DaoCloud Enterprise 5.0

2022 年金秋送爽之际，「DaoCloud 道客」融合开源社区最流行的众多云原生技术，由上百位行业大咖经过为期两年日夜不辍的开发、测试和验证，理想主义的花最终盛开在了浪漫主义的地里，现隆重推出新一代云原生容器化综合平台，引领时下正如火如荼的云原生浪潮。

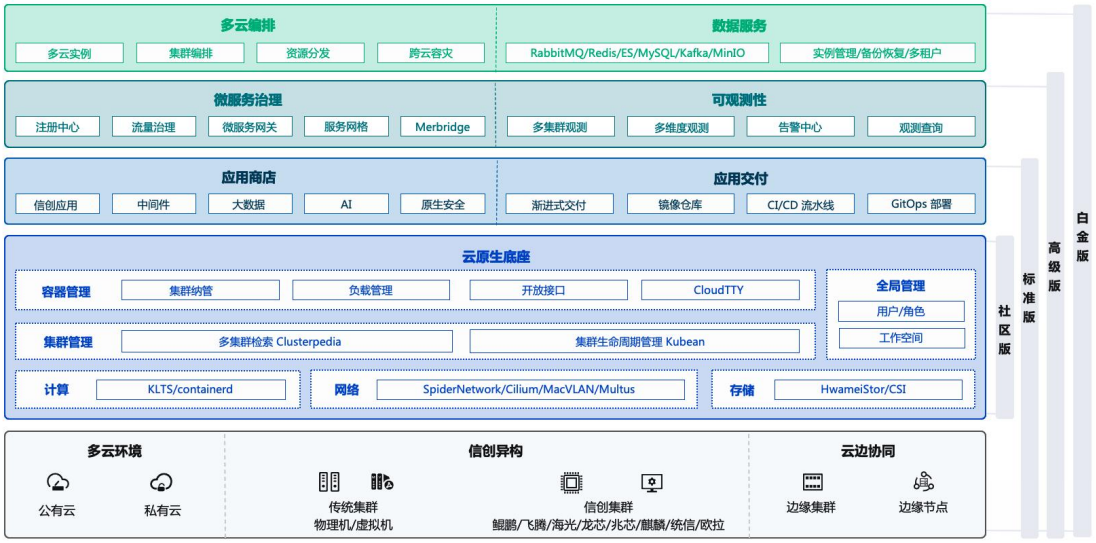
DaoCloud Enterprise 5.0 (DCE 5.0) 是高性能可扩展的云原生操作系统，在任何基础设施上、任意环境上提供一致、稳定的体验，支持信创异构、边云协同、多云编排， 融合前沿的服务网格和微服务技术，将可观测性贯穿于每一个流量生发的始终，助力洞察集群、节点、应用和服务的详细指标， 通过动态仪表盘和拓扑大图可视化掌握应用健康状态。原生支持 DevOps 开发运维模式，实现应用交付的全流程标准化和自动化， 原生集成各类精选数据库和中间件，使得运维治理更高效。

DCE 5.0 的各个产品模块独立解耦，灵活升级，业务无感知，开放对接超百家云原生生态产品，形成完整的解决方案体系， 经近千家行业客户生产场景锤炼，构建坚实可靠的数字底座，释放云原生生产力，助力企业定义数字边界。

- :fontawesome-solid-jet-fighter-up: **安装** 安装社区版和商业版
- :octicons-container-16: **容器管理** 集群/节点/负载等基础设施
- :fontawesome-solid-user-group: **全局管理** 设置登录、访问权限、外观
- :material-monitor-dashboard: **可观测性** 一站式图形化仪表盘
- :material-microsoft-azure-devops: **应用工作台** CI/CD 流水线
- :material-cloud-check: **多云编排** 多云实例/负载/策略
- :material-engine: **微服务引擎** 微服务治理中心和网关
- :material-table-refresh: **服务网格** Istio 定制增强版
- :material-middleware: **中间件** ES, Kafka, MinIO, MySQL 等
- :material-warehouse: **镜像仓库** 镜像集成和托管服务
- :material-dot-net: **网络** 多 CNI 融合方案

- :floppy_disk: **存储** 容器化存储综合方案

DaoCloud Enterprise 5.0



模块图

DCE 5.0 云原生操作系统提供了 9 大能力，自由搭配各种模块，可以应对海量应用场景。

这些模块就像乐高搭积木一样，糅合社区最优秀的几十种开源技术，经过众多辩证选型、攻坚克难、编码调试、海量测试，“十年磨一剑，一朝试锋芒”，全新搭建的新一代容器化平台能够满足企业上云的各类场景需求。

===“多云编排”

支持多云和混合云的统一集中管理，提供跨云资源检索及跨云的应用部署、发布和运维能力，实现多云应用高效管控，提供基于集群资源的应用弹性扩缩，实现全局负载均衡，具备故障恢复能力，有效解决多云应用灾备问题，助力企业构建多云、混合云的数字基础设施。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、信创异构

![多云](https://docs.daocloud.io/daocloud-docs-images/docs/images/01multi-cloud.png)

===“数据中间件服务”

专为有状态应用设计的云原生本地存储能力，满足中间件高 I/O 的存储需求，提升运维管理效率。精选各类数据库、分布式消息和日志检索等中间件，
提供多租户、部署、观测、备份、运维操作等全生命周期的中 间件管理能力，实现数据服务的自助化申请、弹性扩展、 高并发处理和稳定高可用。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[精选中间件](middleware/index.md)

![数据](https://docs.daocloud.io/daocloud-docs-images/docs/images/02data.png)

===“微服务治理”

提供非侵入式流量治理功能，支持无感接入传统微服务、云原生微服务和开源微服务框架，实现企业现有微服务体系及新旧微服务体系的融合治理，
支持微服务从开发、部署、接入、观测、运维的全生命周期管理，提供高性能云原生微服务网关，保证微服务应用的连续可用性；引入自主开源的 eBPF 网格加速技术，全面提高流量转发效率。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[微服务治理](skoala/intro/index.md)、[服务网格](mspider/intro/index.md)、[可观测性](insight/intro/index.md)、[应用工作台](amamba/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![微服务](https://docs.daocloud.io/daocloud-docs-images/docs/images/03msgov.png)

===“可观测性”

基于日志、链路、指标、eBPF 等技术手段，全面采集服务数据，深入获取请求链路信息，动态观测、多维度掌控集群、节点、应用和服务的实时变化，
通过统一控制面实现所有集群及负载观测数据的查询，引入拓扑分析技术可视化掌握应用健康状态，实现秒级故障定位。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[可观测性](insight/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![观测](https://docs.daocloud.io/daocloud-docs-images/docs/images/04insight.png)

===“应用商店”

收录来自大数据、AI、中间件等十大领域生态伙伴的软件产品，实现生态技术、产品、运营服务等能力的整合，提供开箱即用的生态应用软件，面向企业实际业务需求，打造完整的解决方案体系。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

===“应用交付”

通过一致性可推广的应用交付流程实现自助式上云，支持柔性租户体系，动态适配用户组织架构规划和实时资源分配，基于云原生化的 CI/CD 流水线，集成丰富的工具链并支持流水线高效并发执行流转，
自动化完成应用的构建、部署，创新性引入 Gitops、渐进式交付能力体系，实现应用更精细的管理运维。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[应用工作台](amamba/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[镜像仓库](kangaroo/index.md)

![应用交付](https://docs.daocloud.io/daocloud-docs-images/docs/images/06appdeliv.png)

===“信创异构”

采用信创云原生技术架构，兼容国产芯片及服务器，支持信创操作系统及信创应用生态体系，屏蔽底层异构基础设施的复杂性，把传统操作系统从需要长期积累的软件生态兼容适配中解放出来，
实现混合异构集群的灵活调度，保证信创应用运行环境的稳定高可靠，助力信创进程进一步提速。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[集群生命周期管理](community/kubean.md)、[信创中间件](middleware/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![信创](https://docs.daocloud.io/daocloud-docs-images/docs/images/08xinchuan.png)

===“云边协同”

将云原生能力延伸至边缘，采用边缘集群、边缘节点模式，将数据中心算力下移，端设备算力上移，统一管控和调度离散、异构的计算资源，解决在海量边、端设备上统一完成大规模应用交付、运维、管控的诉求，实现云边的真正一体化。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[云边协同](kant/intro/index.md)

![云边](https://docs.daocloud.io/daocloud-docs-images/docs/images/09cloud-edge.png)

===“云原生底座”

提供云原生计算、网络、存储等能力，兼容各种集群接入，支持集群从部署、版本升级、证书变更、配置变更、回收等全生命周期管理，突破 K8s API 性能瓶颈，实现企业超大规模用户并发使用多集群。
针对企业环境，提供场景化的网络方案，实现当前企业网络基础设施复用的最大化，降低企业使用云原生应用门槛。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![云原生底座](https://docs.daocloud.io/daocloud-docs-images/docs/images/07base.png)

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

DaoCloud Enterprise 5.0

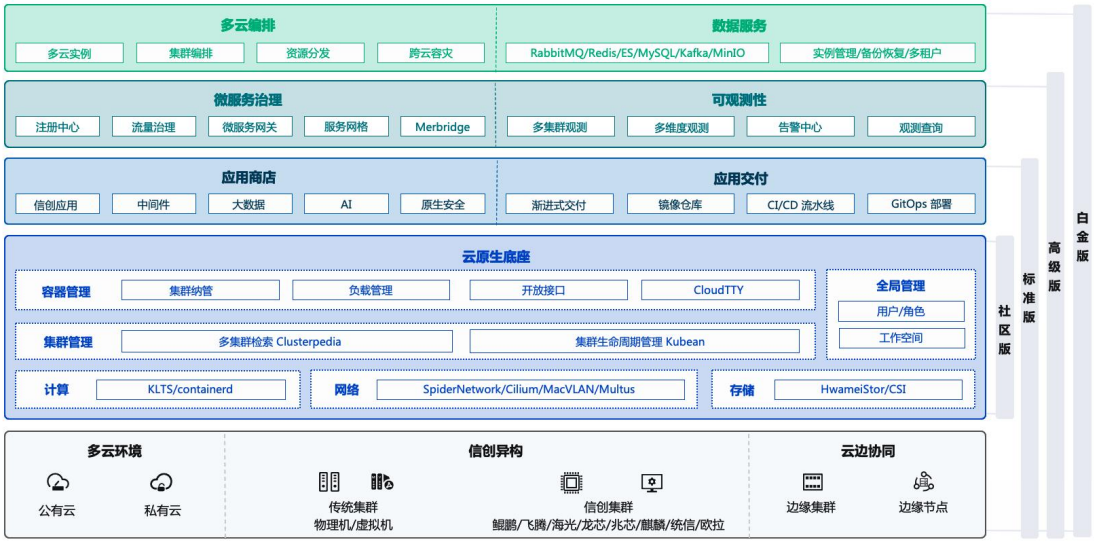
2022 年金秋送爽之际，「DaoCloud 道客」融合开源社区最流行的众多云原生技术，由上百位行业大咖经过为期两年日夜不辍的开发、测试和验证，理想主义的花最终盛开在了浪漫主义的地里，现隆重推出新一代云原生容器化综合平台，引领时下正如火如荼的云原生浪潮。

DaoCloud Enterprise 5.0 (DCE 5.0) 是高性能可扩展的云原生操作系统，在任何基础设施上、任意环境上提供一致、稳定的体验，支持信创异构、边云协同、多云编排， 融合前沿的服务网格和微服务技术，将可观测性贯穿于每一个流量生发的始终，助力洞察集群、节点、应用和服务的详细指标， 通过动态仪表盘和拓扑大图可视化掌握应用健康状态。原生支持 DevOps 开发运维模式，实现应用交付的全流程标准化和自动化， 原生集成各类精选数据库和中间件，使得运维治理更高效。

DCE 5.0 的各个产品模块独立解耦，灵活升级，业务无感知，开放对接超百家云原生生态产品，形成完整的解决方案体系， 经近千家行业客户生产场景锤炼，构建坚实可靠的数字底座，释放云原生生产力，助力企业定义数字边界。

- :fontawesome-solid-jet-fighter-up: **安装** 安装社区版和商业版
- :octicons-container-16: **容器管理** 集群/节点/负载等基础设施
- :fontawesome-solid-user-group: **全局管理** 设置登录、访问权限、外观
- :material-monitor-dashboard: **可观测性** 一站式图形化仪表盘
- :material-microsoft-azure-devops: **应用工作台** CI/CD 流水线
- :material-cloud-check: **多云编排** 多云实例/负载/策略
- :material-engine: **微服务引擎** 微服务治理中心和网关
- :material-table-refresh: **服务网格** Istio 定制增强版
- :material-middleware: **中间件** ES, Kafka, MinIO, MySQL 等
- :material-warehouse: **镜像仓库** 镜像集成和托管服务
- :material-dot-net: **网络** 多 CNI 融合方案
- :floppy_disk: **存储** 容器化存储综合方案

DaoCloud Enterprise 5.0



模块图

DCE 5.0 云原生操作系统提供了 9 大能力，自由搭配各种模块，可以应对海量应用场景。

这些模块就像乐高搭积木一样，糅合社区最优秀的几十种开源技术，经过众多辩证选型、攻坚克难、编码调试、海量测试，“十年磨一剑，一朝试锋芒”，全新搭建的新一代容器化平台能够满足企业上云的各类场景需求。

===“多云编排”

支持多云和混合云的统一集中管理，提供跨云资源检索及跨云的应用部署、发布和运维能力，实现多云应用高效管控，提供基于集群资源的应用弹性扩缩，实现全局负载均衡，具备故障恢复能力，有效解决多云应用灾备问题，助力企业构建多云、混合云的数字基础设施。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、信创异构

![多云](https://docs.daocloud.io/daocloud-docs-images/docs/images/01multi-cloud.png)

===“数据中间件服务”

专为有状态应用设计的云原生本地存储能力，满足中间件高 I/O 的存储需求，提升运维管理效率。精选各类数据库、分布式消息和日志检索等中间件，
提供多租户、部署、观测、备份、运维操作等全生命周期的中 间件管理能力，实现数据服务的自助化申请、弹性扩展、 高并发处理和稳定高可用。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[精选中间件](middleware/index.md)

![数据](https://docs.daocloud.io/daocloud-docs-images/docs/images/02data.png)

===“微服务治理”

提供非侵入式流量治理功能，支持无感接入传统微服务、云原生微服务和开源微服务框架，实现企业现有微服务体系及新旧微服务体系的融合治理，
支持微服务从开发、部署、接入、观测、运维的全生命周期管理，提供高性能云原生微服务网关，保证微服务应用的连续可用性；引入自主开源的 eBPF 网格加速技术，全面提高流量转发效率。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[微服务治理](skoala/intro/index.md)、[服务网格](mspider/intro/index.md)、[可观测性](insight/intro/index.md)、[应用工作台](amamba/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![微服务](https://docs.daocloud.io/daocloud-docs-images/docs/images/03msgov.png)

===“可观测性”

基于日志、链路、指标、eBPF 等技术手段，全面采集服务数据，深入获取请求链路信息，动态观测、多维度掌控集群、节点、应用和服务的实时变化，
通过统一控制面实现所有集群及负载观测数据的查询，引入拓扑分析技术可视化掌握应用健康状态，实现秒级故障定位。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[可观测性](insight/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![观测](https://docs.daocloud.io/daocloud-docs-images/docs/images/04insight.png)

===“应用商店”

收录来自大数据、AI、中间件等十大领域生态伙伴的软件产品，实现生态技术、产品、运营服务等能力的整合，提供开箱即用的生态应用软件，面向企业实际业务需求，打造完整

的解决方案体系。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

===“应用交付”

通过一致性可推广的应用交付流程实现自助式上云，支持柔性租户体系，动态适配用户组织架构规划和实时资源分配，基于云原生化的 CI/CD 流水线，集成丰富的工具链并支持流水线高效并发执行流转，
自动化完成应用的构建、部署，创新性引入 Gitops、渐进式交付能力体系，实现应用更精细的管理运维。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[应用工作台](amamba/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[镜像仓库](kangaroo/index.md)

![应用交付](https://docs.daocloud.io/daocloud-docs-images/docs/images/06appdeliv.png)

===“信创异构”

采用信创云原生技术架构，兼容国产芯片及服务器，支持信创操作系统及信创应用生态体系，屏蔽底层异构基础设施的复杂性，把传统操作系统从需要长期积累的软件生态兼容适配中解放出来，
实现混合异构集群的灵活调度，保证信创应用运行环境的稳定高可靠，助力信创进程进一步提速。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[集群生命周期管理](community/kubean.md)、[信创中间件](middleware/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![信创](https://docs.daocloud.io/daocloud-docs-images/docs/images/08xinchuan.png)

===“云边协同”

将云原生能力延伸至边缘，采用边缘集群、边缘节点模式，将数据中心算力下移，端设备算力上移，统一管控和调度离散、异构的计算资源，解决在海量边、端设备上统一完成大规模应用交付、运维、管控的诉求，实现云边的真正一体化。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)、[云边协同](kant/intro/index.md)

![云边](https://docs.daocloud.io/daocloud-docs-images/docs/images/09cloud-edge.png)

===“云原生底座”

提供云原生计算、网络、存储等能力，兼容各种集群接入，支持集群从部署、版本升级、证书变更、配置变更、回收等全生命周期管理，突破 K8s API 性能瓶颈，实现企业超大规模用户并发使用多集群。
针对企业环境，提供场景化的网络方案，实现当前企业网络基础设施复用的最大化，降低企业使用云原生应用门槛。

涉及的模块：[全局管理](ghippo/intro/index.md)、[容器管理](kpanda/intro/index.md)、[云原生网络](network/intro/index.md)、[云原生存储](storage/index.md)

![云原生底座](https://docs.daocloud.io/daocloud-docs-images/docs/images/07base.png)

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

数据服务权限设计说明

数据服务模块建立在工作空间之上，数据服务的配置管理及各个中间件模块，对应工作空间各个角色的权限映射如下：

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
配置管理	配置列表	查看列表 配置名称搜索 创建配置 更新配置 删除配置			
MySQL	MySQL 实例列表	查看列表 实例名称搜索 创建实例 更新实例配置 删除实			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		例			
	MySQL 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
		查看实例备份列表			
		创建实例备份			
		修改实例自动备份任务			
		使用备份创建			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		新实例			
	备份配置管理	备份配置列表			
		创建备份配置			
		修改备份配置			
		删除备份配置			
	配置参数	查看配置参数			
		修改配置参数			
RabbitMQ	RabbitMQ 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	RabbitMQ 实例详情	实例概览			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
Elasticsearch	Elasticsearch实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	Elasticsearch实例详情	实例概览			
		实例监控			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
Redis	Redis 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	Redis 实例详情	实例概览			
		实例监控			
		查看实例配置			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		参数修改实例配置参数			
		查看实例访问密码			
	备份配置管理	备份配置列表			
		创建备份配置			
		修改备份配置			
		删除备份配置			
	配置参数	查看配置参数			
		修改配置参数			
Kafka	Kafka 实例列表	查看列表			
		实例名称搜索			
		创建实例			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		更新实例配置			
		删除实例			
	Kafka 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
	配置参数	查看配置参数			
		修改配置参数			
MinIO	MinIO 实例列表	查看列表			
		实例名称搜索			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		创建实例			
		更新实例配置			
		删除实例			
	MinIO 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
	配置参数	查看配置参数			
		修改配置参数			
PostgreSQL	PostgreSQL 实例列表	查看列表			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	PostgreSQL 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
	配置参数	查看配置参数			
		修改配置参数			

数据服务介绍

DCE 5.0 针对实际应用场景，精选了一些经典的数据服务中间件，通过前后端开发，能够满足各类应用场景的开发和维护。

用户可以按需安装/启用以下数据服务中间件，即插即用：

- Elasticsearch 搜索服务：目前首选的全文搜索引擎
- Kafka 消息队列：常用于消息传输的数据管道
- MinIO 对象存储：非常热门的轻量对象存储方案
- MySQL 数据库：简单易用、性能更高的开源关系型数据库
- RabbitMQ 消息队列：常用于交易数据的传输管道
- Redis 缓存服务：高性能的开源内存数据库
- PostgreSQL 数据库：功能更丰富、扩展性更强的开源关系型数据库
- MongoDB 数据库：面向文档的 NoSQL 数据库管理系统

数据服务学习路径

上述几个数据服务中间件的学习路径大致相同，此处以 RabbitMQ 为例，简单说明学习路径。

!!! info

在下方流程流程图中点击相应文字可以直接跳转到对应的操作指南页面。

graph TD

B(选择工作空间) --> C{部署/创建实例}
C -.->D[更新/删除实例]
C -.->E[实例概览]
C -.->F[实例监控]
C -.->G[数据迁移]

C -.->H[卸载中间件]

click B "https://docs.daocloud.io/middleware/rabbitmq/user-guide/login/"
click C "https://docs.daocloud.io/middleware/rabbitmq/user-guide/create/"
click D "https://docs.daocloud.io/middleware/rabbitmq/user-guide/update/"
click E "https://docs.daocloud.io/middleware/rabbitmq/user-guide/view/"
click F "https://docs.daocloud.io/middleware/rabbitmq/user-guide/insight/"
click G "https://docs.daocloud.io/middleware/rabbitmq/user-guide/migrate/"
click H "https://docs.daocloud.io/middleware/rabbitmq/quickstart/install/#_1"

快速使用

Elasticsearch Kafka MinIO MySQL RabbitMQ Redis PostgreSQL MongoDB

数据服务与 Kubernetes 版本适配

目前数据服务已正式发布了 RabbitMQ、Elasticsearch、MySQL、Redis、Kafka、MinIO、PostgreSQL 七个数据服务中间件。

Kubernetes 版本适配情况

数据服务	功能	1.26	1.25	1.24	1.23	1.22	备注
RabbitMQ	Operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						
	删除实例						
ElasticSearch	operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						
	删除实例						
MySQL	operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						

数据服务	功能	1.26	1.25	1.24	1.23	1.22	备注
	删除实例						
Redis	operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						
	删除实例						
Kafka	operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						
	删除实例						
MinIO	operator						
	创建实例						
	编辑实例						
	查询实例						
	实例连接						
	删除实例						
PostgreSQL	operator						
	创建实例						
	编辑实例						
	查询实例						

数据服务	功能	1.26	1.25	1.24	1.23	1.22	备注
	实例连接						
	删除实例						

PostgreSQL Release Notes

本页列出 PostgreSQL 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.4.0

新增

- **新增** 备份管理能力。

优化

- **优化** 监控图表，去除干扰元素并新增时间范围选择

修复

- **修复** 监控图表部分 Panel 无法展示的问题。

2023-06-30

v0.3.0

新功能

- **新增** 对接全局管理审计日志模块

优化

- **优化** 监控图表，去除干扰元素并新增时间范围选择

2023-04-27

v0.1.2

新功能

- **新增** Mcamel-PostgreSQL UI 模块上线，支界面化管理
- **新增** 安装 Mcamel-PostgreSQL PgAdmin 离线镜像版本能力。
- **新增** Mcamel-PostgreSQL PgAdmin 支持 LoadBalancer 类型。

优化

- **优化** Mcamel-PostgreSQL 在 IPV6 的情况下 Exporter 没有权限连接到 PostgreSQL
- **优化** Mcamel-PostgreSQL 调度策略增加滑动按钮

2023-04-20

v0.1.1

新功能

- **新增** Mcamel-PostgreSQL 支持 快速启动 PostgreSQL 集群
- **新增** Mcamel-PostgreSQL 支持 ARM 环境下部署
- **新增** Mcamel-PostgreSQL 支持 自定义角色
- **新增** Mcamel-PostgreSQL 支持 PostgreSQL 生命周期管理的后端接口
- **新增** Mcamel-PostgreSQL 支持 查看日志
- **新增** Mcamel-PostgreSQL 支持 PostgreSQL 15
- **新增** Mcamel-PostgreSQL 支持 PostgreSQL UI 管理界面

2023-03-29

v0.0.2

新功能

- **新增** Mcamel-PostgreSQL 支持 PostgreSQL 实例创建

技术特性及基本概念

- **多版本并发控制（MVCC）**

  PostgreSQL 的 MVCC 机制使得并发控制能力得到了很大的提升，允许事务以一种非阻塞的方式执行，从而避免了死锁的情况。

- **复制和高可用性**

  支持多种复制和高可用性方案，包括主从复制、流复制和逻辑复制等。这些方案可以提供数据冗余和自动故障转移等能力，从而提高系统的可用性。

- **安全性**

  提供严格的安全保护措施，包括访问控制列表（ACL）、加密传输、密码认证和审计日志等。这些措施能够保证数据库数据的安全和完整性。

- **扩展性**

  允许用户通过扩展来增强其功能。例如，用户可以自定义类型、自定义函数、自定义操作符和自定义索引等等。这些扩展能够增强 PostgreSQL 的能力和适应性。

- **性能优化**

  良好的性能优化机制，可以通过各种手段对性能进行优化，包括索引优化、查询优化和配置优化等。同时，PostgreSQL 提供了强大的统计信息和实时监控工具，可以帮助用户进行性能问题的排查和优化。

- **支持 JSON 和 JSONB 数据类型**

  支持直接存储和处理 JSON 格式的数据，这让 PostgreSQL 成为一个非常有用的 NoSQL 数据库。

- **支持全文检索**

内置全文检索功能，可以对大量的文本数据进行高效的搜索和匹配。

- **支持地理信息数据类型**

  支持对地理信息数据进行存储和查询，这让 PostgreSQL 成为一个非常有用的 GIS 数据库。

- **支持分区表**

  支持将一个大型的表分成多个小型表进行存储和查询，这可以提高查询效率和管理方便性。

- **支持并发控制**

  通过 MVCC 机制来提高并发控制能力，同时 PostgreSQL 还支持多种隔离级别，包括 Read Committed、Repeatable Read 和 Serializable 等。

- **支持 PL/SQL 和 PL/Python 等存储过程语言**

  支持多种存储过程语言，包括 PL/SQL、PL/Python 和 PL/Perl 等，这使得 PostgreSQL 可以结合其他编程语言进行开发和扩展。

什么是 PostgreSQL

PostgreSQL 是一种开源的关系型数据库管理系统（RDBMS），它采用了美国加州大学伯克利分校开发的 Postgres 作为基础，并由 PostgreSQL 全球开发团队进行开发和维护。它是一款功能丰富、可扩展性强、安全可靠、兼容性好的数据库，已广泛应用于企业、互联网公司和开源社区中。

PostgreSQL 的特点包括：

- 具备完整的 ACID 事务支持，支持多版本并发控制（MVCC）机制，保证数据一致性。
- 能够灵活地支持各种数据类型，包括数值、日期、时间、JSON、XML 等。
- 提供灵活的扩展机制，包括自定义函数、PL/SQL 存储过程、外部表等。
- 能够支持复杂查询，包括聚合、子查询、连接等。
- 支持存储过程和触发器，以实现数据库应用逻辑的封装。
- 具备高度安全性，包括 SSL 加密、数据的备份和自动恢复机制等。
- 提供跨平台的支持，可以在各种操作系统下运行，包括 Linux、Windows、Unix 等。
- 具备高可用性和可扩展性，支持主从复制、流复制和连接池等机制。

PostgreSQL 还有一些其他的特点，如稳定性强、性能优异、易于部署和管理等。它也被广泛应用于大规模的企业级应用开发、互联网应用和数据仓库等领域。

在 PostgreSQL 社区中，有大量的贡献者和开发者，他们为 PostgreSQL 项目做出了各种各样的贡献，包括功能开发、错误修复、文档编写等。PostgreSQL 社区也提供了各种资源和支持，包括邮件列表、论坛、官方文档等，对开发者提供全面的帮助和支持。

容器化 PostgreSQL

容器化技术通过将应用和其依赖项打包在一个独立的运行环境中，从而实现了一种轻量级、便携和可重复性的部署方式。这种技术对于 PostgreSQL 数据库的安装、配置和管理非常有用。

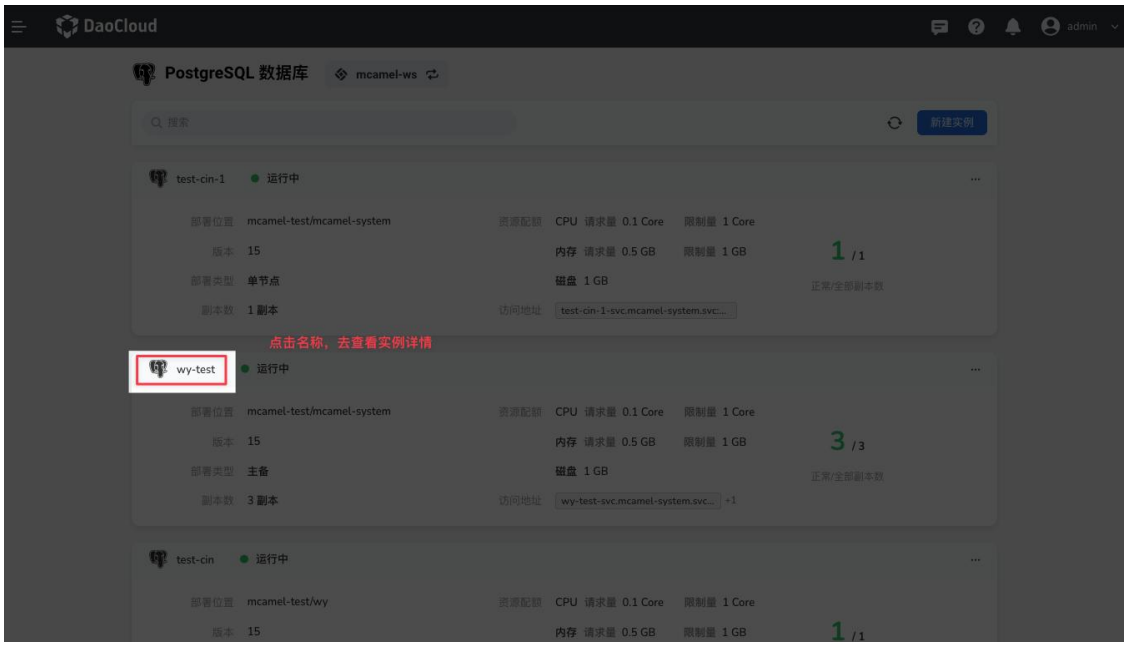
- 简化部署和管理：容器化技术可以轻松地对 PostgreSQL 进行安装、配置和部署，减少开发人员和系统管理员的工作量。
- 快速复制和扩展：容器化 PostgreSQL 能够很容易地进行复制和扩展，从而满足大规模数据集的需求。
- 跨平台支持：由于容器化是一种轻量级的虚拟化技术，因此可以在任何支持该技术的服务器上运行。
- 可靠性和安全性：容器化技术提供了一个隔离的运行环境，可以有效地保护 PostgreSQL 实例免受外部攻击和运行环境的影响。 创建 PostgreSQL 实例

查看 PostgreSQL 日志

操作步骤

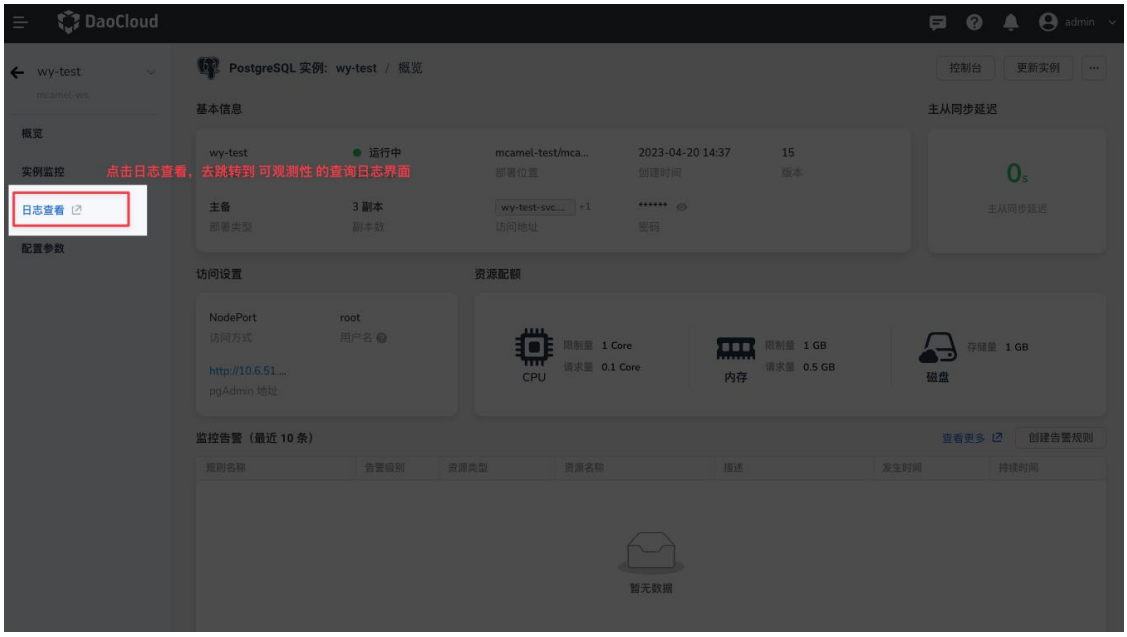
通过访问每个 PostgreSQL 的实例详情，页面；可以支持查看 PostgreSQL 的日志。

1. 在 PostgreSQL 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



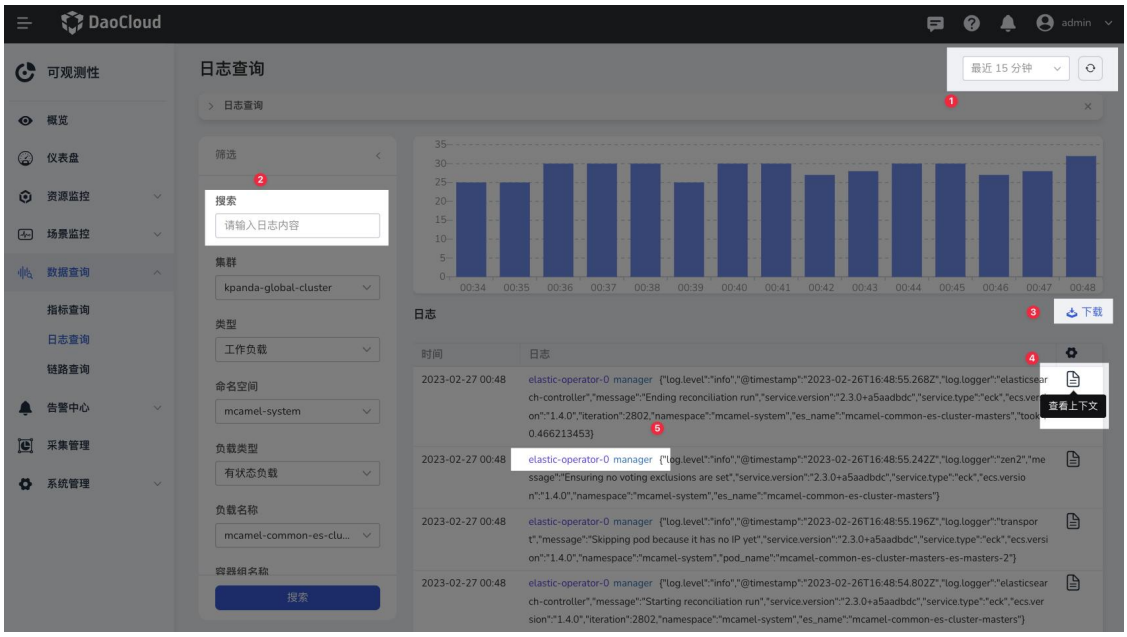
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

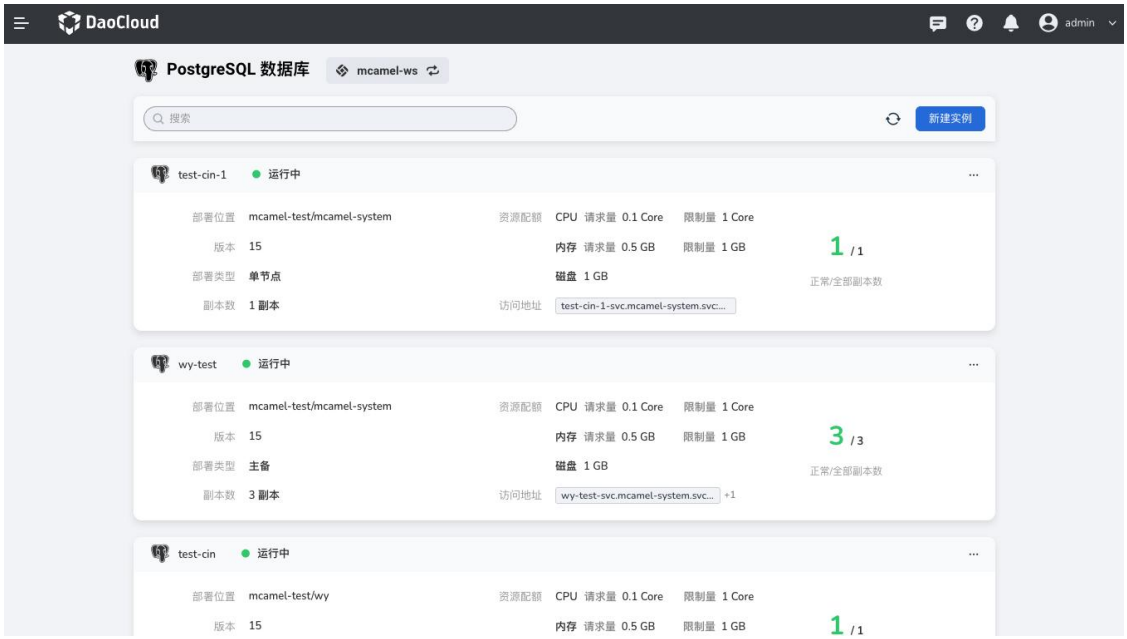

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

创建 PostgreSQL 实例

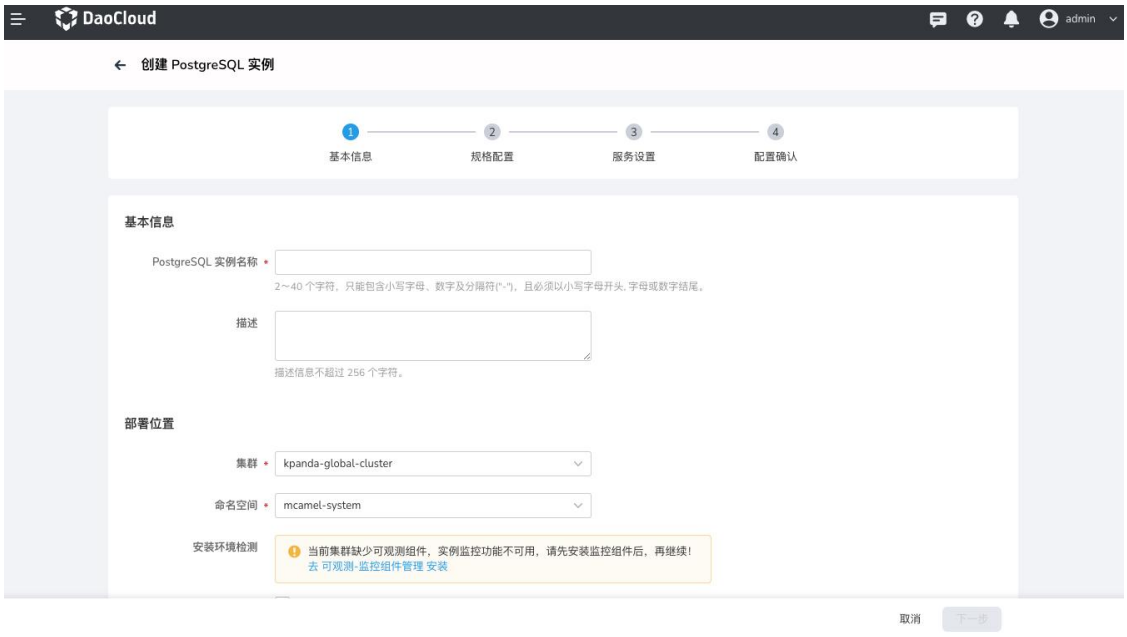
接入 PostgreSQL 数据库后，参照以下步骤创建 PostgreSQL 实例。

1. 在实例列表中，点击右上角的 新建实例。



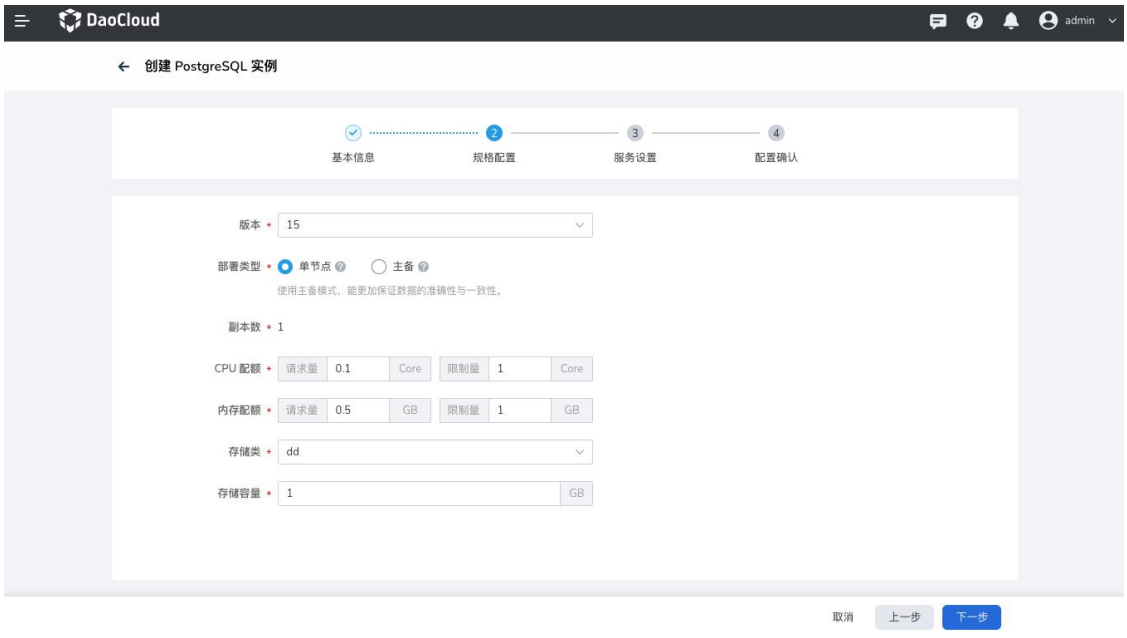
基本信息

2. 在创建 PostgreSQL 实例页面中，配置基本信息后，点击下一步。



基本信息

3. 选择部署类型、CPU、内存和存储等规格配置后，点击下一步。



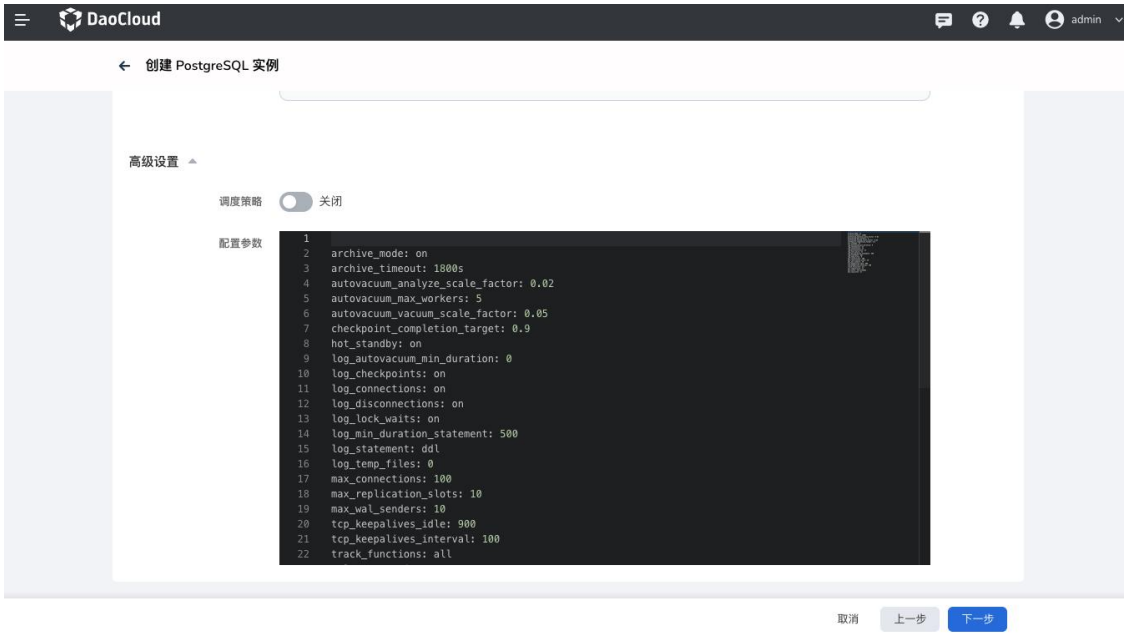
规格配置

4. 设置用户名和密码等服务设置，默认采用 ClusterIP 作为访问方式。



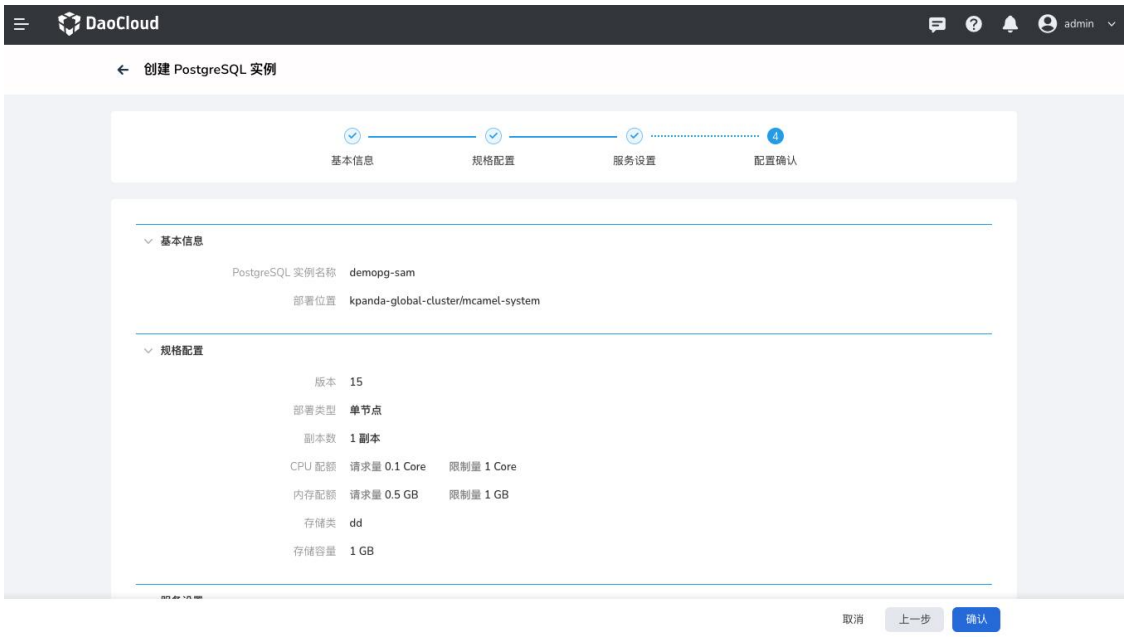
服务设置

5. 支持优化配置 PostgreSQL 的初始化配置，提供了默认通用配置。



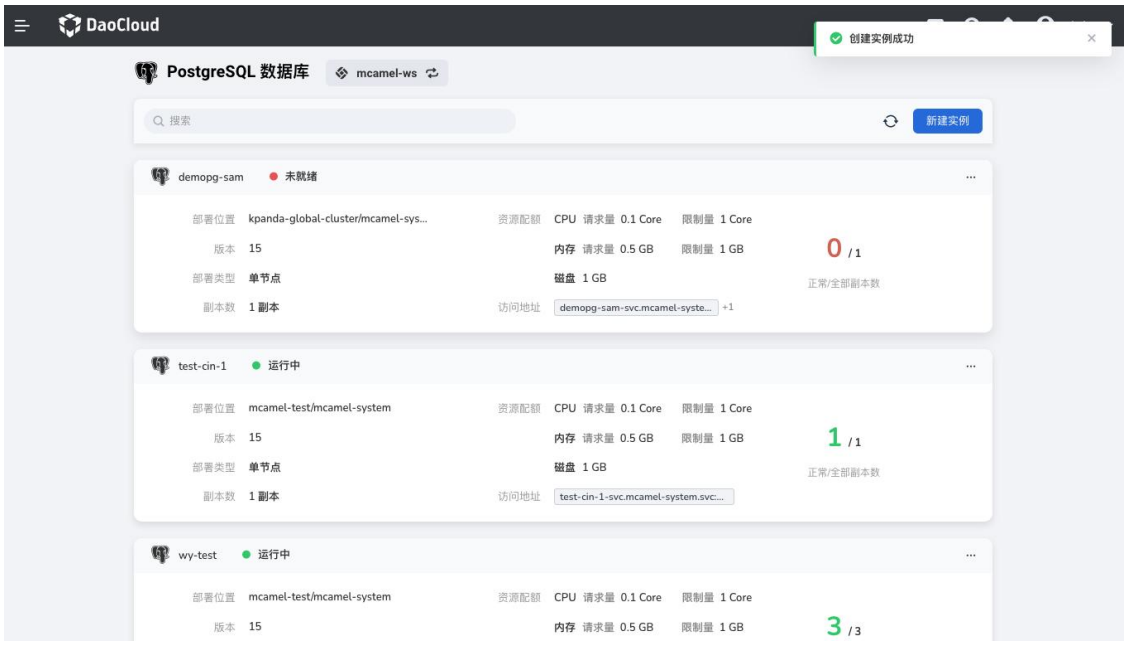
服务设置

6. 确认基本信息、规格配置、服务设置的信息准确无误后，点击确认。



确认

7. 返回实例列表，屏幕将提示创建实例成功。

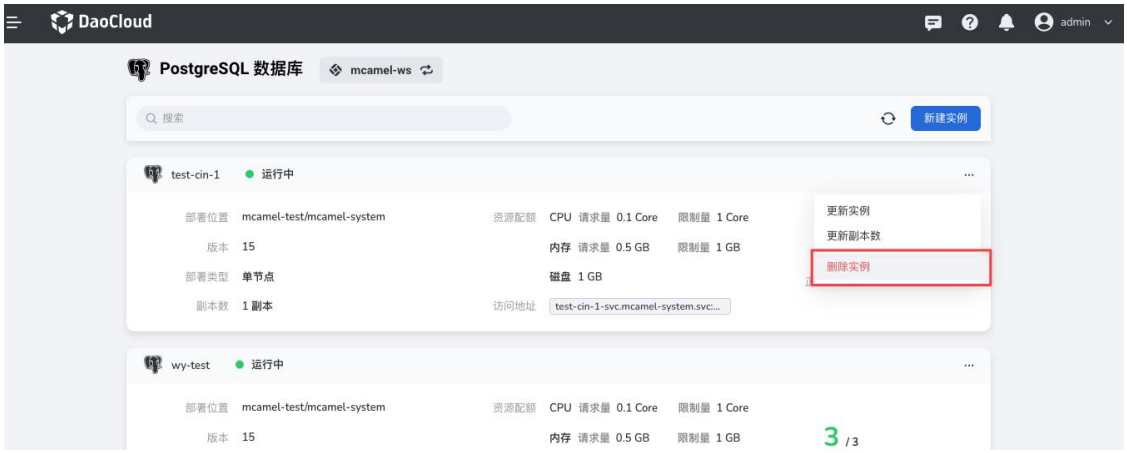


成功创建

删除 PostgreSQL 实例

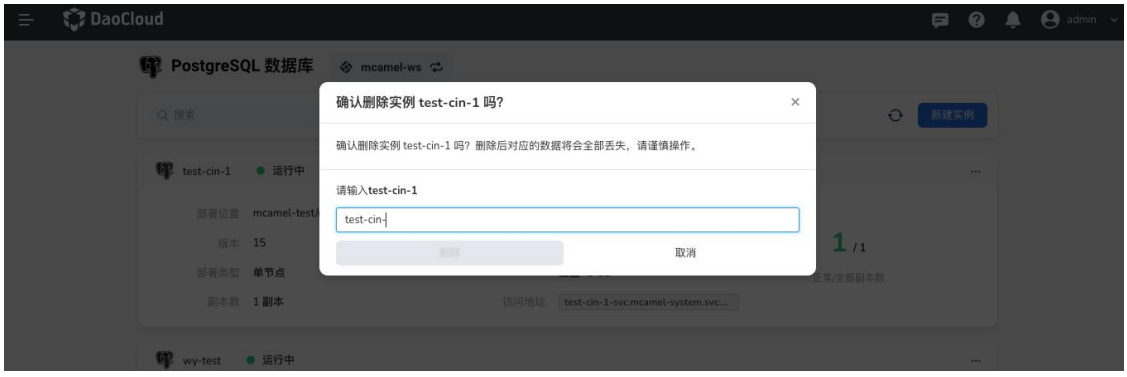
如果想要删除一个 PostgreSQL 实例，可以执行如下操作：

1. 在 PostgreSQL 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。



点击删除

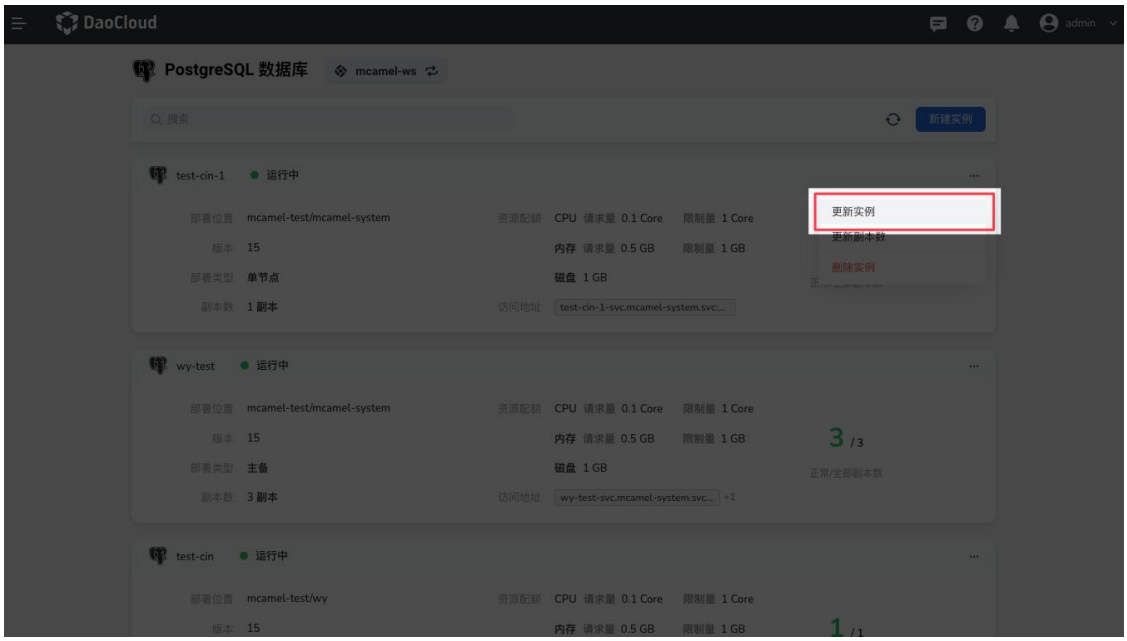
!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

更新 PostgreSQL 实例

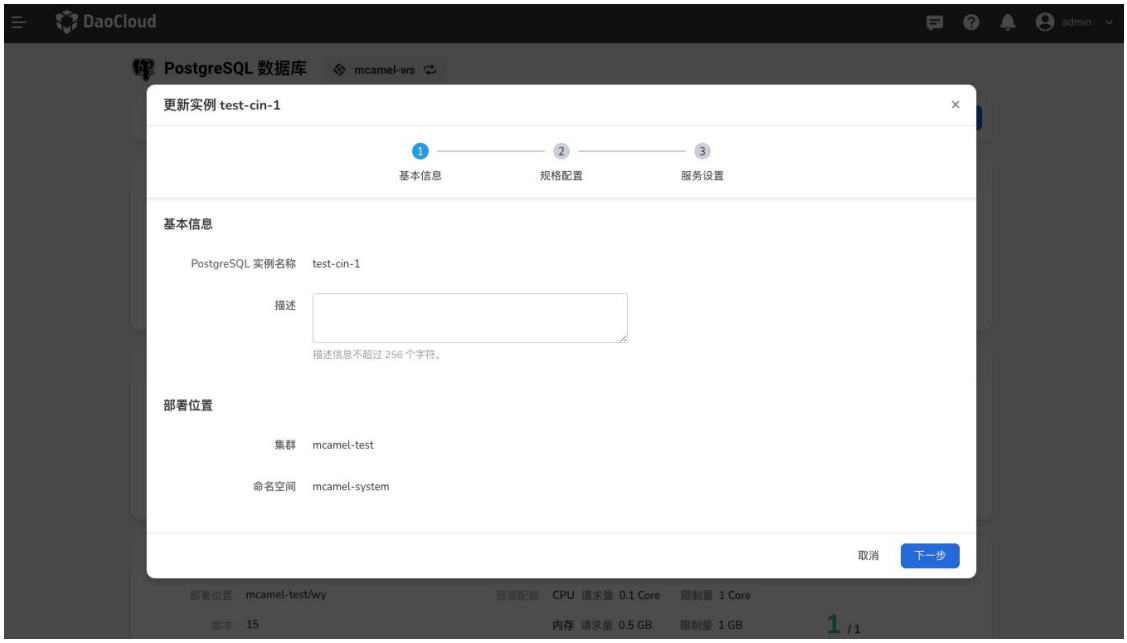
如果想要更新或修改 PostgreSQL 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



更新实例

2. 基本信息：只能修改描述。然后点击下一步。



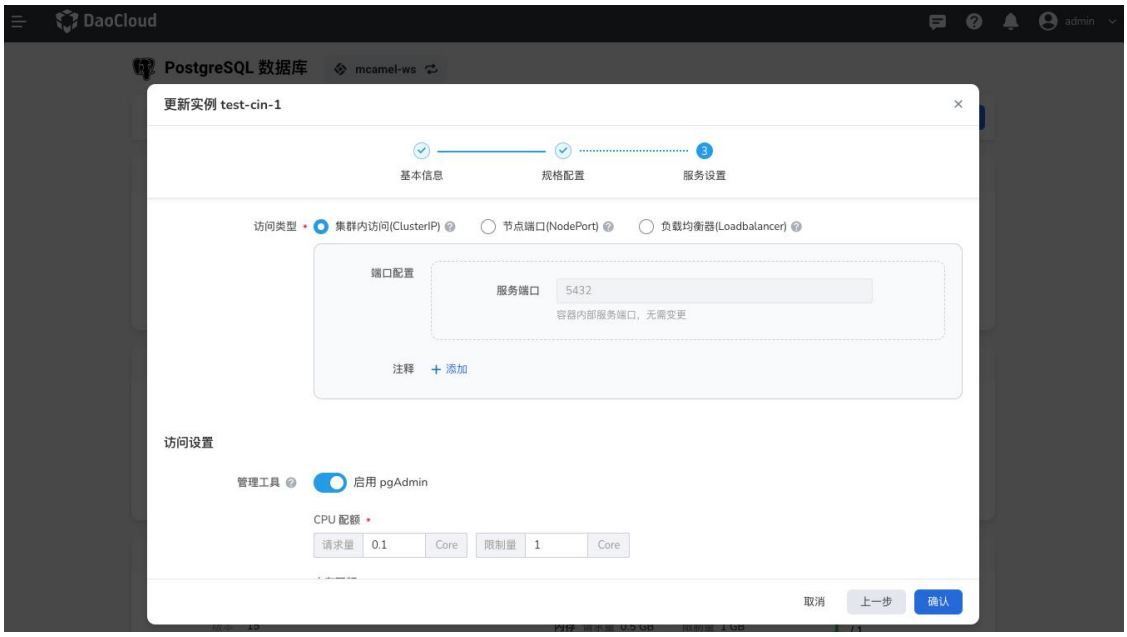
基本信息

3. 修改规格配置后点击下一步。



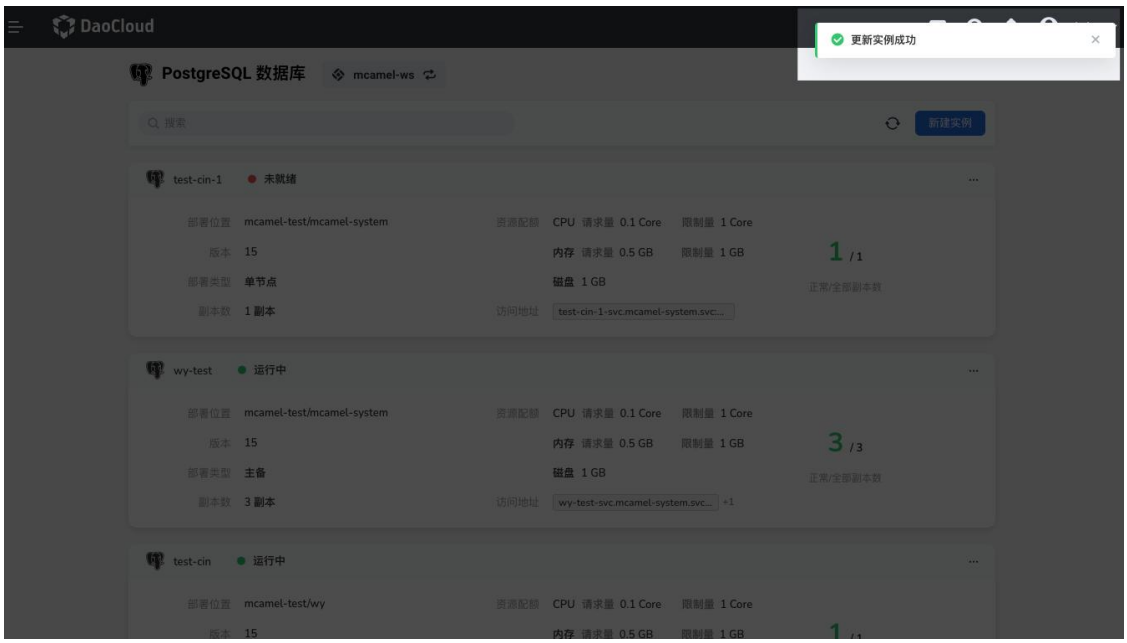
规格配置

4. 修改服务设置后点击确认。



服务设置

5. 实例配置更新成功，可以在右上角看到更新结果提示。



更新完成

Redis 缓存服务 Release Notes

本页列出 Redis 缓存服务的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.10.0

新功能

- **新增** mcamel-redis 访问白名单配置。

优化

- **优化** mcamel-redis 创建实例对话框添加默认反亲和标签值，简化配置过程。
- **优化** mcamel-redis 数据恢复界面。
- **优化** mcamel-redis 前端界面权限展示信息。

修复

- **修复** mcamel-redis 关闭节点亲和性失败。

2023-06-30

v0.9.0

新功能

- **新增** mcamel-redis 不同命名空间下无法创建同名 Redis。
- **新增** mcamel-redis 非 MCamel 纳管的 Redis 可能被误操作的问题。

优化

- **优化** mcamel-redis 备份管理页面结构样式展示。
- **优化** mcamel-redis 备份 Job 中的密码展示问题。
- **优化** mcamel-redis 监控图表，去除干扰元素并新增时间范围选择。
- **优化** mcamel-redis ServiceMonitor 闭环安装。

2023-05-30

v0.8.0

新功能

- **新增** mcamel-redis 可配置实例反亲和性。
- **新增** mcamel-redis 对接全局管理审计日志模块。
- **修复** mcamel-redis 删除 Redis 后，备份相关内容残留。

修复

- **修复** mcamel-redis 哨兵集群的 Service 地址展示错误。

2023-04-27

v0.7.1

新功能

- **新增** mcamel-redis 详情页面展示相关的事件
- **新增** mcamel-redis 列表接口支持 Cluster 与 Namespace 字段过滤
- **新增** mcamel-redis 自定义角色

修复

- **修复** mcamel-redis 优化 调度策略增加滑动按钮

2023-03-29

v0.6.2

新功能

- **新增** mcamel-redis 支持自动化备份恢复。

修复

- **修复** 没有导出备份恢复的离线镜像。
- **修复** mcamel-redis 没有导出备份恢复的离线镜像。
- **修复** mcamel-redis 修复了若干已知问题，提升了系统稳定性和安全性。

文档

- **新增** 新增备份功能使用文档。

2023-02-23

v0.5.0

API

- **新增** mcamel-redis helm-docs 模板文件。
- **新增** mcamel-redis 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-redis 支持 cloud shell。
- **新增** mcamel-redis 支持导航栏单独注册。
- **新增** mcamel-redis 支持查看日志。
- **新增** mcamel-redis 更新单例/集群模式 Operator 的版本。
- **新增** mcamel-redis 展示 common Redis 集群。
- **新增** mcamel-redis Operator 对接 chart-syncer。
- **修复** mcamel-redis 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-redis 工作空间 Editor 用户无法查看实例密码。
- **修复** mcamel-redis 不能解析出正确的 Redis 版本号。
- **修复** mcamel-redis 无法修改 Port 的问题。
- **升级** mcamel-redis 升级离线镜像检测脚本。

文档

- **新增** 日志查看操作说明，支持自定义查询、导出等功能。

2022-12-25

v0.4.0

API

- **新增** mcamel-redis NodePort 端口冲突提前检测。
- **新增** mcamel-redis 节点亲和性配置。
- **修复** mcamel-redis 单例和集群设置 nodeport 无效的问题。
- **修复** mcamel-redis 集群模式下，从节点不可以设置为 0 的问题。

2022-11-28

v0.2.6

API

- **修复** 更新 Redis 时校验部分字段错误
- **改进** 密码校验调整为 MCamel 低等密码强度
- **改进** 提升哨兵模式的版本依赖，v1.1.1=>v1.2.2，重要变更为支持 k8s 1.25+
- **新增** 支持在 ARM 环境安装主备模式的 Redis 集群
- **新增** sc 扩容提示
- **新增** 返回列表或者详情时的公共字段
- **新增** 增加返回告警列表
- **新增** 校验 Service 注释
- **修复** 服务地址展示错误

2022-10-26

v0.2.2

API

- **新增** 获取用户列表接口
- **新增** 支持 arm 架构
- **新增** redis 实例全生命周期的管理
- **新增** redis 实例的监控部署
- **新增** 支持 redis 哨兵，单例和集群一键部署
- **新增** 支持 ws 权限隔离
- **新增** 支持在线动态扩容
- **升级** release notes 脚本

什么是 Redis 缓存服务

Redis 缓存服务是 DaoCloud 提供的一款内存数据库缓存服务，兼容了 Redis 和 Memcached 两种内存数据库引擎，为您提供开箱即用、安全可靠、弹性扩容、便捷管理的在线分布式缓存能力，满足用户高并发及数据快速访问的业务诉求。

- 开箱即用

  提供单机、高可用集群、Cluster 集群、读写分离类型的缓存实例，拥有从 128M 到 1024G 的丰富内存规格。您可以通过 UI 控制台直接创建，无需单独准备服务器资源。

  所有 Redis 版本采用容器化部署，秒级完成创建。

- 安全可靠

  借助 DaoCloud 全局管理、审计日志等安全管理服务，全方位保护实例数据的存储与访问。

  灵活的容灾策略，主备/集群实例从单集群内部署，到支持多集群多云部署。

- 弹性伸缩

  提供对实例内存规格的在线扩容与缩容服务，帮助您实现基于实际业务量的成本控制，达到按需使用的目标。

- 便捷管理

  可视化 Web 管理界面，在线完成实例重启、参数修改等操作。还提供 RESTful API，方便您进一步实现实例自动化管理。

创建 Redis 实例

适用场景

很多大型电商网站、视频直播和游戏平台等，存在 **大规模数据访问**，对 **数据查询效率要求高**，且 **数据结构简单**，**不涉及太多关联查询**。 这种场景使用 Redis，在速度上对传统磁盘数据库有很大优势，能够有效减少数据库磁盘 IO，提高数据查询效率，减轻管理维护工作量，降低数据库存储成本。 Redis 对传统磁盘数据库是一个重要的补充，成为了互联网应用，尤其是支持高并发访问的互联网应用必不可少的基础服务之一。

以下举几个典型样例：

1. 电商网站 – 秒杀抢购

   电商网站的商品类目、推荐系统以及秒杀抢购活动，适宜使用 Redis 缓存数据库。

   例如秒杀抢购活动，并发高，对于传统关系型数据库来说访问压力大，需要较高的硬件配置（如磁盘 IO）支撑。 Redis 数据库单节点 QPS 支撑能达到 10 万，轻松应对秒杀并发。实现秒杀和数据加锁的命令简单，使用 SET、GET、DEL、RPUSH 等命令即可。

2. 视频直播 – 消息弹幕

   直播间的在线用户列表，礼物排行榜，弹幕消息等信息，都适合使用 Redis 中的 SortedSet 结构进行存储。

   例如弹幕消息，可使用 ZREVRANGEBYSCORE 排序返回，在 Redis 5.0 中，新增了 zpopmax、zpopmin 命令，更加方便消息处理。

3. 游戏应用 – 游戏排行榜

在线游戏一般涉及排行榜实时展现，比如列出当前得分或战力最高的 10 个用户。 使用 Redis 的有序集合存储用户排行榜非常合适，有序集合使用非常简单，提供多达 20 个操作集合的命令。

4. 社交 App – 返回最新评论/回复

   在 web 类应用中，常有“最新评论”之类的查询，如果使用关系型数据库，往往涉及到按评论时间逆排序，随着评论越来越多，排序效率越来越低，且并发频繁。

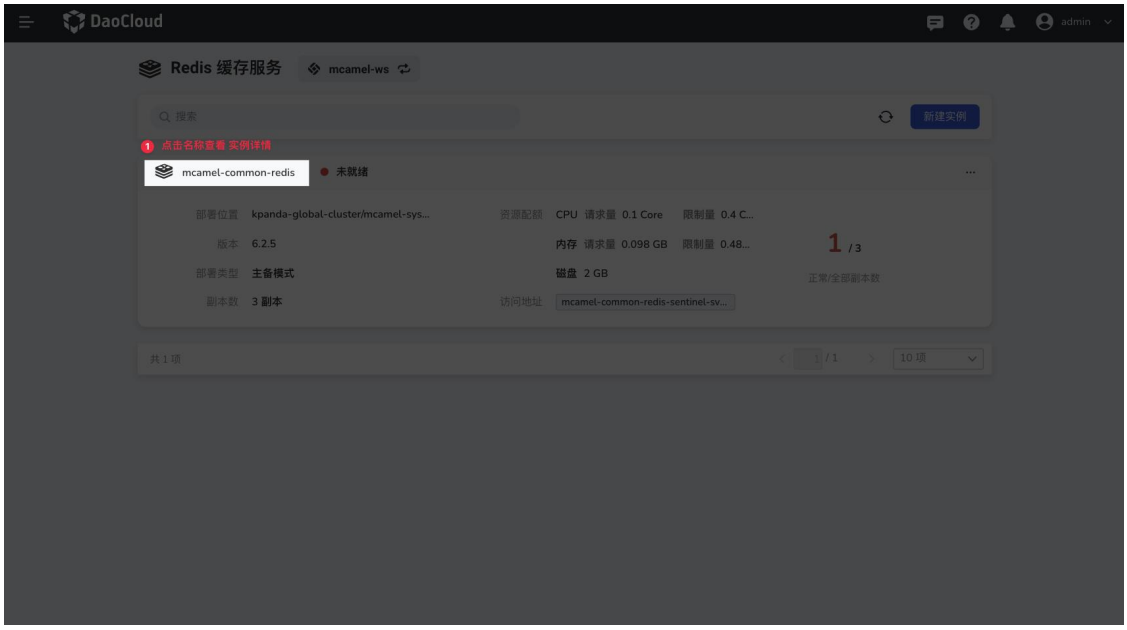
   使用 Redis 的 List（链表）来存储最新 1000 条评论，当请求的评论数在这个范围，就不需要访问磁盘数据库，直接从缓存中返回，减少数据库压力的同时，提升 App 的响应速度。

查看 Redis 日志

操作步骤

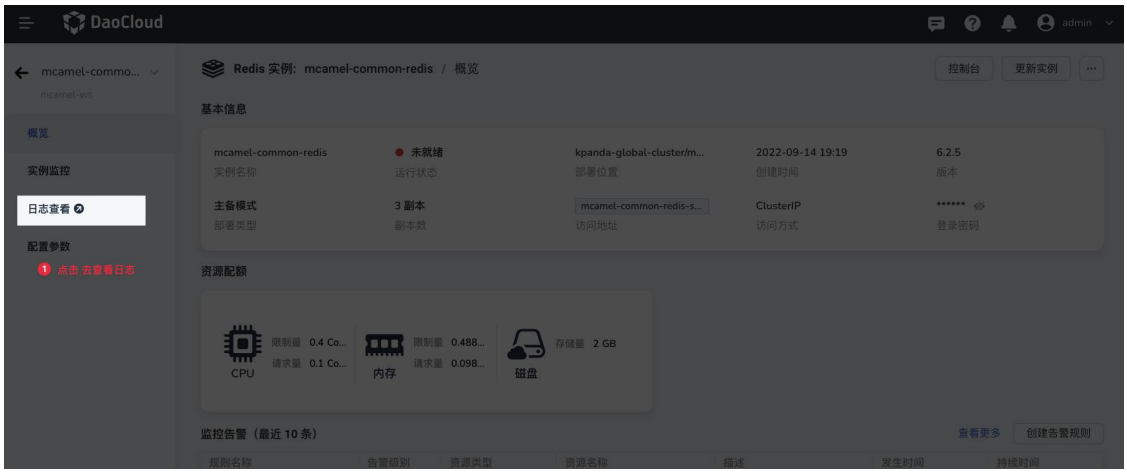
通过访问每个 Redis 的实例详情，页面；可以支持查看 Redis 的日志。

1. 在 Redis 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



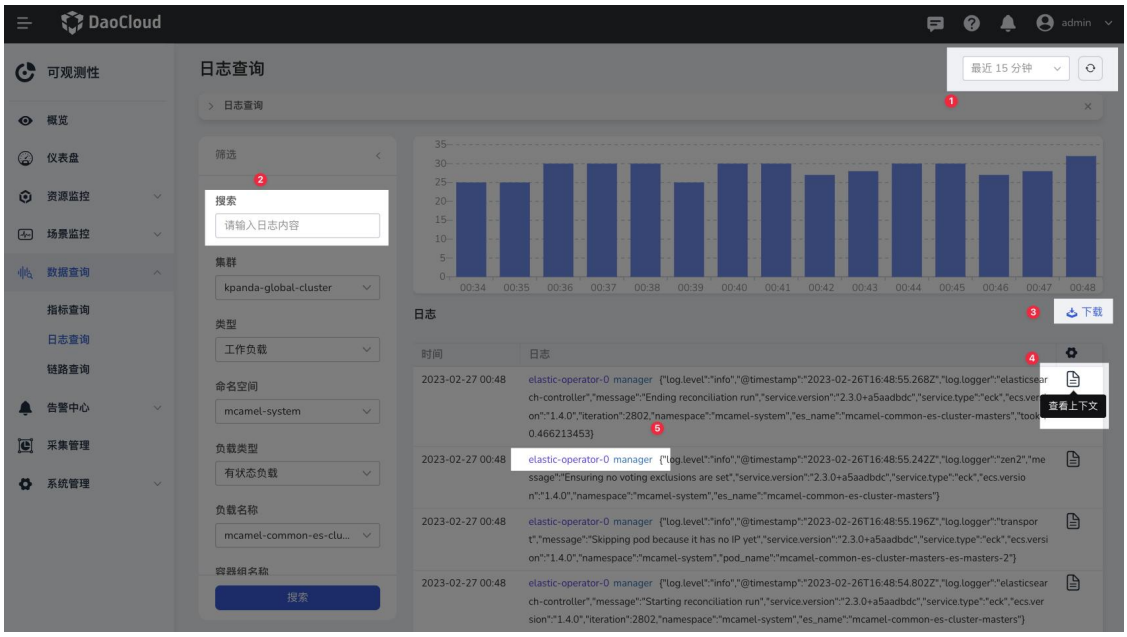
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

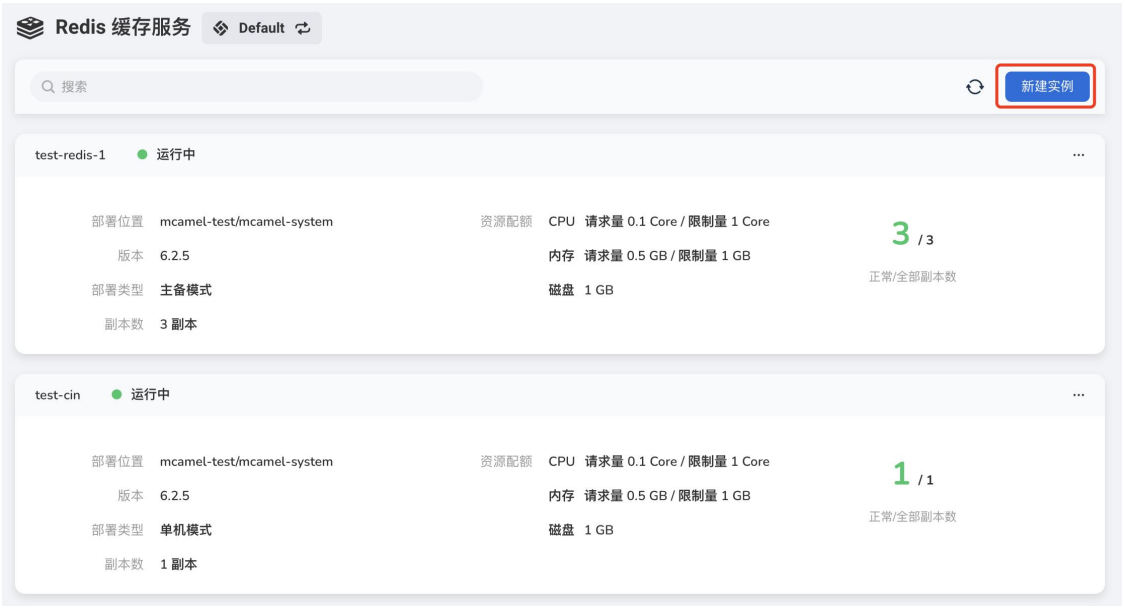

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

创建 Redis 实例

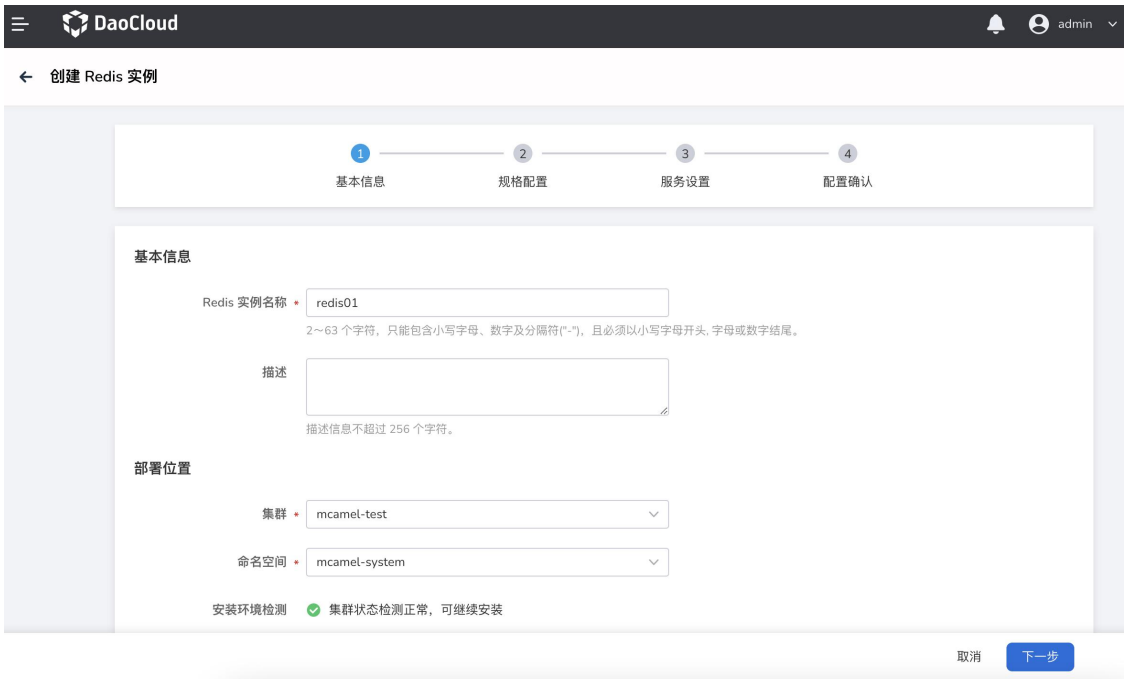
接入 Redis 缓存服务后，参照以下步骤创建 Redis 实例。

1. 在 Redis 缓存服务的实例列表中，点击新建部署按钮。



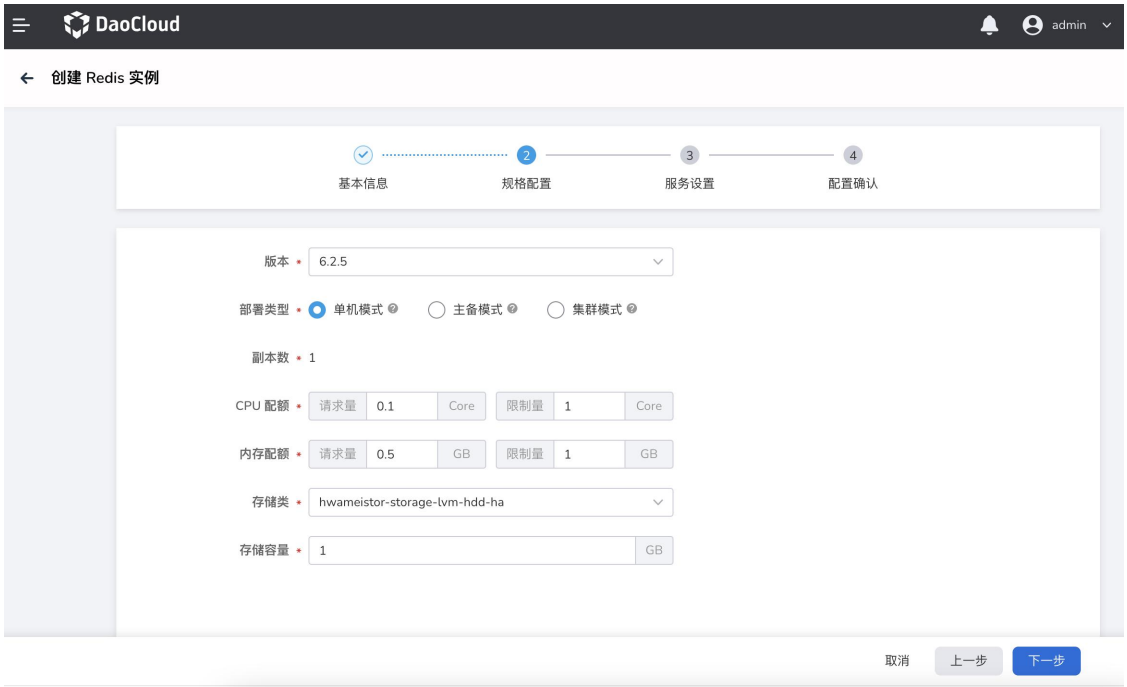
基本信息

2. 在创建 Redis 实例页面中，配置基本信息后，点击下一步。



基本信息

3. 选择部署类型、CPU、内存和存储等规格配置后，点击下一步。



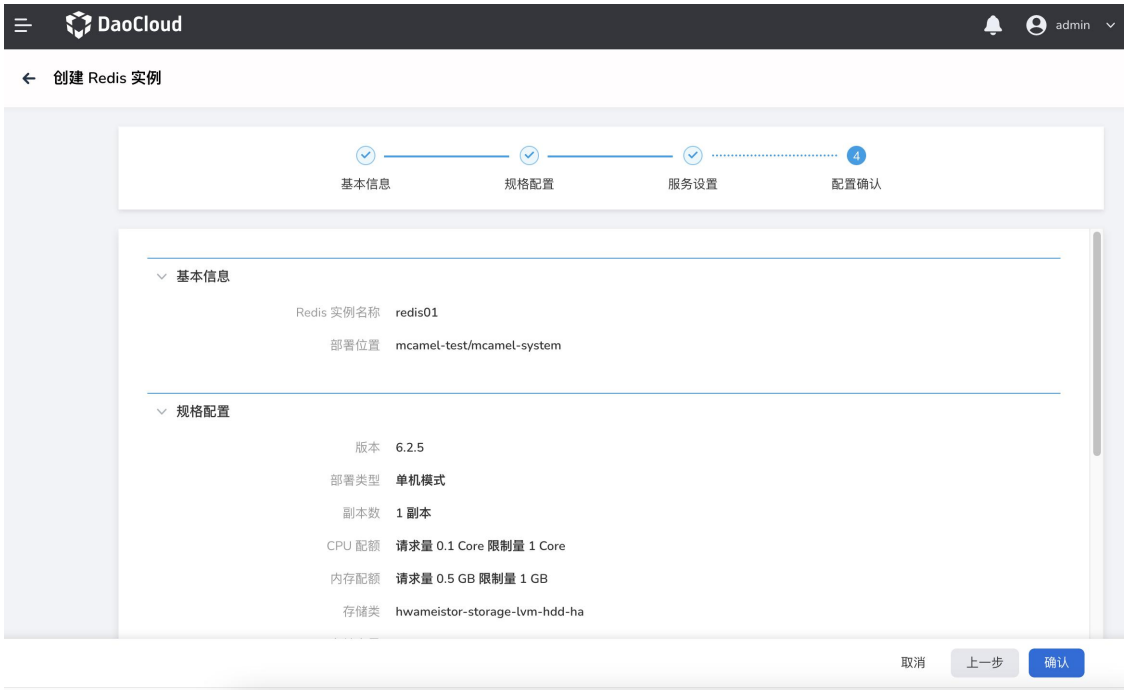
规格配置

4. 设置用户名和密码等服务设置，默认采用 ClusterIP 作为访问方式。



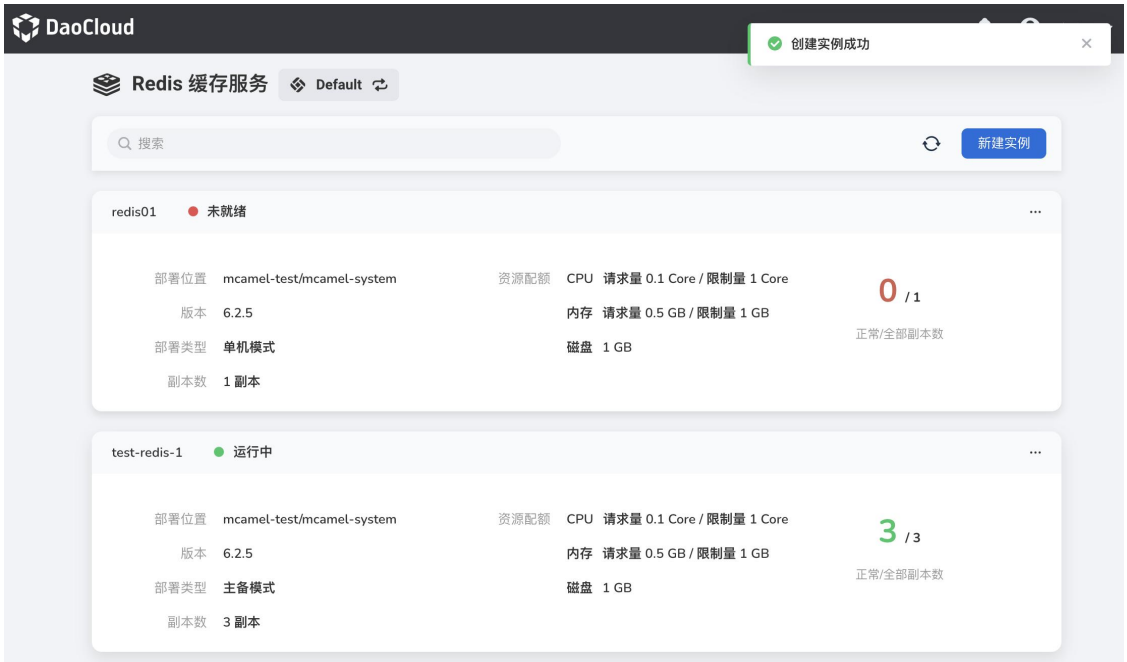
服务设置

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击确认。



确认

6. 返回实例列表，屏幕将提示创建实例成功。新创建的实例状态为未就绪，等待片刻后将变为运行中。

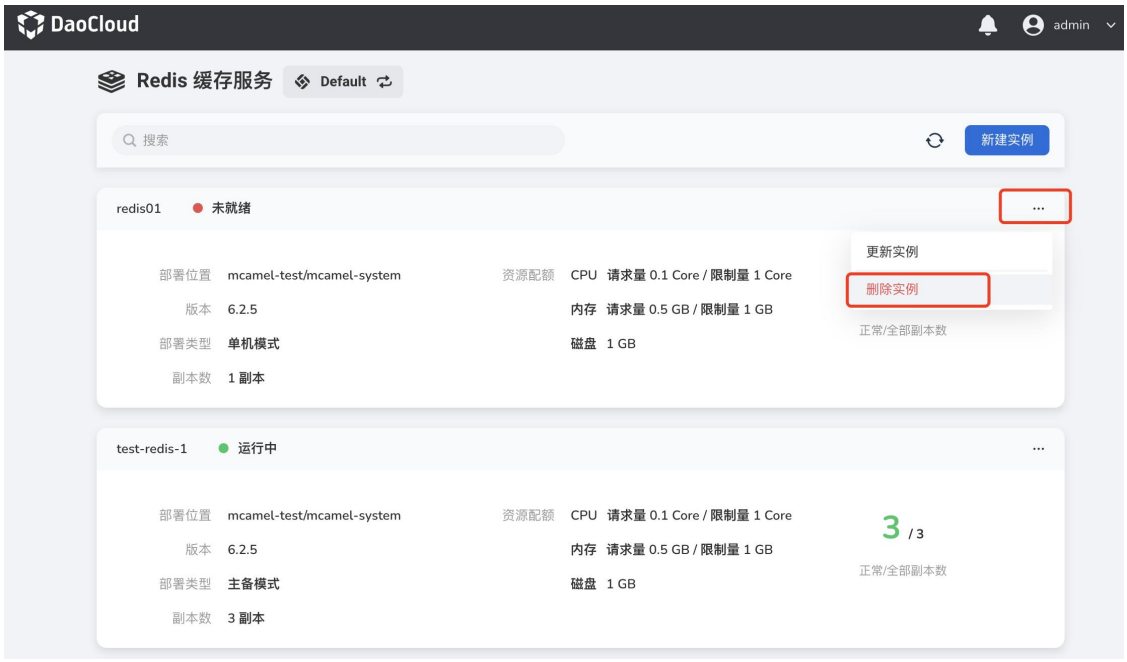


成功创建

删除 Redis 实例

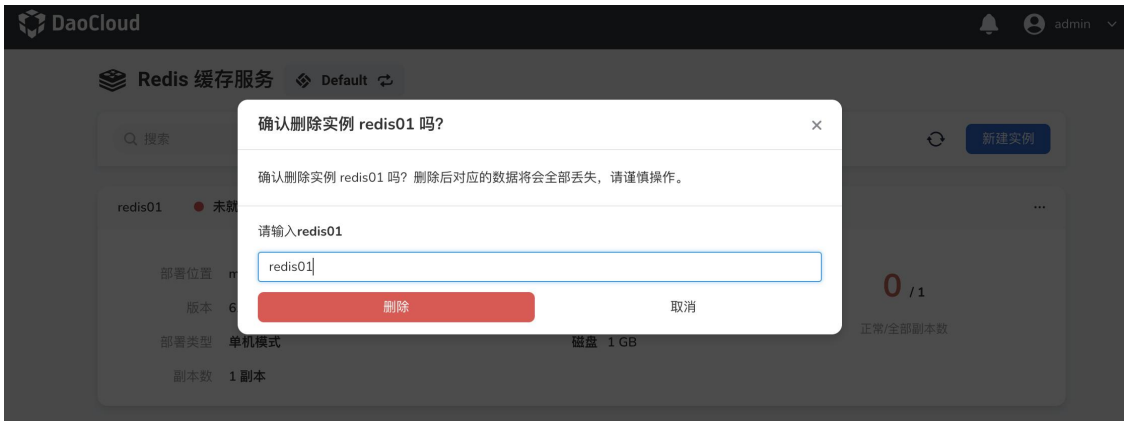
如果想要删除一个 Redis 实例，可以执行如下操作：

1. 在 Redis 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



选择删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。



点击删除

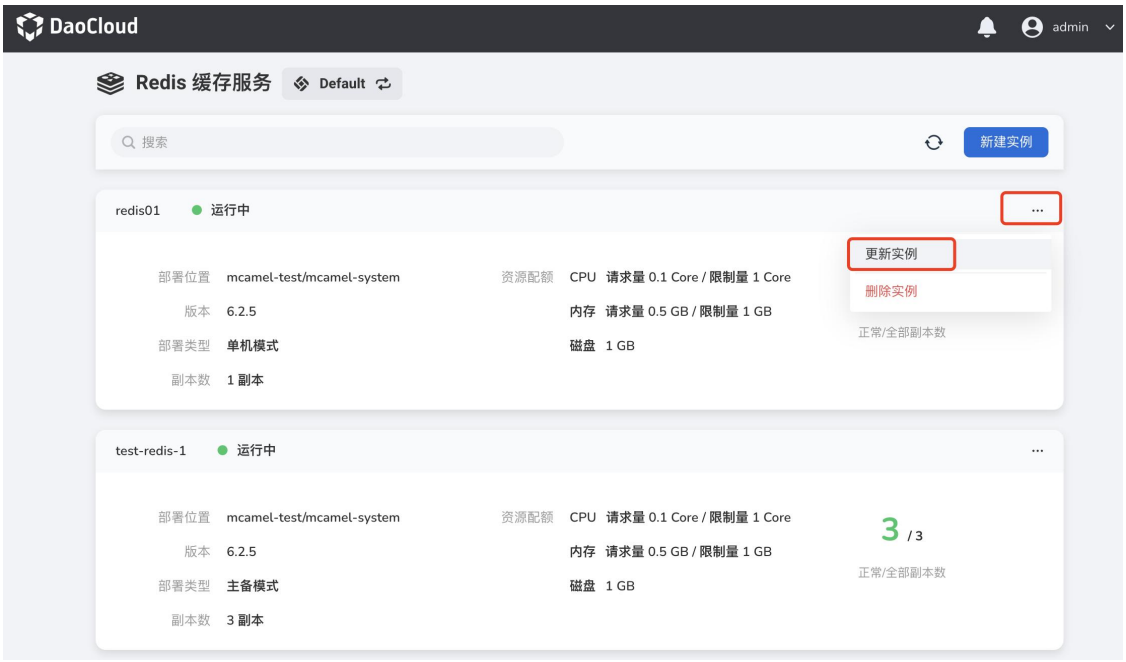
!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

更新 Redis 实例

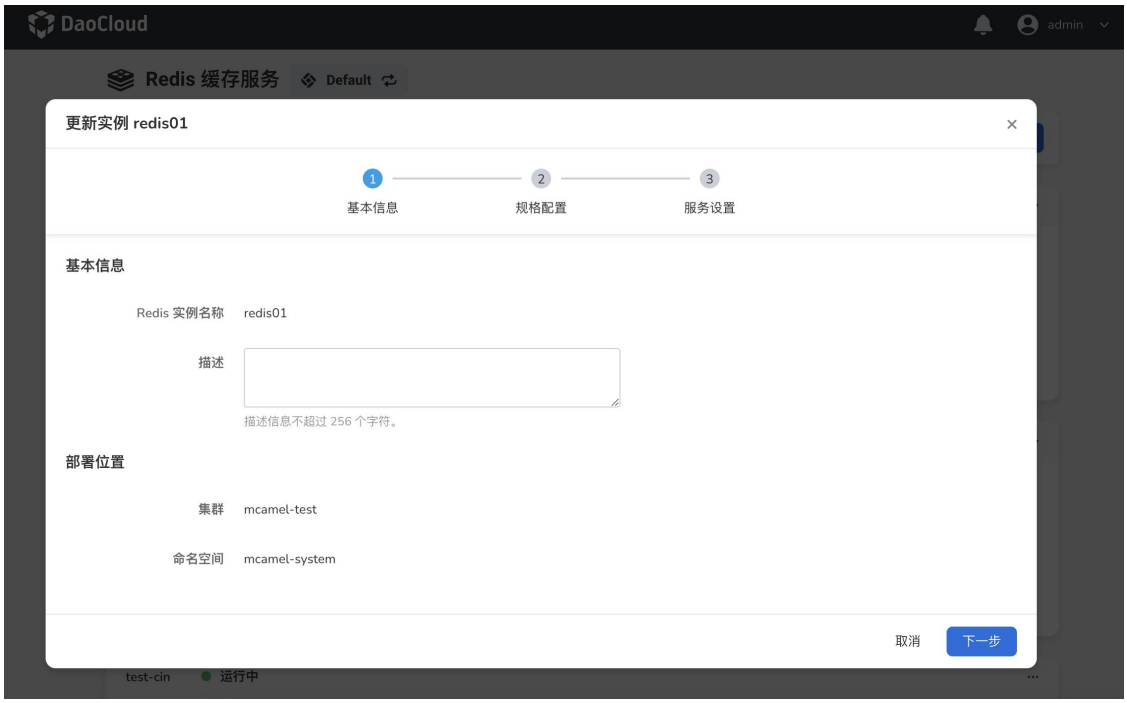
如果想要更新或修改 Redis 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



更新实例

2. 基本信息：只能修改描述。然后点击下一步。



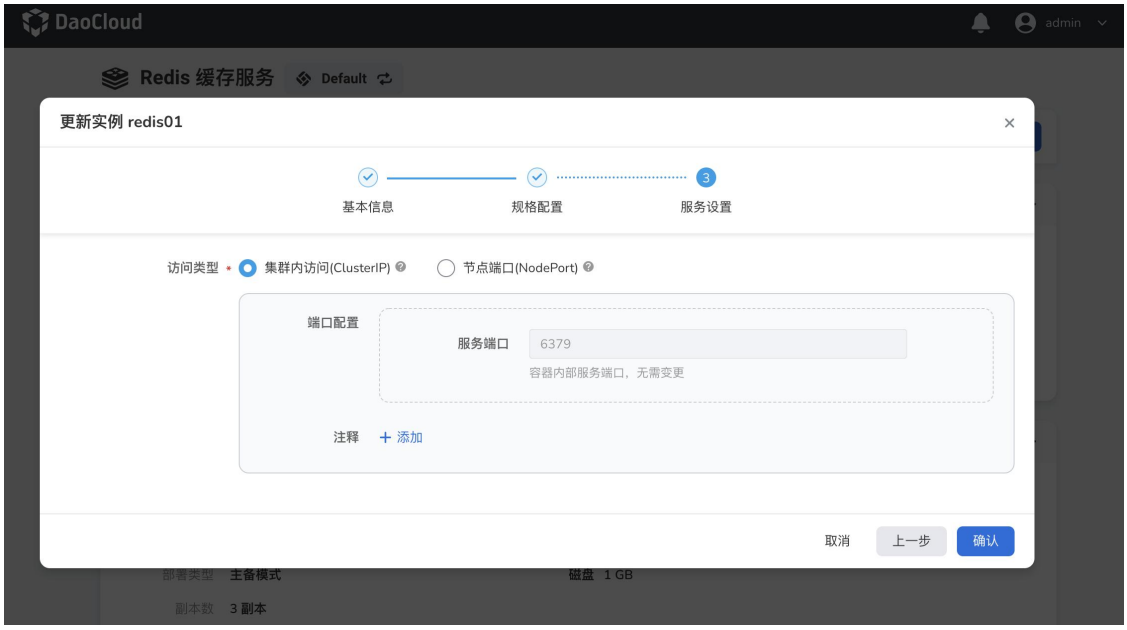
基本信息

3. 修改规格配置后点击下一步。



规格配置

4. 修改服务设置后点击确认。



服务设置

跨集群数据同步

Redis 作为开源的内存数据存储系统，具备出色的性能、高可用性和实时性能，可满足应用和服务的大量访问请求以及数据存储需求。然而，由于缺乏内置的数据安全保护功能，因此在实际应用中，需要采取一定的容灾技术来保障 Redis 数据的可靠性。

容灾技术主要通过数据复制和数据备份等手段实现，以提供冷备和热备两种备份方式。在网络、服务器或数据库发生故障导致数据丢失时，通过主备切换和数据恢复等方式，确保企业数据的安全性，确保业务的连续性。

Redis-Shake 是一个用于合并、同步和迁移 Redis 数据的工具，可以基于不同的 Redis 模式，提供相应的数据持续同步方案，本文将介绍不同模式间的数据同步与数据恢复配置方法。

集群模式 to 集群模式

哨兵模式 to 哨兵模式

集群模式 to 哨兵模式

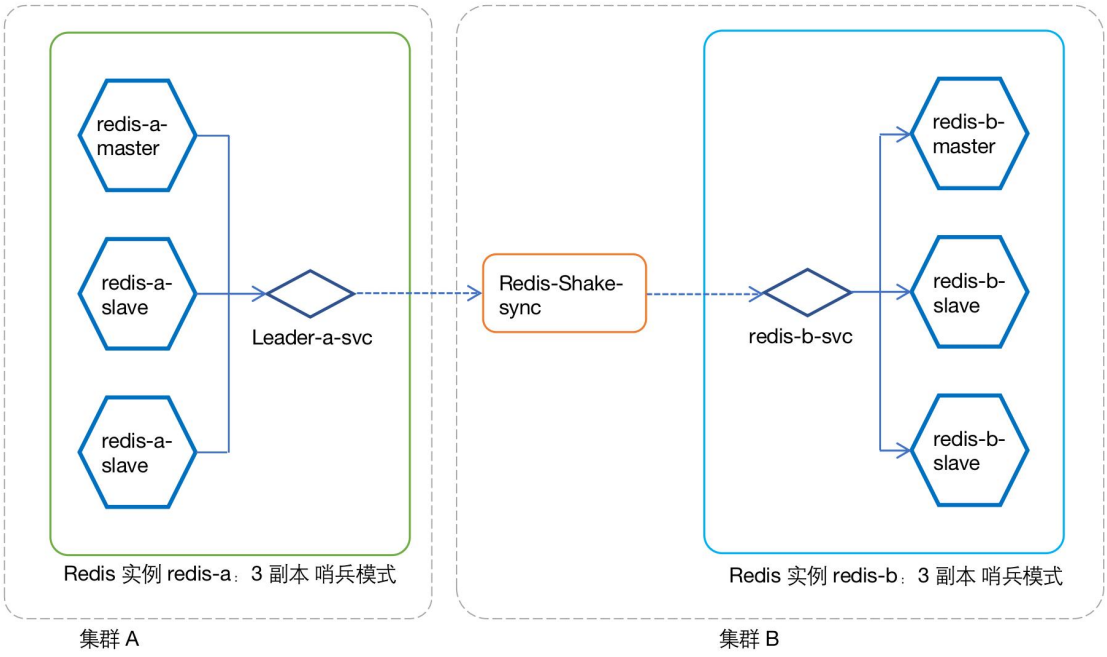
哨兵模式 to 哨兵模式

Redis-shake 支持哨兵模式实例间的数据同步与迁移能力，现以 3 副本哨兵模式 Redis 为例，演示跨集群同步配置方法。

假设 Redis 实例 A 与 Redis 实例 B 处于不同集群，二者均为 3 副本哨兵模式，现将实例 A 作为主实例，实例 B 作为从实例搭建同步结构，提供以下灾备支持：

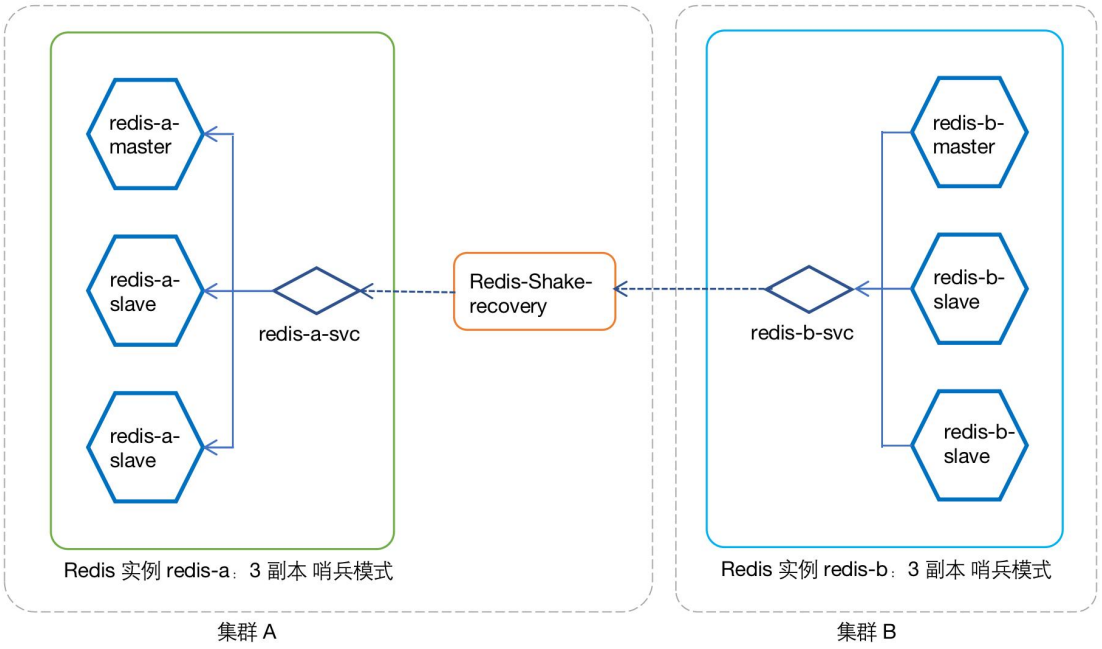
- 正常状态下，由实例 A 对外提供服务，并持续同步数据 实例 A >> 实例 B ；
- 当主实例 A 故障离线后，由实例 B 对外提供服务；
- 待实例 A 恢复上线后， 实例 B >> 实例 A 回写增量数据；
- 实例 A 数据恢复完成后，切换为初始状态，即由实例 A 提供服务，并向实例 B 持续数据同步。

图例：数据同步 实例 A >> 实例 B



sync

图例：数据恢复 实例 B >> 实例 A



recovery

!!! note

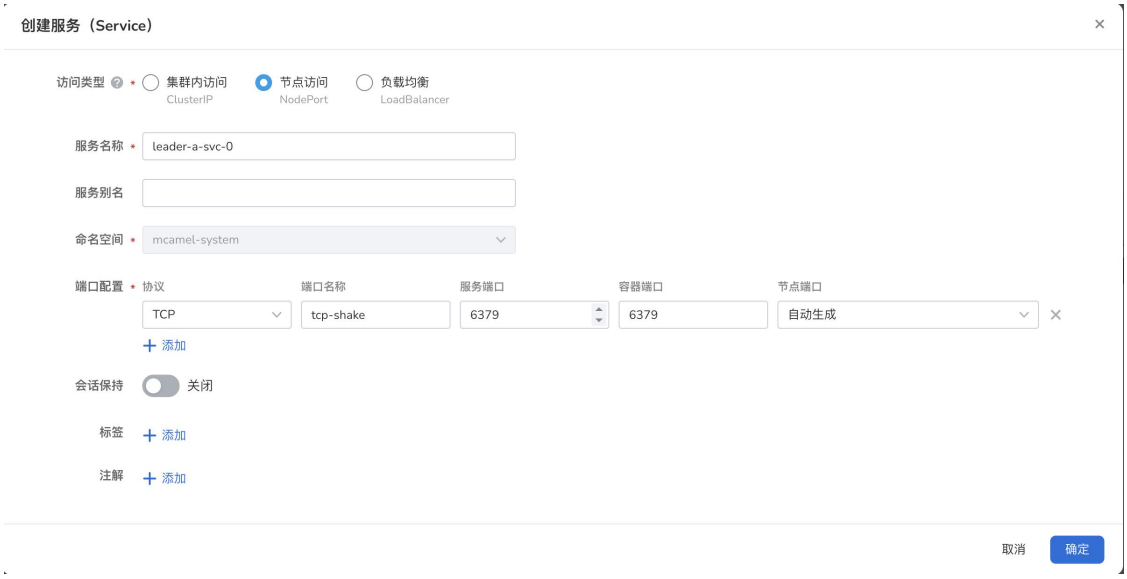
数据同步与数据恢复需要通过不同的 Redis-shake 组完成，因此在本例中需要分别创建 `Redis-shake-sync`、`Redis-shake-recovery` 两个 Redis-shake 实例。

数据同步部署

为源端实例配置服务

为 Redis 实例创建一个 Nodeport 服务，用于 Redis-Shake 的数据同步访问。本例中需要为实例 A 创建 1 个服务，以下为创建过程 ：

1. 进入容器管理 -{源端集群}- 有状态工作负载：选择工作负载 redis-a，创建一个 Nodeport 服务，容器端口和服务端口均为 6379



2. 更新该服务。并确定工作负载选择器包含以下标签

```
app.kubernetes.io/component: redis
app.kubernetes.io/managed-by: redis-operator
app.kubernetes.io/name: redis-a
app.kubernetes.io/part-of: redis-failover
redis-failover: master
redisfailovers.databases.spotahome.com/name: redis-a
```



SVC

Redis-shake 部署

Redis-shake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在目标端部署 redis-shake，配置方式如下。

1. 创建配置项

在容器管理 -{目标端集群}- 配置与存储 - 配置项为 Redis-shake 实例创建配置项 redis-sync。导入文件 sync.toml （文件内容见附录），并注意需要修改以下内容：



conf

- source.address：源端 redis-a 的服务地址：

  address = "10.233.109.145:6379"

- 源端实例的访问密码：可在 Redis 实例的概览页获取该信息：

  password = "3wPxzWffdn" *# keep empty if no authentication is required*



conf

- 目标端实例访问地址，此处为目标端 redis-b 的默认 clusterIP 服务的地址：

```
address = "172.30.120.202:32283"
```

- 目标端实例的访问密码，可在数据服务模块下的 Redis 实例概览页获取该信息:

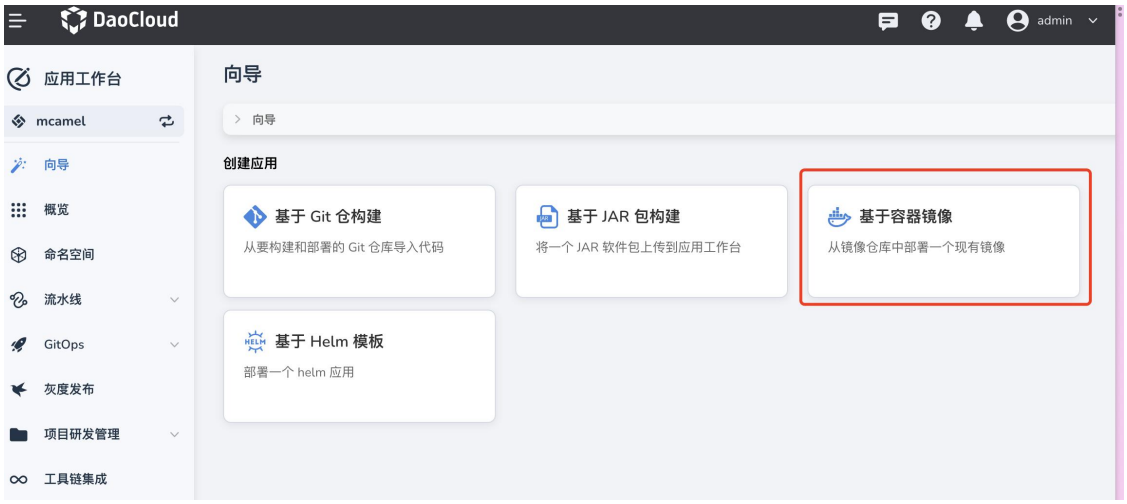
```
password = "3wPxzWffdn" # keep empty if no authentication is required
```

- 目标端类型需设置为 standalone：

```
[target]
type = "standalone" # "standalone" or "cluster"
```

2. 创建 Redis-shake

1. 打开应用工作台，选择向导–基于容器镜像，创建一个应用 Redis-shake-sync：



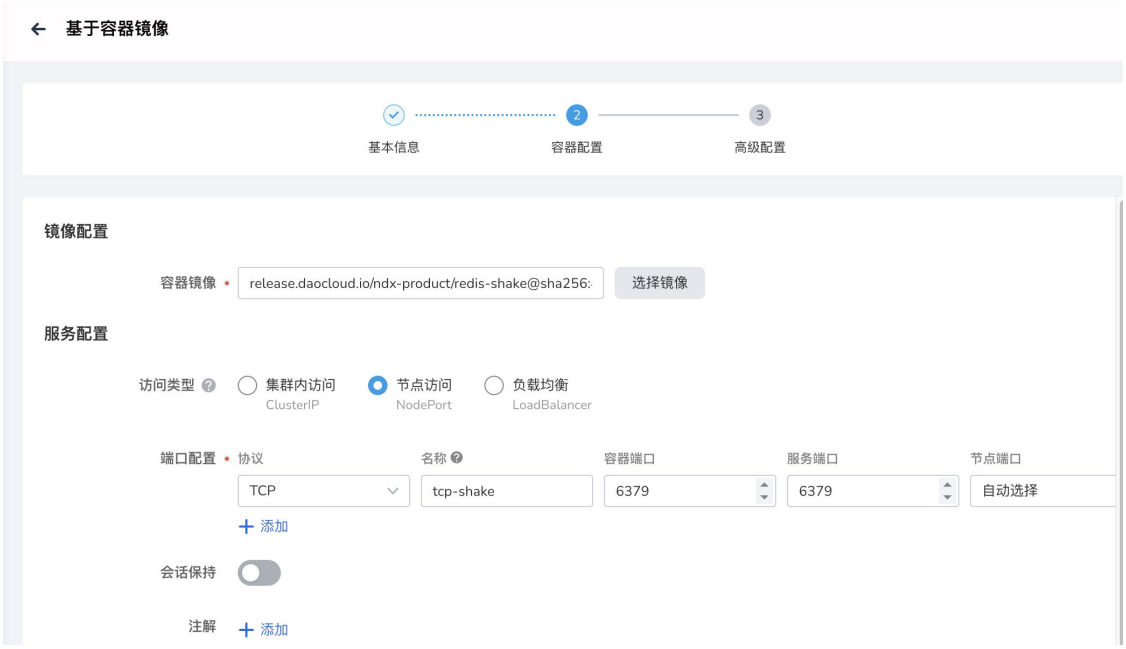
sync

2. 参考如下说明填写应用配置。

   - 应用所属集群、命名空间需与 Redis 实例一致；
   - 镜像地址：

     release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc1e211fb9cb894c4dc85e94287395a32fc3dc

- 默认服务的访问类型为 Nodeport，容器端口和服务端口设置为 6379。



sync

- 高级设置 – 生命周期 – 启动命令 – 运行参数填入：

/etc/sync/sync.toml



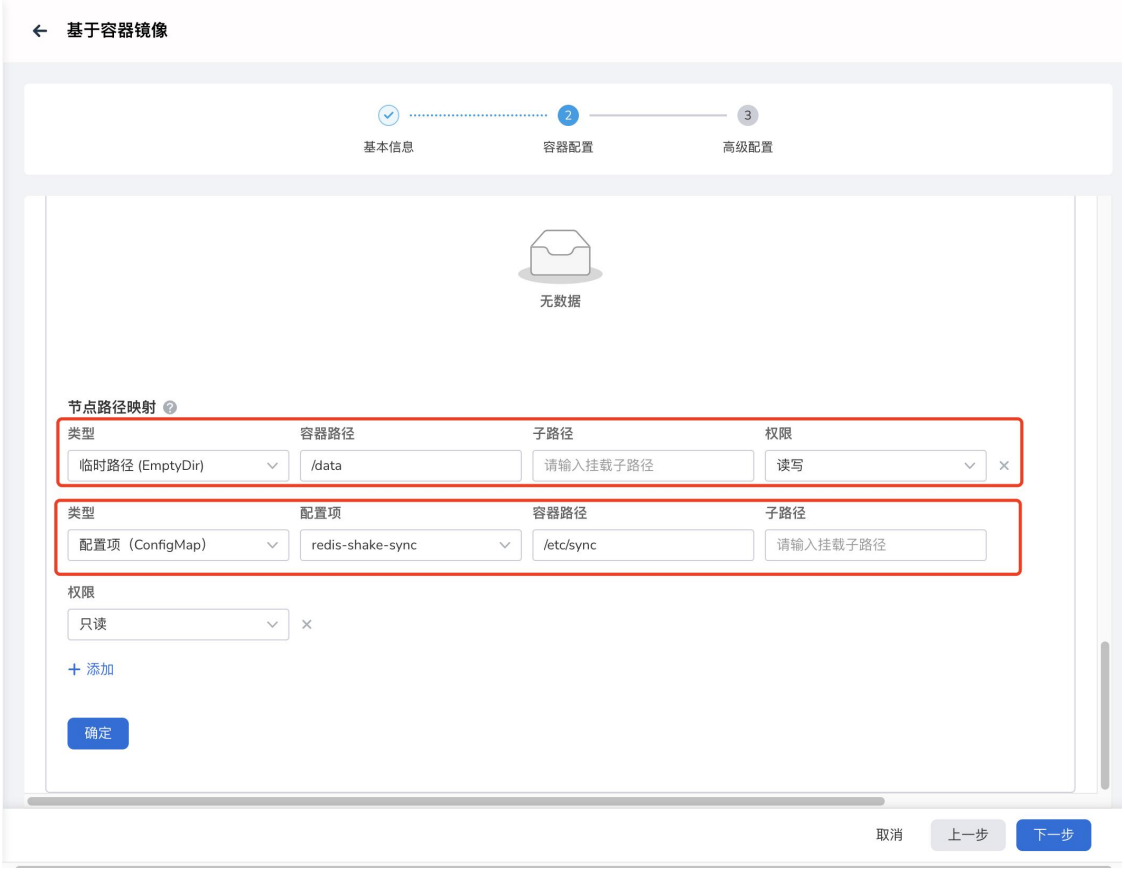
sync

- 高级设置 – 数据存储：添加配置项 redis-sync，路径必须设置为：

  /etc/sync

- 高级设置 – 数据存储：添加一个临时路径，容器路径必须为：

  /data



sync

3. 点击确定，完成 Redis-shake 创建。

完成 Redis-shake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 redis-cli 工具验证同步，这里就不做赘述。

数据恢复

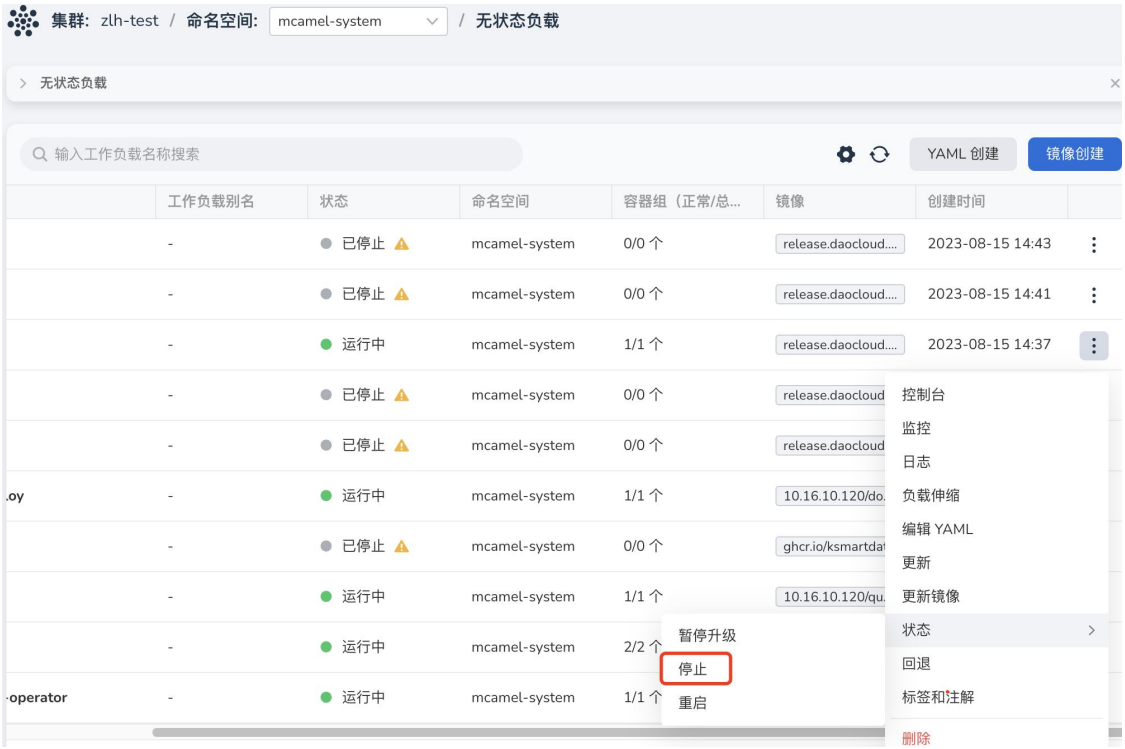
当源端实例恢复上线后，首先需要从目标端实例恢复增量数据，因此需要在 **源端** 再次部署 1 个 redis-shake 实例，实现 实例 **B >> 实例 A** 的数据回传，此处配置方法与数据同步过程类似，执行 **反方向** 配置部署即可，完成 redis-shake 创建后，即自动开始数据恢复。

!!! note

源端实例上线前，请先关闭用于正常同步的 `Redis-shake-sync`，避免发生错误的数据同步覆盖掉新增数据。

复原主从关系

如需复原初始的主从同步关系 **实例 A >> 实例 B**，需在容器管理中停止源端集群中的 3 个 Redis-shake-recovery 实例，重新启动目标端集群中的 3 个 Redis-shake-sync 实例，即可重建初始主从关系。 如需复原初始的主从同步关系 实例 **A >> 实例 B**，需在容器管理中暂停源端集群中的 Redis-shake-recovery 实例，重新启动目标端集群中的 Redis-shake-sync 实例，即可重建初始主从关系。



sync

附录

sync.toml

```
type = "sync"

[source]
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
address = "10.233.109.145:6379"
username = "" # keep empty if not using ACL
password = "3wPxzWffdn" # keep empty if no authentication is required
tls = false
elasticache_psync = "" # using when source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373

[target]
type = "standalone" # "standalone" or "cluster"
```

```
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
# When the target is a cluster, write the address of one of the nodes.
# redis-shake will obtain other nodes through the `cluster nodes` command.
address = "10.233.103.2:6379"
username = "" # keep empty if not using ACL
password = "Aa123456" # keep empty if no authentication is required
tls = false

[advanced]
dir = "data"

# runtime.GOMAXPROCS, 0 means use runtime.NumCPU() cpu cores
ncpu = 4

# pprof port, 0 means disable
pprof_port = 0

# metric port, 0 means disable
metrics_port = 0

# log
log_file = "redis-shake.log"
log_level = "info" # debug, info or warn
log_interval = 5 # in seconds

# redis-shake gets key and value from rdb file, and uses RESTORE command t
o
# create the key in target redis. Redis RESTORE will return a "Target key nam
e
# is busy" error when key already exists. You can use this configuration item
# to change the default behavior of restore:
```

```
# panic:   redis-shake will stop when meet "Target key name is busy" error.
# rewrite: redis-shake will replace the key with new value.
# ignore:  redis-shake will skip restore the key when meet "Target key name is
  busy" error.
rdb_restore_command_behavior = "rewrite"  # panic, rewrite or skip

# pipeline
pipeline_count_limit = 1024

# Client query buffers accumulate new commands. They are limited to a fixed
# amount by default. This amount is normally 1gb.
target_redis_client_max_querybuf_len = 1024_000_000

# In the Redis protocol, bulk requests, that are, elements representing single
# strings, are normally limited to 512 mb.
target_redis_proto_max_bulk_len = 512_000_000
```

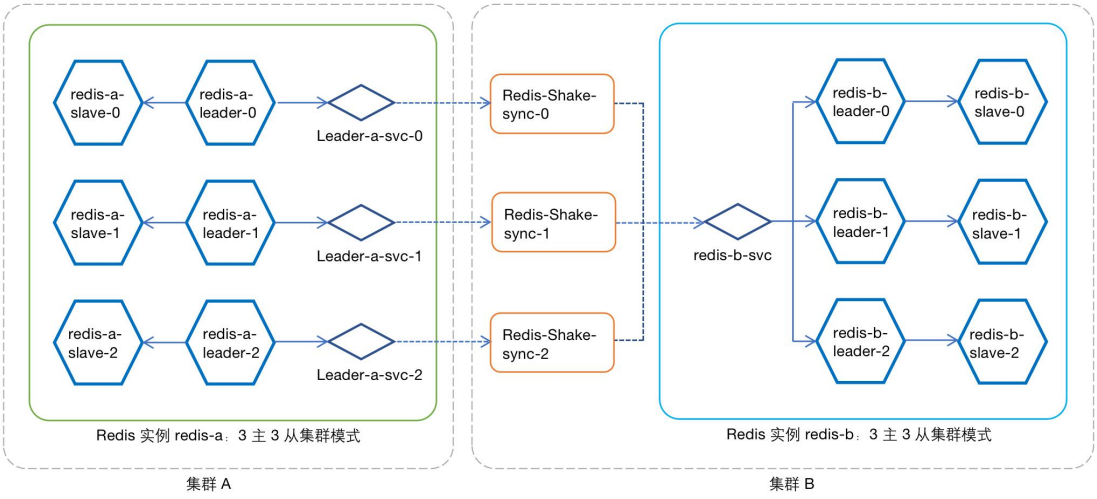
集群模式 to 集群模式

Redis-shake 支持集群模式实例间的数据同步与迁移能力，现以以下 3 主 3 从集群模式 Redis 场景为例，演示跨集群同步配置方法。

假设 Redis 实例 A 与 Redis 实例 B 处于不同集群，二者均为 3 主 3 从集群模式，现将实例 A 作为主实例，实例 B 作为从实例搭建同步结构，提供以下灾备支持：

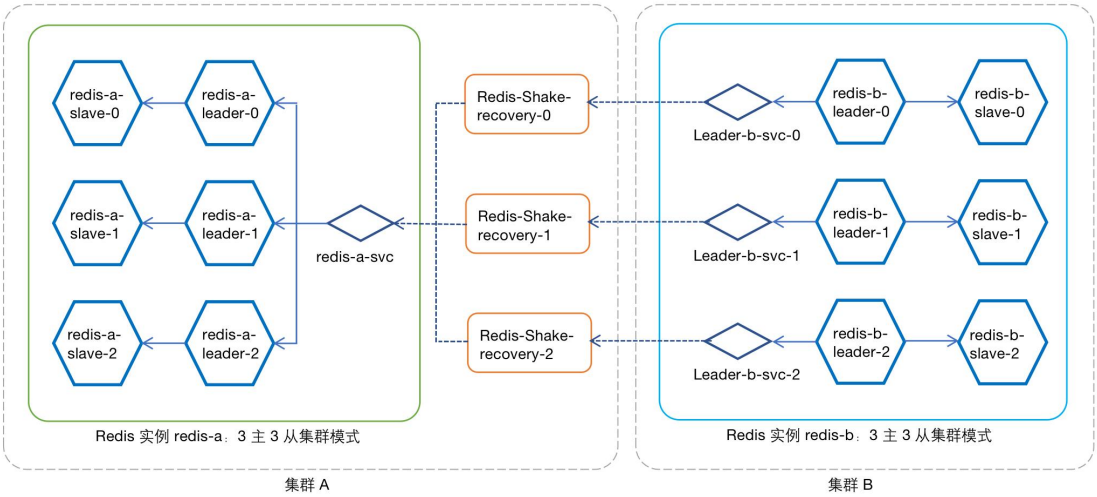
- 正常状态下，由实例 A 对外提供服务，并持续同步数据 实例 A >> 实例 B ；
- 当主实例 A 故障离线后，由实例 B 对外提供服务；
- 待实例 A 恢复上线后， 实例 B >> 实例 A 回写增量数据；
- 实例 A 数据恢复完成后，切换为初始状态，即由实例 A 提供服务，并向实例 B 持续数据同步。

图例：数据同步 实例 A >> 实例 B



sync

图例：数据恢复 实例 B >> 实例 A



recovery

!!! note

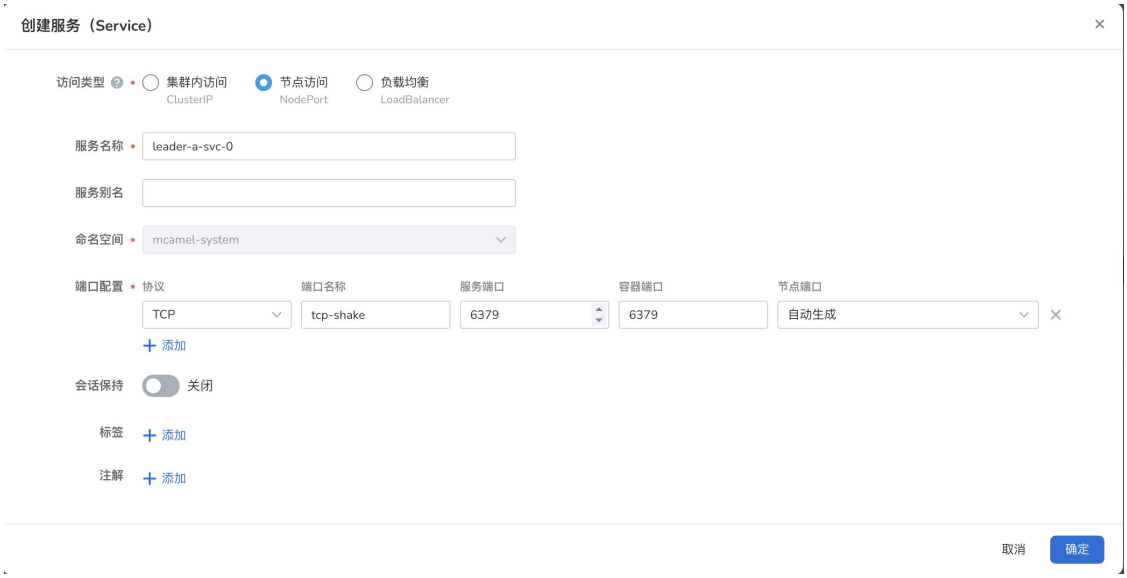
数据同步与数据恢复需要通过不同的 Redis-shake 组完成，因此在本例中实际要创建两组（Redis-shake-sync/Redis-shake-recovery）共 6 个 Redis-shake 实例。

数据同步部署

为源端实例配置服务

为 Redis 实例的每一个 Leader Pod 创建一个 Nodeport 服务，用于 Redis-Shake 的数据同步访问。本例中需要为实例 A 的 3 个 Leader Pod 分别创建服务，下面以 Pod redis-a-leader-0 为例，为其创建服务：

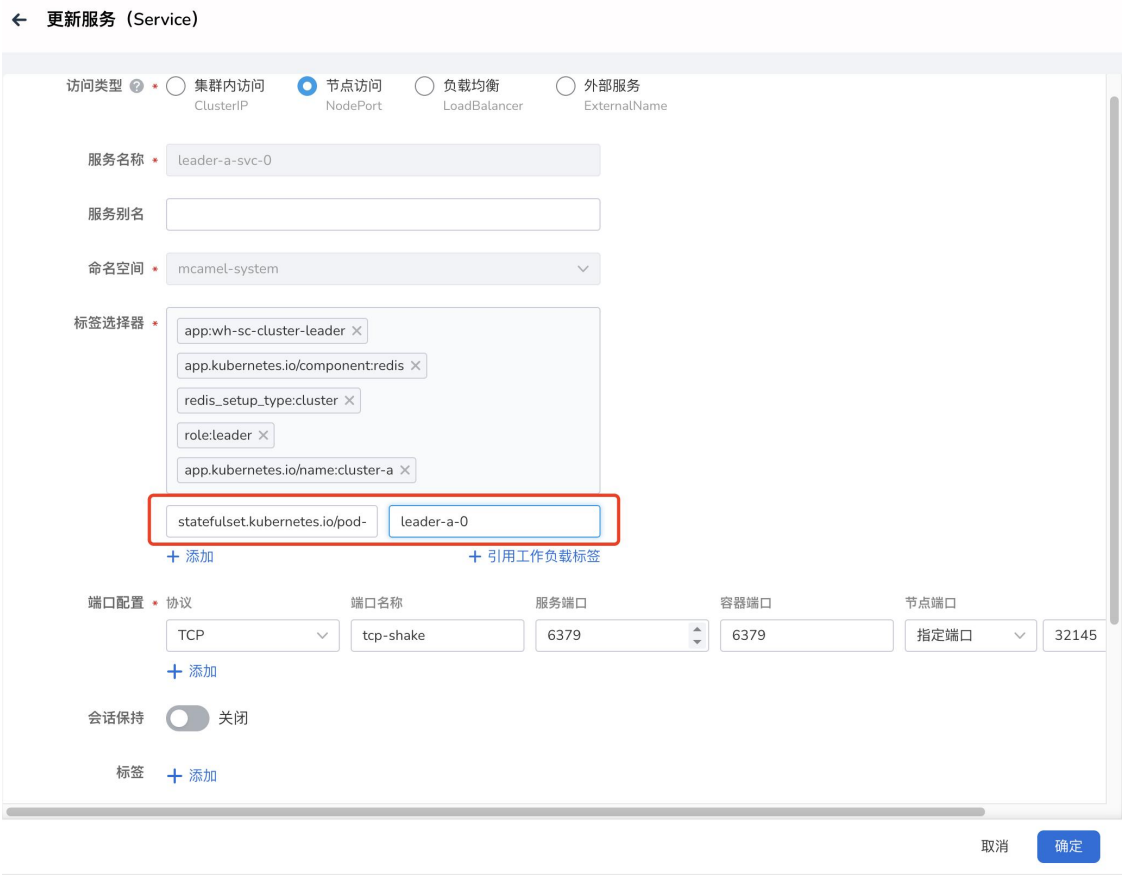
1. 进入容器管理 - 源端集群 - 有状态工作负载：选择工作负载 redis-a-leader，为其创建一个服务，命名为 redis-a-leader-svc-0，访问类型为 Nodeport，容器端口和服务端口均为 6379。



SVC

2. 更新该服务。并确定工作负载选择器包含以下标签

```
app.kubernetes.io/component: redis
app.kubernetes.io/managed-by: redis-operator
app.kubernetes.io/name: redis-a
app.kubernetes.io/part-of: redis-failover
role: leader
# 注意 pod-name 一定要选择正确的 leader pod 名称
statefulset.kubernetes.io/pod-name: redis-a-leader-0
```



svc

重试执行以上操作，为 redis-a-leader-1 redis-a-leader-2 分别创建服务。

Redis-shake 部署

Redis-shake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在目标端部署 redis-shake，配置方式如下。

在集群模式下，需要为源端 Redis 实例 cluster-a 的每一个 Leader Pod 部署独立的 Redis-shake。以 redis-a-leader-0 为例，创建 Redis-shake-sync-0：

1. 创建配置项

在容器管理 –{目标端集群}– 配置与存储 – 配置项为 Redis-shake 实例创建配置项 redis-sync-0。导入文件 sync.toml（文件内容见附录），并注意需要修改以下内容：



conf

- source.address：源端 redis-a-leader-0 的 redis-a-leader-svc-0 服务地址：

  address = "10.233.109.145:6379"

- 源端实例的访问密码：可在【中间件】实例的概览页获取该信息：

  password = "3wPxzWffdn" *# keep empty if no authentication is required*

- 目标端实例访问地址，此处为目标端 redis-b 的默认 clusterIP 服务 redis-b-leader 的地址：

  address = "172.30.120.202:32283"

- 目标端实例的访问密码，可在数据服务模块下的 Redis 实例概览页获取该信息:

```
password = "3wPxzWffdn" # keep empty if no authentication is required
```



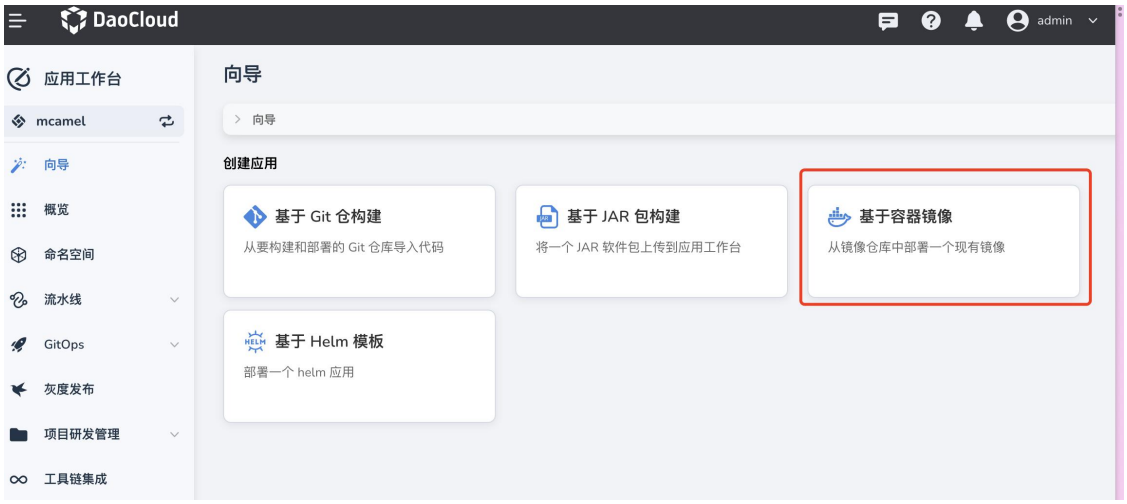
svc

- 目标端类型需设置为 cluster：

```
[target]
type = "cluster" # "standalone" or "cluster"
```

2. 创建 Redis-shake

1. 打开应用工作台，选择向导–基于容器镜像，创建一个应用 Redis–shake–sync–0：



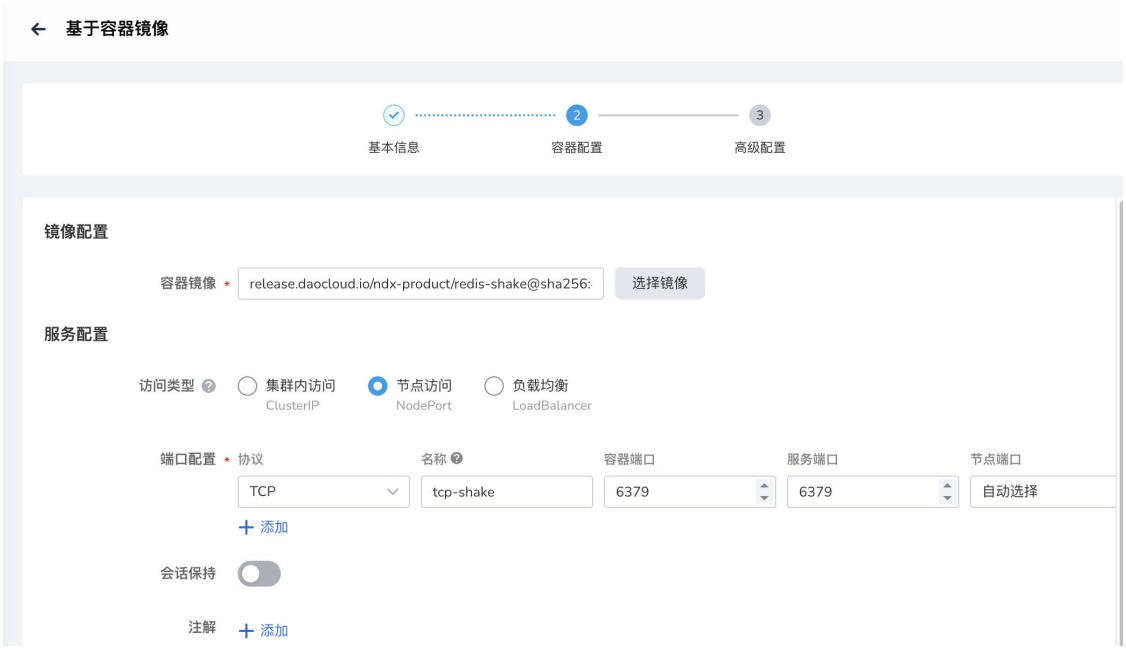
sync

2. 参考如下说明填写应用配置。

- 应用所属集群、命名空间需与 Redis 实例一致；
- 镜像地址：

  release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc1e211fb9cb894c4dc85e94287395a32fc3dc

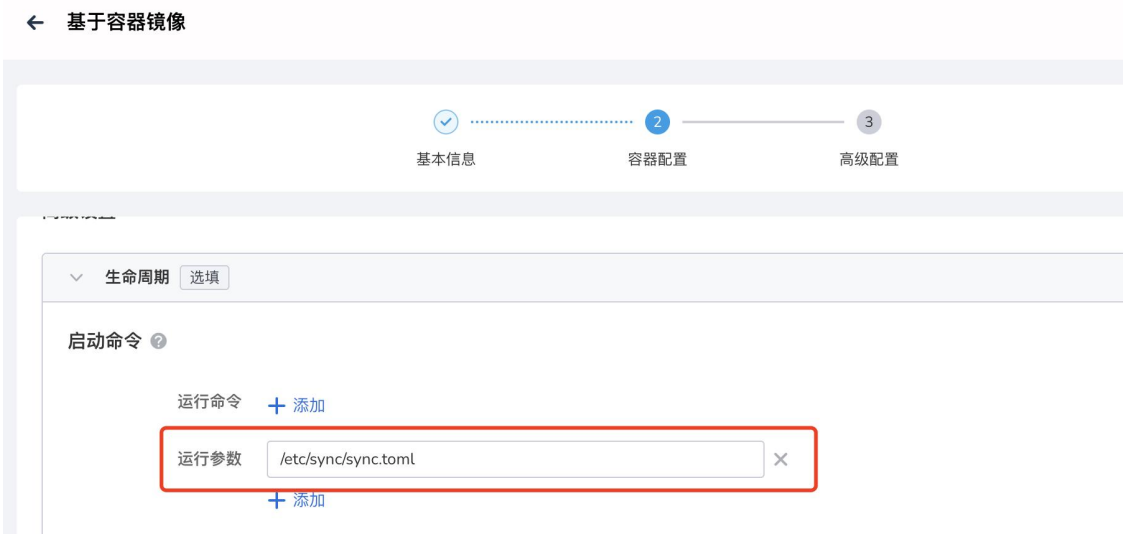
- 默认服务的访问类型为 Nodeport ，容器端口和服务端口设置为 6379。



sync

- 高级设置 – 生命周期 – 启动命令 – 运行参数 填入：

  /etc/sync/sync.toml



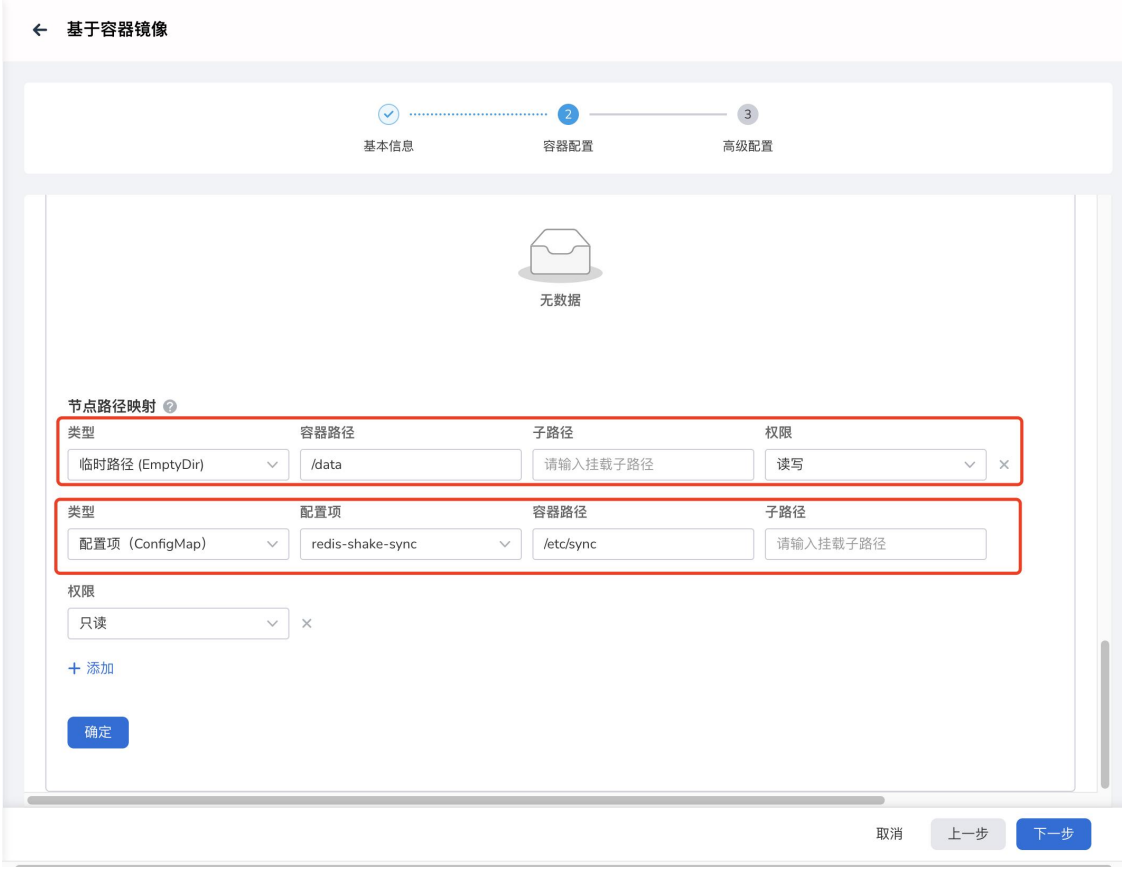
sync

- 高级设置 – 数据存储：添加配置项 redis-sync-0，路径必须设置为：

  /etc/sync

- 高级设置 –数据存储：添加一个临时路径，容器路径必须为：

  /data



sync

3. 点击确定，完成 Redis-shake 创建。

重复执行以上操作，分别为其他两个 Leader Pod 创建 Redis-shake-sync-1，Redis-shake-sync-2。

完成 Redis-shake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 redis-cli 工具验证同步，这里就不做赘述。

数据恢复

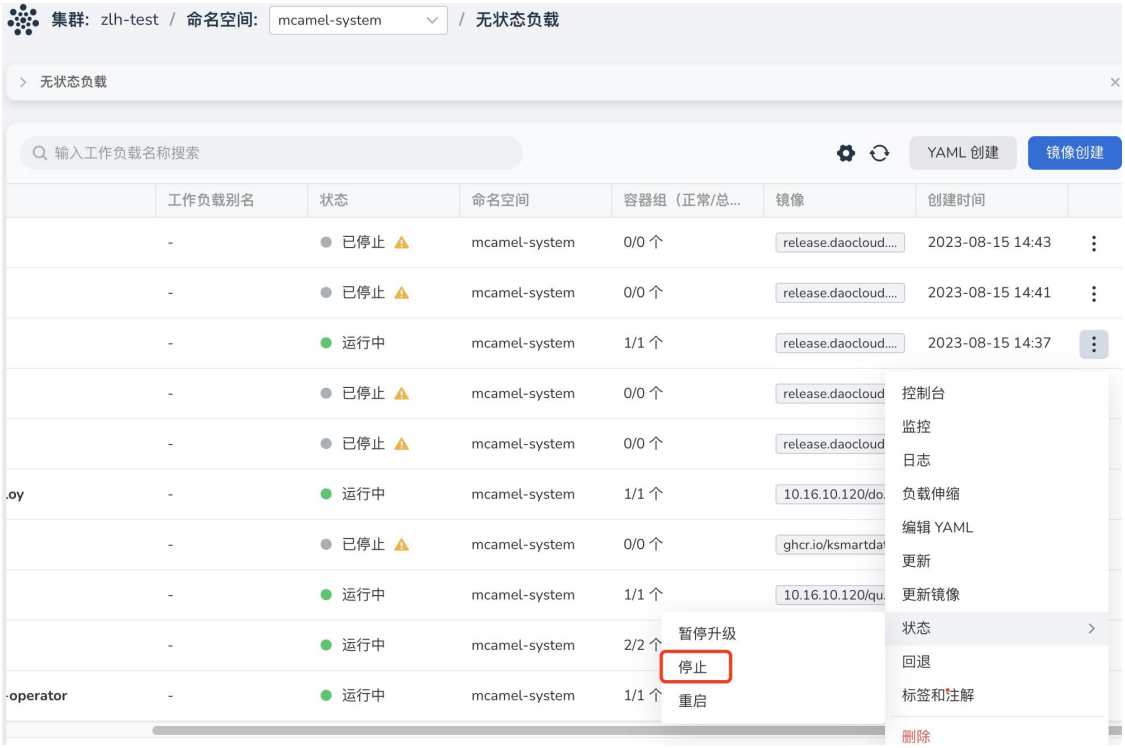
当源端实例恢复上线后，首先需要从目标端实例恢复增量数据，因此需要在 **源端集群** 再次部署 3 个 redis-shake 实例，实现 **实例 B >> 实例 A** 的数据回传，此处配置方法与数据同步过程类似，执行 **反方向** 配置部署即可，完成 redis-shake 创建后，即自动开始数据恢复。

!!! note

源端实例上线前，请先关闭运行在 **目标端集群** 的 3 个 Redis-shake 实例 `Redis-shake-sync`，避免发生错误的数据同步覆盖掉新增数据。

复原主从关系

如需复原初始的主从同步关系 **实例 A >> 实例 B**，需在容器管理中停止源端集群中的 3 个 Redis-shake-recovery 实例，重新启动目标端集群中的 3 个 Redis-shake-sync 实例，即可重建初始主从关系。



sync

附录

sync.toml

```
type = "sync"

[source]
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
```

```
address = "10.233.109.145:6379"
username = "" # keep empty if not using ACL
password = "3wPxzWffdn" # keep empty if no authentication is required
tls = false
elasticache_psync = "" # using when source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373

[target]
type = "cluster" # "standalone" or "cluster"
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
# When the target is a cluster, write the address of one of the nodes.
# redis-shake will obtain other nodes through the `cluster nodes` command.
address = "10.233.103.2:6379"
username = "" # keep empty if not using ACL
password = "Aa123456" # keep empty if no authentication is required
tls = false

[advanced]
dir = "data"

# runtime.GOMAXPROCS, 0 means use runtime.NumCPU() cpu cores
ncpu = 4

# pprof port, 0 means disable
pprof_port = 0

# metric port, 0 means disable
metrics_port = 0

# log
log_file = "redis-shake.log"
```

```
log_level = "info" # debug, info or warn
log_interval = 5 # in seconds

# redis-shake gets key and value from rdb file, and uses RESTORE command t
o
# create the key in target redis. Redis RESTORE will return a "Target key nam
e
# is busy" error when key already exists. You can use this configuration item
# to change the default behavior of restore:
# panic:   redis-shake will stop when meet "Target key name is busy" error.
# rewrite: redis-shake will replace the key with new value.
# ignore:  redis-shake will skip restore the key when meet "Target key name is
 busy" error.
rdb_restore_command_behavior = "rewrite" # panic, rewrite or skip

# pipeline
pipeline_count_limit = 1024

# Client query buffers accumulate new commands. They are limited to a fixed
# amount by default. This amount is normally 1gb.
target_redis_client_max_querybuf_len = 1024_000_000

# In the Redis protocol, bulk requests, that are, elements representing single
# strings, are normally limited to 512 mb.
target_redis_proto_max_bulk_len = 512_000_000
```

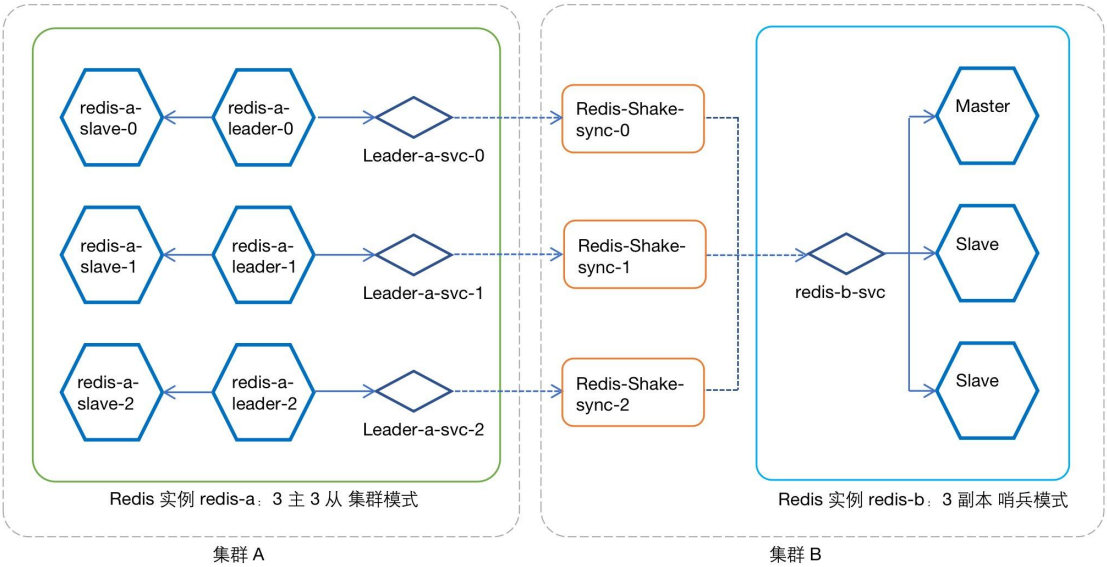
集群模式 to 哨兵模式

Redis-shake 支持不同部署模式实例间的数据同步与迁移能力，现以 3 主 3 从集群模式实例与 3 副本哨兵模式实例的场景为例，演示不同模式之间的同步配置方法。

假设 Redis 实例 A 为 3 主 3 从集群模式，实例 B 为 3 副本哨兵模式，两实例处于不同集群。现将实例 A 作为主实例，实例 B 作为从实例搭建同步结构，提供以下灾备支持：

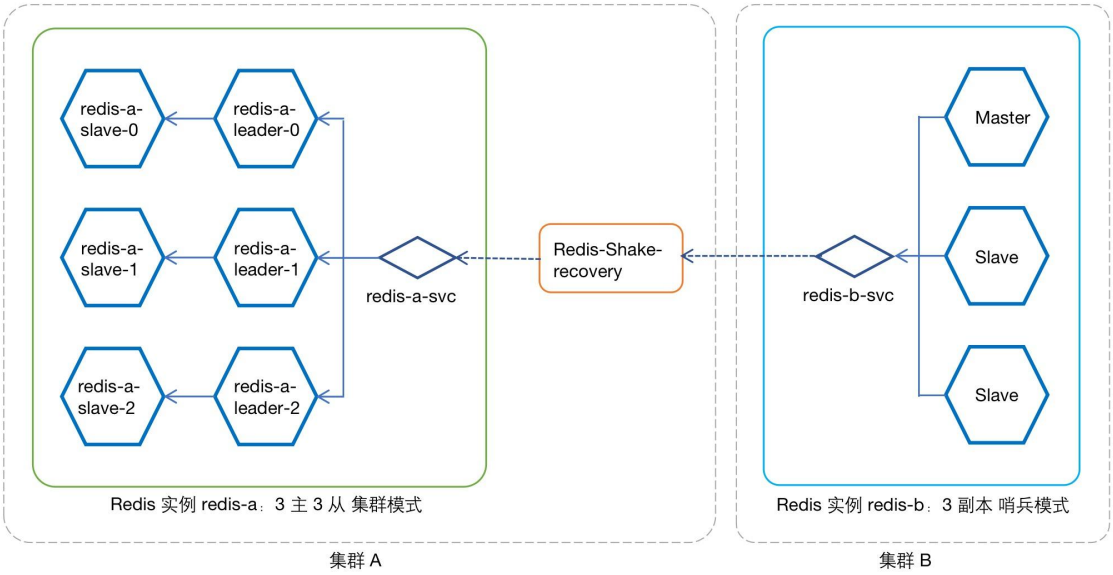
- 正常状态下，由实例 A 对外提供服务，并持续同步数据 实例 A >> 实例 B ；
- 当主实例 A 故障离线后，由实例 B 对外提供服务；
- 待实例 A 恢复上线后， 实例 B >> 实例 A 回写增量数据；
- 实例 A 数据恢复完成后，切换为初始状态，即由实例 A 提供服务，并向实例 B 持续数据同步。

图例：数据同步 实例 A >> 实例 B



sync

图例：数据恢复 实例 B >> 实例 A



recovery

!!! note

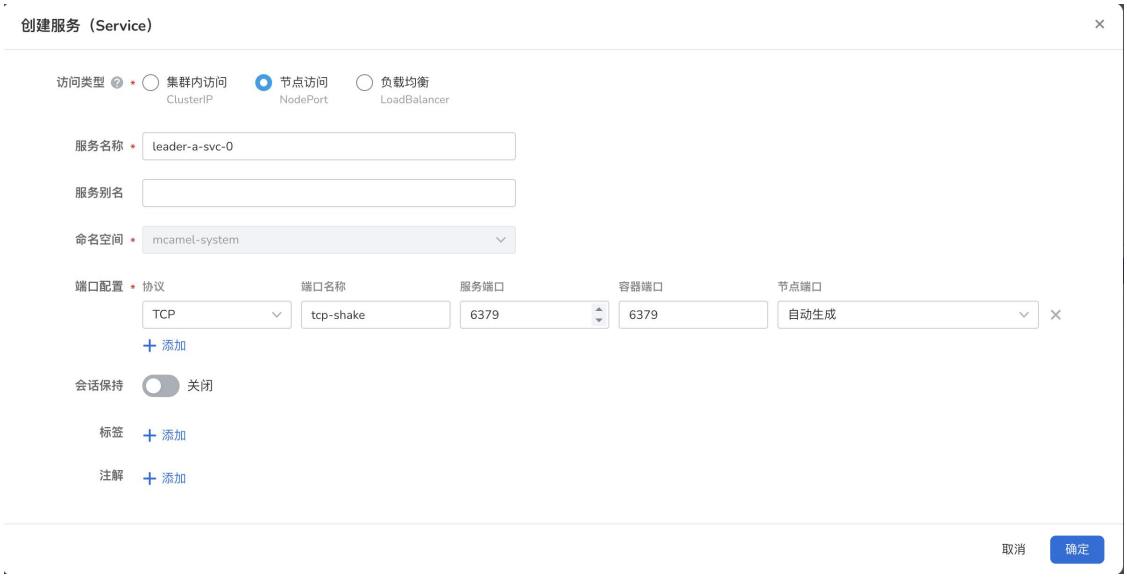
由上图可见，数据传输的方式与传输目标实例有关，在`数据同步`中目标实例 B 为哨兵模式，仅需部署一个 Redis-shake（Redis-shake-sync），在`数据恢复`中目标实例 A 为集群模式，则需要为每个 leader 节点分别部署一个 Redis-shake（Redis-shake-recovery）。

数据同步部署

为 实例 A 配置服务

为 Redis 实例的每一个 Leader Pod 创建一个 Nodeport 服务，用于 Redis-Shake 的数据同步访问。本例中需要为实例 A 的 3 个 Leader Pod 分别创建服务，下面以 Pod redis-a-leader-0 为例，为其创建服务：

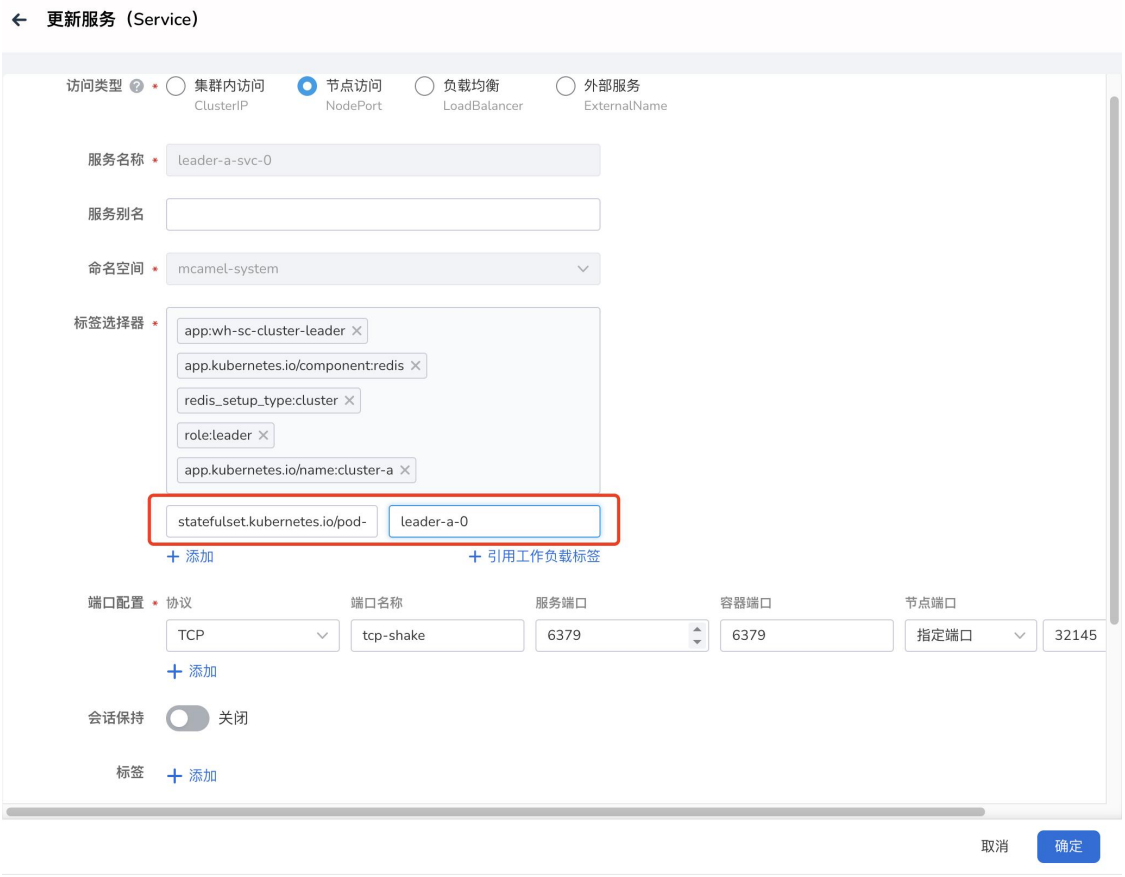
1. 进入容器管理 – 源端集群 – 有状态工作负载：选择工作负载 redis-a-leader，为其创建一个服务，命名为 redis-a-leader-svc-0，访问类型为 Nodeport，容器端口和服务端口均为 6379。



SVC

2. 更新该服务。并确定工作负载选择器包含以下标签

```
app.kubernetes.io/component: redis
app.kubernetes.io/managed-by: redis-operator
app.kubernetes.io/name: redis-a
app.kubernetes.io/part-of: redis-failover
role: leader
# 注意 pod-name 一定要选择正确的 leader pod 名称
statefulset.kubernetes.io/pod-name: redis-a-leader-0
```



svc

重试执行以上操作，为 redis-a-leader-1 redis-a-leader-2 分别创建服务。

Redis-shake 部署

Redis-shake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在目标端部署 redis-shake，配置方式如下。

1. 创建配置项

在容器管理 -{目标端集群}- 配置与存储 - 配置项为 Redis-shake 实例创建配置项 redis-sync。导入文件 sync.toml （文件内容见附录），并注意需要修改以下内容：



conf

- source.address：源端 redis-a 的服务地址：

  address = "10.233.109.145:6379"

- 源端实例的访问密码：可在 Redis 实例的概览页获取该信息：

  password = "3wPxzWffdn" *# keep empty if no authentication is required*



conf

- 目标端实例访问地址，此处为目标端 redis-b 的默认 clusterIP 服务的地址：

```
address = "172.30.120.202:32283"
```

- 目标端实例的访问密码，可在数据服务模块下的 Redis 实例概览页获取该信息:

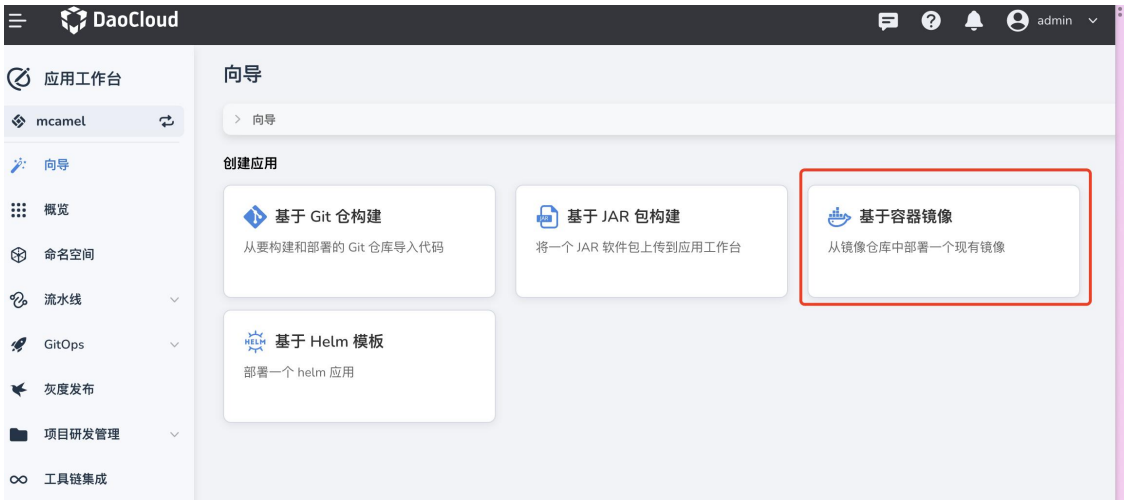
```
password = "3wPxzWffdn" # keep empty if no authentication is required
```

- 目标端类型需设置为 standalone：

```
[target]
type = "standalone" # "standalone" or "cluster"
```

2. 创建 Redis-shake

1. 打开应用工作台，选择向导–基于容器镜像，创建一个应用 Redis-shake-sync：



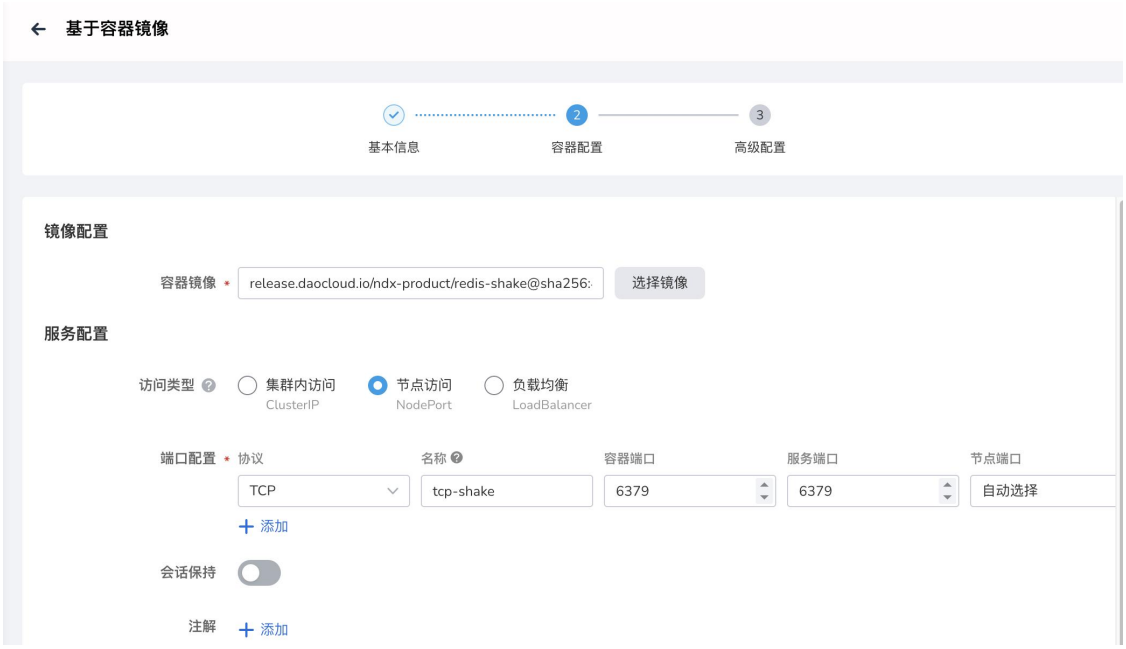
sync

2. 参考如下说明填写应用配置。

   - 应用所属集群、命名空间需与 Redis 实例一致；
   - 镜像地址：

     release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc1e211fb9cb894c4dc85e94287395a32fc3dc

- 默认服务的访问类型为 Nodeport，容器端口和服务端口设置为 6379。



sync

- 高级设置 – 生命周期 – 启动命令 – 运行参数填入：

/etc/sync/sync.toml



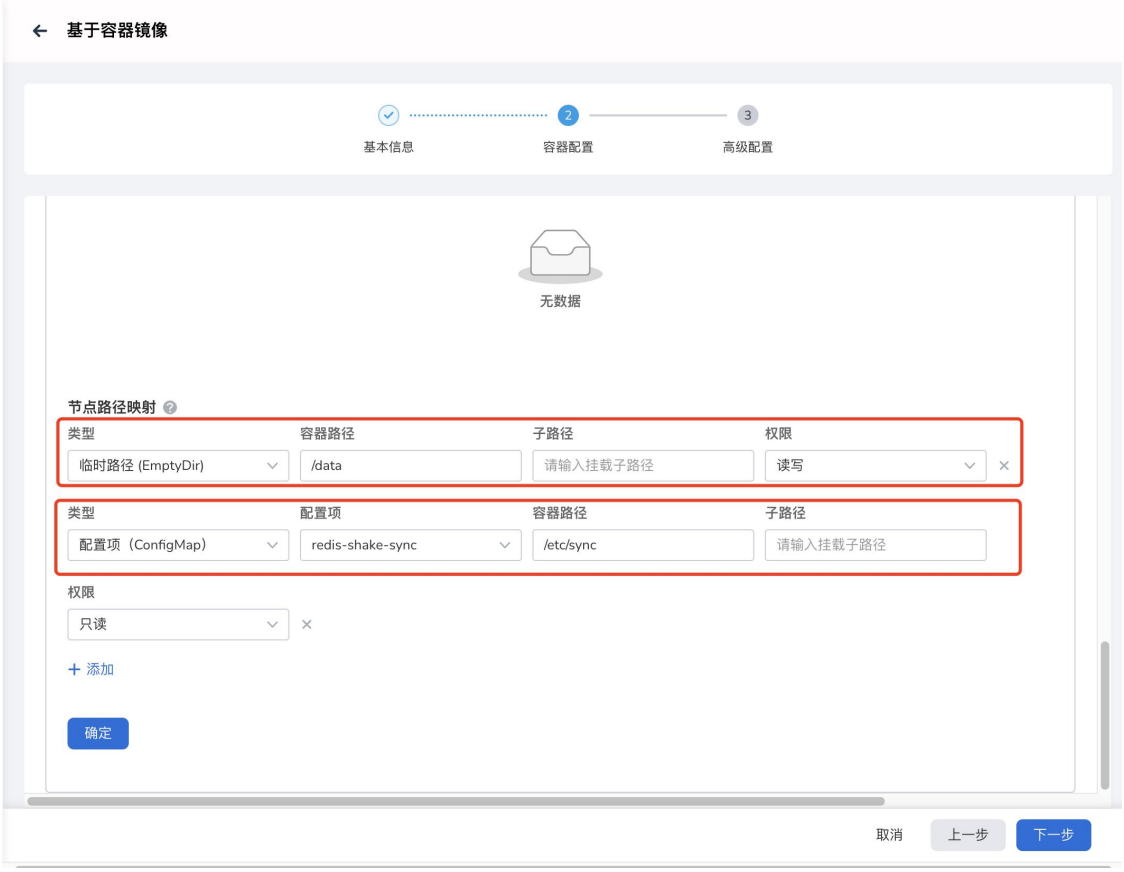
sync

- 高级设置 – 数据存储：添加配置项 redis-sync，路径必须设置为：

  /etc/sync

- 高级设置 – 数据存储：添加一个临时路径，容器路径必须为：

  /data



sync

3. 点击确定，完成 Redis-shake 创建。

完成 Redis-shake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 redis-cli 工具验证同步，这里就不做赘述。

数据恢复

当源端实例 **实例 A** 发生故障离线后，将由目标端 **实例 B** 提供服务，该过程必然产生新增数据。**实例 A** 恢复上线后，首先需要从 **实例 B** 恢复增量数据，因 **实例 A** 为集群模式，因此需要在 **源端集群** 部署 3 个 redis-shake 实例，实现 **实例 B >> 实例 A** 的数据回传，配置方式如下。

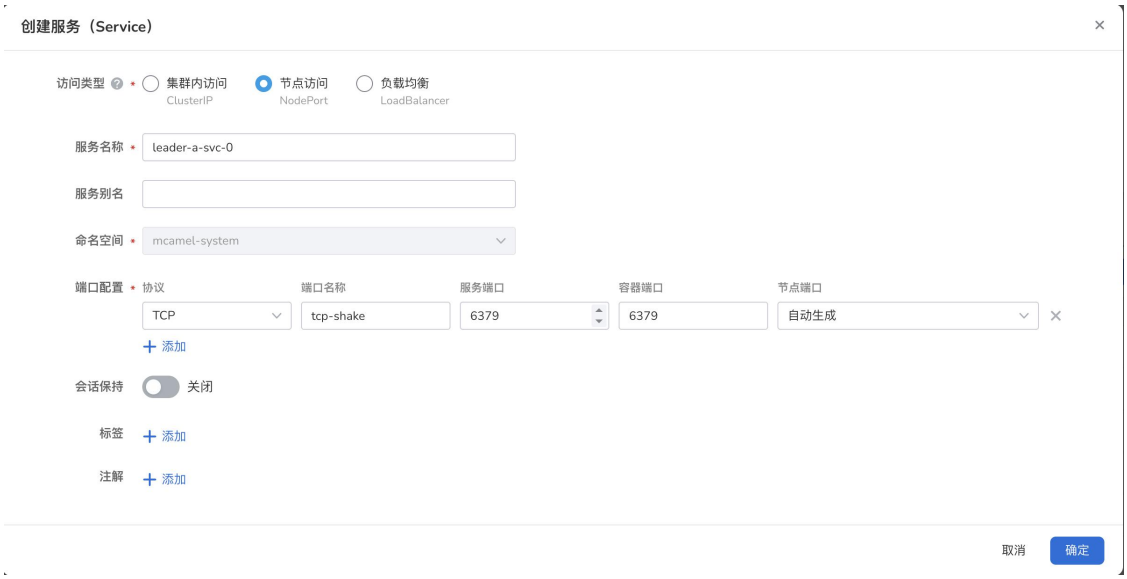
!!! note

源端实例上线前，请先关闭运行在 **目标端集群** 上运行的 Redis-shake 实例 `Redis-shake-sync`，避免发生错误的数据同步覆盖掉新增数据。

为 实例 B 配置服务

此时的数据源为 **实例 B**，为哨兵模式，因此仅需要创建 1 个服务，以下为创建过程 ：

1. 进入容器管理 – {源端集群} – 有状态工作负载：选择工作负载 redis-b，创建一个 Nodeport 服务，容器端口和服务端口均为 6379



svc

2. 更新该服务。并确定工作负载选择器包含以下标签

app.kubernetes.io/component: redis

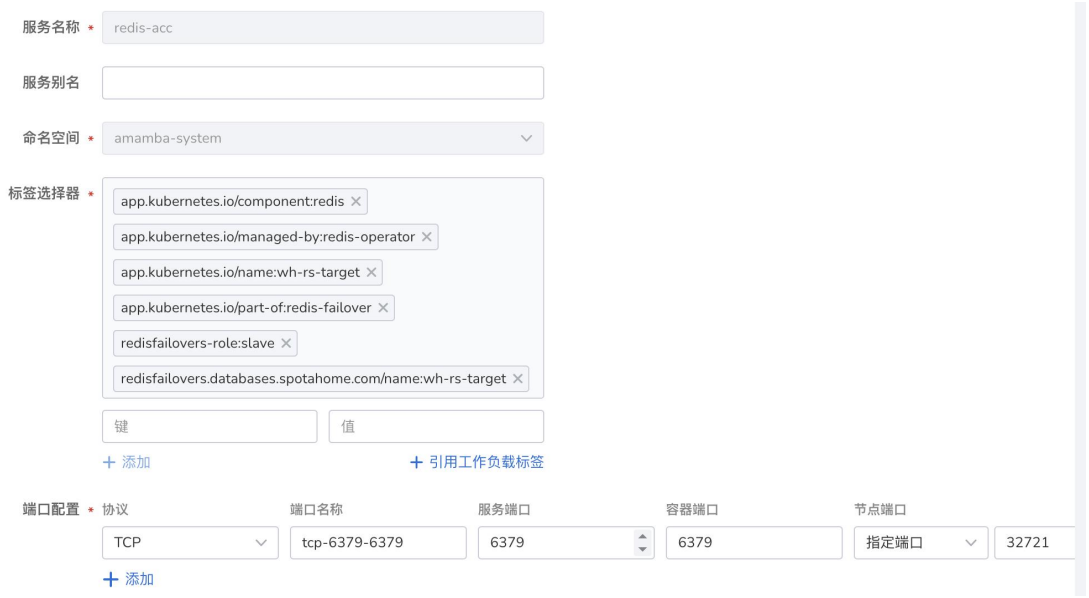
app.kubernetes.io/managed-by: redis-operator

app.kubernetes.io/name: redis-b

app.kubernetes.io/part-of: redis-failover

redis-failover: master

redisfailovers.databases.spotahome.com/name: redis-b



SVC

Redis-shake 部署

Redis-shake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在目标端部署 redis-shake，配置方式如下。

在集群模式下，需要为源端 Redis 实例 cluster-a 的每一个 Leader Pod 部署独立的 Redis-shake。以 redis-a-leader-0 为例，创建 Redis-shake-recovery-0：

1. 创建配置项

在容器管理 -{目标端集群}- 配置与存储 - 配置项为 Redis-shake 实例创建配置项 redis-recovery-0。导入文件 sync.toml（文件内容见附录），并注意需要修改以下内容：



conf

- source.address：源端 redis-b 的 redis-b-svc 服务地址：

  address = "10.233.109.145:6379"

- 源端实例的访问密码：可在【中间件】实例的概览页获取该信息：

  password = "3wPxzWffdn" *# keep empty if no authentication is required*

- 目标端实例访问地址，此处为 **实例 A** redis-a 的默认 clusterIP 服务 redis-a-leader-0 的地址：

  address = "172.30.120.202:32283"

- 目标端实例的访问密码，可在数据服务模块下的 Redis 实例概览页获取该信息:

  password = "3wPxzWffdn" *# keep empty if no authentication is required*



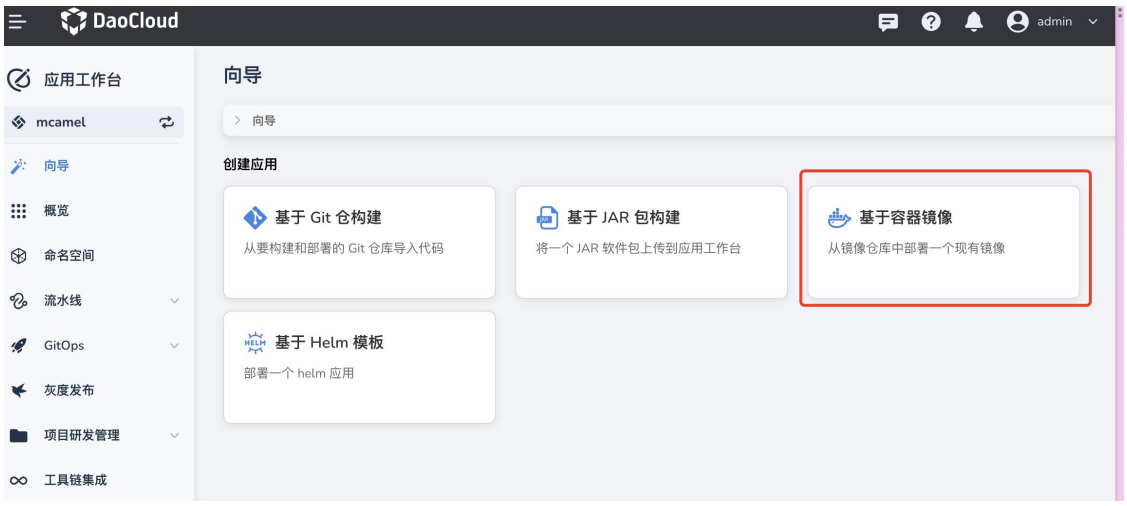
svc

- 目标端类型需设置为 cluster：

```
[target]
type = "cluster" # "standalone" or "cluster"
```

2. 创建 Redis-shake

1. 打开应用工作台，选择向导–基于容器镜像，创建一个应用 Redis–shake–recovery–0：



sync

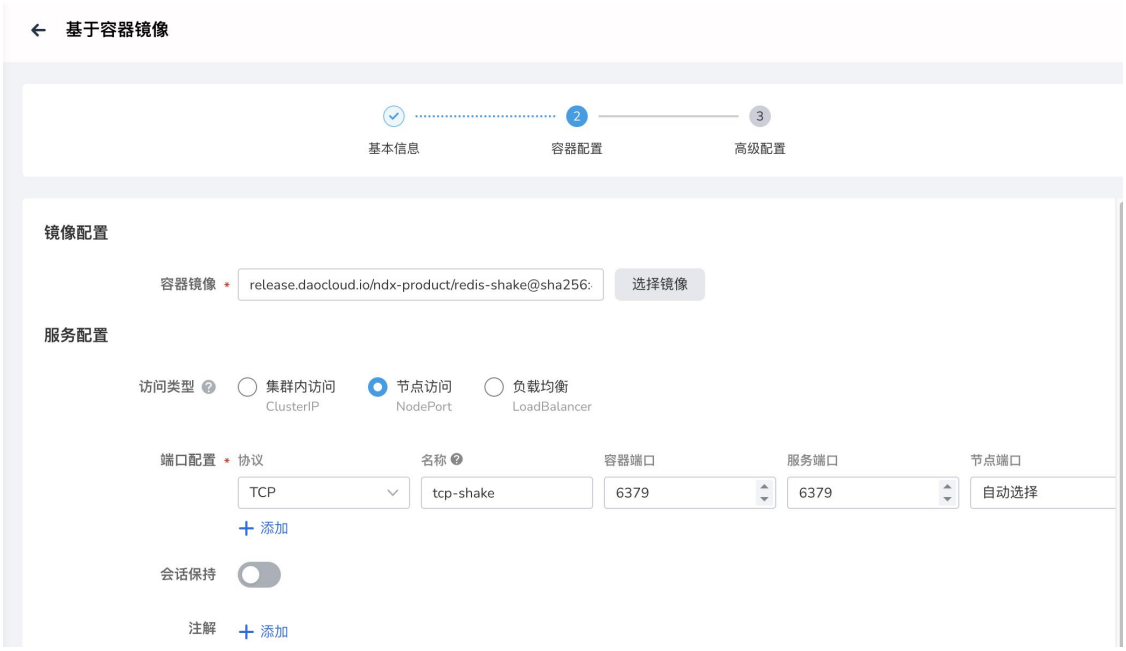
2. 参考如下说明填写应用配置。

   – 应用所属集群、命名空间需与 Redis 实例一致；

   – 镜像地址：

release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc1e211fb9cb894c4dc85e94287395a32fc3dc

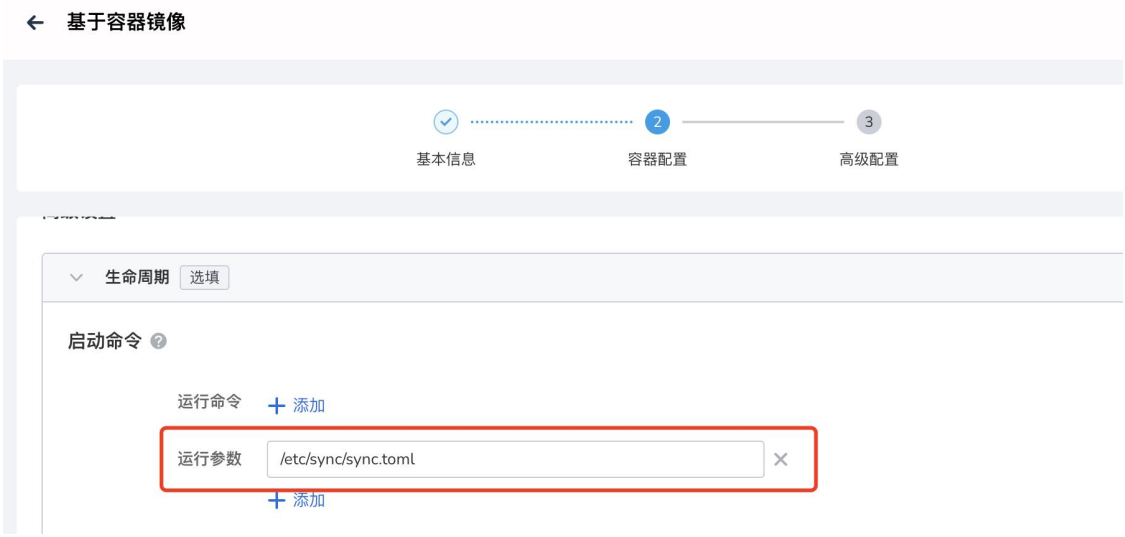
- 默认服务的访问类型为 Nodeport ，容器端口和服务端口设置为 6379。



sync

- 高级设置 – 生命周期 – 启动命令 – 运行参数 填入：

/etc/sync/sync.toml



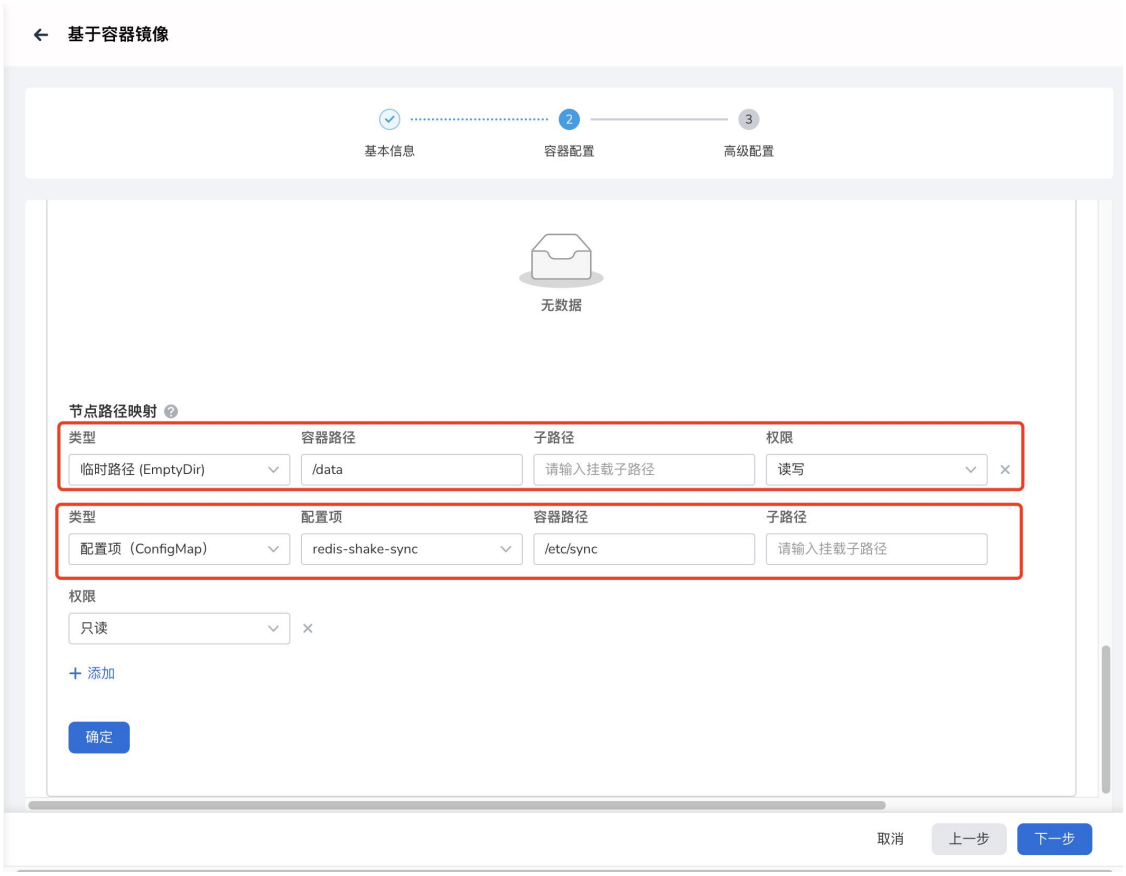
sync

- 高级设置 – 数据存储：添加配置项 redis-recovery-0，路径必须设置为：

  /etc/sync

- 高级设置 –数据存储：添加一个临时路径，容器路径必须为：

  /data



sync

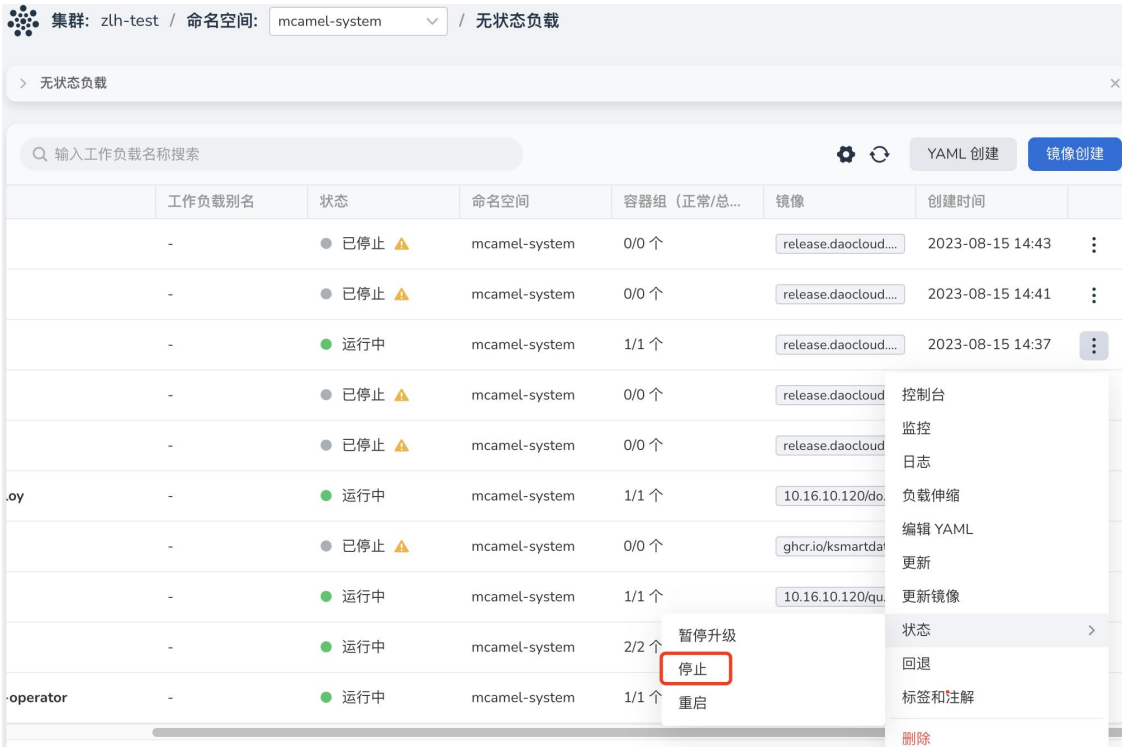
3. 点击确定，完成 Redis-shake 创建。

重复执行以上操作，分别为其他两个 Leader Pod 创建 Redis-shake-recovery-1，Redis-shake-recovery-2。

完成 Redis-shake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 redis-cli 工具验证同步，这里就不做赘述。

复原主从关系

如需复原初始的主从同步关系 **实例 A >> 实例 B**，需在容器管理中停止源端集群中的 3 个 Redis-shake-recovery 实例，重新启动目标端集群中的 3 个 Redis-shake-sync 实例，即可重建初始主从关系。



sync

附录

sync.toml

??? note “请点击查看配置文件”

```toml
type = "sync"

[source]
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
address = "10.233.109.145:6379"
username = "" # keep empty if not using ACL
password = "3wPxzWffdn" # keep empty if no authentication is required
tls = false
elasticache_psync = "" # using when source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373
```

```
[target]
type = "cluster" # "standalone" or "cluster"
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
# When the target is a cluster, write the address of one of the nodes.
# redis-shake will obtain other nodes through the `cluster nodes` command.
address = "10.233.103.2:6379"
username = "" # keep empty if not using ACL
password = "Aa123456" # keep empty if no authentication is required
tls = false

[advanced]
dir = "data"

# runtime.GOMAXPROCS, 0 means use runtime.NumCPU() cpu cores
ncpu = 4

# pprof port, 0 means disable
pprof_port = 0

# metric port, 0 means disable
metrics_port = 0

# log
log_file = "redis-shake.log"
log_level = "info" # debug, info or warn
log_interval = 5 # in seconds

# redis-shake gets key and value from rdb file, and uses RESTORE command t
o
# create the key in target redis. Redis RESTORE will return a "Target key nam
```

```
e
# is busy" error when key already exists. You can use this configuration item
# to change the default behavior of restore:
# panic:   redis-shake will stop when meet "Target key name is busy" error.
# rewrite: redis-shake will replace the key with new value.
# ignore:  redis-shake will skip restore the key when meet "Target key name is
 busy" error.
rdb_restore_command_behavior = "rewrite" # panic, rewrite or skip

# pipeline
pipeline_count_limit = 1024

# Client query buffers accumulate new commands. They are limited to a fixed
# amount by default. This amount is normally 1gb.
target_redis_client_max_querybuf_len = 1024_000_000

# In the Redis protocol, bulk requests, that are, elements representing single
# strings, are normally limited to 512 mb.
target_redis_proto_max_bulk_len = 512_000_000
```

RabbitMQ Release Notes

本页列出 RabbitMQ 消息队列的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.12.3

新功能

- **新增** 工作负载反亲和能力。
- **新增** UI 界面的权限访问限制。

2023-06-30

v0.11.0

新功能

- **新增** LoadBalancer 服务类型

优化

- **优化** 监控图表，去除干扰元素并新增时间范围选择

2023-04-27

v0.10.1

新功能

- **新增** mcamel-rabbitmq 详情页面展示相关的事件
- **新增** mcamel-rabbitmq 支持自定义角色
- **新增** mcamel-rabbitmq 支持接入链路追踪

优化

- **优化** mcamel-rabbitmq 调度策略增加滑动按钮

2023-03-30

v0.9.1

优化

- **优化** mcamel-rabbitmq 改进 Operator 镜像为加速地址。

2023-03-28

v0.9.0

新功能

- **新增** mcamel-rabbitmq 支持中间件链路追踪适配。
- **新增** 在安装 mcamel-rabbitmq 根据参数配置启用链路追踪。

2023-02-23

v0.8.0

新功能

- **新增** mcamel-rabbitmq helm-docs 模板文件。
- **新增** mcamel-rabbitmq 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-rabbitmq 支持 cloud shell。
- **新增** mcamel-rabbitmq 支持导航栏单独注册。
- **新增** 日志查看操作说明，支持自定义查询、导出等功能。

升级

- **升级** mcamel-rabbitmq 升级离线镜像检测脚本。

修复

- **新增** mcamel-rabbitmq 支持查看日志。
- **修复** mcamel-rabbitmq 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-rabbitmq 工作空间 Editor 用户无法查看实例密码。

2022-12-25

v0.7.0

新功能

- **新增** mcamel-rabbitmq NodePort 端口冲突提前检测。
- **新增** mcamel-rabbitmq 节点亲和性配置。

优化

- **优化** mcamel-rabbitmq-ui 中间件样式走查优化。

2022-11-28

v0.6.4

新功能

- **新增** 获取用户列表接口
- **新增** 支持多架构的镜像，配置方式为 depend.arm64-img.rabbitClusterImageFormat: xxxx
- **新增** 支持 sc 扩容拦截，当 sc 不支持扩容的时候，直接拦截掉
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回 alerts
- **新增** 校验 Service 注释

优化

- **优化** 密码校验调整为 MCamel 中等密码强度

修复

- **修复** 页面控制台可能访问到错误的端口

2022-10-27

v0.6.1

新功能

- **新增** 增加覆盖率
- **新增** 前端的 UI 注册功能

- **新增** 性能增强
- **新增** 列表页增加分页功能
- **新增** 增加修改配置的功能
- **新增** 增加返回可修改配置项的功能
- **新增** 更改创建实例的限制为集群级别，原来为 namespace 级别
- **新增** 增加监控地址的拼接功能
- **新增** 增加可以修改版本号的功能
- **新增** 修改底层 update 逻辑为 patch 逻辑
- **新增** RabbitMQ e2e 测试覆盖率 17.24% 左右
- **新增** 增加 RabbitMQ 性能压测报告
- **新增** 增加 RabbitMQ bug 抽查
- **新增** 对接 ghippo 增加 workspace 接口
- **新增** 对接 insight 通过 crd 注入 dashboard
- **新增** 将时间戳 api 字段统一调整为 int64
- **新增** 单测覆盖率提升到 53%
- **优化** 更新 release note 脚本，执行 release-process 规范
- **新增** 新增功能说明
- **新增** 创建 RabbitMQ
- **新增** RabbitMQ 数据迁移
- **新增** 实例监控
- **新增** 首次进入 RabbitMQ
- **新增** 适用场景

产品优势

RabbitMQ 具有以下产品优势。

- 行业领先
  - 支持 ActiveMQ 的快速部署与生命周期管理。ActiveMQ 是目前最流行的开源多协议消息中间件，支持 JavaScript, C, C++, Python, .Net 等语言编写的应用和平台，能够跨应用交换消息。通过消息控制台实现对生产集群、消费集群、消息查询等的管理操作。
  - 采用业界先进、成熟的技术路线，保证平台的稳定可靠性；同时充分考虑用户交互的便利性和界面操作的易用性，采用流行的灵活且易扩展系统架构，使用先进的技术框架路线进行设计与开发，支持基于 API 调用各组件协同工作，能够满足系统纵向集成与横向整合的信息交互。
  - 采用可靠的技术架构，支持系统稳定运行，每个组件都能提供高可用性，能够保障关键组件冗余与高可用。
  - 能够将自身运行的内在状态数据，通过采集、分析、处理后，通过合理的聚合，归纳为指标数据，方便运维人员在最短时间内了解到系统运行实时状态，同时可提供对应的 API 供其它系统读取状态指标数据。
- 生命周期管理
  - 支持图形化或通过 YAML 创建、更新和删除 RabbitMQ 实例。
  - 支持热部署、热更新，除通用技术服务组件升级、平台升级等重大升级改造外，系统部署和升级无需停机。
- 图形化界面操作

支持以图形化方式创建、查看、更新、删除以及弹性扩容。管理员可以根据自己的偏好，定制图形化参数。

- 容器化消息平台

  RabbitMQ 基于 Kubernetes 和 Docker，原生支持容器化部署，将资源利用率提高到最大。

- 兼容性和开放性

  - RabbitMQ 兼容 Intel CPU 和国产化 CPU（如飞腾 ARM，华为鲲鹏 ARM 和海光 x86 等），兼容国产操作系统（如麒麟操作系统 v10，统信服务器操作系统 UOS 等），这些均有授权的电子证明和证书。
  - 具有开放的体系架构，提供开放标准 API 接口保证基础能力、管理服务、日志、监控等资源和服务被高效的调度、管理与使用，支持第三方管理平台通过 API 接口等方式实现相关编排与使用。

基本概念

本节列出有关 RabbitMQ 涉及的专有名词及术语，方便您更好地理解相关概念并使用 RabbitMQ 消息队列。

- 消息（Message）

  消息一般分为两部分：消息体和标签。标签也称为消息头，主要用来描述这条消息。消息体是消息的内容，是一个 json 体或者数据等。

  消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括 routing-key（路由键）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。

  生产者发布消息，消费者使用消息，生产者和消费者彼此并无直接关系。

- 消息标识（Message ID）

  消息标识是消息的可选属性，类型为一个短字符串（short string）。

- 队列（Queue）

  队列用于存储消息，生产者将消息送到队列，消费者从队列中获取消息。多个消费者可以同时订阅同一个队列，队列里的消息分配给不同的消费者。每个消息都会被投入到一个或多个队列里。

- 消息存活时间

  消息在队列中的有效期。某条消息在队列中的留存时间超过配置的消息存活时间时，则该消息过期。消息存活时间的值必须为非负整型数，单位为毫秒。例如，某条消息的存活时间的值是 1000，则代表该消息最多会在队列中存活 1 秒。

- 延时消息

生产者将消息发布到消息队列 RabbitMQ 版服务端，但并不期望这条消息立马投递，而是延迟一定时间后才投递到消费者进行消费，该消息即延时消息。

- 生产者（Publisher）

  消息的生产者，也是一个向交换器发布消息的客户端应用程序。即向队列发布消息的一方。发布消息的最终目的在于将消息内容传递给其他系统/模块，使对方按照约定处理该消息。

- 消费者（Consumer）

  消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。消费者订阅 RabbitMQ 的队列。当消费者使用一条消息时，只是使用消息的消息体。在消息路由的过程中，会丢弃标签，存入到队列中的只有消息体。

- 代理（Broker）

  消息中间件的服务节点，即消息队列服务器实体。

什么是 RabbitMQ

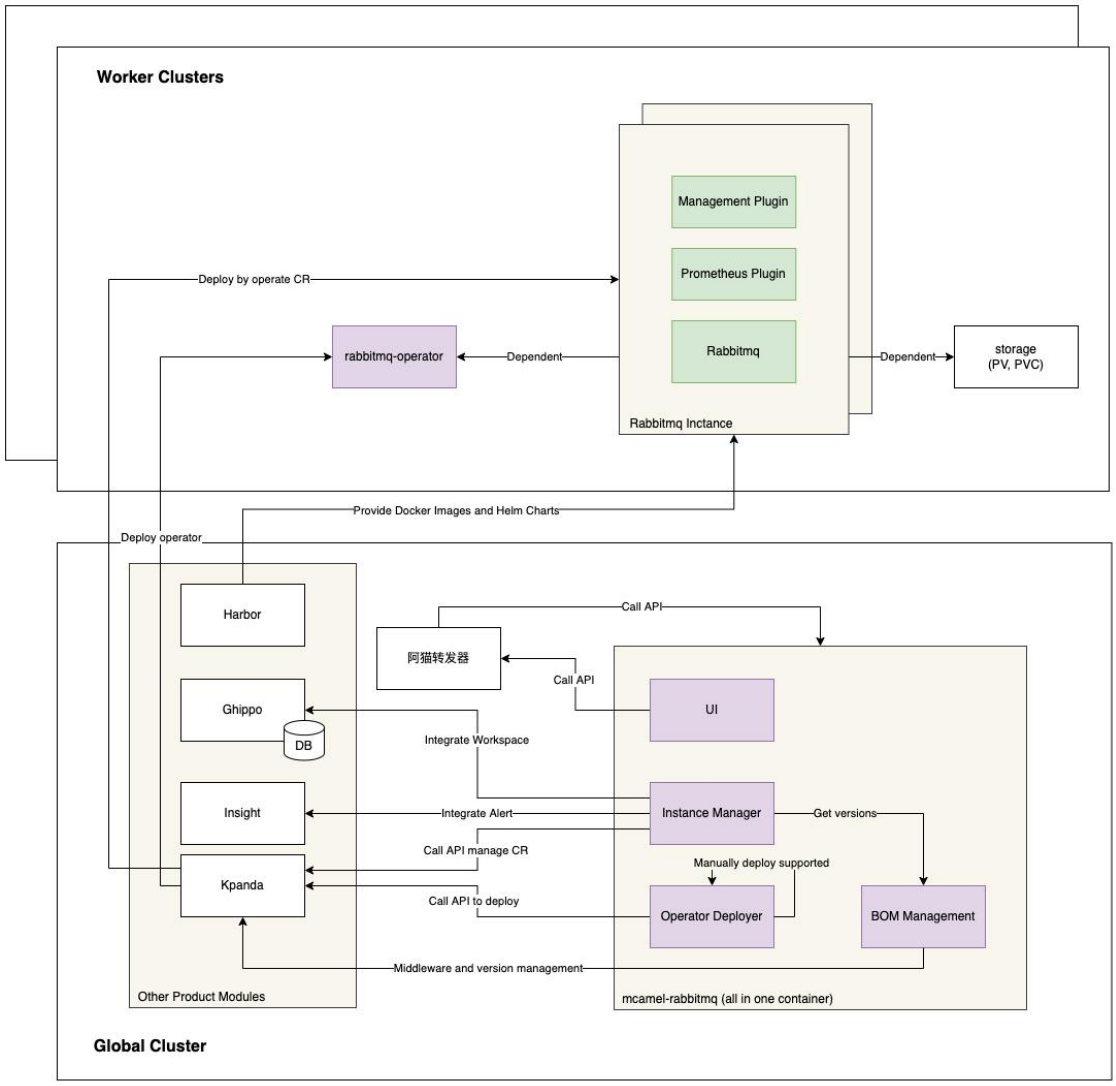
RabbitMQ 是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件），具有高吞吐、低延迟、可扩容等特性。

RabbitMQ 是应用层协议的一个开放标准，为面向消息的中间件设计，基于此协议的客户端与消息中间件来传递消息，不受产品、开发语言等条件的限制。 这是一个企业级的真正具备低延迟、高并发、高可用、高可靠特性，可支撑万亿级数据洪峰的分布式消息中间件服务。



rmq 主界面

其系统架构及数据流如下图。



架构数据流

创建 RabbitMQ 实例

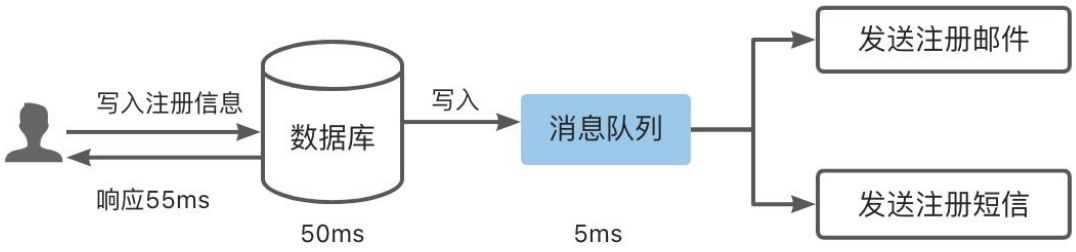
适用场景

RabbitMQ 适用的场景广泛，本节列觉了几个典型场景。

异步处理

场景说明：用户注册后，需要发送注册邮件和注册短信。

引入消息队列后，用户的响应时间就等于写入数据库的时间 + 写入消息队列的时间（这个可以忽略不计）。 引入消息队列后处理后，响应时间是串行的 3 倍，是并行的 2 倍。

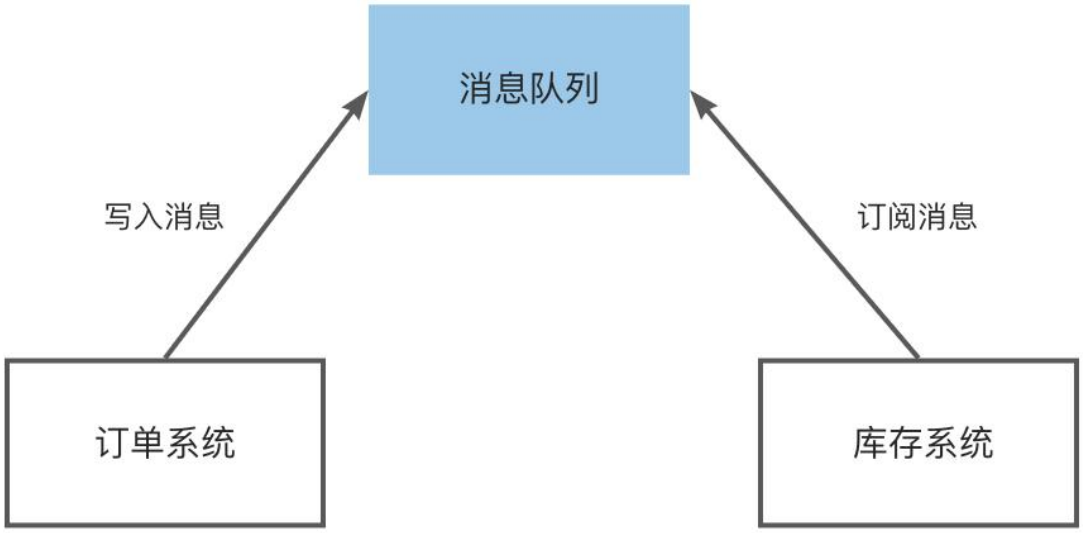


异步处理

应用解耦

场景说明：在促销活动时用户下单数量激增，用户下单后，订单系统需要通知库存系统。

传统的做法就是订单系统调用库存系统的接口，这会导致库存系统出现故障时，订单会失败。 如果使用消息队列（如下图），用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功。 订阅下单的消息，获取下单消息，进行库操作。就算库存系统出现故障，消息队列也能保证消息的可靠投递，不会导致消息丢失。

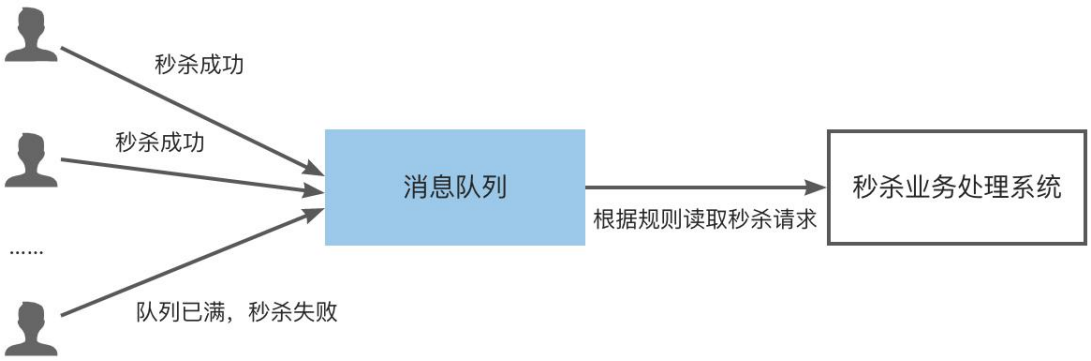


应用解耦

流量削峰

场景说明：某平台策划的秒杀活动，一般会因为流量过大，导致应用不可用。

为了解决这个问题，一般在应用前端加入消息队列。 通过消息队列可以控制活动人数，超过此一定阀值的订单直接丢弃，同时可以缓解短时间的高流量压垮应用。服务器收到用户的请求之后，首先写入消息队列，假如消息队列长度超过最大值，则直接抛弃用户请求或跳转到错误页面。 秒杀业务根据消息队列中的请求信息，再做后续处理。



流量削峰

功能特性

RabbitMQ 的通用功能特性包括：

- 可靠性（Reliability）

  RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。

- 消息集群（Clustering）

  多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker。

- 高可用队列（Highly Available Queues）

  队列可以在集群中的主机上进行镜像，使得在部分节点出问题的情况下队列仍然可用。

- 多种协议（Multi-protocol）

  RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等。

- 多语言客户端（Many Clients）

  RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等。

- 管理界面（Management UI）

  RabbitMQ 提供了一个易用的图形用户界面，使得用户可以监控和管理消息 Broker 的方方面面。

- 跟踪机制（Tracing）

  如果消息异常，RabbitMQ 提供了消息跟踪机制，用户可以轻松找出发生了什么。

- 插件机制（Plugin System）

  RabbitMQ 提供了许多插件，支持从多方面进行扩展，也可以编写自己的插件。

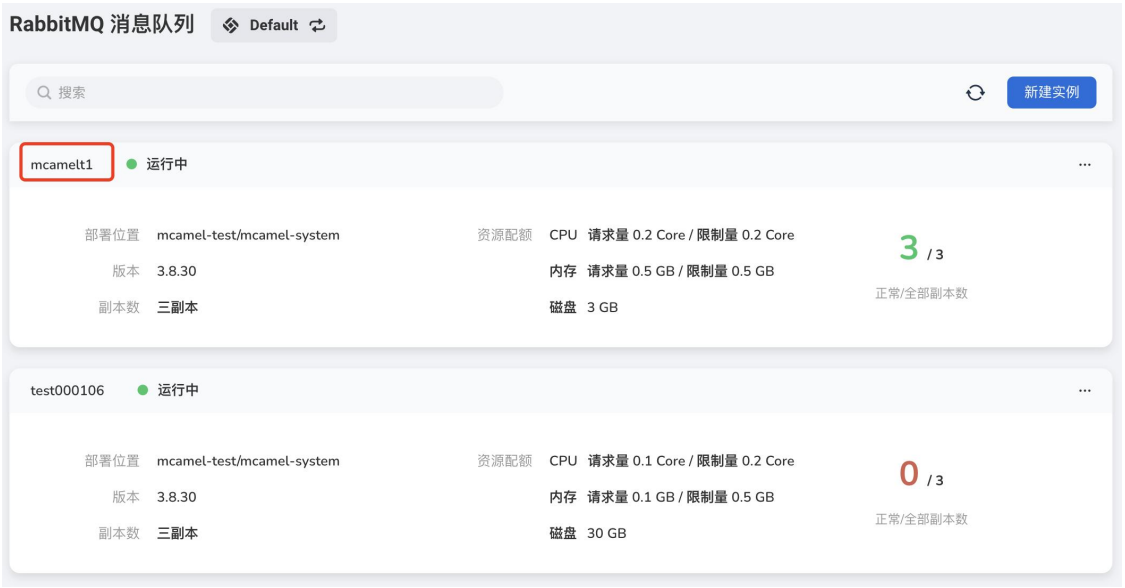
在 DCE 5.0 中部署 RabbitMQ 后，还将支持以下特性：

- 支持单节点和多节点 RabbitMQ 集群部署
- 支持 RabbitMQ Managerment 插件，提供管理页面
- 支持 RabbitMQ Prometheus 插件，暴露监控指标
- 使用 ServiceMonitor 对接 Prometheus 抓取指标
- 支持 RabbitMQ 集群的扩容和滚动升级

实例监控

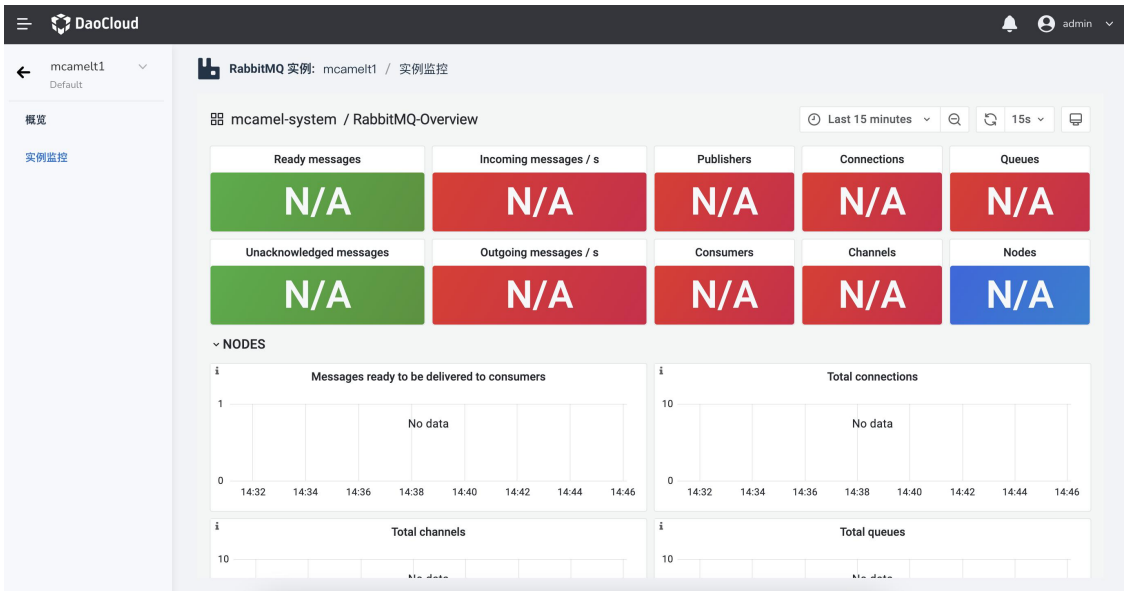
RabbitMQ 内置了 Prometheus 和 Grafana 监控模块。

1. 在消息队列页面中，点击某个名称。



点击某个名称

2. 在左侧导航栏，点击实例监控，可以接入监控模块。



实例监控

各项监控指标如下。

Panel 名	指标名称	说明
connections	连接数	该指标用于统计 RabbitMQ 实例中的总连接数。
channels	通道数	该指标用于统计 RabbitMQ 实例中的总通道数。
queues	队列数	该指标用于统计 RabbitMQ 实例中的总队列数。
consumers	消费者数	该指标用于统计 RabbitMQ 实例中的总消费者数。
publish	生产速率	统计 RabbitMQ 实例中实时消息生产速率。
socket_used	Socket 连接数	该指标用于统计当前节点 RabbitMQ 所使用的 Socket 连接数。
CPU Usage	CPU 使用量	该指标用于统计节点 CPU 使用量。

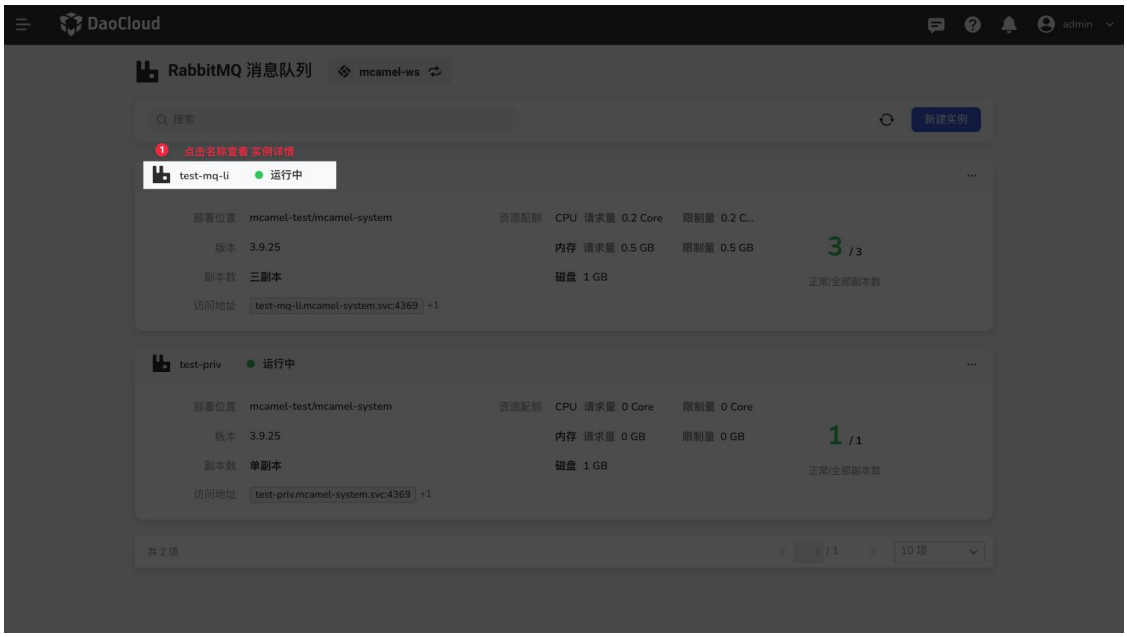
Panel 名	指标名称	说明
Memory Usage	内存使用量	该指标用于统计节点内存使用量。

查看 RabbitMQ 日志

操作步骤

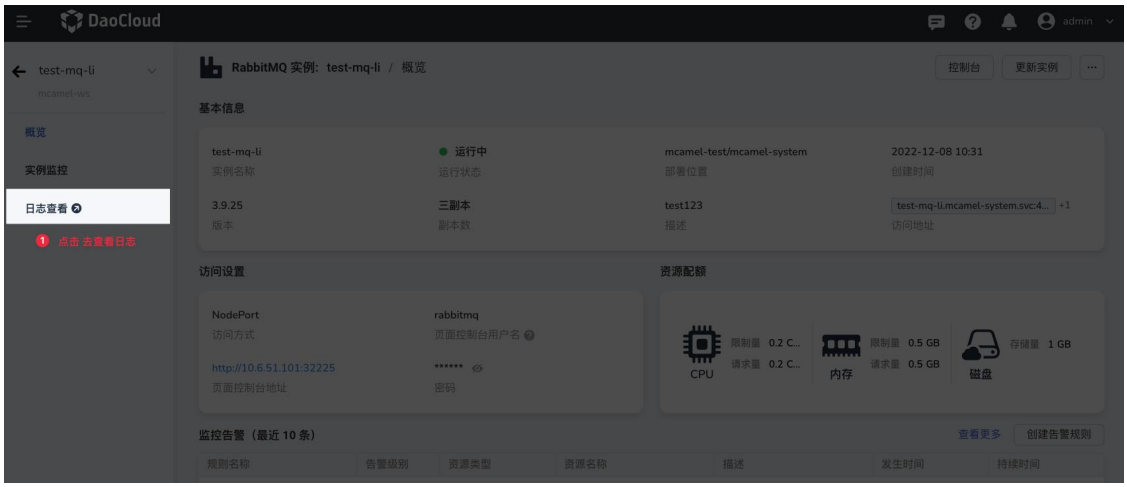
通过访问每个 RabbitMQ 的实例详情，页面；可以支持查看 RabbitMQ 的日志。

1. 在 RabbitMQ 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



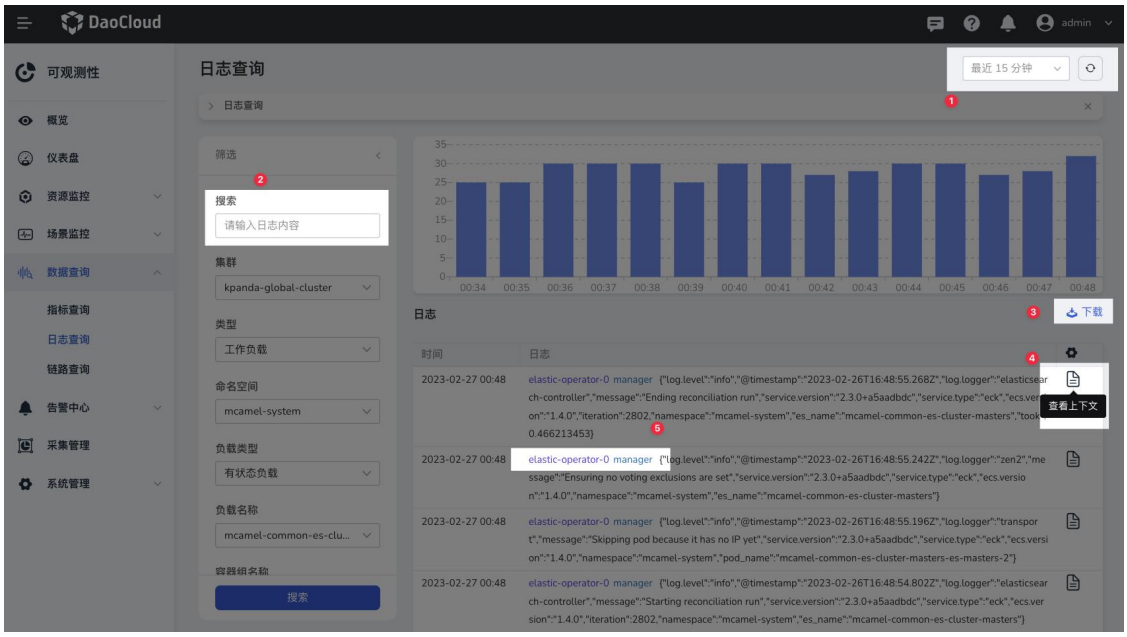
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

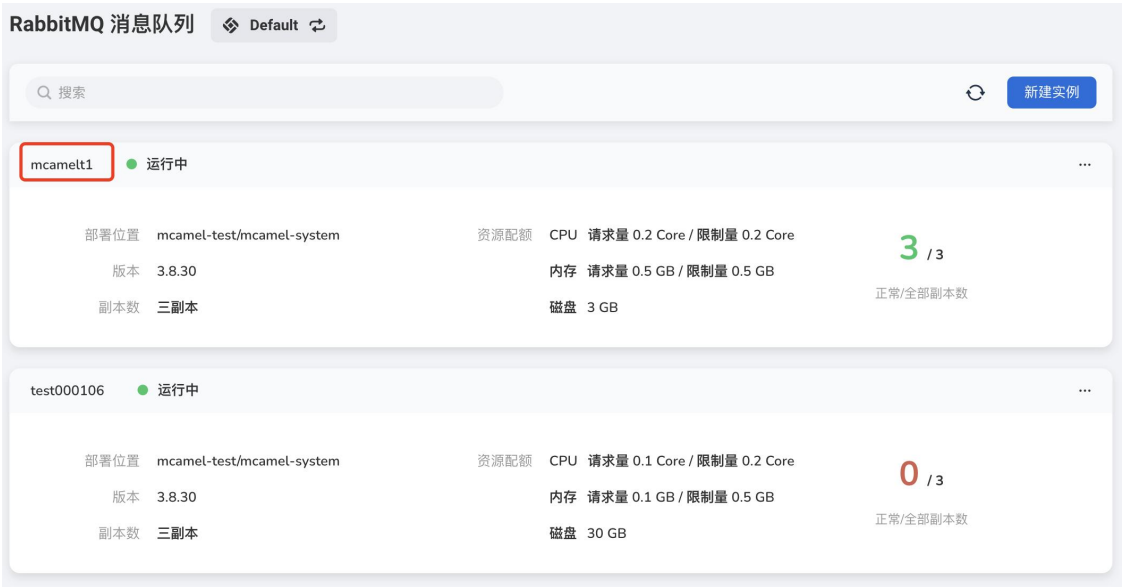

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

查看消息队列

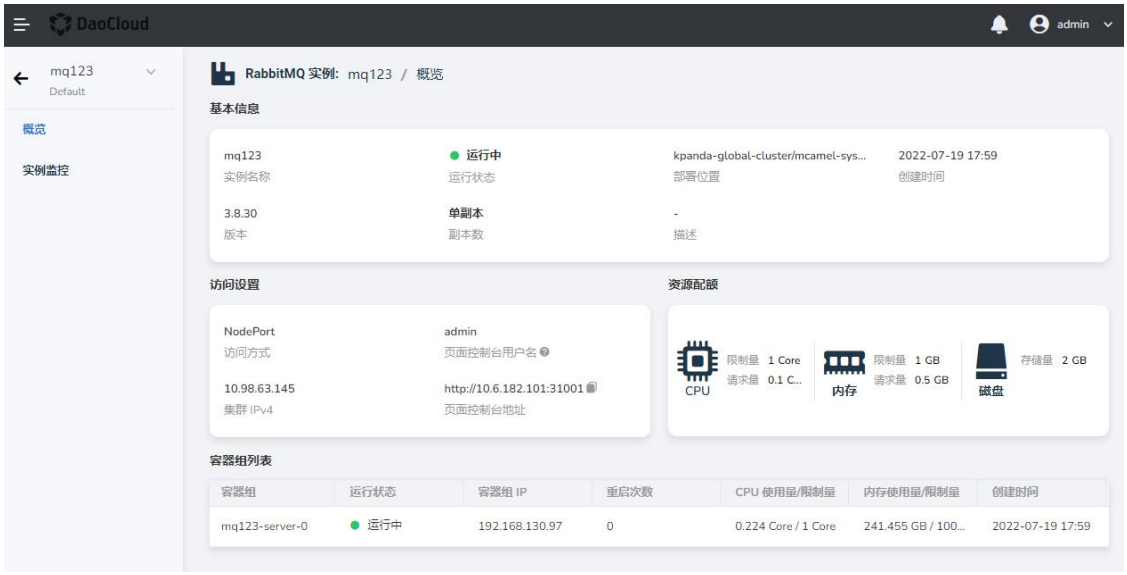
本节说明如何查看 RabbitMQ 消息队列。

1. 在消息队列页面中，点击某个名称。



点击某个名称

2. 进入消息队列概览，查看访问设置、资源配额和 Pod 列表等信息。


DaoCloud
admin
mq123
Default
概览
实例监控
RabbitMQ 实例: mq123 / 概览
基本信息
mq123
实例名称
运行中
运行状态
kpanda-global-cluster/mcamel-sys...
部署位置
2022-07-19 17:59
创建时间
3.8.30
版本
单副本
副本数
-
描述
访问设置
NodePort
访问方式
admin
页面控制台用户名
10.98.63.145
集群 IPv4
http://10.6.182.101:31001
页面控制台地址
资源配额
CPU
限制量 1 Core
请求量 0.1 C...
内存
限制量 1 GB
请求量 0.5 GB
磁盘
存储量 2 GB
容器组列表
容器组
运行状态
容器组 IP
重启次数
CPU 使用量/限制量
内存使用量/限制量
创建时间
mq123-server-0
运行中
192.168.130.97
0
0.224 Core / 1 Core
241.455 GB / 100...
2022-07-19 17:59

查看

RabbitMQ 数据迁移

RabbitMQ 的数据包括元数据（RabbitMQ 用户、vhost、队列、交换和绑定）和消息数据，其中消息数据存储在单独的消息存储库中。

由于业务需要，要求把 rabbitmq-cluster-a 集群上的数据迁移到 rabbitmq-cluster-b 集群上。

数据迁移步骤

!!! info

从 V3.7.0 开始，RabbitMQ 将所有消息数据存储在 `msg_stores/vhosts` 目录中，并存储在每个 vhost 的子目录中。
每个 vhost 目录都以哈希命名，并包含一个带有 vhost 名称的 `.vhost` 文件，因此可以单独备份特定 vhost 的消息集。
了解[更多信息](https://www.rabbitmq.com/backup.html)。

RabbitMQ 数据迁移，可以采用如下两种方案：

- 方案一：不迁移数据，先切换生产端，再切换消费端。
- 方案二：先迁移数据，然后同时切换生产端和消费端。

方案一

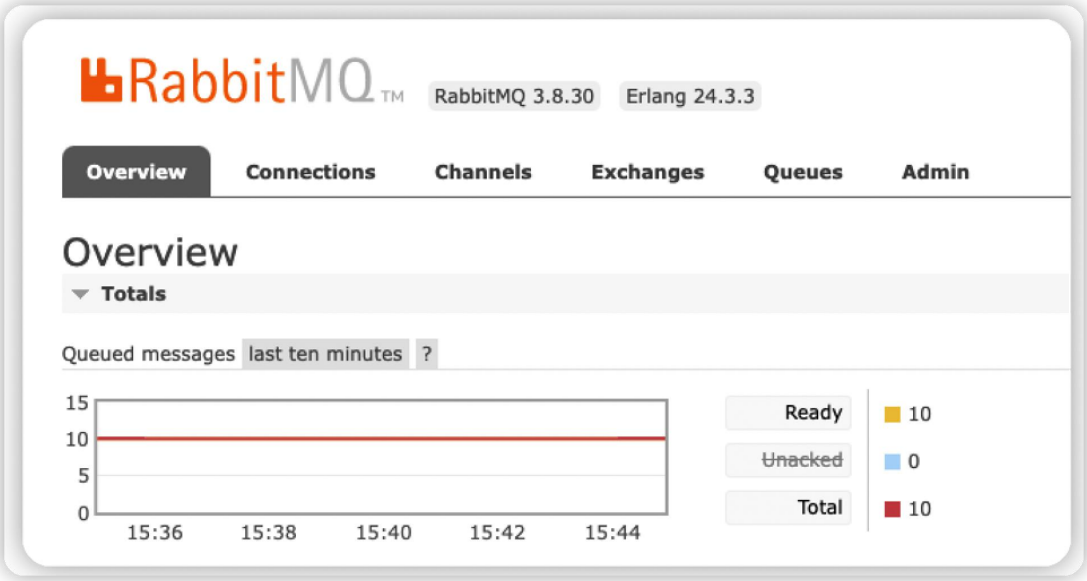
不迁移数据，先切换生产端，再切换消费端。

操作流程

1. 将消息生产端切换到集群 rabbitmq-cluster-b，不再生产消息到 rabbitmq-cluster-a 集群中。
2. 消费端同时消费 rabbitmq-cluster-a 和 rabbitmq-cluster-b 集群中的消息。当 rabbitmq-cluster-a 集群中消息全部消费完后，将消息消费端切换到 rabbitmq-cluster-b 集群中，完成数据迁移。

验证方法

- 在 RabbitMQ Management Web UI 页面查看。



查看

- 调用 API 查看

curl –s –u username:password –XGET http://ip:port/api/overview

参数说明：

  - username：使用 rabbitmq-cluster-a 集群的 RabbitMQ Management WebUI 的帐号
  - password：使用 rabbitmq-cluster-a 集群的 RabbitMQ Management WebUI 的密码
  - ip：使用 rabbitmq-cluster-a 集群的 RabbitMQ Management WebUI 的 IP 地址
  - port：使用 rabbitmq-cluster-a 集群的 RabbitMQ Management WebUI 的端口号

- 在 Overview 视图中，消费消息数（Ready）以及未确定的消息数（Unacked）都为 0，说明消费完成。



消息数为 0

方案二

先迁移数据，然后同时切换生产端和消费端。借助 shove 插件完成数据迁移。

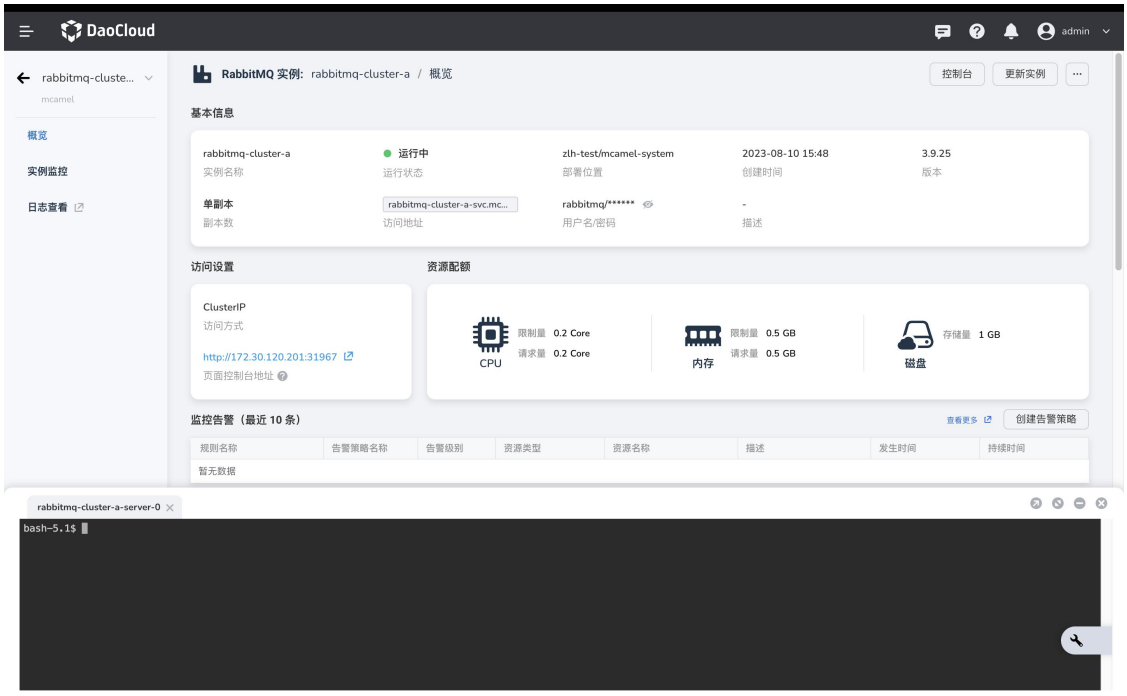
> shovel 迁移数据的原理是消费 rabbitmq-cluster-a 集群中的消息，将消息生产到 rabbitmq-cluster-b 集群中，迁移后 rabbitmq-cluster-a 集群中的消息被清空，建议离线迁移，业务会出现中断。

rabbitmq-cluster-a 和 rabbitmq-cluster-b 均需要开启并配置 shovel 插件。

启用 shovel

开启插件

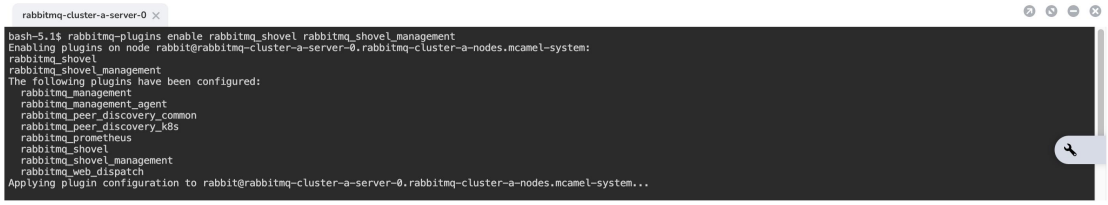
1. 进入中间件 -> RabbitMQ 的实例列表，进入 rabbitmq-cluster-a 的概览页面点击控制台按钮;



控制台

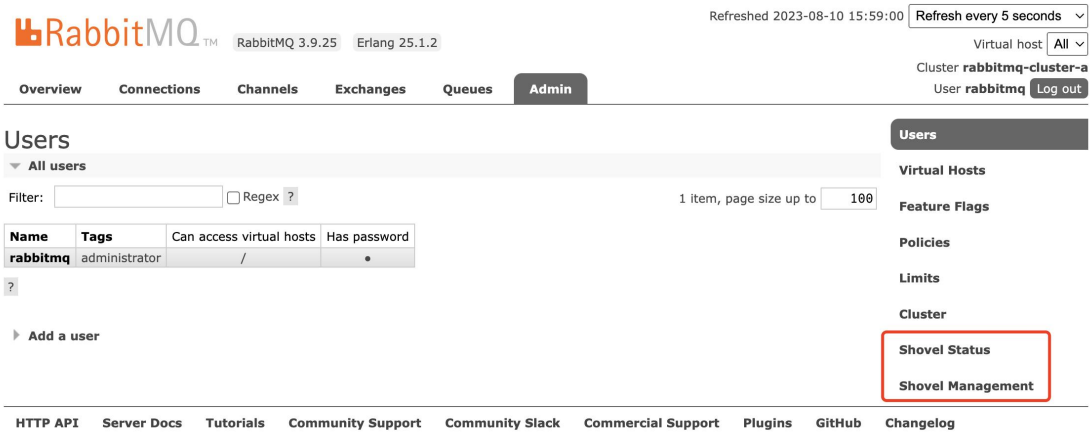
2. 执行以下命令，该过程可能会持续一两分钟：

   rabbitmq-plugins enable rabbitmq_shovel rabbitmq_shovel_management



执行命令

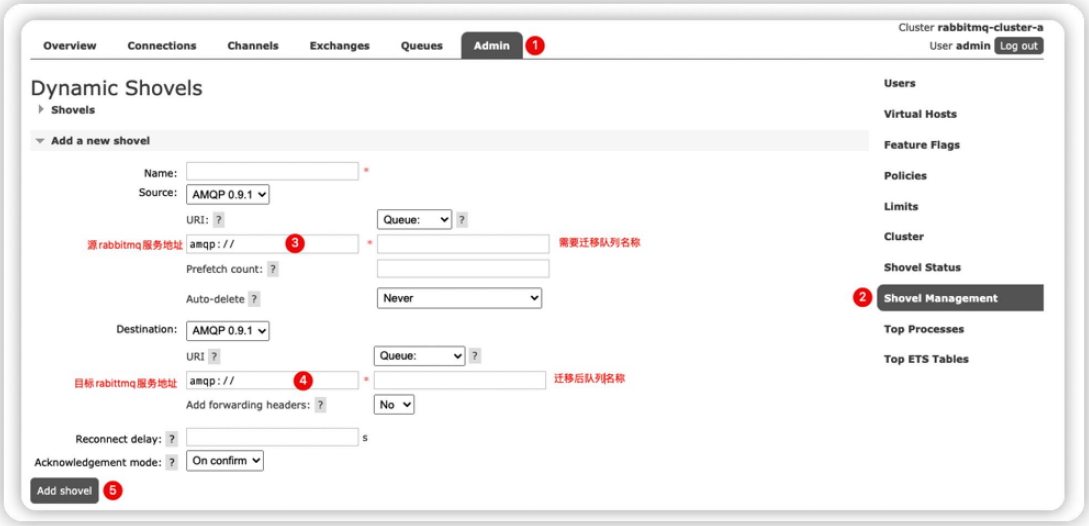
3. 进入 RabbitMQ 管理平台，在 admin 页签下可以看到 shovel 相关的插件信息。



开启插件

进入实例 rabbitmq-cluster-b 的概览页面，再次执行以上操作。

基本配置



插件配置

参数说明：

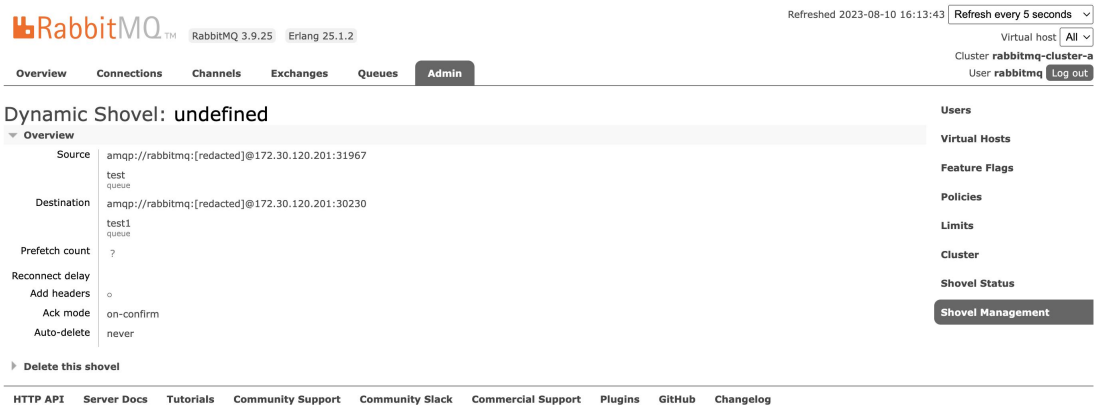
- Name：配置 shovel 的名称。
- Source：指定协议类型、连接的源集群地址，源端的类型。

- Prefech count: 表示 shovel 内部缓存（从源端集群到目的实例之间的缓存部分）的消息条数。
- Auto-delete: 默认为 Never，表示不删除本集群消息，如果设置为 After initial length transferred，则在消息转移完成后删除。
- Destination: 指定协议类型，连接目标集群地址，目标端的类型。
- Add forwarding headers: 设置为 true，则会在转发的消息内添加 x-shovelled 的 header 属性。
- Reconnect delay：指定在 Shovel link 失效的情况下，重新建立连接前需要等待的时间，单位为秒。如果设置为 0，则不会进行重连动作，即 Shovel 会在首次连接失效时停止工作。默认为 5 秒。
- Acknowledgement mode：参考 Federation 的配置。
    - no ack 表示无须任何消息确认行为；
    - on publish 表示 Shovel 会把每一条消息发送到目的端之后再向源端发送消息确认；
    - on confirm 表示 Shovel 会使用 publisher confirm 机制，在收到目的端的消息确认之后再向源端发送消息确认。

!!! note

服务地址（图中的 3 和 4）设置格式：amqp://用户名:密码@{rabbitmq 服务地址}

下图是一个简单的配置示例



配置示例

启动迁移

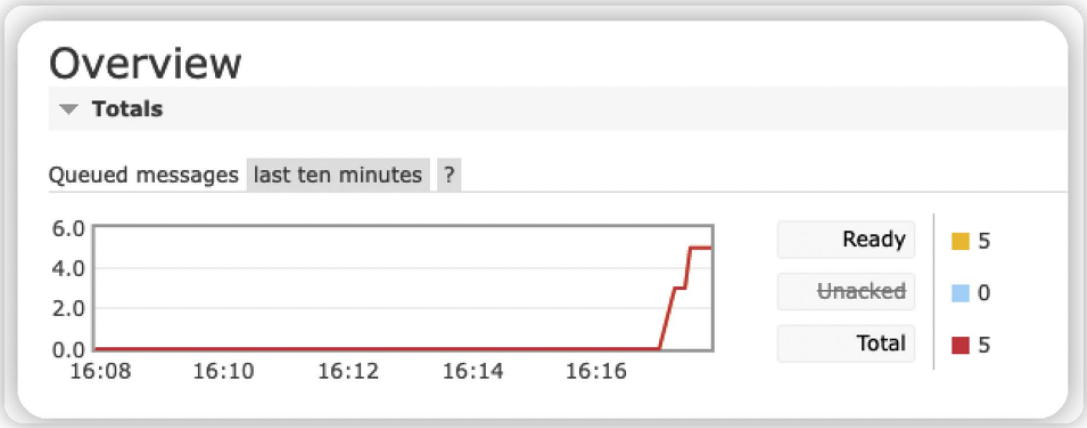
迁移任务将自动启动，当 shovel 状态为 running 时，表示迁移开始，如下图所示。



运行状态

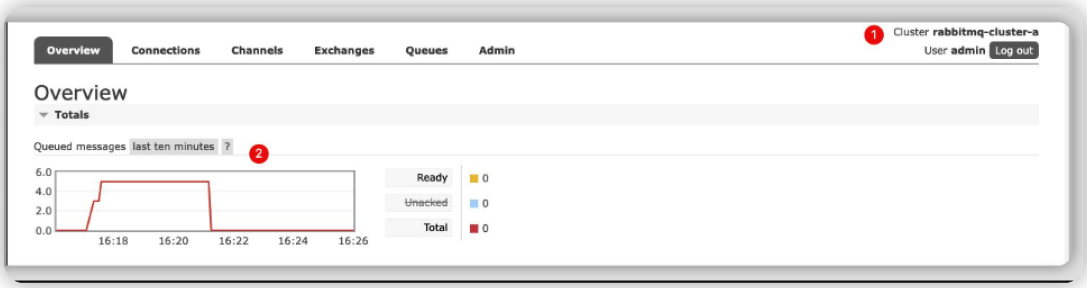
迁移前后观察两个集群的队列状态，可明显看到数据迁移变化：

- 迁移启动前 rabbitmq-cluster-a 集群消息情况。



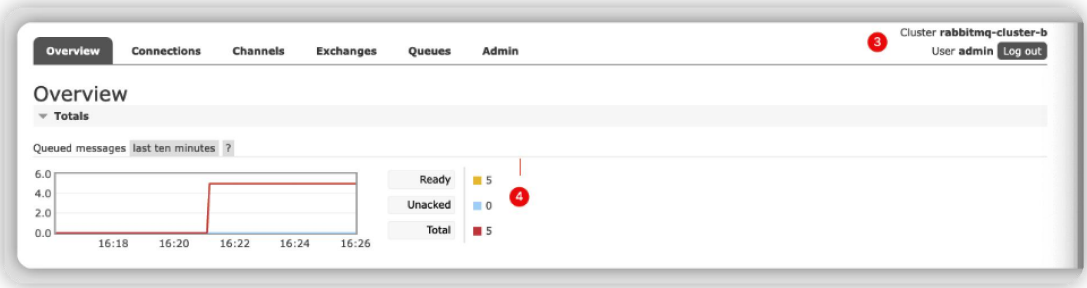
迁移前消息

- 迁移启动后 rabbitmq-cluster-a 集群消息情况，可见队列消息已迁出。



集群消息

- 迁移启动后 rabbitmq-cluster-b 集群消息情况，可见队列消息已迁入该集群。



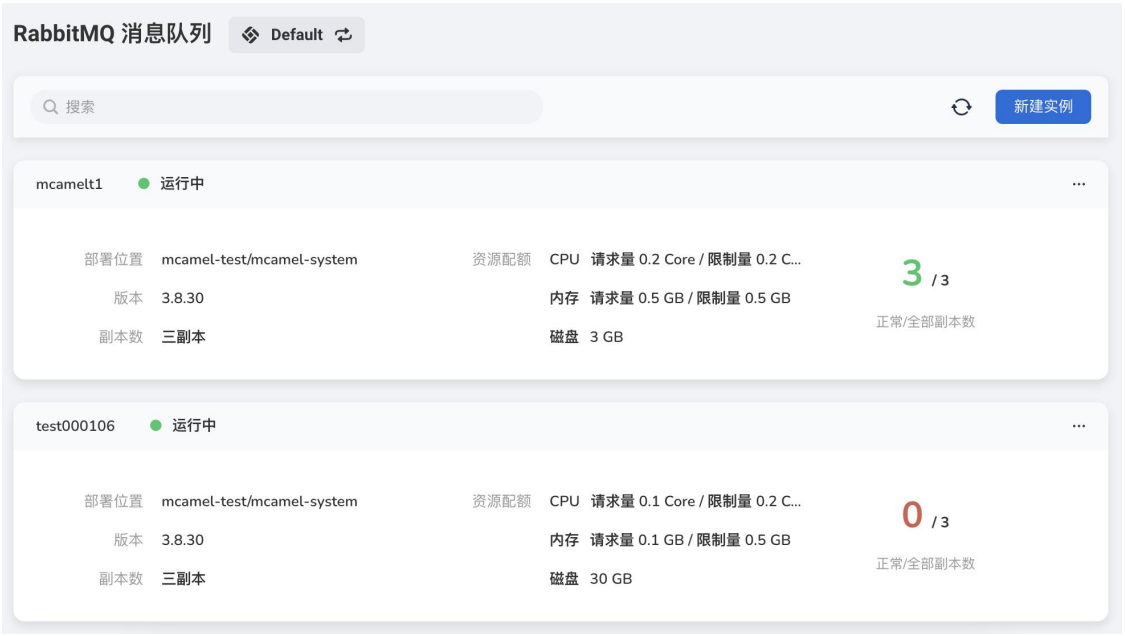
集群消息

数据迁移完成后，即可将生产端、消费端切换至 rabbitmq-cluster-b 集群中，完成迁移过程。

创建 RabbitMQ

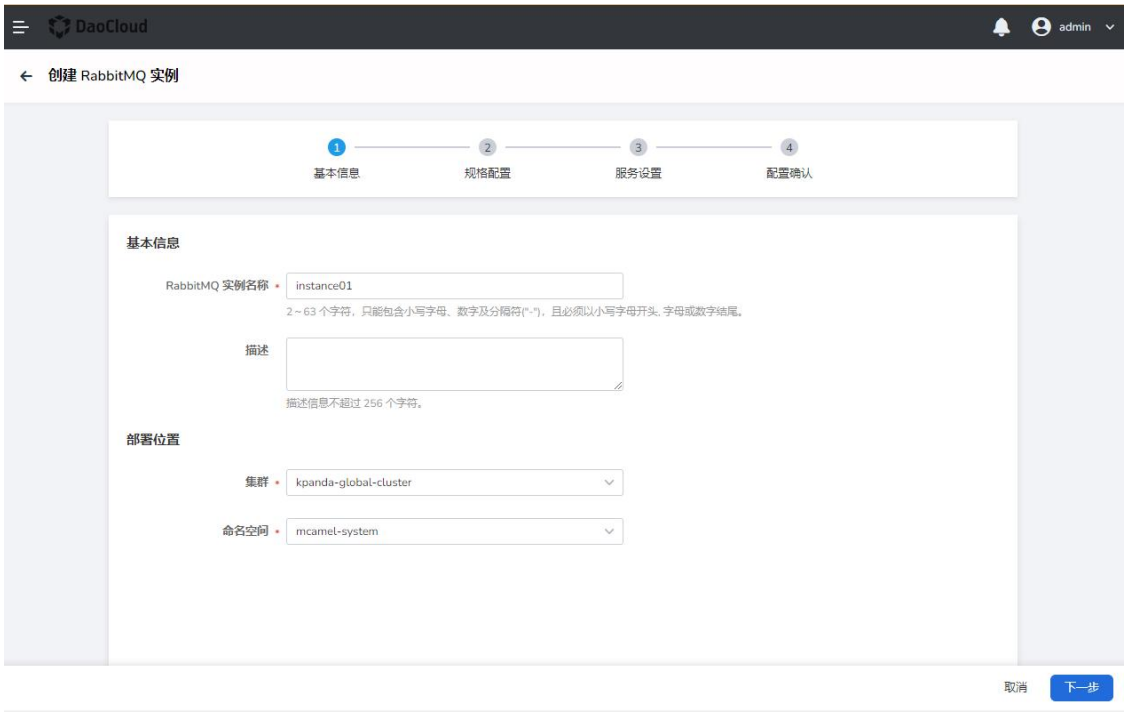
在 RabbitMQ 消息队列中，执行以下操作：

1. 在右上角点击新建实例。



点击新建实例

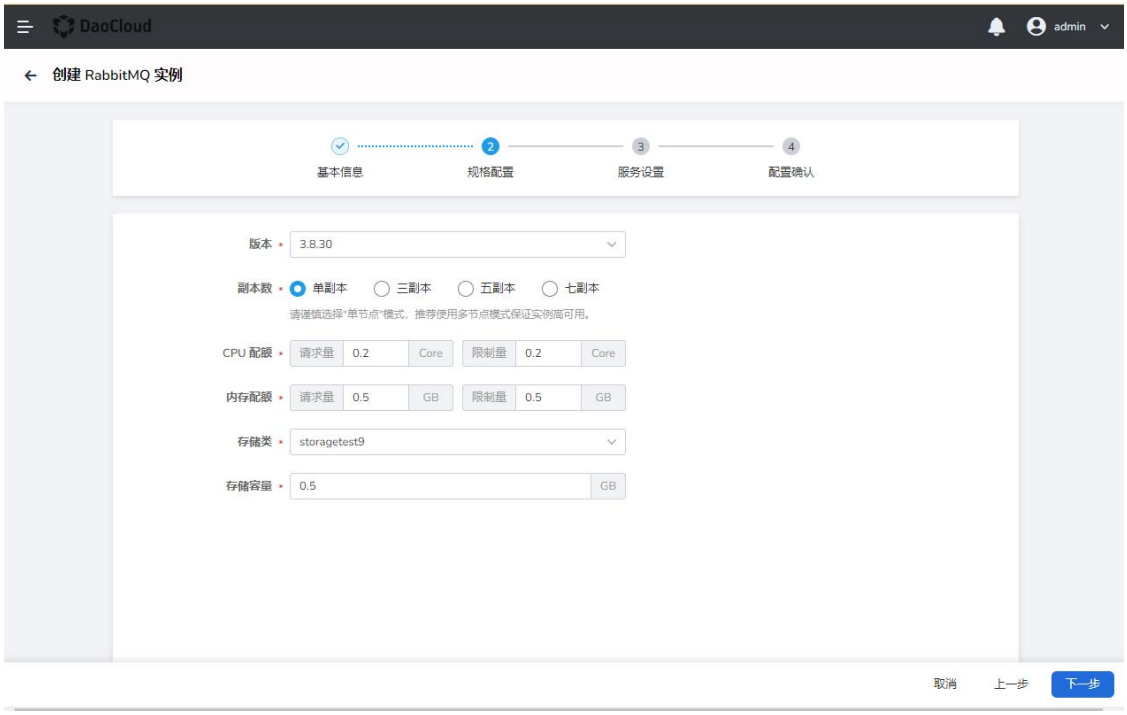
2. 在创建 RabbitMQ 实例页面中，设置基本信息后，点击下一步。



基本信息

3. 配置规格后，点击下一步。

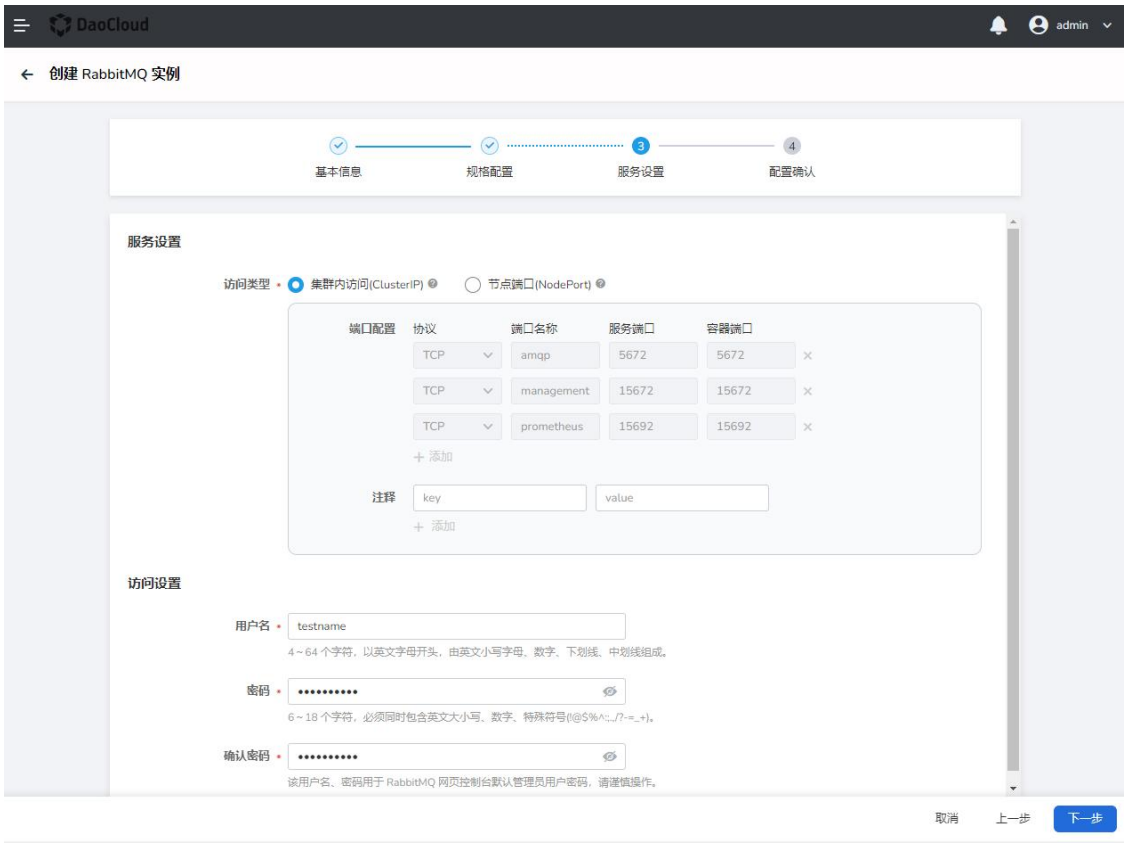
   - 版本：RabbitMQ 的版本号，当前仅支持 RabbitMQ 3.7.20。
   - 副本数：支持 1、3、5、7 副本数。
   - 资源配额：根据实际情况选择规则。
   - 存储卷：选择 RabbitMQ 实例的存储卷和储存空间总量。



配置规格

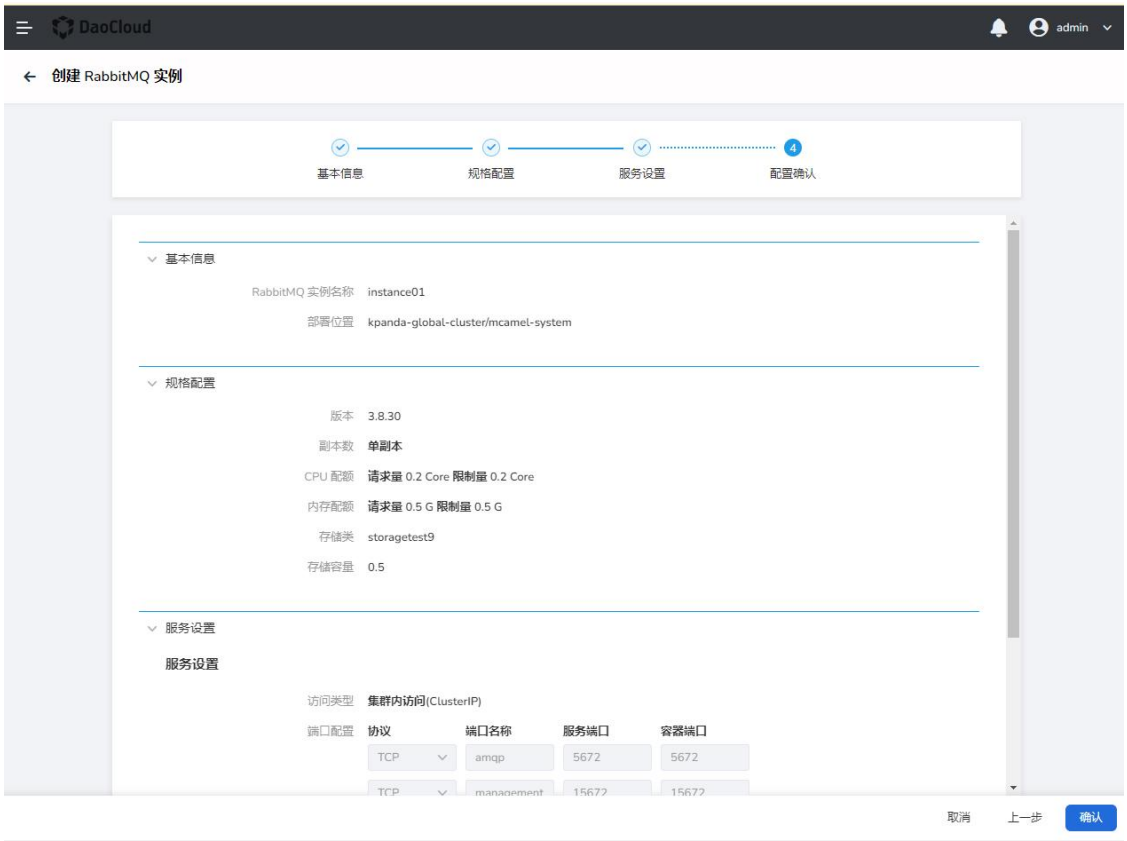
4. 服务设置后，点击下一步。

   - 访问方式：可以选择集群内访问还是 Nodeport 访问。
   - 服务设置：设置连接 RabbitMQ 实例的用户名、密码。



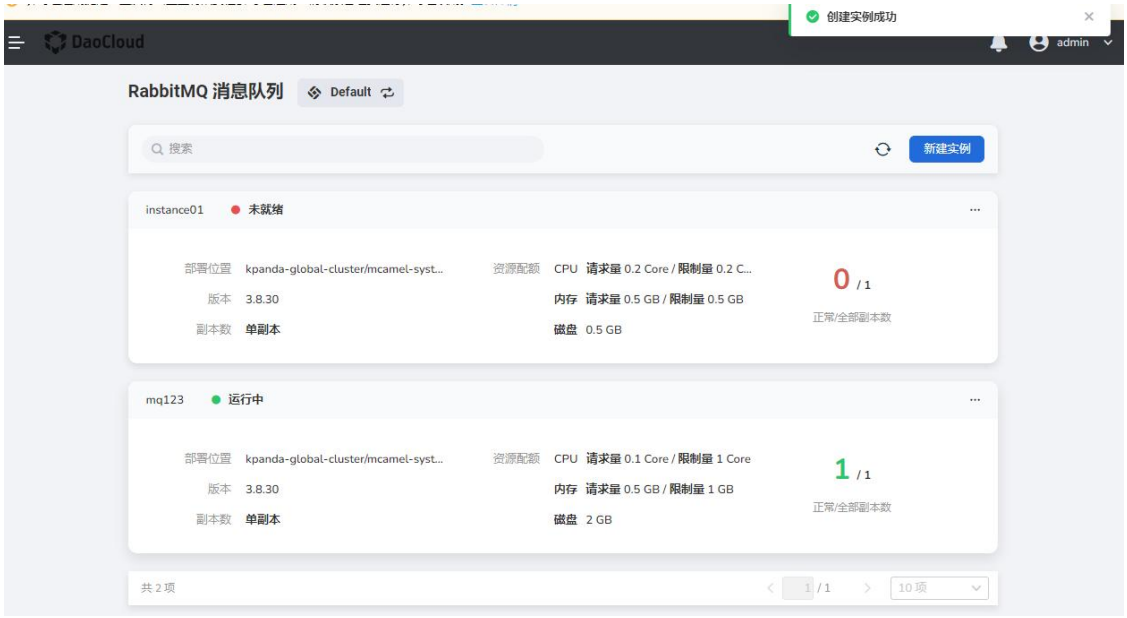
服务设置

5. 确认实例信息无误，点击确认完成创建。



确认

6. 在实例列表页查看实例是否创建成功。刚创建的实例状态为未就绪，等几分钟后该状态变为运行中。

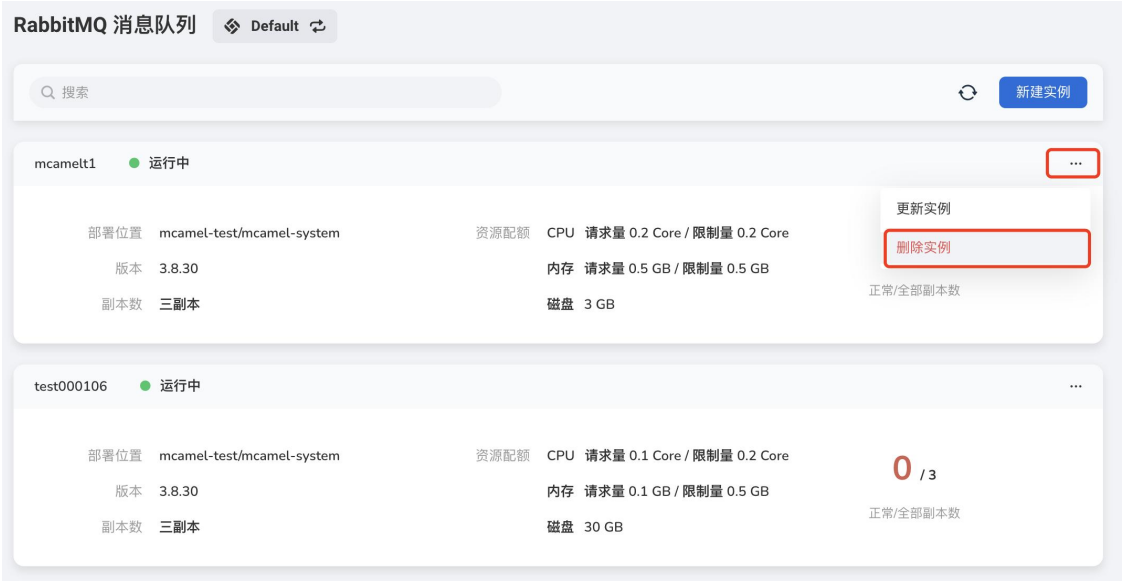

创建实例成功
DaoCloud
admin
RabbitMQ 消息队列
Default
搜索
新建实例
instance01
未就绪
部署位置 kpanda-global-cluster/mcamel-syst...
版本 3.8.30
副本数 单副本
资源配额 CPU 请求量 0.2 Core / 限制量 0.2 C...
内存 请求量 0.5 GB / 限制量 0.5 GB
磁盘 0.5 GB
0 / 1
正常/全部副本数
mq123
运行中
部署位置 kpanda-global-cluster/mcamel-syst...
版本 3.8.30
副本数 单副本
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.5 GB / 限制量 1 GB
磁盘 2 GB
1 / 1
正常/全部副本数
共 2 项
1 / 1
10 项

状态

删除 RabbitMQ

如果想要删除一个消息队列，可以执行如下操作：

1. 在消息队列中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



选择删除实例

2. 在弹窗中输入该消息队列的名称，确认无误后，点击删除按钮。



点击删除

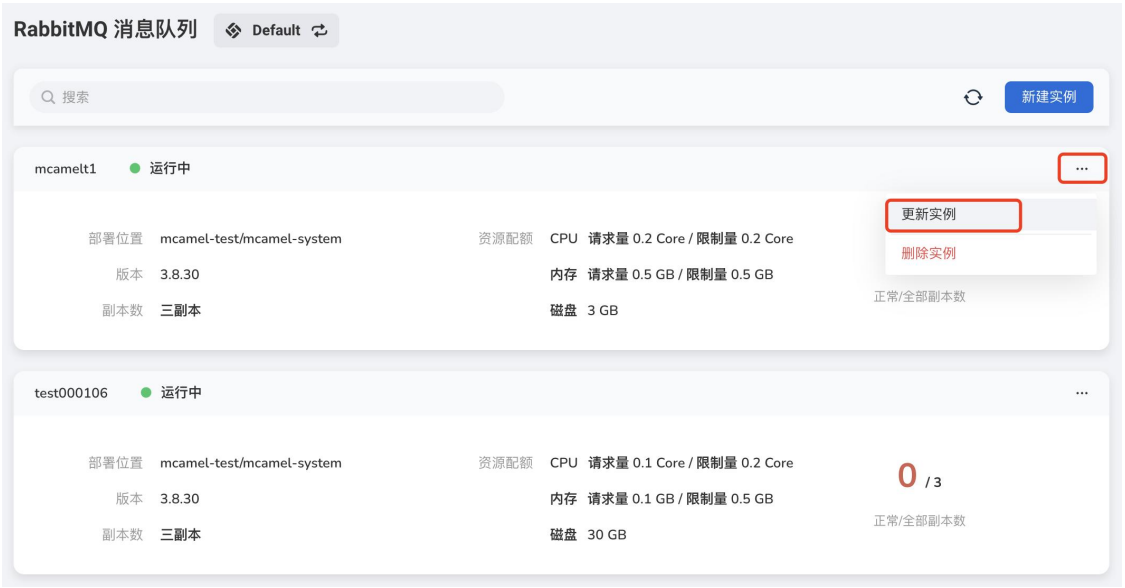
!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。

更新 RabbitMQ

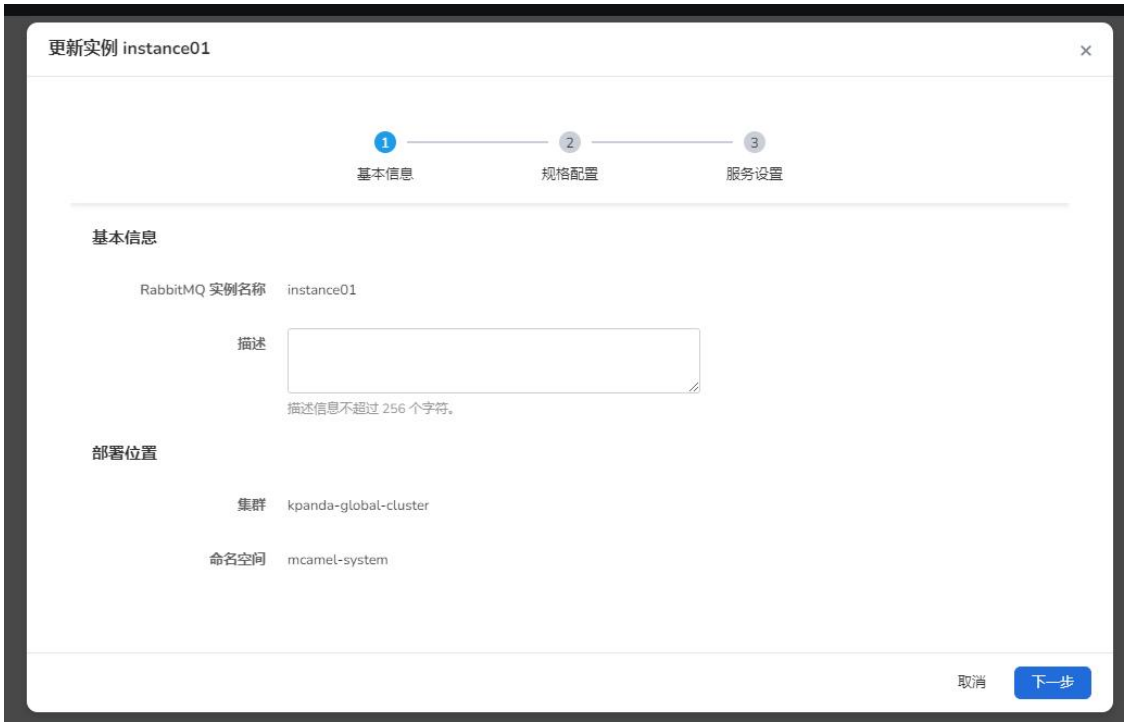
如果想要更新或修改 RabbitMQ 的资源配置，可以按照本页说明操作。

1. 在消息队列中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



选择更新实例

2. 修改基本信息后，点击下一步。



基本信息

3. 修改规格配置后，点击下一步。



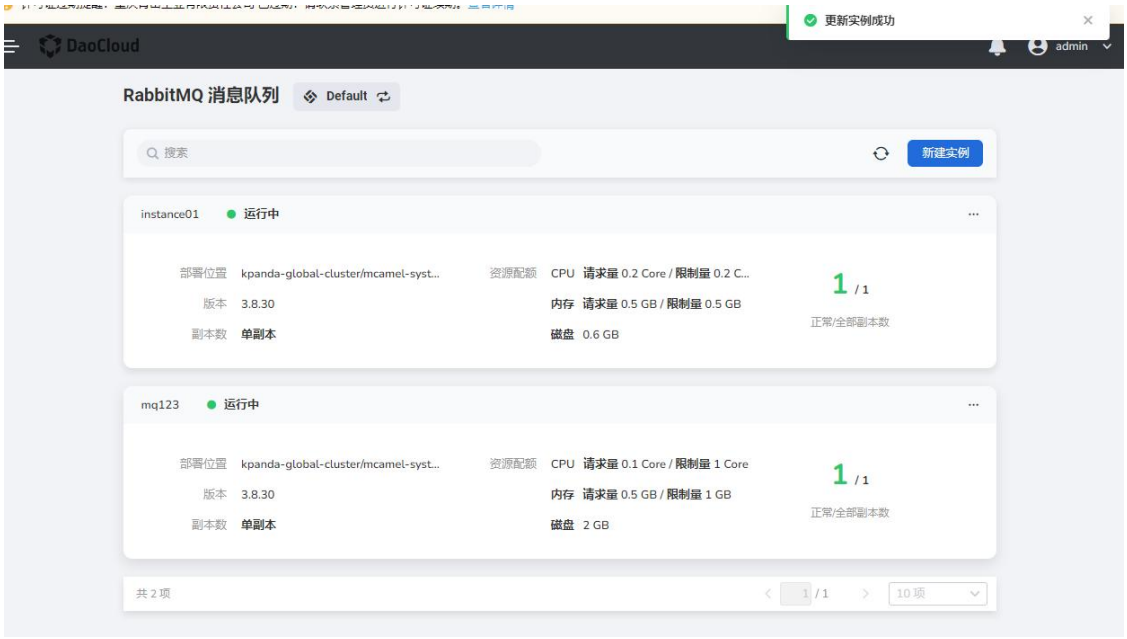
规格配置

4. 修改服务设置后，点击确认。



服务设置

5. 返回消息队列，屏幕右上角将显示消息：更新实例成功。


更新实例成功
DaoCloud
admin
RabbitMQ 消息队列
Default
搜索
新建实例
instance01
运行中
部署位置 kpanda-global-cluster/mcamel-syst...
版本 3.8.30
副本数 单副本
资源配额 CPU 请求量 0.2 Core / 限制量 0.2 C...
内存 请求量 0.5 GB / 限制量 0.5 GB
磁盘 0.6 GB
1 / 1
正常/全部副本数
mq123
运行中
部署位置 kpanda-global-cluster/mcamel-syst...
版本 3.8.30
副本数 单副本
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.5 GB / 限制量 1 GB
磁盘 2 GB
1 / 1
正常/全部副本数
共 2 项
1 / 1
10 项

成功

为 RabbitMQ 添加自定义插件

问题描述

RabbitMQ 有很多插件，但是在安装 RabbitMQ 时，只能安装默认的插件，如果需要安装其他插件，需要在安装完成后，手动安装。

解决方案

在创建的 RabbitMQ 的 yaml 中，增加 initContainer，用于下载插件，然后将插件挂在到 RabbitMQ 的插件目录中，最后在 rabbitmq 的 config 中，增加了插件的配置，并在启用的插件模块中启用插件。

示例

这里以 rabbitmq_message_timestamp-3.8.0.ez 为例，主要做的内容如下：

- 增加了 initContainer，用于下载插件
- 然后将插件挂在到 RabbitMQ 的插件目录中
- 在 rabbitmq 的 config 中，增加了插件的配置
- 并在启用的插件模块中启用插件

修改示例实例代码

```yaml
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  ...
status:
  ...
spec:
```

```
  image: docker.m.daocloud.io/library/rabbitmq:3.9.25-management-alpine
  override:
    service:
      spec:
        ports:
          - name: amqp
            port: 5672
            protocol: TCP
            targetPort: 5672
          - name: management
            port: 15672
            protocol: TCP
            targetPort: 15672
          - name: prometheus
            port: 15692
            protocol: TCP
            targetPort: 15692
    statefulSet:
      spec:
        template:
          spec:
            # 以下为增加部分
+           volumes:
+             - name: community-plugins
+               emptyDir: { }
+           initContainers:
+             - command:
+                 - sh
+                 - -c
+                 - curl -L -v https://github.com/rabbitmq/rabbitmq-message-timestamp/releases/download/v3.8.0/rabbitmq_message_timestamp-3.8.0.ez --output rabbitmq_message_timestamp-3.8.0.ez
```

```
+                image: docker.m.daocloud.io/curlimages/curl:7.70.0
+                imagePullPolicy: IfNotPresent
+                name: copy-community-plugins
+                resources:
+                  limits:
+                    cpu: 100m
+                    memory: 500Mi
+                  requests:
+                    cpu: 100m
+                    memory: 500Mi
+                terminationMessagePolicy: FallbackToLogsOnError
+                volumeMounts:
+                  - mountPath: /community-plugins/
+                    name: community-plugins
        # 以上为增加部分
            containers:
              - name: rabbitmq
                ports:
                  - containerPort: 5672
                    name: amqp
                    protocol: TCP
                  - containerPort: 15672
                    name: management
                    protocol: TCP
                  - containerPort: 15692
                    name: prometheus
                    protocol: TCP
                resources: {}
                # 以下为增加部分
+               volumeMounts:
+                 - mountPath: /opt/rabbitmq/community-plugins
+                   name: community-plugins
```

```
                    # 以上为增加部分
  persistence:
    storage: 1Gi
    storageClassName: hwameistor-storage-lvm-hdd
  rabbitmq:
+   envConfig: |
+     PLUGINS_DIR=/opt/rabbitmq/plugins:/opt/rabbitmq/community-plugins
    # 以上一行为增加部分
    additionalConfig: |

      log.console.level = info
      default_user=rabbitmq
      default_pass=UE81O6Y4^w$iWP86g
      cluster_partition_handling = pause_minority
      vm_memory_high_watermark_paging_ratio = 0.99
      disk_free_limit.relative = 1.0
      collect_statistics_interval = 10000
    additionalPlugins:
+     - rabbitmq_message_timestamp   # 增加次插件启用
      - rabbitmq_peer_discovery_k8s
      - rabbitmq_prometheus
      - rabbitmq_management
  replicas: 1
  resources:
    limits:
      cpu: 200m
      memory: 512Mi
    requests:
      cpu: 200m
      memory: 512Mi
  secretBackend: {}
  service:
```

```
    type: ClusterIP
  terminationGracePeriodSeconds: 604800
  tls: {}
```

这里的插件采用 github 官方作为下载地址，生成使用，建议自行维护插件位置，以免下载失败。

注意事项

目前支持手工在 YAML 编辑自定义资源的方式增加对应的插件，存在一定的操作风险性，建议谨慎操作。

MinIO 对象存储 Release Notes

本页列出 MinIO 对象存储的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.7.3

新功能

- **新增** UI 界面的权限访问限制。

2023-06-30

v0.7.0

新功能

- **新增** 对接全局管理审计日志模块

优化

- **优化** 监控图表，去除干扰元素并新增时间范围选择

2023-04-27

v0.5.1

新功能

- **新增** mcamel-minio 详情页面展示相关的事件
- **新增** mcamel-minio 支持自定义角色

优化

- **优化** mcamel-minio 调度策略增加滑动按钮

v0.4.1

发布日期：2023-03-28

API

- **修复** mcamel-minio 页面展示 LoadBalancer 地址错误。
- **修复** mcamel-minio 删除 MinIO 时不应该校验野生存储配置的问题。
- **修复** mcamel-minio 修复 创建 Bucket 偶尔会失败。

其他

- **升级** mcamel-minio golang.org/x/net 到 v0.7.0。

v0.3.0

发布日期：2023-02-23

API

- **新增** mcamel-minio helm-docs 模板文件。
- **新增** mcamel-minio 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-minio 支持 cloud shell。
- **新增** mcamel-minio 支持导航栏单独注册。
- **新增** mcamel-minio 支持查看日志。
- **新增** mcamel-minio Operator 对接 chart-syncer。
- **修复** mcamel-minio 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-minio 工作空间 Editor 用户无法查看实例密码。
- **升级** mcamel-minio 升级离线镜像检测脚本。

文档

- **新增** 日志查看操作说明，支持自定义查询、导出等功能。

v0.2.0

发布日期：2022-12-25

API

- **新增** mcamel-minio NodePort 端口冲突提前检测。
- **新增** mcamel-minio 节点亲和性配置。

- **修复** mcamel-minio 修复单实例的时候，状态显示异常的问题。
- **修复** mcamel-minio 创建实例时未校验名字。

UI

- **优化** mcamel-minio 取消认证信息输入框。

v0.1.4

发布日期：2022-11-28

- **改进** 更新获取前端的接口，增加 sc 列表是否可以扩容
- **改进** 密码校验调整为 MCamel 中等密码强度
- **新增** 创建 MinIO 集群时配置 Bucket
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警列表
- **新增** 校验 Service 注释
- **修复** 创建 MinIO 时，密码校验从 between 调整为 length
- **改进** 完善优化复制功能
- **改进** 实例详情 - 访问设置，移除 集群 IPv4
- **改进** 中间件密码校验难度调整
- **新增** minio 支持创建的时候内置 BUCKET 创建
- **新增** 对接告警能力
- **新增** 新增判断 sc 是否支持扩容并提前提示功能
- **优化** 优化安装环境检测的提示逻辑&调整其样式
- **优化** 中间件样式走查优化

v0.1.2

发布日期：2022-11-08

- **新增** 获取用户列表接口
- **新增** minio 实例创建

- **新增** minio 实例的修改
- **新增** minio 实例的删除
- **新增** minio 实例的配置修改
- **新增** minio 实例支持 nodeport 的 svc
- **新增** minio 实例的监控数据导出
- **新增** minio 实例的监控查看
- **新增** 多租户全局管理的对接
- **新增** mcamel-minio-ui 的创建/列表/修改/删除/查看
- **新增** APIServer/UI 支持 mtls
- **修复** 单例模式下，只有一个 pod，修复 grafana 无法获取数据问题
- **优化** 完善了计算器功能

MinIO 常见术语

本页说明 MinIO 相关概念。

概念	含义
Object 对象	存储的基本对象，例如文件、图片等
Bucket 桶	用于存储 Object 的逻辑空间，相互之间隔离，类似于系统中的顶层文件夹
Drive 驱动磁盘	即存储数据的磁盘，所有的对象数据都存储在 Drive 中
Set 磁盘集	即一组 Drive 的集合，分布式部署根据集群规模自动划分一个或多个 Set

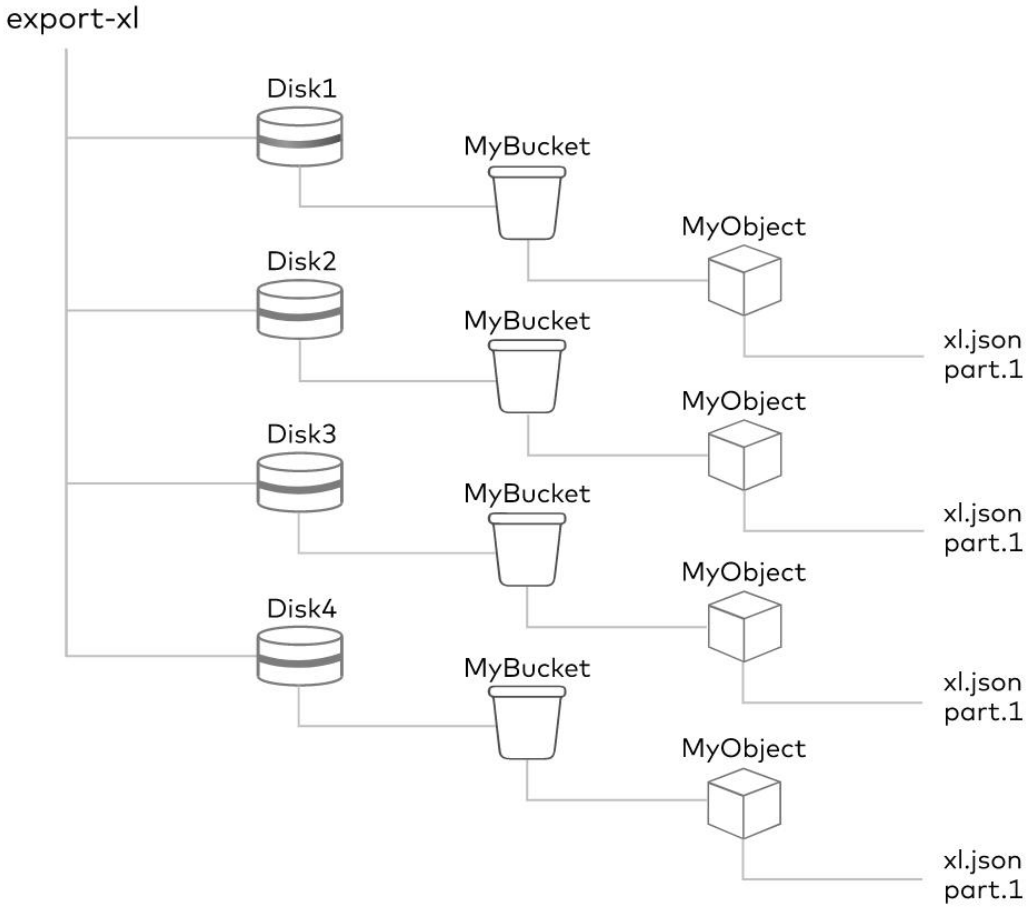
- 一个 Object 存储在一个 Set
- 一个集群划分为多个 Set
- 一个 Set 中的 Drive 尽可能分布在不同的节点上
- 一个 Set 包含的 Drive 数量是固定的，默认由系统根据集群规模自动计算

更多概念的阐述如下所述。

擦除码 Erasure Code

MinIO 使用按对象的嵌入式擦除编码保护数据，该编码以汇编代码编写，可提供最高的性能。 MinIO 使用 Reed-Solomon 代码将对象划分为 n/2 个数据和 n/2 个奇偶校验块，也可以将对象配置为任何所需的冗余级别。 这意味着在 12 个驱动器设置中，将一个对象分片为 6 个数据和 6 个奇偶校验块。 即使丢失了多达 5 个（（n / 2）—1）个驱动器（无论是奇偶校验还是数据），仍然可以从其余驱动器可靠地重建数据。 MinIO 的实现可确保即使丢失或无法使用多个设

备，也可以读取对象或写入新的对象。 最后，MinIO 的擦除代码位于对象级别，并且可以一次修复一个对象。



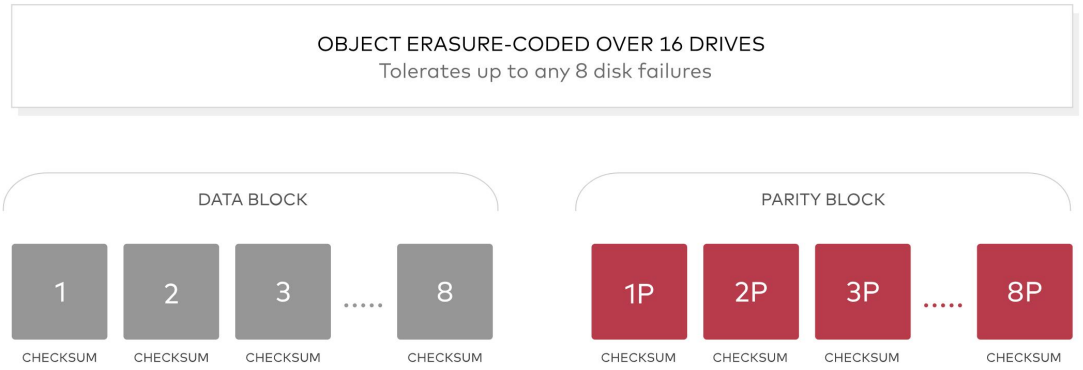
擦除码

Bitrot 保护

无声的数据损坏或 Bitrot 是磁盘驱动器面临的严重问题，导致数据在用户不知情的情况下损坏。 原因多种多样：驱动器老化，电流尖峰，磁盘固件错误，虚假写入，读/写方向错误，驱动程序错误，意外覆盖等。 但结果是一样的：数据泄漏。

MinIO 对高速哈希算法的优化实现可确保它永远不会读取损坏的数据，可以实时捕获和修复损坏的对象。 通过在 READ 上计算哈希值，并在 WRITE 上从应用程序、整个网络以及内存/驱动器的哈希值来确保端到端的完整性。 该实现旨在提

高速度，并且可以在 Intel CPU 的单个内核上实现超过每秒 10 GB 的哈希速度。



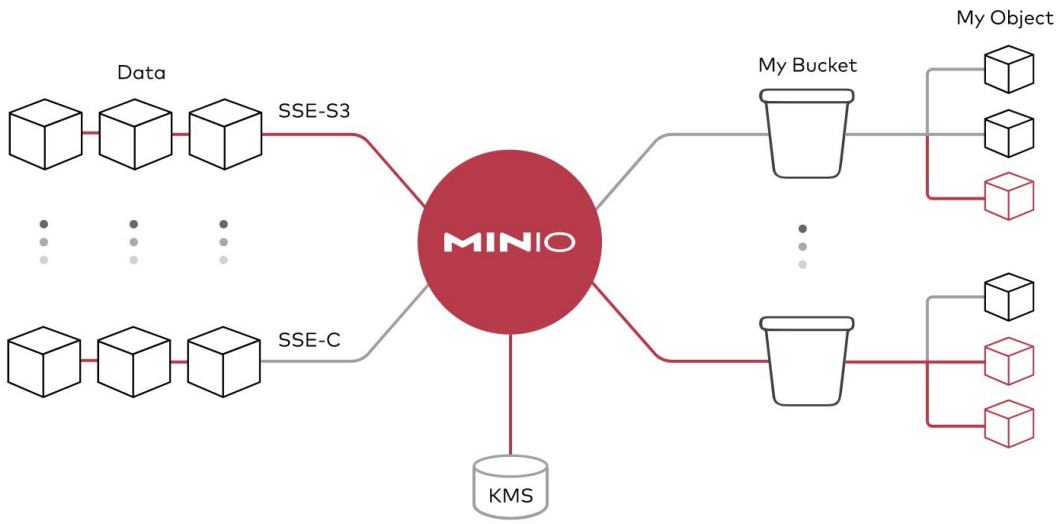
Bitrot 保护

加密

加密活跃数据与保护静态数据不同。 MinIO 支持多种复杂的服务器端加密方案，以保护所有位置的存储数据。 MinIO 的方法可确保机密性、完整性和真实性，而性能开销却可以忽略不计。 使用 AES-256-GCM、ChaCha20-Poly1305 和 AES-CBC 支持服务器端和客户端加密。 加密的对象使用 AEAD 服务器端加密进行了防篡改。 此外，MinIO 与所有常用的密钥管理解决方案（例如 HashiCorp Vault）兼容并经过测试。

MinIO 使用密钥管理系统（KMS）支持 SSE-S3。 如果客户端请求 SSE-S3 或启用了自动加密，则 MinIO 服务器会使用唯一的对象密钥对每个对象进行加密，该对象密钥受 KMS 管理的主密钥保护。 由于开销极低，因此可以为每个应用程序和实例打开自动加密。

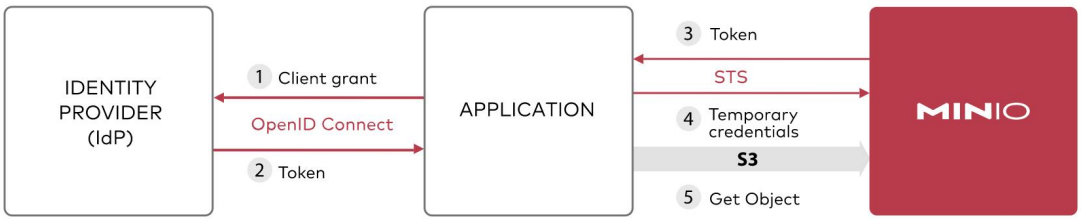
启用 WORM 后，MinIO 会禁用所有可能会使对象数据和元数据发生变异的 API。 这意味着一旦写入数据就可以防止篡改。这对于许多不同的法规要求具有实际应用。



加密

身份认证和管理

MinIO 支持身份管理中最先进的标准，并与 OpenID connect 兼容提供商以及主要的外部 IDP 供应商集成。 这意味着访问是集中的，密码是临时的和轮换的，而不是存储在配置文件和数据库中。 此外，访问策略是细粒度的且高度可配置的，这意味着支持多租户和多实例部署变得简单。

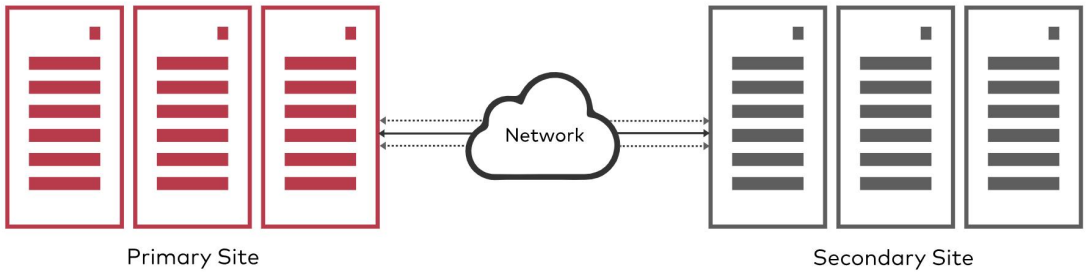


身份认证和管理

连续复制

传统复制方法的挑战在于它们无法有效扩展到几百 TB。 话虽如此，每个人都需要一种复制策略来支持灾难恢复，并且该策略需要跨越地域、数据中心和云。 MinIO 的连续复制旨在用于大规模的跨数据中心部署。通过利用 Lambda 计算通知和对象元数据，它可以高效、快速地计算增量。

Lambda 通知确保与传统的批处理模式相反，更改可以立即传播。 连续复制意味着即使发生高动态数据集，如果发生故障，数据丢失也将保持在最低水平。 最后，就像 MinIO 所做的一样，连续复制是多厂商的，这意味着您的备份位置可以是从 NAS 到公共云的任何位置。




```
mc mirror --watch --recursive dc1/ dc2/
```

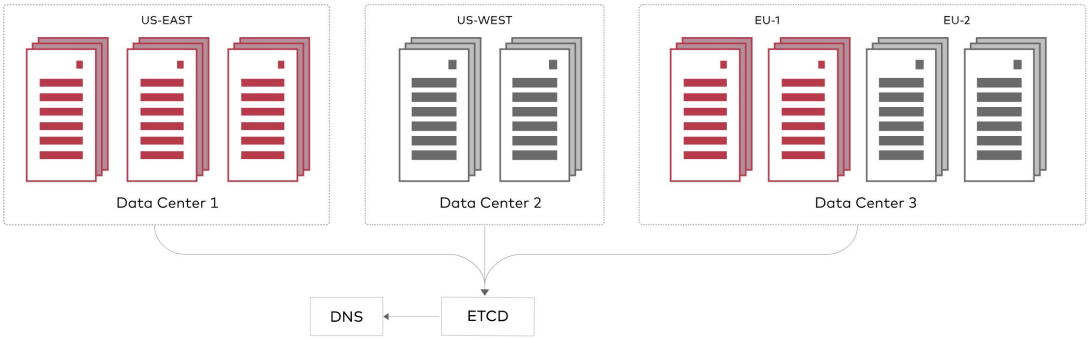
连续复制

全局一致性

现代企业到处都有数据。MinIO 允许将这些各种实例组合在一起以形成统一的全局命名空间。 具体来说，最多可以将 32 个 MinIO 服务器组合成一个分布式模式集，并且可以将多个分布式模式集组合成一个 MinIO 服务器联合。 每个 MinIO Server Federation 都提供统一的管理员和命名空间。

MinIO Federation Server 支持无限数量的分布式模式集。

这种方法的影响在于，对象存储可以为大型的、地理上分散的企业进行大规模扩展， 同时保留从以下位置容纳各种应用程序（S3 Select、MinSQL、Spark、Hive、Presto、TensorFlow、H20）的能力。具有单一控制台。



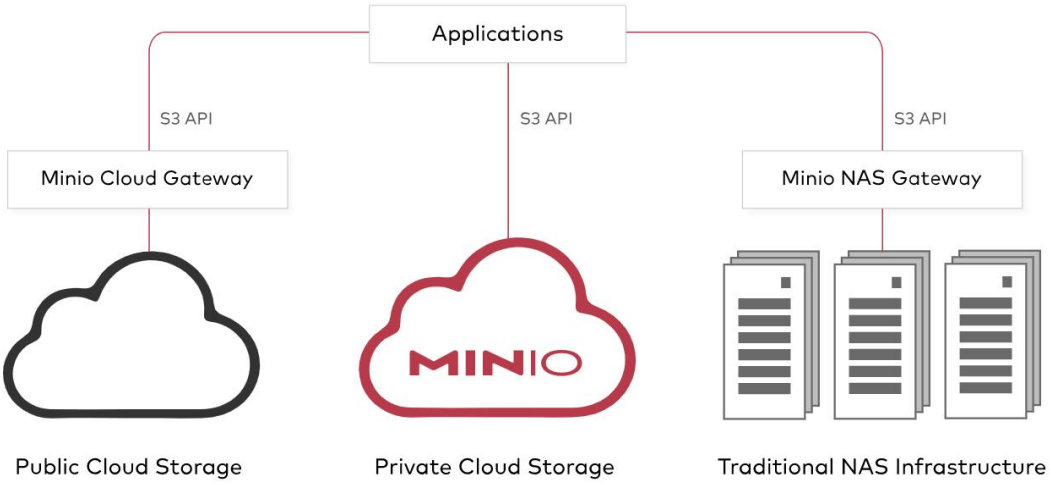
全局一致性

多云网关

所有企业都在采用多云策略，这也包括私有云。 因此，您的裸机虚拟化容器和公共云服务（包括 Google、Microsoft 和阿里等非 S3 提供商）必须看起来完全相同。 尽管现代应用程序具有高度的可移植性，但为这些应用程序提供支持的数据却并非如此。

MinIO 应对的主要挑战是：无论数据位于何处都都能使数据可用。 MinIO 在裸机、网络连接存储和每个公共云上运行。 更重要的是，MinIO 通过 Amazon S3 API 从应用程序和管理角度确保您对数据的看法完全相同。

MinIO 可以走得更远，使您现有的存储基础架构与 Amazon S3 兼容。 其影响是深远的。现在，组织可以从文件到块真正统一其数据基础架构，所有这些都显示为可通过 Amazon S3 API 访问的对象，而无需迁移。


Applications
S3 API
S3 API
S3 API
Minio Cloud Gateway
Minio NAS Gateway
MINIO
Public Cloud Storage
Private Cloud Storage
Traditional NAS Infrastructure

多云网关

什么是 MinIO

MinIO 是一款热门、轻量、开源的对象存储方案，完美兼容 AWS S3 协议，友好支持 K8s。MinIO 是专为 AI 等云原生工作负载而设计的，可以为高性能的云原生数据机器学习、大数据分析、海量存储的基础架构等提供数据工作负载。

DCE 的 MinIO 模块是基于开源 MinIO 构建的对象存储中间件，即插即用。DaoCloud 为其开发了简单易用的图形化界面，向用户提供计算、存储和带宽资源独占的 MinIO 专享实例。

MinIO 之所以广受欢迎，是因为其具有以下特点：

1. 高性能。

   MinIO 是全球领先的对象存储服务先锋，目前在全世界有数百万的用户。在标准硬件上，读/写速度高达每秒几百 GB。

2. 可扩展性。

   MinIO 借鉴了 Web 缩放器，为对象存储带来了简单的扩缩模型。在部署 MinIO 时，扩展从单个群集开始。

3. 云原生支持。

   MinIO 是在过去几年内从 0 开始打造的一款存储方案，符合一切云原生计算的架构和构建流程，并且包含最新的云计算技术和理念。

4. 纯开源。

   MinIO 基于 Apache V2 license 100% 开放源代码。这就意味着 MinIO 的客户能够自动、无限制、自由免费地使用和集成 MinIO、自由创新和创造、自由修改和完善、自由再次发行新版本和组合软件。

5. 兼容 S3 存储。

AWS 的 S3 API（接口协议）是在全球范围内达到共识的对象存储协议，是全世界内大家都认可的标准。

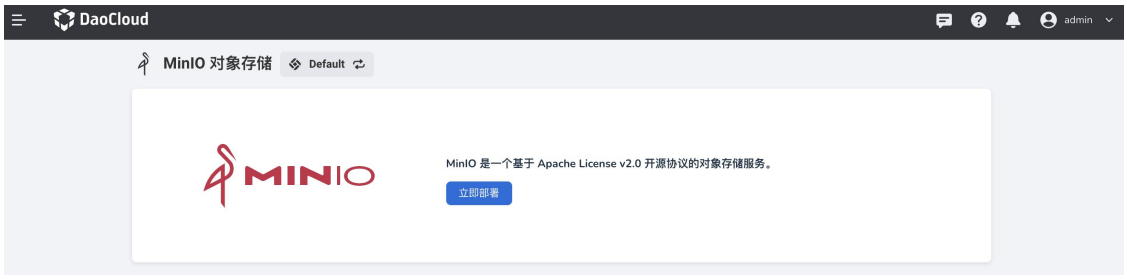
6. 极简。

   极简主义是 MinIO 的指导性设计原则。简单性减少了出错的机会，提高了正常运行时间，提供了可靠性，同时简单性又是性能的基础。

7. 支持多云。

   可以创建数以百万计的实例部署在私有云、公有云和边缘计算环境。

MinIO 为云原生而设计，可以作为轻量级容器运行，由外部编排服务（如 Kubernetes）管理。整个服务器约为几十 MB 的静态二进制文件，即使在高负载下也可以高效利用 CPU 和内存资源，使得企业可以在共享硬件上共同托管大量租户。



MinIO 主界面

创建 MinIO 实例

使用场景

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。 它兼容 AWS S3 云存储服务接口，非常适合于存储大容量非结构化的数据， 例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从 KB 到最大 TB 不等。

常见的使用场景有：

- 网盘：海量文件
- 社交网站：海量图片
- 电商网站：海量商品图片
- 视频网站：海量视频文件

每个 MinIO 集群都是分布式 MinIO 服务器的集合，每个节点一个进程。 MinIO 作为单个进程在用户空间中运行，并使用轻量级的协同例程来实现高并发。 将驱动器分组到擦除集（默认情况下，每组 16 个驱动器），然后使用确定性哈希算法将对象放置在这些擦除集上。

MinIO 专为大规模、多数据中心云存储服务而设计。 每个租户都运行自己的 MinIO 集群，该集群与其他租户完全隔离，从而使租户能够免受升级、更新和安全事件的任何干扰。 每个租户通过联合跨地理区域的集群来独立扩展。

实例监控

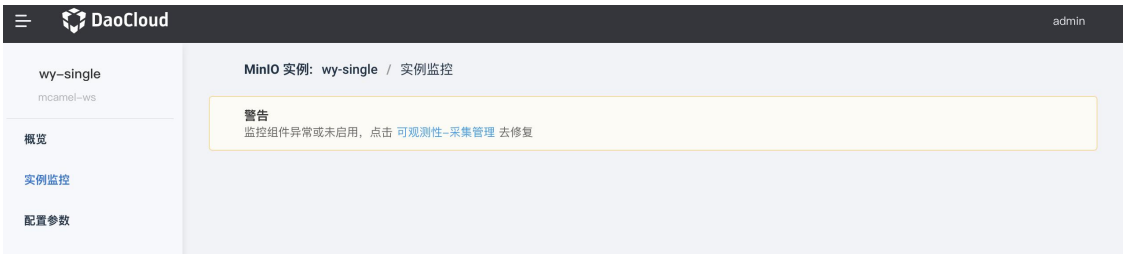
MinIO 内置了 Prometheus 和 Grafana 监控模块。

1. 在实例列表页面中，找到想要查看监控信息的实例，点击该实例的名称。



点击某个名称

2. 在左侧导航栏点击实例监控。

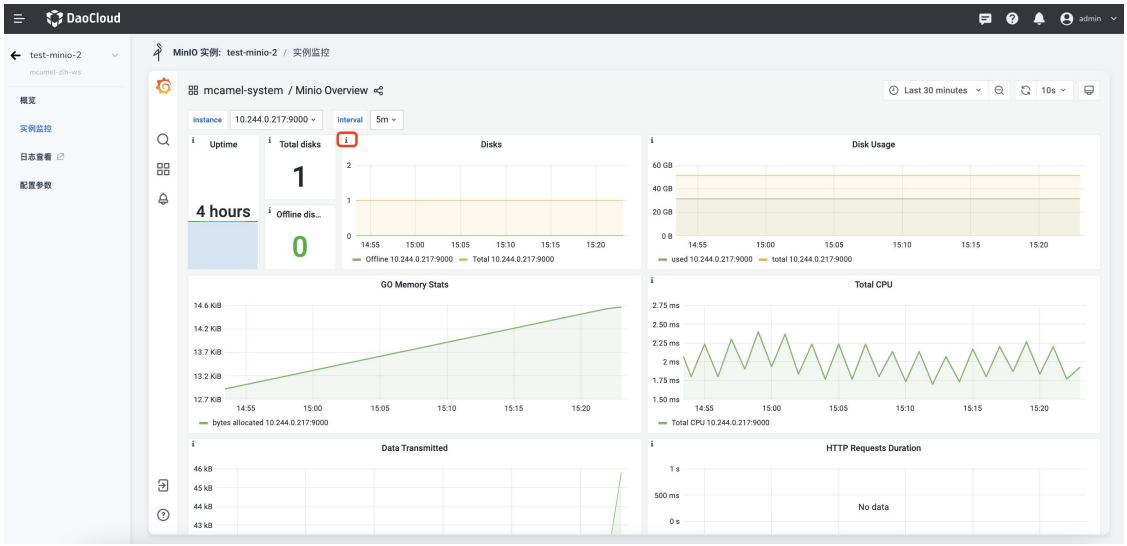


点击实例监控

!!! note

如果提示监控组件异常，请按提示[安装 insight-agent 插件](../../../insight/quickstart/install/install-agent.md)。

3. 查看实例的监控信息。点击红框里的信息符号可查看每个指标的含义说明。


DaoCloud
admin
test-minio-2
mcamel-zth-ws
概览
实例监控
日志查看
配置参数
MinIO 实例: test-minio-2 / 实例监控
mcamel-system / Minio Overview
Last 30 minutes
10s
instance
10.244.0.217:9000
interval
5m
Uptime
4 hours
Total disks
1
Offline dis...
0
Disks
Offline 10.244.0.217:9000
Total 10.244.0.217:9000
Disk Usage
used 10.244.0.217:9000
total 10.244.0.217:9000
GO Memory Stats
bytes allocated 10.244.0.217:9000
Total CPU
Total CPU 10.244.0.217:9000
Data Transmitted
HTTP Requests Duration
No data

点击实例监控

查看 MinIO 日志

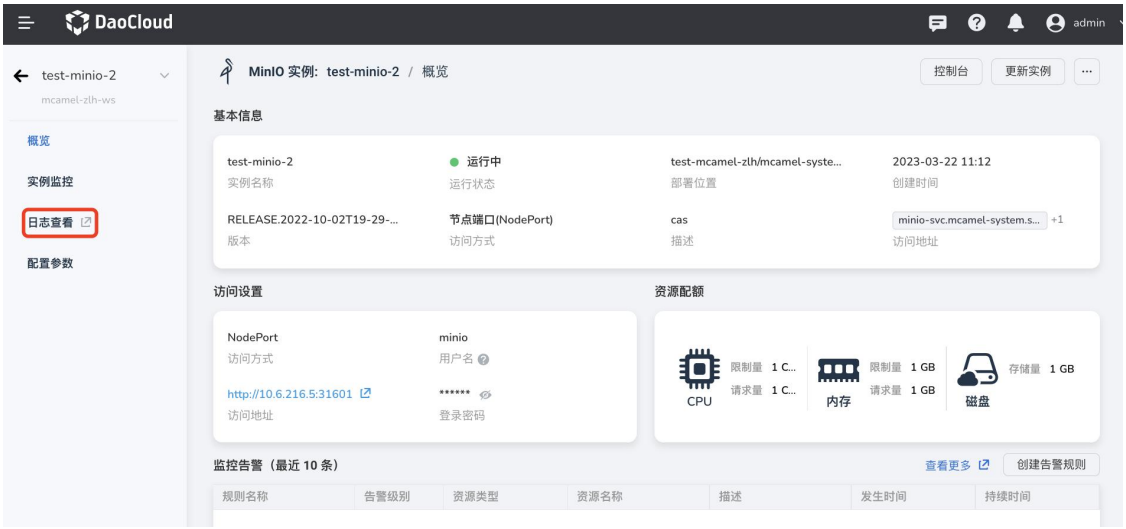
通过访问每个 MinIO 的实例详情页面；可以支持查看 MinIO 的日志。

1. 在 MinIO 实例列表中，找到想要查看日志的实例，点击实例名称进入详情页面。



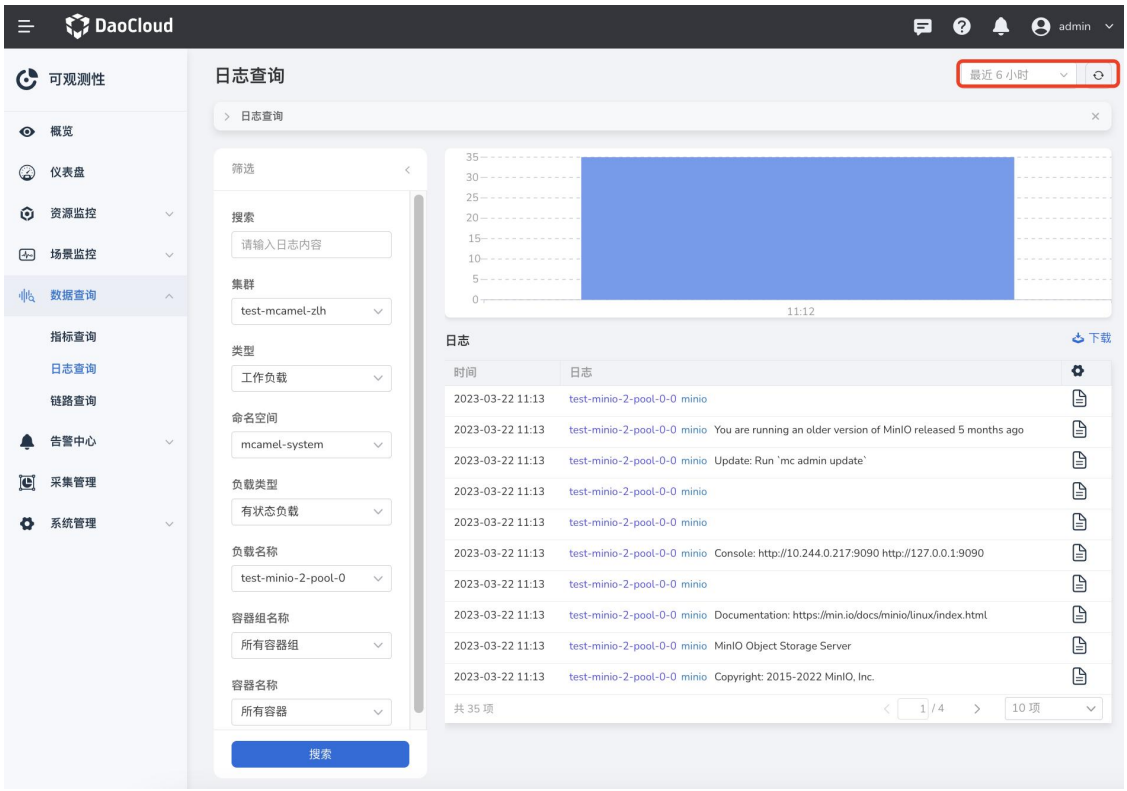
image

2. 在左侧菜单栏点击日志查看。



image

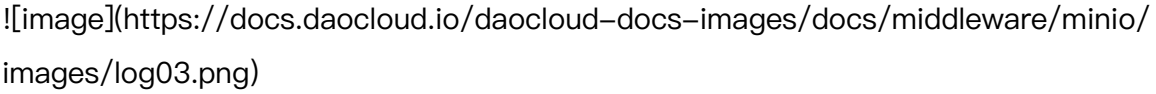
3. 根据需要调整日志查询的时间范围和刷新周期，具体可参考日志查询。



image

!!! note “常用操作”

* 自定义日志查询时间范围：在日志页面右上角可以切换日志的查询时间范围
* 通过关键字检索日志：在左侧`搜索框`下输入关键字即可查询带有特定内容的日志
* 查看日志的时间分布情况：通过柱状图查看在某一时间范围内的日志数量
* 查看日志的上下文：在日志最右侧点击 `查看上下文` 即可
* 导出日志：在柱状图下方点击`下载`即可



MinIO 的身份管理

DCE 5.0 提供的 MinIO 服务自带网页控制台（Web Console）。了解 MinIO 的身份管理（identity management）有助于快速了解如何在 MinIO 内安全有效地管理子账号。

本文简单介绍 MinIO 的身份管理规则，更多详细说明可参考 MinIO 的官方文档。

用户

默认情况下，MinIO 使用内置的 IDentity Provider（IDP）来完成身份管理。除了 IDP，还支持第三方 OIDC 和 LDAP 的方式。

用户由一对 username 和 password 组成。在 MinIO 的语境中，username 又被称为 access key（注意与后面 service account 层级的 access key 区分开来），password 又称为 secret key。

root 用户

在启动 MinIO 时，可以通过环境变量的方式设置 MinIO 集群中 root 用户的账号密码，分别是以下两个变量：

- MINIO_ROOT_USER
- MINIO_ROOT_PASSWORD

root 用户拥有所有资源的所有操作权限。

> 注意：如果要变更 root 用户，需要重启 MinIO 集群中所有的节点。

普通用户

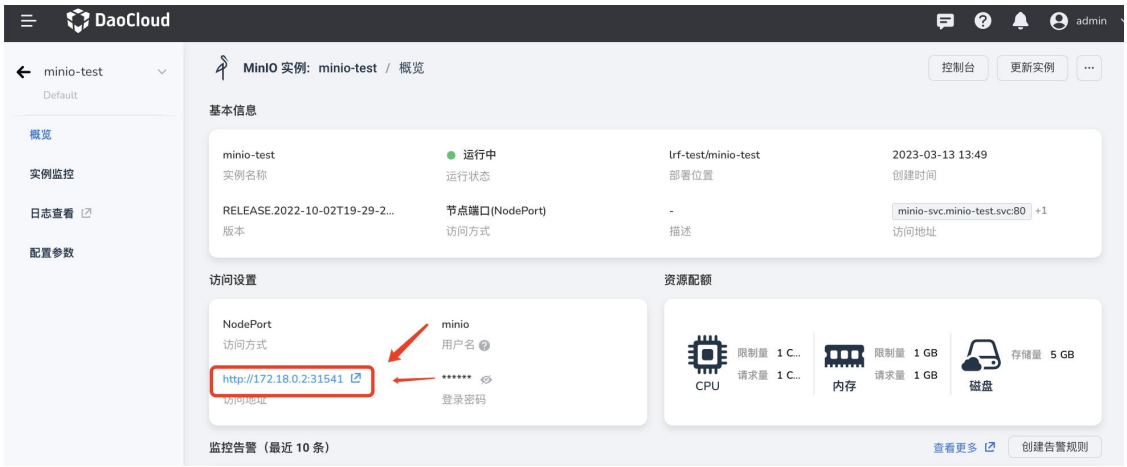
支持通过三种方式创建普通用户：

- Web Console，在 UI 界面中通过表单进行创建

- mc，使用 CLI 命令行创建
- Operator CR，使用 CR 进行创建

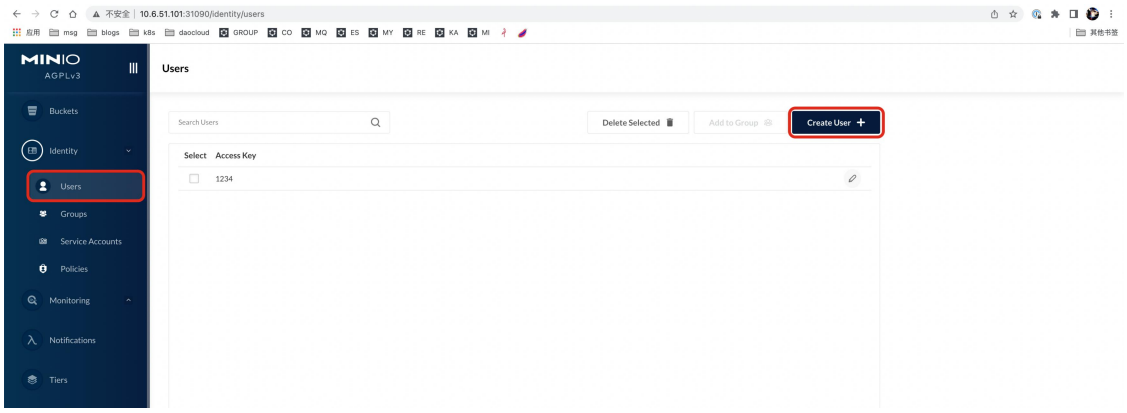
Console 创建

1. 在 DCE 5.0 的 MinIO 实例详情页面，点击访问地址，使用右侧的用户名和密码即可登录该实例的 Console 控制台。



登录 Console

2. 登录 Console 控制台之后根据下图所示，创建用户。



通过 Console 创建普通用户

mc 创建

需要事先安装 mc 命令，并配置连接到 MinIO 实例

创建用户：

ALIAS 指 MinIO 实例的别名

```
mc admin user add ALIAS ACCESSKEY SECRETKEY
```

授予权限：

USERNAME 指 MinIO 用户的用户名，即 ACCESSKEY

```
mc admin policy set ALIAS readwrite user=USERNAME
```

operator CR 创建

如果是通过 cr 安装 MinIO，也可以通过 users 字段来指定普通用户的 secret：

```
type TenantSpec struct {
    ....
    ....
    ....
    // *Optional* +
    //
    // An array of https://kubernetes.io/docs/concepts/configuration/secret/[Kubernetes opaque secrets] to use for generating MinIO users during tenant provisioning. +
    //
    // Each element in the array is an object consisting of a key-value pair `name: <string>`, where the `<string>` references an opaque Kubernetes secret. +
    //
    // Each referenced Kubernetes secret must include the following fields: +
    //
    // * `CONSOLE_ACCESS_KEY` - The "Username" for the MinIO user +
    //
    // * `CONSOLE_SECRET_KEY` - The "Password" for the MinIO user +
    //
    // The Operator creates each user with the `consoleAdmin` policy by default. You can change the assigned policy after the Tenant starts. +
    // +optional
```

```
    Users []*corev1.LocalObjectReference `json:"users,omitempty"`
    ....
    ....
    ....
}
```

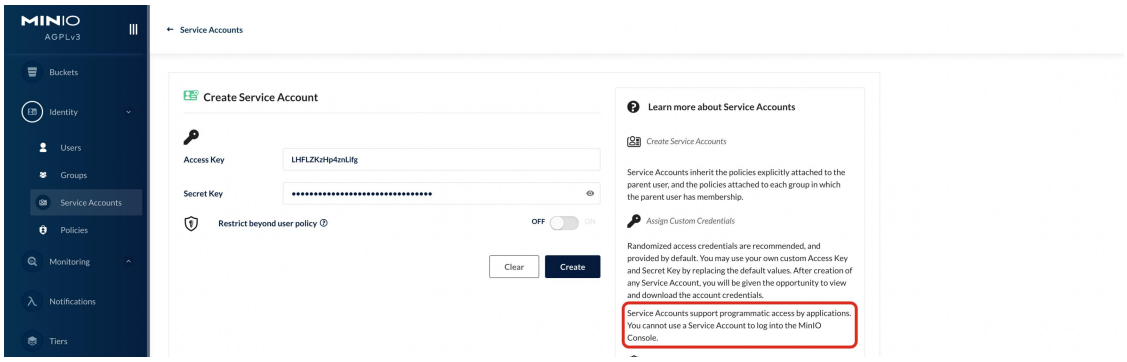
服务账号

服务账号（Service Account）通常使用用户登录 console 或者通过 mc 命令对 MinIO 进行管理操作。但如果应用程序需要访问 MinIO，则通常使用 Service Account（这是比较正式的叫法，某些上下文中也称之为 access key）。

一个用户可以创建多个 Service Account。

> 注意：无法通过 Service Account 登录 MinIO console，这也是它与用户最大的不同之处。

Console 创建



通过 console 创建 service account

mc 命令创建

```
mc [GLOBALFLAGS] admin user svcacct add       \
                                [--access-key]  \
                                [--secret-key]  \
                                [--policy]      \
                                ALIAS
                                USER
```

有关 MinIO 用户的详细说明，可参考 User Management

用户组

用户组，顾名思义即多个用户形成的集合。通过用户组结合授权策略可以批量管理一组用户的权限。通过授权策略可以为用户组分配资源权限，该组中的用户会继承用户组的资源权限。

MinIO 用户的权限分为两部分：用户原本具有的权限 + 从所在用户组继承而来的权限。在 MinIO 语境中，用户仅具有其明确被授予或从用户组继承而来的授权。如果用户没有明确获得（被直接授予或继承）某一资源的权限，则无法访问该资源。

有关 MinIO 用户组的详细说明，可参考 Group Management

授权策略

MinIO 使用基于策略的访问控制 (PBAC)来管理用户对哪些资源具有哪些权限。每条策略通过规定一些动作或条件来限制用户和用户组具有的权限。

内置策略

MinIO 内置了四种策略可以直接分配给用户或用户组。为用户/用户组授权时需要使用 mc admin policy set 命令，具体可参考 mc admin policy

- readonly：对 MinIO 副本中的所有存储桶和存储对象具有 **只读** 权限
- readwrite：对 MinIO 副本中的所有存储桶和存储对象具有 **读写** 的权限
- diagnostics：对 MinIO 副本具有 **诊断** 权限
- writeonly：对 MinIO 副本中的所有存储桶和存储对象具有 **只写** 权限

策略文件示例

MinIO 授权策略文件的模式和亚马逊云 IAM Policy 相同。

```
{
    "Version" ： "2012-10-17",
```

```
    "Statement" : [
        {
            "Effect" : "Allow",
            "Action" : [ "s3:<ActionName>", ... ],
            "Resource" : "arn:aws:s3:::*",
            "Condition" : { ... }
        },
        {
            "Effect" : "Deny",
            "Action" : [ "s3:<ActionName>", ... ],
            "Resource" : "arn:aws:s3:::*",
            "Condition" : { ... }
        }
    ]
}
```

有关 MinIO 授权策略的详细说明，可参考 Policy Management

配置参数

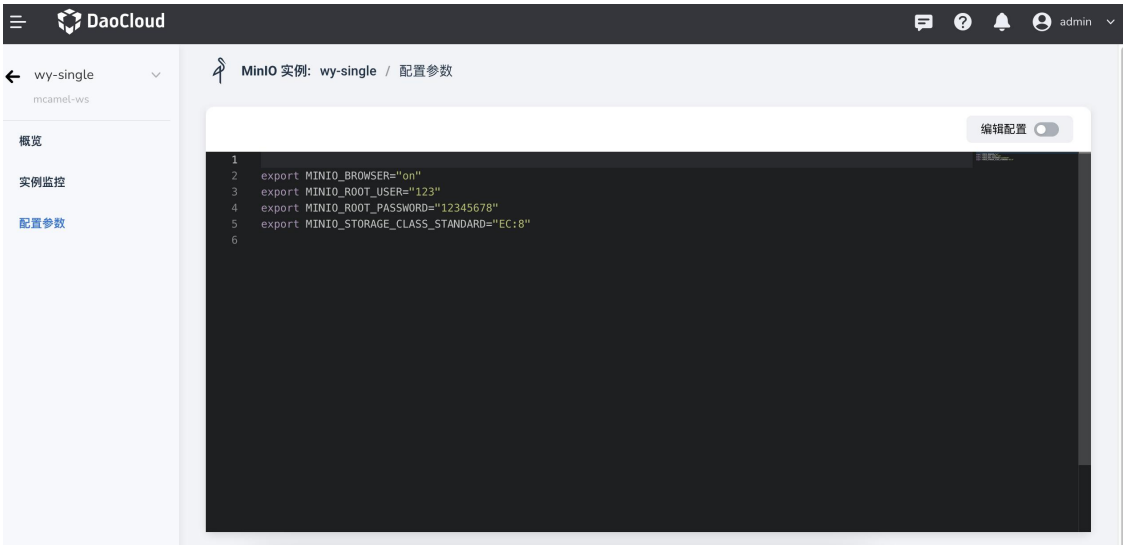
MinIO 内置了参数配置 UI 界面。

1. 在 MinIO 实例列表中，找到想要配置参数的 MinIO 实例，点击实例名称。



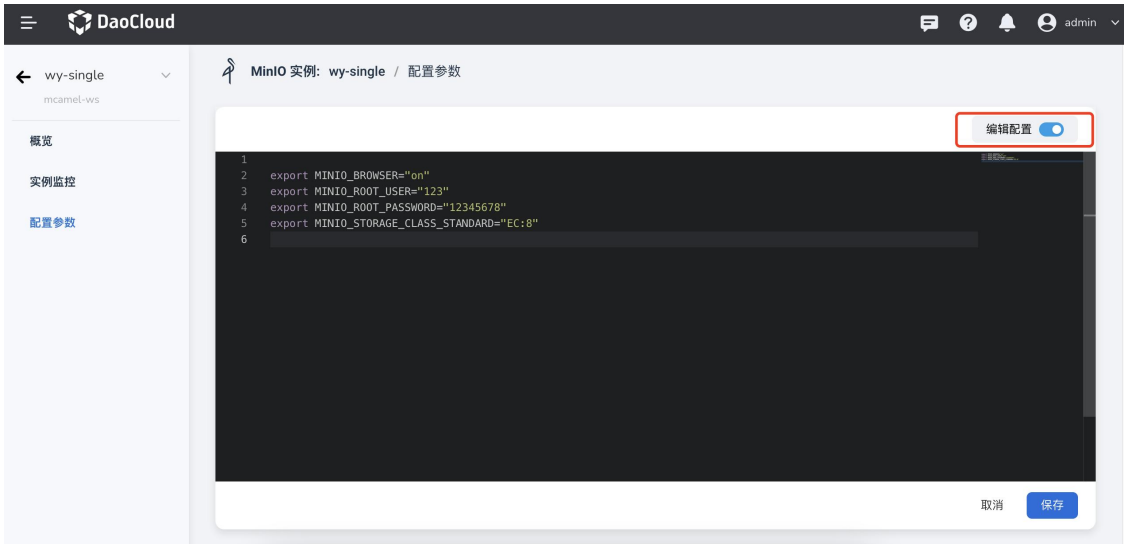
点击某个名称

2. 在左侧导航栏，点击配置参数。



点击配置参数

3. 打开编辑配置，对 MinIO 的各项参数进行添加、删除、修改。



配置参数

4. 点击保存，修改内容将立即生效。

创建 MinIO 实例

1. 从左侧导航栏选择 Minio 存储。



选择 minio 存储

2. 可以点击列表右上角的新建实例按钮。

   如果是首次使用，需要先选择工作空间，然后点击立即部署创建 MinIO 实例。



点击新建实例

3. 参考下方信息填写实例基本信息，然后点击下一步。

!!! note

- 实例名称、所在集群/命名空间在实例创建之后不可修改
- 注意查看输入框下方的填写要求，输入符合要求的内容
- 如未通过安装环境检测，需要根据提示安装组件之后方可进行下一步



基本信息

4. 参考下方信息填写配置规格，然后点击下一步。

    - 部署模式在实例创建之后不可更改
    - 生产模式下建议采用高可用部署模式
    - 高可用模式下需要至少 4 个副本

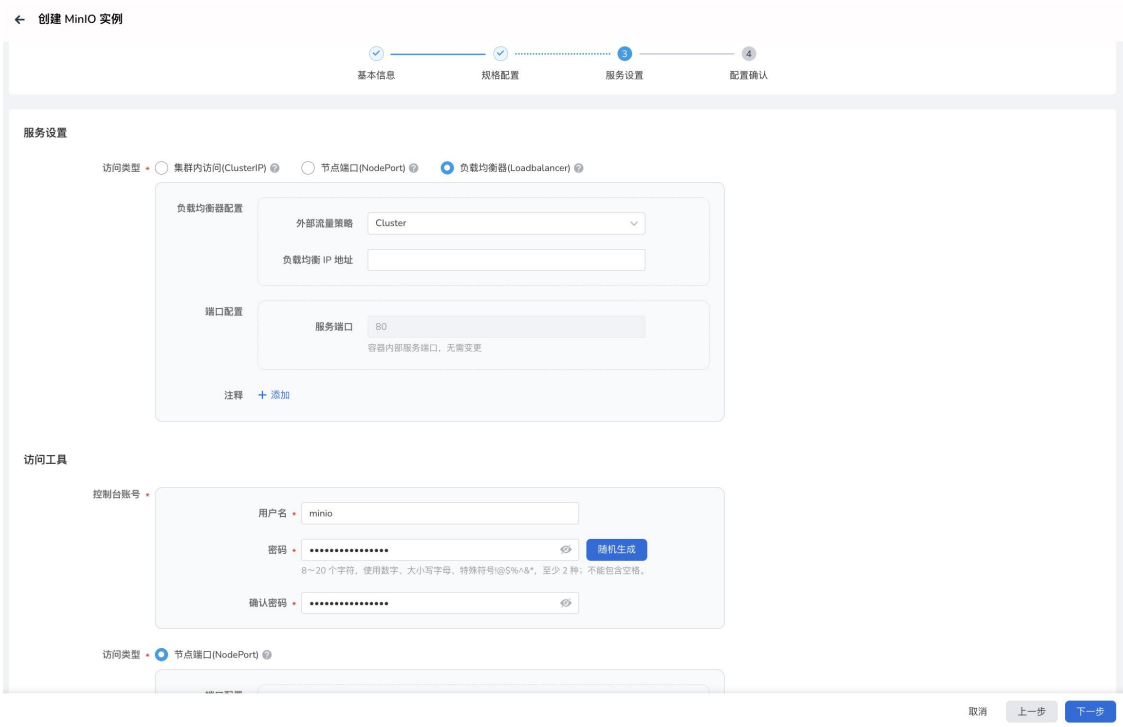
- 存储类：所选的存储类应有足够的可用资源，否则会因资源不足导致实例创建失败
- 存储容量：每个磁盘具有多少容量。**实例创建之后不可调低**
- 每副本磁盘数：为每个副本提供多少个次盘。**实例创建之后不可调低**



配置规格

5. 参考下方信息填写服务设置，点击下一步。
   - 集群内访问：只能在同一集群内部访问服务
   - 节点端口：通过节点的 IP 和静态端口访问服务，支持从集群外部访问服务
   - 负载均衡器：使用云服务提供商的负载均衡器使得服务可以公开访问

- 负载均衡器/外部流量策略：规定服务将外部流量路由到节点本地还是集群范围内的断点
    - Cluster：流量可以转发到集群中其他节点上的 Pod
    - Local：流量只能转发到本节点上的 Pod
- 控制台账号：访问此新建实例时需要用到的用户名、密码



访问模式

??? note “点击查看高级配置说明”

– Bucket 名称：在此实例下新建一个存储桶，设置新建存储桶的名称

– 调度策略/条件：设置 Pod 调度的节点亲和性，可参考 Kubernetes 官方文档[节点亲和性](https://kubernetes.io/zh-cn/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity)

– 尽量满足：先尝试调度到满足规则的节点。如果找不到匹配的节点，也会执行 Pod 调度

– 必须满足：只有找到满足规则的节点时才执行 Pod 调度

– 调度策略/权重：为满足每条调度策略的节点设置权重，优选使用权重高的策略。取值范围 1 到 100

– 调度策略/选择器

– In：节点必须包含所选的标签，并且该标签的取值必须 **属于** 某个取值集合。多个值用 `；` 隔开

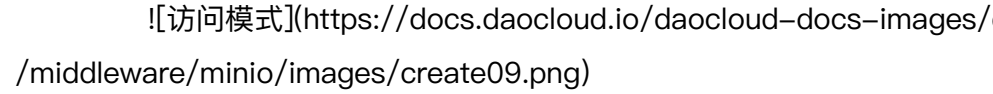
– NotIn：节点必须包含所选的标签，并且该标签的取值必须 **不属于** 某个取值集合。多个值用 `；` 隔开

– Exists：节点包含某个标签即可，不关注标签的具体取值

– DoesNotExists：节点不包含某个标签，不关注标签的具体取值

– Gt：节点必须包含某个标签，并且标签的取值必须大于某个整数

– Lt：节点必须包含某个标签，并且标签的取值必须小于某个整数



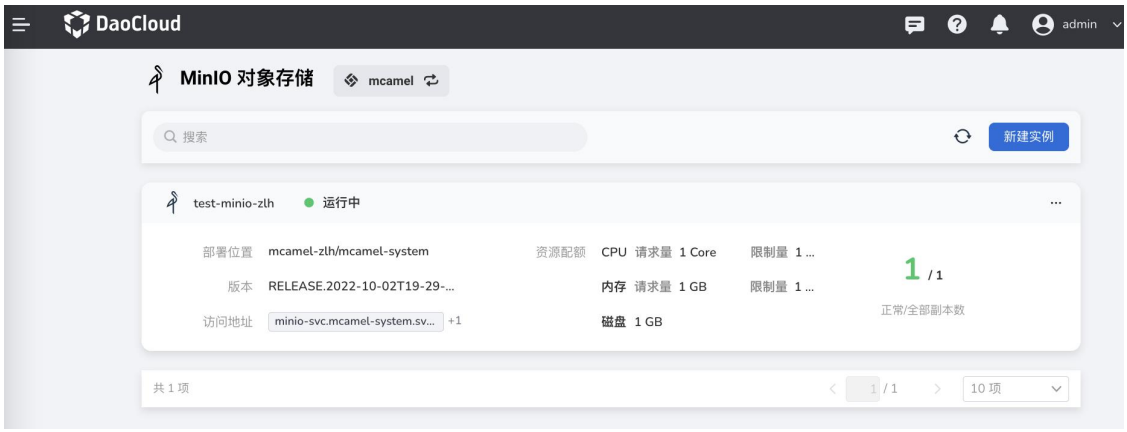
6. 确认实例配置信息无误，点击确认完成创建。



点击确认

7. 返回实例列表页查看实例是否创建成功。

   创建中的实例状态为未就绪，等所有相关容器成功启动之后状态变为运行中。

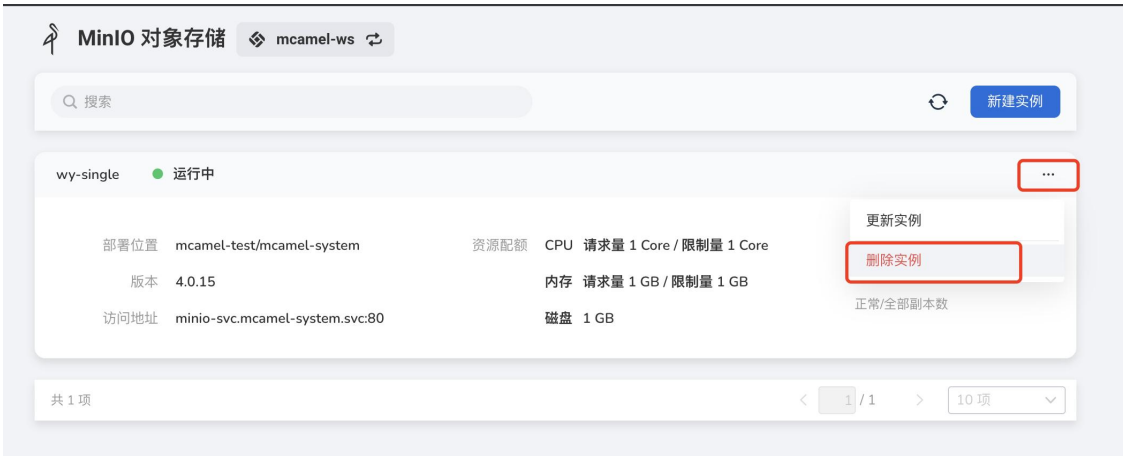

DaoCloud
admin
MinIO 对象存储
mcamel
搜索
新建实例
test-minio-zlh
运行中
部署位置
mcamel-zlh/mcamel-system
版本
RELEASE.2022-10-02T19-29-...
访问地址
minio-svc.mcamel-system.sv...
+1
资源配额
CPU 请求量 1 Core
限制量 1 ...
内存 请求量 1 GB
限制量 1 ...
磁盘 1 GB
1 / 1
正常/全部副本数
共 1 项
1 / 1
10 项

状态

删除 MinIO

如果想要删除一个实例列表，可以执行如下操作：

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



选择删除实例

2. 在弹窗中输入该实例列表的名称，确认无误后，点击删除按钮。

!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。



点击删除

更新 MinIO 实例

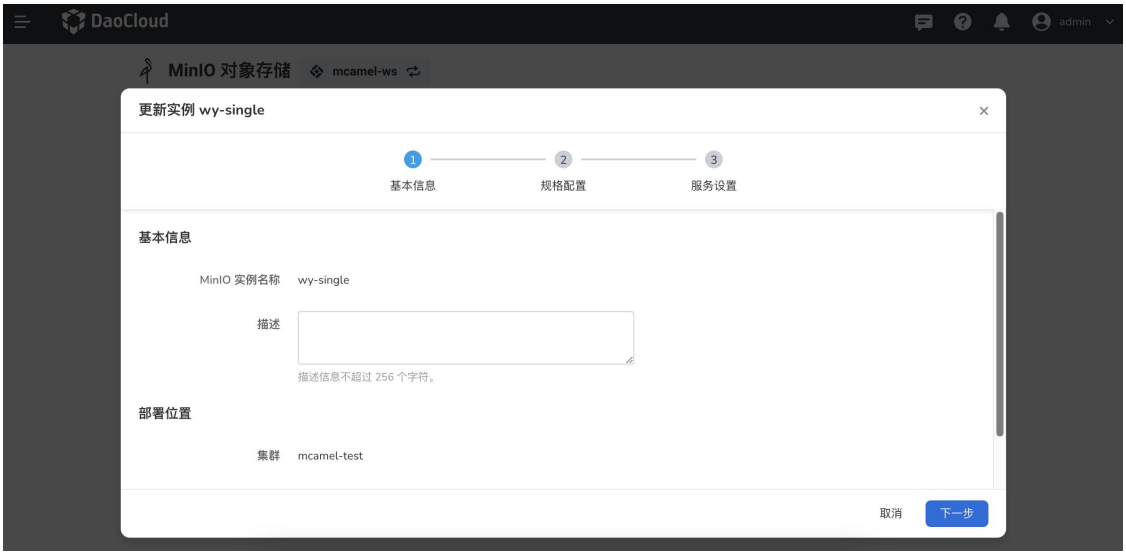
如果想要更新或修改 MinIO 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



选择更新实例

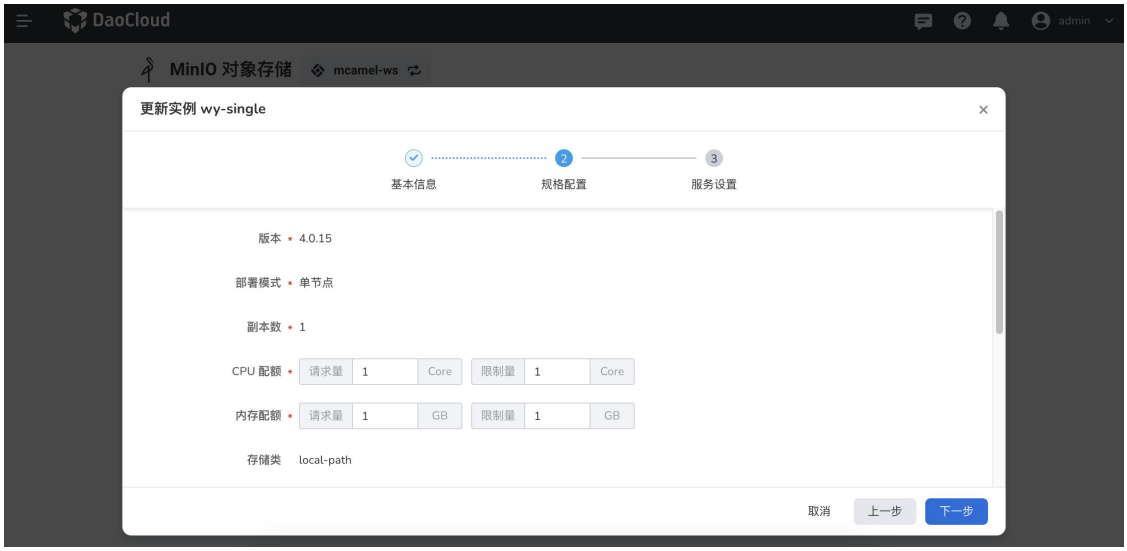
2. 修改基本信息，然后点击下一步。
   - 仅支持修改描述信息
   - 实例名称、部署位置不可修改



基本信息

3. 修改规格配置，然后点击下一步。

   - 仅支持修改：CPU 配额和内存配额
   - 不可修改：版本、部署模式、存储类、容量



规格配置

4. 修改服务设置，然后点击确认。



服务设置

5. 返回实例列表，屏幕右上角将显示消息：更新实例成功。



成功

MongoDB Release Notes

本页列出 MongoDB 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.2.0

新功能

- **新增** 对 log 目录所在 pvc 的配置能力。

优化

- **优化** 创建实例对话框增加反亲和默认标签，简化配置过程。
- **优化** 可以在非 Operator 所在命名空间创建 MongoDB 实例。
- **优化** 增加前端界面权限相关的展示信息

2023-06-30

v0.1.0

新功能

- **新增** Mcamel-MongoDB 支持 MongoDB 的实例创建、查看、删除等管理功能

什么是 MongoDB

MongoDB 是一种面向文档的 NoSQL 数据库管理系统，它以灵活的数据模型和可扩展性而闻名。下面是 MongoDB 的一些重要特点和概念：

面向文档的数据模型：MongoDB 存储数据的方式是以文档的形式，文档是一种类似于 JSON 的结构，它是一个键值对的集合，可以嵌套和包含多种类型的数据。这种灵活的数据模型使得 MongoDB 非常适合存储复杂的、非结构化的数据。

分布式架构和高可用性：MongoDB 能够水平扩展到多个服务器，以应对大规模数据和高并发请求。它支持主从复制和分片（sharding）机制，可以实现数据的冗余备份和负载均衡，提供高可用性和扩展性。

查询和索引：MongoDB 提供丰富的查询功能，支持多种查询条件和操作符。它还支持灵活的索引机制，可以根据需要创建多个索引来优化查询性能。

强大的数据处理功能：MongoDB 内置了许多数据处理功能，例如聚合管道（aggregation pipeline）、地理空间索引和查询、全文搜索等。这些功能使得 MongoDB 可以在数据库层面上处理复杂的数据操作和分析任务。

开发者友好性：MongoDB 提供了丰富的 API 和驱动程序，支持多种编程语言。它还具有简单而直观的命令行界面和图形用户界面，使开发人员能够快速上手和使用。

MongoDB 被广泛应用于 Web 应用、大数据分析、物联网、实时数据处理等领域，它的灵活性和可扩展性使得开发人员能够更高效地处理和存储不断增长的数据。


MongoDB
MongoDB 是一款基于分布式文件存储的数据库，类型介于关系数据库和非关系数据库之间，旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。
立即部署

创建

功能特性

MongoDB 拥有丰富的功能，以下是一些它的主要功能：

- 高可用性和冗余备份

  MongoDB 支持主从复制和副本集（replica set）机制，保证数据的冗余备份和高可用性。主从复制通过将数据从主节点复制到从节点，实现数据的持久性和故障恢复；副本集是一组互相通信的 MongoDB 实例，当主节点不可用时，自动选举新的主节点。

- 水平扩展和负载均衡

  MongoDB 支持分片（sharding）机制，可以将数据水平分割并分布在多个服务器上。这样可以实现数据的横向扩展，提高系统的存储容量和处理能力。MongoDB 提供自动的数据分布和负载均衡，使得数据的访问和查询在整个集群中得到均衡分配。

- 强大的查询和索引功能

  MongoDB 支持丰富的查询条件和操作符，可以进行灵活的查询。它还支持多种索引类型，包括单键索引、复合索引、文本索引和地理空间索引等，以提高查询性能和响应速度。

- 数据处理和聚合管道

  MongoDB 提供了强大的数据处理功能，通过聚合管道（aggregation pipeline）可以进行复杂的数据操作和分析。聚合管道允许串联多个操作，如过滤、排序、分组、计算等，以获得所需的结果。

- 地理空间和全文搜索

MongoDB 支持地理空间数据的存储和查询，可以进行地理位置相关的操作和搜索。同时，它还提供了全文搜索的功能，可以进行文本关键字的搜索和索引。

- 完整的事务支持

  MongoDB 从版本 4.0 开始引入了多文档事务的支持。这允许开发人员在一个或多个文档上执行原子性的读写操作，并保持数据的一致性。

- 安全性和权限管理

  MongoDB 提供了丰富的安全性功能，支持身份验证、访问控制和加密通信等。开发者可以为用户和角色定义不同的权限级别，以保护数据的安全性。

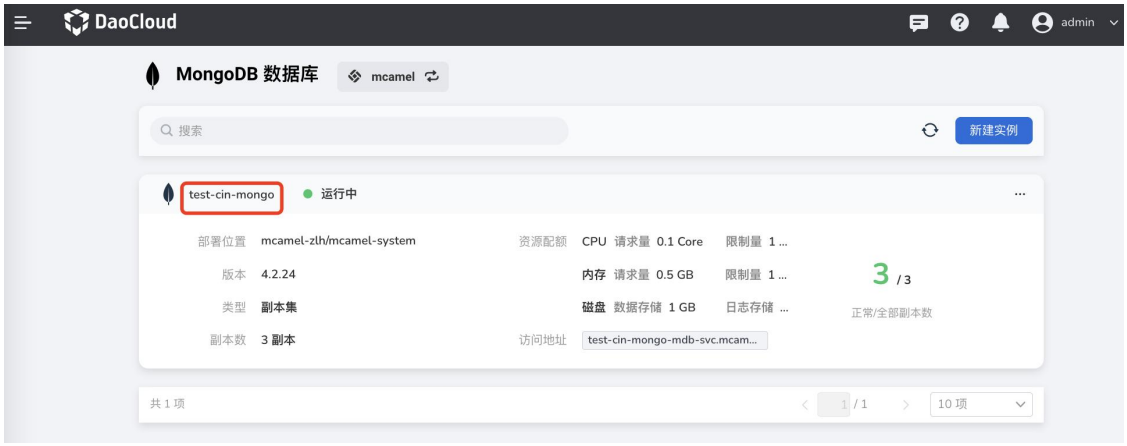
以上是 MongoDB 的一些主要功能，它的灵活性、可扩展性和丰富的功能使得开发人员能够应对各种不同的数据管理和处理需求。

查看 MongDB 日志

操作步骤

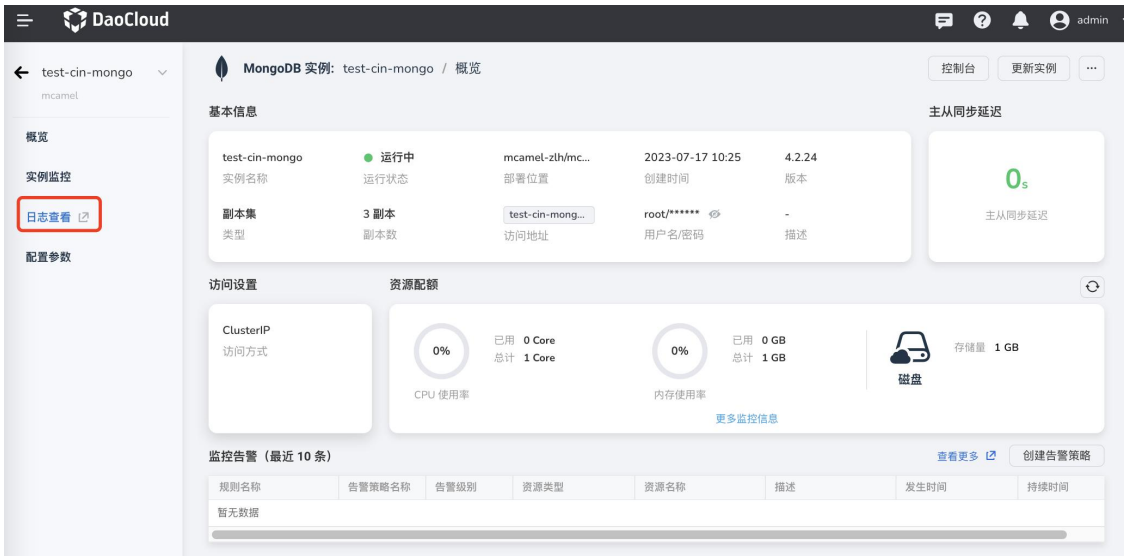
通过访问每个 MongDB 的实例详情页面，可以支持查看 MongDB 的日志。

1. 在 MongDB 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



日志

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



日志

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面可以很方便的进行日志查看。常用操作说明如下：

- 支持自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持查看日志的上下文，点击右侧 上下文 图标即可
- 支持导出日志

创建 MongoDB 实例

接入 MongoDB 数据库后，参照以下步骤创建 MongoDB 实例。

1. 在实例列表中，点击右上角的 新建实例。



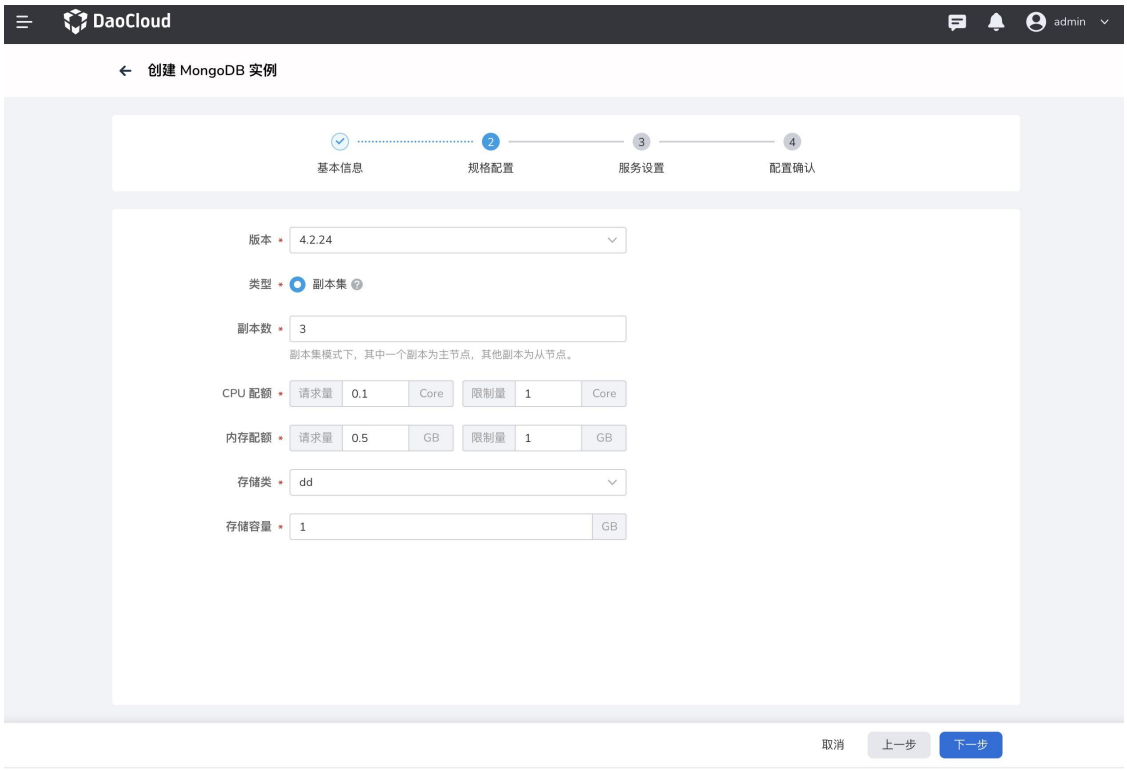
创建

2. 在创建 MongoDB 实例页面中，配置基本信息后，点击下一步。



创建

3. 选择部署类型、CPU、内存和存储等规格配置后，点击下一步。

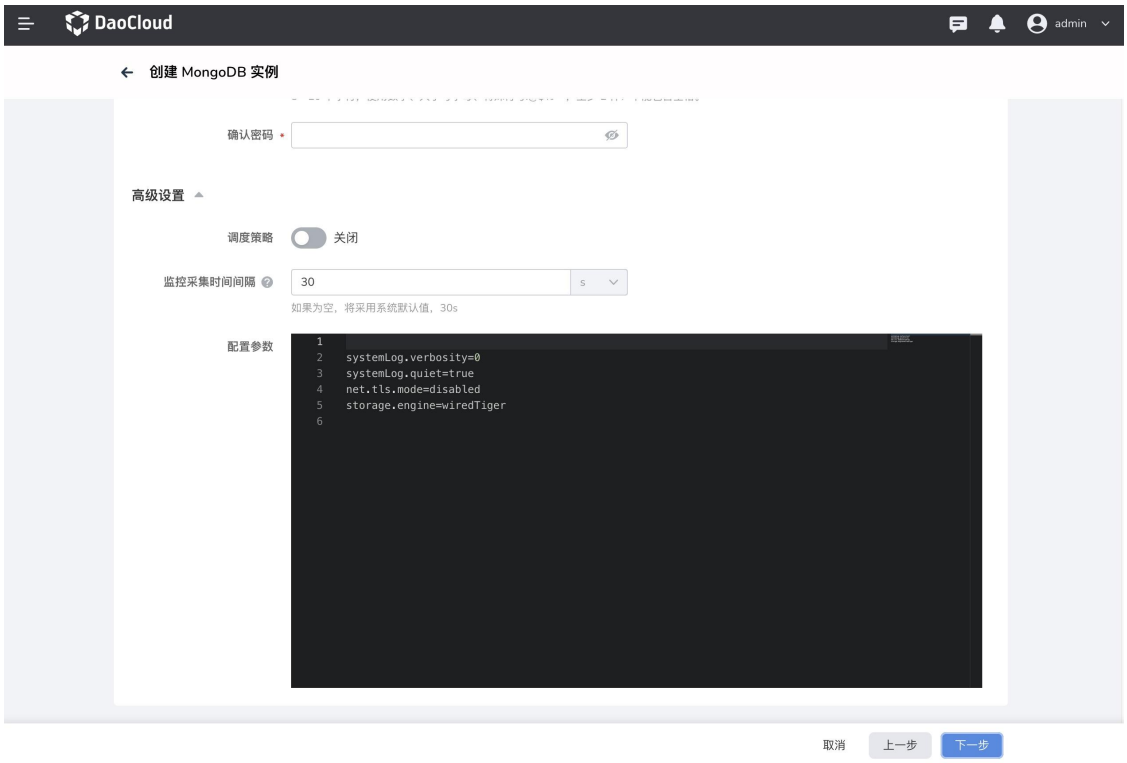


创建

4. 设置用户名和密码等服务设置，默认采用 ClusterIP 作为访问方式，用户还可以配置亲和性、配置参数等多项设置，可在通用设置中查看。

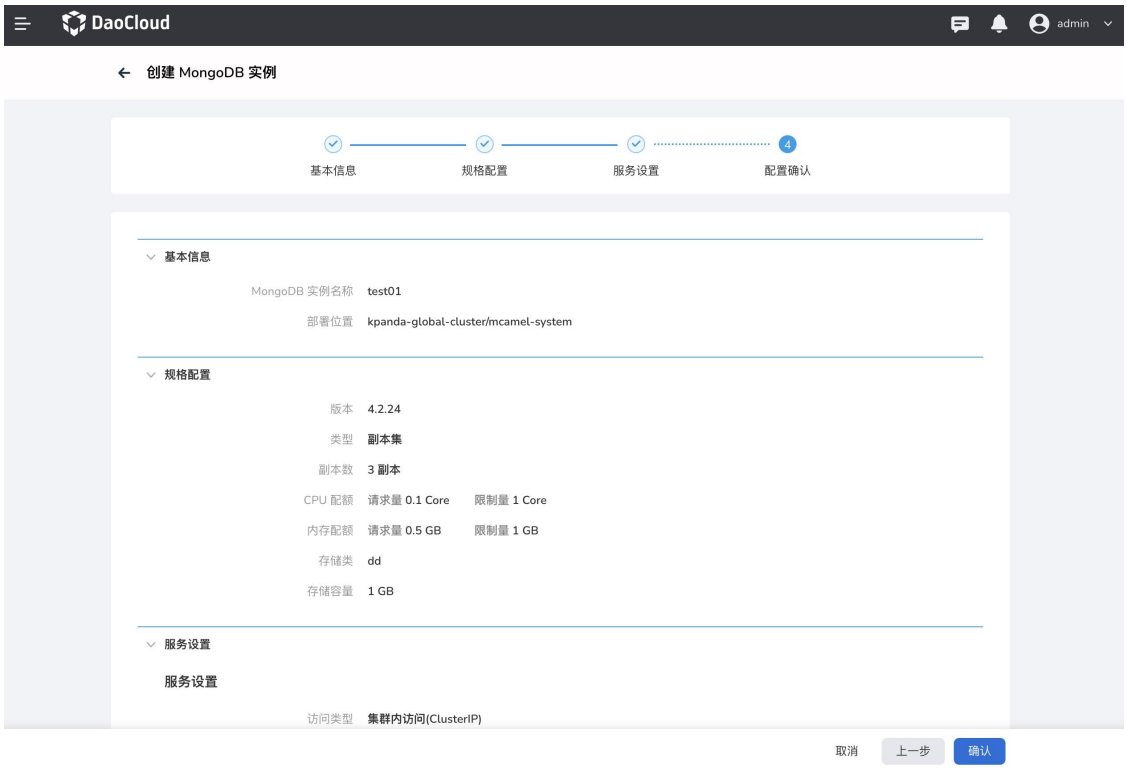

DaoCloud
admin
创建 MongoDB 实例
基本信息
规格配置
3
服务设置
4
配置确认
服务设置
访问类型 * 集群内访问(ClusterIP) 节点端口(NodePort) 负载均衡器(LoadBalancer)
端口配置
服务端口 27017
容器内部服务端口，无需变更
注释 + 添加
访问设置
用户名 * root
密码 * 随机生成
8～20 个字符，使用数字、大小写字母、特殊符号!@$%^*，至少 2 种；不能包含空格。
确认密码 *
高级设置
取消
上一步
下一步

创建



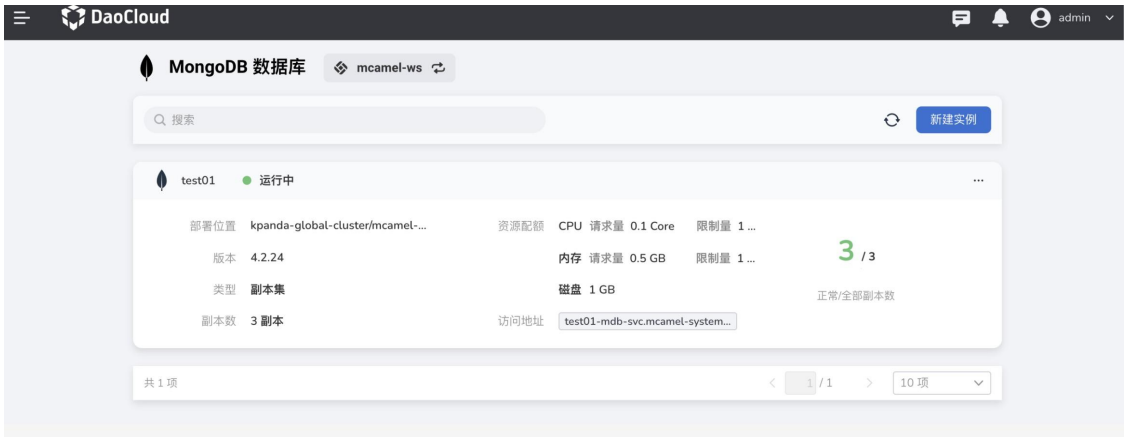
创建

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击确认。



创建

6. 返回实例列表，屏幕将提示创建实例成功。

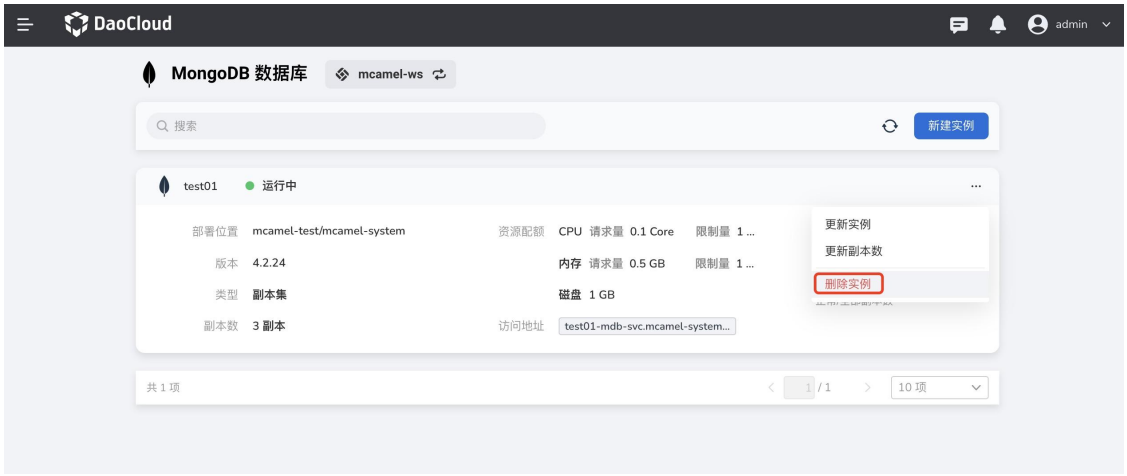


查看

删除 MongDB 实例

如果想要删除一个 MongDB 实例，可以执行如下操作：

1. 在 MongDB 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。



删除实例

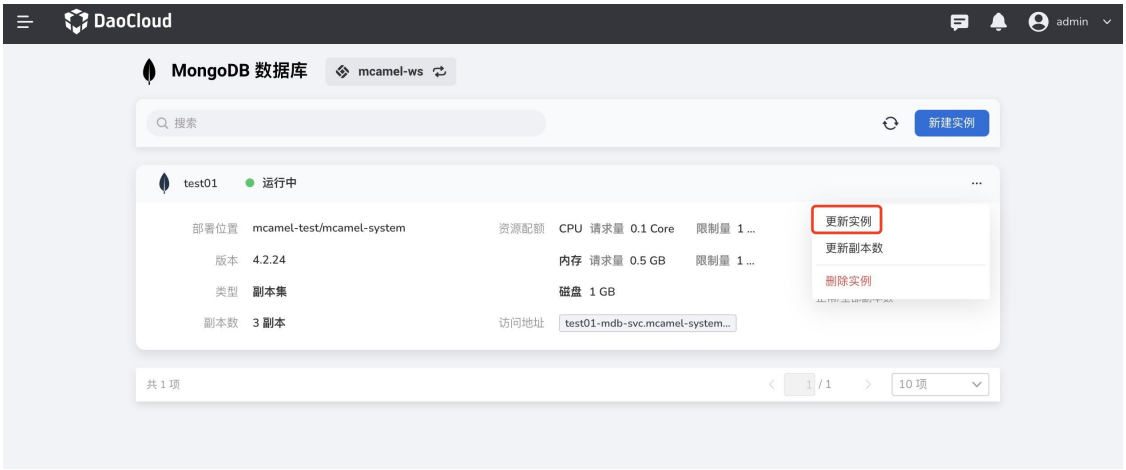
!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

更新 MongoDB 实例

如果想要更新或修改 MongoDB 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



更新实例

2. 基本信息：只能修改描述。然后点击下一步。



基本信息

3. 修改规格配置后点击下一步。



规格配置

4. 修改服务设置后点击确认。


更新实例 test01
基本信息
规格配置
3
服务设置
服务设置
访问类型 *
集群内访问(ClusterIP)
节点端口(NodePort)
负载均衡器(LoadBalancer)
端口配置
服务端口
27017
容器内部服务端口，无需变更
注释
+ 添加
高级设置
调度策略
关闭
监控采集时间间隔
30
s
如果为空，将采用系统默认值，30s
取消
上一步
确认

服务设置

首次进入中间件

首次使用 DCE 5.0 的 Elasticsearch 搜索服务、Kafka 消息队列、MinIO 对象存储、MySQL 数据、RabbitMQ 消息、PostgreSQL 数据库、Redis 数据库、MongoDB 数据库等中间件时，需要选择工作空间。

有关工作空间的详细介绍，可参考工作空间与层级。

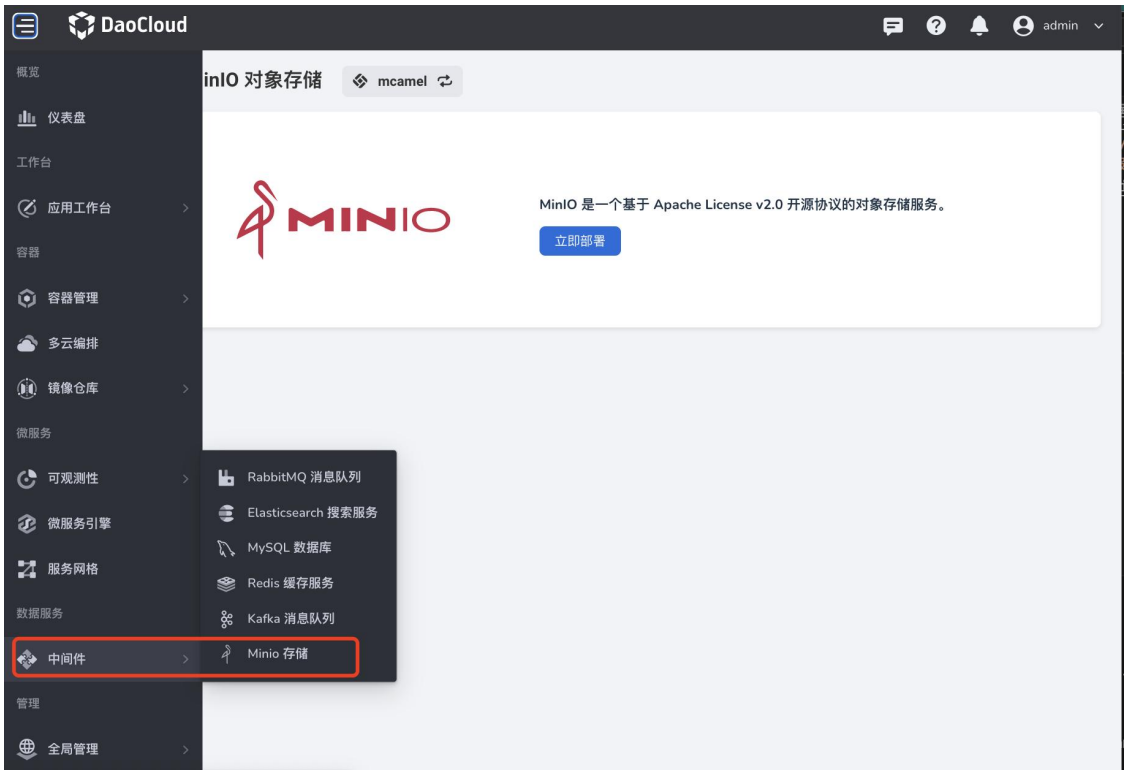
下面以 MinIO 为例介绍如何选择工作空间，其他数据服务组件同理。

前提条件

1. 在 DCE 5.0 平台中创建工作空间
2. 如需直接创建中间件实例，还需要在 DCE 5.0 容器管理模块中创建或接入集群。

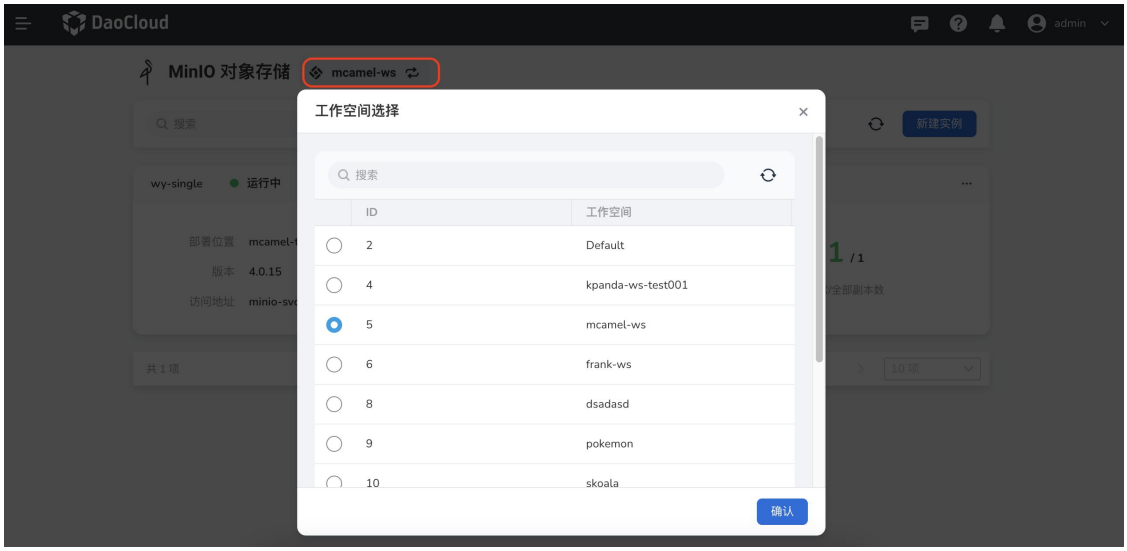
操作步骤

1. 在左侧导航栏中选择中间件 –> MinIO 存储。



选择工作空间

2. 在弹窗中选择一个工作空间后，点击确认。

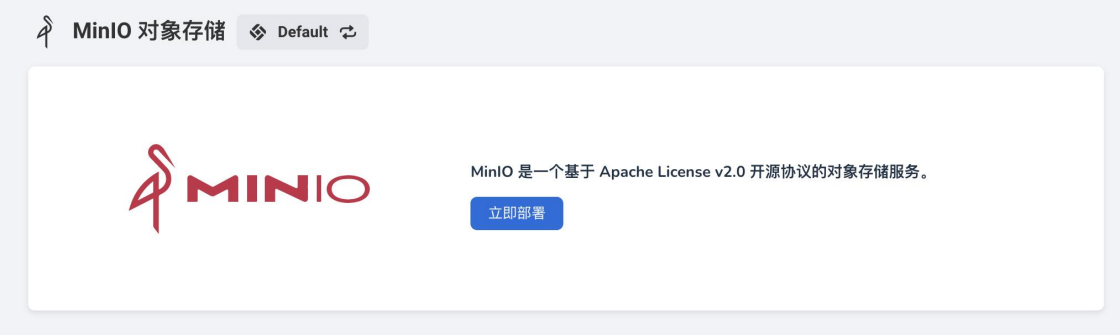


选择工作空间

!!! note

如果未出现弹窗/想要切换工作空间，可手动点击红框中的切换图标选择新的工作空间。

3. 初次使用时，可以点击立即部署来创建 MinIO 实例。



立即部署

工作负载反亲和性

创建 Elasticsearch 搜索服务、Kafka 消息队列、MinIO 对象存储、MySQL 数据、RabbitMQ 消息、PostgreSQL 数据库、Redis 数据库、MongoDB 数据库实例时，可以在服务设置页面配置工作负载反亲和性。

工作负载反亲和性的作用原理是，在一定的拓扑域（作用域）范围内，如果检测到已经有工作负载带有反亲和性配置中添加的某个标签，则不会将新建的工作负载部署到该拓扑域。这样做的好处是：

- 性能优化

  通过工作负载反亲和性，将实例的多个副本部署到不同的节点/可用区/区域，避免副本间的资源竞争，确保每个副本都有充足的可用资源，从而提供应用的性能和可靠性。

- 故障隔离

  实例副本分布在不同的节点/可用区/区域，有效避免了单点故障问题。某个环境中的副本出现故障时，其他环境中的副本不受影响，从而保障服务整体的可用性。

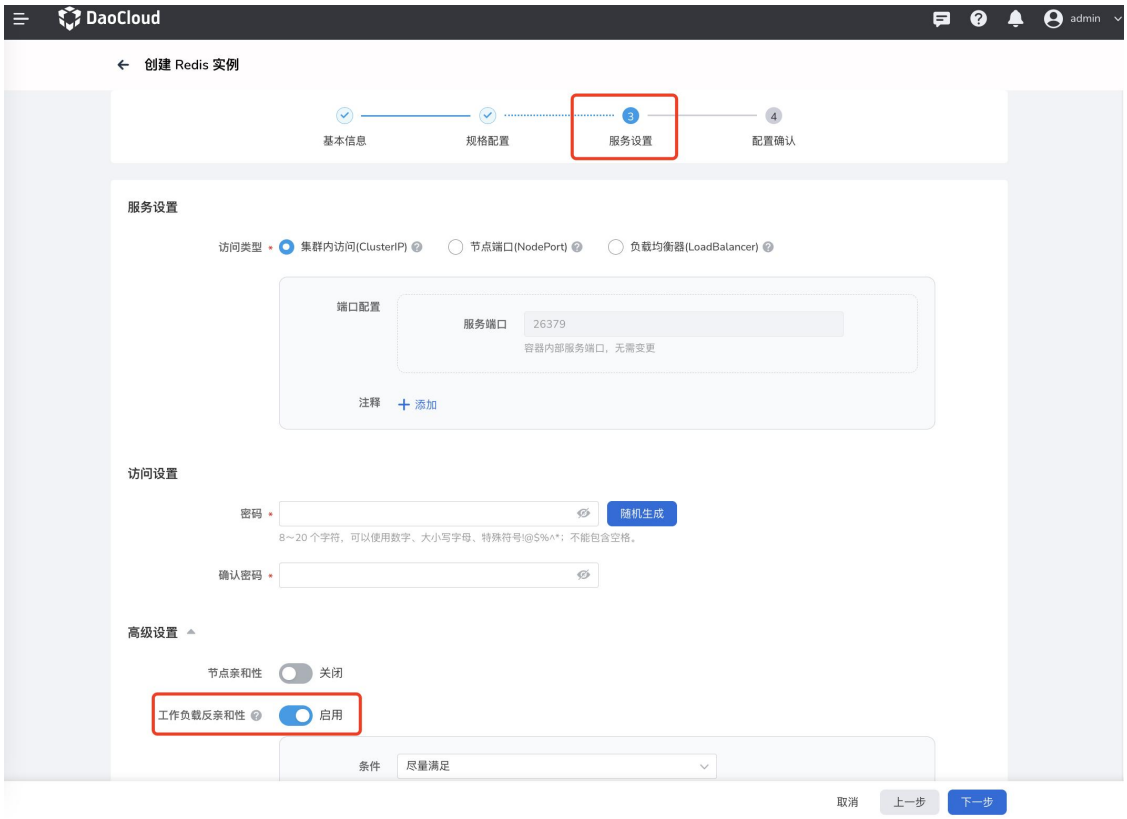
操作步骤

下面以 Redis 为例，介绍如何设置工作负载反亲和性。

!!! note

本文侧重介绍如何配置`工作负载反亲和性`。如需了解详细的 `Redis` 实例创建过程，可参考[创建实例](../redis/user-guide/create.md)。

1. 在创建 Redis 实例过程中，在服务设置->高级设置中启用工作负载反亲和性。



创建

2. 参考以下说明填写工作负载反亲和性的配置。

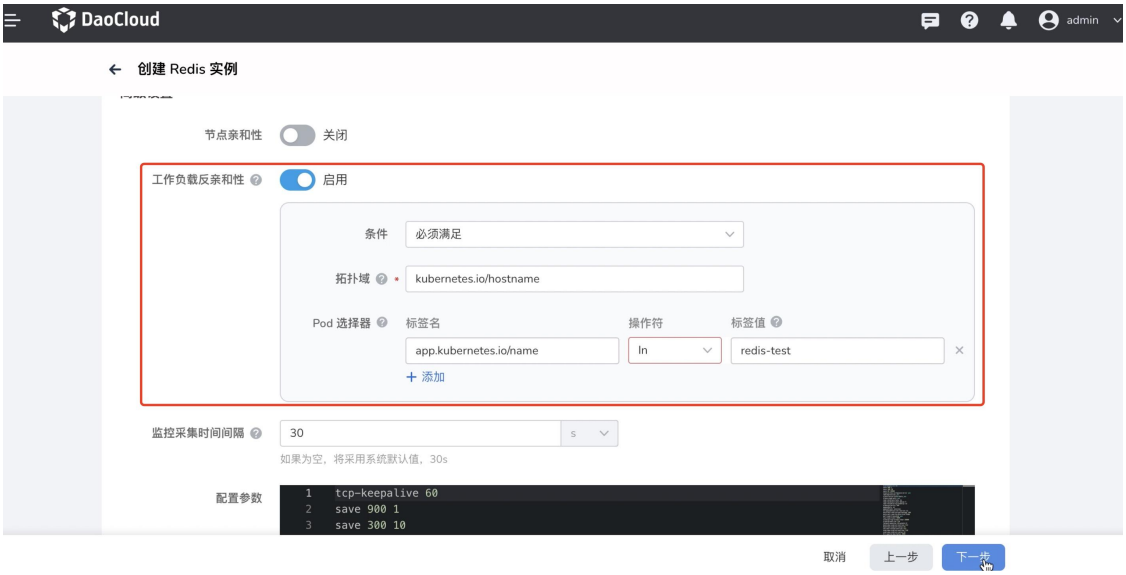
   - 条件：分为尽量满足和必须满足两种。

     - 尽量满足：尽量尝试满足反亲和性要求，最终的部署结果不一定满足反亲和性要求
     - 必须满足：必须满足反亲和性要求。如果找不到可调度的节点/可用区/区域，则 Pod 会一直处于 Pending 状态。

   - 权重：条件设置为必须满足时无需设置权重。条件设置为尽量满足时，为反亲和性规则设置权重值，优先满足权重高的规则。

   - 拓扑域：拓扑域定义反亲和性的作用范围，可以为节点标签、zone 标签、region 标签，也可以由用户自定义。

- Pod 选择器：设置 Pod 标签。**同一个拓扑域内，只能有一个带有此标签的 Pod。**
- 有关反亲和性以及各种操作符的详细介绍，可参考操作符



创建

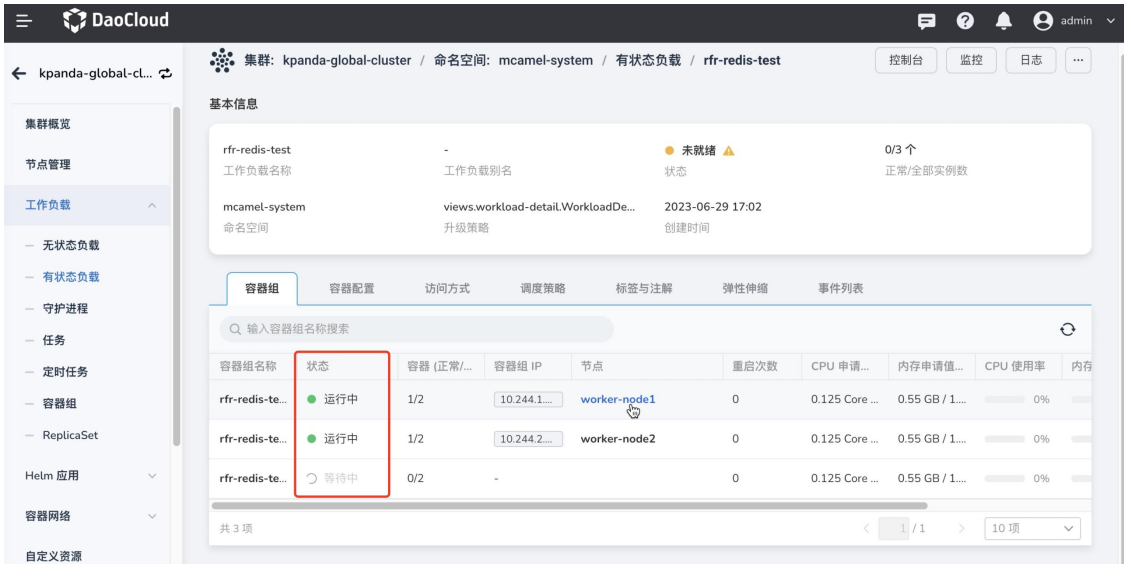
!!! note

上图配置的含义是，在同一个节点上只能有一个带有 `app.kubernetes.io/name` 标签且值为 `redis-test` 的 Pod。如果没有满足条件的节点，则 Pod 会一直处于 Pending 状态。

3. 参考创建实例完成后续操作。

效果验证

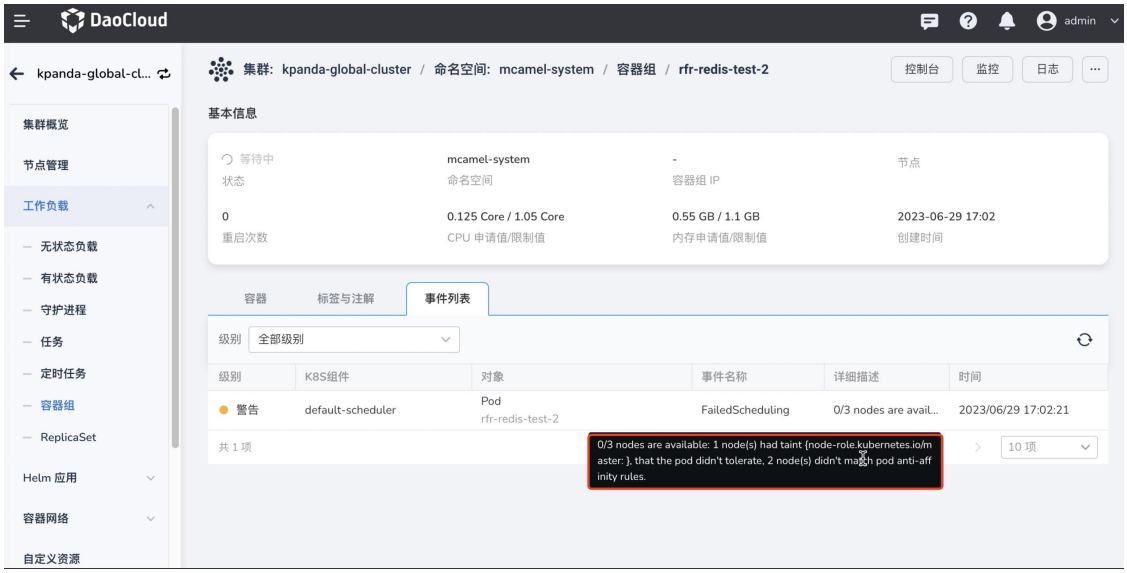
配置好工作负载反亲和性并且实例创建成功后，进入容器管理模块查看 Pod 调度信息。



查看 Pod

可见一共 3 个 Pod，两个已经在正常运行并且分布在不同的节点上。

第三个 Pod 处于等待状态，点击 Pod 名称查看详情，发现原因是污点和反亲和性规则导致没有可以部署的节点。



事件日志

!!! success

这说明前面配置的工作负载反亲和性已经生效，即一个节点上之后只能有一个带有 `app.kubernetes.io/name` 标签且值为 `redis-test` 的 Pod。如果没有满足条件的节点，则 Pod 一直处于 Pending 状态。

数据服务自定义角色权限说明

数据服务模块允许用户自定义角色权限，用户可以在 全局管理中创建自定义角色并为角色指定各中间件的权限。

创建自定义角色时可以为角色指定三种权限：创建/编辑、查看、删除。

!!! note

部分权限选项会自动关联相关依赖权限，例如：选中`删除`权限会自动关联勾选`查看`权限，请勿取消依赖权限，避免目标权限无法正常使用。

这三种权限在数据服务模块内有权执行的具体操作如下：

数据服务	操作对象	操作	创建/编辑	查看	删除
配置管理	配置列表	查看列表			
		配置名称搜索			
		创建配置			
		更新配置			
		删除配置			
MySQL	MySQL 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	MySQL 实例详情	实例概览			
		实例监控			

数据服务	操作对象	操作	创建/编辑	查看	删除
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
		查看实例备份列表			
		创建实例备份			
		修改实例自动备份任务			
		使用备份创建新实例			
	备份配置管理	备份配置列表			
		创建备份配置			
		修改备份配置			
		删除备份配置			
	配置参数	查看配置参数			
		修改配置参数			
RabbitMQ	RabbitMQ 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			

数据服务	操作对象	操作	创建/编辑	查看	删除
	RabbitMQ 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
Elasticsearch	Elasticsearch 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	Elasticsearch 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
Redis	Redis 实例列表	查看列表			
		实例名称搜索			
		创建实例			

数据服务	操作对象	操作	创建/编辑	查看	删除
		更新实例配置			
		删除实例			
	Redis 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
		创建实例备份			
		修改实例自动备份任务			
		使用备份创建新实例			
	备份配置管理	备份配置列表			
		创建备份配置			
		修改备份配置			
		删除备份配置			
	配置参数	查看配置参数			
		修改配置参数			
Kafka	Kafka 实例列表	查看列表			
		实例名称搜索			
		创建实例			

数据服务	操作对象	操作	创建/编辑	查看	删除
		更新实例配置			
		删除实例			
	Kafka 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
	配置参数	查看配置参数			
		修改配置参数			
MinIO	MinIO 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	MinIO 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问			

数据服务	操作对象	操作	创建/编辑	查看	删除
		密码			
	配置参数	查看配置参数			
		修改配置参数			
PostgreSQL	PostgreSQL 实例列表	查看列表			
		实例名称搜索			
		创建实例			
		更新实例配置			
		删除实例			
	PostgreSQL 实例详情	实例概览			
		实例监控			
		查看实例配置参数			
		修改实例配置参数			
		查看实例访问密码			
	配置参数	查看配置参数			
		修改配置参数			

Kafka 消息队列 Release Notes

本页列出 Kafka 消息队列的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.7.3

新功能

- **新增** UI 界面的权限访问限制。

2023-06-30

v0.7.0

新功能

- **新增** 对接全局管理审计日志模块
- **新增** LoadBalancer 服务类型

优化

- **优化** 监控图表，去除干扰元素并新增时间范围选择
- **优化** ServiceMonitor 闭环安装

2023-04-27

v0.5.1

新功能

- **新增** mcamel-kafka 详情页面展示相关的事件
- **新增** mcamel-kafka 支持自定义角色

优化

- **优化** mcamel-kafka 调度策略增加滑动按钮

2023-03-28

v0.4.0

新功能

- **新增** mcamel-kafka 支持中间件链路追踪适配。
- **新增** 安装 mcamel-kafka 根据参数配置启用链路追踪。

优化

- **优化** mcamel-kafka 优化 Kafka 的默认配置。
- **升级** golang.org/x/net 到 v0.7.0
- **升级** GHippo SDK 到 v0.14.0

2023-02-23

v0.3.0

新功能

- **新增** mcamel-kafka helm-docs 模板文件。
- **新增** mcamel-kafka 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-kafka 支持 cloud shell。
- **新增** mcamel-kafka 支持导航栏单独注册。
- **新增** mcamel-kafka 支持查看日志。
- **新增** mcamel-kafka Operator 对接 chart-syncer。

优化

- **优化** mcamel-kafka 升级离线镜像检测脚本。

修复

- **修复** mcamel-kafka 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-kafka 工作空间 Editor 用户无法查看实例密码。
- **新增** 日志查看操作说明，支持自定义查询、导出等功能。

2022-12-25

v0.2.0

新功能

- **新增** mcamel-kafka NodePort 端口冲突提前检测。
- **新增** mcamel-kafka 节点亲和性配置。

优化

- **优化** mcamel-kafka manager 去掉 probe，防止 kafka 没准备好不能打开 manager。

- **优化** mcamel-kafka zooEntrance 重新打包镜像地址为 1.0.0。

2022-11-28

v0.1.6

新功能

- **改进** 完善优化复制功能
- **改进** 实例详情 – 访问设置，移除集群 IPv4
- **改进** 中间件密码校验难度调整
- **新增** 对接告警能力
- **新增** 新增判断 sc 是否支持扩容并提前提示功能
- **优化** 优化安装环境检测的提示逻辑 & 调整其样式
- **优化** 中间件样式走查优化
- **修复** 离线镜像有数字和大写无法被扫描到

2022-11-08

v0.1.4

新功能

- **修复** 更新时无法校验到正确字段，如 managerPass
- **改进** 密码校验调整为 MCamel 低等密码强度
- **新增** 返回是否可以更新 sc 容量的校验

- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警
- **新增** 校验 Service 注释
- **修复** operator 用名字选择
- **修复** 服务地址展示错误
- **修复** Kafka 使用 NodePort 时，创建失败

2022-10-28

v0.1.2

新功能

- **新增** 同步 Pod 状态到实例详情页
- **优化** workspace 界面逻辑调整
- **优化** 不符合设计规范的样式调整
- **优化** password 获取逻辑调整
- **优化** cpu&内存请求量应该小于限制量逻辑调整

2022-9-25

v0.1.1

新功能

- **新增** 支持 kafka 列表查询，状态查询，创建，删除和修改
- **新增** 支持 kafka-manager 对 kafka 进行管理
- **新增** 支持 kafka 的指标监控，查看监控图表
- **新增** 支持 ghippo 权限联动
- **新增** mcamel-elasticsearch 获取用户列表接口
- **优化** 更新 release note 脚本，执行 release-process 规范

Kafka 常见术语

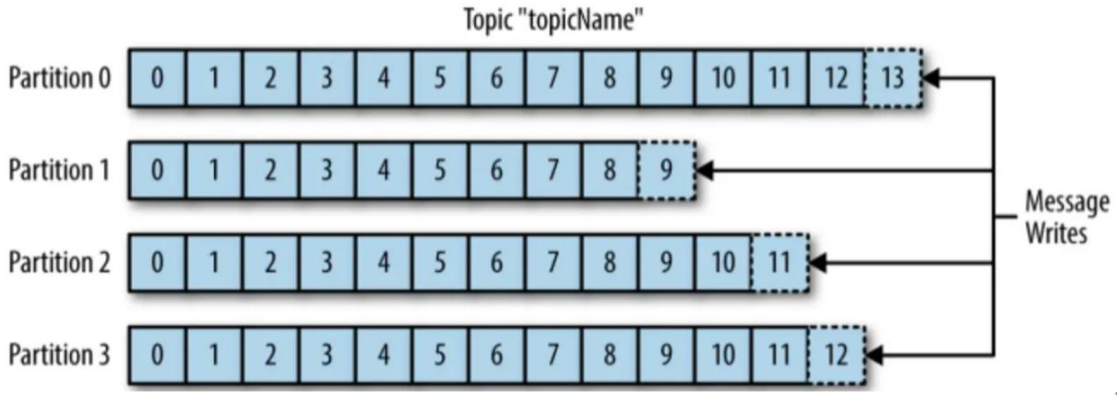
- 消息和批次

  Kafka 的基本数据单元称为 message（消息），为减少网络开销，提高效率，多个消息会被放入同一批次（Batch）中后再写入。

- 主题和分区

  Kafka 的消息通过 Topic（主题）进行分类，一个主题可以分为若干个 Partition（分区），一个分区就是一个提交日志（commit log）。 消息以追加的方式写入分区，然后以先入先出的顺序读取。 Kafka 通过分区来实现数据的冗余和伸缩性，分区可以分布在不同的服务器上，这意味着一个 Topic 可以横跨多个服务器，以提供比单个服务器更强大的性能。

  由于一个 Topic 包含多个分区，因此无法在整个 Topic 范围内保证消息的顺序性，但可以保证消息在单个分区内的顺序性。

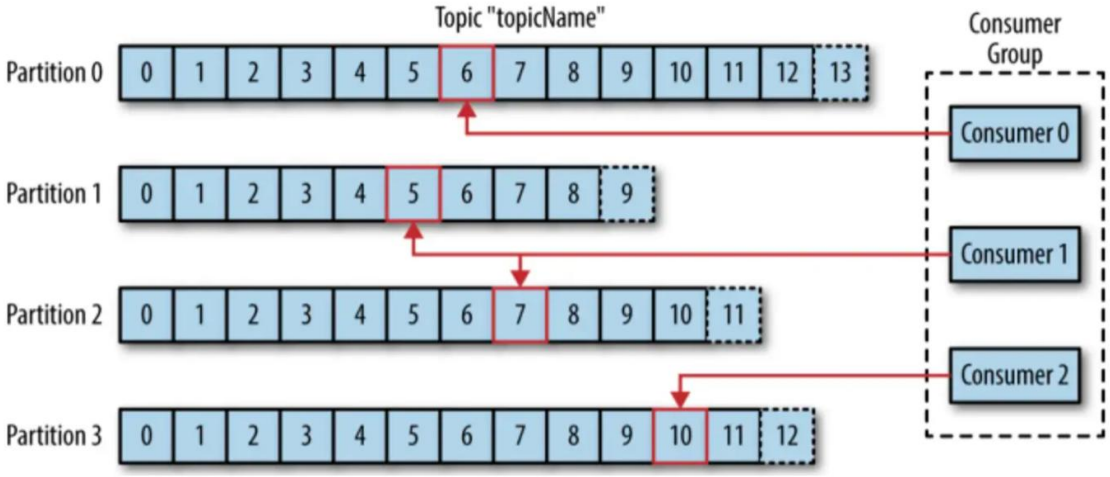


主题和分区

- 生产者和消费者

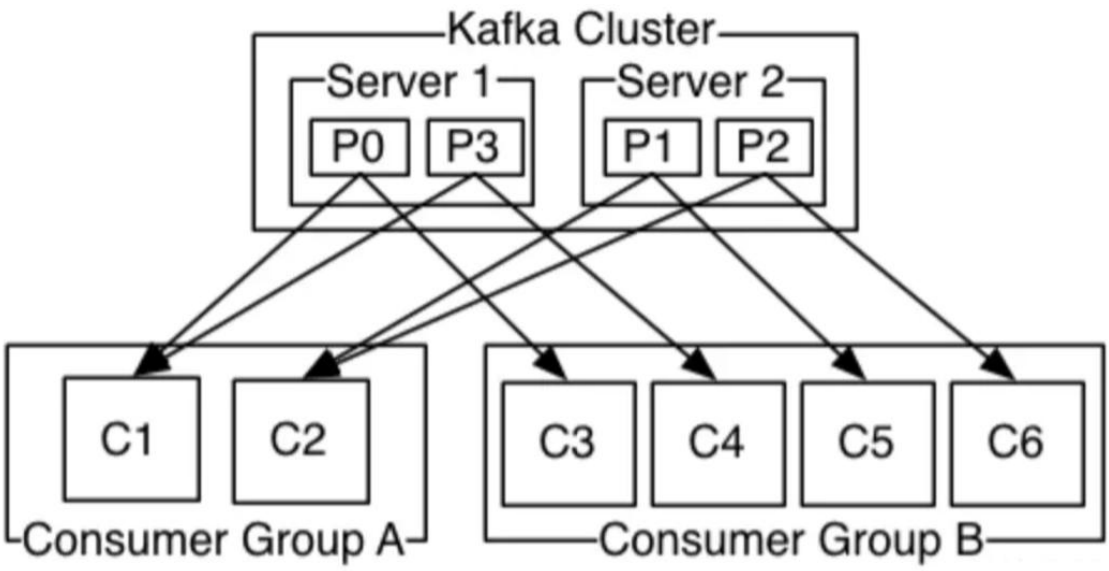
生产者负责创建消息。一般情况下，生产者在把消息均衡地分布到在主题的所有分区上，而并不关心消息会被写到哪个分区。 如果我们想要把消息写到指定的分区，可以通过自定义分区器来实现。

消费者是消费者群组的一部分，消费者负责消费消息。消费者可以订阅一个或者多个主题，并按照消息生成的顺序来读取它们。 消费者通过检查消息的偏移量（offset）来区分读取过的消息。 偏移量是一个不断递增的数值，在创建消息时，Kafka 会把它添加到其中，在给定的分区里，每个消息的偏移量都是唯一的。 消费者把每个分区最后读取的偏移量保存在 Zookeeper 或 Kafka 上，如果消费者关闭或者重启，它还可以重新获取该偏移量，以保证读取状态不会丢失。



消费者

一个分区只能被同一个群组中的一个消费者读取，但可以被不同群组中的多个消费者共同读取。 多个群组中的消费者共同读取同一个主题时，彼此之间互不影响。



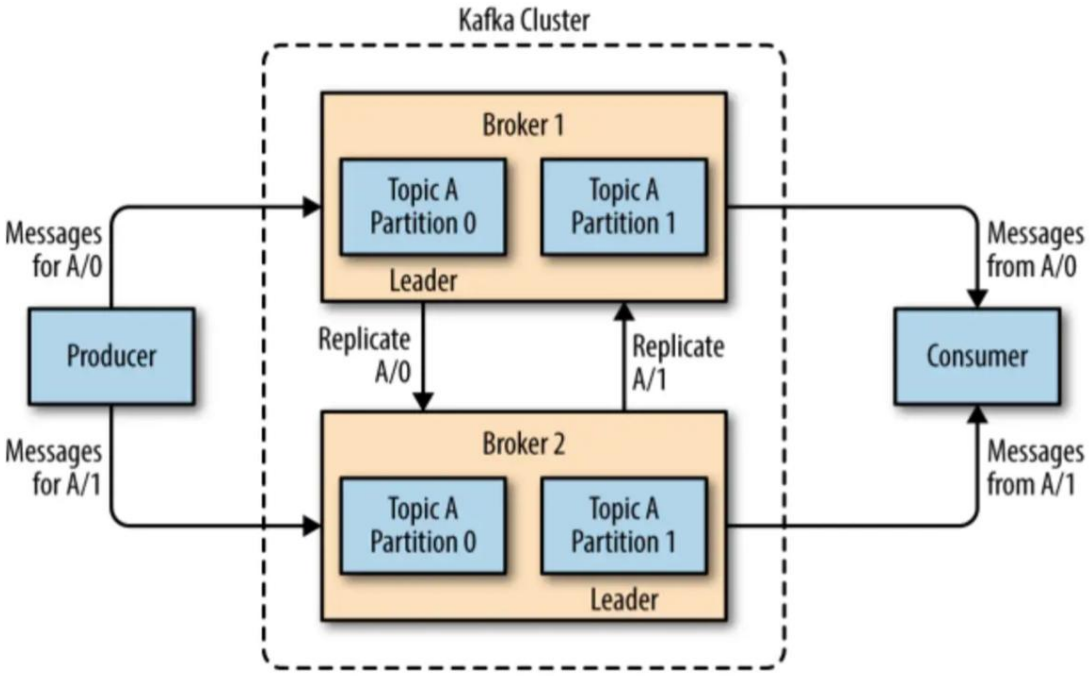
消费者

- Broker 和 Cluster

  一个独立的 Kafka 服务器被称为 Broker。Broker 接收来自生产者的消息，为消息设置偏移量，并提交消息到磁盘保存。 Broker 为消费者提供服务，对读取分区的请求做出响应，返回已经提交到磁盘的消息。

  Broker 是集群（Cluster）的组成部分。 每个集群都会选举出一个 Broker 作为集群控制器（Controller），集群控制器负责管理工作，包括将分区分配给 Broker 和监控 Broker。

  在集群中，一个分区（Partition）从属一个 Broker，该 Broker 被称为分区的首领（Leader）。 一个分区可以分配给多个 Broker，这个时候会发生分区复制。 这种复制机制为分区提供了消息冗余，如果有一个 Broker 失效，其他 Broker 可以接管领导权。



broker 和 cluster

什么是 Kafka

Kafka 模块是一款基于开源软件 Kafka 提供的分布式消息队列服务。 DaoCloud 为其开发了简单易用的图形化界面，向用户提供计算、存储和带宽资源独占的 Kafka 专享实例。

Kafka 是一个拥有高吞吐、可持久化、可水平扩展，支持流式数据处理等多种特性的分布式消息流处理中间件，采用分布式消息发布与订阅机制，在日志收集、流式数据传输、在线/离线系统分析、实时监控等领域有广泛的应用。

目前支持的特性如下：

- 创建/更新/删除 Kafka 实例
- 可靠性保证机制
- 支持自定义参数配置
- 支持集群高可用
- 多语言客户端支持
- 简单易用的图形界面



kafka 主界面

创建 Kafka 实例

使用场景

相比支持广播、事务消息、消息路由、死信队列、优先级队列等且广泛应用于秒杀、流控、系统解耦等场景的 RabbitMQ，Kafka 消息队列适用于构建实时数据管道、流式数据处理、第三方解耦、流量削峰去谷等场景，具有大规模、高可靠、高并发访问、可扩展且完全托管的特点。

与 RabbitMQ 对比

正所谓没有最好的技术，只有最合适的技术。每个消息中间件服务都有自己的优劣，以下对 RabbitMQ 和 Kafka 做一个简单的对比。

	Kafka	RabbitMQ
性能	单节点 QPS 达百万级，吞吐量高	单节点 QPS 为万级，吞吐量低
可靠性	多副本机制，数据可靠性高	多副本机制，数据可靠性高
功能	持久化事务消息单分区级别的顺序性	持久化优先级队列延迟队列死信队列事务消息
消费模式	消息过滤客户端主动拉取消息回溯广播消费	客户端主动拉取和服务端推送广播消费
客户端支持	只支持 Kafka 自定义协议采用 Scala 和 Java 编写支持 SSL/SASL 认证和读写权限控制	支持 MQTT、STOMP 等多种协议采用 Erlang 编写支持 SSL/SASL 认证和读写权限控制

	Kafka	RabbitMQ
服务可用性	采用集群部署，分区与多副本的设计，使用单代理宕机对服务无影响，且支持消息容量的线性提升	支持集群部署，集群代理数量有多种规格
其他	消息堆积流量控制：支持 client 和 user 级别，通过主动设置可将流控作用于生产者或消费者	消息追踪消息堆积多租户流量控制：流控基于 Credit-based 算法，是内部被动触发的保护机制，作用于生产者层面

总之，Kafka 采用拉取（Pull）方式消费消息，吞吐量相对更高，适用于海量数据收集与传递场景，例如日志采集和集中分析。

而 RabbitMQ 基于 Erlang 语言开发，不利于做二次开发和维护，适用于对路由、负载均衡、数据一致性、稳定性和可靠性要求很高，对性能和吞吐量要求没那么高的场景。

典型场景

Kafka 作为一款热门的消息队列中间件，具备高效可靠的消息异步传递机制，主要用于不同系统间的数据交流和传递，在企业解决方案、金融支付、电信、电子商务、社交、即时通信、视频、物联网、车联网等众多领域都有广泛应用。

1. 异步通信

   将业务中属于非核心或不重要的流程部分使用消息异步通知的方式发给目标系统，这样主业务流程无需同步等待其他系统的处理结果，从而达到系统快速响应的目的。

   如网站的用户注册场景，在用户注册成功后，还需要发送注册邮件与注册短信，这两个流程使用 Kafka 消息服务通知邮件发送系统与短信发送系统，从而提升注册流程的响应速度。

2. 错峰流控与流量削峰

   在电子商务或大型网站中，上下游系统处理能力存在差异，处理能力高的上游系统的突发流量可能会对处理能力低的某些下游系统造成冲击，需要提高系统的可用性的同时降低系统实现的复杂性。 电商大促销等流量洪流突然来袭时，可以通过队列服务堆积缓存订单等信息，在下游系统有能力处理消息的时候再处理，避免下游订阅系统因突发流量崩溃。 消息队列提供亿级消息堆积能力，3 天的默认保留时长，消息消费系统可以错峰进行消息处理。

   另外，在商品秒杀、抢购等流量短时间内暴增场景中，为了防止后端应用被压垮，可在前后端系统间使用 Kafka 消息队列传递请求。

3. 日志同步

   在大型业务系统设计中，为了快速定位问题，全链路追踪日志，以及故障及时预警监控，通常需要将各系统应用的日志集中分析处理。

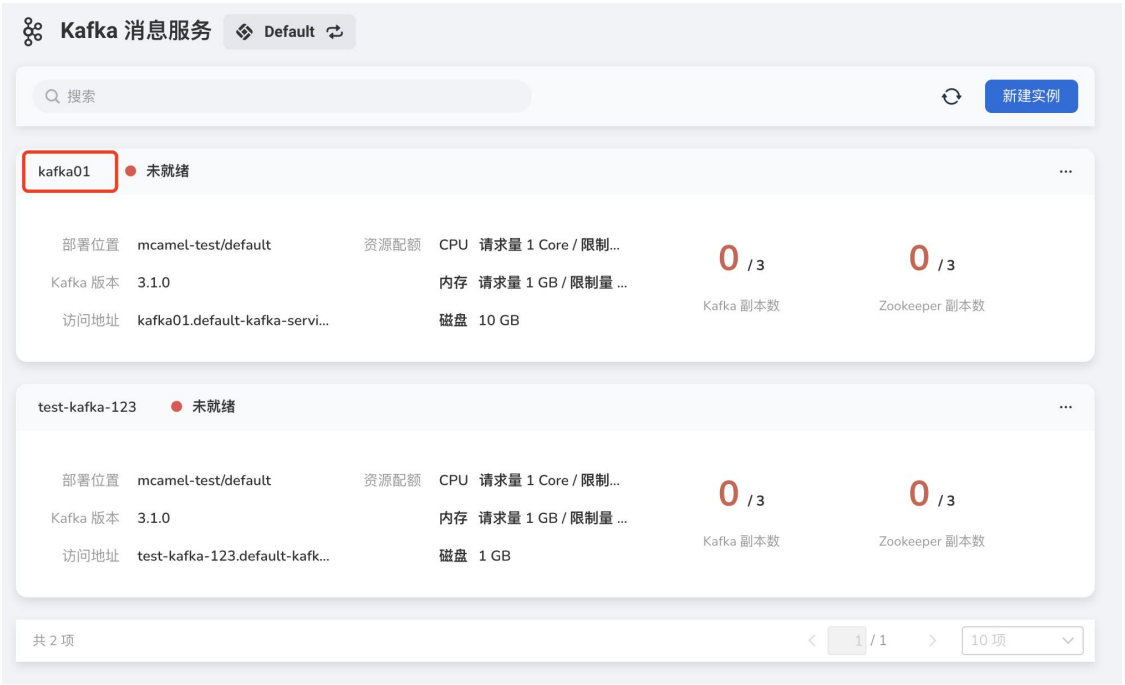
   Kafka 设计初衷就是为了应对大量日志传输场景，应用通过可靠异步方式将日志消息同步到消息服务，再通过其他组件对日志做实时或离线分析，也可用于关键日志信息收集进行应用监控。

   日志同步主要有三个关键部分：日志采集客户端，Kafka 消息队列以及后端的日志处理应用。

实例监控

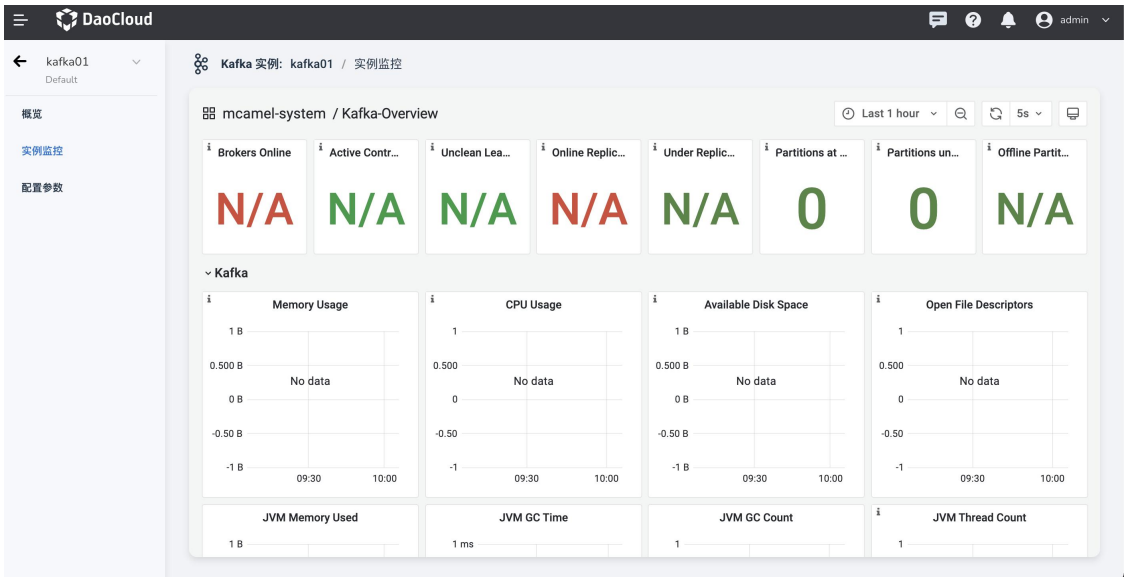
Kafka 内置了 Prometheus 和 Grafana 监控模块。

1. 在消息队列页面中，点击某个名称。



点击某个名称

2. 在左侧导航栏，点击实例监控，可以接入监控模块。


DaoCloud
admin
kafka01
Default
概览
实例监控
配置参数
Kafka 实例: kafka01 / 实例监控
mcamel-system / Kafka-Overview
Last 1 hour
5s
Brokers Online
N/A
Active Contr...
N/A
Unclean Lea...
N/A
Online Replic...
N/A
Under Replic...
N/A
Partitions at ...
0
Partitions un...
0
Offline Partit...
N/A
Kafka
Memory Usage
1 B
0.500 B
0 B
-0.50 B
-1 B
No data
09:30
10:00
CPU Usage
1
0.500
0
-0.50
-1
No data
09:30
10:00
Available Disk Space
1 B
0.500 B
0 B
-0.50 B
-1 B
No data
09:30
10:00
Open File Descriptors
1
0.500
0
-0.50
-1
No data
09:30
10:00
JVM Memory Used
1 B
JVM GC Time
1 ms
JVM GC Count
1
JVM Thread Count
1

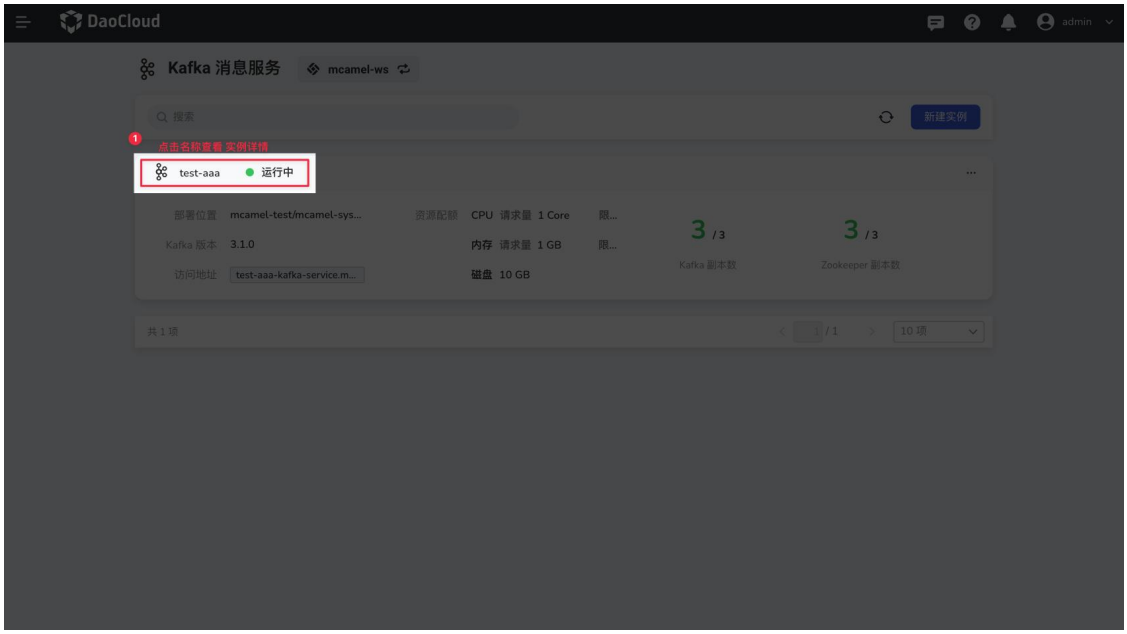
点击实例监控

查看 Kakfa 日志

操作步骤

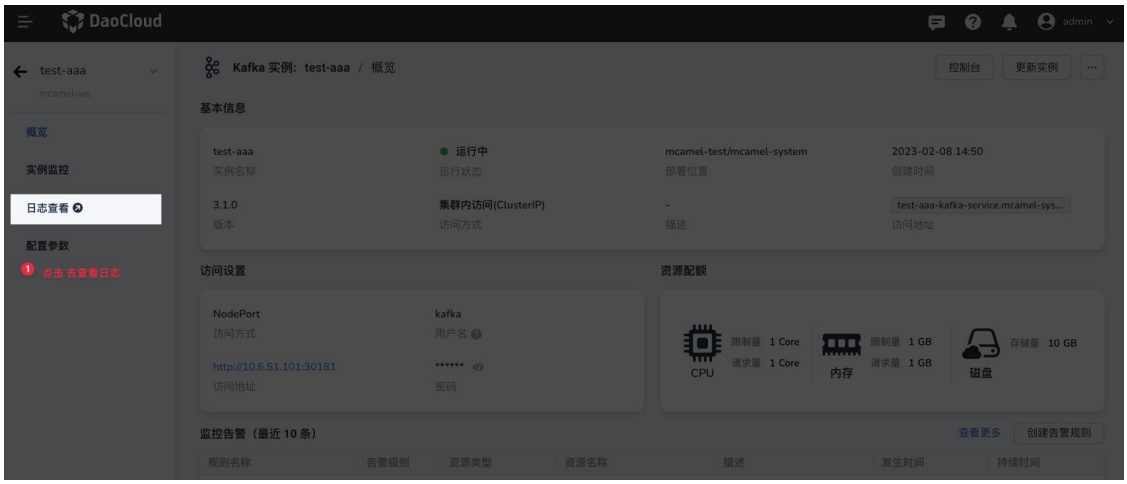
通过访问每个 Kakfa 的实例详情，页面；可以支持查看 Kakfa 的日志。

1. 在 Kakfa 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



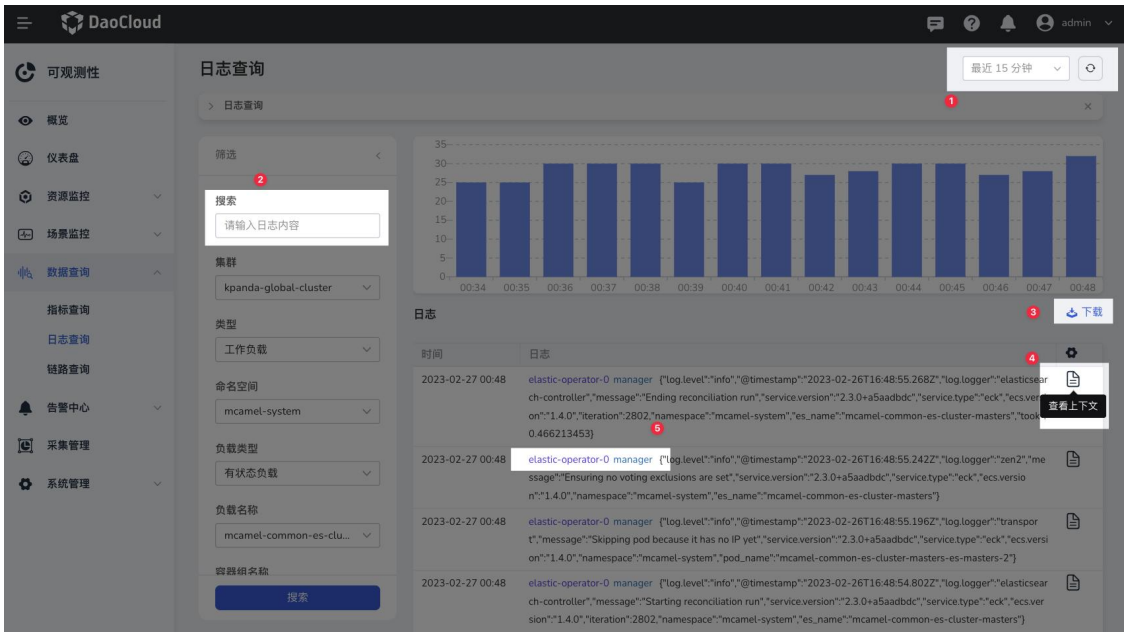
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

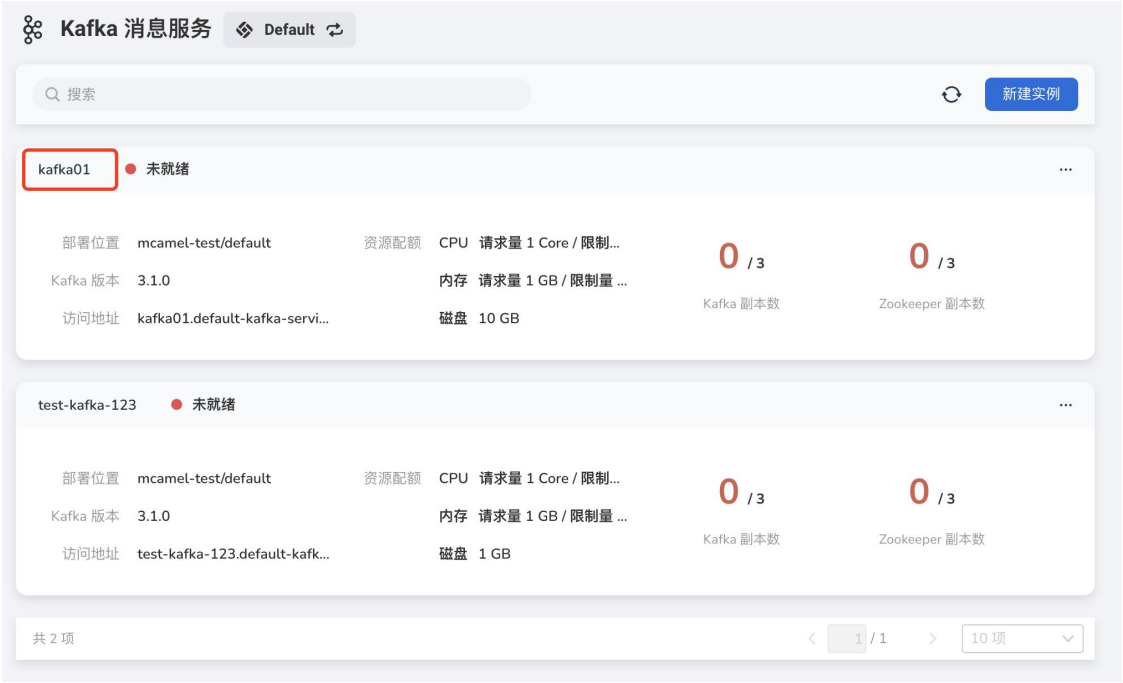

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

查看消息队列

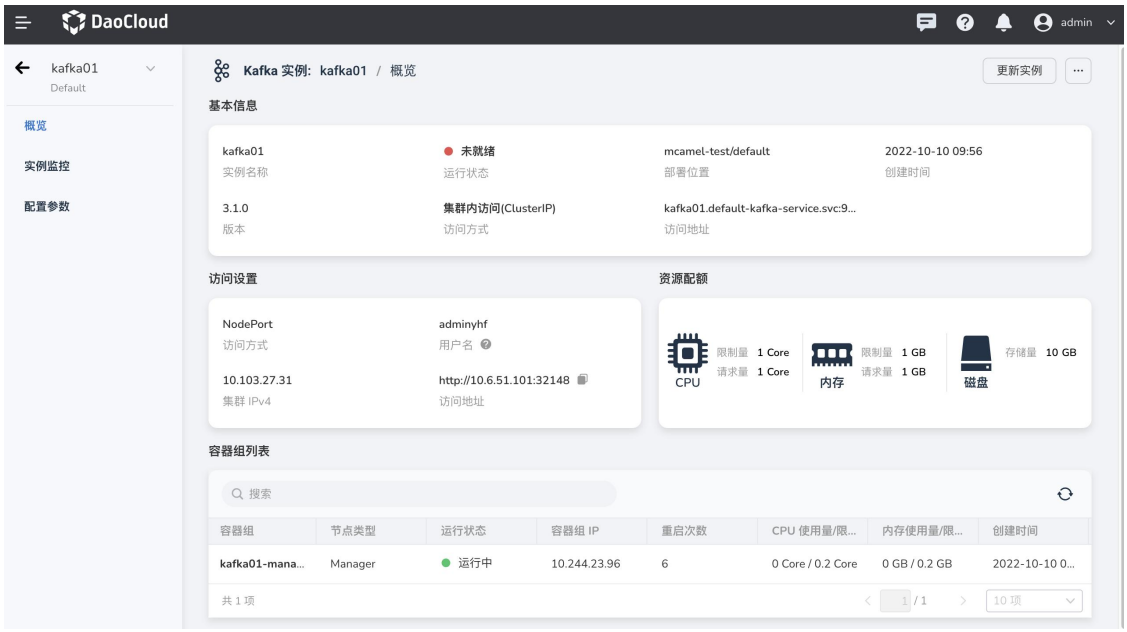
本节说明如何查看 Kafka 消息队列。

1. 在消息队列页面中，点击某个名称。



点击某个名称

2. 进入消息队列概览，查看基本信息、访问设置、资源配额和 Pod 列表等信息。

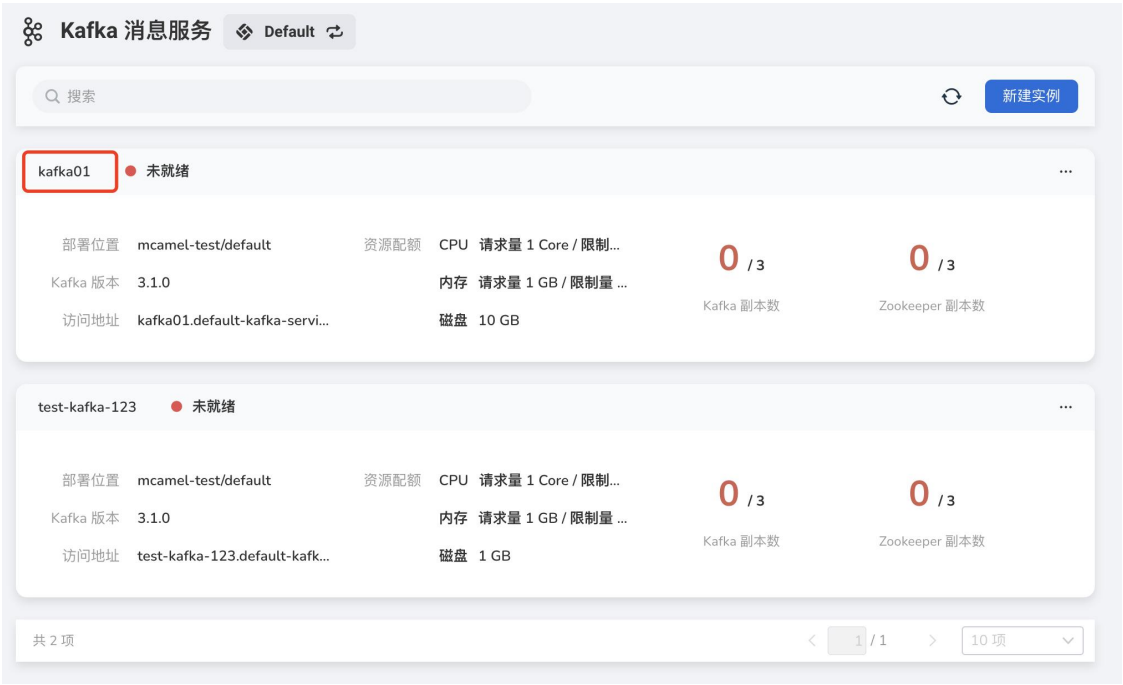

DaoCloud
admin
kafka01
Default
概览
实例监控
配置参数
Kafka 实例: kafka01 / 概览
更新实例
基本信息
kafka01
实例名称
未就绪
运行状态
mcamel-test/default
部署位置
2022-10-10 09:56
创建时间
3.1.0
版本
集群内访问(ClusterIP)
访问方式
kafka01.default-kafka-service.svc:9...
访问地址
访问设置
NodePort
访问方式
adminyhf
用户名
10.103.27.31
集群 IPv4
http://10.6.51.101:32148
访问地址
资源配额
CPU
限制量 1 Core
请求量 1 Core
内存
限制量 1 GB
请求量 1 GB
磁盘
存储量 10 GB
容器组列表
搜索
容器组
节点类型
运行状态
容器组 IP
重启次数
CPU 使用量/限...
内存使用量/限...
创建时间
kafka01-mana...
Manager
运行中
10.244.23.96
6
0 Core / 0.2 Core
0 GB / 0.2 GB
2022-10-10 0...
共 1 项
1 / 1
10 项

查看信息

配置参数

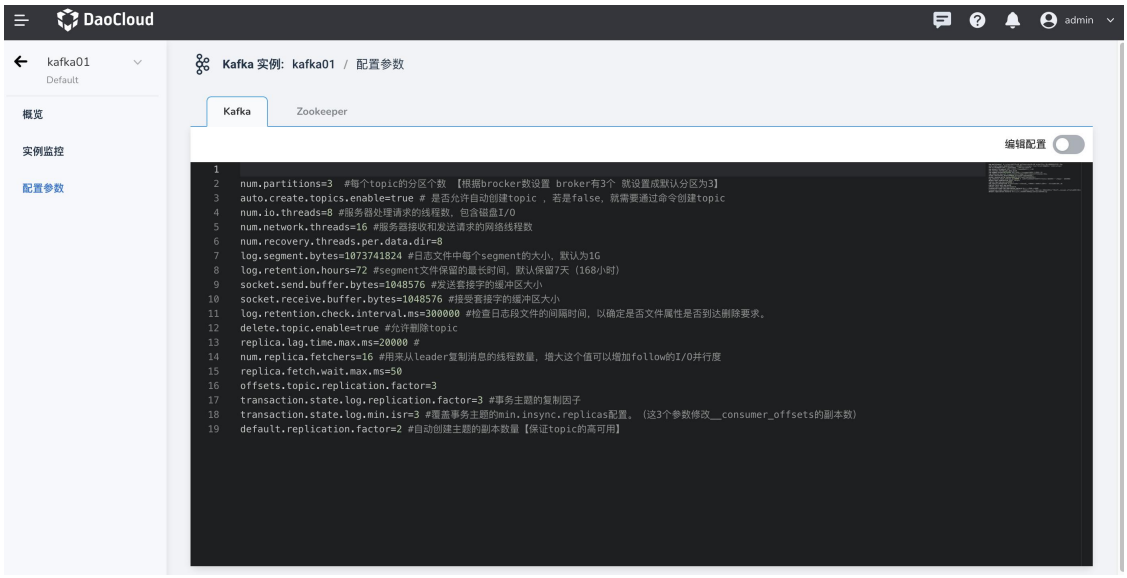
Kafka 内置了参数配置 UI 界面。

1. 在消息队列页面中，点击某个名称。



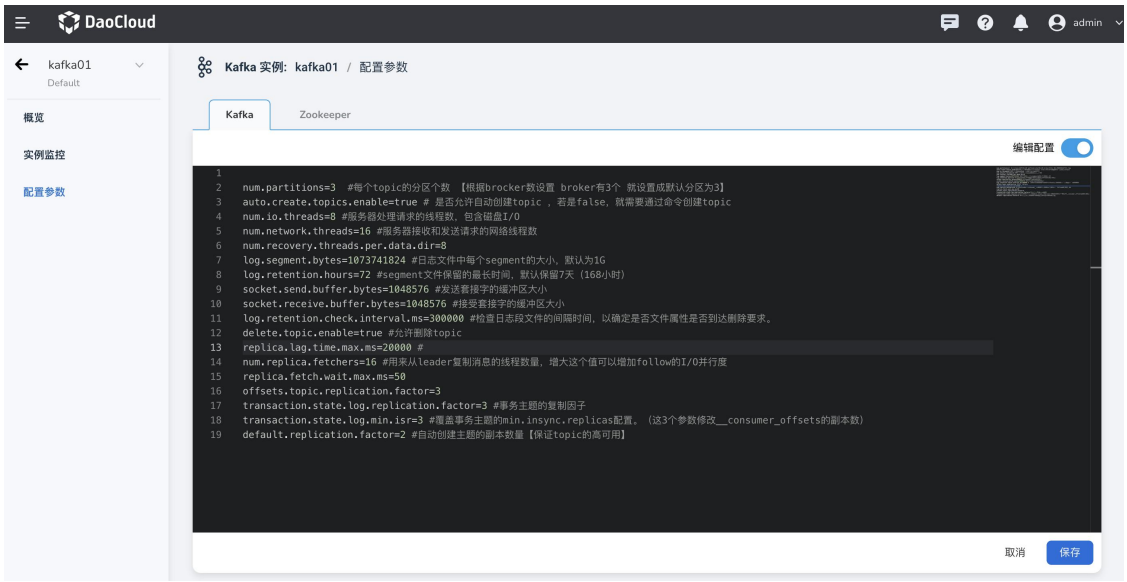
点击某个名称

2. 在左侧导航栏，点击配置参数。



点击配置参数

3. 点击编辑配置滑块开关，可以很方便地配置 Kafka 和 Zookeeper 的各项参数。



配置参数

4. 点击保存，参数将立即生效。

创建 Kafka

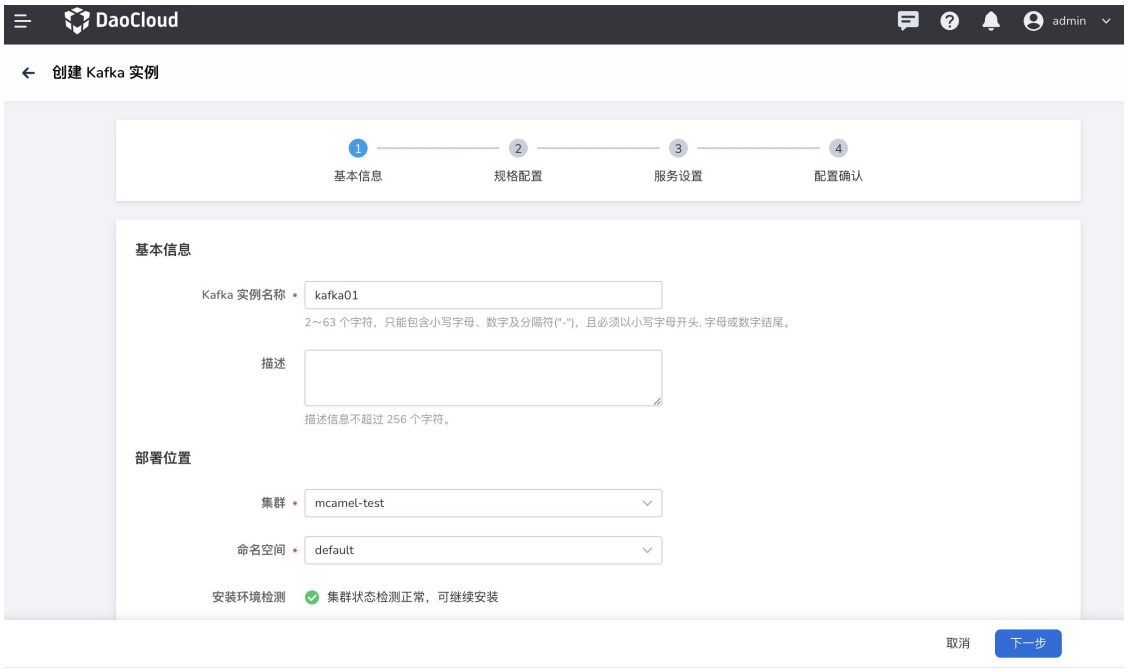
在 Kafka 消息队列中，执行以下操作创建 Kafka 实例。

1. 在右上角点击新建实例。



新建实例

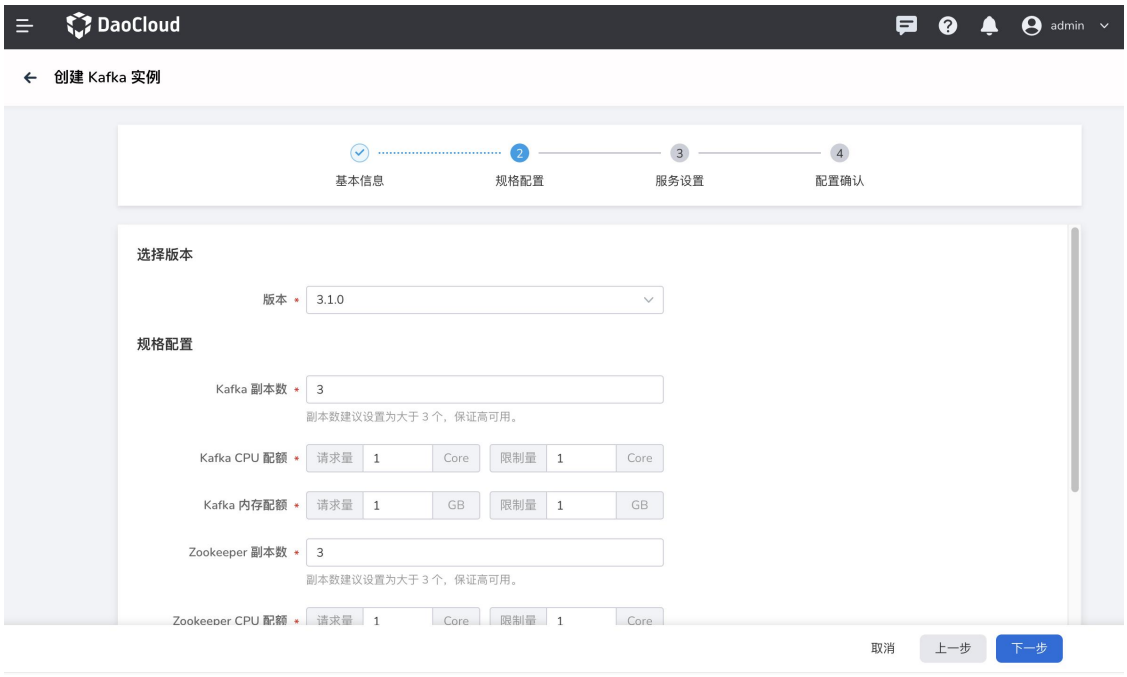
2. 在创建 Kafka 实例页面中，设置基本信息后，点击下一步。



设置基本信息

3. 配置规格后，点击下一步。

   - 版本：Kafka 的版本号，当前仅支持 Kafka 3.1.0。
   - 副本数：支持 1、3、5、7 副本数。
   - 资源配额：根据实际情况选择规则。
   - 存储卷：选择 Kafka 实例的存储卷和储存空间总量。



配置规格

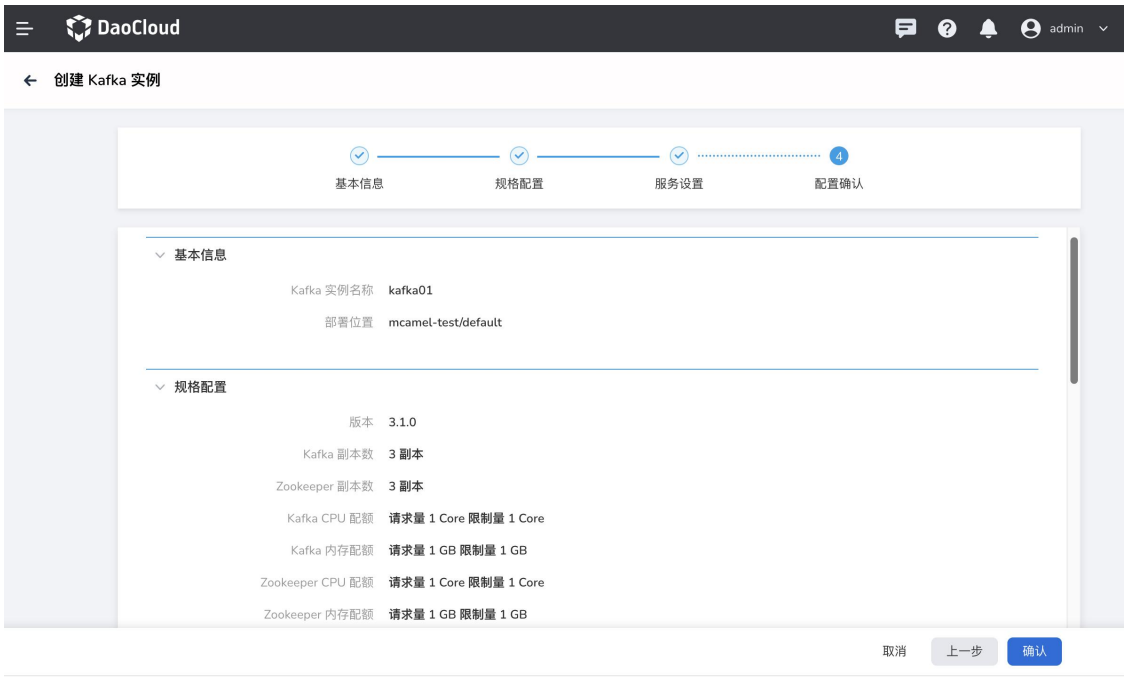
4. 服务设置后，点击下一步。

   - 访问方式：可以选择集群内访问还是 Nodeport 访问。
   - 服务设置：设置连接 Kafka 实例的用户名、密码。



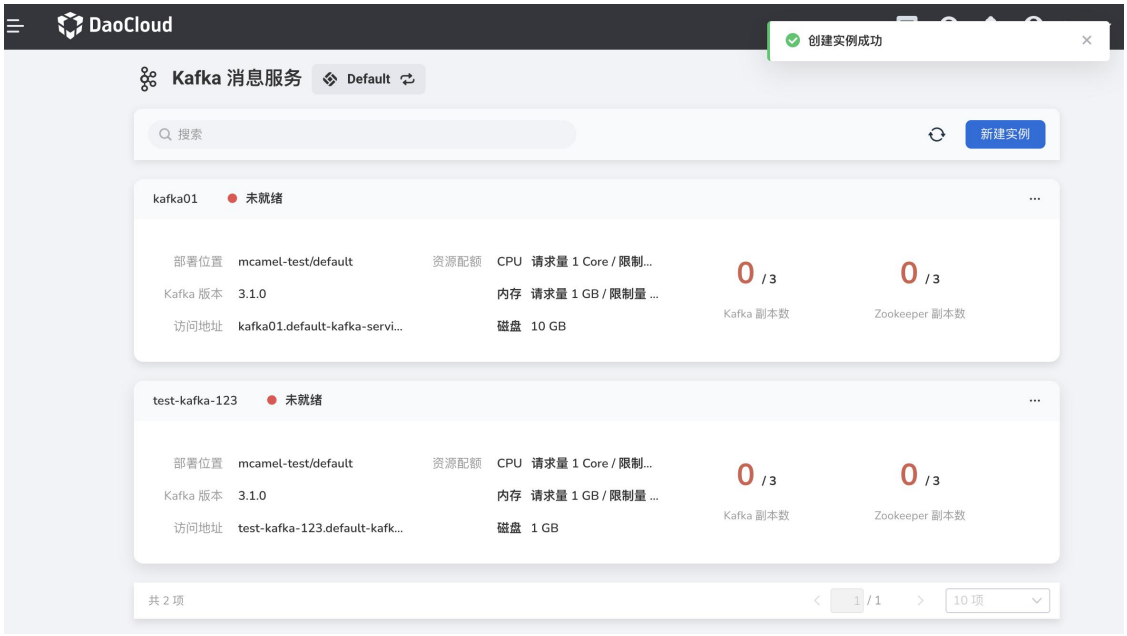
服务设置

5. 确认实例配置信息无误，点击确认完成创建。



点击确认

6. 在实例列表页查看实例是否创建成功。刚创建的实例状态为未就绪，等几分钟后该状态变为运行中。

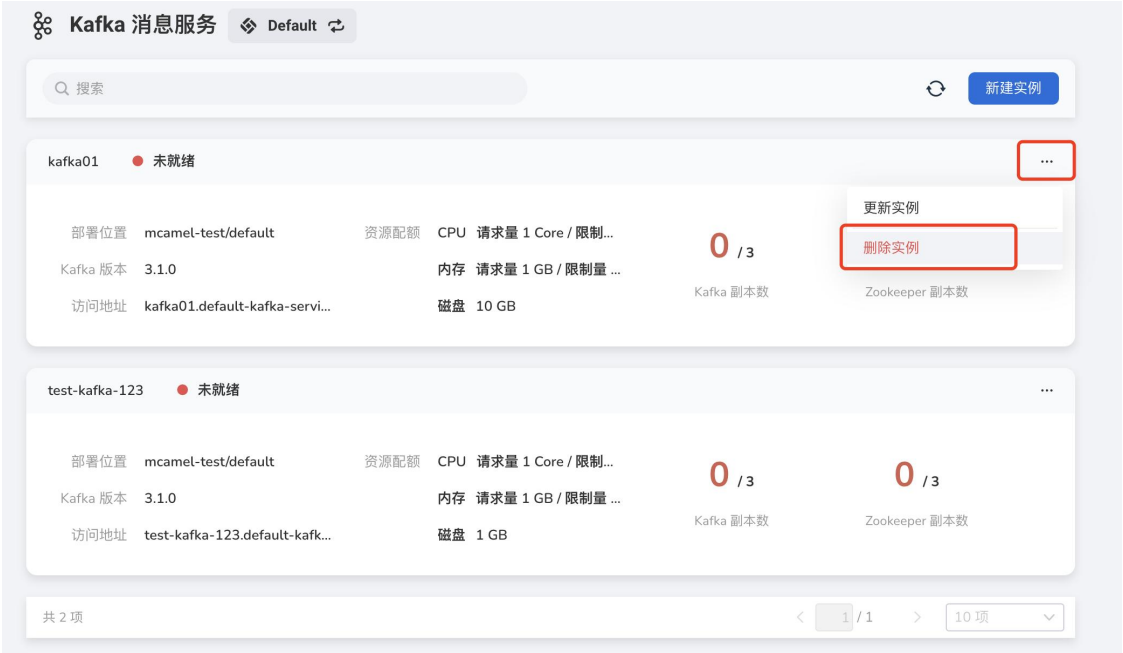


查看状态

删除 Kafka

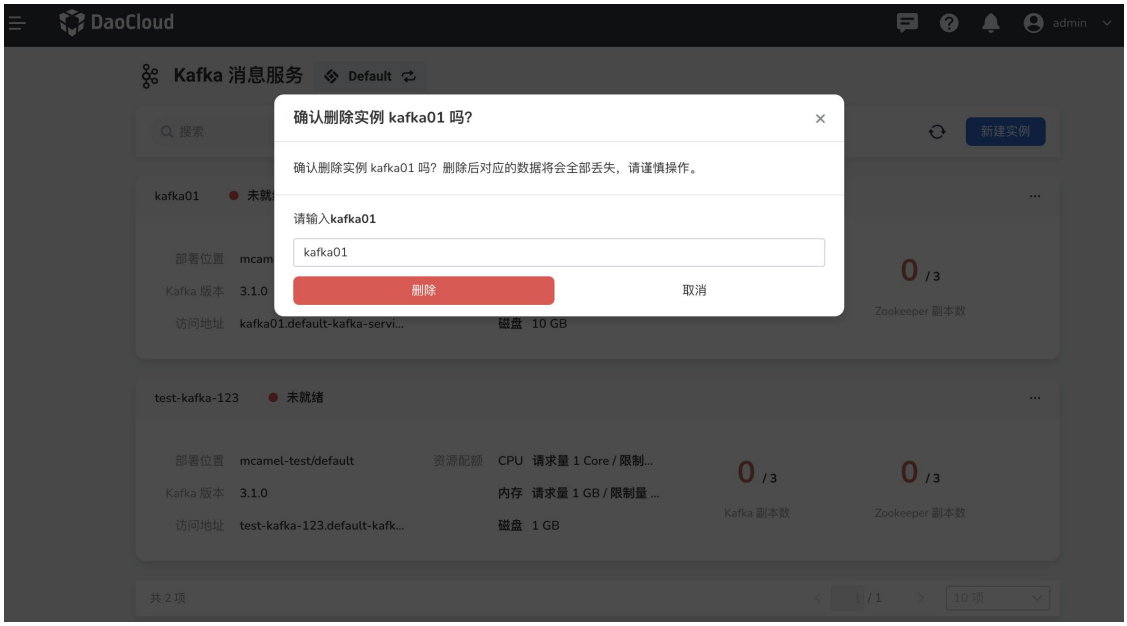
如果想要删除一个消息队列，可以执行如下操作：

1. 在消息队列中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



选择删除实例

2. 在弹窗中输入该消息队列的名称，确认无误后，点击删除按钮。



点击删除

!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。

MySQL Release Notes

本页列出 MySQL 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.10.3

新功能

- **新增** UI 界面的权限访问限制。

2023-06-30

v0.10.0

新功能

- **优化** mcamel-mysql 备份管理页面结构样式展示。
- **优化** mcamel-mysql 监控图表，去除干扰元素并新增时间范围选择。
- **优化** mcamel-mysql 存储容量指标来源，使用中立指标。
- **优化** mcamel-mysql ServiceMonitor 闭环安装。

2023-05-30

v0.9.0

新功能

- **新增** mcamel-mysql 对接全局管理审计日志模块。
- **新增** mcamel-mysql 可配置实例监控数据采集间隔时间。
- **修复** mcamel-mysql 安装 MySQL Operator 多副本时，Raft 集群无法正常建立。
- **修复** mcamel-mysql 升级 MySQL Operator 多副本时 PodDisruptionBudget 版本到 v1。

2023-04-27

v0.8.2

新功能

- **新增** mcamel-mysql 详情页面展示相关的事件
- **新增** mcamel-mysql openapi 列表接口支持 Cluster 与 Namespace 字段过滤
- **新增** mcamel-mysql 自定义角色
- **新增** mcamel-mysql 对接 HwameiStor，支持存储容量展示（需要手动创建 HwameiStor exporter ServiceMonitor）

升级

- **升级** 优化 调度策略增加滑动按钮

2023-03-28

v0.7.0

新功能

- **新增** mcamel-mysql 支持中间件链路追踪适配。
- **新增** 安装 mcamel-mysql 根据参数配置启用链路追踪。
- **新增** mcamel-mysql PhpMyAdmin 支持 LoadBalancer 类型。

升级

- **升级** golang.org/x/net 到 v0.7.0
- **升级** GHippo SDK 到 v0.14.0
- **优化** mcamel-mysql common-mysql 支持多个实例优化。
- **优化** mcamel-mysql 排障手册增加更多处理方法。

2023-02-23

v0.6.0

新功能

- **新增** mcamel-mysql helm-docs 模板文件。

- **新增** mcamel-mysql 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-mysql 支持 cloud shell。
- **新增** mcamel-mysql 支持导航栏单独注册。
- **新增** mcamel-mysql 支持查看日志。
- **新增** mcamel-mysql 更新 Operator 版本。
- **新增** mcamel-mysql 展示 common MySQL 在实例列表中。
- **新增** mcamel-mysql 支持 MySQL8.0.29。
- **新增** mcamel-mysql 支持 LB。
- **新增** mcamel-mysql 支持 Operator 对接 chart-syncer。
- **新增** mcamel-mysql Operator 的 finalizers 权限以支持 openshift。
- **新增** UI 增加 MySQL 主从复制延迟情况展示
- **新增** 文档 增加 日志查看操作说明，支持自定义查询、导出等功能。

优化

- **升级** mcamel-mysql 升级离线镜像检测脚本。

修复

- **修复** mcamel-mysql 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-mysql 工作空间 Editor 用户无法查看实例密码。
- **修复** mcamel-mysql 配置文件里 expire-logs-days 参数定义重复。
- **修复** mcamel-mysql 8.0 环境下 binlog 过期时间不符合预期。
- **修复** mcamel-mysql 备份集列表会展示出同名集群旧的备份集。

2022-12-25

v0.5.0

新功能

- **新增** NodePort 端口冲突提前检测。
- **新增** 节点亲和性配置。

- **新增** 创建备份配置时，可校验 Bucket。
- **修复** arm 环境中无法展示默认配置。
- **修复** 创建实例时 name 校验与前端不一致。
- **修复** 重新接入变更名字的集群后，配置管理地址展示错误。
- **修复** 保存自动备份配置失败。
- **修复** 自动备份集无法展示的问题。

2022-11-08

v0.4.0

新功能

- **新增** 增加 MySQL 生命周期管理接口功能
- **新增** 增加 MySQL 详情接口功能
- **新增** 基于 grafana crd 对接 insight
- **新增** 增加与 ghippo 服务对接接口
- **新增** 增加与 kpanda 对接接口
- **新增** 单测覆盖率提升到 30%
- **新增** 新增备份恢复功能
- **新增** 新增备份配置接口
- **新增** 实例列表接口新增备份恢复来源字段
- **新增** 获取用户列表接口
- **新增** mysql-operator chart 参数，来指定 metric exporter 镜像
- **新增** 支持 arm64 架构
- **新增** 添加 arm64 operator 镜像打包
- **新增** 支持密码脱敏
- **新增** 支持服务暴露为 nodeport
- **新增** 支持 mtls

- **新增** 文档 第一次文档网站发布
- **新增** 文档 基本概念
- **新增** 文档 Concepts
- **新增** 文档 首次使用 MySQL
- **新增** 文档 删除 MySQL 实例

优化

- **优化** 将时间戳 api 字段统一调整为 int64

修复

- **修复** 修复备份列表接口模糊搜索无效
- **修复** 修复依赖漏洞
- **修复** 备份 Job 被删除后，无法展示备份任务列表
- **修复** 当镜像有大写和数字时不能被抓取到的问题

2022-10-18

v0.3

新功能

- **新增** 增加 MySQL 生命周期管理接口功能
- **新增** 增加 MySQL 详情接口功能
- **新增** 基于 grafana crd 对接 insight
- **新增** 增加与 ghippo 服务对接接口
- **新增** 增加与 kpanda 对接接口
- **新增** 单测覆盖率提升到 30%
- **新增** 新增备份恢复功能
- **新增** 新增备份配置接口
- **新增** 实例列表接口新增备份恢复来源字段
- **修复** 修复备份列表接口模糊搜索无效
- **优化** 将时间戳 api 字段统一调整为 int64

产品优势

MySQL 具有以下产品优势。

- 行业领先
  - 支持 ActiveMQ 的快速部署与生命周期管理。ActiveMQ 是目前最流行的开源多协议数据库，支持 JavaScript, C, C++, Python, .Net 等语言编写的应用和平台，能够跨应用交换消息。通过消息控制台实现对生产集群、消费集群、消息查询等的管理操作。
  - 采用业界先进、成熟的技术路线，保证平台的稳定可靠性；同时充分考虑用户交互的便利性和界面操作的易用性，采用流行的灵活且易扩展系统架构，使用先进的技术框架路线进行设计与开发，支持基于 API 调用各组件协同工作，能够满足系统纵向集成与横向整合的信息交互。
  - 采用可靠的技术架构，支持系统稳定运行，每个组件都能提供高可用性，能够保障关键组件冗余与高可用。
  - 能够将自身运行的内在状态数据，通过采集、分析、处理后，通过合理的聚合，归纳为指标数据，方便运维人员在最短时间内了解到系统运行实时状态，同时可提供对应的 API 供其它系统读取状态指标数据。
- 生命周期管理
  - 支持图形化或通过 YAML 创建、更新和删除 MySQL 实例。
  - 支持热部署、热更新，除通用技术服务组件升级、平台升级等重大升级改造外，系统部署和升级无需停机。

- 图形化界面操作

  支持以图形化方式创建、查看、更新、删除以及弹性扩容。管理员可以根据自己的偏好，定制图形化参数。

- 容器化消息平台

  MySQL 基于 Kubernetes 和 Docker，原生支持容器化部署，将资源利用率提高到最大。

- 兼容性和开放性

  - MySQL 兼容 Intel CPU 和国产化 CPU（如飞腾 ARM，华为鲲鹏 ARM 和海光 x86 等），兼容国产操作系统（如麒麟操作系统 v10，统信服务器操作系统 UOS 等），这些均有授权的电子证明和证书。
  - 具有开放的体系架构，提供开放标准 API 接口保证基础能力、管理服务、日志、监控等资源和服务被高效的调度、管理与使用，支持第三方管理平台通过 API 接口等方式实现相关编排与使用。

基本概念

本页的词汇表定义了 MySQL 背后的核心概念，以帮助您构建 MySQL 工作原理的思维模型，并了解文档在使用某些术语时所指代的内容。

- ACID 合规

  ACID 合规性是 Atomiciy、Consistency、Isolation 和 Durability 这 4 个单词的首字母缩写，是一组由原子性、一致性、隔离性和持久性组成的数据库特性，可确保高效完成数据库事务。

- ACL

  访问控制列表或 ACL 是控制对系统资源访问的用户权限列表。

- Connection Status Metric（连接状态指标）

  连接状态指标是衡量创建、连接和运行的线程数与数据库连接限制相关的指标。

- DBaaS（数据库即服务）

  数据库即服务、托管数据库服务，简称为 DBaaS，这是一种云服务，允许用户在订阅的基础上访问云数据库系统，而无需拥有个人云数据系统。

- E2EE

  英文全称为 End-to-end encryption，即端到端加密，或简称为 E2EE，是一种通信系统，它为所有人加密消息和消息服务，但接收消息的用户和发送消息的用户除外。

- Failover（故障转移）

故障转移是一种高可用性（HA）机制，可监控服务器的故障并在主服务器发生故障时将流量或操作重新路由到备用服务器。

- High Availability（高可用性）

  高可用性（HA）是一种基础架构设计方法，专注于减少停机时间和消除单点故障。

- Hot Standby（热备）

  热备用是侦听主节点何时发生故障以便备用节点取代其位置的行为。

- Index vs. Sequential Reads Metric（索引与顺序读取指标）

  索引与顺序读取指标图显示了使用索引的读取占主服务器上所有数据库（模式）的读取总数的比例。

- LUKS Disk Encryption（LUKS 磁盘加密）

  Linux 统一密钥设置磁盘加密（LUKS）是 Linux 存储设备的开源磁盘加密规范。

- Machine Type（机器类型）

  机器类型是用于虚拟机（VM）实例的一组虚拟化硬件资源。

- Node Plan（节点计划）

  节点计划、数据库或集群配置是节点规格的硬件计划。

- Operations Throughput Metric（运营吞吐量指标）

  操作吞吐量指标是对服务器上所有数据库的获取、插入、更新和删除操作的吞吐量的度量。

- Point-In-Time-Recovery（恢复时间点）

恢复时间点（简称 PITR）确保进行自动备份，以便恢复在服务器先前状态下创建的数据。

- Port（端口）

  端口是网络连接的通信端点。使用每个传输协议所用的端口号来标识一个端口。

- Read-Only Node（只读节点）

  只读节点是集群主节点的副本。

- SQL Mode（SQL 模式）

  SQL 模式或 sql_mode 是一个 MySQL 系统变量，用于配置 MySQL 服务器的操作特性。

- SSL Certificate（SSL 证书）

  SSL 证书是描述网站身份的数字凭证。

- Standby Node（备用节点）

  备用节点是在热备模式时设为空闲待用的节点。

- Tag（标记）

  标记是与资源相关联的关键字，有助于管理资源所有权并组织对资源的查找和操作。

什么是 MySQL

MySQL 是应用最广泛的开源关系数据库，是许多常见网站、应用程序和商业产品使用的主要关系数据存储，具有高吞吐、低延迟、可扩展等特性。

它是应用层协议的一个开放标准，是面向消息的中间件而设计，基于此协议的客户端与数据库来传递消息，不受产品、开发语言等条件的限制。

创建 MySQL 实例

功能特性

本页说明有关 MySQL 的功能特性。

- 高可用架构

  支持主从热备架构，灵活满足各类可用性需求，稳定可靠的性能远超业界平均水准。

- 快速部署

  可在线快速部署实例，图形化界面操作简便，节省采购、部署、配置等自建数据库工作，缩短项目周期，帮助业务快速上线。

- 确保数据安全

  基于实时副本技术，适配公有云、混合云和私有云，严密账号确保数据安全。

- 更低成本

  可依据业务需求即时开通所需资源，无需在业务初期采购高成本硬件，有效减少初期的资产投入，避免资源闲置浪费。

- 弹性扩缩

  可依据业务压力弹性扩缩数据库资源，满足不同业务阶段对于数据库性能和存储空间的需求。

- 自动运维

  可设置自动备份策略、监控告警策略、自动扩容策略等。

在 DCE 5.0 中部署 MySQL 后，还将支持以下特性：

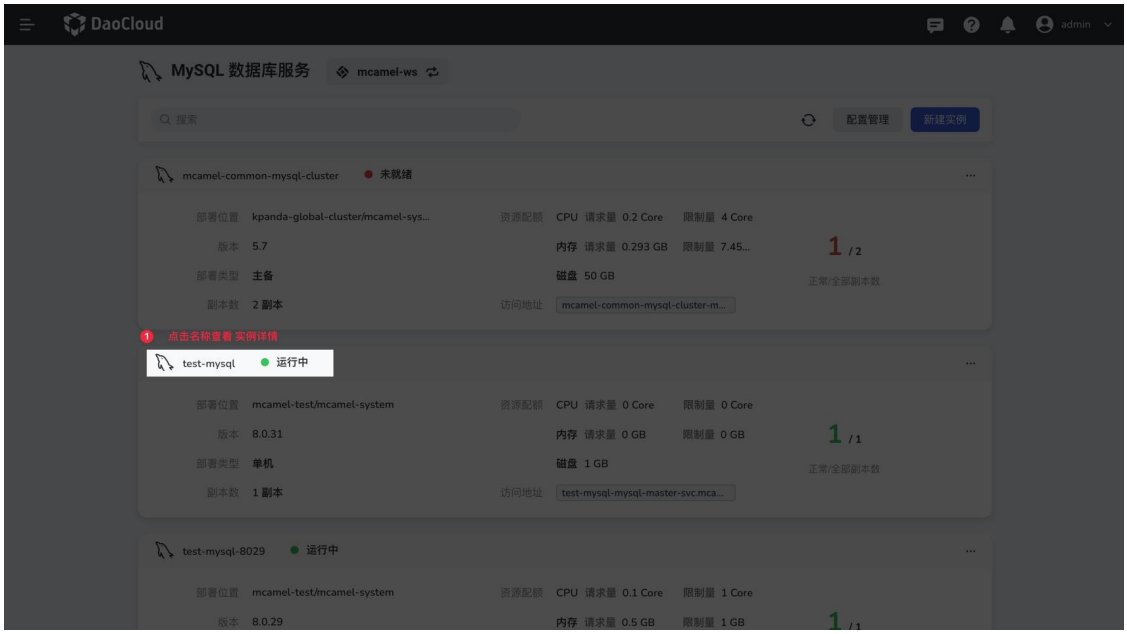
- 基于 Orchestrator 实现 MySQL 高可用和拓扑管理
- 支持单节点和主备模式
- 支持 phpmyadmin，提供管理页面
- 基于 mysqld-exporter 暴露指标
- 使用 Grafana Operator 集成 MySQL Dashboard，展示监控数据
- 使用 S erviceMonitor 对接 Prometheus 抓取指标
- 支持备份、恢复（依赖支持 S3 协议的存储）

查看 MySQL 日志

操作步骤

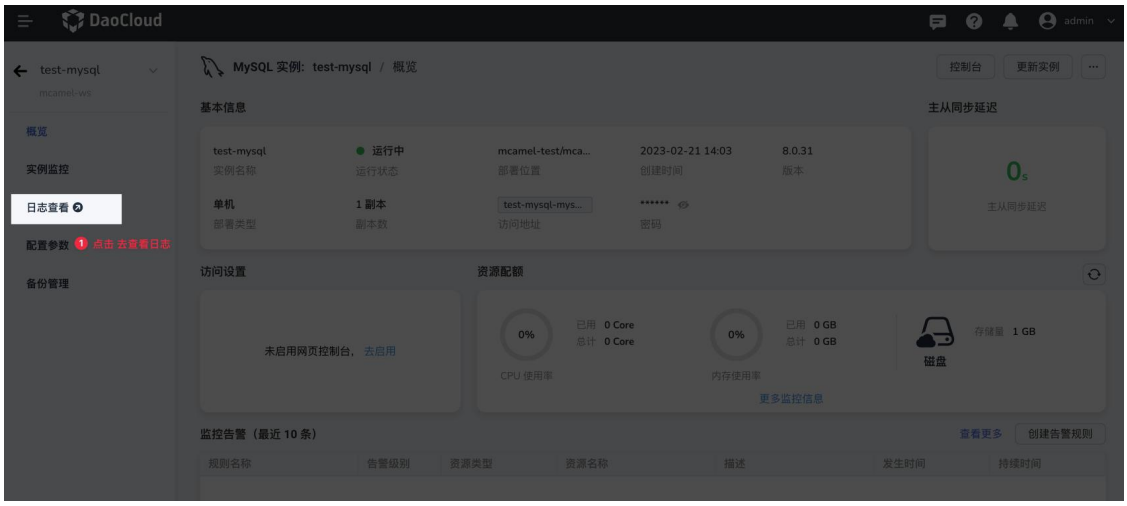
通过访问每个 MySQL 的实例详情，页面；可以支持查看 MySQL 的日志。

1. 在 MySQL 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



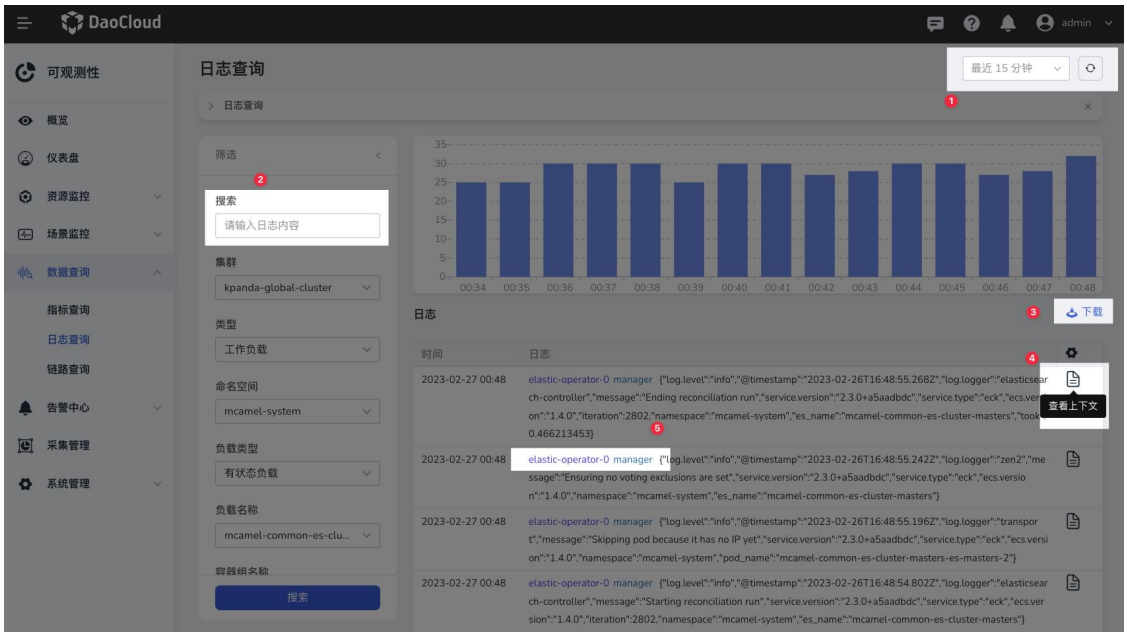
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

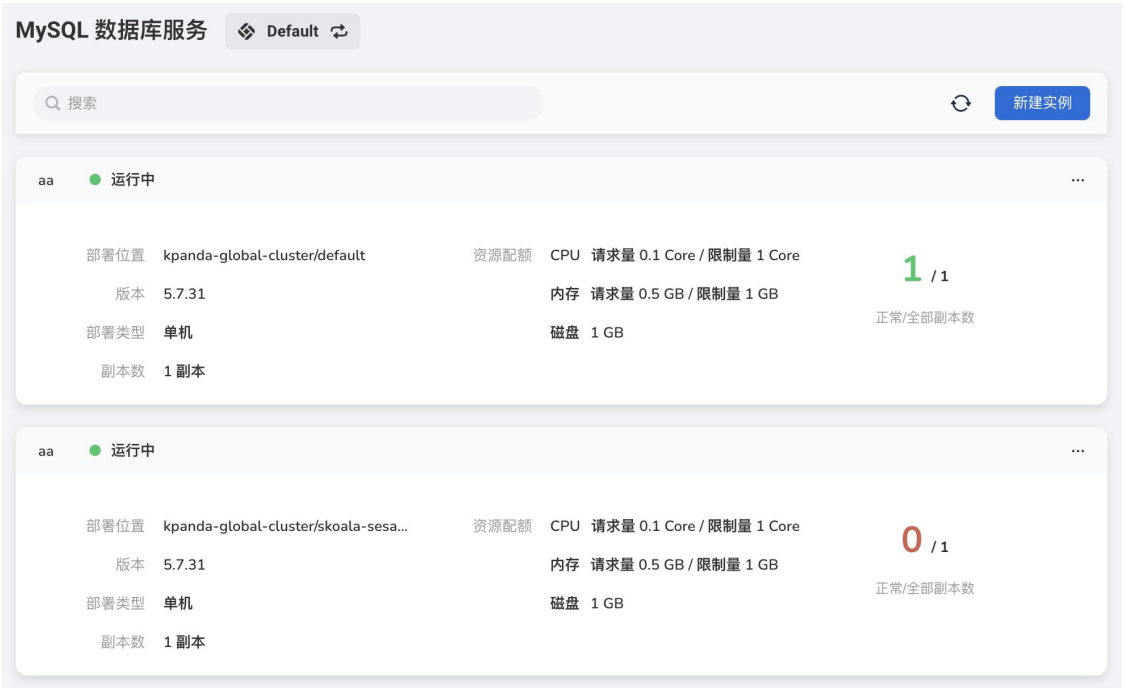

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

创建 MySQL 实例

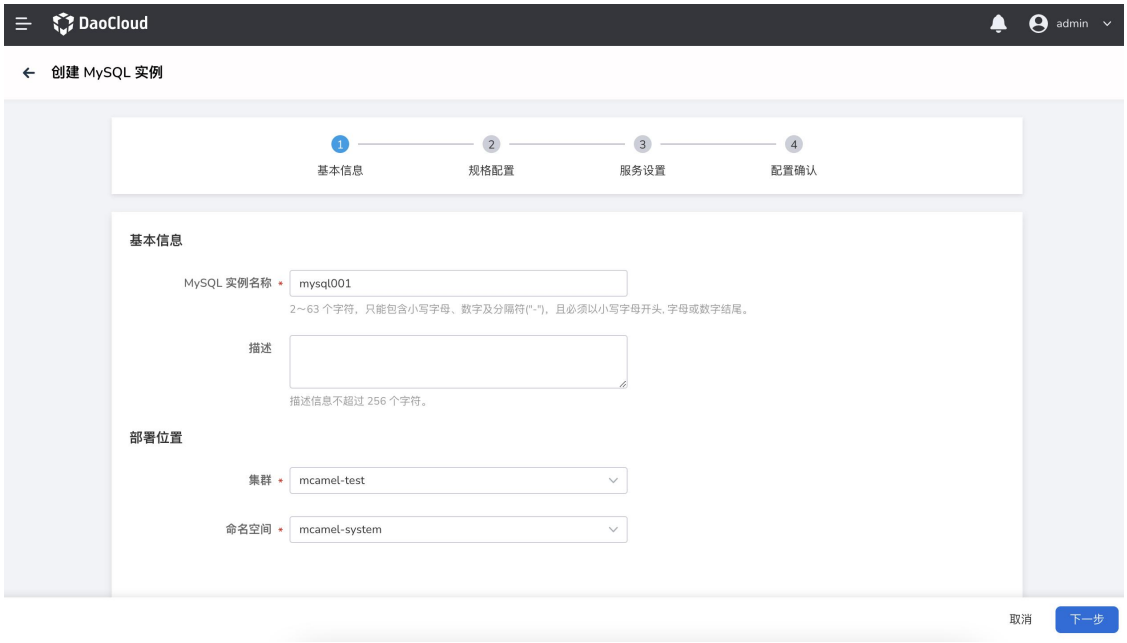
接入 MySQL 数据库后，参照以下步骤创建 MySQL 实例。

1. 在实例列表中，点击右上角的 新建实例。



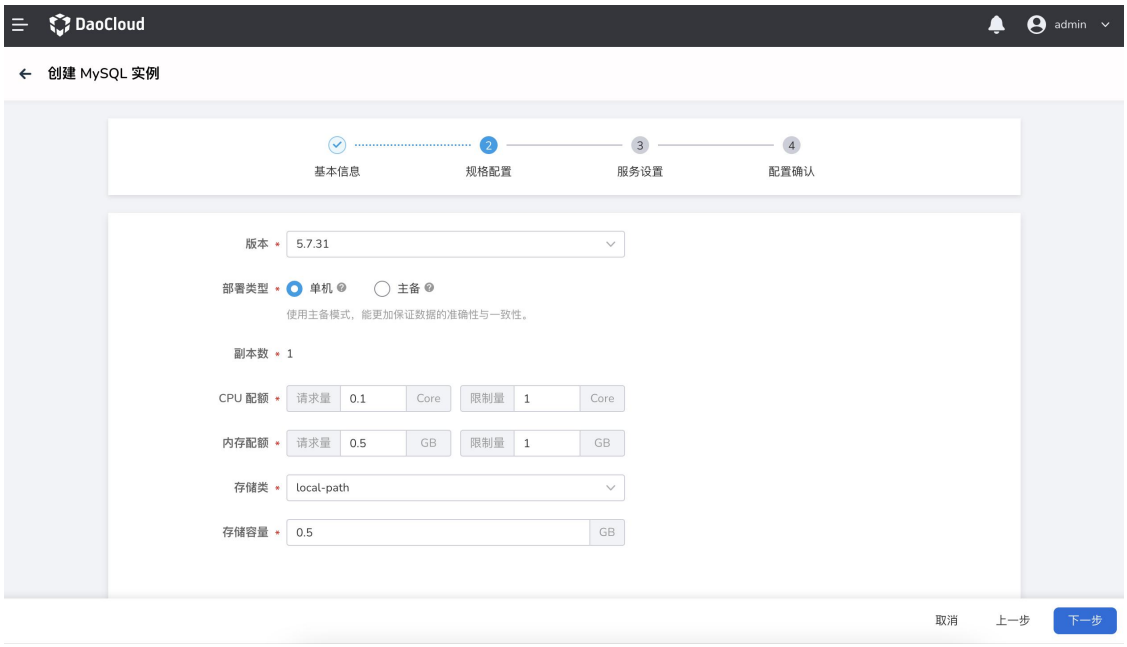
基本信息

2. 在创建 MySQL 实例页面中，配置基本信息后，点击下一步。



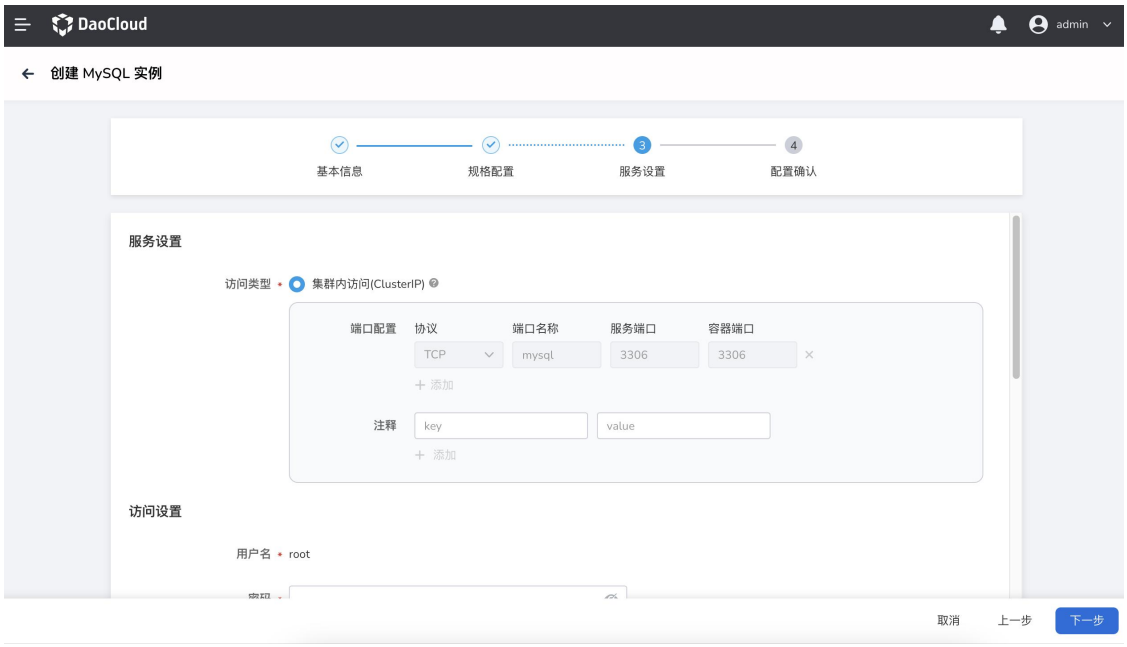
基本信息

3. 选择部署类型、CPU、内存和存储等规格配置后，点击下一步。



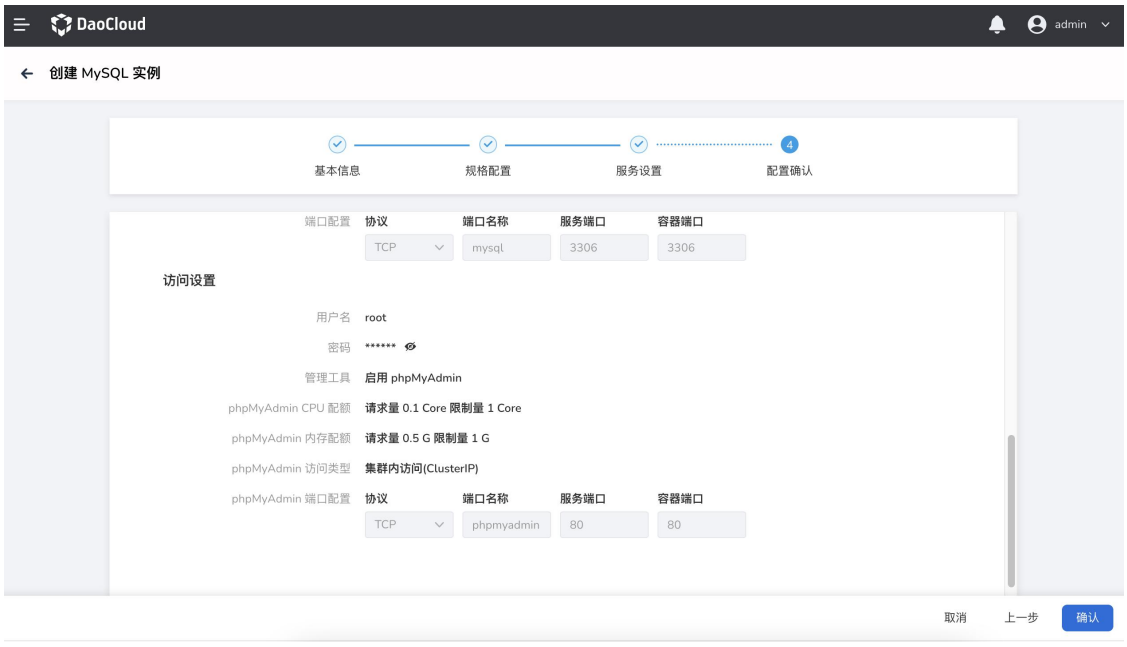
规格配置

4. 设置用户名和密码等服务设置，默认采用 ClusterIP 作为访问方式。



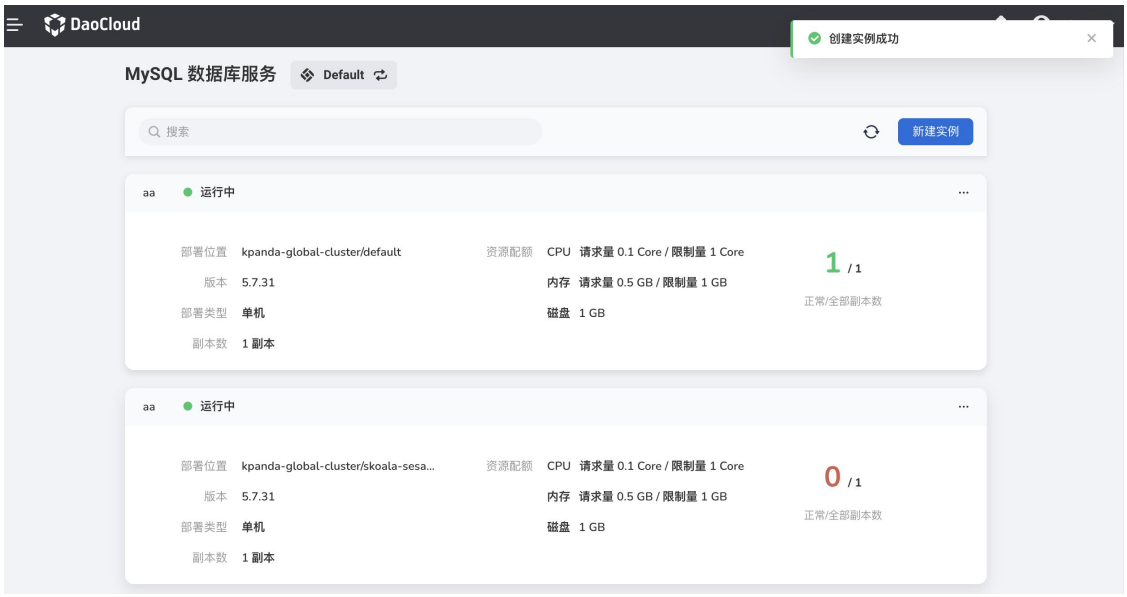
服务设置

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击确认。



确认

6. 返回实例列表，屏幕将提示创建实例成功。

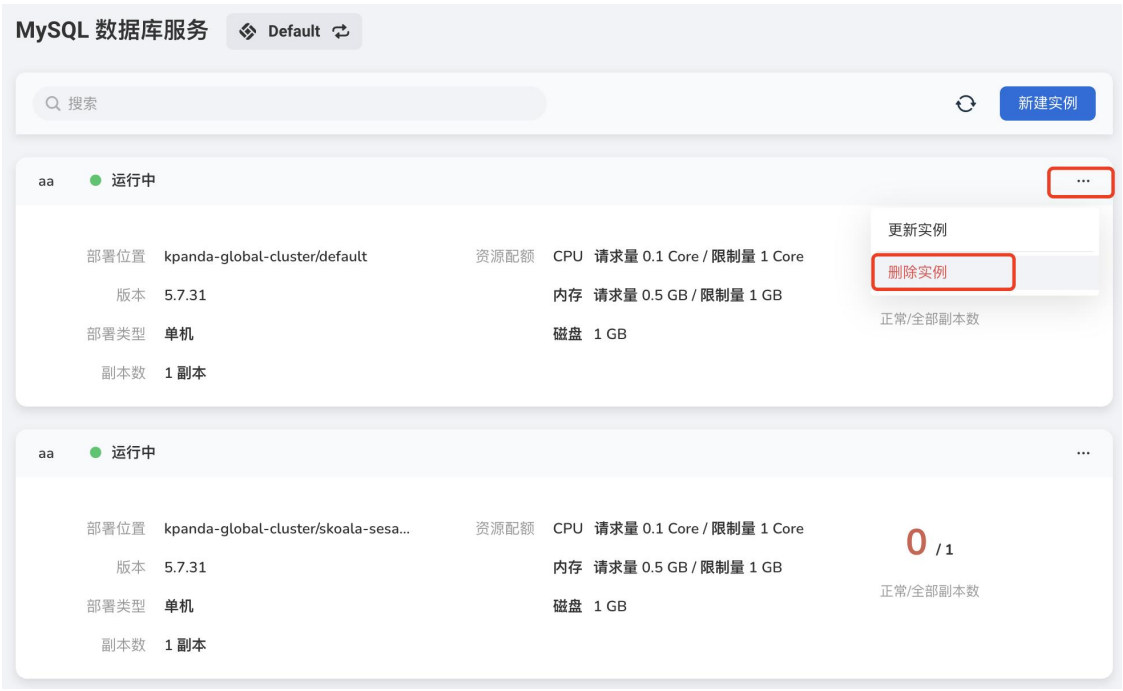

DaoCloud
创建实例成功
MySQL 数据库服务
Default
搜索
新建实例
aa
运行中
部署位置 kpanda-global-cluster/default
版本 5.7.31
部署类型 单机
副本数 1 副本
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.5 GB / 限制量 1 GB
磁盘 1 GB
1 / 1
正常/全部副本数
aa
运行中
部署位置 kpanda-global-cluster/skoala-sesa...
版本 5.7.31
部署类型 单机
副本数 1 副本
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.5 GB / 限制量 1 GB
磁盘 1 GB
0 / 1
正常/全部副本数

成功创建

删除 MySQL 实例

如果想要删除一个 MySQL 实例，可以执行如下操作：

1. 在 MySQL 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。



点击删除

!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

更新 MySQL 实例

如果想要更新或修改 MySQL 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



更新实例

2. 基本信息：只能修改描述。然后点击下一步。



基本信息

3. 修改规格配置后点击下一步。



规格配置

4. 修改服务设置后点击确认。


更新实例 aa
基本信息
规格配置
3
服务设置
访问类型 * 集群内访问(ClusterIP)
端口配置
协议
端口名称
服务端口
容器端口
TCP
mysql
3306
3306
添加
注释
key
value
添加
访问设置
管理工具
关闭 phpMyAdmin
取消
上一步
确认

服务设置

MySQL Pod

可以通过以下命令快速的查看当前集群上所有 MySQL 实例的健康状态：

```
kubectl get mysql -A
```

输出类似于：

```
NAMESPACE          NAME                              READY   REPLICAS   AGE
ghippo-system      test                              True    1          3d
mcamel-system      mcamel-common-mysql-cluster       False   2          62d
```

针对不同的副本状态，排障方案如下文所述。

Pod running = 0/4，状态为 Init:Error

遇到此类问题时，首先应该查看 master 节点的（sidecar）日志信息。

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep master | awk '{print $1}' | xargs -I {} kubectl logs -f {} -n mcamel-system -c sidecar
```

??? note “输出内容示例”

```none
2023-02-09T05:38:56.208445-00:00 0 [Note] [MY-011825] [Xtrabackup] perl binary not found. Skipping the version check
2023-02-09T05:38:56.208521-00:00 0 [Note] [MY-011825] [Xtrabackup] Connecting to MySQL server host: 127.0.0.1, user: sys_replication, password: set, port: not set, socket: not set
2023-02-09T05:38:56.212595-00:00 0 [Note] [MY-011825] [Xtrabackup] Using server version 8.0.29
2023-02-09T05:38:56.217325-00:00 0 [Note] [MY-011825] [Xtrabackup] Executing LOCK INSTANCE FOR BACKUP ...
```

2023-02-09T05:38:56.219880-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Found tables with row versions due to INSTANT ADD/DROP columns
2023-02-09T05:38:56.219931-00:00 0 [ERROR] [MY-011825] [Xtrabackup] This feature is not stable and will cause backup corruption.
2023-02-09T05:38:56.219940-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Please check https://docs.percona.com/percona-xtrabackup/8.0/em/instant.html for more details.
2023-02-09T05:38:56.219945-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Tables found:
2023-02-09T05:38:56.219951-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/USER_SESSION
2023-02-09T05:38:56.219956-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATION_EXECUTION
2023-02-09T05:38:56.219960-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATION_FLOW
2023-02-09T05:38:56.219968-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATOR_CONFIG
2023-02-09T05:38:56.219984-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/CLIENT_SESSION
2023-02-09T05:38:56.219991-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/IDENTITY_PROVIDER
2023-02-09T05:38:56.219998-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/PROTOCOL_MAPPER
2023-02-09T05:38:56.220005-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/RESOURCE_SERVER_SCOPE
2023-02-09T05:38:56.220012-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/REQUIRED_ACTION_PROVIDER
2023-02-09T05:38:56.220018-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/COMPONENT
2023-02-09T05:38:56.220027-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/RESOURCE_SERVER
2023-02-09T05:38:56.220036-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc

```
loak/CREDENTIAL
2023-02-09T05:38:56.220043-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/FED_USER_CREDENTIAL
2023-02-09T05:38:56.220049-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/MIGRATION_MODEL
2023-02-09T05:38:56.220054-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/REALM
2023-02-09T05:38:56.220062-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/CLIENT
2023-02-09T05:38:56.220069-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/REALM_ATTRIBUTE
2023-02-09T05:38:56.220075-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keyc
loak/OFFLINE_USER_SESSION
2023-02-09T05:38:56.220084-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Plea
se run OPTIMIZE TABLE or ALTER TABLE ALGORITHM=COPY on all listed tabl
es to fix this issue.
E0209 05:38:56.223635 1 deleg.go:144] sidecar "msg"="failed waiting for
xtrabackup to finish" "error"="exit status 1"
```

登录 master 节点的 MySQL，执行 alter 表结构：

```
[root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cl
uster-mysql | grep master
mcamel-common-mysql-cluster-mysql-0

#获取密码
[root@master-01 ~]$ kubectl get secret -n mcamel-system mcamel-common-m
ysql-cluster-secret -o=jsonpath='{.data.ROOT_PASSWORD}' | base64 -d

[root@master-01 ~]$ kubectl exec -it mcamel-common-mysql-cluster-mysql-0
-n mcamel-system -c mysql -- /bin/bash
```

 # 注意：修改表结构需要 root 权限登录

~bash:mysql –uroot –p

??? note“SQL 语句如下”

```sql
use keycloak;
ALTER TABLE USER_SESSION ALGORITHM=COPY;
ALTER TABLE AUTHENTICATION_EXECUTION ALGORITHM=COPY;
ALTER TABLE AUTHENTICATION_FLOW ALGORITHM=COPY;
ALTER TABLE AUTHENTICATOR_CONFIG ALGORITHM=COPY;
ALTER TABLE CLIENT_SESSION ALGORITHM=COPY;
ALTER TABLE IDENTITY_PROVIDER ALGORITHM=COPY;
ALTER TABLE PROTOCOL_MAPPER ALGORITHM=COPY;
ALTER TABLE RESOURCE_SERVER_SCOPE ALGORITHM=COPY;
ALTER TABLE REQUIRED_ACTION_PROVIDER ALGORITHM=COPY;
ALTER TABLE COMPONENT ALGORITHM=COPY;
ALTER TABLE RESOURCE_SERVER ALGORITHM=COPY;
ALTER TABLE CREDENTIAL ALGORITHM=COPY;
ALTER TABLE FED_USER_CREDENTIAL ALGORITHM=COPY;
ALTER TABLE MIGRATION_MODEL ALGORITHM=COPY;
ALTER TABLE REALM ALGORITHM=COPY;
ALTER TABLE CLIENT ALGORITHM=COPY;
ALTER TABLE REALM_ATTRIBUTE ALGORITHM=COPY;
ALTER TABLE OFFLINE_USER_SESSION ALGORITHM=COPY
```

Pod running = 2/4

此类问题很可能是因为 MySQL 实例使用的磁盘用量达到了 100%，可以在 master 节点上运行以下命令检测磁盘用量。

kubectl get pod -n mcamel-system | grep cluster-mysql | awk '{print $1}' | xargs -I {} kubectl exec {} -n mcamel-system -c sidecar -- df -h | grep /var/lib/mysql

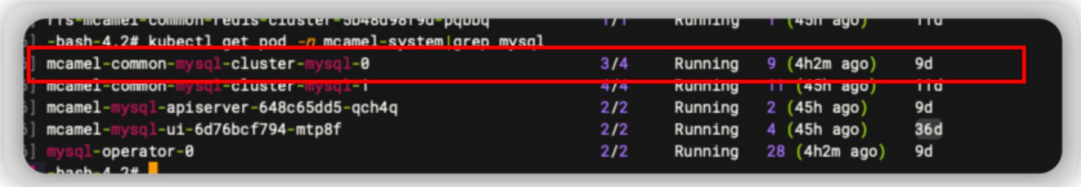
输出类似于：

```
/dev/drbd43001          50G   30G   21G  60% /var/lib/mysql
/dev/drbd43005          80G   29G   52G  36% /var/lib/mysql
```

如果发现某个 PVC 满了，则需要为 PVC 扩容。

kubectl edit pvc data-mcamel-common-mysql-cluster-mysql-0 -n mcamel-system *# 修改 request 大小即可*

Pod running = 3/4



image

使用 kubectl describe 上图中框起来的 Pod，发现异常提示： Warning Unhealthy 4m50s (x7194 over 3h58m) kubelet Readiness probe failed:

此时需要手工进行修复，这是目前开源 mysql-operator 版本的 Bug

修复方式有两种：

- 重启 mysql-operator，或者
- 手工更新 sys_operator 的配置状态

kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'update sys_operator.status set value="1" WHERE name="configured"'

MySQL 健康检查

如果您发现遇到的问题，未包含在故障排查内，可以快速跳转到页面底部，提交您的问题。

常规的 MySQL 健康状态检查，可以通过一句命令快速查看 MySQL 实例的整体状态:

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep mysql
```

输出类似于：

```
mcamel-common-mysql-cluster-auto-2023-03-28t00-00-00-backujgg9m    0/1
     Completed            0                   27h
mcamel-common-mysql-cluster-auto-2023-03-29t00-00-00-backusgf59    0/1
    Completed            0                   3h43m
mcamel-common-mysql-cluster-mysql-0                                 4/4
 Running              6 (11h ago)      25h      yes        master
mcamel-common-mysql-cluster-mysql-1                                 4/4
Running              690 (11h ago)    4d20h    yes        replica
mcamel-mysql-apiserver-9797c7f76-bvf5n                              2/2
Running              0                22h
mcamel-mysql-ui-7ffd9dd8db-d5jfm                                    2/2
Running              0                25m
mysql-operator-0                                                    2/2     Ru
nning              109 (47m ago)    2d21h
```

如上所示，如果主备节点（master replica）的状态均为 yes,说明 MySQL 为正常状态。

其他故障

CR 创建数据库失败报错

数据库运行正常，使用 CR 创建数据库出现了报错，此类问题的原因有：mysql root 密码有特殊字符



image

1. 获取查看原密码：

   [root@master-01 ~]$ kubectl get secret -n mcamel-system mcamel-common-mysql-cluster-secret -o=jsonpath='{.data.ROOT_PASSWORD}' | base64 -d

2. 如果密码含有特殊字符 - ，进入 MySQL 的 shell 输入原密码出现以下错误

   bash-4.4# mysql -uroot -p
   Enter password:
   ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)

3. 清理重建：

   - 方法一：清理数据目录，删除 Pod 等待 sidecar running 以后，再删除一次数据目录，再删除 Pod 即可恢复：

   [root@master-01 ~]# kubectl exec -it mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c sidecar -- /bin/sh

```
sh-4.4# cd /var/lib/mysql
sh-4.4# ls | xargs rm -rf
```

- 方法二：删除 PVC 再删除 Pod，只需要处理一次,即可恢复：

```
kubectl delete pvc data-mcamel-common-mysql-cluster-mysql-1 -n mcamel-system
```

```
kubectl delete pod mcamel-common-mysql-cluster-mysql-1 -n mcamel-system
```

MySQL Operator

未指定 storageClass

由于没有指定 storageClass，导致 mysql-operator 无法获取 PVC 而处于 pending 状态。

如果采用 Helm 启动，可以做如下设置：

1. 关闭 PVC 的申请

   orchestrator.persistence.enabled=false

2. 指定 storageClass 去获取 PVC

   orchestrator.persistence.storageClass={storageClassName}

如果使用其他工具，可以修改 value.yaml 内对应字段，即可达到和 Helm 启动一样的效果。

MySQL 主备关系

MySQL 的主备关系故障相对比较复杂，基于不同现象，会有不同的解决方案。 1. 执行以下命令确认 MySQL 状态:

```bash
kubectl get mysql -A
```

输出类似于:

```
NAMESPACE NAME READY REPLICAS AGE
ghippo-system test True 1 3d
mcamel-system mcamel-common-mysql-cluster False 2 62d
```

2. 关注 Ready 字段值为 False 的库（这里为 True 的判断是延迟小于 30s 同步），查看 MySQL 从库的日志

   ```
   [root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mysql | grep ERROR
   ```

   当实例状态为 False 时，可能存在以下几类故障，可以结合库日志信息排查修复。

实例状态为 false 但日志无报错信息

如果从库的日志中没有任何错误 ERROR 信息，说明 False 只是因为主从同步的延迟过大，可对从库执行以下命令进一步排查:

1. 寻找到从节点的 Pod

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
```

输出类似于：

```
mcamel-common-mysql-cluster-mysql-1
```

2. 设置 binlog 参数

```
kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'set global sync_binlog=10086;'
```

3. 进入 MySQL 的容器

```
kubectl exec -it mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf
```

4. 在 MySQL 容器中执行查看命令，获取从库状态。

Seconds_Behind_Master 字段为主从延迟，如果取值在 0~30，可以认为没有延迟；表示主从可以保持同步。

??? note “SQL 语句如下”

```sql
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for source to send event
                  Master_Host: mcamel-common-mysql-cluster-mysql-0.mysql.mcamel-system
                  Master_User: sys_replication
                  Master_Port: 3306
                Connect_Retry: 1
              Master_Log_File: mysql-bin.000304
          Read_Master_Log_Pos: 83592007
```

Relay_Log_File: mcamel-common-mysql-cluster-mysql-1-relay-bin.000002
Relay_Log_Pos: 83564355
Relay_Master_Log_File: mysql-bin.000304
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 83564299
Relay_Log_Space: 83592303
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:

```
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 100
                    Master_UUID: e17dae09-8da0-11ed-9104-c2f9484728f
d
               Master_Info_File: mysql.slave_master_info
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
        Slave_SQL_Running_State: Replica has read all relay log; waiting f
or more updates
            Master_Retry_Count: 86400
                    Master_Bind:
        Last_IO_Error_Timestamp:
       Last_SQL_Error_Timestamp:
                 Master_SSL_Crl:
            Master_SSL_Crlpath:
            Retrieved_Gtid_Set: e17dae09-8da0-11ed-9104-c2f9484728fd:2
1614244-21621569
              Executed_Gtid_Set: 4bc2107c-819a-11ed-bf23-22be07e4eaff:
1-342297,
 7cc717ea-7c1b-11ed-b59d-c2ba3f807d12:1-619197,
 a5ab763a-7c1b-11ed-b5ca-522707642ace:1-178069,
 a6045297-8743-11ed-8712-8e52c3ace534:1-4073131,
 a95cf9df-84d7-11ed-8362-5e8a1c335253:1-493942,
 b5175b1b-a2ac-11ed-b0c6-d6fbe05d7579:1-3754703,
 c4dc2b14-9ed9-11ed-ac61-36da81109699:1-945884,
 e17dae09-8da0-11ed-9104-c2f9484728fd:1-21621569
                  Auto_Position: 1
          Replicate_Rewrite_DB:
                  Channel_Name:
            Master_TLS_Version:
        Master_public_key_path:
          Get_master_public_key: 0
```

```
                    Network_Namespace:
1 row in set, 1 warning (0.00 sec)
```

5. 主从同步后 Seconds_Behind_Master 小于 30s，设置 sync_binlog=1

   ```
   kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'set global sync_binlog=1';
   ```

6. 如果此时依然不见缓解，可以查看从库的宿主机负载或者 IO 是否太高，执行以下命令：

   ```
   [root@master-01 ~]$ uptime
   11:18  up 1 day, 17:49, 2 users, load averages: 9.33 7.08 6.28
   ```

   load averages 在正常情况下 3 个数值都不应长期超过 10；如果超过 30 以上，请合理调配下该节点的 Pod 和磁盘。

从库日志出现复制错误

如果从库 Pod 日志中出现从库复制错误，可能由多种原因引起，下文将针对不同情况介绍判断及修复方法。

purged binlog 错误

注意以下示例，如果出现关键字 purged binlog，通常需要对从库执行重建处理。

??? note “错误示例”

```bash
[root@demo-alpha-master-01 /]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mysql | grep ERROR
2023-02-08T18:43:21.991730Z 116 [ERROR] [MY-010557] [Repl] Error reading packet from server for channel '': Cannot replicate because the master purged required binary logs. Replicate the missing transactions from elsewhere, or provisi
```

on a new slave from backup. Consider increasing the master's binary log expiration period. The GTID sets and the missing purged transactions are too long to print in this message. For more information, please see the master's error log or the manual for GTID_SUBTRACT (server_errno=1236)
2023-02-08T18:43:21.991777Z 116 [ERROR] [MY-013114] [Repl] Slave I/O for channel '': Got fatal error 1236 from master when reading data from binary log: 'Cannot replicate because the master purged required binary logs. Replicate the missing transactions from elsewhere, or provision a new slave from backup. Consider increasing the master's binary log expiration period. The GTID sets and the missing purged transactions are too long to print in this message. For more information, please see the master's error log or the manual for GTID_SUBTRACT', Error_code: MY-013114
```

重建操作如下：

1. 寻找从节点的 Pod

   ```
   [root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
   mcamel-common-mysql-cluster-mysql-1
   ```

2. 寻找从节点的 PVC

   ```
   [root@master-01 /]$ kubectl get pvc -n mcamel-system | grep mcamel-common-mysql-cluster-mysql-1
   data-mcamel-common-mysql-cluster-mysql-1
                   Bound    pvc-5840569e-834f-4236-a5c6-878e41c55c85
    50Gi       RWO            local-path                33d
   ```

3. 删除从节点的 PVC

   ```
   [root@master-01 /]$ kubectl delete pvc data-mcamel-common-mysql-cluster-mysql-1 -n mcamel-system
   ```
   ```

persistentvolumeclaim "data-mcamel-common-mysql-cluster-mysql-1" deleted

4. 删除从库的 Pod

```
[root@master-01 /]$ kubectl delete pod mcamel-common-mysql-cluster-mysql-1 -n mcamel-system
pod "mcamel-common-mysql-cluster-mysql-1" deleted
```

主键冲突错误

??? note “错误实例”

```bash
[root@demo-alpha-master-01 /]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mysql | grep ERROR
2023-02-08T18:43:21.991730Z 116 [ERROR] [MY-010557] [Repl] Could notexecute Write_rows event on table dr_brower_db.dr_user_info; Duplicate entry '24' for key 'PRIMARY', Error_code:1062; handler error HA_ERR_FOUND_DUPP_KEY; the event's master logmysql-bin.000010, end_log_pos 5295916
```

如果在错误日志中看到：Duplicate entry '24' for key 'PRIMARY', Error_code:1062; handler error HA_ERR_FOUND_DUPP_KEY;,

说明出现了主键冲突，或者主键不存在的错误。此时，可以以幂等模式恢复或插入空事务的形式跳过错误：

方法 1：幂等模式恢复

1. 寻找到从节点的 Pod

```
[root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
mcamel-common-mysql-cluster-mysql-1
```

2. 设置 mysql 幂等模式

```
[root@master-01 ~]$ kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;set global slave_exec_mode="IDEMPOTENT";set global sync_binlog=10086;start slave;'
```

方法 2：插入空事务跳过错误

```
mysql> stop slave;
mysql> SET @@SESSION.GTID_NEXT= 'xxxxx:105220'; /* 具体数值，在日志里面提到 */
mysql> BEGIN;
mysql> COMMIT;
mysql> SET SESSION GTID_NEXT = AUTOMATIC;
mysql> START SLAVE;
```

执行完成以上操作后，观察从库重建的进度：

```
# 进入 mysql 的容器
[root@master-01 ~]$ kubectl exec -it mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf
```

执行以下命令，查看从库的主从延迟状态字段 Seconds_Behind_Master，如果取值在 0~30，表示已没有主从延迟，主库和从库基本保持同步。

```
mysql> show slave status\G;
```

确认主从同步后（Seconds_Behind_Master 小于 30s），执行以下命令，设定 MySQL 严格模式：

```
[root@master-01 ~]$ kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;set global slave_exec_mode="STRICT";set global sync_binlog=10086;start slave;
```

主从库复制错误

当从库出现类似 [Note] Slave: MTS group recovery relay log info based on Worker-Id 0, group_r 的错误信息，可以执行如下操作：

1. 寻找到从节点的 Pod

   ```
   [root@master-01 ~]# kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
   mcamel-common-mysql-cluster-mysql-1
   ```

2. 设置让从库跳过这个日志继续复制

   ```
   [root@master-01 ~]# kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;reset slave;change master to MASTER_AUTO_POSITION = 1;start slave;';
   ```

   提示:

   1.这种情况可以以幂等模式执行。

   2.此种类型错误也可以重做从库。

主备 Pod 均为 replica

1. 通过以下命令，发现两个 MySQL 的 Pod 均为 replica 角色，需修正其中一个为 master。

   ```
   [root@aster-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role|grep mysql
   mcamel-common-mysql-cluster-mysql-0                                4/4
     Running    5 (16h ago)    16h    no          replica
   mcamel-common-mysql-cluster-mysql-1                                4/4
     Running    6 (16h ago)    16h    no          replica
   mysql-operator-0                                                   2/2      Running    1 (16h ago)    16h
   ```

2. 进入 MySQL 查看：

   kubectl exec -it mcamel-common-mysql-cluster-mysql-0 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf

3. 查看 slave 的状态信息，查询结果为空的就是原来的 master，如下方示例中 mysql-0 对应的内容：

   ??? note “状态信息示例“

   ```sql
   -- mysql-0
   mysql> show slave status\G;
   empty set, 1 warning (0.00 sec)


   -- mysql-1
   mysql> show slave status\G;
   *************************** 1. row ***************************
                  Slave_IO_State: Waiting for source to send event
                     Master_Host: mcamel-common-mysql-cluster-mysql-0.mysql.mcamel-system
                     Master_User: sys_replication
                     Master_Port: 3306
                    Connect_Retry: 1
                  Master_Log_File: mysql-bin.000004
              Read_Master_Log_Pos: 38164242
                   Relay_Log_File: mcamel-common-mysql-cluster-mysql-1-relay-bin.000002
                    Relay_Log_Pos: 38164418
            Relay_Master_Log_File: mysql-bin.000004
                 Slave_IO_Running: Yes
                 Slave_SQL_Running: Yes
                   Replicate_Do_DB:
   ```

```
              Replicate_Ignore_DB:
              Replicate_Do_Table:
          Replicate_Ignore_Table:
          Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
                       Last_Errno: 0
                       Last_Error:
                     Skip_Counter: 0
              Exec_Master_Log_Pos: 38164242
                  Relay_Log_Space: 38164658
                  Until_Condition: None
                   Until_Log_File:
                    Until_Log_Pos: 0
              Master_SSL_Allowed: No
              Master_SSL_CA_File:
              Master_SSL_CA_Path:
                 Master_SSL_Cert:
               Master_SSL_Cipher:
                  Master_SSL_Key:
           Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                 Last_IO_Errno: 0
                 Last_IO_Error:
                Last_SQL_Errno: 0
                Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 100
                    Master_UUID: c16da70b-ad12-11ed-8084-0a580a8102
56
               Master_Info_File: mysql.slave_master_info
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
```

```
            Slave_SQL_Running_State: Replica has read all relay log; waiting f
or more updates
                  Master_Retry_Count: 86400
                         Master_Bind:
              Last_IO_Error_Timestamp:
             Last_SQL_Error_Timestamp:
                       Master_SSL_Crl:
                  Master_SSL_Crlpath:
                  Retrieved_Gtid_Set: c16da70b-ad12-11ed-8084-0a580a810256:
537-59096
                   Executed_Gtid_Set: c16da70b-ad12-11ed-8084-0a580a81025
6:1-59096
                       Auto_Position: 1
                Replicate_Rewrite_DB:
                       Channel_Name:
                  Master_TLS_Version:
              Master_public_key_path:
                Get_master_public_key: 0
                    Network_Namespace:
1 row in set, 1 warning (0.01 sec)
```

4. 针对 master 的 mysql shell 执行重置操作：

   mysql > **stop** slave;**reset** slave;

5. 此时再手动编辑 master 的 Pod：role replica => master ,healthy no => yes。

6. 针对 slave 的 mysql shell 执行：

   mysql > **start** slave;

7. 如果主从没有建立联系，在 slave 的 mysql shell 执行：

```
-- 注意替换下 {master-host-pod-index}
mysql > change master to master_host='mcamel-common-mysql-cluster-mysql-{master-host-pod-index}.mysql.mcamel-system',master_port=3306, master_user='root',master_password='{password}',master_auto_position=1,MASTER_HEARTBEAT_PERIOD=2,MASTER_CONNECT_RETRY=1, MASTER_RETRY_COUNT=86400;
```

主备数据不一致

当主从实例数据不一致时，可以执行以下命令完成主从一致性同步：

```
pt-table-sync --execute --charset=utf8 --ignore-databases=mysql,sys,percona --databases=amamba,audit,ghippo,insight,ipavo,keycloak,kpanda,skoala dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql-0.mysql.mcamel-system,P=3306 dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql.mysql.mcamel-system,P=3306 --print

pt-table-sync --execute --charset=utf8 --ignore-databases=mysql,sys,percona --databases=kpanda dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql-0.mysql.mcamel-system,P=3306 dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql-1.mysql.mcamel-system,P=3306 --print
```

使用 pt-table-sync 即可完成数据补充，示例中是 mysql-0=> mysql-1 补充数据。

这种场景往往适用于主从切换，发现新从库有多余的已执行的 gtid 在重做之前补充数据。

这种补充数据只能保证数据不丢失，如果新主库已经删除的数据会被重新补充回去，是一个潜在的风险，如果是新主库有数据，会被替换成老数据，也是一个风险。

Elasticsearch 索引服务 Release Notes

本页列出 Elasticsearch 索引服务的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.9.3

新功能

- **新增** UI 界面的权限访问限制。

2023-06-30

v0.9.0

新功能

- **新增** mcamel-elasticsearch 节点反亲和配置
- **新增** mcamel-elasticsearch 监控图表，去除干扰元素并新增时间范围选择

优化

- **优化** mcamel-elasticsearch ServiceMonitor 闭环安装
- **修复** mcamel-elasticsearch 监控图表，去除干扰元素并新增时间范围选择

2023-05-30

v0.8.0

新功能

- **新增** mcamel-elasticsearch 新增 对接全局管理审计日志模块。
- **新增** mcamel-elasticsearch 新增 可配置实例监控数据采集间隔时间。
- **新增** mcamel-elasticsearch 修复 Pod 列表分页展示有误。

2023-04-27

v0.7.2

新功能

- **新增** mcamel-elasticsearch 详情页面展示相关的事件
- **新增** mcamel-elasticsearch 支持自定义角色

优化

- **优化** mcamel-elasticsearch 调度策略增加滑动按钮
- **修复** mcamel-elasticsearch 在纳管集群时可能会中断重试的问题

2023-03-28

v0.6.0

新功能

- **新增** mcamel-elasticsearch 支持中间件链路追踪适配。
- **新增** 在安装 mcamel-elasticsearch 根据参数配置启用链路追踪。
- **新增** mcamel-elasticsearch Kibana 支持 LoadBalancer 类型。

升级

- **升级** golang.org/x/net 到 v0.7.0
- **升级** GHippo SDK 到 v0.14.0

2023-02-23

v0.5.0

新功能

- **新增** mcamel-elasticsearch helm-docs 模板文件。
- **新增** mcamel-elasticsearch 应用商店中的 Operator 只能安装在 mcamel-system。
- **新增** mcamel-elasticsearch 支持 cloud shell。
- **新增** mcamel-elasticsearch 支持导航栏单独注册。

- **新增** mcamel-elasticsearch 支持查看日志。
- **新增** mcamel-elasticsearch Operator 对接 chart-syncer。
- **新增** mcamel-elasticsearch 支持 LB。
- **新增** 日志查看操作说明，支持自定义查询、导出等功能。

升级

- **升级** mcamel-elasticsearch 升级离线镜像检测脚本。

修复

- **修复** mcamel-elasticsearch 实例名太长导致自定义资源无法创建的问题。
- **修复** mcamel-elasticsearch 工作空间 Editor 用户无法查看实例密码。
- **修复** mcamel-elasticsearch 密码不能使用特殊字符的问题。
- **修复** mcamel-elasticsearch 超出索引导致 panic 的问题。

2022-12-25

v0.4.0

新功能

- **新增** mcamel-elasticsearch 获取集群已经分配的 NodePort 列表接口。
- **新增** mcamel-elasticsearch 增加状态详情。
- **新增** mcamel-elasticsearch 节点亲和性配置。

优化

- **优化** mcamel-elasticsearch 可以展示公共的 es，纳管之前是不可以删除的。

- **优化** mcamel-elasticsearch 增加健康状态返回。

修复

- **修复** mcamel-elasticsearch 修复 kb 不存在时候，删除会失败的 BUG。

- **修复** mcamel-elasticsearch 修复 es exporter 离线失效的问题。

- **修复** mcamel-elasticsearch 修复 es 创建成功后没有返回 ports 信息的 bug。
- **修复** mcamel-elasticsearch 查询实例列表和详情时，Kibana 的服务类型不符合预期。

v0.3.6

2022-11-28

- **改进** 密码校验调整为 MCamel 中等密码强度
- **改进** 角色可以升级
- **新增** 新增 sc 扩容提示
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警
- **新增** 校验 Service 注释
- **修复** 更新实例后，集群使用了错误的镜像，导致集群状态异常
- **修复** 使用 NodePort 时，更新实例报错
- **升级** 中依赖的 eck operator 版本为 2.3.0
- **优化** 在某些版本的 K8s 集群中，默认 FD 不足，无法启动的问题
- **优化** 减小 elasticsearch 容器的运行权限

2022-10-28

v0.3.4

新功能

- **新增** 同步 pod 状态到实例详情页
- **新增** 获取用户列表接口
- **新增** 支持 arm 架构

优化

- **优化** workspace 界面逻辑调整
- **优化** 不符合设计规范的样式调整
- **优化** password 获取逻辑调整
- **优化** cpu&内存请求量应该小于限制量逻辑调整
- **优化** 实例版本不允许修改，下拉框应该为文本

修复

- **修复** 更新实例服务设置，确认无反应，无法提交

2022-9-25

v0.3.2

新功能

- **新增** 列表页增加分页功能
- **新增** 增加修改配置的功能
- **新增** 增加返回可修改配置项的功能
- **新增** 更改创建实例的限制为集群级别，原来为 namespace 级别
- **新增** 增加监控地址的拼接功能
- **新增** 增加可以修改版本号的功能
- **新增** 修改底层 update 逻辑为 patch 逻辑
- **新增** 将时间戳 api 字段统一调整为 int64
- **新增** 单测覆盖率提升到 43%
- **新增** 对接全局管理增加 workspace 接口
- **新增** 对接 insight 通过 crd 注入 dashboard
- **新增** 更新 release note 脚本，执行 release-process 规范
- **新增** 支持 helm 部署 eck-operator
- **新增** 支持 helm 部署 mcamel-elasticsearch 服务
- **新增** 第一次文档网站发布
- **新增** 功能说明

- **新增** 产品优势
- **新增** 什么是 Elasticsearch
- **新增** 基本概念
- **新增** 集群容量规划

产品优势

Elasticsearch 很快

由于 Elasticsearch 是在 Lucene 基础上构建而成的，所以在全文本搜索方面表现十分出色。 Elasticsearch 同时还是一个近实时的搜索平台，这意味着从文档索引操作到文档变为可搜索状态之间的延时很短，一般只有一秒。 因此，Elasticsearch 非常适用于对时间有严苛要求的用例，例如安全分析和基础设施监测。

Elasticsearch 具有分布式的本质特征

Elasticsearch 中存储的文档分布在不同的容器中，这些容器称为分片，可以进行复制以提供数据冗余副本，以防发生硬件故障。 Elasticsearch 的分布式特性使得它可以扩展至数百台（甚至数千台）服务器，并处理 PB 量级的数据。

Elasticsearch 包含一系列广泛的功能

除了速度、可扩展性和弹性等优势以外，Elasticsearch 还有大量强大的内置功能（例如数据汇总和索引生命周期管理），可以方便用户更加高效地存储和搜索数据。

Elastic Stack 简化了数据采集、可视化和报告过程

通过与 Beats 和 Logstash 进行集成，用户能够在向 Elasticsearch 中索引数据之前轻松地处理数据。 同时，Kibana 不仅可针对 Elasticsearch 数据提供实时可视化，同时还提供 UI 以便用户快速访问应用程序性能监测（APM）、日志和基础设施指标等数据。

Elasticsearch 支持多种编程语言

- Java
- JavaScript (Node.js)

- Go
- .NET (C#)
- PHP
- Perl
- Python
- Ruby

基本概念

本节列出有关 Elasticsearch 涉及的专有名词及术语。

- 节点（node）

  数据节点：存储索引数据的节点，主要对文档进行增删改查、聚合等操作。

  专有主节点：对集群进行操作，例如创建或删除索引，跟踪哪些节点是集群的一部分，并决定哪些分片分配给相关的节点。稳定的主节点对集群的健康非常重要，默认情况下集群中的任一节点都可能被选为主节点。

  协调节点：分担数据节点的 CPU 开销，从而提高处理性能和服务稳定性。

- 索引（Index）

  用于存储 Elasticsearch 的数据，是一个或多个分片分组在一起的逻辑空间。

- 分片（Shard）

  索引可以存储数据量超过 1 个节点硬件限制的数据。为满足这样的需求，Elasticsearch 提供了一个能力，将一个索引拆分为多个，称为 Shard。当您创建一个索引时，您可以根据实际情况指定 Shard 的数量。每个 Shard 托管在集群中的任意一个节点中，且每个 Shard 本身是一个独立的、全功能的“索引”。 Shard 的数量只能在创建索引前指定，且在索引创建成功后无法修改。

- 副本（replicas）

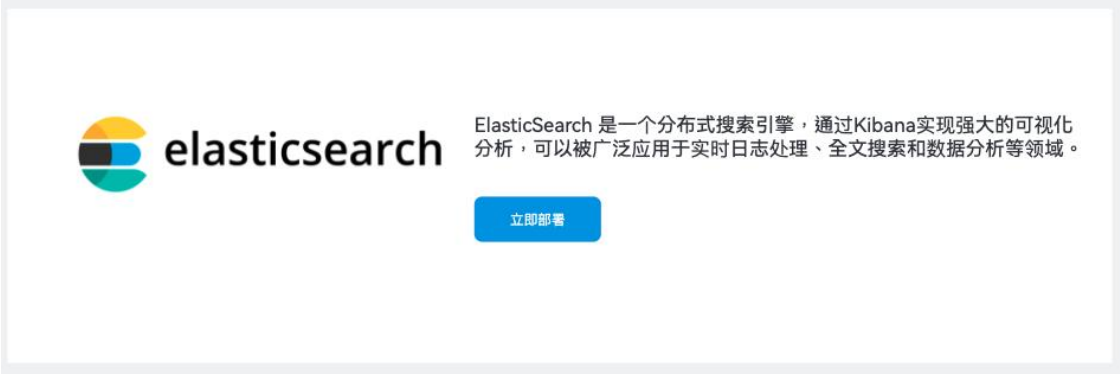
  replicas 是索引的备份，Elasticsearch 可以设置多个副本。写操作会先在主分片上完成，然后分发到副本分片上。 因为索引的主分片和副本分片都

可以对外提供查询服务，所以副本能够提升系统的高可用性和搜索时的并发性能。但如果副本太多，也会增加写操作时数据同步的负担。

什么是 Elasticsearch

Elasticsearch（下文简称 Elastic）是目前全文搜索引擎的首选。它可以快速地存储、搜索和分析海量数据。Elastic 的底层是开源库 Lucene，但是 Lucene 无法直接用，必须自己写代码去调用它的接口。Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。

DCE 5.0 内置的搜索服务基于 Elasticsearch，能够提供分布式搜索服务，为用户提供结构化、非结构化文本以及基于 AI 向量的多条件检索、统计、报表。完全兼容 Elasticsearch 原生接口。它可以帮助网站和 APP 搭建搜索框，提升用户的搜索体验；也可以用于搭建日志分析平台，助力企业实现数据驱动运维，数据驱动运营；它的向量检索能力可以帮助客户快速构建基于 AI 的图搜、推荐、语义搜索、人脸识别等丰富的应用。



欢迎界面

ElasticSearch 的工作管理

原始数据会从多个来源（包括日志、系统指标和网络应用程序）输入到 Elasticsearch 中。数据采集旨在 Elasticsearch 中进行索引之前解析、标准化并充实这些原始数据的过程。这些数据在 Elasticsearch 中完成索引之后，用户便可针对数据运行复杂的查询，并使用聚合来检索自身数据的复杂汇总。在 Kibana 中，用户可以基于自己的数据创建强大的可视化。

Kibana 是什么？

Kibana 是一款适用于 Elasticsearch 的数据可视化和管理工具，可以提供实时的直方图、线形图、饼状图和地图。Kibana 同时还包括诸如 Canvas 和 Elastic Maps 等高级应用程序；其中 Canvas 允许用户基于自身数据创建定制的动态信息图表，而 Elastic Maps 则可用来对地理空间数据进行可视化。

创建 Elasticsearch 实例

适用场景

Elasticsearch 在速度和可扩展性方面都表现出色，而且还能够索引多种类型的内容，这意味着其可用于多种使用场景：

- 应用程序搜索
- 网站搜索
- 企业搜索
- 日志处理和分析
- 基础设施指标和容器监测
- 应用程序性能监测
- 地理空间数据分析和可视化
- 安全分析
- 业务分析

功能说明

Elasticsearch 以 JSON 文档的形式存储数据。 每个文档都会在一组键（字段或属性的名称）与它们对应的值（字符串、数字、布尔值、日期、数值组、地理位置或其他类型的数据）之间建立联系。

Elasticsearch 使用的是一种名为倒排索引的数据结构，这一结构的设计可以允许十分快速地进行全文本搜索。 倒排索引会列出在所有文档中出现的每个特有词汇，并且可以找到包含每个词汇的全部文档。

在索引过程中，Elasticsearch 会存储文档并构建倒排索引，这样用户便可以近实时地对文档数据进行搜索。 索引过程是在索引 API 中启动的，通过此 API 您既可向特定索引中添加 JSON 文档，也可更改特定索引中的 JSON 文档。

Elasticsearch 支持的通用功能特性如下：

分类	特性	说明
分布式集群	集群部署集群监控	监控集群、节点、索引的运行状况
搜索管理	索引配置管理、结构定义、索引重建	搜索管理平台提供配置功能
全文搜索	搜索功能、排序功能、统计分析功能	通过 RESTful API 方式提供
数据采集	ElasticSearch 数据导入 API、Maxcompute 数据导入工具、全量/增量采集方式	丰富的原生数据采集接口，集成 Maxcompute 数据导入工具
服务鉴权	服务级别的用户鉴权机制	统一的用户鉴权设置

在 DCE 5.0 中部署 Elasticsearch 后，还将支持以下特性：

- 支持 Elasticsearch 专有节点、热数据节点、冷数据节点、数据节点角色部署
- 集成 Kibana
- 基于 elasticsearch-exporter 暴露指标
- 基于 Grafana Operator 集成 Elasticsearch Dashboard，展示监控数据
- 使用 ServiceMonitor 对接 Prometheus 抓取指标
- 基于工作空间 Workspace 多租户化管理

集群规格和容量规划

前言

当您准备创建 Elasticsearch 实例时，建议提前评估集群所需的资源。一般情况下需要评估所需磁盘容量、集群规格、存储的文件类型、大小和数量等。基于这些资源评估，预先规划合适的 Elasticsearch 集群配置，可以减少后续使用过程中的问题和风险。

> 重要知识: ES 集群的物理磁盘容量 **不等于** ES 集群可使用数据空间大小

本文基于实际测试结果和用户使用经验，提供了相对通用的评估方法。但业务是瞬息万变的，所以本文的目的是在您创建 Elasticsearch 时给出一个相对合适的建议。实际上 DCE 5 也提供了优化集群规格和容量规划的能力，方便后续对资源进行升级。

事先评估项

要进行 Elasticsearch 的容量规划，需要考虑以下因素：

- 使用场景：主要使用的业务场景，不同场景下产生的数据量不同
- 文档数量：确定要存储的文档数量，以及每个文档的大小
- 存储需求：确定所需的存储空间，以便存储所有文档以及任何相关数据
- 索引需求：选择适当的索引策略和设置以确保性能和可扩展性
- 查询需求：确定查询负载的复杂性和大小，以便为其分配足够的资源
- 集群规模：确定集群的大小和数量，以便支持预期的负载并提供高可用性和容错性

存储容量规划

基础要求规划

在存储容量规划的开始时，需要预先确认两个使用场景和数据要求规划：

- 使用场景
  - 日志场景
  - 搜索场景
  - 数据分析场景
  - 数据库加速场景
  - 通用场景
- 数据要求
  - 源数据大小或者预估文档条数和单个文档的大小
  - 每日数据增量情况
  - 数据保留时长
  - 需要的副本数

ES 其他业务开销

Elasticsearch 中除了数据之外，还有其他的开销，这些也是影响 Elasticsearch 服务存储容量的主要原因，如：

- 索引开销：可以使用 _cat/indices?v API 和 pri.store.size 值计算确切的开销计算，通常比源数据大 10%（_all 参数等未计算）
- 操作系统预留空间：默认操作系统会保留 5%的文件系统供您处理关键流程、系统恢复以及防止磁盘碎片化问题等
- Elasticsearch 内部开销：段合并、日志等内部操作，一般预留 20%
- 副本数量： 副本有利于增加数据的可靠性，但同时会增加存储成本
- 安全阈值： 为了预防突发的数据增长，请大致保留 15% 的容量空间作为安全阈值

所以，为了数据的安全和稳定，我们建议您的磁盘使用率不要超过 85% ；或者在即将达到 85%，应该尽快扩充升级。

快速计算公式

完整公式	简化版本
源数据 * (1 + 副本数量) * (1 + 索引开销) / (1 – Linux 预留空间) / (1 – 内部开销) = 最小存储要求	源数据 * (1 + 副本数量) * 1.45 = 最小存储要求

如果有 500G 数据存储并且需要一个副本，则最低存储要求更接近 500 * 2 * 1.1 / 0.95 / 0.8 = 1.5T。

关于节点磁盘的配置

使用场景不同，单节点最大承载数据量也会不同，具体如下：

- 数据加速、查询聚合等场景：单节点磁盘最大容量 = 单节点内存大小（GB）* 10
- 日志写入、离线分析等场景：单节点磁盘最大容量 = 单节点内存大小（GB）* 50
- 通用场景：单节点磁盘最大容量 = 单节点内存大小（GB）* 30

ES 集群实例配置推荐

在生产环境部署推荐配置：尽量一个节点只承担一个角色。不同节点所需要的计算资源不一样。不同角色分离后，可以按需扩展互不影响。

- 集群最大节点数 = 单节点 CPU * 5
- 单节点磁盘最大容量
  - 搜索类场景：单节点磁盘最大容量 = 单节点内存大小（GB）* 10。
  - 日志类等场景：单节点磁盘最大容量 = 单节点内存大小（GB）* 50。

配置	最大节点数	单节点磁盘最大容量（查询）	单节点磁盘最大容量（日志）
4 核 16G	20	160 GB	800 GB
8 核 32G	40	320 GB	1.5 TB
16 核 64G	80	640 GB	2 TB

分片数量规划

适用场景：

- 日志类，写入频繁，查询较少，单个分片 30G 左右
- 搜索类，写入少，查询频繁，单个分片不超过 20G

每个 Elasticsearch 索引被分为多个分片，数据按哈希算法打散到不同的分片中。由于索引分片的数量影响读写性能和故障恢复速度，建议提前规划。

分片使用概要

- Elasticsearch 在 7.x 版本中，每个索引默认为 1 个主分片和 1 个副本分片
- 在单节点上，7.x 版本最大分片数量为 1000
- 单个分片大小尽量保持在 10–50G 之间为最佳体验，一般推荐在 30G 左右
  - 分片过大可能使 Elasticsearch 的故障恢复速度变慢
  - 分片过小可能导致非常多的分片，因为每个分片会使用占用一些 CPU 和内存，从而导致读写性能和内存不足的问题。
- 当分片数量超过数据节点数量时，建议分片数量接近数据节点的整数倍，便于将分片均匀的分布到数据节点中。
- 对日志场景，建议启用 ILM 功能。在发现分片大小不合理时，通过该功能及时调整分片数量。

索引分片资源占用

每个索引和每个分片都需要一些内存和 CPU 资源。在大多数情况下，一小组大分片比许多小分片使用更少的资源。

段在分片的资源使用中起着重要作用。大多数分片包含几个段，用于存储其索引数据。 Elasticsearch 将段元数据保存在 JVM 堆内存中，以便可以快速检索它以进行搜索。 随着分片的增长，它的段被合并成更少、更大的段。这减少了段的数量，这意味着更少的元数据保存在堆内存中。

为了减少索引数量并避免造成过大且无序的映射，可以考虑在同一索引中存储类似结构的数据，而不要基于数据来源将数据分到不同的索引中。 很重要的一点是在索引/分片的数量和每个单独索引的映射大小之间实现良好平衡。 由于集群状态会加载到每个节点（包括主节点）上的堆内存中，而且堆内存大小与索引数量以及单个索引和分片中的字段数成正比关系，所以还需要同时监测主节点上的堆内存使用量并确保其大小适宜，这一点很重要。

分片过小会导致段过小，进而致使开销增加。您要尽量将分片的平均大小控制在至少几 GB 到几十 GB 之间。 对时序型数据用例而言，分片大小通常介于 20GB 至 40GB 之间。

由于单个分片的开销取决于段数量和段大小，所以通过 forcemerge 操作强制将较小的段合并为较大的段能够减少开销并改善查询性能。 理想状况下，应当在索引内再无数据写入时完成此操作。请注意：这是一个极其耗费资源的操作，所以应该在非高峰时段进行。

每个节点上可以存储的分片数量与可用的堆内存大小成正比关系，但是 Elasticsearch 并未强制规定固定限值。 这里有一个很好的经验法则：确保对于节点上已配置的每个 GB，将分片数量保持在 20 以下。 如果某个节点拥有 30GB 的堆内存，那其最多可有 600 个分片，但是在此限值范围内，您设置的分片数量越少，效果就越好。 一般而言，这可以帮助集群保持良好的运行状态。

更多信息，请参考：

- 减少集群分片数
- 我在 Elasticsearch 集群内应该设置多少个分片?

分片计算公式

(元数据 + 增长空间) * (1 + 索引开销) / 所需的分片大小 = 主分片的大约数量

假设有 80GiB 的数据。希望将每个分片保持在 30GiB 左右。因此，您的分片数量应大约为 80 * 1.1 / 30 = 3

如何管理分片

进入到 Kibana 的索引管理界面，找到 Stack Managment

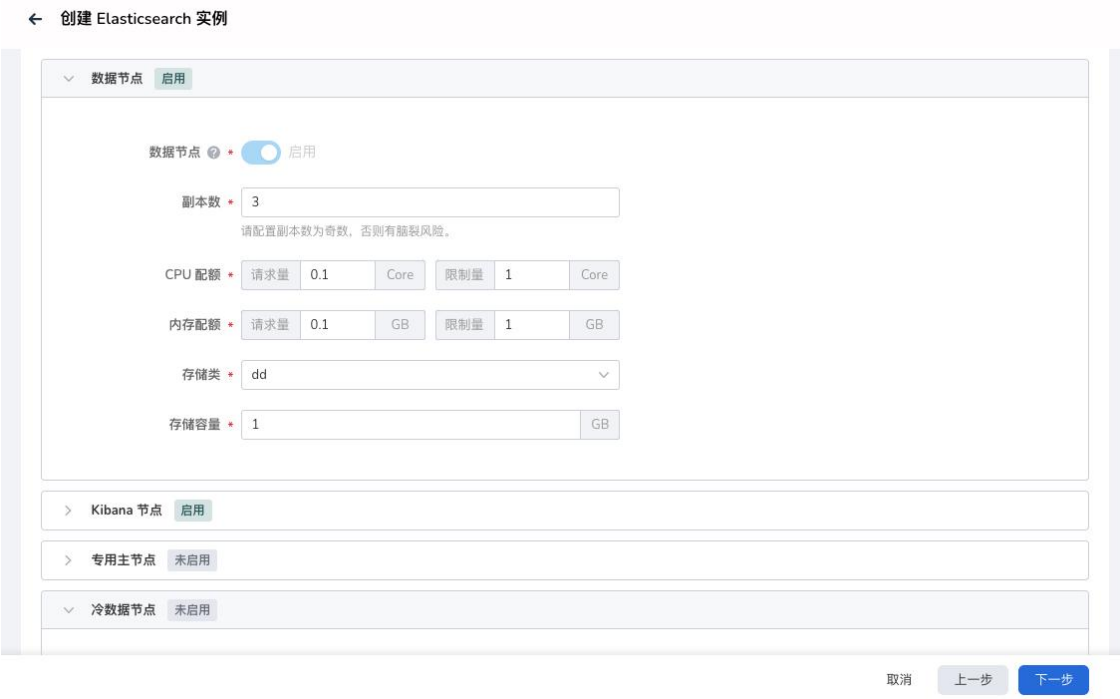
使用索引生命周期管理（ILM）自动管理索引，管理策略如下：

- 根据索引大小，自动 rollover
- 根据索引创建时间，自动 rollover
- 根据文档数量，自动 rollover

索引生命周期执行策略，默认每 10 分钟执行一次，可以通过修改 indices.lifecycle.poll_interval 参数来控制检查频率。

数据服务内创建 Elasticsearch

中间件 elasticsearch 创建的页面



创建实例

节点介绍： – 数据节点： Data Node 是 Elasticsearch 用于存储数据的节点，负责存储数据，用于存储大量数据，所以对存储的压力非常高 – Kibana 节点： 启用对应 Kibana 控制台， 为 Elasticsearch 提供了界面化的管理窗口 – 专用主节点： 专用节点（Dedicated Node）是一种配置选项，该选项指定节点只用于执行特定任务。与数据节点（Data Node）不同，专用节点不存储任何数据，而是专门用于执行集群管理或搜索操作

如上图中的，存储容量配置，请根据上述的容量计算公式，规划对应的存储容量。是否启用专用主节点，主要考虑是否需要专用主节点来支持增强集群的稳定性。

结语

如果你有关于 Elasticsearch 如何进行容量规划的需求，可在页面底部进行讨论。

基于 Hwameistor 的 Elasticsearch 迁移实践

由于 Kubernetes 自身的特性，有状态应用部署完成后是否可以迁移取决于底层 CSI 的能力。然而当集群出现资源不均等意外情况时，需要跨节点迁移相关的有状态应用。

本文以 Elasticsearch 为例，参考 Hwameistor 官方提供的迁移指南，演示使用 Hwameistor 时如何跨节点迁移数据服务中间件。

演示环境

从集群信息、ES 安装信息、PVC 三方面进行介绍演示环境：

===“集群信息”

```bash
[root@prod-master1 ~]# kubectl get node
NAME                STATUS     ROLES                AGE     VERSION
prod-master1     Ready      control-plane     15h     v1.25.4
prod-master2     Ready      control-plane     15h     v1.25.4
prod-master3     Ready      control-plane     15h     v1.25.4
prod-worker1     Ready      <none>               15h     v1.25.4
prod-worker2     Ready      <none>               15h     v1.25.4
prod-worker3     Ready      <none>               15h     v1.25.4
```

===“ES 安装信息”

```bash
[root@prod-master1 ~]# kubectl get pods -o wide | grep es-cluster-masters-es-data

mcamel-common-es-cluster-masters-es-data-0 Running prod-worker1
```

mcamel-common-es-cluster-masters-es-data-1 Running prod-worker3
mcamel-common-es-cluster-masters-es-data-2 Running prod-worker2
```

===“ES 使用的 PVC”

```bash
kubectl -n mcamel-system get pvc -l elasticsearch.k8s.elastic.co/statefulset-name=mcamel-common-es-cluster-masters-es-data
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-0 Bound pvc-61776435-0df5-448f-abb9-4d06774ec0e8 35Gi RWO hwameistor-storage-lvm-hdd 15h
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1 Bound pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 35Gi RWO hwameistor-storage-lvm-hdd 15h
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-2 Bound pvc-955bd221-3e83-4bb5-b842-c11584bced10 35Gi RWO hwameistor-storage-lvm-hdd 15h
```

演示目标

将 prod-worker3 节点上的 mcamel-common-es-cluster-masters-es-data-1（以下简称演示应用/esdata-1）有状态应用跨节点迁移到 prod-master3 节点。

准备工作

确定需要迁移的 PV

使用如下命令查找演示应用 esdata-1 对应的 PV 磁盘，明确需要迁移哪个 PV。

1. 查看演示应用绑定的 PVC
```

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pod mcamel-common-es-cluster-masters-es-data-1 -ojson | jq .spec.volumes[0]
{
  "name": "elasticsearch-data",
  "persistentVolumeClaim": {
    "claimName": "elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1"
  }
}
```

2. 查看该 PVC 绑定的 PV

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1
NAME                                                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS                AGE
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1   Bound    pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c   35Gi       RWO            hwameistor-storage-lvm-hdd   17h
```

3. 确认该 PV 绑定的应用是否为需要迁移的应用，即此文中的演示应用 esdata-1

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pv pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 35Gi RWO Delete Bound mcamel-system/elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1 hwameistor-storage-lvm-hdd 17h ```
```

上述信息证明，需要迁移的 PV 为 pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c。

停止运行待迁移应用

1. 查看当前正在运行的应用

```
[root@prod-master1 ~]# kubectl -n mcamel-system get sts
NAME                                          READY   AGE
elastic-operator                              2/2     20h
mcamel-common-es-cluster-masters-es-data      3/3     20h
mcamel-common-kpanda-mysql-cluster-mysql      2/2     20h
mcamel-common-minio-cluster-pool-0            1/1     20h
mcamel-common-mysql-cluster-mysql             2/2     20h
mysql-operator                                1/1     20h
rfr-mcamel-common-redis-cluster               3/3     20h
```

2. 停止运行 ES operator

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=0 sts elastic-operator
```

3. 停止运行 ES：

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=0 sts mcamel-common-es-cluster-masters-es-data
# --- wait about 3 mins ----
```

4. 确认 ES 已经停止运行

```
[root@prod-master1 ~]# kubectl -n mcamel-system get sts
NAME                                          READY   AGE
elastic-operator                              0/0     20h
mcamel-common-es-cluster-masters-es-data      0/0     20h
mcamel-common-kpanda-mysql-cluster-mysql      2/2     20h
mcamel-common-minio-cluster-pool-0            1/1     20h
```

mcamel-common-mysql-cluster-mysql 2/2 20h
mysql-operator 1/1 20h
rfr-mcamel-common-redis-cluster 3/3 20h

视频演示如下：



开始迁移

有关此过程的详细说明，可参考 Hwameistor 官方文档：迁移数据卷

1. 建立迁移任务

```
[root@prod-master1 ~]# cat migrate.yaml
apiVersion: hwameistor.io/v1alpha1
kind: LocalVolumeMigrate
metadata:
```

```
  namespace: hwameistor
  name: migrate-es-pvc # 任务名称
spec:
  sourceNode: prod-worker3 # 来源 node，可以通过 `kubectl get ldn` 获取
  targetNodesSuggested:
  - prod-master3
  volumeName: pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c # 需要迁移的 pvc
  migrateAllVols: false
```

2. 执行迁移命令

```
[root@prod-master1 ~]# kubectl apply -f migrate.yaml
```

此时会在 hwameistor 命名空间创建一个 pod，用于执行迁移动作。

3. 查看迁移状态

```
[root@prod-master1 ~]# kubectl get localvolumemigrates.hwameistor.io migrate-es-pvc -o yaml
apiVersion: hwameistor.io/v1alpha1
kind: LocalVolumeMigrate
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"hwameistor.io/v1alpha1","kind":"LocalVolumeMigrate","metadata":{"annotations":{},"name":"migrate-es-pvc"},"spec":{"migrateAllVols":false,"sourceNode":"prod-worker3","targetNodesSuggested":["prod-master3"],"volumeName":"pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c"}}
  creationTimestamp: "2023-04-30T12:24:17Z"
  generation: 1
  name: migrate-es-pvc
  resourceVersion: "1141529"
```

```
  uid: db3c0df0-57b5-42ef-9ec7-d8e6de487767
spec:
  abort: false
  migrateAllVols: false
  sourceNode: prod-worker3
  targetNodesSuggested:
  - prod-master3
  volumeName: pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c
status:
  message: 'waiting for the sync job to complete: migrate-es-pvc-datacopy-elasticsearch-data-mcamel'
  originalReplicaNumber: 1
  state: SyncReplica
  targetNode: prod-master3
```

4. 迁移完成后，查看迁移结果

```
[root@prod-master1 ~]# kubectl get lvr
NAME CAPACITY NODE STATE SYNCED DEVICE AGE
pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 37580963840 prod-master3 Ready true /dev/LocalStorage_PoolHDD/pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 129s
```

恢复 common-es

1. 启动 ES operator

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=2 sts elastic-operator
```

2. 启动 ES

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=3 sts mcamel-common-es-cluster-masters-es-data
```

相关问题

HwameiStor 使用 rclone 来迁移 PV，而 rclone 在迁移过程中可能会丢失权限（参考 rclone#1202 和 hwameistor#830）。如果权限丢失，ES 会启动失败并反复启动，陷入恶性循环。

遇到类似问题时可以通过下述步骤排查并解决故障。

确认问题

使用以下命令查看 Pod 日志：

``bash kubectl –n mcamel–system logs mcamel–common–es–cluster–masters–es–data–0 –c elasticsearch

如果日志中包含如下错误信息，则可以确认为权限丢失造成的问题。

```log
java.lang.IllegalStateException: failed to obtain node locks, tried [[/usr/share/elasticsearch/data]] with lock id [0]; maybe these locations are not writable or multiple nodes were started without increasing [node.max_local_storage_nodes] (was [1])?
```

解决故障

1. 运行下命令修改 ES 的 CR

   ```
   kubectl –n mcamel–system edit elasticsearches.elasticsearch.k8s.elastic.co mcamel–common–es–cluster–masters
   ```

2. 为 ES 的 Pod 添加一个 initcontainer，内容如下：

   ```
   – command:
     – sh
     – –c
     – chown –R elasticsearch:elasticsearch /usr/share/elasticsearch/data
   ```

```
      name: change-permission
      resources: {}
      securityContext:
        privileged: true
```

initcontainer 在 CR 中的位置如下：

```
spec:
  ...
  ...
  nodeSets:
  - config:
      node.store.allow_mmap: false
    count: 3
    name: data
    podTemplate:
      metadata: {}
      spec:
        ...
        ...
        initContainers:
        - command:
          - sh
          - -c
          - sysctl -w vm.max_map_count=262144
          name: sysctl
          resources: {}
          securityContext:
            privileged: true
        - command:
          - sh
          - -c
          - chown -R elasticsearch:elasticsearch /usr/share/elasticsearch
```

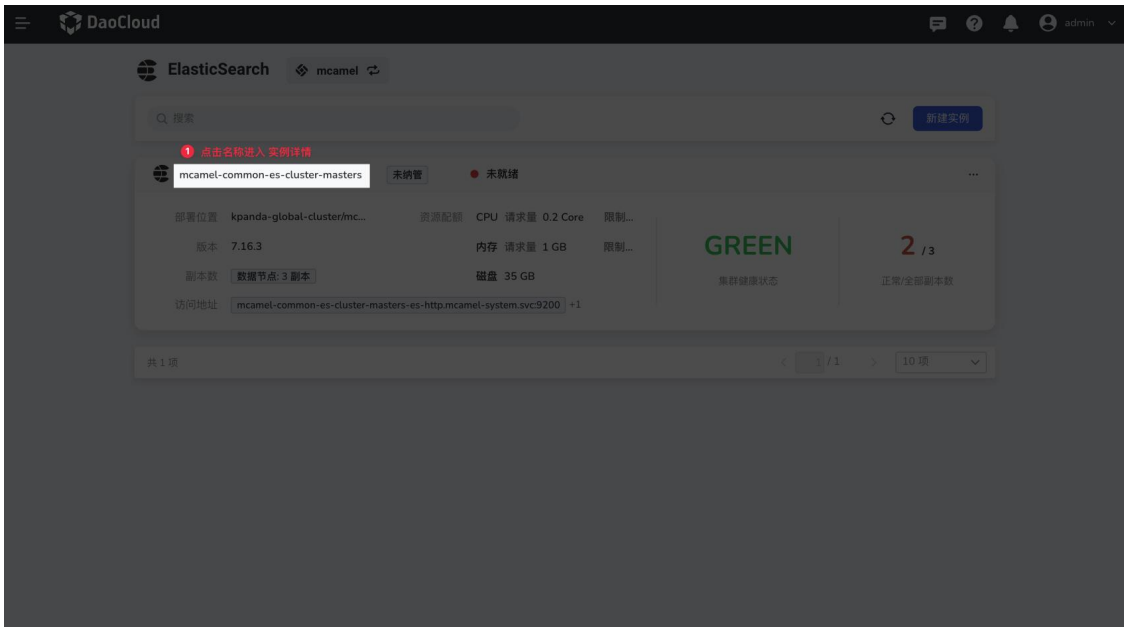
```
/data
        name: change-permission
        resources: {}
        securityContext:
          privileged: true
```

查看 Elasticsearch 日志

操作步骤

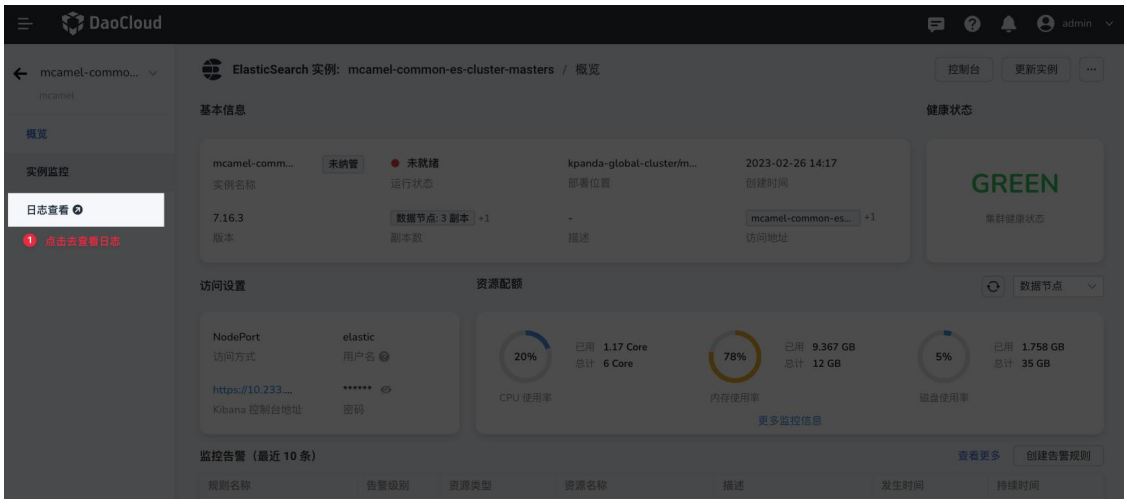
通过访问每个 Elasticsearch 的实例详情，页面；可以支持查看 Elasticsearch 的日志。

1. 在 Elasticsearch 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。



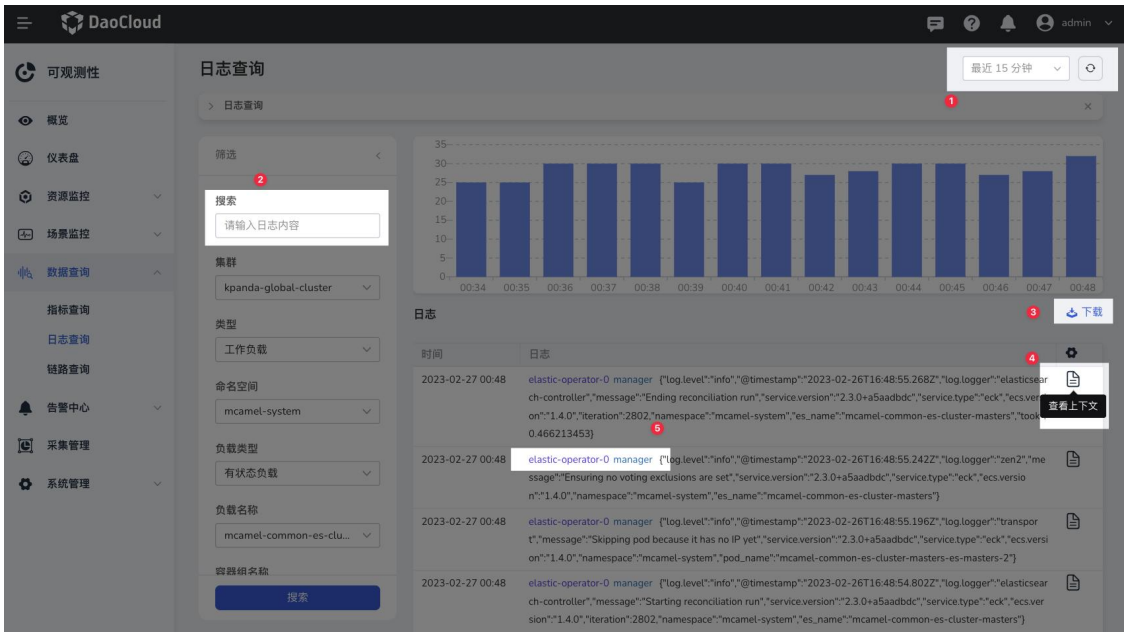
image

3. 点击 日志查看 即可进入到日志查看页面（Insight 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 上下文 图标即可
- 支持 导出日志

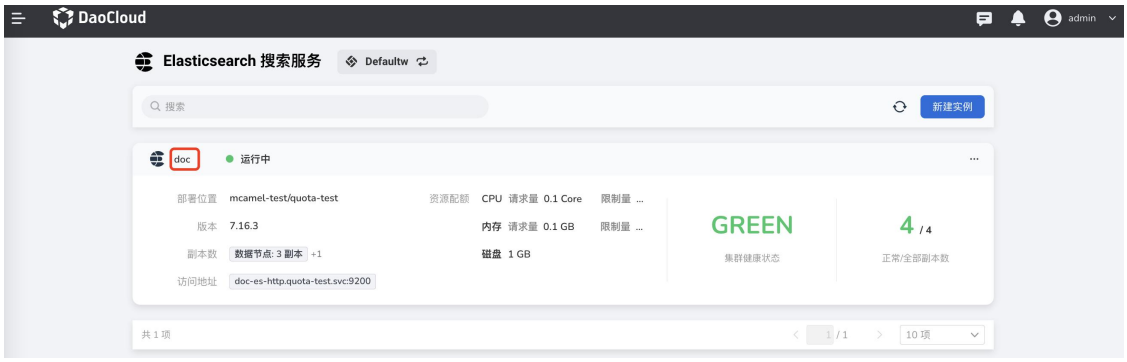

DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统管理
日志查询
日志查询
最近 15 分钟
筛选
搜索
请输入日志内容
集群
kpanda-global-cluster
类型
工作负载
命名空间
mcamel-system
负载类型
有状态负载
负载名称
mcamel-common-es-clu...
搜索
35
30
25
20
15
10
5
0
00:34
00:35
00:36
00:37
00:38
00:39
00:40
00:41
00:42
00:43
00:44
00:45
00:46
00:47
00:48
日志
下载
时间
日志
2023-02-27 00:48
elastic-operator-0 manager
查看上下文
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager
2023-02-27 00:48
elastic-operator-0 manager

image

实例概览

本文介绍 Elasticsearch 实例的基本信息。

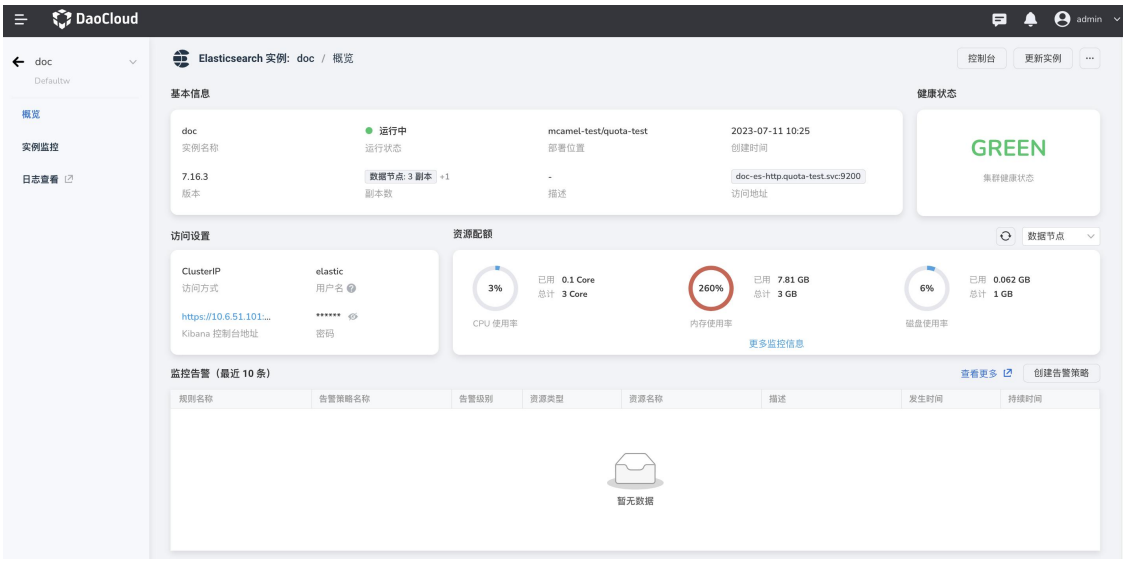
1. 在 Elasticsearch 搜索服务的首页点击实例名称

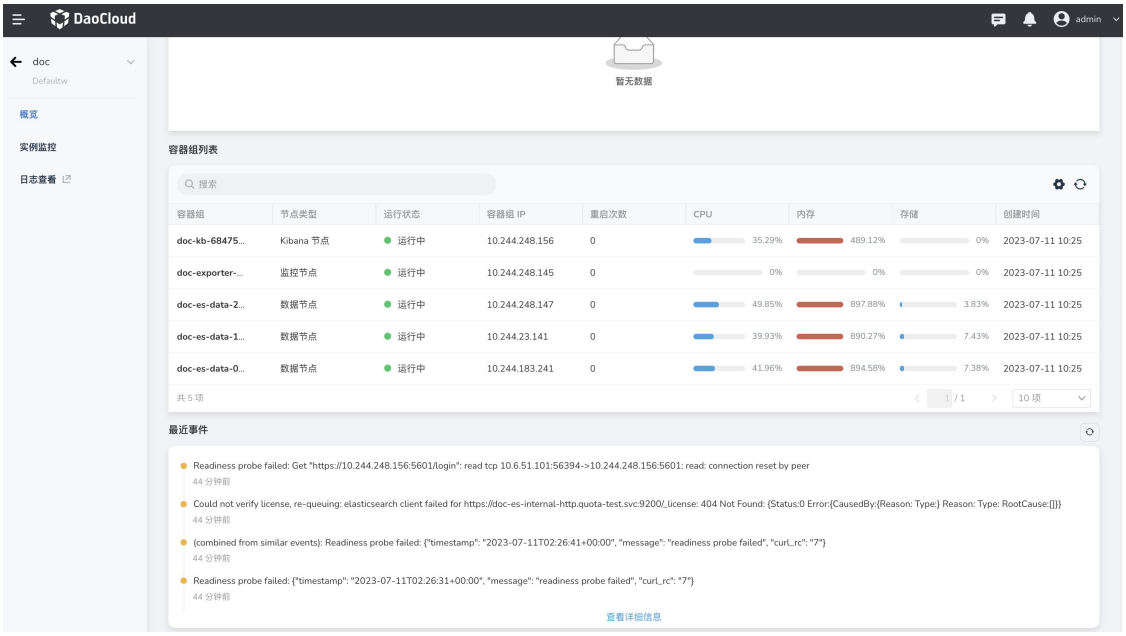


概览

2. 查看实例的概览信息，具体包括：

   - 基本信息：实例名称、运行状态、部署位置、创建时间、版本、副本数、访问地址（Elasticsearch 集群内部的访问地址）等
   - 健康状态：分为
   - 访问设置：访问方式、用户名和密码、Kibana 控制台地址等
   - 资源配额：CPU 和内存使用率
   - 监控告警：当前实例的告警信息，可以转到 DCE 5.0 可观测性模块查看告警详情或创建告警策略
   - 容器组列表：当前实例的容器组信息，包括容器组名称、节点类型、运行状态、IP、CPU、内存、存储信息、创建时间等
   - 最近事件：当前实例的事件信息


DaoCloud
admin
doc
Defaultw
概览
实例监控
日志查看
Elasticsearch 实例: doc / 概览
控制台
更新实例
基本信息
doc
实例名称
运行中
运行状态
mcamel-test/quota-test
部署位置
2023-07-11 10:25
创建时间
7.16.3
版本
数据节点: 3 副本 +1
副本数
-
描述
doc-es-http.quota-test.svc:9200
访问地址
健康状态
GREEN
集群健康状态
访问设置
ClusterIP
访问方式
elastic
用户名
https://10.6.51.101:...
Kibana 控制台地址
******
密码
资源配额
数据节点
3%
CPU 使用率
已用 0.1 Core
总计 3 Core
260%
内存使用率
已用 7.81 GB
总计 3 GB
6%
磁盘使用率
已用 0.062 GB
总计 1 GB
更多监控信息
监控告警（最近 10 条）
查看更多
创建告警策略
规则名称
告警策略名称
告警级别
资源类型
资源名称
描述
发生时间
持续时间
暂无数据


DaoCloud
admin
doc
Defaultw
概览
实例监控
日志查看
暂无数据
容器组列表
搜索
容器组
节点类型
运行状态
容器组 IP
重启次数
CPU
内存
存储
创建时间
doc-kb-68475...
Kibana 节点
运行中
10.244.248.156
0
35.29%
489.12%
0%
2023-07-11 10:25
doc-exporter-...
监控节点
运行中
10.244.248.145
0
0%
0%
0%
2023-07-11 10:25
doc-es-data-2...
数据节点
运行中
10.244.248.147
0
49.85%
897.88%
3.83%
2023-07-11 10:25
doc-es-data-1...
数据节点
运行中
10.244.23.141
0
39.93%
890.27%
7.43%
2023-07-11 10:25
doc-es-data-0...
数据节点
运行中
10.244.183.241
0
41.96%
894.58%
7.38%
2023-07-11 10:25
共 5 项
1 / 1
10 项
最近事件
Readiness probe failed: Get "https://10.244.248.156:5601/login": read tcp 10.6.51.101:56394->10.244.248.156:5601: read: connection reset by peer
44 分钟前
Could not verify license, re-queuing: elasticsearch client failed for https://doc-es-internal-http.quota-test.svc:9200/_license: 404 Not Found: {Status:0 Error:{CausedBy:{Reason: Type:} Reason: Type: RootCause:[]}}
44 分钟前
(combined from similar events): Readiness probe failed: {"timestamp": "2023-07-11T02:26:41+00:00", "message": "readiness probe failed", "curl_rc": "7"}
44 分钟前
Readiness probe failed: {"timestamp": "2023-07-11T02:26:31+00:00", "message": "readiness probe failed", "curl_rc": "7"}
44 分钟前
查看详细信息

实例监控

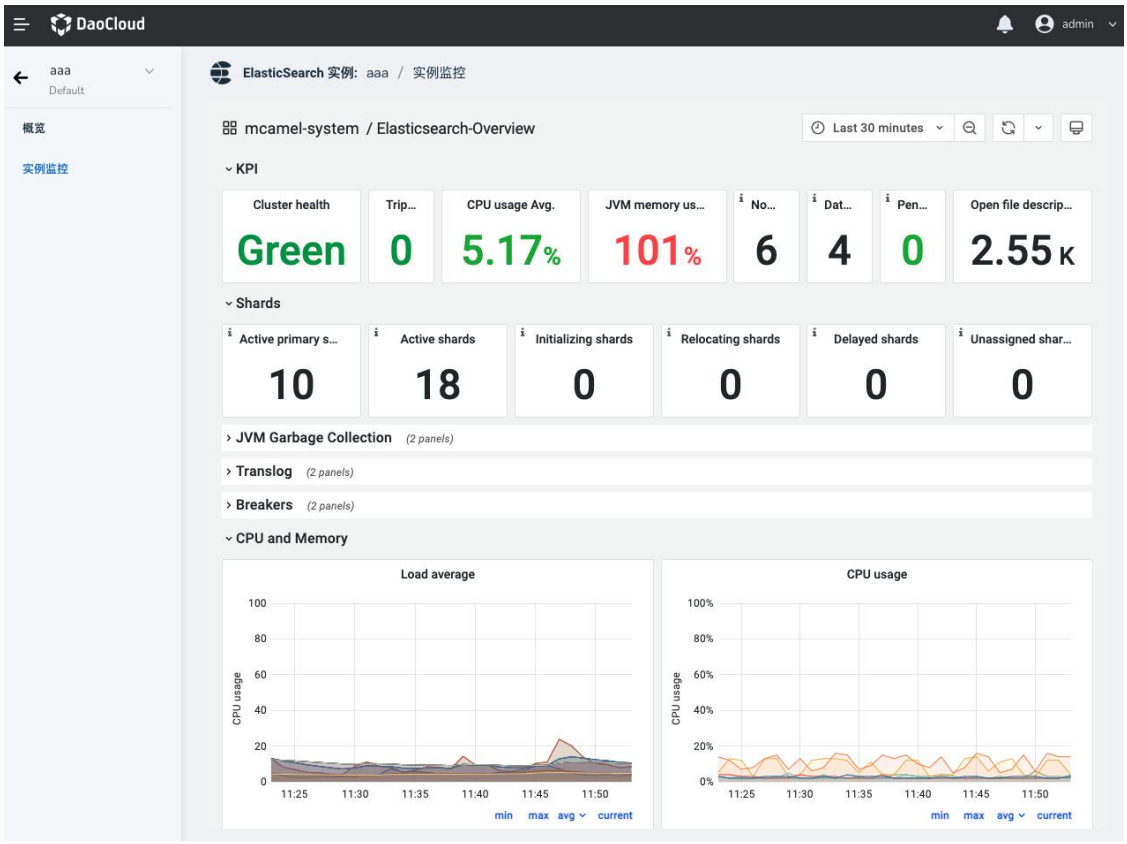
本页说明如何从图表面板快速获取 Elasticsearch 有关的信息以及一些注意点和小提示。

1. 在 Elasticsearch 实例列表中点击一个实例名称。



实例监控

2. 在左侧导航栏点击实例监控，进入实例监控大屏。


DaoCloud
admin
aaa
Default
概览
实例监控
ElasticSearch 实例: aaa / 实例监控
mcamel-system / Elasticsearch-Overview
Last 30 minutes
KPI
Cluster health
Green
Trip...
0
CPU usage Avg.
5.17%
JVM memory us...
101%
No...
6
Dat...
4
Pen...
0
Open file descrip...
2.55K
Shards
Active primary s...
10
Active shards
18
Initializing shards
0
Relocating shards
0
Delayed shards
0
Unassigned shar...
0
JVM Garbage Collection (2 panels)
Translog (2 panels)
Breakers (2 panels)
CPU and Memory
Load average
CPU usage
100
80
60
40
20
0
100%
80%
60%
40%
20%
0%
11:25
11:30
11:35
11:40
11:45
11:50
min
max
avg
current

实例监控

创建 Elasticsearch 实例

在 Elasticsearch 实例列表中，执行以下操作创建一个新的实例。

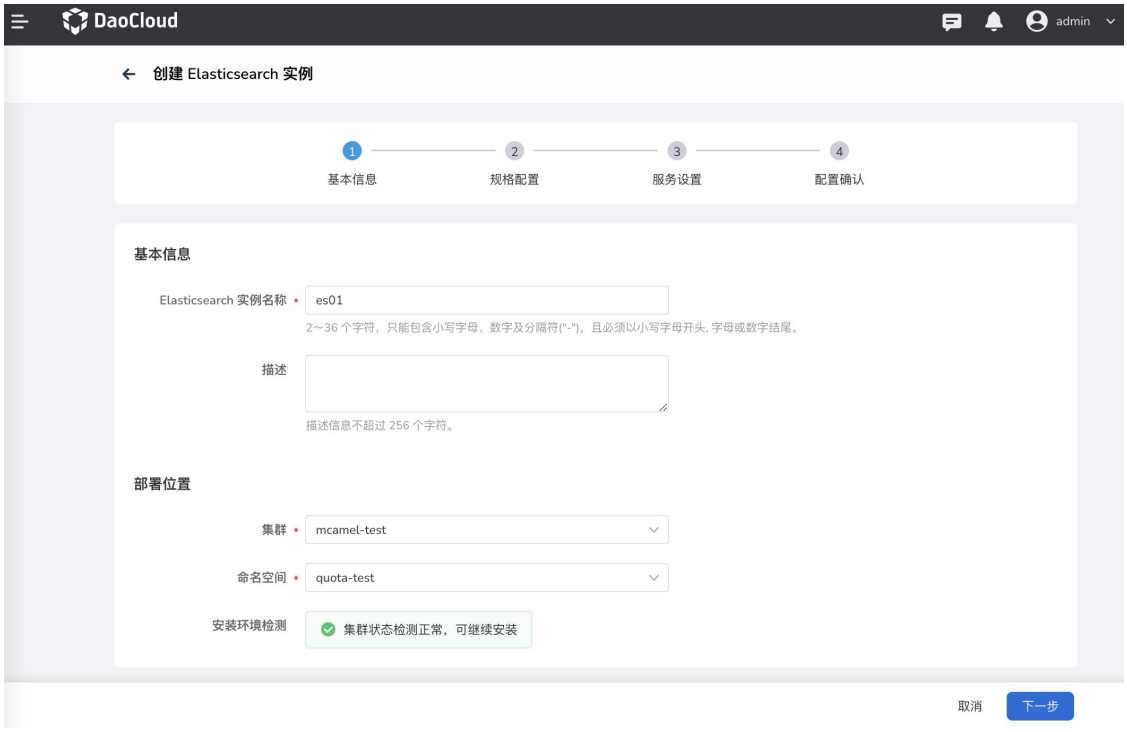
1. 在 Elasticsearch 搜索服务的首页右上角点击新建实例。



新建实例

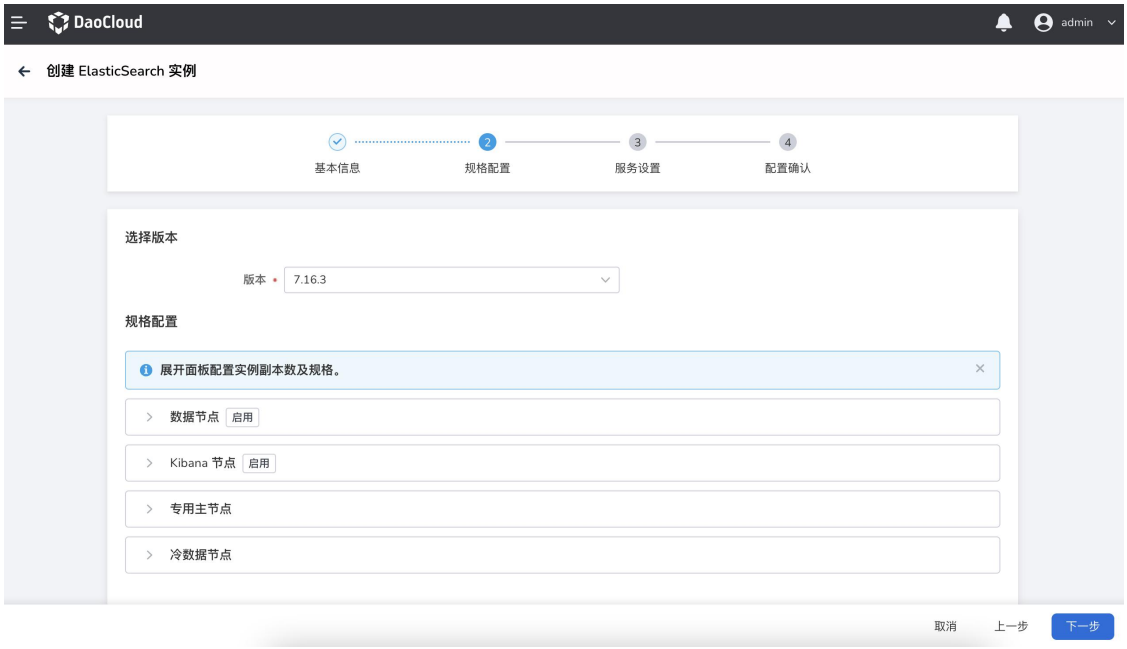
2. 填写实例的基本信息。通过安装环境检测后，点击下一步。

   如未通过安装环境检测，页面会给出失败原因和操作建议。常见原因是缺少相关组件，根据页面提示安装对应的组件即可。



基本信息

3. 选择版本：选择基于哪个 Elasticsearch 版本创建实例，目前仅支持 7.16.3



规格配置

4. 参考以下信息填写实例的规格配置。

===“数据节点（默认启用）”

– 用于存储数据，执行增删改查、搜索、聚合等数据相关操作。数据节点对资源要求较高，需要配置充足的资源。
– **如果未启用`专用主节点`，由`数据节点`充当`专用主节点**。
– 最少 1 副本，最多 50 个副本，默认 3 副本。
– 副本数建议为基数，否则存在脑裂风险。

![热数据节点](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-1.png)

===“Kibana 节点（默认启用）”

– Kibana 是适用于 Elasticsearch 的可视化分析平台，可搜索、查看存储在索引中的数据并与其进行交互。
– 默认启用 `Kibana 节点`，用于存放 Elasticsearch 可视化数据的节点。
– 默认为 1 副本，不可修改。

![Kibana 节点](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-2.png)

===“专用主节点”

– 主节点负责集群范围内的轻量化操作，例如创建/删除索引、监听其他类型节点的状态、决定如何分配数据分片等。
– **如果不启用`专用主节点`，则由数据节点充当主节点**。这样可能会存在数据节点和主节点竞争资源的情况，影响系统稳定性。
– 启用`专用主节点`后，主节点与`数据节点`分离，有利于保障服务的稳定性。
– 默认 3 个副本，不可修改。

![专用主节点](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-3.png)

===“冷数据节点”

– 存储历史数据等查询频率低且基本无需写入的数据。
– 最少 2 副本，最多 50 副本，默认 3 副本。
– 如果业务中同时存在“查询频率高/写入压力大“和”查询频率/低基本无写入“的数据，建议启用`冷数据节点`，实现冷热数据分离。
– 启用`冷数据节点`后，会自动开启`专用主节点`。

![冷数据节点](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-4.png)

5. 参考以下说明填写服务设置

   - 访问类型：Elasticsearch 实例对应的 Service 的类型。有关各种类型的详细说明，可参考服务类型

   - 访问设置：访问 Elasticsearch 实例的用户名和密码，以及 Kibana 的访问类型。

     服务设置

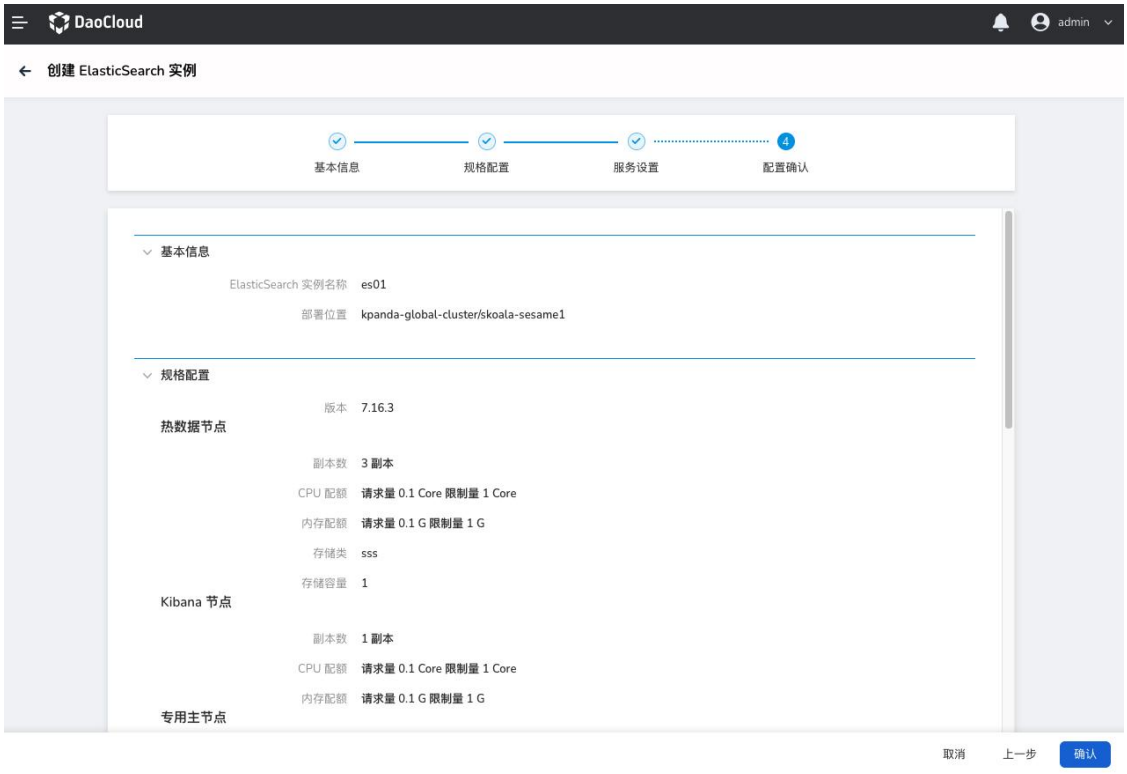
   - 节点亲和性：启用后，只能/尽量将 Elasticsearch 实例调度到带有特定标签的节点上。

   - 工作负载亲和性：在拓扑域（反亲和性的作用范围）内，根据反亲和性将工作负载下的 Pod 分发到多个节点中，避免多个 Pod 被集中调度到某一个节点，造成节点过载。相关的视频教程可参考工作负载反亲和性

   - 监控采集时间间隔：实例监控的数据采集时间间隔。如果不设置，将采用全局设置。默认为 30s。

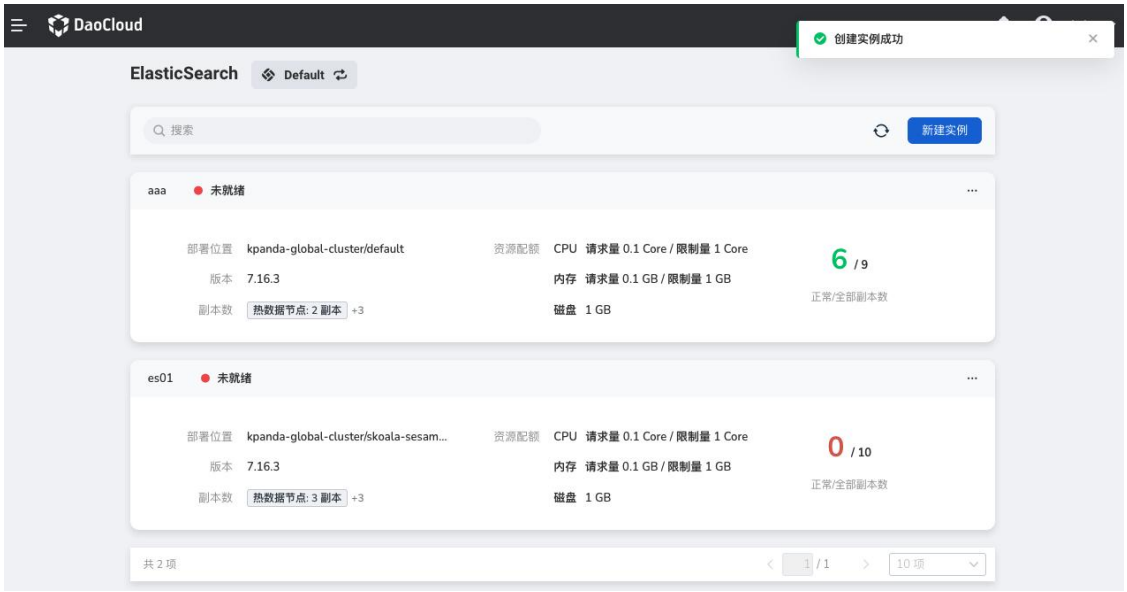
     服务设置

6. 检查所填信息，确认无误后点击确认。如需修改可点击上一步返回修改配置。



配置确认

实例创建成功后，页面会自动跳转到 Elasticsearch 实例列表，可查看所有实例的基本信息和状态。


DaoCloud
创建实例成功
ElasticSearch
Default
搜索
新建实例
aaa
未就绪
部署位置 kpanda-global-cluster/default
版本 7.16.3
副本数 热数据节点: 2 副本 +3
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.1 GB / 限制量 1 GB
磁盘 1 GB
6 / 9
正常/全部副本数
es01
未就绪
部署位置 kpanda-global-cluster/skoala-sesam...
版本 7.16.3
副本数 热数据节点: 3 副本 +3
资源配额 CPU 请求量 0.1 Core / 限制量 1 Core
内存 请求量 0.1 GB / 限制量 1 GB
磁盘 1 GB
0 / 10
正常/全部副本数
共 2 项
1 / 1
10 项

创建成功

删除 Elasticsearch 实例

如果想要删除一个 Elasticsearch 实例，可以执行如下操作：

!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

1. 在 Elasticsearch 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择删除实例。



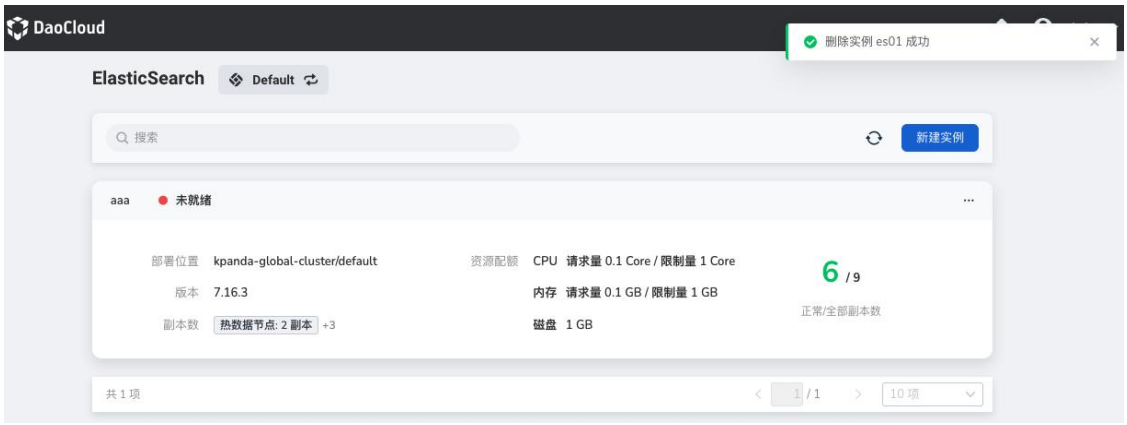
删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。



删除实例

3. 自动返回实例列表，屏幕提示：删除实例 xxx 成功。

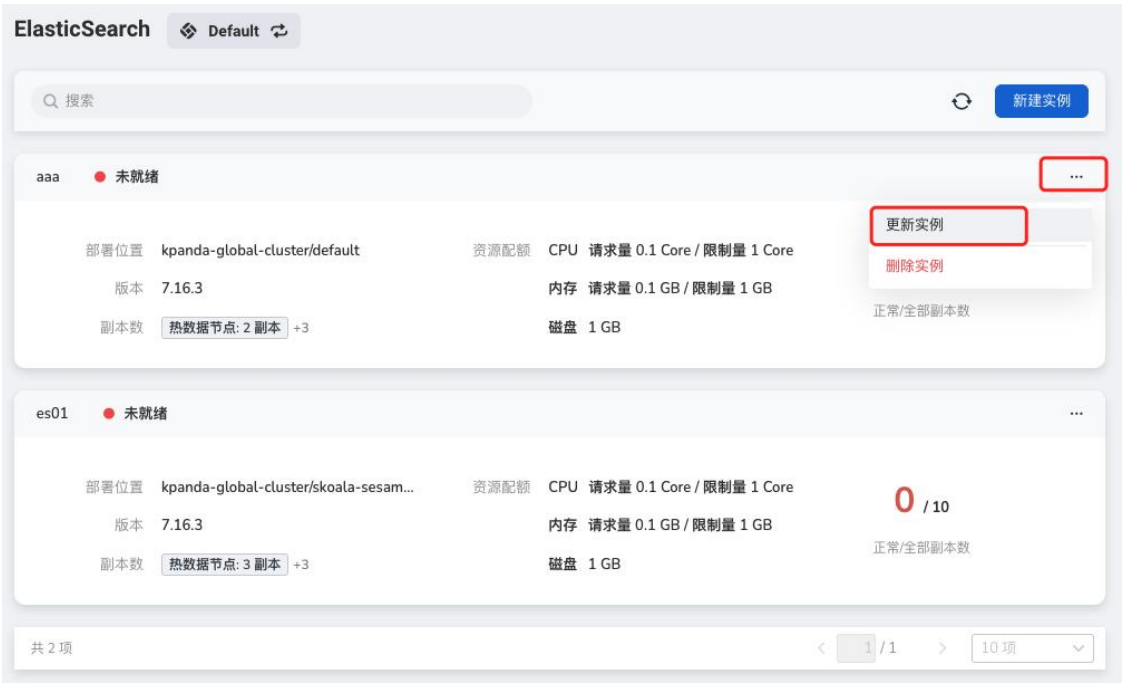


删除实例

更新 Elasticsearch 实例

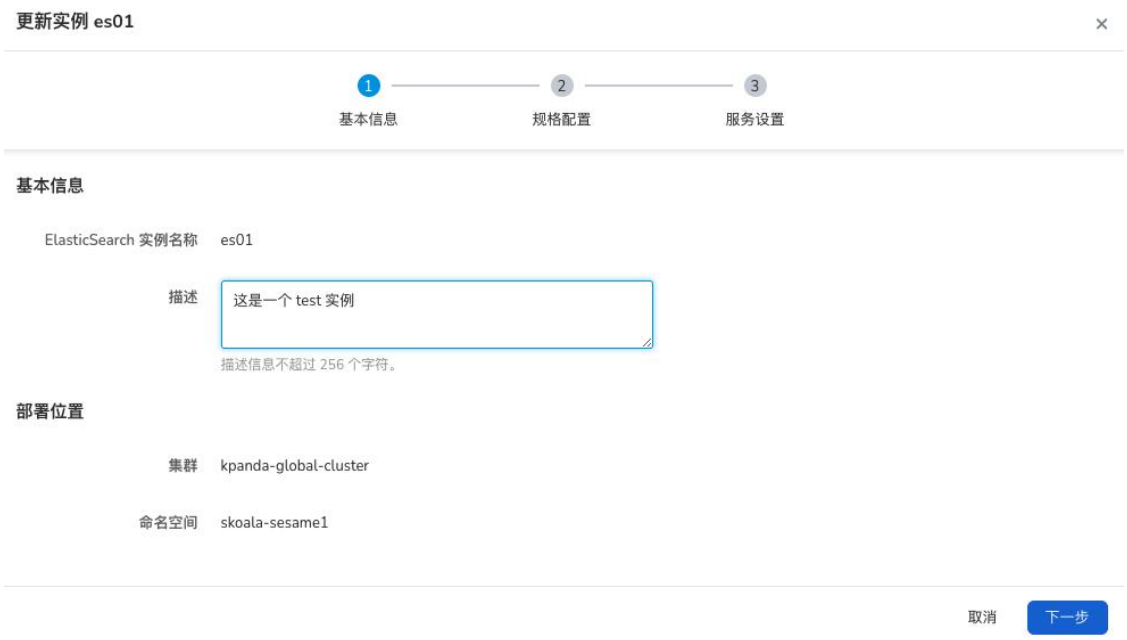
如果想要更新或修改 Elasticsearch 实例的资源配置，可以按照本页说明操作。

1. 在 Elasticsearch 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择更新实例。



更新实例

2. 修改基本信息后，点击下一步。此处暂时只支持修改描述信息。



更新实例

3. 修改规格配置（包括热数据节点、Kibana 节点、专用主节点和冷数据节点）后，点击下一步。



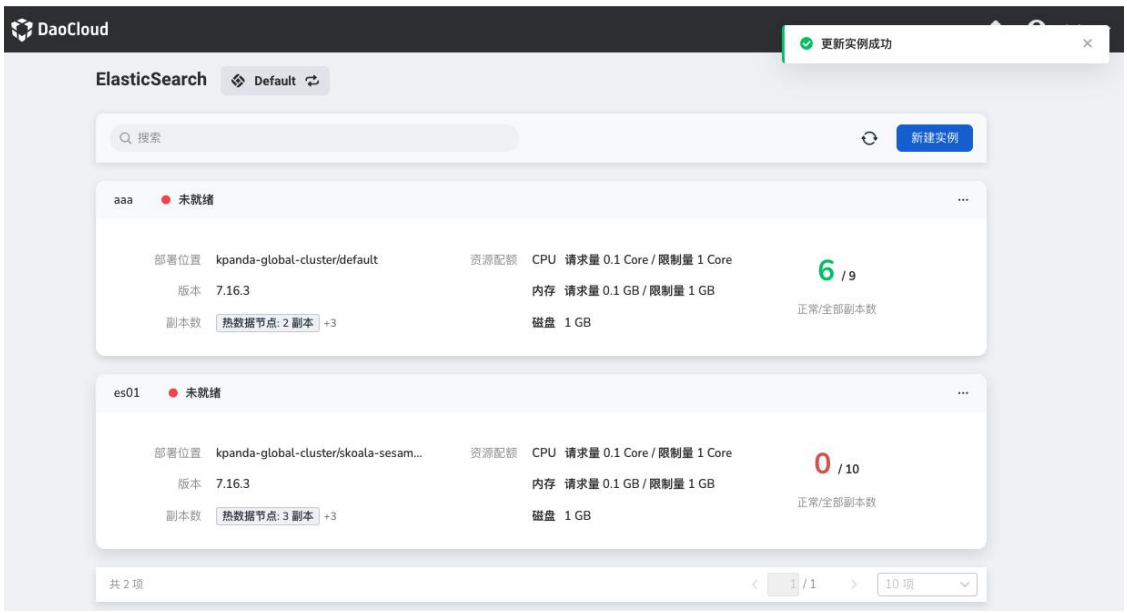
更新实例

4. 修改服务设置后，点击确认。



更新实例

5. 返回消息队列，屏幕右上角将显示消息：更新实例成功。



更新实例

Elasticsearch 排障手册

本文将持续统计和梳理常见的 Elasticsearch 异常故障以及修复方式。若遇到使用问题，请优先查看此排障手册。

> 如果您发现遇到的问题未包含在本手册，可以快速跳转到页面底部，提交您的问题。

Elasticsearch PVC 磁盘容量满

存储依赖 hwameistor

报错信息

{"type": "server", "timestamp": "2022-12-18T10:47:08,573Z", "level": "ERROR", "component": "o.e.m.f.FsHealthService", "cluster.name": "mcamel-common-es-cluster-masters", "node.name": "mcamel-common-es-cluster-masters-es-masters-0", "message": "health check of [/usr/share/elasticsearch/data/nodes/0] failed", "cluster.uuid": "afIglgTVTXmYO2qPFNvsuA", "node.id": "nZRiBCUZQymQVV1son34pA" ,
"stacktrace": ["java.io.IOException: No space left on device",
"at sun.nio.ch.FileDispatcherImpl.write0(Native Method) ~[?:?]",
"at sun.nio.ch.FileDispatcherImpl.write(FileDispatcherImpl.java:62) ~[?:?]",
"at sun.nio.ch.IOUtil.writeFromNativeBuffer(IOUtil.java:132) ~[?:?]",
"at sun.nio.ch.IOUtil.write(IOUtil.java:97) ~[?:?]",
"at sun.nio.ch.IOUtil.write(IOUtil.java:67) ~[?:?]",
"at sun.nio.ch.FileChannelImpl.write(FileChannelImpl.java:285) ~[?:?]",
"at java.nio.channels.Channels.writeFullyImpl(Channels.java:74) ~[?:?]",
"at java.nio.channels.Channels.writeFully(Channels.java:96) ~[?:?]",
"at java.nio.channels.Channels$1.write(Channels.java:171) ~[?:?]",
"at java.io.OutputStream.write(OutputStream.java:127) ~[?:?]",
"at org.elasticsearch.monitor.fs.FsHealthService$FsHealthMonitor.monitorFSHealth(FsHealthService.java:170) [elasticsearch-7.16.3.jar:7.16.3]",

"at org.elasticsearch.monitor.fs.FsHealthService$FsHealthMonitor.run(FsHealthService.java:144) [elasticsearch-7.16.3.jar:7.16.3]",
"at org.elasticsearch.threadpool.Scheduler$ReschedulingRunnable.doRun(Scheduler.java:214) [elasticsearch-7.16.3.jar:7.16.3]",
"at org.elasticsearch.common.util.concurrent.ThreadContext$ContextPreservingAbstractRunnable.doRun(ThreadContext.java:777) [elasticsearch-7.16.3.jar:7.16.3]",
"at org.elasticsearch.common.util.concurrent.AbstractRunnable.run(AbstractRunnable.java:26) [elasticsearch-7.16.3.jar:7.16.3]",
"at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136) [?:?]",
"at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635) [?:?]",
"at java.lang.Thread.run(Thread.java:833) [?:?]"] }

解决方式

1. 扩容 PVC（从 1Gi 修改为 10Gi）

   kubectl edit pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0 -n mcamel-system

   ```
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: 10Gi
   ```

2. PVC 扩容日志

   查看 elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0 扩容日志信息。

   kubectl describe pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0 -n mcamel-system

Name: elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0
Namespace: mcamel-system
StorageClass: hwameistor-storage-lvm-hdd
Status: Bound
Volume: pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3
Labels: common.k8s.elastic.co/type=elasticsearch
elasticsearch.k8s.elastic.co/cluster-name=mcamel-common-es-cluster-masters
elasticsearch.k8s.elastic.co/statefulset-name=mcamel-common-es-cluster-masters-es-masters
Annotations: pv.kubernetes.io/bind-completed: yes
pv.kubernetes.io/bound-by-controller: yes
volume.beta.kubernetes.io/storage-provisioner: lvm.hwameistor.io
volume.kubernetes.io/selected-node: xulongju-worker03
Finalizers: [kubernetes.io/pvc-protection]
Capacity: 10Gi
Access Modes: RWO
VolumeMode: Filesystem
Used By: mcamel-common-es-cluster-masters-es-masters-0
Events:

Type	Reason	Age	From	Message
Normal	WaitForPodScheduled	51m (x18 over 55m)	persistentvolume-controller	waiting for pod mcamel-common-es-cluster-masters-es-masters-0 to be scheduled

Normal WaitForFirstConsumer 50m (x7 over 56m) persistentvolume-controller
waiting for first consumer to be created before binding
Normal ExternalProvisioning 50m persistentvolume-controller
waiting for a volume to be created, either by external provisioner "lvm.hwameistor.io" or manually created by system administrator
Normal Provisioning 50m lvm.hwameistor.io_hwameistor-local-storage-csi-controller-68c9df8db8-kzdgn_680380b5-fc4d-4b82-ba80-5681e99a8711 External provisioner is provisioning volume for claim "mcamel-system/elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0"
Normal ProvisioningSucceeded 50m lvm.hwameistor.io_hwameistor-local-storage-csi-controller-68c9df8db8-kzdgn_680380b5-fc4d-4b82-ba80-5681e99a8711 Successfully provisioned volume pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3
Warning ExternalExpanding 3m39s volume_expand
Ignoring the PVC: didn't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.
Warning VolumeResizeFailed 3m39s external-resizer lvm.hwameistor.io
resize volume "pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3" by resizer "lvm.hwameistor.io" failed: rpc error: code = Unknown desc = volume expansion not completed yet
Warning VolumeResizeFailed 3m39s external-resizer lvm.hwameistor.io
resize volume "pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3" by resizer "lvm.hwameistor.io" failed: rpc error: co

de = Unknown desc = volume expansion in progress

Normal Resizing 3m38s (x3 over 3m39s) external –resizer lvm.hwameistor.io

External resizer is resizing volume pvc–42309e19–b74f–45b4–9284–9c68b7dd93b3

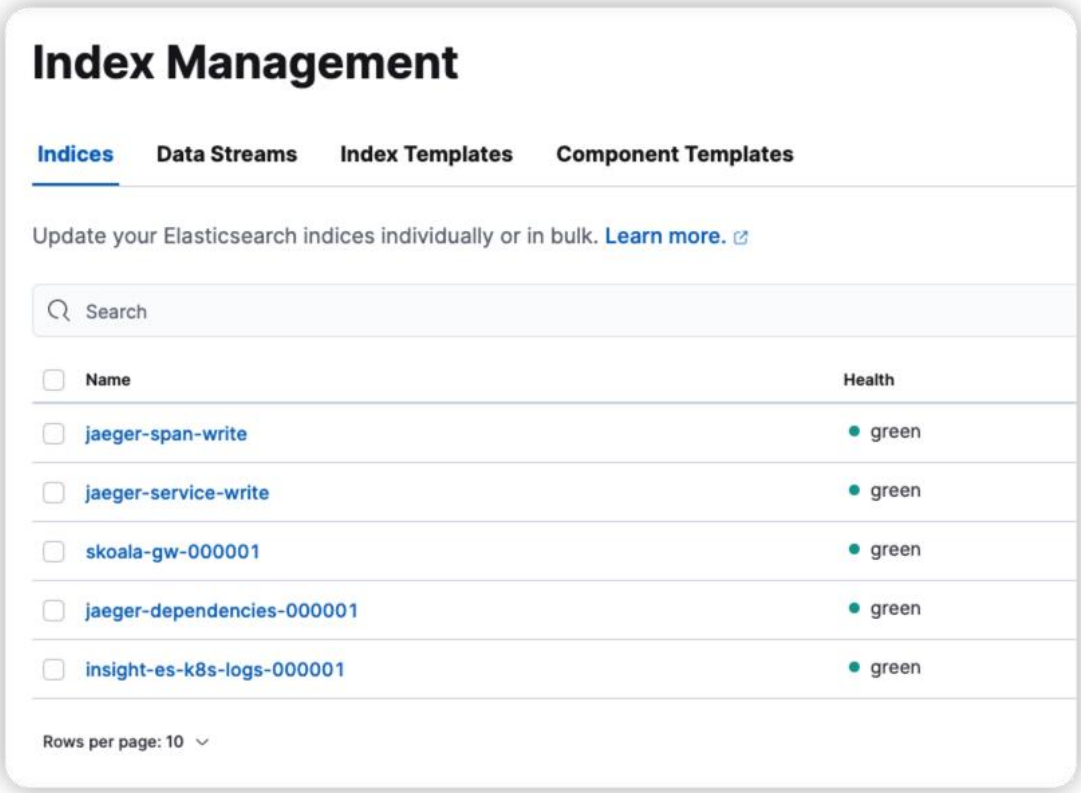
Normal FileSystemResizeRequired 3m38s external –resizer lvm.hwameistor.io

Require file system resize of volume on node

Normal FileSystemResizeSuccessful 2m42s kubelet

Elasticsearch 业务索引别名被占用

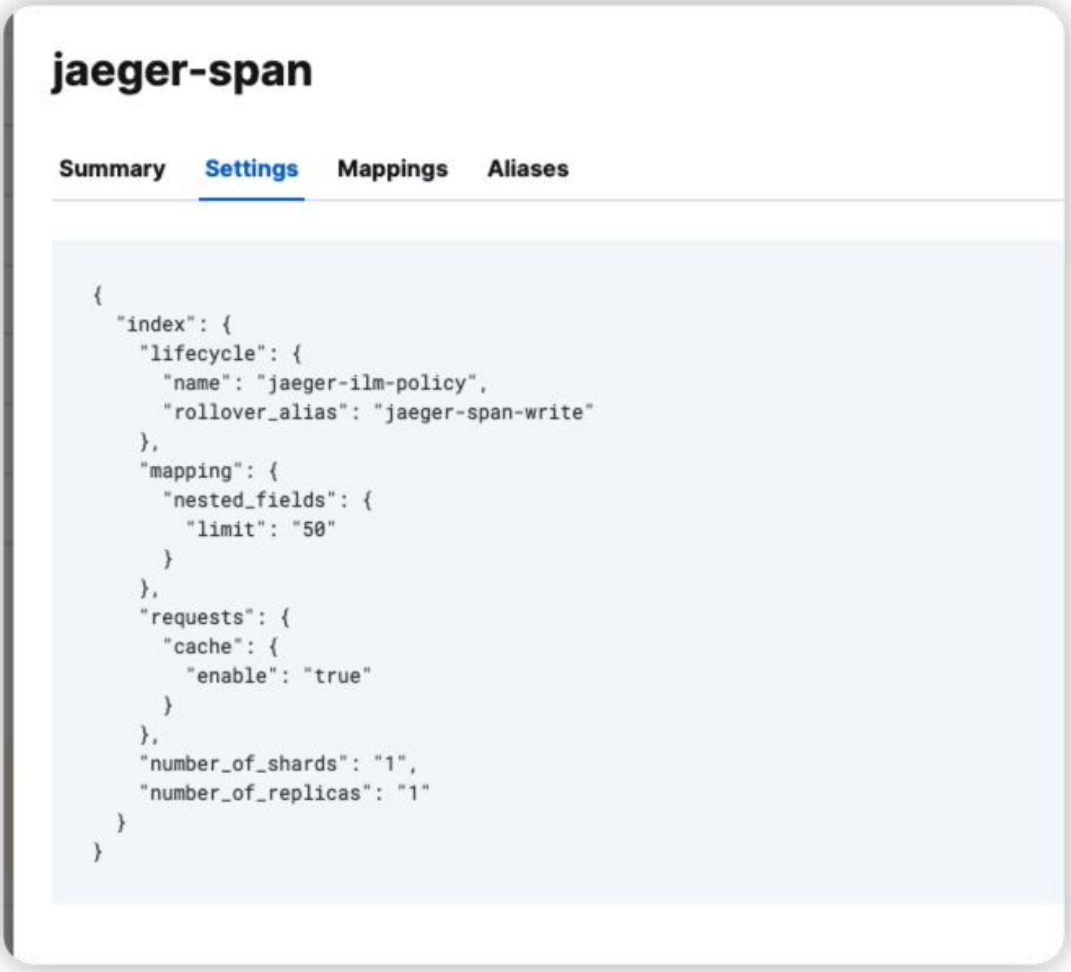
现象：索引别名被占用



image

此图中*-write 为别名，例如 jaeger-span-write，需要对此别名进行处理

查看业务索引模板中使用的别名 rollover_alias 对应值



image

临时处理方式：进入 es pod 容器内执行以下脚本：

1. 修改 TEMPLATE_NAME 对应值
2. 修改 INDEX_ALIAS 对应值
3. 需要进入 elasticsearch pod 中执行该脚本
4. 修改里面 elastic 用户的密码值（ES_PASSWORD=xxxx）

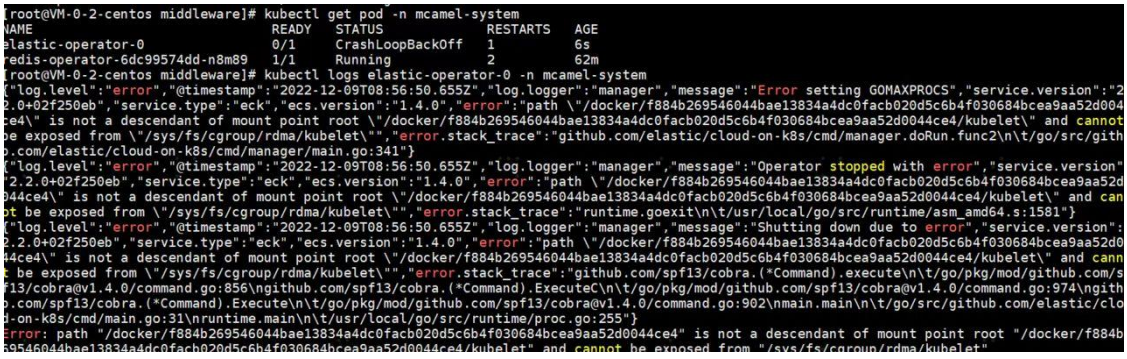
```
#!/bin/bash
# Add a template/policy/index
TEMPLATE_NAME=insight-es-k8s-logs
INDEX_ALIAS="${TEMPLATE_NAME}-alias"
ES_PASSWORD="DaoCloud"
ES_URL=https://localhost:9200
while [[ "$(curl -s -o /dev/null -w '%{http_code}\n' -u elastic:${ES_PASSWORD}
 $ES_URL -k)" != "200" ]]; do sleep 1; done
curl -XDELETE -u elastic:${ES_PASSWORD} -k "$ES_URL/${INDEX_ALIAS}"
curl -XPUT -u elastic:${ES_PASSWORD} -k "$ES_URL/${TEMPLATE_NAME}-00
0001" -H 'Content-Type: application/json' -d'{"aliases": {'\""${INDEX_ALIAS}"\"':
{"is_write_index": true }}}'
```

> 注意：此脚本存在一定失败几率，取决于数据写入速度，作为临时解决方式。

真实情况需要停止数据源的写入情况，再执行上述方法。

报错 Error setting GoMAXPROCS for operator

报错信息



image

环境信息：

kind 版本：0.17.0

containerd:1.5.2

k8s:1.21.1

解决方式

升级版本：

kind：1.23.6

runc version 1.1.0

报错 Terminating due to java.lang.OutOfMemoryError: Java heap space

完整的报错信息如下：

{"type": "server", "timestamp": "2023-01-04T14:44:05,920Z", "level": "WARN", "component": "o.e.d.PeerFinder", "cluster.name": "gsc-cluster-1-master-es", "node.name": "gsc-cluster-1-master-es-es-data-0", "message": "address [127.0.0.1:9305], node [null], requesting [false] connection failed: [][127.0.0.1:9305] connect_exception: Connection refused: /127.0.0.1:9305: Connection refused", "cluster.uuid": "JOa0U_Q6T7WT60SPYiR1lg", "node.id": "_zlorWVeRbyrUMYf9wJgfQ" }
{"type": "server", "timestamp": "2023-01-04T14:44:06,379Z", "level": "WARN", "component": "o.e.m.j.JvmGcMonitorService", "cluster.name": "gsc-cluster-1-master-es", "node.name": "gsc-cluster-1-master-es-es-data-0", "message": "[gc][15375] overhead, spent [1.3s] collecting in the last [1.3s]", "cluster.uuid": "JOa0U_Q6T7WT60SPYiR1lg", "node.id": "_zlorWVeRbyrUMYf9wJgfQ" }
{"timestamp": "2023-01-04T14:44:06+00:00", "message": "readiness probe failed", "curl_rc": "28"}
java.lang.OutOfMemoryError: Java heap space
Dumping heap to data/java_pid7.hprof ...
{"timestamp": "2023-01-04T14:44:11+00:00", "message": "readiness probe failed", "curl_rc": "28"}
{"timestamp": "2023-01-04T14:44:14+00:00", "message": "readiness probe failed", "curl_rc": "28"}
{"timestamp": "2023-01-04T14:44:17+00:00", "message": "readiness probe failed",

"curl_rc": "28"}
{"timestamp": "2023-01-04T14:44:21+00:00", "message": "readiness probe failed", "curl_rc": "28"}
{"timestamp": "2023-01-04T14:44:26+00:00", "message": "readiness probe failed", "curl_rc": "28"}
{"timestamp": "2023-01-04T14:44:31+00:00", "message": "readiness probe failed", "curl_rc": "28"}
Heap dump file created [737115702 bytes in 25.240 secs]
Terminating due to java.lang.OutOfMemoryError: Java heap space

解决方式

如果在条件允许的情况下，可以进行资源及容量规划。

kubectl edit elasticsearch mcamel-common-es-cluster-masters -n mcamel-system



image

OCP 环境安装 Elasticsearch 时报错 Operation not permitted

报错信息

```
[root@10-29-16-95 ~]# kubectl logs mcamel-common-es-cluster-masters-es-masters-0
+ ulimit -HSn 65535
+ /bin/tini -- /usr/local/bin/docker-entrypoint.sh eswrapper
chroot: cannot change root directory to '/': Operation not permitted
[root@10-29-16-95 ~]#
```

image

解决方式

```
        failureThreshold: 3
        initialDelaySeconds: 10
        periodSeconds: 5
        successThreshold: 1
        timeoutSeconds: 5
      resources:
        limits:
          cpu: "2"
          memory: 2Gi
        requests:
          cpu: 200m
          memory: 1Gi
      securityContext:
        capabilities:
          add:
          - SYS_CHROOT
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /mnt/elastic-internal/downward-api
        name: downward-api
        readOnly: true
      - mountPath: /usr/share/elasticsearch/bin
        name: elastic-internal-elasticsearch-bin-local
      - mountPath: /mnt/elastic-internal/elasticsearch-config
        name: elastic-internal-elasticsearch-config
-- INSERT --
```

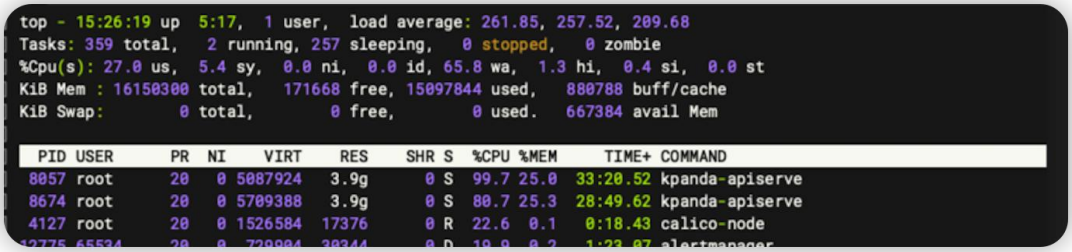
image

某个节点磁盘读吞吐异常、CPU workload 很高

异常信息



image



image

解决方式

如果 es 在此节点，可以将 ES 进程杀掉恢复。

数据写入 Elasticsearch 时报错 status:429, es_rejected_execution_exception

完整的报错信息如下：

[2023/03/23 09:47:16] [error] [output:es:es.kube.kubeevent.syslog] error: Output {"took":0,"errors":true,"items":[{"create":{"_index":"insight-es-k8s-logs-000067","_type":"_doc","_id":"MhomDlcBLVS7yRloG6PF","status":429,"error":{"type":"es_rejected_execution_exception","reason":"rejected execution of org.elasticsearch.action.support.replication.TransportWriteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoolExecutor[name = mcamel-common-es-cluster-masters-es-data-0/write, queue capacity = 10000, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2, active threads = 2, queued tasks = 10000, completed tasks = 11472149]]"}}},{"create":{"_index":"insight-es-k8s-logs-000067","_type":"_doc","_id":"MxomDlcBLVS7yRloG6PF","status":429,"error":{"type":"es_rejected_execution_exception","reason":"rejected execution of org.elasticsearch.action.support.replication.TransportWriteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoolExecutor[name = mcamel-co

mmon-es-cluster-masters-es-data-0/write, queue capacity = 10000, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2, active threads = 2, queued tasks = 10000, completed tasks = 11472149]]"}}},{"create":{"_index":"insight-es-k8s-logs-000067","_type":"_doc","_id":"NBomDIcBLVS7yRloG6PF","status":429,"error":{"type":"es_rejected_execution_exception","reason":"rejected execution of org.elasticsearch.action.support.replication.TransportWriteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoolExecutor[name = mcamel-common-es-cluster-masters-es-data-0/write, queue capacity = 10000, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2, active threads = 2, queued tasks = 10000, completed tasks = 11472149]]"}}}]}

解决方式

- 方式 1：产生 429 错误的原因是 Elasticsearch 写入并发过大，Elasticsearch 来不及处理导致，可以适当降低写入并发并控制写入量。
- 方式 2：在资源允许的情况下，可以适当调大队列大小

```
nodeSets:
  - config:
      node.store.allow_mmap: false
      thread_pool.write.queue_size: 1000 #增加/调大此参数的值
```

方式 1 和方式 2 可以配合使用。

Cilium 基于 eBPF 实现 socket 加速

原创作者：tanberBro



home

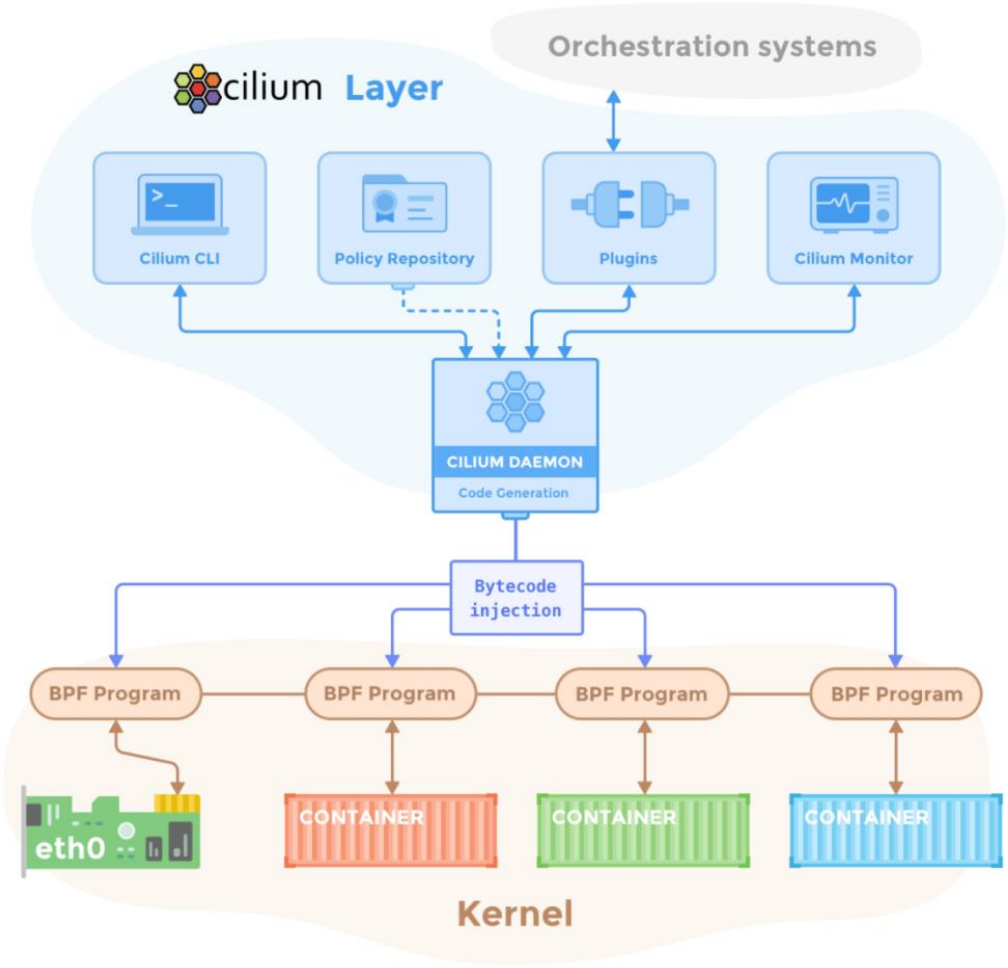
随着云原生的不断发展，越来越多的应用部署在云上。其中有些应用对实时性要求非常严苛，这使得我们必须提升这些应用性能，达到更快的服务速度。

场景

为了达到更快的服务速度，一个场景是：当两个互相调用的应用部署在同一个节点上的时候，每次请求和返回都需要经过 socket 层、TCP/IP 协议栈、数据链路层、物理层。 如果请求和返回绕过 TCP/IP 协议栈，直接在 socket 层将数据包重定向到对端 socket，那将大大减少发送数据包耗时，从而加快服务速度。 基于这个思路，eBPF 技术通过映射存储 socket 信息，利用帮助函数实现了将数据包重定向到对端 socket 层的能力。Cilium 正是基于 eBPF 这个能力，实现了 socket 层加速效果。

架构

Cilium 整体架构如下：

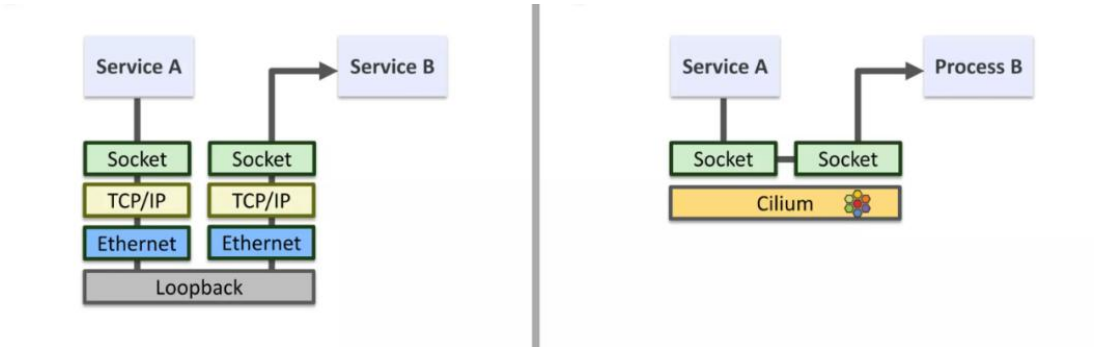


架构

Cilium 在 daemon 组件实现 socket 层加速，daemon 组件会在集群中每个节点都启动一个 Pod，所以 socket 加速效果会应用在每个节点上。

能力概述

Cilium 在处理源端和目标端在同一个节点的时候，可以在 socket 层将流量直接重定向到目标端 socket，这样直接绕过了整个 TCP/IP 协议栈，进而达到加速的效果，如下图所示。



加速效果

实现原理

原理概述

Cilium 使用以下 eBPF 程序和映射实现上述能力：

- eBPF 映射包含：
  - Cilium_sock_ops（类型：BPF_MAP_TYPE_SOCKHASH）：存储 socket 信息。
- eBPF 程序包含：
  - obpf_sockmap （类型：BPF_PROG_TYPE_SOCK_OPS）：拦截系统中 socket 建立连接操作，并将 socket 信息存储到 cilium_sock_ops。
  - obpf_redir_proxy（类型：BPF_PROG_TYPE_SK_MSG）：拦截系统中 sendmsg 系统调用，提取 key，将数据直接重定向到对端 socket。

bpf_sockmap 需要 attach 到 cgroup 上，在 Cilium 中会被 attach 到 /run/cilium/cgroupv2，所以 bpf_sockmap 可以作用到系统中所有属于该 cgroup 的进程。

代码实现

以下代码基于 cilium/cilium v1.13 分支。Cilium 使用 C 语言编写 eBPF 代码，并通过 go 语言 compile、load 和 attach eBPF 代码。

eBPF 映射

cilium_sock_ops：该映射使用 sock_key 作为 key，sock_key 存储源 IP、目标 IP、协议、源端口、目标端口，即 socket 五元组。

bpf/sockops/bpf_sockops.h

```
struct sock_key {
    union {
        struct {
            __u32           sip4;
            __u32           pad1;
            __u32           pad2;
            __u32           pad3;
        };
        union v6addr        sip6;
    };
    union {
        struct {
            __u32           dip4;
            __u32           pad4;
            __u32           pad5;
            __u32           pad6;
        };
        union v6addr        dip6;
    };
    __u8  family;
    __u8  pad7;
    __u16 pad8;
    __u32 sport;
    __u32 dport;
} __packed;
```

```
struct {
    __uint(type, BPF_MAP_TYPE_SOCKHASH);
    __type(key, struct sock_key);
    __type(value, int);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
    __uint(max_entries, SOCKOPS_MAP_SIZE);
} SOCK_OPS_MAP __section_maps_btf;
```

eBPF 程序

bpf_sockmap：BPF_PROG_TYPE_SOCK_OPS 类型的程序和其他类型程序不同，它会在多个地方执行，参数 skops->op 可获取当前程序执行的地方。 这里会在主动建立连接、被动建立连接完成，并且协议族是 AF_INET6（TCP/IPv6）、AF_INET（TCP/IPv4）时，进入 bpf_sock_ops_ipv4 处理 socket。bpf_sock_ops_ipv6 会判断目标 IPV4 不为空，也会进入 bpf_sock_ops_ipv4 做进一步处理。

bpf/sockops/bpf_sockops.c

```
__section("sockops")
int cil_sockops(struct bpf_sock_ops *skops)
{
    __u32 family, op;

    family = skops->family;
    op = skops->op;

    switch (op) {
    case BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB:
    case BPF_SOCK_OPS_ACTIVE_ESTABLISHED_CB:
#ifdef ENABLE_IPV6
        if (family == AF_INET6)
            bpf_sock_ops_ipv6(skops);
```

```
#endif
#ifdef ENABLE_IPV4
        if (family == AF_INET)
            bpf_sock_ops_ipv4(skops);
#endif
        break;
    default:
        break;
    }

    return 0;
}
```

bpf_sock_ops_ipv4：这里首先调用 sk_extract4_key 提取出 key，也就是 cilium_sock_ops 的 key；调用 sk_lb4_key 提取出 lb4_key， 调用 lb4_lookup_service，根据 lb4_key 查询映射：cilium_lb4_services_v2 获取 L4 负载信息，如果找到说明目标是 L4 负载地址， 直接 return，结束调用，向下使用 L4/L3 协议栈处理；调用 lookup_ip4_remote_endpoint 根据目标 IP 查询映射：cilium_ipcache， 该映射存储了所有网络信息和身份信息的对应关系，找到目标 IP 的身份 ID；调用 policy_sk_egress 进行策略裁决； 调用 redirect_to_proxy 判断是否要重定向到 proxy（verdict > 0），如果是的话，将 key 的源 IP、源端口换成 proxy 的地址， 调用 bpf 帮助函数 bpf_sock_hash_update 将 key 存储到 cilium_sock_ops 中；如果不是的话，调用 __lookup_ip4_endpoint， 根据目标 IP 查询映射：cilium_lxc，该映射存储了所有本地的 Pod 信息、主机信息， 如果找到，说明是本机内部访问，调用 bpf 帮助函数 bpf_sock_hash_update 将 key 存储到 cilium_sock_ops 中。

bpf/sockops/bpf_sockops.c

```
static inline void bpf_sock_ops_ipv4(struct bpf_sock_ops *skops)
{
    struct lb4_key lb4_key = {};
```

```
__u32 dip4, dport, dst_id = 0;
struct endpoint_info *exists;
struct lb4_service *svc;
struct sock_key key = {};
int verdict;

sk_extract4_key(skops, &key);

/* If endpoint a service use L4/L3 stack for now. These can be
 * pulled in as needed.
 */
sk_lb4_key(&lb4_key, &key);
svc = lb4_lookup_service(&lb4_key, true, true);
if (svc)
    return;

/* Policy lookup required to learn proxy port */
if (1) {
    struct remote_endpoint_info *info;

    info = lookup_ip4_remote_endpoint(key.dip4);
    if (info != NULL && info->sec_label)
        dst_id = info->sec_label;
    else
        dst_id = WORLD_ID;
}

verdict = policy_sk_egress(dst_id, key.sip4, (__u16)key.dport);
if (redirect_to_proxy(verdict)) {
    __be32 host_ip = IPV4_GATEWAY;
```

```
        key.dip4 = key.sip4;
        key.dport = key.sport;
        key.sip4 = host_ip;
        key.sport = verdict;

        sock_hash_update(skops, &SOCK_OPS_MAP, &key, BPF_NOEXIST);
        return;
    }

    /* Lookup IPv4 address, this will return a match if:
     * - The destination IP address belongs to the local endpoint manage
     *   by Cilium.
     * - The destination IP address is an IP address associated with the
     *   host itself.
     * Then because these are local IPs that have passed LB/Policy/NAT
     * blocks redirect directly to socket.
     */
    exists = __lookup_ip4_endpoint(key.dip4);
    if (!exists)
        return;

    dip4 = key.dip4;
    dport = key.dport;
    key.dip4 = key.sip4;
    key.dport = key.sport;
    key.sip4 = dip4;
    key.sport = dport;

    sock_hash_update(skops, &SOCK_OPS_MAP, &key, BPF_NOEXIST);
}
```

bpf_redir_proxy：BPF_PROG_TYPE_SK_MSG 类型程序会在 socket 上调用 sendmsg 时执行，这里就是执行 bpf_redir_proxy。 该程序首先调用 sk_extract4_key 提取出 key，也就是 cilium_sock_ops 的 key；调用 lookup_ip4_remote_endpoint 根据目标 IP 查询映射：cilium_ipcache，该映射存储了所有网络信息和身份信息的对应关系，找到目标 IP 的身份 ID； 调用 policy_sk_egress 进行策略裁决；如果裁决通过，则调用 bpf 帮助函数 bpf_msg_redirect_hash， 该函数通过传入的 key 查询映射 cilium_sock_ops，获取对端 socket 信息，进而将消息重定向到对端 socket 的 ingress 方向，完成 socket 层的重定向。

bpf/sockops/bpf_redir.c

```
__section("sk_msg")
int cil_redir_proxy(struct sk_msg_md *msg)
{
    struct remote_endpoint_info *info;
    __u64 flags = BPF_F_INGRESS;
    struct sock_key key = {};
    __u32 dst_id = 0;
    int verdict;

    sk_msg_extract4_key(msg, &key);

    /* Currently, pulling dstIP out of endpoint
     * tables. This can be simplified by caching this information with the
     * socket to avoid extra overhead. This would require the agent though
     * to flush the sock ops map on policy changes.
     */
    info = lookup_ip4_remote_endpoint(key.dip4);
    if (info != NULL && info->sec_label)
        dst_id = info->sec_label;
    else
```

```
        dst_id = WORLD_ID;

    verdict = policy_sk_egress(dst_id, key.sip4, (__u16)key.dport);
    if (verdict >= 0)
        msg_redirect_hash(msg, &SOCK_OPS_MAP, &key, flags);
    return SK_PASS;
}
```

compile、load 和 attach eBPF 程序

Cilium 使用 go 语言，调用外部命令 clang llc bpftool 进行 compile、load 和 attach。

当 Cilium 配置 sockops-enable: “true”，且内核支持 BPF Sock ops，daemon 启动时进行 init 操作的时候进行 compile、load 和 attach，代码如下：

daemon/cmd/daemon.go

```
.....
        if option.Config.SockopsEnable {
            eppolicymap.CreateEPPolicyMap()
            if err := sockops.SockmapEnable(); err != nil {
                return fmt.Errorf("failed to enable Sockmap: %w", err)
            } else if err := sockops.SkmsgEnable(); err != nil {
                return fmt.Errorf("failed to enable Sockmsg: %w", err)
            } else {
                sockmap.SockmapCreate()
            }
        }
.....
```

sockops.SockmapEnable：首先调用 clang… | llc …将 bpf_sockops.c 编译成 bpf_sockops.o； 然后将 bpf_sockops.o 加载到内核，加载之后

bpf_sockops.o 会 pin 到/sys/fs/bpf/bpf_sockops（默认），再调用 bpftool cgroup attach...将/sys/fs/bpf/bpf_sockops attach 到/run/cilium/cgroupv2（默认）cgroup 上，最后将 cilium_sock_ops pin 到 /sys/fs/bpf/tc/globals/cilium_sock_ops。

pkg/sockops/sockops.go

```
func SockmapEnable() error {
    err := bpfCompileProg(cSockops, oSockops)
    if err != nil {
        return err
    }
    progID, mapID, err := bpfLoadAttachProg(oSockops, eSockops, sockMap)
    if err != nil {
        return err
    }
    log.Infof("Sockmap Enabled: bpf_sockops prog_id %d and map_id %d loaded", progID, mapID)
    return nil
}
```

sockops.SkmsgEnable：首先调用 clang... | llc ...将 bpf_redir.c 编译成 bpf_redir.o；然后将 bpf_redir.o 加载到内核，加载之后 bpf_redir.o 会 pin 到 /sys/fs/bpf/bpf_redir（默认），根据 bpf_redir 调用 bpftool prog show pinned...获取 bpf_redir 程序 ID，调用 bpftool map show 获取 bpf_sockops 所使用的的映射 ID，最后调用 bpftool prog attach...根据程序 ID、映射 ID 将 bpf_redir attach 到 cilium_sock_ops。

pkg/sockops/sockops.go

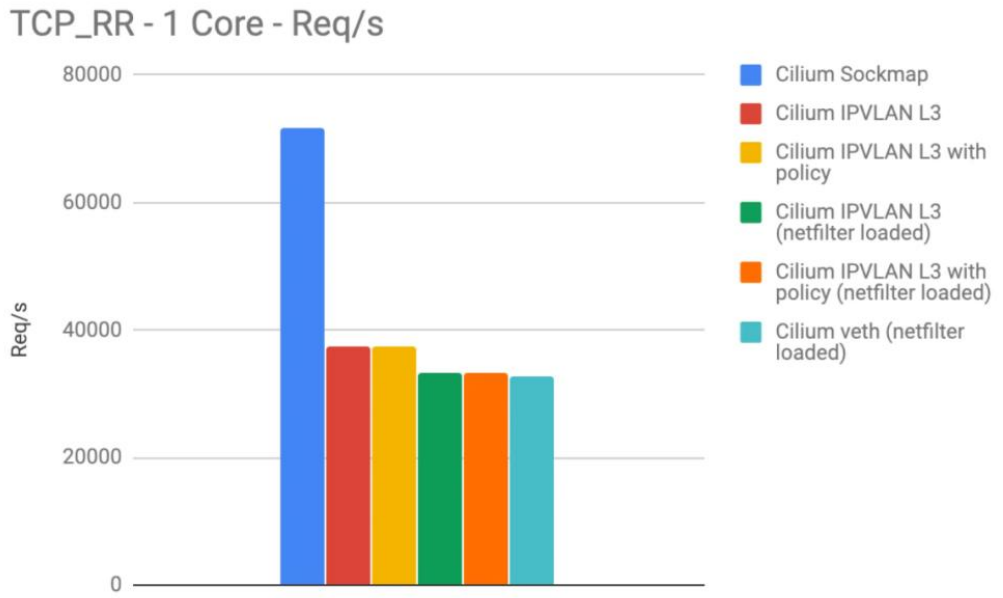
```
func SkmsgEnable() error {
    err := bpfCompileProg(clPC, olPC)
    if err != nil {
```

```
            return err
        }

        err = bpfLoadMapProg(oIPC, eIPC)
        if err != nil {
            return err
        }
        log.Info("Sockmsg Enabled, bpf_redir loaded")
        return nil
}
```

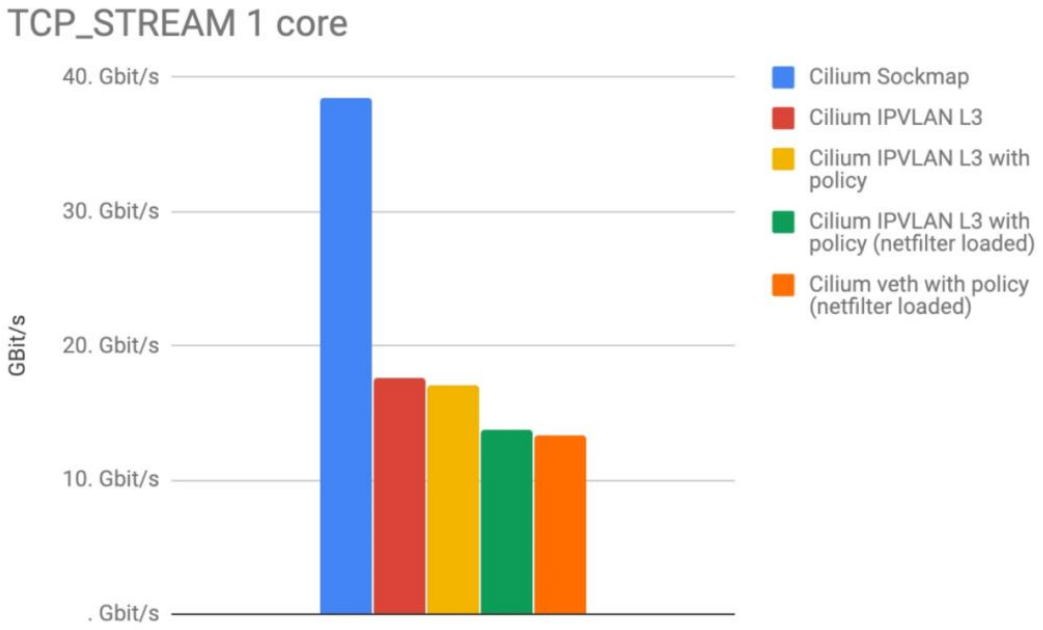
小结

综上所述，使用 cilium_sock_ops 映射做 socket 信息存储载体，通过 bpf_sockmap 程序拦截并存储 socket 信息， 通过 bpf_redir_proxy 调用 bpf_msg_redirect_hash 实现 socket 重定向。通过 Cilium daemon 组件， 将 eBPF 程序和映射 load 到内核中，并 attach 到/run/cilium/cgroupv2 上，实现同一节点 socket 通讯加速， 由于加速的范围是基于 cgroup 监听的所有主机上的 socket，所以只要是同一节点上的 socket 之间通信都是可以完成加速的。

加速效果

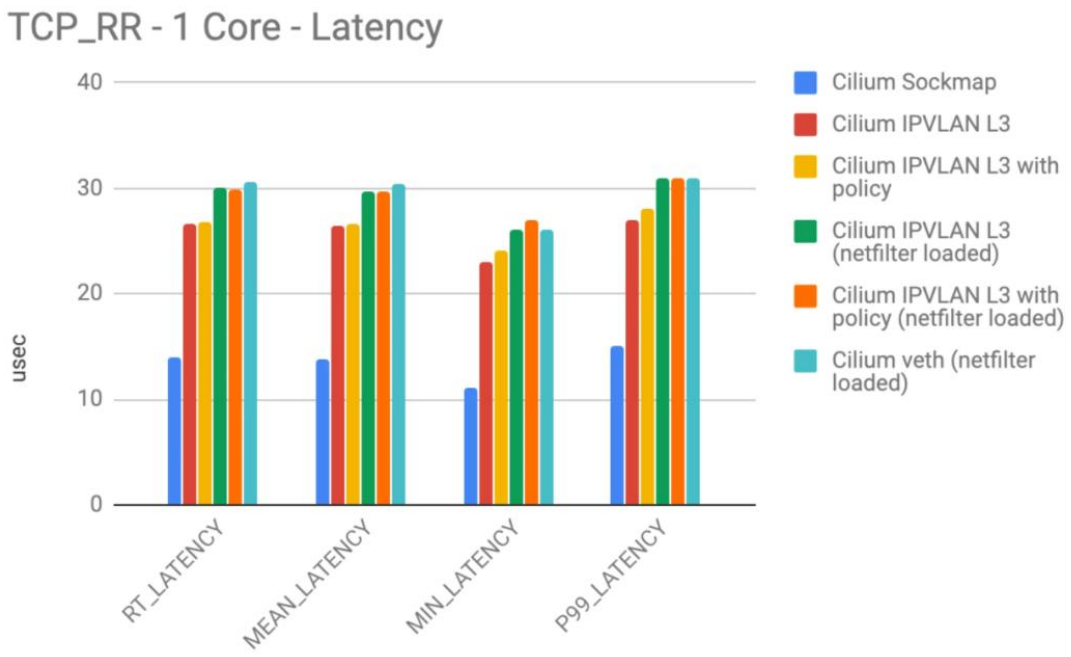


加速效果



加速效果

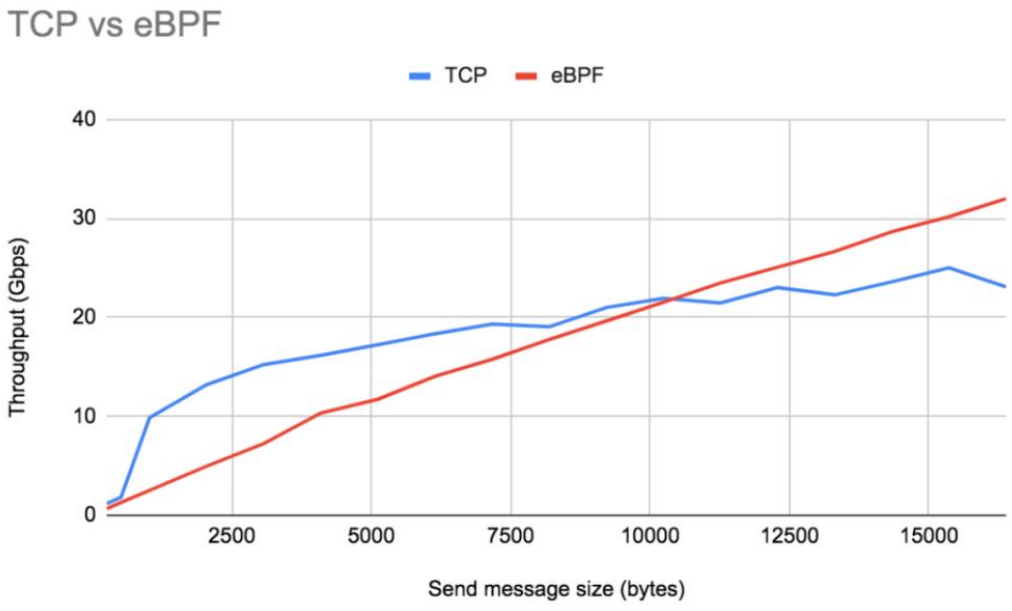
如上图展示了，使用 eBPF socket 加速（蓝色）之后，请求数/s 和吞吐量都成倍增加。



加速效果

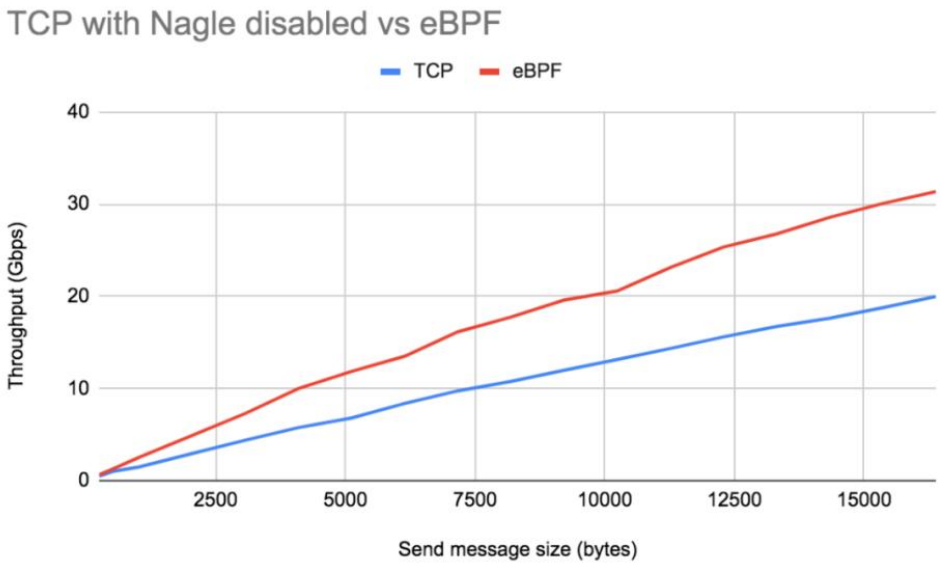
如上图展示了，eBPF socket 加速（蓝色）和 TCP/IP 栈延迟对比，eBPF socket 加速之后的性能优于常规 TCP/IP 栈。

接下来看看当发送和接收多种长度的消息时加速效果对比，深入分析加速原理：



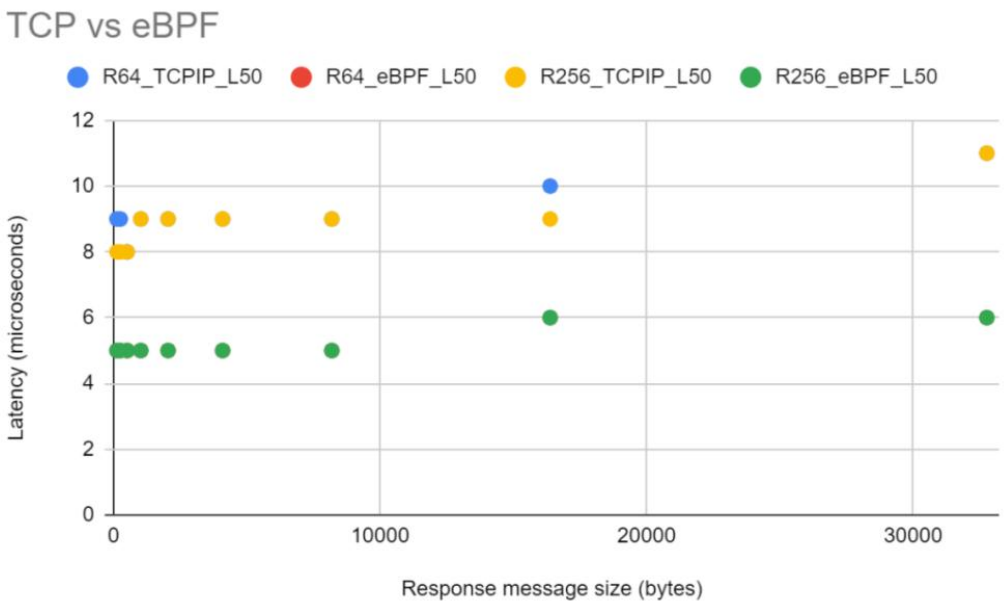
加速效果

如上图展示了，使用 eBPF socket 加速之后，吞吐量和发送消息大小呈线性关系。这是因为当应用程序发送较大的消息时， 几乎没有额外的开销，但是当发送消息比较小时，使用 TCP/IP 协议栈，反而吞吐量会大于 eBPF socket 加速之后的吞吐量， 这是由于 TCP/IP 栈默认开启了 Nagle 算法。Nagle 的算法是用来解决小数据包在慢速网络中泛滥导致拥塞的问题， 在该算法中只要有一个 TCP 段在小于 TCP MSS 大小的情况下未被确认，就会进行批处理传输数据操作。 这种批处理导致一次传输更多的数据并分摊开销，所以能超过 eBPF socket 加速之后吞吐量。但是随着发送消息越来越大， 超过 MSS，TCP/IP 栈就会失去其批处理优势，在这些大数据包发送大小下，eBPF socket 加速凭借其低开销远远超过启用 Nagle 算法的 TCP/IP 栈的吞吐量。



加速效果

如上图展示了，在禁用 Nagle 算法的情况下，与 eBPF socket 加速相比，常规 TCP 的吞吐量增益完全消失了。 TCP/IP 栈和 eBPF socket 加速的性能都按预期线性增加，由于 eBPF 的每次发送调用的成本开销是固定的， 所以 eBPF 具有比常规 TCP/IP 更大的斜率。对于较大的发送消息大小和较小的 TCP MSS，这种性能差距更为明显。



加速效果

如上图展示了，eBPF socket 加速和 TCP/IP 栈延迟对比，eBPF socket 加速优于常规 TCP/IP 栈。 性能优于常规 TCP/IP 栈近 50%。与 TCP/IP 栈相比，eBPF 通过将数据包从源套接字的传输队列重定向到目标套接字的接收队列，从而消除了任何协议级别的开销（慢启动、拥塞避免、流量控制等）。此外，请求消息大小的大小对延迟没有影响。

参考资料

- Welcome to Cilium's documentation!
- How to use eBPF for accelerating Cloud Native applications
- Lessons from using eBPF (and bypassing TCP/IP) for accelerating Cloud Native applications
- Accelerating Envoy and Istio with Cilium and the Linux Kernel
- Congestion Control in IP/TCP Internetworks
- Cilium 1.4: Multi-Cluster Service Routing, DNS Authorization, IPVLAN support, Transparent Encryption, Flannel Integration, Benchmarking other CNIs

有效提升资源利用率，让闲置的算力跑起来

本文作者：云原生研究院产品经理 Stella



提升资源利用率

资源利用率现状

据 Gartner 最新预测：“2023 年全球企业对云服务的支出预计将增长 20.7%，达到 6000 亿美元， 高于 2022 年的 4900 亿美元以及 18.8% 的增长预测，并且 PaaS 增长率将达到 23.2%。 到 2025 年，超过 85% 的企业将采用优先上云的原则。”与此同时，许多企业表示：“在没有云原生技术的情况下，将无法全面执行数字战略。”

对于已经拥抱云原生的企业来说，云技术为他们创造了丰厚的价值。 由于现代企业大多希望构建出具有高度可扩展性、弹性和灵活性的应用程序， 并能够快速更新迭代满足自身和客户需求，目前没有哪项技术能像云技术一样帮助企业实现这些目标。 尤其在提升资源利用率与节约成本方面，云原生已连续多年排名第一，成为了企业降本增效的最佳路径。

但随着许多企业规模持续扩大和业务量的不断增长，云基础设施服务成本高昂和资源利用率低下的问题也越来越突出， 并非所有计算资源都能得到充分利用，云上

浪费性支出依然是令众多企业头疼的话题。 根据 Flexera《2022 年云状况》报告数据显示：“优化云的现有使用率已连续六年成为受访企业的首要问题，云成本支出预计增加 29% ，其中估计 32% 的云支出会被浪费掉。”另外，中国通信院在去年发表的关于云原生成本管理白皮书中指出：“Kubernetes 资源预留过多，普遍存在 50% 以上的浪费。”而刚过去的 2022 北美 KubeCon 会议中，“成本”也被提及多达 15 次。虽然云原生在提升企业资源利用率方面表现优异，但是还有很大的进步空间。因此不论过去还是现在，如何进一步帮助企业有效提升资源利用率并充分发挥上云的价值，都是云原生从业者们义不容辞的责任。

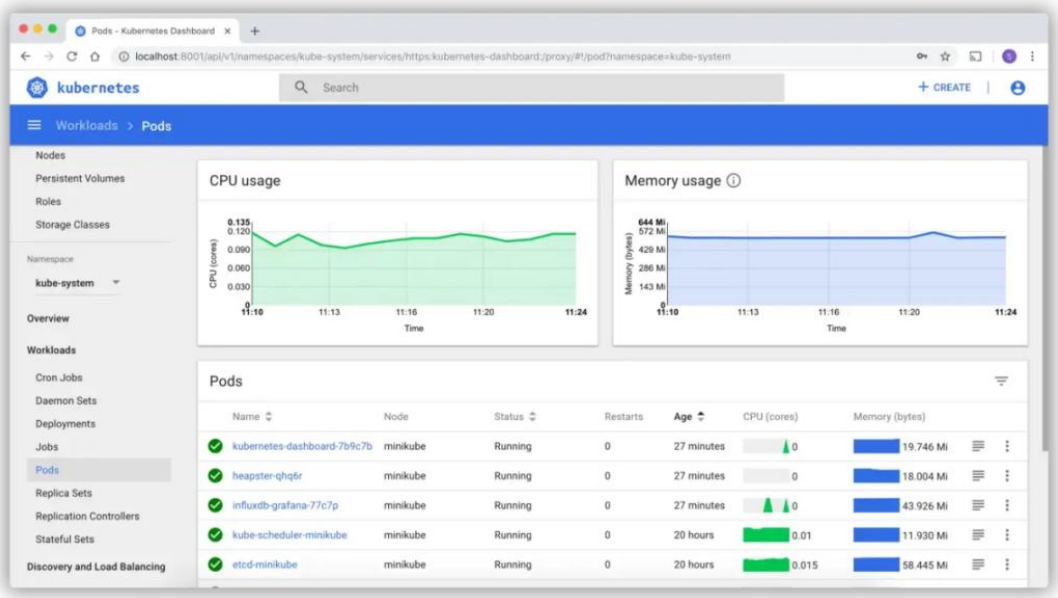
如何进行资源优化

哪些资源在被浪费?

想要进行资源优化，必须先识别出哪些业务分配了过剩的资源。 而资源使用率通常通过 CPU、内存、磁盘、进出带宽等指标数据来衡量。 一般的公有云或私有云厂商都会对这些指标的采集与查看提供现成的解决方案。

通过对历史指标的查看，可以快速了解资源的实际利用率以及周期性规律，在此基础上，才可进行细粒度的资源利用率优化。

1. 合理设置 Request 和 Limit



Request 和 Limit

Kubernetes 中创建一个 Pod 时，可以指定容器对 CPU 和内存的资源请求量(即 request)，以及资源限制量(即 limit)。 其中 Request 字段代表 Pod 对资源需求的最小值，每个 Kubernetes 节点可以分配给 Pod 的 CPU 和内存数量都是一定的。 调度器在调度时只会考虑那些未分配资源量且满足 Pod 请求量的节点。如果节点的未分配资源量小于 Pod 的资源请求量， 那么调度器则不会把 Pod 调度到此节点上。

在实践中，一个集群可能会有多个团队共同使用，各个团队为了应对流量的高峰时段，保证自己的工作负载的服务质量， 通常会为 Pod 申请较多的资源，也就是将 Request 设置得较大。这样 Request 和实际使用之间的差值就会被浪费，而且这个差值的资源，是不能被其它工作负载所使用的。所以为了满足所有团队对资源的需求，集群中就需要加入更多的节点。 与此同时，更多的节点也带来了更多的资源浪费。

所以，合理地为容器设置 Request 和 Limit 值，对优化资源利用率至关重要，而设置 Pod 的 Request 和 Limit 也是一项艺术， 需要考虑应用的工作负载、

集群的规模以及其他因素。为了合理地设置 Pod 的资源请求和限制，建议考虑以下几点：

1. 运行在 Pod 中的容器工作负载和资源需求。如果容器的工作负载很高，则需要更多的 CPU 和内存。
2. 为 Pod 设置的资源限制应该足够大，以便容器能够按需使用资源。如果设置的资源限制过小，则容器可能会被过早地杀死。
3. 为 Pod 设置的资源请求应该尽可能接近容器的实际资源需求，以帮助 Kubernetes 进行调度决策。 如果资源 请求过高，则 Pod 可能无法调度到节点；如果过低，则可能会浪费节点上的资源。

为了确定 Pod 的资源请求和限制，可以使用 Kubernetes 的资源监控工具，例如 kubectl top 命令或者 Prometheus 和 Grafana。 这些工具可以帮助企业了解 Pod 和节点的资源使用情况，并根据实际需求调整资源请求和限制。

此外，还可以使用自动伸缩功能，根据 Pod 的资源使用情况动态调整副本数量，这样可以确保集群中的资源得以有效利用。

2. 自动扩缩容

Kubernetes 默认提供三种自动扩缩容的方式，分别是：

- Horizontal Pod Autoscaler (HPA)：横向自动扩缩容
- Vertical Pod Autoscaler (VPA)：垂直自动扩缩容
- Cluster Autoscaler (CA)：集群节点自动扩缩容

HPA

HPA 是一种自动伸缩功能，用于根据实际负载动态调整 Pod 的副本数量。 HPA 会定期监控 Deployment、ReplicationController、ReplicaSet 或者 StatefulSet 中 Pod 的资源使用情况， 根据设定的规则动态调整副本数量。例如，可以设定当 CPU 使用率超过 80% 时，HPA 就会增加副本数量； 当 CPU 使用率低于 50% 时，HPA 就 会减少副本数量。

通过 HPA，可以有效地提高 Deployment、ReplicationController、ReplicaSet 或者 StatefulSet 中 Pod 的资源利用率。

VPA

与 HPA 不同，VPA 意味着将更多资源（例如：内存或 CPU）分配给正在运行中的 Pod。 VPA 通过监控 Pod 的资源使用情况，并根据设定的目标使用率自动调整容器的资源请求和限制。 例如，如果 Pod 的 CPU 使用率较低，VPA 就会减少容器的 CPU 请求和限制； 如果 Pod 的 CPU 使用率较高，VPA 就会增加容器的 CPU 请求和限制。

使用 VPA 有如下优势:

- Pod 资源用其所需，提升集群节点使用效率
- 不必运行基准测试任务来确定 CPU 和内存请求的合适值
- VPA 可以随时调整 CPU 和内存请求，无需人为操作，因此可以减少维护时间。

需要注意的是：VPA 目前还没有生产就绪，在使用之前需要了解资源调节对应用的影响。

CA

CA 也是 Kubernetes 中的一种自动伸缩功能，可以根据集群的资源使用情况动态调整节点数量。 CA 通过监控集群 的资源使用情况，并根据设定的阈值自动调整节点数量。 例如，如果集群的 CPU 使用率超过了阈值，CA 就会增加 节点数量； 如果集群的 CPU 使用率低于阈值，CA 就会减少节点数量。 通过 CA 可以有效地提高 Kubernetes 集群的资源利用率。 当集群的资源使用情况发生变化时，CA 可以自动调整节点数量，使得集群能够更好地利用可用的资源。

小结

通过使用弹性扩缩容，让业务能够通过日常请求的高低峰值，对服务器资源进行自动调整并高效利用资源， 这也是上云的本质能力。另外在容器中，弹性扩缩容一般分为容器层和 node 节点层，当业务负载的变化触发容器层的扩缩容， 容器的

扩缩容会使节点集群的可分配资源量变化，并发生 node 节点的扩缩容。 业务上线不仅不需要提前申请过多的资源，而且申请下来的冗余资源也可以按需使用。

3. 错峰调度

Kubernetes 错峰调度是一种策略，可通过设置 Pod 的调度策略来实现，用于控制在同一时间内运行的作业数量。 目的是避免在系统负载过高的情况下运行大量作业，并允许在系统负载较低的时间运行更多的作业。 这种方法可以有效地利用计算资源，并在集群中的节点之间均衡负载。 可以使用 Kubernetes 的调度程序或第三方调度程序来实现错峰调度。错峰调度通常用于以下场景：

- 在系统负载较低的时间运行更多的作业，以提高资源利用率。
- 避免在系统负载过高的时间运行大量作业，以免造成系统瘫痪。
- 在集群中的节点之间均衡负载，以提高系统可靠性。

举个例子，假设你有一个每天运行一次的定期作业，该作业在早上 8 点开始运行。 如果你使用错峰调度策略，则可以调整作业的开始时间，使其在早上 6 点或 10 点开始运行， 而不是在早上 8 点。这样就可以避免在早上 8 点运行大量作业时造成的系统负载过高的问题， 在集群的资源使用较低的时段调度这些 Pod，从而减少集群的资源使用峰值。

4. 在线离线混部

在线离线业务混合部署是指在一个 Kubernetes 集群中同时运行离线业务和在线业务：

- 在线业务：需要实时处理的业务，例如网站、游戏、移动应用等。这类业务通常对延迟要求较低，但是对资源要求较高。
- 离线业务：不需要实时处理的业务，例如批处理作业、数据分析和备份等。这类业务通常消耗较多资源，但是可以允许较长的延迟。

例如，某公司可能有一个网站，用来提供网上购物和物流服务。该服务对稳定性的要求较高， 因为在线网站的流畅性关乎用户的使用体验。为了确保网站的可用性和响应速度， 在部署时这类服务的 QoS 等级较高，在分配资源时享有更高优先

级。 同时网站的资源使用率与网站流量成正比，白天高峰，夜里低峰，有明显的流量潮汐特性。

同时该公司可能需要定期备份其客户数据库，并对过去一年的销售数据进行分析，以确定接下来的营销策略。 为了实现这个目标，公司可以使用 Kubernetes 集群来运行两个离线业务工作作业： 一个用于数据备份，一个用于数据分析。数据备份工作作业可以每天凌晨运行一次， 将客户数据库的数据备份到云存储中。数据分析工作作业可以每周末运行一次，对过去一年的销售数据进行分析，并生成报告。

由此可见在线服务的流量存在周期性，对延时敏感，资源使用率低，对稳定性、服务资源与负载有着很高的要求； 而离线作业通常可以容忍较高的时延，并且失败任务可重启。这两种类型的服务负载在分时间段使用、资源互补上存在极大的优化空间。 例如，可以为离线业务设置较低的资源请求和限制，并使用较低优先级的调度策略。 这样，当在线业务需要资源时，离线业务就会自动让出资源。通过在离线业务混合部署， 可以有效地利用集群中的资源，并同时保证在线业务的可用性和响应速度。

总结

Kubernetes 可以使用多种方式来提升资源利用率。通过设置 Pod 的资源请求和限制、使用自动伸缩功能、错峰调度策略以及混部技术， 皆可有效地优化 Kubernetes 集群的计算资源，更好地满足业务需求并节省成本。 当然，在实际应用中，我们还可以根据具体的业务场景进行更多的优化。 总的来说，要提升 Kubernetes 的资源利用率，只要对应用程序的运行方式和资源使用情况进行全面分析， 并使用适当的工具和方法来提升资源利用率，就能够让 Kubernetes 集群更加高效地运行。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

安装 Kubernetes 教程 (KLTS 版)

本文以 DaoCloud 自主维护的 KLTS (Kubernetes Long Term Support) 为例，简要介绍如何安装 Kubernetes。

准备工作

- 准备一台兼容的 Linux 主机。Kubernetes 项目为基于 Debian 和 Red Hat 的 Linux 发行版以及一些不提供包管理器的发行版提供通用的指令。
- 每台主机至少 2 GB 或更多的内存（如果内存太少将影响应用的运行）
- CPU 2 核或更多
- 集群中所有主机的网络连通（公网和内网）
- 单个节点上不能有重复的主机名、MAC 地址或 product_uuid，请参阅确保每个节点上 MAC 地址和 product_uuid 的唯一性
- 开启主机上的某些端口，请参阅检查所需端口。
- 禁用交换分区。为了保证 kubelet 正常工作，您必须禁用交换分区。

确保节点上 MAC 地址和 product_uuid 的唯一性

- 使用命令 ip link 或 ifconfig -a 来获取网络接口的 MAC 地址
- 使用 sudo cat /sys/class/dmi/id/product_uuid 命令来校验 product_uuid

一般来讲，硬件设备拥有唯一的地址，但是有些虚拟机的地址可能会重复。Kubernetes 使用 MAC 地址和 product_uuid 来确定集群中的唯一节点。 如果这些值在每个节点上不唯一，可能会导致安装失败。

检查网络适配器

如果您有一个以上的网络适配器，同时您的 Kubernetes 组件通过默认路由不可达， 我们建议您预先添加 IP 路由规则，这样 Kubernetes 集群就可以通过对应的适配器完成连接。

允许 iptables 检查桥接流量

确保 br_netfilter 模块被加载。这一操作可以通过运行 lsmod | grep br_netfilter 来完成。若要显式加载该模块，可执行命令 sudo modprobe br_netfilter。

为了让您的 Linux 节点上的 iptables 能够正确地查看桥接流量，您需要确保在 sysctl 配置中将 net.bridge.bridge-nf-call-iptables 设置为 1。例如：

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

更多细节请查阅网络插件需求页面。

检查所需端口

控制平面节点

协议	方向	端口范围	作用	使用者
TCP	入站	6443	Kubernetes API 服务器	所有组件
TCP	入站	2379-2380	etcd 服务器客户端 API	kube-apiserver、etcd
TCP	入站	10250	Kubelet API	kubelet 自身、控制平面组件

协议	方向	端口范围	作用	使用者
TCP	入站	10251	kube-scheduler	kube-scheduler 自身
TCP	入站	10252	kube-controller-manager	kube-controller-manager 自身

工作节点

协议	方向	端口范围	作用	使用者
TCP	入站	10250	Kubelet API	kubelet 自身、控制平面组件
TCP	入站	30000-32767	NodePort 服务	所有组件

以上是 NodePort 服务的默认端口范围。

使用 * 标记的任意端口号都可以被覆盖，所以您需要保证定制的端口是开放的。

虽然控制平面节点已经包含了 etcd 的端口，您也可以使用自定义的外部 etcd 集群，或指定自定义端口。

您使用的 Pod 网络插件（见下）也可能需要某些特定端口开启。由于各个 Pod 网络插件都有所不同，请参阅相应文档中的端口要求。

设置节点名字

命令的语法格式如下：

```
hostnamectl set-hostname your-new-host-name
echo "127.0.0.1 $(hostname)" >> /etc/hosts
echo "::1       $(hostname)" >> /etc/hosts
```

关闭 Swap

执行以下命令关闭 Swap：

```
swapoff -a
```

如果需要永久关闭，请编辑 /etc/fstab 文件，注释掉 Swap 的挂载路径。

关闭 Selinux

执行以下命令关闭 Selinux：

```
setenforce 0
```

如果需要永久关闭，请编辑 /etc/sysconfig/selinux 将 SELINUX=enforcing 替换为 SELINUX=disabled。

安装运行时

为了在 Pod 中运行容器，Kubernetes 使用容器运行时（Container Runtime）。

如果是 Linux 节点

默认情况下，Kubernetes 使用容器运行时接口（Container Runtime Interface，CRI）来与您所选择的容器运行时交互。

如果您不指定运行时，则 kubeadm 会自动尝试检测系统上已经安装的运行时，方法是扫描一组众所周知的 Unix 域套接字。

下面的表格列举了一些容器运行时及其对应的套接字路径：

运行时	域套接字
Docker	/var/run/dockershim.sock
Containerd	/run/containerd/containerd.sock

运行时　　域套接字

CRI-O　　/var/run/crio/crio.sock

如果同时检测到 Docker 和 Containerd，则优先选择 Docker。 这是必然的，即使您仅安装了 Docker，因为 Docker 18.09 附带了 Containerd，所以两者都是可以检测到的。 如果检测到其他两个或多个运行时，则 kubeadm 输出错误信息并退出。

kubelet 通过内置的 dockershim CRI 实现与 Docker 集成。

对于 Docker

===“基于 Red Hat 发行版”

执行以下命令安装基于 Red Hat 发行版的 Docker：

```bash
yum install docker
```

===“基于 Debian 发行版”

执行以下命令安装基于 Debian 发行版的 Docker：

```bash
apt-get install docker.io
```

对于 containerd

containerd 官方默认只提供 amd64 架构的下载包，如果您采用的是其他基础架构， 可以从 Docker 官方仓库安装 containerd.io 软件包。在安装 Docker 引擎中 找到为各自的 Linux 发行版设置 Docker 存储库和安装 containerd.io 软件包的有关说明。

也可以使用以下源代码构建。

```
VERSION=1.5.4
wget -c https://github.com/containerd/containerd/releases/download/v${VERSION}/containerd-${VERSION}-linux-amd64.tar.gz
tar xvf containerd-${VERSION}-linux-amd64.tar.gz -C /usr/local/
mkdir /etc/containerd/ && containerd config default > /etc/containerd/config.toml
wget -c -O /etc/systemd/system/containerd.service https://raw.githubusercontent.com/containerd/containerd/main/containerd.service
systemctl start containerd && systemctl enable containerd
```

如果是其它操作系统

默认情况下，kubeadm 使用 docker 作为容器运行时。kubelet 通过内置的 dockershim CRI 实现与 Docker 集成。

对于 Docker

===“基于 Red Hat 发行版”

执行以下命令安装基于 Red Hat 发行版的 Docker：

```bash
yum install docker
```

===“基于 Debian 发行版”

执行以下命令安装基于 Debian 发行版的 Docker：

```bash
apt-get install docker.io
```

对于 containerd

containerd 官方默认只提供 amd64 架构的下载包，如果您采用的是其他基础架构， 可以从 Docker 官方仓库安装 containerd.io 软件包。在安装 Docker 引擎中 找到为各自的 Linux 发行版设置 Docker 存储库和安装 containerd.io 软件包的有关说明。

也可以使用以下源代码构建。

```
VERSION=1.5.4
wget -c https://github.com/containerd/containerd/releases/download/v${VERSION}/containerd-${VERSION}-linux-amd64.tar.gz
tar xvf containerd-${VERSION}-linux-amd64.tar.gz -C /usr/local/
mkdir /etc/containerd/ && containerd config default > /etc/containerd/config.toml
wget -c -O /etc/systemd/system/containerd.service https://raw.githubusercontent.com/containerd/containerd/main/containerd.service
systemctl start containerd && systemctl enable containerd
```

参阅容器运行时以了解更多信息。

安装 KLTS

KLTS 提供了基于 deb 和 rpm 软件源的安装方式，您可以选择适合的安装方式。

设置 KLTS 软件源

===“基于 Red Hat 的发行版”

执行以下代码设置下载 KLTS 的软件源：

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://raw.githubusercontent.com/klts-io/kubernetes-lts/rpm-v${VERSIO
```

N}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://raw.githubusercontent.com/klts-io/others/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

===“基于 Debian 的发行版”

执行以下代码设置下载 KLTS 的软件源：

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://raw.githubusercontent.com/klts-io/kubernetes-lts/deb-v
${VERSION} stable main
deb [trusted=yes] https://raw.githubusercontent.com/klts-io/others/deb stable m
ain
EOF

apt-get update
```

===“基于 Red Hat 的发行版，国内加速 ��”

!!! note
```

以下加速均来自第三方，安全和稳定性不做保障，仅建议测试环境使用!!!

执行以下代码设置下载 KLTS 的软件源：

=== "/etc/hosts"

```bash
curl https://raw.githubusercontent.com/wzshiming/github-hosts/master/hosts >>/etc/hosts

VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://raw.githubusercontent.com/klts-io/kubernetes-lts/rpm-v${VERSION}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://raw.githubusercontent.com/klts-io/others/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

=== "hub.fastgit.org"

```bash
```

VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://hub.fastgit.org/klts-io/kubernetes-lts/raw/rpm-v${VERSION}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://hub.fastgit.org/klts-io/others/raw/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

=== "ghproxy.com"

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://ghproxy.com/https://raw.githubusercontent.com/klts-io/kubernetes-lts/rpm-v${VERSION}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://ghproxy.com/https://raw.githubusercontent.com/klts-io/others
```
```

/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

=== "raw.githubusercontents.com"

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://raw.githubusercontents.com/klts-io/kubernetes-lts/rpm-v${VERSION}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://raw.githubusercontents.com/klts-io/others/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

=== "raw.staticdn.net"
```

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/yum.repos.d/klts.repo
[klts]
name=klts
baseurl=https://raw.staticdn.net/klts-io/kubernetes-lts/rpm-v${VERSION}/\$basearch/
enabled=1
gpgcheck=0
[klts-other]
name=klts-others
baseurl=https://raw.staticdn.net/klts-io/others/rpm/\$basearch/
enabled=1
gpgcheck=0
EOF

yum makecache
```

===“基于 Debian 的发行版，国内加速 ��”

!!! note

以下加速均来自第三方，安全和稳定性不做保障，仅建议测试环境使用!!!

执行以下代码设置下载 KLTS 的软件源：

=== "/etc/hosts"

```bash
curl https://raw.githubusercontent.com/wzshiming/github-hosts/master/hosts >>/etc/hosts
```

```
VERSION=1.18.20-lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://raw.githubusercontent.com/klts-io/kubernetes-lts/deb-v${VERSION} stable main
deb [trusted=yes] https://raw.githubusercontent.com/klts-io/others/deb stable main
EOF

apt-get update
```

=== "hub.fastgit.org"

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://hub.fastgit.org/klts-io/kubernetes-lts/raw/deb-v${VERSION} stable main
deb [trusted=yes] https://hub.fastgit.org/klts-io/others/raw/deb stable main
EOF

apt-get update
```

=== "ghproxy.com"

```bash
VERSION=1.18.20-lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://ghproxy.com/https://raw.githubusercontent.com/klts
```

–io/kubernetes–lts/deb–v${VERSION} stable main
deb [trusted=yes] https://ghproxy.com/https://raw.githubusercontent.com/klts
–io/others/deb stable main
EOF

apt–get update
```

=== "raw.githubusercontents.com"

```bash
VERSION=1.18.20–lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://raw.githubusercontents.com/klts–io/kubernetes–lts/
deb–v${VERSION} stable main
deb [trusted=yes] https://raw.githubusercontents.com/klts–io/others/deb sta
ble main
EOF

apt–get update
```

=== "raw.staticdn.net"

```bash
VERSION=1.18.20–lts.2
cat << EOF > /etc/apt/sources.list.d/klts.list
deb [trusted=yes] https://raw.staticdn.net/klts–io/kubernetes–lts/deb–v${VER
SION} stable main
deb [trusted=yes] https://raw.staticdn.net/klts–io/kubernetes–lts/deb stable
main

EOF

apt-get update
```

开始安装 KLTS

===“基于 Red Hat 的发行版”

执行以下命令安装：

```bash
yum install kubeadm kubelet kubectl
```

===“基于 Debian 的发行版”

执行以下命令安装：

```bash
apt-get install kubeadm kubelet kubectl
```

开机自动启动 Kubelet

执行以下命令开机自动启动 Kubelet：

systemctl enable kubelet

拉取依赖镜像

===“默认”

执行以下命令 pull 依赖的镜像：

```bash
VERSION=1.18.20-lts.2
REPOS=ghcr.io/klts-io/kubernetes-lts
```
```

kubeadm config images pull --image-repository ${REPOS} --kubernetes-version v${VERSION}
```

===“国内加速 ”

执行以下命令 pull 依赖的镜像：

```bash
VERSION=1.18.20-lts.2
REPOS=ghcr.m.daocloud.io/klts-io/kubernetes-lts
kubeadm config images pull --image-repository ${REPOS} --kubernetes-version v${VERSION}
```

后续对 kubeadm 的操作都需要加上 --image-repository 和 --kubernetes-version 以主动指定镜像。

初始化控制面节点

===“默认”

执行以下命令初始化控制面的节点：

```bash
VERSION=1.18.20-lts.2
REPOS=ghcr.io/klts-io/kubernetes-lts
kubeadm init --image-repository ${REPOS} --kubernetes-version v${VERSION}
```

===“国内加速 ”

执行以下命令初始化控制面的节点：

```bash
VERSION=1.18.20-lts.2
```
```

```
REPOS=ghcr.m.daocloud.io/klts-io/kubernetes-lts
kubeadm init --image-repository ${REPOS} --kubernetes-version v${VERSION}
```

脚本一键安装

除了上述正常安装方式外，KLTS 还支持脚本自动完成安装流程。

```
wget https://github.com/klts-io/klts/raw/main/install.sh
chmod +x install.sh
./install.sh

Usage: ./install.sh [OPTIONS]
  -h, --help : Display this help and exit
  --kubernetes-container-registry=ghcr.io/klts-io/kubernetes-lts : Kubernetes container registry
  --kubernetes-version=1.18.20-lts.1 : Kubernetes version to install
  --containerd-version=1.3.10-lts.0 : Containerd version to install
  --runc-version=1.0.2-lts.0 : Runc version to install
  --kubernetes-rpm-source=https://github.com/klts-io/kubernetes-lts/raw/rpm-v1.18.20-lts.2 : Kubernetes RPM source
  --containerd-rpm-source=https://github.com/klts-io/containerd-lts/raw/rpm-v1.3.10-lts.0 : Containerd RPM source
  --runc-rpm-source=https://github.com/klts-io/runc-lts/raw/rpm-v1.0.2-lts.0 : Runc RPM source
  --others-rpm-source=https://github.com/klts-io/others/raw/rpm : Other RPM source
  --kubernetes-deb-source=https://github.com/klts-io/kubernetes-lts/raw/deb-v1.18.20-lts.2 : Kubernetes DEB source
  --containerd-deb-source=https://github.com/klts-io/containerd-lts/raw/deb-v1.3.10-lts.0 : Containerd DEB source
  --runc-deb-source=https://github.com/klts-io/runc-lts/raw/deb-v1.0.2-lts.0 : Runc DEB source
  --others-deb-source=https://github.com/klts-io/others/raw/deb : Other DEB
```

source

--focus=enable-iptables-discover-bridged-traffic,disable-swap,disable-selinux,setup-source,install-kubernetes,install-containerd,install-runc,install-crictl,install-cniplugins,setup-crictl-config,setup-containerd-cni-config,setup-kubelet-config,setup-containerd-config,daemon-reload,start-containerd,status-containerd,enable-containerd,start-kubelet,status-kubelet,enable-kubelet,images-pull,control-plane-init,status-nodes,show-join-command : Focus on specific step

--skip='' : Skip on specific step

更多安装方式

- 保姆式部署三节点 K8s 集群
- 使用 kubeadm 创建集群
- 使用 kOps 安装 K8s
- 使用 Kubespray 安装 K8s

边缘原生应用准则白皮书

发布日期：2023 年 1 月 17 日（第一版），2022 年 10 月 24 日（草稿）

原文链接：边缘原生应用准则白皮书

目标

“边缘原生”一词已在行业博文等许多地方被提及，比如 Gartner、 Macrometa、FutureCIO。 而 State of the Edge 和 Linux Foundation (LF) 这些组织也在讨论边缘原生应用，但还没有关注边缘原生应用准则。

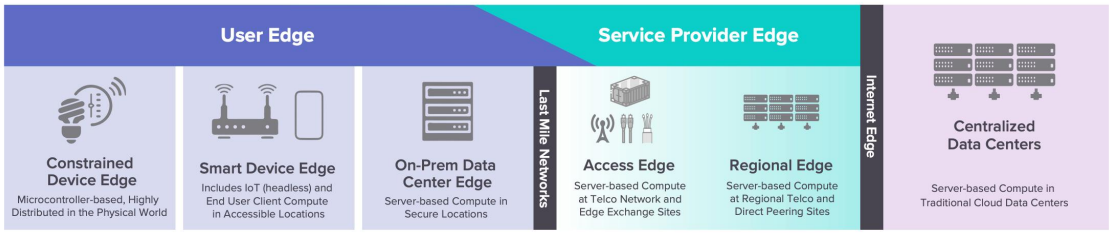
本白皮书侧重于边缘原生应用以及如何定义这些应用准则。

什么是边缘？

边缘计算使数据处理更接近源头，例如在工厂中控制机器人。未来五年，边缘计算将变得更加普遍， 该行业预计从 2022 年到 2030 年将增长 38.9%。 很多公司正在看到以下将计算能力放在边缘的好处:

- 减少延迟
- 带宽管理
- 增强隐私数据的安全
- 不可靠网络下的稳定运行

有多种边缘计算定义存在，但本文将重点关注基于数据处理所处的地理位置的边缘计算。 基于地理位置的边缘被分类为多个类别，具体取决于与用户的距离。下图显示了根据 Linux 基金会边缘白皮书 定义的类别。



edgetype

边缘原生准则与云原生准则存在许多相似之处，但也有一些关键的区别。

云原生 vs 边缘原生

根据 云原生基金会(CNCF) 的定义，云原生技术是：

> **“云原生技术赋予组织在现代动态环境中构建和运行可扩展的应用的能力，例如公有云、私有云和混合云。容器、服务网格、微服务、不可变基础架构和声明式 API 是这种方法的典范。**
>
> **这些技术使得松耦合的系统具有弹性、可管理和可观测性。结合强大的自动化，它们允许工程师频繁且可预测的进行高影响的更改，最大限度的减少繁琐工作。“**

这一广泛的使命对于边缘应用仍然适用，因为开放边缘计算术语表指出，“边缘原生应用”参考了云原生准则：

> **“边缘原生应用是利用边缘计算能力构建的应用，不适合在中心云运行。边缘原生应用参考云原生准则，同时需要考虑边缘的独特特性，例如资源限制、安全性、延迟和自治性。边缘原生应用是利用云计算能力，并能与上游资源协同工作的方式构建的。不关心集中式资源管理、远程管理、编排、 CI/CD 的边缘应用并不真正“原生”，而更类似于传统的本地应用。”**

随着云原生用例涉及传统云以外的边缘位置的数据和事件，新的工具和技术正在不断发展，以实现松耦合的系统，具有弹性、可管控和可观测性，同时管理边缘的独特性。

边缘原生与云原生的相似性

边缘原生与云原生有许多相似之处，本节将描述这些相似之处。

属性	云原生 与 边缘原生
应用和服务的可移植性	应用和服务将它们与基础设施的耦合分离。一个良好编写的应用不需要知道它运行的位置，可以支持在平台之间移植。
可观测性	平台配备了一套良好的文档接口和工具选项，以便检测问题和收集指标。这使得开发人员可以构建具有弹性和高效管理的系统。
可管理性	提供接口和工具选项以规模化管理应用和资源。平台还具有插件机制，以提供基础网络连接、服务和管理功能。
支持多种语言和框架	应用和服务可以使用各种流行的语言和框架实现。

边缘原生与云原生的区别

边缘原生和云原生的广泛使命有相似之处，但开发人员应该意识到不同之处。

属性	云原生	边缘原生
应用模型	大多数微服务组件是支持水平扩展的无状态服务。	虽然服务提供的边缘应用非常相似，但用户边缘应用可能是单独的单体程序；在这些情况下，状态与应用相关联。
数据模型	常见的是支持无状态组件的集中式模型。	常常采用缓存、流式处理、实时和分布式模型。
弹性	快速启动和关闭；通常将底层资源视为无限制。	由于边缘设备的硬件资源受限，弹性受到限制；如果需要更多资源，会通过请求云端进行“垂直”扩展。
稳定性	将稳定性外包给云提供商，使用分布在不	通常依赖于经过强化的基础

属性	云原生	边缘原生
	同地域的冗余节点。	设施，具有面向有状态组件的恢复架构；在许多情况下，稳定性可能比云上的稳定性低。
规模	通常限于少数区域和实例	可以支持大规模区域（高达数万个），支持大量外部设备（高达数十万个）
编排	大型公有或私有云中的编排旨在通过在集中池化的主机上运行工作负载（以水平方式调度）来实现效率和可用性。	边缘是分散的，工作负载以分布式方式部署，通常以指定区域的方式调度。
管控	虽然云原生和边缘原生都是可管控的，但机制有所不同；云原生依赖于集中管控和自动化。	边缘原生需要远程和集中管控的混合方式以及硬件和软件的零接触部署(ZTP)。边缘的运维人员可能没有接受过培训，人数很少，甚至不存在。升级流程需要具有原子性和一致性，防止设备升级失败导致不可用。
网络	应用可以依赖高速网络。	应用需要考虑各种网速（不稳定、比较差、非常好）和功能。包括基于移动和无线的，集成来自非 IP 协议网络的数据和事件。
安全	安全管控的基础设施。	不可信不安全环境中的“零信任”。
硬件配置	很少需要关注硬件配置，能适用于大多数应用。	应用可能具有更高的实时要求，对硬件平台、位置和安

属性	云原生	边缘原生
		全意识有要求。开发人员需要了解更广泛的硬件和接口。
与外部资源交互	应用很少需要与本地硬件资源交互。	在边缘部署的服务通常需要与本地环境交互：相机、传感器、执行器、用户等。

边缘原生应用

边缘原生应用是为边缘设计的应用和服务。它们参考以上相似性和差异性编写。以下是这些应用的核心准则。

边缘原生准则

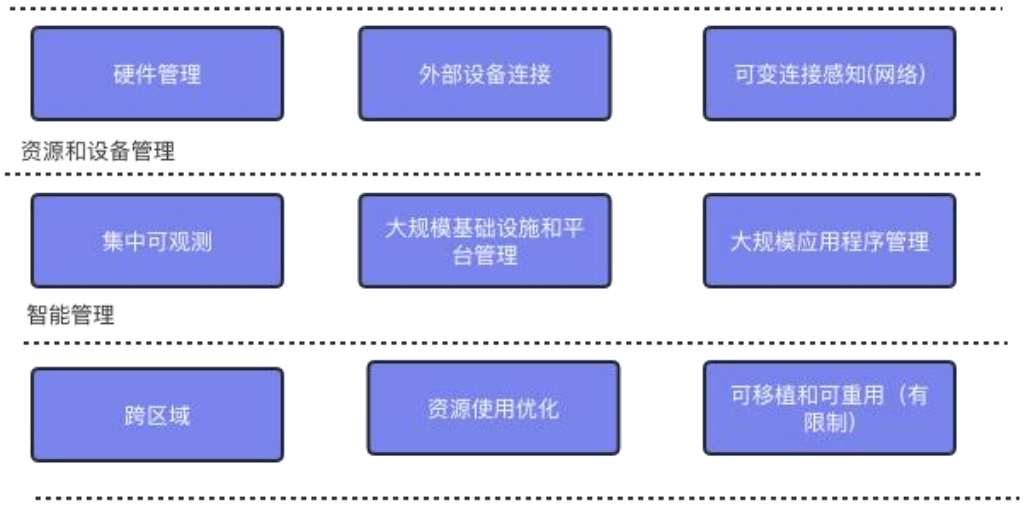
为了实现本文前面提到的边缘原生的使命，边缘原生应用应遵循以下准则。

准则	描述
硬件管理能力	开发人员需要了解广泛的硬件平台和接口，而不是只有同质化的硬件平台。
外部设备连接	应用必须知道如何连接其环境中的设备，并了解运行时的功能变化。例如，它们在初始化配置后，能响应传感器连接/断开或新设备接入的情况。功能并不固定，需要考虑应用的环境，因此编排器需要能够协调应用状态和功能变化。
可变连接感知	应用必须适应不可靠甚至无法使用（完全隔离）的网络连接，使用异步通信、排队和缓存等机制。当边缘从中心站点获取配置时，可能需要使用“拉取”机制来克服规模、网络连接和安全问题。
集中可观测	虽然边缘和云原生应用都需要集中可观测，但边缘原生应用具有独特的考虑因素。边缘原生应用可能被部

准则	描述
	署到大规模实例中，存在运维人员和现场支持受限的情况。因此，需要采用数据分布式收集和集中聚合、开放式环路（人员可观察/可操作）和闭合环路(机器自动化)等技术。可观测性包括指标、日志、数字孪生、警报（事件和警报）和健康监测。
大规模基础设施和平台管理	基础设施和平台的管理在大规模边缘应用上非常重要，需要支持声明式的管理。此外，可能存在一些特殊的要求，例如设备接入、横向扩展限制、管理裸机环境等。在平台层面，部署或管理 Kubernetes 或虚拟化层以及各种插件也是一个问题；需要保持平台层面的供应商中立，以实现应用的可移植性。
大规模应用管理	应用的数量和这些应用的实例数量在边缘可能非常大，需要基于声明式的规则和条件来配置、通过自动化服务生效，以及跨多个应用实例的聚合管理视图。应用也可能有实时需求，这意味着应用和基础设施平台之间的联系（例如使用 GPU、DPU、FPGA、CPU 架构、内核优化、Kubernetes 插件）可能比云应用更紧密。换句话说，应用编排可能触发底层的基础设施和平台编排。
跨区域	应用不止部署在一个区域，存在跨区域的延迟和故障。事实上，边缘应用也可能跨公有云、私有云。
资源使用优化	由于边缘计算资源受限，应用必须持续优化资源使用。按需调整应用，基于部署位置和可用性意图的迁移和伸缩。这意味着在一天的时间里存在不同的运行工作负载。
应用可移植和可重用（有限制）	抽象层试图通过供应商中立的 PaaS 提供与基础设施和平台无关的可移植性。但由于本地资源、硬件平

准则	描述
	台、安全、移动网络等限制，配置选项需要适应本地差异。

边缘原生准则分组



edgegroup

这九项准则可以归纳为一组较小的五项准则。硬件管理、外部设备连接、可变连接感知(网络)都可以在更广泛的资源和硬件管理准则下考虑。 同样，边缘应用可大规模管理、可集中观测以及具有可管理的基础设施和平台，这些都可以归类为大规模管理的准则。 以下是扩展的五个准则：跨区域、资源使用优化、可移植和可重用限制、资源和硬件管理以及规模管理。

结论和后续步骤

本文为第一版，可能会进行修订。后面会有一些与本文子内容相关的论文。

如何参与

CNCF IoT Edge 工作组有定期会议、邮件列表和 Slack。有关最新信息，请参阅工作组 GitHub 页面的通信部分。 我们欢迎读者参与，介绍 Edge 相关项目，为小组的工作领域提出想法，或帮助修改本白皮书和起草后续文件。

边缘原生开源项目和计划的工作列表

作为本文的一部分，CNCF IoT Edge 工作组正在收集开源项目的工作列表，这些项目可帮助应用开发人员实现本文中概述的边缘原生应用准则。

可以在此电子表格中或通过二维码找到该列表。 要获得添加项目的编辑权限，请加入 IoT Edge Working Group Google 群组。



边缘原生项目列表的二维码

贡献者

作者

- Amar Kapadia, Aarna Networks
- Brandon Wick, Aarna Networks
- Joel Roberts, Cisco
- Kate Goldenring, Fermyon
- Dejan Bosanac, Red Hat
- Tomoya Fujita, Sony US Lab
- Ravi Chunduru, Verizon
- Natalie Fisher, VMware
- Steven Wong, VMware

审稿人

- Frédéric Desbiens, Eclipse Foundation

- Prakash Ramchandran, eOTF
- Mark Abrams, SUSE

参考资料

Linux 基金会边缘白皮书： https://www.lfedge.org/wp-content/uploads/2020/07/LFedge_Whitepaper.pdf

开放边缘计算术语表 [v2.1.0] State of the Edge： https://github.com/State-of-the-Edge/glossary/blob/master/edge-glossary.md#edge-native-application

云原生组织(CNCF)章程：
https://github.com/cncf/foundation/blob/main/charter.md

Gartner “云原生不等于边缘原生”：
https://blogs.gartner.com/thomas_bittman/2020/04/17/cloud-native-isnt-edge-native/

Macrometa “边缘原生非云原生”： https://www.macrometa.com/blog/edge-native-is-not-cloud-native

Future CIO “云原生和边缘原生区别”： https://futurecio.tech/cloud-native-versus-edge-native-know-the-difference/

《边缘计算市场规模、份额和趋势分析报告》，按组件（硬件、软件、服务、边缘管理平台）、应用、行业垂直、地区和细分预测，2022 年至 2030 年：
https://www.grandviewresearch.com/industry-analysis/edge-computing-market

2023-08-15

Kubernetes 1.28 震撼发布，Sidecar Containers 迎面而来

作者：carlory



1.28 planternetes

太平洋时间 2023 年 8 月 15 日，Kubernetes 1.28 正式发布。此版本距离上版本发布时隔 4 个月，是 2023 年的第二个版本。

新版本中 release 团队跟踪了 46 个 enhancements。其中 12 个功能升级为稳定版，14 个已有功能进行优化升级为 Beta， 另有 20 个 Alpha 级别的功能，大多数为全新功能。1.28 版本包含了很多重要功能以及用户体验优化，接下来将详细介绍部分重要功能。

重要功能

[架构] 版本偏差策略扩展至 3 个版本

从 1.28 控制平面/1.25 节点开始，Kubernetes 版本偏差策略将支持的控制平面/节点偏差扩展到 3 个版本。 这使得节点的年度次要版本升级成为可能，同时保持受支持的次要版本。

更多信息，请访问 https://kubernetes.io/releases/version-skew-policy/

[API Machinery] KEP-2876: CRD 基于通用表达式语言（CEL）的验证规则 Beta

CRD 基于通用表达式语言（CEL）的验证规则，在 v1.25 中就已经升级为 Beta。通过将 CEL 表达式直接集成在 CRD 中， 可以是开发者在不使用 Webhook 的情况下解决大部分对 CR 实例进行验证的用例。在未来的版本，我们将继续扩展 CEL 表达式的功能，以支持默认值和 CRD 转换。

此特性使用 x-kubernetes-validations 扩展包含在 CustomResourceDefinition 模式定义中。 在 v1.27 中，ValidationRule 新增了可选字段 messageExpression 提供可读性的失败信息。 在 v1.28 中添加可选字段“reason”和“fieldPath”，允许用户在验证失败时指定原因和字段路径。

```
x-kubernetes-validations:
- rule: "self.x <= self.maxLimit"
  messageExpression: '"x exceeded max limit of " + string(self.maxLimit)'
  reason: "xExceededMaxLimit"
  fieldPath: "spec.x"
```

[API Machinery] KEP-3488: 基于通用表达式语言（CEL）的准入控制 Beta

基于通用表达式语言（CEL）的准入控制是可定制的，对于 kube-apiserver 接受到的请求，可以使用 CEL 表达式来决定是否接受或拒绝请求，可作为 Webhook 准入控制的一种替代方案。在 v1.28 中，CEL 准入控制被升级为 Beta，同时添加了一些新功能，包括但不限于：

1. ValidatingAdmissionPolicy 类型检查现在可以正确处理 CEL 表达式中的“authorizer”变量。
2. ValidatingAdmissionPolicy 支持对 messageExpression 字段进行类型检查。

3. 在接受单个请求的过程中，ValidatingAdmissionPolicy 准入插件将对每个唯一的授权者表达式执行不超过一次的授权检查。 对相同授权者表达式的所有评估将产生相同的决策。

```
# 例子
expression: authorizer.path('/healthz').check('GET').allowed()
messageExpression: authorizer.path('/healthz').check('GET').reason()
```

4. kube-controller-manager 组件新增 ValidatingAdmissionPolicy 控制器，用来对 ValidatingAdmissionPolicy 中的 CEL 表达式做类型检查，并将原因保存在状态字段中。

5. 支持需要权限检查的准入控制用例，通过使用“authorizer”变量及其方法来，可以实现以下场景（包括但不限于):

   - 验证只有具有特定权限的用户才能设置特定字段。
   - 验证只有负责 finalizer 的控制器才能将其从 finalizers 字段中删除。

6. 支持变量组合，可以在 ValidatingAdmissionPolicy 中定义变量，然后在定义其他变量时使用它。

7. 新增 CEL 库函数支持对 Kubernetes 的 resource.Quantity 类型进行解析。

[API Machinery] KEP-3488: 基于通用表达式语言 (CEL) 的准入 Webhook 匹配条件 Beta

将 CEL 表达式过滤器引入 Webhook，以允许请求打向 Webhook 的范围变得更小。向准入 Webhook 添加“匹配条件”， 作为定义 Webhook 范围的现有规则的扩展。matchCondition 是一个 CEL 表达式，必须评估为 true 才能将准入请求发送到 Webhook。 如果 matchCondition 的计算结果为 false，则将跳过该请求的 Webhook（隐式允许）。

ValidatingAdmissionPolicy 是一项令人兴奋的新功能，我们希望它能够大大减少对准入 Webhook 的需求 但它并不是有意尝试涵盖所有可能的用例，而是旨在改善那些无法迁移的 webhooks 的情况。

[API Machinery] KEP-4020: Kube APIServer 混合版本互操作代理 Alpha

当集群具有多个混合版本的 apiserver 组件时（例如在升级/降级期间或运行时配置更改和部署回滚发生时）， 并非每个 apiserver 都可以为每个版本的每个资源提供服务。

为了解决这个问题，过滤器被添加到聚合器的处理程序链中，聚合器将客户端代理到能够处理其请求的 apiserver 组件。

当在集群上执行升级或降级时，在一段时间内，apiserver 处于不同的版本，并且能够提供不同的内置资源集 （不同的组、版本和资源都是可能的）。

[API Machinery] KEP-4008: CRD Validation Ratcheting Alpha

如果 Patch 请求没有更改任何不合法的字段，这将允许 CR 验证失败。将验证逻辑从控制器转移到前端的能力， 是提高 Kubernetes 项目可用性的长期目标。

[API Machinery] KEP-4080: 新增通用控制平面存储库 Alpha

使 kube-apiserver 构建在一个新的临时存储库上，该存储库使用 k8s.io/apiserver， 但具有更大、精心选择的 kube-apiserver 功能子集，以便可重用。

这个过程是渐进的：我们将从一个新的存储库开始，它不会向 k8s.io/apiserver 添加任何内容， 然后逐步将通用功能从 kube-apiserver 移动到它下面。新的存储库被命名为 k8s.io/generic-controlplane。

节点 KEP-4009: 设备插件 API 添加对 CDI 标准设备的支持 Alpha

CDI 提供了一种将复杂设备注入容器的标准化方法（即逻辑上需要注入多个 /dev 节点才能工作的设备）。 这项新功能使插件开发人员能够利用 1.27 中添加到 CRI 的 CDIDevices 字段将 CDI 设备直接传递到支持 CDI 的运行时（其中 containerd 和 crio-o 是最新版本）。

有关 CDI 本身的更多信息，请访问 https://github.com/container-orchestrated-devices/container-device-interface

节点 KEP-753: 原生支持 Sidecar 容器 Alpha

它为 init 容器引入了 restartPolicy 字段，并使用这个字段来指示 init 容器是 sidecar 容器。Kubelet 将按照 restartPolicy=Always 的顺序与其他 init 容器一起启动 init 容器，但它不会等待其完成，而是等待容器启动完成。

启动完成的条件是启动探测成功（或者未定义启动探测）并且 postStart 处理程序完成。 此条件用 ContainerStatus 类型的字段 Started 表示。有关选择此信号的注意事项，请参阅“Pod 启动完成条件”部分。

字段 restartPolicy 仅在 init 容器上被接受。现在唯一支持的值是“Always”。不会定义其他值。此外，该字段可为空，因此默认值为“无值”。容器的 restartPolicy 的其他值将不被接受，容器将遵循当前实现的逻辑。

Sidecar 容器不会阻止 Pod 完成 – 如果所有常规容器都已完成，Sidecar 容器将被终止。 在 sidecar 启动阶段，重启行为将类似于 init 容器。如果 Pod restartPolicy 为 Never， 则启动期间失败的 sidecar 容器将不会重新启动，整个 Pod 将失败。如果 Pod restartPolicy 为 Always 或 OnFailure， 则会重新启动。一旦 sidecar 容器启动（postStart 完成且启动探测成功），即使 Pod restartPolicy 为 Never 或 OnFailure， 这些容器也会重新启动。此外，即使在 Pod 终止期间，sidecar 容器也会重新启动。

为了最大限度地减少 sidecar 容器的 OOM 杀死，这些容器的 OOM 调整将匹配或超过 Pod 中常规容器的 OOM 分数调整。

其他 init 容器不同，Sidecar 容器还支持以下配置：

- Sidecar 容器的 PostStart 和 PreStop 生命周期处理程序
- 所有探针（启动、就绪、活跃）

Sidecar 的就绪探测将有助于确定整个 Pod 的就绪情况。

节点 KEP-2400: 节点 Swap 内存支持 Beta

在 Kubernetes 1.22 之前，节点不支持使用内存交换，如果在节点上检测到交换，kubelet 将默认无法启动。 每个系统管理员或 Kubernetes 用户在设置和使用 Kubernetes 方面都采用同一种方式：禁用交换空间。

从 1.22 开始，可以在每个节点上启用内存交换 Alpha 功能支持。此更改允许管理员选择在 Linux 节点上配置交换， 将块存储的一部分视为额外的虚拟内存。要在节点上启用交换，必须在 kubelet 上启用 NodeSwap 功能门， 并且 —fail-swap-on 命令行标志或 failSwapOn 配置设置必须设置为 false。

在 1.28 版本，该功能被升级为 Beta，功能门默认为关闭，该功能仅支持应用于 Bustable 类型的容器组且仅支持 Cgroup v2 (对 v1 的支持将被删除)。对于 BestEffort 类型的容器组，在存在节点压力的情况下应该被驱逐，Swap 交换区将被禁用。

Burstable 类型的容器组的 Swap 限制的公式为：(/)*。 此外，内存限制等于内存请求的容器也无法访问 Swap 交换区。

[Instrumentation] KEP-3498: 指标静态分析扩展机制 GA

静态分析扩展机制升级到指标稳定性框架，它包括对 BETA 级指标的支持和指标文档的自动生成。 有关各个稳定性级别（以及指标本身）的更多信息，请参见 https://kubernetes.io/docs/reference/instrumentation/metrics/

Apps KEP-3939: Job API 新增 Pod 替换策略支持 Alpha

Kubernetes 1.28 为 Job API 添加了一个新字段，允许用户指定是否希望控制平面在前一个 Pod 开始终止时立即创建新 Pod（现有行为），或者仅在现有 Pod 完全终止时创建新 Pod（新的、 可选行为）。

许多常见的机器学习框架（例如 Tensorflow 和 JAX）需要每个索引都有唯一的 Pod。在旧的行为中， 如果属于索引作业的 Pod 进入终止状态（由于抢占、驱逐或其他外部因素），则会创建一个替换 Pod，但由于与旧 Pod 发生冲突， 该 Pod 立即无法启动 尚未关闭。

在前一个 Pod 完全终止之前出现替换 Pod 也可能会导致资源稀缺或预算紧张的集群出现问题。 这些资源可能很难获取，因此可能需要很长时间才能找到资源，并且只有在现有 Pod 终止后才能找到节点。 如果启用集群自动缩放程序，过早创建替换 Pod 可能会产生不需要的扩展。

想要了解更多信息，请访问 https://kubernetes.io/docs/concepts/workloads/controllers/job/#delayed-creation-of-replacement-pods

Apps KEP-3850: 每个索引的 Job 重试 BackOff 限制 Alpha

目前，索引作业的索引共享单个回退限制。当作业达到此共享退避限制时，作业控制器将整个作业标记为失败， 并清理资源，包括尚未运行完成的索引。因此，现有的实现并没有涵盖工作量真正达到要求的情况。 例如，如果将索引作业用作一组长时间运行的集成测试的基础，则每次测试运行将只能发现单个测试失败。

embarrassingly parallel (https://en.wikipedia.org/wiki/Embarrassingly_parallel)： 每个索引完全独立于其他指标。在 1.28 中，新增 backoffLimitPerIndex 等字段，扩展 Job API 以支持索引作业， 其中回退限制是针对每个索引的，并且尽管某些索引失败，作业仍可以继续执行。有关更多信息， 请阅读 Kubernetes 文档中的处理 Pod 和容器故障: https://kubernetes.io/docs/concepts/workloads/controllers/job/#handling-pod-and-container-failures

存储 KEP-2268: 节点非优雅关闭功能 GA

节点非优雅关闭功能在 v1.28 中移至 GA。如果原始节点意外关闭或可能由于硬件故障或操作系统无响应而最终处于不可恢复状态， 则此功能允许有状态工作负载在不同节点上重新启动。目前，此功能需要人为干预才能生效，但我们正在寻找可能的途径来努力使其自动化。 当原始节点因故障导致无法优雅关闭时，集群管理员可以通过向原始节点打 node.kubernetes.io/out-of-service=out-of-

service=nodeshutdown:NoExecute 或 node.kubernetes.io/out-of-service=out-of-service=hardwarefailure:NoExecute 污点来触发此功能。

存储 KEP-3333: 追溯默认 StorageClass 分配功能 GA

追溯默认 StorageClass 分配功能在 v1.28 中移至 GA。此功能允许为现有未绑定持久卷的声明追溯分配默认的存储类， 从而使更改默认 StorageClass 变得更加容易。

[Windows] KEP-2371: 新增对 Windows 节点的支持 Beta

它允许用户指定 Kubelet PodAndContainerStatsFromCRI featuregate，仅从 CRI 获取 Windows Pod 和容器统计信息。

[Windows] SIG 新项目: windows-service-proxy

它是基于 Kpng 构建的，用于支持在 Windows 节点实现的服务代理的实验项目。Kpng 同样也是一 Kubernetes SIG 下的项目，它旨在针对目前 kube-proxy 存在的一些问题，重新思考实现架构，为 Kubernetes 提供一个新的服务代理实现的实验项目。

DaoCloud 社区贡献

本次发布中， DaoCloud 重点贡献了 sig-node，sig-scheduling、sig-storage 和 kubeadm 相关内容，具体功能点如下：

- [API] kube-apiserver 不再允许用户重新启用已被弃用属于 policy/v1beta1 组的 API 资源
- 网络 添加对 Pod 容器端口中的冲突端口更新/修补的警告
- 网络 Pod API 中添加了对“status.hostIPs”字段的 Alpha 支持，并添加了对“status.hostIPs”的向下 API 支持。要使用其中任何一个，都需要启用 Alpha 功能门“PodHostIPs”。
- 调度 当调度器执行 Reserve 插件时，如果插件的调度结果为不可调度时，调度器将此插件的名称记录在失败的插件列表中，以便当观测到相关资源事件时，重新计算决定是否将 Pod 放入到可调度队列中。

- [kubeadm] crictl pull 应当使用 -i 设置容器镜像服务的端点
- Apps 当 Cronjob 控制器在协调 Cronjob 创建 Job 资源时，如果因命名空间的状态是 Terminating 而导致失败时，直接返回错误，而不是重试
- Apps 修复 kubectl create job test-job --from=cronjob/a-cronjob 命令未更新 Cronjob status.lastSuccessfulTime 的问题
- Apps force_delete_pods_total 和 force_delete_pod_errors_total 指标对所有 pod 删除行为进行计数。
- 存储 弃用对 Ceph RBD 卷的 CSI 迁移的支持
- [Instrumentation] kube-controller-manager 和 kube-scheduler 使用结构化和上下文日志记录。

在 v1.28 发布过程中，DaoCloud 参与近百个问题修复和功能研发，作为作者约有 98 个提交， 详情请见贡献列表 （该版本的两百多位贡献者中有来自 DaoCloud 的 15 位）。

在 Kubernetes v1.28 的发布周期中，DaoCloud 的多名研发工程师取得了不少成就。

其中，Mygit 翻译了很多近期的官网博客和文档，成为了 SIG-Docs-Zh 的维护者。DaoCloud 还有两位中文文档维护者— windsonsea and Mentjiao.liu。

殷纳成为 SIG-scheduler 的 Approver。徐俊杰成为 kubeadm 项目的 Approver。

蔡威成为 Containerd 社区的 Reviewer，并且是 KCD 北京站 2023 的组织者和主持人。 在 KCD 北京，张世明带来了社区热点项目 Kwok（Kubernetes WithOut Kubelet） 的相关分享， 主题为“Kwok 低成本模拟集群”，同时蒋兴彦作为另一场讲演的共同讲演者，向与会者分享了“Karmada 跨集群弹性伸缩场景与实现剖析”的主题演讲。

Kubecon 2023

在即将在上海举行的 KubeCon China 2023 全球峰会中，DaoCloud 将为大家带来十几个云原生相关的主题， 包括多个维护者主题，AI 相关的云原生实践，还有多云和金融汽车等落地实践等，敬请期待。

刘齐均、潘远航和徐俊杰入选 2023 Kubecon 评审委员会；同时韩小鹏也被邀请参加 IstioCon 的评审委员会。

开源展台

Clusterpedia (https://clusterpedia.io)

名字借鉴自 Wikipedia，同样也展现了 Clusterpedia 的核心理念 —— 多集群的百科全书。通过聚合多集群资源，在兼容 Kubernetes OpenAPI 的基础上额外提供了更加强大的检索功能，让用户更快更方便地在多集群中获取到想要的任何资源。当然 Clusterpedia 的能力并不仅仅只是检索查看，未来还会支持对资源的简单控制，就像 wiki 同样支持编辑词条一样。

Merbridge (https://merbridge.io)

Merbridge 专为服务网格设计，使用 eBPF 代替传统的 iptables 劫持流量，能够让服务网格的流量拦截和转发能力更加高效。

Hwameistor (https://hwameistor.io)

它是一款 Kubernetes 原生的容器附加存储（CAS）解决方案，将 HDD、SSD 和 NVMe 磁盘形成本地存储资源池进行统一管理，使用 CSI 架构提供分布式的本地数据卷服务，为有状态的云原生应用或组件提供数据持久化能力。

Containerd、Kubespray、Piraeus、SIG-Scheduling、SIG-Node 等展台，同样也有 DaoCloud 的同学在现场为大家解答问题，欢迎大家前来交流。

主题演讲

- 9.27 上午 11 点 Kay Yan、Kante Yin、Alex Zheng、XingYan Jiang 将带来以下主题演讲：

- 《SIG 集群生命周期：Kubespray 的新功能 | SIG Cluster Lifecycle: What's New in Kubespray – Kay Yan, DaoCloud & Peng Liu, Jinan Inspur Data Technology Co, Ltd.》

议题详情介绍：
https://kccncosschn2023.sched.com/event/1PTJt/sigzhong-shi-chang-potodaepkubesprayzha-xia-sig-cluster-lifecycle-whats-new-in-kubespray-kay-yan-daocloud-peng-liu-jinan-inspur-data-technology-co-ltd，Maintainer Track | 3 夹层 3M5A 会议室 | 3M ROOM 3M5A

- 《使用 KubeRay 和 Kueue 在 Kubernetes 中托管 Sailing Ray 工作负载 | Sailing Ray Workloads with KubeRay and Kueue in Kubernetes – Jason Hu, Volcano Engine & Kante Yin, DaoCloud》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTGw/zhi-kuberayrekueuenanokubernetesfa-sailing-raydu-zhe-sailing-ray-workloads-with-kuberay-and-kueue-in-kubernetes-jason-hu-volcano-engine-kante-yin-daocloud，3 层 307 会议室| 3F ROOM 307

- 《金融行业云原生存储的最佳实践 | Best Practice for Cloud Native Storage in Finance Industry – Yang Cao & Jie Chu, Shanghai Pudong Development Bank; Alex Zheng, Daocloud》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTFU/shu-gun-chan-chang-zha-huan-best-practice-for-cloud-native-storage-in-finance-industry-yang-cao-jie-chu-shanghai-

pudong-development-bank-alex-zheng-daocloud，会议室 2 | 2F ROOM 2

- 《突破集群边界，在大规模上自动扩展工作负载 | Break Through Cluster Boundaries to Autoscale Workloads Across Them on a Large Scale – Wei Jiang, Huawei Cloud & XingYan Jiang, DaoCloud》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTJ7/di-wu-zhong-shi-yi-dui-daelsnanomao-kek-du-zhe-break-through-cluster-boundaries-to-autoscale-workloads-across-them-on-a-large-scale-wei-jiang-huawei-cloud-xingyan-jiang-daocloud，3 层 305A 会议室| 3F ROOM 305A

- 9.27 下午 2:45 Iceber Gu 将带来以下主题演讲：
  - 《项目更新和深入探讨：containerd | Project Update and Deep Dive: Containerd – Wei Fu, Microsoft & Iceber Gu, DaoCloud》

    议题详情介绍：
    https://kccncosschn2023.sched.com/event/1PTKB/tu-ju-recheng-daepcontainerd-project-update-and-deep-dive-containerd-wei-fu-microsoft-iceber-gu-daocloud，Maintainer Track | 3 夹层 3M3 会议室 | 3M ROOM 3M3

- 9.27 下午 2:55 – 3:00 Peter Pan 将带来以下主题演讲：
  - 《 闪电演讲: 带有人工智能能力的云原生 Kubernetes 运维 | Lightning Talk: Sailing Kubernetes Operation with AI Power – Peter Pan, Daocloud》

议题详情介绍：
https://kccncosschn2023.sched.com/event/1PTFl/clcong-shu-xi-xia-xia-zha-chang-kubernetes-ai-gu-lightning-talk-sailing-kubernetes-operation-with-ai-power-peter-pan-daocloud，3 层 302 会议室| 3F ROOM 302

- 9.27 下午 3:50 Mengjiao Liu、Kante Yin 将带来以下主题演讲：
  - 《SIG 仪器仪表：介绍、深入探讨和最新发展 | SIG Instrumentation: Intro, Deep Dive and Recent Developments – Mengjiao Liu, DaoCloud & Shivanshu Raj Shrivastava, Tetrate》

    议题详情介绍：
    https://kccncosschn2023.sched.com/event/1PTJq/sig-jnai-daeptao-cheng-re-sig-instrumentation-intro-deep-dive-and-recent-developments-mengjiao-liu-daocloud-shivanshu-raj-shrivastava-tetrate， Maintainer Track | 3 夹层 3M3 会议室 | 3M ROOM 3M3
  - 《SIG-Scheduling 介绍与深入探讨 | SIG-Scheduling Intro & Deep Dive – Qingcan Wang, Shopee & Kante Yin, DaoCloud》

    议题详情介绍：
    https://kccncosschn2023.sched.com/event/1PTKH/sig-scheduling-tao-cheng-sig-scheduling-intro-deep-dive-qingcan-wang-shopee-kante-yin-daocloud，Maintainer Track | 3 夹层 3M5A 会议室 | 3M ROOM 3M5A
- 9.27 下午 4:40 MinJie Huang & WenJie Song 将带来以下主题演讲：
  - 《构建一个主动-主动的高可用 Kubernetes 控制平面集群 | Building an Active-Active HA Kubernetes Control Plane Cluster

– MinJie Huang & WenJie Song, DaoCloud; Jiashun Dai, SAIC General Motors》

议题详情介绍：(https://kccncosschn2023.sched.com/event/1PTFj/zha-pan-zhi-kubernetesxiao-zhong-shi-building-an-active-active-ha-kubernetes-control-plane-cluster-minjie-huang-wenjie-song-daocloud-jiashun-dai-saic-general-motors，3 层 301 明珠厅| 3F THE PEARL HALL 301

- 9.28 上午 11:50 Paco Xu、Michael Yao 将带来以下主题演讲：
  - 《Kubernetes SIG 节点介绍和深入探讨 | Kubernetes SIG Node Intro and Deep Dive – Paco Xu, DaoCloud》

    议题详情介绍：https://kccncosschn2023.sched.com/event/1PTJk/kubernetes-sigze-tao-recheng-kubernetes-sig-node-intro-and-deep-dive-paco-xu-daocloud， Maintainer Track | 3 夹层 3M3 会议室 | 3M ROOM 3M3
  - 《Kubernetes 文档和本地化 | Kubernetes Documentation and Localization – Michael Yao, DaoCloud & Xin Li, Qihoo》

    议题详情介绍：https://kccncosschn2023.sched.com/event/1PTFs/kubernetes-repico-kubernetes-documentation-and-localization-michael-yao-daocloud-xin-li-qihoo-360，3602 层会议室 3 | 2F ROOM 3
- 9.28 下午 2:45 Hongbing Zhang、Shiming Zhang、Paco Xu 将带来以下主题演讲：

- 《云原生边缘计算与 KubeEdge：更新和未来 | Cloud Native Edge Computing with KubeEdge: Updates and Future – Fei Xu, Huawei Cloud & Hongbing Zhang, Shanghai DaoCloud Network Technology》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTKc/chang-yi-sui-dou-zhao-kubeedgedaep-cloud-native-edge-computing-with-kubeedge-updates-and-future-fei-xu-huawei-cloud-hongbing-zhang-shanghai-daocloud-network-technology，
  Maintainer Track | 3 夹层 3M5A 会议室 | 3M ROOM 3M5A

- 《深入研究：KWOK | Deep Dive: KWOK – Shiming Zhang, DaoCloud & Hao Liang, Tencent》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTJn/ye-ge-daepkwok-deep-dive-kwok-shiming-zhang-daocloud-hao-liang-tencent， Maintainer Track | 3 夹层 3M3 会议室 | 3M ROOM 3M3

- 《如何在大型集群中加速 Pod 的启动? | How Can Pod Start-up Be Accelerated on Nodes in Large Clusters? – Paco Xu, DaoCloud & Byron Wang, Birentech》

  议题详情介绍：
  https://kccncosschn2023.sched.com/event/1PTFR/nanonfzhong-shi-fu-pod-zha-dyags-how-can-pod-start-up-be-accelerated-on-nodes-in-large-clusters-paco-xu-daocloud-byron-wang-birentech，2 层会议室 1 | 2F ROOM 1

其他需要了解的功能

Apps

- kubernetes 控制器弃用 —volume-host-cidr-denylist 和 —volume-host-allow-local-loopback 命令行参数。
- 为从 CronJobs 调度的 Job 对象添加了新的注释 batch.kubernetes.io/cronjob-scheduled-timestamp。
- StatefulSet 将索引设置为 Pod 标签 statefulset.kubernetes.io/pod-index。
- kube-controller-manager：LegacyServiceAccountTokenCleanUp 功能门现在可作为 Alpha 使用（默认情况下关闭）。启用后，legacy-service-account-token-cleaner 控制器循环会删除在 —legacy-service-account-token-clean-up-period 指定的时间内未使用的服务账户令牌密钥（默认为一年），服务账户令牌是从 ServiceAccount 对象的 .secrets 列表中引用的，而不是从 pod 中引用的。
- 添加了对 pod hostNetwork 字段选择器的支持。

APIMachinery

- client-go：在观察大量不经常更改的对象时，改进了反射器缓存的内存使用
- 添加了 ConcientListFromCache 功能门，允许 apiserver 从缓存提供一致的资源列表。

CLI

- 在 registry.k8s.io/kubectl 中为 kubectl 添加了一个与其他镜像相同的架构的容器镜像（linux/amd64 linux/arm64 linux/s390x linux/ppc64le）
- 向 kubectl 添加了新的命令行参数 —interactive。新的命令行参数允许用户以交互方式确认每个资源的删除请求。

网络

- 设置 hostNetwork: true 并声明端口的 Pod 会自动设置 hostPort 字段。以前，这种情况会发生在 Deployment、DaemonSet 或其他工作负载

API 的 PodTemplate 中。现在，只有在创建实际 Pod 时才会设置 hostPort。

节点

- kubelet: —azure-container-registry-config 被弃用并在未来的版本中会被删除。 请使用 —image-credential-provider-config 和 —image-credential-provider-bin-dir 来设置。
- 如果使用 cgroups v2，则将通过 memory.oom.group 为容器 cgroup 启用 cgroup 感知 OOM 杀手。 这会导致 cgroup 中的进程被视为一个单元，并在 cgroup 中任何进程发生 OOM 终止时同时终止。
- PodResources API GA。
- 实现了 Container Runtime 的 cgroup 驱动程序的自动检测，以便在 kubelet 中使用。

调度

- kube-scheduler: 删除了 —lock-object-namespace 和 —lock-object-name。 请使用 —leader-elect-resource-namespace 和 —leader-elect-resource-name 或 ComponentConfig 来配置这些参数.
- 在“KubeSchedulerConfiguration”中添加了新的配置选项“delayCacheUntilActive”， 当“kube-scheduler”发生 leader 身份变化时，可以在内存效率和调度速度之间提供权衡。

Auth

- KMSv1 已弃用，以后只会接收安全更新。请改用 KMSv2。在未来版本中， 设置 —feature-gates=KMSv1=true 以使用已弃用的 KMSv1 功能。

存储

- GCE PD 树内插件在 v1.28 中删除（CSI 迁移完成）。
- 从 pvc.Status 中删除了 resizeStatus 枚举并替换为 AllocatedResourceStatus。

其他需要了解的功能

本次发布的主题是 Planternetes（这个新造词源于 Plant + Kuberenetes，意为绿植、环保、生机盎然的容器云编排）。



planternetes

每个 Kubernetes 版本都是我们数以千计的社区人员辛勤工作的结晶。 这个版本背后的人有行业资深人士、前辈和社区新人，他们向我们贡献了独特的经验，创造了具有全球影响力的集体工作。

就像花园一样，我们的版本有不断变化的成长、挑战和机遇。这个主题颂扬了我们为取得今天的成绩而付出的心血和努力。 和谐共处，我们成长得更好。

升级注意事项

自定义调度程序插件开发人员需要采取的行动。这是调度框架中 EnqueueExtension 的重大变化。EnqueueExtension 中的 EventsToRegister 将返回值从 ClusterEvent 更改为 ClusterEventWithHint。ClusterEventWithHint 允许每个插件通过名为 QueueingHintFn 的回调函数过滤掉更多无用的事件。当调度队列收到集群事件时，在将每个 Pod 从不可调度的 pod 池移动到 activeQ/backoffQ 之前，会调用上一个调度周期中拒绝每个 Pod 的插件的 QueueingHintFn。 为了向后兼容，nil QueueingHintFn 被视为始终返回 QueueAfterBackoff。因此，如果您只想保留现有行为， 可以注册 ClusterEventWithHint，其中不包含 QueueingHintFn。但是，从调度性能的角度来看， 注册适当的 QueueingHintFn 当然更好。

Ceph RBD 树内插件已在 v1.28 中弃用，并计划在 v1.31 中删除 （没有计划进行 CSI 迁移，建议使用 RBD 模式的 Ceph CSI 第三方存储驱动程序作为替代方案）。

Ceph FS 树内插件已在 v1.28 中弃用，并计划在 v1.31 中删除（没有计划进行 CSI 迁移）。 建议使用 Ceph CSI 第三方存储驱动程序作为替代方案）。

历史文档

- 近两年功能增加最多！Kubernetes 1.27 正式发布
- Kubernetes 正式发布 v1.26，稳定性显著提升
- Kubernetes 1.25 正式发布，多方面重大突破
- Kubernetes 1.24 走向成熟的 Kubernetes
- Kubernetes 1.23 正式发布，有哪些增强?

- Kubernetes 1.22 颠覆你的想象：可启用 Swap，推出 PSP 替换方案，还有……
- Kubernetes 1.21 震撼发布 | PSP 将被废除，BareMetal 得到增强

参考资料

1. Kubernetes 1.28 发布团队：

   https://github.com/kubernetes/sig-release/blob/master/releases/release-1.28

2. Kubernetes 1.28 变更日志 ：

   https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md

3. Kubernetes 1.28 主题讨论：

   https://github.com/kubernetes/sig-release/discussions/2271

云原生混沌工程–Chaos Mesh 实战篇



chaos mesh

通过《云原生混沌工程 — Chaos Mesh 总览篇》和《云原生混沌工程 — Chaos Mesh 控制器篇》这两篇文章的介绍，可以大概了解 Chaos Mesh 在做什么样的事情以及基本的工作原理。接下来，从安装和使用 Chaos Mesh 的角度，来具体介绍一下 Chaos Mesh 的能力。

存储

1. 安装 nfs sevrer：

```
sudo dnf install nfs-utils
sudo systemctl enable --now nfs-server
```

2. 创建 nfs server 的导出的文件系统：

```
sudo mkdir -p /srv/nfs4/k8s
```

3. 挂载实际的目录：

```
sudo mkdir /opt/k8s
sudo mount --bind /opt/k8s /srv/nfs4/k8s
```

4. 想要这个挂载持久化，添加下面的条目到 /etc/fstab 文件，以下是编辑完的结果：

```
cat /etc/fstab

/opt/k8s /srv/nfs4/k8s none bind 0 0
```

5. 导出文件系统，以下是编辑完的结果：

```
cat /etc/exports

/srv/nfs4 10.1.3.0/24(rw,sync,no_root_squash,no_subtree_check,crossmnt,fsid=0)
/srv/nfs4/k8s 10.1.3.0/24(rw,sync,no_root_squash,no_subtree_check)

sudo exportfs -ra
```

6. 查看倒出来的文件系统：

```
sudo exportfs -v
```

7. 安装 Helm

```
wget https://get.helm.sh/helm-v3.8.1-linux-amd64.tar.gz
tar -zxvf helm-v3.8.1-linux-amd64.tar.gz
mv linux-amd64/helm /usr/local/bin/helm
```

安装 nfs-subdir-external-provisioner

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=10.1.3.210 --set nfs.path=/k8s --set image.repository=k8s.m.daocloud.io/sig-storage/nfs-subdir-external-provisioner --set image.tag=v4.0.2 --set storageClass.defaultClass=true
```

数据库

1. 准备 pvc mysqlpvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: mysql
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
    storage: 20Gi
```

```
kubectl apply -f mysqlpvc.yaml
```

2. 安装 mysql mysql.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  selector:
    matchLabels:
    app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
```

```
        - image: mysql:5.6
          name: mysql-con
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: dangerous
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  labels:
    app: mysql
spec:
  type: NodePort
  selector:
    app: mysql
  ports:
    - protocol: TCP
    port: 3306
    targetPort: 3306

kubectl apply -f mysql.yaml
```

3. 检查 mysql 的安装

```
kubectl get pvc|grep mysql-pv-claim

mysql-pv-claim                Bound     pvc-c64c8f7b-0408-49e2-953e-09
d36003a3d8    20Gi          RWO                nfs-client       6d3h

kubectl get pods|grep mysql-deployment

mysql-deployment-68bd7964d7-x9c8h                        1/1       Running
  0              6d3h
```

4. 连接数据库

```
kubectl run my-release-mysql-client --rm --tty -i --restart='Never' --i
mage  docker.io/bitnami/mysql:8.0.32-debian-11-r8 --namespace default
 --env MYSQL_ROOT_PASSWORD=dangerous --command -- bash
kubectl exec -it my-release-mysql-client bash
mysql -h 10.1.3.210 -P32265 -uroot -pdangerous
```

安装 Chaos Mesh

```
kubectl create ns chaos-mesh
helm repo add chaos-mesh https://charts.chaos-mesh.org
helm install chaos-mesh chaos-mesh/chaos-mesh -n=chaos-mesh --set chaos
Daemon.runtime=containerd --set chaosDaemon.socketPath=/run/containerd/con
tainerd.sock --set controllerManager.leaderElection.enabled=false --set controlle
rManager.replicaCount=1 --set dashboard.env.DATABASE_DRIVER=mysql --set d
ashboard.env.DATABASE_DATASOURCE=root:dangerous@tcp'(10.1.3.210:32265)'/c
haosmesh?parseTime=true
```

检查 Chaos Mesh

```
kubectl get po -n chaos-mesh

NAME                                          READY    STATUS     RESTARTS
  AGE
chaos-controller-manager-77558f4c96-zv8s4    1/1       Running    0            5d
```

```
chaos-daemon-2nm9v                          1/1     Running   0          6
d2h
chaos-daemon-mrvh6                          1/1     Running   0          6
d2h
chaos-dashboard-544c89d476-d29kt            1/1     Running   0          6
d2h
```

使用 Chaos Mesh

准备

部署一个 nginx 容器用于测试使用。

```
helm repo add my-repo https://charts.bitnami.com/bitnami
helm install nginx --set service.type=NodePort my-repo/nginx
```

登录

- 输入令牌登录：



input token

- 点击 link 来打开令牌的辅助生成器：



input token

- 复制授权的 yaml，进行授权，并生成登录的 token，之后使用 ServiceAccount 的 name 和生成的 token 登录：

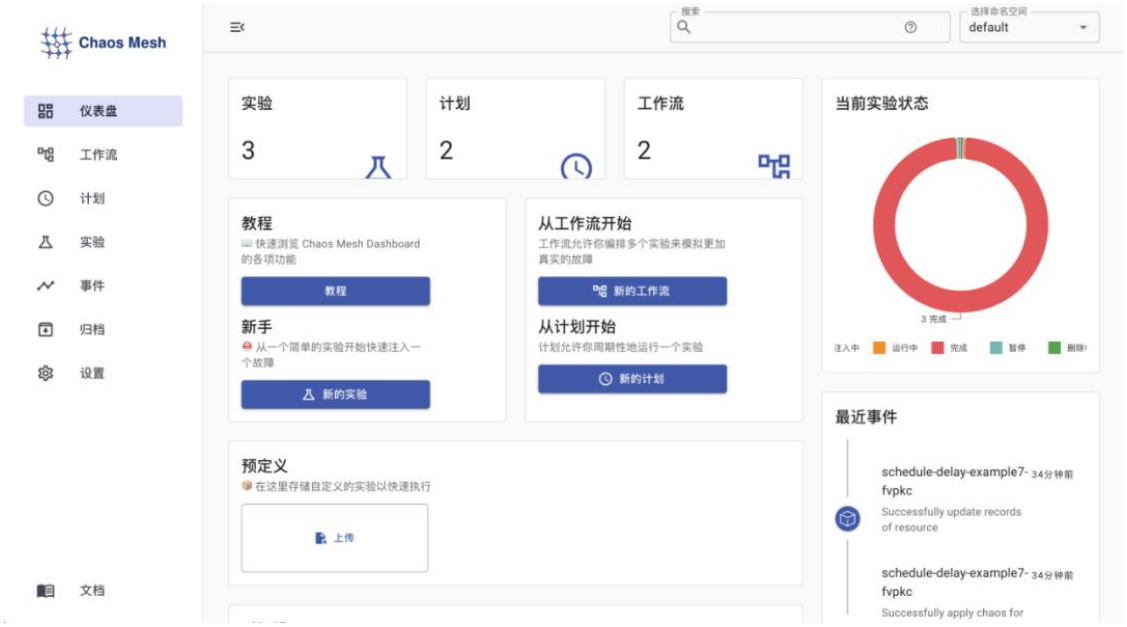
```
vi rbac.yaml

kind: ServiceAccount
apiVersion: v1
metadata:
  namespace: default
  name: account-default-viewer-xfddq


---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: role-default-viewer-xfddq
rules:
- apiGroups: [""]
```

```
  resources: ["pods", "namespaces"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["chaos-mesh.org"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]


---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-default-viewer-xfddq
  namespace: default
subjects:
- kind: ServiceAccount
  name: account-default-viewer-xfddq
  namespace: default
roleRef:
  kind: Role
  name: role-default-viewer-xfddq
  apiGroup: rbac.authorization.k8s.io

kubectl apply -f rbac.yaml
kubectl create token account-default-viewer-xfddq

eyJhbGciOiJSUzI1NiIsImtpZCI6IjZjbVpnWEpzNWxrNHdDNXBZWGZaYlltYldqV
zRDR1RRcnozVS1Oc0pGVk0ifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVz
LmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjc3ODM4MzQ0
LCJpYXQiOjE2Nzc4MzQ3NDQsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZW
ZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3
BhY2UiOiJkZWZhdWx0Iiwic2VydmljZWFjY291bnQiOnsibmFtZSI6ImFjY291bn
QtZGVmYXVsdC1tYW5hZ2VyLXR5eHB4IiwidWlkIjoiY2VlOWQ5OTItMjEwYy00
NWRjLTk2ODQtMjUwYTdlZTdlMTk2In19LCJuYmYiOjE2Nzc4MzQ3NDQsInN1
YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmFjY291bnQtZGVm
```

YXVsdC1tYW5hZ2VyLXR5eHB4In0.ffnSlmHHfewt8B5ErxIxMRbvk8gloKjazF3ocD9jFkqHMcNedqLo1XZSTE7lDlbdKgKJept6Vq03R4GusW5doKqC51lalwh-aWLFYL_lJZ-wAOenU5-HLM_L3SgFSYluSCwraLfwcatLuzqtkOT_hO8lMoGYLEJQghw6iHlogXf-Z6ckPGt6VGsP6bU40Xaz2EVQw76qJwg2HpkAEbx-ucF6lID_J9Pg5pVsMBW75lgYy3FgITOPXKh85klm15VqopvNJCRMRHjg-5RcoLBbhlhBVuU9FBwVRqt0TwVXX2CH-f6H8LUlMcDDbrHjBntnOfiBO4r1dru2X38rujRpVg

- 使用 account-default-viewer-xfddq 作为名称，生成出来的 token 作为令牌，就可以登录到对应 namespace。 支持 namespace 和 cluster 范围的令牌申请。

仪表盘



dashboard

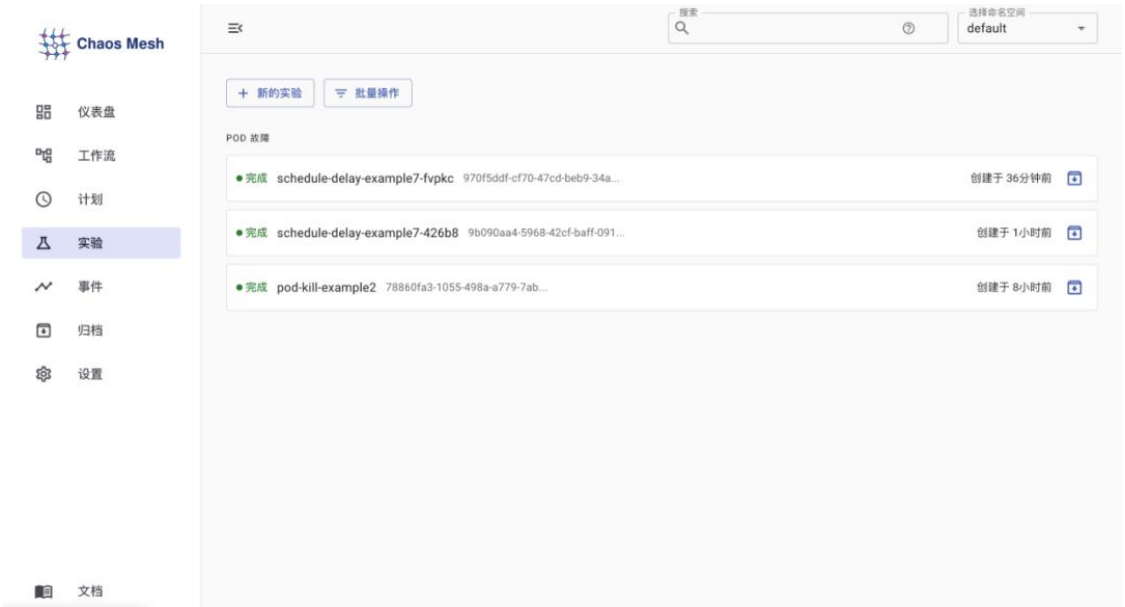
实验

以 PodChaos 为例，创建一个 Chaos 类型的实验，完成删除 Pod 的故障注入。

- 定义：

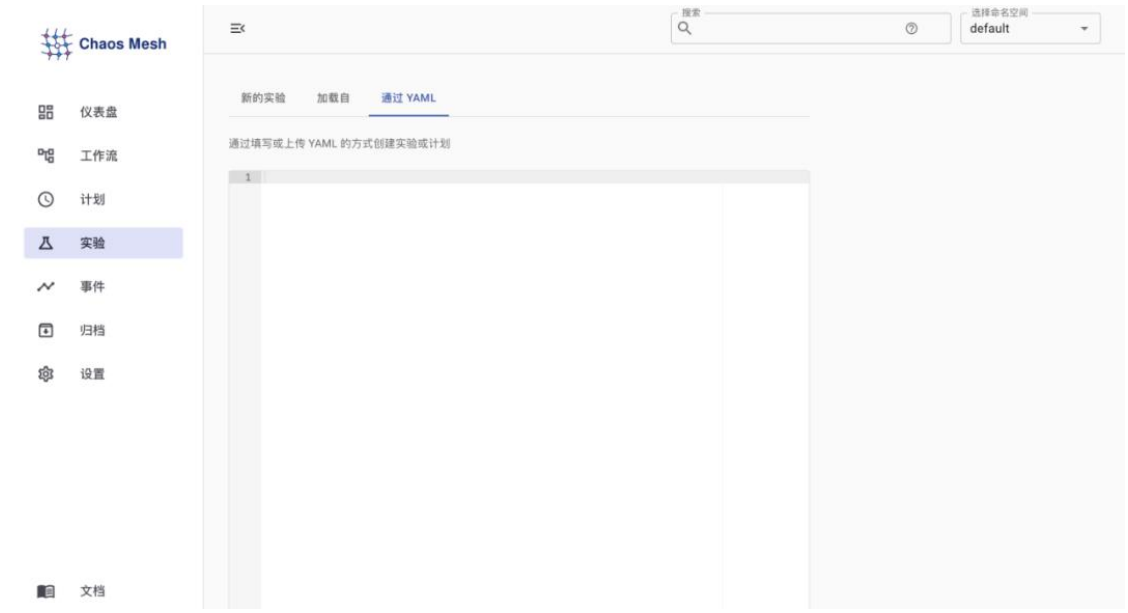
```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: pod-kill-example2
  namespace: default
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces:
      - default
    labelSelectors:
      "app.kubernetes.io/instance": "nginx"
```

- 列表：



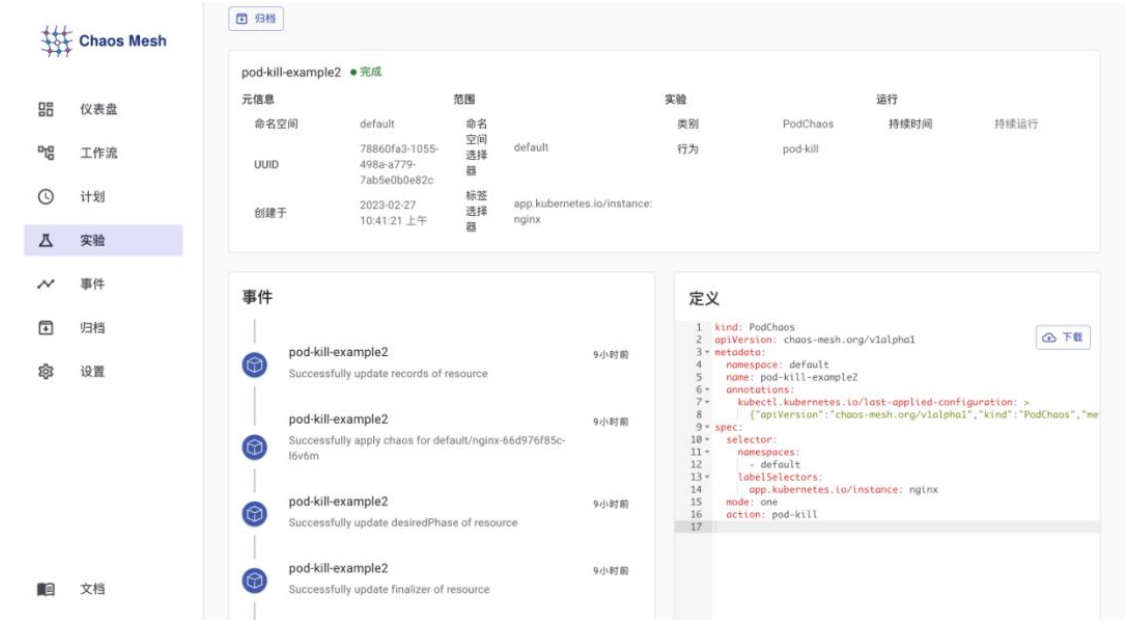
list

- 创建：



create

- 详情：其中包含元数据，完整的 yaml 定义，以及对应的事件。同时 可以将其进行归档，归档完之后就只能在归档的菜单中可以查看到， 因为对应的资源对象已经从 Kubernetes 中删除了，只保存在了数据中。



details

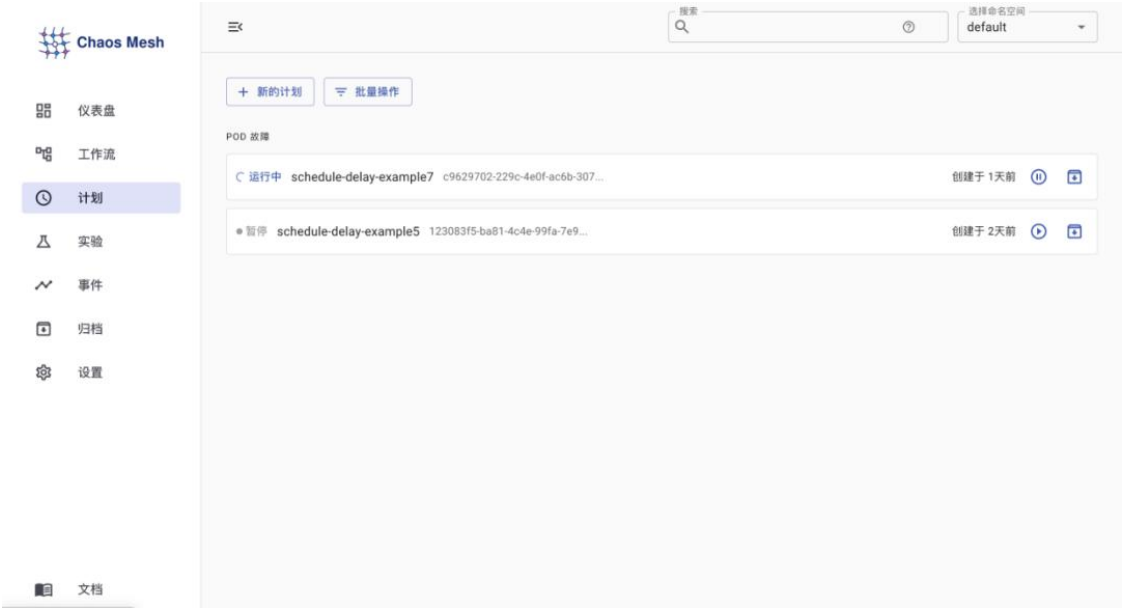
计划

定义一个在每小时的 05 分会定时执行的计划，计划中的任务是删除选中的 nginx 的 Pod。

- 定义：

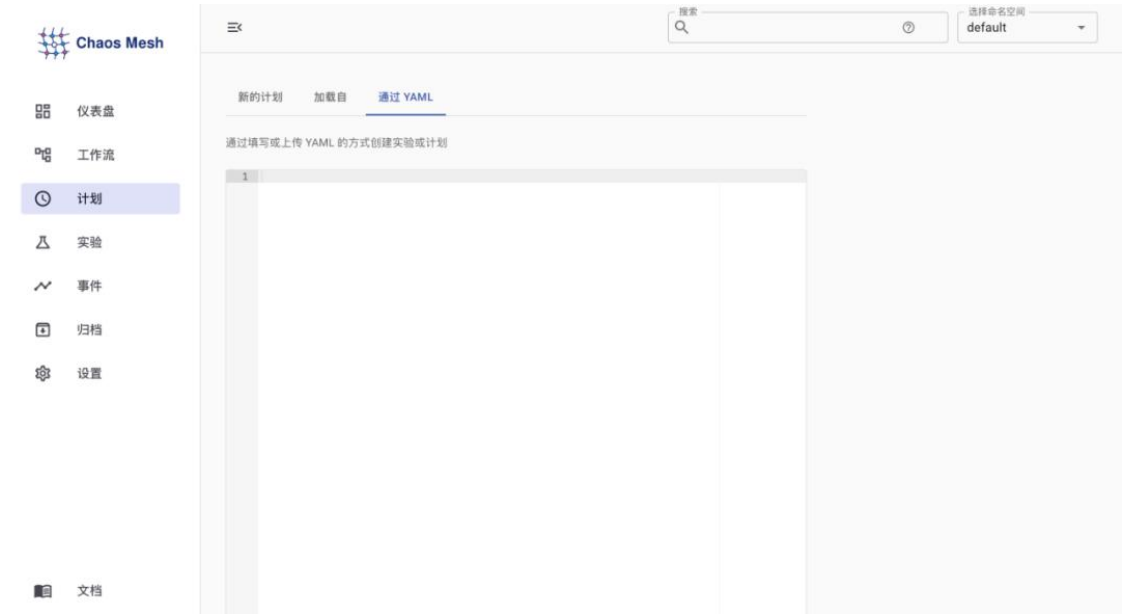
```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: Schedule
metadata:
  name: schedule-delay-example
  namespace: default
spec:
  schedule: "5 * * * *"
  historyLimit: 2
  concurrencyPolicy: Allow
  type: "PodChaos"
  podChaos:
    action: "pod-kill"
    mode: one
    selector:
      namespaces:
        - default
    labelSelectors:
        "app.kubernetes.io/instance": "nginx"
```

- 列表：



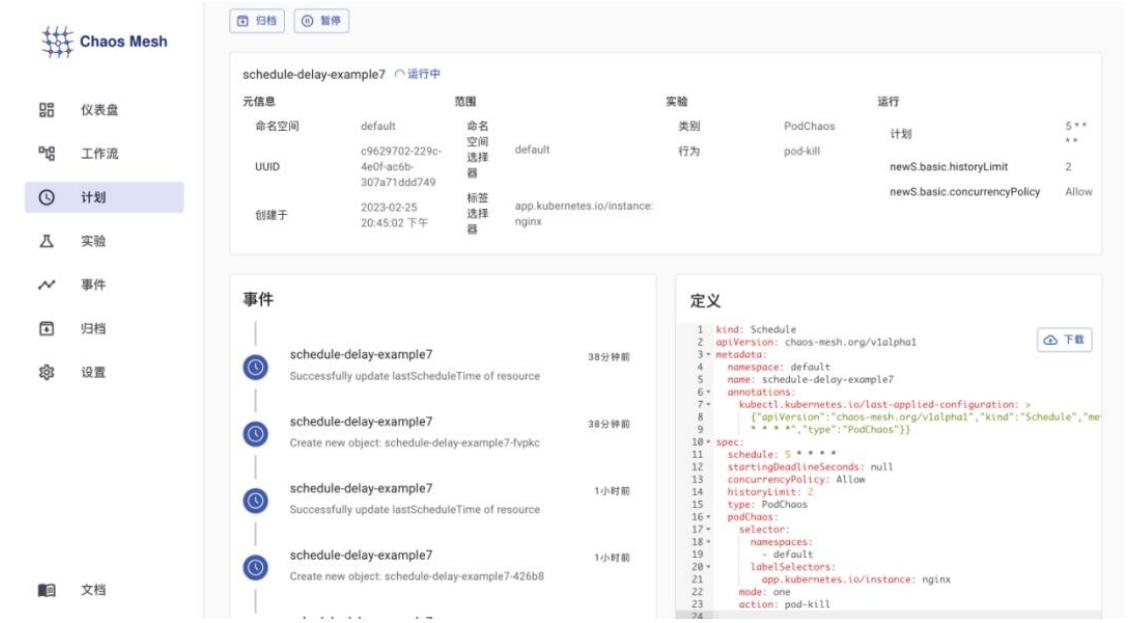
list

- 创建：



create

- 详情：其中包含元数据，完整的 yaml 定义，以及对应的事件。同时 可以将其进行归档，归档完之后就只能在归档的菜单中可以查看到， 因为对应的资源对象已经从 Kubernetes 中删除了，只保存在了数据中。



details

工作流

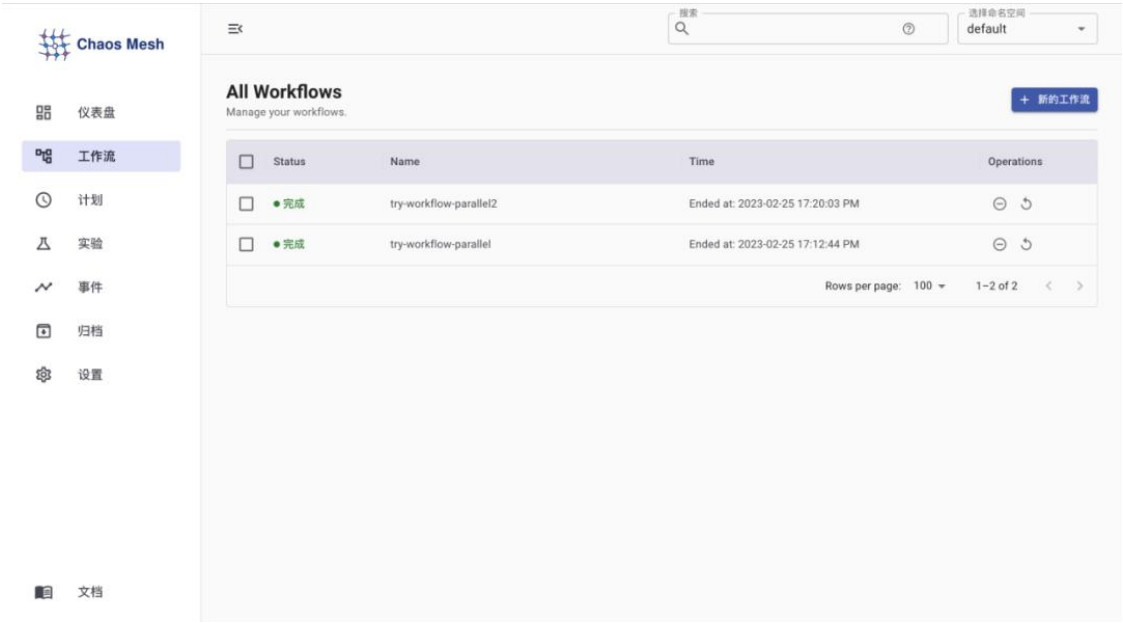
定义一个工作流，包含入口节点 the-entry 和 Chaos 类型的节点 workflow-pod-chaos。

- 定义：

```
apiVersion: chaos-mesh.org/v1alpha1
kind: Workflow
metadata:
  name: try-workflow-parallel
spec:
  entry: the-entry
  templates:
    - name: the-entry
      templateType: Parallel
      deadline: 240s
      children:
        - workflow-pod-chaos
```

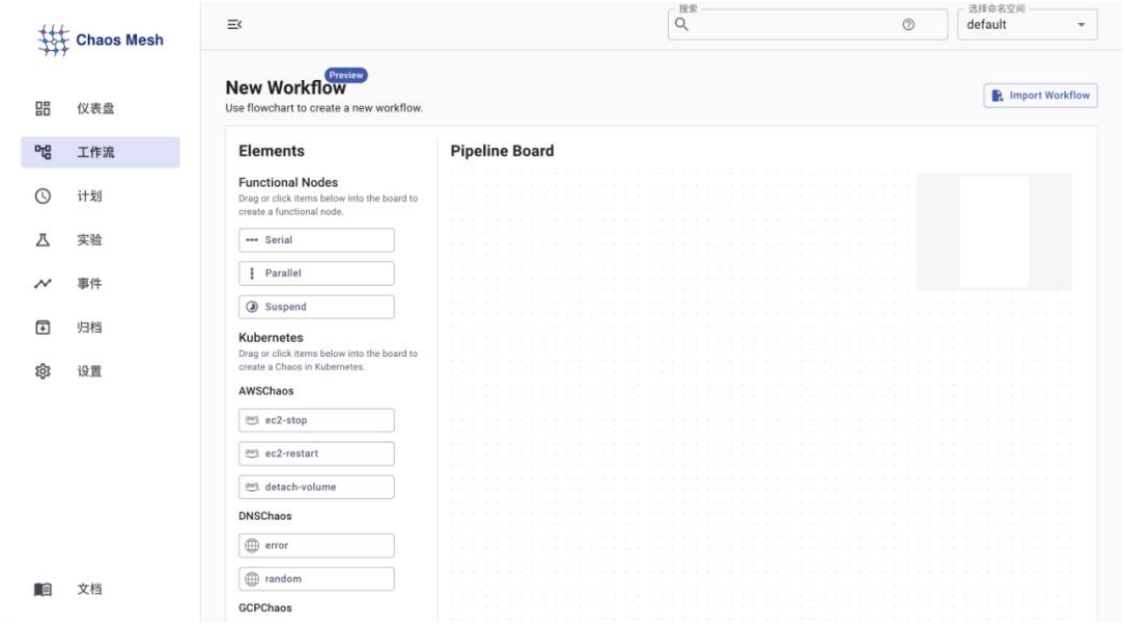
```
  - name: workflow-pod-chaos
    templateType: PodChaos
    deadline: 20s
    podChaos:
      action: pod-kill
      mode: one
      selector:
        namespaces:
          - default
        labelSelectors:
          "app.kubernetes.io/instance": "nginx"
```

- 列表：



list

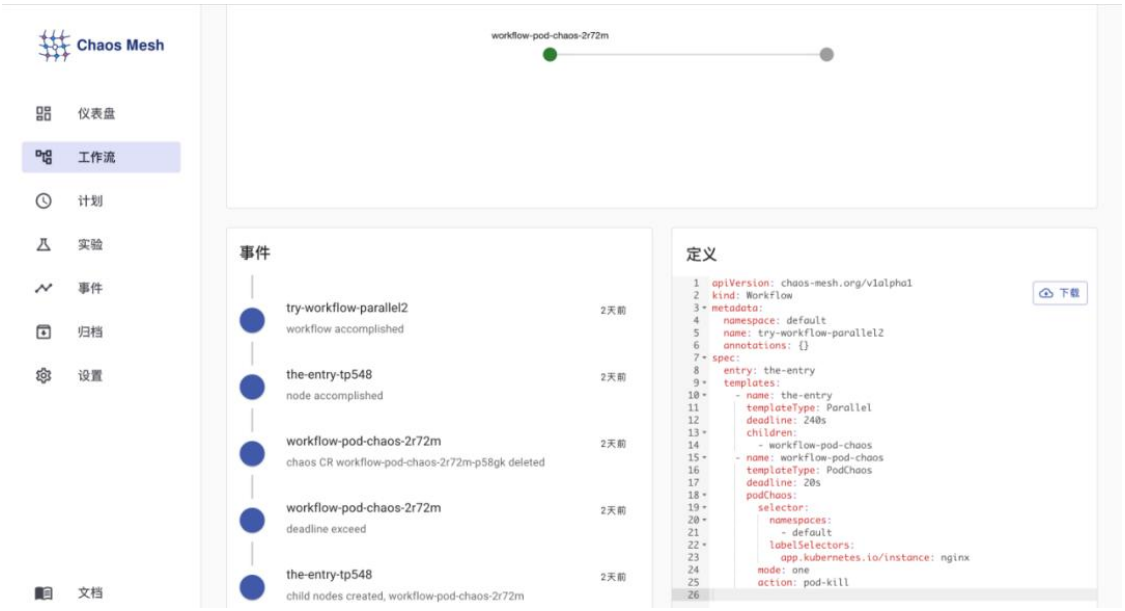
- 创建：



create

为了快速创建去看效果，直接通过 kubectl 命令行 apply 的方式创建 Workflow 资源对象，页面操作会慢一点。

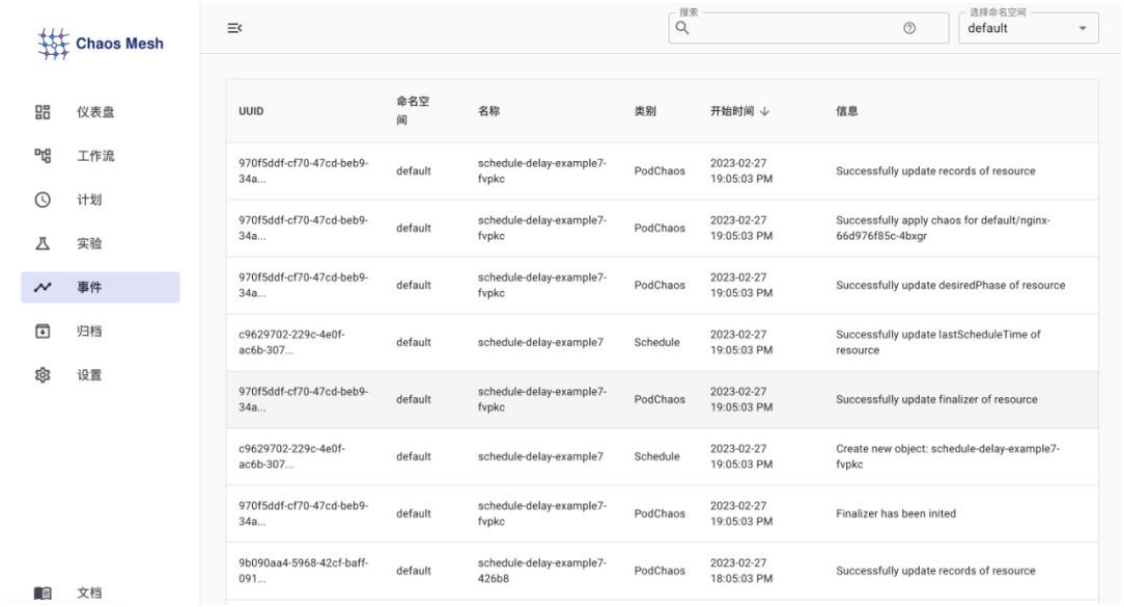
- 详情：其中包含工作流运行的拓扑情况，完整的 yaml 定义，以及对应的事件。同时 可以将其进行归档，归档完之后就只能在归档的菜单中可以查看到，因为对应的资源对象已经从 Kubernetes 中删除了，只保存在了数据中。以及在工作流运行完成之后，在过程中创建的 Chaos 类型的资源对象也会被自动删除掉。



details

事件

这里汇聚了所有资源对象相关的事件，可以进行查询操作。这些事件数据是从数据库中查出来的，之前的文章提到过数据库中事件数据的来源的原理。同时，支持可以设置事件数据的过期时间，一旦到达过期时间，就会将其从数据库中删除掉。

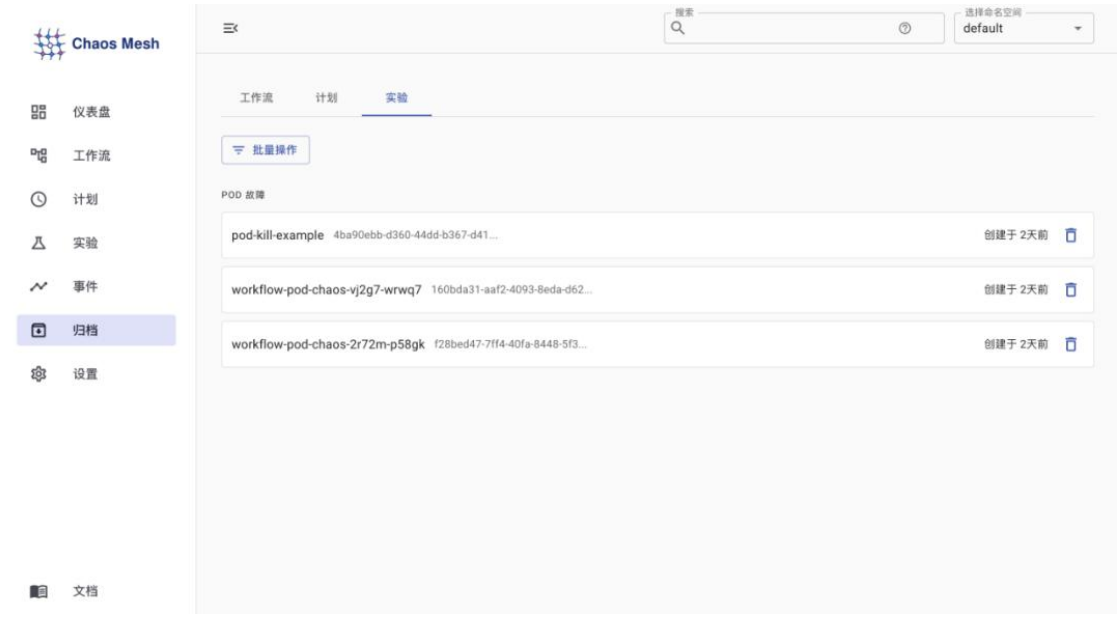


details

归档

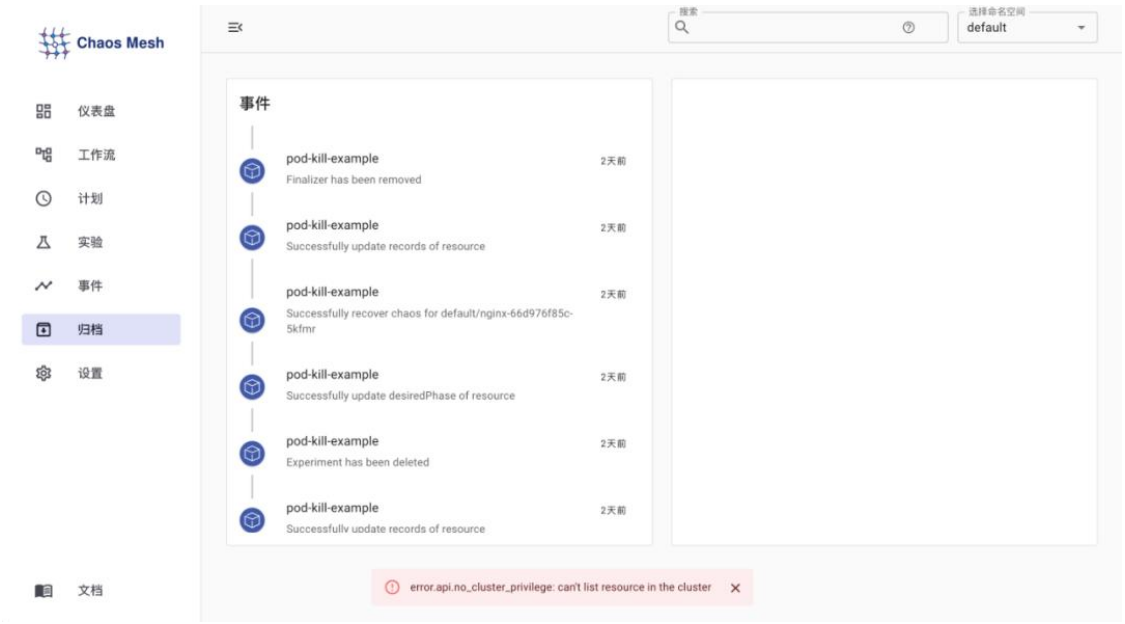
当在页面上对相关资源对象点击归档，或者是从底层 Kubernetes 直接删除资源对象，都是可以将其变成归档状态，归档的数据是保存在数据库中的。归档完的数据就会在这里进行展示，同时可以设置归档数据的过期时间，一旦到达过期时间，就会将其从数据库中删除掉。

- 列表：



list

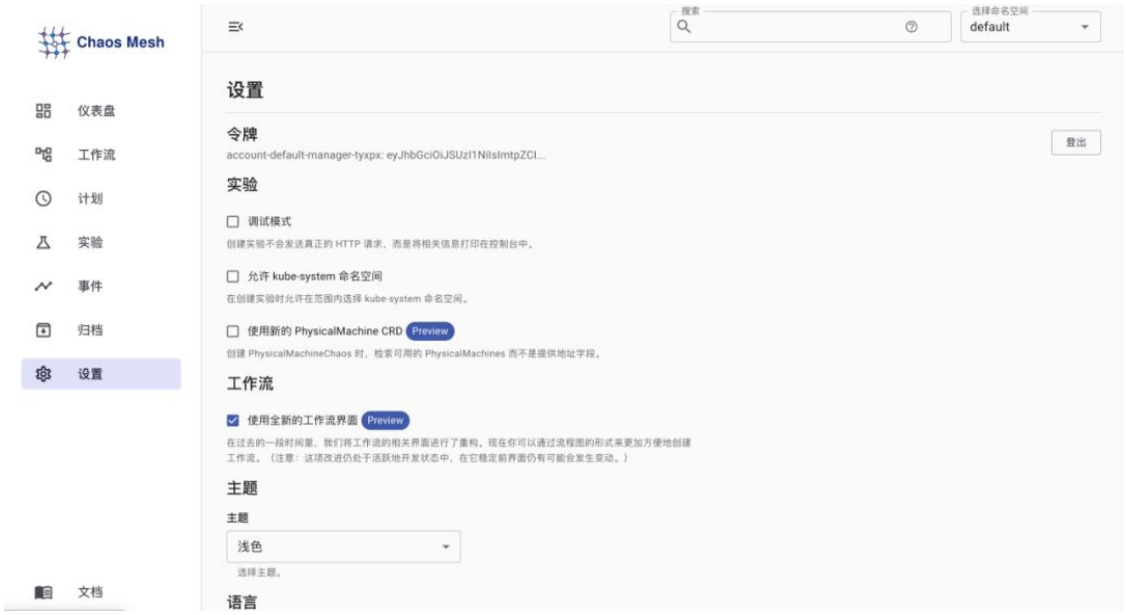
- 详情：已经归档的实验，计划，工作流对象是不能查看详情了，原因是归档的资源对象在 Kubernetes 中已经不存在了。



details

设置

主要包含登录/登出，主题色彩，中英文设置，还有一些其它的设置开关。



setting

StatusCheck

在目前的版本中，页面上还不支持显示 StatusCheck 的类型，这里以命令行的方式来展示 Continuous 和 Synchronous 两种方式的 StatusCheck 的使用。

- Continuous 类型：

```
apiVersion: chaos-mesh.org/v1alpha1
kind: StatusCheck
metadata:
  name: status-check-example
spec:
  mode: Continuous
  type: HTTP
  intervalSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3
  http:
    url: http://10.1.3.210:30028
    method: GET
    criteria:
      statusCode: "200"
```

检查状态：

```
kubectl describe StatusCheck status-check-example
```

```
Name:         status-check-example
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  chaos-mesh.org/v1alpha1
Kind:         StatusCheck
Metadata:
```

```
  Creation Timestamp:  2023-03-05T09:20:57Z
  Generation:          1
  ......
Spec:
  Failure Threshold:  3
  Http:
    Criteria:
      Status Code:      200
    Method:             GET
    URL:                http://10.1.3.210:30029
  Interval Seconds:     30
  Mode:                 Continuous
  Records History Limit:  100
  Success Threshold:    1
  Timeout Seconds:      5
  Type:                 HTTP
Status:
  Conditions:
    Last Probe Time:       2023-03-05T09:22:23Z
    Last Transition Time:  2023-03-05T09:21:23Z
    Reason:
    Status:                True
    Type:                  SuccessThresholdExceed
    Last Probe Time:       2023-03-05T09:22:23Z
    Last Transition Time:  2023-03-05T09:20:53Z
    Reason:
    Status:                False
    Type:                  Completed
    Last Probe Time:       2023-03-05T09:22:23Z
    Last Transition Time:  2023-03-05T09:20:53Z
    Reason:
    Status:                False
```

```
    Type:                   DurationExceed
    Last Probe Time:        2023-03-05T09:22:23Z
    Last Transition Time:   2023-03-05T09:20:53Z
    Reason:
    Status:                 False
    Type:                   FailureThresholdExceed
  Count:                    3
  Records:
    Outcome:        Success
    Start Time:     2023-03-05T09:21:23Z
    Outcome:        Success
    Start Time:     2023-03-05T09:21:53Z
    Outcome:        Success
    Start Time:     2023-03-05T09:22:23Z
  Start Time:       2023-03-05T09:20:53Z
Events:
  Type     Reason                           Age    From            Mess
age
  ----     ------                           ----   ----            ---
----
  Normal   StatusCheckExecutionSucceed       69s    statuscheck     HTT
P execution of status check succeed
  Normal   StatusCheckSuccessThresholdExceed  69s    statuscheck     succ
ess threshold exceed
  Normal   StatusCheckSuccessThresholdExceed  69s    statuscheck     succ
ess threshold exceed
  Normal   StatusCheckSuccessThresholdExceed  69s    statuscheck     succ
ess threshold exceed
  Normal   StatusCheckExecutionSucceed       39s    statuscheck     HTT
P execution of status check succeed
  Normal   StatusCheckSuccessThresholdExceed  39s    statuscheck     succ
ess threshold exceed
```

```
  Normal  StatusCheckSuccessThresholdExceed  39s   statuscheck  succ
ess threshold exceed
  Normal  StatusCheckSuccessThresholdExceed  39s   statuscheck  succ
ess threshold exceed
  Normal  StatusCheckSuccessThresholdExceed  39s   statuscheck  succ
ess threshold exceed
  Normal  StatusCheckExecutionSucceed        9s    statuscheck  HTT
P execution of status check succeed
  Normal  StatusCheckSuccessThresholdExceed  9s    statuscheck  succ
ess threshold exceed
  Normal  StatusCheckSuccessThresholdExceed  9s    statuscheck  succ
ess threshold exceed
```

- Synchronous 类型：

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: StatusCheck
metadata:
  name: status-check-example
spec:
  mode: Synchronous
  type: HTTP
  intervalSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3
  http:
    url: http://10.1.3.210:30028
    method: GET
    criteria:
      statusCode: "200"
```

检查状态：

```
kubectl describe StatusCheck status-check-example

Name:              status-check-example
Namespace:         default
Labels:            <none>
Annotations:       <none>
API Version:       chaos-mesh.org/v1alpha1
Kind:              StatusCheck
Metadata:
  Creation Timestamp:  2023-03-05T09:25:15Z
  ......
Spec:
  Failure Threshold:  3
  Http:
    Criteria:
      Status Code:          200
    Method:                 GET
    URL:                    http://10.1.3.210:30029
  Interval Seconds:         30
  Mode:                     Synchronous
  Records History Limit:    100
  Success Threshold:        1
  Timeout Seconds:          5
  Type:                     HTTP
Status:
  Completion Time:  2023-03-05T09:25:53Z
  Conditions:
    Last Probe Time:          2023-03-05T09:25:53Z
    Last Transition Time:     2023-03-05T09:25:53Z
    Reason:                   StatusCheckSuccessThresholdExceed
    Status:                   True
    Type:                     Completed
```

Last Probe Time: 2023-03-05T09:25:53Z
Last Transition Time: 2023-03-05T09:25:12Z
Reason:
Status: False
Type: DurationExceed
Last Probe Time: 2023-03-05T09:25:53Z
Last Transition Time: 2023-03-05T09:25:12Z
Reason:
Status: False
Type: FailureThresholdExceed
Last Probe Time: 2023-03-05T09:25:53Z
Last Transition Time: 2023-03-05T09:25:53Z
Reason:
Status: True
Type: SuccessThresholdExceed
Count: 1
Records:
Outcome: Success
Start Time: 2023-03-05T09:25:42Z
Start Time: 2023-03-05T09:25:12Z
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Normal StatusCheckExecutionSucceed 4m40s statuscheck HTTP execution of status check succeed
Normal StatusCheckCompleted 4m29s statuscheck status check completed: StatusCheckSuccessThresholdExceed
Normal StatusCheckSuccessThresholdExceed 4m29s statuscheck success threshold exceed

总结

经过安装和实践，可以更好地理解使用 Chaos Mesh 的一些方式。这里只是举了一些例子帮助理解，对于其它更多的使用方式可以参考官方的文档。

Chaos Mesh 简介： https://chaos-mesh.org/zh/docs/

作者：熊先生

5.0 应用工作台能力介绍

如今随着云原生理念迅速普及，云原生技术给企业数字化转型的过程中带来了巨大的便利。但是企业云原生过程化中面临着一些列挑战：

- 多租户设计下如何满足逻辑层面的资源隔离？如何满足物理资源的隔离？
- 频繁的持续构建，如何及时在更早的阶段找到问题、修复 bug，持续保证代码和环境符合预期？
- 云原生下的技术种类复杂，如何利用好云原生建设云原生下的 CI/CD 系统？
- 云原生技术带来了应用部署复杂性，如何降低开发者的认知负担？
- 全自动的应用部署，如何让新功能发布可控、可预测、可逆？

面对这样的挑战，DCE 5.0 基于 Kubernetes 等云原生技术打造的应用工作台应运而生，集 DaoCloud DevOps 实践、云原生社区前沿研发理念于一体。 为企业建立一套自动化的从源代码构建到持续部署应用，且可以覆盖云原生下所有形态应用全生命周期管理的开发运维一体化的环境。帮助企业缩短研发周期，提供应用交付效率。

图形化用户交互

数字经济时代，云原生的应用已成为企业数字化转型的必选项，所以应用工作台以应用为中心，基于 DevOps 理念，解决云原生应用开发、自动化交付、运维全生命周期。降低企业应用云原生门槛，提高应用交付频率。应用工作台涉及：层级多租户资源、云原生应用、流水线 CI/CD 流水线、GitOps、渐进式交付等模块。

用户在使用过程中主要存在三个阶段：**开发阶段、交付阶段、运维阶段。**

1. **在开发阶段**：应用工作台提供了流水线功能，企业可以在流水线中定义编译代码、代码检查、构建云原生制品流程，帮助开发者及时测试代码，从而

提高代码质量。另外整个流水线一次设置，多次复用，大大降低了开发人员的工作负担，减少了不必要的重复劳动。

2. **在交付阶段**：应用工作台给予目前 GitOps 理念，以自动化方式，频繁且持续性的将企业的应用部署到生产环境。作为上一阶段的延伸，自动化的持续部署起到了承上启下的作用，是对软件的研发的全流程管控的闭环。应用工作台可自动且持续监测代码仓库与生产环境中的应用进行比对，来确保生产环境与仓库中的期望状态保持一致，以实现全自动化部署。另外应用工作台还支持在持续部署的基础上对接外部能力，从而实现渐进式交付下的灰度发布等高级发布策略。

3. **在运维阶段**：应用工作台针对多种形态的云原生应用，提供了统一的观测平面，包含监控、告警、日志等信息。还提供了对应用升级、回滚、停止和删除应用等功能。



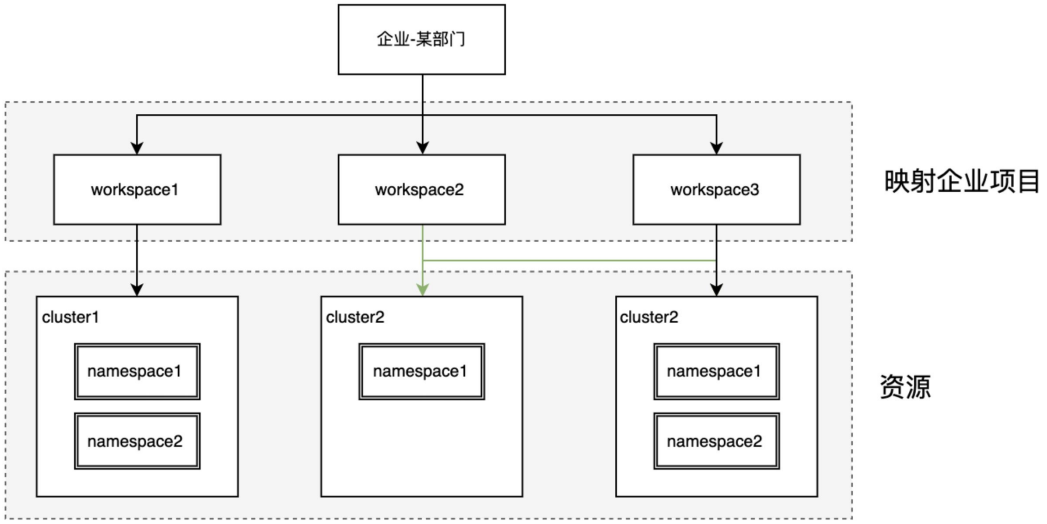
三个阶段

功能特性

应用工作台提供了层级资源管理、云原生应用、CI/CD 流水线、GitOps、渐进式交付核心服务，来满足企业不同场景的需求。

层级资源管理

在企业环境中，企业下的每一个部门由不同的项目构成，一个一个的项目就映射到了我们的工作空间（workspace）。下图是应用工作台提供了基于 Kubernetes 的多租户管理方案。



层级资源

前提条件：用户可以在 5.0 全局管理模块被设置为工作空间的管理员来参与工作空间的管理，从而参与项目的协同。

特点：

- 工作空间是最小的租户单元。
- 工作空间支持弱绑定集群的模式，从而使用户获得跨集群、跨命名空间共享资源的能力。
- 工作空间支持强绑定集群的模式，从而使用户获得独享集群资源的能力。
- 工作空间的管理员可以在该工作空间关联的集群中创建命名空间资源，并为命名空间进行配额管理。

支持多种形态云原生应用的部署，包括源代码、Jar 包、镜像一键完成应用部署，并且在创建过程中可使用微服务的注册发现、服务治理能力，还支持通过应用商城

一键部署 Helm 应用、OLM 应用。降低了企业上云的门槛，简单易用、敏捷高效。

云原生应用

应用工作台以应用为中心，涵盖了云原生技术下多种形态的应用，应用工作台对云原生社区内的不用形态应用采取了更包容的态度，而不是按照自己对应用的理解，在平台中定义一套自己的应用规范。旨在帮助用户无需学习云原生知识，即可“用好云”。

目前应用工作台支持的云原生应用有：



云原生应用

应用工作台面向应用开发和运维，覆盖应用全生命周期，包括应用的创建、删除、配置修改、自动扩缩及自动运维。 并且支持 SpringCloud、Dubbo、ServiceMesh 服务治理架构，与 5.0 微服务引擎、5.0 服务网格无缝集成：

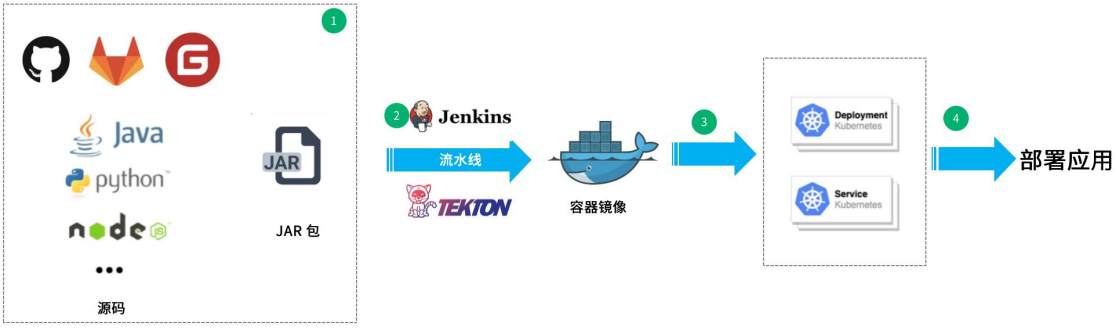


集成

CI/CD 流水线

流水线是一个自定义的 CI/CD 流水线模式，定义了包含构建、测试和发布的完整构建过程。应用工作台基于流水线能力具有以下优势：

- 不同来源构建应用：支持使用源码、软件包（Jar）实现应用的一键部署。
- 双引擎：支持 Jenkins 和 Tekton 作为应用工作台的流水线系统引擎。
- 丰富的流水线模板：内置了多条流水线官方模板，大大降低了用户的使用门槛，可以适应不同业务场景，满足用户的日常需求。



流水线

GitOps

GitOps 是一种为云原生应用实现持续部署的理念。应用工作台全面拥抱 GitOps，GitOps 的核心思想是拥有一个 Git 仓库，并将应用系统的申明式基础架构和应用程序存放在 Git 仓库中进行版本控制。 GitOps 结合 Kubernetes 能够利用自动交付流水线将更改应用到指定的任意多个集群中，从而解决跨云部署的一致性问题。

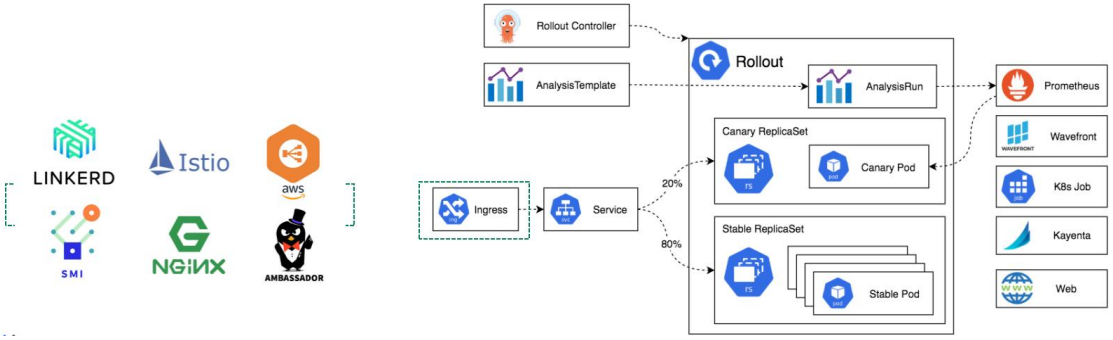


GitOps

渐进式发布

为了应用部署的安全性，企业都采取了灰度发布的解决方案，这种部署策略使应用部署更加安全，但是仍然缺乏自动化。将服务发布到生产和分析指标的过程仍然是一个手动过程。这将促使我们进入下一阶段，即渐进式交付。

应用工作台基于 Argo Rollouts 实现了渐进式的灰度发布，可以让开发人员可以选择他们自己的分析指标，自定义他们的渐进式发布的步骤，甚至选择他们自己的入口或服务网格提供商来进行流量控制。



渐进式发布

常见问答

1. **什么是 DevOps**

   DevOps 是开发（Dev）和运营（Ops）的综合体，是指将开发、质量保证和 IT 运营集成到统一的环境以及软件交付过程中。 通过采用 DevOps 文化、做法和工具，企业能够更好地响应业务需求、更快的构建应用程序，并更快地实现业务目标。

   DevOps 包括以下活动和操作：

   - **持续集成（CI）**是指如下的做法：将所有开发人员代码频繁合并到一个中央代码库中，然后执行自动化生成和测试过程。 目标是快速发现和纠正代码问题、简化部署并确保代码质量。
   - **持续交付（CD）**是指如下的做法：自动生成、测试代码并将其部署到类似生产的环境。目标是确保代码始终可供部署。 添加持续交付来创建完整 CI/CD 流水线的做法有助于尽早检测到代码缺陷。并确保能够在极短的时间内发布已经过适当测试的更新。

- **持续部署** 是一个附加的过程，可自动引入通过 CI/CD 流水线传递的所有更新，并将其部署到生产环境。 持续部署需要可靠的自动测试和高级过程规划， 不一定适合所有团队。
- **持续监视** 是指在 DevOps 和 IT 运营生命周期的每个阶段中整合监视功能所要采用的流程和技术。 监视有助于确保应用程序在从开发环境转移到生产环境时，应用程序和基础结构保持正常的运行状况、性能和可靠性。持续监视是基于 CI 和 CD 的概念。

2. **什么是流水线?**

   流水线可以自动生成和测试代码项目，使其可供他人使用。 适用于任何语言或项目类型。将持续集成（CI）和持续交付（CD）组合在一起，可以测试并生成代码，并将代码发送给任何目标。 应用工作台还支持可视化的方式来编排流水线，企业可以将现有研发流程通过流水线以可视化的方式呈现出来。

3. **流水线如何构建一个应用程序?**

   应用工作台提供了从 Git 导入代码、上传 Jar 包来创建应用程序。 在创建的过程中，只要填写流水线所需的部分信息，应用工作台就会帮您创建一条流水线资源，自动化完成从源码构建、测试、发布应用至 Kubernetes 集群环境中。 旨在帮助企业以更快的速度将软件持续交付，同时降低风险。

4. **支持哪些云原生应用?**

   Kubernetes 原生应用、Helm 应用、OAM 应用、OLM 应用。

5. **如何运维多种形态的云原生应用?**

   应用工作台推出以应用为中心，统一观测平面，包含监控、告警、日志等信息。 让不同形态的云原生应用运维更轻松方便。

了解应用工作台

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

增强 K8s 可观测性：API Server Tracing 特性进阶至 Beta

作者：mengjiao-liu



图片

在分布式系统中，很难弄清楚问题在哪里。 想象一个场景，这也是 Kubernetes 集群管理员最常遇到的问题，Pod 无法正常启动，这时候作为管理员， 我们会先去思考这可能是哪个组件出了问题，然后去对应的组件查找日志， 结果发现问题可能是另一个组件导致的， 这时候又去查找另外的日志，这还是最好的情况，我们能通过日志发现线索。有时候线索不是那么明显， 我们可能就需要苦思冥想，到底是哪里出了问题， 有时候需要依靠猜测，花费了很多时间进行定位问题， 这个时候就需要管理员对集群的各个组件都有比较全面的了解，这使得学习和排障的成本高昂。 这种情况下，**如果我们有分布式追踪，就可以清晰的看到是哪个组件出现了异常，快速定位到有问题的地方。**

分布式系统通常存在不确定性问题，或过于复杂而无法在本地重现。跟踪通过分解请求流经分布式系统时发生的情况， 使调试和理解分布式系统变得不那么令人生畏。 **分布式跟踪是一种工具，旨在在这些情况下提供帮助，而 Kubernetes API 服务器也许是能够调试的、最重要的 Kubernetes 组件。**

在 Kubernetes 中，API Server 是管理和调度所有集群资源的核心组件，它接收和处理来自各种客户端的请求， 并将它们转化为底层资源操作。因此，API Server 的稳定性和可观测性对 Kubernetes 的整体健康状态至关重要。

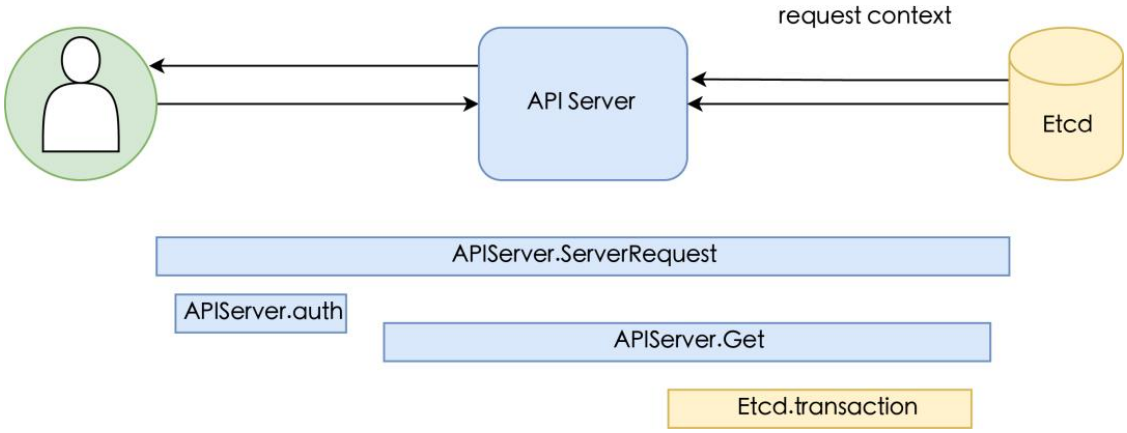
为了提高 Kubernetes API Server 的可观测性，帮助管理员更好地管理和维护 Kubernetes 集群。为此， **Kubernetes 引入了 APIServer Tracing， 该特性在 Kubernetes API Server 中添加更多的跟踪信息，并将其收集到后端收集器中。** 通过这些跟踪信息，管理员可以更容易地追踪请求的来源和流转，了解请求的处理时间和结果， 从而更容易地发现和解决问题。这些信息还可以用于性能优化和容量规划等方面。

接下来，让我们开始对此特性一探究竟吧。

Kubernetes API Server 追踪

设计详情：KEP APIServer Tracing #647

追踪概览图



图片

开启

- APIServerTracing 特性门控（v1.27+ 不再需要特性门控）
- tracing-config-file 配置文件

现状

负责的小组：由 sig instrumentation 负责

迭代版本：1.22 alpha，1.27 到达 beta 版本

追踪的组件：API → etcd tracing

演示

步骤：

1. 启动 Jaeger
2. 启动 APIServer tracing（包含启动 etcd）
3. 查看 Jaeger UI 观测追踪

启动 Jaeger 容器

Jaeger 是比较流行的分布式追踪 UI 工具，也是云原生计算基金会（CNCF）主持的第 7 个顶级项目（2019 年 10 月毕业）。 这里选择 Jaeger 作为数据收集和存储的后端，以及可视化数据的 UI。jaegertracing/all-in-one 是专为快速本地测试而设计的可执行文件，可启动 Jaeger UI、收集器、查询和代理，并带有内存存储组件。

```
docker run -d --name jaeger \
  -e COLLECTOR_ZIPKIN_HOST_PORT=:9411 \
  -e COLLECTOR_OTLP_ENABLED=true \
  -p 6831:6831/udp \
  -p 6832:6832/udp \
  -p 5778:5778 \
  -p 16686:16686 \
  -p 4317:4317 \
  -p 4318:4318 \
  -p 14250:14250 \
  -p 14268:14268 \
  -p 14269:14269 \
  -p 9411:9411 \
  jaegertracing/all-in-one:1.43
```

详情： https://www.jaegertracing.io/docs/1.43/getting-started/

启动 Kubernetes API Server tracing

启动 Kubernetes APIServer tracing 在本文提供了两种方式进行测试。 如果你是 Kubernetes 开发者，你可以直接在 Kubernetes 交互测试中直接测试； 如果你是 Kubernetes 集群管理员，你可以直接在集群中配置相关参数。

启动 Kubernetes 本地交互测试

测试文件：test/integration/apiserver/tracing/tracing_test.go

修改 API Server tracing 测试代码&配置

本地的交互测试代码需要修改下配置，以便将其收集的数据发送到 Jaeger。

#test/integration/apiserver/tracing/tracing_test.go:125

```
if err := os.WriteFile(tracingConfigFile.Name(), []byte(fmt.Sprintf(`
apiVersion: apiserver.config.k8s.io/v1beta1
kind: TracingConfiguration
samplingRatePerMillion: 1000000
endpoint: %s`, "0.0.0.0:4317")), os.FileMode(0755)); err != nil {
  t.Fatal(err)
 }
```

启动 etcd

需要配置的参数：

```
--experimental-enable-distributed-tracing=true
--experimental-distributed-tracing-address=0.0.0.0:4317
--experimental-distributed-tracing-service-name=etcd
```

修改代码：

#test/integration/framework/etcd.go:82

```
customFlags := []string{
```

```
  "--experimental-enable-distributed-tracing",
  "--experimental-distributed-tracing-address=0.0.0.0:4317",
  "--experimental-distributed-tracing-service-name=etcd",
 }

currentURL, stop, err := RunCustomEtcd("integration_test_etcd_data", customFlags, output)
```

运行测试

```
cd ./test/integration/apiserver/tracing
go test -run TestAPIServerTracing
```

在 Kubernetes 集群中配置 API Server tracing

这里以 kubeadm 安装的 Kubernetes 集群为例。

在 kube-apiserver.yaml 配置清单中配置特性门控 APIServerTracing=true（1.27 及以上版本不再需要配置此特性门控）。

配置 tracing-config-file 文件,此处我们将此文件保存在 /etc/kubernetes/apitracing-config.yaml。

```
apiVersion: apiserver.config.k8s.io/v1beta1
kind: TracingConfiguration
endpoint: 10.6.9.3:4317
samplingRatePerMillion: 100000  #采样频率，根据自身需要设置
vim /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --feature-gates=APIServerTracing=true
    - --tracing-config-file=/etc/kubernetes/apitracing-config.yaml
```

保存退出即可，kubelet 会自动重启 APIServer。

在 etcd.yaml 配置清单中配置以下参数：

```
vim /etc/kubernetes/manifests/etcd.yaml
spec:
  containers:
    - command:
        - etcd
        - --experimental-distributed-tracing-address=<JaegerIP:4317>
        - --experimental-distributed-tracing-service-name=etcd
        - --experimental-enable-distributed-tracing=true
```

保存退出即可，kubelet 会自动重启 etcd。

查看 Jaeger

这时候我们可以访问 Jaeger 了。地址：http://:16686/ 在 Jaeger 界面我们可以清晰的看到请求的追踪路径。



图片

青色行来自 API 服务器，包括对 /api/v1/nodes 的服务请求，并向 ETCD 发出 grpc Range RPC。黄色线来自 ETCD 处理 Range RPC。

结语

SIG instrumentation 正在积极推动 Kubernetes 组件可追踪，现在 APIServer Tracing 和 kubelet Tracing 在 Kubernetes v1.27 都已到达 Beta 版本，敬请期待！

Kubernetes 本地开发工具比较：Telepresence、Gefyra 和 mirrord

作者: Eyal Bukchin (MetalBear)

译者: Michael Yao (DaoCloud)

Kubernetes 的开发周期是一个不断演化的领域，有许多工具在寻求简化这个过程。 每个工具都有其独特的方法，具体选择通常取决于各个项目的要求、团队的专业知识以及所偏好的工作流。

在各种解决方案中，我们称之为“本地 K8s 开发工具”的一个类别已渐露端倪，这一类方案通过将本地运行的组件连接到 Kubernetes 集群来提升 Kubernetes 开发体验。 这样可以在云环境中快速测试新代码，避开了 Docker 化、CI 和部署这样的传统周期。

在本文中，我们将比较这个类别中的三个解决方案：Telepresence、Gefyra 和我们自己的挑战者 mirrord。

Telepresence

Telepresence 是这类工具中最早也最成熟的解决方案， 它使用 VPN（或更具体地说，一个 tun 设备）将用户的机器（或本地运行的容器）与集群的网络相连。它支持拦截发送到集群中特定服务的传入流量，并将其重定向到本地端口。 被重定向的流量还可以被过滤，以避免完全破坏远程服务。 它还提供了一些补充特性，如支持文件访问（通过本地挂载卷将其挂载到 Pod 上）和导入环境变量。Telepresence 需要在用户的机器上安装一个本地守护进程（需要 root 权限），并在集群上运行一个 Traffic Manager 组件。此外，它在 Pod 上以边车的形式运行一个 Agent 来拦截所需的流量。

Gefyra

Gefyra 与 Telepresence 类似，也采用 VPN 连接到集群。 但 Gefyra 只支持将本地运行的 Docker 容器连接到集群。 这种方法增强了在不同操作系统和本地设置环境之间的可移植性。 然而，它的缺点是不支持原生运行非容器化的代码。

Gefyra 主要关注网络流量，不支持文件访问和环境变量。 与 Telepresence 不同，Gefyra 不会改变集群中的工作负载， 因此如果发生意外情况，清理过程更加简单明了。

mirrord

作为这三个工具中最新的工具，mirrord 采用了一种不同的方法， 它通过将自身注入到本地二进制文件中（在 Linux 上利用 LD_PRELOAD，在 macOS 上利用 DYLD_INSERT_LIBRARIES）， 并重写 libc 函数调用，然后代理到在集群中运行的临时代理。 例如，当本地进程尝试读取一个文件时，mirrord 会拦截该调用并将其发送到该代理， 该代理再从远程 Pod 读取文件。这种方法允许 mirrord 覆盖进程的所有输入和输出，统一处理网络访问、文件访问和环境变量。

通过在进程级别工作，mirrord 支持同时运行多个本地进程，每个进程都在集群中的相应 Pod 上下文中运行， 无需将这些进程容器化，也无需在用户机器上获取 root 权限。

摘要

比较 Telepresence、Gefyra 和 mirrord

Telepresence

Gefyra

mirrord

集群连接作用域

整台机器或容器

容器

进程

开发者操作系统支持

Linux、macOS、Windows

Linux、macOS、Windows

Linux、macOS、Windows (WSL)

传入的流量特性

拦截

拦截

拦截或镜像

文件访问

支持

不支持

支持

环境变量

支持

不支持

支持

需要本地 root

是

否

否

如何使用

CLI

Docker Desktop 扩展

CLI

Docker Desktop 扩展

CLI

Visual Studio Code 扩展

IntelliJ 插件

结论

Telepresence、Gefyra 和 mirrord 各自提供了独特的方法来简化 Kubernetes 开发周期， 每个工具都有其优缺点。Telepresence 功能丰富但复杂，mirrord 提供无缝体验并支持各种功能， 而 Gefyra 则追求简单和稳健。

你的选择应取决于项目的具体要求、团队对工具的熟悉程度以及所需的开发工作流。 无论你选择哪个工具，我们相信本地 Kubernetes 开发方法都可以提供一种简单、有效和低成本的解决方案， 来应对 Kubernetes 开发周期中的瓶颈，并且随着这些工具的不断创新和发展，这种本地方法将变得更加普遍。

5.0 资源管理能力介绍

随着企业规模的扩大，管控的资源必定日渐庞杂复：成千上万台的服务器、机器设备、电子器件等有形固定资产；日常办公软件、专业财务/人事/OA/项目管理/开发等专业软件；管理层、雇员、职员等资源。

如何高效管理这些资源?



resource

基于资源的管理能力本质上是建立一套与企业相关的，基于资源使用的关系结构。在 DCE 5.0 中主要通过工作空间与层级模块实现，该模块具有全局一致性的特点，方便您基于此关系结构对平台上多个子模块内的各种资源进行高效规划和管理。

工作空间与层级模块中有两个概念：层级和工作空间。 层级可以映射为企业中的项目、环境、供应商等概念，方便您基于企业的业务或生态环境构建出体现资源关系的企业层级关系。 同时层级具有权限继承能力，上级层级管理员能够创建和管理下级层级及工作空间，有效解决层级结构中的一次性授权问题。 工作空间与资源直接关联，支持资源独享、资源共享等多维度，跨集群的资源管理能力，满足数

百个集群下的资源快速授权和资源灵活调配需求。 同时，工作空间与层级模块还支持移动层级和移动工作空间功能，随时应对企业层级关系调整等带来的资源调整。

资源管理的用户交互

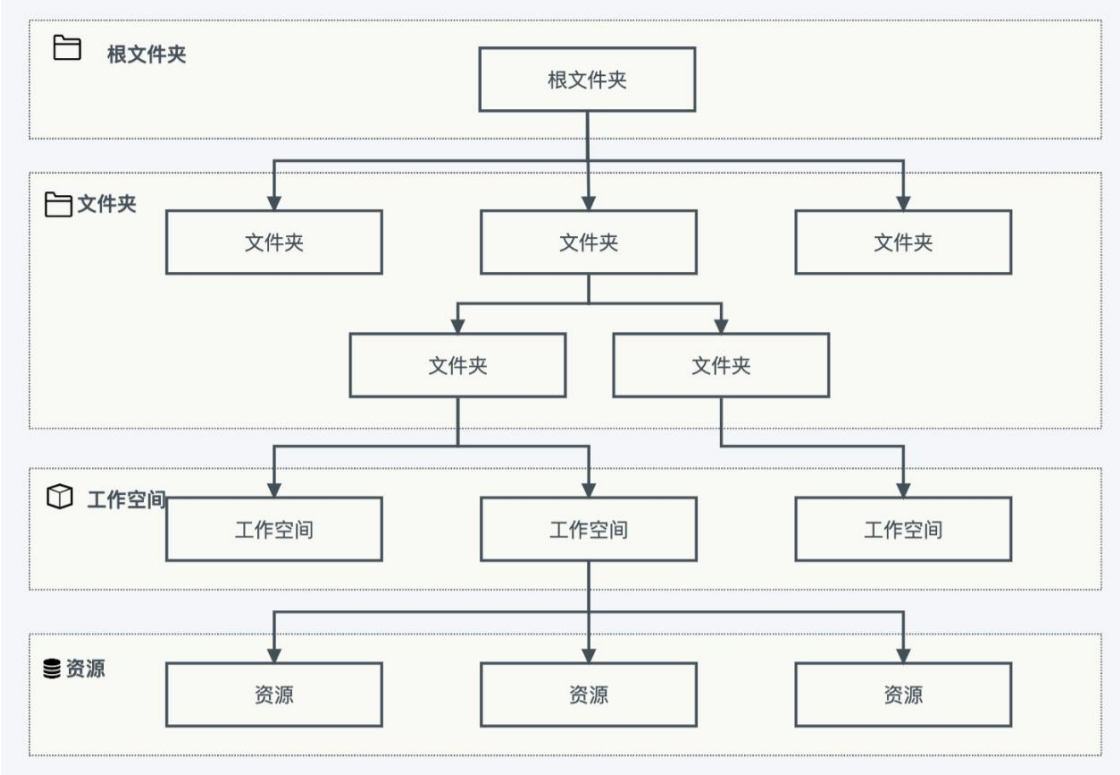
资源管理模块是一个具有层级的资源隔离和资源分组模块，主要解决资源统一授权、资源分组以及资源限额问题。 资源管理模块有两个概念：层级和工作空间。在资源管理模块中您可以通过层级构建企业业务层级关系，通过工作空间中的资源组和共享资源管理资源，使资源管理模块的用户（用户组）之间能够共享工作空间中的资源。

资源处于资源管理模块层级结构的最低层级，资源包括 Cluster、Namespace、Pipeline、网关等。 所有这些较低层级的资源的父级只能是工作空间，而工作空间作为资源容器是一种资源分组单位。 工作空间通常代指一个项目或环境，每个工作空间的资源相对于其他工作空间中的资源时逻辑隔离的。 您可以通过工作空间中的授权，授予用户（用户组）同一组资源的不同访问权限。

从层次结构的底层算起，工作空间位于第一层，且包含资源。 除共享资源外，所有资源有且仅有一个父项。所有工作空间也有且仅有一个父级层级。

层级是在工作空间基础之上的进一步分组机制，具有层级结构。一个层级可以包含工作空间、其他层级或两者的组合，能够形成树状的组织关系。 借助层级您可以映射企业业务层级关系，按照部门对工作空间进行分组。层级不直接与资源挂钩，而是通过工作空间间接实现资源分组。 层级有且仅有一个父级层级，根层级是层次结构的最高层级，因此没有父级，层级和工作空间均挂靠到根层级下。

同时，用户（用户组）在层级中能够通过层级结构继承来自于父项的权限。 用户在层次结构中每个节点权限来自于直接在该节点获取的权限和继承其父项权限的组合结果，权限之间是加合关系不存在互斥。



resource

资源通过工作空间进行分组，而工作空间中存在两种分组模式，分别是资源组和共享资源。 一个资源只能加入一个资源组，资源组与工作空间一一对应，资源被加入到资源组后，工作空间的所有者将获得资源的管理权限，相当于该资源的所有者。 而对于共享资源来说，多个工作空间可以共享同一个或者多个资源。资源的所有者，可以选择将自己拥有的资源共享给工作空间使用，一般共享时资源所有者会限制被共享工作空间能够使用的资源额度。 资源被共享后，工作空间的所有者仅具有资源限额下的资源使用权限，无法管理资源或者调整工作空间能够使用的资源量。 同时共享资源对于资源本身也具有一定的要求，仅具有层级的资源能够被共享，比如 Cluster-Namespace，Cluster 资源所有者能够将 Cluster 资源分享给不同的工作空间使用，并且限制工作空间在此 Cluster 上的使用额度，工作空间所有者在资源限额内能够创建多个 Namespace，但是 Namespace 的资源额度总和不能超过 Cluster 在该工作空间的资源限额。对于 Kubernetes 资源，当前能够分享的资源类型仅有 Cluster。

申请社区免费体验

资源管理要素

安装 DCE 5.0 后，在初始化阶段平台会自动为用户创建根层级，根层级位于资源管理模块资源层级结构的顶层，没有父级。资源关系依据根层级向下分布，最多支持 5 级层级。

层级

平台管理员根据企业现有的层级结构在根层级下创建新的层级。 层级可以映射为企业的项目、环境、供应商等概念，层级具有层级结构和权限继承，是资源管理模块资源层级结构的节点。 一个层级可以包含工作空间、其他层级或两者的组合。新创建的层级必须具有父级，且下级层级和工作空间会继承父级文件的权限，因此用户在层级或工作空间的实际权限是来自于继承父级层级的权限和在该节点获得的权限之和。

工作空间

工作空间是资源管理模块中的资源隔离单元，具有资源组和共享资源两种模式，与资源直接关联。 工作空间有且只有一个父级层级，工作空间及其下的资源会继承父级层级的权限。

在 DCE 5.0 中，由于资源存在不同的授权方式，因此衍生出了强依赖于工作空间授权的资源，如网关、Pipeline 等；以及可以选择性绑定到工作空间的资源，如 Cluster、Namespace。

应用工作台、微服务引擎、服务网格的资源由于以工作空间为顶层概念，资源完全依赖于工作空间进行授权，因此资源会被自动绑定到某个工作空间，工作空间的所有者即是资源的所有者。 此类资源无需手动绑定，资源也无需在工作空间的资源组进行显示，用户（用户组）使用资源的前提是被授予工作空间的相应权限。 而对于 Cluster、Namespace 资源，由于容器管理有独立的授权模块，资源所有者有选择性绑定的权利。 资源被手动绑定到工作空间的资源组，表示允许工作空间所有者能够管理使用该资源，此时工作空间所有者相当于资源所有者，被绑定的资源将在工作空间的资源组进行显示，并可以随时解绑。 资源被绑定到到工作空间

的共享资源，表示允许资源所有者在资源限额内使用资源，当前能够被共享的资源类型仅有 Cluster。

资源限额

共享资源不意味着被共享者可以无限的使用被共享的资源，一般资源所有者会限制工作空间在该资源上能够使用的额度上限。 比如，当 Cluster 被共享到工作空间后，工作空间所有者就获得了在该 Cluster 上创建 Namespace 的权利。 若 Cluster 所有者设置了工作空间在该 Cluster 上的资源额度上限为 Quota=100，那么 Namespace 的 Quota 总和不能超过 100。 同样的，当多个 Cluster 被共享到该工作空间时，那么每个 Cluster 所有者都可以设置工作空间在各个 Cluster 上的额度使用上限。 另外，除了 Quota 外，资源所有者还可以通过 CPU、内存、Secret 等多个维度对资源额度上限进行限制。 （资源所有者能够共享资源的前提：同时是 Cluster 和工作空间的所有者）。

移动层级和工作空间

部门关系或项目关系的调整随时可能使资源的授权关系发生变化，为了能够随机应变，移动层级和移动工作空间功能应用而生。 以移动层级为例，由于资源、工作空间、层级的权限具有继承关系，因此移动层级后，原父级层级将失去对该层级以及其子层级、工作空间、空间中资源的管理权限。 新的父级获得对该层级以及其子层级、工作空间、空间中资源的管理权限。 对于层级本身以及其下的子层级授权关系没有发生任何变化。工作空间同理。

谁需要资源管理

需要对 K8S、微服务引擎、服务网格等资源进行管理，且资源具有一定规模的企业用户

解决什么问题

1. 企业人员众多同时存在多层级结构，如多级供应商，多级部门等，而资源往往聚焦在少部分运维人员手中，资源下发与资源管理耗费大量的人力物力
2. 企业层级结构多变，无法灵活应对部门调整带来的人员权限和资源归属变化
3. 以项目为驱动力的企业，对于同一批人，同样的资源需要进行多次授权

4. 对于同一种资源，多人拥有相同的权限存在安全风险

适用场景

1. 探索更为便捷的资源下发与资源管理方式，减少人力物力消耗，缩减成本
2. 资源管理方式映射企业层级结构，按照层级进行分权自制，更为灵活的应对企业变化，同时接入更多供应商，扩大业务范围
3. 更为精细化的资源权限控制，同一种资源设置管理、使用、只读等多种权限，为不同的成员下发不同的权限，将安全风险降到最低，进一步降低事故率

FAQs

1. 什么场景下需要使用层级?

   答：层级可以映射为企业的项目、环境、供应商等概念，具有层级结构和权限继承能力。可以有效应对企业遇到的多级供应商的资源分配问题。

2. 什么场景下需要使用工作空间?

   答：资源与工作空间直接关联，同时资源会继承工作空间的权限。 当资源数量很大时，通过工作空间对资源进行统一授权能够有效减少资源运维的工作量。 另外在 DCE 5.0 中，应用工作台、微服务引擎、服务网格中的资源依赖于工作空间授权，因此创建此类资源的前提是拥有工作空间的相应权限。

3. 层级、工作空间及资源的关系?

   答：资源与工作空间直接关联，资源权限继承工作空间的权限；而工作空间一定归属于某个层级，且工作空间权限继承层级权限，所以拥有层级权限的用户相当于对其下工作空间中的资源有相应权限，但是资源不与层级直接绑定。

4. 移动层级或者工作空间，资源和权限是否会发生变化?

答：资源绑定在工作空间下，会随着层级或者工作空间的移动而移动。 因为资源继承工作空间的权限，工作空间继承层级的权限，所以移动工作空间或者层级后会改变原有的权限继承关系，原来层级或者工作空间的上级失去对层级或者工作空间的管理权限，目标层级及其上级开始拥有对该层级或者工作空间的管理权限。

5. 谁能够创建 Folder?

   答：Admin、Workspace Owner 和上一级 Folder Admin。

6. 谁能够创建 Workspace?

   答：Admin、Workspace Owner、Folder Admin。

7. 想要获得 微服务治理、应用工作台、服务网格权限找谁授权?

   答：微服务治理、应用工作台、服务网格下的资源均自动绑定 Workspace，且权限强依赖于 Workspace，需要通过获得 Workspace 权限继而获取相关的资源权限。 能够进行 Workspace 授权的有 Admin、Workspace Owner、Folder Admin、Workspace Admin。

了解服务网格

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

CNCF 平台工程白皮书

!!! quote

引子（译者 [panpan0000](https://github.com/panpan0000）注）：2022 年，“平台工程”这个概念很火热，也在 Gartner 的炒作周期曲线上。
还有言论说“DevOps 已死，平台工程才是未来“。开发者不愿意和基础设施打交道，企业发展又需要自己的基础设施。
“平台工程”统一这两个矛盾点，或者说“平台工程”是 DevOps 的下一站。

![cncf curve](images/cncf-blog01.png)

![cncf comments](images/cncf-blog02.png)

但关于平台工程的诠释，一直比较模糊。几天前，CNCF 的应用交付技术小组（App-Delivery TAG）（TAG：Technical Advisory Groups）
发布了[《平台白皮书》](https://tag-app-delivery.cncf.io/whitepapers/platforms/)
让我们一起来看看吧。

概述

受到 DevOps 成功推动跨职能合作的启发，平台工程（Platform Engineering）也开始在企业中作为明确的协作形式而涌现。 “平台”所提供的基础能力、框架和经验，提供便利并加速着内部用户的工作（如应用程序开发者、数据科学家和信息处理者）。 特别是在云计算方面，“平台”已经帮助企业实现了云计算长期承诺的价值： 如快速产品发布、跨基础设施的可移植性、更安全更弹性的产品，以及更高的开发者生产力。

本文旨在帮助企业技术决策者们（企业管理者、企业架构师、平台组管理者），去调研、规划、推广内部的云计算平台。 我们相信”平台”对企业的价值流（Value Streams）会产生重要影响，但因为这个影响是间接性的， 所以管理者们的共识

和支持，对于平台团队的可持续性和成功是至关重要的。 在本文中，我们将讨论几个要点：定义、衡量、实施和最大化平台的价值。

目录：

1. 为什么需要一个平台
2. 平台是什么
3. 一个成功的平台应具备的属性
4. 一个成功的平台团队应具备的属性
5. 实施平台的挑战
6. 如何度量平台的成功
7. 平台的能力

为何需要一个平台?

平台和平台工程在云计算世界中是一个热门话题。在深入平台构建的定义、技术和度量之前，让我们首先探讨这火热的平台概念所提供的价值。

过去二三十年间的软件流程改进，为开发者提供了丰富的基础架构（如计算、网络、存储、开发者服务如构建、测试、交付和可观测性等）， 显著提高了开发的敏捷性。但这种自治和流程改进，也逐渐使得基础服务的运营职责，（从基础团队）左移到了产品开发团队， 迫使他们在基础架构方面花费越来越多的时间和认知精力，减少他们为组织（和产品）产生价值的时间。

为了能让开发团队重新聚焦于他们的核心任务，和减少组织内重复工作，这个目标驱动着企业去实施云原生计算平台。 通过投资这个平台，企业可以收获：

1. 减轻产品团队的认知负担，从而加速产品的开发和交付。
2. 通过专家的配置和管理，提高依赖于平台能力的产品的可靠性和弹性。
3. 在团队之间重用、共享平台工具和知识，加速产品的开发和交付。
4. 通过治理平台能力，治理与其相关的用户、工具和流程，来减少产品和服务在安全、监管和功能问题上的风险。

5. 充分利用公共云和托管服务提供商，实现降本增效， 同时保持对用户体验的控制。

这些好处来源于三方面原因：一方面是因为少量平台团队可服务于许多产品团队，从而放大了他们的影响； 第二个方面是因为平台团队整合了常见功能的管理，并促进了治理；第三个方面是因为平台团队强调用户界面和体验优于一切。

平台专家团队不仅减轻了产品开发团队的常规工作，还优化了这些产品中使用的平台能力。 平台团队还能维护了一组公共的约定范式、知识和工具，在这共通的基础上，开发者能够快速为其他团队和产品做出贡献。 这种模式还能（通过技术手段）把治理和管控的目的融入模板、模式和能力中。 最后，因为平台团队在公有云或私有化供应商之上，集中构筑一致体验， 所以能对那些基础但无差别的服务的使用更加高效：如数据库、身份访问、基础架构操作和应用程序生命周期。

什么是平台

云原生计算平台，是根据平台用户的需求能力的集成。它确保了一致的体验，集成通用应用场景的典型服务能力。 良好的平台提供一致的用户体验来使用其服务，例如 Web 门户、项目模板和自助式 API。

据 Atlassian 的说法： “平台团队创建的能力，可以被众多流程一致的产品开发团队使用，而几乎没有额外的开销..... 平台团队最小化了团队的资源和认知负担....平台团队可以将不同产品的体验统一起来。”

Martin Fowler 和 Evan Bottcher 说过，“数字平台是一组自助式 API、工具、服务、知识和支持的基础，这些都被组织到一个统一的内部产品中。自治的开发团队可以利用平台，更快交付产品功能，减少协调。”

平台支持的特定能力和场景，应由利益相关者和用户的需求确定。虽然平台“提供”这些能力，但需要注意的是，平台团队不应总是自己“实现”它们。 可以让托管服务提供商或专门的内部团队来维护和实现，而平台则是提供一致性的“最薄合理层”，以满足组织的要求。 例如，一个非常简单的“平台”：可以是一个 wiki 页面， 包含从提供商获取能力的方法和流程。

由于这些平台的目标用户，不多不少，正是企业的内部使用者，因此我们通常将它们称为“内部”平台。

平台对云原生架构息息相关，因为它们比以前的范例，更分离支持能力与特定于应用程序的逻辑。 在类似云的环境中，资源和能力通常是独立管理的，并与业务紧密集成； 这些资源可能包括数据库和对象存储、消息队列和代理、可观察性采集器和仪表板、用户目录和身份验证系统、任务执行器和协调器等等。 内部平台不单单向企业团队提供这些资源能力，并以一种能够被应用简单集成的方式来提供。

平台成熟度

最初，内部平台能提供单一服务的能力和一致性体验：比如 CI 的流水线 runner、数据库或密钥（secret）存储。 随着它们的成熟，内部平台还提供了这些服务的“组合”：比如用于关键场景（如 Web 应用程序开发、如数据分析（即 MLOps））的自助式模板。

平台的使用中，可能会经历以过程：

1. 产品开发人员可以按需提供能力，并立即使用它们来运行系统，例如计算、存储、数据库或身份认证信息。
2. 产品开发人员可以按需提供服务空间，并使用它们来运行管道和任务，存储 artifacts 和配置，和收集遥测数据。
3. 第三方软件的管理员，可以按需提供依赖项，例如数据库，并轻松安装和运行之。
4. 产品开发人员可以从模板（包括运行时和开发时所需的服务）中，按需提供完整的环境，例如 Web 开发或 MLOps。
5. 产品开发人员和经理可以通过自动化的工具和标准仪表板，观察已部署服务的功能、性能和成本。

内部平台通过单个能力或能力组合，提供一致、符合要求的体验，最终使其用户更轻松、更高效地交付有价值的产品。

平台的属性

在定义了平台是什么，以及为什么组织希望构建一个平台之后，让我们识别一些影响平台成功的关键属性。

1. 平台即产品（Platfrom as a Product）：平台是为了满足其用户的需求而存在，类似于任何其他软件产品，应基于这些需求进行设计和演进。 平台应提供必要的能力，优先支持大多数产品团队的常见需求，而非优先考虑仅某个团队使用的具体能力，以最大化提供的价值。
2. 用户体验：平台应通过一致的界面，并专注于用户体验。平台应努力满足不同用户的需求，这可能意味着 GUI、API、命令行工具、IDE 和门户都要有。例如，平台通常提供部署应用程序的能力。开发人员可以通过 IDE 来使用这种能力，测试人员可以使用命令行工具， 而产品经理可以使用基于 GUI 的 Web 门户。
3. 文档和快速入门：文档是成功软件产品的关键方面。为了能够使用平台的产品，用户需要文档和示例。 平台应随着适当的文档一起交付，以满足其用户的需求。它还应提供工具，以加速使用平台服务的新项目的上手。 例如，平台可以提供一个可复用的工作流，为在 Kubernetes 上构建、扫描、测试、部署和观察 Web 应用程序。 这样的典型工作流，可以作为一个新手上手模板和文档，这个组合通常被称为“黄金路径”。
4. 自助式：平台应是自助式的。用户必须能够自主且自动地请求和接收能力。这是允许一个平台团队能支撑许多产品团队并按需“扩容”的关键。平台能力应按需提供，并仅需最少的手动干预。 例如，用户应能运行命令行工具或在 Web 门户上填写表单来申请一个数据库，并能看到它的 locator 和认证信息。
5. 减少认知负担：平台的一个重要目标是减少产品团队的认知负荷。平台应封装实现细节，并隐藏可能从其架构中产生的任何复杂性。 例如，平台可能将某些服务委托给云提供商，但不应向用户暴露此类细节。 同时，平台应允许用户按需配置和查看某些服务。用户不应负责运维底层服务。 例如，用户经常需要一个数据库，但他们不应该管理数据库服务器。

6. 可选和可组合：平台旨在使产品开发更加高效，而不是”成为障碍“。 平台应该是可组合的，并使产品团队仅使用部分能力。当平台不具备某些能力时，产品团队还应能自己部署和使用外部组件。 例如，如果平台不提供 Graph 数据库，而这是产品所需的，那么产品团队应能够自行提供和运维 Graph 数据库。
7. 默认安全：平台应默认情况下是安全的，并提供符合组织定义的规则，和标准的合规性和验证能力。

平台团队的属性

平台团队负责平台功能的接口和体验，例如 Web 门户、自定义 API 和“黄金路径模板”。 一方面，平台团队与实施基础设施和支持服务的团队合作，以定义一致的体验；另一方面，他们与产品和用户团队合作。收集反馈并确保这些体验符合要求。

以下是平台团队应负责的工作：

1. 研究平台用户需求并规划 Roadmap
2. 在企业内宣贯和传播平台的价值
3. 管理和开发界面，包括门户、API、文档、模板、以及 CLI 工具

最重要的是，平台团队必须了解平台用户的需求，以告知并不断改进其平台提供的功能和界面。 了解用户需求的方法包括用户访谈、交互式黑客马拉松、问题跟踪和问卷调查，以及通过可观察性工具直接观察使用情况。 例如，平台团队可以发布一个表单供用户提交功能申请，主持 Roadmap 会议以分享即将推出的功能，并分析用户的使用方式以调整优先级。

收集外部反馈和周到设计是平台团队工作的一方面；另一方面是主动对外营销和宣传。 如果该平台真正是根据用户需求构建的，那么这些用户将很高兴使用所提供的功能。 平台团队可以通过内部营销活动，包括公告、引人入胜的演示以及定期反馈和沟通会议，来提高用户采用度。 这里的关键是在用户所在的地方与他们会面，并让他们踏上与平台互动并从中受益的旅程。

平台团队不一定运维计算、网络、存储或其他服务。实际上一个内部平台应尽可能依赖外部提供的服务和能力； 平台团队只有在无法从托管供应商或内部基础设施团队那里获得这些能力时，才应构建和维护自己的能力。 相反，平台团队主要负责接口（即 GUI、CLI 和 API）以及平台提供的服务和功能的用户体验。

例如，平台中的网页可能会有个按钮来为应用程序提供身份认证；而该功能的实现可能是通过云托管的身份服务。 内部平台团队可能会管理网页和 API，但不会管理实际的服务实现。平台团队通常应仅在其他地方无法提供所需功能时才考虑创建和维护自己的功能。

平台的挑战

虽然平台承诺了很多价值，但它们也带来了如下挑战，实施者应牢记这些挑战：

1. 平台团队必须像对待产品一样对待他们的平台，并与用户一起开发它们
2. 平台团队必须谨慎选择优先级，和初始合作的应用开发团队
3. 平台团队必须寻求企业领导层的支持并展示对价值流的影响

也许最重要的是将平台视为面向客户的产品，其成功直接取决于其用户和产品的成功； 因此，平台团队与应用程序团队等用户合作，对平台的功能和用户体验进行优先排序、规划、实施和迭代至关重要。 平台团队在没有反馈的情况下发布功能和体验，或者依靠自上而下强制推动采用，几乎肯定会遭到用户的抵制和不满，并丢失很多承诺的价值。为了解决这个问题，平台团队应该从一开始就要有产品经理角色，来分享路线图、收集反馈并普遍理解和代表平台用户的需求。

在采用平台时，首先选择正确的功能和体验至关重要。经常需要且区分度不高的功能， 如 CI Pipeline、数据库和可观察性，可能是一个很好的起点。 平台团队也可以选择首先关注少数优秀的应用程序团队。来自他们的反馈改善了首次平台体验； 他们未来还能为平台宣传给后续采用者。

最后，在大型企业中，获得领导的支持至关重要。许多企业领导者将 IT 基础架构视为与其主要价值流完全脱节的支出， 并可能试图限制分配给 IT 平台的成本和

资源。从而导致实施不力、承诺无法兑现和挫败感。 为了缓解这种情况，平台团队需要证明他们对产品和价值流团队的直接影响和关系（见前两段）， 将平台团队展示为产品团队的战略合作伙伴，共同为客户提供价值。

如何赋能平台团队

平台团队面临着许多导致认知负担的挑战。就像同行的应用程序团队一样，这一挑战随着产品的用户的数量和多样性而增加。

重要的是将平台团队的精力集中在他们特定业务所独有的经验和能力上。减轻平台团队负担的方法包括：

1. 寻求在托管供应商的实施之上，构建最薄的可行平台层
2. 利用开源框架和工具包，创建供应用程序团队使用的文档、模板和能力组合
3. 确保平台团队配备适合其领域和客户数量的团队成员

如何衡量平台的成功

企业很希望衡量他们的平台投入，是否能实现上面讨论的价值和属性。 此外，在本文中，我们一直强调将内部平台视为产品的重要性，良好的产品管理取决于对产品的定量和定性的衡量。 为了满足这些要求，内部平台团队应不断收集用户反馈，并衡量用户活动。

然而，与内部平台的其他方面一样，平台团队应该尽最小的努力来收集他们需要的反馈。 我们会建议去建立一些指标（但在初期，对用户行为进行简单的调查和分析可能最有用）。

如下，是一些有助于企业和平台团队了解其平台影响的指标类别：

用户满意度和生产力

许多平台追求的第一质量是改善用户体验以提高生产力。反映用户满意度和工作效率的指标包括：

- 活跃用户和保留：包括提供的功能数量和用户增长/流失

- 净推荐值（Net Promoter Score, NPS）或其他衡量用户对产品满意度的调查
- 开发人员生产力的指标，例如 SPACE 框架 4 中讨论的指标

组织效率

从许多平台寻求的另一个好处是:有效地为大量用户群提供共同需求。这靠提供用户自助服务并减少手动步骤和人工干预来实现的， 同时实施“策略管理”以确保安全性和合规性。要衡量平台在减少重复工作方面的收效，请考虑以下措施：

- 从请求到实现服务或功能（例如数据库或测试环境）的延迟
- 构建全新服务并将其部署到生产中的延迟
- 一个新用户提交 Pull Request 到产品的耗时

产品和功能交付

内部平台的最终目标是更快地为客户提供商业价值，因此它对企业自身产品和功能发布的收效，是平台最好的成果展示。 谷歌的 DevOps 研究与评估（DORA）研究所建议 5 跟踪以下指标：

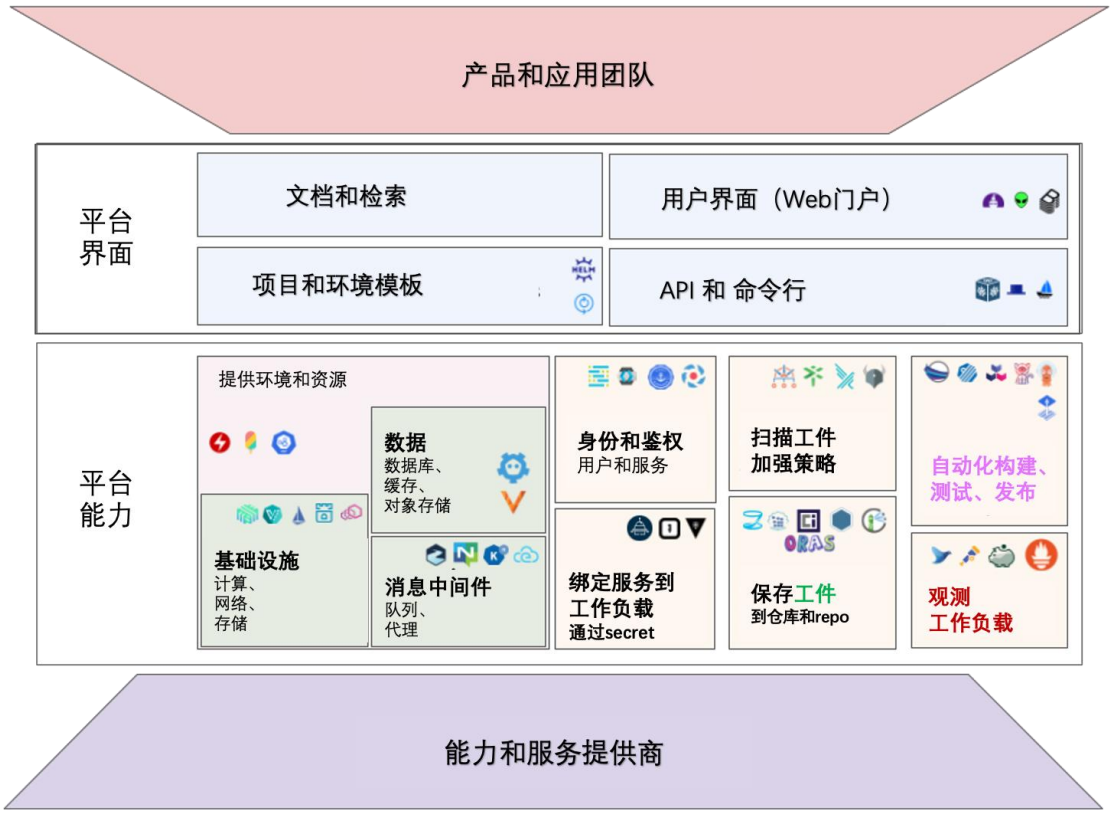
- 部署频率
- 更改的准备时间（Lead time for changes）
- 故障后恢复服务的时间
- 更改失败率

通常，平台团队的一个关键目标，是使基础设施等 IT 功能与企业的价值流（其产品）保持一致。 因此，归根结底，组织的产品和应用程序的成功，才是衡量平台成功与否的真正标准。

平台能力

正如我们所描述的，云原生计算平台提供并组合了来自许多提供商的功能和服务。这些提供商可能是同一企业内的其他团队或第三方，如云服务提供商。 简而言之，平台是从底层“能力提供者”到平台用户（如应用程序开发人员）的桥梁； 并

在此过程中实施和执行安全、性能、成本治理和一致体验所需的实践。 下图说明了产品、平台和能力提供者之间的关系。



platform components

我们在本文中重点讨论了如何构建一个好的平台和平台团队；现在，在最后一节中，我们将描述平台实际可能提供的功能。 此列表旨在指导平台如何建设，并包括云原生应用所需的功能。正如我们一直说的，一个好的平台反映了其用户的需求， 因此最终平台团队应该和用户一起，来挑选和排序平台将要实现的能力。

能力可能包含几个“特征”，诠释父能力的一些具体属性。例如，可观察性（作为一个能力）可能包括用于收集和发布指标、 跟踪和日志以及用于观察成本和能源消耗（这些特征）。考虑组织中每个功能或方面的需求和优先级。 以后的 CNCF 出版物可能会进一步扩展每个领域。

以下是构建云原生计算平台时需要考虑的能力领域：

1. Web 门户：查看和部署产品能力
2. API 和 CLI：自动配置产品和功能
3. “黄金路径”模板和文档：呈现使用的最佳姿势
4. 构建和测试的自动化服务
5. 交付和验证的自动化服务
6. 开发环境：托管 IDE 和远程连接工具等
7. 可观察性：使用采集器和 dashboard 等方式，进行包括功能性、性能和成本的观察
8. 基础设施服务：包括计算运行时、可编程网络以及块和卷存储
9. 数据服务：包括数据库、缓存和对象存储
10. 消息中间件和事件服务：包括 broker 代理、消息队列和 Event Fabrics
11. 身份和密钥管理服务：例如身份和授权、证书和密钥颁发以及静态密钥存储
12. 安全服务：包括代码和 artifact 的静态分析/运行时分析，还有 Policy 增强
13. 工件（artifact）存储：包括容器镜像和特定语言包、自定义二进制文件和库，以及源代码的存储

如下列表，通过关联一些 CNCF 或者 CDF 的项目，帮助读者理解这些能力：

能力	**能力的描述**	**范例 CNCF/CDF 的项目**
Web 门户	发布：文档、服务目录（service catalogs）和项目模板；发布遥测能力	Backstage, Skooner, Ortelius
API	结构化、自动化创建、升级、删除、观测各种服务	Kubernetes, Crossplane, Operator Framework, Helm, KubeVela
黄金模板和文档	用于快速项目开发的优秀集成代码和功能的模板组合	ArtifactHub

能力	能力的描述	范例 CNCF/CDF 的项目
自动化构建和测试	自动化构建和测试软件产品	Tekton, Jenkins, Buildpacks, ko, Carvel
自动化发布和验证	自动化发布和过程观测	Argo, Flux, Keptn, Flagger, OpenFeature
开发环境	支持应用程序和系统的研发	Devfile, Nocalhost, Telepresence, DevSpace
应用观测	收集、分析遥测数据，汇总发布给相关者	OpenTelemetry, Jaeger, Prometheus, Thanos, Fluentd, Grafana, OpenCost
基础设施	运行应用程序代码、连接应用组件，并持久化数据	Kubernetes, Kubevirt, Knative, WasmEdge CNI, Istio, Cilium, Envoy, Linkerd, CoreDNS Rook, Longhorn, Etcd
数据服务	持久化结构化数据	TiKV, Vitess, SchemaHero
消息队列和 Event 服务	使应用程序能够彼此异步通信	Strimzi, NATS, gRPC, Knative, Dapr
身份和密	确保工作负载具有使用资源和功能的认证和密钥能力，使服务能够向	Dex, External Secrets, SPIFFE/SPIRE, Teller, cert-

能力	能力的描述	范例 CNCF/CDF 的项目
钥服务	其他服务标识自己	manager
安全服务	观察运行时行为并报告/修复异常；验证构建和工件的漏洞；根据企业要求限制平台上的活动；通知和/或修复异常	Falco, In-toto, KubeArmor, OPA, Kyverno, Cloud Custodian
工件（artifacts）存储	存储、发布和保护用于生产目的的构建工件；缓存和分析第三方工件；存储源代码	ArtifactHub, Harbor, Distribution, Porter

词汇表

也可以参考 https://glossary.cncf.io/

平台 聚合了开发和交付软件时所需的能力。根据其所应用的场景，一个平台可能被命名为“开发者平台”、“交付平台”、“应用平台”、甚至“云平台”。 旧术语“PaaS（平台即服务）”也很有影响力。

平台 通过提供和管理通用能力，使开发人员和运营商，可以更快地交付应用和服务。 平台在平台用户和平台能力提供者之间架起桥梁，由平台团队构建和维护。

平台能力提供者（Platform capability providers)：开发和维护平台提供的能力。提供者可以是外部组织或内部团队，“能力”可以是基础设施、运行时或其他支持服务。

平台工程师：负责根据平台产品经理提供的需求和指示，开发和维护接口和工具，以赋能用户去配置和集成平台的能力。 平台开发人员通常在平台团队组织架构中。

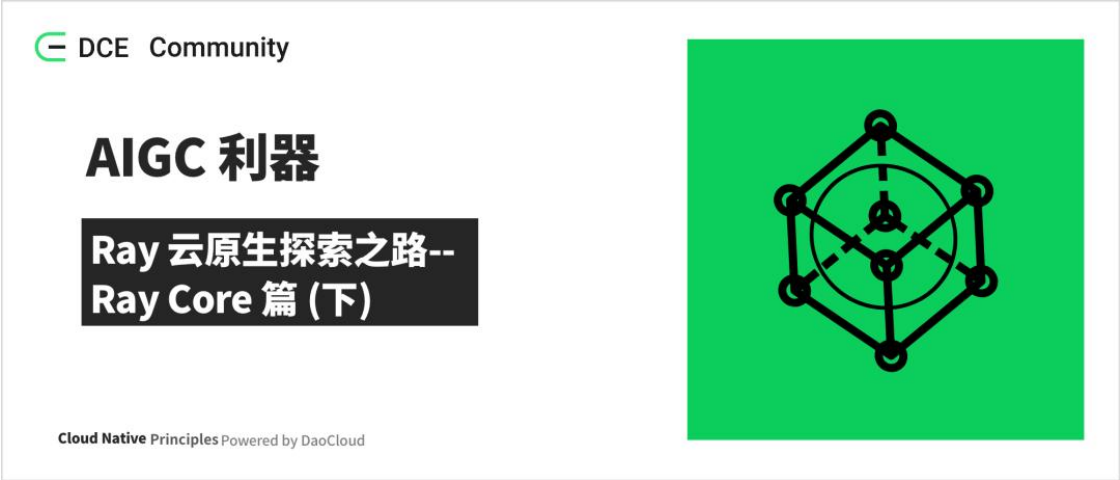
平台产品经理：负责了解平台用户的体验，构建路线图，以解决平台产品的空白、需求和发展机会，并在日常工作中管理平台团队。

平台团队：负责开发和维护平台功能的接口和体验——如 Web 门户、自定义 API 和“黄金路径模板”。 平台团队由平台产品经理管理，并涉及开发人员。随着平台的发展和变得更加先进，其他角色可以成为平台团队的一部分， 包括但不限于运维人员、QA、UI/UX 设计师、文档工程师、开发者布道师。

平台用户：包括但不限于应用程序开发人员和运营商、数据科学家、商业软件的运维人员、信息处理人员—— 那些在平台上运行软件或使用平台提供的功能的人。

最薄可行平台（Thinnest viable platform, TVP）：TVP 是 by Matthew Skelton 和 Manuel Pais. 在《Team Topologies》一书中最初定义的一个概念：“TVP 是保持一种谨慎的平衡：保持平台的【小】，和平台加速和简化软件交付效果之间的平衡。”

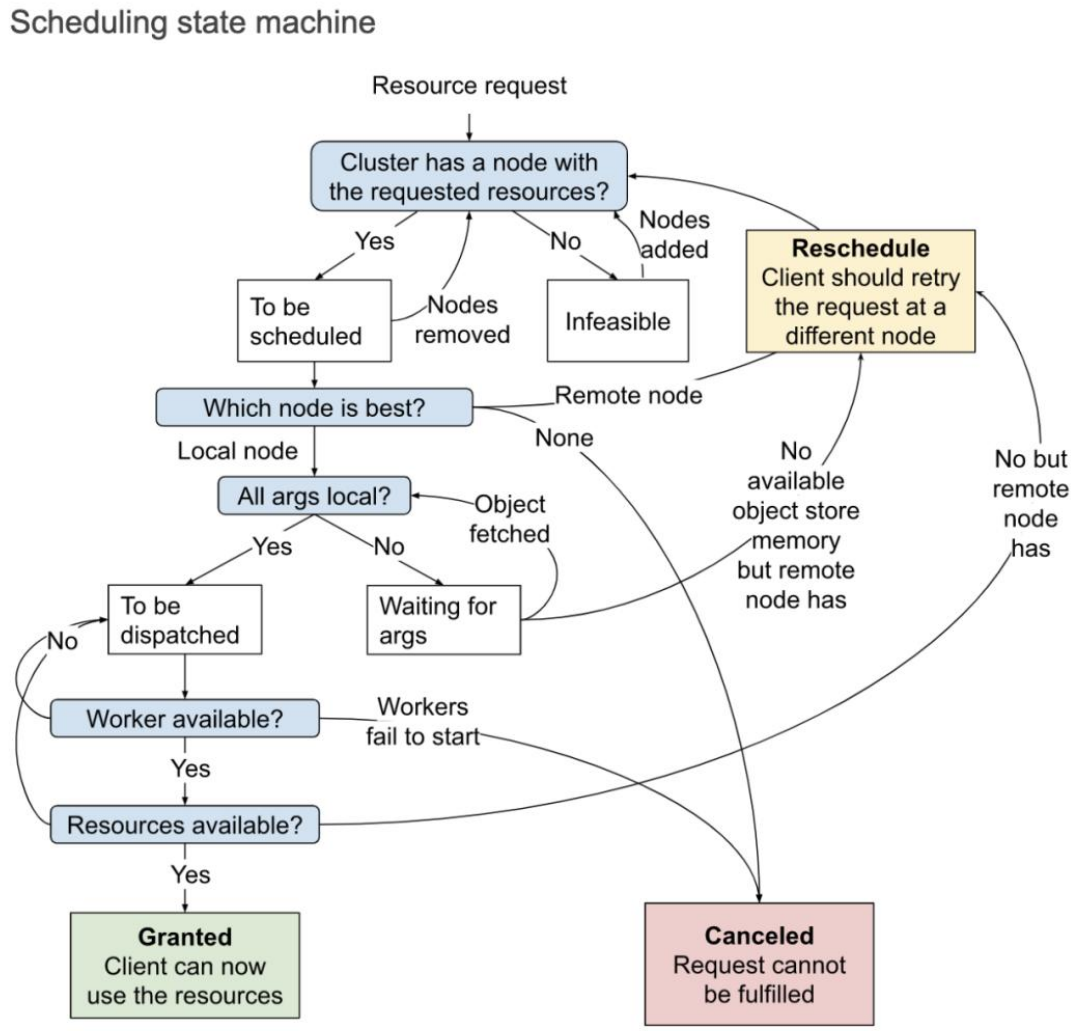
AIGC 利器 Ray 云原生探索之路–Ray Core 篇（下）



cover

在上一篇《AIGC 利器 Ray 云原生探索之路–Ray Core 篇（上)》中，已经基本介绍了 Ray Core 的一些架构、关键概念等，在这里继续补充剩余的一些部分，并结合一些实践的操作，来帮助理解。

调度能力



State machine of raylet distributed scheduling

scheduling

当资源请求（i.e RequestWorkerLease PRC）被 raylet 接受之后，整个过程就如上图的状态机一样，主要包含三个状态:

- Granted: 在这个状态下，client 是可以使用申请的资源和 raylet 返回来的对应的 worker 去运行 actor 和 task 的。
- Reschedule: 如果有比本地节点更合适的节点的时候，就需要重新调度到别的节点，这个过程就是重新调度的过程。 在这个过程中，本地节点是可以从集群视角去看到集群全部节点的资源使用情况。

- Canceled: 当申请的资源无法被满足的时候，就会取消本地的调度。比如一个 client 请求了一个特定的机器，但是这个机器挂了。 再比如，给 task 请求分配的运行环境不能正常的创建出来，就会导致 raylet 不能启动 worker 去运行 task，这个时候调度的请求也会被取消。

调度策略主要包含：

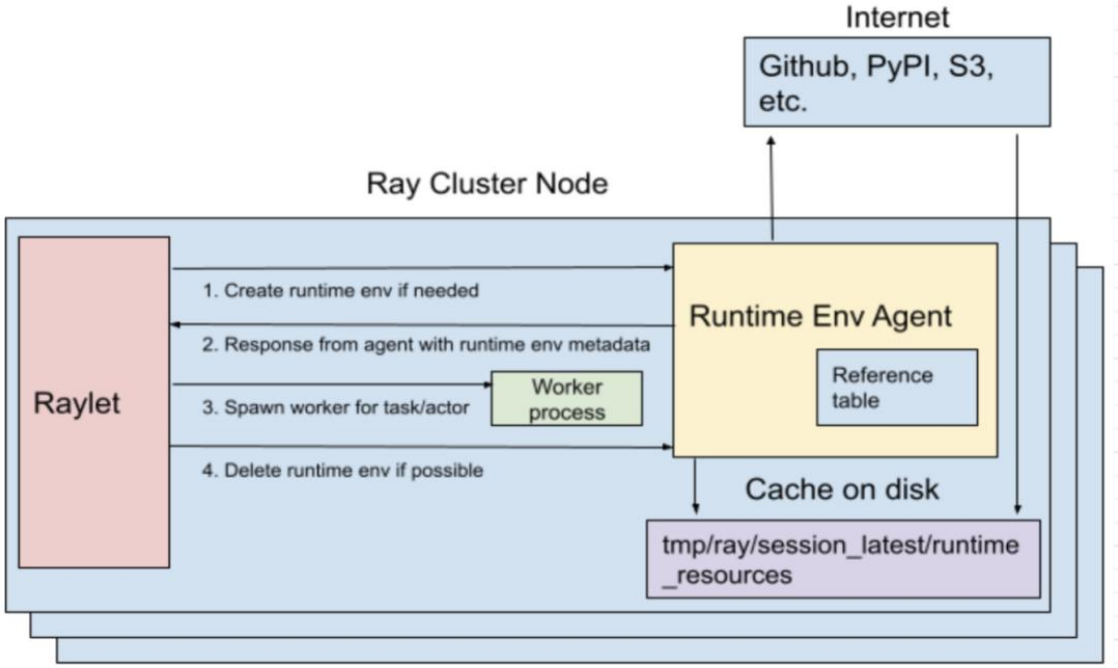
- Hybrid Policy：如果不指定特定的调度策略，这个就是一种默认的调度策略。该策略首先是尝试将 task 放在自己本地来运行， 但是当本地节点的资源使用量超过了配置的阈值（默认为 50%），就将 task 分配到其它的机器上，这个分配机器的过程是按照机器的 ID 进行排序 （保证每次都按照相同顺序来），这里有第一远程节点、第二远程节点等等，依此类推，直到所有节点上的关键资源利用率都超过阈值。 它将选择资源利用率最低的节点。该策略是在 bin-packing 和 load-balancing 之间进行平衡。bin-packing 的场景就是先紧着第一台机器，等第一台机器实在不行了，再到第二台，这种会造成大部分有 task 的机器的负载都很高， 会导致一些不确定的问题，机器的负载高了，什么问题都可能发生。load-balancing 就是下面介绍的 spread 的方式，简单理解就是， 有 task 大家一起均摊来运行，这种会有资源碎片化的问题。

- Spread Policy：该策略使用轮询的方式，在具有可用资源的节点之间分配 task，这样可以保证所有的 task 在整个集群中比较均衡地分布， 不至于某一些机器的负载很高，其它的机器又很空闲。但这种调度的问题是，可能会导致资源碎片化。

- Node Affinity Policy：使用此策略，用户可以明确指定 task 或 actor 应运行的目标节点。如果目标节点处于活动状态， 则任务或参与者只会在那里运行。如果目标节点已挂了，则取决于亲和力是否是软限制，task 或 actor 可能会被安排到其他节点或无法被安排。

- Data Locality Policy：简单理解就是，task 可能依赖很多参数对象，task 依赖的参数对象分布在哪个节点多，就调度到哪个节点去， 这样就不需要将依赖的参数对象搬来搬去了。就是所谓的，数据本地化策略。
- Placement Group Policy：task 和 actor 将在 placement group 所在的位置运行。

运行时环境



An overview of the lifecycle of a runtime environment.

cri

运行时环境（Runtime Env），定义了 python 脚本运行所需要的依赖，包括文件包，还有环境变量等，它是在运行时动态的被安装的， 可以提供给一个 Ray 的 job，或者 actors，或者 tasks 去运行。

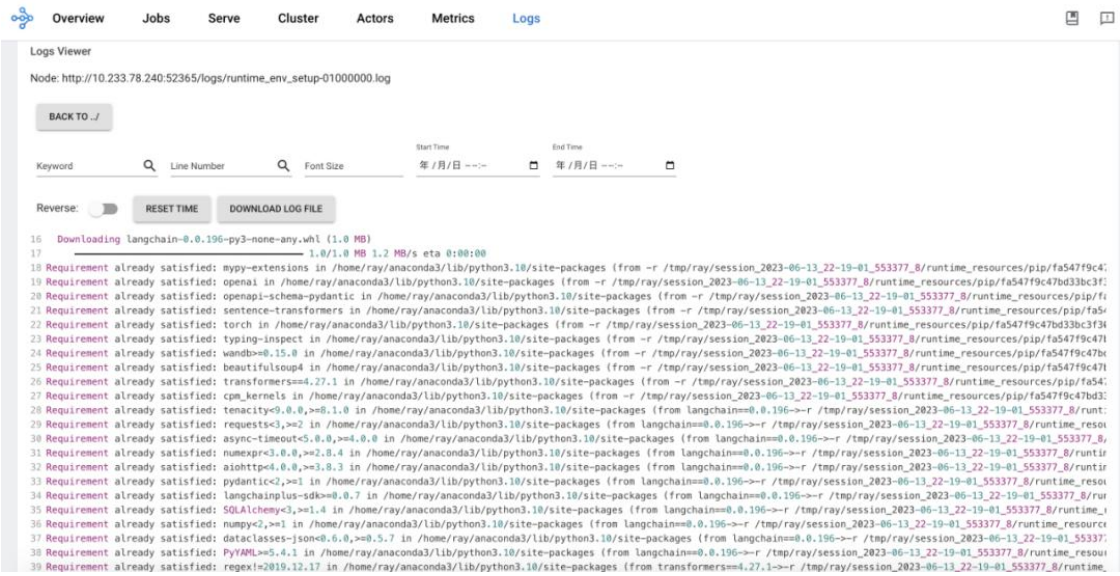
运行时环境的安装和删除是通过运行在每一个 node 上的 RuntimeEnvAgent gRPC server 服务来完成的。 在调度 tasks 和 actors 的时候，RuntimeEnvAgent 组件也是一个关键的组件。

当一个 actor 或者 task 需要运行时环境的时候，raylet 会发送一个 gRPC 请求到 RuntimeEnvAgent 去创建运行时环境，如果不存在就创建。 当创建一个环境的时候会涉及到以下的一些工作：通过 pip install 去下载和安装包；为 worker process 设置环境变量； 在启动一个 worker process 之前，先调用 conda activate；从远程的云存储中下载文件等。

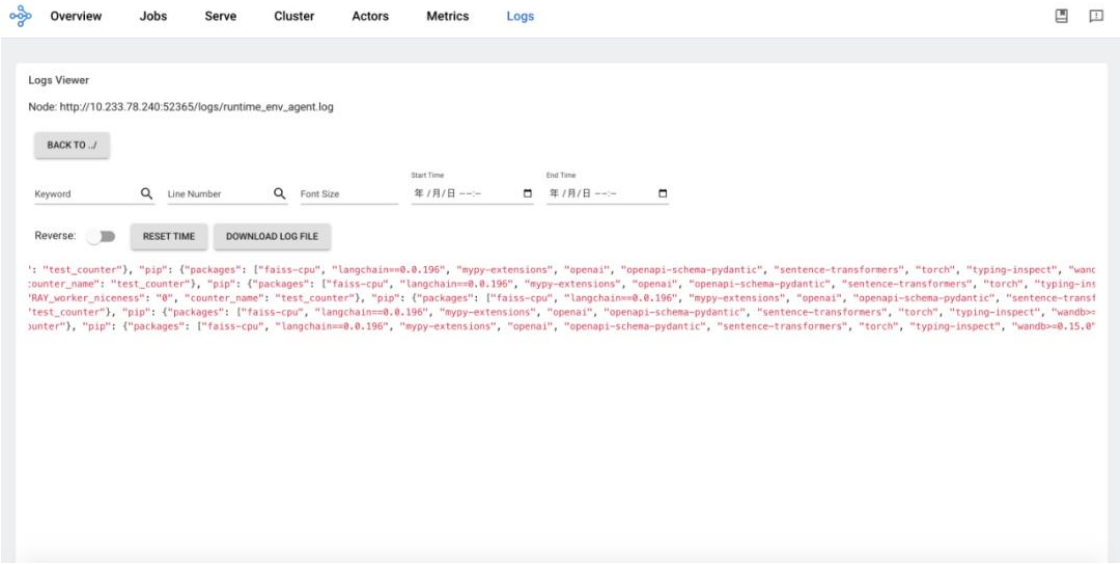
运行时环境中的一些资源，如下载的文件、被安装的 conda 环境，这些会被缓存在每一个 node 上， 这样在不同的 jobs、actors、tasks 之间可以共享这些资源。只有当缓存的大小超过一定的大小， 那些当前没有被任何 jobs、actors、tasks 使用的资源就可以从缓存中删除。

可以在程序中指定，也可以在提交 job 的时候，设置 Runtime Env。同时也支持为 actor 或者 task 指定特定的 runtime env。

```
# Option 1: Starting a single-node local Ray cluster or connecting to existing local cluster
ray.init(runtime_env=runtime_env)

# Option 2: Using Ray Jobs API (Python SDK)
from ray.job_submission import JobSubmissionClient

client = JobSubmissionClient("http://<head-node-ip>:8265")
job_id = client.submit_job(
    entrypoint="python my_ray_script.py",
    runtime_env=runtime_env,
)

# Option 3: Using Ray Jobs API (CLI). (Note: can use --runtime-env to pass a YAML file instead of an inline JSON string.)
$ ray job submit --address="http://<head-node-ip>:8265" --runtime-env-json='{"working_dir": "/data/my_files", "pip": ["emoji"]}' -- python my_ray_script.py
```



ui

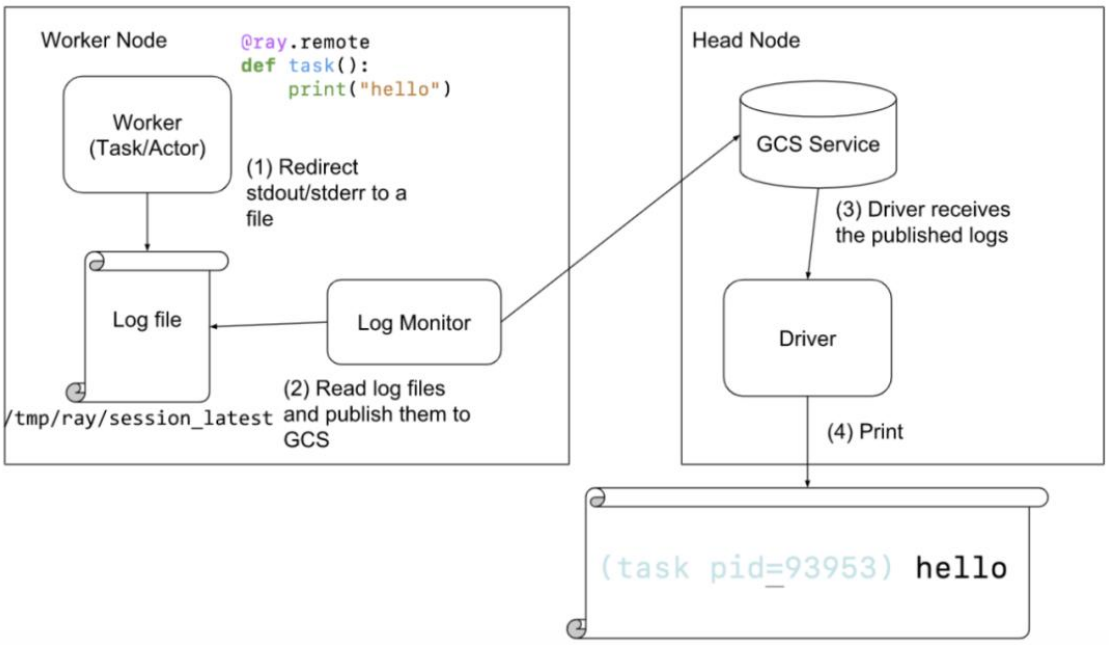


ui

日志能力

当一个 actor 或者 task 打印日志到 stdout or stderr，这些 stdout or stderr 会被自动的重定向到对应的 worker 的日志文件中， 因为 actor 和 task 的运行都是在一个 worker process 中的。log monitor 组件会运行在每一个 node 上，同时它会周期性地读取 Ray 的这些本地日志文件，然后通过 GCS 的

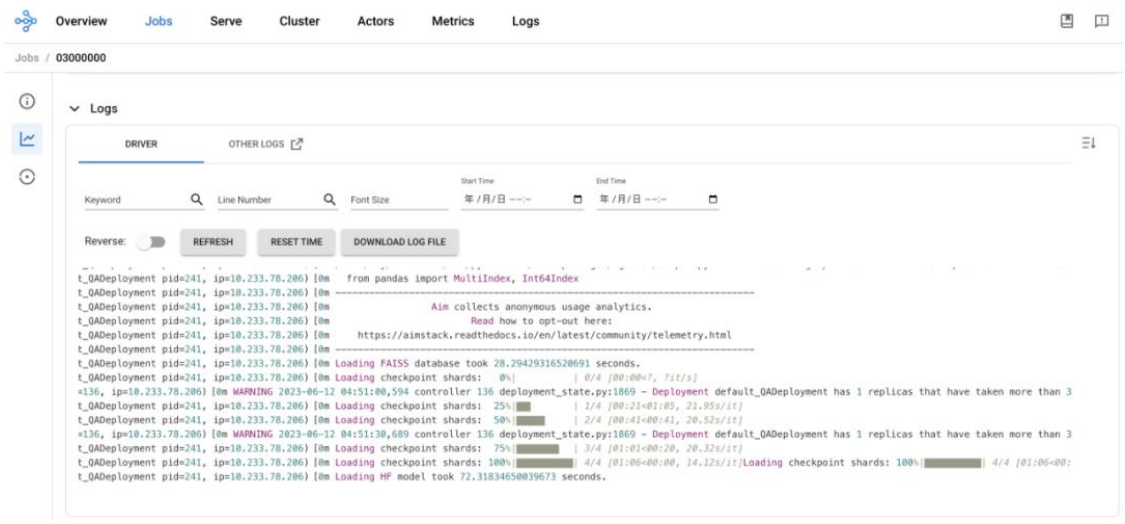
pubsub 机制，发布这些日志消息给到 driver 程序。driver 程序会负责聚合和打印所有来自于 actors 和 tasks 的日志。



log arch

```
(base) [root@10-1-3-210 raytest]# RAY_ADDRESS='http://10.1.3.210:8265' ray job submit --working-dir . -- python test0.py
Job submission server address: http://10.1.3.210:8265
2023-06-08 15:21:18,194 INFO dashboard_sdk.py:364 -- Package gcs://_ray_pkg_d8e10a0b6435346d.zip already exists, skipping upload.

-------------------------------------------------------
Job 'raysubmit_E9VefP31Q4YEJKng' submitted successfully
-------------------------------------------------------

Next steps
  Query the logs of the job:
    ray job logs raysubmit_E9VefP31Q4YEJKng
  Query the status of the job:
    ray job status raysubmit_E9VefP31Q4YEJKng
  Request the job to be stopped:
    ray job stop raysubmit_E9VefP31Q4YEJKng

Tailing logs until the job exits (disable with --no-wait):
2023-06-08 15:21:20,373 INFO worker.py:1314 -- Using address 10.1.3.210:6379 set in the environment variable RAY_ADDRESS
2023-06-08 15:21:20,373 INFO worker.py:1432 -- Connecting to existing Ray cluster at address: 10.1.3.210:6379...
2023-06-08 15:21:20,392 INFO worker.py:1616 -- Connected to Ray cluster. View the dashboard at 10.1.3.210:8265
[0, 1, 4, 9]

------------------------------------------
Job 'raysubmit_E9VefP31Q4YEJKng' succeeded
------------------------------------------
```

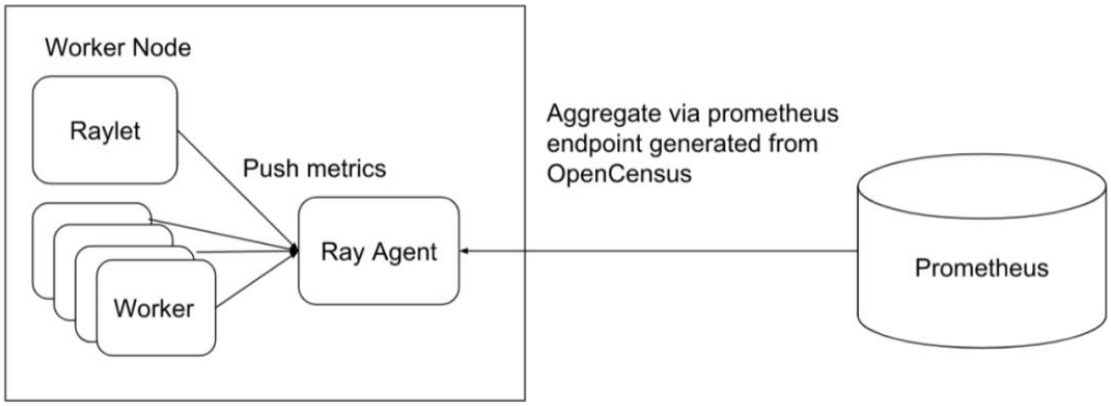
log



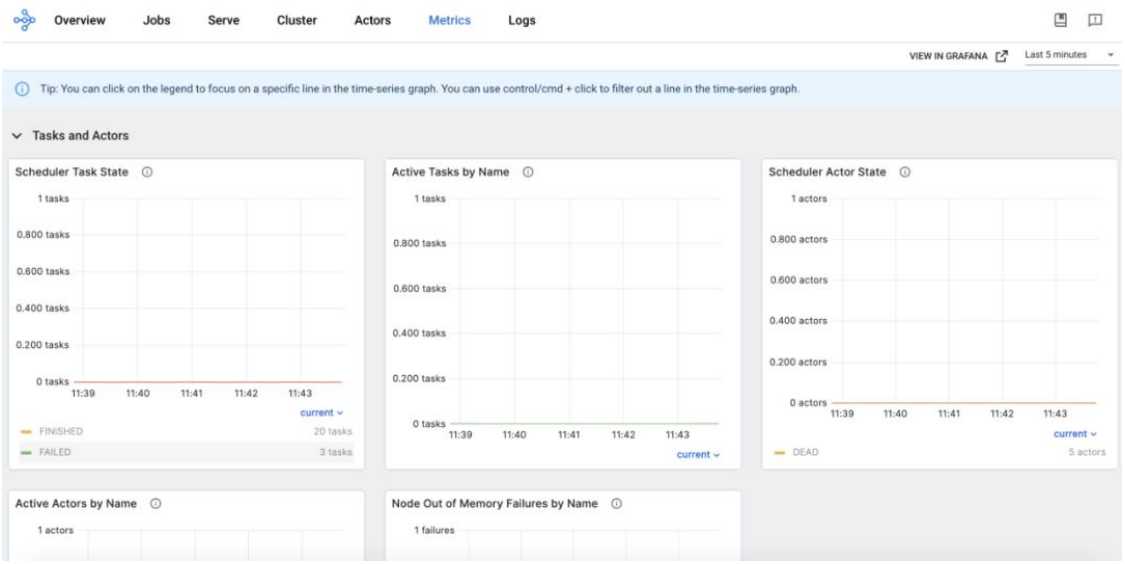
log

Metrics 能力

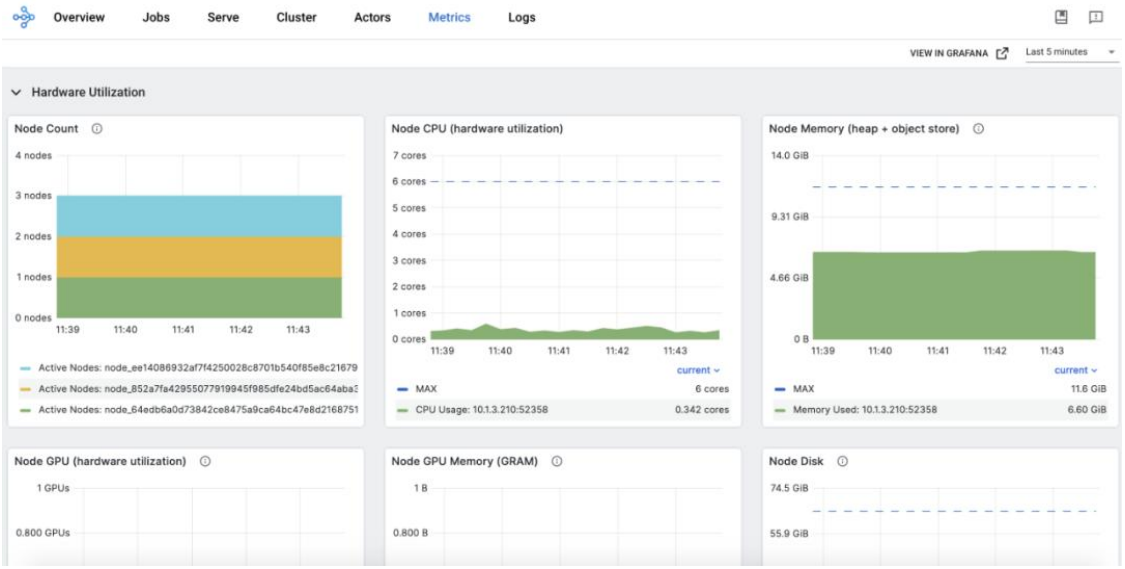
Ray 原生地集成了 OpenCensus，同时默认支持将 metrics 数据导出到 Prometheus。所有的 Ray 的组件， 包括 GCS、raylet、workers 将自己的 metrics 数据 push 到它们本地的 Ray agent 进程中， 然后每个节点的 Ray agent 通过 OpenCensus 去暴露 metrics 数据，作为 Prometheus 的一个 exporter， 这样数据就可以被 Prometheus pull 到 Prometheus 里。



metrics



metrics



metrics

GPU 能力

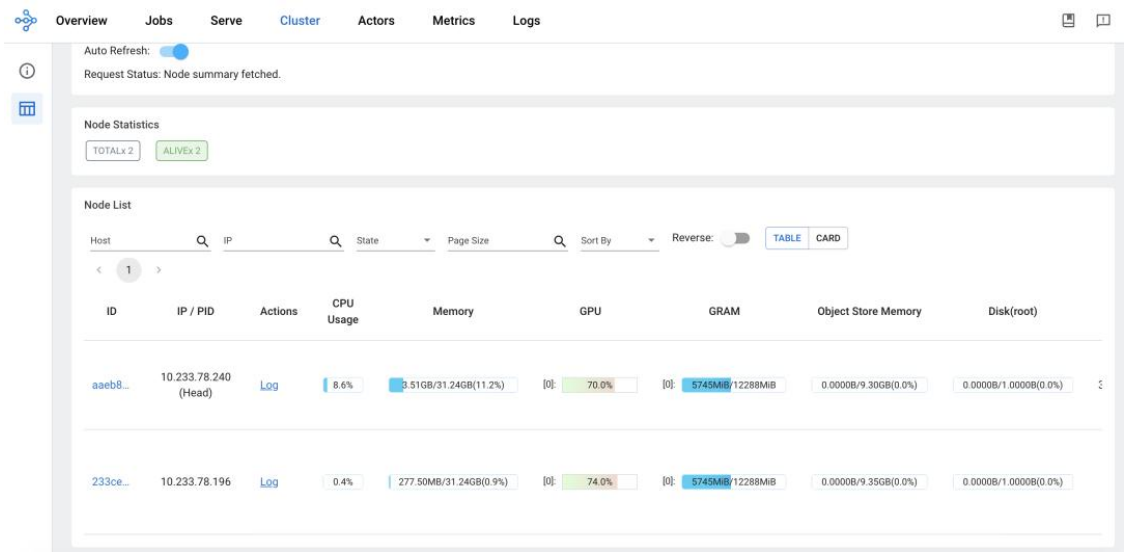
默认情况下，Ray 将通过自动检测主机硬件的方式得知 GPU 的设备数量，同时设置给自己，当然也可以通过设置来覆盖自动检测出来的结果。 如果一个 actor 或者 task 的执行需要 GPU 资源的时候，可以通过给 actor 或者 task 指定需要的资源请求，如@ray.remote(num_gpus=1)。 这个时候，Ray 会调度 actor 或者 task 到满足资源需求的机器，也就是有空闲 GPU 的机器去运行 actor 或

者 task， 同时在 actor 或者 task 在运行之前，通过设置 CUDA_VISIBLE_DEVICES 环境变量，将 GPU 指定给 actor 或者 task。下面举例说明了这个过程：

```
ray.init(num_gpus=2)

@ray.remote(num_gpus=1)
class GPUActor:
    def ping(self):
        print("ray.get_gpu_ids(): {}".format(ray.get_gpu_ids()))
        print("CUDA_VISIBLE_DEVICES: {}".format(os.environ["CUDA_VISIBLE_DEVICES"]))


@ray.remote(num_gpus=1)
def use_gpu():
    print("ray.get_gpu_ids(): {}".format(ray.get_gpu_ids()))
    print("CUDA_VISIBLE_DEVICES: {}".format(os.environ["CUDA_VISIBLE_DEVICES"]))


gpu_actor = GPUActor.remote()
ray.get(gpu_actor.ping.remote())

// 因为 actor（gpu_actor）使用了第一个 GPU，所以 task（use_gpu）将会使用第二个 GPU。
ray.get(use_gpu.remote())

# (GPUActor pid=52420) ray.get_gpu_ids(): [0]
# (GPUActor pid=52420) CUDA_VISIBLE_DEVICES: 0
# (use_gpu pid=51830) ray.get_gpu_ids(): [1]
# (use_gpu pid=51830) CUDA_VISIBLE_DEVICES: 1
```

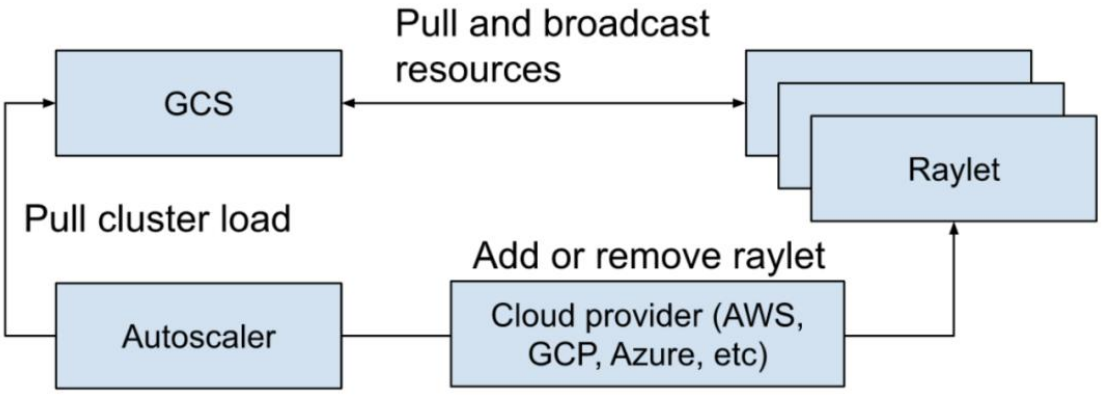
在云原生的场景下，需要使容器可以识别 GPU 的设备，同时将设备信息传递给到 Ray 内部。Ray 的官方的 docker 镜像版本中， 提供了 GPU 版本的镜像了。在使用的时候，需要在 RayJob 的定义中指定类似“ nvidia.com/gpu: 1 ”这样的资源申请方式， 同时在 actor 或者 task 上指定需要的 GPU 请求，如：

```
@ray.remote(num_gpus=1)
class GPUActor:
    def say_hello(self):
        print("I live in a pod with GPU access.")
```



gpu

Autoscaler 能力



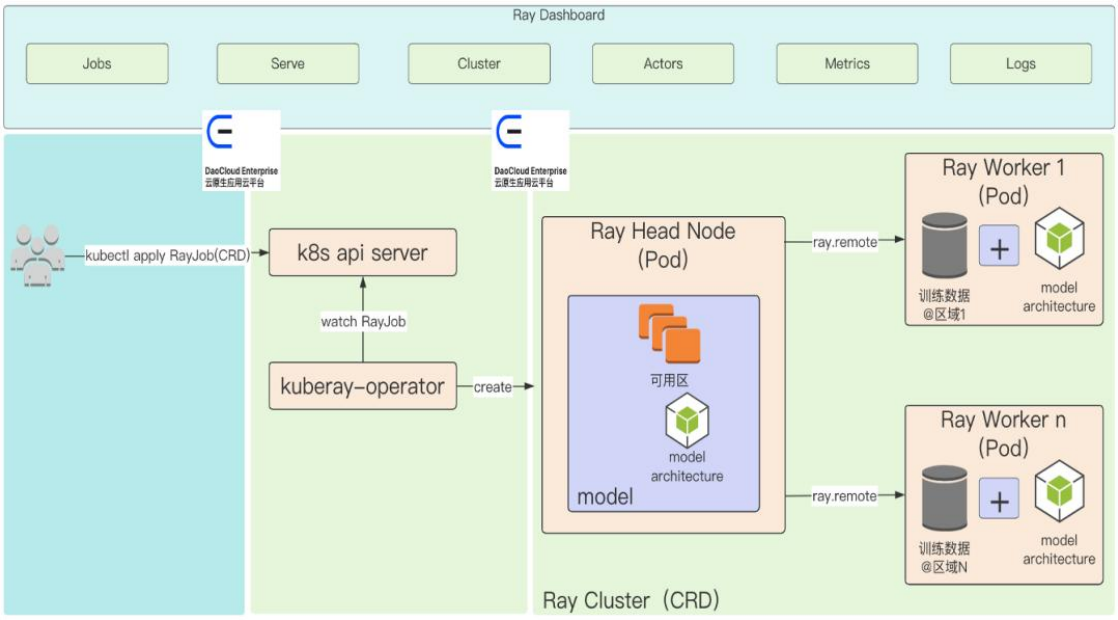
Autoscaler

Ray Autoscaler 负责从集群中添加和删除节点。它查看分布式调度器公开的逻辑资源需求、集群中当前的节点、集群的节点配置文件， 计算所需的集群配置，并执行将集群移动到所需状态的操作。Autoscaler 会从 GCS 中去拉取当前集群的负载情况，然后调用云厂商的实现去添加和移除节点。

自动扩缩容的主要工作包含以下几点：

1. 当应用提交 actors、tasks、placement groups 去请求资源，如 CPU、GPU 等。
2. 调度器会查看资源的要求和可用的资源，然后决定 tasks 应该放置的位置，或者资源不满足的时候就会阻塞，这个信息会被作为一个快照放到 GCS 中。
3. Autoscaler 作为一个独立的进程运行，它周期性地从 GCS 中抓取步骤 2 中提到的快照， 然后从中查看集群的可用资源、请求的资源、有什么 tasks 是阻塞的，以及 worker 节点被设置的配置， 然后运行 bin-packing 算法（在调度部分提到过）去计算一下，在能保证运行中的 tasks/actors/placement group 和阻塞的这些 tasks/actors/placement group 的资源都能满足的情况下，一共需要多少节点。
4. 然后 Autoscaler 使用 node provider interface 去添加和移除节点，各种 cloud provider 可以去实现自己的 node provider interface，如 AWS、GCP、Azure、Kubernetes 以及 on-premise 的数据中心。
5. 当新的节点启动了，也注册到 Ray 的集群中了，这些节点就可以接收应用的工作负载了。

实践



practice

安装 KubeRay：

```
helm repo add kuberay https://ray-project.github.io/kuberay-helm/

# Install both CRDs and KubeRay operator v0.5.0.
helm install kuberay-operator kuberay/kuberay-operator --version 0.5.0

# Check the KubeRay operator Pod in `default` namespace
kubectl get pods
# NAME                                READY   STATUS    RESTARTS   AG
E
# kuberay-operator-7fc785cbbd-9jcpv   1/1     Running   0          14d
```

创建 RayJob：

```
apiVersion: ray.io/v1alpha1
kind: RayJob
metadata:
  name: rayjob-sample
```

```yaml
spec:
  entrypoint: python /home/ray/samples/sample_code.py
  # runtimeEnv decoded to '{
  #     "pip": [
  #         "requests==2.26.0",
  #         "pendulum==2.1.2"
  #     ],
  #     "env_vars": {
  #         "counter_name": "test_counter"
  #     }
  #}'
  runtimeEnv: ewogICAgInBpcCI6IFsKICAgICAgICAicmVxdWVzdHM9PTIuMjYuMCIsCiAgICAgICAgInBlbmR1bHVtPT0yLjEuMiIKICAgIF0sCiAgICAiZW52X3ZhcnMiOiB7Im
NvdW50ZXJfbmFtZSI6ICJ0ZXN0X2NvdW50ZXIifQp9Cg==
  rayClusterSpec:
    rayVersion: '2.4.0' # should match the Ray version in the image of the cont
ainers
    # Ray head pod template
    headGroupSpec:
      # the following params are used to complete the ray start: ray start --he
ad --block --redis-port=6379 ...
      rayStartParams:
        dashboard-host: '0.0.0.0'
        num-cpus: '1' # can be auto-completed from the limits
      #pod template
      template:
        spec:
          containers:
            - name: ray-head
              image: rayproject/ray:2.4.0
              ports:
                - containerPort: 6379
```

```
            name: gcs-server
          - containerPort: 8265 # Ray dashboard
            name: dashboard
          - containerPort: 10001
            name: client
          - containerPort: 8000
            name: serve
        resources:
          limits:
            cpu: "1"
          requests:
            cpu: "200m"
        volumeMounts:
          - mountPath: /home/ray/samples
            name: code-sample
    volumes:
      # You set volumes at the Pod level, then mount them into containe
rs inside that Pod
      - name: code-sample
        configMap:
          # Provide the name of the ConfigMap you want to mount.
          name: ray-job-code-sample
          # An array of keys from the ConfigMap to create as files
          items:
            - key: sample_code.py
              path: sample_code.py
  workerGroupSpecs:
    # the pod replicas in this group typed worker
    - replicas: 1
      minReplicas: 1
      maxReplicas: 5
      # logical group name, for this called small-group, also can be function
```

```
al
          groupName: small-group
          rayStartParams: {}
          #pod template
          template:
            spec:
              containers:
                - name: ray-worker # must consist of lower case alphanumeric c
haracters or '-', and must start and end with an alphanumeric character (e.g. '
my-name',  or '123-abc'
                  image: rayproject/ray:2.4.0
                  lifecycle:
                    preStop:
                      exec:
                        command: [ "/bin/sh","-c","ray stop" ]
                  resources:
                    limits:
                      cpu: "1"
                    requests:
                      cpu: "200m"
########################Ray code sample#################################
####
# this sample is from https://docs.ray.io/en/latest/cluster/job-submission.html#q
uick-start-example
# it is mounted into the container and executed to show the Ray job at work
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: ray-job-code-sample
data:
  sample_code.py: |
```

```
    import ray
    import os
    import requests

    ray.init()

    @ray.remote
    class Counter:
        def __init__(self):
            # Used to verify runtimeEnv
            self.name = os.getenv("counter_name")
            self.counter = 0

        def inc(self):
            self.counter += 1

        def get_counter(self):
            return "{} got {}".format(self.name, self.counter)

    counter = Counter.remote()

    for _ in range(5):
        ray.get(counter.inc.remote())
        print(ray.get(counter.get_counter.remote()))

    print(requests.__version__)
```

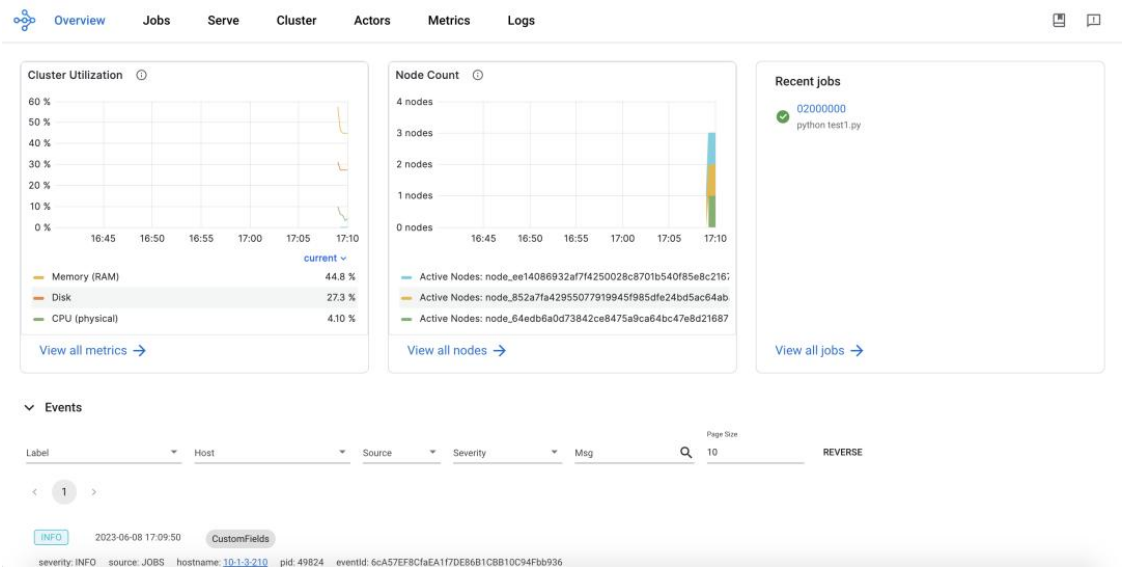
查看 RayJob：

```
$ kubectl get rayjob
NAME                AGE
rayjob-sample     14d
```

```
$ kubectl get raycluster
NAME                                 DESIRED WORKERS   AVAILABLE WORKERS   STATUS   AGE
rayjob-sample-raycluster-25mkr       1                 1                   ready    14d
```

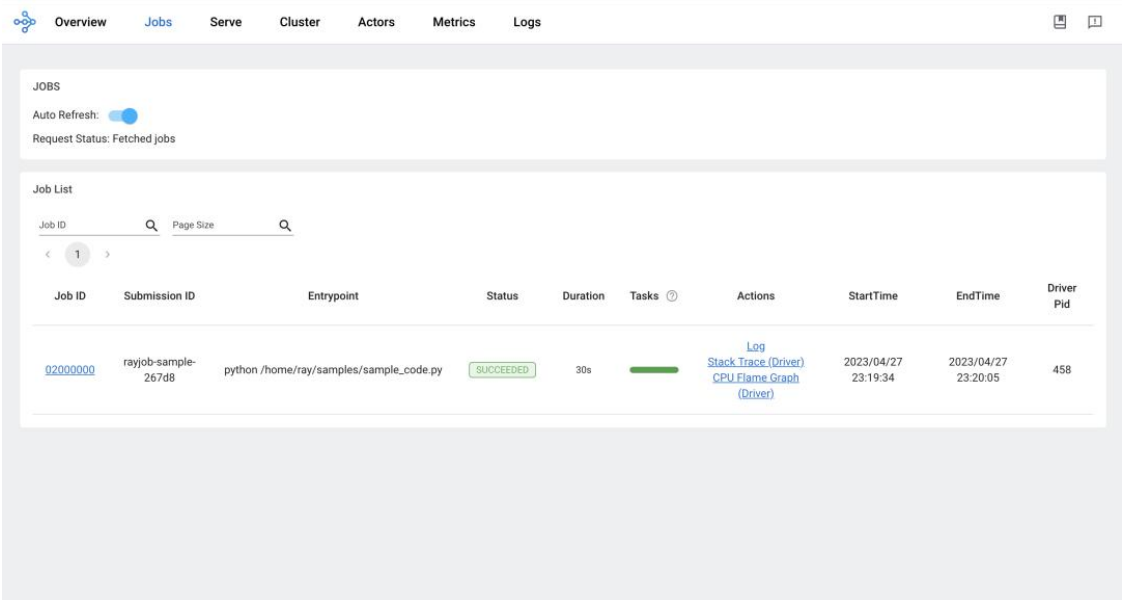
访问 Ray Dashboard：

- 全局视图：



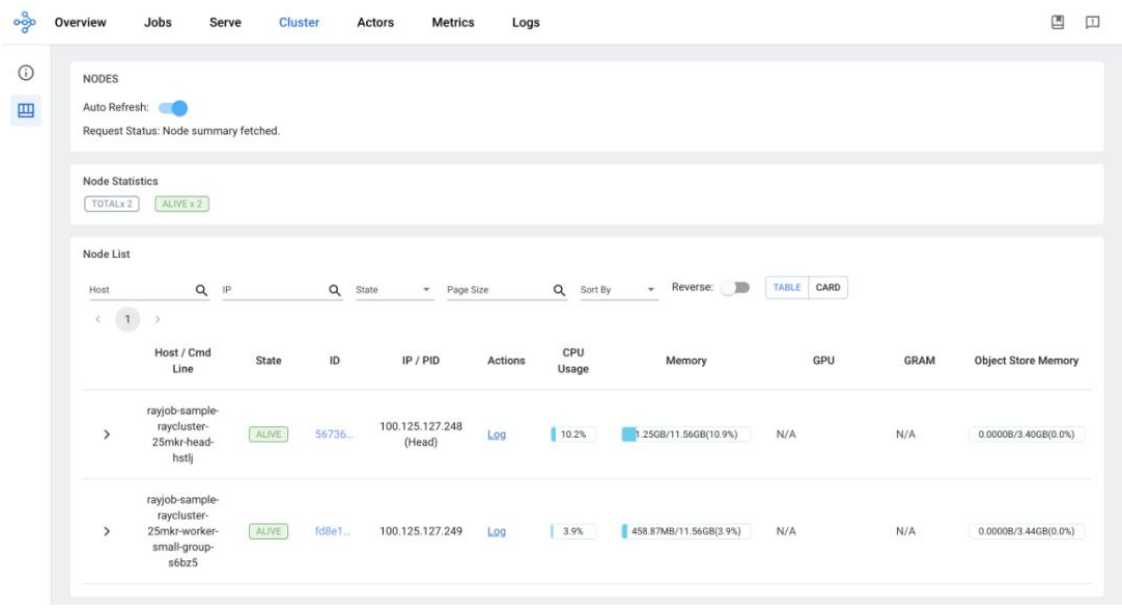
global

- Job 视图：



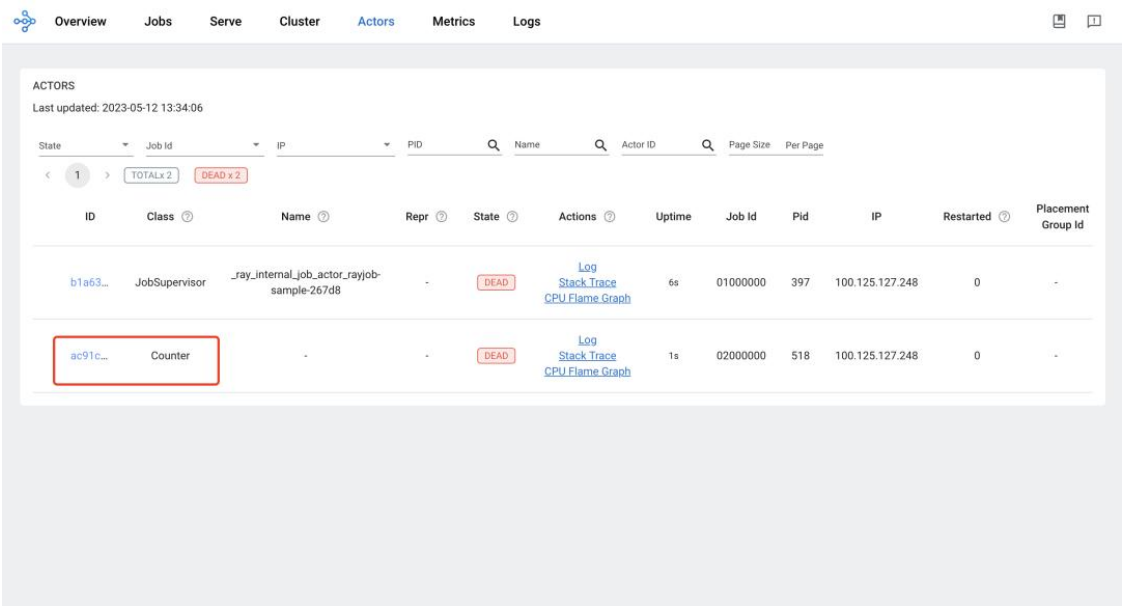
job

- 集群视图：



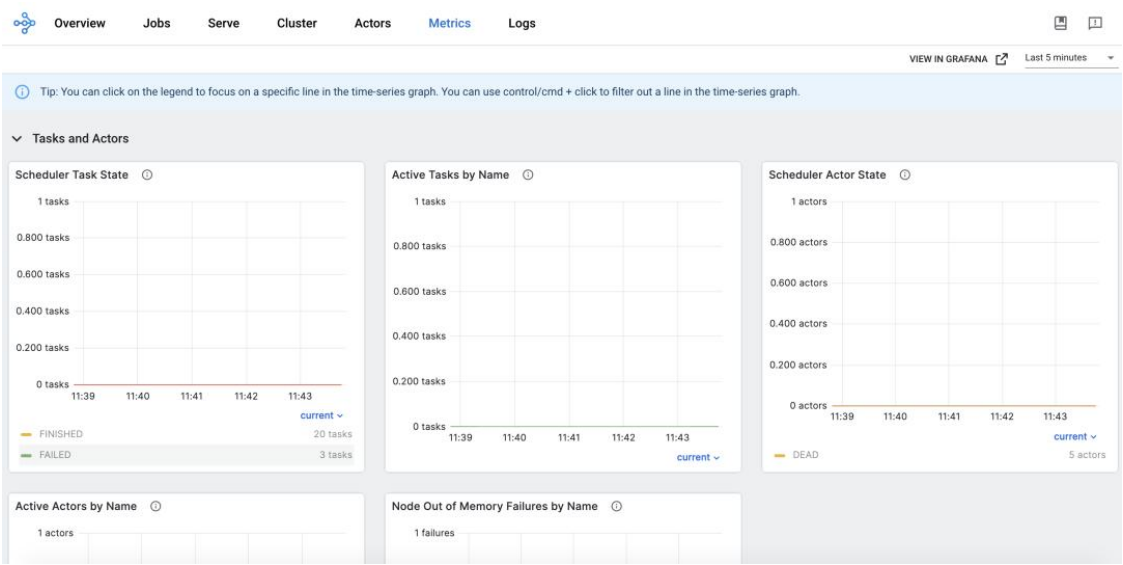
cluster

- Actor 视图：



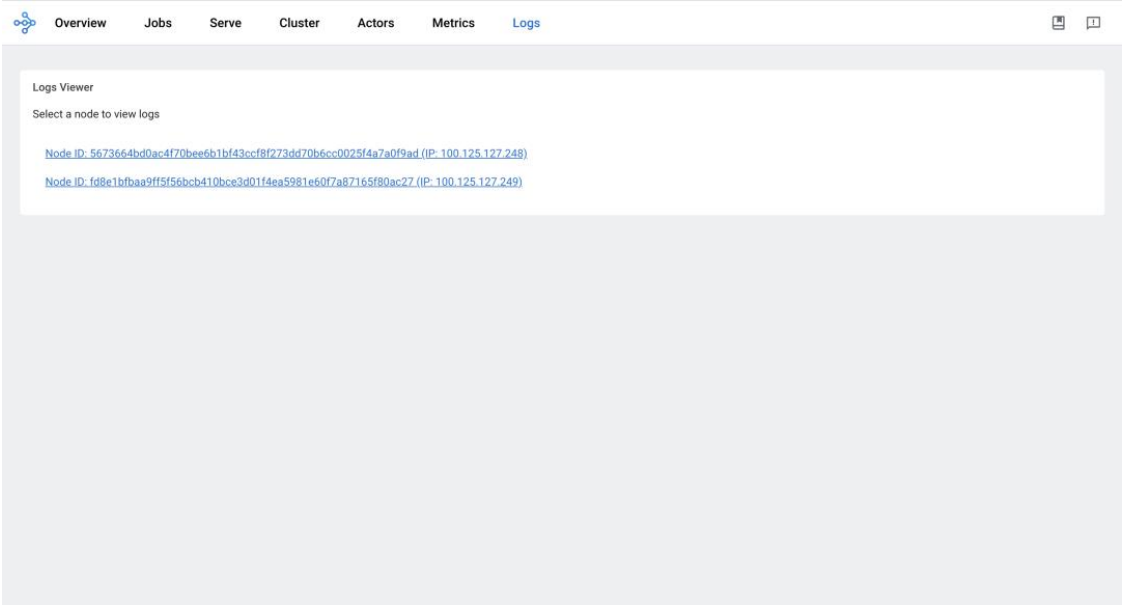
actor

- Metrics 视图：



metrics

- Log 视图：



log

总结

以上从 Ray Core 本身介绍了相关的知识，同时也基于社区 Kuberay 结合 Kubernetes 的方案，通过实践展示了一下 Ray Core 的相关能力。

参考链接：

- Ray Documentation
- Ray v2 Architecture

云原生联邦中间件–FedState 正式开源上线



image

在云原生的场景下，有状态的服务，也有了很大的发展。有的玩法是基于 Yaml 的包装来手动安装， 有的基于 Helm Charts 的方式封装，有的是基于 Operator 的方式的来封装的。目前社区的玩法中， 以 Helm Chart 和 Operator 偏多，特别是基于 Operator 的方式，因为它是面向编程友好的方式， 可以根据需求进行各种能力的开发。现在社区开源的各种中间件，数据库的 Operator 也是层出不穷的， 但是主要是集中在单集群的玩法，这个和 Operator 的框架本身是有关系的。在多云/联邦的环境下， 怎么设计和实现基于多集群、多数据中心以及混合云场景下，有状态服务的编排、调度、部署和自动化运维等能力？ 最新开源的 FedState 项目正是在尝试解决这样的场景问题。

FedState 的开源仓库地址在：https://github.com/fedstate/fedstate 。 接下来，整体介绍一下 FedState，同时结合 MongoDB 为例来分析。

需求分析

首先先来大概分析一下，客户对中间件在多云场景下的使用需求大概有哪些。

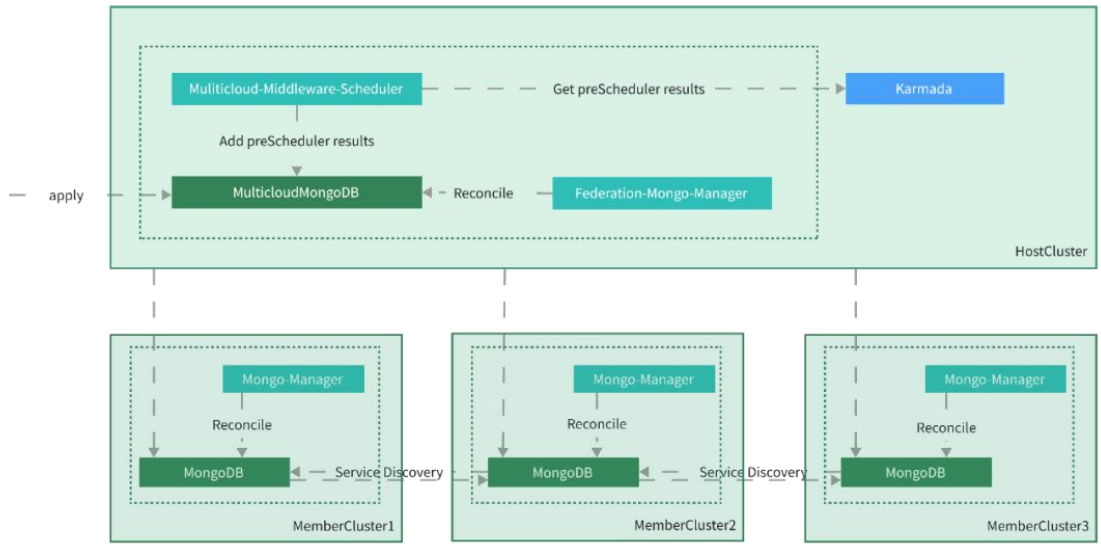
- 跨容器集群/数据中心部署中间件

- 统一控制面和数据面的多层架构规范
- 成员容器集群的资源统计能力
- 根据成员容器集群资源合理调度中间件负载
- 根据不同中间件的特性灵活适配去完成调度
- 跨容器集群中间件组件之间的发现
- 集群的统一外部访问入口
- 中间件集群的跨多集群的拓扑
- 中间件集群的扩容
- 中间件集群的缩容
- 跨容器集群的数据复制/容灾能力
- 中间件集群的自动化运维能力
- 云原生方式提供服务
- 页面化管理中间件集群的能力
- 开放的 API 能力，方便上层能力的建设
- 存储的开放性，适配 CSI
- 租户隔离
- 多版本中间件支持
- 监控，告警等能力

架构介绍

目前 FedState 除了实现了联邦中间件的框架之外，还内置实现了联邦 MongoDB 和单集群 MongoDB 的中间件能力。 控制平面和数据平面是相对独立的。可以直接用单集群的 MongoDB Manager 在单集群的场景下使用， 也可以使用联邦场景下的 MongoDB 能力。同时，联邦的 MongoDB 是使用了单集群模式的 MongoDB Manager。

这里，以联邦 MongoDB 为例。以下架构图介绍了 FedState 在多云/多集群环境中，围绕 MongoDB 副本集，提供的整体能力。



图片

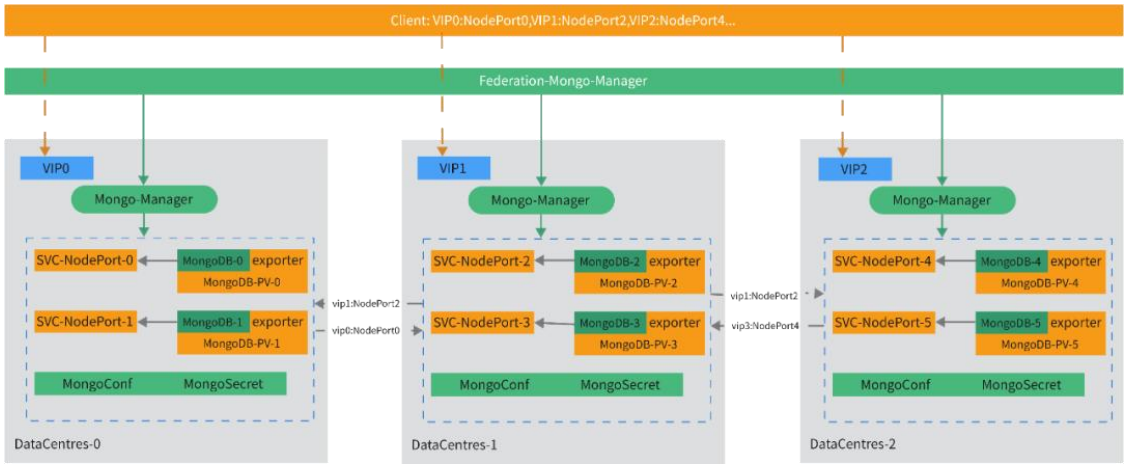
FedState 自身包含以下组件：

Multicloud-Middleware-Scheduler：负责收集工作集群的资源使用情况，根据资源使用情况给出合理的调度方式。 同时不同的中间件有自己的特性，会有对应中间件的特殊调度逻辑，帮助各种中间件更好的选择适合自己的调度结果。 可以根据需要扩展对不同类型的中间件的调度策略和实现。

Federation-Mongo-Manager ： 可以理解成 MongoDB 控制面的 Operator，主要负责这些中间件服务按需配置、 进行合理调度以及通过 Karmada 进行分发。可以根据需要扩展不同的中间件进来。

Mongo-Manager：可以理解成 MongoDB 数据面的 Operator，主要负责真正的服务负载的处理。 可以根据需要扩展不同的中间件进来。这里也可以选择优秀的开源的 Operator， 只要这个 Operator 本身满足控制平面对于数据平面 Operator 的要求就可以， 只要在控制平面进行扩展，就可以支持这种中间件，让其具备联邦中间件的能力。

关键能力



图片

1. 联邦 MongoDB 实例的对外访问方式：Vip0:NodePort0，Vip1:NodePort1…
2. 各个成员集群间 MongoDB 服务的东西向流量以及服务发现，使用来对应集群 VIP+服务所对应的 NodePort 方式
3. 除了支持联邦 MongoDB 副本集的创建与更新等能力以外，还可以直接用数据面的 Operator 来在单集群的模式线，创建和管理 MongoDB 的集群
4. 根据不同的中间件的特性，分布不同的角色的节点部署在不同的集群中，保证中间件的高可用的要求
5. 支持手动方式，调整集群规模，来完成扩容和缩容的能力
6. 支持故障迁移，基于 Karmada 和自定义的迁移策略
7. 支持 Prometheus 方式的监控，收集 MongoDB 各个节点的 metrics 数据
8. 自定义配置文件，可以根据需要，设置 MongoDB 支持的各种配置项
9. 支持 MongoDB 集群认证，完成客户端连接的认证
10. 支持 CSI 存储，持久化中间件数据
11. 支持 MongoDB 集群的拓扑，清晰地观测各个 MongoDB 节点的运行状态
12. 支持 MongoDB 副本集的方式，组建 MongoDB 集群

业务模型

以 MongoDB 为例，介绍在联邦和单集群下提供的一些常见能力介绍：

联邦模式下的 MongoDB 集群

创建和管理联邦有状态服务的资源对象，会根据配置分发有状态服务到各成员集群， 设置需要的存储，开启监控，密码配置等，创建成功后，会在状态中展示 MongoDB 在各个成员集群上的部署拓扑以及访问地址。 这种模式适合联邦集群下，使用控制平面和数据平面的 MongoDB Operator 安装和管理跨集群的 MongoDB 集群的场景。

??? note “点击查看 MultiCloudMongoDB YAML 示例”

```yaml
apiVersion: middleware.fedstate.io/v1alpha1
kind: MultiCloudMongoDB
metadata:
  name: multicloudmongodb-sample
spec:
  replicaset: 5 # 副本数
  export: # 监控配置
    enable: true
    resource:
      limits:
        cpu: "500m"
        memory: 512Mi
      requests:
        cpu: "200m"
        memory: 256Mi
  resource: # 有状态服务资源配置
    limits:
      cpu: "2"
      memory: 512Mi
```

```
      requests:
        cpu: "1"
        memory: 512Mi
  storage: # 存储设置
    storageClass: managed-nfs-storage
    storageSize: 1Gi
  imageSetting: # 镜像设置
    image: mongo:3.6
    imagePullPolicy: Always
    imagePullSecret: "my-image-secret"
  auth: # 密码设置
    rootPasswd: "mypasswd"
  config: # 有状态服务配置设置
    arbiter: false
    configRef: "my-custome-configmap"
  scheduler: # 调度设置，调度模型等配置
    schedulerMode: Uniform
  spreadConstraints: # 调度配置，节点选择等相关配置
    nodeSelect:
      deploy: mongo
```

单集群模式下的 MongoDB 集群

创建和管理有状态服务的资源对象，这种模式适合单集群下，使用数据平面的 MongoDB Operator 直接安装和管理 MongoDB 集群的场景。

??? note “点击查看 MongoDB YAML 示例”

```yaml
apiVersion: middleware.fedstate.io/v1alpha1
kind: MongoDB
metadata:
  name: mongodb-sample
```

```
spec:
  members: 1 # 副本数
  image: mongo:3.6 # 可以指定某个 mongo 版本进行部署，默认为 mongo 6.0 版本
  imagePullSecret: # 镜像拉取认证信息
    username: admin
    password: admin
  imagePullPolicy: Always # 镜像拉取策略
  config: # 参考 mongo 的配置进行填入
    - name: LOG_LEVEL
      value: info
  customConfigRef: mongo-operator-mongo-default-config # 自定义 mongo config，指定 cm name，默认为 mongo-default-config
  rootPassword: "123456" # 指定初始密码
  resources:
    limits:
      cpu: "1"
      memory: 512Mi
    requests:
      cpu: "1"
      memory: 512Mi
  persistence: # 持久化参数
    storage: 1Gi
    storageClassName: "" # 存储类型，默认为空，使用默认 sc
  metricsExporterSpec:
    enable: true # 监控是否开启，默认为 true
    resources:
      limits:
        cpu: "0.1"
        memory: 128Mi
      requests:
        cpu: "0.1"
        memory: 128Mi
```

```
  podSpec:
    nodeSelector: # 节点选择器
    securityContext: # pod 安全上下文
    topologySpreadConstraints: # 拓扑分布约束
    affinity: # 亲和和反亲和
    tolerations: # 污点容忍
```

案例介绍

虽然支持联邦和单集群两种模式，这里的案例以联邦的方式展示案例。

在控制平面下发联邦 MongoDB 集群的 CR

```yaml
apiVersion: middleware.fedstate.io/v1alpha1
kind: MultiCloudMongoDB
metadata:
  name: multicloudmongodb-sample
spec:
  replicaset: 5
  export:
    enable: false
  resource:
    limits:
      cpu: "2"
      memory: 512Mi
    requests:
      cpu: "1"
      memory: 512Mi
  storage:
    storageClass: managed-nfs-storage
    storageSize: 1Gi
  imageSetting:
    image: mongo:3.6
    imagePullPolicy: Always
```

查看控制平面的 MongoDB CR 实例状态

??? note “点击查看 MultiCloudMongoDB YAML 示例”

```yaml
apiVersion: middleware.fedstate.io/v1alpha1
kind: MultiCloudMongoDB
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"middleware.fedstate.io/v1alpha1","kind":"MultiCloudMongoDB",
"metadata":{"annotations":{},"name":"multicloudmongodb-sample","namespace":"fe
deration-mongo-operator"},"spec":{"export":{"enable":false},"imageSetting":{"image
":"mongo:3.6","imagePullPolicy":"Always"},"replicaset":5,"resource":{"limits":{"cpu":"
2","memory":"512Mi"},"requests":{"cpu":"1","memory":"512Mi"}},"storage":{"storageC
lass":"managed-nfs-storage","storageSize":"1Gi"}}}
    schedulerResult: '{"ClusterWithReplicaset":[{"cluster":"10-29-14-21","replicaset
":4},{"cluster":"10-29-14-25","replicaset":1}]}'
  creationTimestamp: "2023-05-25T07:06:26Z"
  finalizers:
  - multiCloudMongoDB.finalizers.middleware.fedstate.io
  generation: 1
  name: multicloudmongodb-sample
  namespace: federation-mongo-operator
  resourceVersion: "72770747"
  uid: 56c69f88-6c52-4886-a922-9ebf7c156ba5
spec:
  auth:
    rootPasswd: 39nZzksAmXE=
  config: {}
  export:
    resource: {}
  imageSetting:
```

```
    image: mongo:3.6
    imagePullPolicy: Always
    imagePullSecret: {}
  member: {}
  replicaset: 5
  resource:
    limits:
      cpu: "2"
      memory: 512Mi
    requests:
      cpu: "1"
      memory: 512Mi
  scheduler:
    schedulerMode: Uniform
    schedulerName: multicloud-middleware-scheduler
  spreadConstraints: {}
  storage:
    storageClass: managed-nfs-storage
    storageSize: 1Gi
status:
  conditions:
  - lastTransitionTime: "2023-05-25T07:07:57Z"
    message: Service Dispatch Successful And Ready For External Service
    reason: ServerReady
    status: "True"
    type: ServerReady
  - lastTransitionTime: "2023-05-25T07:06:26Z"
    message: 'The number of member clusters is the same as the number of c
ontrol plane
      copies, check, SpecReplicaset: 5'
    reason: CheckSuccess
    status: "True"
```

```
    type: ServerCheck
  - lastTransitionTime: "2023-05-25T07:06:26Z"
    message: 'Get Scheduler Result From MultiCloudMongoDB Annotations Success (federation-mongo-operator/multicloudmongodb-sample):
      {"ClusterWithReplicaset":[{"cluster":"10-29-14-21","replicaset":4},{"cluster":"10-29-14-25","replicaset":1}]}'
    reason: GetSchedulerSuccess
    status: "True"
    type: ServerScheduledResult
  externalAddr: 10.29.5.103:33498,10.29.5.103:38640,10.29.5.103:37661,10.29.5.103:35880,10.29.5.107:38640
  result:
  - applied: true
    cluster: 10-29-14-21
    connectAddrWithRole:
      10.29.5.103:33498: SECONDARY
      10.29.5.103:35880: SECONDARY
      10.29.5.103:37661: SECONDARY
      10.29.5.103:38640: PRIMARY
    currentRevision: multicloudmongodb-sample-666cb9cb8
    replicasetSpec: 4
    replicasetStatus: 4
    state: Running
  - applied: true
    cluster: 10-29-14-25
    connectAddrWithRole:
     10.29.5.107:38640: SECONDARY
    currentRevision: multicloudmongodb-sample-866554df84
    replicasetSpec: 1
    replicasetStatus: 1
    state: Running
```

```
  state: Health
```

查看数据平面派生出来的 MongoDB CR 实例

??? note “点击查看 MongoDB YAML 示例”

```yaml
apiVersion: middleware.fedstate.io/v1alpha1
kind: MongoDB
metadata:
  annotations:
    resourcebinding.karmada.io/name: multicloudmongodb-sample-mongodb
    resourcebinding.karmada.io/namespace: federation-mongo-operator
    resourcetemplate.karmada.io/uid: 66c5d156-88f7-445f-b068-15dcb327e452
  creationTimestamp: "2023-05-25T07:06:26Z"
  finalizers:
  - mongodb.finalizers.middleware.fedstate.io
  generation: 1
  labels:
    app.kubernetes.io/instance: multicloudmongodb-sample
    app.multicloudmongodb.io/vip: 10.29.5.103
    propagationpolicy.karmada.io/name: multicloudmongodb-sample
    propagationpolicy.karmada.io/namespace: federation-mongo-operator
    resourcebinding.karmada.io/key: 8484fbdb6f
    work.karmada.io/name: multicloudmongodb-sample-8484fbdb6f
    work.karmada.io/namespace: karmada-es-10-29-14-21
  name: multicloudmongodb-sample
  namespace: federation-mongo-operator
  resourceVersion: "232973338"
  selfLink: /apis/middleware.fedstate.io/v1alpha1/namespaces/federation-mongo-operator/mongodbs/multicloudmongodb-sample
  uid: f53b0751-ba6e-4172-b52a-e4ba611b522f
spec:
```

```
  dbUserSpec: {}
  image: mongo:3.6
  imagePullPolicy: IfNotPresent
  imagePullSecret: {}
  memberConfigRef: multicloudmongodb-sample-hostconf
  members: 4
  metricsExporterSpec:
    enable: false
  persistence:
    storage: 1Gi
  resources:
    limits:
      cpu: "2"
      memory: 512Mi
    requests:
      cpu: "1"
      memory: 512Mi
  rootPassword: 39nZzksAmXE=
  rsInit: true
  type: ReplicaSet
status:
  conditions:
  - lastTransitionTime: "2023-05-25T07:07:25Z"
    message: replset-0
    status: "True"
    type: rsInit
  - lastTransitionTime: "2023-05-25T07:07:38Z"
    message: replset-0
    status: "True"
    type: userRoot
  - lastTransitionTime: "2023-05-25T07:07:38Z"
    message: replset-0
```

```
    status: "True"
    type: userClusterAdmin
  currentInfo:
    members: 4
    resources:
      limits:
        cpu: "2"
        memory: 512Mi
      requests:
        cpu: "1"
        memory: 512Mi
  currentRevision: multicloudmongodb-sample-666cb9cb8
  replset:
  - _id: 0
    health: 1
    name: 10.29.5.103:33498
    state: 2
    stateStr: SECONDARY
    syncSourceHost: 10.29.5.107:38640
    syncingTo: 10.29.5.107:38640
  - _id: 1
    health: 1
    name: 10.29.5.103:38640
    state: 1
    stateStr: PRIMARY
    syncSourceHost: ""
    syncingTo: ""
  - _id: 2
    health: 1
    name: 10.29.5.107:38640
    state: 2
    stateStr: SECONDARY
```

```
    syncSourceHost: 10.29.5.103:38640
    syncingTo: 10.29.5.103:38640
  - _id: 3
    health: 1
    name: 10.29.5.103:37661
    state: 2
    stateStr: SECONDARY
    syncSourceHost: 10.29.5.107:38640
    syncingTo: 10.29.5.107:38640
  - _id: 4
    health: 1
    name: 10.29.5.103:35880
    state: 2
    stateStr: SECONDARY
    syncSourceHost: 10.29.5.107:38640
    syncingTo: 10.29.5.107:38640
  state: Running
```

社区

随着对有状态服务的跨云分发和自动化运维能力的需求不断增加，围绕这个场景的方案探索和实践，将变得更加有意义。

现在云原生联邦中间件 – FedState 项目已经在 Github 开源上线了，欢迎对多云、联邦场景有状态服务感兴趣的开发者， 来一起体验和试用，也非常欢迎大家一起参与进来一起贡献。

项目地址：https://github.com/fedstate/fedstate

通过 Docker 和 kind 在 macOS 电脑上安装社区版

本页说明如何使用 macOS 笔记本电脑创建单节点的 kind 集群，然后在线安装 DCE 5.0 社区版。

!!! tip

这是针对初学者的简化安装体验步骤，实际生产很少会使用 macOS，
原文作者是 [panpan0000](https://github.com/panpan0000)。

硬件环境

确认 MacBook 的性能和资源是否满足需求。最低配置为：

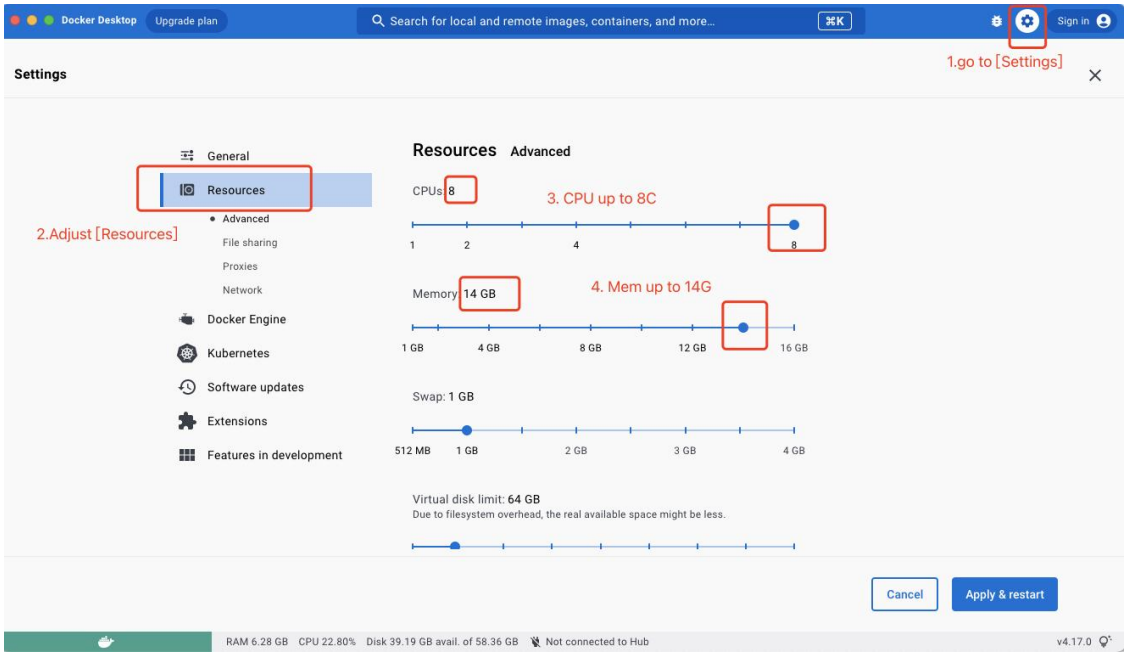
- CPU：**8 核**
- 内存：**16G**
- 磁盘剩余空间：大于 20G

安装和调整 Docker

根据 MacBook 的芯片（Intel 或 M1），安装 Docker Desktop。

调整容器资源上限：

1. 启动 Docker。
2. 点击右上角的 ，以打开 Settings 页面。
3. 点击左侧的 Resources，将启动容器的资源上限调节到 8C14G，点击 Apply & Restart 按钮。



调整资源

安装 kind

按照实际电脑情况，以下任选其一，安装 kind。 如果遇到其他问题，请参阅 kind 官方安装说明。

===“Mac 是 Intel 芯片”

```shell
[ $(uname -m) = x86_64 ]&& curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-darwin-amd64
```

===“Mac 是 M1/ARM 芯片”

```shell
[ $(uname -m) = arm64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-darwin-arm64
chmod +x ./kind
sudo mv kind /usr/local/bin/kind
```

===“通过 Homebrew 安装 kind”

安装 Homebrew：

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

安装 kind

```shell
brew install kind
```

最后执行以下命令，确认 kind 安装成功：

```
kind version
```

创建 kind 配置文件

暴露集群内的 32088 端口到 kind 对外的 8888 端口（可自行修改）：

```
cat > kind_cluster.yaml << EOF
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 32088
    hostPort: 8888
EOF
```

kind 创建 K8s 集群

以 K8s 1.25.3 版本为例，执行以下命令创建一个 K8s 集群：

```
kind create cluster --image docker.m.daocloud.io/kindest/node:v1.25.3 --name=fire-kind-cluster --config=kind_cluster.yaml
```

确认 kind 集群创建是否成功：

```
docker exec -it fire-kind-cluster-control-plane  kubectl get no
```

期望输出：

```
NAME                                STATUS   ROLES           AGE   VERSION
fire-kind-cluster-control-plane     Ready    control-plane   18h   v1.25.3
```

安装 DCE 5.0 社区版

1. 安装依赖项

   ```
   cat <<EOF | docker exec -i fire-kind-cluster-control-plane  bash
   curl -LO https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite.sh
   bash install_prerequisite.sh online community
   apt-get update && apt-get install -y wget
   EOF
   ```

2. 下载 dce5-installer 二进制文件

   ```
   docker exec -it fire-kind-cluster-control-plane /bin/bash
   ```

   假定 VERSION=v0.5.0

   ```
   export VERSION=v0.5.0;
   curl -Lo ./dce5-installer  https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/dce5-installer-$VERSION
   chmod +x ./dce5-installer
   exit
   ```

3. 安装 DCE5 社区版

1. 先获取本机 IP

   myIP=$(ip r get 1.1.1.1| awk '{print $NF}')

   如果报错 zsh: command not found: ip，有 2 个方案：

   - 执行 myIP=$(ifconfig en0| grep "inet[ ]" | awk '{print $2}')
   - 或通过 brew install iproute2mac 这类命令安装 iproute2mac 后重试。

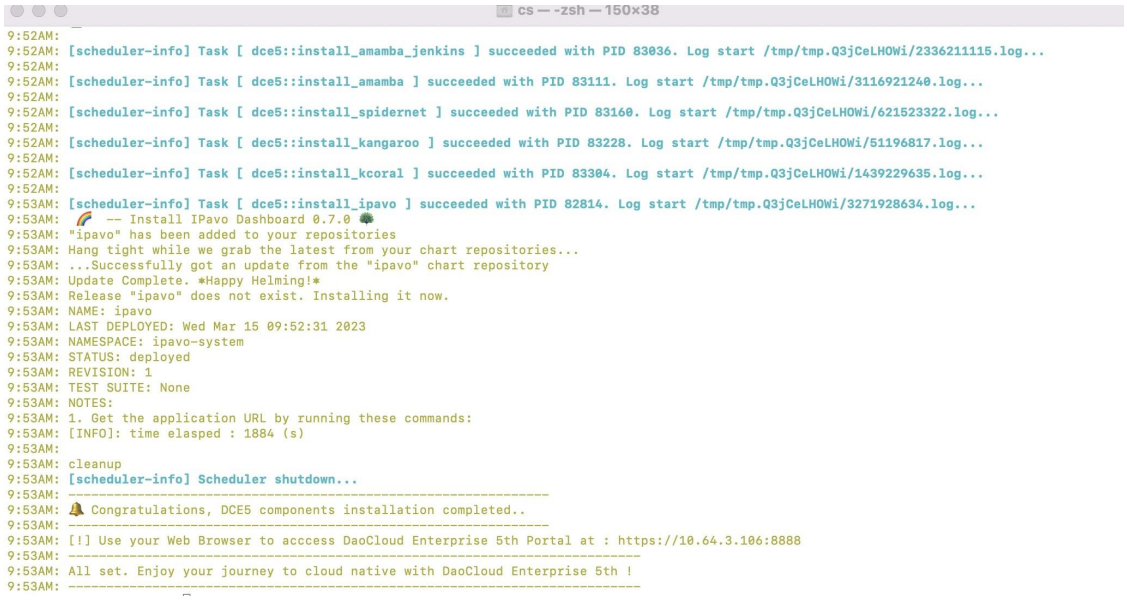
2. 开始安装，大概用时 30 分钟，取决于镜像拉取的网速

   docker exec -it fire-kind-cluster-control-plane bash -c "./dce5-installer install-app -z -k $myIP:8888"

4. 在安装过程中可以另起一个终端窗口，执行如下命令，观察 Pod 启动情况。

   docker exec -it fire-kind-cluster-control-plane kubectl get po -A -w

   当看到以下提示，则表示 DCE 5.0 社区版安装成功了。



success

5. 输入默认用户和密码（admin/changeme）登录后，系统将提示申请许可密钥。

体验使用

申请到许可证后，将进入 DCE 5.0 主界面，显示当前安装的组件、集群/节点/资源、告警等信息。

您可以尝试：

- 创建一个用户， 加入一个用户组， 赋予角色权限
- 定制软件界面
- 接入一个集群
- 管理你的节点
- 创建一个负载
- 更多请查阅文档站页面

卸载

1. 卸载 DCE 5.0 社区版。
2. 删除 kind 集群。

   ```
   kind delete cluster --name=fire-kind-cluster
   ```

3. 卸载 kind 本身。

   ```
   rm -f $(which kind)
   ```

4. 在应用列表中卸载 Docker。

至此您的 MacBook 恢复到了最初状态 ��

边缘场景玩转 seccomp 配置文件

作者: Sascha Grunert

译者: Michael Yao (DaoCloud)

Security Profiles Operator (SPO) 是一个功能丰富的 Kubernetes operator，相比以往可以简化 seccomp、SELinux 和 AppArmor 配置文件的管理。 从头开始记录这些配置文件是该 Operator 的关键特性之一，这通常涉及与大型 CI/CD 系统集成。 是否能够在边缘场景中测试 Operator 的记录能力是 SPO 的最新开发工作之一， 非常有助于轻松玩转 seccomp 配置文件。

使用 spoc record 记录 seccomp 配置文件

v0.8.0 版本的 Security Profiles Operator 附带一个名为 spoc 的全新命令行接口， 在各种其他场景中记录和回放 seccomp 配置文件的一些辅助工具不在这篇博文的讨论范围内。

记录 seccomp 配置文件需要执行一个二进制文件，这个二进制文件可以是一个只需调用 uname(2) 的简单 Golang 应用程序：

```
package main

import (
    "syscall"
)

func main() {
    utsname := syscall.Utsname{}
    if err := syscall.Uname(&utsname); err != nil {
        panic(err)
```

```
    }
}
```

可通过以下命令从代码中构建一个二进制文件：

```
> go build -o main main.go
> ldd ./main
        not a dynamic executable
```

现在可以从 GitHub 下载最新的 spoc 二进制文件， 并使用它在 Linux 上运行应用程序：

```
> sudo ./spoc record ./main
10:08:25.591945 Loading bpf module
10:08:25.591958 Using system btf file
libbpf: loading object 'recorder.bpf.o' from buffer
...
libbpf: prog 'sys_enter': relo #3: patched insn #22 (ALU/ALU64) imm 16 -> 16
10:08:25.610767 Getting bpf program sys_enter
10:08:25.610778 Attaching bpf tracepoint
10:08:25.611574 Getting syscalls map
10:08:25.611582 Getting pid_mntns map
10:08:25.613097 Module successfully loaded
10:08:25.613311 Processing events
10:08:25.613693 Running command with PID: 336007
10:08:25.613835 Received event: pid: 336007, mntns: 4026531841
10:08:25.613951 No container ID found for PID (pid=336007, mntns=4026531841,
 err=unable to find container ID in cgroup path)
10:08:25.614856 Processing recorded data
10:08:25.614975 Found process mntns 4026531841 in bpf map
10:08:25.615110 Got syscalls: read, close, mmap, rt_sigaction, rt_sigprocmask, m
advise, nanosleep, clone, uname, sigaltstack, arch_prctl, gettid, futex, sched_get
affinity, exit_group, openat
10:08:25.615195 Adding base syscalls: access, brk, capget, capset, chdir, chmod,
```

chown, close_range, dup2, dup3, epoll_create1, epoll_ctl, epoll_pwait, execve, faccessat2, fchdir, fchmodat, fchown, fchownat, fcntl, fstat, fstatfs, getdents64, getegid, geteuid, getgid, getpid, getppid, getuid, ioctl, keyctl, lseek, mkdirat, mknodat, mount, mprotect, munmap, newfstatat, openat2, pipe2, pivot_root, prctl, pread64, pselect6, readlink, readlinkat, rt_sigreturn, sched_yield, seccomp, set_robust_list, set_tid_address, setgid, setgroups, sethostname, setns, setresgid, setresuid, setsid, setuid, statfs, statx, symlinkat, tgkill, umask, umount2, unlinkat, unshare, write
10:08:25.616293 Wrote seccomp profile to: /tmp/profile.yaml
10:08:25.616298 Unloading bpf module

我必须以 root 用户身份执行 spoc，因为它将在内部运行一个 ebpf 程序， 通过重用 Security Profiles Operator 本身的相同代码部分。 我可以看到 bpf 模块已成功加载，并且 spoc 已将所需的跟踪点附加到该模块。 随后该模块将使用其挂载命名空间跟踪主要应用程序并处理记录的系统调用数据。 ebpf 程序的本质是监视整个内核的上下文，这意味着 spoc 跟踪系统的所有系统调用，但不会干涉其执行过程。

这些日志表明 spoc 找到了包括 uname 在内的 read、close、mmap 等系统调用。 除 uname 之外的所有系统调用都来自 Golang 运行时及其垃圾回收，这已经为我们演示中的基本应用增加了开销。 我还可以从日志行 Adding base syscalls: … 中看到 spoc 将一堆基本系统调用添加到了生成的配置文件中。 这些系统调用由 OCI 运行时（如 runc 或 crun）使用以便能够运行容器。 这意味着 spoc 可用于记录直接被容器化的 seccomp 配置文件。 这种行为可以通过在 spoc 中使用 --no-base-syscalls/-n 禁用，或通过 --base-syscalls/-b 命令行标志进行自定义。这对于使用除了 crun 和 runc 之外的不同 OCI 运行时或者如果我只想记录应用的 seccomp 配置文件并将其与另一个基本配置文件堆叠时非常有帮助。

生成的配置文件现在位于 /tmp/profile.yaml， 但可以使用 --output-file value/-o 标志更改默认位置：

```
> cat /tmp/profile.yaml

apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  creationTimestamp: null
  name: main
spec:
  architectures:
    - SCMP_ARCH_X86_64
  defaultAction: SCMP_ACT_ERRNO
  syscalls:
    - action: SCMP_ACT_ALLOW
      names:
        - access
        - arch_prctl
        - brk
        - ...
        - uname
        - ...
status: {}
```

seccomp 配置文件 CRD 可直接与 Security Profiles Operator 一起使用，统一在 Kubernetes 中进行管理。 spoc 还可以通过使用 --type/-t raw-seccomp 标志生成原始的 seccomp 配置文件（格式为 JSON）：

```
> sudo ./spoc record --type raw-seccomp ./main
...
52.628827 Wrote seccomp profile to: /tmp/profile.json

> jq . /tmp/profile.json

{
  "defaultAction": "SCMP_ACT_ERRNO",
```

```
  "architectures": ["SCMP_ARCH_X86_64"],
  "syscalls": [
    {
      "names": ["access", "...", "write"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

实用程序 spoc record 允许我们直接从任何能够在内核中运行 ebpf 代码的 Linux 系统中记录复杂的 seccomp 配置文件。但它还可以做更多事情： 例如修改 seccomp 配置文件并使用 spoc run 进行测试。

使用 spoc run 运行 seccomp 配置文件

spoc 还能够使用应用了 seccomp 配置文件的二进制文件，轻松测试对其所做的任何修改。 要执行此操作，只需运行：

```
> sudo ./spoc run ./main
10:29:58.153263 Reading file /tmp/profile.yaml
10:29:58.153311 Assuming YAML profile
10:29:58.154138 Setting up seccomp
10:29:58.154178 Load seccomp profile
10:29:58.154189 Starting audit log enricher
10:29:58.154224 Enricher reading from file /var/log/audit/audit.log
10:29:58.155356 Running command with PID: 437880
>
```

看起来应用程序已成功退出，这是可以预期的，因为我尚未修改先前记录的配置文件。 我还可以使用 --profile/-p 标志指定配置文件的自定义位置，但这并不是必需的， 因为我没有修改默认输出位置。spoc 将自动确定它是基于原始的（JSON）还是基于 CRD 的 （YAML）seccomp 配置文件，然后将其应用于这个过程。

Security Profiles Operator 支持 log enricher 特性， 通过解析审计日志提供附加的 seccomp 相关信息。 spoc run 以同样的方式使用 enricher 向最终用户提供更多数据以调试 seccomp 配置文件。

现在我不得不修改配置文件来查看输出中有价值的信息。 例如，我可以移除允许的 uname 系统调用：

```
> jq 'del(.syscalls[0].names[] | select(. == "uname"))' /tmp/profile.json > /tmp/no-uname-profile.json
```

然后尝试用新的配置文件 /tmp/no-uname-profile.json 来运行：

```
> sudo ./spoc run -p /tmp/no-uname-profile.json ./main
10:39:12.707798 Reading file /tmp/no-uname-profile.json
10:39:12.707892 Setting up seccomp
10:39:12.707920 Load seccomp profile
10:39:12.707982 Starting audit log enricher
10:39:12.707998 Enricher reading from file /var/log/audit/audit.log
10:39:12.709164 Running command with PID: 480512
panic: operation not permitted

goroutine 1 [running]:
main.main()
        /path/to/main.go:10 +0x85
10:39:12.713035 Unable to run: launch runner: wait for command: exit status 2
```

好的，这符合预期！应用的 seccomp 配置文件阻止了 uname 系统调用，导致出现“operation not permitted”错误。此错误提示过于宽泛，没有提供关于 seccomp 阻止了什么的任何提示。 通常情况下，如果 seccomp 禁止某个系统调用，很难预测应用程序会做出什么行为。 可能应用程序像这个简单演示中一样终止，但也可能导致奇怪的异常行为使得应用程序根本无法停止。

现在，如果我将配置文件的默认 seccomp 操作从 SCMP_ACT_ERRNO 更改为 SCMP_ACT_LOG，就像这样：

```
> jq '.defaultAction = "SCMP_ACT_LOG"' /tmp/no-uname-profile.json > /tmp/no-uname-profile-log.json
```

那么 log enricher 将提示我们 uname 系统调用在使用 spoc run 时被阻止：

```
> sudo ./spoc run -p /tmp/no-uname-profile-log.json ./main
10:48:07.470126 Reading file /tmp/no-uname-profile-log.json
10:48:07.470234 Setting up seccomp
10:48:07.470245 Load seccomp profile
10:48:07.470302 Starting audit log enricher
10:48:07.470339 Enricher reading from file /var/log/audit/audit.log
10:48:07.470889 Running command with PID: 522268
10:48:07.472007 Seccomp: uname (63)
```

应用程序现在不会再终止，但 seccomp 将行为记录到 /var/log/audit/audit.log 中， 而 spoc 会解析数据以将其直接与我们的程序相关联。将日志消息生成到审计子系统中会带来巨大的性能开销， 在生产系统中应小心处理。当在生产环境中以审计模式运行不受信任的应用时，也会带来安全风险。

本文的演示希望让你了解如何使用 Security Profiles Operator 各项特性所赋予的全新辅助工具来调试应用程序的 seccomp 配置文件问题。 spoc 是一个灵活且可移植的二进制文件，适用于资源有限的边缘场景， 甚至是 Kubernetes 本身可能无法利用其全部功能的场景中。

感谢阅读这篇博文！如果你有兴趣了解更多，想提出反馈或寻求帮助，请通过 Slack (#security-profiles-operator) 或邮件列表直接与我们联系。

参考资料

- 英文博客：Having fun with seccomp profiles on the edge
- 使用 seccomp 限制容器的系统调用

保姆式安装 DCE 5.0 社区版

作者：Peter Pan, SAMZONG

本文在 3 个节点的集群中完成了从 0 到 1 的 DCE 5.0 社区版安装，包含了 K8s 集群、依赖项、网络、存储等细节及更多注意事项。

> 现阶段版本迭代较快，本文的安装方式可能与最新版有所差异，请以产品文档的安装说明为准。

集群规划

使用 3 台 UCloud 的 VM，配置均为 8 核 16G。

角色	主机名	操作系统	IP	配置
control-plane	k8s-master01	CentOS 8.3	10.23.*	8 核 16G 系统盘 200GB
worker-node	k8s-work01	CentOS 8.3	10.23.*	8 核 16G 系统盘 200GB
worker-node	k8s-work02	CentOS 8.3	10.23.*	8 核 16G 系统盘 200GB

本示例采用的组件为：

- Kubernetes：1.25.8
- CRI：containerd（因为新版本 K8s 已经不再直接支持 Docker）
- CNI：Calico
- StorageClass：local-path

- DCE 5.0 社区版：v0.5.0

准备节点

本节所述的所有操作都是必需的。

节点配置

安装前先分别对 3 个节点做了一些必要的设置。

1. 配置主机名。修改主机名（可选），避免主机名重名即可。

   ```
   hostnamectl set-hostname k8s-master01
   hostnamectl set-hostname k8s-work01
   hostnamectl set-hostname k8s-work02
   ```

   建议修改后 exit 退出 SSH 会话，重新登录以显示新的主机名。

2. 禁用 Swap

   ```
   swapoff -a
   sed -i '/ swap / s/^/#/' /etc/fstab
   ```

3. 关闭防火墙（可选）

   ```
   systemctl stop firewalld
   systemctl disable firewalld
   ```

4. 设置内核参数并允许 iptables 进行桥接流量

   加载 br_netfilter 模块：

   ```
   cat <<EOF | tee /etc/modules-load.d/kubernetes.conf
   br_netfilter
   EOF

   # 加载模块
   ```

```
sudo modprobe overlay
sudo modprobe br_netfilter
```

修改内核参数如 ip_forward 和 bridge-nf-call-iptables：

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# 刷新配置
sysctl --system
```

安装容器运行时（containerd）

1. 如果是 CentOS 8.x，要先卸载系统预装的 Podman，否则会版本冲突（[注意]��）

```
yum erase podman buildah -y
```

2. 安装依赖

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

3. 安装 containerd，可以用二进制也可以用 yum 包（yum 是 docker 社区维护的，本例中使用 yum 包）

```
sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
sudo yum makecache
yum install containerd.io -y
ctr -v  # 显示安装的版本，例如 ctr containerd.io 1.6.20
```

4. 修改 containerd 的配置文件

```
# 删除自带的 config.toml，避免后续 kubeadm 出现错误 CRI v1 runtime API is not implemented for endpoint
mv /etc/containerd/config.toml /etc/containerd/config.toml.old

# 重新初始化配置
sudo containerd config default | sudo tee /etc/containerd/config.toml

# 更新配置文件内容：使用 systemd 作为 cgroup 驱动，并且替代 pause 镜像地址
sed -i 's/SystemdCgroup\ =\ false/SystemdCgroup\ =\ true/' /etc/containerd/config.toml
sed -i 's/k8s.gcr.io\/pause/k8s-gcr.m.daocloud.io\/pause/g' /etc/containerd/config.toml # 老的 pause 地址
sed -i 's/registry.k8s.io\/pause/k8s-gcr.m.daocloud.io\/pause/g' /etc/containerd/config.toml
sudo systemctl daemon-reload
sudo systemctl restart containerd
sudo systemctl enable containerd
```

5. 安装 CNI

```
curl -JLO https://github.com/containernetworking/plugins/releases/download/v1.2.0/cni-plugins-linux-amd64-v1.2.0.tgz
mkdir -p /opt/cni/bin && tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.2.0.tgz
```

6. 安装 nerdctl（可选）

```
curl -LO https://github.com/containerd/nerdctl/releases/download/v1.2.1/nerdctl-1.2.1-linux-amd64.tar.gz
tar xzvf nerdctl-1.2.1-linux-amd64.tar.gz
mv nerdctl /usr/local/bin
nerdctl -n k8s.io ps # 查看容器
```

安装 k8s 集群

安装 k8s 二进制组件

在三个节点上都需要如下操作：

1. 安装 Kubernetes 软件源（这里采用国内阿里云的源加速）

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

2. 将 SELinux 设置为 permissive 模式（相当于将其禁用）

```
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

3. 安装 Kubernetes 组件，版本以 1.25.8 为例（DCE 5.0 对 1.26 暂时不支持）

```
export K8sVersion=1.25.8
sudo yum install -y kubelet-$K8sVersion
sudo yum install -y kubeadm-$K8sVersion
sudo yum install -y kubectl-$K8sVersion
sudo systemctl enable --now kubelet
```

kubeadm 安装第一个 master 节点

1. 预先下载镜像以加速安装，使用 DaoCloud 的加速仓库

   ```
   # 指定 K8s 版本，并拉取镜像
   kubeadm config images pull --image-repository k8s-gcr.m.daocloud.io --kubernetes-version=v1.25.8
   ```

2. 调用 kubeadm 初始化第一个节点（使用 DaoCloud 加速仓库）

   !!! note

   如下 Pod CIDR 不能与宿主机物理网络的网段重合（该 CIDR 未来还需要跟 Calico 的配置一致）。

   ```
   sudo kubeadm init --kubernetes-version=v1.25.8 --image-repository=k8s-gcr.m.daocloud.io --pod-network-cidr=192.168.0.0/16
   ```

   经过十几分钟，你能看到打印成功的信息如下（请记住最后打印出的 kubeadm join 命令和相应 token，后续会用到 ）

   ```
   Your Kubernetes control-plane has initialized successfully!
   To start using your cluster, you need to run the following as a regular user:

   mkdir -p $HOME/.kube
   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   sudo chown $(id -u):$(id -g) $HOME/.kube/config

   Alternatively, if you are the root user, you can run:
   export KUBECONFIG=/etc/kubernetes/admin.conf
   You should now deploy a pod network to the cluster.
   Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
   https://kubernetes.io/docs/concepts/cluster-administration/addons/
   ```

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 10.23.207.16:6443 --token p4vw62.shjjzm1ce3fza6q7 \
--discovery-token-ca-cert-hash sha256:cb1946b96502cbd2826c52959d0400b6e214e06cc8462cdd13c1cb1dc6aa8155
```

3. 配置 kubeconfig 文件，以便用 kubectl 更方便管理集群

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
kubectl get no # 你能看到第一个节点，但是仍然 NotReady
```

4. 安装 CNI，以 Calico 为例

【请以官方安装方案为准。参考官方 Calico 安装文档】

1. 先安装 calico-operator

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml
```

2. 再下发经过修改/配置过的 CR

```
# 下载配置文件模板
curl -LO https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/custom-resources.yaml
grep cidr custom-resources.yaml #打印出默认的 CIDR
# 请确认 calico 配置文件里的 CIDR 和之前 kubeadm init 时的 CIDR 是一致的！！！否则请修改!!!
vim custom-resources.yaml
kubectl apply -f custom-resources.yaml
```

3. 等待部署成功

```
kubectl get po -n calico-system -w # 等待 Pod 都 Running
kubectl get no # 可以看到第一个节点变为 ready 状态了
```

接入其他 worker 工作节点

最后在其他 worker 节点执行 join 命令。 在上述 master 节点执行 kubeadm init 时最后会在屏幕打出（注意三个参数都是跟环境相关的，请勿直接拷贝）

```
kubeadm join $第一台 master 的 IP:6443 --token p...7 --discovery-token-ca-cert-hash s....x
```

成功 join 之后，输出类似于：

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

在 master 节点确认节点都被接入，并且等待其都变为 Ready 状态。

```
kubectl get no -w
```

安装默认存储 CSI（使用本地存储）

```
# 参考： https://github.com/rancher/local-path-provisioner
wget https://raw.githubusercontent.com/rancher/local-path-provisioner/v0.0.24/deploy/local-path-storage.yaml
sed -i "s/image: rancher/image: docker.m.daocloud.io\/rancher/g" local-path-storage.yaml # 替换 docker.io 为实际镜像
sed -i "s/image: busybox/image: docker.m.daocloud.io\/busybox/g" local-path-storage.yaml
kubectl apply -f local-path-storage.yaml
kubectl get po -n local-path-storage -w # 等待 Pod 都 running

# 把 local-path 设置为默认 SC
```

```
kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
kubectl get sc # 可以看到形如：local-path (default)
```

安装 DCE 5.0 社区版

现在一切准备就绪，开始安装 DCE 5.0 社区版。

安装基础依赖

```
curl -LO https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite.sh
bash install_prerequisite.sh online community
```

下载 dce5-installer

```
export VERSION=v0.5.0
curl -Lo ./dce5-installer https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/dce5-installer-$VERSION
chmod +x ./dce5-installer
```

确认节点的外部可达 IP 地址

1. 如果你的浏览器跟 master 节点的 IP 是可以直通的，无需额外操作。
2. 如果 master 节点的 IP 是内网（比如本示例的公有云机器）：
    - 请在公有云中为其创建外网可达的 IP
    - 请在公有云配置中，在该主机的的防火墙规则中，允许 32088 端口的进出
    - 如上的 32088 端口是 kubectl -n istio-system get svc istio-ingressgateway 的 NodePort 端口



image

执行安装

1. 如果你的浏览器跟 master 节点的 IP 是可以直通的，直接执行

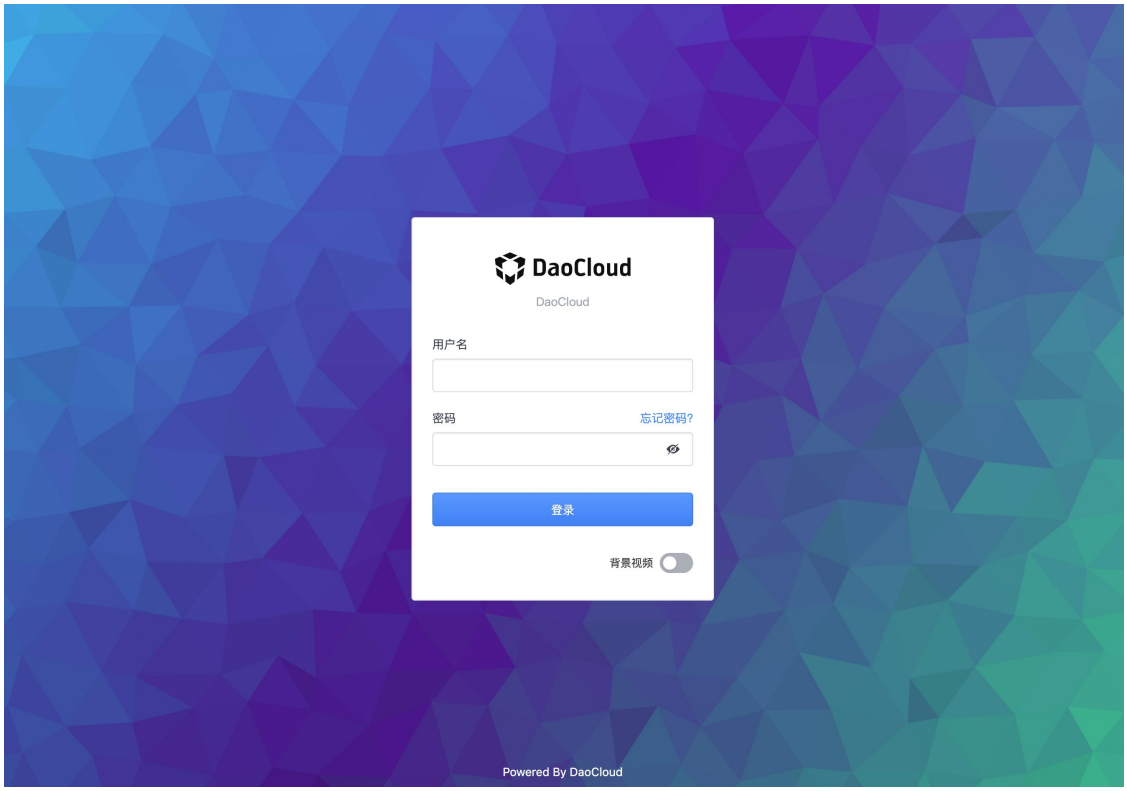
   ./dce5-installer install-app -z -k

2. 如果 master 节点的 IP 是内网（比如本示例的公有云机器），请确认上述外部 IP 和防火墙配置完毕，然后执行如下命令：

   ./dce5-installer install-app -z -k $外部 IP:32088

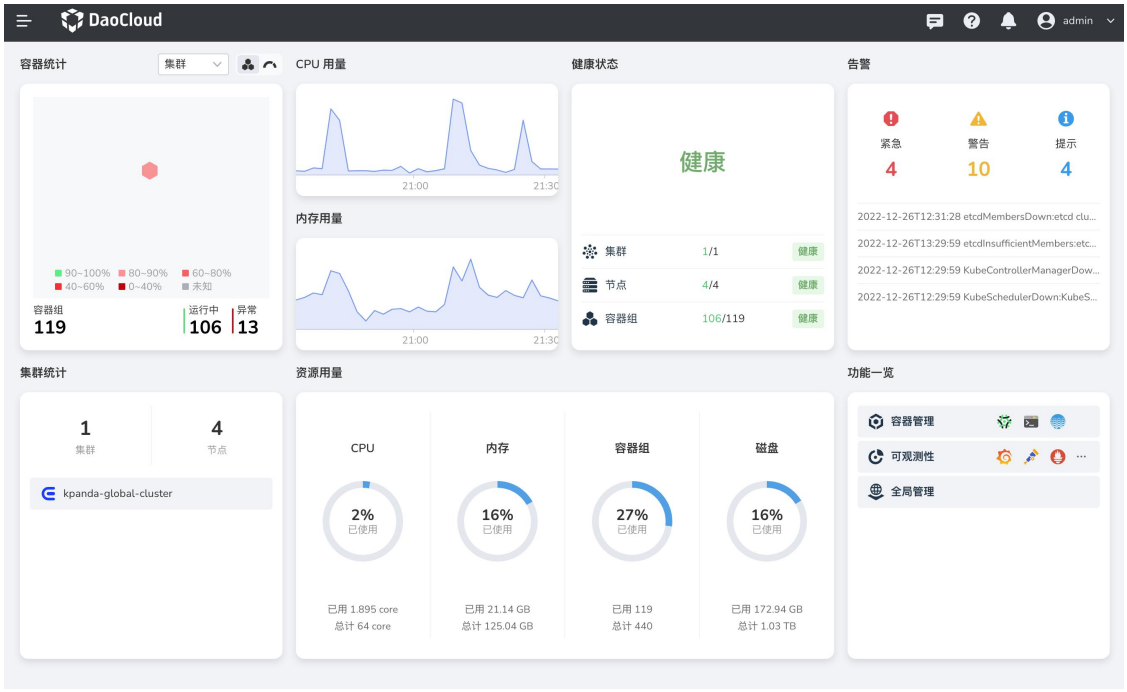
   注意：上述的 32088 是 kubectl -n istio-system get svc istio-ingressgateway 的 NodePort 端口

3. 在浏览器中打开登录界面。



登录

4. 以用户名 admin 密码 changeme 登录 DCE 5.0。



成功登录

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

Spiderpool：Calico 固定应用 IP 的一种新选择



spiderpool

Calico 是一套开源的网络和网络安全解决方案，也是作为 Kubernetes 容器网络解决方案（CNI: Container Network Interface）的一种实现。 它基于 Linux 纯三层路由实现，在一些场景下用户可以将 Pod 子网的路由通过 Calico BGP 模式宣告到网关， 来自集群外的客户端可以直接通过 Pod 的 IP 访问 Pod，同时还可以保留客户端的源 IP。

当前痛点

在 Calico Underlay 网络模式下，用户还希望 Deployment 或 StatefulSet 类型应用的 IP 地址，能够被固定或是在一个指定的 IP 范围中浮动，因为:

- Pod 的 IP 地址常常受防火墙策略管控，防火墙只会允许特定的 IP 或者 IP 范围内的目标访问
- 传统微服务应用直接使用 Pod IP 进行微服务注册
- 一些场景下业务 IP 为固定的 IP，不会变化

Calico 通过在 Pod 中注入注解: cni.projectcalico.org/ipAddrs 可以实现 Pod 级别的 IP 固定，但是 **在使用中我们发现以下不足** ：

- 固定 IP 只在 Pod 级别生效，对 Deployment、StatefulSet 类型无能为力；
- 需要管理员保证 Pod 的注解 IP 是不重叠的，避免出现 IP 冲突。尤其是在大规模集群下，难以排查冲突的 IP；
- 固定 IP 地址的配置、管理繁琐，不具有云原生化。

解决方式：Spiderpool

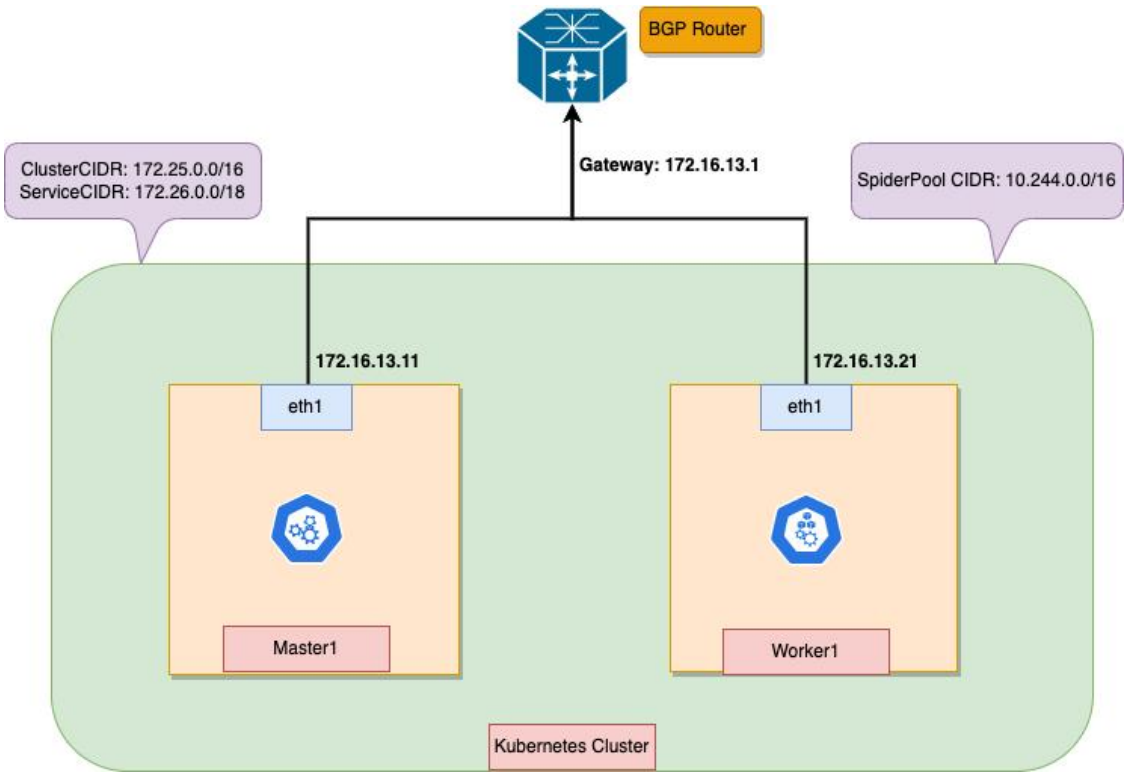
针对上述不足，我们尝试通过 Spiderpool 来实现这点需求。 Spiderpool 是一个 Kubernetes 的 IPAM 插件项目，其主要针对于 Underlay 网络的 IP 地址管理需求而设计， 能够为任何兼容第三方 IPAM 插件的 CNI 项目所使用。它主要具有以下几个特点：

- 能为 Deployment、StatefulSet 类型自动化分配固定 IP 地址，并且做到 IP 地址数量随副本数量自动扩缩容
- 以 CRD 方式管理、配置 IP 池，极大减少运维成本
- 支持第三方控制器创建的 Pod
- 支持为 Pod 多网卡指定不同子网

下面将对 Spiderpool 进行展开介绍。

安装 Spiderpool

以下是笔者参考 Spiderpool 官方文档搭建了一套 Calico BGP 模式搭配 Spiderpool 的环境，网络环境拓扑图如下:



spiderpool

根据自身环境创建 SpiderSubnet 实例: nginx-subnet-v4，体验 Spiderpool 固定 IP 等功能。

```
[root@master ~]# kubectl get ss
NAME                VERSION   SUBNET          ALLOCATED-IP-COUNT   TOTAL-IP-COUNT
nginx-subnet-v4     4         10.244.0.0/16   0                    25602
```

自动为应用固定 IP 池

该功能支持从 SpiderSubnet(10.244.0.0/16) 子网中自动创建 IP 池，并绑定到应用 Pod。并且支持 Pod IP 固定、IP 池数量随副本数量自动扩缩容等功能。

使用 Spiderpool 自动池功能，创建两个副本的 Nginx Deployment 应用:

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
```

```
  metadata:
    name: nginx
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        annotations:
          ipam.spidernet.io/subnet: '{"ipv4":["nginx-subnet-v4"]}'
          ipam.spidernet.io/ippool-ip-number: '+3'
        labels:
          app: nginx
      spec:
        containers:
        - name: nginx
          image: nginx
          imagePullPolicy: IfNotPresent
          ports:
          - name: http
            containerPort: 80
            protocol: TCP
```

- ipam.spidernet.io/subnet：Spiderpool 从“nginx-subnet-v4”子网中随机选择一些 IP 来创建固定 IP 池，并与应用绑定
- ipam.spidernet.io/ippool-ip-number：‘+3’表示 IP 池的 IP 总量为比副本数多三个，解决应用滚动发布时有临时 IP 可用

当应用 Deployment 被创建，Spiderpool 中自动创建了一个名为 auto-nginx-v4-eth0-452e737e5e12 的 IP 池， 并与应用绑定。IP 范围为：10.244.100.90-10.244.100.95，池 IP 数量为 5：

```
[root@master1 ~]# kubectl get po -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP
              NODE      NOMINATED NODE   READINESS GATES
nginx-644659db67-6lsmm   1/1     Running   0          12s   10.244.10
0.93   worker5   <none>           <none>
nginx-644659db67-n7ttd   1/1     Running   0          12s   10.244.100.
91     master1   <none>           <none>

[root@master1 ~]# kubectl get sp
NAME                                VERSION   SUBNET          ALLOCATED-
IP-COUNT   TOTAL-IP-COUNT   DEFAULT   DISABLE
auto-nginx-v4-eth0-452e737e5e12     4         10.244.0.0/16   2
          5                false     false

[root@master ~]# kubectl get sp auto-nginx-v4-eth0-452e737e5e12 -o jsonpath
='{.spec.ips}'
["10.244.100.90-10.244.100.95"]
```

Pod 的 IP 被固定在自动池: auto-nginx-v4-eth0-452e737e5e12(10.244.100.90-10.244.100.95) 中范围内，重启 Pod 其 IP 也被固定在此范围内:

```
[root@master1 ~]# kubectl get po -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP
              NODE      NOMINATED NODE   READINESS GATES
nginx-644659db67-szgcg   1/1     Running   0          23s   10.244.10
0.90   worker5   <none>           <none>
nginx-644659db67-98rcg   1/1     Running   0          23s   10.244.10
0.92   master1   <none>           <none>
```

扩容副本，新副本的 IP 地址从自动池: auto-nginx-v4-eth0-452e737e5e12(10.244.100.90-10.244.100.95) 中自动分配，并且 IP 池随副本数量自动扩容:

```
[root@master1 ~]# kubectl scale deploy nginx --replicas 3  # scale pods
deployment.apps/nginx scaled

[root@master1 ~]# kubectl get po -o wide
NAME                     READY   STATUS    RESTARTS   AGE     IP
              NODE      NOMINATED NODE   READINESS GATES
nginx-644659db67-szgcg   1/1     Running   0          1m      10.244.100.
90    worker5   <none>           <none>
nginx-644659db67-98rcg   1/1     Running   0          1m      10.244.100.
92    master1   <none>           <none>
nginx-644659db67-brqdg   1/1     Running   0          10s     10.244.100.
94    master1   <none>           <none>

[root@master1 ~]# kubectl get sp
NAME                                VERSION   SUBNET          ALLOCATED-
IP-COUNT   TOTAL-IP-COUNT   DEFAULT   DISABLE
auto-nginx-v4-eth0-452e737e5e12     4         10.244.0.0/16   3
       6                false     false
```

手动指定 IP 池

在某些场景下，用户希望直接应用 IP 从一个固定的 IP 范围中分配，而不是由 Spiderpool 随机分配，下面是将展示此功能:

创建 IP 池:

```
cat << EOF | kubectl apply -f -
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: nginx-v4-ippool
  spec:
    ipVersion: 4
    subnet: 10.244.0.0/16
```

```
  ips:
  - 10.244.120.10-10.244.120.20
```

spec.subnet 表示该 IP 池属于哪一个子网

spec.ips 固定 IP 地址范围，范围为 10.244.120.10-10.244.120.20，共计 10 个 IP

通过注解 ipam.spidernet.io/ippool 手动指定 IP 池：nginx-v4-ippool，创建应用：nginx-m：

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-m
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        ipam.spidernet.io/ippool: '{"ipv4":["nginx-v4-ippool"]}'
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
        - name: http
```

```
        containerPort: 80
        protocol: TCP
```

Spiderpool 从 nginx-v4-ippool 池中分配两个 IP 并绑定到应用。Pod 无论重启或是扩容，仍然从该池中分配 IP ，达到固定 IP 的效果:

```
[root@master1 ~]# kubectl get po -o wide | grep nginx-m
NAME                          READY   STATUS    RESTARTS   AGE     IP
              NODE      NOMINATED NODE   READINESS GATES
nginx-m-7c879df6bc-26dcq      1/1     Running   0          23s     10.244.
120.12    worker5   <none>           <none>
nginx-m-7c879df6bc-nwdtp      1/1     Running   0          23s     10.244.
120.14    master1   <none>           <none>

[root@master1 ~]# kubectl get sp
NAME                                VERSION   SUBNET          ALLOCATED-
IP-COUNT   TOTAL-IP-COUNT   DEFAULT   DISABLE
auto-nginx-v4-eth0-452e737e5e12     4         10.244.0.0/16   3
          6                false     false
nginx-v4-ippool                     4         10.244.0.0/16   2
          11               false     false
```

结论

经过测试: 集群外客户端可直接通过 Nginx Pod 的 IP 正常访问，集群内部通讯 Nginx Pod 跨节点也都通信正常（包括跨 Calico 子网）。 在 Calico underlay 场景下，我们可借助 Spiderpool 轻松帮助我们实现 Deployment 等类型应用固定 IP 的需求，这也为该场景下提供了一个新的选择。

更多功能介绍参考 Spiderpool 官方文档。

了解 DCE 5.0 云原生网络

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

Kubernetes 1.27：加快了 Pod 启动速度

投稿：Paco Xu、Michael Yao

如何在大型集群中加快节点上的 Pod 启动？这是企业中集群管理员常常会面临的问题。

这篇博文重点介绍了从 kubelet 一侧加快 Pod 启动的方法。此方法不涉及通过 kube-apiserver 由 controller-manager 创建 Pod 所用的时间段， 也不包含 Pod 的调度时间或在其上执行 Webhook 的时间。

本文从 kubelet 的角度考虑，提及了一些重要的影响因素，但并不会详尽罗列所有诱因。

Kubernetes v1.27 已正式发布，本文也强调了 v1.27 中有助于加快 Pod 启动的一些主要变更。

并行容器镜像拉取

拉取镜像总是需要一些时间的，更糟的是镜像拉取默认是以串行方式作业的。 换言之，kubelet 一次只会向镜像服务发送一个镜像拉取请求。 其他的镜像拉取请求必须等到正在处理的拉取请求被完成才能进行。

要启用并行镜像拉取，请在 kubelet 配置中将 serializeImagePulls 字段设置为 false。 当 serializeImagePulls 被禁用时，将立即向镜像服务发送镜像拉取请求，并可以并行拉取多个镜像。

设定并行镜像拉取最大值有助于防止节点因镜像拉取而过载

我们在 kubelet 中引入了一个新特性，可以在节点级别设置并行镜像拉取的限值。 此限值限制了可以同时拉取的最大镜像数量。如果有个镜像拉取请求超过了这个限值， 该请求将被阻止，直到其中一个正在进行的镜像拉取完成为止。 在启用此特性之前，请确保容器运行时的镜像服务可以有效处理并行镜像拉取。

要限制并行镜像拉取的数量，你可以在 kubelet 中配置 maxParallelImagePulls 字段。 将 maxParallelImagePulls 的值设置为 **n** 后，并行拉取的镜像数将不能超过 **n** 个。 超过此限值的任何其他镜像拉取请求都需要等到至少一个正在进行的拉取被完成为止。

你可以在关联的 KEP 中找到更多细节： Kubelet 并行镜像拉取数限值 (KEP-3673)。

提高了 kubelet 默认 API 每秒查询限值

为了在节点上具有多个 Pod 的场景中加快 Pod 启动，特别是在突然扩缩的情况下， kubelet 需要同步 Pod 状态并准备 ConfigMap、Secret 或卷。这就需要大带宽访问 kube-apiserver。

在 v1.27 之前的版本中，kubeAPIQPS 的默认值为 5，kubeAPIBurst 的默认值为 10。 然而在 v1.27 中，kubelet 为了提高 Pod 启动性能，将这些默认值分别提高到了 50 和 100。 值得注意的是，提高 kubelet 的 API QPS 限值并不是唯一的原因。

1. 现在它有可能会被大幅限制（默认 QPS = 5）
2. 在大型集群中，它们仍然可能产生相当大的负载，因为数量很多
3. 有一个专用的 PriorityLevel 和 FlowSchema，这点我们可以轻松控制

以前在具有 50 个以上 Pod 的节点中，我们经常在 Pod 启动期间在 kubelet 上遇到 volume mount timeout。 特别是在使用裸金属节点时，我们建议集群操作员将 kubeAPIQPS 提高到 20，kubeAPIBurst 提高到 40。

更多细节请参阅 KEP https://kep.k8s.io/1040 和 PR#116121。

事件驱动的容器状态更新

在 v1.27 中，Evented PLEG （PLEG 是英文 Pod Lifecycle Event Generator 的缩写，表示“Pod 生命周期事件生成器”） 进阶至 Beta 阶段。Kubernetes 为 kubelet 提供了两种方法来检测 Pod 的生命周期事件， 例如容器中最后一个进程关闭。在 Kubernetes v1.27 中，**基于事件的** 机制已进阶至 Beta， 但默认

被禁用。如果你显式切换为基于事件的生命周期变更检测，则 kubelet 能够比依赖轮询的默认方法更快地启动 Pod。默认的轮询生命周期更改机制会增加明显的开销， 这会影响 kubelet 处理不同任务的并行能力，并导致性能和可靠性问题。出于这些原因，我们建议你将节点切换为使用基于事件的 Pod 生命周期变更检测。

更多细节请参阅 KEP https://kep.k8s.io/3386 和容器状态从轮询切换为基于 CRI 事件更新。

必要时提高 Pod 资源限值

在启动时，某些 Pod 可能会消耗大量的 CPU 或内存。 如果 CPU 限值较低，则可能会显著降低 Pod 启动过程的速度。 为了改善内存管理，Kubernetes v1.22 引入了一个名为 MemoryQoS 的特性门控。 该特性使 kubelet 能够在容器、Pod 和 QoS 级别上设置内存 QoS，以便更好地保护和确保内存质量。 尽管此特性门控有所好处，但如果 Pod 启动消耗大量内存，启用此特性门控可能会影响 Pod 的启动速度。

Kubelet 配置现在包括 memoryThrottlingFactor。该因子乘以内存限制或节点可分配内存， 可以设置 cgroupv2 memory.high 值来执行 MemoryQoS。 减小该因子将为容器 cgroup 设置较低的上限，同时增加了回收压力。 提高此因子将减少回收压力。默认值最初为 0.8，并将在 Kubernetes v1.27 中更改为 0.9。 调整此参数可以减少此特性对 Pod 启动速度的潜在影响。

更多细节请参阅 KEP https://kep.k8s.io/2570。

更多说明

在 Kubernetes v1.26 中，新增了一个名为 pod_start_sli_duration_seconds 的直方图指标， 用于显示 Pod 启动延迟 SLI/SLO 详情。此外，kubelet 日志现在会展示更多与 Pod 启动相关的时间戳信息，如下所示：

Dec 30 15:33:13.375379 e2e-022435249c-674b9-minion-group-gdj4 kubelet[8362]: I1230 15:33:13.375359 8362

pod_startup_latency_tracker.go:102] "Observed pod startup duration" pod="kube-system/konnectivity-agent-gnc9k" podStartSLOduration=-9.223372029479458e+09 pod.CreationTimestamp="2022-12-30 15:33:06 +0000 UTC" firstStartedPulling="2022-12-30 15:33:09.258791695 +0000 UTC m=+13.029631711" lastFinishedPulling="0001-01-01 00:00:00 +0000 UTC" observedRunningTime="2022-12-30 15:33:13.375009262 +0000 UTC m=+17.145849275" watchObservedRunningTime="2022-12-30 15:33:13.375317944 +0000 UTC m=+17.146157970"

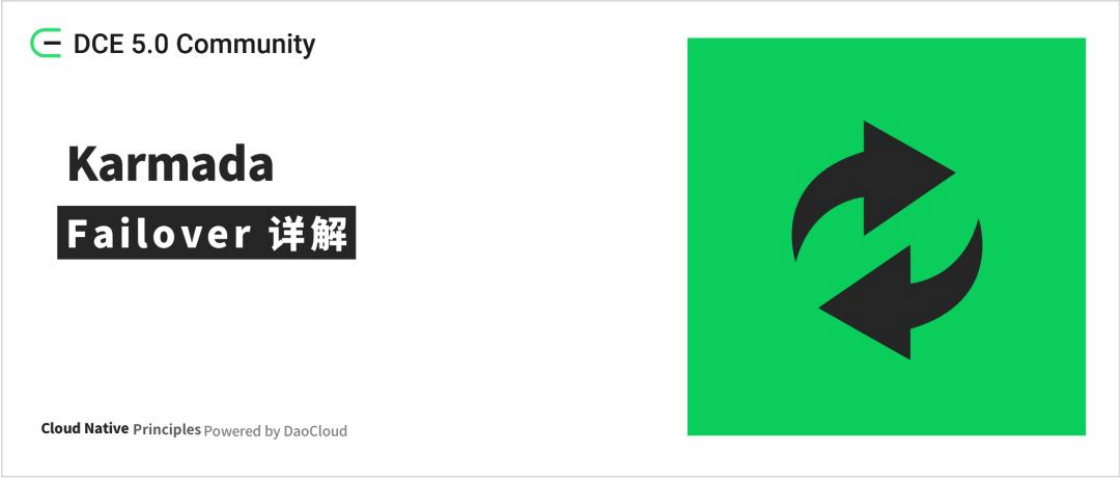
SELinux 挂载选项重标记功能在 v1.27 中升至 Beta 版本。 该特性通过挂载具有正确 SELinux 标签的卷来加快容器启动速度， 而不是递归地更改卷上的每个文件。更多细节请参阅 KEP https://kep.k8s.io/1710。

为了确定 Pod 启动缓慢的原因，分析指标和日志可能会有所帮助。 其他可能会影响 Pod 启动的因素包括容器运行时、磁盘速度、节点上的 CPU 和内存资源。

SIG Node 负责确保快速的 Pod 启动时间，而解决大型集群中的问题则属于 SIG Scalability 的范畴。

Karmada Failover 详解

作者：Fish-pro



karmada failover

多云时代，如何实现应用跨数据中心，跨可用区和跨集群高可用，成为我们探讨的新话题。 在单个集群中，如果集群发生故障，那么在集群中的所有应用将不可被访问。 是否有方式帮助我们在集群发生故障时，应用自动迁移到新的集群，保证应用持续的对外访问呢？ 显然，**Karmada 作为目前社区最火热的多云项目，提供了这样的能力。**

Karmada (Kubernetes Armada) 使用户能够跨多个集群运行云原生应用，而不需更改已有的应用。 通过使用 Kubernetes 原生 API 提供高级调度功能，实现了真正开放的多云。Karmada 旨在为多云和混合云场景下的多集群应用管理提供便捷的自动化，具有集中式多云管理、高可用性和故障恢复等关键特性。 **本文基于 Karmada 的 release 版本 v1.4.2，将和大家一起来探究一下 Karmada 跨集群的故障恢复是如何实现的** ， 有哪些控制器和调度器参与到这个过程中，以及每个控制器在这个过程中承担了什么样的能力，调度器承担了什么样的能力， 以及如何保证用户业务的高可用性和连续性?

如果您在阅读本文之前，还未了解或使用过 Karmada，推荐阅读：

1. Karmada 官方文档
2. 云原生多云应用利器 — Karmada 总览篇
3. 云原生多云应用利器 — Karmada 控制器
4. 云原生多云应用利器 — Karmada 调度器

为什么需要故障转移

首先，我们来梳理一下，实现故障转移的必要性：

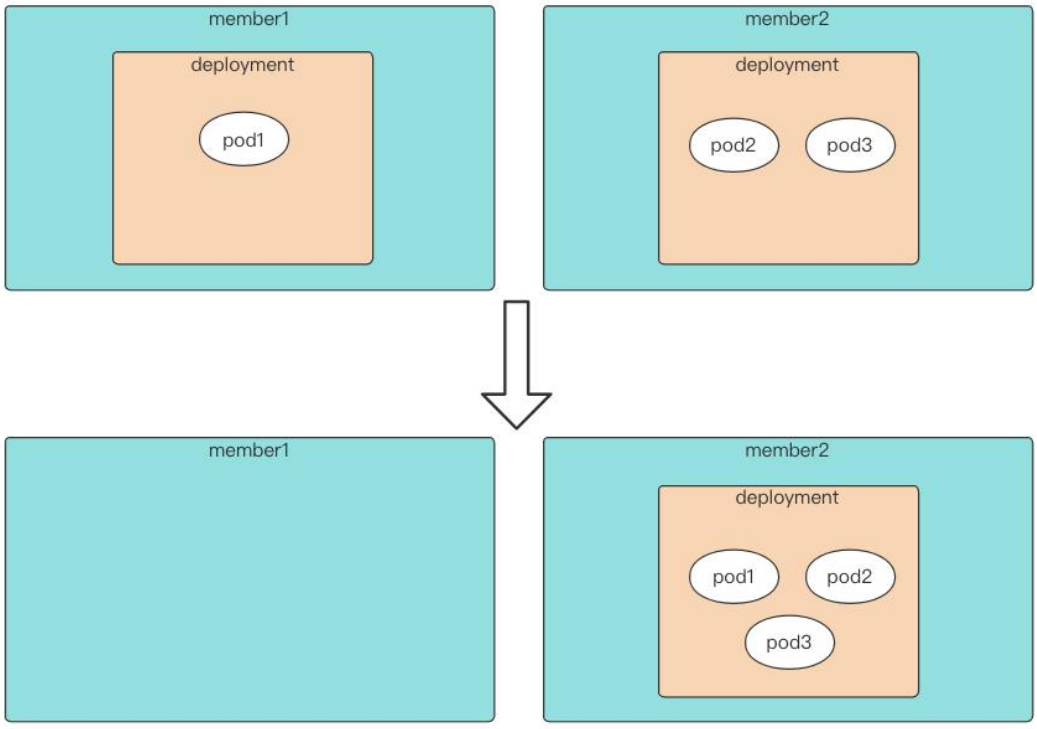
- 管理员在 Karmada 控制平面上部署离线应用，并将 Pod 实例分配到多个集群。 当集群发生故障时，管理员希望 Karmada 将故障集群中的 Pod 实例迁移到满足条件的其他集群中。
- 普通用户通过 Karmada 控制平面在集群中部署在线应用。应用包括数据库实例、服务器实例和配置文件。 此时，集群故障。客户希望将整个应用程序迁移到另一个合适的集群。在应用迁移过程中，确保业务不中断。
- 管理员升级集群后，集群中作为基础设施的容器网络和存储设备将发生变化。 管理员希望在升级集群前将集群中的应用迁移到其他合适的集群中。在迁移过程中，必须持续提供服务。

Karmada 故障恢复示例

Karmada 故障恢复支持两种方式：

- Duplicated（全量调度策略）。当满足 pp (propagationPolicy) 限制的未调度的候选集群数量不少于调度失败的集群数量时，将调度失败的集群重新调度到候选集群。
- Divided（副本拆分调度策略）。集群故障时，调度器和控制器配合尝试将故障集群副本迁移到其他运行正常的集群。

本文以 Divided 为例：



副本拆分策略

Divided

1. 下载 Karmada 官方 v1.4.2 sourece code 后，使用 hack/local-up-karmada.sh，启动本地的 Karmada。 启动后，自动纳管了三个工作集群，其中集群 member1 和 member2 使用 push 模式，member3 使用 pull 模式。

   export KUBECONFIG=$HOME/.kube/karmada.config
   kubectl --kubeconfig $HOME/.kube/karmada.config config use-context karmada-apiserver

   Switched to context "karmada-apiserver".

   kubectl get cluster

   NAME VERSION MODE READY AGE
   member1 v1.23.4 Push True 32m

```
member2     v1.23.4     Push     True      32m
member3     v1.23.4     Pull     True      31m
```

2. 在 Karmada 控制平面部署如下的应用配置，可以发现我们定义了一个副本数为 3 的 nginx 的应用，同时定义了一个传播策略。 传播策略中，使用 clusterNames 的方式指定了集群亲和性，需要调度到集群 member1 和 member2 上。 同时在副本调度策略中，采用副本拆分的方式进行调度，遵循 member1 权重为 1，member2 权重为 2 的静态权重方式进行调度。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
    app: nginx
  template:
    metadata:
    labels:
        app: nginx
    spec:
    containers:
    - image: nginx
        name: nginx
---
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
```

```yaml
metadata:
  name: nginx-propagation
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
    replicaScheduling:
      replicaDivisionPreference: Weighted
      replicaSchedulingType: Divided
      weightPreference:
        staticWeightList:
        - targetCluster:
            clusterNames:
              - member1
          weight: 1
        - targetCluster:
            clusterNames:
              - member2
          weight: 2
```

应用下发后，我们将会看到如下结果，集群 member1 上传播了一个副本数为 1 的 deployment，集群 member2 上传播了一个副本数为 2 的 deployment。 合计三个副本，符合我们调度预期。查看控制平面资源信息，发现在控制平面的成员集群对应执行命名空间下，创建了对应的 work， 这里的 work 即是通过传播策略和覆盖策略作用后实际需要在成员集群上传播的资源对象的载体， 同时，资源 deployment 的 rb

(ResourceBinding) 中看到，deployment 调度到了集群 member1 和集群 member2。

```
kubectl create -f failover.yaml

deployment.apps/nginx created
propagationpolicy.policy.karmada.io/nginx-propagation created

kubectl get deploy,pp

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx         3/3     3            3           2m

NAME                                                  AGE
propagationpolicy.policy.karmada.io/nginx-propagation   119s

kubectl get work -A | grep nginx

karmada-es-member1   nginx-687f7fb96f                  True        20
m
karmada-es-member2   nginx-687f7fb96f                  True        20
m

kubectl get rb nginx-deployment -o yaml

...
spec:
clusters:
- name: member2
    replicas: 2
- name: member2
    replicas: 1
replicas: 3
resource:
    apiVersion: apps/v1
```

```
    kind: Deployment
    name: nginx
    namespace: default
    resourceVersion: "5776"
    uid: 530aa301-760a-48a7-ada0-fc3a2112564b
...
```

```
karmadactl get po
```

```
NAME                     CLUSTER   READY   STATUS    RESTARTS   AGE
nginx-85b98978db-d7q92   member2   1/1     Running   0          110s
nginx-85b98978db-xmbp9   member2   1/1     Running   0          110s
nginx-85b98978db-97xbx   member1   1/1     Running   0          110s
```

```
karmadactl get deploy
```

```
NAME    CLUSTER   READY   UP-TO-DATE   AVAILABLE   AGE     ADOPTION
nginx   member2   2/2     2            2           3m15s   Y
nginx   member1   1/1     1            1           3m15s   Y
```

3. 模拟集群发生故障，由于安装集群是使用 kind 启动的，那么我们直接暂停集群 member1 的容器， 模拟实际情况中，集群由于网络或者集群本身问题，从而在联邦中失联。

```
docker ps -a
```

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                       NAMES
8794507af450   kindest/node:v1.23.4   "/usr/local/bin/entr…"   52 minut
```

es ago Up 51 minutes 127.0.0.1:40000->6443/tcp member2-control-plane
cc57b0eb54fe kindest/node:v1.23.4 "/usr/local/bin/entr..." 52 minutes ago Up 51 minutes 127.0.0.1:35728->6443/tcp karmada-host-control-plane
5ac1815cd40e kindest/node:v1.23.4 "/usr/local/bin/entr..." 52 minutes ago Up 51 minutes 127.0.0.1:39837->6443/tcp member1-control-plane
f5e5f753dcb8 kindest/node:v1.23.4 "/usr/local/bin/entr..." 52 minutes ago Up 51 minutes 127.0.0.1:33529->6443/tcp member3-control-plane

docker stop member1-control-plane

member1-control-plane

暂停成功，我们再来看一下实际情况，可以发现，集群暂停成功后，控制平面的 resource template 状态依然是‘3/3’，符合整体的预期。 获取成员集群上的 deployment，集群 member1 网络不可达。集群 membe2 上获取到了副本数为 3 的 deployment，故障发生后集群 member2 新增了 1 个副本。 通过查看 Karmada 资源 rb (rersourcebinding) 和 pp(propagationPolicy) 后，可以发现，故障转移后，deployment 的资源绑定只调度到集群 member2。 但是有个问题，从 rb (rersourcebinding) 的配置中，我们可以看到，此时资源未调度到集群 member1，但是此时集群 member1 对应的执行命名空间下仍然存在对应的 work，这是为什么呢？不急，我们来继续进一步探究。

kubectl get cluster

NAME VERSION MODE READY AGE
member1 v1.23.4 Push False 43m
member2 v1.23.4 Push True 43m
member3 v1.23.4 Pull True 42m

kubectl get deploy,pp

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx         3/3     3            3           11m

NAME                                                  AGE
propagationpolicy.policy.karmada.io/nginx-propagation   11m
```

karmadactl get deploy

```
NAME    CLUSTER   READY   UP-TO-DATE   AVAILABLE   AGE   ADOPTION
nginx   member2   3/3     3            3           12m   Y
error: cluster(member1) is inaccessible, please check authorization or network
```

karmadactl get po

```
NAME                     CLUSTER   READY   STATUS    RESTARTS   AGE
nginx-85b98978db-8zj5k   member2   1/1     Running   0          3m18s
nginx-85b98978db-d7q92   member2   1/1     Running   0          12m
nginx-85b98978db-xmbp9   member2   1/1     Running   0          12m
error: cluster(member1) is inaccessible, please check authorization or network
```

kubectl get rb nginx-deployment -o yaml

```
...
spec:
clusters:
- name: member2
```

```
    replicas: 3
replicas: 3
resource:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
    namespace: default
    resourceVersion: "5776"
    uid: 530aa301-760a-48a7-ada0-fc3a2112564b
...
```

```
kubectl get work -A | grep nginx

karmada-es-member1     nginx-687f7fb96f                    True          30
m
karmada-es-member2     nginx-687f7fb96f                    True          30
m
```

将集群状态恢复，模拟集群被运维管理员修复的情况，此时副本的变化又将会如何呢？ 可以发现，当故障集群恢复后，集群 member2 上的副本数不变，同时，控制平面在集群 member1 对应执行命名空间下的 work 被删除，集群 member1 上的 deployment 被删除。

```
kubectl get cluster

NAME          VERSION     MODE     READY     AGE
member1     v1.23.4     Push     True       147m
member2     v1.23.4     Push     True       147m
member3     v1.23.4     Pull     True       146m

karmada(imagesloy,po

NAME                                CLUSTER     READY     STATUS       RESTA
RTS     AGE
pod/nginx-85b98978db-2p8hn     member2     1/1       Running     0
```

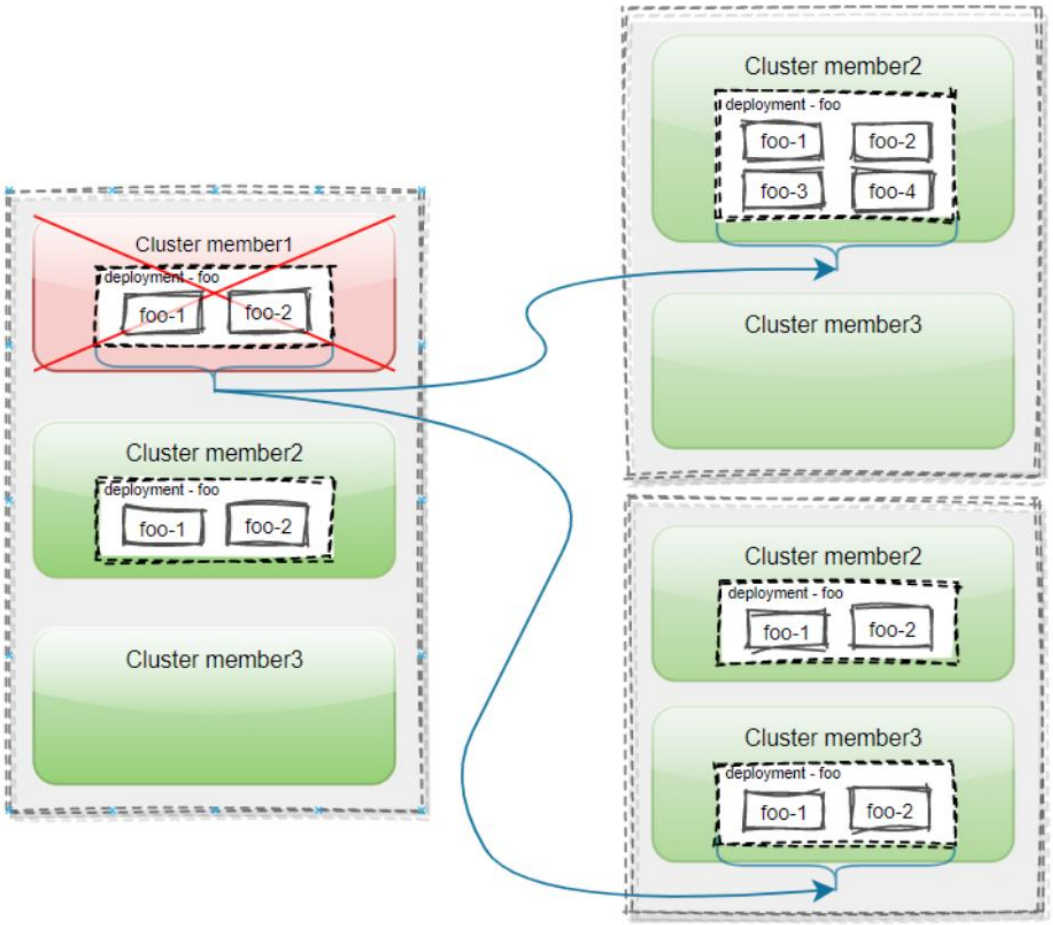
```
  73m
pod/nginx-85b98978db-7j8xs     member2     1/1       Running     0
 73m
pod/nginx-85b98978db-m897g     member2     1/1       Running     0
    70m

NAME                          CLUSTER     READY     UP-TO-DATE     AVAILA
BLE     AGE
deployment.apps/nginx     member2     3/3       3                3                7
3m
```

```
kubectl get work | grep nginx

No resources found in default namespace.
```

从以上的示例，我们可以发现，当集群发生故障时，Karmada 会自动调节其余成员集群上的副本， 从而来满足用户整体的副本期望，从而达到集群级别的故障恢复。当故障集群恢复后，转移的副本会保持不变， 原有调度集群上资源会被删除，资源不会调度回故障的集群。

Karmada 故障恢复实现原理

在多集群场景下，用户应用可能部署在多个集群中，以提高业务的高可用性。 在 Karmada 中，当集群发生故障或用户不想继续在集群上运行应用时，集群状态将被标记为不可用，并添加两个污点。 (images(images 检测到集群故障后，控制器将从故障集群中移除应用。然后，被移除的应用将被调度到满足要求的其他集群。 这样可以实现故障转移，保证用户业务的高可用性和连续性。



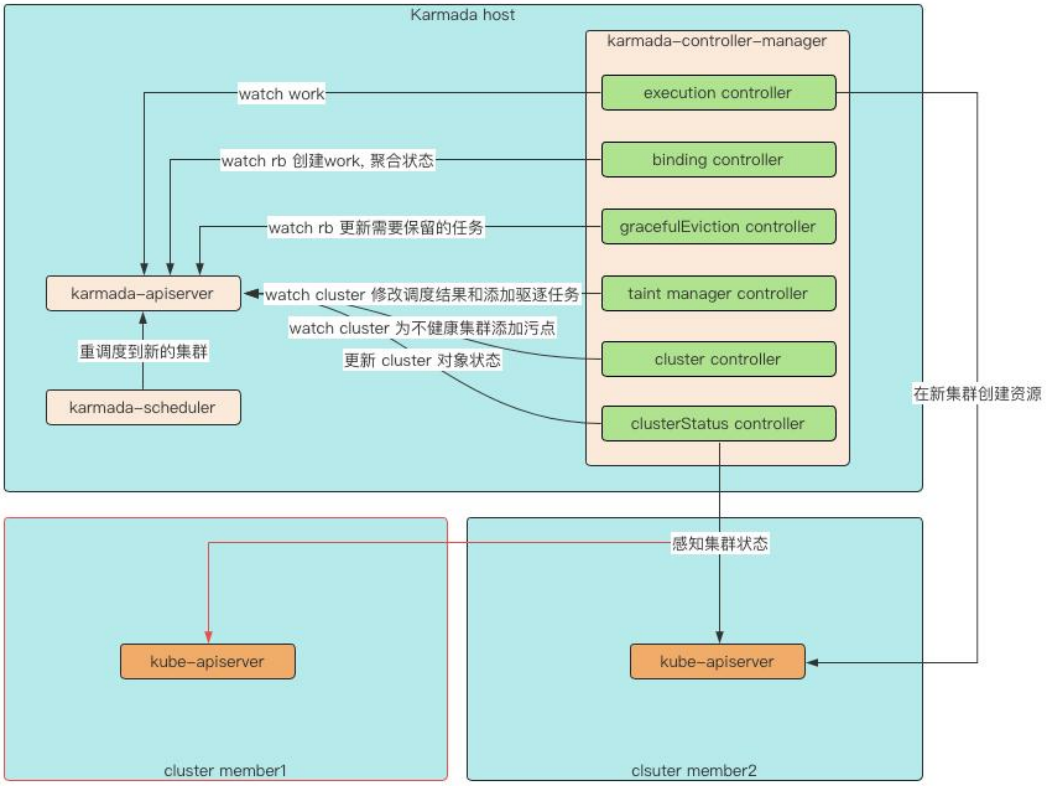
failover

图源：Karmada 官方文档 https://karmada.io

如上图，用户在 Karmada 中加入了三个集群: member1、member2 和 member3。 部署一个名为 foo 的 deployment，它有两个副本，在 Karmada 控制平面上创建。 通过使用 pp (PropagationPolicy) 将 deployement 传播到集群 member1 和 member2。

当集群 member1 发生故障时，如果是副本拆分策略，集群 member1 上的 Pod 将被清除并迁移到满足要求的集群 member2。 如果是全量策略，集群 member1 上的 deployment 将会被移除并迁移到满足要求的集群 member3。

Karmada 实现故障转移，主要由 karmada-controller-manager 中的 6 个控制器和 karmada-scheduler 参与完成。



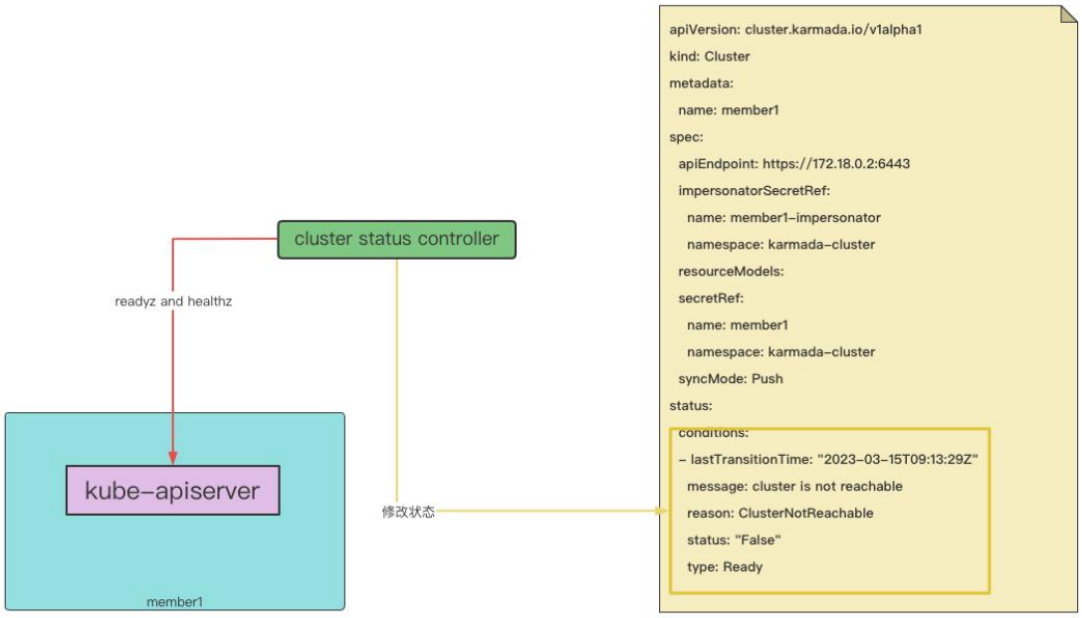
failover

1. clusterStatus controller：感知集群状态，并将集群状态写入到控制平面的 cluster 资源对象的状态中。
2. cluster controller：在控制平面创建执行命名空间（这里的执行命名空间指的是 Karmada 会在控制平面为每一个成员集群创建一个具有特殊命名规则的命名空间，该命名空间用于存放 work ），根据 conditions 判断是否需要为集群打上污点。
3. taint-manager controller：根据集群污点，传播策略中容忍的集群污点，对比计算，确定是否要将集群上的资源驱逐。
4. karmada-scheduler：根据传播策略和集群情况，为资源选择最佳的调度集群。

5. gracefulEviction controller：保证在新的集群上的资源状态健康后，再移除驱逐集群上的资源对象。
6. binding controller：根据调度结果，应用传播策略中的规则，在控制平面集群对应执行命名空间下创建 work， 聚合 work 状态，更新 rb(ResourceBinding) 和 resource template 状态。
7. execution controller：根据成员集群对应执行命名空间下的 work，在成员集群中创建资源。

接下来看一下每个控制器和调度器，在集群故障恢复的场景中，分别承担了什么样的能力。

clusterStatus controller



clusterStatus controller

clusterStatus controller 主要用于同步集群实际状态。当集群无法访问时，clusterStatus controller 就会感知到集群状态处于非在线状态，此时便会更新控制平面的 cluster 资源对象的状态，修改 conditions 中 type 为 Ready 的 condition 状态为‘False’， 以下为主要实现代码。

首先，通过访问成员集群 kube-apiserver 的‘/readyz’和‘/healthz’，获取到成员集群的在线状态和健康状态。

```
func getClusterHealthStatus(clusterClient *util.ClusterClient) (online, healthy bool)
 {
    healthStatus, err := healthEndpointCheck(clusterClient.KubeClient, "/readyz")
    if err != nil && healthStatus == http.StatusNotFound {
        // do health check with healthz endpoint if the readyz endpoint is not i
nstalled in member cluster
        healthStatus, err = healthEndpointCheck(clusterClient.KubeClient, "/healt
hz")
    }

    if err != nil {
        klog.Errorf("Failed to do cluster health check for cluster %v, err is : %v
 ", clusterClient.ClusterName, err)
        return false, false
    }

    if healthStatus != http.StatusOK {
        klog.Infof("Member cluster %v isn't healthy", clusterClient.ClusterName)
        return true, false
    }

    return true, true
}
```

其次，根据在线状态和健康状态初始化 conditions。

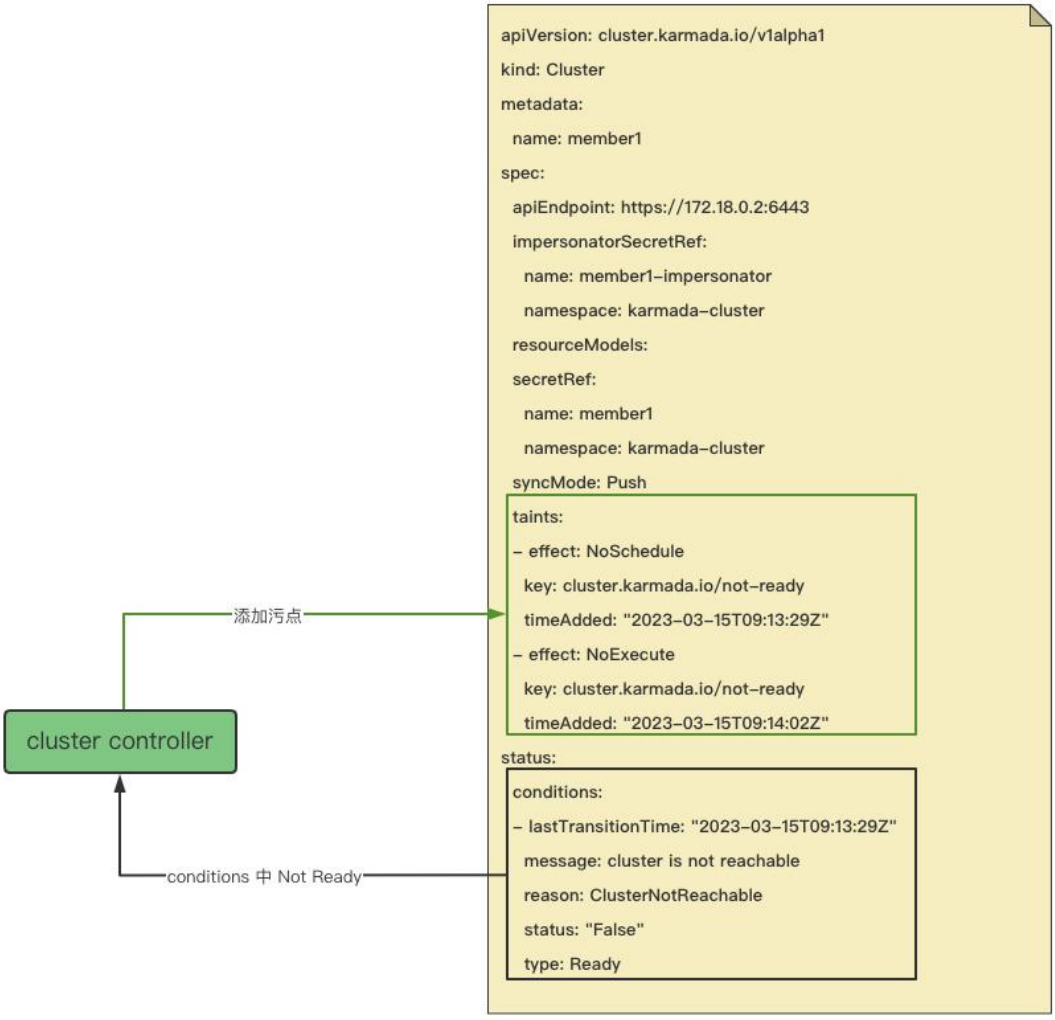
```
func generateReadyCondition(imagesalthy bool) metav1.Condition {
    if !online {
        return util.NewCondition(clusterv1alpha1.ClusterConditionReady, clusterNot
ReachableReason, clusterNotReachableMsg, metav1.ConditionFalse)
```

```
    }
    if !healthy {
        return util.NewCondition(clusterv1alpha1.ClusterConditionReady, clusterNot
Ready, clusterUnhealthy, metav1.ConditionFalse)
    }

    return util.NewCondition(clusterv1alpha1.ClusterConditionReady, clusterReady,
clusterHealthy, metav1.ConditionTrue)
}
```

最后，如果集群已经下线，此时初始化的 condition 状态不为‘True’， 则认为需要更新 cluster 资源对象的 condiitions。 conditions 中 type 为 Ready 的 condition 状态将会被修改为‘False’，然后返回。

```
online, healthy := getClusterHealthStatus(clusterClient)
observedReadyCondition := generateReadyCondition(online, healthy)
readyCondition := c.clusterConditionCache.thresholdAdjustedReadyCondition(clust
er, &observedReadyCondition)

// cluster is offline after retry timeout, update cluster status immediately and r
eturn.
if !online && readyCondition.Status != metav1.ConditionTrue {
    klog.V(2).Infof("Cluster(%s) still offline after %s, ensuring offline is set.",
        cluster.Name, c.ClusterFailureThreshold.Duration)
    setTransitionTime(cluster.Status.Conditions, readyCondition)
    meta.SetStatusCondition(¤tClusterStatus.Conditions, *readyCondition)
    return c.updateStatusIfNeeded(cluster, currentClusterStatus)
}
```

cluster controller



clusterStatus controller

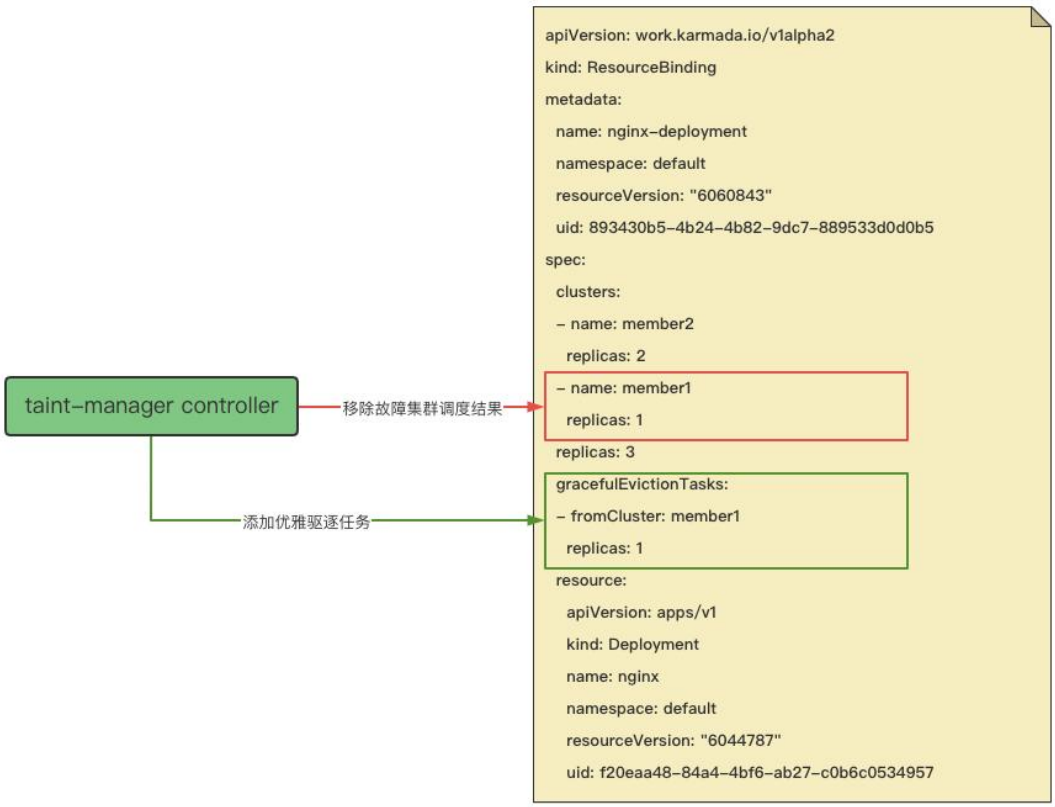
cluster conntroller 会根据集群当前状态下的 conditions，判断是否需要为集群打上不可调度和不可执行的污点。 以下代码为核心实现逻辑，当 conditions 中 type 为 Ready 的 condition 状态为‘False’时， 执行 UpdateClusterControllerTaint 函数添加 effect 为 NoSchedule 和 NoExecute 的污点。

```
func (c *Controller) taintClusterByCondition(ctx context.Context, cluster *clusterv
1alpha1.Cluster) error {
    currentReadyCondition := meta.FindStatusCondition(cluster.Status.Conditions,
```

```
clusterv1alpha1.ClusterConditionReady)
    var err error
    if currentReadyCondition != nil {
        switch currentReadyCondition.Status {
        case metav1.ConditionFalse:
            // Add NotReadyTaintTemplateForSched taint immediately.
            if err = utilhelper.UpdateClusterControllerTaint(ctx, c.Client, []*corev1.
Taint{NotReadyTaintTemplateForSched}, []*corev1.Taint{UnreachableTaintTemplate
ForSched}, cluster); err != nil {
                klog.ErrorS(err, "Failed to instantly update UnreachableTaintForS
ched to NotReadyTaintForSched, will try again in the next cycle.", "cluster", clus
ter.Name)
            }
        case metav1.ConditionUnknown:
            // Add UnreachableTaintTemplateForSched taint immediately.
            if err = utilhelper.UpdateClusterControllerTaint(ctx, c.Client, []*corev1.
Taint{UnreachableTaintTemplateForSched}, []*corev1.Taint{NotReadyTaintTemplate
ForSched}, cluster); err != nil {
                klog.ErrorS(err, "Failed to instantly swap NotReadyTaintForSched
 to UnreachableTaintForSched, will try again in the next cycle.", "cluster", cluste
r.Name)
            }
        case metav1.ConditionTrue:
            if err = utilhelper.UpdateClusterControllerTaint(ctx, c.Client, nil, []*co
rev1.Taint{NotReadyTaintTemplateForSched, UnreachableTaintTemplateForSched},
cluster); err != nil {
                klog.ErrorS(err, "Failed to remove schedule taints from cluster,
will retry in next iteration.", "cluster", cluster.Name)
            }
        }
    } else {
        // Add NotReadyTaintTemplateForSched taint immediately.
```

```
            if err = utilhelper.UpdateClusterControllerTaint(ctx, c.Client, []*corev1.Taint{NotReadyTaintTemplateForSched}, nil, cluster); err != nil {
                klog.ErrorS(err, "Failed to add a NotReady taint to the newly added cluster, will try again in the next cycle.", "cluster", cluster.Name)
            }
        }
        return err
}
```

taint-manager controller



clusterStatus controller

taint-manager controller 控制器随 cluster controller 同步启动，可参数配置是否启动，默认启动。 它感知 cluster 资源对象的变更事件，当感知到集群拥有 effect 为 NoExecute 的污点时， 会获取到该集群上的所有命名空间级别和集群级别的 rb (ResourceBinding)，然后放入对应的驱逐处理队列。 驱逐处理消费者

（worker）会判断获取到 rb(ResourceBinding) 对应的 pp (PropagationPolicy)，看 pp (PropagationPolicy) 中是否存在集群污点容忍，如果 pp (PropagationPolicy) 中的集群污点容忍和集群污点匹配， 那么直接跳过，认为该集群上的资源不需要驱逐。否则认为需要驱逐。如果需要驱逐，那么此时会判断是否开启了优雅驱逐， 优雅驱逐默认开启，为 rb (ResourceBinding) 对象写入优雅驱逐任务，即 rb.spec.gracefulEvictionTasks 中增加一条优雅驱逐任务。

```
needEviction, tolerationTime, err := tc.needEviction(cluster, binding.Annotations)
if err != nil {
    klog.ErrorS(err, "Failed to check if binding needs eviction", "binding", fedKey.ClusterWideKey.NamespaceKey())
    return err
}

// Case 1: Need eviction now.
// Case 2: Need eviction after toleration time. If time is up, do eviction right now.
// Case 3: Tolerate forever, we do nothing.
if needEviction || tolerationTime == 0 {
    // update final result to evict the target cluster
    if features.FeatureGate.Enabled(features.GracefulEviction) {
        binding.Spec.GracefulEvictCluster(cluster, workv1alpha2.EvictionProducerTaintManager, workv1alpha2.EvictionReasonTaintUntolerated, "")
    } else {
        binding.Spec.RemoveCluster(cluster)
    }
    if err = tc.Update(context.TODO(), binding); err != nil {
        helper.EmitClusterEvictionEventForResourceBinding(binding, cluster, tc.EventRecorder, err)
        klog.ErrorS(err, "Failed to update binding", "binding", klog.KObj(binding))
        return err
```

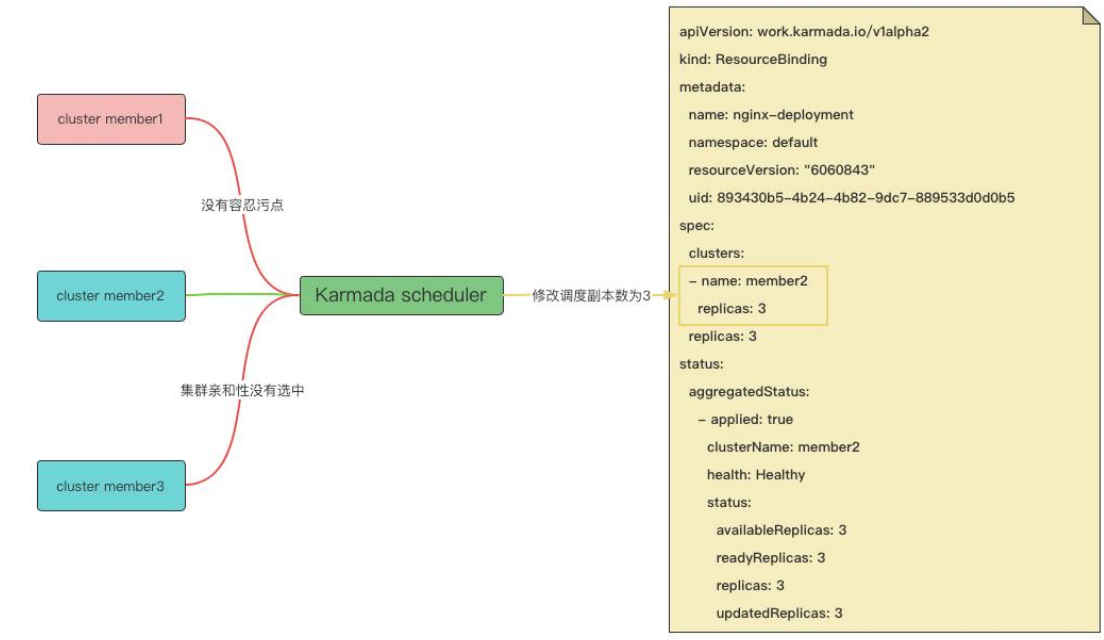
```
        }
        if !features.FeatureGate.Enabled(features.GracefulEviction) {
            helper.EmitClusterEvictionEventForResourceBinding(binding, cluster, tc.Ev
entRecorder, nil)
        }
    } else if tolerationTime > 0 {
        tc.bindingEvictionWorker.AddAfter(fedKey, tolerationTime)
    }
```

可以发现，写入驱逐任务时，会将优雅驱逐任务对应的集群，从 rb.spec.clusters 中移除，也就是会修改调度结果 （这里需要特别说明一下，调度结果就是 Karmada 调度器根据传播策略和集群情况为资源选择的调度分发集群， 调度结果会记录在 rb (ResourceBinding) 的 spec.clusters 属性中） 。 也就是说由于集群故障，会触发调度器重新调度，资源应该从故障的集群上驱逐，在新的集群上创建。

```
// This function no-(images cluster does not exist.
func (s *ResourceBindingSpec) GracefulEvictCluster(name, producer, reason, mes
sage string) {
    // find the cluster index
    var i int
    for i = 0; i < len(s.Clusters); i++ {
        if s.Clusters[i].Name == name {
            break
        }
    }
    // not found, do nothing
    if i >= len(s.Clusters) {
        return
    }

    // build eviction task
```

```
    evictingCluster := s.Clusters[i].DeepCopy()
    evictionTask := GracefulEvictionTask{
        FromCluster: evictingCluster.Name,
        Reason:      reason,
        Message:     message,
        Producer:    producer,
    }
    if evictingCluster.Replicas > 0 {
        evictionTask.Replicas = &evictingCluster.Replicas
    }

    s.GracefulEvictionTasks = append(s.GracefulEvictionTasks, evictionTask)
    s.Clusters = append(s.Clusters[:i], s.Clusters[i+1:]...)
}
```

Karmada scheduler



Karmada scheduler

Karmada scheduler，如果调度的目标副本和期望的目标副本不等，也就是当rb.spec.clusters（资源调度集群结果）下所有的副本之和不等于

rb.spec.replicas（资源期望副本）时，调度器认为这是一次扩容或者缩容操作，从而触发重调度操作。 而在 taint-manager controller 中，由于集群故障，资源从故障集群上驱逐了，taint-manger controller 修改了调度结果，移除了 rb.spec.clusters 下的一个集群，导致 rb.spec.clusters 下所有的副本之和小于 rb.spec.replicas，所以这是一次扩容操作。

这里是否需要重新调度的判断，核心逻辑为‘replicasSum != bindingSpec.Replicas’。 移除了调度结果中的集群 member1 的调度结果，所以此时 replicasSum 值为 2， 但是 bindingSpec.Replicas 值为 3，两者不等所以函数返回值为真。

```
// IsBindingReplicasChanged will check if the sum of replicas is different from the replicas of object
func IsBindingReplicasChanged(bindingSpec *workv1alpha2.ResourceBindingSpec, strategy *policyv1alpha1.ReplicaSchedulingStrategy) bool {
    if strategy == nil {
        return false
    }
    if strategy.ReplicaSchedulingType == policyv1alpha1.ReplicaSchedulingTypeDuplicated {
        for _, targetCluster := range bindingSpec.Clusters {
            if targetCluster.Replicas != bindingSpec.Replicas {
                return true
            }
        }
        return false
    }
    if strategy.ReplicaSchedulingType == policyv1alpha1.ReplicaSchedulingTypeDivided {
        replicasSum := GetSumOfReplicas(bindingSpec.Clusters)
        return replicasSum != bindingSpec.Replicas
    }
```

```
    return false
}
```

由于函数 IsBindingReplicasChanged 返回为真，并且设置了 ReplicaScheduling，则会执行重调度操作。

```
if rb.Spec.Placement.ReplicaScheduling != nil && util.IsBindingReplicasChanged(&rb.Spec, rb.Spec.Placement.ReplicaScheduling) {
    // binding replicas changed, need reschedule
    klog.Infof("Reschedule ResourceBinding(%s/%s) as replicas scaled down or scaled up", namespace, name)
    err = s.scheduleResourceBinding(rb)
    metrics.BindingSchedule(string(ScaleSchedule), utilmetrics.DurationInSeconds(start), err)
    return err
}
```

在重调度逻辑中，由于集群 member1 存在污点，所以会被调度插件过滤掉，集群 member3 没有被集群亲和性选中。结果，只有 member2 满足要求。

```
// Filter checks if the given tolerations in placement tolerate cluster's taints.
func (p *TaintToleration) Filter(
    _ context.Context,
    bindingSpec *workv1alpha2.ResourceBindingSpec,
    _ *workv1alpha2.ResourceBindingStatus,
    cluster *clusterv1alpha1.Cluster,
) *framework.Result {
    // skip the filter if the cluster is already in the list of scheduling results,
    // if the workload referencing by the binding can't tolerate the taint,
    // the taint-manager will evict it after a graceful period.
    if bindingSpec.TargetContains(cluster.Name) {
        return framework.NewResult(framework.Success)
    }
```

```
    filterPredicate := func(t *corev1.Taint) bool {
        return t.Effect == corev1.TaintEffectNoSchedule || t.Effect == corev1.Tain
tEffectNoExecute
    }

    taint, isUntolerated := v1helper.FindMatchingUntoleratedTaint(cluster.Spec.Tain
ts, bindingSpec.Placement.ClusterTolerations, filterPredicate)
    if !isUntolerated {
        return framework.NewResult(framework.Success)
    }

    return framework.NewResult(framework.Unschedulable, fmt.Sprintf("cluster(s)
had untolerated taint {%s}", taint.ToString()))
}
```

在计算调度副本时，由于设置了静态权重，所以要以静态权重方式 （静态权重方式是指，根据管理员设定的比例对副本进行拆分，举例，如果期望副本总数为 9， 集群 member1 静态权重为 1，集群 member2 静态权重为 2，那么集群 member1 上的副本计算方式是 9*1/3=3， 集群 member2 上的副本计算方式为 9*2/3=6）进行副本拆分。但是，因为只有一个集群满足要求， 所以在设置集群调度副本时，集群 member2 的副本调度结果为 3，所以调度器会将 rb.spec.clusters （资源调度集群结果）下的集群 member2 上的副本调度结果由 2 修改为 3。

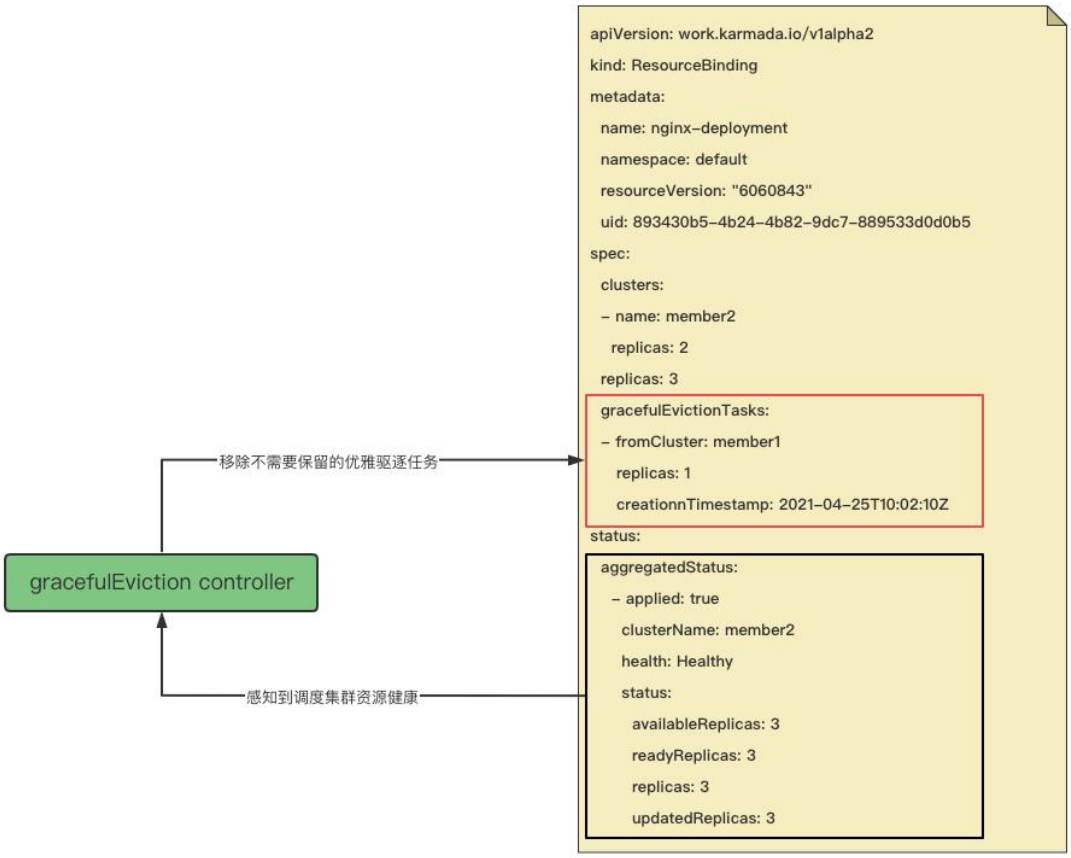
```
// TakeByWeight divide replicas by a weight list and merge the result into previ
ous result.
func (a *Dispenser) TakeByWeight(w ClusterWeightInfoList) {
    if a.Done() {
        return
    }
    sum := w.GetWeightSum()
    if sum == 0 {
```

```
            return
        }

        sort.Sort(w)

        result := make([]workv1alpha2.TargetCluster, 0, w.Len())
        remain := a.NumReplicas
        for _, info := range w {
            replicas := int32(info.Weight * int64(a.NumReplicas) / sum)
            result = append(result, workv1alpha2.TargetCluster{
                Name:     info.ClusterName,
                Replicas: replicas,
            })
            remain -= replicas
        }
        // TODO(Garrybest): take rest replicas by fraction part
        for i := range result {
            if remain == 0 {(image(images
                break
            }
            result[i].Replicas++
            remain--
        }

        a.NumReplicas = remain
        a.Result = util.MergeTargetClusters(a.Result, result)
    }
```

以下为故障恢复时，截取调度器部分日志，可以清晰看到整个集群筛选和副本计算的过程。 集群 member1 由于没有容忍污点被过滤掉，集群 member3 没有被集群亲和性选中被过滤掉，最终调度结果为集群 member2，调度副本为 3。

I0217 06:46:53.057843 1 scheduler.go:390] Reschedule ResourceBinding(default/nginx-deployment) as replicas scaled down or scaled up
I0217 06:46:53.057888 1 scheduler.go:473] "Begin scheduling resource binding" resourceBinding="default/nginx-deployment"
I0217 06:46:53.083460 1 generic_scheduler.go:119] cluster "member3" is not fit, reason: cluster(s) didn't match the placement cluster affinity constraint
I0217 06:46:53.083524 1 generic_scheduler.go:119] cluster "member1" is not fit, reason: cluster(s) had untolerated taint {cluster.karmada.io/not-ready: }
I0217 06:46:53.083542 1 generic_scheduler.go:73] feasible clusters found: [member2]
I0217 06:46:53.083567 1 generic_scheduler.go:146] Plugin ClusterLocality scores on default/nginx => [{member2 100}]
I0217 06:46:53.083590 1 generic_scheduler.go:146] Plugin ClusterAffinity scores on default/nginx => [{member2 0}]
I0217 06:46:53.083601 1 generic_scheduler.go:79] feasible clusters scores: [{member2 100}]
I0217 06:46:53.085488 1 util.go:72] Target cluster: [{member2 99}]
I0217 06:46:53.085532 1 select_clusters.go:19] select all clusters
I0217 06:46:53.085548 1 generic_scheduler.go:85] selected clusters: [member2]
I0217 06:46:53.085571 1 scheduler.go:489] ResourceBinding default/nginx-deployment scheduled to clusters [{member2 3}]
I0217 06:46:53.151403 1 scheduler.go:491] "End scheduling resource binding" resourceBinding="default/nginx-deployment"

gracefulEviction controller



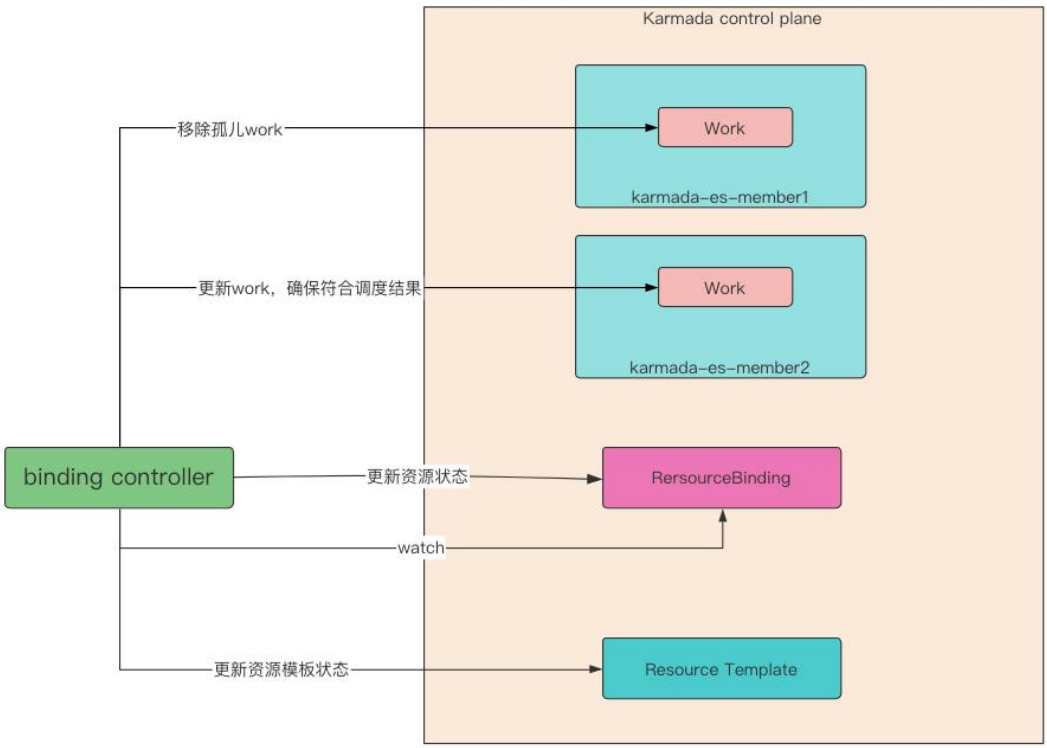
gracefulEviction controller

gracefulEviction controller 感知带有 rb.spec.gracefulEvictionTasks 的 rb(ResourceBinding) 资源的变更， 即只关心需要优雅驱逐的 rb (ResourceBinding)。然后根据调度目标集群，获取到调度结果集群的资源健康状态， 如果调度目标集群所有资源健康，则认为不需要保留优雅驱逐任务，从而移除驱逐集群任务，也就是将集群 member1 上的优雅驱逐任务移除。从中我们可以发现，优雅驱逐实现的逻辑是，保证了新的副本在符合要求的集群上 Ready 后， 再移除优雅驱逐任务，也就是才能删除调度到故障集群的旧资源。

```
// assessEvictionTasks assesses each task according to graceful eviction rules and
// returns the tasks that should be kept.
func assessEvictionTasks(bindingSpec workv1alpha2.ResourceBindingSpec,
```

```
    obse(images[]workv1alpha2.AggregatedStatusItem,
    timeout time.Duration,
    now metav1.Time,
) ([]workv1alpha2.GracefulEvictionTask, []string) {
    var keptTasks []workv1alpha2.GracefulEvictionTask
    var evictedClusters []string

    for _, task := range bindingSpec.GracefulEvictionTasks {
        // set creation timestamp for new task
        if task.CreationTimestamp.IsZero() {
            task.CreationTimestamp = now
            keptTasks = append(keptTasks, task)
            continue
        }

        // assess task according to observed status
        kt := assessSingleTask(task, assessmentOption{
            scheduleResult: bindingSpec.Clusters,
            timeout:        timeout,
            observedStatus: observedStatus,
        })
        if kt != nil {
            keptTasks = append(keptTasks, *kt)
        } else {
            evictedClusters = append(evictedClusters, task.FromCluster)
        }
    }
    return keptTasks, evictedClusters
}
```

binding controller



Image

当 rb (ResourceBinding) 资源发生变更时，会被 binding controller 所感知。首先会移除孤儿 work（work 用于存放经过传播策略规则后，在成员集群上需要真正创建的资源对象模板。 孤儿 work 是指，带有 rb （ReesourceBinding）资源对应标签的，但是随着 rb 的变更，work 已不是 rb 所期望的 work）。 除了 rb(ResourceBinding) 中三种属性中包含的集群，其他集群对应的执行命名空间下的 work 都会认为是孤儿 work： 第一种是 rb.spec.clusters，资源调度集群结果；第二种是 rb.spec.requiredBy，依赖应用的资源列表集群； 第三种是 rb.spec.gracefulEvictionTasks，优雅驱逐任务中的集群。在 gracefulEviction controller 中， 我们知道，需要新调度集群资源 Ready 后，任务才能被删除。这里就很巧妙， 这样就能保证新启动的副本在符合要求的集群上 Ready 之前，故障集群 work 永远不会被移除。

在示例中，可以发现当故障转移发生后，故障集群对应执行命名空间下仍然存在 work，理论上， 故障集群下 work 已经被判定为孤儿，为什么没有删除成功呢。进一步探究会发现，集群 member1 命名空间下的 work 的 DeletionTimestamp 不为空。没删除的原因是 work 存在 finalizer， 该 finalizer 是 binding controller 创建 work 时添加的。这里不能直接移除 work 的原因是， 需要保证在成员集群上的资源被移除后，才能移除 work。否则直接移除 work，会导致集群 member1 恢复后， 集群 member1 上的资源被保留，造成资源残留。所以，需要等待故障集群恢复后，execution controller 移除 work 对应在成员集群上的资源成功后，才能移除 finalizer，这样 work 就会被删除。

kubectl -n karmada-es-member1 get work nginx-687f7fb96f -oyaml

```
...
apiVersion: work.karmada.io/v1alpha1
kind: Work
metadata:
  annotations:
    resourcebinding.karmada.io/name: nginx-deployment
    resourcebinding.karmada.io/namespace: default
  creationTimestamp: "2023-02-17T08:53:08Z"
  deletionGracePeriodSeconds: 0
  deletionTimestamp: "2023-02-17T08:55:39Z"
  finalizers:
  - karmada.io/execution-controller
  generation: 2
  labels:
    resourcebinding.karmada.io/key: 687f7fb96f
  name: nginx-687f7fb96f
  namespace: karmada-es-member1
  resourceVersion: "28134"
  uid: d215a03c-522d-4a89-9c5d-e87e27338a03
...
```

以下为 binding controller 的核心代码，核心逻辑是:

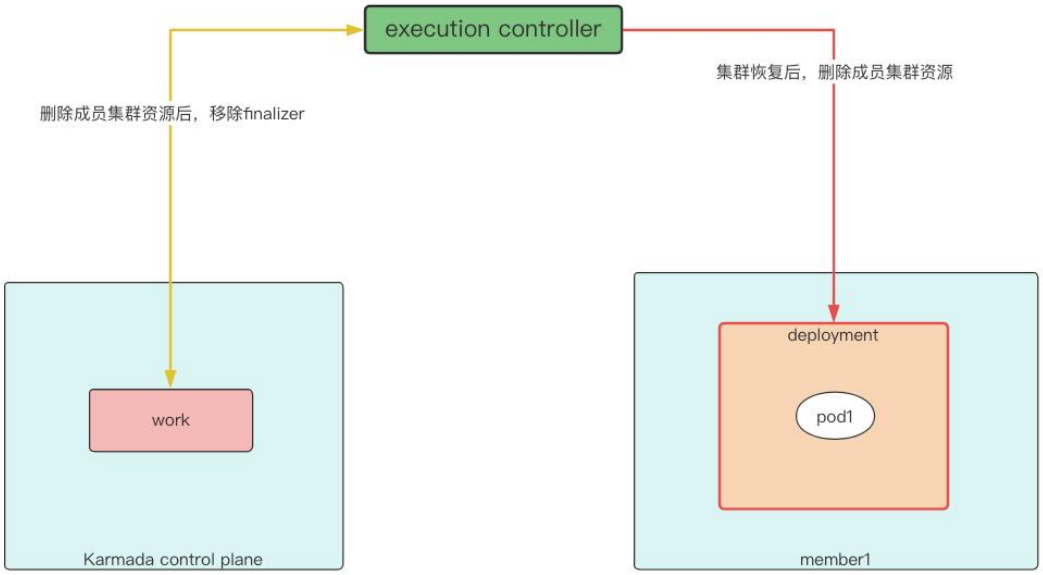
- 移除孤儿 work；
- 确保 rb (ResourceBinding) 期望 work 符合预期；
- 聚合 work 的状态，记录到 rb (ResourceBinding) 的状态中；
- 根据 rb (ResourceBinding) 的聚合状态，更新 resource template 状态， resource template 是指存放在 Karmada 控制平面上的资源模板对象。

```
// syncBinding will sync resourceBinding to Works.
func (c *ResourceBindingController) syncBinding(binding *workv1alpha2.ResourceBinding) (controllerruntime.Result, error) {
    if err := c.removeOrphanWorks(binding); err != nil {
        return controllerruntime.Result{Requeue: true}, err
    }

    workload, err := helper.FetchWorkload(c.DynamicClient, c.InformerManager, c.RESTMapper, binding.Spec.Resource)
    if err != nil {
        if apierrors.IsNotFound(err) {
            // It might happen when the resource template has been removed but the garbage collector hasn't removed
            // the ResourceBinding which dependent on resource template.
            // So, just return without retry(requeue) would save unnecessary loop.
            return controllerruntime.Result{}, nil
        }
        klog.Errorf("Failed to fetch workload for resourceBinding(%s/%s). Error: %v.",
            binding.GetNamespace(), binding.GetName(), err)
        return controllerruntime.Result{Requeue: true}, err
    }
```

```
    var errs []error
    start := time.Now()
    err = ensureWork(c.(imagesesourceInterpreter, workload, c.OverrideManager,
binding, apiextensionsv1.NamespaceScoped)
    metrics.ObserveSyncWorkLatency(binding.ObjectMeta, err, start)
    if err != nil {
        klog.Errorf("Failed to transform resourceBinding(%s/%s) to works. Error:
 %v.",
            binding.GetNamespace(), binding.GetName(), err)
        c.EventRecorder.Event(binding, corev1.EventTypeWarning, events.EventRea
sonSyncWorkFailed, err.Error())
        c.EventRecorder.Event(workload, corev1.EventTypeWarning, events.EventR
easonSyncWorkFailed, err.Error())
        errs = append(errs, err)
    } else {
        msg := fmt.Sprintf("Sync work of resourceBinding(%s/%s) successful.",
binding.Namespace, binding.Name)
        klog.V(4).Infof(msg)
        c.EventRecorder.Event(binding, corev1.EventTypeNormal, events.EventReas
onSyncWorkSucceed, msg)
        c.EventRecorder.Event(workload, corev1.EventTypeNormal, events.EventRe
asonSyncWorkSucceed, msg)
    }
    if err = helper.AggregateResourceBindingWorkStatus(c.Client, binding, worklo
ad, c.EventRecorder); err != nil {
        klog.Errorf("Failed to aggregate workStatuses to resourceBinding(%s/%
s). Error: %v.",
            binding.GetNamespace(), binding.GetName(), err)
        errs = append(errs, err)
    }
    if len(errs) > 0 {
        return controllerruntime.Result{Requeue: true}, errors.NewAggregate(errs)
```

```
    }

    err = c.updateResourceStatus(binding)
    if err != nil {
        return controllerruntime.Result{Requeue: true}, err
    }

    return controllerruntime.Result{}, nil
}
```

execution controller



execution controller

execution controller 的能力是 **将成员集群对应执行命名空间的 work 中存放的实际需要下发的资源在对应成员集群创建、更新或删除。** 所以，它的执行逻辑是和 work 的状态和集群的状态是息息相关的。如果 work 的 DeletionTimestamp 不为空，也就是 work 已经被删除， 则会继续判断集群状态，如果集群状态为 Ready，则会尝试删除成员集群上的资源。示例中，当集群发生故障后，资源转移到集群 member2 上所有副本都 Ready ，此时 gracefulEviction controller 会

移除集群 member1 优雅驱逐任务，binding controller 认为 member1 上的 work 是孤儿，所以会删除 work。此时集群 member1 执行命名空间下的 work DeletionTimestamp 不为空， 同时集群 Not Ready，所以会进入 cluster.DeletionTimestamp.IsZero() 的判断分支，从而重新入队列，work finalizer 不会被移除，所以示例中 work 一直存在。当集群恢复后，则会进入 util.IsClusterReady(&cluster.Status) 分支， 此时会尝试删除成员集群上 work 对应的资源，删除成功后，执行 removeFinalizer，将 work 上的 finalizer 移除， 这样 work 就会被删除。所以示例中集群恢复后，集群 member1 执行命名空间下的 work 和成员集群上的 deployment 都被移除了。

```
if !work.DeletionTimestamp.IsZero() {
    // Abort deleting workload if cluster is unready when unjoining cluster, othe
rwise the unjoin process will be failed.
    if util.IsClusterReady(&cluster.Status) {
        err := c.tryDeleteWorkload(clusterName, work)
        if err != nil {
            klog.Errorf("Failed to delete work %v, namespace is %v, err is %v",
 work.Name, work.Namespace, err)
            return controllerruntime.Result{Requeue: true}, err
        }
    } else if cluster.DeletionTimestamp.IsZero() { // cluster is unready, but not t
erminating
        return controllerruntime.Result{Requeue: true}, fmt.Errorf("cluster(%s) not
 ready", cluster.Name)
    }

    return c.removeFinalizer(work)
}
```

以上就是 Karmada 实现故障恢复的原理，从中可以看出，它是**由多个控制器和调度器协同完成的** 。 如果其中一个控制器或调度器出问题，那么都会影响故障恢复的功能。笔者曾经提过一个故障转移不符合预期的 issue 2769， 最终发现

是由于资源状态计算错误引起的，详细的修改见 pull request 2928。这就是由于在故障转移过程中， 开启了优雅驱逐，优雅驱逐会获取调度目标集群上的资源健康状态，计算是否都是 Healthy，如果都是 Healthy 则会移除优雅驱逐任务，但是由于状态计算错误，导致实际转移的目标集群上资源都是 Healthy，但是优雅驱逐控制器计算为 Healthy，所以导致优雅驱逐任务一直保留，从而出现 issue 中问题。再如 issue 2989 中，提到 work 状态仍然为 Healthy，这是由于成员集群状态为 Not Ready，从而导致 execution controller 在更新 work 状态前就重新入队列， 所以 work 状态中仍然是正确的，但是此时 work 的 DeletionTimestamp 不为空。

总结

当前版本的 Karmada 可以实现副本拆分策略和全量策略，但是在真正的业务场景中，例如有状态应用如何实现故障转移？ 转移后如何保证应用的网络和存储？这个有待进一步实际操作验证，需要根据实际情况具体问题具体分析。

同时，笔者在同客户交流过程中，有客户提到，集群故障恢复后，是否可以进行重调度，将副本调度回到恢复的集群。 目前，在参与社区会议中提到这一点，目前社区的方向是，将来可能会提供给客户选择是否重调度的能力。 也就是说，将来的 Karmada 版本**可能会支持用户自定义是否要调度回恢复的集群** 。 目前没有调度回原集群的原因是，考虑到重调度对应用稳定性的影响。 同时，目前在最新的社区会议中，有讨论 **增强 Failover 的 feature，会在传播策略中增加 Failover 更多配置能力** ， 具体的讨论这里就不展开了，我们一起期待。

综上所述，**基于 Karmada 应用可以实现集群级别的高可用。** 随时多云的发展，Karmada 被越来越多的人所使用，我们相信，Karmada 会在越来越多的场景中解决实际问题。 在笔者参与贡献的 CNCF 项目贡献中，目前，Karmada 是 issue 回复最快，代码合并请求处理最快的项目， 这得益于社区主导者良好的代码贡献管理机制。所以，基于增长的使用用户， 社区快速的响应，我相信，Karmada 将会带给我们越来越多惊艳的能力。

参考资料

- Karmada 源码
- Karmada 官方文档
- 云原生多云应用利器 — Karmada 总览篇
- 云原生多云应用利器 — Karmada 控制器
- 云原生多云应用利器 — Karmada 调度器

DCE 5.0 涉及多少开源项目

时常听闻客户、社区成员、贡献者、公司内部的售前、交付、项目组在询问“DCE 到底涉及了哪些开源项目？”

为此，我们做了一个简单的汇总。下文列出了截止到 2023-02-14，DCE 5.0 社区版所涉及的开源项目。

!!! info

集成和调用的所有相关项目将跟随社区版本持续演进，动态更新。

容器管理

涉及的开源项目有直接集成和间接调用两种。

直接集成

容器管理模块直接集成的项目包括但不限于：

1. DATA-DOG/go-sqlmock v1.5.0
2. Miniredis v2.22.0
3. cloudtty v0.0.3-0.20220525041218-c0b20763ee9f
4. clusterpedia-io/api v0.0.0-20220802044336-d3ea49998d11
5. clusterpedia-io/client-go v0.0.0-20220721030555-ba14329a2df5
6. clusterpedia-io/fake-apiserver v0.0.0-20220726040952-67538a31d6c2
7. Docker v2.8.1+incompatible
8. fsnotify v1.5.4
9. yaml v1.0.0
10. go-openapi/runtime v0.24.1
11. go-openapi/strfmt v0.21.3

12. go-redis/redis v8.11.5
13. golang/mock v1.6.0
14. google/go-cmp v0.5.8
15. google/uuid v1.3.0
16. gorilla/mux v1.8.0
17. grpc-ecosystem/go-grpc-middleware v1.3.0
18. grpc-ecosystem/grpc-gateway v2.10.3
19. heroku/docker-registry-client v0.0.0-20211012143308-9463674c8930
20. kubernetes-csi/external-snapshotter v4.2.0
21. moby v20.10.17+incompatible
22. onsi/ginkgo v2.1.4
23. onsi/gomega v1.20.1
24. patrickmn/go-cache v2.1.0+incompatible
25. prometheus/client_golang v1.13.0
26. russross/blackfriday v2.1.0
27. shirou/gopsutil v3.22.7
28. soheilhy/cmux v0.1.5
29. spf13/cobra v1.5.0
30. spf13/pflag v1.0.5
31. spf13/viper v1.10.1
32. stretchr/testify v1.8.0
33. golang.org/x/sync v0.0.0-20220601150217-0de741cfad7f
34. golang.org/x/term v0.0.0-20210927222741-03fcf44c2211
35. golang.org/x/text v0.3.7
36. golang.org/x/tools v0.1.10
37. google.golang.org/grpc v1.46.2

38. google.golang.org/protobuf v1.28.1
39. gopkg.in/yaml.v2 v2.4.0
40. gopkg.in/yaml.v3 v3.0.1
41. gorm.io/driver/mysql v1.3.2
42. gorm.io/driver/postgres v1.3.1
43. gorm.io/gorm v1.23.2
44. helm.sh/helm/v3 v3.8.0
45. k8s.io/api v0.24.3
46. k8s.io/apiextensions-apiserver v0.24.0
47. k8s.io/apimachinery v0.24.3
48. k8s.io/apiserver v0.24.0
49. k8s.io/client-go v0.24.3
50. k8s.io/code-generator v0.24.1
51. k8s.io/component-base v0.24.0
52. k8s.io/component-helpers v0.24.0
53. k8s.io/klog v2.60.1
54. k8s.io/kubectl v0.24.0
55. k8s.io/kubernetes v1.24.0
56. k8s.io/metrics v0.24.0
57. k8s.io/utils v0.0.0-20220210201930-3a6ce19ff2f9
58. kubean.io/api
59. sigs.k8s.io/cli-utils v0.32.0
60. sigs.k8s.io/cluster-api v1.1.3
61. sigs.k8s.io/controller-runtime v0.12.1
62. sigs.k8s.io/yaml v1.3.0

间接调用

容器管理模块间接调用的项目包括但不限于：

1. github.com/Azure/go-ansiterm v0.0.0-20210617225240-d185dfc1b5a1
2. github.com/Masterminds/semver/v3 v3.1.1
3. github.com/NYTimes/gziphandler v1.1.1
4. github.com/PuerkitoBio/purell v1.1.1
5. github.com/PuerkitoBio/urlesc v0.0.0-20170810143723-de5bf2ad4578
6. github.com/alicebob/gopher-json v0.0.0-20200520072559-a9ecdc9d1d3a
7. github.com/asaskevich/govalidator v0.0.0-20210307081110-f21760c49a8d
8. github.com/beorn7/perks v1.0.1
9. github.com/blang/semver v3.5.1+incompatible
10. github.com/blang/semver/v4 v4.0.0
11. github.com/cespare/xxhash/v2 v2.1.2
12. github.com/containerd/containerd v1.5.13
13. github.com/coreos/go-semver v0.3.0
14. github.com/coreos/go-systemd/v22 v22.3.2
15. github.com/davecgh/go-spew v1.1.1
16. github.com/dgryski/go-rendezvous v0.0.0-20200823014737-9f7001d12a5f
17. github.com/docker/cli v20.10.11+incompatible
18. github.com/docker/docker v20.10.12+incompatible
19. github.com/docker/docker-credential-helpers v0.6.4
20. github.com/docker/go-connections v0.4.0
21. github.com/docker/go-metrics v0.0.1
22. github.com/docker/go-units v0.4.0

23. github.com/docker/libtrust v0.0.0-20160708172513-aabc10ec26b7
24. github.com/emicklei/go-restful v2.9.5+incompatible
25. github.com/envoyproxy/protoc-gen-validate v0.6.2
26. github.com/evanphx/json-patch v4.12.0+incompatible
27. github.com/felixge/httpsnoop v1.0.1
28. github.com/go-errors/errors v1.0.1
29. github.com/go-logr/logr v1.2.0
30. github.com/go-ole/go-ole v1.2.6
31. github.com/go-openapi/analysis v0.21.2
32. github.com/go-openapi/errors v0.20.2
33. github.com/go-openapi/jsonpointer v0.19.5
34. github.com/go-openapi/jsonreference v0.19.6
35. github.com/go-openapi/loads v0.21.1
36. github.com/go-openapi/spec v0.20.4
37. github.com/go-openapi/swag v0.21.1
38. github.com/go-openapi/validate v0.22.0
39. github.com/go-sql-driver/mysql v1.6.0
40. github.com/gobuffalo/flect v0.2.4
41. github.com/gogo/protobuf v1.3.2
42. github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da
43. github.com/golang/protobuf v1.5.2
44. github.com/google/btree v1.0.1
45. github.com/google/gnostic v0.5.7-v3refs
46. github.com/google/gofuzz v1.2.0
47. github.com/google/shlex v0.0.0-20191202100458-e7afc7fbc510

48. github.com/gregjones/httpcache v0.0.0-20180305231024-9cad4c3443a7
49. github.com/grpc-ecosystem/go-grpc-prometheus v1.2.0
50. github.com/grpc-ecosystem/grpc-gateway v1.16.0
51. github.com/hashicorp/hcl v1.0.0
52. github.com/imdario/mergo v0.3.12
53. github.com/inconshreveable/mousetrap v1.0.0
54. github.com/jackc/chunkreader/v2 v2.0.1
55. github.com/jackc/pgconn v1.11.0
56. github.com/jackc/pgio v1.0.0
57. github.com/jackc/pgpassfile v1.0.0
58. github.com/jackc/pgproto3/v2 v2.2.0
59. github.com/jackc/pgservicefile v0.0.0-20200714003250-2b9c44734f2b
60. github.com/jackc/pgtype v1.10.0
61. github.com/jackc/pgx/v4 v4.15.0
62. github.com/jhump/protoreflect v1.12.0
63. github.com/jinzhu/inflection v1.0.0
64. github.com/jinzhu/now v1.1.4
65. github.com/josharian/intern v1.0.0
66. github.com/json-iterator/go v1.1.12
67. github.com/klauspost/compress v1.13.6
68. github.com/liggitt/tabwriter v0.0.0-20181228230101-89fcab3d43de
69. github.com/lufia/plan9stats v0.0.0-20211012122336-39d0f177ccd0
70. github.com/magiconair/properties v1.8.5
71. github.com/mailru/easyjson v0.7.7

72. github.com/matttproud/golang_protobuf_extensions v1.0.2-0.20181231171920-c182affec369
73. github.com/mitchellh/mapstructure v1.4.3
74. github.com/moby/locker v1.0.1
75. github.com/moby/term v0.0.0-20210619224110-3f7ff695adc6
76. github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd
77. github.com/modern-go/reflect2 v1.0.2
78. github.com/monochromegane/go-gitignore v0.0.0-20200626010858-205db1a8cc00
79. github.com/morikuni/aec v1.0.0
80. github.com/munnerz/goautoneg v0.0.0-20191010083416-a7dc8b61c822
81. github.com/mxk/go-flowrate v0.0.0-20140419014527-cca7078d478f
82. github.com/oklog/ulid v1.3.1
83. github.com/opencontainers/go-digest v1.0.0
84. github.com/opencontainers/image-spec v1.0.2
85. github.com/opentracing/opentracing-go v1.2.0
86. github.com/pelletier/go-toml v1.9.4
87. github.com/peterbourgon/diskv v2.0.1+incompatible
88. github.com/pkg/errors v0.9.1
89. github.com/pmezard/go-difflib v1.0.0
90. github.com/power-devops/perfstat v0.0.0-20210106213030-5aafc221ea8c
91. github.com/prometheus/client_model v0.2.0
92. github.com/prometheus/common v0.37.0
93. github.com/prometheus/procfs v0.8.0

94. github.com/sirupsen/logrus v1.8.1
95. github.com/spf13/afero v1.6.0
96. github.com/spf13/cast v1.4.1
97. github.com/spf13/jwalterweatherman v1.1.0
98. github.com/subosito/gotenv v1.2.0
99. github.com/tklauser/go-sysconf v0.3.10
100. github.com/tklauser/numcpus v0.4.0
101. github.com/xlab/treeprint v1.1.0
102. github.com/yuin/gopher-lua v0.0.0-20220504180219-658193537a64
103. github.com/yusufpapurcu/wmi v1.2.2
104. go.etcd.io/etcd/api/v3 v3.5.1
105. go.etcd.io/etcd/client/pkg/v3 v3.5.1
106. go.etcd.io/etcd/client/v3 v3.5.1
107. go.mongodb.org/mongo-driver v1.10.0
108. go.opentelemetry.io/contrib v0.20.0
109. go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc v0.20.0
110. go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp v0.20.0
111. go.opentelemetry.io/otel v0.20.0
112. go.opentelemetry.io/otel/exporters/otlp v0.20.0
113. go.opentelemetry.io/otel/metric v0.20.0
114. go.opentelemetry.io/otel/sdk v0.20.0
115. go.opentelemetry.io/otel/sdk/export/metric v0.20.0
116. go.opentelemetry.io/otel/sdk/metric v0.20.0
117. go.opentelemetry.io/otel/trace v0.20.0
118. go.opentelemetry.io/proto/otlp v0.7.0
119. go.starlark.net v0.0.0-20200306205701-8dd3e2ee1dd5

120. go.uber.org/atomic v1.9.0
121. go.uber.org/multierr v1.6.0
122. go.uber.org/zap v1.19.1
123. golang.org/x/crypto v0.0.0-20220622213112-05595931fe9d
124. golang.org/x/mod v0.6.0-dev.0.20220106191415-9b9b3d81d5e3
125. golang.org/x/net v0.0.0-20220425223048-2871e0cb64e4
126. golang.org/x/oauth2 v0.0.0-20220411215720-9780585627b5
127. golang.org/x/sys v0.0.0-20220520151302-bc2c85ada10a
128. golang.org/x/time v0.0.0-20220210224613-90d013bbcef8
129. golang.org/x/xerrors v0.0.0-20200804184101-5ec99f83aff1
130. gomodules.xyz/jsonpatch/v2 v2.2.0
131. google.golang.org/appengine v1.6.7
132. google.golang.org/genproto v0.0.0-20220519153652-3a47de7e79bd
133. gopkg.in/inf.v0 v0.9.1
134. gopkg.in/ini.v1 v1.66.2
135. gorm.io/datatypes v1.0.6
136. k8s.io/cli-runtime v0.24.0
137. k8s.io/gengo v0.0.0-20211129171323-c02415ce4185
138. k8s.io/kube-openapi v0.0.0-20220328201542-3ee0da9b0b42
139. oras.land/oras-go v1.1.0
140. sigs.k8s.io/apiserver-network-proxy/konnectivity-client v0.0.30
141. sigs.k8s.io/json v0.0.0-20211208200746-9f7c6b3444d2
142. sigs.k8s.io/kustomize/api v0.11.4
143. sigs.k8s.io/kustomize/kyaml v0.13.6
144. sigs.k8s.io/structured-merge-diff/v4 v4.2.1

全局管理

涉及的开源项目有直接集成和间接调用两种。

直接集成

全局管理模块直接集成的项目包括但不限于：

1. ghippo.io/api v0.0.0
2. github.com/DATA-DOG/go-sqlmock v1.5.0
3. github.com/Nerzal/gocloak/v10 v10.0.1
4. github.com/agiledragon/gomonkey/v2 v2.8.0
5. github.com/coreos/go-oidc v2.2.1+incompatible
6. github.com/dgrijalva/jwt-go v3.2.0+incompatible
7. github.com/gin-gonic/gin v1.8.1
8. github.com/go-logr/zapr v1.2.3
9. github.com/go-playground/validator/v10 v10.11.0
10. github.com/go-sql-driver/mysql v1.6.0
11. github.com/golang-jwt/jwt/v4 v4.4.3
12. github.com/golang-migrate/migrate/v4 v4.15.2
13. github.com/google/uuid v1.3.0
14. github.com/grpc-ecosystem/go-grpc-middleware v1.3.0
15. github.com/grpc-ecosystem/grpc-gateway/v2 v2.10.0
16. github.com/grpc-ecosystem/protoc-gen-grpc-gateway-ts v1.1.1
17. github.com/jackc/pgconn v1.13.0
18. github.com/jackc/pgerrcode v0.0.0-20201024163028-a0d42d470451
19. github.com/jackc/pgx/v4 v4.17.2
20. github.com/jinzhu/configor v1.2.1
21. github.com/lestrrat-go/jwx/v2 v2.0.4
22. github.com/lib/pq v1.10.5

23. github.com/nicksnyder/go-i18n/v2 v2.2.0
24. github.com/onsi/ginkgo/v2 v2.3.1
25. github.com/pkg/errors v0.9.1
26. github.com/robfig/cron/v3 v3.0.1
27. github.com/smartystreets/goconvey v1.7.2
28. github.com/soheilhy/cmux v0.1.5
29. github.com/stretchr/testify v1.8.1
30. github.com/uptrace/opentelemetry-go-extra/otelgorm v0.1.17
31. github.com/urfave/cli/v2 v2.3.0
32. go.opentelemetry.io/contrib/instrumentation/github.com/gin-gonic/gin/otelgin v0.36.3
33. go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc v0.32.0
34. go.opentelemetry.io/otel v1.11.0
35. go.opentelemetry.io/otel/exporters/otlp/otlptrace v1.10.0
36. go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc v1.10.0
37. go.opentelemetry.io/otel/exporters/prometheus v0.30.0
38. go.opentelemetry.io/otel/metric v0.32.1
39. go.opentelemetry.io/otel/sdk v1.10.0
40. go.opentelemetry.io/otel/sdk/metric v0.31.0
41. go.uber.org/zap v1.19.1
42. golang.org/x/oauth2 v0.0.0-20220309155454-6242fa91716a
43. golang.org/x/sync v0.1.0
44. golang.org/x/text v0.6.0
45. google.golang.org/grpc v1.50.1
46. google.golang.org/grpc/cmd/protoc-gen-go-grpc v1.1.0

47. google.golang.org/protobuf v1.28.1
48. gopkg.in/natefinch/lumberjack.v2 v2.0.0
49. gorm.io/driver/mysql v1.4.4
50. gorm.io/driver/postgres v1.4.4
51. gorm.io/gorm v1.23.10
52. istio.io/api v0.0.0-20220304230428-0cdff8a2764f
53. istio.io/client-go v1.13.2
54. k8s.io/apimachinery v0.24.3
55. k8s.io/client-go v0.24.3
56. k8s.io/code-generator v0.24.3
57. k8s.io/klog/v2 v2.90.0
58. kpanda.io/api v0.0.0
59. sigs.k8s.io/controller-runtime v0.12.3

间接调用

全局管理模块间接调用的项目包括但不限于：

1. github.com/cenkalti/backoff/v4 v4.1.3
2. github.com/decred/dcrd/dcrec/secp256k1/v4 v4.0.1
3. github.com/emicklei/go-restful/v3 v3.10.1
4. github.com/go-logr/stdr v1.2.2
5. github.com/go-openapi/swag v0.22.3
6. github.com/goccy/go-json v0.9.10
7. github.com/google/gnostic v0.6.9
8. github.com/gopherjs/gopherjs v0.0.0-20181017120253-0766667cb4d1
9. github.com/jtolds/gls v4.20.0+incompatible
10. github.com/lestrrat-go/blackmagic v1.0.1
11. github.com/lestrrat-go/httpcc v1.0.1

12. github.com/lestrrat-go/httprc v1.0.4
13. github.com/lestrrat-go/iter v1.0.2
14. github.com/lestrrat-go/option v1.0.0
15. github.com/munnerz/goautoneg v0.0.0-20191010083416-a7dc8b61c822
16. github.com/pelletier/go-toml/v2 v2.0.1
17. github.com/smartystreets/assertions v1.2.0
18. github.com/uptrace/opentelemetry-go-extra/otelsql v0.1.17
19. go.opentelemetry.io/otel/exporters/otlp/internal/retry v1.10.0
20. go.opentelemetry.io/otel/trace v1.11.0
21. go.opentelemetry.io/proto/otlp v0.19.0
22. k8s.io/kube-openapi v0.0.0-20230127205639-68031ae9242a
23. cloud.google.com/go v0.99.0
24. github.com/Azure/go-autorest v14.2.0+incompatible
25. github.com/Azure/go-autorest/autorest v0.11.18
26. github.com/Azure/go-autorest/autorest/adal v0.9.16
27. github.com/Azure/go-autorest/autorest/date v0.3.0
28. github.com/Azure/go-autorest/logger v0.2.1
29. github.com/Azure/go-autorest/tracing v0.6.0
30. github.com/BurntSushi/toml v1.2.0
31. github.com/Masterminds/goutils v1.1.0
32. github.com/Masterminds/semver v1.5.0
33. github.com/Masterminds/sprig v2.22.0+incompatible
34. github.com/beorn7/perks v1.0.1
35. github.com/cespare/xxhash/v2 v2.1.2
36. github.com/cpuguy83/go-md2man/v2 v2.0.1
37. github.com/davecgh/go-spew v1.1.1

38. github.com/envoyproxy/protoc-gen-validate v0.6.2
39. github.com/evanphx/json-patch v4.12.0+incompatible
40. github.com/fsnotify/fsnotify v1.5.1
41. github.com/gin-contrib/sse v0.1.0
42. github.com/go-logr/logr v1.2.3
43. github.com/go-openapi/jsonpointer v0.19.6
44. github.com/go-openapi/jsonreference v0.20.2
45. github.com/go-playground/locales v0.14.0
46. github.com/go-playground/universal-translator v0.18.0
47. github.com/go-resty/resty/v2 v2.6.0
48. github.com/gogo/protobuf v1.3.2
49. github.com/golang/glog v1.0.0
50. github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da
51. github.com/golang/protobuf v1.5.2
52. github.com/google/go-cmp v0.5.9
53. github.com/google/gofuzz v1.2.0
54. github.com/hashicorp/errwrap v1.1.0
55. github.com/hashicorp/go-multierror v1.1.1
56. github.com/huandu/xstrings v1.3.2
57. github.com/iancoleman/strcase v0.2.0
58. github.com/imdario/mergo v0.3.12
59. github.com/jackc/chunkreader/v2 v2.0.1
60. github.com/jackc/pgio v1.0.0
61. github.com/jackc/pgpassfile v1.0.0
62. github.com/jackc/pgproto3/v2 v2.3.1

63. github.com/jackc/pgservicefile v0.0.0-20200714003250-2b9c44734f2b
64. github.com/jackc/pgtype v1.12.0
65. github.com/jinzhu/inflection v1.0.0
66. github.com/jinzhu/now v1.1.5
67. github.com/josharian/intern v1.0.0
68. github.com/json-iterator/go v1.1.12
69. github.com/leodido/go-urn v1.2.1
70. github.com/mailru/easyjson v0.7.7
71. github.com/mattn/go-isatty v0.0.14
72. github.com/matttproud/golang_protobuf_extensions v1.0.2-0.20181231171920-c182affec369
73. github.com/mitchellh/copystructure v1.0.0
74. github.com/mitchellh/reflectwalk v1.0.0
75. github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd
76. github.com/modern-go/reflect2 v1.0.2
77. github.com/opentracing/opentracing-go v1.2.0
78. github.com/pmezard/go-difflib v1.0.0
79. github.com/pquerna/cachecontrol v0.0.0-20171018203845-0dec1b30a021
80. github.com/prometheus/client_golang v1.12.1
81. github.com/prometheus/client_model v0.2.0
82. github.com/prometheus/common v0.32.1
83. github.com/prometheus/procfs v0.7.3
84. github.com/russross/blackfriday/v2 v2.1.0
85. github.com/segmentio/ksuid v1.0.4

86. github.com/sirupsen/logrus v1.8.1
87. github.com/spf13/pflag v1.0.5
88. github.com/stretchr/objx v0.5.0
89. github.com/tjfoc/gmsm v1.4.2-0.20220114090716-36b992c51540
90. github.com/ugorji/go/codec v1.2.7
91. go.uber.org/atomic v1.7.0
92. go.uber.org/multierr v1.8.0
93. golang.org/x/crypto v0.0.0-20220722155217-630584e8d5aa
94. golang.org/x/mod v0.7.0
95. golang.org/x/net v0.5.0
96. golang.org/x/sys v0.4.0
97. golang.org/x/term v0.4.0
98. golang.org/x/time v0.0.0-20220224211638-0e9765cccd65
99. golang.org/x/tools v0.5.0
100. gomodules.xyz/jsonpatch/v2 v2.2.0
101. google.golang.org/appengine v1.6.7
102. google.golang.org/genproto v0.0.0-20220317150908-0efb43f6373e
103. gopkg.in/inf.v0 v0.9.1
104. gopkg.in/square/go-jose.v2 v2.5.1; indi7509080b3f373e
105. gopkg.in/yaml.v2 v2.4.0
106. gopkg.in/yaml.v3 v3.0.1
107. istio.io/gogo-genproto v0.0.0-20211208193508-5ab4acc9eb1e
108. k8s.io/api v0.24.3
109. k8s.io/apiextensions-apiserver v0.24.2
110. k8s.io/component-base v0.24.2
111. k8s.io/gengo v0.0.0-20221011193443-fad74ee6edd9
112. k8s.io/utils v0.0.0-20220210201930-3a6ce19ff2f9

113. sigs.k8s.io/json v0.0.0-20211208200746-9f7c6b3444d2
114. sigs.k8s.io/structured-merge-diff/v4 v4.2.3
115. sigs.k8s.io/yaml v1.3.0

可观测性

涉及的开源项目有直接集成和间接调用两种。

直接集成

可观测性模块直接集成的项目包括但不限于：

1. ghippo.io/api v0.0.0
2. github.com/DATA-DOG/go-sqlmock v1.5.0
3. github.com/elastic/go-elasticsearch/v7 v7.16.0
4. github.com/gorilla/mux v1.8.0
5. github.com/json-iterator/go v1.1.12
6. github.com/mwitkow/go-conntrack v0.0.0-20190716064945-2f068394615f
7. github.com/oklog/ulid/v2 v2.1.0
8. github.com/pkg/errors v0.9.1
9. github.com/prometheus-community/prom-label-proxy v0.4.0
10. github.com/prometheus/client_golang v1.14.0
11. github.com/prometheus/common v0.39.0
12. github.com/prometheus/prometheus v1.8.2-0.20210811141203-dcb07e8eac34
13. github.com/soheilhy/cmux v0.1.5
14. github.com/spf13/cobra v1.6.1
15. github.com/spf13/pflag v1.0.5
16. github.com/spf13/viper v1.14.0
17. github.com/stretchr/testify v1.8.1

18. go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc v0.37.0
19. go.opentelemetry.io/otel v1.11.2
20. go.opentelemetry.io/otel/exporters/otlp/otlptrace v1.11.2
21. go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc v1.4.1
22. go.opentelemetry.io/otel/exporters/prometheus v0.30.0
23. go.opentelemetry.io/otel/metric v0.34.0
24. go.opentelemetry.io/otel/sdk v1.11.2
25. go.opentelemetry.io/otel/sdk/metric v0.31.0
26. go.opentelemetry.io/otel/trace v1.11.2
27. go.uber.org/zap v1.24.0
28. google.golang.org/grpc v1.51.0
29. google.golang.org/protobuf v1.28.1
30. gorm.io/driver/mysql v1.3.4
31. gorm.io/driver/postgres v1.3.7
32. gorm.io/gorm v1.23.8
33. gotest.tools v2.2.0+incompatible
34. insight.io/api v0.0.0
35. k8s.io/apimachinery v0.24.6
36. k8s.io/client-go v0.24.6
37. kpanda.io/api v0.0.0
38. sigs.k8s.io/yaml v1.3.0
39. github.com/VictoriaMetrics/metricsql v0.43.0
40. github.com/VictoriaMetrics/operator/api v0.0.0-20220623030345-4957cfea49fb
41. github.com/aquasecurity/esquery v0.2.0
42. github.com/deckarep/golang-set/v2 v2.1.0

43. github.com/dgrijalva/jwt-go v3.2.0+incompatible
44. github.com/gocarina/gocsv v0.0.0-20220729221910-a7386ae0b221
45. github.com/golang-migrate/migrate/v4 v4.15.2
46. github.com/grpc-ecosystem/go-grpc-middleware v1.3.0
47. github.com/grpc-ecosystem/grpc-gateway/v2 v2.10.3
48. github.com/hashicorp/golang-lru v0.5.4
49. github.com/kr/pretty v0.3.1
50. github.com/muesli/cache2go v0.0.0-20211005105910-8e46465cca4a
51. github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring v0.57.0
52. github.com/prometheus-operator/prometheus-operator/pkg/client v0.57.0
53. github.com/robfig/cron/v3 v3.0.1
54. github.com/uptrace/opentelemetry-go-extra/otelgorm v0.1.15
55. gopkg.in/gomail.v2 v2.0.0-20160411212932-81ebce5c23df
56. moul.io/zapgorm2 v1.1.3
57. sigs.k8s.io/controller-runtime v0.12.0
58. github.com/agiledragon/gomonkey/v2 v2.7.0
59. github.com/gogo/protobuf v1.3.2
60. gopkg.in/yaml.v2 v2.4.0

间接调用

可观测性模块间接调用的项目包括但不限于：

1. github.com/evanphx/json-patch v5.6.0+incompatible
2. github.com/google/gofuzz v1.2.0
3. github.com/pelletier/go-toml v1.9.5
4. github.com/pelletier/go-toml/v2 v2.0.5

5. github.com/rogpeppe/go-internal v1.9.0
6. github.com/VictoriaMetrics/VictoriaMetrics v1.77.1
7. github.com/VictoriaMetrics/fasthttp v1.1.0
8. github.com/VictoriaMetrics/metrics v1.18.1
9. github.com/asaskevich/govalidator v0.0.0-20210307081110-f21760c49a8d
10. github.com/beorn7/perks v1.0.1
11. github.com/cenkalti/backoff/v4 v4.2.0
12. github.com/cespare/xxhash/v2 v2.1.2
13. github.com/davecgh/go-spew v1.1.1
14. github.com/efficientgo/tools/core v0.0.0-20210201224146-3d78f4d30648
15. github.com/emicklei/go-restful/v3 v3.9.0
16. github.com/envoyproxy/protoc-gen-validate v0.6.8
17. github.com/fatih/structs v1.1.0
18. github.com/fsnotify/fsnotify v1.6.0
19. github.com/go-kit/log v0.2.1
20. github.com/go-logfmt/logfmt v0.5.1
21. github.com/go-logr/logr v1.2.3
22. github.com/go-logr/stdr v1.2.2
23. github.com/go-openapi/analysis v0.21.2
24. github.com/go-openapi/errors v0.20.2
25. github.com/go-openapi/jsonpointer v0.19.5
26. github.com/go-openapi/jsonreference v0.20.0
27. github.com/go-openapi/loads v0.21.1
28. github.com/go-openapi/runtime v0.24.1
29. github.com/go-openapi/spec v0.20.6

30. github.com/go-openapi/strfmt v0.21.3
31. github.com/go-openapi/swag v0.22.3
32. github.com/go-openapi/validate v0.22.0
33. github.com/go-sql-driver/mysql v1.6.0
34. github.com/gogo/googleapis v1.4.1
35. github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da
36. github.com/golang/protobuf v1.5.2
37. github.com/google/gnostic v0.6.9
38. github.com/google/go-cmp v0.5.9
39. github.com/google/uuid v1.3.0
40. github.com/hashicorp/errwrap v1.1.0
41. github.com/hashicorp/go-multierror v1.1.1
42. github.com/hashicorp/hcl v1.0.0
43. github.com/imdario/mergo v0.3.13
44. github.com/inconshreveable/mousetrap v1.0.1
45. github.com/jackc/chunkreader/v2 v2.0.1
46. github.com/jackc/pgconn v1.13.0
47. github.com/jackc/pgio v1.0.0
48. github.com/jackc/pgpassfile v1.0.0
49. github.com/jackc/pgproto3/v2 v2.3.1
50. github.com/jackc/pgservicefile v0.0.0-20200714003250-2b9c44734f2b
51. github.com/jackc/pgtype v1.12.0
52. github.com/jackc/pgx/v4 v4.17.0
53. github.com/jinzhu/inflection v1.0.0
54. github.com/jinzhu/now v1.1.5

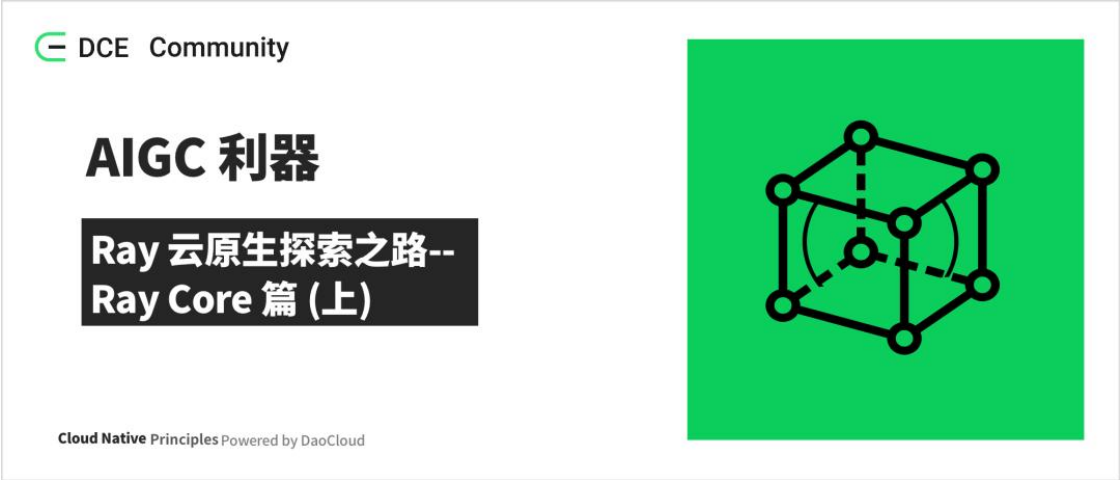
55. github.com/josharian/intern v1.0.0
56. github.com/jpillora/backoff v1.0.0
57. github.com/klauspost/compress v1.15.11
58. github.com/kr/text v0.2.0
59. github.com/lib/pq v1.10.6
60. github.com/magiconair/properties v1.8.6
61. github.com/mailru/easyjson v0.7.7
62. github.com/matttproud/golang_protobuf_extensions v1.0.4
63. github.com/mitchellh/mapstructure v1.5.0
64. github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd
65. github.com/modern-go/reflect2 v1.0.2
66. github.com/munnerz/goautoneg v0.0.0-20191010083416-a7dc8b61c822
67. github.com/oklog/ulid v1.3.1
68. github.com/opentracing/opentracing-go v1.2.0
69. github.com/pmezard/go-difflib v1.0.0
70. github.com/prometheus/alertmanager v0.23.0
71. github.com/prometheus/client_model v0.3.0
72. github.com/prometheus/procfs v0.8.0
73. github.com/shopspring/decimal v1.3.1
74. github.com/spf13/afero v1.9.2
75. github.com/spf13/cast v1.5.0
76. github.com/spf13/jwalterweatherman v1.1.0
77. github.com/stretchr/objx v0.5.0
78. github.com/subosito/gotenv v1.4.1
79. github.com/uber/jaeger-client-go v2.30.0+incompatible

80. github.com/uber/jaeger-lib v2.4.1+incompatible
81. github.com/uptrace/opentelemetry-go-extra/otelsql v0.1.15
82. github.com/valyala/bytebufferpool v1.0.0
83. github.com/valyala/fastrand v1.1.0
84. github.com/valyala/fasttemplate v1.2.1
85. github.com/valyala/histogram v1.2.0
86. github.com/valyala/quicktemplate v1.7.0
87. go.mongodb.org/mongo-driver v1.10.0
88. go.opentelemetry.io/otel/exporters/otlp/internal/retry v1.11.2
89. go.opentelemetry.io/proto/otlp v0.19.0
90. go.uber.org/atomic v1.10.0
91. go.uber.org/multierr v1.8.0
92. golang.org/x/crypto v0.0.0-20220722155217-630584e8d5aa
93. golang.org/x/net v0.4.0
94. golang.org/x/oauth2 v0.3.0
95. golang.org/x/sys v0.3.0
96. golang.org/x/term v0.3.0
97. golang.org/x/text v0.5.0
98. golang.org/x/time v0.0.0-20220609170525-579cf78fd858
99. gomodules.xyz/jsonpatch/v2 v2.2.0
100. google.golang.org/appengine v1.6.7
101. gopkg.in/alexcesaro/quotedprintable.v3 v3.0.0-20150716171945-2caba252f4dc
102. gopkg.in/inf.v0 v0.9.1
103. gopkg.in/ini.v1 v1.67.0
104. gopkg.in/yaml.v3 v3.0.1
105. k8s.io/api v0.24.6

106. k8s.io/apiextensions-apiserver v0.24.0
107. k8s.io/component-base v0.24.0
108. k8s.io/klog/v2 v2.80.1
109. k8s.io/kube-openapi v0.0.0-20220928191237-829ce0c27909
110. k8s.io/utils v0.0.0-20220728103510-ee6ede2d64ed
111. sigs.k8s.io/json v0.0.0-20211208200746-9f7c6b3444d2
112. sigs.k8s.io/structured-merge-diff/v4 v4.2.3

未完待续（例如网络、存储等基础模块还未包含在上述列表）

AIGC 利器 Ray 云原生探索之路 - Ray Core 篇 (上)

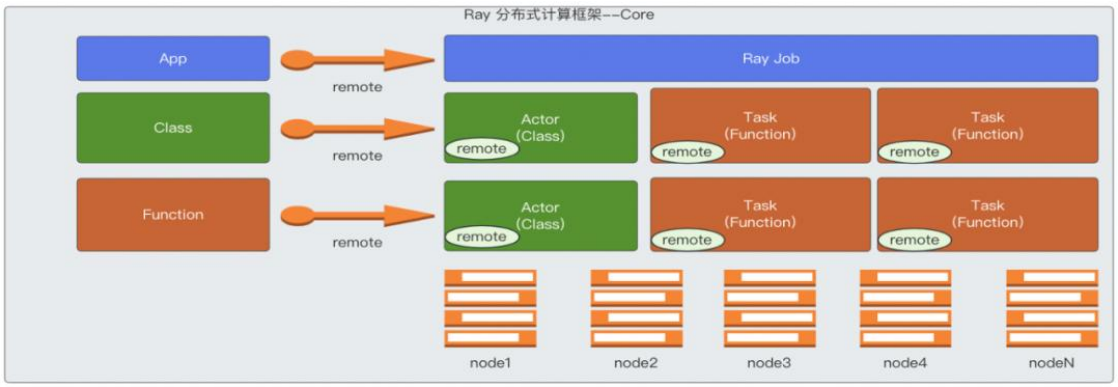


start

本文介绍 Ray 分布式计算的底座：Ray Core 是怎样的一种分布式的计算框架。

基本概念

Ray Core 是 Ray 的底层的分布式的计算框架，使用基于 actor 模型来实现的一套计算框架， 它可以将 Python 的一个 Class 或者一个 Function 转成分布式的 actor 和 task， 在所有的机器上分布式的运行，并且 tasks/actor 之间可以通过分布式的对象存储能力来共享和传递 object 来完成分布式的一些协作。
Ray Core 的这些能力为其它的更高层的模块提供了一个分布式计算的基础框架。



concepts

```python
import ray
import os
import requests

ray.init()

@ray.remote
class Counter:
    def __init__(self):
        # used to verify runtimeEnv
        self.name = os.getenv("counter_name")
        self.counter = 0
    def inc(self):
        self.counter += 1
    def get_counter(self):
        return "{} got {}".format(self.name, self.counter)

counter = Counter.remote()

for _ in range(5):
    ray.get(counter.inc.remote())
    print(ray.get(counter.get_counter.remote()))

print(requests.__version__)
```

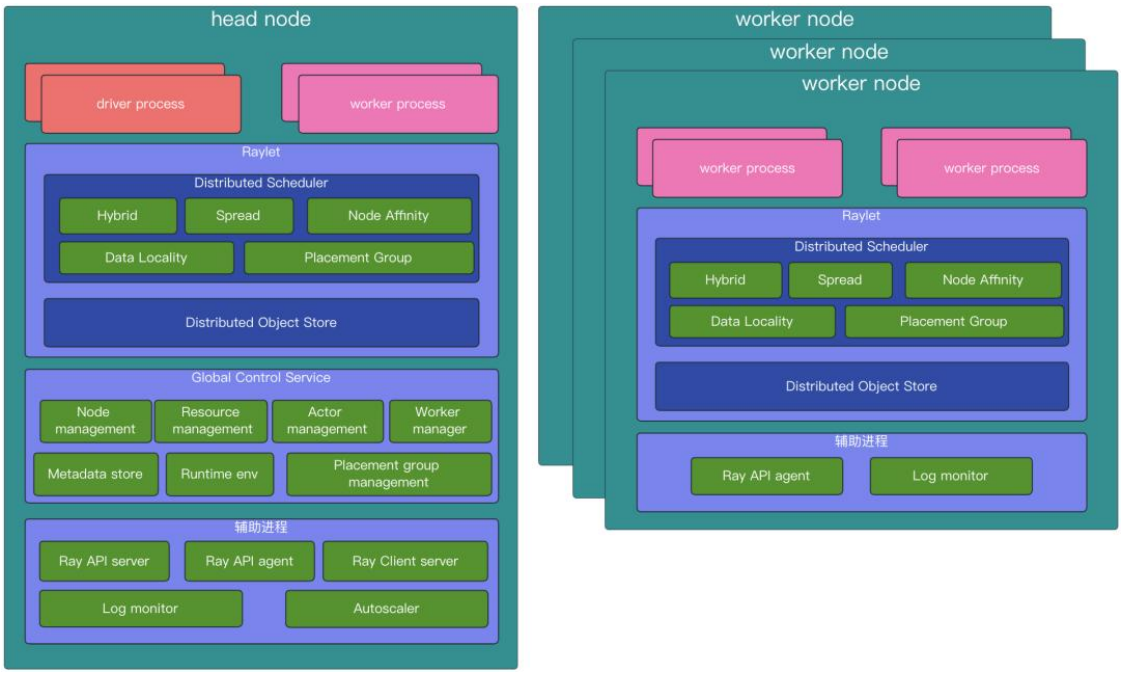
说明：

- class Counter 在标注了 @ray.remote 注解之后，同时使用 remote() 调用，如 counter = Counter.remote()， 这里的 counter 就是一个 actor 的 handle，而 Counter 就是一个 actor。而 class 中定义的 function 在使用 remote() 调用的时候，就表现成了一个 task。actor 是有状态，对于 Synchronous, Single-Threaded 类型的 actor， 对于同一个调用者

中按照顺序调用类成员方法的时候，需要保证顺序，才能保证结果的正确性，但是不同的调用者之间是可以并发处理的。 对于 Asynchronous 或 Threaded 类型的 actor，即使是同一个调用者在调用成员方法的时候，也是可以不用保证调用顺序，进行异步的执行。

- 在一般的 function（不是 class 的成员方法）的定义上也标注了 @ray.remote 注解之后，在调用的时候使用 remote()， 就表现成一个 task。task 可以分布在不同机器上，并行的进行运行，而且是异步的方式，同时是无状态的。

整体架构



architecture

driver process ： driver 是程序的根，或者理解成 main 程序，这里指的是运行了 ray.init()方法的 code。 driver process 是一种特殊的 worker process ，用于执行 driver 的 process，它会执行顶层的应用（如 __main__ in Python）。它能提交 task，但是它不是自己去执行 task。同时 driver processes 能运行在任何节点上，只是默认是运行在 head 节点。 driver process 也会处理来自 log

monitor 发送过来的 worker 节点上日志。同时，jobs 和 drivers 的关系是 1:1 的关系。

worker process ： worker process 负责 task 的提交和执行。一个 worker process 就是一个 python 的进程。 worker process 可以是无状态的，这种 worker process 可以被用来重复地执行一般的 task，这里一般的 task 指的就是使用了 ray.remote 标注的 function。worker process 也可以用于执行有状态的 actor，这里有状态的 actor 指的是使用了 ray.remote 的 class，在一个 actor 被实例化的时候，就会创建一个对应的 worker process，专门用于这个 actor 的运行，以及 actor 内的成员方式所对应的 task。每一个 worker process 和一个特定的 job 是相关的，默认初始化的 worker 的数量和所在机器的 cpu 核数一样。

Distributed Scheduler ： Distributed Scheduler 是 Raylet 组件的一部分，负责资源管理，task 放置的位置， 保证 task 运行所需要的参数对象可以从分布式对象存储的对象中获取。每个 Raylet 组件会跟踪本地节点的资源， 当一个资源请求被同意之后，Raylet 就会减少本地可用资源，一旦资源被使用完之后返还回来了，Raylet 又会增加本地的可用资源， 所以 Raylet 有一个强一致性的本地的可用资源的视图。同时，Raylet 也接收来自控制平面 GCS 服务的一些关于其它节点的资源使用的信息， 这个信息对于分布式调度是很有用的。比如，在集群范围内，可以均衡一点的调度 task。GCS 会周期性的去所有的 Raylet 拉取每一个 Raylet 的本地节点的可用资源，然后将这些信息通过广播的方式告知所有节点的 Raylet。一般的 task 提交是使用的分布式的方式提交 task， 也就是对于这些 task 的调度已经是去中心化了，举例来说，一个 task 的调用者在调度 task 的时候，首先会选择一个合适的 Raylet 发送 RequestWorkerLease PRC 请求，默认是发给本地节点的 Raylet 中的调度器，去申请运行 task 需要的资源。 但是也会根据数据本地化的原则，节点粘性的一些特性选择其它节点的 Raylet。当 Raylet 中的调度器经过分析之后， 觉得当前的节点就是合适的节点，那这个 task 就可以在本地的节点运行，如果本地 Raylet 觉得本地节点是不合适的， 那会告知任务的提交者去一个合适节点去申请资源，此时，任务的提交者就会去被推

荐的节点的 Raylet 的调度器去申请运行 task 的资源。 以此类推，以类似的机制处理调度的请求，直到找到一个合适的节点，这个过程就是一个去中心化的分布式的调度方式。

Distributed Object Store ： 负责存储和转移大对象。这里是分布式的方式支持对象存储。在本地存储的大对象， 在其它的节点上也是可以引用，在 task 执行的时候，如果 task 被调度的节点上，没有对应需要的对象， 就会到这个对象的节点去获取这个对象到本地，然后执行 task。这样不同节点的 task 之间是可以共享和引用这些对象的。

Global Control Service ： GCS 是一个 server，它是 ray 的控制平面，在新版的 ray 中，GCS 加强了容错能力， 这样 GCS 就可以运行在任意的节点，以及多个节点，而不是仅仅运行在 head 节点。GCS 服务主要包含以下能力：

1. 节点管理：管理节点的增加和删除，同时广播这些信息给所有节点的 Raylets，这样所有的 Raylet 就能感知到节点的变化。

2. 资源管理：广播每一个 Raylet 可用的资源给整个集群，这样可以保证每一个 Raylet 都可以及时的更新自己的全局的可用资源的视图， 在分布式调度 task 的时候是很重要的。Ray 会先自动探测节点的资源数据作为可以调度的逻辑资源的设置。

3. Actor 管理：管理 actor 的创建和删除请求，同时也监控 actor 的存活情况，以及在 actor 失败的时候重新创建 actor。 所有的 actor 在被调度之前是需要先注册到 GCS 中。

4. Placement group 管理：配合完成 Placement group 的创建和删除。管理 Placement group 的生命周期，同时使用 two phase commit protocol 创建 Placement group。Placement group 是属于 job 或者 actor 的，当 job 或者 actor 是处于 dead 状态，那 Placement group 就会被删除，这时所有使用了 Placement group 预留资源的 actors 和 tasks 都会被 kill，同时释放资源。

5. 元数据存储：提供了一个键值对的存储能力，这些数据可以让任何 worker 访问到。这里只存储一些小的元数据。 主要包括集群的 dashboard address，Remote function 的定义（当一个 worker 被指定运行一个 task， 它会去 GCS 下载该 task 对应的 function 的定义），Runtime environment 的数据 （默认情况下，Runtime environment 的工作目录是保存在 GCS），还有一些其它的组件也将元数据保存在 GCS， 比如 Ray Serve 组件将部署的信息就保存在 GCS 中。
6. Worker 管理器：处理 Raylet 上报上来的失败的 worker。
7. 运行时环境：管理运行时环境的包，同时也会统计这些包的使用量，以及回收。运行时环境的包是可以共享的，在资源不足的时候会回收一部分。

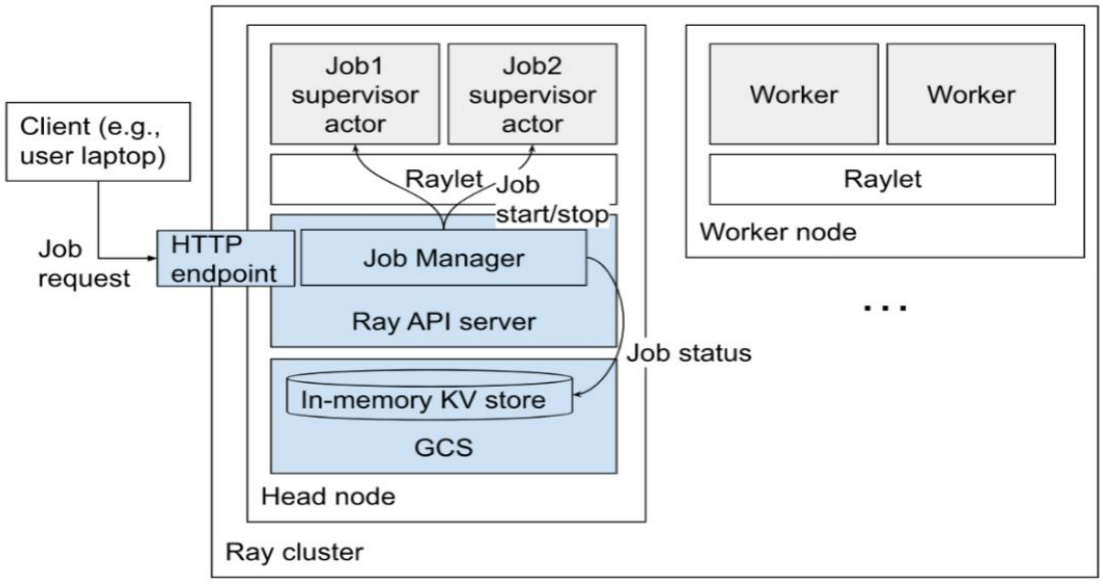
API server ： 不仅仅包含了 dashboard server，这里的 dashboard server 也可以理解成 dashboard 的 backend，同时还是提交 job 的 api 入口，也提供了一些集群状态的 api。

API agent ： 收集本地节点的指标数据，用于去聚合集群层面的健康数据。也负责给 task 和 actor 去安装运行时环境用于执行 task 和 actor。

Autoscaler ： 基于集群的资源的使用情况，以及 job 对资源的需求，去增加和移除集群的节点。

Log monitor ： 当 task 和 actor 打印 logs 到它的 stdout or stderr，这些输出会被自动的重定向到对应的 worker 的日志文件。 在每一个节点上都会运行一个 log monitor，它会定期的从本地的 ray 日志文件中读取这些日志文件，然后通过 GCS pubsub 的机制， 发布这些日志消息到 head 的 driver 中去。

Job 介绍



Image

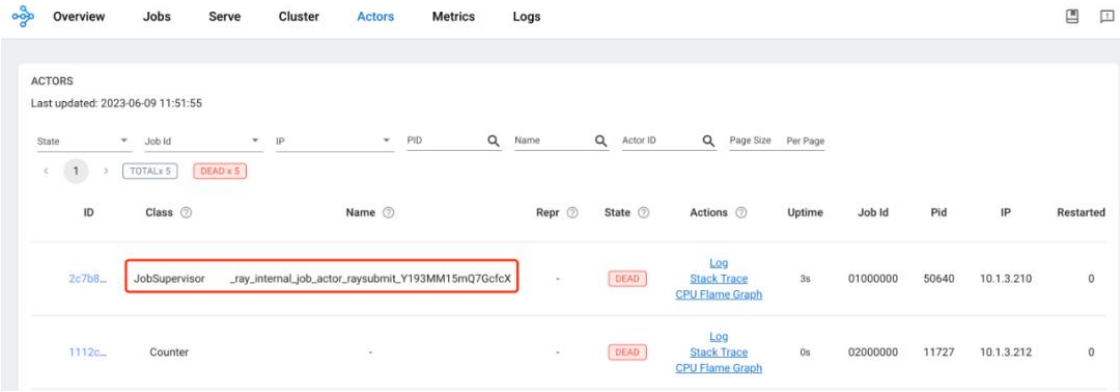
每个 job 由其对应专用的 job supervisor actor 来管理，job supervisor actor 是运行在 head 节点上， 后续会支持将 job supervisor actor 也可以运行在非 head 节点来减小 head 节点的压力。 job supervisor actor 运行在用户指定的运行环境下面，job 的 entrypoint command 作为其子进程进行运行， 同时会继承 job supervisor actor 运行的环境。这个 actor 在 ray 的 dashboard 中可以查看到，它是一个内部类型的 actor。

job manager 会管理这些 job supervisor actor 以及它的日志，同时 job 自己的 entrypoint script 的输出也会被写到 head 节点，这些日志文件可以通过 Http 的方式访问，也可以在 ray dashborad 上查看。

job 会报告自己的状态（如，PENDING、RUNNING、SUCCEEDED）和一些消息，这些被存储在 GCS 中，同时可以通过 API 的方式获取到。

如果想要停止一个 job，需要给对应的 job supervisor actor 发送停止事件，这个过程本身是异步的， job supervisor actor 会负责中断 job 的子进程以及更新 job 的状态。

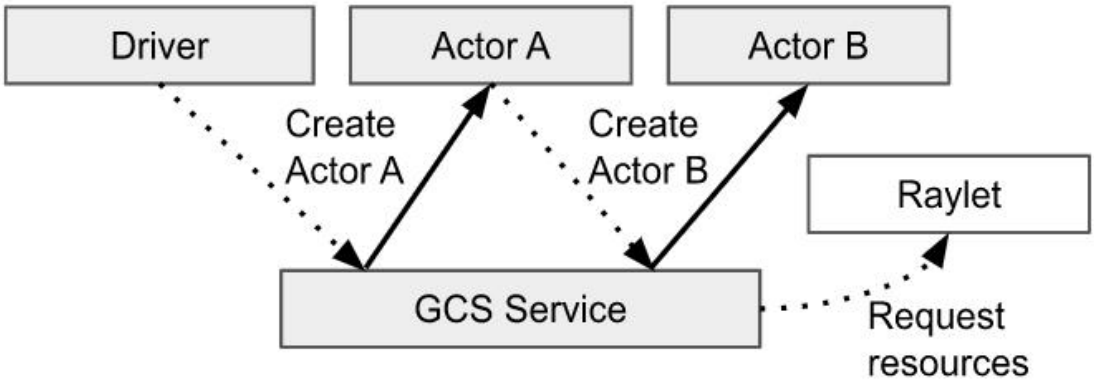
一个 job 会包含相关的 actors、tasks、objects 和 runtime env。jobs 和 drivers 的关系是 1:1 的关系。



Image

Actor 介绍

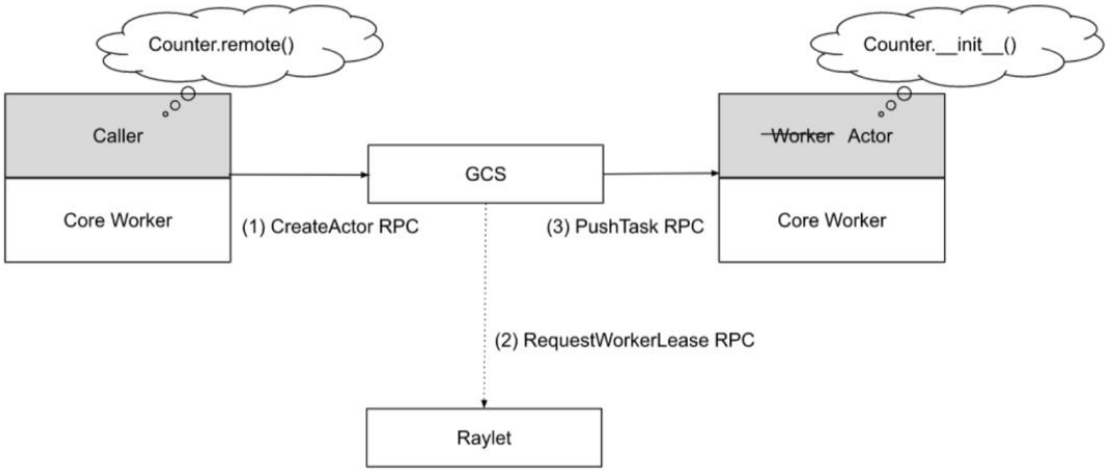
1. actor 的生命周期：



Image

- actor 的生命周期和它的元数据都是被 GCS 服务所管理的，每一个 actor 的 client 会缓存这个元数据在本地， 这些元数据的信息中有和发送 Tasks 相关的信息，比如 actor 的地址，client 可以通过这个地址直接通过 gRPC 发送 task 给 actor，这里发送的 task 主要指 actor 的成员方法所对应的 task。不像一般 task 的提交， 一般 task 的任务提交完全是由调用者控制什么时候调度以及运行，通过分布式调度完成调度工作。而 actor 的整个生命周期

都是由 GCS 集中式的管理，包括 actor 的调度请求也是由 GCS 发起的，而不是 actor 的调用者。actor 被实例化之后会创建一个 worker process，这个 worker 是当前这个 actor 专享的（不像给一般 task 使用的 worker，这种 worker 是可以被 task 重复使用的），所有这个 actor 的成员方法的调用（也就是 task），会被调度这个指定的 worker， 这些方法可以访问和修改这个 worker 的状态。



Image

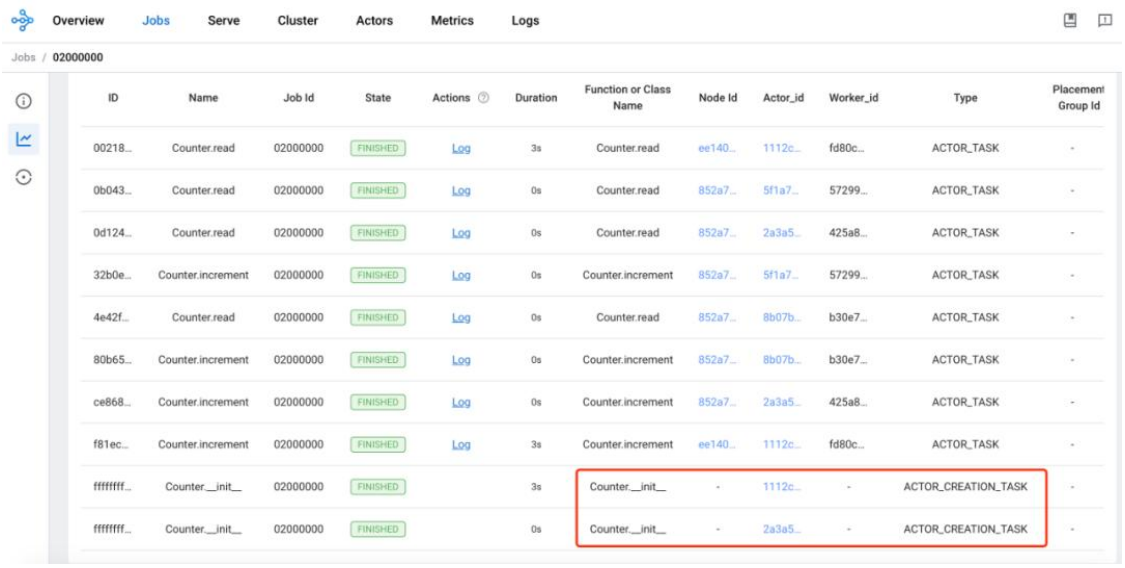
- 当创建一个 actor 的时候，负责创建 actor 的 worker 会构建一个 task，这个 task 叫做 actor creation task， actor creation task 会运行 actor 的构造函数。负责创建这个 actor 的 worker 会等待 actor creation task 运行所依赖的一些外部条件就绪了之后，会异步的注册这个 actor 到 GCS，对于 detached actors，注册的处理过程是同步的方式， 对于 non-detached actors 的注册处理过程是异步的方式，然后 GCS 会负责调度 actor creation task 去创建 actor， 就好像 GCS 是这个 actor 的调用者一样。
- 同时，就像方法调用一样，创建的 actor 会返回一个 actor handle 给调用者。这个调用者就可以用这个 actor handle 去调用 actor 的方法，也可以理解成发送 Tasks，因为 actor 的成员方法也是一

个 remote 的分布式的 task。 当一个 task 的参数是用的 actor handle，这个 task 不会立即调度，而是要等到“actor creation task”这个任务完成。

- actor 的 tasks 执行和一般的 task 是类似的，这些 actor tasks 通过 gRPC 被直接提交给 actor process，但是这些 task 不会直接运行，除非它的依赖都已经是 ready 的。但是 actor 的 task 和一般的 task 有两个主要的不同的地方：

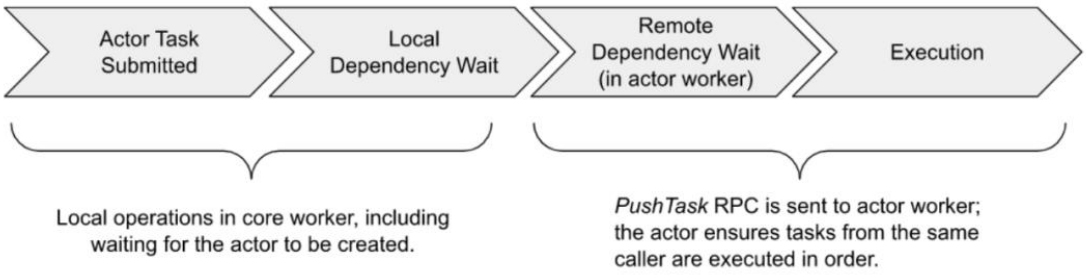
    1. 默认情况下，actor task 执行是不需要资源申请的，因为 actor 在创建的时候已经申请了资源。 actor task 就是 actor 的成员方法被调用的创建出来的方法。
    2. 对于每一个 actor 的调用者来说，同一个调用者调用的 actor 的方法时，也就是 task 是按照调用顺序执行的， 这样就可以保证 actor 有状态的特性，这种模式的 actor 是 Synchronous, Single-Threaded Actor。
    3. 还有一种是 Asynchronous 或 Threaded Actor，这种 actor 的方法是可以异步，并且无顺序要求的。

- 默认情况下，actors 在失败的时候是不会重启的，但是为了支持故障恢复，可以设置相关参数让其支持失败之后重启的能力， 这里可以设置 max_restarts 和 max_task_retries 选项在 ray.remote() 和 .options() 里面。



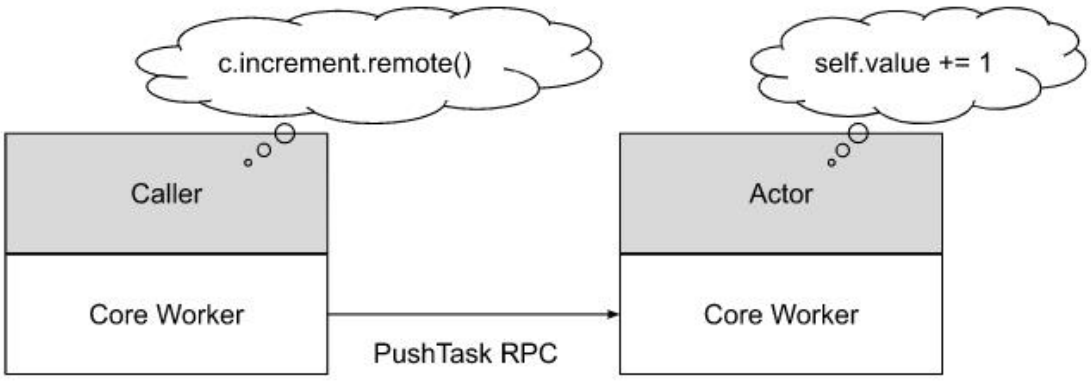
Image

2. actor task 的执行：



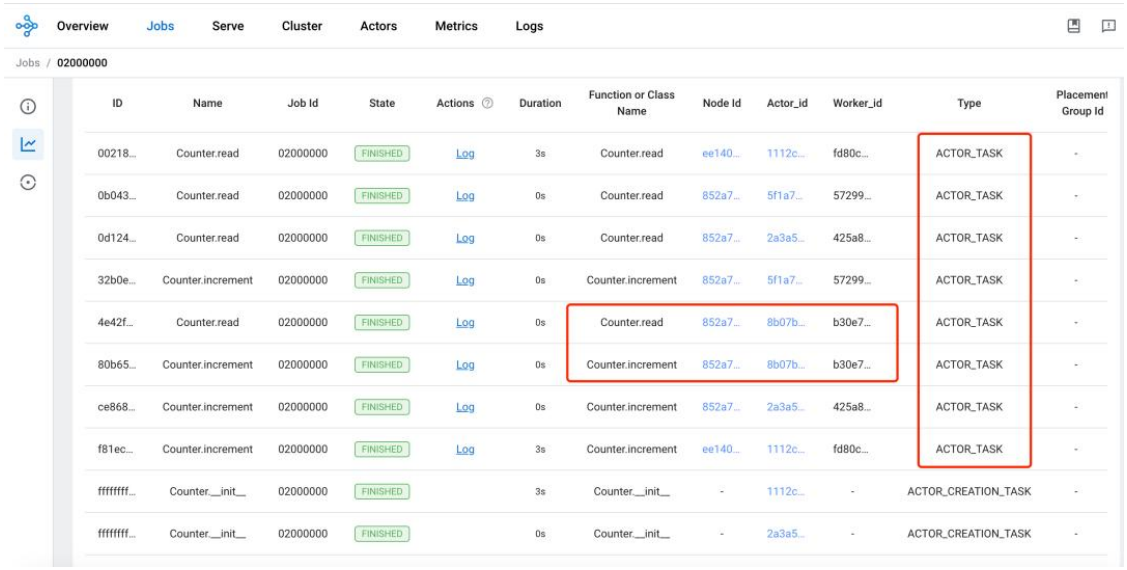
Image

- actor task 指的是 actor 的成员方法被调用的时候所创建的 task。一个 actor 可能有无数个调用者， 一个 actor handle 就代表一个调用者，同时 actor handle 中包含了 actor 的 RPC 地址， 这样调用者就可以连接到 actor 进行提交 actor task 给 actor 对应的 worker process。



Image

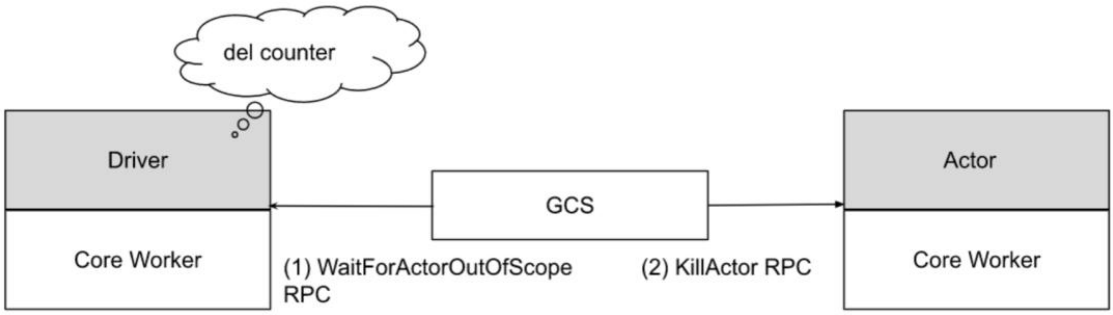
- 一个 actor 可以并发处理很多的调用，上图仅仅显示了一次调用。



ID	Name	Job Id	State	Actions	Duration	Function or Class Name	Node Id	Actor_id	Worker_id	Type	Placement Group Id
00218...	Counter.read	02000000	FINISHED	Log	3s	Counter.read	ee140...	1112c...	fd80c...	ACTOR_TASK	-
0b043...	Counter.read	02000000	FINISHED	Log	0s	Counter.read	852a7...	5f1a7...	57299...	ACTOR_TASK	-
0d124...	Counter.read	02000000	FINISHED	Log	0s	Counter.read	852a7...	2a3a5...	425a8...	ACTOR_TASK	-
32b0e...	Counter.increment	02000000	FINISHED	Log	0s	Counter.increment	852a7...	5f1a7...	57299...	ACTOR_TASK	-
4e42f...	Counter.read	02000000	FINISHED	Log	0s	Counter.read	852a7...	8b07b...	b30e7...	ACTOR_TASK	-
80b65...	Counter.increment	02000000	FINISHED	Log	0s	Counter.increment	852a7...	8b07b...	b30e7...	ACTOR_TASK	-
ce868...	Counter.increment	02000000	FINISHED	Log	0s	Counter.increment	852a7...	2a3a5...	425a8...	ACTOR_TASK	-
f81ec...	Counter.increment	02000000	FINISHED	Log	3s	Counter.increment	ee140...	1112c...	fd80c...	ACTOR_TASK	-
ffffffff...	Counter.__init__	02000000	FINISHED		3s	Counter.__init__	-	1112c...	-	ACTOR_CREATION_TASK	-
ffffffff...	Counter.__init__	02000000	FINISHED		0s	Counter.__init__	-	2a3a5...	-	ACTOR_CREATION_TASK	-

Image

3. actor dead：



Actor termination is also through the GCS.

Image

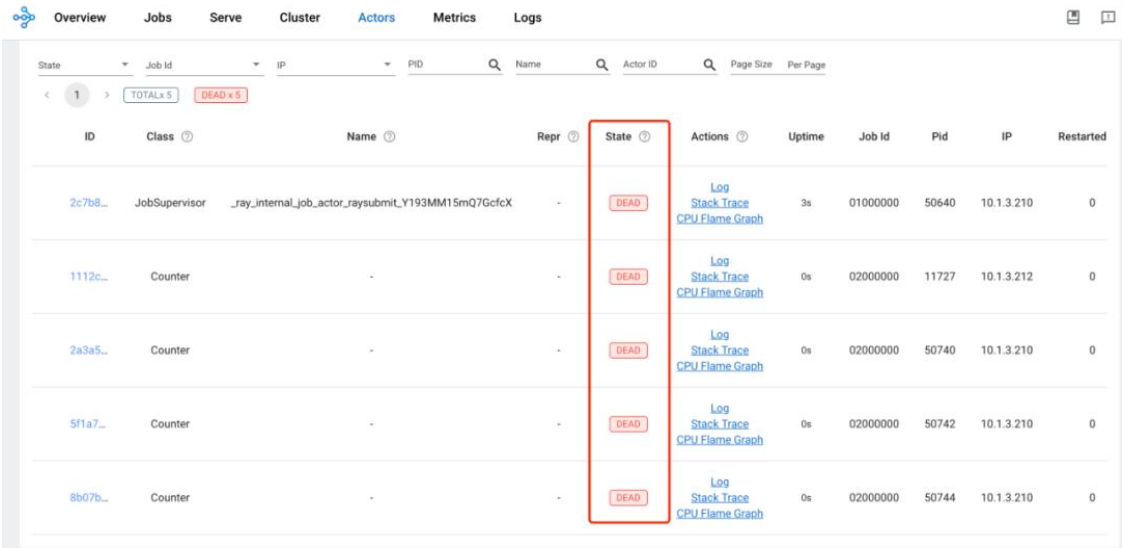
- actor 可能是 detached 类型的也可能是 non-detached 类型，detached 是同步的场景，non-detached 是异步的场景， 在使用中推荐使用 non-detached 的类型，同时也是默认的方式，当一个 actor 的所有的 actor handle 超出范围了（通过引用计数的方式来跟踪）， 这个时候，actor 的创建者会通知 GCS，然后 GCS 发送一个 KillActor RPC 给对应的 actor，之后 actor 将会退出它的进程， actor 可以被自动的垃圾回收。或者当 GCS 检测到 actor 的创建者退出了，也会中断对应的 actor，actor 可以被自动的垃圾回收。 或者是 actor 对应的 job 退出了， 也会中断对应的 actor，actor 也可以被自动地垃圾回收。不管是哪种情况，提交给 actor 的， 但是还没有执行的 tasks 就会报 RayActorError 错误。而对于 detached 类型的 actor，在不再需要的时候，是需要应用自己手动的方式去删除的。

- actor 在运行期间，也有可能遇到不可预期的错误导致退出，那提交给 actor 的 task 也是会报 RayActorError 错误。

- Ray 中支持设置 actor 失败之后的重启次数，如果设置了 max_restarts，那在 actor 的 owner 还存活的时候， GCS 会尝试通过 actor creation task 的方式，去重启 actor。在这个过程中，所有想要调用 actor 的客户端就会将要调用的 tasks 缓存起

来，在 actor 重启成功了之后再提交过去，如果 actor 重启固定次数之后，还是没有启动好， 那缓存起来的，还没有运行的 actor task 就会全部被标记成失败。

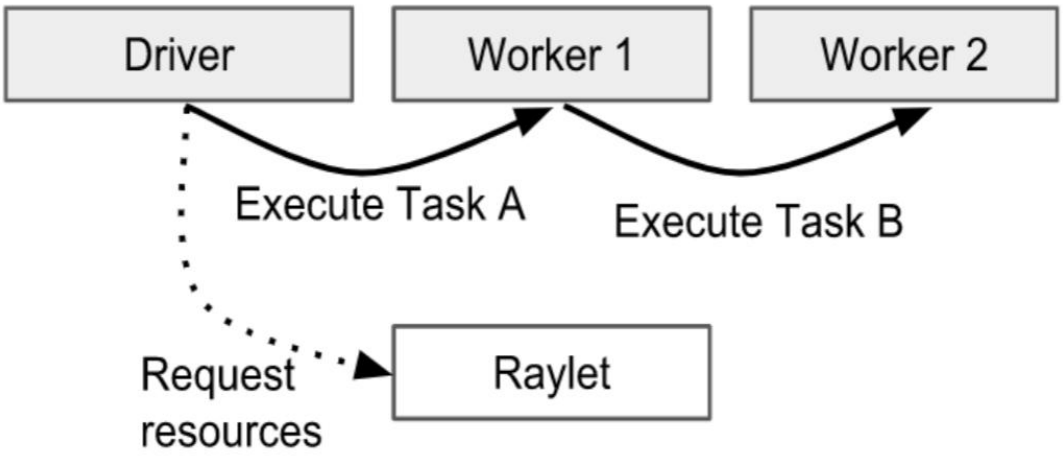
- 同时在 actor 重启成功之后，也有一个参数 max_restarts 来控制 actor task 的重启次数。



Image

Task 介绍

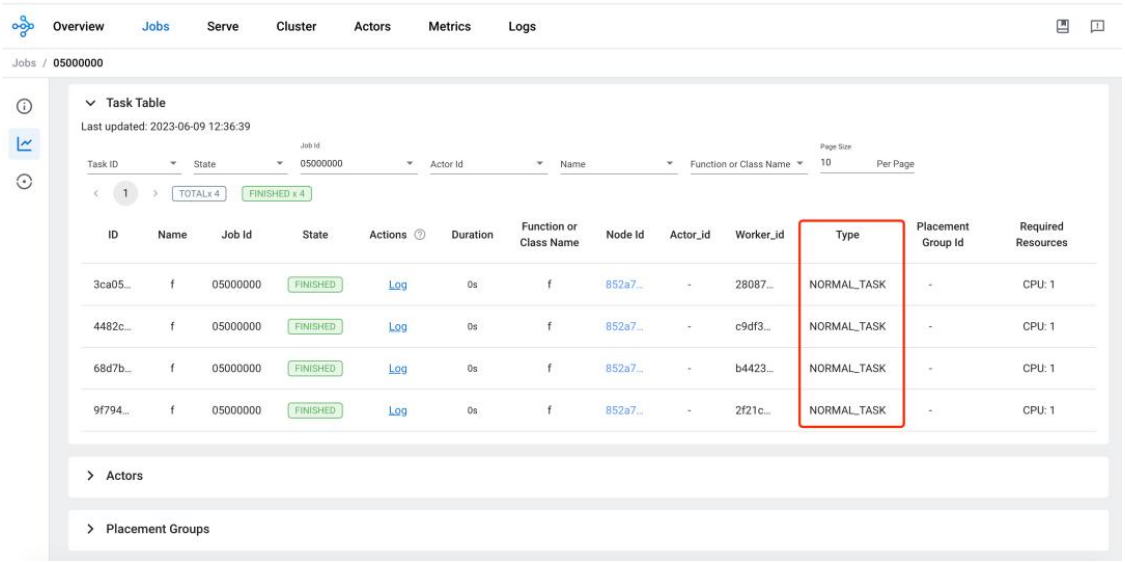
1. task 的生命周期：



Image

- 这里的 task 指的是一般的 task（NORMAL_TASK）。task 的持有者会负责保证一个被提交的 task 的执行，这里的 task 的持有者可以理解成调用 function 的某一个 process，这个 process 也是 function 返回值的持有者。task 的持有者会去负责和 raylet 通信去申请执行 task 需要的资源，这里为什么是和 raylet 通信，因为 raylet 中包含了分布式调度器。上图中 Driver 是 task A 的持有者，Worker1 是 task B 的持有者，所以 Driver 会负责申请 task A 执行需要的资源，而 Worker1 会负责 task B 执行需要的资源。
- 在申请资源之前，首先需要解决的 task 的依赖，才会去分布式调度器去申请资源。这里的 task 的依赖主要指的是 task（function）的参数依赖，因为参数可以是其它的 task 的返回值。分布式调度器会尝试去申请资源，同时去从分布式对象存储中去获取 task 的依赖（ObjectRef） 到本地节点，一旦资源和依赖都就绪了，分布式调度器会同意资源的申请，然后返回一个可以执行这个 task 的 worker 的地址给到 task 的持有者。
- 然后 task 的持有者就会通过 gRPC 的方式，去提交这个 task 到分布式调度器返回的 worker 中去运行。在 task 执行成功之后，负责执行 task 的 worker 需要保存 task 的返回值。如果 task 的

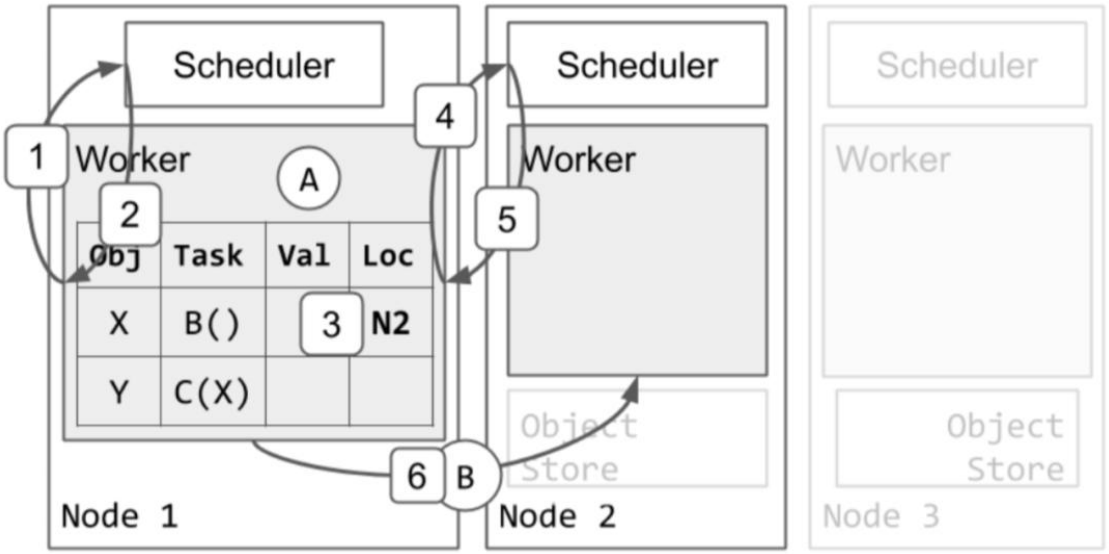
返回值很小，worker 就直接将返回值返回给 task 的持有者， 如果 task 的返回值比较大，那就存到分布式对象存储中，返回给 task 的持有者一个存放的位置。



Image

2. 分布式 task 调度：

下图中，task A 已经在 worker1 中运行，task B 和 task C 已经被提交到了 worker1，所以在 worker1 的本地的关系表中已经包含了 X 和 Y 对象， X 代表 task B，Y 代表了 task C。首先来看看怎么样来调度 task B。

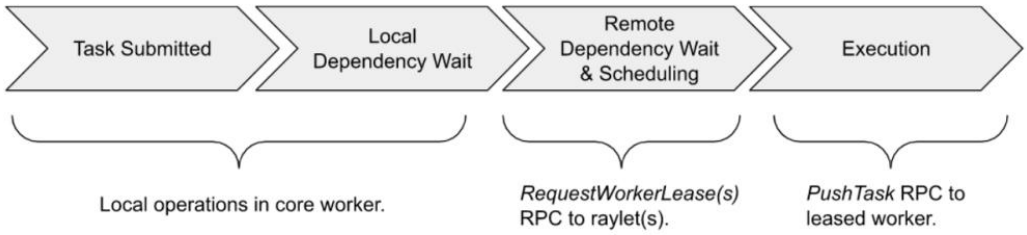


Image

- worker1 去询问本地的 raylet 中的调度器去给 task B 去申请资源。
- 本地调度器告诉 worker1 去尝试 node2 去请求调度。
- worker1 更新了本地的关系表去表示 task B 现在是阻塞在 node2 了。
- worker1 去询问 node2 的调度器，去为 task B 申请资源。
- node2 的调度器同意了 worker1 的资源请求，同时返回了 worker2 的地址给了 worker1。只要 task B 还在占用 worker2 去运行，node2 的调度器会保证没有其它的 tasks 会被再指定到 worker2 上去。
- 之后，worker1 就发送了 task B 到 worker2 去执行了。

3. Task 的执行：

Task execution



The scheduling workflow of a normal Ray task.

Image

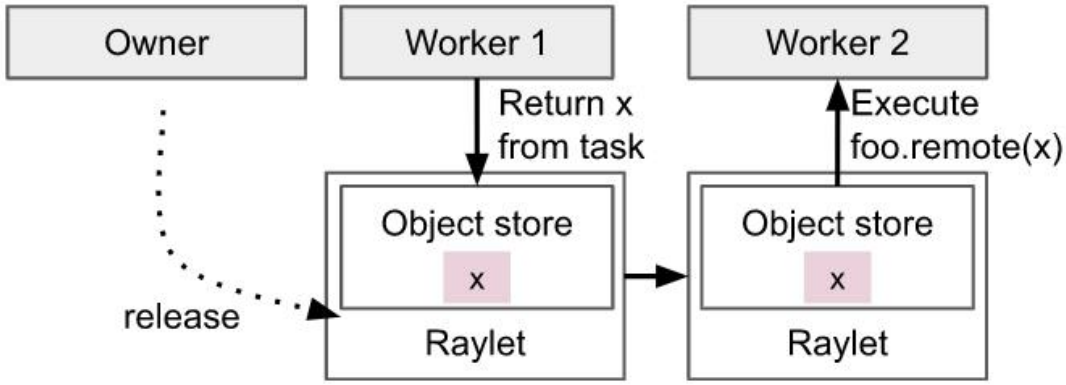
- 依赖解决：task 的调用者在去分布式调度器去为 task 申请资源之前，会等待所有 task 运行所需要的依赖都就绪了。 很多时候，任务的调用者也是其它任务的持有者，比如 foo.remote(bar.remote())，在这个例子中，task 的调用者是执行 foo.remote 这个方法的进程，同时在 foo.remote()中又依赖了 bar.remote 这个 task 的返回值，而 bar.remote 这个 task 也是属于 foo.remote 调用者的，所以这个时候不会直接调度 foo 这个 task，而是会等 bar 这个 task 已经完成了， 才会去调用 foo 这个 task。在这个情况下，bar 这个 task 的返回值会被保存在本地的进程中，直接给到 foo 这个 task 作为参数进行运行 foo 这个 task。如果 foo 这个 task 依赖的参数是不在本地的，比如是存在分布式对象存储中的大对象， 那在调度 foo 这个 task 之前会将 foo 依赖的大对象拉到本地来，然后再完成调度和运行 foo 这个 task。
- 资源满足：一个 task 的调用者通过发送资源请求给一个更合适的 raylet，也就是更合适的分布式调度器，那怎么选择哪个 raylet 是更合适的，从以下三个方面考虑：
    1. 通过数据本地化优先， 如果一个 task 依赖了很多对象参数，那就会现在那些节点上能包含了绝大多数参数对象的节

点， 因为不需要再去读取很多的参数对象了，因为大部分都在某一个节点了， 那就调度这个节点上，至于还有少部分没有的对象参数，只好从分布式对象存储中去读取到本地了。

2. 通过节点粘性，如果目标 raylet 是通过 NodeAffinitySchedulingStrategy 策略指定的，那就直接到对应的节点的 raylet 去。
3. 也是默认的一种方式，就是认为本地的 raylet 就是开始认为最合适的调度器，虽然最终可能不一定被调度在本地。

- 在选择好了合适的 raylet 之后，raylet 首先会将资源的请求放到队列中，如果这个 raylet 同意了资源请求， 就返回给调用者当前 raylet 它的本地的 worker 的地址。

Object 介绍



Image

Object 是一个应用的 value，这些 value 是由 task 返回的或通过 ray.put 创建的。 一个 Object 的 value 是不可变的值，一旦创建就不能修改，它可以被存储，也可以被引用，只要是在 Ray cluster 集群范围内。

task（function）是可以有返回值的，这个返回值是一个 Object，当在某一个 worker 中，去调用这个 function 的时候就会去提交创建这个 task 的请求，那这个 task 的返回值就属于这个 worker 的，这个 worker 就是 owner。 所以

在这个情况下，Object 的 owner 是通过提交 creating task 来创建初始 ObjectRef 的 worker。 同时这个 worker 会保存这个 Object 的引用计数。

ray.put(obj) 这种方式也是可以创建一个 Object，当在一个 worker 中，调用 ray.put 的时候， 所以在这种情况下，Object 的 owner 是通过调用 ray.put 来创建初始 ObjectRef 的 worker。 同时这个 worker 会保存这个 Object 的引用计数。

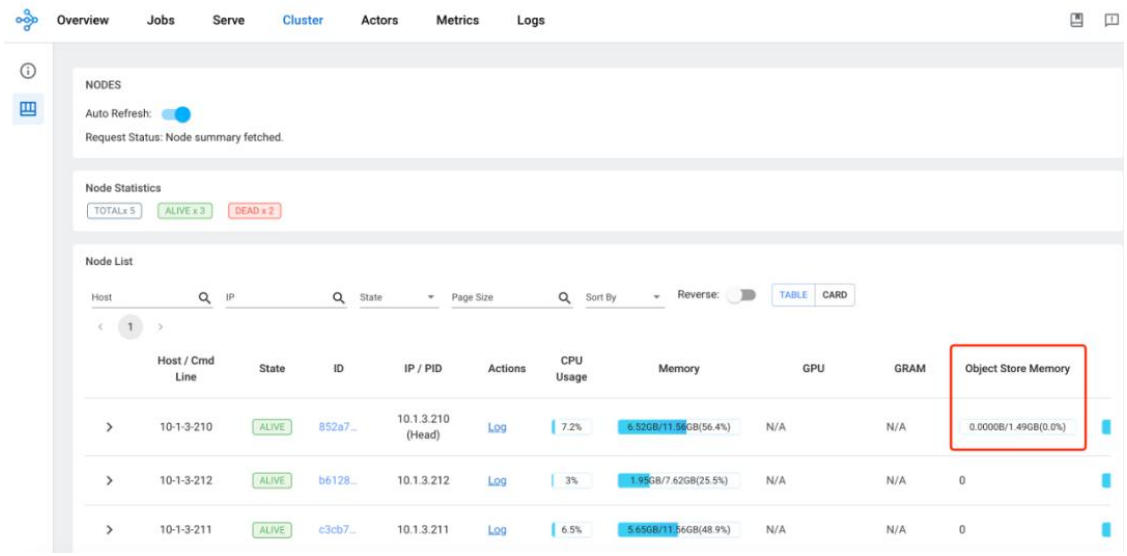
当前，worker 可以使用 ObjectRef 来引用对象。Object 的生命周期是由它的 owner 管理的。 Ray 会保证，如果 Object 的 owner 是存活的，那么 Object 最终可能被解析为它的 value （或者在 worker 失败的情况下抛出错误空）。如果 owner 挂了，尝试获取该 Object 的值将抛出异常，即使该 Object 仍然存在物理副本。

Object 是可以被保存在 owner's in-process memory store，也可以保存在分布式对象存储中。 in-process memory store 是被分配在 owner 的堆内存的，同时不会限制内存的大小，因为在 Ray 中， 只有小的 Object 才会保存在 worker 的 in-process memory store，如果小的 Object 太多了， 也会导致 worker 因 OOM 而被 kill。那些保存在分布式对象存储中的大的 Object 首先会被保存在共享的内存对象存储中。 共享的内存对象存储会强制一个用户指定容量的限制（默认是机器内存的 30%），同时在达到容量限制的时候会保存到本地磁盘。这样做是为了减少每个 Object 的内存占用和解析时间。

有两种方式可以获取到 ObjectRef 对应的 Object 的 value。1. 一种是通过在一个 ObjectRef(s) 上， 调用 ray.get 的方式；2. 另一种是传递一个 ObjectRef 作为 task 的参数，当 worker 去执行这个 task 的时候， 就会解析这个 ObjectRef，然后使用解析出来的 value 去替换 task 的参数。

对于一个小的 Object，可以直接从它的 owner 的 in-process store 中去解析出来。 如果是被保存在分布式对象中的大的 Object，就必须通过分布式协议去解析这个大的 Object 了。

当一个 task A 的返回值是一个大的 Object，它会被保存在运行这个 task A 的节点 A 的 local shared memory store 中，当另一台节点 B 的 task B 要用 taskA 的返回值作为 task B 的参数， 那首先会从调用 task B 的调用者，也就是 taskB 的 owner 中查找 object directory， 从中得知 task A 的返回值这个大的 Object 的保存为位置，这里可以简单理解为节点 A， 接下来，task B 所在的 raylet 是会去节点 A 的 raylet 中的分布式对象存储中去读取 task A 的返回值对象， 这时会使用到分布式协议，将需要的对象复制到本地。这里提一下 object directory ，在 object directory 就保存了 object 的位置信息，在之前的版本是保存在 GCS 中， 现在的版本是保存在 owners 中的（这里的 owner 就是 task 的调用者）。 当在节点 A 的 task C 要使用 task B 的返回值，这个时候就会直接从节点 A 的 local shared memory store 中直接去获取了。



Image

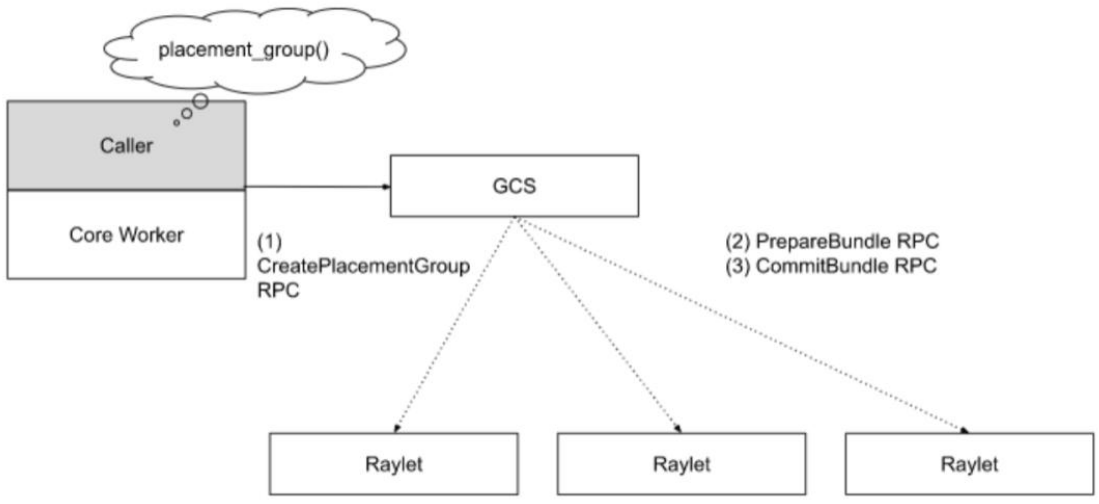
Placement Group 管理

描述：Placement Groups 主要是用来申请预留一些资源用于去运行 actors 或 tasks。actors 或者 tasks 可以在运行的时候指定 PlacementGroupSchedulingStrategy。Placement Groups 在预留资源的时候的表现形式是使用 Bundles 来表示，Bundles 是一组资源的集合，如 bundle 0 为 {“CPU”: 1}, bundle 1 为 {“CPU”: 1, “GPU”: 4}。 一个 Placement Group 中可

以包含多个 bundles，需要注意的是 bundle 是在 Placement Groups 的最小单元， 而且一个 bundle 可以被预留的前提是，资源的申请不能超过单台机器的资源剩余。举例来说，集群的机器中， 具有最充裕资源的机器剩下了 cpu :5 ，gpu :5，如果定义了一个 Placement Group，Placement Group 中定义了一个 bundle，这个 bundle 定义了 {“CPU”: 5, “GPU”: 9}，那这个 bundle 是不能被调度的， 也可以理解成这个 Placement Group 是无法被满足以及调度的。Placement Group 的资源预留也是可以跨机器预留的。 比如集群有 3 台机器，让每一台机器预留 {“CPU”: 1, “GPU”: 1}，然后 tasks 分布在这三个机器运行。通过 Placement Groups 的能力，可以让 actors 或者 tasks 以 gang scheduling（组调度，要么一起运行，要么都不要运行）的方式运行。

```
# 创建一个 actor，并使用 placement group。
actor = Actor.options(
    scheduling_strategy=PlacementGroupSchedulingStrategy(
        placement_group=pg,
    )
).remote()
```

Placement Group 创建：



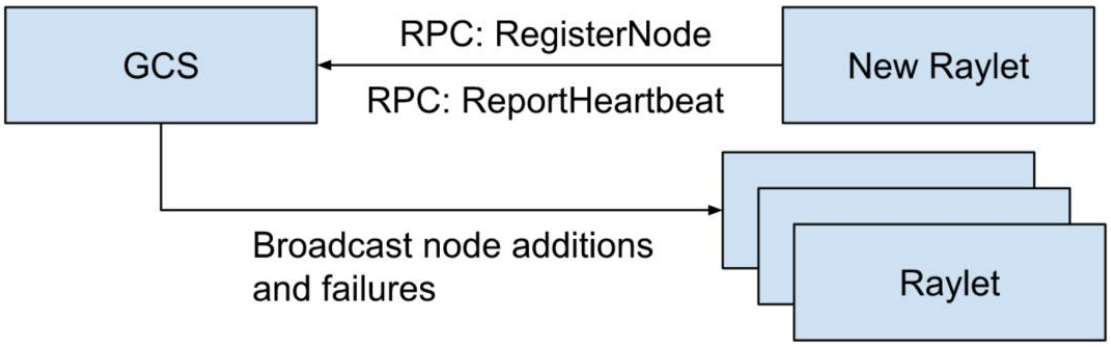
Image

当应用请求去创建一个 placement group 的时候，应用所在的 worker 会基于 RPC 去发送一个同步的请求给到 GCS， GCS 会先将请求保存起来，然后排队。因为 placement group 的资源申请可能会跨节点，所以 Ray 使用了两阶段提交保证原子性， 如果在这个过程中，有任何的 raylet 挂了，那这次创建过程就会回滚，然后这次创建的请求是重新进入队列排队。 不像 tasks、actors、objects、placement groups 是不需要引用计数的。因为 placement groups 的持有者是 job 或者 detached actor，当 own 这个 placement group 的 job 或者 detached actor 是 dead 状态了， 那这个 placement group 也就会被自动的删除了，所以不需要引用计数的方式，因为会自动的垃圾回收。 同时也支持用户直接调用 api 的方式去删除一个 placement group。placement group 也可以像 actor 一样有 detected， placement group 也可以有 detached placement groups，需要手动创建和手动删除的，举例来说一个 task 使用了 detached 类型的 placement group，那这个 placement group 不会随着 task 的销毁而自动被删除，detached placement groups 的生命周期是超越了它的 owner 的。当一个 placement group 被销毁了，所有使用这个 placement group 的 actors， tasks 都会被 kill，同时 placement group 预留的资源会释放出来。

在云原生的方案中，也有类似能力的组件可以平替 placement group，如 Kueue 等来完成组调度的能力。

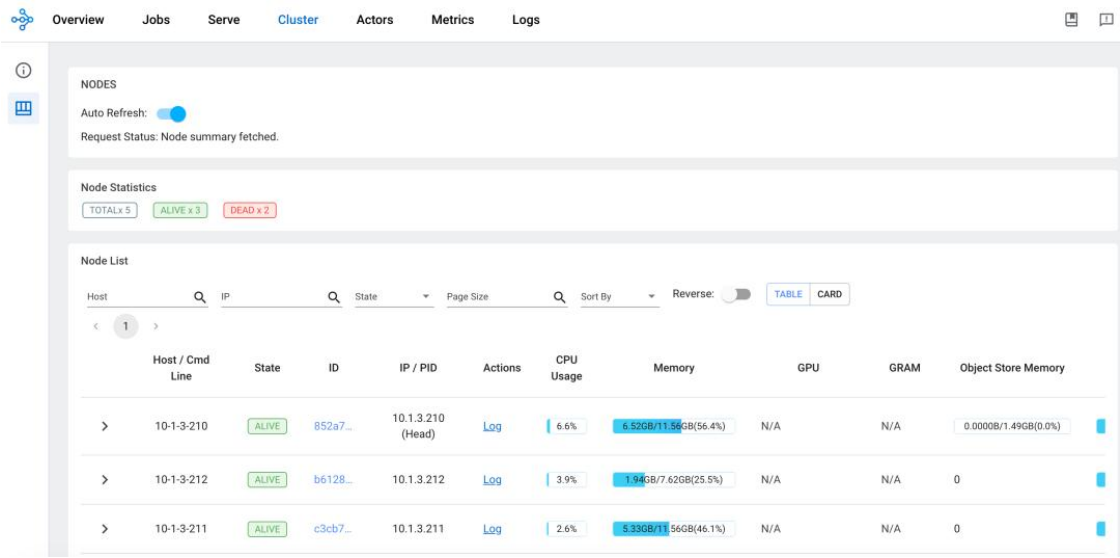
节点管理

当 raylet 启动的时候，它会注册到 GCS 中去，GCS 将注册上来的 raylet 的相关信息保存在存储中， 一旦 raylet 被成功注册到 GCS，这个注册的事情信息就会被广播给所有的 raylets。



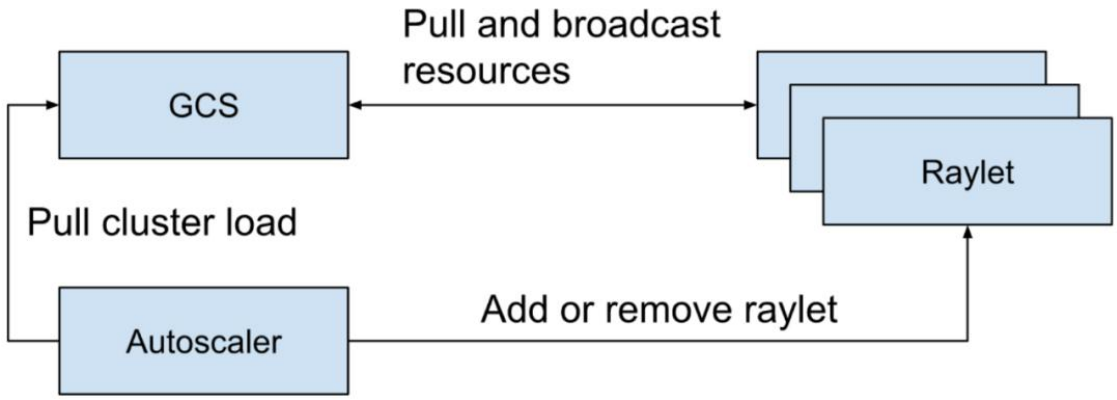
Image

当节点完成注册之后，GCS 会通过周期性的监控检测去监控这个节点的 raylet 的健康不健康。 同时 GCS 也会拉取这个 raylet 的资源视图，然后将这个信息也广播其它的 raylets。 如果这个 raylet 失败了，GCS 也会将这个失败的消息广播给其它的 raylets。一旦 raylet 收到了这些消息， 它们就开始清理相关的状态。同时 raylet 也会统计它所在节点的本地进程中哪些是挂掉的，去告知 GCS， 然后 GCS 再将这个信息再广播给其它的 raylets。这个有助于去清理系统中相关不一致的状态，去更好的帮助完成 tasks 的提交。



Image

资源管理



Image

GCS 负责保证所有的 raylets 有最新的集群资源使用的视图。这样分布式调度器就可以更有效的完成调度， 如果这个视图是不够新的，那就可能会给分布式调度器造成调度不准确的问题，导致 actors/tasks 不能很好的被调度到合适的位置。默认，GCS 会每隔 100ms 从所有的注册到自己的 raylets 去拉取最新的资源使用情况的视图，也会间隔 100ms 去广播给其它的 raylets。 同时 GCS 也会为 Autoscaler 组件提供当前集群的负载的数据，以帮助 Autoscaler 去判断什么时候给集群增加机器和从集群中移除机器。

总结

从上述的分析中，可以看到 Ray Core 中的整体架构、关键能力、基本的工作方式等。上述为 Ray Core 的部分分析， 还有一部分的分析会放到下一篇，同时在下一篇中会有实践的内容，可以更直观一点地认识 Ray Core。

参考链接：

- Ray 文档站
- Ray v2 架构

2022 年 DCE 5.0 攻坚语录集

Inspire People, Enable People! 致程序猿里超有文采的 panpan0000 同学。

*[panpan0000]: 技术圈的诗人，被代码耽误的文学家

!!! tip

这是一个没有硝烟的时间战场，是一场开疆拓土的光荣远征，是一次振奋人心的技术攀登，是一回一往无前的破釜沉舟 :material-sail-boat:

==扬起旗帜，吹响号角，夺下那个山头！For the DaoCloud 5th！==

- :leaves:{ .lg .middle } **3 月春风里**

在上海肆虐的疫情下， 每个人都在用自己的方式守护， 感谢每一位坚强隐忍、静待花开的你们。 所有的冬天都必将过去， 没有一个春天不会回来。

- :material-virus:{ .lg .middle } **4 月封控中**

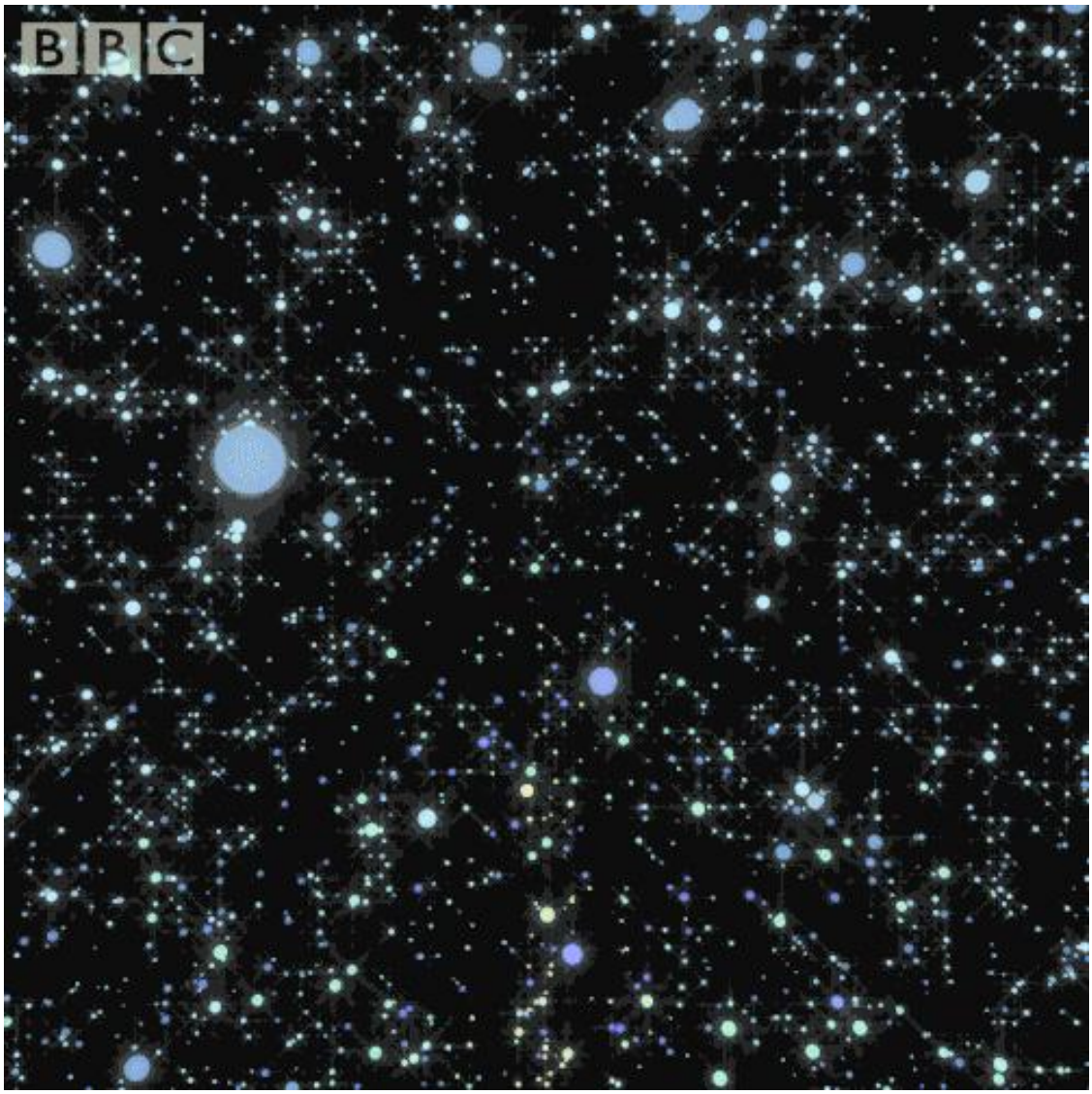
这个 4 月，就着泡面 ��，穿着睡衣 ��，头势飘逸 �� 敲着代码 ��，上班开会 ，下班抢菜 �� 做过的抗原比 bug 还多，git 的记录比你的乱发还长。 即使疫情阴霾，即使周边喧嚣，你还能保持阳光 �� 坚持，热爱 105℃ 的你。

- :desktop:{ .lg .middle } **5 月居家办公**

---

5 月万箭齐发，第五代开发已渐入佳境。 通过细细的网线，四地连成一心，这归功于：

```yaml title=“封城中的坚守”你的坚守: 这魔幻的上海两月，终会载入这历史性产品的历史性一页

你的风采: 一张张活起来的页面，联而贯之，梦想逐渐开始兑现

你的努力: 新的技术跃然纸上，新的能力呼之欲出，振奋人心 ```

- :hugging:{ .lg .middle } **6 月解封了**

---

那些波澜不惊的日复一日，会在某天让你看到坚持的意义。

当打开 demo-dev 站点发出赞叹“真漂亮啊，秒杀友商”的时候； 当“关于”的界面中，五彩斑斓地闪耀出你的名字的时候； 当你用一个个实打实的功能，回应几个月期待的时候； 当不经意种下自主开源种子，某天突然发现花香满园的时候。

我的伙伴：你所热爱的东西 / 真得有一天 / 会反过来拥抱你！

- :love_letter:{ .lg .middle } **7 月阳光下星辰里**

---

念念不忘，必有回响，功不唐捐，玉汝于成。

世界上大部分事，都没有太大意义；

真理和热爱除外。
```



动态星空

- :material-run-fast:{ .lg .middle } **8 月炎夏之际**

“人定胜天”一直是一个违心的误读；“人定”实指志向坚定、战胜自己。所以哪有什么天生如此，只有我们的天天坚持。

8 个月过去了，夺旗团队迎来更多的伙伴， 老同志们引领着前路，新同学们奋力跟上。 每个团队都有着厚积薄发的高光，也经历过韬光养晦的低

谷。 这 8 个月，时间回答了成长，成长回答梦想，而梦想决定了你现在的模样：

```yaml title="程序猿的日常" 你在半夜两点才合上的电脑 你在文档里字字谨慎的推敲 你在床上还辗转萦绕的思考 你在台风里仍惦记的 git 提交

一切热情和荣耀 在 9 月即将而来的实践面前 将赢得检验和回报 走吧！去发光 / 而不是被照亮！ ```

- :stars:{ .lg .middle } **9 月核酸之光**

---

夜深，不服输的你，还在痛苦的抗争， 犹如宇宙中漂浮的无力。

在技术的巨壑中前行，宛如广袤宇宙之中的渺小和孤独； 你只拥有模糊的漆黑世界和清晰的自己。

但只有如此黑暗之时，你才能发现星辰的濯濯生辉； 倘若不曾见过黑夜，那日出也不会如此明媚动人。

何去何从，信念使然…



*宇宙之光*

- :material-flower-tulip:{ .lg .middle } **10 月秋风起**
```

与其抱怨身处黑暗，不如提灯前行； 因为人生最重要的，不是你现在所处的位置，而是你面朝的方向。

董宇辉总说：“苦难，能给予我们坚持的力量”， 其实，力量的源泉还有对明天的期待。

秋风已深，凛冬将至。 但是我知道：你们心里 / 有一个 / 不可战胜的夏天。



那朵花

- :sunny:{ .lg .middle } **11 月我还行**

海明威说：

“生活总是让我们遍体鳞伤，但到后来，那些受伤的地方一定会变成我们最强壮的地方。”

在这个结痂上，愿你长出羽翼，去拥抱星河万里。

也许你未必 / 一直光芒万丈 / 但始终 / 温暖有光 /



move forward

- :fontawesome-solid-hourglass-end:{ .lg .middle } **12 月放开管控**

2022 年画上了句号。 这个句号，带着病毒带来的精神肉体的双重挑战，也带着对大疫三年后百废待兴、否极泰来的期许。

5.0 夺旗转眼又是一年了，你还记得一年前的自己么？ 淞沪路边被你日夜来回踏过的砖， 新江湾城里被你从绿看到黄再到秃的树， 多少次发誓明天再也不吃老乡鸡， 园区前将永远消失的核酸亭…

打开电脑，面对紫色悦动的登录界面，你也会眼眶湿润； 是为了末世电影一般的 2022， 也是为了这一年日夜熬过的 12 个里程碑。

> 面对不完美的世界，总有人选择挺身而出。 去奔赴，去热爱，去见证。 是你，带着跨越瘟疫与巨变的勇气， 像贺炜说的“请相信，世界上总有一些美好，值得我们全力以赴。”艰难困苦，玉汝于成！

!!! quote

生活总是让我们遍体鳞伤，但到后来，那些受伤的地方一定会变成我们最强壮的地方 ❤❤



测试分析

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

博客文章

本页汇总 DCE 5.0 及云原生技术相关的博客和公众号文章，默认按时间排序。

- 20230824 | Kubernetes 本地开发工具比较：Telepresence、Gefyra 和 mirrord

  随着 K8s 项目的成熟，K8s 本地开发工具已浮出水面， 这一类方案通过将本地运行的组件连接到 Kubernetes 集群来提升 Kubernetes 开发体验。 这样可以在云环境中快速测试新代码，避开了 Docker 化、CI 和部署这样的传统周期。

- 20230815 | Kubernetes 1.28 震撼发布

  太平洋时间 2023 年 8 月 15 日，Kubernetes 1.28 正式发布。此版本距离上版本发布时隔 4 个月，是 2023 年的第二个版本。

- 20230808 | 尝鲜：K8sGPT 赋能 Kubernetes

  K8sGPT 是一个非常新潮的云原生智能工具，它为 Kubernetes 云原生软件工程师（SRE）赋予了超能力。

- 20230612 | AIGC 利器 Ray 云原生探索之路 – Ray Core 下篇

  本文详细阐述 Ray 的各项能力和参数，还介绍了一些应用操作实践。

- 20230612 | AIGC 利器 Ray 云原生探索之路 – Ray Core 上篇

  本文介绍 Ray 分布式计算的底座：Ray Core 是怎样的一种分布式的计算框架。

- 20230609 | Istio 1.18 新鲜出炉：正式宣布 Ambient 模式

1.18 版本是 Istio 在 2023 年发布的第二个版本，也是首个支持 Ambient 模式的版本。

本次版本更新引入了许多新功能和变化，包括但不限于 Ambient 模式的 Istio、增强的 Kubernetes Gateway API、 针对未自动注册的虚拟机的健康检查支持、新增对 Metrics 的过期支持、增强的 istioctl analyze 等。

- 20230605 | 云原生联邦中间件—FedState 正式开源上线

  在云原生的场景下，有状态的服务，也有了很大的发展。在多云/联邦的环境下， 怎么设计和实现基于多集群、多数据中心以及混合云场景下，有状态服务的编排、调度、部署和自动化运维等能力？ 最新开源的 FedState 项目正是在尝试解决这样的场景问题。

- 20230601 | API Server Tracing 特性进阶至 Beta

  在 Kubernetes 中，API Server 是管理和调度所有集群资源的核心组件，它接收和处理来自各种客户端的请求， 并将它们转化为底层资源操作。因此，API Server 的稳定性和可观测性对 Kubernetes 的整体健康状态至关重要。

- 20230515 | 边缘场景玩转 seccomp 配置文件

  Security Profiles Operator (SPO) 是一个功能丰富的 Kubernetes Operator， 相比以往可以简化 seccomp、SELinux 和 AppArmor 配置文件的管理。 从头开始记录这些配置文件是该 Operator 的关键特性之一，这通常涉及与大型 CI/CD 系统集成。 是否能够在边缘场景中测试 Operator 的记录能力是 SPO 的最新开发工作之一， 非常有助于轻松玩转 seccomp 配置文件。

- 20230515 | 云原生混沌工程—Chaos Mesh 实战篇

  安装和使用 Chaos Mesh，了解 Chaos Mesh 的能力。

- 20230509 | 安装 Kubernetes 教程（KLTS 版）

以 DaoCloud 自主维护的 KLTS (Kubernetes Long Term Support) 为例， 简要介绍安装 Kubernetes 的准备工作、安装步骤。

- 20230508 | Cilium 基于 eBPF 实现 socket 加速

  随着云原生的不断发展，越来越多的应用部署在云上。其中有些应用对实时性要求非常严苛，这使得我们必须提升这些应用性能，达到更快的服务速度。 为了达到更快的服务速度，一个场景是：当两个互相调用的应用部署在同一个节点上的时候，每次请求和返回都需要经过 socket 层、TCP/IP 协议栈、数据链路层、物理层。 如果请求和返回绕过 TCP/IP 协议栈，直接在 socket 层将数据包重定向到对端 socket，那将大大减少发送数据包耗时，从而加快服务速度。 基于这个思路，eBPF 技术通过映射存储 socket 信息，利用帮助函数实现了将数据包重定向到对端 socket 层的能力。Cilium 正是基于 eBPF 这个能力，实现了 socket 层加速效果。

- 20230428 | Kubernetes 1.27：加快了 Pod 启动速度

  如何在大型集群中加快节点上的 Pod 启动？这是企业中集群管理员常常会面临的问题。 这篇博文重点介绍了从 kubelet 一侧加快 Pod 启动的方法。此方法不涉及通过 kube-apiserver 由 controller-manager 创建 Pod 所用的时间段， 也不包含 Pod 的调度时间或在其上执行 Webhook 的时间。

- 20230427 | 云原生监控 – VictoriaMetrics 之基础篇

  说到云原生监控方案，第一时间基本上都会想到 Prometheus+AlertManager+Grafana 的一套成熟解决方案。Prometheus 作为监控核心，具备强大的数据模型、高效率运作、丰富的监控能力、强大的查询语言 PromQL、 简单易用、管理方便等特点。但是 Prometheus 目前在高可用层面上做得还并不完美。为此，在开源社区中， 孕育出了许多替代、增强方案，VictoriaMetrics 属于其中较为优异

的一个，是一个快速、经济高效且可扩展的监控解决方案和时间序列数据库。

- 20230418 | Karmada Failover 详解

  多云时代，如何实现应用跨数据中心，跨可用区和跨集群高可用，成为我们探讨的新话题。 在单个集群中，如果集群发生故障，那么在集群中的所有应用将不可被访问。 是否有方式帮助我们在集群发生故障时，应用自动迁移到新的集群，保证应用持续的对外访问呢?

- 20230417 | CNCF 平台工程白皮书

  2022 年，“平台工程”这个概念很火热，也在 Gartner 的炒作周期曲线上。 还有言论说“DevOps 已死，平台工程才是未来“。开发者不愿意和基础设施打交道，企业发展又需要自己的基础设施。 “平台工程”统一这两个矛盾点，或者说“平台工程”是 DevOps 的下一站。

- 20230412 | 近两年功能增加最多！Kubernetes 1.27 正式发布

  Kubernetes 1.27 正式发布。此版本距离上版本发布时隔 4 个月，是 2023 年的第一个版本。 新版本中 release 团队跟踪了 **60 个 enhancements**，比之前版本都要多得多。

- 20230411 | Spiderpool：Calico 固定应用 IP 的新选择

  Spiderpool 是一个 Kubernetes 的 IPAM 插件项目，主要针对 Underlay 网络的 IP 地址管理需求而设计， 能够为任何兼容第三方 IPAM 插件的 CNI 项目所使用。

- 20230405 | 保姆式安装 DCE 5.0 社区版

  本文在 3 个节点的集群中完成了从 0 到 1 的 DCE 5.0 社区版安装， 包含了 K8s 集群、依赖项、网络、存储等细节及更多注意事项。

- 20230317 | 边缘原生应用准则白皮书

物联网边缘工作组（IOT Edge Working Group）一直在探索边缘原生的定义， 以及“云原生”和“边缘原生”之间的异同，并发布了《边缘原生应用准则白皮书》。

- 20230315 | 在 Linux 上安装单机 DCE 5.0 社区版

  讲述如何在一台 Linux 机器上使用 Docker 和 kind 在线安装 DCE 5.0 社区版。 这是一种极简安装方式，便于学习和体验，性能优于 macOS 单机版。

- 20230315 | 在 macOS 上安装单机 DCE 5.0 社区版

  使用 macOS 笔记本电脑创建单节点的 kind 集群，然后在线安装 DCE 5.0 社区版。 适合初学者体验和学习，不适合生产环境。

- 20230301 | 开源项目 KWOK 介绍

  是什么样的开源项目会在发布 5 个月内，就被 Apple、IBM、腾讯、华为纷纷使用?

  KWOK 是 Kubernetes WithOut Kubelet 的缩写，即没有 Kubelet 的 Kubernetes。 帮助你在几秒钟内搭建一个由数千个节点构成的集群，用少量资源模拟几千个真实的节点。

- 20230214 | DCE 5.0 社区版包含多少开源项目

  时常听闻客户、社区成员、贡献者、公司内部的售前、交付、项目组在询问“DCE 到底涉及了哪些开源项目？”这篇博文详细列出了社区版所包含的开源项目。

- 20230214 | 「DaoCloud 道客」与 Kubernetes

  讲述「DaoCloud 道客」如何借力 Kubernetes 打造新一代企业级云原生应用云平台 —— DaoCloud Enterprise 5.0，以及如何回馈开源社区，践行云原生信仰。

- 20230201 | 2023 年云原生预测

  基于 CNCF 发布的云原生报告畅谈 2023 年云原生领域的各项技术和趋势发展。

- 20230201 | DCE 5.0 攻坚语录集

  2022 年，上海，疫情肆虐，封城、管控、居家，程序猿们奔走在病毒 `` 的缝隙中，那一年是 DCE 5.0 攻坚的日子，也是每个中国人艰难的一年。

- 20221209 | K8s 1.26 正式发布

  Kubernetes 正式发布了主题为 Electrifying 的 v1.26。 作为 2022 年最后的一个版本，增加了很多新的功能，同时在稳定性上也得到显著提升，本文从多个角度介绍了 1.26 版本的更新。

- 20221130 | Karmada 资源解释器

  Karmada 已被越来越多的企业，应用到多云和混合云场景中。 在实际应用过程中，用户经常会遇到，通过 PropagationPolicy 来分发各种资源到成员集群的使用场景。 这就要求分发的资源类型，不仅仅要包含常见的 Kubernetes 原生或知名扩展资源，同时也需要能支持分发用户的自定义资源。 因此 Karmada 引入了内置解释器来解析常见的 Kubernetes 原生或知名扩展资源，同时又设计了自定义解释器来解释自定义资源的结构，并且在近期提出了可配置解释器方案。 对无论是自定义资源，还是常见的 Kubernetes 原生资源，都可以提供更加灵活可配置的自定义方法，来提取资源的指定信息，例如副本数、状态等。

- 20221125 | KubeCon 2022 北美站 | 精彩看点回顾

  2022 年 11 月落幕的云原生顶级会议 2022 KubeCon 北美站上，来自世界各地的云原生技术专家、产品或解决方案供应商和使用者带来了 300 多场精彩纷呈的演讲。 主题涵盖 Kubernetes、GitOps、可观测性、

eBPF 、网络、服务网格和安全等。 本文精心挑选了此次会议上的几个热点话题演讲，进行简单介绍，感受每一次演讲和讨论背后所蕴藏的云原生趋势。

- 20221123 | 走进车企的数字原生之路 | 论道回顾

  11 月 18 日，由「DaoCloud 道客」主办的「论道原生 | 云原生数字生态私享会 · 走进车企」成功举办。 本次活动从车企的具体案例出发，主要分享了云原生在汽车行业的应用与实践。让我们一起回顾一下本次活动的精彩内容。

- 20221115 | SpiderPool – 云原生容器网络 IPAM 插件

  SpiderPool 来源于容器网络落地实践的经验积累，是「Daocloud 道客」开源的原生容器网络 IPAM 插件（github：https://github.com/spidernet-io/spiderpool）， 主要用于搭配 Underlay CNI 实现容器云平台精细化的管理和分配 IP。

- 20221110 | 源于热爱，始于坚持，不忘初心 — 「DaoCloud 道客」八岁生日快乐！

  时光荏苒，岁月如梭。自 2014 年 11 月成立以来，在 DaoClouders 不懈的耕耘中，「DaoCloud 道客」已经走过了八个年头。 在 11 月 8 日下午，「DaoCloud 道客」全体船员为 「DaoCloud 道客」举办了生日会。让我们一起来看看「DaoCloud 道客」八岁生日会的盛况吧！

- 20221105 | DaoCloud 是 K8s 资深认证服务商

  DaoCloud 早在 2017 年就首次顺利通过了 Kubernetes 认证，是国内最早涉足并得到 CNCF 官方认可的服务商， 同时也是国内最早获得 Kubernetes Training Partner (KTP) 认证的厂商。 目前经 CNCF 官方认证可支持的 K8s 版本包括：v1.25, v1.24, v1.23, v1.20, v1.18, v1.15, v1.13, v1.9, v1.7

- 20221105 | 原生思维看金融数字化转型

  2022 年 11 月 5 日至 6 日，由西南财经大学、成都市地方金融监督管理局和成都市温江区人民政府联合主办的第五届国际金融科技论坛 2022 在蓉成功举办。 5 日下午以“数字经济赋能金融科技创新”为主题的论坛邀请「DaoCloud 道客」创始人兼首席执行官、云原生计算基金会大使陈齐彦先生做主题演讲。本文根据演讲内容整理。

- 20221104 | HwameiStor 入选 CNCF 全景图：生产可运维的云原生本地存储系统

  近日，CNCF（云原生计算基金会）[1] 发布了最新版的云原生全景图 [2]。「DaoCloud 道客」自主开源的云原生本地存储系统 HwameiStor，被收录在 CNCF 云原生全景图中的 RunTime（运行时）层的 Cloud Native Storage（云原生存储）象限，成为 CNCF 推荐的云原生本地存储项目。

- 20221028 | Kubean 集群生命周期管理

  Kubean 采用 Kubespray 作为底层技术依赖，一方面简化了集群部署的操作流程，降低了用户的使用门槛。 另一方面在 Kubespray 能力基础上增加了集群操作记录、离线版本记录等诸多新特性。 Kubean 还提供了界面化创建集群的能力（需要结合社区版 DCE 5.0 容器管理功能），让新手用户也能一键创建和管理集群。

- 20221028 | 激活医疗、人工智能的数字新动能 | 云原生底座

---

[1] CNCF：全称 Cloud Native Computing Foundation (云原生计算基金会)，隶属于 Linux 基金会，成立于 2015 年 12 月，是非营利性组织，致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术，普及云原生应用。

[2] 云原生全景图：由 CNCF 从 2016 年 12 月开始维护，汇总了社区成熟和使用范围较广、具有最佳实践的产品和方案，并加以分类，为企业构建云原生体系提供参考，在云生态研发、运维领域具有广泛影响力。

10 月 28 日，由「DaoCloud 道客」主办的「论道原生 | 云原生数字生态私享会 · 上海」成功举办。 本次活动从云原生思维和技术出发，分享了云原生在医疗和人工智能方面的应用与实践。让我们一起回顾一下本次活动的精彩内容吧。

- 20221026 | DCE 5.0 容器管理能力介绍

  说明 DCE 5.0 容器管理模块提供的能力。

- 20221018 | DCE 5.0 资源管理能力介绍

  说明 DCE 5.0 全局管理模块提供的能力。

- 20220925 | DCE 5.0 应用工作台能力介绍

  说明 DCE 5.0 应用工作台模块提供的能力。

- 20220914 | Merbridge 入选 eBPF 全景图

  花开自有期，绽放亦有时。在所有贡献者的共同努力和呵护下，Merbridge 的花苞徐徐绽放开来。 2022 年 4 月初，Merbridge 顺利入选 CNCF 云原生全景图（Cloud Native Landscape）， 进入 Orchestration & Management（编排与管理）层的 Service Mesh（服务网格）象限，成为 CNCF 推荐的云原生服务网格加速器。

- 20220909 | Clusterpedia 使用心得

  Kubernetes 带来的云原生技术革命已席卷全球，越来越多的业务应用运行在 Kubernetes 平台上， 随着业务规模的扩展，集群数量与日俱增，多集群内部资源管理和检索越来越复杂，多云管理面临巨大挑战。 在多集群时代，我们可以通过 Cluster-api 来批量创建管理集群，使用 Karmada/Clusternet 来分发部署应用。 不过我们貌似还是缺少了什么功能，我们要如何去统一地查看多个集群中的资源呢?

- 20220908 | 华为与「DaoCloud 道客」推出面向元宇宙的云边协同超融合一体机

  2022 年 9 月 2 日，在世界人工智能大会上，华为与上海道客网络科技有限公司联合推出面向元宇宙创新业务的“云边协同超融合一体机”， 将云原生能力下沉至边缘，提供实时的虚拟数字世界体验，实现真正的云边一体化元宇宙。

- 20220905 | 如何打好存储基础，筑就云原生应用基石？| 论道原生

  应用的云原生化极大地提升了自身的可用性、稳定性、扩展性、性能等核心能力，同时也深刻改变了应用的方方面面， 存储作为应用运行的基石，也不可避免地受到了冲击。云原生时代背景下，给存储带来了怎样的挑战，又该如何应对呢？ 第十一期论道原生，「DaoCloud 道客」携华为，分享云原生存储解决方案。

- 20220810 | Cluster API 检索从未如此简单

  Clusterpedia 是一个 CNCF 沙箱项目，用于跨集群复杂的资源检索。 名字源于 Wikipedia，寓意是打造多集群的百科全书，可以与多个集群同步资源， 并在与 Kubernetes OpenAPI 兼容的基础上，提供更强大的搜索功能，以帮助您快速、简便、有效地获取任何多集群资源。

- 20220808 | DCE 5.0 多云编排能力介绍

  说明 DCE 5.0 多云编排模块提供的能力。

- 20220708 | DCE 5.0 服务网格能力介绍

  说明 DCE 5.0 服务网格模块提供的能力。

- 20220622 | Clusterpedia 作为首个多云检索开源项目正式进入 CNCF 沙箱

2022 年 6 月 15 日，云原生计算基金会（CNCF）宣布 Clusterpedia 正式被纳入 CNCF 沙箱（Sandbox）项目。 Clusterpedia 是由「DaoCloud 道客」于 2021 年年底推出的开源项目，是一个通过加持 kubectl 就能完成多集群资源复杂检索的神器， 也是目前 CNCF 唯一专注于多云信息检索的项目，正被广泛使用。

- 20220611 | CloudTTY：下一代云原生开源 Cloud Shell

  CloudTTY 是一个云原生开源项目， 基于 kubernetes 之上，解决了一系列集群之上的“网页命令行”权限控制下的功能需求。

- 20220609 | KubeCon EU 热门云原生技术分享 | 精彩看点回顾

  在刚刚过去的年度最顶级的云原生旗舰会议 KubeCon + CloudNativeCon Europe 2022 上， 全球的云原生技术专家、产品或解决方案供应商和使用者，对云原生进行广泛充分地交流和讨论。 本文从云原生的外部融合、自身演进、内部相关功能特性这三个方面，分享一些大会上热门的云原生开源项目，一起走进云原生。

- 20220606 | Merbridge CNI 模式

  Merbridge CNI 模式的出现，旨在能够更好地适配服务网格的功能。 之前没有 CNI 模式时，Merbridge 能够做得事情比较有限。 其中最大的问题是不能适配注入 Istio 的 Sidecar Annotation，这就导致 Merbridge 无法排除某些端口或 IP 段的流量等。 同时，由于之前 Merbridge 只处理 Pod 内部的连接请求，这就导致，如果是外部发送到 Pod 的流量，Merbridge 将无法处理。

- 20220606 | DCE 5.0 研发背景

  简述新一代云原生操作系统 DaoCloud Enterprise 5.0 诞生的背景。

- 20220530 | 云原生布局开始加速，市场前景如何？收好这份指南

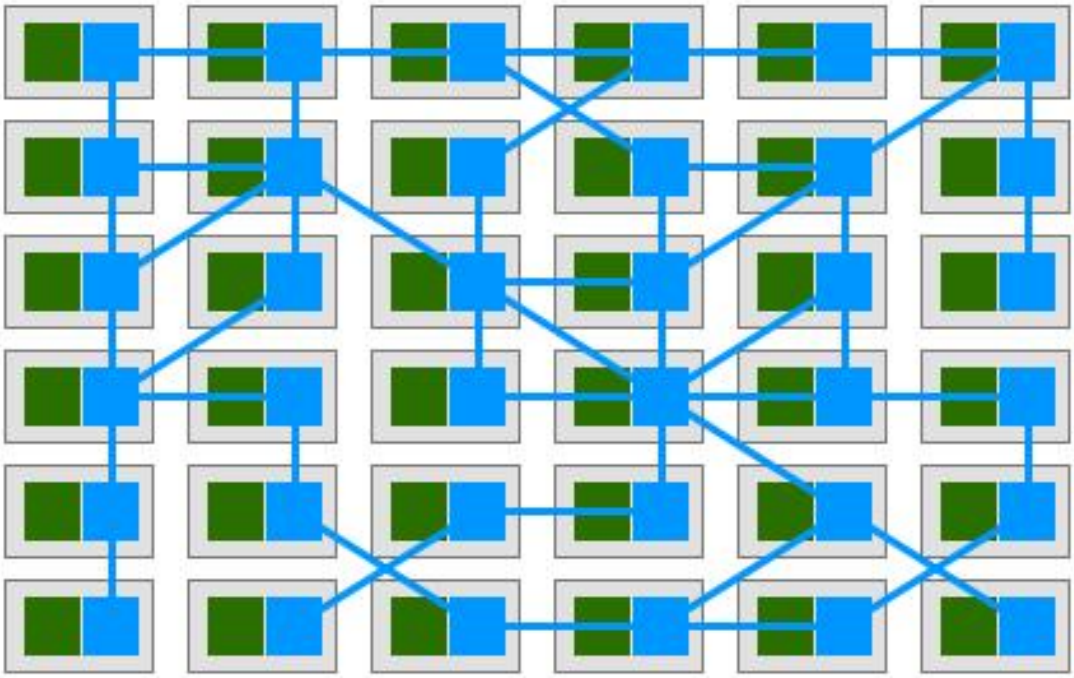
本文以 Infographic 模式图解说明了当前快速发展的云原生与云计算浪潮。

- 20220520 | 走进可观测性 | 论道原生

  2018 年起可观测性（Observability）被引入 IT 领域，CNCF-Landscape 组织创建了 Observability 的分组。 自此，可观测性逐渐取代传统的系统监控，从被动监控转向主动观测应用关联的各类数据，成为云原生领域的最热门话题之一。

5.0 服务网格能力介绍

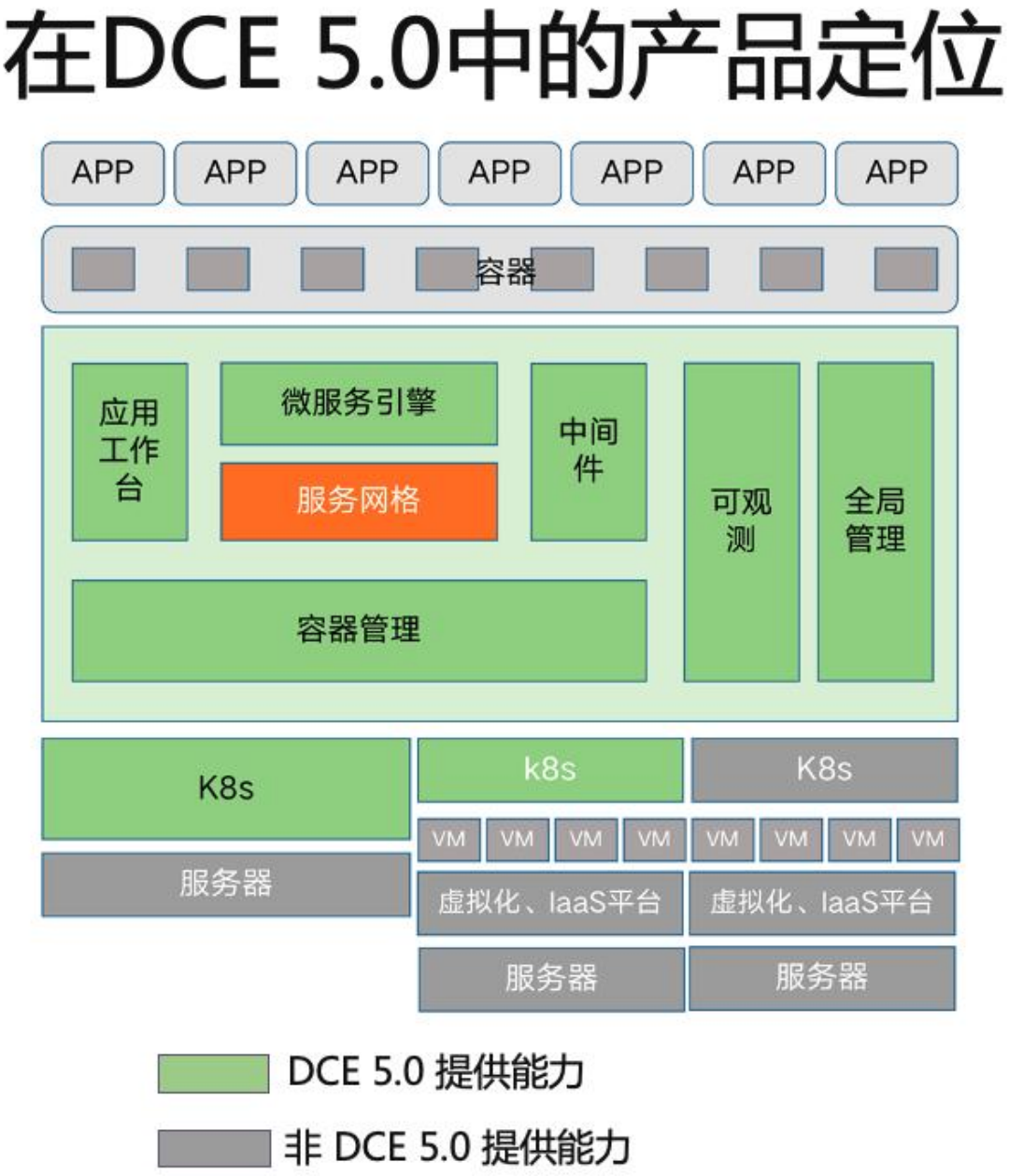
下图是一个网格状服务的正常模型，如何管理这样复杂的网格服务?



whatmesh

DaoCloud 服务网格是一种具备高性能、高易用性的全托管服务网格产品，通过提供完整的非侵入式的微服务治理解决方案，能够很好的支持多云多集群复杂环境的统一治理，以基础设施的方式为用户提供服务流量治理、安全性治理、服务流量监控、以及对传统微服务（SpringCloud、Dubbo）接入支持。并兼容社区原生 Istio 开源服务网格，提供原生 Istio 接入管理能力。较高的层次上，服务网格有助于降低服务治理的复杂性，并减轻开发运维团队的压力。

道客服务网格作为 DCE 5.0 产品的体系一员，无缝对接容器管理平台，可以为用户提供开箱即用的上手体验，并作为基础设施为微服务引擎提供容器微服务治理支持，方便用户通过单一平台对各类微服务系统做统一管理。



mspiderlocation

流量治理

流量治理是一个非常宽泛的话题，例如：

- 动态修改服务间访问的负载均衡策略，如根据某个请求特征做会话保持。
- 同一个服务有两个版本在线，将一部分流量切到某个版本上。

- 对服务进行保护，例如限制并发连接数、限制请求数、隔离故障服务实例等。
- 动态修改服务中的内容，或者模拟一个服务运行故障等。

在服务网格中实现这些服务治理功能时无须修改任何应用的代码。 服务网格可以为管理的服务提供非侵入的流量治理能力。 根据服务的协议，提供策略化、场景化的网络连接管理。 在应用拓扑上对选定服务的选定端口，根据需要配置各种不同的治理规则。

场景优势

重试

服务访问失败自动重试，从而提高总体访问成功率和质量。支持配置 HTTP 请求重试次数、重试超时时间和重试条件。

超时

服务访问超时自动处理，快速失败，从而避免资源锁定和请求卡顿。支持配置 HTTP 请求超时时间。

连接池

通过连接池管理，可以对四层协议配置 TCP 的最大连接数、连接超时时间、最大无响应次数、最短空闲时间和健康检查间隔， 对七层协议配置 HTTP 最大请求数、最大重试次数、最大等待请求数、每连接最大请求数、连接最大空闲时间，从而防止一个服务的失败级联影响到整个应用。

离群检测

通过离群检测配置实例被驱逐前的连续错误次数、驱逐间隔时长、最小驱逐时间、最大驱逐比例等参数，从而定期考察被访问的服务实例的工作情况，如果连续出现访问异常，则将服务实例标记为异常并进行隔离，在一段时间内不为其分配流量。过一段时间后，被隔离的服务实例会再次被解除隔离，尝试处理请求。如果还不正常，则被隔离更长的时间。从而实现异常服务实例的故障隔离和自动故障恢复。

负载均衡

配置各种负载均衡策略，如随机、轮询、最少连接，还可以配置一致性哈希将流量转发到特定的服务实例上。

HTTP 头域

灵活增加、修改和删除指定 HTTP 头域，包括将 HTTP 请求转发到目标服务之前对 Headers 的操作，以及将 HTTP 响应回复给客户端前，对 Headers 的操作，以非侵入方式管理请求内容。

故障注入

通过对选定的服务注入中断故障、延时故障来构造故障场景，无需修改代码即可进行故障测试。

端到端的透明安全

众所周知，将传统的单体应用拆分为一个个微服务固然带来了各种好处，包括更好的灵活性、可扩缩性、重用性，但微服务也同样面临着特殊的安全需求：

- 为了抵御中间人攻击，需要用到流量加密。
- 为了提供灵活的服务访问控制，需要用到 TLS 和细粒度访问策略。
- 为了决定哪些人在哪些时间可以做哪些事，需要用到审计工具。

面对这些需求服务网格提供全面的安全解决方案，包括身份验证策略、透明的 TLS 加密以及授权和审计工具。

场景优势

非侵入安全

服务网格是以一种安全基础设施的方式向用户提供透明的安全能力，让不涉及安全问题的代码安全运行，让不太懂安全的人可以开发和运维安全的服务，不用修改业务代码就能提供服务访问安全。 应用服务网格提供了一个透明的分布式安全层，并提供了底层安全的通信通道，管理服务通信的认证、授权和加密，提供 Pod 到

Pod、服务到服务的通信安全。 开发人员在这个安全基础设施层上只需专注于应用程序级别的安全性。

多集群安全

在多集群场景下服务网格提供了全局的服务访问安全。多个集群的网格共享一套根证书，给数据面的服务实例分发密钥和证书对，并定期替换密钥证书，根据需要撤销密钥证书。 在服务间访问时，网格的数据面代理就会代理本地服务和对端进行双向认证、通道加密。 这里的双向认证的服务双方可以来自两个不同的集群，从而做到跨集群的透明的端到端双向认证。

细粒度授权

在认证的基础上，就可以进行服务间的访问授权管理，可以控制某个服务，或者服务的一个特定接口进行授权管理。 如只开放给特定的一个 Namespace 下的服务，或者开放给某个特定的服务。 源服务和目标服务可以在不同的集群，甚至源服务的不同实例在不同的集群，目标服务的不同实例在不同的集群。

服务运行监控

运营容器化的基础设施带来了一系列新的挑战。我们需要增强容器、评估 API 端点的性能以及识别出基础设施中的有害部分。 服务网格可在不修改代码的情况下实现 API 增强，并且不会带来服务延迟。

服务网格为网格内的所有服务通信生成详细的遥测，这种遥测技术提供了服务行为的可观察性，允许运营商对其应用程序进行故障排除、维护和优化，而不会给服务开发人员带来任何额外负担。 通过服务网格，运营商可以全面了解被监控的服务如何与其他服务以及组件本身进行交互。

场景优势

非侵入监控数据采集

在复杂应用的场景下，服务间的访问拓扑、调用链、监控等都是对服务整体运行状况进行管理，以及服务访问异常时进行定位定界的必要手段。 服务网格技术的一

项重要能力就是以应用非侵入的方式提供这些监控数据的采集，用户只需关注自己的业务开发，无需额外关注监控数据的生成。

丰富性能监控能力

基于网格生成服务访问数据，集成各种不同的性能监控服务，提供跨集群智能的服务运行管理。包括跨集群的服务调用链、服务访问拓扑和服务运行健康状态。通过跨集群的全局视图来关联服务间的访问状况。

图形化用户交互

在服务网格中，为用户实现以下图形化的交互性功能：

1. 对服务网格下的网格资源创建、管理、设置能力，用户可以根据需要把集群资源分配至合理的网格中，实现聚合管理；
2. 为用户接入集群资源，或把用户已部署 ISTIO 治理组件的集群资源置于服务网格平台管控下；
3. 提供对网格下管控资源的工作空间绑定功能，通过工作空间实现网格资源的权限分享，方便各子产品间互动；
4. 以聚合管理方式，为用户提供命名空间、服务的统一查看、操作功能；
5. 针对命名空间、服务的流量治理功能，并提供表单和 YAML 两种创建、编辑形式，用户几乎可以使用原生 ISTIO 提供的所有流量治理功能（路由、分流、离群检测等）；
6. 针对命名空间、服务的安全治理功能，并提供表单和 YAML 两种创建、编辑形式，用户可以配合流量治理实现网格内零信任环境和访问授权功能；
7. 对网格内服务通信的流量拓扑查看功能和状态图表查看功能；流量拓扑会为用户提供各类网格内节点间通信状态、使用协议、性能参数等各项信息展示，用户可以通过搜索筛选功能迅速定位问题；状态图表可以从全局、服务、工作负载等多个纬度为用户提供系统内各状态信息；

了解服务网格

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

Linux 单机在线体验 DCE 5.0 社区版

本页说明如何通过 Docker 和 kind 在一台 Linux 机器上在线安装单机 DCE 5.0 社区版。

!!! tip

这是一种极简安装方式，便于学习和体验，性能优于 [macOS 单机版](230315-install-on-macos.md)。原文作者是 [panpan0000](https://github.com/panpan0000)。

准备工作

- 准备一台 Linux 机器，资源建议：CPU > 8 核、内存 > 12 GB、磁盘空间 > 100 GB
- 确保这台机器能够连通公网
- 操作系统：CentOS 7 或 Ubuntu 22.04

检查系统资源和联网情况：

```
set -e
if [ $(free -g|grep Mem | awk '{print $2}')                -lt 12 ]; then (echo "insufficient memory! (should >=12G)"; exit 1); fi
if [ $(grep 'processor' /proc/cpuinfo |sort |uniq |wc -l) -lt 8 ]; then (echo "insufficient CPU! (should >=8C)"; exit 1); fi
if [ $(df -m / |tail -n 1 | awk '{print $4}') -lt 30720 ]; then (echo "insufficient free disk space of root partition!(should >=30G)"; exit 1); fi
ping daocloud.io -c 1 &> /dev/null || ( echo "no connection to internet! abort." && exit 1; )
echo "precheck pass.."
```

预期输出如下：

```
precheck pass..
```

安装 Docker

如果主机上已有 Docker，并且版本高于 1.18，则可跳过此步骤。

=== “CentOS”

依次执行以下命令，大概需要 5 分钟左右：

```bash
set -e
if [ -x "$(command -v docker)" ];then
  echo "docker already installed : version = "$(docker -v);
  exit 0
fi
```

```bash
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
sudo yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
sudo sed -i 's+download.docker.com+mirrors.aliyun.com/docker-ce+' /etc/yum.repos.d/docker-ce.repo
sudo yum makecache fast
sudo yum -y install docker-ce
sudo service docker start
sudo systemctl enable docker
sudo yum install -y wget
```

=== “Ubuntu”

依次执行以下命令，大概需要 5 分钟左右：

```bash
set -e
```

```
if  [ -x "$(command -v docker )" ] ;then
   echo "docker already installed : version = "$(docker -v);
   exit 0
fi
```

```bash
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository --yes "deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get -y update
sudo apt-get -y install docker-ce
sudo apt-get -y install wget
sudo service docker start
sudo systemctl enable docker
```

!!! note

- 如果机器上已有 Podman，但是没有 Docker，仍需要安装 Docker。
- 这是因为一个已知问题：Podman 虽然可以启动 kind，但是会出现文件句柄不足的问题，并且重启会出现 IP 对不上的问题。
- Docker 安装问题请参阅 [Docker 官方安装说明](https://docs.docker.com/desktop/install/linux-install/)。

kind 集群

1. 下载 kind 的二进制文件包。

   ```
   curl -Lo ./kind https://qiniu-download-public.daocloud.io/kind/v0.17.0/kind-linux-amd64
   chmod +x ./kind
   ```

```
old_kind=$(which kind)
if [ -f "$old_kind" ]; then mv ./kind $old_kind; else mv ./kind /usr/bin/ki
nd ; fi
```

查看 kind 版本：

```
kind version
```

预期输出如下：

```
kind v0.17.0 go1.19.2 linux/amd64
```

2. 创建 kind_cluster.yaml 配置文件。暴露集群内的 32088 端口到 kind 对外的 8888 端口（可自行修改）。

```
cat > kind_cluster.yaml << EOF
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 32088
    hostPort: 8888
EOF
```

3. 通过 kind 创建一个名为 fire-kind-cluster 的 K8s 集群，以 k8s 1.25.3 为例。

```
kind create cluster --image docker.m.daocloud.io/kindest/node:v1.25.3 --
name=fire-kind-cluster --config=kind_cluster.yaml
```

此步骤大概需要 3~5 分钟，取决于镜像下载的网速。预期输出如下：

```
Creating cluster "fire-kind-cluster" ...
 √ Ensuring node image (docker.m.daocloud.io/kindest/node:v1.25.3) ��
```

```
 √ Preparing nodes ��
 √ Writing configuration ��
 √ Starting control-plane ��
 √ Installing CNI ��
 √ Installing StorageClass ��
Set kubectl context to "kind-fire-kind-cluster"
```

4. 查看新创建的 kind 集群。

   ```
   kind get clusters
   ```

   预期输出如下：

   ```
   fire-kind-cluster
   ```

安装 DCE 5.0 社区版

1. 安装依赖项，另请参阅依赖项安装说明

   ```
   curl -LO https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite.sh
   bash install_prerequisite.sh online community
   ```

2. 在主机下载 dce5-installer 二进制文件（也可以通过浏览器下载）

   ```
   export VERSION=v0.5.0
   curl -Lo ./dce5-installer  https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/dce5-installer-$VERSION
   chmod +x ./dce5-installer
   ```

3. 执行以下命令开始安装。

   ```
   myIP=$(ip -o route get 1.1.1.1 | cut -d " " -f 7)
   ./dce5-installer install-app -z -k $myIP:8888
   ```

   !!! note

- kind 集群仅支持 NodePort 模式。
- 如果是公有云机器，则只能手动指定公网 IP：`./dce5-installer install-app -z -k {公网 IP}:8888`

4. 安装完成后，命令行会提示安装成功。恭喜您！

   现在可以通过屏幕提示的 URL（默认为 https://${主机 IP}:8888），使用 **默认的账号和密码（admin/changeme）** 探索全新的 DCE 5.0 啦！



安装成功

「DaoCloud 道客」与 Kubernetes

为数字世界寻找全局最优解

挑战

「DaoCloud 道客」云原生领域的创新领导者，成立于 2014 年底，拥有自主知识产权的核心技术，致力于打造开放的云操作系统，为企业数字化转型赋能。这样的目标与使命决定了公司从创立之初就在云原生的世界里耕耘。与传统业务场景不同的是，云原生业务离不开容器，云操作系统更是依赖容器作为基础设施。因此「DaoCloud 道客」面临的首要挑战就是如何高效管理调度多个容器，如何保证容器间的正常通信。

此外，如今的云原生处在高速发展时期，各类云原生技术方案层出不穷且各有利弊，令人眼花缭乱。用户期望的并非解决眼前问题的局部最优解，而是要寻找全局最优解。如何整合这些独立的项目，取长补短，构建整体的云原生解决方案，这是「DaoCloud 道客」面临的又一挑战，也是云原生行业的发展难题。

解决方案

Kubernetes 作为容器编排的事实标准，无疑是首选的容器解决方案。「DaoCloud 道客」架构师兼开源团队负责人徐俊杰（Paco）表示“Kubernetes 是目前容器生态里面比较基础的一环，绝大多数服务都是基于 Kubernetes 部署的，应用绝大多数都是在 Kubernetes 集群中运行和管理。“

面对层出不穷的技术方案，「DaoCloud 道客」研发副总裁潘远航（Peter）认为，“在众多技术面前，坚持以 Kubernetes 为核心，整合周边最佳实践和先进技术，打造一个适合的平台和方案，才是寻找全局最优解的正确路径。”

影响

在拥抱云原生的过程中，「DaoCloud 道客」不断向 Kubernetes 等优秀的 CNCF 开源项目学习，逐渐形成了以 DaoCloud Enterprise 云原生应用云平台为

核心的产品架构。「DaoCloud 道客」坚持以 Kubernetes 等世界领先的云原生技术为支点，为军工、金融、制造、能源、政务、零售等垂直行业提供了前沿的云原生解决方案，为浦发银行、华泰证券、富国基金、上汽集团、海尔、复旦大学、屈臣氏、吉致汽车金融、国家电网等各行各业的优秀企业都量身定制了满意的数字化转型方案。

!!! note “”

“随着 DaoCloud Enterprise 越来越强大，客户覆盖面越来越广，有些客户需要使用 Kubernetes 而不是 Swarm 进行应用编排。Kubernetes 作为一套功能强大的应用编排系统，有着强大的社区支持，也受到很多大公司的青睐。我们作为提供商需要满足用户的需求。”

[刘齐均（Kebe）](https://github.com/kebe7jun),「DaoCloud 道客」服务网格专家

「DaoCloud 道客」成立的初衷就是帮助传统企业进行数字化转型，实现应用上云。公司成立之后发布的首秀产品 DaoCloud Enterprise 1.0 便是一款基于 Docker 的容器引擎平台，可以轻松打包构建镜像并运行容器。随着应用数量的增加，容器越来越多，如何协调和调度这些容器逐渐成为制约产品性能的主要瓶颈。DaoCloud Enterprise 2.0 开始采用 Docker Swarm 管理容器，但随着容器调度系统越来越复杂，Docker Swarm 也开始显得力不从心。

此时恰逢 Kubernetes 崭露头角，凭借多样的功能、稳定的性能、及时的社区支持、强大的兼容性等优势迅速发展成为容器编排的业界标准。Paco 表示“企业容器平台需要容器编排来规范化应用上云的过程。Kubernetes 在 2016 – 2017 年逐渐成为容器编排的事实标准，我们在 2017 年就开始同时支持 Docker Swarm 和 Kubernetes 了。”

经过一系列的评估，2017 年发布的 DaoCloud Enterprise 2.8 版本开始正式采用 Kubernetes（v1.6.7）作为容器编排工具。此后，2018 年发布的 DaoCloud Enterprise 3.0 采用 Kubernetes 1.10 版本，2021 年发布的 DaoCloud

Enterprise 4.0 采用 Kubernetes 1.18 版本。2022 年发布的 DaoCloud Enterprise 5.0 支持 Kubernetes 1.23 至 1.26 版本。

六年时间里发布的四个主要版本一直都在坚定不移地使用 Kubernetes，这足以说明当时的选择是正确的。「DaoCloud 道客」用实际经验证明了 Kubernetes 是容器编排的最佳选择，也用自身行动证明了自己一直都是 Kubernetes 的忠实拥趸。

!!! note

“借力 K8S 的价值体系『自动化大于人工』，产研团队从 0 到 1 完成了从研发构建自动化，测试自动化，安全自动化，发布自动化保证了软件交付质量，其次实现了智能化协作沟通，包括产品需求及定义体系、产品多语言协作体系、产品缺陷修复协作体系、疑难杂症攻坚体系，极大的提升了产研同部门、跨部门的协作效率，这是我们走向世界一流基础设施软件产品的基石。”

叶挺，「DaoCloud 道客」产品创新副总裁

在 Kubernetes 的助力下，「DaoCloud 道客」 的产品性能更优，更具竞争力。「DaoCloud 道客」坚持以 Kubernetes 为核心，整合周边最佳实践和先进技术，打造出 DaoCloud Enterprise 云原生应用云平台，提供应用商店、应用交付、微服务治理、可观测性、数据服务、多云编排、信创异构、云边协同等能力。DaoCloud Enterprise 5.0 是集云原生技术大成的完全形态。

- 「DaoCloud 道客」为上海浦发银行部署 Kubernetes 平台后，应用部署效率提升 82%，交付周期从半年缩短到一个月，交易成功率达到 99.999%；
- 四川天府银行落地基于 Kubernetes 的云原生平台，将弹性响应时间由数小时大幅缩减到平均 2 分钟，产品迭代周期从两个月缩短为两周，应用上线时间缩短 76.76%；
- 为某合资车企搭建基于 Kubernetes 的云原生平台后，将其交付周期从两个月缩短到一两周，应用部署成功率提升 53%，应用上线效率提高 24

倍；为某跨国零售集团部署基于 Kubernetes 的多个云原生平台模块，为其减少了 46% 的应用部署问题，将监控定位效率提升 90% 以上；

- 为某大型综合类券商搭建统一的云原生 PaaS 平台，使其业务流程效率提升 30%，资源成本节约 35% 左右；
- 为富国基金打造基于 Kubernetes 的新一代云原生 PaaS 平台，将标准中间件部署时间从数小时缩短至数分钟，中间件运维能力提升 50%，容器化程度提升 60%，资源利用率提升 40%。

另一方面，「DaoCloud 道客」自身的产品研发工作也是基于 Kubernetes 进行的。公司基于 Kubernetes 部署了 Gitlab，形成了“Gitlab —> PR —> 自动化测试 —> 构建发布“的产品开发流程，显著提升了开发效率，减少了重复测试的工作量，实现了应用的自动发布。这样一来，大大节省了运维成本，技术人员可以为开发产品投入更多的时间与精力，打磨出更优秀的云原生产品。

!!! note “”

“我们的开发者很踊跃地贡献开源，沉淀技术实力，在 Kubernetes 和 Istio 社区都有越来越多的贡献。公司第五代产品走的也是开源路线，为云原生技术添砖加瓦，完善技术生态。”

徐俊杰 Paco,「DaoCloud 道客」架构师/ 开源 & AD 团队负责人

「DaoCloud 道客」深度参与贡献 Kubernetes 等多项云原生开源项目，在云原生开源社区中的参与度、贡献度持续增长。在过去一年里，「DaoCloud 道客」在 Kubernetes 的开源榜单累计贡献度位居全球第三（基于 Stackalytics 网站 2023/01/05 的数据）。

在 2022 年 8 月由 Kubernetes 官方组织的社区贡献者访谈活动中，接见了来自亚太地区的 4 位优秀贡献者，其中 Shiming Zhang 和 Paco Xu 都来自「DaoCloud 道客」，二人均是 SIG Node 的 Reviewer。此外在 2022 Kubecon 北美站上，「DaoCloud 道客」的 Kante Yin 荣获 Kubernetes 2022 年度贡献者奖。

此外，「DaoCloud 道客」也在坚持践行云原生信仰，持续回馈云原生社区，开源了 Clusterpedia、Kubean、CloudTTY、KLTS.io、Merbridge、HwameiStor、Spiderpool、Piraeus 等优秀项目，不断完善 Kubernetes 生态体系。其中：

- Clusterpedia 兼容 Kubernetes OpenAPI，实现了多集群资源的同步，提供了更强大的搜索功能，可以快速、轻松、有效地获取集群内所有资源信息。
- Kubean 支持快速创建 Kubernetes 集群以及其他厂商的集群。
- CloudTTY 是专为 Kubernetes 云原生环境打造的 Web 终端和 Cloud Shell Operator，可以通过一个 Web 页面随时随地管理 Kubernetes 集群。
- KLTS 为 Kubernetes 早期版本提供长期免费的维护支持。
- Piraeus 是适用于 Kubernetes 的高性能、高可用性、简单安全的存储解决方案。

「DaoCloud 道客」融合自身在各行各业的实战经验，持续贡献 Kubernetes 开源项目，致力于让以 Kubernetes 为代表的云原生技术更平稳、高效地落地到产品和生产实践中。

!!! note “”

“「DaoCloud 道客」作为首批 CNCF 官方认证的云原生技术培训伙伴，将持续开展赋能培训、项目指导等活动，携手伙伴，为客户导入云原生，共同打造云原生能力的最佳实践路径。”

郑松,「DaoCloud 道客」中国区技术总经理

「DaoCloud 道客」研发副总裁潘远航（Peter）认为“企业用户需要的是一个全局最优解，这个最优解可以理解为是涵盖多云编排、信创异构、应用交付、可观测性、云边协同、微服务治理、应用商店、数据服务等能力的最大公约数。”在如今的云原生生态体系里，这些功能都离不开 Kubernetes 作为底层的容器编排技

术。这就意味着「DaoCloud 道客」在寻找数字世界最优解的过程中也离不开 Kubernetes，未来的产品研发也将继续以 Kubernetes 为基础。

此外，「DaoCloud 道客」一直致力于 Kubernetes 的培训、推广活动。2017 年，公司凭借核心产品云原生应用云平台 DaoCloud Enterprise 成为全球首批通过 CNCF Kubernetes 兼容性认证的厂家。2018 年，公司成为 CNCF 认证的 Kubernetes 服务提供商，并成为全球首批获得 CNCF 官方认证的 Kubernetes 培训合作伙伴，全面拥抱 Kubernetes 技术生态。

2022 年 11 月 18 日，由 CNCF 和「DaoCloud 道客」、华为云、四川天府银行、 OPPO 联合发起的「Kubernetes Community Days 成都站」成功举办，聚集了来自云原生领域开源社区的最终用户、贡献者和技术专家，分享关于云原生的多行业实践、热门开源项目、社区贡献心得等丰富内容。未来，「DaoCloud 道客」将继续为 Kubernetes 贡献自己的力量，通过项目培训、社区贡献等活动不断扩大 Kubernetes 的影响力。

Kubernetes 正式发布 v1.26，稳定性显著提升



1.26 release

太平洋时间 2022 年 12 月 8 号 Kubernetes 正式发布了主题为 Electrifying 的 v1.26。

作为 2022 年最后的一个版本，增加了很多新的功能，同时在稳定性上也得到显著提升，我们将从以下多个角度来介绍 1.26 版本的更新。

更新概览：

- **Kube APIServer:** 作为 Kubernetes 请求的入口，本次更新增加了 4 个 KEP 新功能，并且在响应压缩上做了一些优化，另外还有 2 个功能也从 Alpha 升级到了 Beta。
- **节点:** 我们将与 kubelet 关系最为密切的更新放在这里，主要包括 4 个新增的 KEP 功能，并且有 4 个功能在该版本 GA。
- **存储:** 存储方面主要是增加了从其他命名空间（跨命名空间)的快照中分配卷的功能，并且有 2 个存储相关功能进入了 Beta 阶段，3 个功能正式 GA。
- **网络:** 主要是对 Kube Proxy 的更新，包括一个性能优化的 KEP，并且有 2 个功能进入 Beta，4 个功能正式 GA。

- **资源控制与调协:** 主要是针对在 kube-controller-manager 中相关资源 controller 的更新，同样也有 2 个新增的 KEP 功能，有 2 个功能由 Alpha 升级到 Beta，还有 1 个功能正式 GA。
- **调度器:** 主要新增了一个重要 KEP 功能 – PodSchedulingReadiness 用于控制 Pod 何时可以被调度器调度，并且有 1 个功能由 Alpha 升级到 Beta。
- **可观测性:** 在可观测性上主要是增加了一种新的暴露组件状态的机制 – Component Health SLIs，另外各组件中也增加了很多指标。
- **kubectl 命令，kubeadm 以及 client-go** 也有一些优化和 Bug Fix。
- 对于已经 GA 的功能，根据 Kubernetes 的版本迭代策略，**在 1.26 中也移除了 11 个 Feature Gates**，如果在组件命令中继续设置了这些 Feature Gate，那么组件就会无法正常启动。

下面，我们先来看一些比较重要的、对升级有影响的 API 废弃与更改。

API 废弃与更改

PR#110618 Kubelet 不再支持 v1alpha2 版本的 CRI，接入的容器运行时必须实现 v1 版本的容器运行时接口。

这意味着 Kubernetes v1.26 将不支持 containerd 1.5.x 及更早的版本；需要升级到 containerd 1.6.x 或更高版本后，才能将该节点的 kubelet 升级到 1.26。

PR#112306 flowcontrol.apiserver.k8s.io 增加 v1beta3 版本，并将 v1beta2 设为最优版本，在 1.27 中将设置 v1beta3 为最优版本。

PR#113336 CSIMigrationvSphere 功能已经 GA 并且该功能无法被关闭。

!!! tip

官方提示：如果你需要使用 Windows、XFS 或者 raw block 的话，不要升级到 Kubernetes v1.26。等到 vSphere CSI Driver 在 v2.7.x 后的版本中增加相关的支持，即可进行升级。

PR#113710 kube-controller-manager 命令中 --pod-eviction-timeoutflag 被废弃，并且和 --enable-taint-managerflag 一起在 1.27 被移除。

PR#112643 DynamicKubeletConfig 在 1.23 中废弃，kubelet 中的逻辑已经在 1.24 中移除，本次更新将 FeatureGateDynamicKubeletConfig 和 APIServer 内的逻辑移除。

PR#112120 Kube 组件中删除一些无效的 klog 相关的 flags。

Kubernetes APIServer

KEP-2799 Reduction of Secret-based Service Account Tokens

新增 Alpha Feature Gate —— LegacyServiceAccountTokenTracking 来控制是否开启该功能。

当开启 LegacyServiceAccountTokenTracking 时，基于 secret 的 sa token 会使用 label kubernetes.io/legacy-token-last-used 来记录最后的使用时间。

KEP-3488 CEL for Admission Control

相关 PR：PR#113314、 PR#113349、 PR#112994、 PR#112792、 PR#112926、 PR#112858

基于 Kubernetes v1.25 提供的 KEP-2876 CRD 验证表达式语言， 该功能在 admissionregistration.k8s.io/v1alpha1 下增加一个新的资源 —— ValidatingAdmissionPolicy，允许在不使用 Validation Webhook 时实现字段验证。

```
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: ValidatingAdmissionPolicy
metadata:
  name: "demo-policy.example.com"
Spec:
  failurePolicy: Fail
  matchConstraints:
```

```
    resourceRules:
    - apiGroups:   ["apps"]
      apiVersions: ["v1"]
      operations:  ["CREATE", "UPDATE"]
      resources:   ["deployments"]
  validations:
    - expression: "object.spec.replicas <= 5"
```

这会要求资源的 spec.replicas 字段小于等于 5。

新增 Alpha Feature Gate —— ValidatingAdmissionPolicy 来控制是否开启该功能

KEP-3352 Aggregated Discovery PR#113171

用户当前获取 Discovery API 只能去遍历请求 Group 和 Version API，而该功能将这些调用减少到只需要请求 /api 和 /apis 两个接口即可。

新增 Alpha Feature Gate —— AggregatedDiscoveryEndpoint 来控制是否开启该功能。

KEP-3325 Auth API to get self user attributes PR#111333

在 authentication/v1alpha1 组中新增资源 SelfSubjectReview 来提供给用户查询自己在 Kubernetes 中映射的用户信息。

并且新增了 kubectl alpha auth whoami 命令来方便查询。

kubectl alpha auth whoami -o yaml

```
apiVersion: authentication.k8s.io/v1alpha1
kind: SelfSubjectReview
status:
  userInfo:
    username: jane.doe
    uid: b79dbf30-0c6a-11ed-861d-0242ac120002
    groups:
    - students
```

```
  - teachers
  - system:authenticated
extra:
  skills:
  - reading
  - learning
  subjects:
  - math
  - sports
```

新增 Alpha Feature Gate —— APISelfSubjectAttributesReview 来控制是否开启该功能。

PR#112193 APIServer 增加 --aggregator-reject-forwarding-redirect flag， 用户可以设置为 false，来继续转发 AA (Aggregated API) Server 的重定向响应，默认为 true。

PR#113015 可以通过 --encryption-provider-config 文件指定自定义资源，这些自定义资源就可以在 etcd 中加密存储。

Response Compression

PR#112299 根据从数千个生产 Kubernetes 集群收集的负载测试和生产数据，社区观察到 Kubernetes APIServer 中的 gzip 压缩目前不是最优的。

ISSUE: kubernetes/kubernetes#112296

PR#112309 在 kubeconfig 增加 DisableCompression 字段，设置为 true 时，要求不再对响应进行压缩。

PR#112580 在 kubectl 中增加 --disable-compression flag，设置为 true 时，要求不再对响应进行压缩。

功能稳定性升级

Alpha -> Beta

Feature Gate	KEP
LegacyServiceAccountTokenNoAutoGeneration	KEP-2799 Reduction of Secret-based Service Account Tokens
APIServerIdentity	KEP-1965 kube-apiserver identity

节点（Kubelet）

KEP-3063 dynamic resource allocation PR#111023

新增 resource.k8s.io/v1alpha1 组并且在该 Group 下增加动态资源分配相关的资源：‘ResourceClaim’，‘ResourceClass’，‘ResourceClaimTemplate’，‘PodScheduling’。

新增 Alpha Feature Gate —— DynamicResourceAllocation 来控制是否开启该功能。

新的 API 比 Kubernetes 现有的设备插件（Device Plugins）功能更灵活，因为它允许 Pod 请求特殊类型的资源，这些资源可以在节点级、集群级或按照用户设置的其他模式提供。

同样 Pod 结构也增加了相应对动态资源分配的支持。

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: with-resource
    image: busybox
    command: ["sh", "-c", "set && mount && ls -la /dev/"]
    resources:
      claims:
      - name: resource
  resourceClaims:
  - name: resource
```

```
    source:
      resourceClaimName: shared-claim
      # resourceClaimTemplateName: test-inline-claim-template
```

KEP-3545 Improved multi-numa alignment in Topology Manager PR#112914

该功能通过对 TopologyManager 的优化来更好的处理 NUMA (Non-Uniform Memory Access) 节点。

在 kubelet config 和 kubectl 命令中分别增加一个新的可配置项 topologyManagerPolicyOptions 字段和 --topology-manager-policy-options flag 来设置 Topology Manager Policy 的额外配置。

并且新增三个 Alpha Feature Gate 来控制对 Topology Manager Policy 的配置：

- TopologyManagerPolicyOptions
- TopologyManagerPolicyAlphaOptions
- TopologyManagerPolicyBetaOptions

TopologyManagerPolicyOptions 控制是否支持 TopologyManagerPolicyOptions 功能，而另外两个分别用来控制是否可以设置 TopologyManagerPolicy 的 Alpha 和 Beta 级别的 Options。

当然使用 TopologyManager 功能还需要开启 TopologyManager Feature Gate，不过该 Feature Gate 已经处于 Beta 阶段，并不需要主动设置。

KEP-3386 Kubelet Evented PLEG for Better Performance PR#111384

该功能让 kubelet 在跟踪节点中 Pod 状态时，通过尽可能依赖容器运行时接口 (CRI) 的通知来减少定期轮训，这会减少 kubelet 对 CPU 的使用

新增 Alpha Feature Gate —— EventedPLEG 来控制是否开启该功能。

PR#86139 在过去版本中，在容器 preStop 和 postStart 生命周期回调中使用 httpGet 时， 即使将 schemes 设置为 HTTPS，也依然使用 http 来访问，并且设置请求中也不会应用用户设置的 Headers。

```
lifecycle:
  postStart:
    port: 443
    httpGet:
      scheme: HTTPS
    httpHeadlers:
    - name: HEADER
      value: VALUE
```

该 PR 修复了这些问题，并且当 https 访问异常时会回退回 http 请求，当发生回退时， 会为该 Pod 创建一个 LifecycleHTTPFallback 事件，并且更新 kubelet_lifecycle_handler_http_fallbacks_total 指标。

另外 **新增了一个 Alpha Feature Gate** – ConsistentHTTPGetHandlers，用户可以在 kubelet 中设置 --feature-gates=ConsistentHTTPGetHandlers=false 来关闭该回退行为。

KEP-3503 Windows 中允许指定 Pod 是否加入到节点的网络命名空间 PR#112961

新增 Alpha Feature Gate —— WindowsHostNetworking 来控制是否开启该功能。

PR#112414 允许将 /etc/resolv.conf 中多行选项合并为单行设置到 Pod 中。

```
options ndots:1 attempts:3
options ndots:1 attempts:3 ndots:5

->

options ndots:5 attempts:3
```

BUG FIX

PR#113041 修复了执行 kubectl exec 时，由于 lifecycle.preStop 导致 kubelet 由于容器名称重复而选择了错误容器的问题。

PR#108832 当容器设置了 limit.cpu，但是 requests.cpu 为“0”时，cgroups cpuShares 取最小值 2，而不是使用 limit.cpu。

PR#112184 kubelet 只设置 --cloud-provider 或者 --node-ip 时，会确保清除节点中无效的 annotation - alpha.kubernetes.io/provided-node-ip。

PR#113481 修复 kubectl logs --timestamps 查看日志时，出现混乱的随机时间戳的问题。

PR#112518 修复了启用 PodDisruptionConditions 特性门控时，Pod 在打了 NoExecute 污点的节点上继续运行的问题。

PR#112123 将 cpuCFSQuotaPeriod 的最小值从 1us 设置为 1ms，设置 1ms 以下的值将会验证失败。

相关 PR: PR#112077、 PR#111520、 PR#63437

功能稳定性升级

Beta -> GA

Feature Gate	KEP
CPUManager	KEP-3057 Graduate to CPUManager to GA
DevicePlugins	KEP-3573 Graduate DeviceManager to GA
KubeletCredentialProviders	KEP-2133 Kubelet Credential Provider
WindowsHostProcessContainers	KEP-1981 Support for Windows privileged containers

存储

KEP-3294 Provision volumes from cross-namespace snapshots

相关 PR: PR#113186，PR#kubernetes-csi/external-rpovisioner#805

在 Kubernetes 1.26 之前，通过 VolumeSnapshot，用户可以从快照中分配卷。但是它无法无法在 PersistentVolumeClaim 中绑定到其他名称空间的 VolumeSnapshots。

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
spec:
  storageClassName: csi-hostpath-sc
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  dataSourceRef:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: new-snapshot-demo
    namespace: prod
  volumeMode: Filesystem
```

该功能支持用户通过设置 PVC 中新增的字段 spec.dataSourceRef.namespace 来跨命名空间从快照中分配卷

新增 Alpha Feature Gate —— CrossNamespaceVolumeDataSource 来控制是否开启该功能

功能稳定性升级

Alpha -> Beta

Feature Gate	KEP
RetroactiveDefaultStorageClass	KEP-3333 Retroactive default StorageClass assignement
NodeOutOfServiceVolumeDetach	KEP-2268 Non-graceful node shutdown

Beta -> GA

Feature Gate	KEP
CSIMigrationAzureFile	KEP-625 In-tree storage plugin to CSI Driver Migration
CSIMigrationvSphere	KEP-1491 vSphere in-tree to CSI driver migration
DelegateFSGroupToCSIDriver	KEP-2317 Allow Kubernetes to supply pod's fsgroup to CSI driver on mount

网络

PR#110268 优化 kube-proxy 性能，它只发送在调用 iptables-restore 中更改的规则，而不是整个规则集

新增 Alpha Feature Gate —— MinimizeIPTablesRestore 来控制是否开启该功能。

PR#108250 kube-proxy 新增 flag—iptables-localhost-nodeports 来允许用户禁止通过 localhost 访问 NodePort 的 Service。

PR#111806 如果用户要求使用 ipvs，但是系统没有正确配置时，不再回退到 iptables 模式，而是返回错误。

PR#113363 如果 kube-proxy 检测到分配给 Node 的 pod.Spec.PodCIDRs 已经改变，那么它将重新启动。

PR#112133 移除已经废弃的“userspace”代理模式。

功能稳定性升级

Alpha -> Beta

Feature Gate	KEP
ProxyTerminatingEndpoints	KEP-1669 Proxy Terminating Endpoints
ExpandedDNSConfig	KEP-2595 Expanded DNS configuration

Beta -> GA

Feature Gate	KEP
MixedProtocolLBService	KEP-1435 Support of mixed protocols in Services with type=LoadBalancer
ServiceIPStaticSubrange	KEP-3070 Reserve Service IP Ranges For Dynamic and Static IP Allocation
ServiceInternalTrafficPolicy	KEP-2086 Service Internal Traffic Policy
EndpointSliceTerminatingCondition	KEP-1672 Tracking Terminating Endpoints

资源控制与协调

KEP-3017 PodHealthyPolicy for PodDisruptionBudget PR#113375

在 PodDisruptionBudget 资源中新增 spec.unhealthyPodEvictionPolicy 字段来控制不健康的 Pod 何时应该被考虑驱逐。

spec.unhealthyPodEvictionPolicy 字段当前可以设置的值有两个 —— IfHealthyBudget 和 AlwaysAllow

```
spec:
  minAvailable: 2
  selector:
```

```
    matchLabels:
      app: zookeeper
  unhealthyPodEvictionPolicy: IfHealthyBudget
```

新增 Alpha Feature Gate —— PDBUnhealthyPodEvictionPolicy 来控制是否开启该功能

KEP-3335 StatefulSet Start Ordinal PR#112744

StatefulSet 当前是从 0 开始对 Pod 进行编号。

该功能在 StatefulSet 中增加一个 spec.ordinals.start 字段来控制 Pod 的起始编号。

```
apiVersion: apps/v1
kind: StatefulSet
spec:
  ordinals:
    start: 1
```

新增 Alpha Feature Gate —— StatefulSetStartOrdinal 来控制是否开启该功能

PR#112011 如果多个 HPA 涉及到相同的 Pod，那么会停止工作， 并设置 HPA 的 ScalingAction Condition 为 False， Reason 为 AmbiguousSelector，该 PR 还包括多个 HPA 指向同一 Deployment 的情况。

功能稳定性升级

Alpha -> Beta

Feature Gate	KEP
JobPodFailurePolicy	KEP-3329 Retriable and non-retriable Pod failures for Jobs
PodDisruptionConditions	KEP-3329 Retriable and non-retriable Pod failures for Jobs

Beta -> GA

Feature Gate	KEP
JobTrackingWithFinalizers	KEP-2307 Job tracking without lingering Pods

Pod 调度

KEP-3521 Pod Scheduling Readiness

相关 PR: PR#113275， PR#113274， PR#113442

并不是 Pending 状态 的 Pod 都已准备好被调度，有些 Pod 会因为缺少必要资源的状态而无法成功调度，而这也会对调度器中带来额外的工作。

该功能在 Pod 中增加 spec.schedulingGates 字段来控制 Pod 是否已经准备好真正调度。

```
spec:
  schedulingGates:
  - name: <value>
```

只有在 spec.schedulingGates 被清空时，才会开始调度：

```
$ kubectl get pod example-po
NAME          READY   STATUS            RESTARTS   AGE
example-po    0/1     SchedulingGated   0          30s
```

新增 Alpha Feature Gate —— PodSchedulingReadiness 来控制是否开启该功能

PR#111726 在 Scheduler 的调试 Dummper 中输出 Pending 状态的 Pod 信息。

优化与 BUG FIX

PR#111809 在使用 Patch 更新 Pod 状态时， 除了 net.ConnectionRefused 外，遇到 ServiceUnavailable 和 InternalError 错误时也会进行重试。

在 APIServer 暂时无法处理请求时，会出现 ServiceUnavailable 错误。

> E0805 20:54:21.624945 123623 scheduler.go:356] Error updating pod foo: the server is currently unable to handle the request (patch pods foo)

InternalError 通常会由于 webhook 的临时故障而出现。

> E0811 23:32:30.886582 213747 scheduler.go:357] Error updating pod foo: Internal error occurred: failed calling webhook “xyz”: Post “xyz”: context deadline exceeded

功能稳定性升级

Alpha –> Beta

Feature Gate	**KEP**
NodeInclusionPolicy	KEP–3094 Take taints/tolerations into consideration when calculating PodTopologySpread skew

可观测性

KEP-3466 Kubernetes Component Health SLIs

之前并没有标准的格式来暴露 Kubernetes 组件的健康信息，该功能在每个组件中增加了一个新的路径/metrics/slis，以 Prometheus 格式来公开服务级别指标(ServiceLevelIndicator）。

每个组件都需要暴露两个 metrics:

- gauge: 组件当前的健康检查状态

  > # HELP kubernetes_healthcheck [ALPHA] This metric records the result of a single healthcheck.
  > # TYPE kubernetes_healthcheck gauge

```
kubernetes_healthcheck{name="etcd",type="healthz"} 1
kubernetes_healthcheck{name="etcd",type="readyz"} 1
```

- counter: 对每个健康检查检测到的状态的累积计数

```
# HELP kubernetes_healthchecks_total [ALPHA] This metric records the results of all healthcheck.
# TYPE kubernetes_healthchecks_total counter
kubernetes_healthchecks_total{name="etcd",status="success",type="healthz"} 15
kubernetes_healthchecks_total{name="etcd",status="success",type="readyz"} 15
```

- Kube APIServer: PR#112741
- Kube Controller Manager: PR#112978
- Kube Scheduler: PR#113026
- Kubelet: PR#113030
- Kube Proxy: PR#113057
- Cloud Controller Manager: PR#113340

新增 Alpha Feature Gate —— ComponentSLIs 来控制是否开启该功能。

各组件中增加了一些 metrics 指标，并修复了一些指标计算的问题。

Kubectl 命令

子命令增强和修复

PR#109525 kubectl wait 命令支持在 -o jsonpath= 中设置不存在的字段，这在某些字段被异步设置时会很有用

PR#111096 kubectl api-resources 在 -o wide 输出时增加 categories 列，并且增加 —categories 参数来支持根据 categories 过滤。

PR#113819 kubectl alpha event 移动为顶级命令 kubectl events。

PR#111093 修复了 kubectlrollouthistory—revision=–ojson|yaml 来输出 json/yaml 时,返回最新版本而不是指定的 revision。

PR#111571 优化 kubectl label —dry–run 命令的提示信息，避免用户误解为 label 已被设置。

- before

  ```
  $ kubectl label pod foo bar=baz --dry-run=server
  pod/foo labeled
  ```

- after

  ```
  $ kubectl label pod foo bar=baz --dry-run=server
  pod/foo labeled (server dry run)
  ```

PR#112556 优化 kubectl patch 使用 StrategicMerigePatch 更新自定义资源时的错误信息。

PR#112700 修复 kubectl covert 选择了错误的 api 版本的问题。

PR#109505 kubectl annotate 设置与原值相同值的 annotation 时，不再抛出错误。

PR#110907 执行 kubectl apply 时，如果指定了 —namespace，但是没有指定 —prune–allowlist，会删除非命名空间的资源， 该 pr 只是增加打印一个警告，在 1.28 中 kubectl apply 在指定 namespace 时，不再删除无命名空间的资源 pv & namespace。

PR#113116 kubectl apply 增加 —prune–allowlist flag，配合 —prune flag 使用，替代已废弃的 —prune–whitelist flag。

Other

PR#113146 kubectl explain 命令可以通过环境变量 KUBECTL_EXPLAIN_OPENAPIV3 来使用 OpenAPIv3。

PR#112553 Kubectl 转义输出中的终端特殊字符。修复 CVE-2021-25743。

PR#112150 优化 kubectl 对 APIServer 返回的无效请求的显示。

PR#112243，PR#112261 废弃 kubectl run 命令的多个 flag，即使设置了他们也会被忽略。

Shell 补全

PR#113636 kubectl shell 补全支持在 bash 中显示命令描述。

```
bash-5.1$ kubectl a[tab][tab]
alpha          (Commands for features in alpha)
annotate       (Update the annotations on a resource)
api-resources  (Print the supported API resources on the server)
api-versions   (Print the supported API versions on the server, in the form of "group/version")
apply          (Apply a configuration to a resource by file name or stdin)
argo           (The command argo is a plugin installed by the user)
attach         (Attach to a running container)
auth           (Inspect authorization)
autoscale      (Auto-scale a deployment, replica set, stateful set, or replication controller)

bash-5.1$ kubectl --c[tab][tab]
--cache-dir                (Default cache directory)
--certificate-authority    (Path to a cert file for the certificate authority)
--client-certificate       (Path to a client certificate file for TLS)
--client-key               (Path to a client key file for TLS)
```

--cluster (The name of the kubeconfig cluster to use)
--context (The name of the kubeconfig context to use)

PR#105867 提供对 kubectl 插件的 shell 补全，插件可以通过 kube_complete- 来提供插件命令的 shell 补全。

Kubeadm

命令修复和增强

PR#113005 kubeadm join phase control-plane-preapare certs 支持使用 --dry-run 运行。

PR#112945 支持子阶段的 dry-run 模式，例如 kubeadm reset phase cleanup-node --dry-run。

PR#111512 kubeadm init 命令中增加一个新 phase —— show-join-command，用户可以通过 kubeadm init --skip-phase=show-join-command 来跳过打印 join 信息，该阶段无法单独执行。

PR#112172 kubeadm reset 命令增加 一cleanup-tmp-dir flag，它会清理 /etc/kubernetes/tmp 中的内容，默认为 false。

PR#112732 kubeadm 增加对配置中镜像仓库格式的校验。

PR#111783 kubeadm 读取的 kubeconfig 的 CertificateAuthorityData 为空时，会尝试从外部 CertificateAuthority 文件加载 CA 证书。

PR#112508 在 preflight check 时允许 RSA 和 ECDSA 格式的 key。

PR#110972 kubeadm reset 在执行时会尽可能的去清理掉陈旧的数据，旧数据会在每个 reset 阶段执行时被清除，默认的 etcd 数据目录会在 remove-etcd-member 阶段执行时被删除。

PR#112751 修复验证 ClusterConfiguration 网络相关字段(dnsDomain，serviceSubnet，podSubnet）时的 bug。

Other

PR#111277 优化 kubeadm 在运行子命令时的错误提示信息。

PR#112008 由于 1.25 中不再将 node-role.kubernetes.io/master 污点设置在控制面节点中，kubeadm 不再为 CoreDNSDeployment 设置 node-role.kubernetes.io/master 容忍。

PR#112000 kubeadm init|join|upgrade 命令删除 —container-runtime flag，因为自从 dockershim 被移除后，该 flag 仅有一个可以设置的值 —container-runtime=remote。

Client-Go

PR#112200 client-go 的 SharedInformerFactory 增加 Shutdown 方法，来等待 Factory 内所有运行的 informer 都结束。

移除功能

新版本移除了已经 GA 的功能 Feature Gate：

- ServiceLoadBalancerClass
- ServiceLBNodePortControl
- CSRDuration
- DefaultPodTopologySpread
- NonPreemptingPriority
- PodAffinityNamespaceSelector
- PreferNominatedNode
- PodOverhead
- UnversionedKubeletConfigMap
- IndexedJob
- SuspendJob

功能降级

Beta -> Alpha

LocalStorageCapacityIsolationFSQuotaMonitoring 在 v1.25 中升级到 Beta，但是由于 ConfigMap 更新后不会正常同步到 Pod 文件系统的问题而被回退到 Alpha。

历史发布

- Kubernetes 1.25 正式发布，多方面重大突破
- Kubernetes 1.24 走向成熟的 Kubernetes
- Kubernetes 1.23 正式发布，有哪些增强?
- Kubernetes 1.22 颠覆你的想象：可启用 Swap，推出 PSP 替换方案，还有……
- Kubernetes 1.21 震撼发布 | PSP 将被废除，BareMetal 得到增强

参考资料

- KEP-2876 CRD 验证表达式语言
- 报告
- 会议纪要



5.0 容器管理能力介绍

自 2013 年 Docker 开源以来，以 Docker 为代表的容器技术为软件行业带来了颠覆性的变革。数字经济成为当代经济发展的重要驱动力量。 基础设施云化，软件云原生化已成了整个社会科技发展的必然趋势。以金融行业为代表，传统制造业为基石的各行各业都在积极拥抱和探索这一技术发展的新趋势。 云原生技术抽象了底层基础设施，使开发人员能够专注于应用本身开发，而无需担心底层系统架构等运维问题，让应用更加弹性、高效。 Daocloud 第五代容器管理（Kpanda）是基于 Kubernetes 开源技术构建的面向云原生工作负载的容器管理平台，基于原生多云架构，完全解耦底层基础设施平台， 实现多云与多集群统一化管理，进一步简化企业的应用上云流程，降低运维管理和人力成本。 支持一键式创建 Kubernetes 集群，帮助企业快速搭建企业级的容器云平台。

容器管理的主要功能如下：



容器管理功能

集群管理

- 多云多集群统一纳管，支持所有特定版本范围内的任意 Kubernetes 集群纳入容器管理管理范围，实现云上云下多云、混合云容器云平台的统一纳管。
- 集群快速部署，通过 Web 界面快速部署企业级的 Kubernetes 集群，快速搭建企业级容器云台，适配物理机和虚拟机底层环境。
- 集群一键升级，支持一键升级集群 Kubernetes 版本。

- 集群高可用，内置集群容灾、备份能力，支持业务系统在主机故障、机房中断、自然灾害等情况下快速恢复，进一步提高生产环境的稳定性，降低业务中断风险。

节点管理

- 支持对管理集群和基于管理集群创建的工作集群内的节点进行全生命周期管理。
- 实时监控节点状态。支持实时对节点和节点上运行的容器组状态进行监控。
- 支持采用界面 CloudShell 的方式登录并管理节点。
- 提供对节点的调度策略、标签、污点、注解等进行管理的基础云原生能力。

负载管理

- 一站式部署，解耦底层 Kubernetes 平台，一站式部署和运维业务应用，实现工作负载的全生命周期管理。
- 工作负载负载的扩缩容，通过界面可实现工作负载负载的手动/自动扩缩容能力，自定义扩缩容策略，应对流量高峰。
- 工作负载的全生命周期，支持工作负载查看，更新，删除，回滚，时间查看以及升级等全生命周期管理。
- 跨集群负载统一管理能力。

插件管理

- 内置 DaoCloud 自研核心插件库，支持引入 Helm 官方库和私有插件仓库，为用户提供海量的插件。
- 提供对插件进行增删改查的全生命周期管理能力。
- 对插件的安装、更新、删除等操作进行记录，并以运维人员更容易理解的控制台日志进行展现，使插件运维更加云原生化。

网络访问

提供云原生的负载均衡能力，通过固定的 IP 地址和端口进行访问。目前支持的网络访问模式包括：

- 集群内访问（ClusterIP）：仅在集群内访问服务。
- 节点访问（NodePort）：使用节点 IP 进行访问。
- 负载均衡（LoadBlance）:基于公有云厂商的外部负载进行访问。

存储管理

- 数据卷管理：支持本地存储、文件存储、块存储，通过 CSI 能力接入，并提供给工作负载工作负载使用。
- 数据卷动态创建：可支持存储池实现动态创建数据卷
- 存储池管理：兼容业内各类主流存储技术和符合标准 Kubernetes CNI 架构的存储插件。

权限管理

支持多集群权限的统一管理，结合 Kubernetes 原生的 RBAC 角色访问控制和全局管理模块的用户身份认证，提供工作空间级别、集群级别和命名空间级别的权限控制。 基于以往金融、银行、传统制造等多行业的生产运维管理经验，提供开箱即用的多种企业权限控制方案，帮助不同组织架构的企业快速对用户/用户组进行授权。

集群运维

支持集群级、节点级、负载级、容器级的全方位指标监控及告警，集成各类资源状态统一监控看板。提供集群自动/手动巡检能力，帮助运维和开发人员实时了解集群资源状态，保障业务连续性。

开放式 API

提供云原生的 Kubernetes OpenAPI 能力，打通容器管理与不同系统的界限，快速赋能用户业务上云。

对 Kubernetes 友好的 Web 终端 CloudTTY

很多企业在使用云平台管理 Kubernetes 时，出于安全考虑，不支持使用 SSH 的方式直连业务主机，因此需要云 Shell 能力。 Daocloud 第五代容器管理模块引入了自研开源组件 CloudTTY，支持通过 CloudShell 直连集群。并提供通过 Kubectl 工具对集群进行运维操作的能力。

图形化用户交互

容器管理模块是一个多集群管理的聚合平台，支持用户对不同厂商、不同区域的集群进行统一的纳管。 帮助企业快速上云，满足用户多云多集群管理的使用场景。在层级结构上，采用了三级架构的形式，最上层将用户常用的使用场景进行抽象，提炼出集群列表、命名空间、工作负载、权限管理几个模块。

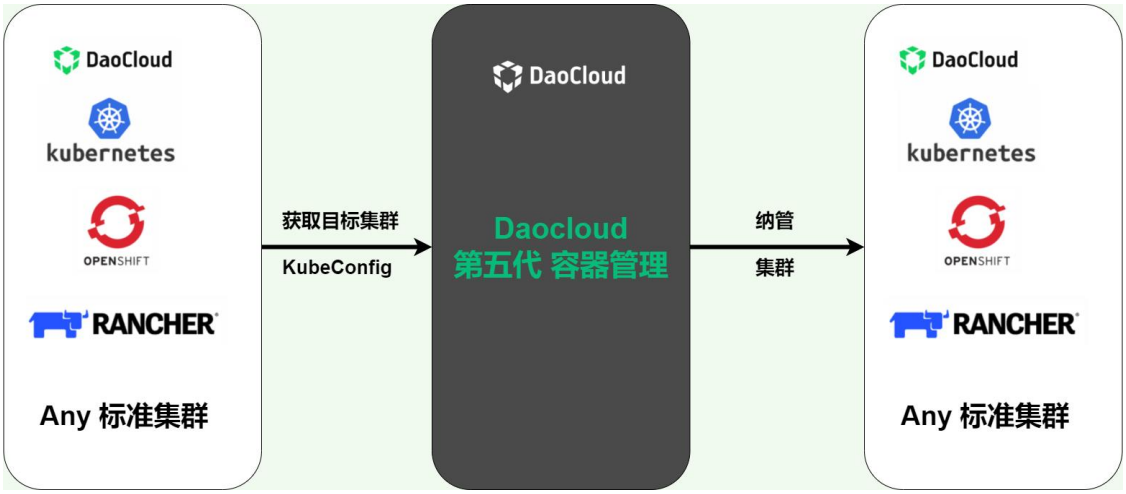
- 用户通过集群列表界面，能够清晰的感知每一个集群的运行状态、资源用量、节点/容器组数量等实时信息。 并且能够基于集群名称进入集群的详情之中，对集群的节点、工作负载、服务与路由、运维管理等功能进行精细化管理。 同时如果用户想使用 kubectl 的方式管理集群，也可以在界面上使用基于 Cloudtty 桌面控制台工具。
- 在命名空间界面，用户能够感知所拥有权限的全部集群下的命名空间的状态，并且能够对不同的命名空间进行配额设置和绑定/解绑工作空间。
- 在工作负载界面，用户能够感知所拥有权限的全部集群下的工作负载状态、资源使用率等工作负载的全局概览信息。
- 在权限管理界面，用户能够基于集群、命名空间两个层级对用户/用户组的权限进行查看以及授权。
- 在集群详情界面，用户能够查阅到详细的集群概览信息，并且能够对集群内的资源进行全生命周期的管理。 包括节点管理、服务与路由管理、工作负载和容器组管理、命名空间管理、Helm 插件管理、存储管理、运维管理等功能都能够在集群详情界面内进行操作。

最佳实践探索

在本地构建起第一个全局服务集群（Kpanda）后，您就能够通过 Web UI 的方式，对容器集群进行管理了。您可以借助容器管理模块实现接入集群、创建集群、应用一键部署等功能操作了。

接入集群

首先前往目标集群的控制节点上获取它的 KubeConfig 文件，然后在容器管理的集群列表处点击接入集群按钮，并将 KubeConfig 文件内的内容作为参数复制粘贴到接入参数的表单内，既完成了对一个集群的接入操作。



img

创建集群

容器管理模块支持创建工作集群和管理集群两种集群类型，用户可根据自身业务需要，选择相应的集群模式。 管理集群和工作集群的差别是在管理集群只运行系统组件，不部署业务，以保障集群的高可用。 工作集群只运行业务相关的负载，同时管理集群支持纳管工作集群，能够基于管理集群创建工作集群，并在创建过程中为工作集群配置网络、节点、插件等操作。

部署工作负载

容器管理模块支持基于 Kubernetes 原生的工作负载类型部署和管理能力，包括：创建、配置、监控、扩容、升级、回滚、删除等全生命周期管理。

- **弹性扩缩** ： 通过界面可实现工作负载的手动/自动扩缩容能力，自定义扩缩容策略，应对流量高峰。
- **容器的生命周期设置** ： 支持创建工作负载时设置回调函数、启动后参数、停止前参数，满足特定场景需求。
- **容器就绪检查、存活检查设置** ： 支持部署工作负载时设定工作负载就绪检查及存活检查：
- **工作负载就绪检查** ： 用于检测用户业务是否就绪，如未就绪，则不转发流量至当前实例。
- **工作负载存活检查** ： 用户检测容器是否正常，如异常，集群会执行容器重启操作
- **容器的环境变量设置** ： 可为业务容器运行环境设定指定环境变量。
- **容器的自动调度** ： 支持工作负载服务调度管理，自动按照主机资源用量进行容器调度，可以指定具体的部署主机，以及通过标签策略进行容器调度。
- **亲和性和反亲和性** ： 支持定义容器组间调度的亲和性和反亲和性；以及容器组与节点间的亲和性和反亲和性，满足业务自定义调度需求。
- **容器安全用户设置** ： 支持设定容器运行用户，如以 Root 权限运行则填写 Root 用户 ID 0。
- **自定义资源（CRD）支持** ： 支持自定义资源的创建、配置、删除等全生命周期管理及基于自定义资源创建实例（CR）的能力。

谁需要容器管理

业务庞大、或跨地区或跨部门需要统一运维管理的大型金融、银行、航空航天、传统制造业等类型的企业。

解决什么问题

1. 在现代企业的软件生成过程中，虽然一般来讲，最佳做法是使用尽可能少的集群，但企业出于各种原因会选择部署多个集群来实现各种技术和业务目

标。 大多数企业至少将各种服务放在不同的集群中，将非生产服务与生产服务分开。在更复杂的场景中，企业可能会选择多个集群来跨层级、语言区域、团队或基础架构提供商分隔服务。

2. 超大规模项目部署在一个集群上，为了保证工作负载的可用性，只能手动不断的往集群上新增节点，但是 Kubernetes 的最大 Pod 数有边界，单纯依靠增加物理节点并不能解决根本问题，且运维难度和成本极速增加。
3. 混合云场景，集群类型复杂，既有远程的边缘融合集群，还有本地自建机房集群和信创集群及公有云集群，集群的 Kubernetes 版本可能还存在差异，如何统一管理这些不同架构的集群问题。
4. 由于企业希望在更多传统工作环境（例如仓库、工厂车间、零售店等）中采用应用现代化改造做法，因此这需要管理更多基础架构组件上的更多工作负载。
5. 当企业担心潜在就地升级问题（即升级自动化故障、应用不稳定或回滚功能）时，他们可以选择将其服务副本部署到新集群中。以这种方式升级需要规划或自动化才能实现，从而确保在升级过程中处理流量管理和状态复制。

应用场景

1. 探索一种针对多机房、跨区域的容器调度方式，实现不同区域集群的统一纳管，减少人力物力消耗，缩减成本。
2. 探索集群级别的弹性扩缩，支持创建集群，并打通集群间的边界，探索跨集群的工作负载部署方式。
3. 探索不同集群类型间的兼容性统一管理，形成资源的统一管理平面。

FAQ

- 问：容器管理兼容哪些厂商的集群?

  答：第五代容器管理兼容包括 AWS、谷歌云、信创架构、边云融合等任何基于标准 Kubeneters ApiServer 的发行版容器集群。



兼容的厂商

- 问：如应对跨集群的网络中断问题?

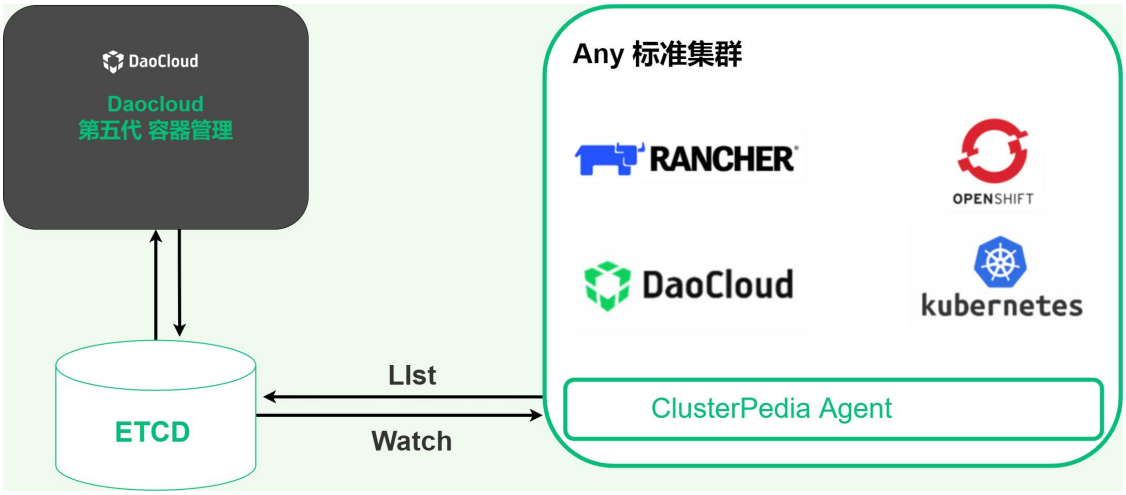
  答：为保证业务高可用，同时将业务部署在不同地域的多个云容器平台上，帮助应用实现多地域的流量分发，并且实现跨云应用在同一平台管理，减少运维成本。 另外当某个云容器平台发生故障时，通过统一流量分发的机制，自动的将业务流量切换到其他云容器平台上。

- 问：如何实现对其它厂商集群的管理?

  答：通过获取其它厂商集群的 KubeConfig 文件，在容器管理模块执行接入集群操作，填入目标集群的 KubeConfig 文件，即完成了对一个容器集群的接入管理。

- 问：如何获取多集群资源的实时状态?

  答：借助自研开源组件 ClusterPedia 通过在目标集群上安装 Agent 组件，结合 Watch 机制能够实时将目标集群的资源变动信息同步到容器管理集群的 ETCD 内。



实时 watch

- 问：容器管理在第五代产品中的定位?

  答：在第五代产品中，容器管理模块处于承上启下的核心位置。对上：对接 DCE 5.0 产品模块及未来定制化模块。对上层的场景化管理模块而言，容器管理模块能够把任何一个标准的 Kubernetes API 对上暴露，无论是制品中心、Karmada、还是应用商店等，只要需要调用 Kubernetes API，就能够通过容器管理这个模块去调用底层的任何平台的 Kubernetes API。对下：对接 Any Kubernetes，例如边缘的 K3S undefined 信创环境、DCE、DKG、DKE、以及外部的 Openshift，Tanzu、CCE 等等都是基于容器管理这个模块。

- 问：谁能够创建集群?

  答：拥有 Global Admin、Kpanda Owner 角色的用户能够去创建集群。

- 问：谁能管理集群?

  答：取决于用户所采用的权限控制解决方案，一般来说除了拥有 Global Admin、Kpanda Owner、Cluster Admin、Cluster Admin 角色的用户外，拥有 Cluster Edit 角色的用户及拥有强绑定的 Floder Admin、

WorkerSpace Admin 角色的用户也能够实现对集群内资源的增删改查操作，以及对用户进行授权。

了解容器管理

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

Istio 1.18 新鲜出炉：正式宣布 Ambient 模式

作者：hanxiaop



istio 1.18

太平洋时间 2023 年 6 月 5 日，Istio 正式发布了 **1.18** 版本。

1.18 版本是 Istio 在 2023 年发布的第二个版本，也是首个支持 Ambient 模式的版本。

本次版本更新引入了许多新功能和变化，包括但不限于 Ambient 模式的 Istio、增强的 Kubernetes Gateway API、 针对未自动注册的虚拟机的健康检查支持、新增对 Metrics 的过期支持、增强的 istioctl analyze 等。

在此次版本更新中，DaoCloud 作为 Istio 社区主要提供商，主要在安装部分的功能和优化、istioctl 的增强以及对 Ambient 模式的适配和文档完善等方面做出了贡献。同时，DaoCloud 韩小鹏作为 1.18 版本的 Release Manager，一直在参与发版工作。

!!! info

Istio 1.18 版本支持的 Kubernetes 版本为 1.24，1.25，1.26，1.27。

接下来，我们将详细讨论一些重要的功能和变化。

Ambient Mesh（环境网格）

在介绍 Ambient 模式之前，我们需要了解当前 Istio 使用的 Sidecar 模式以及它所面临的挑战。

Sidecar 模式是 Istio 目前采用的数据平面模式，该模式中，每个应用程序 Pod 都配备一个 Sidecar 代理（通常为 Envoy）。 这个代理负责处理进出 Pod 的所有网络流量，从而提供 Istio 的核心功能，诸如零信任安全、遥测和流量管理等。

然而，Sidecar 模式也存在一些挑战，包括但不限于以下几点：

- 侵入性强：在 Sidecar 模式中，每个应用程序 Pod 都需要注入一个 Envoy 代理。 这就导致一些应用程序必须在设计和编码阶段考虑到与 Sidecar 代理的交互。此外， 当网格升级时，Sidecar 的升级过程可能会影响应用程序，不能很好地实现网格能力和业务容器的解耦，从而增加了应用开发和运维的复杂性。
- 资源消耗高：Sidecar 模式下，由于每个 Pod 都配有一个 Envoy 代理，这意味着每个 Pod 都需要为其 Sidecar 分配额外的 CPU 和内存资源。这不仅可能导致资源的浪费，也相应地增加了基础设施的成本。
- 安全配置繁琐：虽然 Sidecar 代理提供了安全的核心功能，但目前相对来讲配置繁琐。 例如 mTLS，在 Sidecar 模式中，每个 Pod 或者不同层级的工作负载都需要单独配置安全功能， 运维人员可能需要额外的时间和技术来处理这些繁琐的配置。

正是由于以上这些挑战，Istio 推出了新的数据平面模式 —— Ambient Mesh（环境网格）。 该模式旨在简化操作、扩大应用程序兼容性并降低基础设施成本，同时继续提供 Istio 的核心功能。 有关更多信息，可参考 Istio 环境网格介绍。

关于 Ambient 模式有以下好处：

- 非侵入性：与需要在应用程序的 Pod 中注入边车代理的传统 Istio 架构相比，环境网格不需要侵入应用程序。 只需要打上对应的标签，应用即可自动加入网格，从而减少了网格对应用程序的影响。
- 更高的资源利用率：在边车模式下，每个 pod 都需要为最坏情况分配 CPU 和内存，这一直是一个问题。 环境网格通过共享代理 Ztunnel 有效地减少了资源的浪费。如果需要 Istio 的某些七层功能， 也可以通过精确地为某一个 Service Account 或 Namespace 部署一个 Waypoint 来解决，从而更好地控制资源消耗。
- 性能：在早期，Istio 采用了基于 Envoy 的共享代理模式，然而在开发过程中，发现 Envoy 存在配置繁琐等问题， 因此 Istio 开发了自己的基于 Rust 版本的共享代理（ztunnel）。在未来，预计环境网格将与传统的边车模式具有相当的性能表现。
- 安全性：环境网格通过在每个节点上运行一个共享代理 Ztunnel 来确保安全性。这个代理被部署在每个节点上， 负责处理所有进出节点的流量，可以提供与传统 Istio 网格中边车代理相同的安全性。 一旦为命名空间启用 Ambient 模式，就会创建一个安全覆盖层。 这个覆盖层为应用程序提供 mTLS、遥测、身份验证和 L4 授权等功能，而无需终止或解析 HTTP 流量。

然而，如果 L4 无法满足应用程序的需求，这时 Waypoint 就可以作为 L7 的代理出现。 与 Ztunnel 不同，Waypoint 仍然使用 Envoy 进行流量代理和转发。启用 Ambient 模式的命名空间可以通过部署一个或多个 Waypoint 来处理该命名空间中的 L7 流量， 不同于以往的边车模式，Waypoint 实质上也是像应用程序一样部署单个 Envoy 作为 Pod， 它可以像其他 Kubernetes 资源一样进行自动扩展。

用户可以通过命令行工具 istioctl x waypoint 进行 Waypoint 的配置管理。 例如，通过运行 istioctl x waypoint generate 命令，用户可以生成一个由 Istio 管理的 Kubernetes Gateway API 资源。当此资源在 Kubernetes 中部署时，将自动

创建如 Service、Deployment、Service Account 等相关资源，同时，对 Gateway 的任何修改都会直接更新到这些资源上。

总的来说，环境网格的出现为用户提供了更多的灵活性，同时降低了使用网格的复杂性。

需要注意的是，环境网格目前仍处于 Alpha 阶段，还没有达到生产环境的稳定性，我们期待在未来的 Istio 版本中看到更多更新和改进。

Kubernetes Gateway API 的增强和改动

Istio 1.18 版本引入了对 Kubernetes Gateway API 的多项重要改进和修改：

1. 支持 v1beta1 版本： 在升级到新版本的 Istio 时，注意需要的 Gateway API 版本必须大于 0.6.0+。 可以使用 istioctl x precheck 命令检查是否存在升级问题。
2. Gateway API 自动化部署的管理升级： 任何由 Istio 管理的 K8S Gateway 资源在被创建或更新后， 会自动配置 Service 和 Deployment 资源。如果 Gateway 资源发生变化，相关的配置也会同步更新。 此外，现在 Gateway 资源的部署不再依赖于注入逻辑，而是有独立的创建流程。
3. 去除对 proxy.istio.io/config 注解的支持： 同时，ProxyConfig 资源仅在 Istio 管理的 Gateway 上生效。
4. 修复了 Istiod 对配置更改的处理问题： 如果 Service 和 Deployment 配置发生更改，Istiod 现在会重新处理。
5. 默认不再支持 Alpha 版本的 Gateway API： 可以通过设置 PILOT_ENABLE_ALPHA_GATEWAY_API=true 来重新启用。

值得注意的是，与之前的 Istio 安装不同，安装 Ambient 模式时并不默认包含 IngressGateway。 在未来的开发中，Istio 更倾向于使用 Gateway API 来管理网关。

并发性设置的改变

以前，concurrency 设置在边车和网关的不同安装机制之间存在不一致。在 Istio 1.18 中，这种设置被调整，以保持在不同部署类型之间的一致性。

Istioctl 的增强

1. 对于环境网格模式，新增了 isioctl x waypoint 命令，用于管理 Waypoint 的配置。

2. istioctl analyze 新增了对网关证书配置异常和 Telemetry 资源异常的分析。

移除的功能

1. 移除对 Ingress networking.k8s.io/v1beta1 版本的支持，v1 版本从 Kubernetes 1.19 开始支持。

2. 移除了实验性的 Istio CNI 的 taint controller 功能。

3. 移除对 EndpointSlice 的 discovery.k8s.io/v1beta1 版本的支持，v1 版本从 Kubernetes 1.21 开始支持。

升级注意事项

代理并发性变化

以前，配置代理运行的工作线程数的 concurrency 设置，在 sidecars 和不同的网关安装机制之间配置不一致。 这常常导致网关基于主机机器的物理核心数量运行并发，尽管有 CPU 限制，这导致性能下降和资源使用增加。

在这个版本中，并发配置已经被调整，意在部署类型之间保持一致。新的逻辑将使用 ProxyConfig.Concurrency 设置 （可以在整个网格或每个 pod 上配置），如果未设置会根据分配给容器的 CPU 限制设置并发。例如，2500m 的限制将把并发设置为 3。

在此版本之前，边车遵循这个逻辑，但有时错误地确定了 CPU 限制。网关将永远不会根据并发设置自动适应。

要保留网关始终利用所有核心的旧行为，可以在每个网关上设置 proxy.istio.io/config: concurrency: 0。 然而如果这是期望的情况的话，建议取消设置 CPU 限制。

网关 API 自动部署变化

只有当你使用网关 API 自动部署时， 这个变化才会影响你。请注意，这只适用于 Kubernetes Gateway API，而不是 Istio Gateway。 你可以使用以下命令检查你是否正在使用此特性：

```
kubectl get gateways.gateway.networking.k8s.io -ojson | jq -r '.items[] | select(.spec.gatewayClassName == "istio") | select((.spec.addresses | length) == 0) | "Found managed gateway: " + .metadata.namespace + "/" + .metadata.name'
Found managed gateway: default/gateway
```

如果你看到“Found managed gateway”，那么你可能会受到这个变化的影响。

在 Istio 1.18 之前，托管网关通过创建一个最小的 Deployment 配置来工作，该配置在运行时通过 Pod 注入完全填充。 为了升级网关，用户会重启 Pods 来触发重新注入。

在 Istio 1.18 中，这已经改变，现在通过创建一个完全由 Helm 渲染的部署，并不再依赖注入。 因此，**当它们的版本改变时，网关将通过滚动重启进行更新** 。

此外，使用此特性的用户必须在采用 Istio 1.18 之前将其控制平面升级到 Istio 1.16.5+ 或 1.17.3+。 如果不这样做，可能会导致对同一资源的冲突写入。

具体所有的改动可参考 Istio 1.18 版本的发版信息。

云原生监控 - VictoriaMetrics 之基础篇



VictoriaMetrics

说到云原生监控方案，第一时间基本上都会想到 Prometheus+AlertManager+Grafana 的一套成熟解决方案。 Prometheus 作为监控核心，具备强大的数据模型、高效率运作、丰富的监控能力、强大的查询语言 PromQL、 简单易用、管理方便等特点。但是 Prometheus 目前在高可用层面上做得还并不完美。为此，在开源社区中， 孕育出了许多替代、增强方案，VictoriaMetrics 属于其中较为优异的一个，是一个快速、经济高效且可扩展的监控解决方案和时间序列数据库。

七大特点

1. 它可以作为 Prometheus 的长期储存，且支持 Prometheus 查询 API，可以在 Grafana 中用作 Prometheus 的代替品；
2. 部署简单，无论是单节点版本还是集群版本，都只需要运行所需的组件可执行文件（每个组件都是一个可执行文件），运行前不需要安装任何依赖，易于设置和操作；
3. 使用 vmbackup/vmrestore 工具可以轻松快速地将即时快照备份到 S3 或 GCS；
4. 基于 PromQL 的查询语言实现 MetricsQL，对 PromSQL 进行改造；

5. 读写性能比 InfluxDB 和 TimescaleDB 高达 20 倍；百万时间序列数据下，内存使用比 InfluxDB 少 10 倍， 比 Prometheus、Thanos 或 Cortex 少 7 倍；数据高压缩，与 Prometheus、Thanos 或 Cortex 相比，所需的存储空间最多可减少 7 倍；
6. 具有高延迟 IO 和低 IOPS；
7. 支持从第三方时序数据库获取数据源。

快速接入 Prometheus 获取数据源

数据源接入层面，VictoriaMetrics 支持通过 Prometheus 的远程写入方式直接兼容 Prometheus 的数据写入，同时也支持搜集多个 Prometheus 数据汇总。

```
remote_write:
  - url: http://<victoriametrics-addr>:8428/api/v1/write


# 多个 proemtheus 的话，需要配置每个 Prometheus 的标识符
global:
  external_labels:
    datacenter: dc-123
```

VictoriaMetrics 还支持直接取代 Prometheus 进行 exporter 搜集。

可以使用 -promscrape.config 配置 Prometheus 的 prometheus.yml

针对 Prometheus，VictoriaMetrics 进行了一些优化：

1. 增加了 extra_label=<label_name>=<label_value> 可选的查询支持，可用于基于额外标签进行查询过滤。 例如 /api/v1/query_range?extra_label=user_id=123&extra_label=group_id=456&query=<query>， 会返回额外标签中包含 {user_id="123",group_id="456"} 的结果；
2. 增加了 extra_filters[]=series_selector 可选的查询支持，可用于基于拓展标签进行规则匹配的查询过滤。 例如

/api/v1/query_range?extra_filters[]={env=~"prod|staging",user="xyz"}&query=<query>， 会返回额外标签中包含 {env=~"prod|staging",user="xyz"} 的结果；

3. 支持“start“ 和 “end“，使用多种时间格式，如 1562529662.678、2022-03-29T01:02:03Z、2022-03、1h5m 等；
4. 在 /api/v1/query“和“/api/v1/query_range 中增加了 round_digits 参数，它可用于将响应值四舍五入到小数点后给定的位数；
5. 在 /api/v1/labels 和 /api/v1/label/<labelName>/values 中增加了 limit 参数，用于限制返回条目的数量；
6. 在 /api/v1/series 中增加了 limit 参数，用于限制返回条目的数量；
7. 新增 /api/v1/series/count，返回数据库中时间序列的总数；
8. 新增 /api/v1/status/active_queries，返回当前正在运行的查询列表；
9. 新增 /api/v1/status/top_queries，返回 “topByCount“ 最常执行的查询；返回 “topByAvgDuration“ 平均执行持续时间最长的查询；返回 “topBySumDuration“ 执行时间最长的查询。

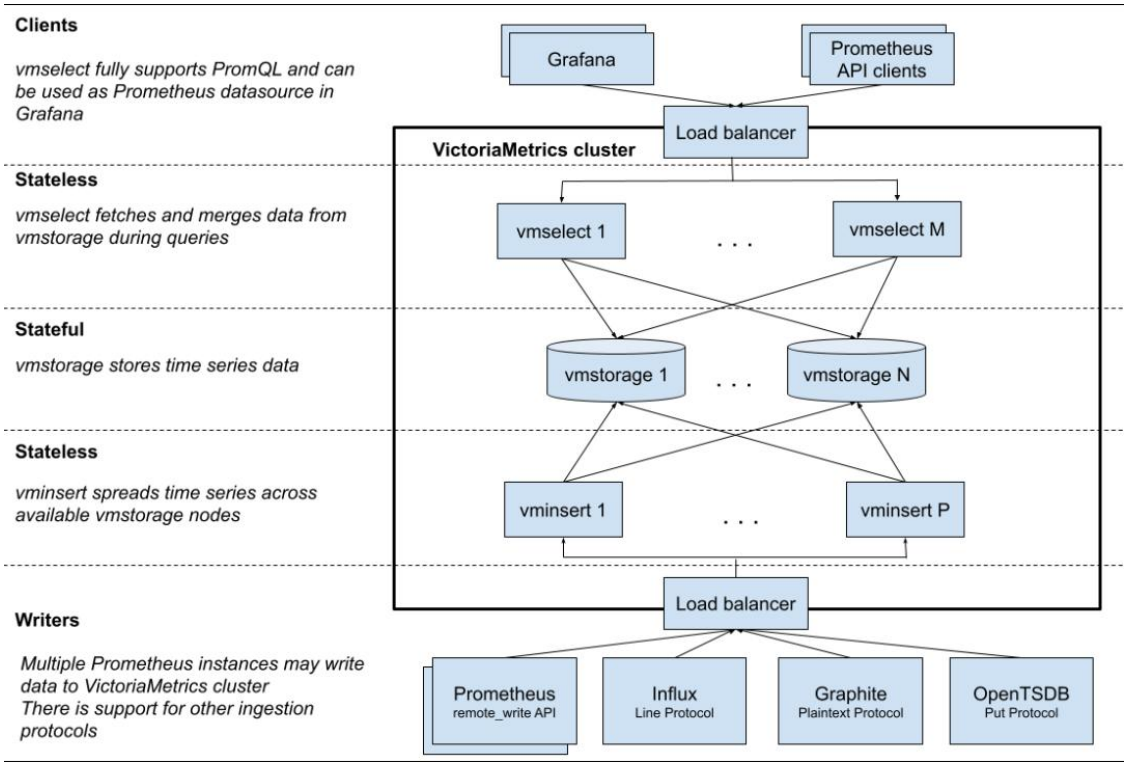
除了支持 Prometheus 作为数据源外，VictoriaMetrics 还支持其他数据源：

1. DataDog agent
2. InfluxDB-compatible agents such as Telegraf
3. Graphite-compatible agents such as StatsD
4. OpenTSDB-compatible agents

架构

面对获取速率低于每秒一百万个数据点的场景下，官方建议使用单节点版本而不是群集版本。 单节点版本可以根据 CPU 内核、RAM 和可用存储空间的数量完美

扩展。与群集版本相比， 单节点版本更易于配置和操作，因此在选择群集版本之前要三思。



Image

VictoriaMetrics 集群由以下服务组成：

1. vmstorage，存储原始数据并返回给定标签过滤器的给定时间范围内的查询数据；
2. vminsert，接受接收的数据，并根据“度量名称及其所有标签”计算出的哈希值，在 vmstorage 节点之间传播数据；
3. vmselect，通过从所有配置的 vmstorage 节点获取所需数据来执行传入查询。

每个服务可以独立扩展，并且可以在最合适的硬件上运行。Vmstorage 节点不了解彼此，不相互通信， 也不共享任何数据。这是一个无共享架构，它提高了群集可用性，简化了群集维护和群集扩展。

VictoriaMetrics 在开源层面，提供以下组件：

1. vmui：负责 vm 页面展示，提供数据查询、指标与基数查询、查询分析、链路分析、job 分析面板等功能；
2. vmagent：负责数据采集、重新标记和过滤收集，并通过 Prometheus 协议将数据存储到 VictoriaMetrics 或其他支持 Prometheus 协议的存储系统中。支持按时间和标签聚合样本后同时复制多个远程存储系统， 且能在传输故障时缓存数据，等待恢复后继续传输；支持抓取暴露数百万时间序列的目标与写入多个租户中；支持 kafka 读写；
3. vminsert：负责数据插入，支持不同格式、不同租户的数据；
4. vmstorage：负责数据存储，具有高压缩率、低资源消耗、高性能的特点；
5. vmselect：负责数据查询，支持数据统一查询与多租户数据隔离查询；
6. vmalert：负责告警，和 Prometheus 一样支持纪录、告警两种规则配置与发送告警通知，允许在注解中使用 Go 模板来格式化数据、迭代或执行表达式，支持跨租户发送警报和记录规则
7. vmbackup：负责数据备份，支持增量备份和全量备份，可以做到每小时、每天、每周和每月备份，支持本地存储、GCS、Azure Blob 存储、S3 存储、任何与 S3 兼容的存储；
8. vmrestore：负责数据还原，支持随时中断与自动从断点恢复。

能力

保存

VictoriaMetrics 使用“–retentionPeriod”命令行标志进行配置，该标志采用一个数字，后跟一个时间单位字符“–h（ours）、d（ays）、w（eeks）、y（ears）”。如果未指定时间单位，则假定为月。例如，“–retentionPeriod=3”表示数据将存储 3 个月，然后删除。默认保留期为一个月。

数据删除

VictoriaMetrics 除了支持配置定时过期外，还支持手动进行数据删除操作，使用“http://<victoriametrics-addr:8428/api/v1/admin/tsdb/delete_series?match[>]=”。删除的时间序列的存储空间不会立即释放，而是在随后的数据文件后台合并过程中释放。请注意，对于前几个月的数据，后台合并可能永远不会发生，因此不会为历史数据释放存储空间。在这种情况下，强制合并可能有助于释放存储空间。

强制合并

VictoriaMetrics 会在后台以每个月为一个分区的形式进行数据压缩，以保持良好的性能。可以使用“http://victoriametrics:8428/internal/force_merge?partition_prefix=YYYY_MM”进行强制数据压缩，会以异步的形式，立马返回请求结果，并在后台执行数据压缩任务。当需要立即删除数据的时候，可以使用强制合并触发数据删除。

数据导入导出

VictoriaMetrics 支持使用专属 Agent 接口导入数据：

1. Prometheus remote_write API
2. DataDog submit metrics API
3. Graphite plaintext protocol
4. OpenTSDB telnet put protocol
5. OpenTSDB http api/put protocol

支持使用 VictoriaMetrics 统一接口导入数据：

1. /api/v1/import，导出 json 格式；
2. /api/v1/import/csv，导出 csv 格式；
3. /api/v1/import/native，导出二进制格式。

相对的，也可以使用导出接口导出数据：

1. /api/v1/export，导出 json 格式；
2. /api/v1/export/csv，导出 csv 格式；
3. /api/v1/export/native，导出二进制格式。

VictoriaMetrics 接口侧，在导入、导出接口中，数据格式保持一致，可以直接使用导出的数据进行数据导入。

消除重复数据

VictoriaMetrics 会根据“–dedup.minScrapeInterval”配置的值进行去重，只留下一个原始样本， 保留配置周期中时间戳最大的数据。比如“–dedup.minScrapeInterval=60s”，将会保留 60s 间隔中最大时间戳的单个原始样本。如果时间戳一致的情况下，随机保留数据。

多个相同配置的 vmagent 或 Prometheus 实例将数据写入同一 VictoriaMetrics 实例，则消除重复数据可减少磁盘空间的使用。

存储

VictoriaMetrics 将以类似 MergeTree 的数据结构存储时间序列数据。插入时，VictoriaMetrics 累积最多 1s 的数据，并将其转储到磁盘上的 storageDataPath>/data/small/YYYY_MM/ 子目录，形成具有以下名称模式的 part：“rowsCount_blocksCount_minTimestamp_maxTimestamp”。 每个 part 由两个“列”组成：值和时间戳。这些是经过排序和压缩的原始时间序列值。 此外，part 包含索引文件，用于在值和时间戳文件中搜索特定 series。

part 会定期合并为较大的 part，生成的部分在 <–storageDataPath>/data/{small，big}/YYYY_MM/tmp 子目录下构造。 当生成的 part 完成后，它将自动从 tmp 移动到自己的子目录，而源部分将自动删除。最终结果是， 源部分被 <–storageDataPath>/data/{small，big}/YYYY_MM/ 目录中的单个较大部分替换。

如果 part 的摘要大小超过可用磁盘空间，VictoriaMetrics 不会合并部件。这可以防止合并过程中出现潜在的磁盘空间不足错误。 在可用磁盘空间不足的情况下，part 的数量可能会随着时间的推移而显著增加。这增加了数据查询期间的开销， 因为 VictoriaMetrics 每次请求都需要从更多的部分读取数据。这就是为什么建议在“-storageDataPath”命令行标志指示的目录下至少有 20% 的可用磁盘空间。

有关合并过程的信息支持在 Grafana 仪表板中查看。

合并过程提高了压缩率，并使磁盘上的部件数量保持相对较低。执行合并过程的好处如下：

1. 它提高了查询性能，因为每次查询都会检查较少的部分；
2. 它减少了数据文件的数量，因为每个部分都包含固定数量的文件。

存储或是合并，都不会只保存部分 part，part 会以数据整体的形式，要么被全部保存成功，要么全部失败。part 是不可变的。

监控

VictoriaMetrics 在“/metrics”页面以 Prometheus exposion 格式导出内部度量。 这些指标可以通过 vmagent 或 Prometheus 获取。或者，当 -selfScrapeInterval 命令行标志设置为持续时间大于 0 时，单节点 VictoriaMetrics 可以自抓取度量。

VictoriaMetrics 在“/api/v1/status/active_queries“ 页面上公开当前正在运行的查询及其执行时间。

VictoriaMetrics 在 ”/api/v1/status/top_queries” 页面上公开了执行时间最长的查询。

TSDB 状态

VictoriaMetrics 以类似于 Prometheus 的方式在“/api/v1/status/TSDB“ 页面返回 TSDB 统计信息：

1. topN=N，其中 N 是响应数量。默认情况下，返回前 10 个；
2. date=YYYY-MM-DD，其中 YYYY-MM-DD 是收集统计数据的日期。默认情况下，收集当天的统计数据；
3. focusLabel=LABEL_NAME，返回 seriesCountByFocusLabelValue 列表中给定 LABEL_NAME 的时间序列数最多的标签值；
4. match[]=SELECTOR，其中 SELECTOR 是一个任意时间序列选择器，用于在统计计算期间考虑序列。默认情况下，将考虑所有系列；
5. extra_label=LABEL=VALUE，拓展标签筛选。

推指标

当出现无法拉取指标的场景下，VictoriaMetrics 支持以 Prometheus 数据格式的方式，通过 push 模式进行指标推送：

1. -pushmetrics.url，推送地址，比如”-pushmetrics.url=http://victoria-metrics:8428/api/v1/import/prometheus“；
2. -pushmetrics.extraLabel，拓展标签，支持以 label=“value” 的形式给 push 数据增加标签；
3. -pushmetrics.interval，push 周期，默认 10s。

缓存

VictoriaMetrics 使用各种内部缓存。在正常关机期间（例如，当 VictoriaMetrics 通过发送 SIGINT 信号停止时）， 这些缓存存储到“<-storageDataPath>/cache”目录中。缓存将在下次 VictoriaMetrics 启动时读取。 有时需要在下次启动时删除这些缓存。这可以通过在重启 VictoriaMetrics 之前将 reset_cache_on_startup 文件放置在“<-storageDataPath>/cache”目录中来执行。

VictoriaMetrics 使用各种内存缓存来加快数据摄取和查询性能。每种类型缓存指标可以在“/metrics”页面导出：

1. vm_cache_size_bytes，实际缓存大小；
2. vm_cache_size_max_bytes，缓存大小限制；
3. vm_cache_requests_total，对缓存的请求数；
4. vm_cache_misses_total，缓存未命中数；
5. vm_cache_entries，缓存中的条目数。

支持在 Grafana 仪表板上查看缓存指标，面板显示了每种类型缓存的当前内存使用情况，以及缓存命中率。如果命中率接近 100%，则缓存效率已经很高，不需要任何调整。

其他

支持众多 Prometheus 具备的功能，比如 Prometheus 的标签重写、联邦等功能。

与 Prometheus 对比

优势：

1. 性能优势。在相同配置、采集压力的情况下，相比使用 Prometheus，存储空间最多可减少 7 倍， 磁盘读写峰值最多可减少 3–4 倍，内存使用最多减少 7 倍；
2. 更优秀的横向拓展、高可用方案。VictoriaMetrics 集群模式，通过组件化各个能力，进行功能层面的架构解耦。 由于负责读写的 vmselect、vminsert 组件都是无状态组件，可以灵活根据读写压力，进行组件的横向扩缩容； 负责存储的 vmstorage 组件虽然是有状态组件，但是当存储压力增加时，同样支持横向扩容，扩容后只需要更新上下游的 vmselect、vminsert 组件配置并重启，即可提升 VictoriaMetrics 集群存储能力；采

集压力的时候， 可以利用扩容 vmagent 与 vmagent 的采集分组能力，进行分散压力采集；

3. 数据多租户能力。VictoriaMetrics 支持把不同类型的数据分别放到不同的租户里，每个租户通过 accountID 或 accountID:projectID 的形式，在请求的 url 里做区分。租户的数量不影响性能， 主要取决于所有租户的活跃总时间序列，每个租户的数据都均匀分布在后端 vmstorage 存储，但是不支持跨租户进行数据查询。

缺点：

1. 没有类似 Prometheus 的 WAL 日志，突然故障可能会丢失部分数据。Prometheus 在接收到数据的时候， 会先把数据写入内存，定期再写入磁盘。为了防止写入磁盘前数据丢失，还会再写入内存的同时简单的写入 WAL 文件里，当出现故障时，就可以通过 WAL 快速恢复当前状态。而 VictoriaMetrics 则在写入过程中， 使用 Go 的 chan 作为数据缓存队列，多协程实现数据处理、压缩、存储等操作，故障下存在缓存数据丢失的情况；

2. 为了更大程度优化存储，会选择丢失部分数据精度。Vmstorage 组件支持配置精度保存范围（1-64，64 表示不丢失），以此在数据读写过程中，提升速度。

安装

使用 helm 进行 VictoriaMetrics 集群模式的部署。

安装 helm，添加 chart helm 仓库：

```
helm repo add vm https://victoriametrics.github.io/helm-charts/
helm repo update
```

导出配置文件，并进行部署配置：

```
helm show values vm/victoria-metrics-cluster > values.yaml
```

可以使用以下命令进行配置检查：

helm install victoria-metrics vm/victoria-metrics-cluster -f values.yaml -n victoria-metrics --debug --dry-run

根据配置文件部署 victoria-metrics-cluster：

helm install victoria-metrics vm/victoria-metrics-cluster -f values.yaml -n victoria-metrics

稍等片刻 Helm 则会机提示安装成功，查询资源可以看到都已经部署起来了。

```
[root@node00 ~]# k get sts -n victoria-metrics
NAME                                                READY   AGE
victoria-metrics-victoria-metrics-cluster-vmstorage   1/2     3m15s
[root@node00 ~]# k get deploy -n victoria-metrics
NAME                                                  READY   UP-TO-DATE   AVAILABLE   AGE
victoria-metrics-victoria-metrics-cluster-vminsert    2/2     2            2           3m31s
victoria-metrics-victoria-metrics-cluster-vmselect    2/2     2            2           3m31s
[root@node00 ~]# k get sts -n victoria-metrics
NAME                                                READY   AGE
victoria-metrics-victoria-metrics-cluster-vmstorage   2/2     3m34s
[root@node00 ~]# k get svc -n victoria-metrics
NAME                                                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                       AGE
victoria-metrics-victoria-metrics-cluster-vminsert    ClusterIP   172.31.202.79   <none>        8480/TCP                      3m43s
victoria-metrics-victoria-metrics-cluster-vmselect    ClusterIP   172.31.134.80   <none>        8481/TCP                      3m43s
victoria-metrics-victoria-metrics-cluster-vmstorage   ClusterIP   None            <none>        8482/TCP,8401/TCP,8400/TCP    3m43s
```

Image

使用

调整 victoria-metrics-victoria-metrics-cluster-vminsert 的 service 与 victoria-metrics-victoria-metrics-cluster-vmselect 的 service，从 ClusterIP 模式改成 NodePort 模式，提供对外访问能力。修改现有 Prometheus 的配置，增加：

```
remote_write:
  - url: http://IP:NodePort/insert/0/prometheus
```

把 Prometheus 的数据，写入 VictoriaMetrics，通过 VictoriaMetrics 的 UI 地址 http://IP:NodePort/select/0/vmui，即可查看 Prometheus 采集到的数据了。



Image



Image

可以看出，两者数据基本一致。

参考资料：https://docs.victoriametrics.com/

2023 年云原生预测



头图

作者：Chris Aniszczyk

我希望每个人都和心爱的人度过了愉快的假期！CNCF 最近发布了关于我们去年完成的所有工作的年度报告。 我建议每个人都抽空仔细阅读这份报告，因为我们花了很多时间来记录社区所做的所有令人惊叹的工作。另外，自从我发布年度预测已经有几年了，所以我为延迟道歉，希望你喜欢今年的清单！

云原生 IDE 成为常态

最近，GitHub Codespaces 完成测试阶段，虽然我们没有很多来自 GitHub 的公共使用数据， 但 Twitter 上似乎有很多正面情绪。此外，GitPod 与该领域的其他公司一起完成了首轮融资，这些公司已经进行了令人印象深刻的融资。

我 100% 相信，短暂的开发工作区以及开发人员在设置工作区时节省的时间，将推动这项技术成为行业规范。 你看到 Uber、Shopify、Slack、Stripe 等公司使用这种形式的开发是有原因的，随着 Codespace 和 Gitpod 等产品成为常态，这些最佳实践将传播到行业的其他地方。

最后，Gitpod 整理了一套优秀的 CDE3（Cloud Developent Environment，云开发环境）原则， 我建议你看看。Redmonk 的优秀员工也分享了我的观点，他们发表了关于“云开发环境之年”的精彩文章，我建议从分析师的角度阅读。

FinOps 成为主流并向左移

几年前，Linux 基金会帮助建立了 FinOps 基金会， 以培养这一领域的创新。FinOps 基金会有了一个很好的开始，从主办第一次 FinOpsX 会议到发起 FinOps 状况调查，以及一些很棒的介绍材料。

为什么今年对 FinOps 来说是重要的一年？云支出在过去几年中大幅增加，正在成为组织内的一项大成本，有时甚至与工资不相上下。

此外，你可以将 FinOps 的就业趋势与 Google 的趋势结合起来，寻找经济增长已经到达拐点的指标。

所有这些市场压力的另一个好处是更多的标准化和开源选项，如 OpenCost。 传统上，破解云账单是一个困难的问题，如果你使用多个云，这个问题会变得更加复杂（没有涵盖所有主要云的云定价和成本管理的开放标准）。

这些市场压力加上全球经济衰退将增加大多数组织的 FinOps 实践，而不仅仅是高科技海湾地区的公司。 与过去相比，FinOps 将成为一个更大的工程问题，在过去，工程团队可以自由支配云消费。 你将看到 FinOps 信息向开发者转移，并最终成为 pull request 基础设施的一部分。

最后，成本管理和 FinOps 将默认成为可观察性解决方案的一部分（例如，Datadog 宣布推出成本管理产品）。 我预计这一领域也会出现大量整合，大型传统公司会通过收购进入该领域。

开源 SBOM 无处不在

美国政府在过去几年中制定了有关提高软件安全性的政策和法律，从 2021 年的行政命令到最近的“美国保护开源软件法案”， 涵盖了无数的安全改进。OpenSSF（Open Source Security Foundation，开源安全基金会）对该法案有一个很好的总结， 以及他们关于如何解决开源安全问题的动员计划。

此外，就在最近，美国政府通过了一项法律，强制要求医疗器械制造商必须生产 SBOM。

不可避免的是，这种趋势将继续下去，并影响开源软件，像 Kubernetes 这样的领先关键项目已经在生产供消费的 SBOM。 在所有行业强制推行 SBOM 的道路上可能会有一些坎坷，但我个人认为，通过监管或行业刚好成熟，这是不可避免的。

这一领域还将有许多新的开源创新、初创公司和项目，它们致力于聚合大量这种安全信息（参见 https://deps.dev 作为一个简单的示例）。 我个人关注的项目有 GUAC、Scorecard、 Sigstore、Witness 等等。

GreenOps 合并到 FinOps

永续发展是一个热门话题，ESG 领域的人员都知道计算基于云的工作负载的碳足迹有多复杂。普华永道最近的研究发现，“财富 1000 强企业中，60% 的企业领导者正在使用或计划使用云来增强 ESG 报告，59%的企业领导者正在使用或计划使用云来改善其 ESG 战略。”

我相信，在我们提高云的效率时，这里会出现一种杰文斯悖论的情况……此外，有趣的研究表明，“未来运行全球自动驾驶汽车车队的计算机所需的能源可能会产生与当今世界所有数据中心一样多的温室气体排放。”

在我看来，GreenOps 是一种专注于云工作负载碳足迹的 FinOps 形式。我希望这些社区合并成一个，并在该领域的开放规范和标准上进行合作， 例如扩展 OpenCost，以包括跨云的碳足迹信息。跨公司和行业的开源合作有很多机会。

GitOps 逐渐成熟，并进入生产高峰期

自从 Alexis Richardson 在 2017 年首次创造了 GitOps 这个词以来，当谈到 GitOps 工具的成熟时，这个领域的事情发生了巨大的变化。 在 CNCF，Argo 和 Flux 项目最近已经毕业， 显示了项目的稳定性和成熟的治理以及快速的采用水平。此外，它们也是 CNCF 生态系统中开源速度最高的项目。

如果你对这个领域感兴趣，我推荐你参与上面的开源项目，并加入 CNCF Open GitOps 工作组。

OpenTelemetry 日趋成熟

如果你看看来自 CNCF 的最新开源项目速度数据， OpenTelemetry 排名第二，仅次于 Kubernetes，对于这样一个年轻的项目来说，这令人难以置信。

在过去的几年里，几乎每一家主要的现代可观测性供应商都致力于集成 OTel。OTel collector 框架将供应商从实现这一功能的需要中解放出来，并使最终用户的生活变得更好。 2023 年，你不仅会看到许多技术先进的公司采用 OTel，还会看到传统的企业最终用户利用这项技术。

Backstage 开发者门户成熟度

开发人员的体验一直是那些达到一定规模以提高工程吞吐量的组织所关心的问题。随着越来越多的组织开始他们的云原生之旅，这对大多数行业来说变得非常重要。在我的上一组预测中，提到“服务目录”将成为一种必需品，但它将不止于此。

在 CNCF 社区，Backstage 是我见过的少数几个甚至在采用 Kubernetes 之前就在传统企业中部署的项目之一。 在这方面有点独特，但它确实被银行或航空公司等传统企业， 以及 Spotify 等尖端科技公司所使用。你可以从他们的 ADOPTERS.md 文件和 BackstageCon 视频中看到项目中的一些疯狂采用。

为了更上一层楼，Backstage 需要巩固它的 API，并继续培养它的插件生态系统，在本质上成为了这个领域的“Jenkins”。

关于 Backstage 和现代开发者门户的另一个有趣的事情是，Gartner 甚至已经注意到并开始在这个领域进行研究，这总是后来成熟的标志。

WebAssembly 创新+启蒙斜坡

我坚信 WebAssembly（Wasm）将在不久的将来成为计算的主导形式，它正在探索浏览器之外的用例，从边缘到服务器工作负载。 我发现 Sapphire Ventures 关于 Wasm 承诺的这篇文章是这个主题中较好的文章之一。 从个人经验来看，

我不断看到 Wasm 出现在面向未来云原生生态系统的更多领域，从 Envoy 中的插件系统重构， 或 WasmCloud 和 WasmEdge 等项目。还有，甚至 Docker 在最近的技术预览版中也支持 Wasm。

然而，随着 Wasm 用例的发现、运行时的成熟，以及技术的全面发展，将会有一些成长的烦恼。用炒作周期的说法， Wasm 将介于幻灭的低谷和启蒙的斜坡之间。虽然有很多关于 Wasm 潜力的正面报道，但当有很多像 WASI 和 tail calls 这样的移动部件没有得到完全支持时，实现这些东西是一个要面对的现实。

此外，我认为你将看到 Cloudflare 等精品云提供商和较小的初创公司为这项技术的成熟铺平道路，超大规模企业将在今年开始提供他们的第一批 Wasm 相关产品。

最后，我想明确一点，我看到了一个容器、Wasm 甚至 VM 将并存的世界……甚至我们在 Docker 的朋友也这么说。

削减成本有利于精品云（或任何“超级云”）

继续今年的成本管理主题，我相信随着组织退后一步评估他们的云使用情况，精品云提供商 （或任何超级云）将从这一趋势中受益。 有关 2023 年这一趋势的示例，请参见 Cloudflare 最近发布的公告…“ Palantir 宣布与 Cloudflare 建立战略合作伙伴关系，重点关注云成本优化”，以及他们如何将其 R2 产品与 S3 产品进行比较。

这些精品云提供商将把自己定位为关注这一特定领域的成本优化和客户服务。他们将在这一领域宣布新的收购和产品，与更大的云竞争。

Kubernetes 有其 Linux 风格的成熟时刻

不提 Kubernetes 我就做不了云原生预测，对不！？ 就在最近，我在云原生生态系统内外发布了一篇关于 2022 年开源项目 velocity 也采用 Kubernetes。还有，Kubernetes 是运行在每一间 Chick-Fil-A 餐厅，甚至有一些基于边缘的计算为你服务！甚至还有人在轨道上运行 Kubernetes，甚至在太空！

当我说 Kubenetes 正在经历其 Linux 风格的成熟时刻时，我的意思是，Linux 最初是为一个特定的业余爱好者用例而构建的， 然后最终更广泛的生态系统扩展到在电话、汽车、实时系统等等上运行。Kubernetes 项目正在经历类似的演变，组织正在扩展 Kubernetes， 以在该项目最初没有设计的新型环境中运行，如嵌入式设备。这些新的用例，将创新推回到 Kubernetes 项目和更广泛的生态系统中， 就像在 Linux 中发生的一样。开源创新的引擎已经准备好了，并将继续下去。

其它预测

- 生成式人工智能将被立法，并在开源社区中引起摩擦。围绕归属、版权和遵守开源基金会和公司政策的有趣问题将会很有趣 （例如，一些公司已经禁止使用 CoPilot 生成的代码）。我们也看到在这一领域对 CoPilot 甚至艺术版权和 Stable Diffusion 的诉讼， 这只会加速并可能导致一些版权法的变化。Heather Meeker 有一篇关于 copyright eating AI 的很棒的博文，我强烈推荐给读者。
- VSCode 将继续增长并主导 IDE 领域。这是一个非常活跃的项目，到目前为止，微软在管理社区方面做得很好。 如果你看看 Stackoverflow 的调查或 Top IDE index 的数据，就会发现 VSCode 将成为几乎所有主流编程语言的主流 IDE （甚至不包括它在 Codespaces 和 Gitpod 中的嵌入式使用）。
- RISC-V 将作为一个开源社区走向成熟，并在嵌入式和移动领域得到广泛应用。就在最近，谷歌宣布 Android 计划支持 RISC-V 作为“Tier 1”架构， 这意味着在不久的将来，你会在 Android 看到 RISC-V。世界各地还存在有利于某些地区采用 RISC-V 的地缘政治逆风。
- 游戏引擎行业的开源创新飞速发展。游戏行业与云原生世界有些不同……大多数 AAA 风格的开发仍然发生在 Windows 机器、巨大的 monorepo 以及 Unity 和 Unreal 等专有游戏引擎上。正如 a16z 在 2016 年所说…… 我们需要游戏领域更多的开源，这一点随着 Bevy、Godot、O3DE 等开源游戏引擎的出现终于实现了。

- 由于监管和安全问题的增加，OSPO 在行业和政府中不断增长。我是 TODO Group 的联合创始人之一， 该组织是开源项目办公室（Open Source Program Office，OSPO）网络的所在地，我见证了 OSPO 在高科技行业中的发展。 随着我们依赖的越来越多的软件是基于开源的，组织将需要一种战略方法来管理创新和安全风险。 此外，欧盟各国政府开始规范 OSPO，其他国家将会效仿。

最后，2023 年快乐，祝好运

我总是有一些更多的预测和趋势要分享，尤其是围绕最终用户驱动的开源、eBPF、服务网格和保护软件供应链， 但我会在今年晚些时候发布更详细的帖子，几个预测就足以开始新的一年了！无论如何，感谢你的阅读， 我希望在 CloudNativeSecurityCon 上见到大家， 当然还有我们于 2023 年 4 月在阿姆斯特丹举行的欧洲 KubeCon + CloudNativeCon 大型会议，已开放注册！

DaoCloud 公司简介

「DaoCloud 道客」云原生领域的创新领导者，成立于 2014 年底，拥有自主知识产权的核心技术，致力于打造开放的云操作系统为企业数字化转型赋能。 产品能力覆盖云原生应用的开发、交付、运维全生命周期，并提供公有云、私有云和混合云等多种交付方式。成立迄今，公司已在金融科技、先进制造、智能汽车、 零售网点、城市大脑等多个领域深耕，标杆客户包括交通银行、浦发银行、上汽集团、东风汽车、海尔集团、屈臣氏、金拱门（麦当劳）等。 目前，公司已完成了 D 轮超亿元融资，被誉为科技领域准独角兽企业。公司在北京、南京、武汉、深圳、成都设立多家分公司及合资公司， 总员工人数超过 400 人，是上海市高新技术企业、上海市“科技小巨人”企业和上海市“专精特新”企业，并入选了科创板培育企业名单。

网址：www.daocloud.io

邮件：info@daocloud.io

电话：400 002 6898

2022-11-16

DaoCloud 是 K8s 资深认证服务商

在云原生领域有这样一个词：KCSP，英文全称 Kubernetes Certified Service Provider，即 CNCF 官方认证的 Kubernetes 服务提供商。

DaoCloud 早在 2017 年就首次顺利通过了 Kubernetes 认证，是国内最早涉足并得到 CNCF 官方认可的服务商， 同时也是国内最早获得 Kubernetes Training Partner (KTP) 认证的厂商。DaoCloud 的开发者们常年积极参与 CNCF 相关社区贡献， 以引领云原生技术潮流为己任，致力于推动云原生社区发展壮大。

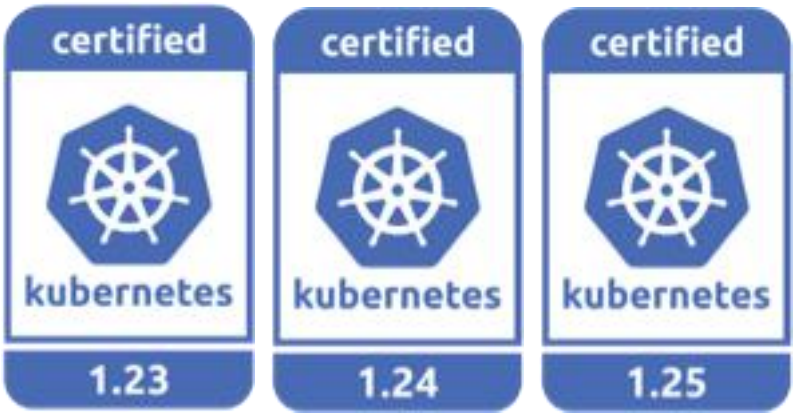
每次 Kubernetes 新版本发布，DaoCloud 都会率先适配增强新特性，并将其推向实际生产环境。 作为一家经过官方严格审核的 Kubernetes 服务商及合作伙伴，DaoCloud 在帮助企业采用 Kubernetes 成功部署大规模集群方面具有深厚的生产经验。

DaoCloud 作为一家久经认证的 KCSP 意味着：

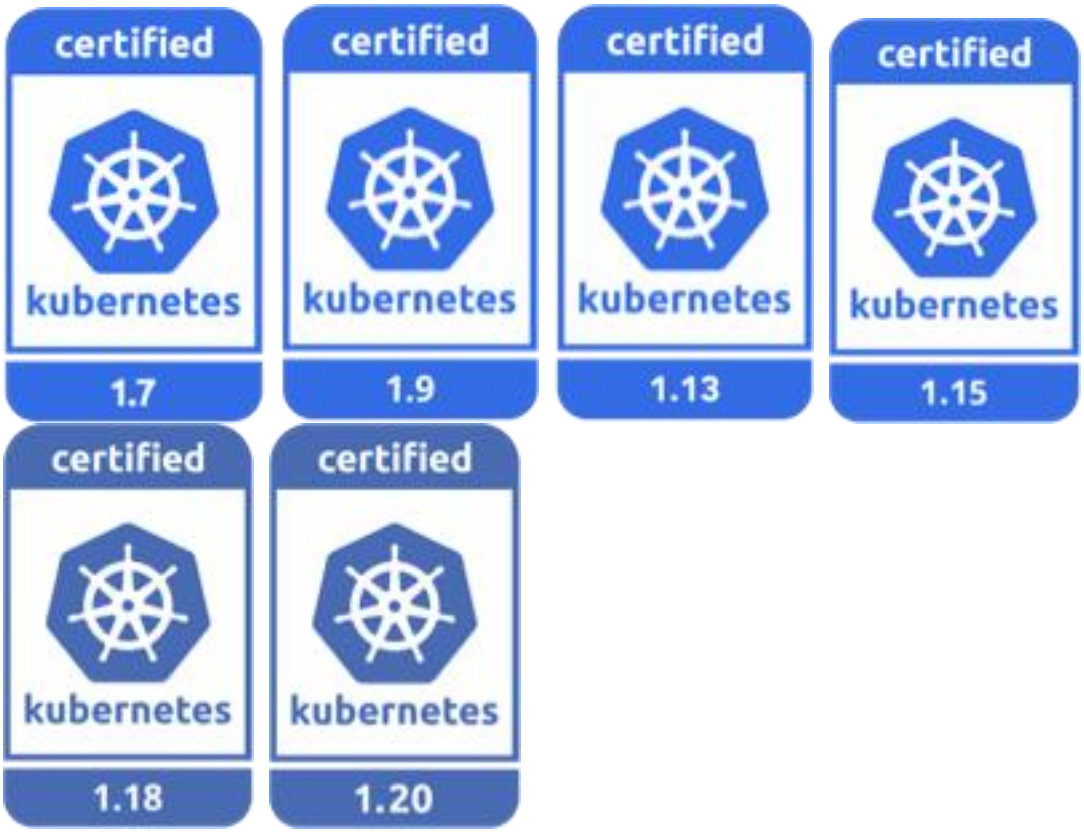
- DaoCloud 是 Kubernetes 官方推荐的资深合作伙伴
- DaoCloud 是经 CNCF 社区公认、有专家资质、能充分胜任 Kubernetes 集群部署专业工作的优良企业
- 最终用户选用 Kubernetes 部署集群时，DaoCloud 是 CNCF 云原生社区公认并推荐的品牌供应商
- 常年位列 KCSP 的 DaoCloud 掌握云原生领域领先技术，时刻引领云原生社区发展方向

目前 DaoCloud 经认证合规且完美支持的 Kubernetes 版本包括：

当前版本：



历史版本：



技术领先

DaoCloud 的研发团队崇尚技术为先，广受 CNCF 社区肯定，担任了很多云原生项目的 Member、Reviewer、Approver/Maintainer 助力社区繁荣发展，还有大咖加入 Steering Committee 负责推进各项底层特性的研发和拓展。

头衔	数量

头衔	数量
Member	52
Reviewer	15
Approver/Maintainer	6
Technical Steering Committee	1
CNCF Ambassador	1
Certified Kubernetes Administrator (CKA)	65
Certified Kubernetes Application Developer (CKAD)	26
Certified Kubernetes Security Specialist (CKS)	8

在 2022 年 8 月由 Kubernetes 官方组织的社区贡献者访谈活动中， 接见了来自亚太地区的 4 位优秀贡献者，其中 2 位都来自 DaoCloud，他们为 Kubernetes 的发展做出了卓越贡献。

DaoCloud 在推动社区技术演进的同时，对 Kubernetes 为首的云原生技术保持敏锐的嗅觉。 以 Kubernetes 1.25 为例，第一个小版本 1.25.1 于 2022-09-14 发布，DaoCloud 经过一个多月的适配调试后于 2022-10-25 提交了合规认证申请 PR， 附带 5 万多行真实的 E2E 测试报告、完善的中英文档站、详尽的资料素材， 于 2022-11-04 成功通过了 v1.25 的 KCSP 审查。

目前能对 Kubernetes 1.25 完美支持并经官方认证的 **国内厂商不超过 5 家** 。有鉴于此，华为云在元宇宙云边协同超融合一体机方面与 DaoCloud 建立战略合作关系，双方在多云编排、智能存储等方面也有深度合作关系。

DaoCloud 围绕 Kubernetes 生态自主开源的社区项目有：

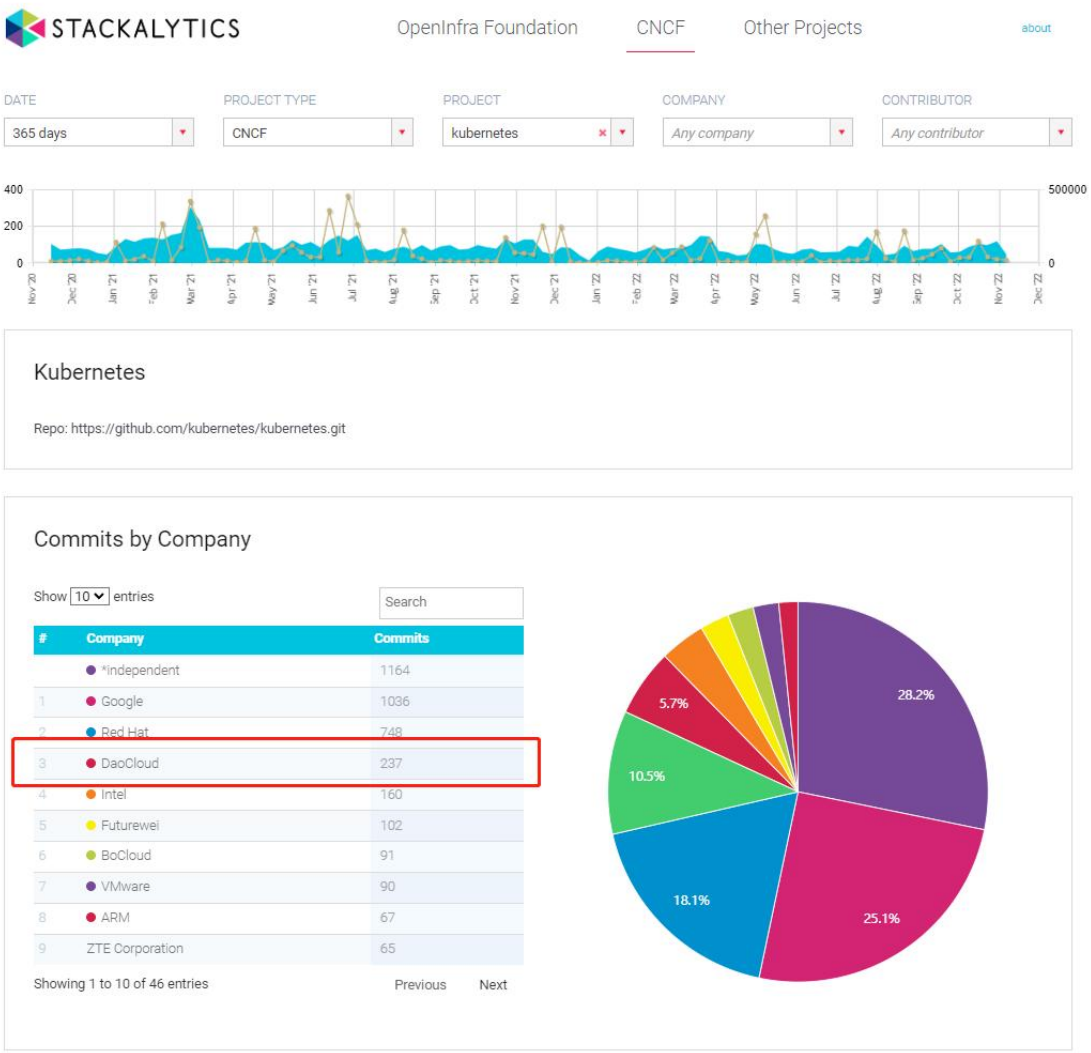
- Clusterpedia：Kubernetes 多集群资源一站式检索百科全书，已入选 CNCF 全景图和 CNCF 沙箱
- CloudTTY：社区首个开源的 Kubernetes 网页版控制台

- Ferry：Kubernetes 多集群通信组件，消除多集群复杂度，如同管理一个单集群
- HwameiStor：高可用的本地存储方案，已入选 CNCF 全景图
- KLTS：对 Kubernetes 10 多个版本的长期维护
- Kubean：基于 kubespray 构建的集群生命周期管理工具
- kwok：用一台笔记本以最小资源模拟成千上万的 kubelet
- Merbridge：使用 eBPF 加速服务网格，已入选 CNCF 全景图
- Spiderpool：云原生网络 IPAM 自动化管理软件，适用于 Underlay CNI

CNCF 贡献排名

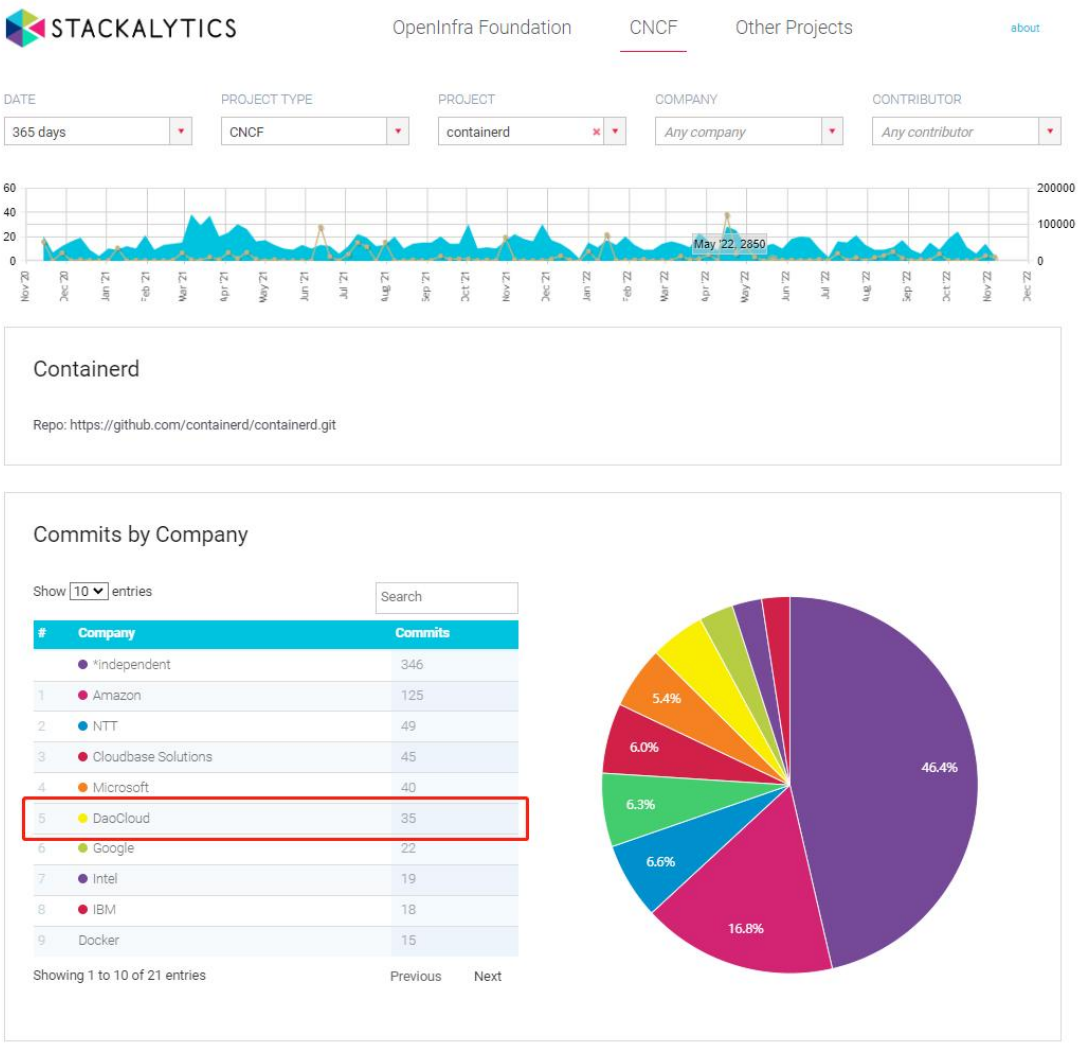
考量一家云原生企业的指标之一是社区贡献，DaoCloud 在 CNCF 社区各云原生项目的总体贡献全球排名领先，在国内首屈一指。

过去 365 天内 DaoCloud 对 **Kubernetes** 社区的贡献 **全球排名第 3**



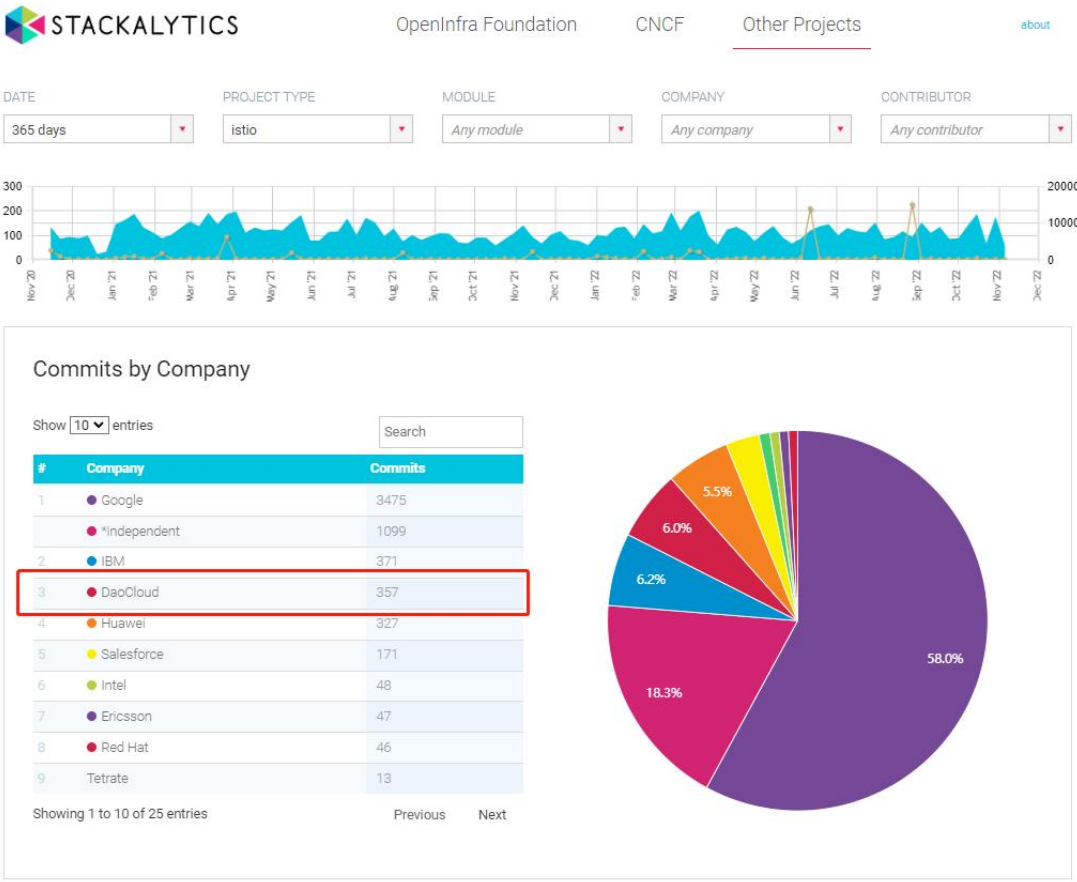
Kubernetes

过去 365 天内 DaoCloud 对 **containerd** 社区的贡献 **全球排名第 5**



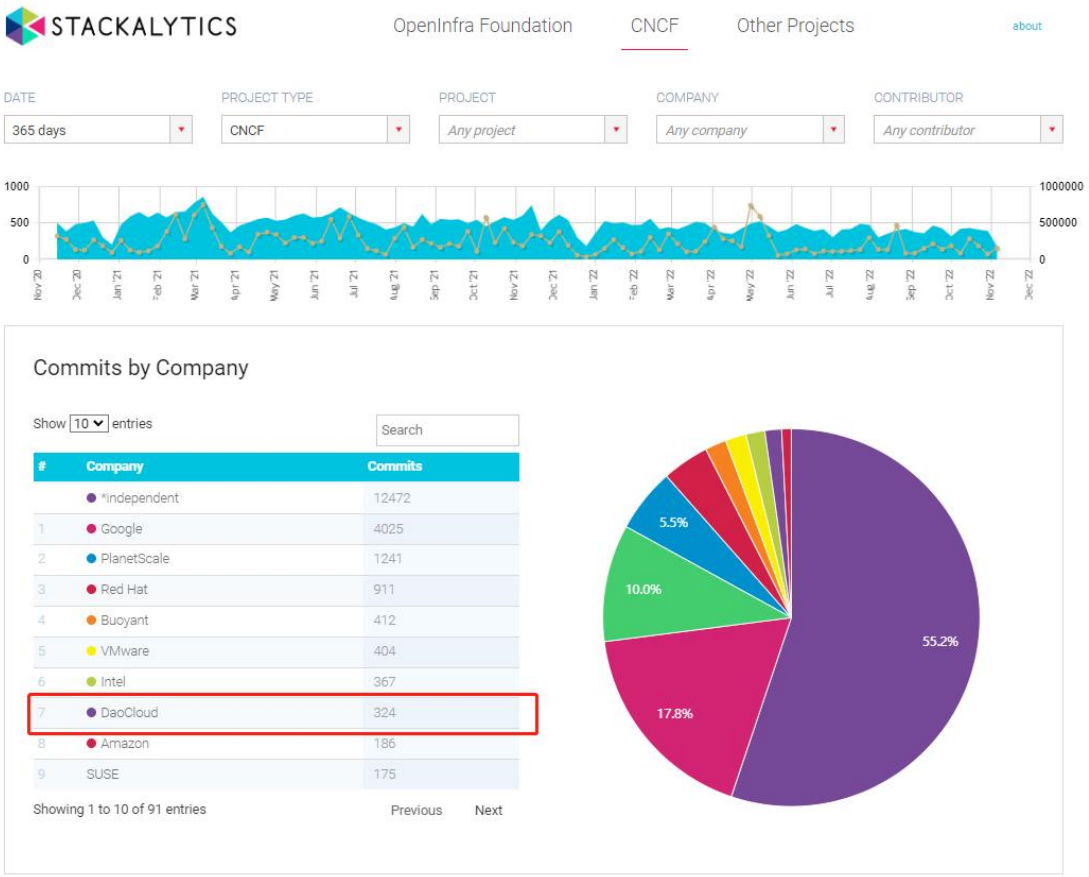
containerd

过去 365 天内 DaoCloud 对 **Istio** 社区的贡献 **全球排名第 3**



Istio

过去 365 天内 DaoCloud 对 **CNCF 社区所有项目** 的累计贡献 **全球排名第 7**



CNCF

参考链接

- https://www.cncf.io/
- https://github.com/cncf/k8s-conformance
- https://kubernetes.io/partners/
- https://www.stackalytics.io/

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

介绍 KWOK

作者: Shiming Zhang (DaoCloud), Wei Huang (Apple), Yibo Zhuang (Apple)

译者: Michael Yao (DaoCloud)

是什么项目会被 Apple、IBM、腾讯、华为纷纷使用？ 你是否曾想过在几秒钟内搭建一个由数千个节点构成的集群，如何用少量资源模拟真实的节点， 如何不耗费太多基础设施就能大规模地测试你的 Kubernetes 控制器?

如果你曾有过这些想法，那你可能会对 KWOK 有兴趣。 KWOK 是一个工具包，能让你在几秒钟内创建数千个节点构成的集群。

什么是 KWOK?

KWOK 是 Kubernetes WithOut Kubelet 的缩写，即没有 Kubelet 的 Kubernetes。 到目前为止，KWOK 提供了两个工具：

kwok
kwok 是这个项目的基石，负责模拟伪节点、Pod 和其他 Kubernetes API 资源的生命周期。

kwokctl
kwokctl 是一个 CLI 工具，设计用于简化创建和管理由 kwok 模拟节点组成的集群。

为什么使用 KWOK?

KWOK 具有下面几点优势：

- **速度**：你几乎可以实时创建和删除集群及节点，无需等待引导或制备过程。

- **兼容性**：KWOK 能够与兼容 Kubernetes API 的所有工具或客户端（例如 kubectl、helm、kui）协同作业。
- **可移植性**：KWOK 没有特殊的软硬件要求。一旦安装了 Docker 或 Nerdctl，你就可以使用预先构建的镜像来运行 KWOK。 另外，二进制文件包适用于所有平台，安装简单。
- **灵活**：你可以配置不同类型的节点、标签、污点、容量、条件等，还可以配置不同的 Pod 行为和状态来测试不同的场景和边缘用例。
- **性能**：你在自己的笔记本电脑上就能模拟数千个节点，无需大量消耗 CPU 或内存资源。

使用场景是什么？

KWOK 可用于各种用途：

- **学习**：你可以使用 KWOK 学习 Kubernetes 概念和特性，无需顾虑资源浪费或其他后果。
- **开发**：你可以使用 KWOK 为 Kubernetes 开发新特性或新工具，无需接入真实的集群，也不需要其他组件。
- **测试**：
  - 你可以衡量自己的应用程序或控制器在使用节点和 Pod 时的扩缩表现如何。
  - 你可以用不同的资源请求或限制创建大量 Pod 或服务，在集群上营造高负载的环境。
  - 你可以通过更改节点状况或随机删除节点来模拟节点故障或网络分区。
  - 你可以通过启用不同的特性门控或 API 版本来测试控制器如何与其他组件交互。

有哪些限制？

KWOK 并非试图完整替代其他什么。当然也有一些限制需要你多加注意：

- **功能性**：KWOK 不是 kubelet。KWOK 在 Pod 生命周期管理、卷挂载和设备插件方面所展现的行为与 kubelet 不同。 KWOK 的主要功能是模拟节点和 Pod 状态的更新。
- **准确性**：需要重点注意 KWOK 还不能确切地反映各种工作负载或环境下真实节点的性能或行为。 KWOK 只能使用一些公式来逼近真实的节点行为。
- **安全性**：KWOK 没有对模拟的节点实施任何安全策略或安全机制。KWOK 假定来自 kube-apiserver 的所有请求都是经过授权且是有效的。

入门指导

如果你对试用 KWOK 感兴趣，请查阅 KWOK 文档了解详情。



在终端上使用 kwokctl 的动图

欢迎参与

如果你想参与讨论 KWOK 的未来或参与开发，可通过以下几种方式参与进来：

- Slack #kwok 讨论一般用法，Slack #kwok-dev 讨论开发问题（访问 slack.k8s.io 获取 KWOK 工作空间的邀请链接）
- 在 sigs.k8s.io/kwok 上提出 Issue/PR/Discussion

我们欢迎所有想要加入这个项目的贡献者，欢迎任何形式的反馈和贡献。

参考链接

- KWOk 代码仓库： https://github.com/kubernetes-sigs/kwok
- KWOk 文档： https://kwok.sigs.k8s.io/
- KWOK slack 频道： https://slack.k8s.io/

尝鲜：K8sGPT 赋能 Kubernetes

文章来源于 Google Cloud， 作者 Jasbir Singh，译者 windsonsea

K8sGPT 是一个非常新潮的云原生智能工具，它为 Kubernetes 云原生软件工程师（SRE）赋予了超能力。 提供了一种简单高效的方式来扫描 Kubernetes 集群并以简单的英文句子诊断集群、节点、Pod 的相关问题。 该工具旨在将工程师们的经验编码到其分析器中，从而帮助提取最相关的信息，并通过人工智能丰富了各种诊断和分析场景。

K8sGPT 将 ChatGPT 的能力引入 K8s 社区，它让用户能够智能化分析集群中的问题，并提供人性化的修复建议。

前提条件

1. 确保已正确安装 K8sGPT，请按照安装指南来安装。
2. 您需要连接到一个 Kubernetes 集群（例如 Minikube、Kind、GKE、EKS、AKS、DIY K8S 集群）。

安装 K8sGPT

安装 K8sGPT 可以采用多种方式，但其中一种简单方法是运行 brew 命令。 您可以参考官方仓库以获取适用于您平台的相关命令。

```
brew tap k8sgpt-ai/k8sgpt
brew install k8sgpt
```

快速入门

准备好开始使用 K8sGPT 分析 Kubernetes 的配置了吗？参照以下步骤快速开始：

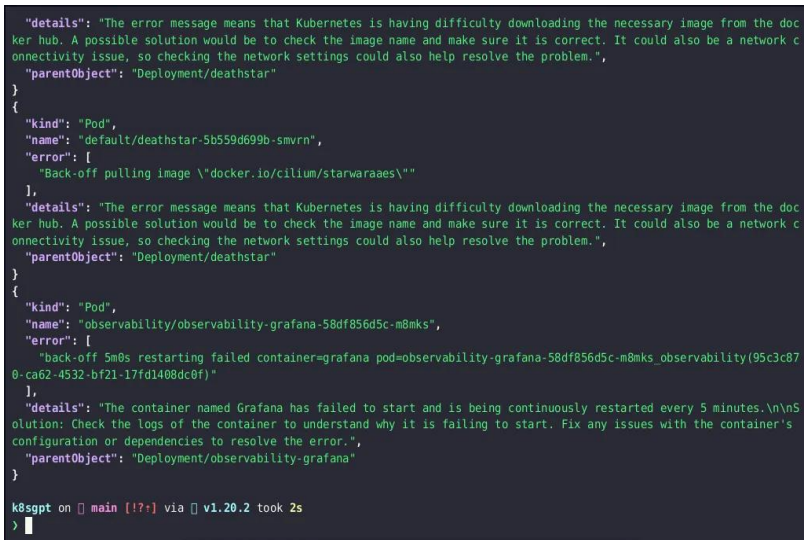
1. 运行命令 k8sgpt generate 来生成 OpenAI 的 API 密钥。 这将在浏览器中打开一个链接以生成 API 密钥。目前，默认的 AI 提供商是 OpenAI，但您可以在将来切换到 Bard 或其他 AI 提供商。

2. 生成 API 密钥后，运行 k8sgpt auth 将其设置到 K8sGPT 中。 还可以使用 --password 参数直接提供 API 密钥。
3. 如果想管理分析器使用的活动过滤条件，请运行 k8sgpt filters。 默认情况下，在分析期间执行所有过滤条件。
4. 运行 k8sgpt analyze 开始扫描。它将分析 Kubernetes 配置并提供发现的任何问题的摘要。
5. 要获得更详细的问题说明，使用 k8sgpt analyze 命令的 --explain 参数。

就是这样！通过这些简单的步骤，您可以快速开始使用 K8sGPT 分析 Kubernetes 配置，确保您的集群安全且性能优化良好。

可以运行以下命令，并以 JSON 格式查看结果，以便更好地理解：

k8sgpt analyze -o json - explain - filter=Pod | jq .



json

K8sGPT 配备了一系列分析器，帮助您检测和解决 Kubernetes 集群中的问题。这些分析器默认包含在 Kubernetes 平台中，并且其中一些甚至会自动启用。 内置的分析器包括 podAnalyzer、pvcAnalyzer、rsAnalyzer、serviceAnalyzer、eventAnalyzer 和 ingressAnalyzer。 此外，您还可以选择添加可选的分析器，例如 hpaAnalyzer 和 pdbAnalyzer。 另外，您甚至可以创建自己的分析器以满

足特定需求。借助于 K8sGPT 的综合分析器集合，您可以确保集群始终平稳高效运行。

```
Usage:
  k8sgpt [command]
Available Commands:
  analyze      This command will find problems within your Kubernetes cluster
  auth         Authenticate with your chosen backend
  completion   Generate the autocompletion script for the specified shell
  filters      Manage filters for analyzing Kubernetes resources
  generate     Generate Key for your chosen backend (opens browser)
  help         Help about any command
  version      Print the version number of k8sgptFlags:
       --config string          config file (default is $HOME/.k8sgpt.git.yaml)
  -h, --help                    help for k8sgpt
       --kubeconfig string      Path to a kubeconfig. Only required if out-of-clust
er.
       --master string          The address of the Kubernetes API server. Override
s any value in kubeconfig. Only required if out-of-cluster.
  -t, --toggle                  Help message for toggleUse "k8sgpt [command] --
help" for more information about a command.
```

在 K8sGPT 中，过滤条件用于管理要分析的资源。您可以通过运行命令 k8sgpt filters list 列出可用的过滤条件。 默认过滤条件已经启用，但您可以根据需要添加或删除过滤条件。要添加过滤条件，请运行 k8sgpt filters add， 然后加上您要添加的过滤条件。多个过滤条件之间可以用逗号分隔。要删除过滤条件，请运行 k8sgpt filters remove，然后加上您要删除的过滤条件。

```
k8sgpt filters list
```

添加一个过滤条件：

```
k8sgpt filters add Service
```

添加多个过滤条件：

```
k8sgpt filters add Ingress,Pod
```

移除一个过滤条件：

```
k8sgpt filters remove Service
```

移除多个过滤条件：

```
k8sgpt filters remove Ingress,Pod
```

要使用默认的分析器运行扫描并获取问题的详细解释，请执行以下操作：

```
k8sgpt generate
k8sgpt auth
k8sgpt analyze --explain
```

内置分析器

默认启用的分析器

- podAnalyzer
- pvcAnalyzer
- rsAnalyzer
- serviceAnalyzer
- eventAnalyzer
- ingressAnalyzer
- statefulSetAnalyzer
- deploymentAnalyzer
- cronJobAnalyzer
- nodeAnalyzer

可选的分析器

- hpaAnalyzer
- pdbAnalyzer

- networkPolicyAnalyzer

用“Service”这类特定的资源过滤结果：

```
k8sgpt analyze --explain --filter=Service
```

用“default”这类特定的命名空间过滤结果：

```
k8sgpt analyze --explain --filter=Pod --namespace=default
```

以 JSON 格式输出结果：

```
k8sgpt analyze --explain --filter=Service --output=json
```

匿名分析

安全性是首要任务，因此我们为您提供了匿名化功能。这个巧妙的匿名技巧将敏感数据（如 Kubernetes 对象名称和标签） 发送到 AI 后端进行分析之前对其进行屏蔽。

这意味着您的数据将安全可靠地保护起来，没有人能窥探不该看的东西。

以下是如何在 K8sGPT 中使用匿名化功能：

```
k8sgpt analyze --explain --anonymize
```

在分析过程中，K8sGPT 会检索敏感数据，然后对其进行屏蔽，然后再发送到 AI 后端。 后端接收到屏蔽后的数据，进行处理，并向用户返回解决方案。

一旦解决方案返回给用户，屏蔽的数据会被实际的 Kubernetes 对象名称和标签所替换。

注意：**匿名化不适用于事件。**

集成命令

您知道吗，您可以将其他工具与 K8sGPT 集成起来？通过 K8sGPT 集成命令，您可以轻松添加和删除这些有用的集成组件。

首先，让我们使用以下命令列出所有可用的集成组件。

```
k8sgpt integrations list
```

接下来，通过运行 k8sgpt integration activate 命令激活一个集成组件，然后输入您想要的集成组件名称。 例如，trivy 是一个流行的集成组件，它将在集群上安装 Trivy 的 Helm chart。

```
k8sgpt integration activate trivy
```

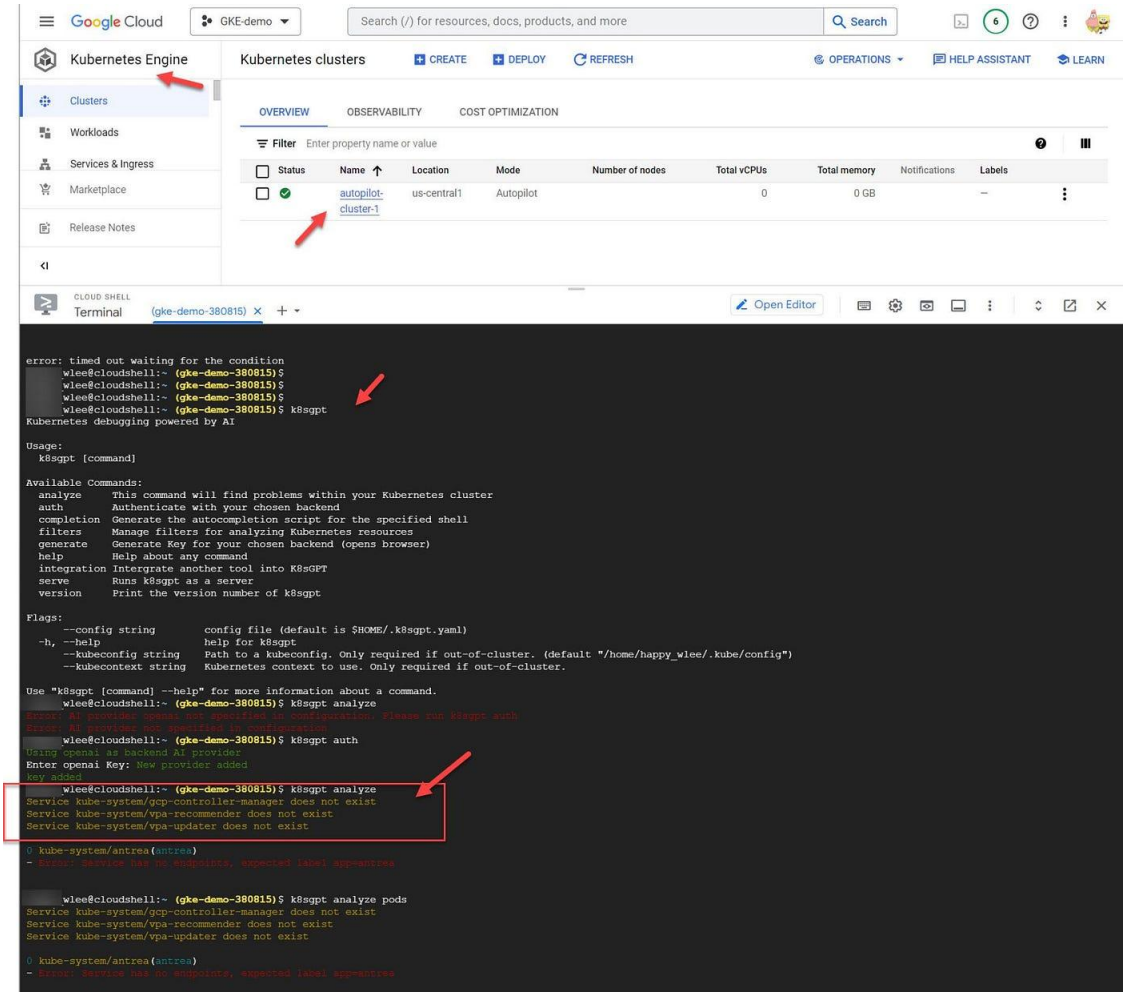
客官请稍候，集成组件还提供了更多可能！ 您甚至可以在分析中使用这些集成组件，只需在 --filter 参数中指定这些集成组件即可。 所以如果您正在使用 Trivy，只需运行以下命令，就可以看到魔法了。

```
k8sgpt analyze --filter VulnerabilityReport
```

如果您需要删除一个集成组件，只需运行以下命令，后面加上集成组件的名称。

```
k8sgpt integration deactivate trivy
```

在 GKE 上使用 K8sGPT



gke

请参考这个 GitHub 仓库 https://github.com/k8sgpt-ai/k8sgpt ，了解更多可用功能的细节，并进行实际操作以体验 K8sGPT 的魅力。

近两年功能增加最多！Kubernetes 1.27 正式发布



k8s 1.27 release

太平洋时间 2023 年 4 月 11 日，Kubernetes 1.27 正式发布。此版本距离上版本发布时隔 4 个月，是 **2023 年的第一个版本。**

新版本中 release 团队跟踪了 **60 个 enhancements**，比之前版本都要多得多。其中 13 个功能升级为稳定版，29 个已有功能进行优化升级为 Beta，另有 18 个 Alpha 级别的功能， 大多数为全新功能。**本版本包含了很多重要功能以及用户体验优化** ，本文会在下一小节进行部分重要功能的详细介绍。

重要功能

镜像仓库从 k8s.gcr.io 切换到 registry.gcr.io

自 Kubernetes 1.25 版本起，Kubernetes 容器镜像仓库域名已经从 k8s.gcr.io 更改为 registry.k8s.io。 2023 年 3 月 20 日之后，k8s.gcr.io 将不再继续直接维护，而是被代理到 registry.k8s.io。

KEP-1847：StatefulSet PVC 自动删除功能特性 Beta

在 v1.23 中引入的 **StatefulSetAutoDeletePVC 功能将在 1.27 版本中升级为 Beta，并默认开启。** 然而，默认开启并不意味着所有 StatefulSet 的 PVC 都将自动删除。

用户可以配置 whenDeleted 或 whenScaled 阶段以触发 Retain 或 Delete 行为。 其中，Retain 是默认行为，只有配置了 Delete 策略的 StatefulSet 在被删除时才会触发对应的 PVC 删除动作。

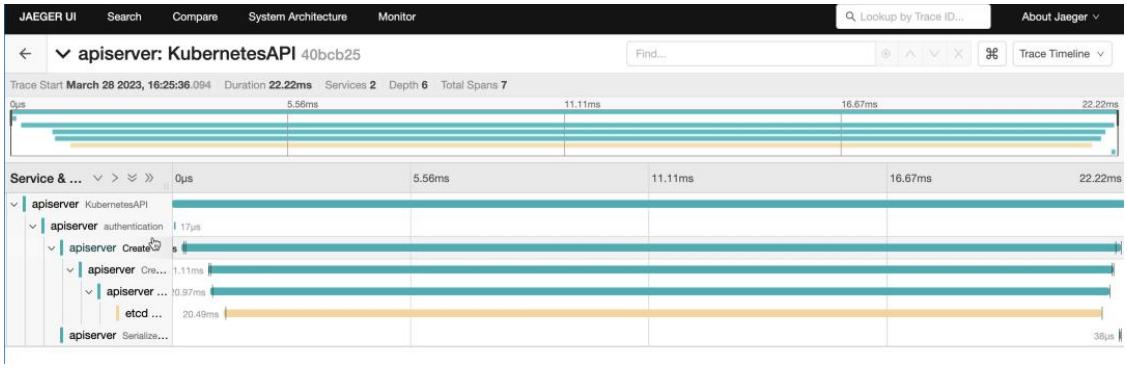
KEP-3453：优化大型集群中 kube-proxy 的 iptables 模式性能

功能 MinimizeIPTablesRestore 在 1.26 版本中引入，并在 1.27 版本中升级为 Beta 并默认启用。 该功能旨在改善大型集群中 kube-proxy 的 iptables 模式性能。

如果您遇到 Service 信息未正确同步到 iptables 的问题，您可以通过将 kube-proxy 启动参数设置为 --feature-gates=MinimizeIPTablesRestore=false 来禁用该功能（并向社区提交问题）。 您还可以查看 kube-proxy 的 metrics 信息中的 sync_proxy_rules_iptables_partial_restore_failures_total 指标来监控规则同步失败的次数。

KEP-2831 和 KEP-647：APIServer 和 Kubelet 的 Tracing 功能 Beta

APIServerTracing 已升级为 Beta 状态，并默认启用。 目前仅支持 kube-apiserver 和 etcd 组件的 Tracing，但未来会添加对 client-go 的支持。 其他组件也将逐步添加 Tracing 能力。kube-apiserver 中的 Tracing 需要指定配置文件才能启用，并且必须指定接收端。



APIServerTracing

Kubelet 和容器运行时通过 CRI 调用的 Tracing 也已经默认开启。 类似地，kubelet 的 KubeletTracing 功能已经默认开启，但是需要配置 Tracing 的接收端才能正常工作。

KEP-3077：上下文日志

上下文日志可以帮助用户理解日志的上下文信息，**更好地让日志帮助用户排错和理解，提升日志的可观测性。** 目前 kube-controller-manager 的迁移工作已经完成，kube-scheduler 的迁移工作将在 1.28 版本中完成。

KEP-1287：Pod 资源的纵向弹性伸缩

Kubernetes 现已支持原地调整 Pod 资源大小， 容器组重建不再是必须的，这很大程度上缓解了横向弹性的冷启动等问题。

- 在之前的版本中，Pod API 不支持修改资源。也就是说，容器定义的资源限制和请求（如 CPU 和内存）是不可变的。 在 1.25 版本中，CRI API 开始支持 Pod 资源限制的热更新。
- 在 Pod 的容器中添加了 resizePolicy 字段，以允许用户控制容器在资源变更时是否重启。
- 在容器状态中添加了 allocatedResources 字段，用于描述为 Pod 分配的节点资源。
- 在容器状态中添加了 resources 字段，用于报告应用于正在运行的容器的实际资源。

- 在 Pod 状态中添加了 resize 字段，用于描述请求调整 Pod 大小的状态。该字段可以是 Proposed（已提出）， InProgress（进行中），Deferred（已延迟）或 Infeasible（不可行）。

KEP-3386：Kubelet 事件驱动 PLEG 升级为 Beta

在节点 Pod 较多的情况下，通过容器运行时的 Event 驱动 Pod 状态更新，能够有效地提升效率。 在 1.27 中，该功能已经达到了 Beta 条件，基础的 E2E 测试任务已经添加。 之所以默认关闭该功能，是因为社区认为该功能还需要补充以下验证：压力测试、恢复测试和带退避逻辑的重试。

1. 压力测试需要在单个 Pod 中创建大量容器以生成 CRI 事件，并观察 latency 值是否超过 1 秒。
2. 恢复测试则是为了验证 Kubelet 在重新启动后能否正确地更新容器状态。
3. 而带退避逻辑的重试则是为了解决 CRI Runtime 宕机时 Kubelet 可能无法连接的问题。

KEP-3476：Volume Group 快照 Alpha（API）

能够在 Pod 的所有卷上同一时间快照，将成为容灾备份和故障恢复场景的重大技术突破。 现在，您不必担心应用程序因备份的卷存在几秒钟差异而无法正确运行。 此外，在安全研究方面，存储卷的组快照功能也将是一个重大变革。 排查问题时，您的快照和 Pod 的状态是可对照的。 需要注意的是该功能并非在 Kubernetes 仓库，VolumeGroupSnapshots API 的定义目前维护在 https://github.com/kubernetes-csi/external-snapshotter

KEP-3838 和 KEP-3521：Pod 调度就绪态功能增强

调度就绪态功能 PodSchedulingReadiness，在 v1.26 作为 Alpha 功能引入，**从 v1.27 开始该功能升级为 Beta，默认开启。**

该功能在 Pod 对象中引入了 .spec.schedulingGates 字段，用来定义 Pod 是否允许开始调度。 此功能最常见的一个场景是：批量调度，一组 Pod 需要等待集群资源满足整组 Pod 同时启动，才能开始调度。

针对受 SchedulingGate 限制悬停（Gated）状态的 Pod，为了让第三方控制器更灵活地控制这些 Pod 的最终调度策略， 新版本中允许 Gated Pod 的 NodeAffinity 和 NodeSelector 可以被修改，但仅允许缩小节点范围。 缩小范围是指支持添加新策略，不能删除或者修改已有策略。

KEP-3243：Deployment 滚动更新过程中的调度优化

PodTopologySpread 调度策略之前只关注标签的键，而不关注标签的值，因此在滚动更新 Deployment 时， 调度无法区分新旧实例，进而导致可能新实例调度不均匀。例如，滚动更新的最后一组 4 个旧 Pod 都在同一个节点， 那么新 Pod 调度为了更加均匀分布，大概率会调度到其他节点；滚动最后删除这一组旧 Pod 后，有可能一个节点没有调度该 Deployment。

在 v1.27 中，PodTopologySpread 调度策略可以区分调度 Pod 标签的值 （这里通常指 Pod 的 pod-template-hash 标签，不同 replica set 对应的 Pod 该标签的值不同）， 这样滚动更新后，**新的 Pod 实例会被调度得更加均匀** 。

KEP-2876：使用通用表达式语言（CEL）来验证 CRD

CustomResourceValidationExpressions 在 v1.25 中就已经升级为 Beta。 验证规则使用通用表达式语言（CEL）来验证定制资源的值。在 v1.27 中，ValidationRule 新增了字段 messageExpression，可以更好地展示提示信息。在之前版本中，只支持固定的失败信息。

```
x-kubernetes-validations:
- rule: "self.x <= self.maxLimit"
  messageExpression: '"x exceeded max limit of " + string(self.maxLimit)'
```

KEP-2258：节点日志查询

v1.27 添加了 NodeLogQuery 特性门控（Feature Gate)， 为集群管理员提供了使用 kubectl 流式查看节点日志的功能，无需登录节点。 该功能目前是 Alpha，需要配置 kubelet 开启特性门控，同时还需要设置 enableSystemLogHandler 和 enableSystemLogQuery 为 true。在 Linux 上， 我们假设系统服务日志可以通过 journald 获得。在 Windows 上， 我们假设系统服务日志可以在应用程

序日志提供程序中获得。在这两个操作系统中， 还可以通过读取 /var/log/ 目录下的文件来获取日志。此功能对 Windows 的支持也在逐步完善， 目前使用 Get-WinEvent 来获取系统和应用程序日志。

不仅如此，目前节点日志获取功能还支持了一些参数。其中 query 表示服务名，可指定 kubelet、containerd 等。patern 通过提供的 PERL 兼容正则表达式过滤日志条目。 此外支持的参数还有 sinceTime、untilTime、tailLines、boot。

```
# Fetch kubelet logs from a node named node-1.example that have the word "e
rror"
kubectl get --raw "/api/v1/nodes/node-1.example/proxy/logs/?query=kubelet&pa
ttern=error"
```

KEP-3659：kubectl apply –prune 重新设计

–prune 在 v1.5 就作为 Alpha 功能引入，提供了自动清理 apply yaml 删除的部分对象， 但是这个过程有些性能问题和缺陷，在一些情况下会造成对象泄漏。常见原因包括 allowlist （之前叫 whitelist）GVK 内容和 apply 内容不匹配，或者命名空间变化。命名空间操作变化的案例， 比如第一次 apply 操作了命名空间 A 和 B；而第二次 apply 如果只 apply 命名空间 A 的资源，那么命名空间 B 的资源将不会被清理。

在 v1.27 中，kubectl 对 apply –prune 的功能进行了重新的设计，**增加了 –applyset 配合使用，目前该功能仍然是 Alpha 级别** ， 因此需要配置 KUBECTL_APPLYSET 环境变量为 1 或者 true，才能启用。在 kubectl apply 时，kubectl 会给对象添加 applyset.kubernetes.io/part-of 标签，在清理时会使用该标签。而为了满足更复杂的场景，还引入了 applyset.kubernetes.io/id 来标识 Parent 对象，以及 toolling、additional-namespaces 来帮助区分对象和增加命名空间信息等，更多详情请阅读 KEP 内容。

DaoCloud 参与功能

本次发布中， DaoCloud 重点贡献了 **sig-node，sig-scheduling 和 kubeadm 相关内容** ，具体功能点如下：

- [client-go] 修复尝试获取 leader lease 的等待时间的问题
- [调度] MinDomainsInPodTopologySpread 功能升级为 Beta
- [调度] 当任何调度程序插件在 PostFilter 中返回 unschedulableAndUnresolvable 状态时，该 Pod 的调度周期立即终止
- [日志] 迁移控制器使用 contextual logging
- [kubeadm] Kubeadm: 添加特性门控 EtcdLearnerMode，它允许将新增的控制节点的 etcd 作为学习者 Learner 加入，然后再升级为投票成员
- [节点] 如果 Pod 的 spec.terminationGracePeriodSeconds 属性值是负数，则会被修改为 1 秒的 terminationGracePeriodSeconds
- [节点] 添加了一个可以限制节点进行并行镜像下载数量的新功能
- [节点] 改进目前 Memory QoS 功能，优化了其在 cgroup v2 场景的适配性
- [节点] Kubelet：将“—container-runtime-endpoint”和“—image-service-endpoint”迁移到 kubelet 配置中
- [节点] Kubelet 默认允许 Pod 使用 net.ipv4.ip_local_reserved_ports sysctl，要求内核版本 3.16+
- [CLI] kubectl.kubernetes.io/default-container 标签正式 GA，主要用于 kubectl 的 logs、exec 等命令来决定默认容器

在 v1.27 发布过程中，DaoCloud 参与 **上百个问题修复和功能研发** ，作为作者约有 **90 个提交** ， 详情请见贡献列表（该版本的两百多位贡献者中有来自 DaoCloud 的 15 位）。在 Kubernetes v1.27 的发布周期中， DaoCloud 的多名研发工程师取得了不少成就。

- wzshiming 主要维护的项目 KWOK (Kubernetes Without Kubelet) 成为社区热点，并在大规模集群模拟方面有效地节约资源，提升效率。
- windsonsea 几乎包揽了近期官网博客的翻译，并成为 SIG-docs-zh 的维护者。
- mengjiao-liu 也是 SIG-docs 的资深中英维护者。

- kerthcet 在即将召开的 2023 年欧洲 KubeCon 上，将分享两个有趣的调度方向的主题，分别是“Sig Scheduling Deep Dive”和“Building a Batch System for the Cloud with Kueue”（属于 Kubernetes Batch + HPC Day）。
- pacoxu 将分享“Kubeadm Deep Dive”的主题。

其他需要了解的功能

APPS:

- PodDisruptionBudget 之前不支持指定不健康 Pod 的处理方法，不健康 Pod 是指 Pod Running 但是状态不是 Ready。我们添加了一个新字段 unhealthyPodEvictionPolicy，允许用户指定这些不健康的 Pod 应该发生什么。该字段在 v1.27 中升级为 Beta。
- “StatefulSetStartOrdinal”功能升级为 Beta，默认允许在 StatefulSet 中配置起始序号。
- CronJob 支持 Timezone 功能进阶至 GA。
- StatefulSetStartOrdinal 特性门控进阶至 Beta，DownwardAPIHugePages kubelet 功能已进阶至 GA。
- Indexed Job 的 API 验证已放宽，允许通过同时更改 parallelism 和 completions 来扩展或者缩小 Indexed Job，但是需要保持 parallelism == completions 同步修改。

API:

- 基于 Kubernetes v1.25 提供的 KEP-2876 CRD 验证表达式语言，该功能增加一个新的资源 —— ValidatingAdmissionPolicy，允许在不使用 Validation Webhook 时实现字段验证。
- 在 1.27 中，Kubernetes 为聚合发现提供了 Beta 支持，通过 /api 和 /apis 发布集群支持的所有资源，而不是每个 Group 分别提供。
- OpenAPIV3 功能 GA，允许 API 服务器发布 OpenAPI V3。社区建议使用 OpenAPI v3，v3 有诸多优势， 其中包括 CustomResourceDefinition

OpenAPI v3 验证模式的无损表示，而 OpenAPI v2 在 CRD validation 中做了有损转换。 kubectl explain 也已经支持了 OpenAPI v3，但是需要配置环境变量 KUBECTL_EXPLAIN_OPENAPIV3 来启用。

- 将 SelfSubjectReview 提升为 Beta 级别。

Auth:

- KMSv2 升级为 Beta，该功能在 1.27 中做了许多优化，比如：在插件 key ID 不变的情况下， 重用 DEK 数据加密密钥，而当 Server 启动时，DEK 会重新随机生成。
- 添加了一个新的 Alpha API：ClusterTrustBundle （certificates.k8s.io/v1alpha1）。
- AdmissionWebhookMatchConditions 功能已进入 Alpha：在 v1Beta 和 v1 API 中， 为 ValidatingWebhookConfiguration 和 MutatingWebhookConfiguration 添加了 MatchConditions 字段。
- 将 LegacyServiceAccountTokenTracking 功能升级为 Beta，用于跟踪基于 Sercet 的 SA token 的使用情况。

CLI:

- kubectl 的 —subresource 支持升级为 Beta，目前 subresource 只支持 status 和 scale。
- 改进 kubectl 插件解析以支持 non-shadowing 子命令。需要配置环境变量 KUBECTL_ENABLE_CMD_SHADOW=true 开启该功能，此时例如 kubectl create foo 执行会首先发现 create 没有 foo 子命令，kubectl 会自动尝试运行 kubectl-create-foo 插件。

网络：

- 当外部 cloud provider 支持提供双栈 IP 时，在 kubelet 中启用 CloudNodeIPs 功能， 您就可以指定双栈的 —node-ip。该功能目前是 Alpha，需要手动开启。ValidatingAdmissionPolicy 添加了

matchConditions 字段，用来支持基于 CEL 的自定义匹配条件。该功能目前仍然是 Alpha 。

- 允许动态扩展可用于服务 Service 的 IP 数量。新增了 MultiCIDRServiceAllocator 功能，目前是 Alpha 级别。
- 新功能 ServiceNodePortStaticSubrange，以启用新的策略在 NodePort 服务端口分配器中， 因此节点端口范围被细分，并且首选从上部分配动态分配的 NodePort 端口为服务。
- 添加了有关工作负载资源（Pod、ReplicaSets、Deployments、Jobs、CronJobs 或 ReplicationControllers）名称无效 DNS 标签的警告。

弹性：

- HPAContainerMetrics 升级为 Beta，该功能允许 HorizontalPodAutoscaler 基于目标 Pod 中各容器 ContainerResource 类型的 metrics 来执行扩缩操作。

节点：

- 动态资源分配功能，使用功能 DynamicResourceAllocation。新增的 API 比 Kubernetes 现有的设备插件 Device Plugin 功能更加灵活。因为它允许 Pod 请求（声明）指定类型的资源，这些资源可以在节点级别、集群级别或任何其他用户自定义实现的模型中使用。
- 用户命名空间支持范围扩大，该功能仍然是 Alpha，但相比 v1.26 支持 StatefulSet。
- GRPC 探针功能 GA。
- Bump default API QPS limits for Kubelet.
- Graduate Kubelet Topology Manager to GA.
- 提高 Kubelet 的默认 API QPS 限制，其中 kubeAPIQPS 和 kubeAPIBurst 都调大了 10 倍。
- Kubelet 的拓扑管理器 Topology Manager 功能 GA。
- seccomp profile 默认值升级至 GA 级别。

日志：

- kube-proxy、kube-scheduler 和 kubelet 有 HTTP API，可以在运行时更改日志 Level。该功能也适用于 JSON 格式日志输出。

Metrics:

- /metrics/slis 现在可用于控制平面组件，可以用来获取当前组件的健康检查指标。

Lease:

- Kubernetes 组件选举现在仅支持使用 Lease。

调度：

- 调度器新增 Metric plugin_evaluation_total。该指标显示特定插件影响调度结果的次数。
- 调度框架在 Filter 和 Score 阶段可以利用 Skip 状态跳过该流程，以提升性能。 在 PreFilter 阶段，如果 Plugin 返回 Skip 信息，那么在后续可以跳过执行该 Plugin 相应的 Filter 和 Score 阶段。

存储：

- NewVolumeManagerReconstruction 功能升级为 Beta。这是 VolumeManager 的重构， 允许 kubelet 在启动期间带上关于现有卷如何挂载的附加信息。
- SELinuxMountReadWriteOncePod 功能升级为 Beta。该功能在卷挂载过程中使用正确的 SELinux 标签， 相比逐个递归更改每个文件的方式，该功能加快了容器启动速度。
- ReadWriteOncePod PV 访问模式功能升级为 Beta。此功能引入了一个新的 ReadWriteOncePod 访问模式， 用于限制 PV 对单个节点上的单个 Pod 的访问。而 ReadWriteOnce 模式限制了单节点访问，但并不限制同一个节点的多个 Pod 同时访问。

- CSINodeExpandSecret 功能升级为 Beta 级别。

版本标志

本次发布的主题是 。Kubernetes 是“Chill Vibes”放松的氛围。



Chill Vibes

这个灵感来源于最近一次发布过程中，在功能冻结后没有出现任何异常情况，这是我们在多次发布中唯一的一次。 我们能够更加轻松地完成此版本的发布，原因在于幕后工作人员为改善发布管理付出了许多努力。 这也正是 **本主题希望庆祝的内容：大家在为提升社区体验方面做出了很多努力。**

特别感谢 Britnee Laverack 创造标志设计图案，并且她还设计了 Kubernetes 1.24: Stargazer 的标志设计图案。

升级注意事项

本节主要介绍 v1.27 中 API 变化，以及功能的移除以及废弃，废弃的功能通常会在 1–2 个版本之后移除。 更多详情请查看 Kubernetes 在 v1.27 中移除的特性和主要变更。

Kubernetes 在 v1.27 中移除的特性和主要变更：https://kubernetes.io/zh-cn/blog/2023/03/17/upcoming-changes-in-kubernetes-v1-27/

其中 **需要重点关注的，IPv6DualStack 外部云供应商特性门控已被删除。** （该功能在 1.23 版本中成为 GA，几个版本之前已删除所有其他组件的特性门控。）如果您仍然手动启用它，则必须立即停止。

k8s.gcr.io 重定向到 registry.k8s.io 相关说明

再次强调，Kubernetes 项目为了托管其容器镜像，使用社区拥有的一个名为 registry.k8s.io. 的镜像仓库。 从 3 月 20 日起，所有来自过期 k8s.gcr.io 仓库的流量将被重定向到 registry.k8s.io。已弃用的 k8s.gcr.io 仓库未来最终将被关停。

其他需要注意的变化

- CSIStorageCapacity 的 storage.k8s.io/v1beta1 API 版本在 v1.24 中已被弃用，将在 v1.27 中被移除。
- 移除 NetworkPolicyEndPort、LocalStorageCapacityIsolation、StatefulSetMinReadySeconds、 IdentifyPodOS、DaemonSetUpdateSurge、EphemeralContainers、CSIInlineVolume、CSIMigration、ControllerManagerLeaderMigration 特性门控，这些特性大部分都是在 v1.25 之前的版本正式 GA。
- kube-apiserver 移除了 —master-service-namespace 命令行参数。

- kube-controller-manager 命令行参数 --enable-taint-manager 和 --pod-eviction-timeout 被移除。
- kubelet 移除了命令行参数 --container-runtime，该参数目前只有一个可选值“remote”并在之前版本中废弃。
- 弃用了 Alpha 状态的 seccomp 注解 seccomp.security.Alpha.kubernetes.io/pod 和 container.seccomp.security.Alpha.kubernetes.io。
- SecurityContextDeny 特性门控已经废弃，将在未来版本移除。
- 由于使用 golang 1.20.2 构建时遇到问题，Linux/arm 将不会在 Kubernetes 1.27 中发布。注意， arm64 仍然支持。

历史版本

- **Kubernetes 正式发布 v1.26，稳定性显著提升**
- **Kubernetes 1.25 正式发布，多方面重大突破**
- **Kubernetes 1.24 走向成熟的 Kubernetes**
- **Kubernetes 1.23 正式发布，有哪些增强?**
- **Kubernetes 1.22 颠覆你的想象：可启用 Swap，推出 PSP 替换方案，还有……**
- **Kubernetes 1.21 震撼发布 | PSP 将被废除，BareMetal 得到增强**

参考资料

- 官方 Changelog
- Kubernetes v1.27 release blog
- KWOK 官网
- v1.27 Release 团队看板

卸载

卸载社区版

依次执行以下命令从集群卸载 DCE 5.0，卸载过程中不会进行任何备份，请谨慎操作。

shell linenums="1" kubectl –n mcamel–system delete mysql mcamel–common–mysql–cluster kubectl –n mcamel–system delete mysql mcamel–common–kpanda–mysql–cluster kubectl –n mcamel–system delete elasticsearches mcamel–common–es–cluster–masters kubectl –n mcamel–system delete redisfailover mcamel–common–redis–cluster kubectl –n mcamel–system delete tenant mcamel–common–minio–cluster kubectl –n ghippo–system delete gateway ghippo–gateway kubectl –n istio–system delete requestauthentications ghippo helm –n mcamel–system uninstall eck–operator mysql–operator redis–operator minio–operator helm –n ghippo–system uninstall ghippo helm –n insight–system uninstall insight–agent insight helm –n ipavo–system uninstall ipavo helm –n kpanda–system uninstall kpanda helm –n istio–system uninstall istio–base istio–ingressgateway istiod kubectl delete namespace mcamel–system ghippo–system insight–system ipavo–system kpanda–system istio–system

安装器 Release Notes

本页列出安装器的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-7-31

v0.10.0

新增

- **新增** 支持 Oracle Linux R8-U7 操作系统
- **新增** 支持灵活暴露 kind 容器映射至宿主机的端口
- **新增** import-artifact 子命令支持根据 clusterConfig.yaml 配置文件中定义的外接服务来导入离线资源

优化

- **优化** 对于使用安装器通过外接 os repo 部署环境后，优化了在容器管理创建集群时可以选择到外接 os repo
- **优化** 重构、抽象 clusterConfig 检测层
- **优化** 优化前置依赖安装脚本的错误提示
- **优化** 在最小化安装过程中 ES 健康状态为 yellow 时允许继续安装
- **优化** 消除 import-artifact 子命令多余的镜像集成步骤
- **优化** 离线资源外接或内建场景下默认展开 clusterConfig 模板中 fullPackagePath 属性

修复

- **修复** 修复外接镜像服务地址检测有误
- **修复** 修复火种 kind 集群输出的错误格式 kubeconfig
- **修复** 修复解压不同版本离线包至统一目录导致 helm 参数出现多个版本 chart 的问题
- **修复** 修复 prerequisite.tgz 中错误的指令集架构信息
- **修复** 修复 import-artifact 不指定-C 导入异常

- **修复** 修复错误的退出指令导致安装器退出提示信息未展示的问题
- **修复** 修复 podman 底座 + kind 重启导致 kube-controller-manager 和 kube-scheduler 证书认证失败问题
- **修复** 修复打印内嵌 manifest 子命令的命令指示符只要指定非 install-app 都会返回全模式 manifest 的问题
- **修复** 修复打印内嵌 manifest 子命令的命令名 typo
- **修复** 修复已存在 amd64 资源，import-artifact 子命令再次导入 arm64 包失败

已知问题

- 升级不支持通过 install-app 子命令，仅支持 create-cluster 子命令
- Redhat 8.6 操作系统火种 kind 重启后 kubelet 服务无法启动，报错：failed to initialize top level QOS containers: root container [kubelet kubepods] doesn't exist，临时解决方案：执行命令 podman restart [containerid] --time
- 安装基于 TencentOS 3.1 的集群时，无法正确识别包管理器，如果需要 TencentOS 3.1 请使用安装器 0.9.0 版本

2023-6-30

v0.9.0

新增

- **新增** istio-ingressgateway 支持了高可用模式，从 v0.8.x 及以前升级到 v0.9.0 时必须执行如下命令：./offline/dce5-installer cluster-create -c clusterConfig.yaml -m manifest.yaml --upgrade infrastructure,gproduct
- **新增** 支持在 clusterConfig.yaml 中配置暴露 火种 kind 地址及端口
- **新增** 安装器在启用画眉存储时，新增前置检查各个节点是否安装 lvm2
- **新增** 安装器内置默认 k8s 版本升级到 v1.26.5
- **新增** 支持在 clusterConfig.yaml 中指定火种 kind 的本地文件挂载路径
- **新增** 整合 ISO 镜像文件导入脚本到安装器二进制中

优化

- **优化** 优化下载脚本
- **优化** 优化 import-artifact 命令逻辑和功能
- **优化** 优化升级过程中 clusterConfig.yaml 中 isoPath 和 osPackagePath 为非必填项
- **优化** 优化安装器临时文件清理机制
- **优化** 优化火种节点复用功能

修复

- **修复** 修复 ES 组件无法在 OCP 启动的问题
- **修复** 修复在 TencentOS 中安装 DCE 后无法访问 UI 界面的问题
- **修复** 修复中间件数据库在 arm64 环境高概率新建数据库失败的问题
- **修复** 修复镜像上传成功检查过程中错误的 shell 扩展

已知问题

- 从 v0.8.x 升级到 v0.9.0 时需要执行如下命令进行检查：

  - 检查 istio-ingressgateway 端口是 80 还是 8080

    ```
    kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].targetPort}'
    ```

  - 检查 istio-ingressgateway 端口是 443 还是 8443

    ```
    kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].targetPort}'
    ```

  输出结果为 80 或 443 时，升级命令需要增加 infrastructure 参数，示例：./offline/dce5-installer cluster-create -c clusterConfig.yaml -m manifest.yaml --upgrade infrastructure,gproduct

  输出结果非上述情况时，升级操作直接参考文档升级 DCE 5.0 产品功能模块

2023-6-15

v0.8.1

优化

- **优化** ipavo 组件升级到 v0.9.3
- **优化** amamba 组件升级到 v0.17.4
- **优化** hwameistor-operator 组件升级到 v0.10.4
- **优化** kangaroo 组件升级到 v0.8.2
- **优化** insight 组件升级到 v0.17.3

修复

- **修复** 修复外接 http 的 Harbor 仓库同步镜像失败的问题
- **修复** 修复 clusterConfig.yaml 配置文件缩进错误的问题
- **修复** 修复基于外接 yum repo 时渲染 localService 配置错误的问题
- **修复** 修复对接外部 jfrog charts 仓库的问题

2023-5-31

v0.8.0

新功能

- **新增** Other Linux 模式支持操作系统 OpenAnolis 8.8 GA
- **新增** 支持操作系统 OracleLinux R9 U1
- **新增** 增加节点状态检测
- **新增** 增加对系统包(OS PKGs)的文件校验
- **新增** 支持节点非 22 端口安装集群
- **新增** 外接文件服务支持 k8s 二进制资源
- **新增** 支持外接 JFrog 镜像及 charts 仓库
- **新增** 支持混合架构部署方案
- **新增** 支持外接 Redis 组件

优化

- **优化** 部署 nacos 实例报镜像缺失

- **优化** 升级集群模块时，重复执行集群安装任务

已知问题

- 离线包解压后的 offline/sample/clusterConfig.yaml 文件存在缩进问题，导致离线部署时会变成在线安装，离线安装前如果要使用 offline/sample/clusterConfig.yaml 文件的话需要手动修改缩进问题，请查看集群配置文件
- Addon 离线包暂不支持上传到 JFrog 外接服务
- 容器管理平台离线模式暂无法支持工作集群添加节点
- 离线场景下使用外置 OS Repo 仓库时，即 clusterConfig.yaml 中定义 osRepos.type=external， 部署 DCE 5.0 成功后无法在容器管理中创建工作集群，临时解决方案如下： global 集群安装完成后立即更新 global 集群 kubean-system 命名空间的 configmap kubean-localservice， 将 yumRepos.external 值中所有双引号改为单引号。如下示例，将文件内的双引号都替换为单引号：

  ```
  yumRepos:
    external: [ "http://10.5.14.100:8081/centos/\$releasever/os/\$basearch","http://10.5.14.100:8081/centos-iso/\$releasever/os/\$basearch" ]
  ```

  替换为：

  ```
  yumRepos:
    external: [ 'http://10.5.14.100:8081/centos/\$releasever/os/\$basearch','http://10.5.14.100:8081/centos-iso/\$releasever/os/\$basearch' ]
  ```

- 版本升级时，insight-agent 存在问题，请参考 insight 升级注意事项

2023-5-30

v0.7.1

优化

- **优化** 升级了监控组件版本

修复

- **修复** 二进制输出社区版 manifest 有误

2023-4-30

v0.7.0

新功能

- **新增** 支持 Other Linux 来部署 DCE 5.0，参考文档
- **新增** 支持了操作系统 OpenEuler 22.03
- **新增** 支持外接 OS Repos，参考集群配置文件说明
- **新增** 支持了内核参数调优，参考集群配置文件说明
- **新增** 支持检测外部 ChartMuseum 和 Minio 服务是否可用

优化

- **优化** 优化了对 tar 等命令的前置校验
- **优化** 优化了升级操作命令行参数
- **优化** 关闭了 Kibana 通过 NodePort 访问，Insight 使用 ES 的 NodePort 或 VIP 访问
- **优化** 优化了并发日志展示，终止任务使用 SIGTERM 信号而不是 SIGKILL

修复

- **修复** 修复在线安装时 Kcoral heml chart 无法查到问题
- **修复** 修复升级时 KubeConfig 无法找到问题

已知问题

- 在线安装 global 集群会失败，需在 clusterConfig.yaml 的 kubeanConfig 块里进行如下配置：

```
kubeanConfig: |-
  calico_crds_download_url: "https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/calico-crds-v3.25.1.tar.gz"
```

同时通过容器管理在线创建工作集群也有相同问题，需在集群创建页面高级配置的自定义参数中添加上述配置，键为 calico_crds_download_url，值为上述 calico_crds_download_url 的值

- Kubean 存在低概率无法创建 spray-job 任务，通过手动删除对应的 clusteroperations CR 资源再重新执行安装命令
- 使用外部 OS Repo 部署 DCE 5.0 后，无法通过容器管理离线创建工作集群，通过手动修改 global 集群 kubean-system 命名空间的 configmap kubean-localservice 来解决。在 yumRepos 下新增如下配置，需要在 external 内填写 clusterConfig.yaml 中配置的外部 OS Repo 地址：

```
yumRepos:
  external: []
```

完成修改后对容器管理创建集群页面的节点配置的 yum 源选择新配置

2023-4-11

v0.6.1

优化

- **优化** 升级了 Kpanda 至 v0.16.1
- **优化** 升级了 Skoala 至 v0.19.4

已知问题

- 采用 7 节点模式安装时，es 专属节点未占成功，预计下个版本修复
- 安装器向火种节点 regsitry 导入镜像时报错 skopeo copy 500 Internal Error -- "NAME_UNKNOWN","message":"repository name not known to registry

2023-4-06

v0.6.0

新功能

- **新增** 支持一键升级 Gproduct 组件

- **新增** 适配了操作系统：UOS V20 1020a / Ubuntu 20.04
- **新增** 支持 OCP (OpenShift Container Platform)安装 DCE 5.0
- **新增** CLI 支持生成 clusterConfig 模板
- **新增** all in one 模式默认启动最小化安装模式
- **新增** Gproduct 组件中新增了 Kcollie 组件
- **新增** 支持社区版将镜像同步到外部仓库

优化

- **优化** 解耦生成离线包的代码和安装流程所需的代码
- **优化** 优化火种节点 inotify 参数
- **优化** 优化全模式在线安装体验
- **优化** 优化 clusterConfig 结构和配置
- **优化** 社区版允许不检查 clusterConfig 格式以及参数
- **优化** 优化安装器执行调度器日志输出

修复

- **修复** 移除了对 wget 的依赖
- **修复** 修复重复解压离线包后安装失败的问题
- **修复** 修复 Minlo 不可重入的问题
- **修复** 修复删除中间件 Redis CR 时继续遗留的 redis pvc
- **修复** 修复 Amamba 和 Amamba-jenkins 并发安装时先后顺序依赖的问题
- **修复** 修复安装器命令行-j 参数解析失败问题

2023-2-28

v0.5.0

新功能

- **新增** 离线包分离 osPackage，需要在集群配置文件中定义 osPackagePath

- **新增** 支持 addon 离线化，需要在集群配置文件中定义 addonOfflinePackagePath
- **新增** 离线安装支持操作系统 REHL 8.4、REHL 7.9

优化

- **优化** 升级了前置依赖工具的版本

修复

- **修复** 安装器命令行 -j 参数效验检测问题
- **修复** 前置依赖工具安装路径问题
- **修复** 主机清单密码纯数字无效问题
- **修复** 运行时为 Docker 时，无法拉取内建仓库镜像问题

已知问题

- 非最小化安装的 REHL8 上由于预装的 runc 导致安装器安装失败，临时解决方案：安装前在上述每台节点上执行 rpm -qa | grep runc && yum remove -y runc
- 非最小化安装的 REHL8 上有非法的内核参数设置，临时解决方案；安装前在上述每台节点上执行 eval $(grep -i 'vm.maxmapcount' /etc/sysctl.conf -r /etc/sysctl.d | xargs -L1 | awk -F ':' '{printf("sed -i -r \"s/(%s)/#\\1/\" %s; ", $2, $1)}') && sysctl --system
- helm 并发安装存在潜在风险，失败后无法继续执行安装

2022-12-30

v0.4.0

新功能

- **新增** clusterConfig 的语法从 v1alpha1 升级为 v1alpha2，语法有不兼容变更，可以查看文档
- **新增** 不再在全局服务集群上安装永久 Harbor 和永久 MinIO
- **新增** 火种节点需要永久存在，用户安装 minio、chart museum、registry
- **新增** 商业版新增安装 contour 作为 默认的 ingress-controller

- **新增** 商业版新增安装 cert-manager
- **新增** 支持私钥模式的集群部署
- **新增** 支持外置镜像仓库进行部署

优化

- **优化** 离线包不再包括操作系统的 ISO，需要单独下载，在纯离线的情况下，需要在 clusterConfig 文件中定义 ISO 的绝对路径
- **优化** 商业版使用 Contour 作为默认的 ingress-controller
- **优化** MinIO 支持使用 VIP
- **优化** coredns 自动注入仓库 VIP 解析
- **优化** 优化离线包制作流程，加速打包 Docker 镜像
- **优化** 优化了离线包的大小
- **优化** 基础设施支持 1.25：升级 redis-operator、eck-operator、hwameiStor
- **优化** 升级到 keycloakx
- **优化** Istio 版本升级至 v1.16.1

修复

- **修复** HwameiStor 的 operator 会随机错误清理集群中的 Job
- **优化** mysql 增强半同步，解决频繁主从切换的数据不一致
- **优化** 解决 mysql 磁盘被 clusterPedia 写爆
- **优化** coredns 自动注入仓库 VIP 解析
- **优化** 公有云上 svc 的修复
- **优化** 修复各种子模块的镜像和 helm 问题

已知问题

- 默认安装模式下暂不支持未分区的 SSD 盘，如果要支持，需要手工干涉。
- 纯离线环境，默认没有应用商店。请手动将火种节点的 chart-museum 接入到 global 集群，仓库地址：http://{火种 IP}:8081，用户名 rootuser，密码 rootpass123

- metallb 社区有已知问题，在主网卡有 dadfailed 的 IPV6 回环地址，metallb 无法工作，安装之前需要确保主网卡没有 dadfailed
- insight-api-server 启动中如果机器太卡，在 Liveness 健康检查周期内，无法完成数据库的初始化（migrate）动作，导致需要手动介入
- clusterConfig 配置文件中里的 iso 路径必须是绝对路径，不支持相对路径
- kubean 默认 k8s 版本和离线包仍然限制在 k8s 1.24 版本，还未能更新到 1.25（PG 暂不支持）
- external-Registry 如果是 Harbor，暂时不会自动创建 Project，需要提前手动创建
- Docker 运行时，无法拉取 built-in 仓库，将在下个版本修复
- 禁用 IPV6 之后，podman 无法启动火种节点的 kind 集群

2022-11-30

v0.3.29

新功能

- **新增** ARM64 支持：构建 ARM64 离线包
- **新增** 支持麒麟 kylin v10 sp2 离线包
- **新增** 基础设施支持 1.25：升级 redis-operator、eck-operator、hwameiStor 等组件
- **新增** 支持私钥模式的集群部署
- **新增** 工作负载基于自定义指标弹性伸缩，更加贴近用户实际业务弹性扩缩容需求

优化

- **优化** 永久 harbor 使用 operator 创建，开启 HTTPS，并使用 Postgressql operator
- **优化** 商业版使用 Contour 作为默认的 ingress-controller
- **优化** MinIO 支持使用 VIP

- **优化** coredns 自动注入仓库 VIP 解析
- **优化** 优化离线包制作流程，加速打包 Docker 镜像

修复

- **修复** 修复了公有云 Service 的问题
- **修复** 修复了各个子模块的镜像和 Helm 的问题
- **修复** 修复了离线包加载的缺陷修复

已知问题

- 因为部分 Operator 需升级到支持 K8s 1.25，导致 DCE 5.0 向下不支持 K8s 1.20
- Kubean 默认 K8s 版本和离线包仍然限制在 1.24 版本，还未能更新到 1.25（由于 postgres-operator 暂不支持）
- Image Load 情况下，istio-ingressgateway imagePullPolicy 为 always
- ARM 版本，不能执行安装脚本的第 16 步（harbor），因为 harbor 暂时不支持 ARM。 需要修改 mainfest.yaml 文件，postgressql operator 为 fasle，执行安装命令时要添加 -j 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
- 在容器管理界面上创建新集群，无法对接 HTTPS 的仓库，需要手动修改 kubean job 的 ConfigMap 和 CR
- 永久 MinIO 的 PVC 的大小默认是 30G，会不够用（承载 kubean 二进制、ISO 以及 Harbor 镜像仓库），需要进行扩容操作

一键下载所有离线包

本页提供了脚本来一键下载安装 DCE 5.0 所需的离线包。

下载脚本

```bash
export VERSION= v0.8.0
curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/download_packages_${VERSION}.sh
chmod +x download_packages_${VERSION}.sh
```

执行脚本

```bash
./download_packages_${VERSION}.sh ${DISTRO} ${INSTALLER_VERSION} ${ARCH}
```

参数说明：

参数	默认值	有效值范围
VERSION	当前下载脚本版本	目前仅 v0.8.0 版本支持
INSTALLER_VERSION	指定下载 DCE 5.0 的版本	有效 DCE 5.0 的发行版本即可
DISTRO	centos7	centos7 kylinv10 redhat8 ubuntu2004
ARCH	amd64	amd64 arm64

!!! note

- amd64 支持的操作系统：centos7、redhat8、ubuntu2004
- arm64 支持的操作系统：kylinv10

安装依赖项

安装 DCE 5.0 之前，需要先安装一些依赖项。

- 对于社区版，请在 K8s Master 节点上安装依赖项。
- 对于商业版，请在火种节点上安装依赖项。

!!! note

安装的依赖项大致包括：

- podman
- helm
- skopeo
- kind
- kubectl
- yq
- minio client

安装过程中如果您的环境存在一些工具且版本低于我们定义的版本，会将对应工具进行强制更新替换。

在线安装依赖项

1. 下载脚本。

   ```
   export VERSION=v0.10.0    # 安装器版本
   curl -LO https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite_${VERSION}.sh
   ```

   为 install_prerequisite_${VERSION}.sh 添加可执行权限：

   ```
   chmod +x install_prerequisite_${VERSION}.sh
   ```

2. 开始在线安装前置依赖。

    - 对于社区版

      ```
      bash install_prerequisite_${VERSION}.sh online community
      ```

    - 对于商业版

      ```
      bash install_prerequisite_${VERSION}.sh online full
      ```

离线安装依赖项

离线安装意味着目标主机的网络处于离线状态，无法下载所需依赖项，所以需先在一个在线环境中制作好离线包。

1. 下载脚本。

   ```
   export VERSION=v0.10.0
   curl -LO https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite_${VERSION}.sh
   ```

2. 下载前置依赖组件离线包。

   ```
   export VERSION=v0.10.0
   curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/prerequisite_${VERSION}_amd64.tar.gz
   ```

   !!! note

   - 如果是 arm 架构，下载地址为：https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/prerequisite_${VERSION}_arm64.tar.gz
   - 确保离线包与脚本在同一个目录层级

3. 执行离线安装。

    - 对于社区版

```
export BINARY_TAR=prerequisite_${VERSION}_amd64.tar.gz
chmod +x install_prerequisite_${VERSION}.sh
./install_prerequisite_${VERSION}.sh offline community
```

- 对于商业版

```
export BINARY_TAR=prerequisite_${VERSION}_amd64.tar.gz
chmod +x install_prerequisite_${VERSION}.sh
./install_prerequisite_${VERSION}.sh offline full
```

接下来就可以安装 DCE 5.0 社区版或商业版了。

安装排障

本页汇总了常见的安装器问题及其排障方案，便于用户快速解决安装过程中遇到的问题。

火种节点重启后 Podman 无法自动恢复

火种节点重启后，等待 kind 启动成功：

1. 执行 podman exec 进入容器。
2. sed –i 's/server: .*:6443/server: https://127.0.0.1:6443/g' /etc/kubernetes/controller–manager.conf
3. sed –i 's/server: .*:6443/server: https://127.0.0.1:6443/g ' /etc/kubernetes/scheduler.conf

禁用 IPv6 后安装时 Podman 无法创建容器

报错信息如下：

ERROR: failed to create cluster: command "podman run ––name kind–control–plane...

解决方案：重新启用 IPv6 或者更新火种节点底座为 Docker。

Podman 相关 issue 地址：
https://github.com/containers/podman/issues/13388

Kind 集群重装 DCE 5.0 时 Redis 卡住重装

问题：Redis Pod 出现了 0/4 running 很久的情况，提示：primary ClusterIP can not unset

1. 在 mcamel–system 命名空间下删除 rfs–mcamel–common–redis

   kubectl delete svc rfs–mcamel–common–redis –n mcamel–system

2. 然后重新执行安装命令

使用 Metallb 时 VIP 访问不通导致 DCE 登录界面无法打开

1. 排查 VIP 的地址是否和主机在同一个网段，Metallb L2 模式下需要确保在同一个网段
2. 如果是在 Global 集群中的控制节点新增了网卡导致访问不通，需要手动宣告配置 L2Advertisement。 请参考 Metallb 这个问题的文档

社区版 fluent-bit 安装失败

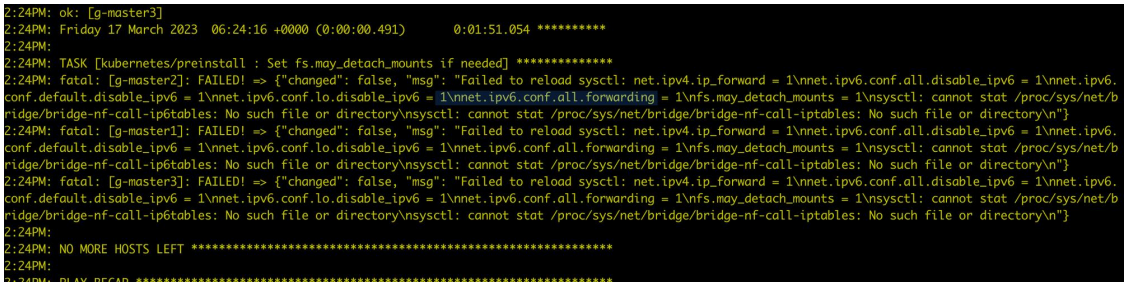
报错：DaemonSet is not ready: insight-system/insight-agent-fluent-bit. 0 out of 2 expected pods are ready

排查查看 Pod 日志是否出现下述关键信息：

[warn] [net] getaddrinfo(host='mcamel-common-es-cluster-masters-es-http.mcamel-system.svc.cluster.local',errrt11):Could not contact DNS servers

出现上述问题是一个 fluent-bit 的 bug，可以参考：https://github.com/aws/aws-for-fluent-bit/issues/233

在 CentOS 7.6 安装时报错



FAQ1

在安装全局服务集群的每个节点上执行 modprobe br_netfilter，将 br_netfilter 加载之后就好了。

CentOS 环境准备问题

运行 yum install docker 时报错：

```
Failed to set locale, defaulting to C.UTF-8
CentOS Linux 8 - AppStream
                                93  B/s |  38  B     00:00
Error: Failed to download metadata for repo 'appstream': Cannot prepare intern
al mirrorlist: No URLs in mirrorlist
```

可以尝试以下述方法来解决：

- 安装 glibc-langpack-en

  ```
  sudo yum install -y glibc-langpack-en
  ```

- 如果问题依然存在，尝试：

  ```
  sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
  sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g'
   /etc/yum.repos.d/CentOS-*
  sudo yum update -y
  ```

安装简介

DCE 5.0 大体分为两个版本：社区版和商业版。

社区版可包括容器管理、全局管理、可观测性三大模块，可永久免费使用。

商业版在社区版的基础上可按需购买服务网格、微服务引擎、多云编排、数据服务中间件、镜像仓库等高级模块，功能更全面，更能适应生产环境需求。

版本

包含的模块

描述

社区版

全局管理

容器管理

可观测性

永久免费授权，3 个模块会保持持续更新，可随时 下载子模块的离线包

商业版

全局管理

容器管理

可观测性

应用工作台

多云编排

微服务引擎

服务网格

精选中间件

镜像仓库

联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898，各个模块可按需自由组合，可随时 下载子模块的离线包

社区版安装流程

DCE 5.0 社区版的安装流程如下图：

```
flowchart TB

subgraph second[使用说明]
    direction TB
    U[ ] -.-
    free[申请免费体验] --- kpanda[容器管理]
    kpanda --- ghippo[全局管理]
    ghippo --- insight[可观测性]
    insight -.- ask[提问!!!]
end

subgraph first[安装社区版]
    direction TB
    S[ ] -.-
    plan[资源规划] --> k8s[准备 K8s 集群]
    k8s --> tools[安装依赖项]
    tools -.-> kind[通过 kind 集群<br>在线安装]
    tools -.-> s1[通过标准 k8s 集群<br>在线安装]
    tools -.-> s2[通过标准 k8s 集群<br>离线安装]
end
```

```
start([fa:fa-user DCE 5.0 社区版<br>安装和使用流程]) --> first
start --> second

classDef grey fill:#dddddd,stroke:#ffffff,stroke-width:px,color:#000000, font-size:1
5px;
classDef white fill:#ffffff,stroke:#000,stroke-width:px,color:#000,font-weight:bold
classDef spacewhite fill:#ffffff,stroke:#fff,stroke-width:0px,color:#000
classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class plan,k8s,tools,kind,s1,s2,kpanda,ghippo,insight,free,ask cluster;
class start plain
class S,U spacewhite

click plan "https://docs.daocloud.io/install/community/resources/"
click k8s "https://docs.daocloud.io/install/community/kind/online/#kind"
click tools "https://docs.daocloud.io/install/install-tools/"
click kind "https://docs.daocloud.io/install/community/kind/online/"
click s1 "https://docs.daocloud.io/install/community/k8s/online/"
click s2 "https://docs.daocloud.io/install/community/k8s/offline/"

click kpanda "https://docs.daocloud.io/kpanda/intro/"
click ghippo "https://docs.daocloud.io/ghippo/intro/"
click insight "https://docs.daocloud.io/insight/intro/"
click free "https://docs.daocloud.io/dce/license0/"
click ask "https://docs.daocloud.io/install/intro/#_4"
```

!!! tip

上图中的蓝色文字可点击跳转

商业版安装流程

DCE 5.0 商业版的安装流程如下图：

```
flowchart TB

    start([fa:fa-user DCE 5.0 商业版<br>安装流程]) -.- arch[部署架构]
    arch --> deploy[部署要求]
    deploy --> prepare[准备工作]
    prepare --> download[下载离线包]
    download --> config[编辑并配置<br>clusterConfig.yaml]
    config --> install[开始安装]

classDef grey fill:#dddddd,stroke:#ffffff,stroke-width:1px,color:#000000, font-size:
15px;
classDef white fill:#ffffff,stroke:#000,stroke-width:1px,color:#000,font-weight:bold
classDef spacewhite fill:#ffffff,stroke:#fff,stroke-width:0px,color:#000
classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class arch,deploy,prepare,download,config,install cluster
class start plain

click arch "https://docs.daocloud.io/install/commercial/deploy-arch/"
click deploy "https://docs.daocloud.io/install/commercial/deploy-requirements/"
click prepare "https://docs.daocloud.io/install/commercial/prepare/"
click download "https://docs.daocloud.io/install/commercial/start-install/#1"
click config "https://docs.daocloud.io/install/commercial/start-install/#2"
click install "https://docs.daocloud.io/install/commercial/start-install/#3"
```

联系我们

DaoCloud Enterprise 5.0 还处于发布初期，安装流程可能会有变更。请收藏此页，关注更新动态，更多操作文档也在制作之中。

- 若有任何安装或使用问题，请提出反馈。
- 欢迎扫描二维码，与开发者畅快交流：



社区版交流群

下载 DCE 5.0 申请社区免费体验

离线资源导入

概述

安装器 dce5-installer (仅支持 v0.9.0 版本及以上) 支持了 import-artifact 命令来导入离线资源。目前可被导入的离线资源包括:

- *.iso 操作系统 ISO 镜像文件
- os-pkgs-${disto}-${kubean_version}.tar.gz Kubean 提供的 osPackage 离线包
- offline-${install_version}-${arch}.tar 安装器全模式离线镜像包:
  - K8S binaries & images
  - DCE5 各个模块的 images 文件和 charts 包

使用场景

场景一

当全局服务集群的操作系统与创建的工作集群操作系统不一致时，需要导入对应工作集群的操作系统的 ISO 镜像文件 和 osPackage 离线包。

如：在 CentOS 的全局服务集群上创建 Ubuntu 操作系统的工作集群

场景二

在混合架构的离线部署场景中，需要在原有的 amd64 离线资源（K8S binaries) && (K8S images + 各个模块的 images 文件和 charts 包）基础上，导入并整合 arm64 的对应资源。

如：如何为工作集群添加异构节点

导入命令介绍

需要提前下载好 dce5-installer 二进制文件。

导入操作系统 ISO 镜像文件

以导入 TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso 为例：

```
# 采用 build-in 内建模式部署火种集群时，我们可以不用指定 clusterConfig.yml 配置文件
dce5-installer import-artifact --iso-path=/home/iso/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso

# 采用 external 外接模式部署火种集群时，我们需要指定 clusterConfig.yml 配置文件
dce5-installer import-artifact -c clusterConfig.yml --iso-path=/home/iso/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso
```

导入 Kubean 提供的 osPackage 离线包

以导入 os-pkgs-tencent31-v0.6.2.tar.gz 为例：

```
# 采用 build-in 内建模式部署火种集群时，我们可以不用指定 clusterConfig.yml 配置文件
dce5-installer import-artifact --os-pkgs-path=/home/os-pkgs/os-pkgs-tencent31-v0.6.2.tar.gz

# 采用 external 外接模式部署火种集群时，我们需要指定 clusterConfig.yml 配置文件
dce5-installer import-artifact -c clusterConfig.yml --os-pkgs-path=/home/os-pkgs/os-pkgs-tencent31-v0.6.2.tar.gz
```

导入安装器离线镜像包 Offline 目录内容

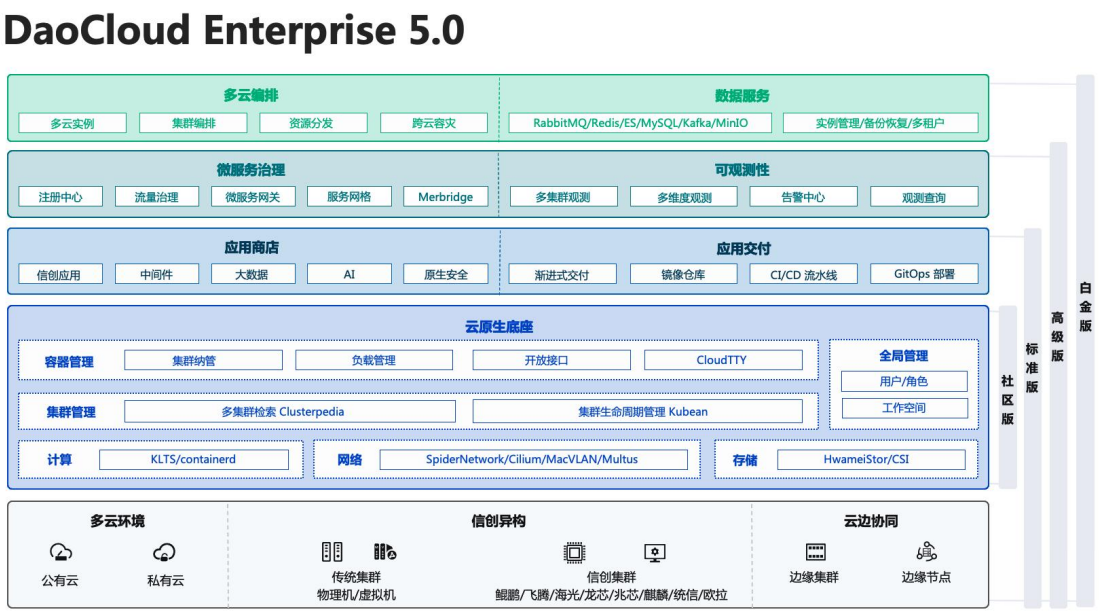
```
# 采用 build-in 内建模式部署火种集群时，我们可以不用指定 clusterConfig.yml 配置文件
dce5-installer import-artifact --offline-path=/home/offline/

# 采用 external 外接模式部署火种集群时，我们需要指定 clusterConfig.yml 配置文件
dce5-installer import-artifact -c clusterConfig.yml --offline-path=/home/offline/
```

同时指定导入多中离线资源

```
# 采用 build-in 内建模式部署火种集群时，我们可以不用指定 clusterConfig.yml 配置文件
dce5-installer import-artifact \
    --offline-path=/home/offline/ \
    --os-pkgs-path=/home/os-pkgs/os-pkgs-tencent31-v0.6.2.tar.gz \
    --iso-path=/home/iso/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso

# 采用 external 外接模式部署火种集群时，我们需要指定 clusterConfig.yml 配置文件
dce5-installer import-artifact -c clusterConfig.yml \
    --offline-path=/home/offline/ \
    --os-pkgs-path=/home/os-pkgs/os-pkgs-tencent31-v0.6.2.tar.gz \
    --iso-path=/home/iso/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso
```

标准版升级到白金版

DCE 5.0 支持多种版本，除社区版以外的其他版本均为商业版，本文演示如何从标准版升级到白金版。



模块图

前提条件

- 您需要有一个 DCE 5.0 的集群环境，参阅离线化部署商业版
- 请确保您的火种机器还存活，并且当前环境对应的离线包还在，否则需要重新下载。

操作步骤

第 1 步：配置 manifest.yaml 文件

将 manifest.yaml 文件里的被 disable 的 components 放开， 开启 enable。

```
skoala:
  enable: false
  helmVersion: 0.26.1
```

```
  variables:
mspider:
  enable: false
  helmVersion: v0.18.0-rc2
  variables:
mcamel-rabbitmq:
  enable: false
  helmVersion: 0.12.2
  variables:
mcamel-elasticsearch:
  enable: false
  helmVersion: 0.9.2
  variables:
...
mcamel-mysql:
  enable: false
  helmVersion: 0.10.2
  variables:
mcamel-redis:
  enable: false
  helmVersion: 0.10.0-rc2
  variables:
mcamel-kafka:
  enable: false
  helmVersion: 0.7.2
  variables:
mcamel-minio:
  enable: false
  helmVersion: 0.7.2
  variables:
mcamel-postgresql:
  enable: false
```

```
    helmVersion: 0.4.0-rc2
    variables:
  mcamel-mongodb:
    enable: false
    helmVersion: 0.2.0-rc2
    variables:
```

第 2 步：执行命令

执行升级命令

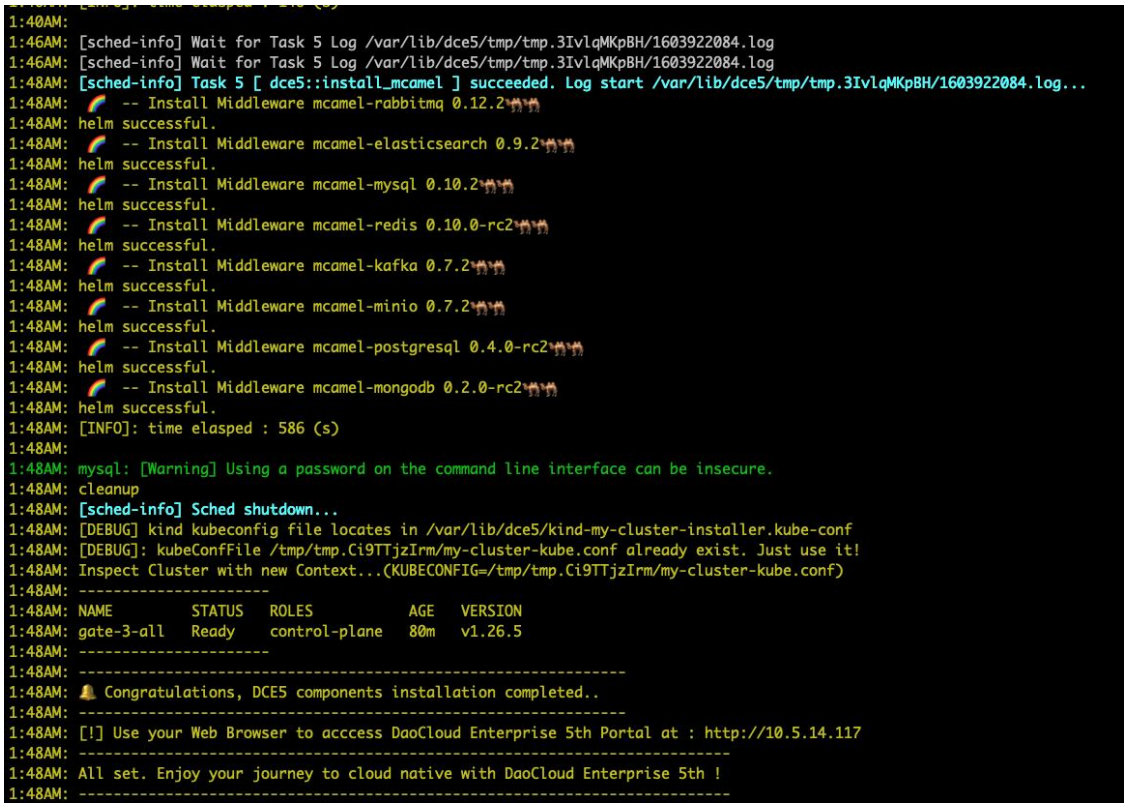
```
./offline/dce5-installer cluster-create -c ./sample/clusterConfig.yaml -m ./sample/manifest.yaml -j 11,12
```

!!! note

clusterConfig.yaml 文件需要与安装时使用的参数保持一致。

-j 11,12 是目前执行标准版升级到白金版的必须参数。

第 3 步：安装成功提示



upgrade

升级 DCE 5.0 组件

DCE 5.0 组件的升级包含升级 DCE 5.0 产品功能模块、升级 DCE 5.0 基础设施模块

其中 DCE 5.0 由容器管理、全局管理、可观测性等十几个子模块构成，主要指 mainfest.yaml 文件中的 components 部分。

DCE 5.0 基础设施模块的组件特指 mainfest.yaml 文件中的 infrastructures 部分。

前提条件

- 您需要有一个 DCE 5.0 的集群环境，参阅离线化部署商业版
- 请确保您的火种机器还存活
- 请确认您想要升级的版本，参阅版本发布说明

离线升级操作步骤

本次操作步骤演示如何从 v0.8.0 升级到 v0.9.0。目前升级到从低版本升级到 v0.9.0 时，需要同时升级 DCE 5.0 产品功能模块和 DCE 5.0 基础设施模块，从而使用到 istio-gateway 组件的高可用功能。

第 1 步：下载 v0.9.0 离线包

可以在下载中心下载最新版本。

CPU 架构	版本	下载地址
AMD64	v0.9.0	https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.9.0-amd64.tar
ARM64	v0.9	https://proxy-qiniu-download-

CPU 架构	版本	下载地址
4	.0	public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.9.0-arm64.tar

下载完毕后解压离线包，以 AMD64 架构离线包为例：

```
tar -xvf offline-v0.9.0-amd64.tar
```

第 2 步：配置集群配置文件 clusterConfig.yaml

!!! note

- 需要确保[集群配置文件](commercial/cluster-config.md) 与安装时使用的参数一致

- 目前仅对 imagesAndCharts 的 builtin 方式进行了测试

文件在解压后的离线包 offline/sample 目录下，参考配置文件如下：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  clusterName: my-cluster
  loadBalancer:
    type: metallb
    istioGatewayVip: 172.30.**.**/32
    insightVip: 172.30.**.**/32
  masterNodes:
    - nodeName: "g-master1"
      ip: 172.30.**.**
      ansibleUser: "root"
      ansiblePass: "*****"
  workerNodes:
    - nodeName: "g-worker1"
```

```
    ip: 172.30.**.**
    ansibleUser: "root"
    ansiblePass: "*****"
  - nodeName: "g-worker2"
    ip: 172.30.**.**
    ansibleUser: "root"
    ansiblePass: "*****"

fullPackagePath: "/home/installer/offline"
osRepos:
  type: builtin
  isoPath: "/home/installer/CentOS-7-x86_64-DVD-2207-02.iso"
  osPackagePath: "/home/installer/os-pkgs-centos7-v0.4.4.tar.gz"
imagesAndCharts:
  type: builtin

addonPackage:
binaries:
  type: builtin   # (1)
```

1. official-service(if omit or empty), builtin or external

第 3 步：配置 mainfest.yaml（可选）

文件在解压后的离线包 offline/sample 目录下。

DCE 5.0 产品功能模块配置

DCE 5.0 产品功能模块的组件特指 mainfest.yaml 文件中的 components 部分。 如果有些产品组件不需要升级，可以在对应组件下选择关闭。如果采用以下配置，更新时将不会对 Kpanda（容器管理）进行升级：

```
components:
  kpanda:
    enable: false
```

```
    helmVersion: 0.16.0
    variables:
```

DCE 5.0 基础设施模块配置

DCE 5.0 基础设施模块的组件特指 mainfest.yaml 文件中的 infrastructures 部分，如下配置就是基础设施中的 hwameiStor 组件：

```
  infrastructures:
    hwameiStor:
      enable: true
      version: v0.10.4
      policy: drbd-disabled
```

!!! note

目前仅支持对当前环境中已经安装的产品组件进行升级，不存在的组件将会跳过升级步骤

第 4 步：开始升级

DCE 5.0 产品功能模块升级

执行升级命令

```
./offline/dce5-installer cluster-create -c ./offline/sample/clusterConfig.yaml -m ./offline/sample/manifest.yaml --upgrade gproduct
```

DCE 5.0 基础设施模块升级

执行升级命令

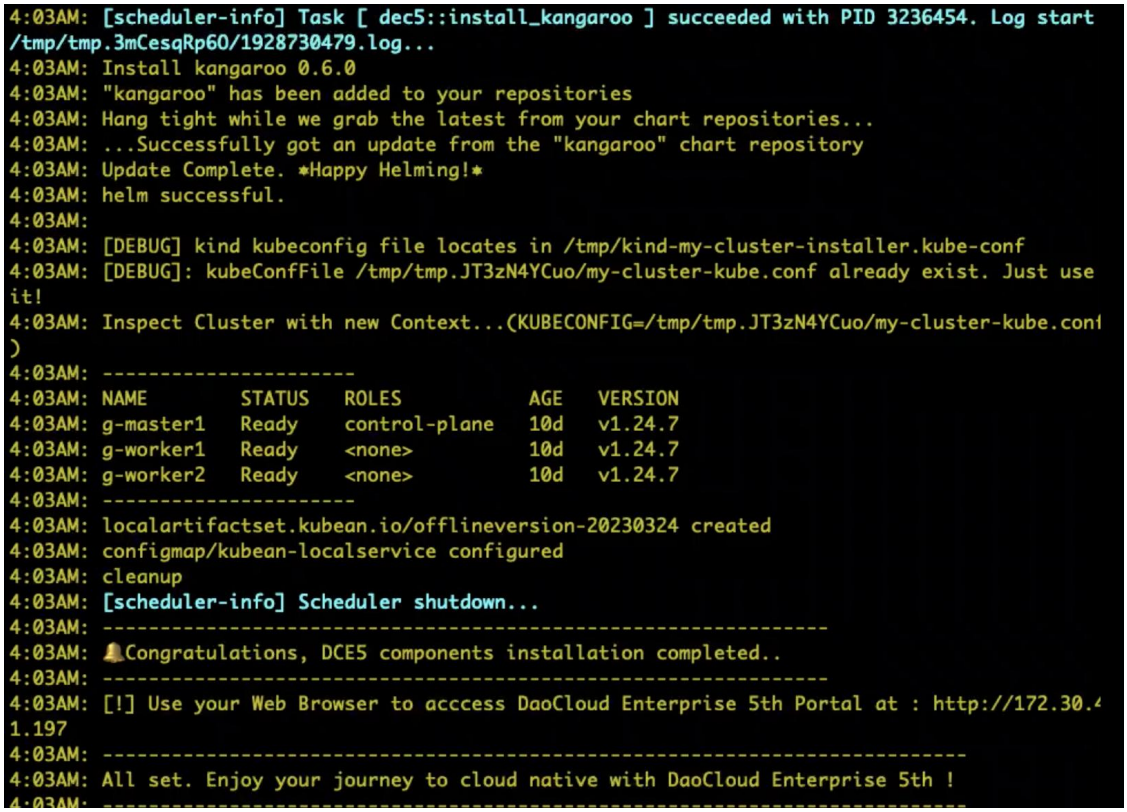
```
./offline/dce5-installer cluster-create -c ./offline/sample/clusterConfig.yaml -m ./offline/sample/manifest.yaml --upgrade infrastructures
```

升级参数说明：

- install-app 或 cluster-create，代表安装 DCE 5.0 的安装模式类型。如果最初的环境是通过 cluster-create 来安装的，则升级时也采用这个命令

- --upgrade 可以简写为 -u，目前支持升级 DCE 5.0 产品功能模块（gproduct）与基础设施模块（infrastructures）
- 如果需要一起升级产品功能模块和基础设施模块，则可以指定参数 --upgrade infrastructures,gproduct

第 5 步：安装成功提示



upgrade

文件下载

本页列出了安装器和镜像包的下载地址。

下载离线镜像包

AMD 架构离线镜像包

1. 在火种节点上下载离线镜像包，可从下载中心下载其他版本的离线包。

   ```
   # 假定 VERSION 为 v0.4.0
   export VERSION=v0.4.0
   wget https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterpris
   e/dce5/offline-centos7-$VERSION-amd64.tar
   ```

2. 解压离线包。

   ```
   tar -xvf offline-centos7-$VERSION-amd64.tar
   ```

ARM 架构离线镜像包

1. 在火种节点上下载离线镜像包，可从下载中心下载其他版本的离线包。

   ```
   # 假定 VERSION 为 v0.4.0
   export VERSION=v0.4.0
   wget https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterpris
   e/dce5/offline-kylin-v10sp2-$VERSION-arm64.tar
   ```

2. 解压离线包。

   ```
   tar -xvf offline-kylin-v10sp2-$VERSION-arm64.tar
   ```

下载 ISO 操作系统镜像文件

CPU架构	操作系统版本	下载地址
AMD64	Centos 7	https://mirrors.tuna.tsinghua.edu.cn/centos/7.9.2009/isos/x86_64/CentOS-7-x86_64-DVD-2009.iso
ARM64	Kylin Linux Advanced Server release V10 (Sword) SP2	申请地址：https://www.kylinos.cn/scheme/server/1.html 注意：麒麟操作系统需要提供个人信息才能下载使用，下载时请选择 V10 (Sword) SP2

部署架构

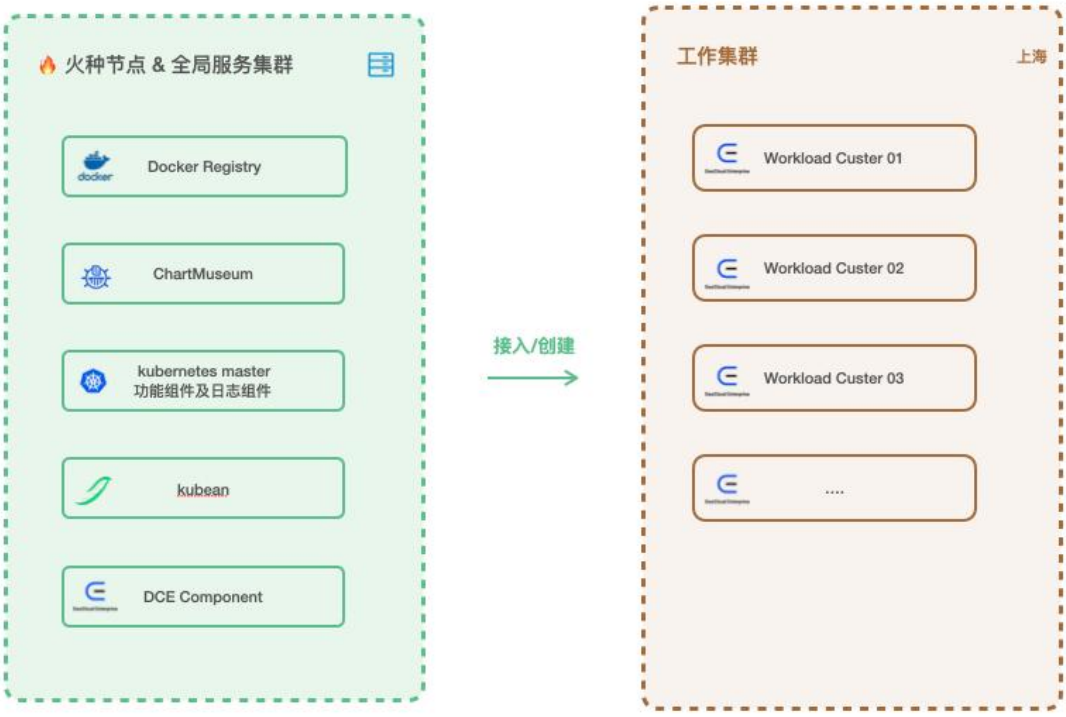
DCE 5.0 提供了三种部署架构：新手尝鲜模式、4 节点、7 节点。

下表介绍了架构中涉及到的术语：

名词	介绍
火种节点	火种节点也称为 bootstrap 节点，安装部署程序的执行，并运行平台所需的镜像仓库和 chart museum。
全局服务集群	用来部署 DCE 5.0 的所有组件，以及 Kubean，其中 Kubean 用来管理集群的生命周期。
工作集群	支撑业务应用和服务（成功安装 DCE 之后再部署工作集群）
DCE Component	DCE 5.0 的所有组件，包含全局管理、容器管理、可观测性、应用工作台、多云编排、镜像仓库、微服务引擎、服务网格、中间件等产品模块。

新手尝鲜模式

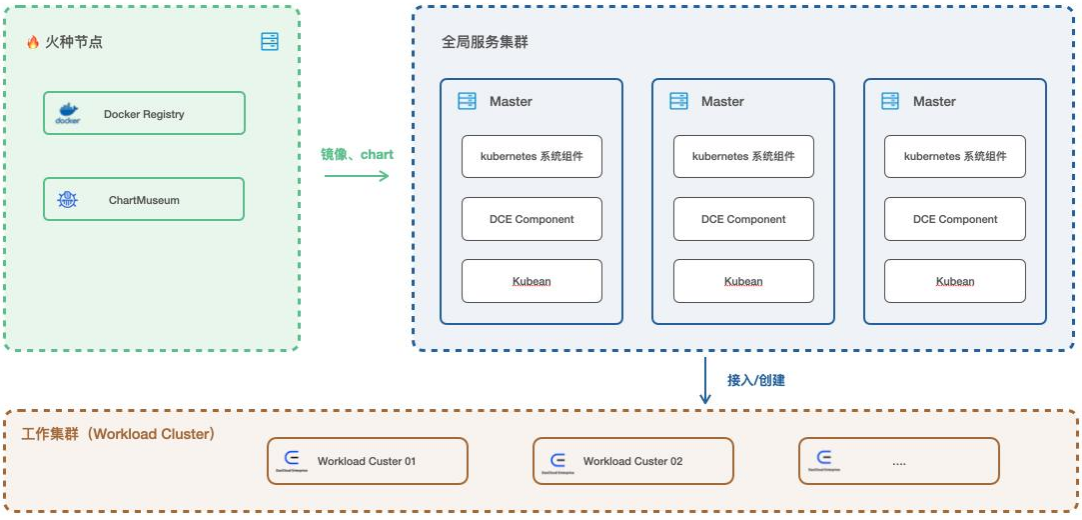
新手尝鲜模式仅需要一台主机，仅建议个人客户体验时采用此模式进行安装 DCE 5.0。 使用新手尝鲜模式时，建议使用最小化安装 DCE 5.0，即安装命令后添加 –z 参数。



allinone

4 节点模式

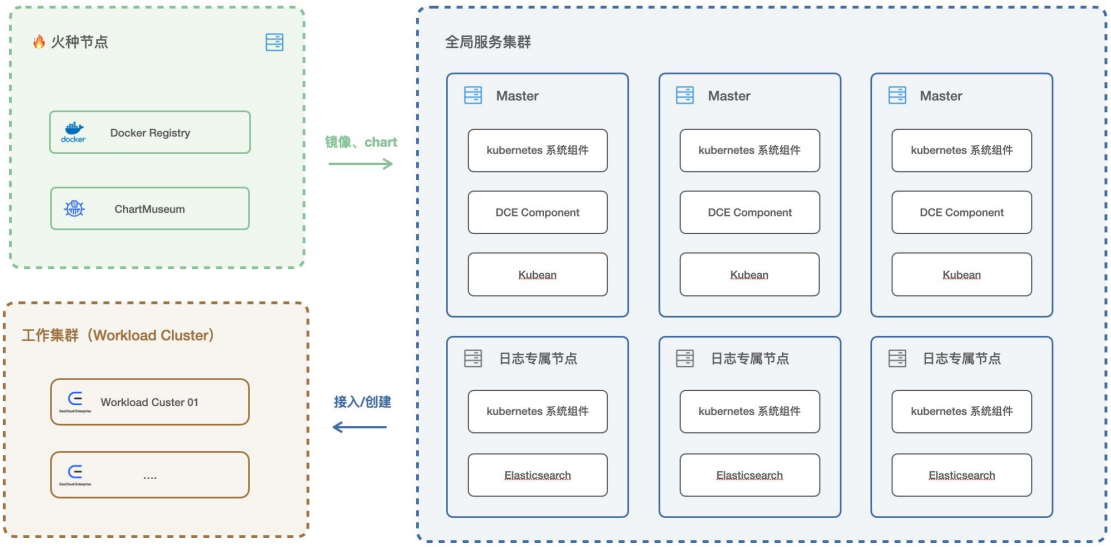
4 节点模式由 1 个火种节点、3 个集群的主节点组成，仅建议 PoC 或测试环境时采用此模式进行安装 DCE 5.0。



four

7 节点模式（1 + 6）

7 节点模式由 1 个火种节点、3 个集群的主节点、3 个集群的工作节点组成，其中工作节点作为 Elasticsearch 组件的专属节点。 建议客户生产环境中采用此模式进行安装 DCE 5.0。



seven

准备工作

本页说明了部署 DCE 5.0 需要进行的准备工作。

机器准备

新手尝鲜模式

参阅新手尝鲜模式说明。

数量	服务器角色	服务器用途	cpu 数量	内存容量	系统硬盘	未分区的硬盘
1	all in one	镜像仓库、chart museum 、global 集群本身	16 core	32G	200G	400G

4 节点模式

参阅 4 节点模式说明。

数量	服务器角色	服务器用途	cpu 数量	内存容量	系统硬盘	未分区的硬盘
1	火种节点	1. 执行安装部署程序 2. 运行平台所需的镜像仓库和 chart museum	2	4G	200G	–
3	控制面	1. 运行 DCE 5.0 组件 2. 运行 kubernetes 系统组件	8	16G	100G	200G

7 节点模式(生产环境推荐)

参阅 7 节点模式说明。

数量	服务器角色	服务器用途	cpu数量	内存容量	系统硬盘	未分区的硬盘
1	火种节点	1. 执行安装部署程序 2. 运行平台所需的镜像仓库和 chart museum	2	4G	200G	–
3	master	1. 运行 DCE 5.0 组件 2. 运行 kubernetes 系统组件	8	16G	100G	200G
3	worker	单独运行日志相关组件	8	16G	100G	200G

前置检查

机器检查

检查项	具体要求	说明
用户权限	root	必须使用 root 用户部署，各个服务器也必须允许 root 用户 ssh 登录
swap	关闭	如果不满足，系统会有一定几率出现 io 飙升，造成 容器运行时 卡死
防火墙	关闭（不强制）	–
selinux	关闭（不强制）	–
时间同步	所有集群节点要求时间必须同步	这是 Docker 和 Kubernetes 官方要求。否则 kube.conf 会报错 Unable to connect to the server: x509: certificate has expired or is not yet
时区	所有服务器时区必须统一	建议设置为 Asia/Shanghai。 参考命令：timedatectl set-timezone Asia/Shanghai

检查项	具体要求	说明
Nameserver	/etc/resolv.conf 至少有一个 Nameserver	CoreDNS 要求，否则会有报错。该 nameserver 在纯离线环境下可以是一个不存在的 IP 地址。Centos8minial 默认没有 /etc/resolv 文件，需要手动创建

火种机器依赖组件检查

检查项	版本要求	说明
podman	v4.4.4	–
helm	⩾ 3.11.1	–
skopeo	⩾ 1.11.1	–
kind	v0.19.0	–
kubectl	⩾ 1.25.6	–
yq	⩾ 4.31.1	–
minio client	mc.RELEASE.2023-02-16T19-20-11Z	

如果不存在依赖组件，通过脚本进行安装依赖组件，安装前置依赖。

```
export VERSION=v0.9.0
# 下载脚本
curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/install_prerequisite_${VERSION}.sh

# 添加可执行权限
chmod +x install_prerequisite_${VERSION}.sh
```

```
# 开始安装
bash install_prerequisite_${VERSION}.sh online full
```

外接组件准备

如果需要使用客户已有的一些组件，可以参考以下文档进行准备：

- 使用外接服务存储 Binaries 资源
- 使用外接镜像仓库与 Chart 仓库存储镜像与 Chart 包
- 使用外接中间件服务
- 使用外接服务存储 OS Repo 资源

离线安装 DCE 5.0 商业版

请在安装之前确保您以阅读并理解部署要求、部署架构、准备工作

请查看产品发布记录，避免所安装版本的已知问题，以及查看新增功能

离线安装步骤

第 1 步：下载离线包

请根据业务环境下载对应的离线包。

离线镜像包 （必需）

离线镜像包包含安装 DCE 5.0 各个产品模块所需的配置文件、镜像资源以及 chart 包。

可以在下载中心下载最新版本。

CPU 架构	版本	下载地址
AMD64	v0.10.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-amd64.tar
ARM64	v0.10.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-arm64.tar

下载完毕后解压离线包：

```
# 以 amd64 架构离线包为例
tar -xvf offline-v0.10.0-amd64.tar
```

addon 离线包 （可选）

addon 离线包包含一些常用的组件的 Helm Chart 离线包，具体清单请参考 addon。

安装器从 v0.5.0 版本，支持了 addon 的离线包导入能力，如果需要支持 addon 中所有的 helm chart 离线化。可以在下载中心下载最新版本。

首先需要事先下载好离线包，并在集群配置文件（clusterConfig.yaml）中定义 addonOfflinePackagePath。

CPU 架构	版本	下载地址
AMD64	v0.10.0	https://qiniu-download-public.daocloud.io/DaoCloud_DigitalX_Addon/addon-offline-full-package-v0.10.0-amd64.tar.gz
ARM64	v0.10.0	https://qiniu-download-public.daocloud.io/DaoCloud_DigitalX_Addon/addon-offline-full-package-v0.10.0-arm64.tar.gz

ISO 操作系统镜像文件 （必需）

ISO 格式的操作系统镜像文件，安装过程中请根据不同操作系统是来下载对应的 ISO 文件。

ISO 操作系统镜像文件需要在集群配置文件中进行配置，请根据操作系统进行下载。

CPU 架构	操作系统版本	下载地址
AMD6	Centos 7	https://mirrors.tuna.tsinghua.edu.cn/centos/7.9.2009/isos/x86_64/CentO

CPU架构	操作系统版本	下载地址
4		S-7-x86_64-DVD-2009.iso
	Redhat 7、8	https://developers.redhat.com/products/rhel/download#assembly-field-downloads-page-content-61451 注意：Redhat 操作系统需要 Redhat 的账号才可以下载
	Ubuntu 20.04	https://releases.ubuntu.com/focal/ubuntu-20.04.6-live-server-amd64.iso
	统信 UOS V20（1020a）	https://cdimage-download.chinauos.com/uniontechos-server-20-1020a-amd64.iso
	openEuler 22.03	https://mirrors.nju.edu.cn/openeuler/openEuler-22.03-LTS-SP1/ISO/x86_64/openEuler-22.03-LTS-SP1-x86_64-dvd.iso
	OracleLinux R9 U1	https://yum.oracle.com/ISOS/OracleLinux/OL9/u1/x86_64/OracleLinux-R9-U1-x86_64-dvd.iso
ARM64	Kylin Linux Advanced Server release V10 (Sword) SP2	申请地址：https://www.kylinos.cn/scheme/server/1.html 注意：麒麟操作系统需要提供个人信息才能下载使用，下载时请选择 V10 (Sword) SP2

osPackage 离线包（必需）

osPackage 离线包是 Kubean 这个开源项目为 Linux 操作系统离线软件源做的补充内容，例如 openEuler 22.03 中缺少了 selinux-policy-35.5-15.oe2203.noarch.rpm。

安装器从 v0.5.0 版本，需要提供操作系统的 osPackage 离线包，并在集群配置文件（clusterConfig.yaml）中定义 osPackagePath。

其中 Kubean 提供了不同操作系统的 osPackage 离线包，可以前往 https://github.com/kubean-io/kubean/releases 查看。

操作系统版本	下载地址
Centos 7	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-centos7-v0.7.4.tar.gz
Redhat 8	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-redhat8-v0.7.4.tar.gz
Redhat 7	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-redhat7-v0.7.4.tar.gz
Kylin Linux Advanced Server release V10 (Sword) SP2	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-kylinv10-v0.7.4.tar.gz
Ubuntu20.04	https://files.m.daocloud.io/github.com/kubean-

操作系统版本	下载地址
	io/kubean/releases/download/v0.7.4/os-pkgs-ubuntu2004-v0.7.4.tar.gz
openEuler 22.03	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-openeuler22.03-v0.7.4.tar.gz
OracleLinux R9 U1	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-oracle9-v0.7.4.tar.gz

统信 UOS V20（1020a）osPackage 部署请参考 UOS V20 (1020a) 操作系统上部署 DCE 5.0

一键下载所需离线包

我们提供了脚本来一键下载安装 DCE 5.0 所需的离线包。

以下是包含的离线包：

- 前置依赖工具离线包
- osPackage 离线包
- 安装器离线包

!!! note

```
由于不同的 ISO 操作系统下载方式不一致，所以暂不支持。
```

第 2 步：配置集群配置文件

集群配置文件位于离线镜像包 offline/sample 目录下，具体的参数介绍请参考 clusterConfig.yaml。

!!! note

目前离线镜像包中提供了标准的 7 节点模式模板。

第 3 步：开始安装

1. 执行以下命令开始安装 DCE 5.0，安装器二进制文件位置为 offline/dce5-installer

   ./offline/dce5-installer cluster-create -c ./offline/sample/clusterConfig.yaml -m ./offline/sample/manifest.yaml

   !!! note

   安装器脚本命令说明：

   - -c 来指定集群配置文件，必选
   - -m 参数指定 manifest 文件，
   - -z 最小化安装
   - -d 开启 debug 模式
   - 更多命令请使用 --help 查询

2. 安装完成后，命令行会提示安装成功。恭喜您！现在可以通过屏幕提示的 URL 使用默认的账号和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

3. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

产品清单文件 manifest.yaml

此 YAML 文件包含了 DCE 5.0 所有模块信息，主要分为基础设施模块、产品功能模块。

升级操作请参考文档 升级 DCE5.0。

manifest 示例

以下是一个 ClusterConfig 文件示例。

yaml title="manifest.yaml" apiVersion: manifest.daocloud.io/v1alpha1 kind: DCEManifest metadata: global: helmRepo: https://release.daocloud.io/chartrepo imageRepo: release.daocloud.io infrastructures: hwameiStor: enable: true version: v0.10.4 policy: drbd-disabled istio: version: 1.16.1 metallb: version: 0.13.9 contour: version: 10.2.2 enable: false cert-manager: version: 1.11.0 enable: false mysql: version: 8.0.29 cpuLimit: 1 memLimit: 1Gi enableAutoBackup: true redis: version: 6.2.6-debian-10-r120 cpuLimit: 400m memLimit: 500Mi components: kubean: enable: true helmVersion: v0.6.6 helmRepo: https://kubean-io.github.io/kubean-helm-chart variables: ghippo: enable: true helmVersion: 0.18.0 variables: kpanda: enable: true helmVersion: 0.19.0+rc4 variables: kcoral: enable: true helmVersion: 0.4.0+rc1 variables: kcollie: enable: true helmVersion: 0.4.0+rc7 variables: insight: enable: true helmVersion: 0.18.0-rc5 variables: insight-agent: enable: true helmVersion: 0.18.0-rc5 features: tracing ipavo: enable: true helmVersion: 0.10.0 variables: kairship: enable: true helmVersion: 0.10.1 variables: amamba: enable: true helmVersion: 0.18.0+alpha.3 features: argocd jenkins: enable: true helmVersion: 0.1.12 helmRepo: https://release.daocloud.io/chartrepo/amamba skoala: enable: true helmVersion: 0.23.0 variables: mspider: enable: true helmVersion: v0.17.0-rc2 variables: mcamel-rabbitmq: enable: true

helmVersion: 0.12.0-rc2 variables: mcamel-elasticsearch: enable: true helmVersion: 0.9.0-rc2 variables: mcamel-mysql: enable: true helmVersion: 0.10.0-rc2 variables: mcamel-redis: enable: true helmVersion: 0.9.0-rc2 variables: mcamel-kafka: enable: true helmVersion: 0.7.0-rc2 variables: mcamel-minio: enable: true helmVersion: 0.7.0-rc2 variables: mcamel-postgresql: enable: true helmVersion: 0.3.0-rc2 variables: mcamel-mongodb: enable: true helmVersion: 0.1.0-rc1 variables: spidernet: enable: true helmVersion: 0.8.0 variables: kangaroo: enable: true helmVersion: 0.9.0 variables: gmagpie: enable: true helmVersion: 0.3.0 variables: dowl: enable: true helmVersion: 0.3.0+rc1

关键字段

该 YAML 文件中的关键字段说明，请参阅下表。其中包含了基础设施所涉及的组件以及产品功能模块涉及的产品。

字段	说明
infrastructures	DCE 5.0 产品基础设施
infrastructures.xxx.enable	是否开启当前模块，默认为 true
infrastructures.xxx.helmVersion	当前模块的 chart 包版本
infrastructures.hwameiStor	HwameiStor 本地存储
infrastructures.istio	Istio 服务网格
infrastructures.metallb	MetalLB 负载均衡器
infrastructures.contour	Contour 入口控制器
infrastructures.cert-manager	Cert Manager 证书管理
infrastructures.mysql	Mysql 数据库
infrastructures.redis	Redis 数据库
components	DCE 5.0 产品功能模块
components.kubean	集群声明周期管理

字段	说明
components.ghippo	全局管理
components.kpanda	容器管理
components.kcoral	应用备份
components.kcollie	集群巡检
components.insight	可观测性
components.insight-agent	可观测性的数据采集组件
components.ipavo	仪表盘
components.kairship	多云编排
components.amamba	应用工作台
components.jenkins	应用工作台的流水线引擎组件
components.skoala	微服务引擎
components.mspider	服务网格
components.mcamel-*	中间件，包含了 ES、Kafka、MinIO 等
components.kangaroo	镜像仓库
components.gmagpie	报表
components.dowl	集群安全

集群配置文件 clusterConfig.yaml

此 YAML 文件包含了集群的各项配置字段，安装之前必须先配置此文件。 该文件将定义部署模式、集群节点信息等关键参数。默认位于 offline/sample/ 目录。

ClusterConfig 示例

以下是一个 ClusterConfig 文件示例。

```yaml title="clusterConfig.yaml"
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  clusterName: my-cluster

  # 火种节点的域名或 IP，默认解析为火种节点默认网关所在网卡的 IP；可手动填入 IP 或域名，若为域名，如果检测到无法解析，将自动建立此域名和火种节点默认 IP 的映射
  # bootstrapNode: auto

  # kind 火种集群的配置，以下为默认值
  #tinderKind:
  # # kind 集群的容器名称
  # instanceName: my-cluster-installer
  # # kind 集群挂载的主机路径
  # resourcesMountPath: /home/kind
  # registryPort: 443
  # minioServerPort: 9000
  # minioConsolePort: 9001
  # chartmuseumPort: 8081

  loadBalancer:

    # NodePort(default), metallb, cloudLB (Cloud Controller)
    type: metallb
    istioGatewayVip: xx.xx.xx.xx/32 # 当 loadBalancer.type 是 metallb 时必填，为 DCE 提供 UI 和 OpenAPI 访问权限
    insightVip: xx.xx.xx.xx/32 # 别丢弃 /32，当 loadBalancer.type 是 metallb 时必填，用作 global 集群的 Insight 数据采集入口，子集群的 insight-agent 可以向这个 VIP 报告数据

  # 指定 ssh 私钥，定义后无需再定义节点的 ansibleUser、ansiblePass
  # privateKeyPath: /root/.ssh/id_rsa_sample
```

masterNodes: – nodeName: “g–master1” # nodeName 将覆盖 hostName，应符合 RFC1123 标准 ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” – nodeName: “g–master2” ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” – nodeName: “g–master3” ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” workerNodes: – nodeName: “g–worker1” ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” nodeTaints: # 对于 7 节点模式：至少 3 个 worker 节点应打污点（仅 ES 节点），如果使用外接 ES 则不需要添加该污点 – “node.daocloud.io/es–only=true:NoSchedule” # nodeLabels: # daocloud.io/hostname: g–worker1 – nodeName: “g–worker2” ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” nodeTaints: – “node.daocloud.io/es–only=true:NoSchedule” # nodeLabels: # daocloud.io/hostname: g–worker2 – nodeName: “g–worker3” ip: xx.xx.xx.xx ansibleUser: “root” ansiblePass: “dangerous” #ansibleSSHPort: “22” nodeTaints: – “node.daocloud.io/es–only=true:NoSchedule” # nodeLabels: # daocloud.io/hostname: g–worker3

ntpServer: # – 0.pool.ntp.org # – ntp1.aliyun.com # – ntp.ntsc.ac.cn

fullPackagePath: “/root/offline” # 解压后的离线包的路径，离线模式下该字段必填

osRepos: # 操作系统软件源

支持 official–service(default), builtin
type: builtin
isoPath: "/root/CentOS–7–x86_64–DVD–2009.iso"
osPackagePath: "/root/os–pkgs–centos7–v0.4.4.tar.gz"

type: external
Set the block below only if target is S3–compatible storage which need to u

pload files automatically(e.g. minio).
isoPath: "/root/CentOS-7-x86_64-DVD-2009.iso"
osPackagePath: "/root/os-pkgs-centos7-v0.4.4.tar.gz"
externalRepoEndpoint: https://external-repo.daocloud.io
externalRepoUsername: rootuser
externalRepoPassword: rootpass123

type: external
Set the block below if target is other storage which cannot or does not need to upload automatically(e.g. nginx).
That requires you to import the required packages(iso, os-pkgs) manually if not all the required offline resources exist.
`centos` as CentOS, RedHat, kylin, AlmaLinux, Fedora or Openeuler
`debian` as Debian
`ubuntu` as Ubuntu
externalRepoType: centos
externalRepoURLs: ['https://extertal-repo.daocloud.io/kubean/centos/\$releasever/os/\$basearch/']

imagesAndCharts: # 镜像仓库和 Chart 仓库源

official-service(default), builtin or external

type: builtin

type: external
IP or domain name
externalImageRepo: https://external-registry.daocloud.io
Set user and password. Optional
externalImageRepoUsername: admin
externalImageRepoPassword: Harbor12345
chartmuseum or harbor
externalChartRepoType: chartmuseum

IP or domain name
externalChartRepo: https://external-charts.daocloud.io:8081
Set user and password. Optional
externalChartRepoUsername: rootuser
externalChartRepoPassword: rootpass123

addonPackage: # 应用商店 addon 离线包，定义后会对 addon 进行离线部署
path: "/root/addon-offline-full-package-v0.4.8-amd64.tar.gz"

binaries: # 二进制可执行文件

official-service(default), builtin
type: builtin

type: external
IP or domain name
externalRepository: https://external-binaries.daocloud.io:9000/kubean

#externalMiddlewares: # database: # kpanda: # - dbDriverName: "mysql" # # Please refer https://gorm.io/docs/connecting_to_the_database.html # dataSourceName: "user:password@tcp(localhost:3306)/dbname" # # readwrite(default) or readonly # accessType: readwrite # # The maximum number of open connections to the database. # #maxOpenConnections: 100 # # The maximum number of connections in the idle connection pool. # #maxIdleConnections: 10 # # The maximum amount of time a connection may be reused. # #connectionMaxLifetimeSeconds: 3600 # # The maximum amount of time a connection may be idle. # #connectionMaxIdleSeconds: 1800 # ghippoApiserver: # - dbDriverName: "mysql" # dataSourceName: "user:password@tcp(localhost:3306)/dbname" # ghippoKeycloak: # - dbDriverName: "mysql" # dataSourceName: "user:password@tcp(localhost:3306)/dbname" # ghippoAuditserver: # - dbDriverName: "mysql" # dataSourceName:

“user:password@tcp(localhost:3306)/dbname” # elasticsearch: # insight: # endpoint: “https://xx.xx.xx.xx:9200” # # login with basic auth or bearer auth # #anonymous: false # # basic auth # username: “username” # password: “password” # S3Storage: # default: # endpoint: “xx.xx.xx.xx:9000” # # Set if you dont want to verify the certificate. # insecure: true # bucket: “bucketname” # accessKey: “YOUR-ACCESS-KEY-HERE” # secretKey: “YOUR-SECRET-KEY-HERE”

Examples as below. More refer to kubespray options setting documentations. # kubeanConfig: |- # this config will set the timezone of nodes , and it won't change timezone if this config is commented out. # ntp_timezone: Asia/Shanghai # # Enable recommended node sysctl settings # node_sysctl_tuning: true # # Extra node sysctl settings while node_sysctl_tuning is enabled # extra_sysctl: [{ name: net.ipv4.tcp_keepalive_time, value: 700 }] # bin_dir: /usr/local/bin # http_proxy: “” # https_proxy: “” # upstream_dns_servers: # - 8.8.8.8 # - 8.8.4.4

k8sVersion only take effect in online mode, don't set it in offline mode. # Unless to install a non-latest k8s version with offline pkg in place. # k8sVersion: v1.25.4 # auditConfig: # logPath: /var/log/audit/kube-apiserver-audit.log # logHostPath: /var/log/kubernetes/audit # policyFile: /etc/kubernetes/audit-policy/apiserver-audit-policy.yaml # logMaxAge: 30 # logMaxBackups: 10 # logMaxSize: 100 # policyCustomRules: > # - level: None # users: [] # verbs: [] # resources: [] # network: # cni: calico # clusterCIDR: 10.233.64.0/18 # serviceCIDR: 10.233.0.0/18 # cri: # criProvider: containerd # criVersion only take effect in online mode, don't set it in offline mode # criVersion: 1.6.8

关键字段

该 YAML 文件中的关键字段说明，请参阅下表。

字段	说明	默认值
clusterName	在 KuBean Cluster 里的 Global 集群命名	-
tinderKind	火种 kind 集群配置	-
tinderKind.instanceName	火种 kind 集群的容器名称	-
tinderKind.resourcesMountPath	kind 集群挂载的主机路径	/home/kind
tinderKind.registryPort	kind 集群中镜像仓库的端口	443
tinderKind.minioServerPort	kind 集群中 MinIO Server 的端口	9000
tinderKind.minioConsolePort	kind 集群中 MinIO Console 的端口	9001
tinderKind.chartmuseumPort	kind 集群中 ChartMuseum 的端口	8081

masterNodes	Global 集群：Master 节点列表，包括 nodeName/ip/ansibleUser/ansiblePass 几个关键字段	-
masterNodes.nodeName	节点名称，将覆盖 hostName	-
masterNodes.ip	节点 IP	-
masterNodes.ansibleUser	节点账号	-
masterNodes.ansiblePass	节点密码	-
masterNodes.ansibleSSHPort	ssh 的端口，默认为 22	22
workerNodes	Global 集群：Worker 节点列表，包括 nodeName/ip/ansibleUser/ansiblePass 几个关键字段	-
privateKeyPath	kuBean 部署集群的 SSH 私钥文件路径，如果填写则不需要定义 ansibleUser、ansiblePass	-
k8sVersion	kuBean 安装集群的 K8s 版本必须跟 KuBean 和离线包相匹配	-
loadBalancer.insightVip	如果负载均衡模式是 metallb，则需要指定一个 VIP，供给 GLobal 集群的 insight 数据收集入口使用，子集群的 insight-agent 可上报数据到这个 VIP	-

loadBalancer.istioGatewayVip	如果负载均衡模式是 metallb，则需要指定一个 VIP，供给 DCE 的 UI 界面和 OpenAPI 访问入口	-
loadBalancer.type	所使用的 LoadBalancer 的模式，物理环境用 metallb，POC 用 NodePort，公有云和 SDN CNI 环境用 cloudLB	NodePort (default)、metallb、cloudLB (Cloud Controller)
fullPackagePath	解压后的离线包的路径，离线模式下该字段必填	-
addonPackage.path	应用商店 addon 包本地文件系统路径	-
imagesAndCharts	镜像仓库和 Chart 仓库源	-
imagesAndCharts.externalChartRepo	外置 Chart 仓库的 IP 或域名	-
imagesAndCharts.externalChartRepoPassword	外置 Chart 仓库的密码，用于推送镜像	-
imagesAndCharts.externalChartRepoType	外置 Chart 仓库的类型，取值为 chartmuseum，harbor	-
imagesAndCharts.externalChartRepoUsername	外置 Chart 仓库的用户名，用于推送镜像	-
imagesAndCharts.externalImageRepo	指定 external 仓库的 IP 或者域名(需指定协议头)	-
imagesAndCharts.externalImageRepoPassword	外置镜像仓库的密码，用于推送镜像	-

imagesAndCharts.externalImageRepoUsername	外置镜像仓库的用户名，用于推送镜像	-
imagesAndCharts.type	镜像与 Chart 的访问模式，取值为 official-service(在线)，buitin(火种内置 registry 和 chartmuseum)，external(外置)	official-service
auditConfig	k8s api-server 的审计日志配置	默认关闭
binaries	二进制可执行文件	-
binaries.externalRepository	外置二进制可执行文件仓库的访问地址，URL 形式	-
binaries.type	二进制可执行文件的访问模式，取值为 official-service(在线)，builtin(火种节点内置的 minio)	official-service
network.clusterCIDR	Cluster CIDR	-
network.cni	CNI 选择，比如 Calico、Cilium	calico
network.serviceCIDR	Service CIDR	-
ntpServer	可用的 NTP 服务器，供给新节点同步时间	-
osRepos	操作系统软件	

源 | -
|
| osRepos.externalRepoType | 外置软件源服务的操作系统类型，取值为 centos(所有红帽系列)，debian，ubuntu | -
|
| osRepos.externalRepoURLs | 外置软件源的访问地址 | -
|
| osRepos.isoPath | 操作系统 ISO 文件的路径，type 为 builtin 时不能为空 | -
|
| osRepos.osPackagePath | 系统包文件的路径 ，type 为 builtin 时不能为空 | -
|
| osRepos.type | 操作系统软件源的访问模式，取值为 official-service(在线)，builtin(火种节点内置的 minio) | official-service |
| kubeanConfig.ntp_timezone | 设置节点的时区，如果不配置该参数，默认按照节点中的时区 | -
|
| kubeanConfig.node_sysctl_tuning | 开启后默认调整 Global 集群的 Systemctl 内核参数 | false
|
| kubeanConfig.extra_sysctl | 设置额外的 Systemctl 内核参数 | /usr/local/bin
|
| externalMiddlewares | 外置中间件 | -
|
| externalMiddlewares.database | 外置数据库 | -
|

externalMiddlewares.database.ghippoApiserver	ghippoApiserver 外置数据库的配置	–
externalMiddlewares.database.ghippoAuditserver	ghippoAuditserver 外置数据库的配置	–
externalMiddlewares.database.ghippoKeycloak	ghippoKeycloak 外置数据库的配置	–
externalMiddlewares.database.kpanda	kpanda 外置数据库的配置	–
externalMiddlewares.database.kpanda[0].accessType	kpanda 外置数据库的访问类型，取值：readwrite，readonly	readwrite
externalMiddlewares.database.kpanda[0].driver	kpanda 外置数据库的类型，取值：mysql	mysql
externalMiddlewares.database.kpanda[0].dataSourceName	kpanda 外置数据库的访数据源信息，用于连接数据库，可参考 https://gorm.io/docs/connecting_to_the_database.html	–
externalMiddlewares.database.kpanda[0].maxOpenConnections	kpanda 外置数据库的最大连接数	10
externalMiddlewares.database.kpanda[0].maxIdleConnections	kpanda 外置数据库的最大空闲连接数	10
externalMiddlewares.database.kpanda[0].connectionMaxLifetimeSeconds	kpanda 外置数据库的最大连接生命周期	0
externalMiddlewares.database.kpanda[0].connectionMaxIdleTimeSeconds	kpanda 外置数据库的最大空闲连接生命周期	0

|

| externalMiddleware.elasticsearch | 外置 Elasticsear
ch | –
|

| externalMiddleware.elasticsearch.insight | insight 所使用的外
置 Elasticsearch 配置 | –
|

| externalMiddleware.elasticsearch.insight.endpoint | insight 所使用的外
置 Elasticsearch 的访问地址 | –
|

| externalMiddleware.elasticsearch.insight.anonymous | insight 所使用的
外置 Elasticsearch 的匿名访问，取值 true，false，配置为 true 时不应再填访问凭证 |
false |

| externalMiddleware.elasticsearch.insight.username | insight 所使用的
外置 Elasticsearch 的访问用户名 | –
|

| externalMiddleware.elasticsearch.insight.password | insight 所使用的
外置 Elasticsearch 的访问密码 | –
|

精简配置说明

离线模式下采用 builtin 方式安装

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
  creationTimestamp: null
spec:
  clusterName: my-cluster
  masterNodes:
```

```
    # nodeName 将覆盖 hostName，应符合 RFC1123 标准
    - nodeName: "g-master1"
      ip: xx.xx.xx.xx
      ansibleUser: "root"
      ansiblePass: "dangerous"
  workerNodes:
  fullPackagePath: "/root/offline"
  osRepos:
    # official-service(if omit or empty), builtin or external
    type: builtin
    isoPath: "/root/CentOS-7-x86_64-DVD-2009.iso"
    osPackagePath: "/root/os-pkgs-centos7-v0.4.4.tar.gz"
  imagesAndCharts:
    # official-service(if omit or empty), builtin or external
    # 目前还不支持 External S3 ..... FIXME
    type: builtin
  addonPackage:
    path: "/root/addon-offline-full-package-v0.4.8-amd64.tar.gz"
  binaries:
    # official-service(if omit or empty), builtin or external
    type: builtin
```

离线模式下采用 external 方式安装

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
  creationTimestamp: null
spec:
  clusterName: my-cluster
  masterNodes:
    # nodeName 将覆盖 hostName，应符合 RFC1123 标准
    - nodeName: "g-master1"
```

```
      ip: xx.xx.xx.xx
      ansibleUser: "root"
      ansiblePass: "dangerous"
  workerNodes:

  fullPackagePath: "/root/offline"
  osRepos:
    # official-service(if omit or empty), builtin or external
    type: external
    # Optional only if external repo already have full required resources
    isoPath: "/root/CentOS-7-x86_64-DVD-2009.iso"
    # Optional only if external repo already have full required resources
    osPackagePath: "/root/os-pkgs-centos7-v0.4.4.tar.gz"
    # `centos` as CentOS, RedHat,kylin AlmaLinux or Fedora
    # `debian` as Debian
    # `ubuntu` as Ubuntu
    externalRepoType: centos
    externalRepoURLs: ["https://extertal-repo.daocloud.io/centos/\$releasever/os
/\$basearch/"]
  imagesAndCharts:
    # official-service(if omit or empty), builtin or external
    # Not Support External S3 so far..... FIXME
    type: external
    # Optional only if external repo already have full required resources
    # IP or domain name
    externalImageRepo: https://external-registry.daocloud.io
    externalImageRepoUsername: admin
    externalImageRepoPassword: Harbor12345
    # chartmuseum or harbor
    externalChartRepoType: chartmuseum
    # IP or domain name
    externalChartRepo: https://external-charts.daocloud.io:8081
```

```
    externalChartUsername: rootuser
    externalChartMuseumPassword: rootpass123
  addonPackage:
    path: "/root/addon-offline-full-package-v0.4.8-amd64.tar.gz"
  binaries:
    # official-service(if omit or empty), builtin or external
    type: external
    # Optional only if external repo already have full required resources
    # IP or domain name
    externalRepository: https://external-binaries.daocloud.io:9000/kubean
```

在线模式采用 official-service 方式安装

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
  creationTimestamp: null
spec:
  clusterName: my-cluster
  masterNodes:
    # nodeName 将覆盖 hostName，应符合 RFC1123 标准
    - nodeName: "g-master1"
      ip: xx.xx.xx.xx
      ansibleUser: "root"
      ansiblePass: "dangerous"
  workerNodes:
```

通过命令行生成 clusterConfig 配置文件模板

全模式 1 节点模式

```
# 官方在线:
./dce5-installer generate-config --install-mode=cluster-create --master=1 --access-type=official-service
# 官方在线简化版:
```

```
./dce5-installer generate-config --master=1

# 内建离线:
./dce5-installer generate-config --install-mode=cluster-create --master=1 --access-type=builtin
# 内建离线简化版:
./dce5-installer generate-config --master=1 --access-type=builtin

# 扩展离线:
./dce5-installer generate-config --install-mode=cluster-create --master=1 --access-type=external
# 扩展离线简化版:
./dce5-installer generate-config --master=1 --access-type=external
```

全模式 4 节点模式

```
# 官方在线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --access-type=official-service
# 官方在线简化版:
./dce5-installer generate-config --master=3

# 内建离线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --access-type=builtin
# 内建离线简化版:
./dce5-installer generate-config --master=3 --access-type=builtin

# 扩展离线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --access-type=external
# 扩展离线简化版:
./dce5-installer generate-config --master=3 --access-type=external
```

全模式 7 节点模式

```
# 官方在线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --worker=3 --access-type=official-service
# 官方在线简化版:
./dce5-installer generate-config --master=3 --worker=3

# 内建离线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --worker=3 --access-type=builtin
# 内建离线简化版:
./dce5-installer generate-config --master=3 --worker=3 --access-type=builtin

# 扩展离线:
./dce5-installer generate-config --install-mode=cluster-create --master=3 --worker=3 --access-type=external
# 扩展离线简化版:
./dce5-installer generate-config --master=3 --worker=3 --access-type=external
```

社区版

```
# 官方在线
./dce5-installer generate-config --install-mode=install-app --access-type=official-service
# 官方在线简化版:
./dce5-installer generate-config --install-mode=install-app

# 内建离线:
./dce5-installer generate-config --install-mode=install-app --access-type=builtin

# 扩展离线:
```

```
./dce5-installer generate-config --install-mode=install-app --access-type=external
```

部署要求

部署 DCE 5.0 时需要先做好软件规划、硬件规划、网络规划。

操作系统要求

架构	操作系统	内核版本	备注（安装指导文档）
AMD64	centos 7.X	Kernel 3.10.0–1127.el7.x86_64 on an x86_64	操作系统推荐 CentOS 7.9 离线安装 DCE 5.0 商业版
	Redhat 8.X	4.18.0–305.el8.x86_64	操作系统推荐 Redhat 8.4 离线安装 DCE 5.0 商业版
	Redhat 7.X	3.10.0–1160.e17.x86	操作系统推荐 Redhat 7.9 离线安装 DCE 5.0 商业版
	Ubuntu 20.04	5.10.104	离线安装 DCE 5.0 商业版
	统信 UOS V20 （1020a）	5.4.0–125–generic	UOS V20 (1020a) 操作系统上部署 DCE 5.0 商业版
	openEuler 22.03	5.10.0–60.18.0.50.oe2203.x86_64	离线安装 DCE 5.0 商业版
	Oracle Linux R9/R8 U1	5.15.0–3.60.5.1.el9uek.x86_64	Oracle Linux R9 U1 操作系统上部署 DCE 5.0 商业版
	TencentOS Server 3.1	5.4.119–19.0009.14	TencentOS Server 3.1 操作系统上部署 DCE

架构	操作系统	内核版本	备注（安装指导文档）
AR M 64	银河麒麟 OS V10 SP2	4.19.90–24.4.v2101.ky10.aarch64	离线安装 DCE 5.0 商业版

!!! note

非上述表中声明的操作系统，可以参考文档 [Other Linux 离线部署 DCE 5.0 商业版](../os-install/otherlinux.md) 进行安装部署。

硬件要求

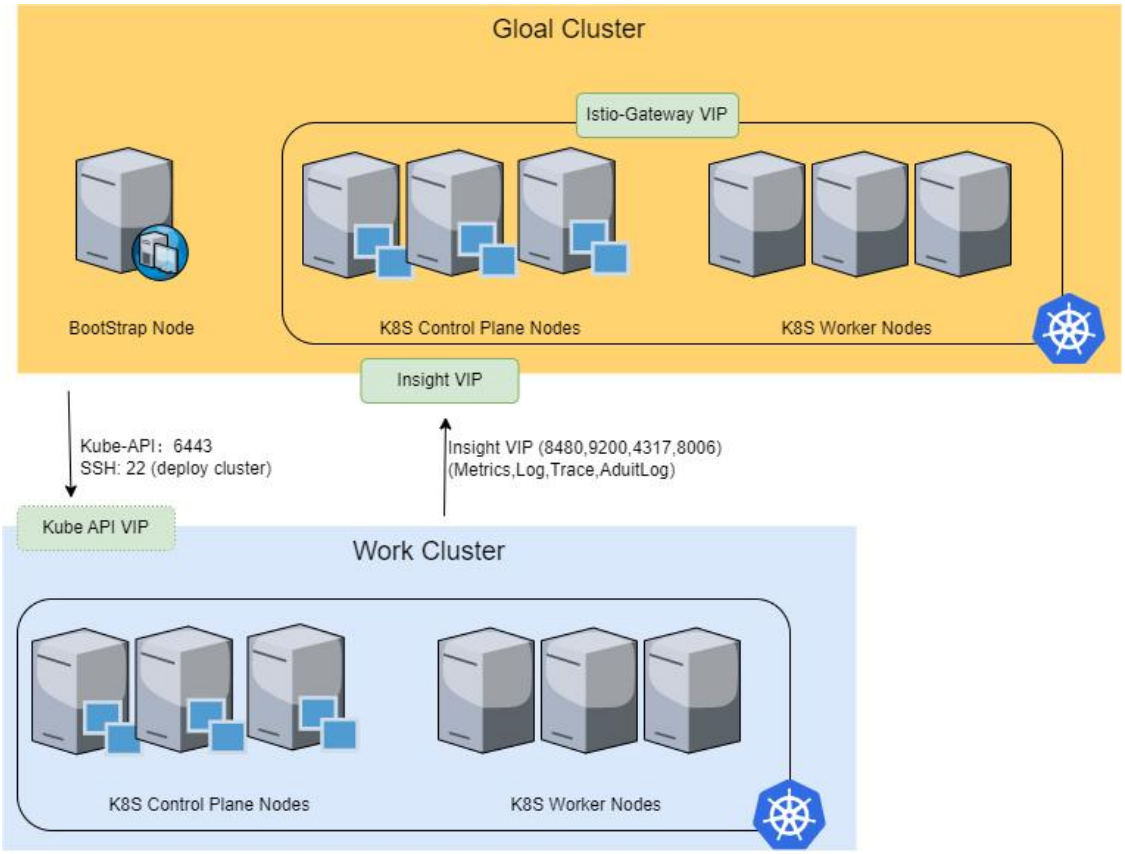
类型	具体要求
CPU	不得超售
内存	不得超售
硬盘	IOPS > 500 吞吐量 >200 MB/s

资源要求，可以参考准备工作

网络要求

网络拓扑

假设使用 VIP 作为全局集群的负载均衡方式：



Network-Topology

网络要求

资源	要求	说明
istioGatewayVip	1 个	如果负载均衡模式是 metallb，则需要指定一个 VIP，供给 DCE 的 UI 界面和 OpenAPI 访问入口
insightVip	1 个	如果负载均衡模式是 metallb，则需要指定一个 VIP，供给 GLobal 集群的 insight 数据收集入口使用，子集群的 insight-agent 可上报数据到这个 VIP
网络速率	1000 M/s	不低于千兆，建议万兆。
协议	-	支持 IPv6。
保留 IP 地址段	需保留两	供 Pod（默认 10.233.64.0/18） 和 Service

资源	要求	说明
	段	（默认 10.233.0.0/18 使用）。如果已经在使用了，可以自定义其他网段避免 IP 地址冲突。
路由	–	服务器有 default 或指向 0.0.0.0 这个地址的路由。
NTP 服务地址	1~4 个	确保您的数据中心有可以访问的 NTP 服务器 IP 地址。
DNS 服务地址	1~2 个	如果您的应用需要 DNS 服务，请准备可以访问 DNS 服务器 IP 地址。

端口要求

为了正常运行，需要打开一些端口。如果网络配置了防火墙规则，则需要确保基础设施组件可以通过特定端口相互通信。 确保所需的以下端口在网络上处于打开状态，并配置为允许主机之间的访问。某些端口是根据配置和用途可选的。

火种节点

Protocol	Port	Description
TCP	443	Docker Registry
TCP	8081	Chart Museum
TCP	9000	Minio API
TCP	9001	Minio UI

Kube 集群（包括 全局集群 和 工作集群）

全局集群 和 工作集群都是通过 Kubean 部署的，因此他们需要打开同样的端口除了标准的 k8s 端口，对于 CNI 和 部分网络组件，也需要打开端口。

k8s Control plane

Protocol	Port	Description
TCP	2379	etcd client port

Protocol	Port	Description
TCP	2380	etcd peer port
TCP	6443	kubernetes api
TCP	10250	kubelet api
TCP	10257	kube-scheduler
TCP	10259	kube-controller-manager

全部 k8s 节点

集群中的每一个节点都需要打开。

Protocol	Port	Description
TCP	22	ssh for ansible
TCP	9100	node exporter(Insight-Agent)
TCP	10250	kubelet api
TCP	30000-32767	kube nodePort range

参考: Kubernetes Docs

Calico (默认)

默认情况下，会使用 Calico 作为 CNI，因此 **全部 k8s 节点** 都需要打开。

Protocol	Port	Description
TCP	179	Calico networking (BGP)
UDP	4789	Calico CNI with VXLAN enabled
TCP	5473	Calico CNI with Typha enabled
UDP	51820	Calico with IPv4 Wireguard enabled
UDP	51821	Calico with IPv6 Wireguard enabled
IPENCAP / IPIP	-	Calico CNI with IPIP enabled

参考: Calico Docs

MetalLB (默认)

当启用 MetalLB 建 VIP 的时候， **全部 k8s 节点** 都需要打开。

Protocol	Port	Description
TCP/UDP	7472	metallb metrics ports
TCP/UDP	7946	metallb L2 operating mode

Cilium (可选)

如果使用 Cilium 作为 CNI，因此 **全部 k8s 节点** 都需要打开。

Protocol	Port	Description
TCP	4240	Cilium Health checks (cilium-health)
TCP	4244	Hubble server
TCP	4245	Hubble Relay
UDP	8472	VXLAN overlay
TCP	9962	Cilium-agent Prometheus metrics
TCP	9963	Cilium-operator Prometheus metrics
TCP	9964	Cilium-proxy Prometheus metrics
UDP	51871	WireGuard encryption tunnel endpoint
ICMP	-	health checks

参考: Cilium Docs

SpiderPool (可选)

如果使用 SpiderPool 作为 CNI，因此 **全部 k8s 节点** 都需要打开。

Protocol	Port	Description
TCP	5710	SpiderPool Agent HTTP Server

Protocol	Port	Description
TCP	5711	SpiderPool Agent Metrics
TCP	5712	SpiderPool Agent gops enabled
TCP	5720	SpiderPool Controller HTTP Server
TCP	5721	SpiderPool Controller Metrics
TCP	5722	SpiderPool Controller Webhook Port
TCP	5723	Spiderpool-CLI HTTP server port.
TCP	5724	SpiderPool Controller gops enabled

参考: SpiderPool Docs

KubeVIP - (可选)

当启用 KubeVIP 建 Kube API VIP 的时候， **全部 Control Plane 节点** 都需要打开。

Protocol	Port	Description
TCP	2112	kube-vip metrics ports

全局集群 其他 需要开放的端口

Istio-Gateway VIP

Protocol	Port	Description	Used By
TCP	80	Istio-Gateway HTTP	Web Browser or API Client
TCP	443	Istio-Gateway HTTPS	Web Browser or API Client

Insight VIP

Protocol	Port	Description	Used By

Protocol	Port	Description	Used By
TCP	8480	Insight VIP for Metrics	所有节点
TCP	9200	Insight VIP for Log	所有节点
TCP	4317	Insight VIP for Trace	所有节点
TCP	8006	Insight VIP for AduitLog	所有节点

工作集群 其他 需要开放的端口

工作集群需要开放 k8s API 的 6443 端口，给到全局管理集群访问。如果要是用部署功能，还要开放 22 端口，给全局集群访问。

Protocol	Port	Description	Used By
TCP	22	各节点 SSH (for ansible)	全局管理集群
TCP	6443	k8s API 访问入口(如 VIP)	全局管理集群

其他各产品需要开放的端口

镜像仓库

Protocol	Port	Description	Used By
TCP	443	访问入口(如 VIP) 的端口	所有节点

客户端浏览器的要求

- Firefox ⩾ 49
- Chrome ⩾ 54

使用外接镜像仓库与 Chart 仓库存储镜像与 Chart 包

本文描述如何使用第三方镜像仓库及 Chart 仓库。

外接镜像仓库与 Chart 仓库 操作步骤

1. 在 集群配置文件 clusterConfig.yaml 中，配置 imagesAndCharts 参数，如下：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  imagesAndCharts:
    type: external

    externalImageRepo: https://external-registry.daocloud.io
    externalImageRepoUsername: admin
    externalImageRepoPassword: Harbor12345

    # 支持 chartmuseum, harbor, jfrog
    externalChartRepoType: chartmuseum
    externalChartRepo: https://external-charts.daocloud.io:8081
    externalChartRepoPassword: rootpass123
  ..........
```

!!! note

- 镜像仓库基本支持社区中开源的所有镜像仓库类型

- Chart 仓库仅支持 chartmuseum、harbor、jfrog 三种

2. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

使用外接中间件服务

本文描述如何使用第三方中间服务，包含：mysql、redis、elasticsearch、S3Storage。

使用外接数据库（MySQL）

前置说明

- DCE 5.0 产品模块中使用了 MySQL 数据库来存储数据，所以仅支持 MySQL 数据库外接
- 下述示例脚本仅用于演示目的，实际应用中应该根据具体的需求进行修改，例如数据库名称、用户名、密码等， 并且可以将以下语句拆分至不同的 DBMS 执行

操作步骤

1. 准备一个 MySQL 数据库，并且具有创建数据库、创建用户、授予权限的权限。
2. 连接到 MySQL 数据库，执行如下 SQL，完成 database、用户的创建并授予对应权限。

```
# ghippo apiserver
CREATE DATABASE ghippo CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'ghippo' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON ghippo.* TO 'ghippo';

# ghippo keycloak
CREATE DATABASE keycloak CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'keycloak' IDENTIFIED BY 'password';
```

```
GRANT ALL PRIVILEGES ON keycloak.* TO 'keycloak';

# ghippo audit
CREATE DATABASE audit CHARACTER SET utf8 COLLATE utf8_general_
ci;
CREATE USER 'audit' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON audit.* TO 'audit';

# kpanda
CREATE DATABASE kpanda CHARACTER SET utf8 COLLATE utf8_gener
al_ci;
CREATE USER 'kpanda' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON kpanda.* TO 'kpanda';

# set sort_buffer_size (used for clusterpedia)
SET GLOBAL sort_buffer_size=8*1024*1024;
SET SESSION sort_buffer_size=8*1024*1024;

# skoala
CREATE DATABASE skoala CHARACTER SET utf8 COLLATE utf8_general
_ci;
CREATE USER 'skoala' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON skoala.* TO 'skoala';

# amamba
CREATE DATABASE amamba CHARACTER SET utf8 COLLATE utf8_gene
ral_ci;
CREATE USER 'amamba' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON amamba.* TO 'amamba';

# insight
```

```
CREATE DATABASE insight CHARACTER SET utf8 COLLATE utf8_genera
l_ci;
CREATE USER 'insight' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON insight.* TO 'insight';

# ipavo
CREATE DATABASE ipavo CHARACTER SET utf8 COLLATE utf8_general_
ci;
CREATE USER 'ipavo' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON ipavo.* TO 'ipavo';

# kcollie
CREATE DATABASE kcollie CHARACTER SET utf8 COLLATE utf8_general
_ci;
CREATE USER 'kcollie' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON kcollie.* TO 'kcollie';

# gmagpie
CREATE DATABASE gmagpie CHARACTER SET utf8 COLLATE utf8_gene
ral_ci;
CREATE USER 'gmagpie' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON gmagpie.* TO 'gmagpie';

# dowl
CREATE DATABASE dowl CHARACTER SET utf8 COLLATE utf8_general_
ci;
CREATE USER 'dowl' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON dowl.* TO 'dowl';
```

3. 在 集群配置文件 clusterConfig.yaml 中，配置 externalMiddlewares.database 参数， 假设数据库访问地址为

localhost:3306；不同的数据库类型有不同的 dataSourceName 配置格式， 详见文档 https://gorm.io/docs/connecting_to_the_database.html

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
  creationTimestamp: null
spec:
  ..............
  externalMiddlewares:
    database:
      kpanda:
        - dbDriverName: "mysql"
          # Please refer https://gorm.io/docs/connecting_to_the_database.html
          dataSourceName: "kpanda:password@tcp(localhost:3306)/dbname"
          # readwrite(default) or readonly
          accessType: readwrite
          # The maximum number of open connections to the database.
          #maxOpenConnections: 100
          # The maximum number of connections in the idle connection pool.
          #maxIdleConnections: 10
          # The maximum amount of time a connection may be reused.
          #connectionMaxLifetimeSeconds: 3600
          # The maximum amount of time a connection may be idle.
          #connectionMaxIdleSeconds: 1800
      ghippoApiserver:
        - dbDriverName: "mysql"
          dataSourceName: "ghippo:password@tcp(localhost:3306)/ghippo"
      ghippoKeycloak:
```

```
    - dbDriverName: "mysql"
      dataSourceName: "keycloak:password@tcp(localhost:3306)/keycl
oak"
  ghippoAuditserver:
    - dbDriverName: "mysql"
      dataSourceName: "audit:password@tcp(localhost:3306)/audit"
  skoala:
    - dbDriverName: "mysql"
      dataSourceName: "skoala:password@tcp(172.30.41.0:3308)/skoal
a"
  amamba:
    - dbDriverName: "mysql"
      dataSourceName: "amamba:password@tcp(172.30.41.0:3308)/am
amba"
  insight:
    - dbDriverName: "mysql"
      dataSourceName: "insight:password@tcp(172.30.41.0:3308)/insig
ht"
  ipavo:
    - dbDriverName: "mysql"
      dataSourceName: "ipavo:password@tcp(172.30.41.0:3308)/ipavo"
  kcollie:
    - dbDriverName: "mysql"
      dataSourceName: "kcollie:password@tcp(172.30.41.0:3308)/kcolli
e"
  gmagpie:
    - dbDriverName: "mysql"
      dataSourceName: "gmagpie:password@tcp(172.30.41.0:3308)/gm
agpie"
  dowl:
    - dbDriverName: "mysql"
      dataSourceName: "dowl:password@tcp(172.30.41.0:3308)/dowl"
```

4. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

使用外接 Redis

操作步骤如下：

1. 在 集群配置文件 clusterConfig.yaml 中，配置 externalMiddlewares.redis 参数：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  externalMiddlewares:
    redis:
      kpanda: "redis://:password@localhost:6379"
  ..........
```

!!! note

- 支持 Redis Standalone、Redis Sentinel、Redis Cluster 三种模式

- Standalone URL 格式为：`redis://[[user]:password@]host[:port][/db-number][?option=value]`

- Sentinel URL 格式为：`redis+sentinel://[[user]:password@]host1[:port1][,host2[:port2]]/master-name[/db-number][?option=value]`

- Cluster URL 格式为：`redis://[[user]:password@]host1[:port1]?addr=host2[:port2][&addr=host3:[port3][&option=value]] 或 rediss://[[user]:password@]host1[:port1]?addr=host2[:port2][&addr=host3:[port3][&option=value]]`

- 目前仅有容器管理产品模块使用到了 Redis 组件

2. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

使用外接 Elasticsearch

操作步骤如下：

1. 在 集群配置文件 clusterConfig.yaml 中，配置 externalMiddlewares.elasticsearch 参数：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  externalMiddlewares:
    elasticsearch:
      insight:
        endpoint: "https://xx.xx.xx.xx:9200"
        # basic auth
        username: "username"
        password: "password"
  ..........
```

!!! note

目前仅有可观测产品模块使用到了 Elasticsearch 组件。
如果使用外接中间件后，不建议使用 7 节点模式下的 worker 节点，不然占用资源。

2. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

使用外接 S3Storage

操作步骤如下：

1. 在 集群配置文件 clusterConfig.yaml 中，配置 externalMiddlewares.S3Storage 参数：

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  externalMiddlewares:
    S3Storage:
      default:
        endpoint: "https://xx.xx.xx.xx:9200"
        # Set if you dont want to verify the certificate.
        insecure: true
        bucket: "bucketname"
        accessKey: "YOUR-ACCESS-KEY-HERE"
        secretKey: "YOUR-SECRET-KEY-HERE"
  ..........
```

2. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

使用外接服务存储 OS Repo 资源

本文描述如何使用第三方存储服务的 OS Repo 资源并且在安装器安装时进行指定。支持两种类型：S3 兼容服务(如 minio)，非 S3 兼容服务(如 nginx)

前提条件

- 根据要部署的环境下载 ISO 操作系统镜像文件
- 根据要部署的环境下载 osPackage 离线包

操作步骤

使用 S3 兼容服务

S3 兼容的服务只需要在 集群配置文件 clusterConfig.yaml 中简单配置即可，无需其他操作。

1. 在 集群配置文件 clusterConfig.yaml 中，配置 osRepo 相关的参数：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  osRepos:
    type: external
    isoPath: "/root/CentOS-7-x86_64-DVD-2009.iso"
    osPackagePath: "/root/os-pkgs-centos7-v0.4.4.tar.gz"
    externalRepoEndpoint: https://external-repo.daocloud.io
    externalRepoUsername: rootuser
    externalRepoPassword: rootpass123
  ..........
```

!!! note

给定的用户名需要具有 bucket 的读写权限。

使用非 S3 兼容服务

非 S3 兼容的服务需要先手动将下载好的 ISO 操作系统镜像文件、osPackage 离线包导入， 然后在集群配置文件 clusterConfig.yaml 中配置相关参数。

以下内容以 CentOS 7.9 x86_64 作为集群节点，使用 nignx 作为 http server， 理论上其他通用 http server 也能支持，需要注意 URL 访问路径和文件路径的映射关系。

1. 确保有一个可用的 nginx 服务，及服务所在节点的登录和文件写入权限；
2. 下载/拷贝 ISO 操作系统镜像文件、osPackage 离线包至 nginx 服务所在节点，并将 ISO 导入脚本从火种节点拷贝至 nginx 服务所在节点；

   !!! note

   ISO 导入脚本在[离线包镜像包](../start-install.md/#_2)中，路径为 `./offline/offline-iso/import_iso.sh`
3. 确定需要导入的路径；
   1. 通过 nginx.conf (nginx -t 命令查看改文件路径) 检测 nginx 服务所在节点的文件路径和 URL 路径的映射关系，下方示例供参考：

```
http {
    server {
        listen          8080;
        server_name  _;
        location / {
            root    /usr/share/nginx/html;
            index   index.html index.htm;
        }
    }
}
```

上方配置说明 nginx http 服务的访问根路径映射本地目录 /usr/share/nginx/html。

2. 如果是普通方式部署的 nginx 服务，则选定导入路径为 /usr/share/nginx/html。

3. 如果是容器部署的 nginx 服务，需要挂载宿主机路径至容器，且挂载的宿主机路径对应着映射了 http 服务的容器本地路径，即存在这样的关系： http-path -> container-path -> host-path。则导入路径应为 host-path，host-path 需要手动按照附录 2 确认。

4. 执行如下命令导入 ISO 操作系统镜像文件、osPackage 离线包：

```
cat > import.sh << "EOF"
[ ! -d "${MAPPING_PATH}" ] && echo "mapping path ${MAPPING_PATH} not found" && exit 1
[ ! -x "${ISO_IMPORT_SH_PATH}" ] && echo "iso import script ${ISO_IMPORT_SH_PATH} not found or not executable" && exit 1
[ ! -f "${OS_PKGS_PATH}" ] && echo "os pkgs ${OS_PKGS_PATH} not found" && exit 1
[ ! -f "${ISO_PATH}" ] && echo "iso ${ISO_PATH} not found" && exit 1
tar -xzvf ${OS_PKGS_PATH} && for arch in amd64 arm64; do tar --strip-components=1 -xzvf os-pkgs/os-pkgs-${arch}.tar.gz -C ${MAPPING_PATH}; done && rm -rf os-pkgs
bash ${ISO_IMPORT_SH_PATH} ${MAPPING_PATH} ${ISO_PATH}
EOF
export MAPPING_PATH="/usr/share/nginx/html"
export ISO_IMPORT_SH_PATH="./import_iso.sh"
export OS_PKGS_PATH="./os-pkgs-centos7-v0.4.5-rc3.tar.gz"
export ISO_PATH="./CentOS-7-x86_64-DVD-2009.iso"
bash ./import.sh
```

其中环境变量 MAPPING_PATH 代表步骤 3 中提及的导入路径

5. 验证是否导入成功

   登录一台全局服务集群节点，假设 nignx 访问地址为 http://10.0.1.1:8080，参考附录 1 进行配置，执行如下命令：

```
cat > /etc/yum.repos.d/test.repo << "EOF"
[test0]
baseurl = http://10.1.1.1:8080/centos/$releasever/os/$basearch
gpgcheck = 0
name = test0

[test1]
baseurl = http://10.1.1.1:8080/kubean/centos-iso/$releasever/os/$basearch
gpgcheck = 0
name = test0
EOF
yum clean all && yum makecache --disablerepo=* --enablerepo=test0,test1
```

   !!! note

   其他操作系统也是类似操作，因为具体的操作系统的包管理器的软件源配置有一些差异

6. 在 集群配置文件 clusterConfig.yaml 中，配置 osRepo 相关的参数，externalRepoURLs 参考附录 1。

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  osRepos:
    type: external
```

```
    # `centos` as CentOS, RedHat,kylin AlmaLinux or Fedora,Openeuler
    # `debian` as Debian
    # `ubuntu` as Ubuntu
    externalRepoType: centos
    externalRepoURLs:
      - 'http://10.0.1.1:8080/centos/\$releasever/os/\$basearch/'
      - 'http://10.0.1.1:8080/centos-iso/\$releasever/os/\$basearch/'
  ..........
```

7. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

附录

1. 操作系统与对应的 RepoURLs

${address_prefix} 替换为 HTTP 服务的外部访问地址，如 http://10.0.1.1:8080

OS	RepoURLs
CentOS	['${address_prefix}/centos/\$releasever/os/\$basearch','${address_prefix}/centos-iso/\$releasever/os/\$basearch']
RedHat	['${address_prefix}/redhat/\$releasever/os/\$basearch','${address_prefix}/redhat-iso/\$releasever/os/\$basearch/BaseOS','${address_prefix}/redhat-iso/\$releasever/os/\$basearch/AppStream']
Kylin V10	['${address_prefix}/kubean/kylin/\$releasever/os/\$basearch','${address_prefix}/kubean/kylin-iso/\$releasever/os/\$basearch']

OS	RepoURLs
UOS V20	[‘${address_prefix}/kubean/uos/\$releasever/os/\$basearch’,‘${address_prefix}/kubean/uos-iso/\$releasever/os/\$basearch/AppStream’,‘${address_prefix}/kubean/uos-iso/\$releasever/os/\$basearch/BaseOS’]
Oracle 9	[‘${address_prefix}/kubean/oracle/\$releasever/os/\$basearch’,‘${address_prefix}/kubean/oracle-iso/\$releasever/os/\$basearch/AppStream’,‘${address_prefix}/kubean/oracle-iso/\$releasever/os/\$basearch/BaseOS’]
OpenEuler 20.03	[‘${address_prefix}/kubean/openeuler/22.03/os/\$basearch’,‘${address_prefix}/kubean/openeuler-iso/22.03/os/\$basearch’]
Ubuntu bionic	[‘deb [trusted=yes] ${address_prefix}/kubean/ubuntu/amd64 bionic/’,‘deb [trusted=yes] ${address_prefix}/kubean/ubuntu-iso bionic main restricted’]
Ubuntu focal	[‘deb [trusted=yes] ${address_prefix}/kubean/ubuntu/am

OS	RepoURLs
	d64 focal/',‘deb [trusted=yes] ${address_prefix}/kubean/ubuntu-iso focal main restricted’]

2. 查看容器卷挂载列表

CLI tool	Command
docker	docker inspect ${CONTAINER_ID} -f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
nerdctl	nerdctl inspect ${CONTAINER_ID} -f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
podman	podman inspect ${CONTAINER_ID} -f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
crictl	crictl inspect -o go-template --template '{{range .status.mounts}}{{printf "hostPath: %s containerPath: %s\n" .hostPath .containerPath }}{{end}}' ${CONTAINER_ID}
ctr	ctr c info ${CONTAINER_ID} --spec 检查 mounts 字段
kubectl	kubectl -n ${NAMESPACE} get pod ${POD_NAME} -oyaml 检查 volumes 和 volumeMounts 字段

使用外接服务存储 Binaries 资源

本文描述如何使用第三方存储服务存储 Binaries 资源并且在安装器安装时进行指定。 支持两种类型：S3 兼容服务（如 minio）、非 S3 兼容服务（如 nginx）。

操作步骤

使用 S3 兼容服务

S3 兼容的服务只需要在 集群配置文件 clusterConfig.yaml 中简单配置即可，无需其他操作。

1. 配置 clusterConfig.yaml，设置 binaries 相关的参数，如下：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  binaries:
    type: external
    externalRepoEndpoint: https://external-repo.daocloud.io
    externalRepoUsername: rootuser
    externalRepoPassword: rootpass123
  ..........
```

!!! note

给定的用户名需要具有 bucket 的读写权限

使用非 S3 兼容服务

非 S3 兼容的服务需要先手动将下载好的镜像离线包 目录下的 offline/kubespray-binary/offline-files.tar.gz binaries 离线包导入， 然后在集群配置文件 clusterConfig.yaml 中配置相关参数。

以下内容以 CentOS 7.9 x86_64 作为集群节点，使用 nignx 作为 http server， 理论上其他通用 http server 也能支持，需要注意 URL 访问路径和文件路径的映射关系。

1. 确保有一个可用的 nginx 服务，及服务所在节点的登录和文件写入权限；
2. 将 binaries 离线包从火种节点拷贝至 nginx 服务所在节点；

   !!! note

   binaries 离线包路径为 `./offline/kubespray-binary/offline-files.tar.gz`

3. 确定需要导入的路径;

   1. 通过 nginx.conf 检测 nginx 服务所在节点的文件路径和 URL 路径的映射关系，下方示例供参考：

```
http {
    server {
        listen          8080;
        server_name  _;
        location / {
            root    /usr/share/nginx/html;
            index   index.html index.htm;
        }
    }
}
```

   上方配置说明 nginx http 服务的访问根路径映射本地目录 /usr/share/nginx/html

2. 如果是普通方式部署的 nginx 服务，则选定导入路径为 /usr/share/nginx/html

3. 如果是容器部署的 nginx 服务，需要挂载宿主机路径至容器，且挂载的宿主机路径对应着映射了 http 服务的容器本地路径， 即存在这样的关系： http-path -> container-path -> host-path。则导入路径应为 host-path。host-path 需要手动按照附录确认。

4. 执行如下命令导入离线 binaries 离线包：

```
cat > import.sh << "EOF"
[ ! -d "${MAPPING_PATH}" ] && echo "mapping path ${MAPPING_PATH} not found" && exit 1
[ ! -f "${BINARIRES_PKG_PATH}" ] && echo "binaries package path ${BINARIRES_PKG_PATH} not found" && exit 1
tar -xzvf ${BINARIRES_PKG_PATH} --strip-components=1 -C ${MAPPING_PATH}
EOF
export MAPPING_PATH="/usr/share/nginx/html"
export BINARIRES_PKG_PATH="./offline-files.tar.gz"
bash ./import.sh
```

其中环境变量 MAPPING_PATH 代表步骤 3 中提及的导入路径。

5. 在 集群配置文件 clusterConfig.yaml 中，配置 binaries 相关的参数。

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  ..........
  binaries:
    type: external
```

```
    externalRepoEndpoint: http://10.0.1.1:8080
.....
```

6. 完成上述配置后，可以继续执行部署 DCE 5.0 商业版。

附录

查看容器卷挂载列表：

CLI tool	Command
docker	docker inspect ${CONTAINER_ID} –f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
nerdctl	nerdctl inspect ${CONTAINER_ID} –f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
podman	podman inspect ${CONTAINER_ID} –f '{{range .Mounts}}{{printf "hostPath: %s containerPath: %s\n" .Source .Destination}}{{end}}'
crictl	crictl inspect –o go–template –– template '{{range .status.mounts}}{{printf "hostPath: %s containerPath: %s\n" .hostPath .containerPath }}{{end}}' ${CONTAINER_ID}
ctr	ctr c info ${CONTAINER_ID} ––spec 检查 mounts 字段
kubectl	kubectl –n ${NAMESPACE} get pod ${POD_NAME} –oyaml 检查 volumes 和 volumeMounts 字段

Oracle Linux R9/R8 U1 操作系统上部署 DCE 5.0 商业版

本文将介绍如何在 Oracle Linux R9/R8 U1 操作系统上部署 DCE 5.0，v0.8.0 及以上支持。

前提条件

- 请提前阅读部署架构，确认本次部署模式。
- 请提前阅读部署要求，确认网络、硬件、端口等是否符合需求。
- 请提前阅读准备工作，确认机器资源及前置检查。

离线安装

1. 下载全模式离线包，可以在下载中心下载最新版本。

CPU架构	版本	下载地址
AMD64	v0.10.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-amd64.tar

   下载完毕后解压离线包：

   ```
   curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-amd64.tar
   tar -xvf offline-v0.10.0-amd64.tar
   ```

2. 下载 Oracle Linux R9/R8 U1 镜像文件。

   ```
   ## Oracle Linux R9 U1
   curl -LO https://yum.oracle.com/ISOS/OracleLinux/OL9/u1/x86_64/Oracl
   ```

eLinux-R9-U1-x86_64-dvd.iso

```
## Oracle Linux R8 U1
curl -LO https://yum.oracle.com/ISOS/OracleLinux/OL8/u7/x86_64/OracleLinux-R8-U7-x86_64-dvd.iso
```

3. 下载 Oracle Linux R9/R8 U1 osPackage 离线包。

```
## Oracle Linux R9 U1
curl -LO https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-oracle9-v0.7.4.tar.gz

## Oracle Linux R8 U1
curl -LO https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.7.4/os-pkgs-oracle8-v0.7.4.tar.gz
```

4. 下载 addon 离线包，可以在下载中心下载最新版本（可选）

5. 设置集群配置文件 clusterConfig.yaml，可以在离线包 offline/sample 下获取该文件并按需修改。 参考配置为：

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  clusterName: oracle-cluster
  loadBalancer:
    type: metallb
    istioGatewayVip: 172.30.41.XXX/32
    insightVip: 172.30.41.XXX/32
  masterNodes:
    - nodeName: "g-master"
      ip: 172.30.41.XXX
      ansibleUser: "root"
```

```
      ansiblePass: "******"
  fullPackagePath: "/root/workspace/offline"
  osRepos:
    type: builtin
    isoPath: "/root/workspace/iso/OracleLinux-R9-U1-x86_64-dvd.iso"
    osPackagePath: "/root/workspace/os-pkgs/os-pkgs-oracle9-v0.5.4.tar.gz"
  imagesAndCharts:
    type: builtin
  binaries:
    type: builtin
  kubeanConfig: |-
    # 打开 sysctl 推荐配置，避免出现`too many open files`问题
    node_sysctl_tuning: true
    # 禁止 kubespray 为 oracel linux 安装 public repo
    use_oracle_public_repo: false
```

!!! note

由于安装过程中 `kpanda-controller-manager` 组件报错 `failed to create fs notify watcher: too many open files.`，
所以需要在 `clusterConfig.yaml` 文件中设置 `node_sysctl_tuning: true`。

6. 开始安装 DCE 5.0。

```
./dce5-installer cluster-create -m ./sample/mainfest.yaml -c ./sample/clusterConfig.yaml
```

!!! note

部分参数介绍，更多参数可以通过 `./dce5-installer --help` 来查看：

- `-z` 最小化安装
- `-c` 指定集群配置文件，使用 NodePort 暴露控制台时不需要指定 `-c`

- `-d` 开启 debug 模式
- `--serial` 指定后所有安装任务串行执行

7. 安装完成后，命令行会提示安装成功。恭喜您！:smile: 现在可以通过屏幕提示的 URL 使用默认的账户和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

8. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

Other Linux 离线部署 DCE 5.0 商业版

本文将介绍如何在 Other Linux 操作系统上部署 DCE 5.0，v0.7.0 及以上支持。

Other Linux 本质上是由于 DCE 对某些 Linux 操作系统没有提供安装系统离线包（OS package），需要客户自己去制作。

已验证操作系统

架构	操作系统	所属系统族	推荐内核
AMD64	统信 UOS V20 (1050d)	Debian	4.19.0-server-amd64
AMD64	AnolisOS 8.8 GA	Redhat	5.10.134-13.an8.x86_64

!!! note

没有验证的操作系统，可以尝试通过本文档的教程尝试部署。

前提条件

- 请提前阅读部署架构，确认本次部署模式。
- 请提前阅读部署要求，确认网络、硬件、端口等是否符合需求。
- 请提前阅读准备工作，确认机器资源及前置检查。

制作操作系统离线包（OS package）

制作及安装

1. 下载制作工具。

```
cd /home
curl -Lo ./pkgs.yml https://raw.githubusercontent.com/kubean-io/kubean/main/build/os-packages/others/pkgs.yml
```

```
curl -Lo ./other_os_pkgs.sh https://raw.githubusercontent.com/kubean-io/kubean/main/build/os-packages/others/other_os_pkgs.sh && chmod +x other_os_pkgs.sh
```

2. 构建操作系统离线包

```
# 指定 pkgs.yml 包配置文件路径（若 pkgs.yml 位于 other_os_pkgs.sh 同级路径，则可以不设置此环境变量）
export PKGS_YML_PATH=/home/pkgs.yml
# 执行系统离线包构建命令
./other_os_pkgs.sh build
```

3. 安装操作系统离线包

```
# 指定 pkgs.yml 包配置文件路径（若 pkgs.yml 位于 other_os_pkgs.sh 同级路径，则可以不设置此环境变量）
export PKGS_YML_PATH=/home/pkgs.yml
# 指定 os pkgs 离线包的路径
export PKGS_TAR_PATH=/home/os-pkgs-${DISTRO}-${VERSION}.tar.gz
# 指定集群 master/worker 节点 IP（多节点 IP 地址以空格分割）
export HOST_IPS='192.168.10.11 192.168.10.12'
# 指定安装的目标节点接入信息（多节点用户名密码需保持一致）
export SSH_USER=root
export SSH_PASS=dangerous
# 执行安装命令，并输出日志
./other_os_pkgs.sh install >>log.txt
```

4. 安装成功后，会输出如下日志：

```
[root@master test]# cat log.txt |egrep 'INFO|WARN'
[WARN]  skip install yq ...
[INFO]  succeed to install package 'python-apt'
[INFO]  succeed to install package 'python3-apt'
[INFO]  succeed to install package 'aufs-tools'
```

```
[INFO]  succeed to install package 'apt-transport-https'
[INFO]  succeed to install package 'software-properties-common'
[INFO]  succeed to install package 'conntrack'
[INFO]  succeed to install package 'apparmor'
[WARN]  the package 'libseccomp2' has been installed
[INFO]  succeed to install package 'ntp'
[WARN]  the package 'openssl' has been installed
[INFO]  succeed to install package 'curl'
[INFO]  succeed to install package 'rsync'
[INFO]  succeed to install package 'socat'
[WARN]  the package 'unzip' has been installed
[WARN]  the package 'e2fsprogs' has been installed
[WARN]  the package 'xfsprogs' has been installed
[INFO]  succeed to install package 'ebtables'
[WARN]  the package 'bash-completion' has been installed
[WARN]  the package 'tar' has been installed
[INFO]  succeed to install package 'ipvsadm'
[INFO]  succeed to install package 'ipset'
[INFO]  All packages for Node (192.168.10.11) have been installed.
```

注意

1. 通过 cat log.txt |egrep 'INFO|WARN'检查安装情况：

   如果出现 failed to install package 关键字，则说明未安装成功，并且最终失败时， 会输出 the packages that failed to install are: ipset ipvsadm xfsprogs。

2. 相同系统族（os family）的不同版本（major version）所对应的包名存在差异:

系统族	版本	包名
Debian	< 11	python-apt
	>= 11	python3-apt

系统族	版本	包名
Redhat Major Version	< 8	libselinux-python
	>= 8	python3-libselinux

开始离线安装

1. 下载全模式离线包，可以在下载中心下载最新版本。

CPU架构	版本	下载地址
AMD64	v0.6.4	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.6.1-amd64.tar

   下载完毕后解压离线包：

   ```
   curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.6.1-amd64.tar
   tar -xvf offline-v0.6.1-amd64.tar
   ```

2. 下载 Other Linux 操作系统镜像，此处以 UnionTech OS Server 20 1050d 为例：

   ```
   curl -LO https://cdimage-download.chinauos.com/uniontechos-server-20-1050d-amd64.iso
   ```

3. 参考上一步制作操作系统离线包。

4. 下载 addon 离线包，可以在下载中心下载最新版本（可选）

5. 设置集群配置文件 clusterConfig.yaml，可以在离线包 offline/sample 下获取该文件并按需修改。

   === “UnionTech OS Server 20 1050d”

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  clusterName: test-cluster
  loadBalancer:
    type: metallb
    istioGatewayVip: 172.30.41.XXX/32
    insightVip: 172.30.41.XXX/32
  masterNodes:
    - nodeName: "g-master1"
      ip: 172.30.41.xxx
      ansibleUser: "root"
      ansiblePass: "******"
  fullPackagePath: "/root/offline"
  osRepos:
    type: none
  imagesAndCharts:
    type: builtin
  binaries:
    type: builtin
  kubeanConfig: |-
  allow_unsupported_distribution_setup: true
    debian_os_family_extensions:
      - "UnionTech OS Server 20\" "
```

=== “AnolisOS 8.8 GA”

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
```

```
metadata:
spec:
  clusterName: test-cluster
  loadBalancer:
    type: metallb
    istioGatewayVip: 172.30.41.XXX/32
    insightVip: 172.30.41.XXX/32
  masterNodes:
    - nodeName: "g-master1"
      ip: 172.30.41.xxx
      ansibleUser: "root"
      ansiblePass: "******"
  fullPackagePath: "/root/offline"
  osRepos:
    type: none
  imagesAndCharts:
    type: builtin
  binaries:
    type: builtin
  kubeanConfig: |-
  allow_unsupported_distribution_setup: true
    redhat_os_family_extensions:
      - "Anolis OS"
```

配置参数说明：

参数	说明	是否必填
spec.kubeanConfig.allow_unsupported_distribution_setup	是否跳过已支持发行版检测	必填
spec.kubeanConfig.debian_os_family_extensions	可通过查看 ansible_os_family 来填	若为 Debian 系统族则需填写

参数	说明	是否必填
spec.kubeanConfig.redhat_os_family_extensions	可通过查看 ansible_os_family 来填写	若为 Redhat 系统族则需填写

如何查看当前发行版环境的系统族标识：

```
export USER=root
export PASS=xxxx
export ADDR=192.168.10.xxx
export ANSIBLE_HOST_KEY_CHECKING=False
ansible -m setup -a 'filter=ansible_os_family' -e "ansible_user=${USER} ansible_password=${PASS}" -i ${ADDR}, all
```

执行成功后将输出以下信息：

```
192.168.10.xxx | SUCCESS => {
    "ansible_facts": {
        "ansible_os_family": "UnionTech OS Server 20\" ",
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```

6. 开始安装 DCE 5.0。

```
./dce5-installer cluster-create -m ./sample/mainfest.yaml -c ./sample/clusterConfig.yaml
```

!!! note

部分参数介绍，更多参数可以通过 `./dce5-installer --help` 来查看：

- `-z` 最小化安装

- `-c` 指定集群配置文件，使用 NodePort 暴露控制台时不需要指定 `-c`
- `-d` 开启 debug 模式
- `--serial` 指定后所有安装任务串行执行

7. 安装完成后，命令行会提示安装成功。恭喜您！:smile: 现在可以通过屏幕提示的 URL 使用默认的账户和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

8. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

UOS V20 (1020a) 操作系统上部署 DCE 5.0 商业版

本文将介绍如何在 UOS V20(1020a) 操作系统上部署 DCE 5.0，v0.6.0 及以上支持。

前提条件

- 请提前阅读部署架构，确认本次部署模式。
- 请提前阅读部署要求，确认网络、硬件、端口等是否符合需求。
- 请提前阅读准备工作，确认机器资源及前置检查。

离线安装

1. 由于安装器依赖 python，所以需要在火种机器中先安装 python3.6。

   ```
   ## 执行以下命令下载依赖
   dnf install -y --downloadonly --downloaddir=rpm/python36

   ## 执行以下命令开始安装
   rpm -ivh python3-pip-9.0.3-18.uelc20.01.noarch.rpm python3-setuptools-39.2.0-7.uelc20.2.noarch.rpm python36-3.6.8-2.module+uelc20+36+6174170c.x86_64.rpm
   ```

2. 下载全模式离线包，可以在下载中心下载最新版本。

CPU架构	版本	下载地址
AMD64	v0.6.1	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.6.1-amd64.tar

   下载完毕后解压离线包：

```
curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.6.1-amd64.tar
tar -xvf offline-v0.6.1-amd64.tar
```

3. 下载 UnionTech Server V20 1020a ISO 镜像文件。

   ```
   curl -LO https://cdimage-download.chinauos.com/uniontechos-server-20-1020a-amd64.iso
   ```

4. 制作 **os-pkgs-uos-20.tar.gz** 文件。

   下载制作脚本

   ```
   curl -Lo ./build.sh https://raw.githubusercontent.com/kubean-io/kubean/main/build/os-packages/others/uos_v20/build.sh
   chmod +x build.sh
   ```

   执行脚本生成 **os-pkgs-uos-20.tar.gz** 文件：

   ```
   ./build.sh
   ```

5. 下载 addon 离线包，可以在下载中心下载最新版本（可选）

6. 设置集群配置文件 clusterConfig.yaml，可以在离线包 offline/sample 下获取该文件并按需修改。 参考配置为：

   ```yaml
   apiVersion: provision.daocloud.io/v1alpha3
   kind: ClusterConfig
   metadata:
   spec:
     clusterName: my-cluster
     loadBalancer:
       type: metallb
       istioGatewayVip: 172.30.41.XXX/32
       insightVip: 172.30.41.XXX/32
     masterNodes:
   ```

```
    - nodeName: "g-master1"
      ip: 10.5.14.XXX
      ansibleUser: "root"
      ansiblePass: "******"
  fullPackagePath: "/home/offline"
  osRepos:
    type: builtin
    isoPath: "/home/uniontechos-server-20-1020a-amd64.iso" ## ISO 的目录
    osPackagePath: "/home/os-pkgs-uos-20.tar.gz" ## os-pkgs 的目录
  imagesAndCharts:
    type: builtin
  addonPackage:
    path: "/home/addon-offline-full-package-v0.5.3-alpha2-amd64.tar.gz" ## addon 的目录
  binaries:
    type: builtin
```

7. 开始安装 DCE 5.0。

./dce5-installer cluster-create -m ./sample/mainfest.yaml -c ./sample/clusterConfig.yaml

!!! note

部分参数介绍，更多参数可以通过 `./dce5-installer --help` 来查看：

- `-z` 最小化安装
- `-c` 指定集群配置文件，使用 NodePort 暴露控制台时不需要指定 `-c`
- `-d` 开启 debug 模式
- `--serial` 指定后所有安装任务串行执行

8. 安装完成后，命令行会提示安装成功。恭喜您！:smile: 现在可以通过屏幕提示的 URL 使用默认的账户和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

9. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

TencentOS Server 3.1 操作系统上部署 DCE 5.0 商业版

本文将介绍如何在 TencentOS Server 3.1 操作系统上部署 DCE 5.0，v0.9.0 及以上支持。

前提条件

- 请提前阅读部署架构，确认本次部署模式。
- 请提前阅读部署要求，确认网络、硬件、端口等是否符合需求。
- 请提前阅读准备工作，确认机器资源及前置检查。

离线安装

1. 下载全模式离线包，可以在下载中心下载最新版本。

CPU 架构	版本	下载地址
AMD64	v0.9.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.9.0-amd64.tar

   下载完毕后解压离线包：

   ```
   curl -LO https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.9.0-amd64.tar
   tar -xvf offline-v0.9.0-amd64.tar
   ```

2. 下载 TencentOS Server 3.1 镜像文件。

   ```
   curl -LO http://mirrors.tencent.com/tlinux/3.1/iso/x86_64/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso
   ```

3. 下载 TencentOS Server 3.1 osPackage 离线包。

```
curl -LO https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.6.6/os-pkgs-tencent31-v0.6.6.tar.gz
```

4. 下载 addon 离线包，可以在下载中心下载最新版本（可选）

5. 设置集群配置文件 clusterConfig.yaml，可以在离线包 offline/sample 下获取该文件并按需修改。 参考配置为：

```
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
spec:
  clusterName: tencent-cluster
  loadBalancer:
    type: metallb
    istioGatewayVip: 172.30.41.XXX/32
    insightVip: 172.30.41.XXX/32
  masterNodes:
    - nodeName: "g-master"
      ip: 172.30.41.XXX
      ansibleUser: "root"
      ansiblePass: "******"
  fullPackagePath: "/root/workspace/offline"
  osRepos:
    type: builtin
    isoPath: "/root/workspace/iso/TencentOS-Server-3.1-TK4-x86_64-minimal-2209.3.iso"
    osPackagePath: "/root/workspace/os-pkgs/os-pkgs-tencent31-v0.6.1.tar.gz"
  imagesAndCharts:
    type: builtin
  binaries:
    type: builtin
```

```
  kubeanConfig: |-
    allow_unsupported_distribution_setup: true
    redhat_os_family_extensions:
      - "TencentOS"
```

!!! note

- TencentOS Server 3.1 操作系统属于 Redhat 系统族，所以需要在 kubean Config 中定义 `redhat_os_family_extensions` 参数。
- 执行如下命令查看 TencentOS Server 3.1 操作系统的 OS Family 标识，输出结果为 `TencentOS`。

    ```bash
    export USER=root
    export PASS=dangerous
    export ADDR=172.30.41.166
    export ANSIBLE_HOST_KEY_CHECKING=False
    ansible -m setup -a 'filter=ansible_os_family' -e "ansible_user=${USER} ansible_password=${PASS}" -i ${ADDR}, all
    ```

6. 开始安装 DCE 5.0。

```
./dce5-installer cluster-create -m ./sample/mainfest.yaml -c ./sample/clusterConfig.yaml
```

!!! note

部分参数介绍，更多参数可以通过 `./dce5-installer --help` 来查看：

- `-z` 最小化安装
- `-c` 指定集群配置文件，使用 NodePort 暴露控制台时不需要指定 `-c`
- `-d` 开启 debug 模式
- `--serial` 指定后所有安装任务串行执行

7. 安装完成后，命令行会提示安装成功。恭喜您！:smile: 现在可以通过屏幕提示的 URL 使用默认的账户和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

8. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

在 OpenShift OCP 上安装 DCE 5.0 商业版

本文将介绍如何在 OCP 上安装 DCE 5.0。

前提条件

- DCE 5.0 默认支持的 Kubernetes 版本为 v1.22.x、v1.23.x、v1.24.x、v1.25.x、v1.26.x
- 已经拥有一个 OCP 环境，并且版本不低于 v1.22.x
- 准备一个私有镜像仓库，并且保证集群可以访问到
- 确保资源充足，建议集群至少还有 12 核 24 GB 的可用资源

离线安装

1. 通过堡垒机登录到 Control plane 节点。

2. 下载全模式离线包，可以在下载中心下载最新版本。

CPU架构	版本	下载地址
AMD64	v0.10.0	https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-amd64.tar
ARM64	v0.10.0	https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.10.0-arm64.tar

   下载完毕后解压离线包：

```
## 以 amd64 架构离线包为例
tar -xvf offline-v0.10.0-amd64.tar
```

3. 设置集群配置文件 clusterConfig.yaml，可以在离线包 offline/sample 下获取该文件并按需修改。

   参考配置为：

```yaml
apiVersion: provision.daocloud.io/v1alpha3
kind: ClusterConfig
metadata:
  creationTimestamp: null
spec:
  loadBalancer:
    type: metallb # 建议 metallb
    istioGatewayVip: 10.5.14.XXX/32
    insightVip: 10.5.14.XXX/32
  fullPackagePath: /home/offline # 离线包目录
  imagesAndCharts:
    type: external
    externalImageRepo: http://10.5.14.XXX:XXXX # 私有镜像仓库地址
    externalImageRepoUsername: admin
    externalImageRepoPassword: Harbor12345
```

4. 配置 manifest 文件（可选），可以在离线包 offline/sample 下获取该文件并按需修改。

   !!! note

   如果要使用 `hwameiStor`作为 StorageClass，请确保当前集群中有没有默认的 StorageClass。
   如果有，需要去除。如果不去除，默认 StorageClass 需要关闭 `hwameiStor`，即更改 enable 的值为 `fasle`。

5. 开始安装 DCE 5.0。

./offline/dce5-installer install-app -m ./offline/sample/manifest.yaml -c ./offline/sample/clusterConfig.yaml --platform openshift -z

!!! note

部分参数介绍，更多参数可以通过 `./dce5-installer --help` 来查看：

- `-z` 最小化安装
- `-c` 指定集群配置文件，使用 NodePort 暴露控制台时不需要指定 -c
- `-d` 开启 debug 模式
- `--platform` 用来声明在哪个 Kubernetes 发行版上部署 DCE 5.0，目前仅支持 openshift
- `--serial` 指定后所有安装任务串行执行

6. 安装完成后，命令行会提示安装成功。恭喜您！:smile: 现在可以通过屏幕提示的 URL 使用默认的账户和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

7. 成功安装 DCE 5.0 商业版之后，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

集群资源规划

您可以在标准 Kubernetes 集群上安装 DCE 5.0，也可以在 kind 集群上安装 DCE 5.0。

标准 Kubernetes 集群资源的要求如下：

资源项	要求
节点硬件	**3 Master + 2 Worker** Master 节点：4 Core，8 G；系统盘 100G Worker 节点：4 Core，8 G；系统盘 100G
K8s 版本	推荐 K8s 官方最高稳定版本，目前推荐 v1.26，最低支持 v1.22
支持的 CRI	Docker 和 containerd
CPU 架构	x86_64 和 ARM64

kind 集群资源的要求如下，仅推荐用于测试和开发。

资源项	要求
节点硬件	CPU > 10 核、内存 > 12 GB、磁盘空间 > 100 GB
K8s 版本	推荐 K8s 官方最高稳定版本，目前推荐 v1.26，最低支持 v1.22
支持的 CRI	Docker 和 containerd
CPU 架构	x86_64 和 ARM64

已有 Kubernetes 集群离线安装社区版

本页简要说明如何在已有 Kubernetes 集群上离线安装 DCE 5.0 社区版。

点击社区版部署 Demo 可观看视频演示。

准备工作

- 准备一个 Kubernetes 集群，集群配置请参考文档资源规划。

  !!! note

  - 存储：需要提前准备好 StorageClass，并设置为默认 SC
  - 确保集群已安装 CoreDNS
  - 如果是单节点集群，请确保您已移除该节点的污点

- 安装依赖项。

  !!! note

  如果集群中已安装所有依赖项，请确保依赖项版本符合要求：

  - helm ⩾ 3.11.1
  - skopeo ⩾ 1.11.1
  - kubectl ⩾ 1.25.6
  - yq ⩾ 4.31.1

下载和安装

1. 在 k8s 集群控制平面节点（Controller Node）下载社区版的对应离线包并解压，或者从下载中心下载离线包并解压。

   假定版本 VERSION=0.10.0

   ```
   export VERSION=v0.10.0
   wget https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5
   ```

```
/offline-community-$VERSION-amd64.tar
tar -xvf offline-community-$VERSION-amd64.tar
```

2. 设置集群配置文件 clusterConfig.yaml

   - 如果是非公有云环境（虚拟机、物理机），请启用负载均衡 (metallb)，以规避 NodePort 因节点 IP 变动造成的不稳定。请仔细规划您的网络，设置 2 个必要的 VIP，配置文件范例如下：

     yaml title="clusterConfig.yaml" apiVersion: provision.daocloud.io/v1alpha3 kind: ClusterConfig spec: loadBalancer: type: metallb istioGatewayVip: 10.6.229.10/32 insightVip: 10.6.229.11/32 fullPackagePath: absolute-path-of-the-offline-directory # 解压离线包后的路径 imagesAndCharts: # 镜像仓库 type: external externalImageRepo: your-external-registry # 镜像仓库地址，必须是 http 或者 https # externalImageRepoUsername: admin # externalImageRepoPassword: Harbor123456

   - 如果是公有云环境，并通过预先准备好的 Cloud Controller Manager 的机制提供了公有云的 k8s 负载均衡能力，配置文件范例如下:

     yaml title="clusterConfig.yaml" apiVersion: provision.daocloud.io/v1alpha3 kind: ClusterConfig spec: loadBalancer: type: cloudLB fullPackagePath: absolute-path-of-the-offline-directory # 解压离线包后的路径 imagesAndCharts: # 镜像仓库 type: external externalImageRepo: your-external-registry # 镜像仓库地址，必须是 http 或者 https # externalImageRepoUsername: admin # externalImageRepoPassword: Harbor123456

   - 如果使用 NodePort 暴露控制台（仅推荐 PoC 使用），配置文件范例如下:

yaml title="clusterConfig.yaml" apiVersion: provision.daocloud.io/v1alpha3 kind: ClusterConfig spec: loadBalancer: type: NodePort fullPackagePath: absolute-path-of-the-offline-directory # 解压离线包后的路径 imagesAndCharts: # 镜像仓库 type: external externalImageRepo: your-external-registry # 镜像仓库地址，必须是 http 或者 https # externalImageRepoUsername: admin # externalImageRepoPassword: Harbor123456

3. 安装 DCE 5.0。

   ./dce5-installer install-app -c clusterConfig.yaml

   !!! note

   - 有关 clusterConfig.yaml 文件设置，请参考[在线安装第 2 步](online.md#_2)。
   - `-z` 最小化安装
   - `-c` 指定集群配置文件。使用 NodePort 暴露控制台时不需要指定 `-c`。
   - `-d` 开启 debug 模式
   - `--serial` 指定后所有安装任务串行执行

4. 安装完成后，命令行会提示安装成功。恭喜您！ 现在可以通过屏幕提示的 URL 使用默认的账号和密码（admin/changeme）探索全新的 DCE 5.0 啦！



success

!!! success

请记录好提示的 URL，方便下次访问。

5. 另外，安装 DCE 5.0 成功之后，您需要正版授权后使用，请参考申请社区免费体验。

已有 Kubernetes 集群在线安装社区版

本页简要说明如何在已有 Kubernetes 集群上在线安装 DCE 5.0 社区版。

!!! note

- 点击[在线安装社区版](../../../videos/install.md)可观看视频演示。
- 如果需要离线安装，请查阅[离线安装步骤](offline.md)。

准备工作

- 准备一个 K8s 集群，集群配置请参考文档集群资源规划。
  - 提前准备好 StorageClass，并设置为默认 SC
  - 确保集群已安装 CoreDNS
  - 如果是单节点集群，确保已移除该节点的污点
- 安装依赖项。

  如果集群中已安装所有依赖项，请确保依赖项版本符合要求：

  - helm ≥ 3.11.1
  - skopeo ≥ 1.11.1
  - kubectl ≥ 1.25.6
  - yq ≥ 4.31.1

下载和安装

1. 在 K8s 集群控制平面节点（Controller Node）下载 dce5-installer 二进制文件（也可以通过浏览器下载）。

   假定 VERSION 为 v0.10.0

   export VERSION=v0.10.0

   ## 如果是 arm 架构请更新 `dce5-installer-$VERSION` 为 `dce5-installer-

$VERSION-linux-arm64`

curl -Lo ./dce5-installer https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/dce5-installer-$VERSION

chmod +x ./dce5-installer

2. 设置配置文件 clusterConfig.yaml

   - 如果使用 NodePort 暴露控制台（仅推荐 PoC 使用），直接执行下一步。
   - 如果是非公有云环境（虚拟机、物理机），请启用负载均衡 (metallb)，以规避 NodePort 因节点 IP 变动造成的不稳定。请仔细规划您的网络，设置 2 个必要的 VIP，配置文件范例如下：

     yaml title="clusterConfig.yaml" apiVersion: provision.daocloud.io/v1alpha3 kind: ClusterConfig spec: loadBalancer: type: metallb istioGatewayVip: 10.6.229.10/32 # (1) insightVip: 10.6.229.11/32 # (2)

     1. 这是 Istio gateway 的 VIP，也是 DCE 5.0 控制台的浏览器访问 IP
     2. 全局服务集群的 Insight-Server 采集所有子集群监控指标的网络路径所用的 VIP

   - 如果是公有云环境，并通过预先准备好的 Cloud Controller Manager 的机制提供了公有云的 K8s 负载均衡能力，配置文件范例如下:

     yaml title="clusterConfig.yaml" apiVersion: provision.daocloud.io/v1alpha3 kind: ClusterConfig spec: loadBalancer: type: cloudLB

3. 安装 DCE 5.0。

./dce5-installer install-app -c clusterConfig.yaml -z

!!! note

- 参数 `-p` 指定解压离线包的 offline 目录。
- 有关 clusterConfig.yaml 文件设置，请参考[在线安装第 2 步](online.md#_2)。
- `-z` 最小化安装
- `-c` 指定集群配置文件。使用 NodePort 暴露控制台时不需要指定 `-c`。
- `-d` 开启 debug 模式
- `--serial` 指定后所有安装任务串行执行

4. 安装完成后，命令行会提示安装成功。恭喜您！ 现在可以通过屏幕提示的 URL 使用 **默认的账号和密码（admin/changeme）** 探索全新的 DCE 5.0 啦！



安装成功

!!! success

请记录好提示的 URL，方便下次访问。

5. 另外，安装 DCE 5.0 成功之后，您需要正版授权后使用，请参考申请社区免费体验。

新手尝鲜在线安装 DCE 5.0 社区版

本页说明如何使用 kind 集群实现新手尝鲜在线安装 DCE 5.0 社区版。

!!! note

点击[在线安装社区版](../../../videos/install.md#3)可观看视频演示。

准备工作

- 准备一台机器，机器资源建议：CPU > 8 核、内存 > 12 GB、磁盘空间 > 100 GB。
- 确保机器能够连通互联网。

执行如下脚本，检查系统资源和联网情况：

```
set -e
if [ $(free -g|grep Mem | awk '{print $2}') -lt 12 ]; then (echo "insufficient memory! (should >=12G)";); fi
if [ $(grep 'processor' /proc/cpuinfo |sort |uniq |wc -l) -lt 8 ]; then (echo "insufficient CPU! (should >=8C)";); fi
if [ $(df -m / |tail -n 1 | awk '{print $4}') -lt 30720 ]; then (echo "insufficient free disk space of root partition!(should >=30G)";); fi
ping daocloud.io -c 1 &> /dev/null || ( echo "no connection to internet! abort.")
echo "precheck pass.."
```

预期输出类似于：

```
precheck pass..
```

安装 Docker

如果您的机器已安装了 Docker 且版本高于 1.18，请跳过这一步。

安装 Docker 时可使用国内源：

https://developer.aliyun.com/mirror/docker-ce

===“如果是 CentOS”

```shell
set -e
if [ -x "$(command -v docker )" ] ;then
    echo "docker already installed : version = "$(docker -v);
else
    echo "docker not found, please install it first."
fi

sudo yum install -y yum-utils device-mapper-persistent-data lvm2
sudo yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
sudo sed -i 's+download.docker.com+mirrors.aliyun.com/docker-ce+' /etc/yum.repos.d/docker-ce.repo
sudo yum makecache fast
sudo yum -y install docker-ce
sudo service docker start
sudo systemctl enable docker
sudo yum install -y wget
```

===“如果是 Ubuntu”

```shell
set -e
if [ -x "$(command -v docker )" ] ;then
    echo "docker already installed : version = "$(docker -v);
else
    echo "docker not found, please install it first."
fi
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-propert
```

```
ies-common
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository --yes "deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get -y update
sudo apt-get -y install docker-ce
sudo apt-get -y install wget
sudo service docker start
sudo systemctl enable docker
```

请注意，如果机器上已有 Podman，但没有 Docker，依然需要安装 Docker。这是因为一个已知问题：Podman 虽然可以启动 kind，但是会出现文件句柄不足的问题，并且重启会出现 IP 对不上的问题。

安装 kind 集群

1. 下载 kind 的二进制文件包。

   ```
   curl -Lo ./kind https://qiniu-download-public.daocloud.io/kind/v0.17.0/kind-linux-amd64
   chmod +x ./kind
   old_kind=$(which kind)
   if [ -f "$old_kind" ]; then mv ./kind $old_kind; else mv ./kind /usr/bin/kind ; fi
   ```

2. 查看 kind 版本。

   ```
   kind version
   ```

   预期输出如下：

   ```
   kind v0.17.0 go1.19.2 linux/amd64
   ```

3. 设置 kind_cluster.yaml 配置文件。

注意，暴露集群内的 32088 端口到 kind 对外的 8888 端口（可自行修改），配置文件示例如下：

```yaml title="kind_cluster.yaml"
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 32088
    hostPort: 8888
```

4. 创建一个名为 fire-kind-cluster 的 K8s v1.25.3 示例集群。

```
kind create cluster --image release.daocloud.io/kpanda/kindest-node:v1.25.3 --name=fire-kind-cluster --config=kind_cluster.yaml
```

预期输出如下：

```
Creating cluster "fire-kind-cluster" ...
 √ Ensuring node image (release.daocloud.io/kpanda/kindest-node:v1.25.3) ��
 √ Preparing nodes ��
 √ Writing configuration ��
 √ Starting control-plane ��
 √ Installing CNI ��
 √ Installing StorageClass ��
Set kubectl context to "kind-fire-kind-cluster"
```

5. 查看新创建的集群。

```
kind get clusters
```

预期输出如下：

```
fire-kind-cluster
```

安装 DCE 5.0 社区版

1. 安装依赖项。

!!! note

如果集群中已安装所有依赖项，请确保依赖项版本符合要求：

- helm ⩾ 3.11.1
- skopeo ⩾ 1.11.1
- kubectl ⩾ 1.25.6
- yq ⩾ 4.31.1

2. 在 kind 主机下载 dce5-installer 二进制文件。

假定 VERSION 为 v0.10.0

```
export VERSION=v0.10.0
curl -Lo ./dce5-installer https://proxy-qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/dce5-installer-$VERSION
chmod +x ./dce5-installer
```

3. 获取 kind 所在主机的 IP，然后开始安装 DCE 5.0。

```
myIP=$(ip -o route get 1.1.1.1 | cut -d " " -f 7)
./dce5-installer install-app -z -k $myIP:8888
```

!!! note

kind 集群仅支持 NodePort 模式。
安装过程持续 30 分钟以上，具体取决于镜像拉取的网速。
可通过以下命令观察安装过程中的 Pod 启动情况：

```shell
docker exec -it fire-kind-cluster-control-plane kubectl get po -A -w
```

4. 安装完成后，命令行会提示安装成功。恭喜您！ 现在可以通过屏幕提示的 URL 使用 **默认的账号和密码（admin/changeme）** 探索全新的 DCE 5.0 啦！



安装成功

!!! success

- 请记录好提示的 URL，方便下次访问。
- 成功安装 DCE 5.0 社区版后，请[申请社区免费体验](../../../dce/license0.md)。
- 如果安装过程中遇到什么问题，欢迎扫描二维码，与开发者畅快交流：

![社区版交流群](https://docs.daocloud.io/daocloud-docs-images/docs/images/assist.png)

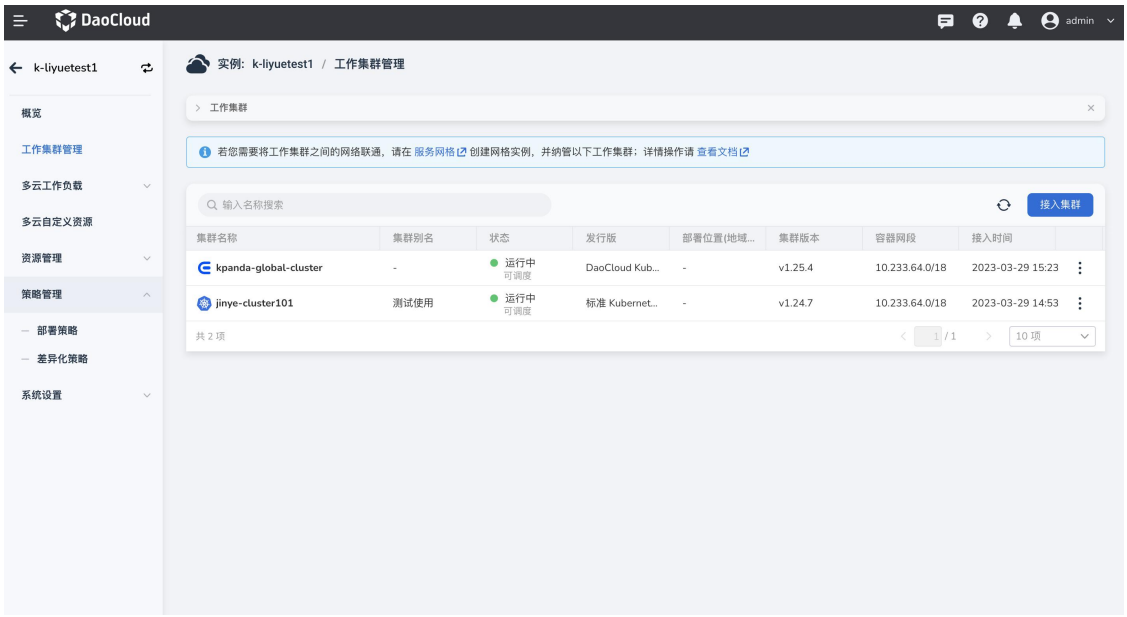
管理工作集群

创建多云实例后，需要先接入工作集群。

接入集群

1. 点击左侧导航栏的工作集群管理，然后在右上角点击接入集群。

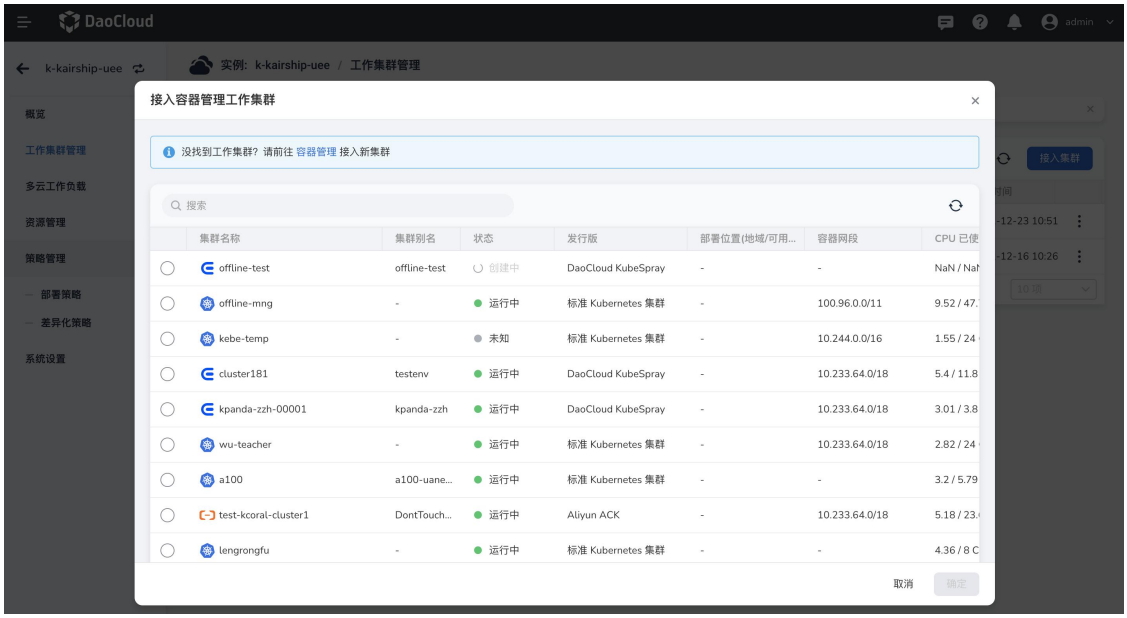
   若想联通工作集群之间的网络，需在服务网格页面创建网格实例并纳管工作集群，操作详情可参考多云网络互联。



接入集群

2. 选择想要添加的工作集群，点击确认。

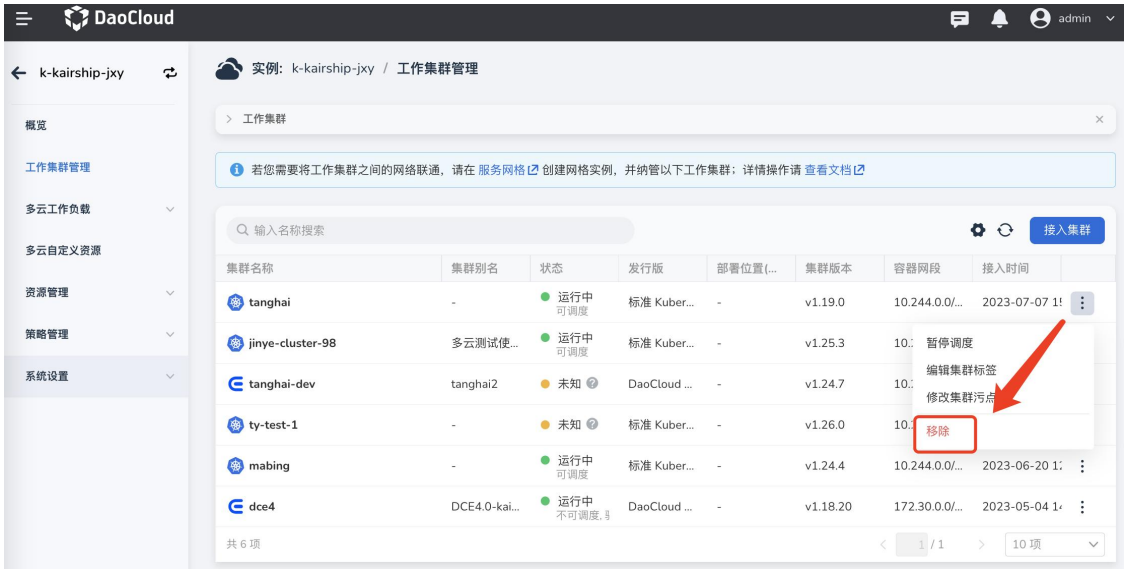
   若列表中没有您想添加的目标集群，可进入容器管理模块接入或创建集群。



确认

移除集群

1. 在集群右侧点击 … 按钮，在弹出菜单中选择移除。



removal

2. 在弹窗中输入集群的名称，确认无误后点击确认移除。



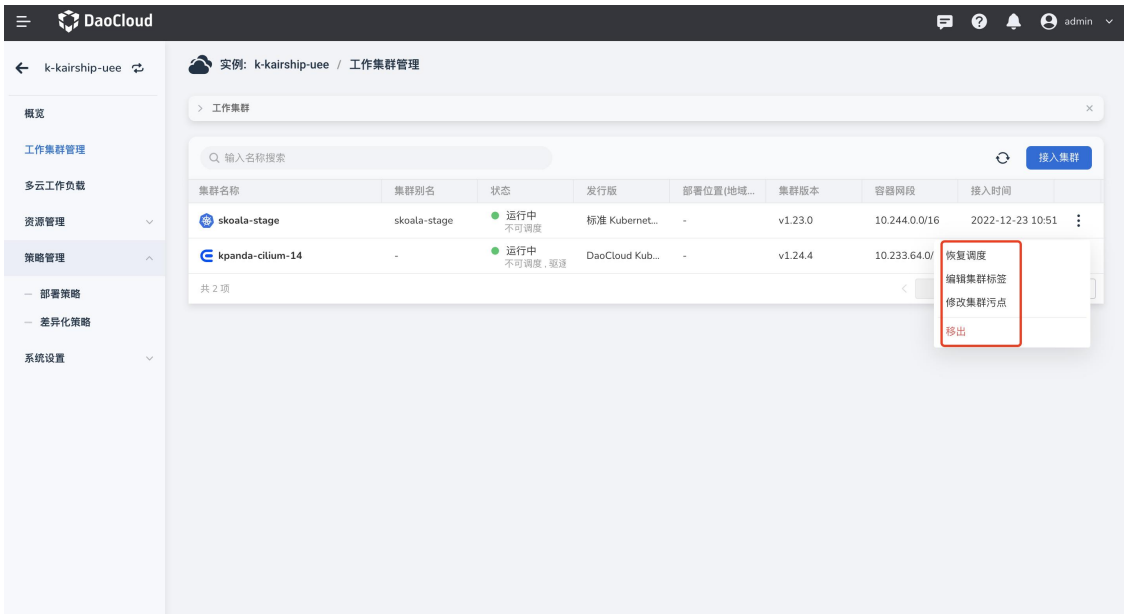
removal

!!! note

– 移除工作集群后，该集群工作负载将不再受此多云实例的管辖。
– 已经分发到此集群的所有多云工作负载将自动迁移到此多云实例管辖的其他工作集群。

更多操作

系统会自动返回工作集群列表。在集群右侧点击 … 按钮可以执行暂停/恢复调度、编辑集群标签、修改集群污点等操作。



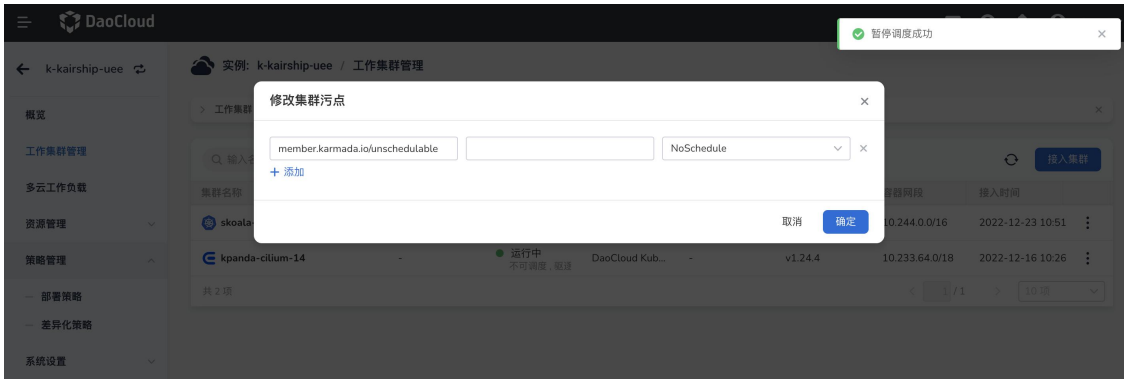
更多操作

暂停/恢复调度

1. 点击暂停调度可以让集群暂停调度，即新的多云资源不会再被分发到该集群。但之前已经分发到该集群的多云资源则不受影响。

2. 此时点击修改集群污点，发现集群被自动加上了污点。

   若想恢复调度也可以点击恢复调度，或者手动去除对应的集群污点。



暂停调度

修改集群污点

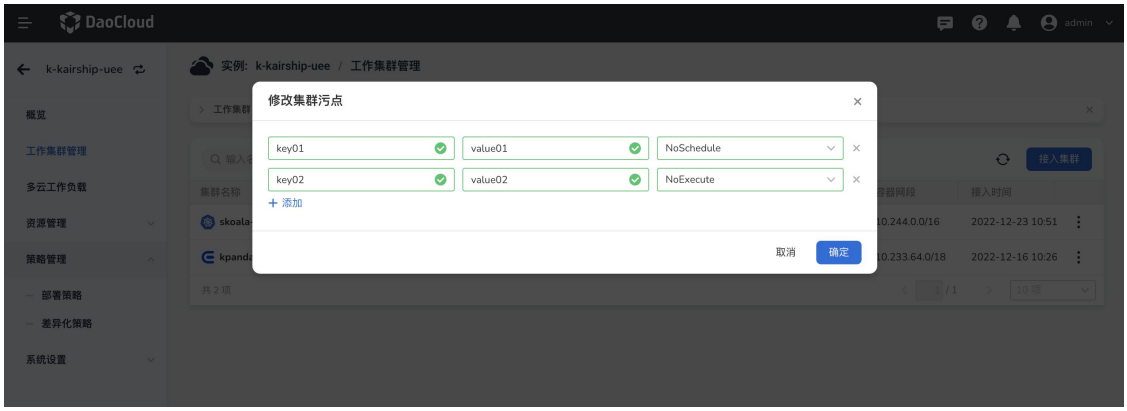
1. 点击修改集群污点，可以控制集群可以部署的工作负载、容器等资源。

2. 在弹框内输入污点的键值信息，值可以为空。

3. 下拉选择污点效果，点击确定。

   支持添加多个污点。目前支持两种污点效果：

效果	描述
NoSchedule	只有拥有和这个污点相匹配的容忍度的 Node 才能被分配到该集群上
NoExecute	定义驱逐行为以应对集群故障。若污点效果定义为 NoExecute，当工作负载或者 Pod 已经在该集群上运行，则会被驱逐。如果没有在该集群运行，也不会再被调度到该集群上


DaoCloud
admin
k-kairship-uee
概览
工作集群管理
多云工作负载
资源管理
策略管理
部署策略
差异化策略
系统设置
实例: k-kairship-uee / 工作集群管理
接入集群
集群名称
容器网段
接入时间
skoala-
kpand
10.244.0.0/16
10.233.64.0/18
2022-12-23 10:51
2022-12-16 10:26
共 2 项
1 / 1
10 项/页
修改集群污点
key01
value01
NoSchedule
key02
value02
NoExecute
添加
取消
确定

修改集群污点

工作空间

工作空间是 DCE 5.0 为管理全局资源而设计的一种资源层级映射。工作空间可以理解为部门下的项目，管理员通过文件夹和工作空间映射企业中的部门和项目关系。有关工作空间的详细说明，可参考工作空间与层级。

注意事项

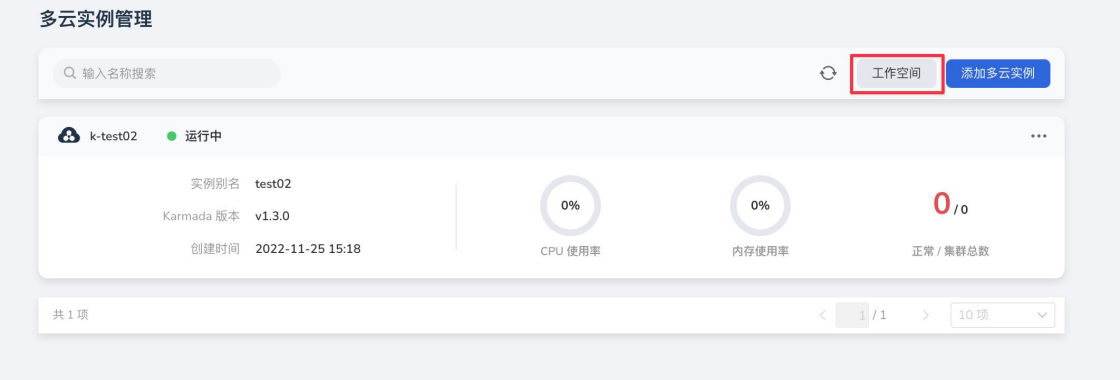
- 当前操作用户应具有 Admin 或 Workspace Admin 权限。有关权限的更多说明，可参考角色和权限管理和工作空间权限。
- 多云实例/命名空间与工作空间绑定之后，多云实例下的所有工作集群会自动同步这种绑定关系，允许对应工作空间的用户在集群中操作对应资源。
- 绑定之后，可以进入全局管理模块的工作空间与层级->资源组查看该资源，详情可参考资源配额
- 目前一个多云实例/命名空间只能绑定到一个工作空间，映射关系为一对一。
    - 这样可以确保资源与权限的稳定匹配，防止资源冲突和权限冲突。
    - 后续会支持将多云实例或命名空间作为共享资源绑定到多个工作空间。此项功能正在开发中，敬请期待。

绑定/解绑工作空间

在多云编排模块中，DCE 5.0 平台的 admin 管理员可以将多云实例或多云命名空间以资源的形式绑定到某个工作空间。

多云实例或命名空间被绑定到工作空间之后，工作空间内的用户就能直接获得该实例或命名空间的相应权限。

1. 在多云编排模块的首页右上角点击工作空间。



管理入口.png

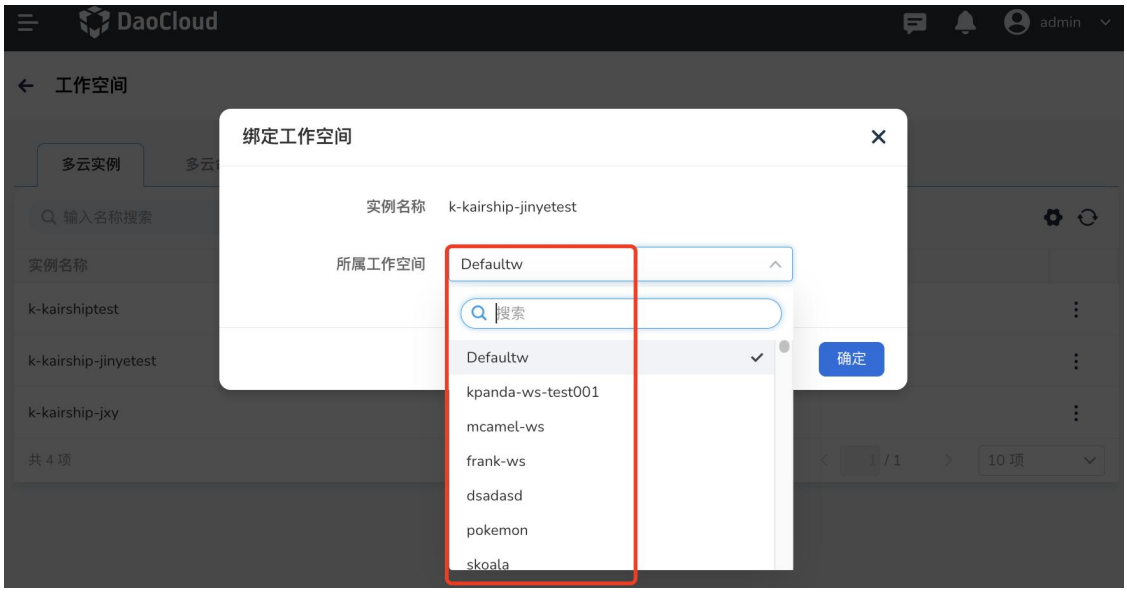
2. 点击多云实例或多云命名空间页签，查看所有多云实例和命名空间当前的绑定状态。

3. 在目标实例/命名空间的最右侧点击更多操作按钮，选择绑定工作空间/解绑工作空间。



管理界面

4. 绑定时选择需要将该实例/命名空间和哪个工作空间绑定。解绑时确认操作即可。

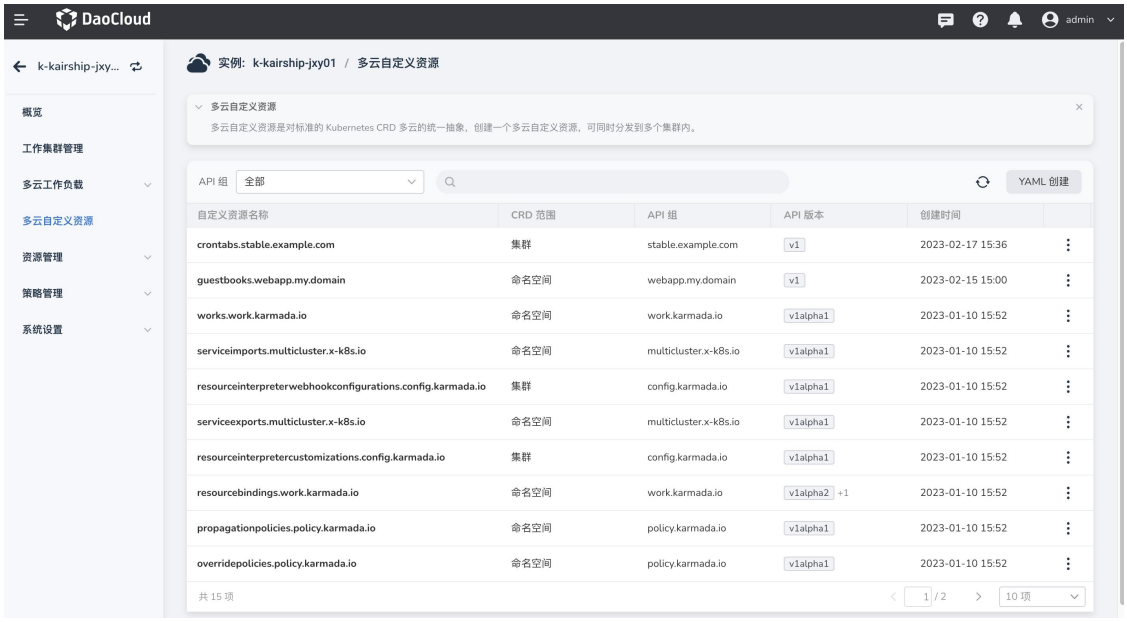

DaoCloud
admin
工作空间
多云实例
输入名称搜索
实例名称
k-kairshiptest
k-kairship-jinyetest
k-kairship-jxy
共 4 项
1 / 1
10 项
绑定工作空间
实例名称
k-kairship-jinyetest
所属工作空间
Defaultw
搜索
Defaultw
kpanda-ws-test001
mcamel-ws
frank-ws
dsadasd
pokemon
skoala
确定


DaoCloud
admin
工作空间
多云实例
输入名称搜索
实例名称
k-kairshiptest
k-kairship-jinyetest
待绑定
k-kairship-jxy
wlp-test
共 4 项
1 / 1
10 项
确认解绑工作空间吗?
确认解除 k-kairshiptest 与工作空间 wlp-test 吗？解绑后工作空间用户将无法访问当前资源，请谨慎操作。
取消
确定

管理多云自定义资源

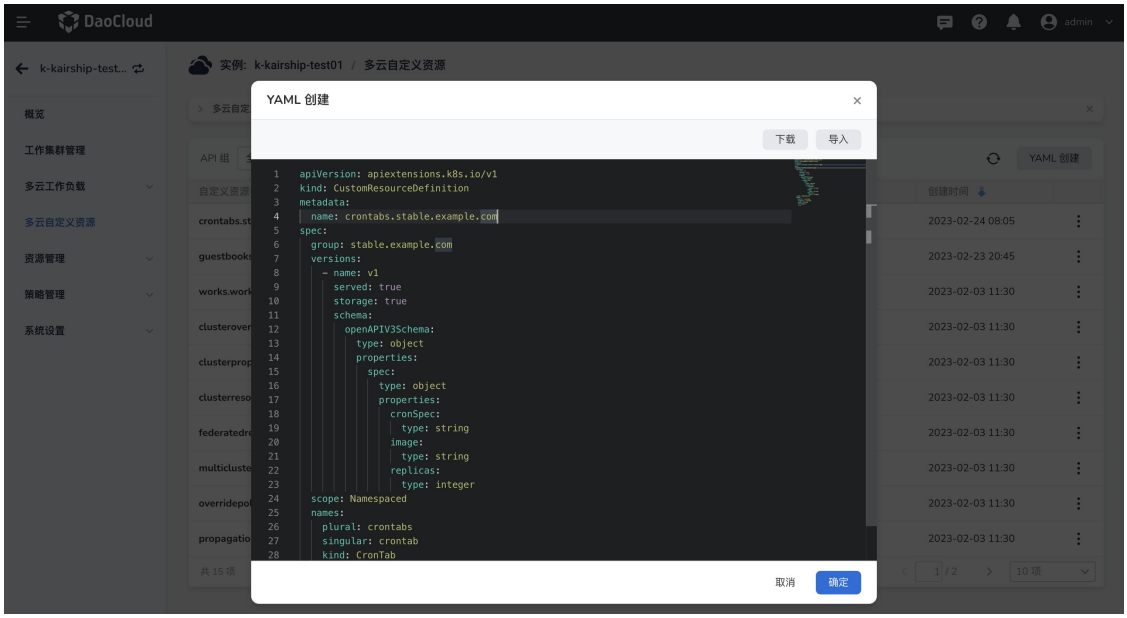
参照以下步骤创建一个多云自定义资源（CRD）。

1. 在左侧导航栏中，点击多云自定义资源，进入自定义资源页面，点击右上角的 YAML 创建按钮。



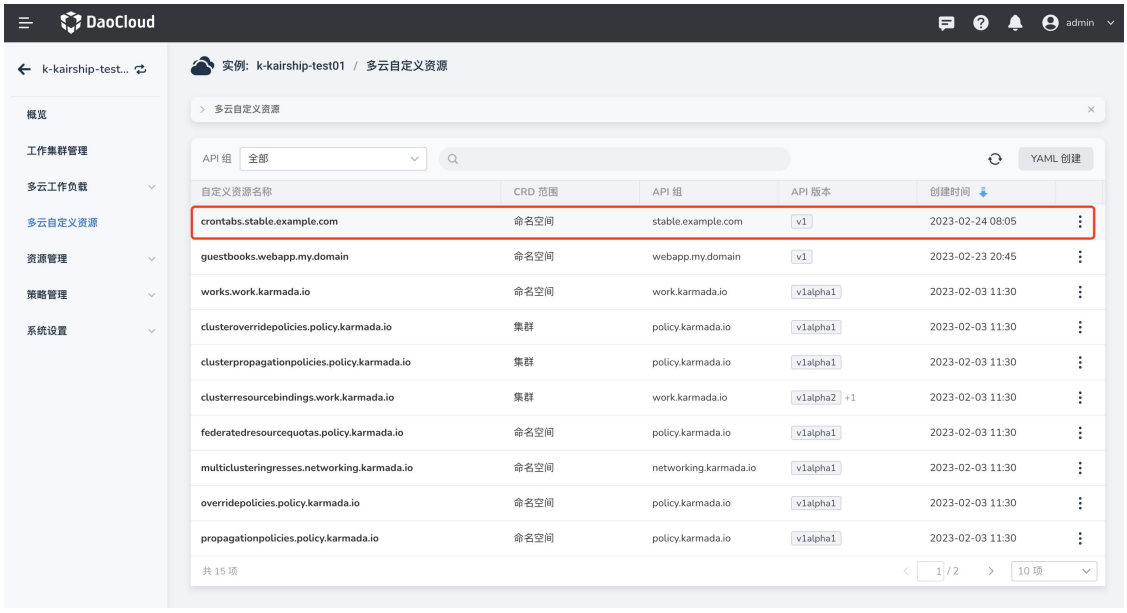
创建 crd

2. 在 YAML 创建页面中，填写 YAML 语句后，点击确定。还支持下载和导入功能。



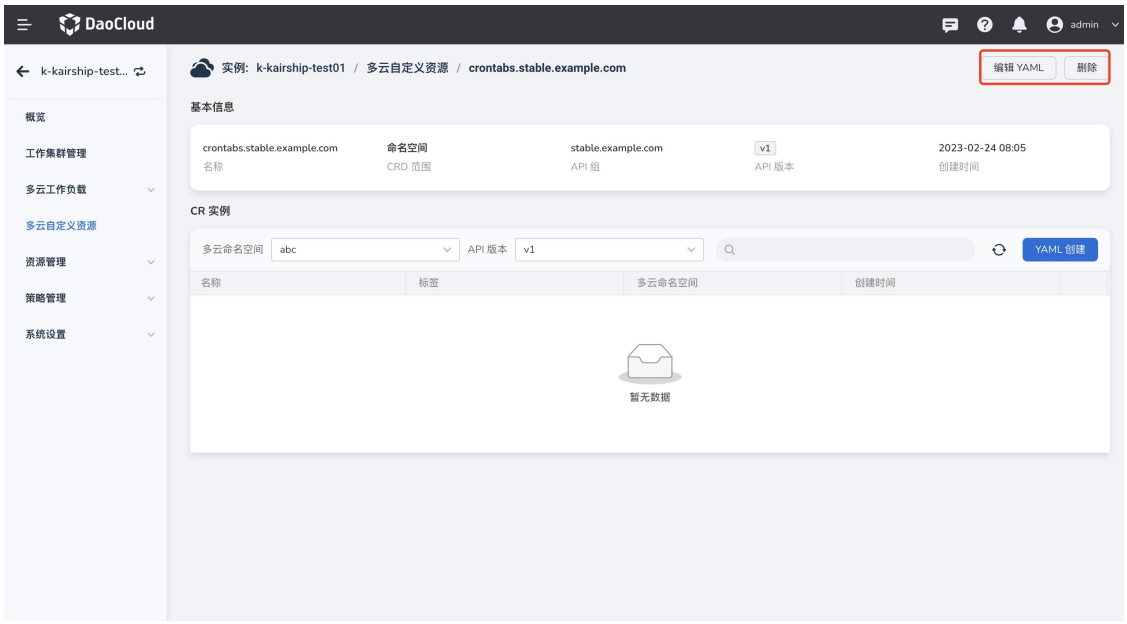
yaml 创建

3. 返回自定义资源列表页，即可查看刚刚创建的名为 crontabs.stable.example.com 的自定义资源。



crd 创建成功

4. 点击名称，进入自定义资源详情页面，在此页面内可以编辑 YAML 信息来更新自定义资源。



crd 详情

自定义资源示例：

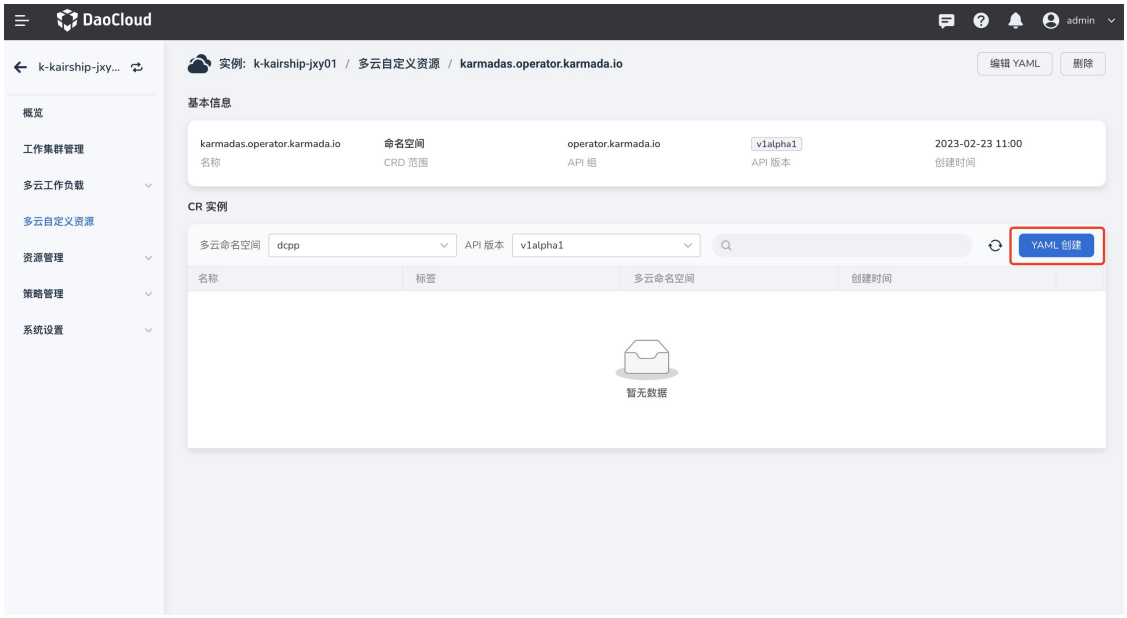
CRD example

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
```

```
            type: object
            properties:
              cronSpec:
                type: string
              image:
                type: string
              replicas:
                type: integer
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
    - ct
```

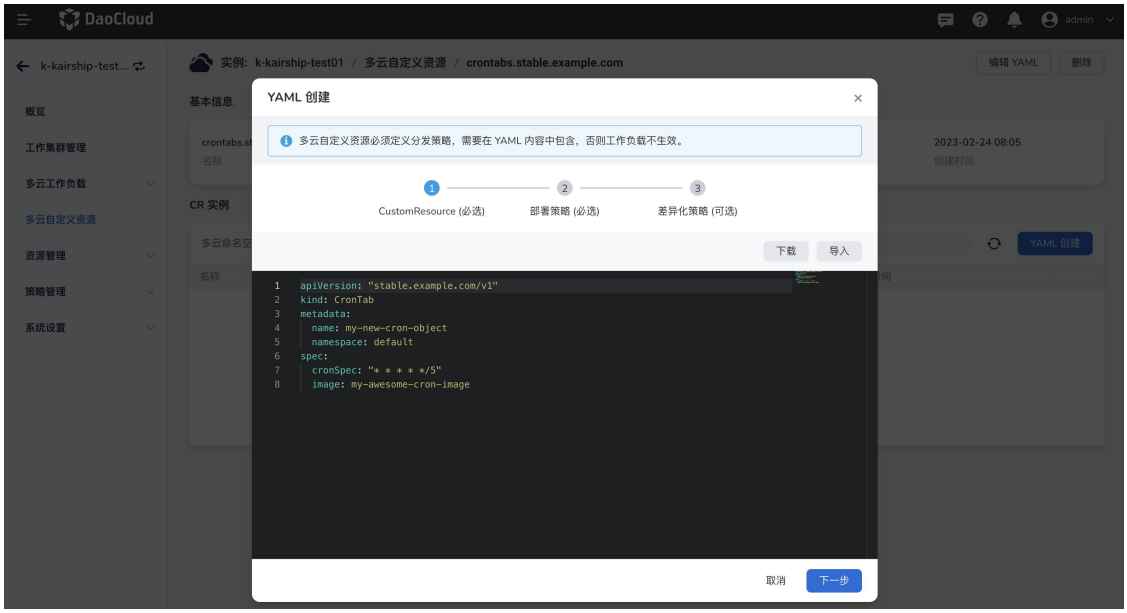
通过 YAML 创建 CR 实例

1. 进入自定义资源详情，点击 CR 列表右侧的 YAML 创建按钮。



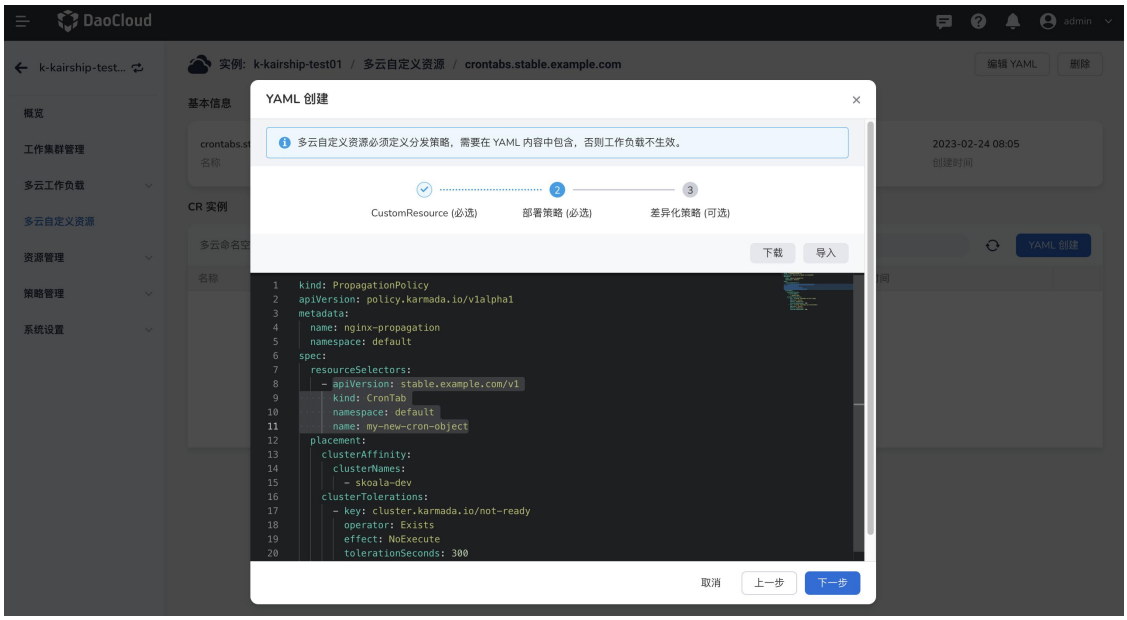
创建 cr

2. 在 YAML 创建页面中，先填写 CR 的 YAML 信息，支持下载和导入功能。



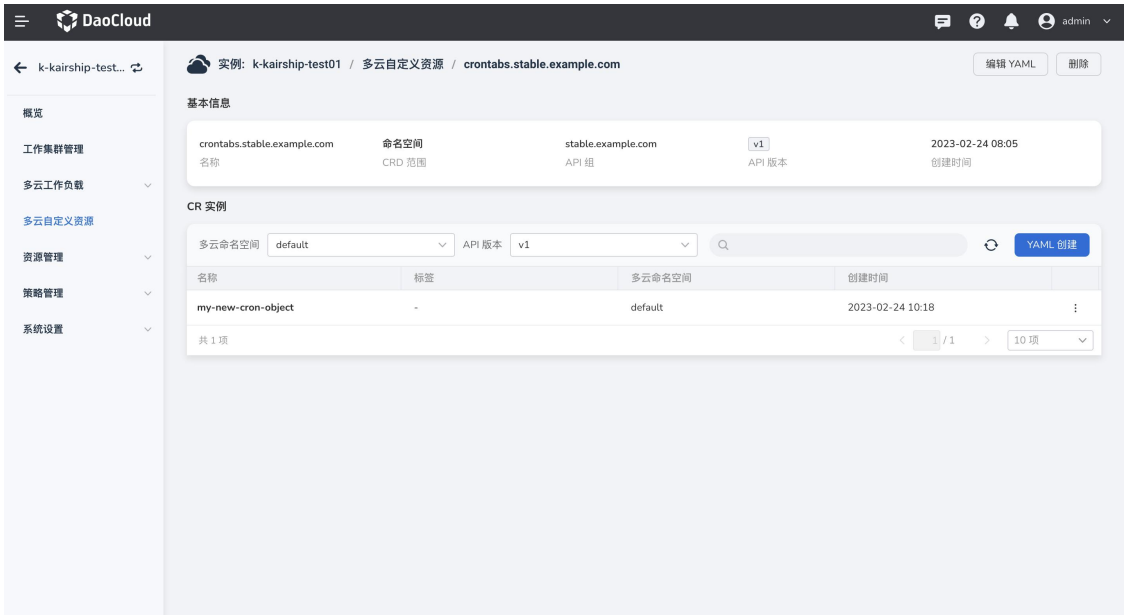
yaml 创建 cr

3. 再填写部署策略，用来指定想要分发在哪些集群。注意需要将部署策略的 YAML 信息根据需要传播的资源的信息进行填写： spec 的 resourceSelector 中 apiVersion、kind、namespace、name 四个参数需要和所需要传播的资源保持一致。 若没有差异化需求，差异化策略可不填。



必填 pp

4. 返回 CR 实例列表页，即可查看刚刚创建的名为 my-new-cron-obiext 的 CR 实例。



创建成功

CR 示例：

CR example

```yaml
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

产品优势

让开发者使用多云像使用一个 Kubernetes 集群，多云编排 Kairship 具有以下优势：

- 兼容 Kubernetes 原生 API
  - 零改造从单集群升级到多集群
- 无缝集成 Kubernetes 单集群工具链生态
  - 告别绑定：多云平台支持，自动分配，自由迁移
  - 不绑定厂商的商业产品
- 丰富的多集群调度
  - 集群亲和性调度，多颗粒多集群高可用部署
  - 多区域（region），多可用区，多集群，多供应商
- 开放中立
  - 来自互联网、金融、制造业、运营商、云厂商等联合发起
  - 开箱即用：面向多场景的内置策略集：两地三中心、同城双活、异地容灾
- 集中式管理
  - 无需顾虑集群位置
  - 支持公有云、私有云、边缘等集群

产品架构

多云编排的管理面主要负责以下功能：

- 多云实例（基于 Karmada）的生命周期管理（LCM）
- 作为多云产品统一的流量入口（OpenAPI、Kairship UI、内部模块 GRPC 调用）
- 代理多云实例的 API 请求（Karmada 原生风格）
- 多云实例内的集群信息（监控、管理、控制）等的聚合
- 多云工作负载等资源的管理和监控
- 后续可能的权限操作

核心组件

多云编排主要包括两个核心组件：kairship apiserver 和 kairship controller-manager。

Kairship apiserver

kairship apiserver 主要担负着多云编排所有流量的入口（OpenAPI、GRPC 等），也是所有 API 的统一入口。protobuf API 优先级最高，通过 proto 定义所有的 API 接口，并以此生成对应的前后端代码，使用 grpw-gateway 同时支持 HTTP Restful 和 GRPC。

启动的时候会从全局管理模块获取操作人的身份信息，用于后续 AuthZ 的安全性校验。

kairship controller-manager

这是多云编排控制器，主要负责实例状态同步、资源搜集、Karmada 实例注册、全局资源注册等。

在多副本部署情况下，通过 leader 机制选举，保持同一个时刻只有一个工作的 Pod（参考 Kubernetes 的 controller-manager 选举机制）。

该组件主要负责多云编排一系列控制逻辑的处理（每个逻辑单独成 controller），通过 list-watch 机制监听特定对象的变更，然后处理对应事件。主要包括：

- virtual-cluster-sync-controller

  多云编排实例 CRD 的 CRUD 事件监听，一旦创建多云编排实例，则同步创建对应的虚拟集群管理资源。

  多云编排实例所有资源的检索（多云工作负载、pp、op）都将通过容器管理模块内部的加速机制完成（借助 Clusterpedia），实现读写分离，进而提高性能。

  实例删除，则同步删除注册在容器管理模块中的 virtual cluster。

- resource statistics controller

  主要搜集多云编排实例中加入的所有集群的统计信息，并将其回写到多云编排实例 CRD 中（例如该实例所管理的集群中总共包含多少 CPU、内存、节点数）。

- status sync controller

  多云编排实例本身的状态同步、统计。

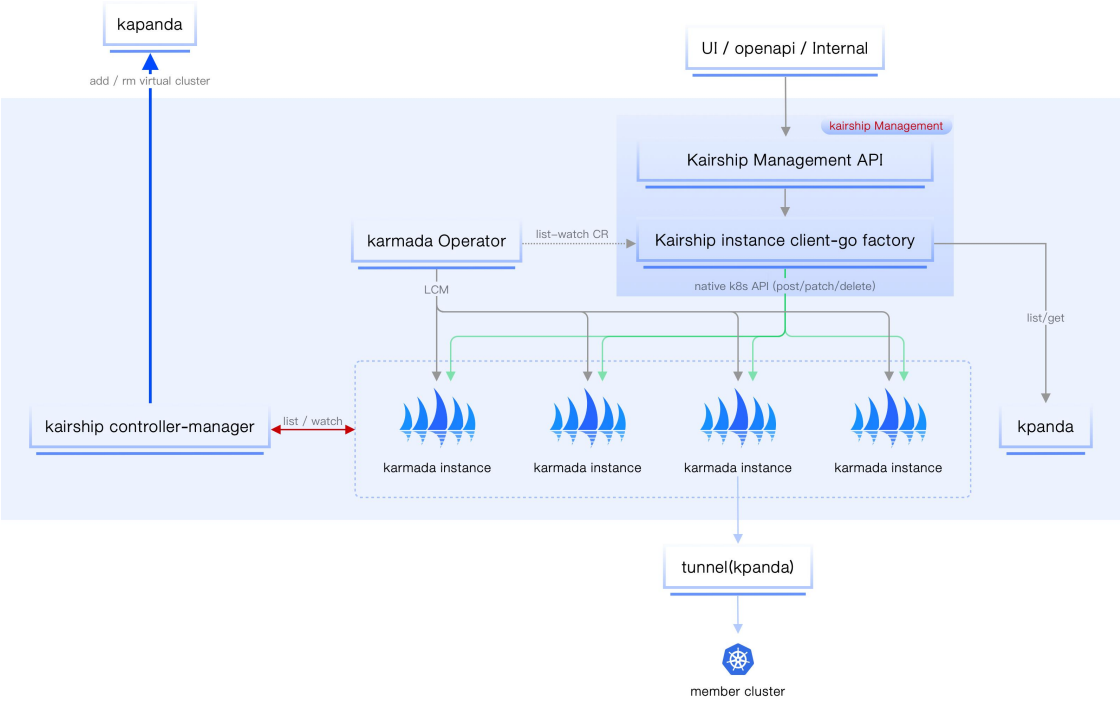
- instance registry controller

  多云编排需要通过自定义资源将平台内所有 Karmada 实例注册到全局管理模块，这样才能在全局管理中完成角色与 Karmada 实例的绑定关系。 最终这些绑定关系会同步到多云编排模块中。

- Ghippo webhook controller

在全局管理模块完成角色与 Karmada 实例的绑定关系之后，通过 sdk 告知多云编排，多云编排据此完成鉴权动作。

数据流图



数据流图

首先需要说明的是，多云实例之间互不感知、相互隔离。

多云编排管理：

- 获取 Karmada 相关的分发策略以及应用的状态信息。
- 获取多云编排实例内的集群、节点的统计、监控信息。
- 编辑、更新、删除相关 Karmada 实例中的多云应用相关的信息（主要围绕 Karmada 工作负载和 pp、op 两个 CRD）。

所有的请求数据流都直接传递到位于全局服务集群的多云编排实例中。

接着，所有访问请求经过多云编排之后将会被分流到对应的实例中。所有 get/list 之类的读请求将会访问容器管理模块，写请求会访问 Karmada 实例，这样可以做到读写分离，加快响应时间。

您可能好奇容器管理模块如何获取多云编排实例的相关资源信息？方法就是，把实例本身作为一个虚拟集群加入到容器管理模块中（不在容器管理中显示）。这样就可以完全借助容器管理模块的能力（搜集加速检索各个 Kubernetes 集群的资源、CRD 等），当在界面中查询某个多云编排实例的资源（Deployment、部署策略、差异化策略等）就可以直接通过容器管理模块进行检索。

多云编排 Release Notes

本页列出多云编排的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.11

新功能

- **新增** 支持在资源详情中查看事件信息

优化

- **优化** 优化集群社区版 karmada operator 项目

2023-06-30

v0.10.3

新功能

- **新增** Service 支持 LoadBalancer 类型
- **新增** 资源新增分发状态展示
- **新增** 多云审计日志

修复

- **修复** 修复审计日志越界
- **修复** 多云实例详情加入工作集群页面出现重复 icon
- **修复** 创建多云工作负载的时候，修改实例数，创建报错
- **修复** 多云资源分发状态，icon 样式问题

优化

- **优化** 部署策略，排除集群下拉数据源优化

2023-05-29

v0.9.1

新功能

- **新增** Service 支持标签、注解表单编辑功能

修复

- **修复** 资源详情页，优化预期调度的集群展示结果
- **修复** kairship-proxy 偶尔不工作的问题
- **修复** cffssl 和 kubectl arm64 版本不存在的问题

优化

- **优化** Ingress 列表，新增集群列
- **优化** 一键转换成员集群 deployment 的相关依赖时的传播策略过多的问题

升级

- **升级** @dao-style/extend 至 v1.2.1
- **升级** CloudTTY API 到 v0.5.2

2023-04-27

v0.8.0

新功能

- **新增** 支持集群调度组
- **新增** 支持镜像选择器选择
- **新增** 高级设置，新增定时重新调度
- **新增** 高级设置，新增集群健康状态阈值设置

修复

- **修复** 部署策略选择 CR 资源类型时，不自动填充 kind
- **修复** 实例删除弹窗，是否同步删除 karmada 实例提示信息调整

移除

- **移除** istio 对 controller-manager 和 proxy 的 sidecar 注入

2023-03-29

v0.7.4

新功能

- **新增** 支持自定义管理集群
- **新增** 负载表单支持数据存储
- **新增** 一键升级负载时支持携带 Service
- **新增** 多云路由支持差异化配置 IngressClass
- **新增** 支持集群层面的部署策略、差异化策略
- **新增** 支持配置 LabelsOverrider、AnnotationsOverrider
- **新增** 自定义角色权限

优化

- **优化** 实例表单，支持隐藏'kairship.io/'为前缀的注解
- **优化** 负载表单，差异化配置引导用户选择集群

修复

- **修复** 修复一键传播 service 失败的问题
- **修复** 修复网格跳转链接样式
- **修复** 前端版本不正确
- **修复** 一键升级工作负载自动传播依赖的 service
- **修复** 过滤子集群 LB 类型的 service 进行一键升级
- **修复** 升级 karmada 的版本到 v1.5.0

2023-02-27

v0.6.3

新功能

- **新增** 添加多云自定义资源模块，支持增删改查、分发功能
- **新增** 添加多云路由模块，支持增删改查、分发功能

优化

- **优化** 负载表单支持自动传播字段

- **优化** 多云 Service 支持指定部署集群功能
- **优化** 负载详情，支持展示预期调度的集群

修复

- **修复** 修复一键升级多云资源不能展示 member 集群的问题
- **修复** cronjob 计算总任务和当前任务数量不正确的问题
- **修复** 生成 pp 资源名称问题
- **修复** Cronjob 无法通过 yaml 更新
- **修复** 自定义资源详情页，部署策略使用状态显示问题
- **修复** 离线包中没有 kairship-ui 的镜像

2022-12-25

v0.5

新功能

- **新增** 添加 cronjob 的增删改查等相关接口
- **新增** 添加 job 的增删改查等相关接口
- **新增** 新增单集群应用一键迁移多集群应用，自动升级依赖资源
- **新增** 给 ListPropagationPolicies 和 ListInstanceOverridePolicies 接口增加 job 和 cronjob 类型
- **新增** ETCD 高可用
- **新增** 部署策略新增 priority 字段
- **新增** 差异化策略新支持 imageOverride, CommandOverrider ArgsOverrider LabelsOverrider AnnotationsOverrider
- **新增** 如果部署策略已经被工作负载使用和关联，部署策略不支持删除
- **新增** 多云工作负载，新增支持 Job、CronJob
- **新增** 差异化策略支持表单化创建、更新
- **新增** 部署策略支持表单化创建、更新
- **新增** 工作集群支持显示驱逐状态

优化

- **优化** 部署策略表单，传播约束新增启用、关闭开关
- **升级** karmada-operator 的版本到 v0.1.9，解决多个实例 etcd 的 pod 反亲和的问题

修复

- **修复** 一个未关联任何角色的用户可以查看所有的实例信息
- **修复** 调度算法为 Duplicated 时，工作负载实例总数统计错误
- **修复** 实例删除时，ghippo 中的数据没有删除
- **修复** 实例删除时，工作集群的 labels 没有被移除
- **修复** 移除集群时，可以在单集群应用多云化界面中看到正在移除的集群，并能添加正在移除的集群
- **修复** 修复在移除 member 集群的过程中，不能对 member 集群的资源进行一键升级
- **修复** 不健康的成员集群无法移除

2022-11-25

v0.4

新功能

- **新增** prometheus 指标 metrics、opentelemetry 链路 trace
- **新增** 创建工作负载指定地域后显示对应集群列表
- **新增** 创建工作负载指定标签后显示对应集群列表
- **新增** failover 故障转移产品化

修复

- **修复** estimator 没有适配离线安装
- **修复** 实例详情页面无状态负载显示异常的问题

2022-10-21

v0.3

新功能

- **新增** 多云编排开启权限校验
- **新增** 多云编排 list instance 接口，根据权限展示数据
- **新增** 多云编排根据用户权限查询 cluster 资源概览信息
- **新增** 多云编排查询所有 member 集群的 labels
- **新增** 多云编排单机集群应用一键转换为多集群应用
- **新增** 多云编排查询 member 集群的 namespace 和 deployment 资源
- **新增** 多云资源创建的提示语

优化

- **优化** 多云编排优化 karmada PropagationPolicy 和 OverridePolicy 的 protobuf 数据结构

修复

- **修复** 多云编排修复实例下所有 PropagationPolicy 资源的排序不生效的问题
- **修复** 多云编排修复移除 member 集群的问题
- **修复** 若干 bug 修复

2022-9-25

v0.2

新功能

- **新增** 对调度时间查询接口
- **新增** 多云服务 ConfigMap 管理接口
- **新增** 批量创建多个资源和 policy 资源
- **新增** Service 增加工作负载的标签
- **新增** 获取所有命名空间下 Service 的接口
- **新增** 注入 istio sidecar

- **新增** 接入集群时，部署 karmada estimator
- **新增** 多云 secret 接口
- **新增** 新增 instance 的 cpu 和 memery 的资源数据收集
- **新增** 新增 instance 的 event 查询 API

2022-8-21

v0.1

新功能

- **新增** 新增 cloudshell 的 API 通过 cloudshell 可以管理 karmada 集群
- **新增** 对多云命名空间的管理接口
- **新增** 多云 service 管理接口
- **新增** 多云工作负载详情相关接口
- **新增** 支持设置集群的污点与容忍
- **新增** 下载 karmada 实例 kubeconfig 接口
- **新增** 提供 instance 的 update API 支持修改 instance 别名和 label

优化

- **优化** 优化 instance 的 API 并收集 karmada 实例的资源统计情况

基本概念

本节的词汇表定义了多云编排背后的核心概念，以帮助您构建多云编排工作原理的思维模型。

- 资源模板：多云编排软件将 Kubernetes Native API 定义用于联合的资源模板，可以轻松集成已被 Kubernetes 采用的现有工具。
- 部署策略：多云编排软件提供了一个独立的传播（置位）策略 API 来定义多集群调度和传播需求。
  - 映射与工作负载支持 1:n。用户无需在每次创建联合应用时都指定调度约束。
  - 使用默认策略，用户可以直接与 Kubernetes API 交互。
- 差异化策略：多云编排软件提供了一个独立的 Override Policy API，用于专门自动化集群相关的配置。
  - 根据成员集群区域覆盖镜像前缀。
  - 根据采用的云提供商来覆盖 StorageClass。

常见问题

1. Karmada 的版本是多少？能否指定版本？是否可以升级？

   当前默认版本 v1.4.0，支持用户自主升级。

2. 单集群应用如何无缝迁移到多云编排？

   可以使用我们的新功能 一键升级为多云工作负载。

3. 是否支持跨集群的应用日志收集？

   暂不支持，之后会增加该功能。

4. 分发到多个集群的工作负载，是否可以在一个视图呈现监控信息？

   支持在统一的视图来查看多云应用，可以监控他们部署到了什么集群，对应的 Service、传播策略等等。

5. 工作负载是否可以跨集群通信？

   支持，参阅多云网络互联。

6. Service 能否实现跨集群服务发现？

   支持，参阅多云网络互联。

7. Karmada 是否有生产级别支持？

   目前还处于 TP 阶段，很多内部组件高可用有待解决（Karmada 依赖的 etcd 等等）。

8. 如何做到故障转移？

   Karmada 原生支持故障转移的功能，在成员集群出现故障的时候，Karmada 会进行智能的重调度，完成故障转移。可参阅故障转移介绍。

9. 多集群的权限问题

   紧密的结合 5.0 现有的权限体系，与 workspace 打通，完成 Karmada 实例与 workspaces 的绑定，解决权限问题。

10. 如何查询多集群的事件?

    多云编排完成了产品级别的整合，会展示所有 Karmada 实例级别的事件。

11. 通过多云编排创建一个多云应用之后，通过容器管理怎么能获取的相关资源信息?

    了解 Karmada 的小伙伴都知道，Karmada control-plane 其本质也就是一个完整 kubernetes 控制面，只是没有任何承载 workload 的节点。因此多云编排在创建多云编排实例的时候，采用了一个取巧的动作，会把实例本身作为一个隐藏的 cluster 加入到容器管理中(不在容器管理中显示)。这样就可以完全借助容器管理的能力(搜集加速检索各个 K8s 集群的资源，CRD 等)，当在界面中查询某个多云编排实例的资源（Deployment、PropagationPolicy、OverridePolicy 等） 就可以直接通过容器管理进行检索，做到读写分离，加快响应时间。

12. 如何自定义 karmada 镜像来源仓库地址?

    Kairship 采用开源的 karmada-operator 进行多实例 LCM 管理；Operator 提供了丰富的自定义能力。支持在启动参数中自定义 karmada 资源镜像的仓库地址。

    可以在容器启动命令中增加 --chat-repo-url 参数进行指定



image

13. 如何连接 Karmada 集群?

    可以在实例概览页面右上角控制台进入，连接 karmada 的控制平面。

14. 是否可以仅删除多云实例，但是不删除 karmada 的组件?

    可以，创建多云实例时，可以选择选择是否勾选实例释放功能。若勾选将同步删除对应的 Karmada 实例；如果不删除，则可以继续通过终端使用，但无法在 多云编排 内管理该 Karmada 实例，建议同步删除。

15. 多云实例内多个工作集群如何实现网络互通?

    需要在服务网格创建网格实例，并且纳管每个工作集群，具体操作可参考多云网络互联。

什么是多云编排

国内的云计算已经发展了近 15 年，随着技术的成熟，企业应用不单单是简单的上云，出现了更多更复杂的多云需求。云计算从单云到多云的发展历程简单示意如下。



单云到多云

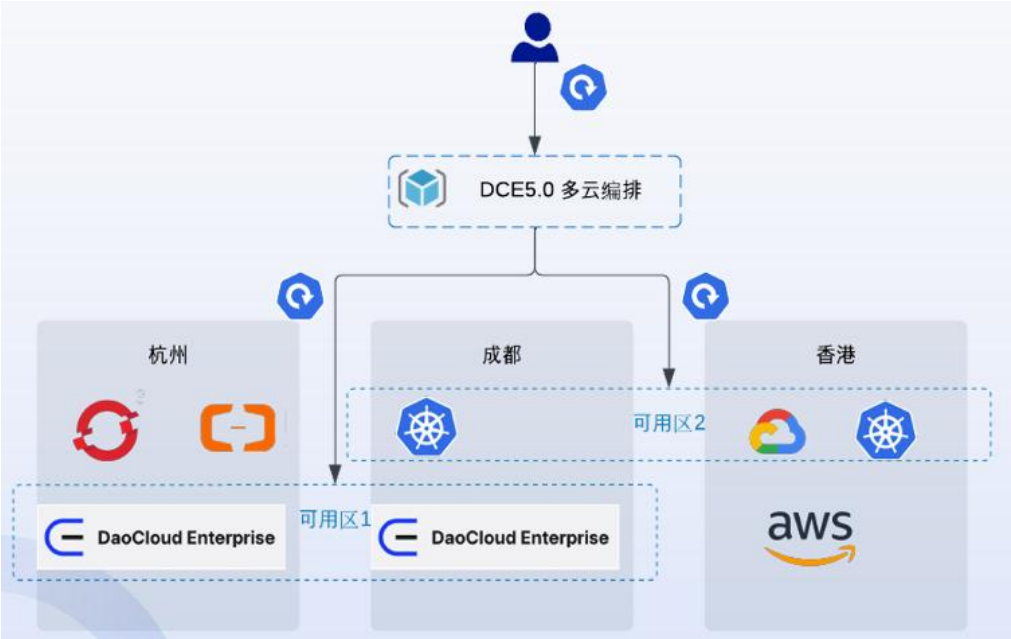
企业从最初简单的只要上云就好，到如今开始选择到底上哪个云，或者上哪些云，从而构建专属于企业自身的多云环境。

经数据统计表明，目前企业选择上云时的多云配比情况如下：

上云方案	市场占比
多个公有云 + 多个私有云	43%
多个公有云 + 一个私有云	29%
一个公有云 + 多个私有云	12%
一个公有云 + 一个私有云	9%
一个公有云/私有云	7%

从上表可以看出，多云是趋势，也是市场主流。这也是多云编排之所以诞生的原因。

多云编排 Kairship（Kubernetes Airship）是一个以应用为中心、开箱即用的多云应用编排平台。 多云编排实现了多云和混合云的集中管理，提供跨云的应用部署、发布和运维的能力；支持基于集群资源的应用弹性扩缩，实现全局负载均衡；提供了故障恢复的能力，彻底解决多云应用灾备的问题。



workload

演示视频：如何将单云应用一键转换为多云应用

下载 DCE 5.0 安装 DCE 5.0

功能总览

多云编排的功能列表如下：

- 统一管理面：多云编排拥有统一的管理面，由该管理面进行多个多云实例的管理，统一请求入口（多云编排实例的 LCM），而所有其他关于多云编排的请求可以部署在 global 集群。
- 多实例：支持创建多个多云实例，实例之间隔离工作，互不感知、互不影响。
- 集群一键导入：支持从现有纳管集群中一键导入集群到多云实例中，并实时同步接入的集群最新信息（删除的同时一并删除）。
- 原生 API 支持：支持所有 Kubernetes 原生 API。
- 多云应用分发：丰富的多云应用分发策略、差异化策略等。
- 应用故障切换：内置提供应用多云故障切换（failover）能力。
- 可观测性：丰富的审计、指标度量，提高可观测性的能力。
- 对接全局管理权限：以工作空间管理用户访问范围，对用户和实例执行鉴权操作。

多云实例管理

- 添加多云实例：支持添加实例时不加入任何集群，以创建空实例，即不包含任何集群的实例。
- 查看多云实例：查看实例列表并支持根据实例名称进行检索

  支持查看当前租户环境内全部的多云实例列表和实例的基础信息、实例内已接入集群的 CPU、内存的使用率情况、实例内已接入集群的状态情况等，支持查看版本信息、实例创建时间和实例状态。

- 一键导入集群：在集群创建完成后，可将已经完成接入容器管理的集群一键导入到多云实例中，支持在接入时查看集群的资源信息，并支持根据集群名称进行检索。
- 移除多云实例：移除多云实例时自动进行校验；为保证数据安全，仅允许当前实例中不包含任何集群时，才能够进行移除多云实例。

实例内集群管理

- 添加集群

  支持动态（热插拔）将新集群加入到当前多云实例中，添加了选取容器管理的集群界面，默认展示当前有权限接入的全部待接入集群，

- 查看集群

  支持查看已接入当前实例的所有集群，以及这些集群的细节信息，包含集群名称、状态、平台、地域、可用区、Kubernetes 版本等。

- 移除集群

  支持动态（热插拔）将集群从当前多云实例中移除。

  移除校验：支持对集群的资源进行校验，告知用户需要事先移除集群内的存在的多个工作负载、命名空间、存储声明、配置与密钥等，提示移除的风险。

- 使用 Kubectl 管理实例资源

  支持获取 kubeconfig 链接信息，用户可在本地终端管理多云实例。

  支持通过网页终端 cloudshell 进行多云实例的管理。

多云工作负载

- 创建多云无状态负载

  镜像创建：界面化创建无状态工作负载，添加差异化配置、分发策略等差异化配置。

差异化设置：支持在创建负载时，进行差异化配置，支持按集群进行差异化配置集群部署 Pod 数/CPU/内存/升级策略，差异化配置集群镜像、部署策略、容器脚本、容器存储、容器日志、调度策略、标签和注解等。

YAML 创建：用户通过 YAML 进行创建工作负载。

YAML 语法校验：支持对用户输入的 YAML 进行语法校验，对错误语法给予提示。

- 多云工作负载详情

  基本信息：支持查看无状态负载的部署详情信息 Pod 数量情况，活跃集群信息。资源负载提供单 Pod 监控查看能力，可跳转到 Insight Pod 日志信息，提供单 Pod 日志查看能力。

  部署信息：查看无状态负载的部署详情信息，并支持多云工作负载的重启/暂停/恢复/释放等操作。

  实例列表：支持按照集群的方式展示工作负载在多个 Kubernetes 集群的 Pod 信息，支持一键快速跳转到对应集群的工作负载详情界面，查看对应 Pod 的监控、日志信息。

  服务列表：支持按照集群的方式切分集群内的工作负载服务信息，支持一键快速跳转到对应集群的工作负载详情界面，查看对应服务的监控、日志信息。

- 更新多云工作负载

  页面编辑：支持通过页面的方式编辑集群

  YAML 编辑：支持通过 YAML 的的方式变更多云工作负载的配置信息

- 删除多云工作负载

  页面删除：支持通过页面的方式进行多云工作负载的删除，同时删除在多集群内的部署

支持 cloudshell 及终端进行删除：支持通过终端进行删除

删除提示：提醒在删除时，需要进行二次确认

资源管理

- 多云命名空间

  多云命名空间管理：支持查看多云命名空间资源列表

  查看多云命名空间列表：提供列表可以查看多集群的命名空间信息

  创建多云命名空间：提供通过界面化创建多云命名空间的能力

  删除多云命名空间：支持对空闲的多云命名空间进行删除

- 多云密钥

  查看多云存储声明列表：支持查看创建的多云 PVC 资源列表

  创建多云存储声明：提供通过界面化和 YAML 多种方式进行存储声明的创建

  删除多云存储声明：支持对空闲存储声明进行删除操作

- 多云配置项

  多云配置项管理：支持查看多云 ConfigMap 资源列表

  多云秘钥管理：支持查看多云 Secret 资源列表

  创建多云配置或多云密钥：支持通过界面化或 YAML 完成对多云配置或多云密钥的创建，并统一显示在用户界面

  删除多云配置或多云密钥：支持对空闲的多云配置或多云秘钥进行删除操作，对于在使用中的配置或秘钥不允许删除

- 多云 Service

  服务列表：支持查看多云 Service 列表

创建服务和路由：支持通过界面化或 YAML 完成对 Service 创建，并统一显示在用户界面

删除服务和路由：支持对 Service 进行删除操作，对于在使用中的给出对应的删除提示

策略中心

- 部署策略

  部署策略列表：支持在界面上查看当前实例的部署策略列表及其关联的多云资源

  创建部署策略：支持以 YAML 的方式维护创建和编辑部署策略信息

  删除部署策略：仅对空闲的部署策略提供删除按钮

- 差异化策略

  差异化策略列表：支持在界面上查看当前实例的部署策略列表及其关联的多云资源

  创建差异化策略：支持以 YAML 的方式维护创建和编辑差异化策略信息

  删除差异化策略：支持在界面上查看当前实例的差异化策略列表以及当前差异化策略关联的多云资源

添加实例

在开始使用多云编排模块提供的各种功能之前，必须先添加一个多云实例，通过该实例才能管理各种多云资源。

!!! note

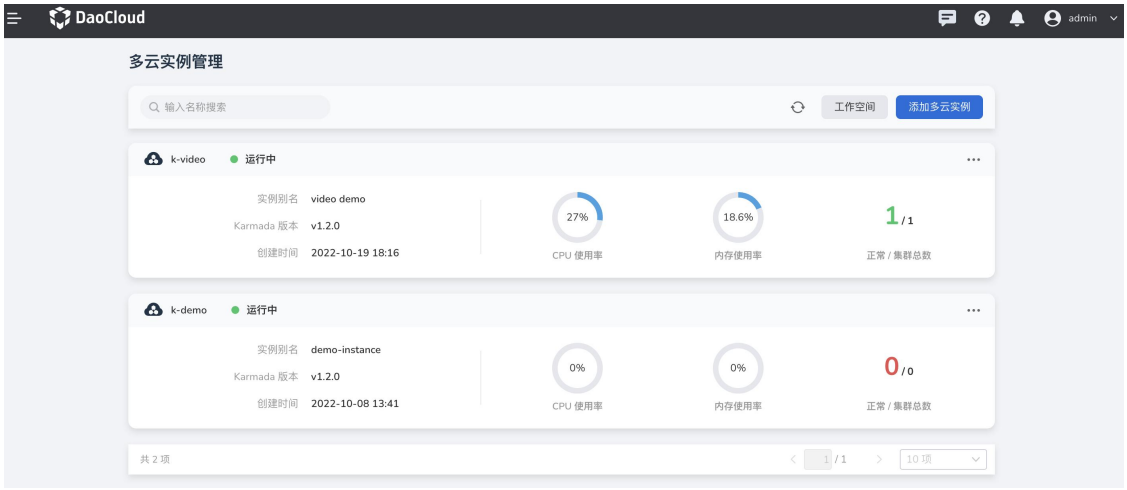
– 由于 DCE 5.0 多云编排模块借用了开源项目 [Karmada](https://karmada.io/) 的能力，所以创建多云实例时会自动创建一个 Karmada 实例。

– 新建的多云实例默认位于[全局管理集群](../../kpanda/user-guide/clusters/cluster-role.md)，以虚拟集群的形式存在，用户无感知。

– 为便于和真实集群进行区分，所有多云实例对应的虚拟集群的名称都带有 `k-` 前缀。

添加多云实例的步骤如下：

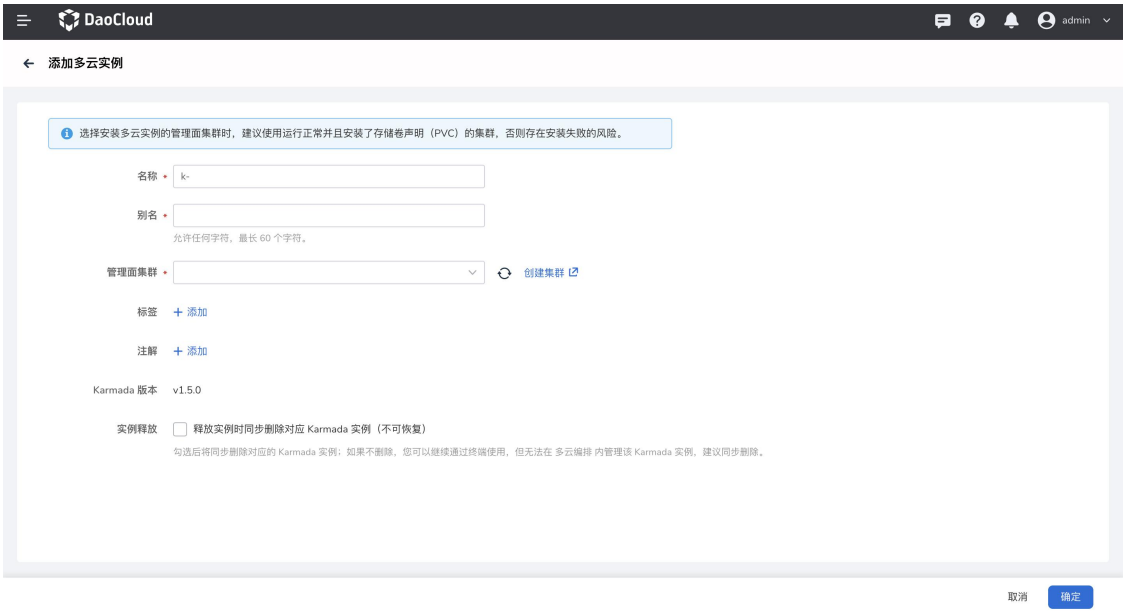
1. 在多云实例列表中，点击右上角的添加多云实例。



add

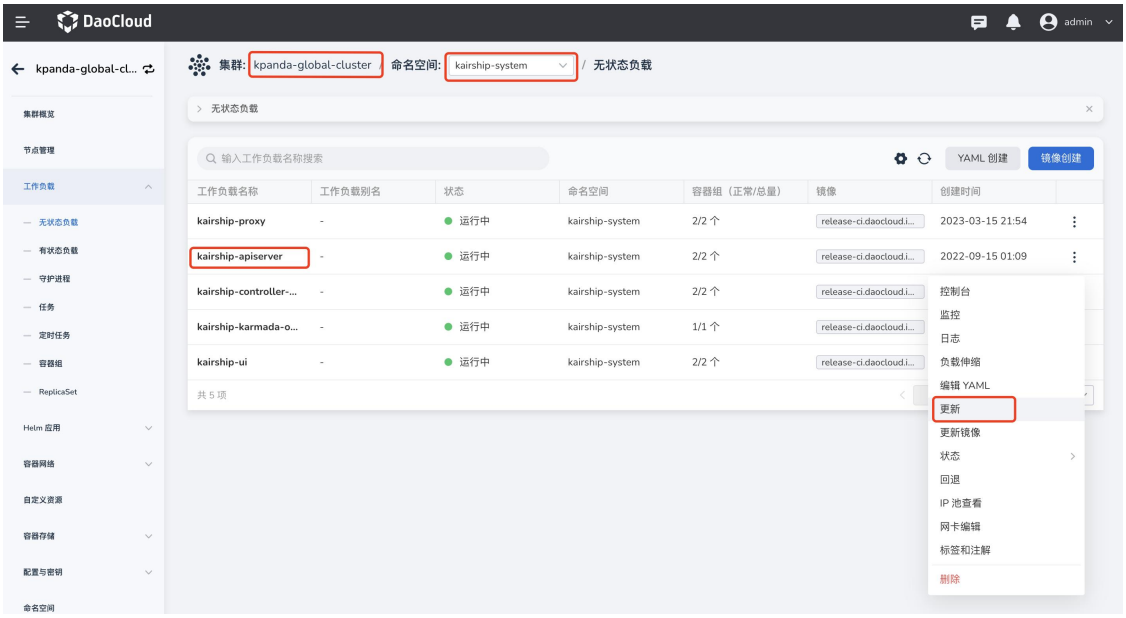
2. 参考下列信息填写各项配置，最后在右下角点击确定。

- 管理面集群：可选的集群列表来自于容器管理模块中接入或创建的集群。如果没有想选的集群，可以去容器管理模块中接入或创建集群。
- 实例释放：如果勾选，删除多云实例时会同步删除对应的 Karmada 实例。如果不删除，可以继续通过终端使用，但无法在 DCE 5.0 的多云编排模块内管理该 Karmada 实例。建议同步删除。



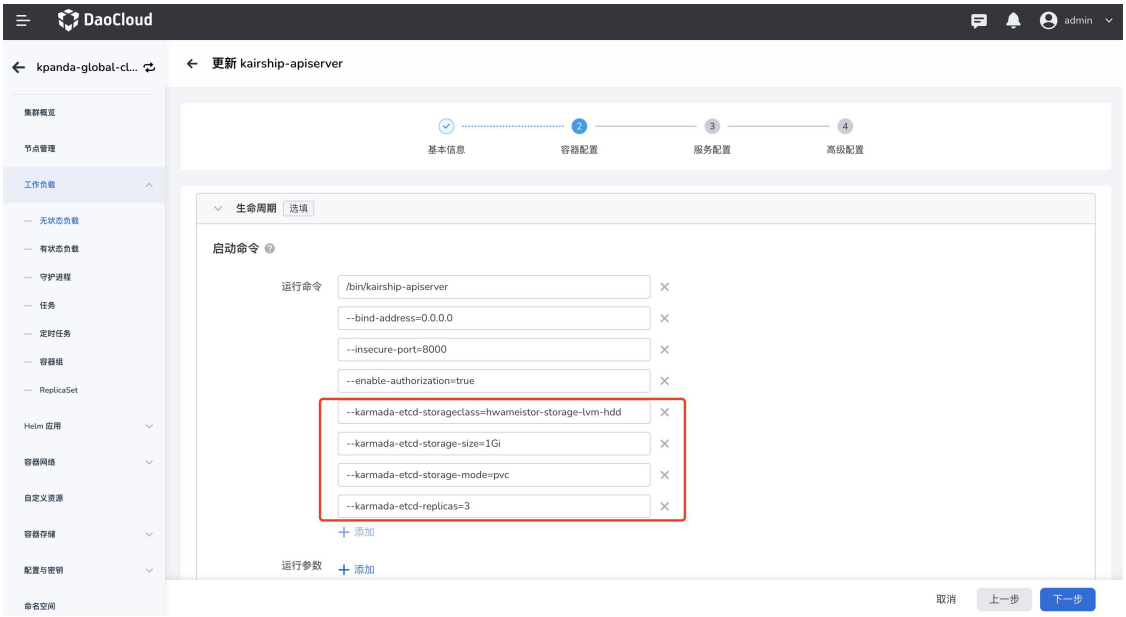
Management

3. 选择安装多云实例的管理面集群时，建议使用运行正常并且安装了存储卷声明（PVC）的集群，否则存在安装失败的风险。若安装失败，可以在对应的管理面集群中查看原因。
   - 在管理面集群详情页面中，选择 kairship-system 命名空间下的 kairship-apiserver 无状态负载，并进行更新操作。



更新无状态负载

- 进入更新页面后，修改容器配置–生命周期的启动命令，可根据管理面集群的真实情况更新。



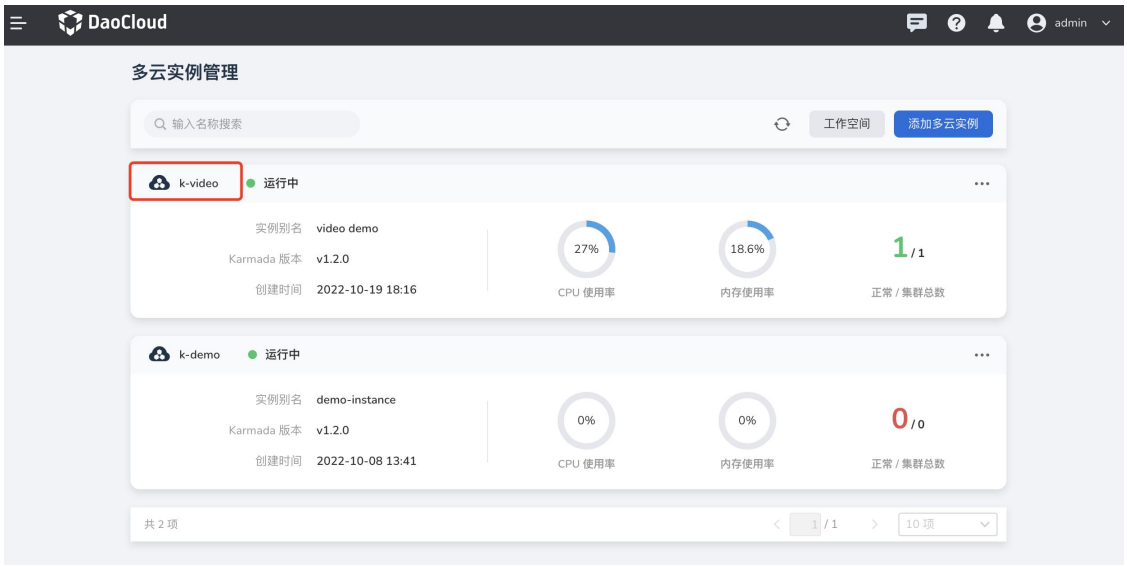
更新启动命令

- 如果管理面集群中不存在存储卷声明（PVC），而是使用 hostPath，尽管可以成功安装，但无法实现高可用功能。

查看多云实例

1. 在多云实例列表中，点击某个实例的名称，可以进入概览页面。

   支持在搜索栏输入实例名称快速检索，搜索支持模糊匹配。



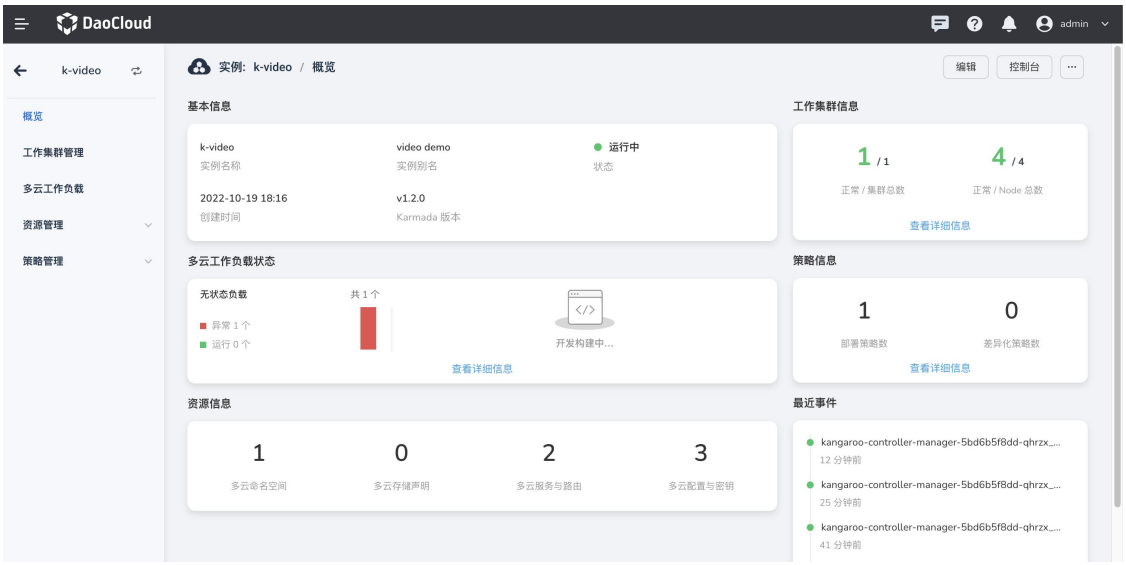
   instance

2. 实例信息展示

   概览页面显示了当前实例的基本信息、多云工作负载状态、资源信息和工作集群等信息。具体包括：

   - 实例名称/别名
   - 实例创建时间
   - 当前实例的 Karmada 版本
   - 实例状态
   - 实例总体的 CPU 使用率。**数据基于当前实例接入的所有工作集群的资源总额**

- 实例总体的内存使用率
- 当前实例中状态正常的集群数/集群总数



instance

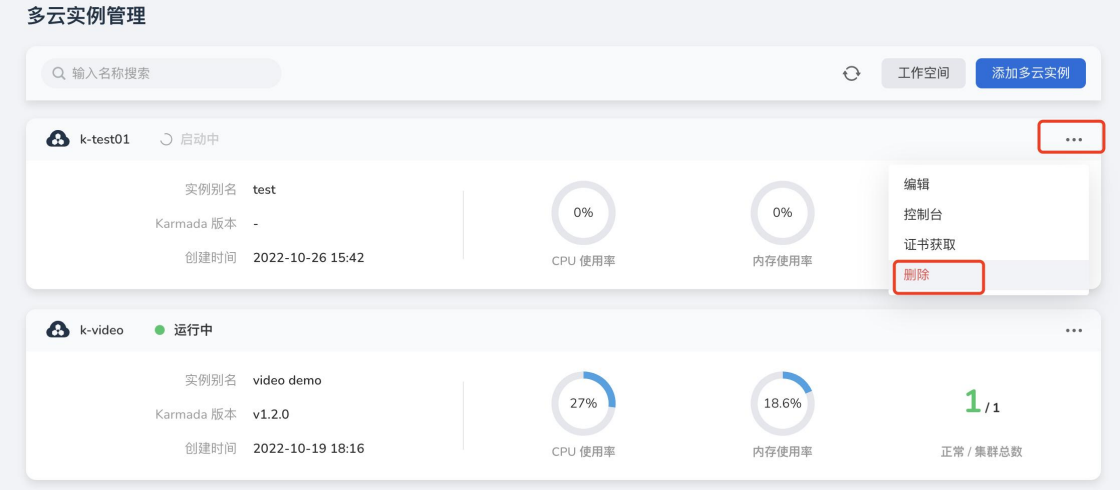
删除实例

如果不再使用某个实例，可以参照以下步骤删除。

!!! note

删除实例之前，需要先删除实例下的集群、工作负载、服务等资源。实例被删除后，该实例相关的所有资源也会被全部删除，请谨慎操作。

1. 在多云实例列表中，点击右侧的　　按钮，在弹出菜单中选择删除。



delete

2. 在弹窗中输入该实例的名称，确认无误后，点击删除按钮。


删除 k-test01 多云实例提醒
您正在操作删除当前多云实例 k-test01 ，您当前有 0 个工作集群，请先移出工作集群；移出所有工作集群后，才可以删除多云实例。
请输入k-test01
k-test01
删除
取消

delete

通过 YAML 创建多云 Deployment

支持通过镜像和 YAML 文件两种方式创建多云无状态负载（Deployment）。您可以根据实际情况选择适合自己的创建方式。

- 镜像创建通过表单填写各项配置，通俗易懂、简单易用，但步骤较为繁琐。
- YAML 创建通过 YAML 文件配置各项信息，操作步骤更少、效率更高，但要求操作者掌握一定的后端技术知识。

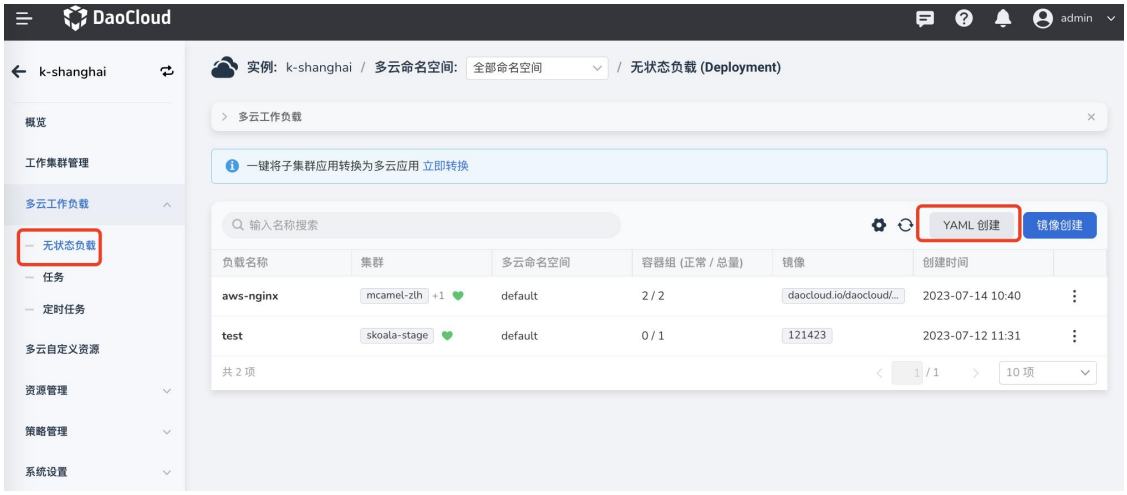
本文介绍如何通过 YAML 创建多云 Deployment。如需了解镜像创建方式，可参考通过镜像创建多云 Deployment。

前提条件

- 创建多云实例
- 在多云实例中添加至少一个工作集群

操作步骤

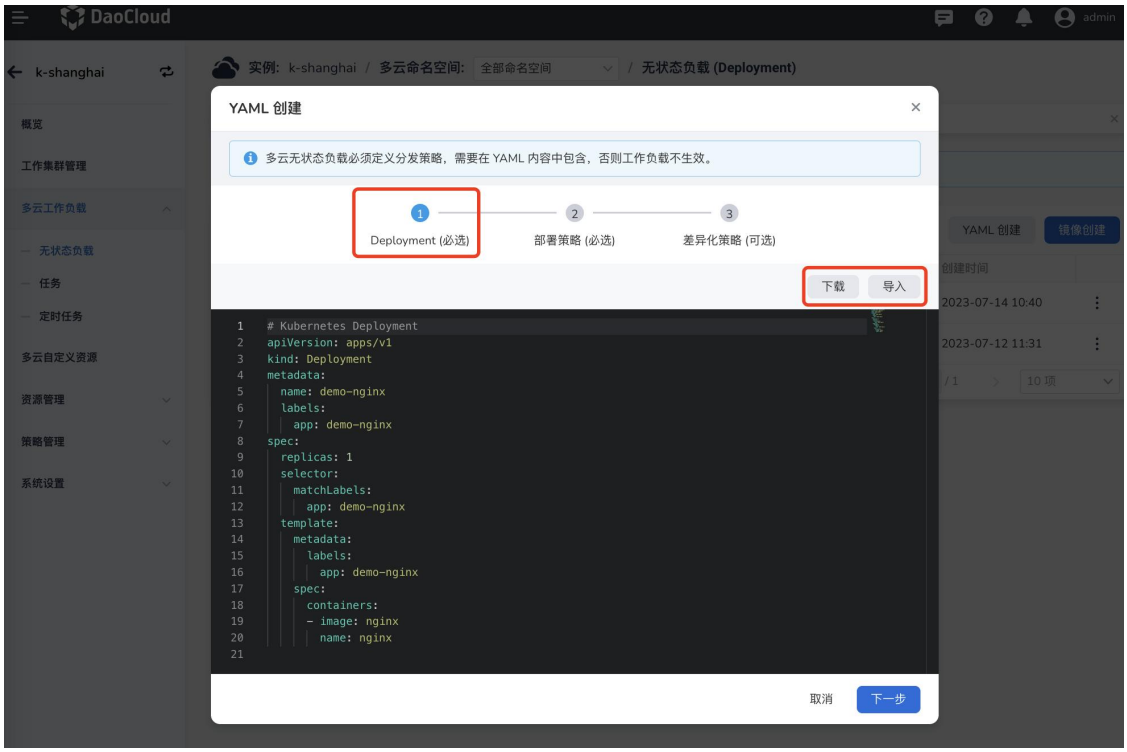
1. 在左侧导航栏中，点击多云工作负载->无状态负载，然后在右上角点击 YAML 创建。



yaml 创建

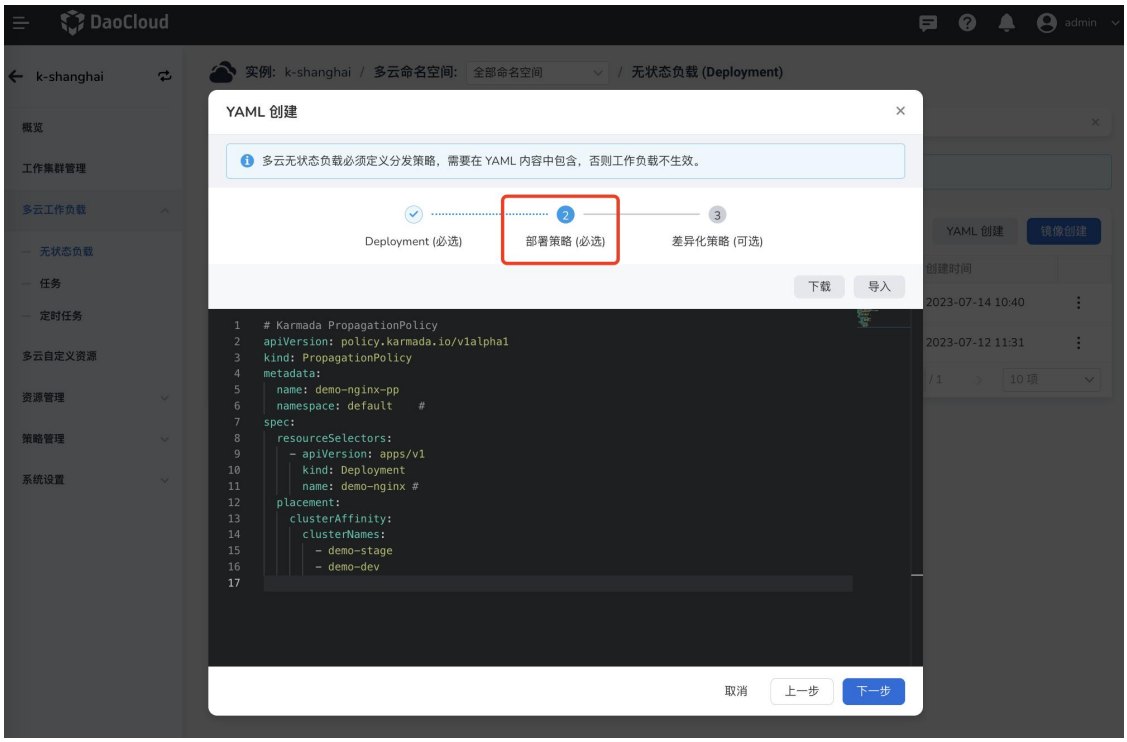
2. 输入或导入 Deployment 的 YAML 文件，然后点击下一步。

   点击下载可以将当前的 YAML 文件下载并保存到本地，便于后续在其他场景中使用。



   输入代码

3. 输入或导入部署策略的 YAML 文件，然后点击下一步。

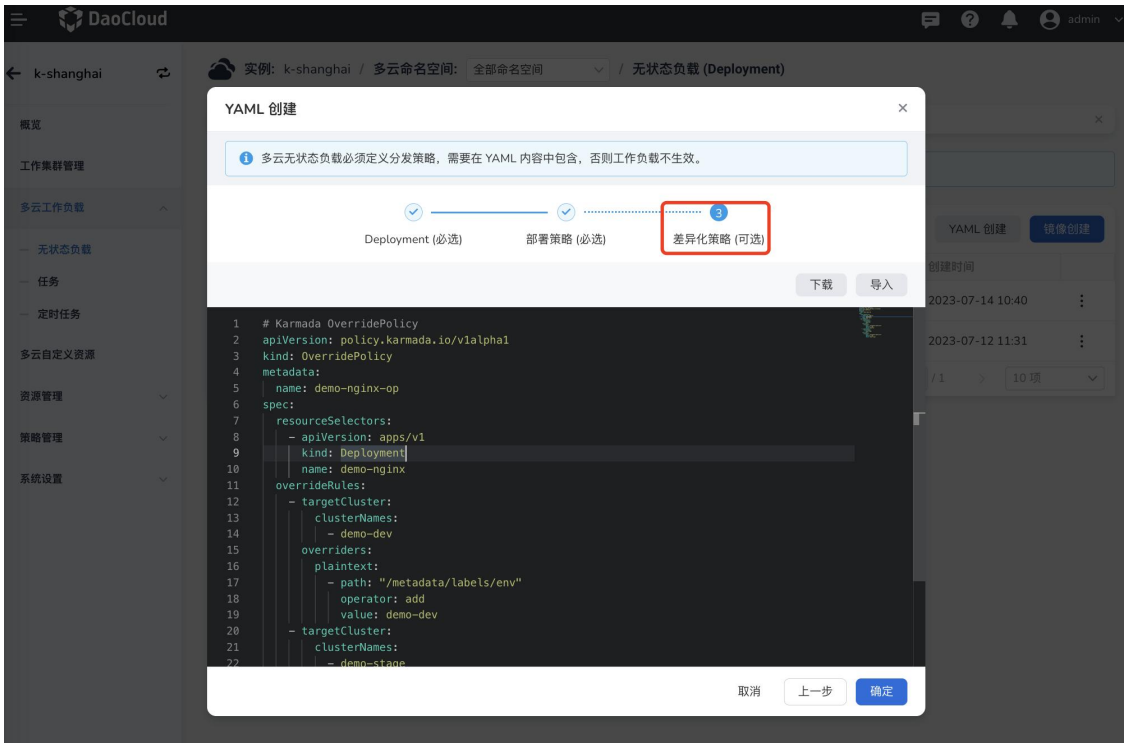


输入 pp

4. 输入或导入差异化策略的 YAML 文件，然后点击确定。

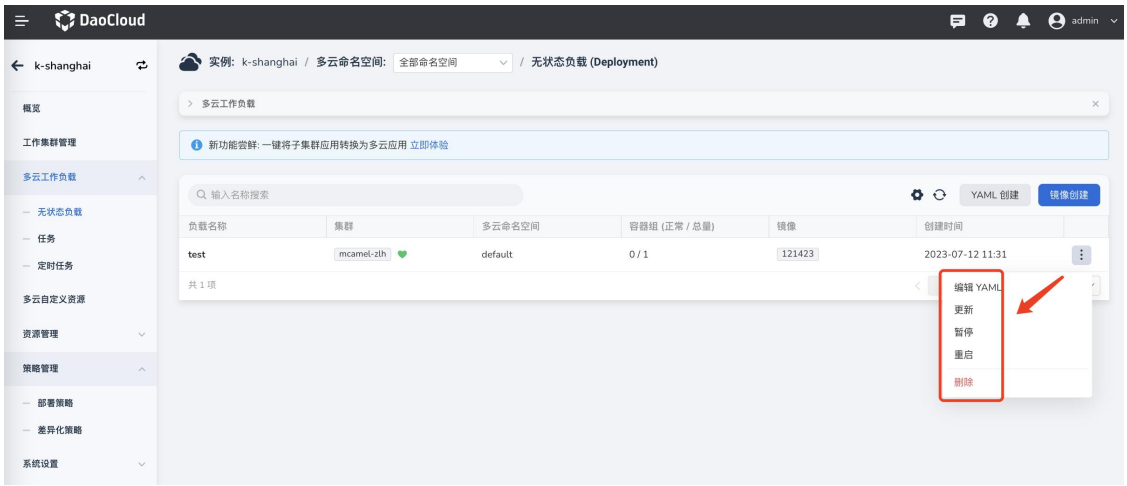
!!! note

差异化策略为可选配置。如无需差异化配置，则不用在此输入任何内容，直接点击`确定`即可。



op

页面会自动返回多云工作负载列表，点击列表右侧的 ⋮，可以编辑修改 YAML，还可以暂停、重启和删除该负载。



more actions

YAML 文件示例

多云编排模块自带的 YAML 编辑器中会自动检测 YAML 语法。如有错误，会用红色波浪线标记错误点。

下面提供一些常用的 YAML 文件示例，您稍加修改就可以使用。

Deployment 示例

```yaml
# Kubernetes Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-nginx
  labels:
    app: demo-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-nginx
  template:
    metadata:
      labels:
        app: demo-nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

部署策略示例

```yaml
# Karmada PropagationPolicy
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
```

```
  name: demo-nginx-pp
  namespace: default      # (1)
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: demo-nginx  # (2)
  placement:
    clusterAffinity:
      clusterNames:
        - demo-stage
        - demo-dev
```

1. The default namespace is default.
2. If no namespace is specified, the namespace is inherited from the parent object scope.

差异化策略示例

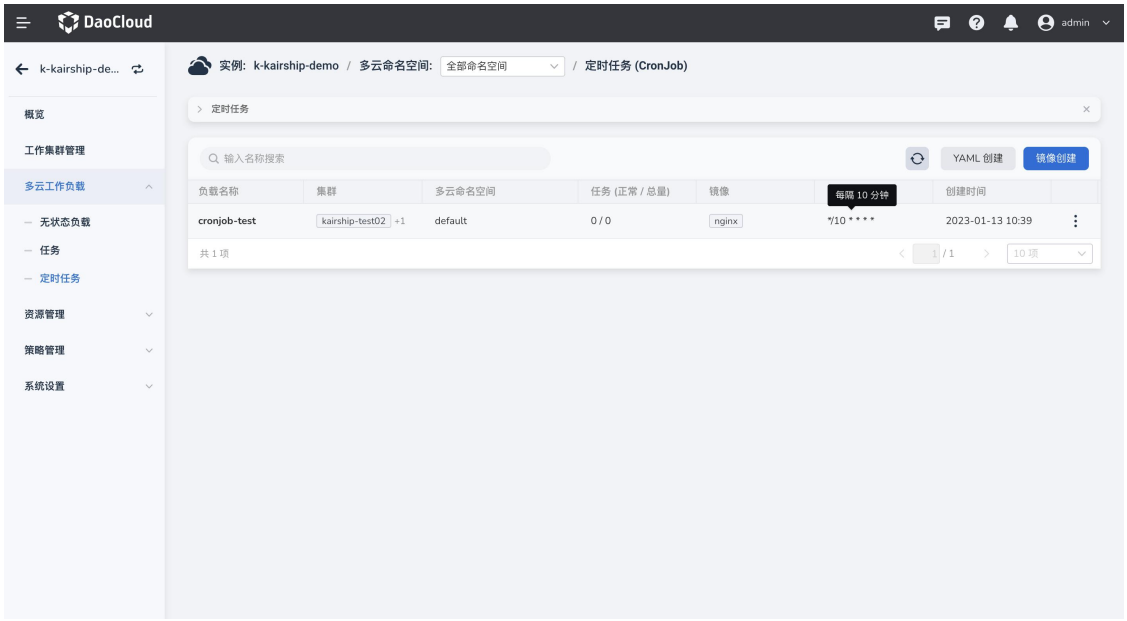
```
# Karmada OverridePolicy
apiVersion: policy.karmada.io/v1alpha1
kind: OverridePolicy
metadata:
  name: demo-nginx-op
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: demo-nginx
  overrideRules:
    - targetCluster:
        clusterNames:
          - demo-dev
      overriders:
```

```
      plaintext:
        - path: "/metadata/labels/env"
          operator: add
          value: demo-dev
  - targetCluster:
      clusterNames:
        - demo-stage
    overriders:
      plaintext:
        - path: "/metadata/labels/env"
          operator: add
          value: demo-stage
```

镜像创建定时任务

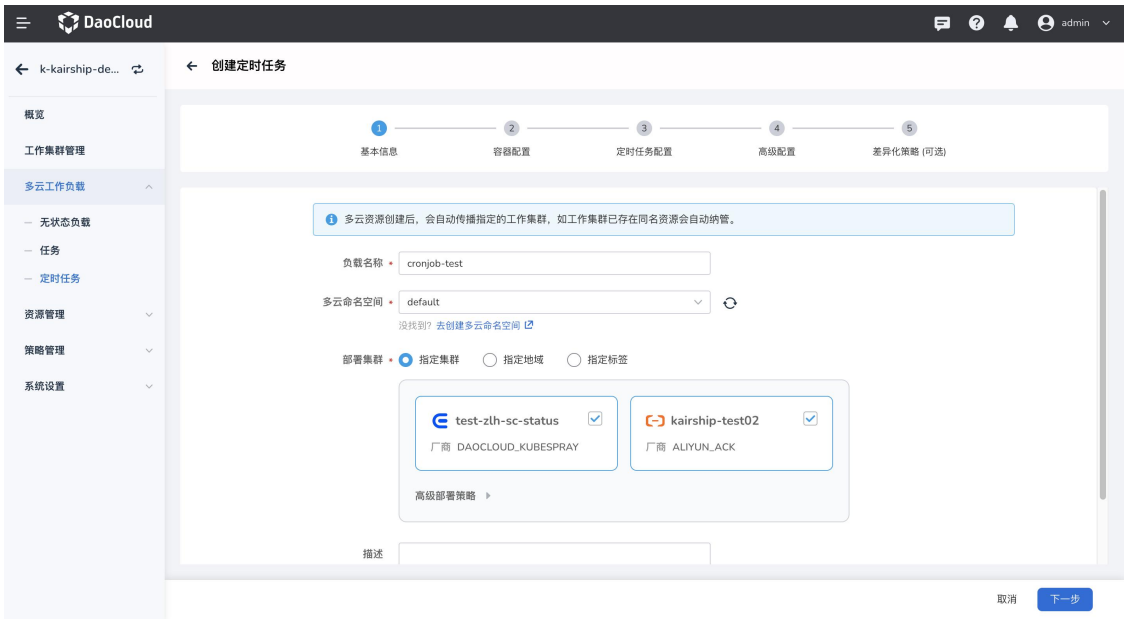
参照以下步骤创建一个定时任务（Cronob）。

在左侧导航栏中，点击多云工作负载，进入定时任务页面，点击右上角的镜像创建按钮。



镜像创建

在创建任务页面中，配置负载的基本信息后，点击下一步。



基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 多云命名空间：选择将新建的任务部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的。
- 部署集群：提供三种选择的方式，决定工作负载将会部署在哪些集群上。
- 实例数：输入工作负载的 Pod 实例数量。默认创建 1 个 Pod 实例。
- 调度策略：提供三种选择方式，决定将工作负载的实例以哪种方式进行分配。
- 描述：输入工作负载的描述信息，内容自定义。字符数量应不超过 512 个。

部署集群分为三种方式：

- 指定集群：通过选择集群名称来指定集群
- 指定地域：通过选择厂商、地域、可用区来指定集群
- 指定标签：通过添加指标来指定集群

- 还可以在高级部署策略中选择排除集群/设置集群污点容忍度/动态区域选择，最终会在底部展示预计会被调度到哪些集群上。

调度策略分为三种方式

- 重复：在所选的所有集群内分别调度相同的实例数
- 聚合：将实例总数尽可能少的调度在所选择的所有集群中
- 动态权重：根据所选择的所有集群实时的可调度最大实例数，等比例运行所填实例总数

在容器配置页面中，配置负载所在容器的基本信息，可选择配置生命周期、健康检查等信息，然后点击下一步。



容器配置

在定时任务配置页面，配置并发策略、定时规则、任务记录、其他配置等信息



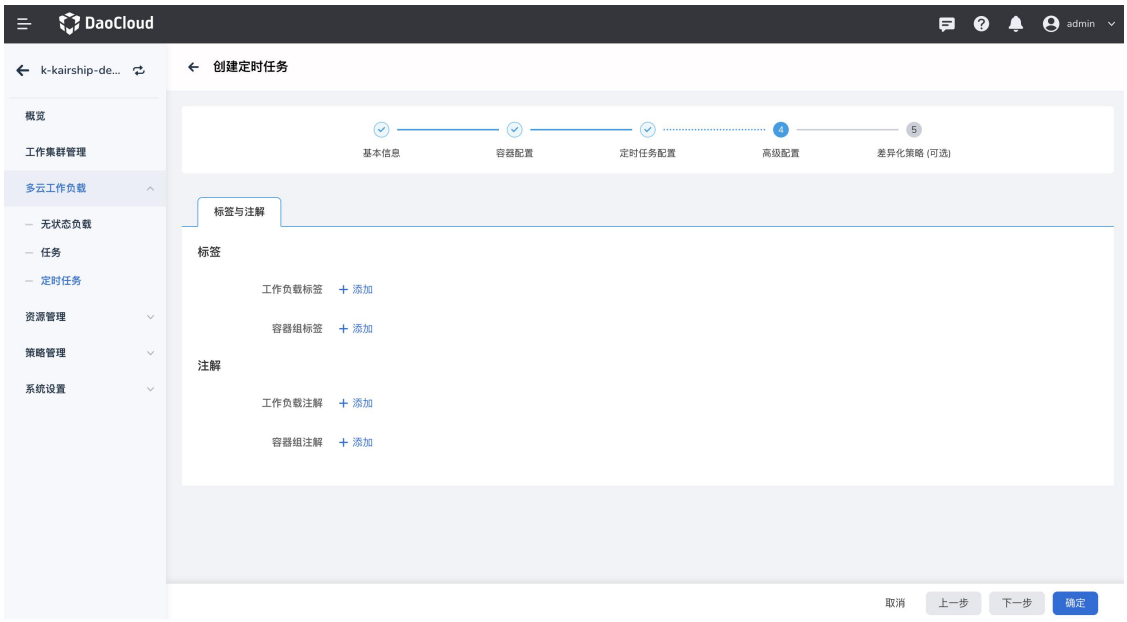
定时任务配置

- 并发策略：是否允许多个 Job 任务并行执行。
  - Allow：可以在前一个任务未完成时就创建新的定时任务，而且多个任务可以并行。任务太多可能抢占集群资源。
  - Forbid：在前一个任务完成之前，不能创建新任务，如果新任务的执行时间到了而之前的任务仍未执行完，CronJob 会忽略新任务的执行。
  - Replace：如果新任务的执行时间到了，但前一个任务还未完成，新的任务会取代前一个任务。

  上述规则仅适用于同一个 CronJob 创建的多个任务。多个 CronJob 创建的多个任务总是允许并发执行。
- 定时规则：基于分钟、小时、天、周、月设置任务执行的时间周期。支持用数字和 * 自定义 Cron 表达式，**输入表达式后下方会提示当前表达式的含义**。有关详细的表达式语法规则，可参考 Cron 时间表语法。
- 任务记录：设定保留多少条任务执行成功或失败的记录。0 表示不保留。

- 超时时间：超出该时间时，任务就会被标识为执行失败，任务下的所有 Pod 都会被删除。为空时表示不设置超时时间。默认值为 360 s。
- 重试次数：任务可重试次数，默认值为 6。
- 重启策略：设置任务失败时是否重启 Pod。

在高级配置页面中，可以配置任务设置和标签与注解。



高级配置

如果您在创建完成后，无需配置差异化时，则可以直接使用 确认 完成多云任务的创建

在差异化配置页面中，选择个性化的容器配置、标签和注解后，点击确定。



差异化配置

您可以在左侧列表区域增加对应的差异化配置项，当您新增一个差异化配置项之后，需要指定对应的集群。 集群的可选择范围仅为刚开始选择的集群，被选择的集群则会使用指定的差异化配置；未被指定的集群仍旧使用默认配置

!!! note

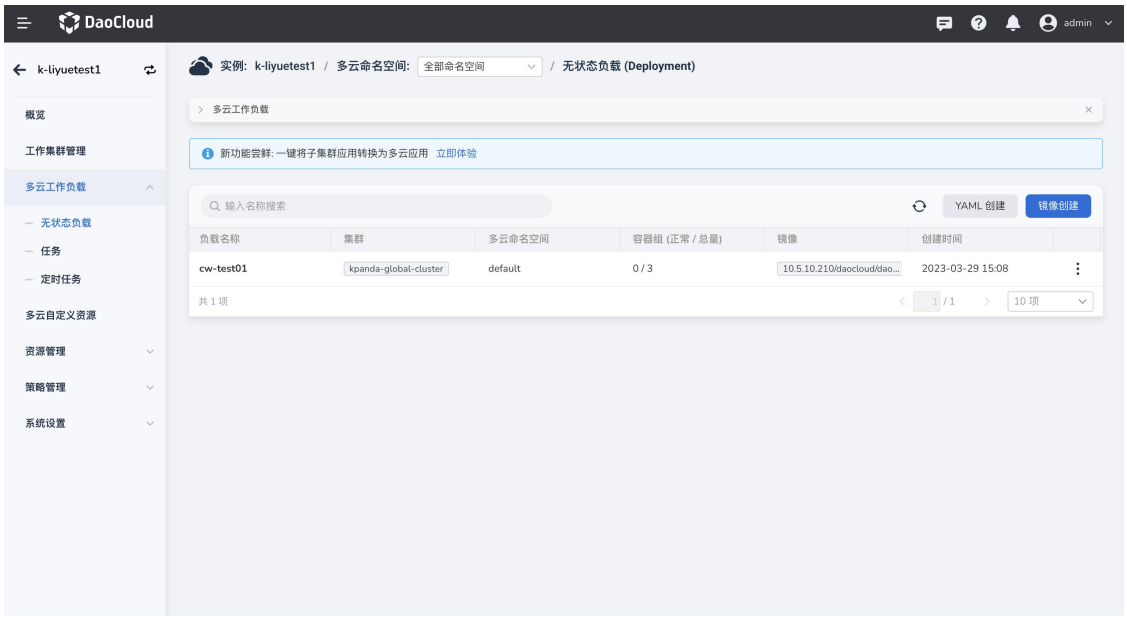
– 通过镜像创建多云工作负载时，如果需要采用指定位置、指定标签的高级能力进行创建时，需要确保工作集群已设定对应的位置或者标签；
增加标签需要在单个集群内增加，可由工作集群管理列表跳转到对应的集群维护。
– 配置副本数时，需要关注对应的调度策略，仅重复时会在多个集群中全部启动配置的副本数。

一键转换为多云工作负载

借助多云编排模块，您可以将单云工作负载一键转换为多云工作负载，操作非常简单，可以极大地提高多云环境中的运维效率。

具体操作步骤如下：

1. 点击多云实例的名称，在左侧点击多云工作负载，选择立即体验新功能。

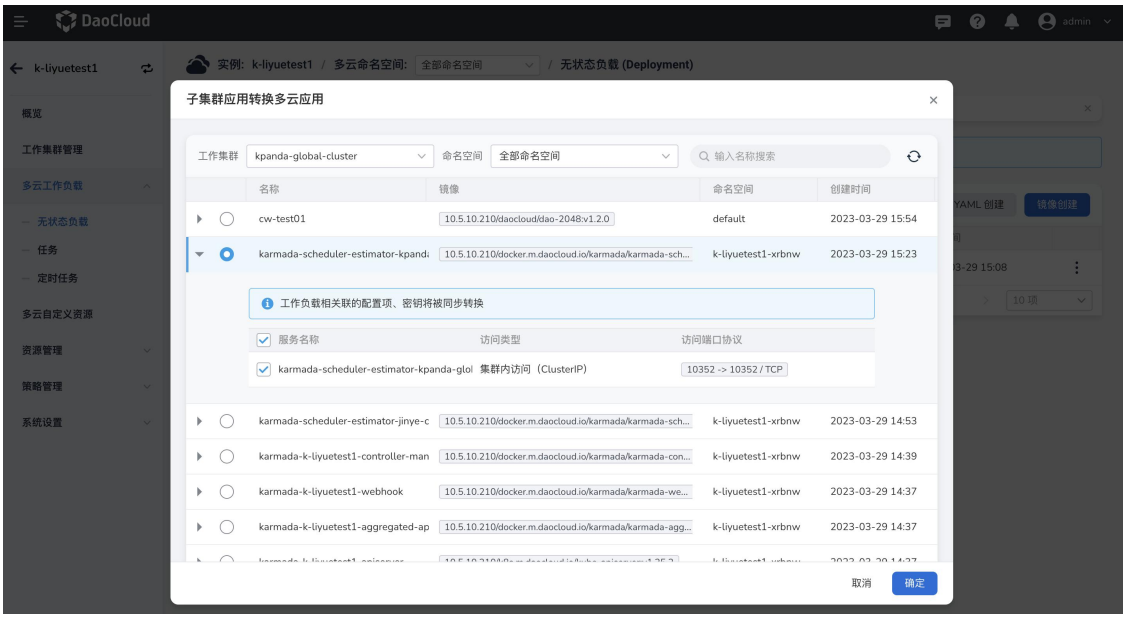


工作负载

2. 选择需要将哪个应用转换为多云应用。

    - 支持通过集群和命名空间筛选应用，或直接通过名称搜索应用
    - 转换时选择是否同步转换应用所关联的配置项和密钥，默认会同步转换。
    - 这里仅展示当前多云实例中已接入的工作集群中的应用。如未找到目标应用，可先将应用所在集群添加到当前工作负载后再次尝试。

- 已经转化为多云应用的资源不支持二次转换，不会出现在列表中。
- 转换过程中，系统会自动创建对应的部署策略，纳管原来的子集群。



转换应用

3. 在弹框右下角点击确定即可完成转换。

   应用转化为多云应用之后，和原生的多云工作负载一样支持编辑、更新等操作。

常见问题

- 被转换的工作负载是否会在转换成功后重启?

  不会重启，支持无感转换。

- 转换成功后，如果在部署策略中删除该工作负载所在的原集群，原集群中的工作负载副本也会一并被删除吗?

  会的。一旦转换成功，该工作负载就是一个标准的多云工作负载。当部署策略发生变化使得工作负载不再传播到原集群时，根据 Karmada 的设计原理，该集群中工作负载副本也会被一并删除以保证一致性。

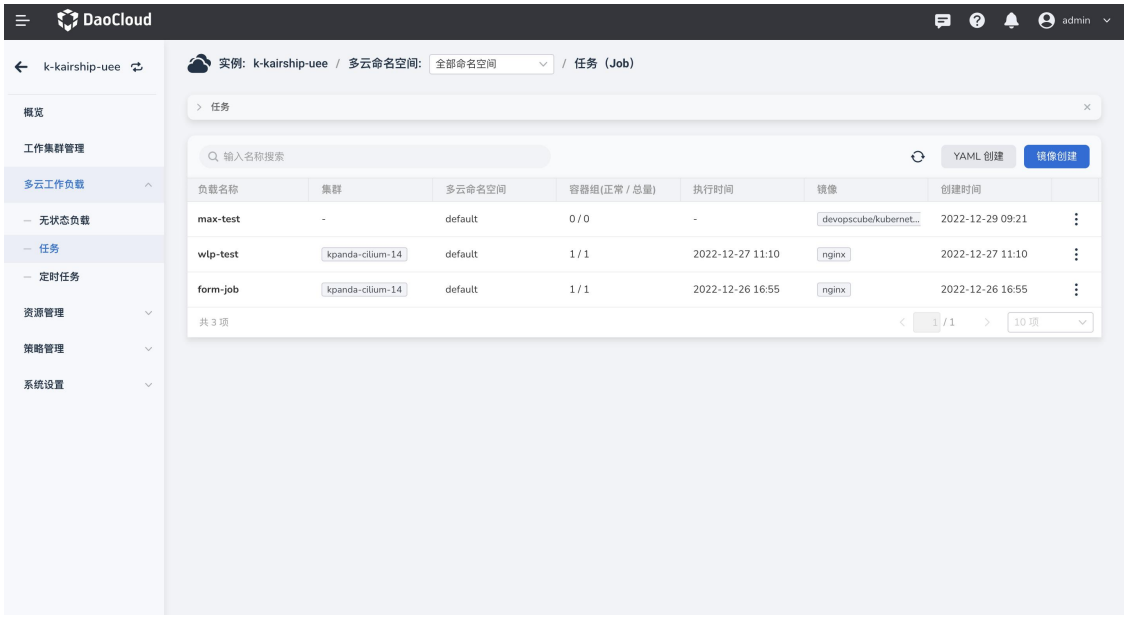
- 当前支持转换哪些 Kubernetes 资源?

  目前仅支持主动显式地将 Deployment 转换为多云资源，如果该 Deployment 关联了对应的配置项和密钥，则用户可以自由选择是否一并将配置项和密钥也转换为多云资源。默认会一并转换。这样做的目的是，当多云化之后的工作负载被分发到其他集群时，该工作负载依赖的资源也同步存在，否则可能会导致工作负载启动异常。

镜像创建任务

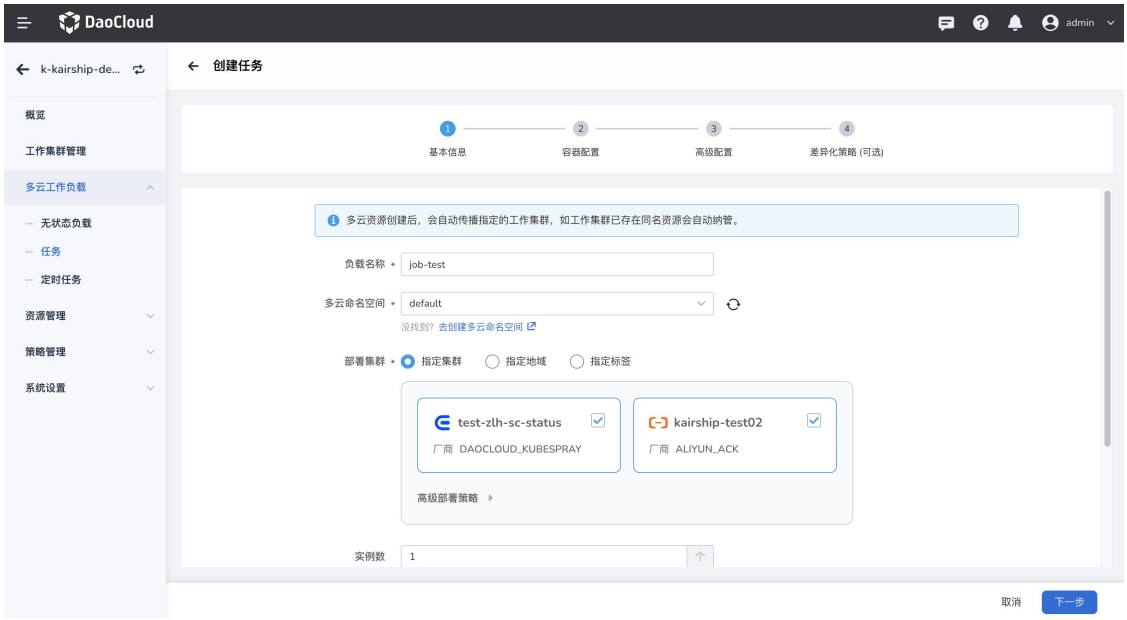
参照以下步骤创建一个任务（Job）。

在左侧导航栏中，点击多云工作负载，进入任务页面，点击右上角的镜像创建按钮。



镜像创建

在创建任务页面中，配置负载的基本信息后，点击下一步。



基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 多云命名空间：选择将新建的任务部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的。
- 部署集群：提供三种选择的方式，决定工作负载将会部署在哪些集群上。
- 实例数：输入工作负载的 Pod 实例数量。默认创建 1 个 Pod 实例。
- 调度策略：提供三种选择方式，决定将工作负载的实例以哪种方式进行分配。
- 描述：输入工作负载的描述信息，内容自定义。字符数量应不超过 512 个。

部署集群分为三种方式：

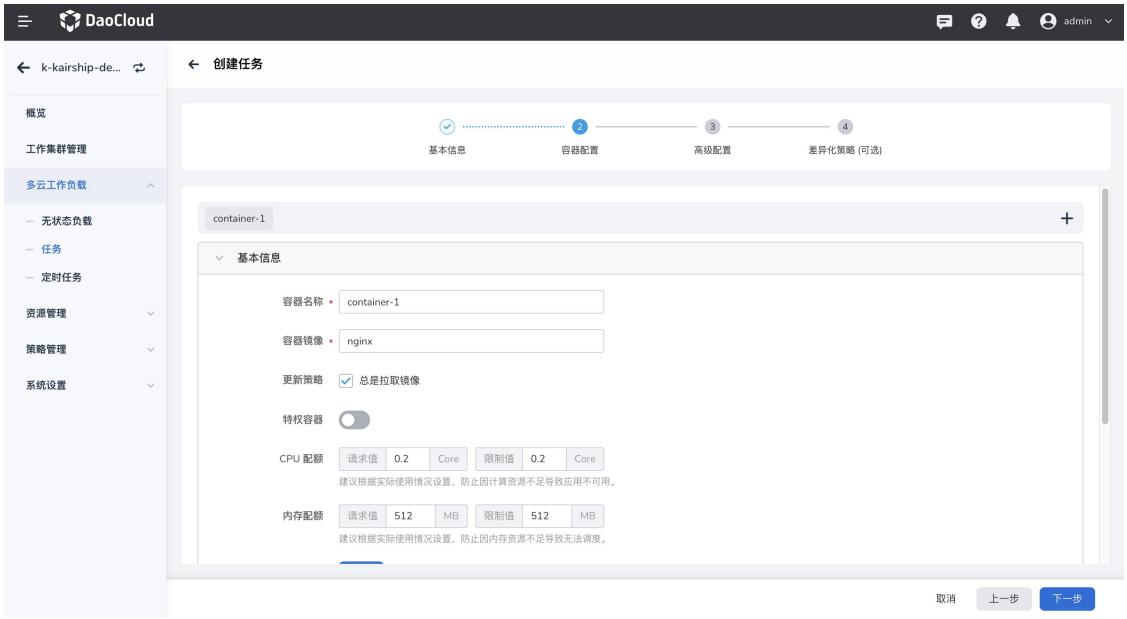
- 指定集群：通过选择集群名称来指定集群
- 指定地域：通过选择厂商、地域、可用区来指定集群
- 指定标签：通过添加指标来指定集群

- 还可以在高级部署策略中选择排除集群/设置集群污点容忍度/动态区域选择，最终会在底部展示预计会被调度到哪些集群上。

调度策略分为三种方式

- 重复：在所选的所有集群内分别调度相同的实例数
- 聚合：将实例总数尽可能少的调度在所选择的所有集群中
- 动态权重：根据所选择的所有集群实时的可调度最大实例数，等比例运行所填实例总数

在容器配置页面中，配置负载所在容器的基本信息，可选择配置生命周期、健康检查等信息，然后点击下一步。



容器配置

在高级配置页面中，可以配置任务设置和标签与注解。



高级配置

任务设置需要包括以下数据：

- 并行数：任务执行过程中允许同时创建的最大 Pod 数，并行数应不大于 Pod 总数。默认为 1。
- 超时时间：超出该时间时，任务会被标识为执行失败，任务下的所有 Pod 都会被删除。为空时表示不设置超时时间。
- 重启策略：设置失败时是否重启 Pod。

如果您在创建完成后，无需配置差异化时，则可以直接使用 确认 完成多云任务的创建

在差异化配置页面中，选择个性化的容器配置、标签和注解后，点击确定。



差异化配置

您可以在左侧列表区域增加对应的差异化配置项，当您新增一个差异化配置项之后，需要指定对应的集群。 集群的可选择范围仅为刚开始选择的集群，被选择的集群则会使用指定的差异化配置；未被指定的集群仍旧使用默认配置

!!! note

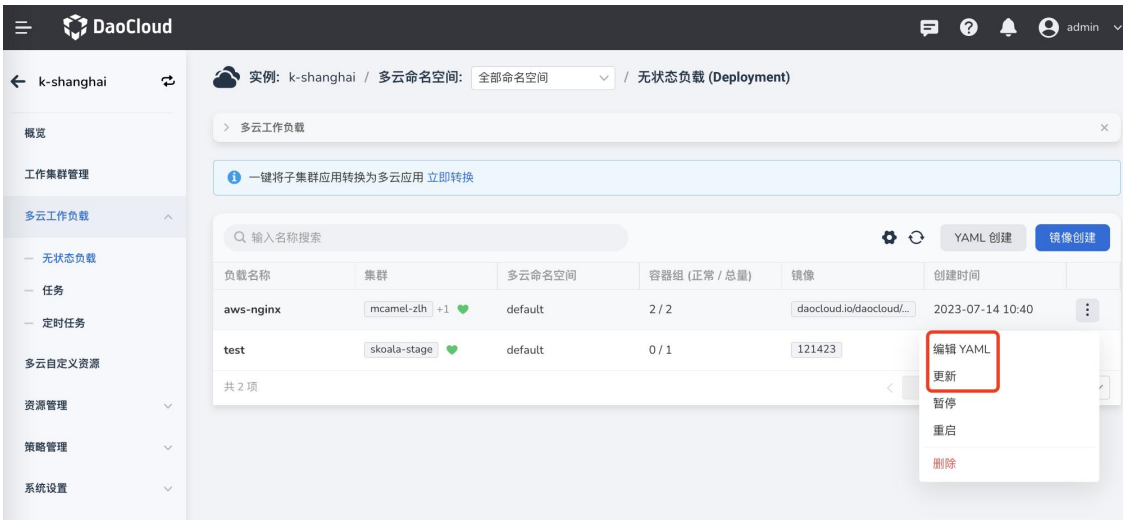
– 通过镜像创建多云工作负载时，如果需要采用指定位置、指定标签的高级能力进行创建时，需要确保工作集群已设定对应的位置或者标签；
增加标签需要在单个集群内增加，可由工作集群管理列表跳转到对应的集群维护。
– 配置副本数时，需要关注对应的调度策略，仅重复时会在多个集群中全部启动配置的副本数。

更新/删除多云工作负载

创建多云无状态工作负载（Deployment）时，可以通过镜像或 YAML 文件两种方式创建。因此，更新无状态工作负载时也可以通过这两种方式更新。

1. 在左侧导航栏点击多云工作负载–>无状态负载，在对应的负载最右侧点击显示更多操作。

    - 选择编辑 YAML 可以通过 YAML 更新无状态工作负载
    - 选择更新可以通过表单更新无状态工作负载
    - 选择删除可以直接删除无状态工作负载
    - 选择暂停可以让无状态工作负载暂时停止运行
    - 选择重启可以重新启动无状态工作负载

    更新无状态负载

2. 修改配置后点击确定即可完成更新。

多云配置项

配置项（ConfigMap）是一种 API 对象，用来将非机密性的数据保存到键值对中。 使用时，Pod 可以将其用作环境变量、命令行参数或者存储卷中的配置文件。

多云配置项可以将多云环境配置信息和容器镜像解耦，便于修改多云应用的配置。

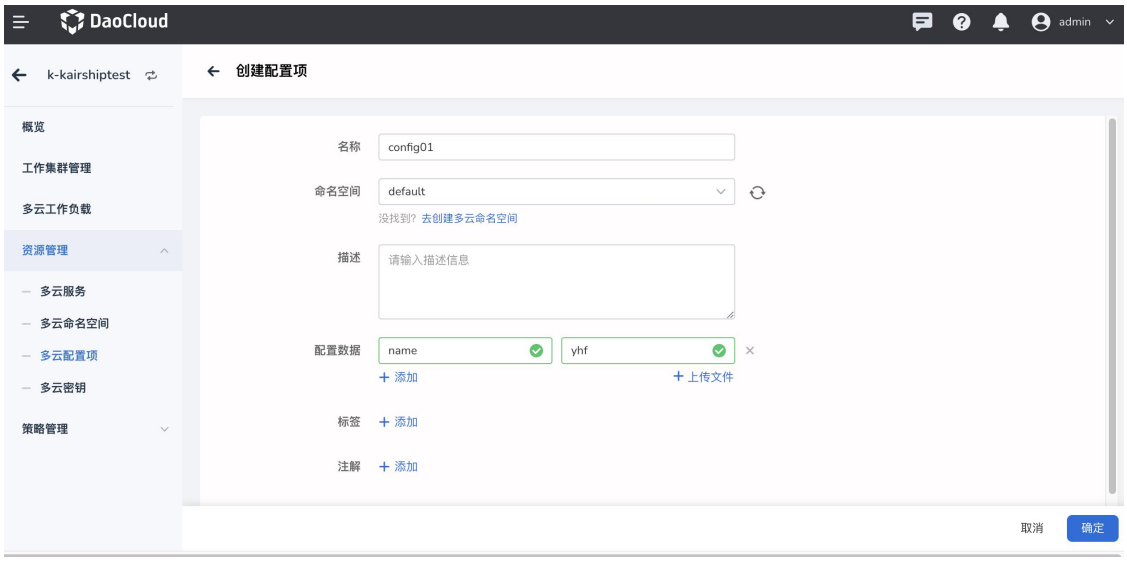
目前提供了两种创建方式：向导创建和 YAML 创建。本文以向导创建为例，参照以下步骤操作。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云配置项，点击右上角的创建配置项按钮。



点击按钮

2. 在创建配置项页面中，输入名称，选择命名空间等信息后，点击确定。



填写表单

3. 返回多云配置项列表，新创建的默认位于第一个。点击列表右侧的 ⋮，可以编辑 YAML、更新、导出和删除配置项。



其他操作

!!! note

若删除一个配置项，其相关信息将一并被删除，请谨慎操作。

YAML 示例

此处列出一个多云配置项的 YAML 示例，您稍加修改就可以使用。

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: cluster-info
  namespace: default
  uid: e8bf0788-d5e6-4b1f-8588-1b58c6d010d5
  resourceVersion: '1647402'
  creationTimestamp: '2022-09-26T07:26:25Z'
  labels:
    a: '1'
    b: '2'
    c: c
  annotations:
    c: '3'
    kairship.io/describe: '123'
    kpanda.io/describe: '13243'
    shadow.clusterpedia.io/cluster-name: k-kairshiptest
data:
  jws-kubeconfig-8utcre: >-
    eyJhbGciOiJIUzI1NiIsImtpZCI6Ijh1dGNyZSJ9..7-9oX6oeZsV5QJ_VsxBKFE7LPFM
mfYX4bQM3IDDBw80
  jws-kubeconfig-faw64f: >-
    eyJhbGciOiJIUzI1NiIsImtpZCI6ImZhdzY0ZiJ9..Hbtgm5MFOOfLekYn-NnGFCj4vm
-D1QS1h-Tm3ywcMr4
  jws-kubeconfig-kew06y: >-
    eyJhbGciOiJIUzI1NiIsImtpZCI6ImtldzA2eSJ9..nOG2817zEvF8tkmPGrE_r1vWM4kv
A-5v6i29EA73Jb0
  kubeconfig: |
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC
```

k1JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJN
d0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJeU1Ea3lOakF6TXprek
5Wb1hEVE15TURreU16QXpNemt6TlZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYS
nVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2d
nRUJBTVhYCkNSS2F6QnlFRithVWNuVHFGVVFST0JxUXZabjc4Q3h5RnI5QVdvU0Nh
K1hFTkZJZVBPaGdudmd2Z1VadzZ1bmYKZWN4ZHJRblltMzRNOE1rQ0dDK21hTDNt
WXJHTUNpVGl0dnNqMllOd3NCMjh4TlZPQ052UG5iZWRKOTFIYWdhbAprQ3psWGR
0STlNLzdCK0xoQWdYcXlPS0NpVTd4U1ZGKzI6dGYvTU9odGlXVmpGR3RxUjZNZUk
3TnRaZmY2cXZxCkJmMUVpSlR2QXBVK0l4NWh6ajJJYUVHWStzTTUzSC8vSnhjVHF
RdmRjVXJOUW5SbXVZS2t0eDV5TWwzTHh5K28KcjJkbElhTVRnQi9GeWF1ZElrTmZz
LzF2a3IxcTdnek5xc1NGSFVhQlhZWUlTYXQ5V1MwRmhpVXpHbENDZjFjdQpaRjBlNU
10V2M2UXRVSXZZenI4Q0F3RUFBYU5aTUZjd0RnWURWUjBQQVFIL0JBUURBZ0trT
UE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZMd0VUMm1P
ZlVQc3hjZHhTZ1Z1VVpvdVRtekRNQlVHQTFVZEVRUU8KTUF5Q0NtdDFZbVZ5Ym1W
MFpYTXdEUVlKS29aSWh2Y05BUUVMQlFBRGdnRUJBQWRQSHlNSW1zc1JLOTY0e
WExbQpHRXVBMzNwUU9wdkJzSWJRZHI4R1diRWoyakk1eUZ1UVBJV3loOGRJTjE1V
nN0YldSekZSZkRHQ1pWSGh6RWdMCks1U1dsMFU3MDVzQUd4UGFaQ3hEVUx0alB
RSEgwNVdodzBaUkYxdHB5K2RPNk5MeWJwTVdpU2FndlFqQmpTMy8KVjZZbHc2N
DFMdDc0eU9QcUJDckFKWUtYOTUzanhEdWZNSjNVV0dBa1VpaVVLWTcycXBCWGx
vNVkyU2RCTmVTZwo5ME1TQ25VdDA2YkRtQ0lMaGI0OVVnZTFvamdldVVFVHJkeG
ppNWplQWp3bHpvZTFRcmY1bnZnRnlNV0tlK05oCnRlajdHRktHRnArVHJxSVh1TmVP
c1dER3o0WldtQ3Z1bm8zRXdhSUdRNThKZWk3YjM5d2F0ZWY1K3FlOStNTWlKZ1BzP
QotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
        server: https://10.6.168.131:6443
      name: ""
    contexts: null
    current-context: ""
    kind: Config
    preferences: {}
    users: null

多云密钥

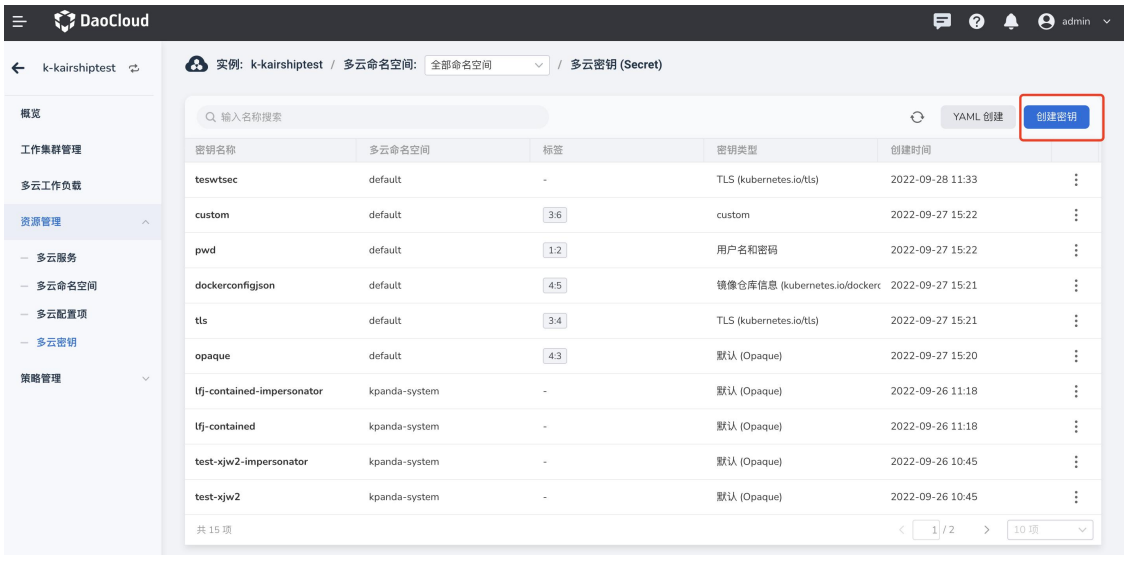
密钥（Secret）是一种包含密码、令牌或密钥等少量敏感信息的对象。 这样的信息可能会被放在 Pod 规约中或者镜像中。 使用 Secret 意味着您不需要在应用程序代码中包含机密数据。

由于创建 Secret 可以独立于使用它们的 Pod，因此在创建、查看和编辑 Pod 的工作流程中暴露 Secret（及其数据）的风险较小。 DCE 和在集群中运行的应用程序也可以对 Secret 采取额外的预防措施，例如避免将机密数据写入非易失性存储。

Secret 类似于 ConfigMap 但专门用于保存机密数据。

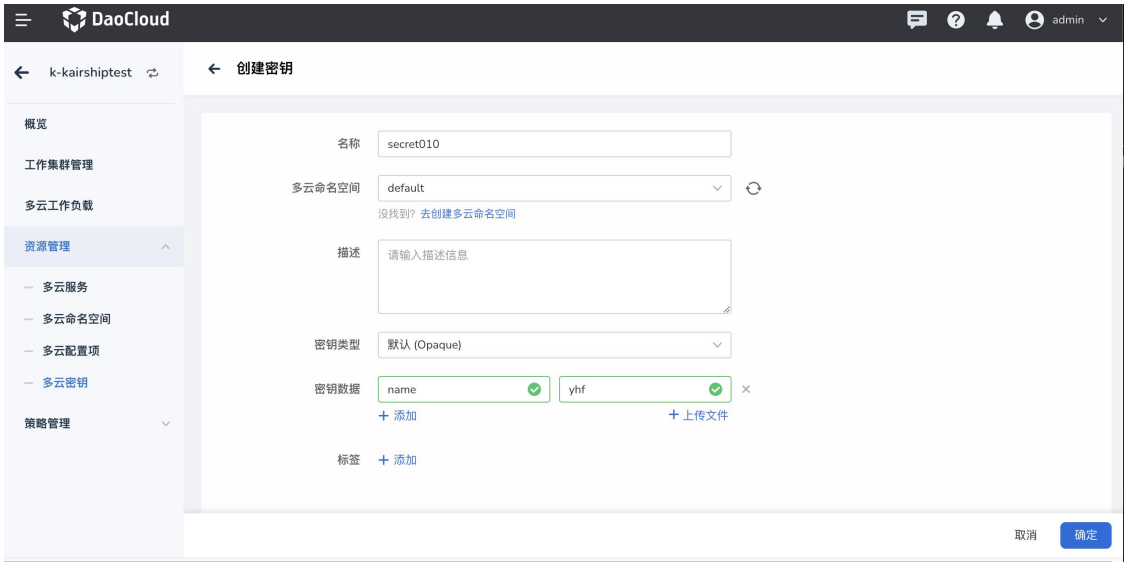
目前提供了两种创建方式：向导创建和 YAML 创建。本文以向导创建为例，参照以下步骤操作。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云密钥，点击右上角的创建密钥按钮。



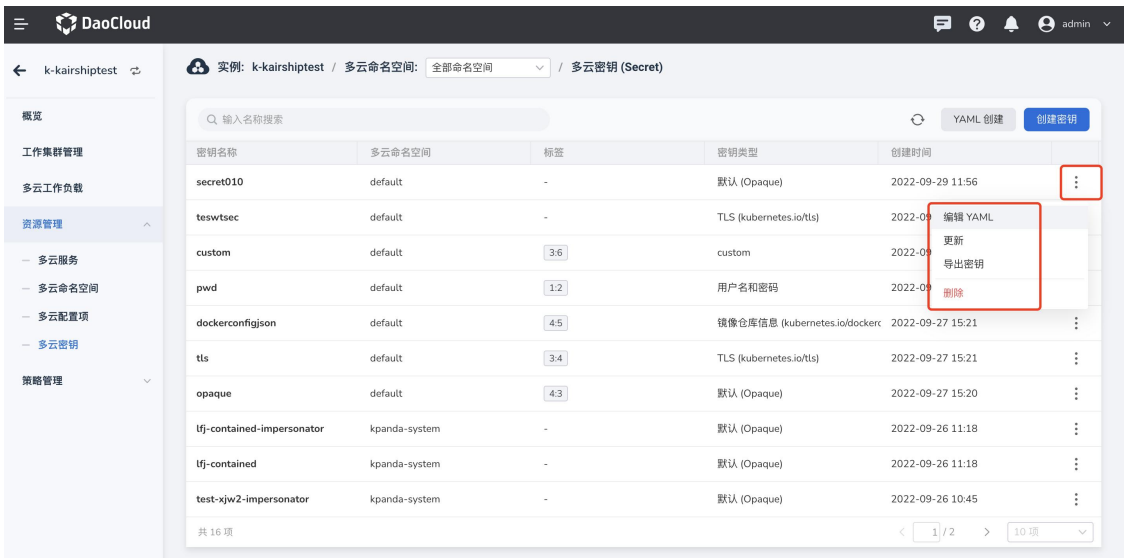
点击按钮

2. 在创建密钥页面中，输入名称，选择命名空间等信息后，点击确定。



填写表单

3. 返回多云密钥列表，新创建的默认位于第一个。点击列表右侧的 ⋮，可以编辑 YAML、更新、导出和删除 Secret。



其他操作

!!! note

若删除一个 Secret，其相关信息将一并被删除，请谨慎操作。

YAML 示例

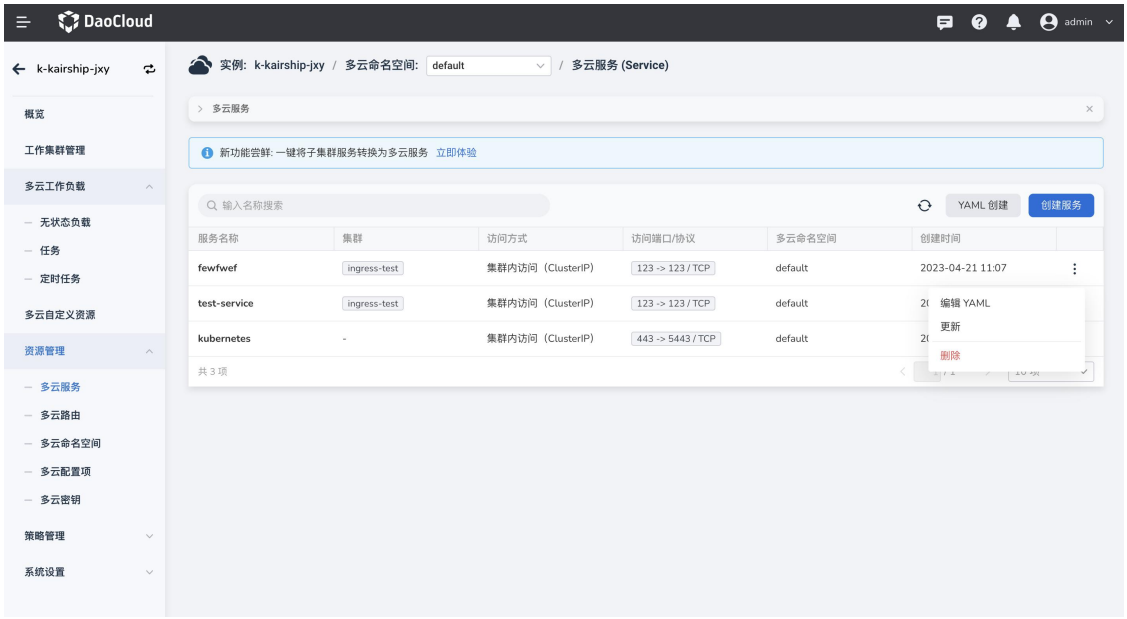
此处列出一个多云 Secret 的 YAML 示例，您稍加修改就可以使用。

```yaml
kind: Secret
apiVersion: v1
metadata:
  name: dockerconfigjson
  namespace: default
  uid: 452c3465-e9a1-4869-8972-2f199ba2750a
  resourceVersion: '1679594'
  creationTimestamp: '2022-09-27T07:21:36Z'
  labels:
    '4': '5'
  annotations:
    kairship.io/describe: test
    shadow.clusterpedia.io/cluster-name: k-kairshiptest
data:
  .dockerconfigjson: >-
    eyJhdXRocyI6eyIxIjp7InVzZXJuYW1lIjoiMiIsInBhc3N3b3JkIjoiMyIsImF1dGgiOiJNam96In19fQ==
type: kubernetes.io/dockerconfigjson
```

更新多云服务

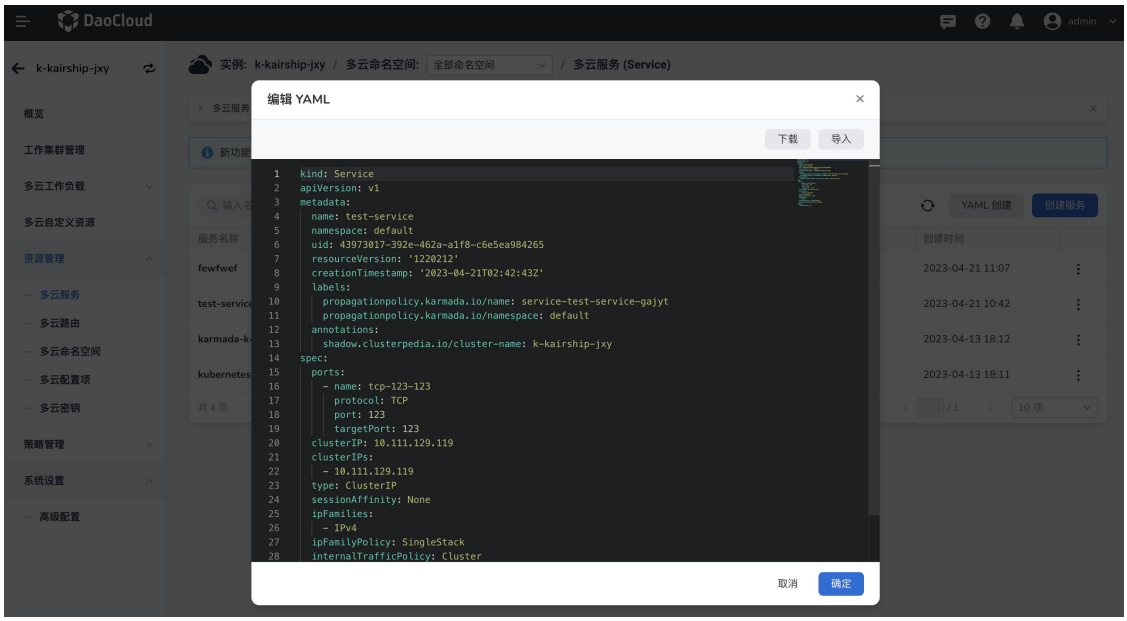
目前提供了两种更新方式：编辑 YAML 和表单更新。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云服务，点击···可以表单更新服务。



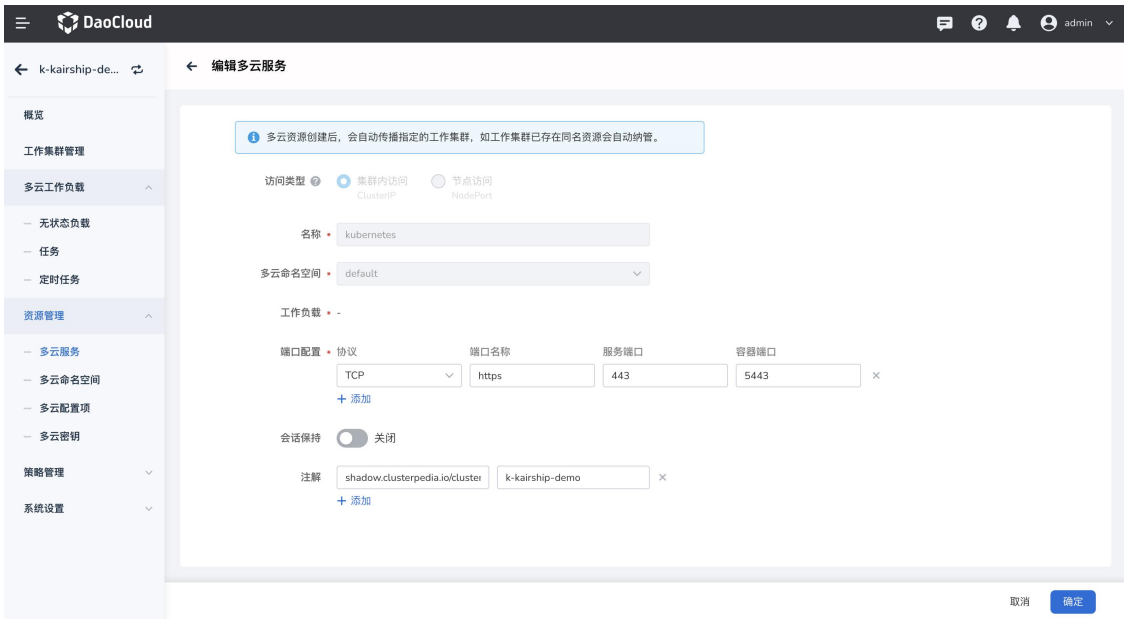
服务列表

2. 在编辑 YAML 弹窗中，修改 YAML 信息，点击确定则更新了。



YAML 更新

3. 点击更新，可以通过表单方式更新服务，访问类型、名称、多云命名空间都无法更新。



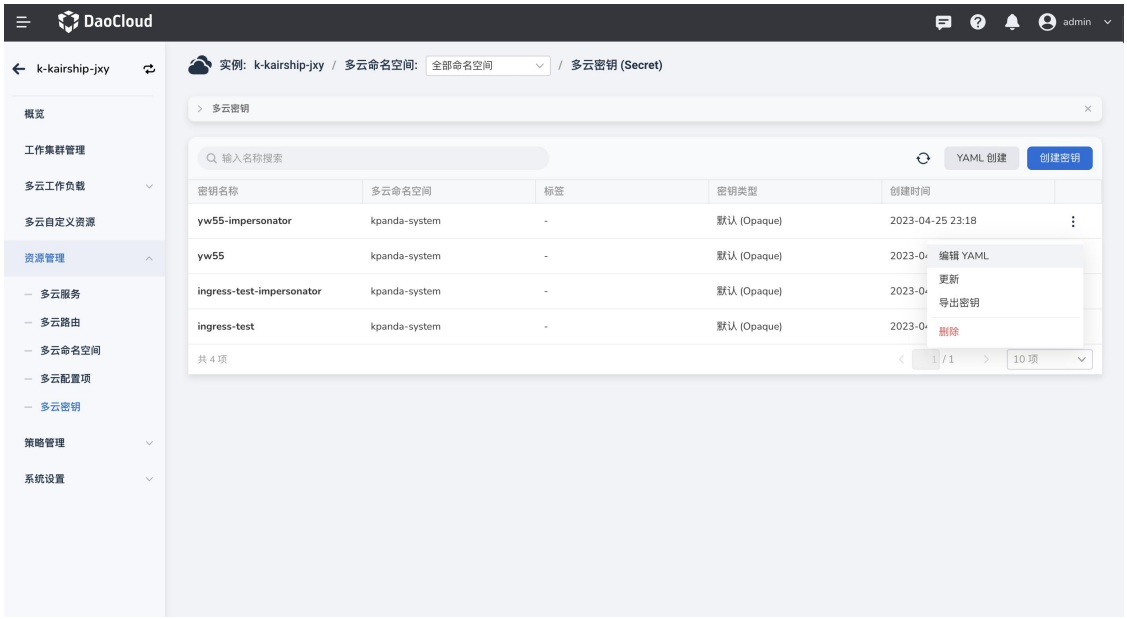
表单更新

4. 完成需要修改的内容后点击确定，更新成功。

更新多云密钥

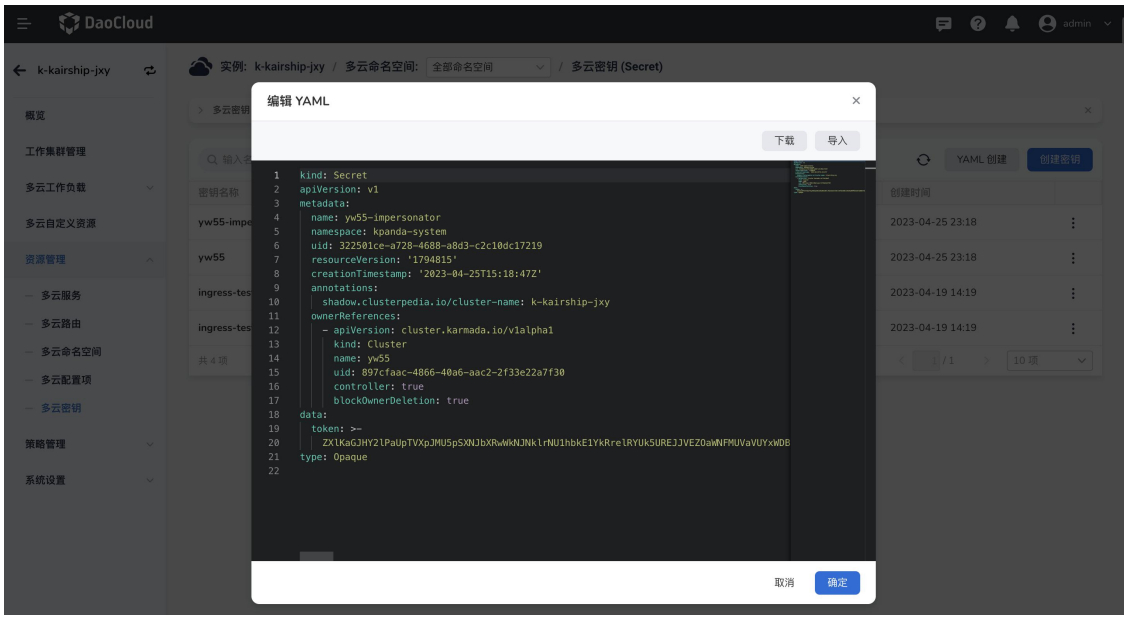
目前提供了两种更新方式：编辑 YAML 和表单更新。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云密钥，点击···可以编辑 YAML 或者更新密钥。



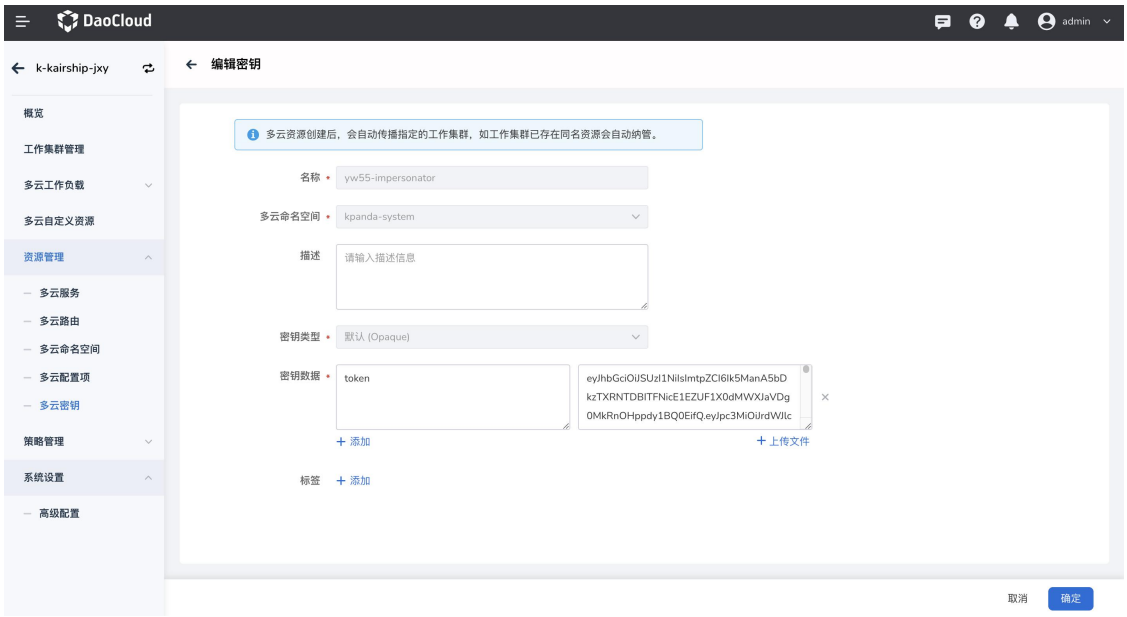
密钥列表

2. 在编辑 YAML 弹窗中，修改 YAML 信息，点击确定则更新了密钥。



YAML 更新

3. 点击更新，可以通过表单方式更新，名称、多云命名空间、密钥类型无法更新。



表单更新

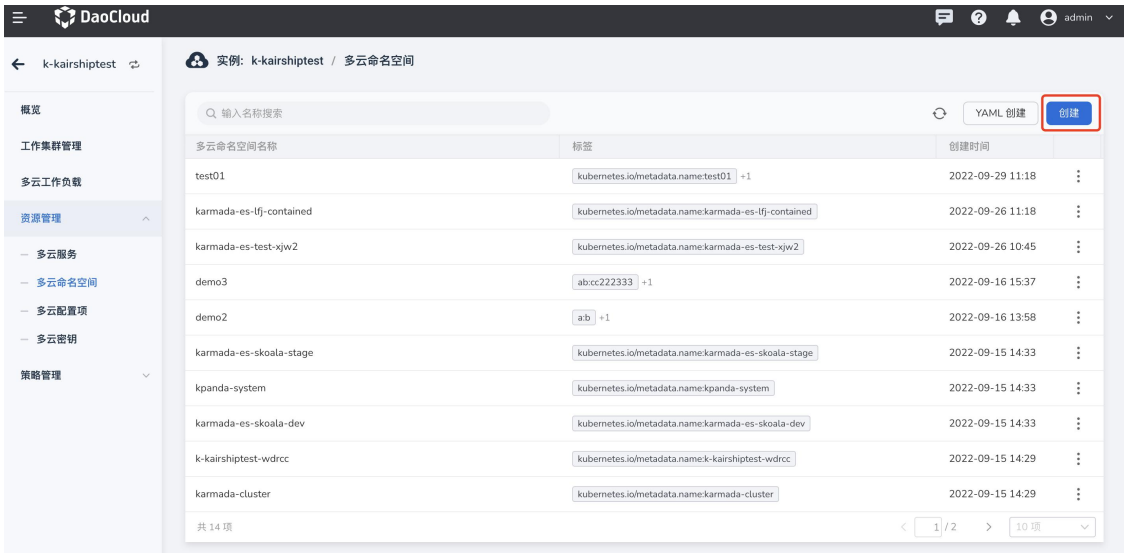
4. 完成需要修改的内容后点击确定，更新成功。

多云命名空间

多云命名空间可以跨云、跨集群管理工作负载。目前提供了基于向导创建的方式。

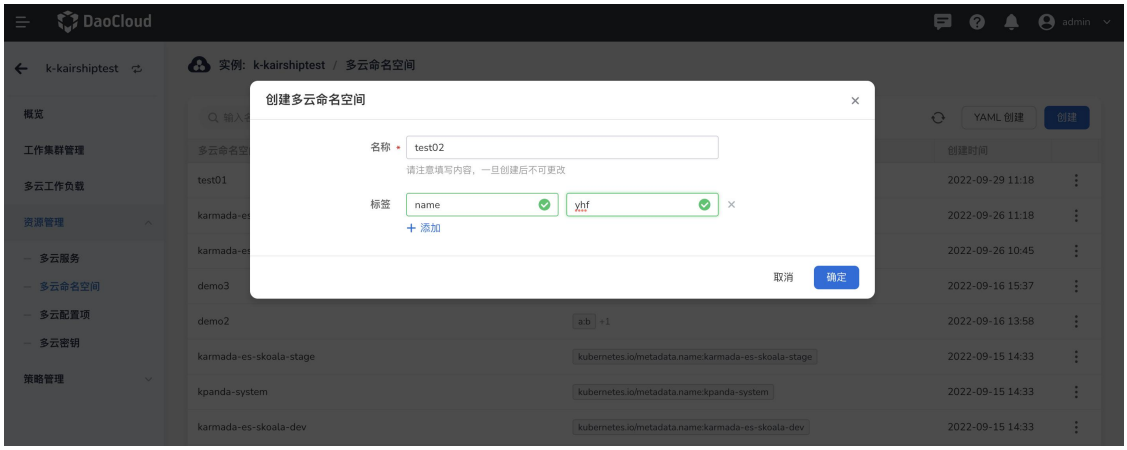
本文以向导创建为例，参照以下步骤操作。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云命名空间，点击右上角的创建按钮。



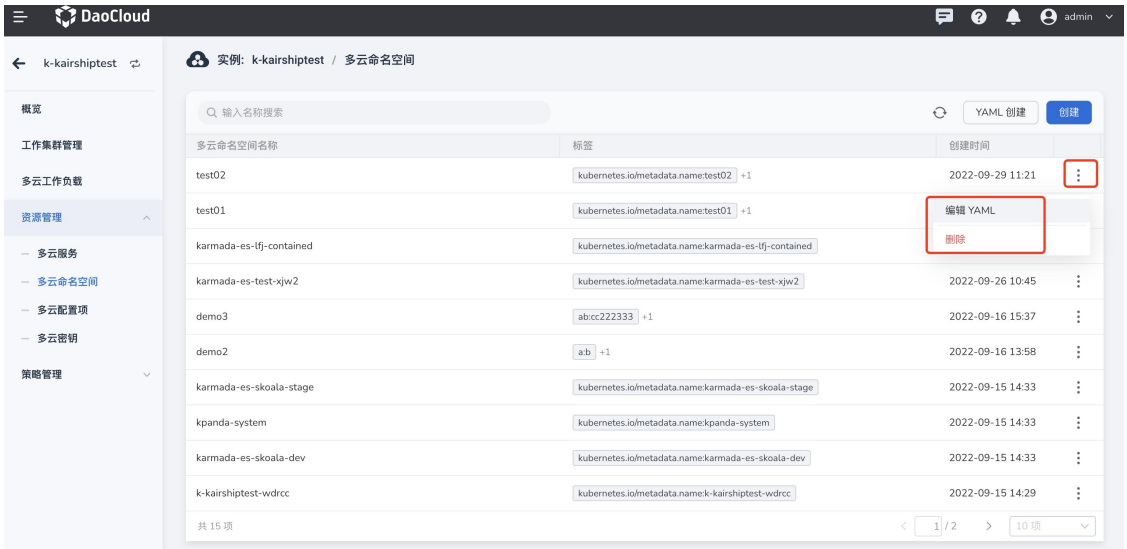
点击创建按钮

2. 在创建多云命名空间页面中，输入名称，添加标签后，点击确定。



填写表单

3. 返回多云命名空间列表，新创建的默认位于第一个。点击列表右侧的 ⋮，可以编辑 YAML 和删除该命名空间。



其他操作

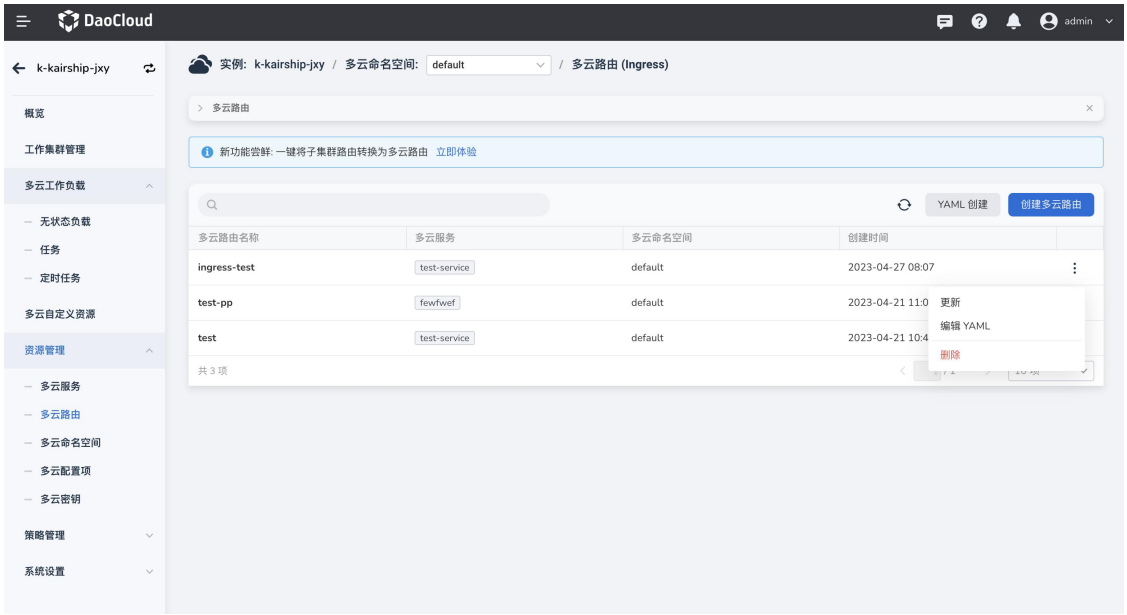
!!! note

若要删除一个命名空间，需要先移除该命名空间下的所有工作负载，删除之后命名空间内的工作负载和服务都会受到影响，请谨慎操作。

更新多云路由

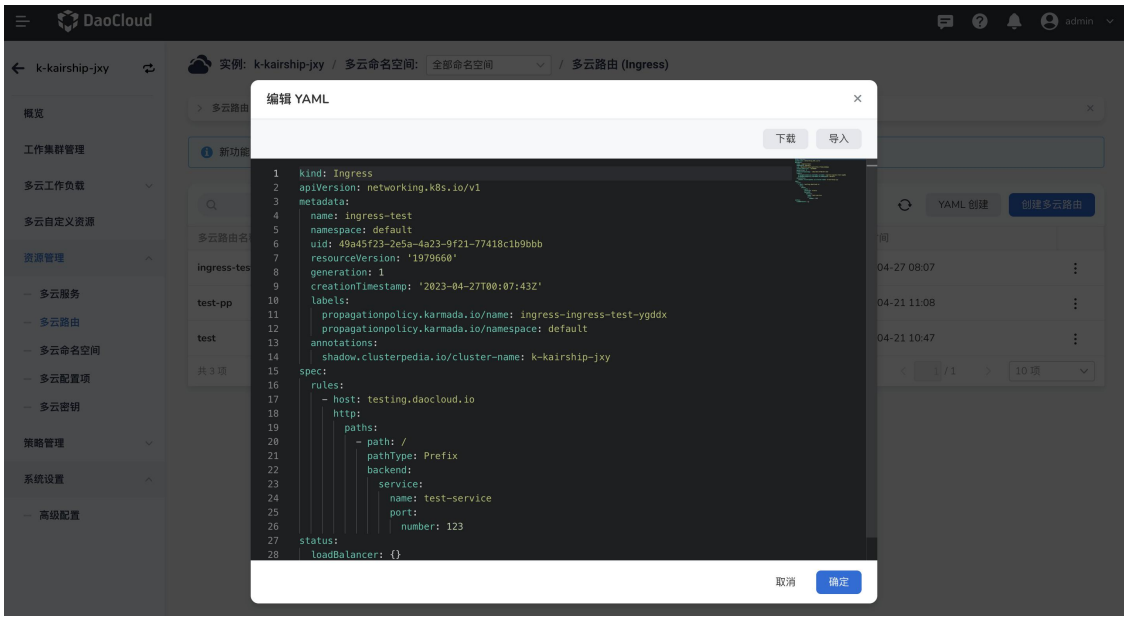
目前提供了两种更新方式：编辑 YAML 和表单更新。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云路由，点击···可以表单更新路由。



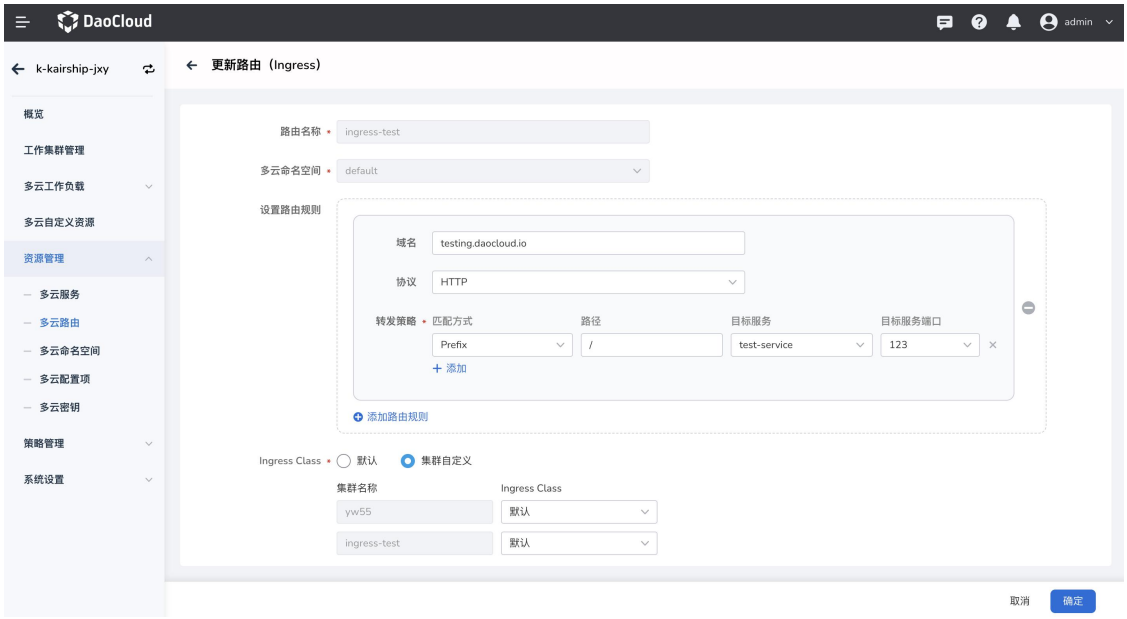
路由列表

2. 在编辑 YAML 弹窗中，修改 YAML 信息，点击确定则更新了路由。



YAML 更新

3. 点击更新，可以通过表单方式更新路由，名称、多云命名空间、Ingress Class 的集群无法更新。



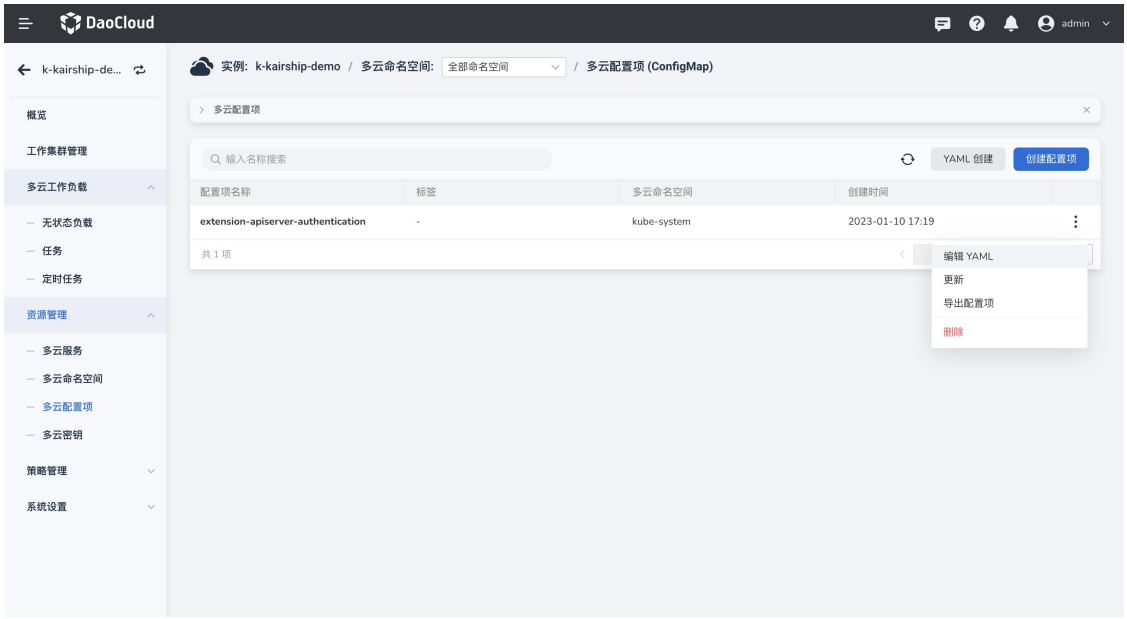
表单更新

4. 完成需要修改的内容后点击确定，更新成功。

更新多云配置项

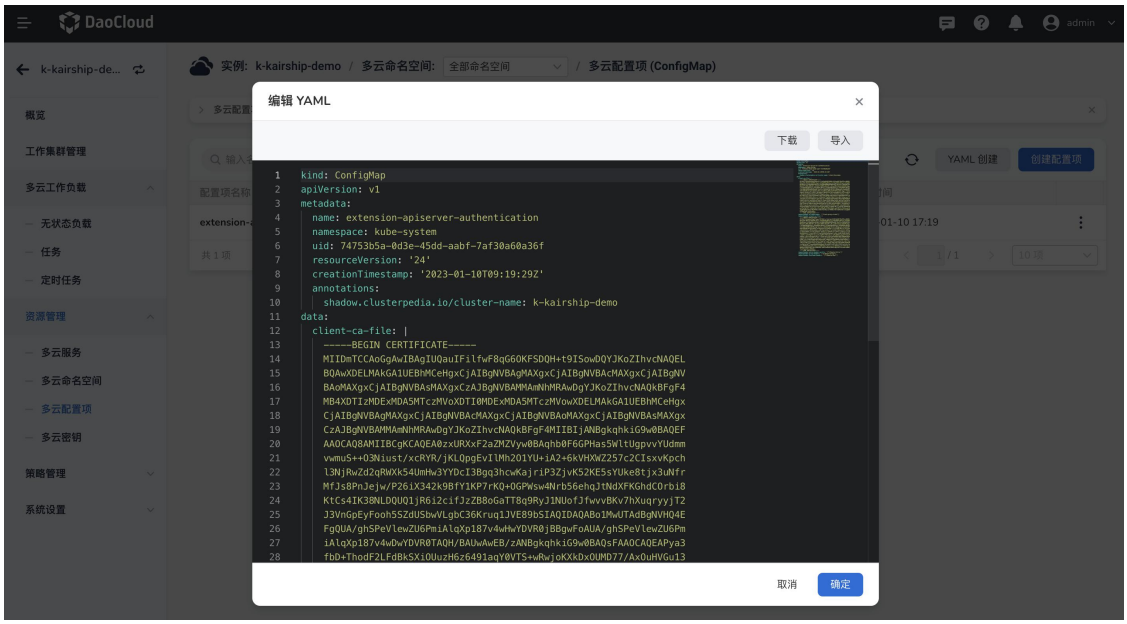
目前提供了两种更新方式：编辑 YAML 和表单更新。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云配置项，点击···可以编辑 YAML 或者更新配置项。



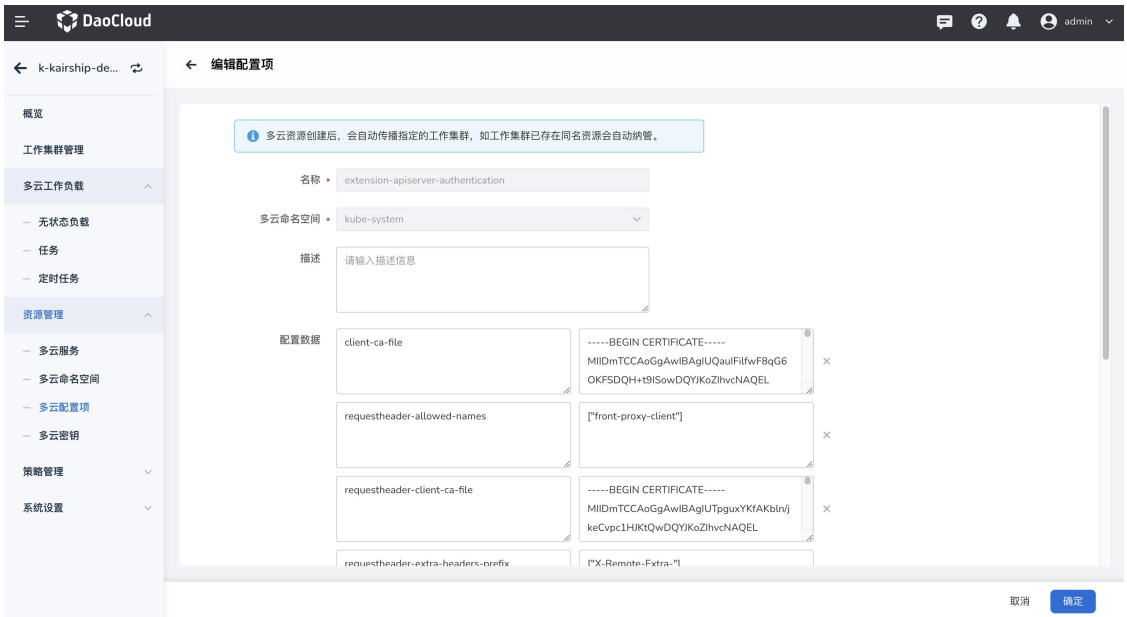
更新配置项

2. 在编辑 YAML 弹窗中，修改 YAML 信息，点击确定则更新了配置项。



编辑 YAML

3. 点击更新，可以通过表单方式更新配置项，负载名称和多云命名空间无法更新。



表单更新

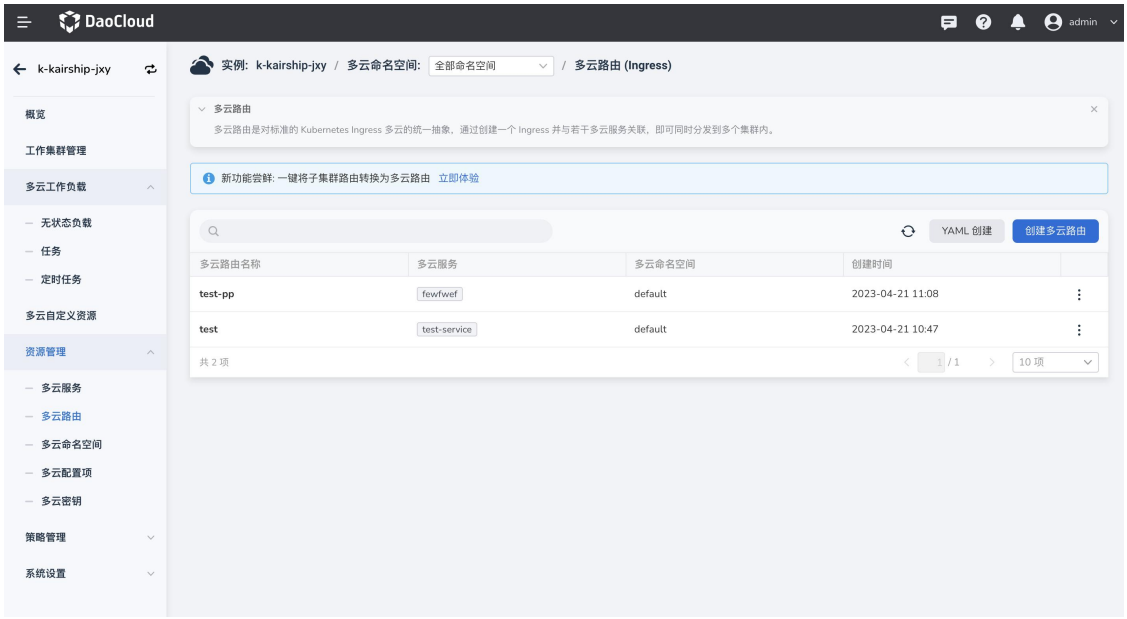
4. 完成需要修改的内容后点击确定，更新成功。

多云路由

多云路由是对标准的 Kubernetes Ingress 多云的统一抽象，通过创建一个 Ingress 并与若干多云服务关联，即可同时分发到多个集群内。

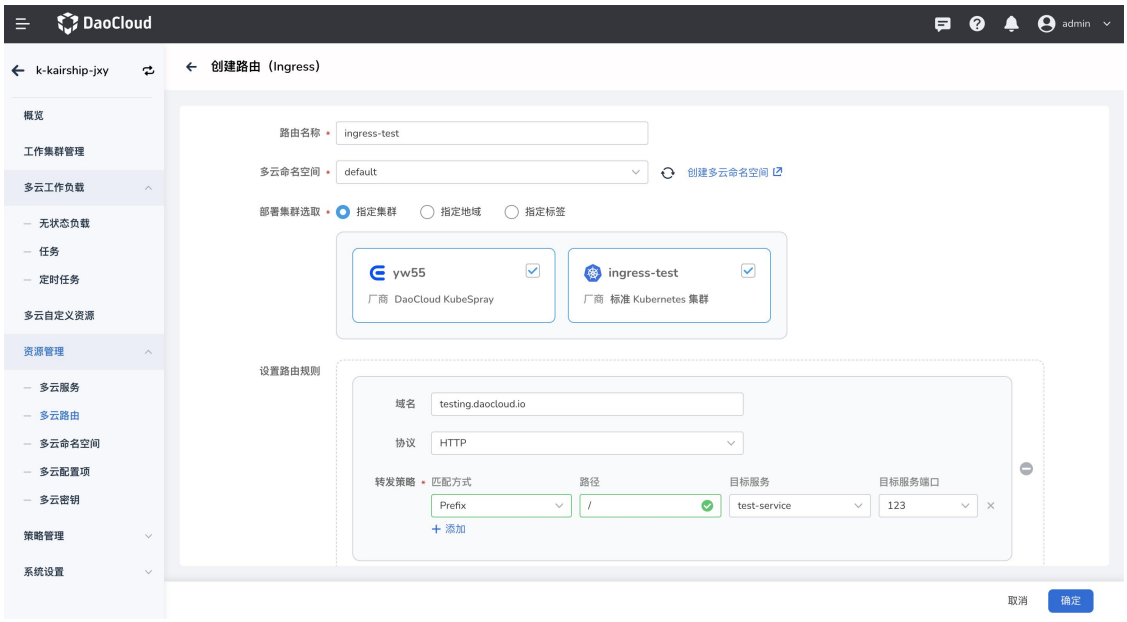
目前提供了两种创建方式：表单创建和 YAML 创建。本文以表单创建为例，参照以下步骤操作。

1. 进入某一个多云实例后，在左侧导航栏中，点击资源管理 -> 多云路由，点击右上角的创建按钮。



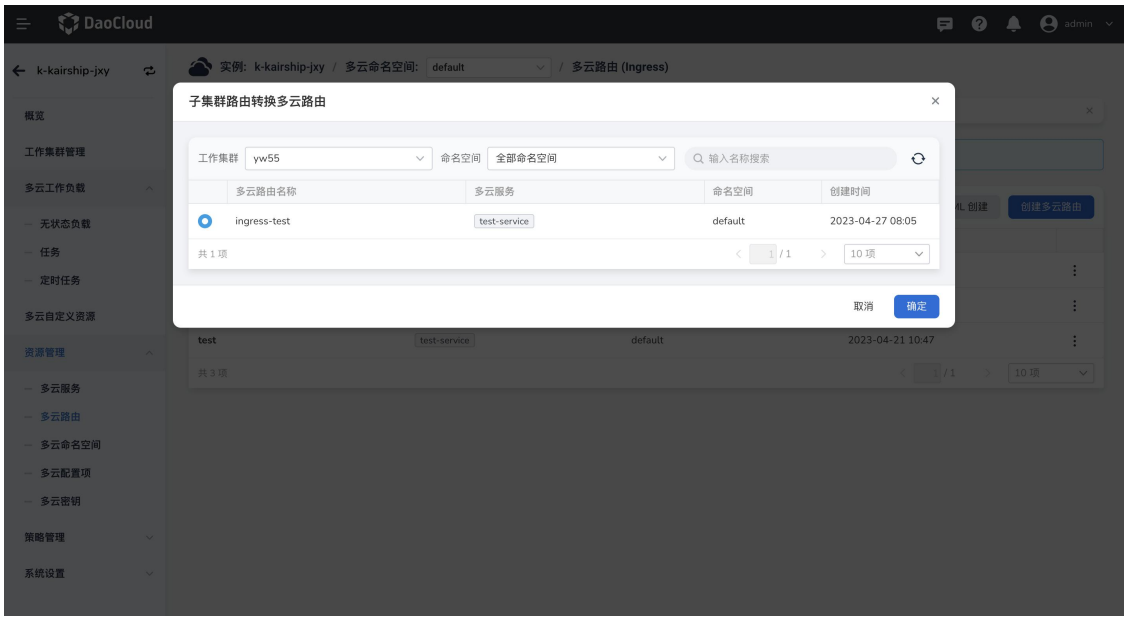
路由列表

2. 在创建多云路由页面中，配置部署位置、设置路由规则、Ingress Class、是否开启会话保持等信息后，点击确定。填写详情可参考创建路由



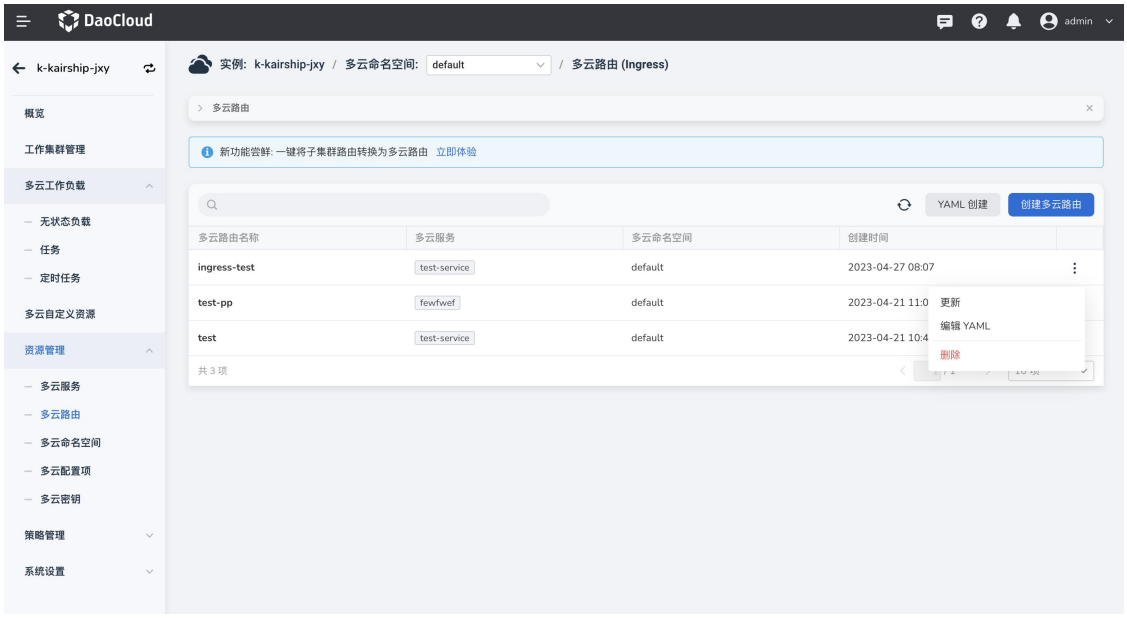
创建多云路由

3. 新功能尝鲜，支持一键将子集群服务转换为多云路由，点击列表页的立即体验，选择指定工作集群和命名空间下的路由，点击确定后可以转换成功。



一键转换

4. 点击列表右侧的 ⋮，可以更新和删除该路由。



更新和删除

!!! note

如果一个路由被删除，该服务相关的信息也将消失，请谨慎操作。

YAML 示例

此处列出一个多云路由的 YAML 示例，您稍加修改就可以使用。

```yaml
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
  name: ingress-test
  namespace: default
  uid: 49a45f23-2e5a-4a23-9f21-77418c1b9bbb
  resourceVersion: '1979660'
  generation: 1
  creationTimestamp: '2023-04-27T00:07:43Z'
  labels:
    propagationpolicy.karmada.io/name: ingress-ingress-test-ygddx
    propagationpolicy.karmada.io/namespace: default
```

```yaml
  annotations:
    shadow.clusterpedia.io/cluster-name: k-kairship-jxy
spec:
  rules:
    - host: testing.daocloud.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: test-service
                port:
                  number: 123
status:
  loadBalancer: {}
```

实现 DCE4 到 DCE5 应用的一键转换

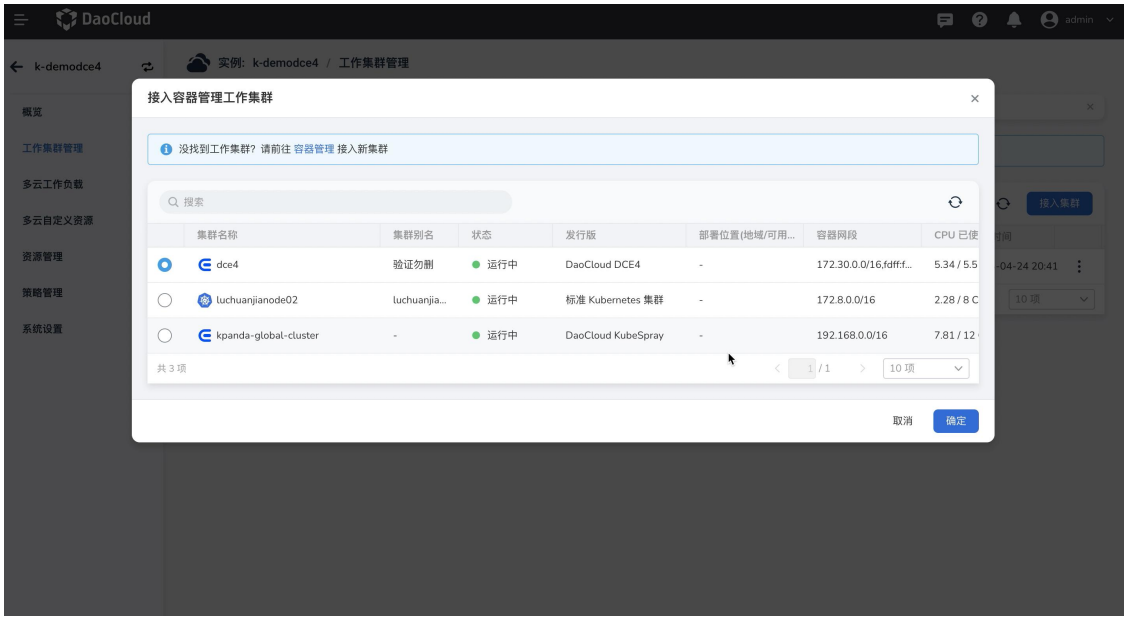
本节将以无状态负载 nginx 为例介绍如何通过多云编排界面实现 DCE4 到 DCE5 应用的一键转换，

前提条件

- 容器管理模块接入集群厂商为 DaoCloud DCE 的集群，可参考已接入 Kubernetes 集群，且能够访问集群的 UI 界面。
- DCE4 集群中工作负载能够正常运行。

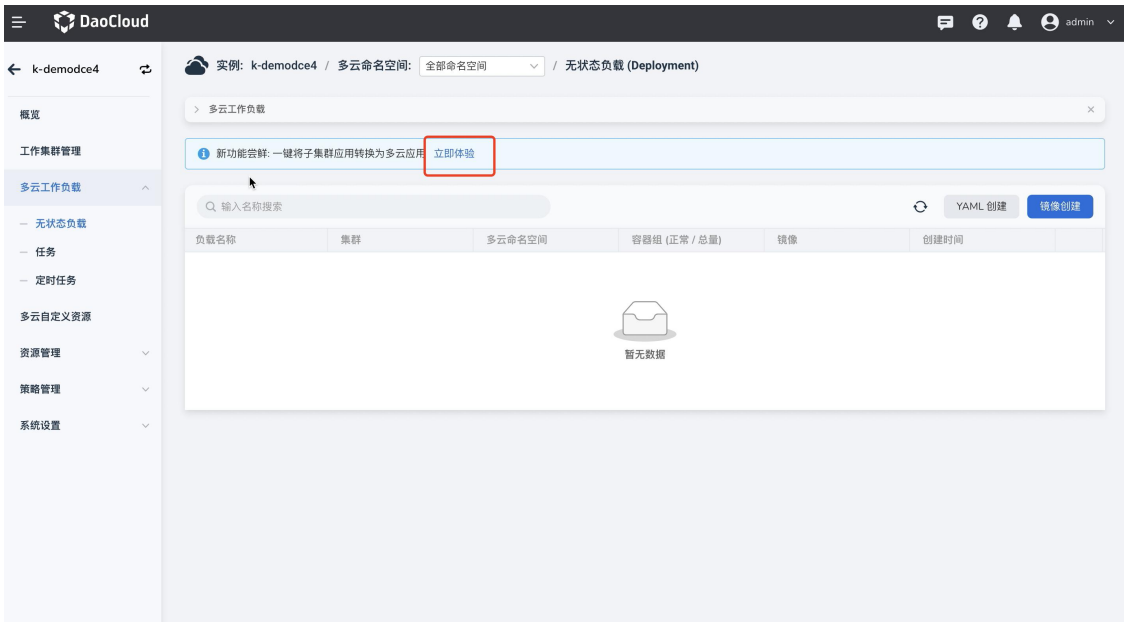
一键转移

1. 进入多云实例–工作负载管理，点击接入集群选择将 DCE4 集群接入多云实例。

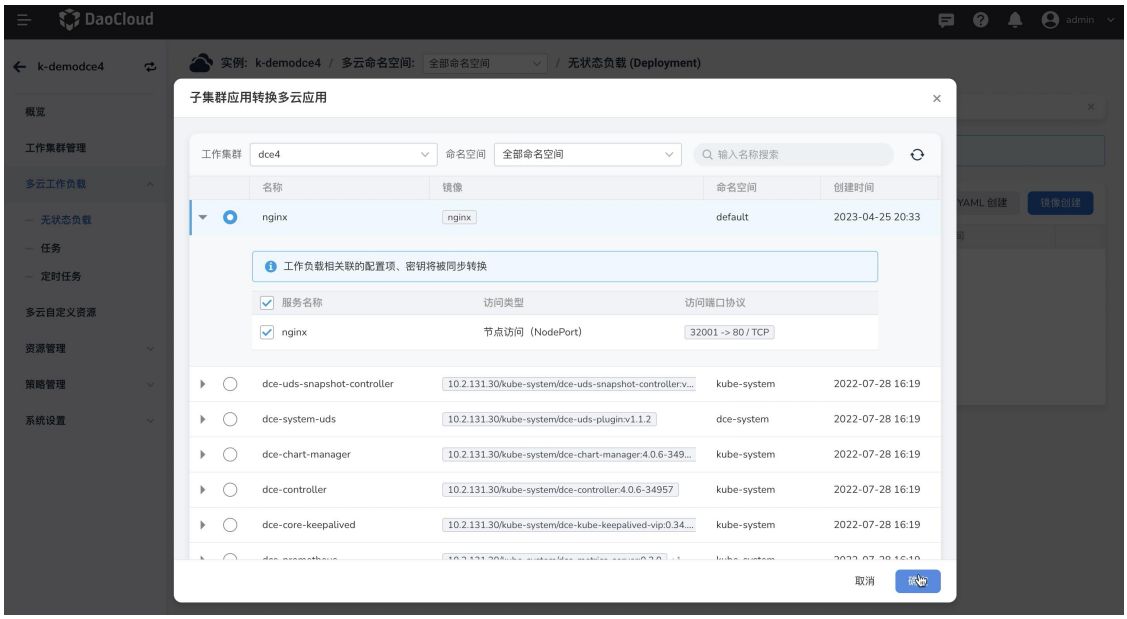


接入集群

2. 进入多云工作负载–无状态负载，点击立即体验，选中目标应用，将会自动勾选其关联的 service，同时相关联的配置项、密钥也会被同步转换。

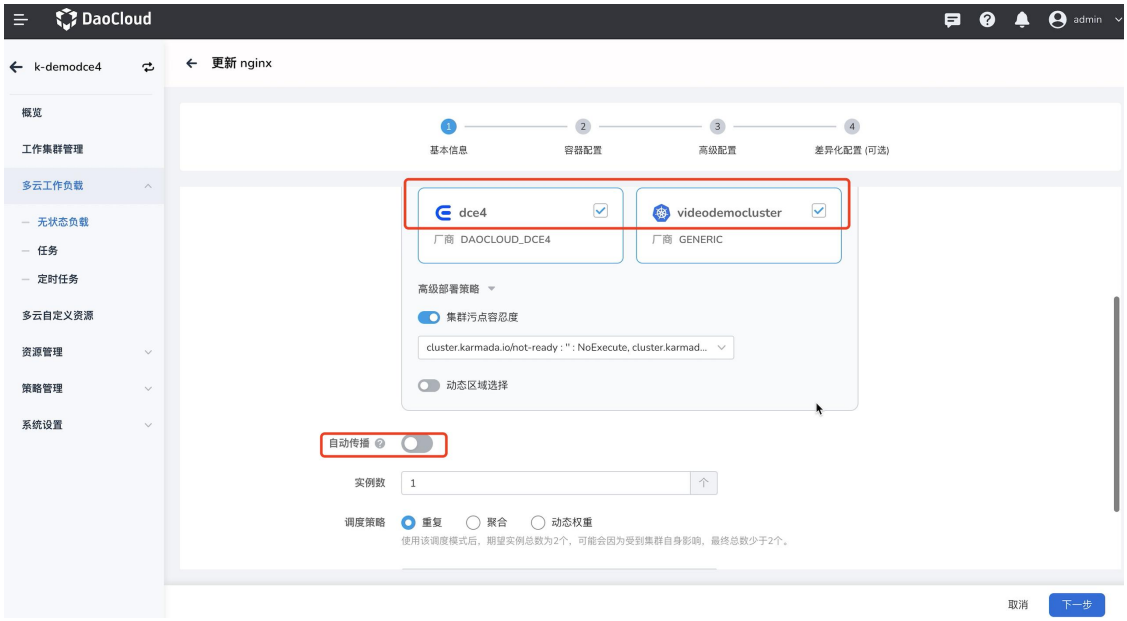


一键转移



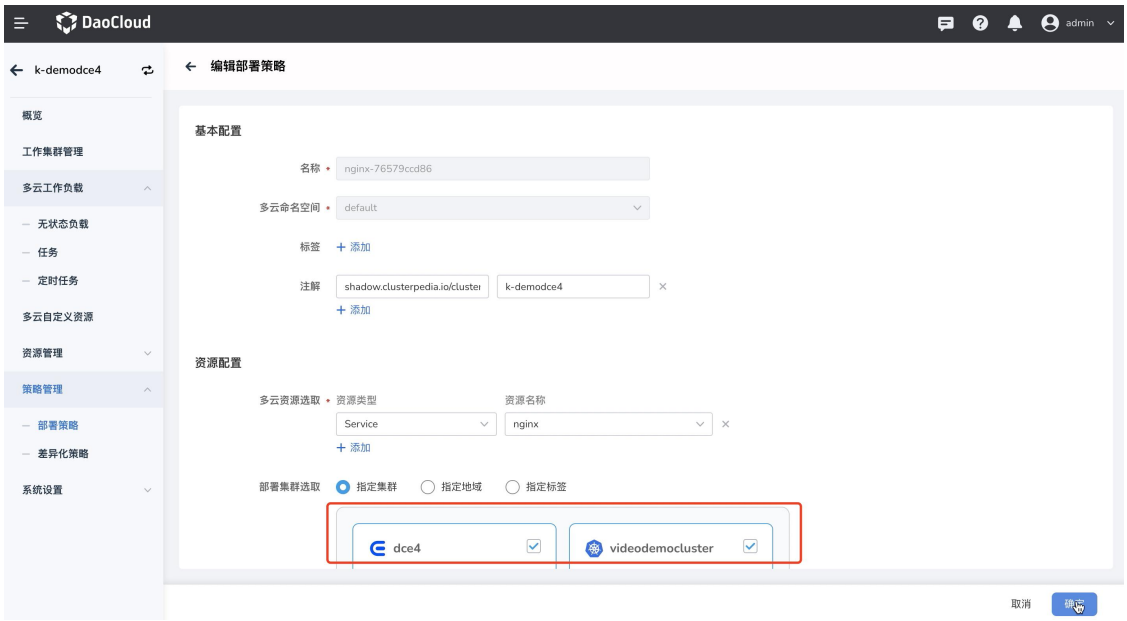
同步转换

3. 转换成功后点击更新，选择目标部署集群，并开启自动传播（将默认自动检测多云工作负载配置中依赖的 ConfigMap 与 Secret 等资源，并实现自动传播）。



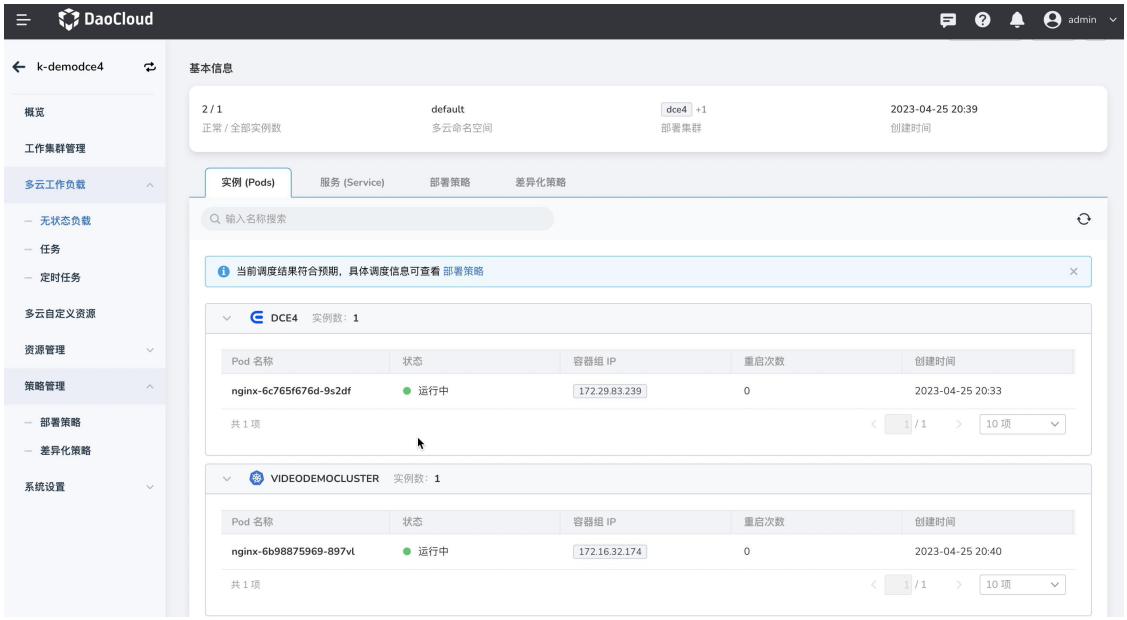
更新 nginx

4. 更新 service 的部署策略，选择部署集群。



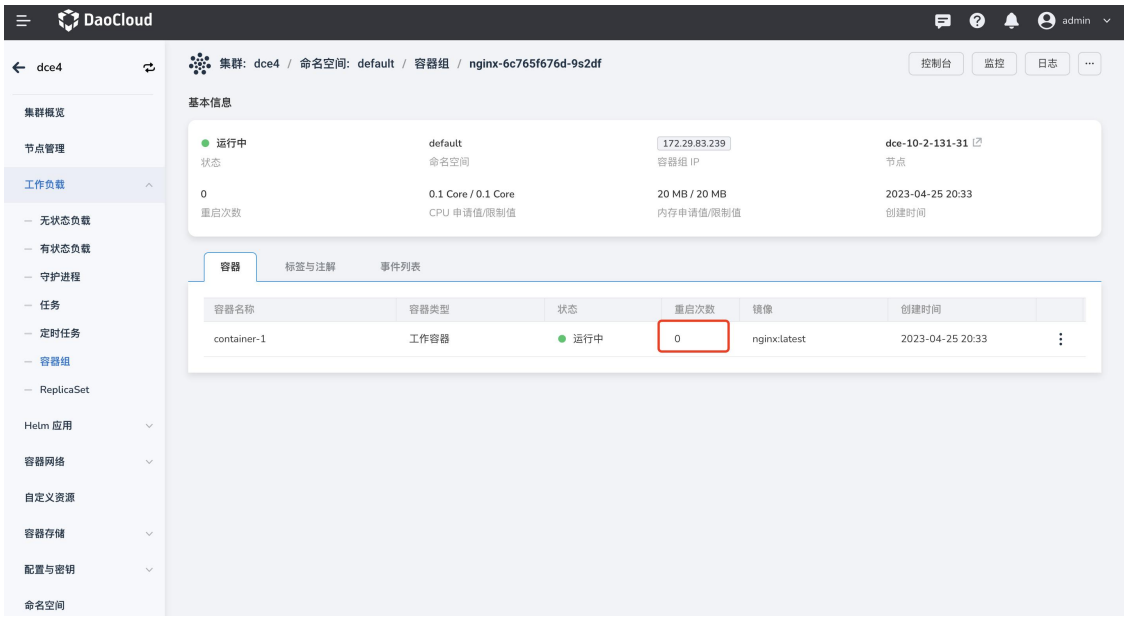
更新部署策略

5. 验证多云 nginx 是否运行成功：在两个集群内 Pod 都成功运行，并且可以正常访问。



验证是否成功

6. DCE4 集群内的工作负载 nginx 不受影响，应用不断服。



不断服

故障转移功能介绍

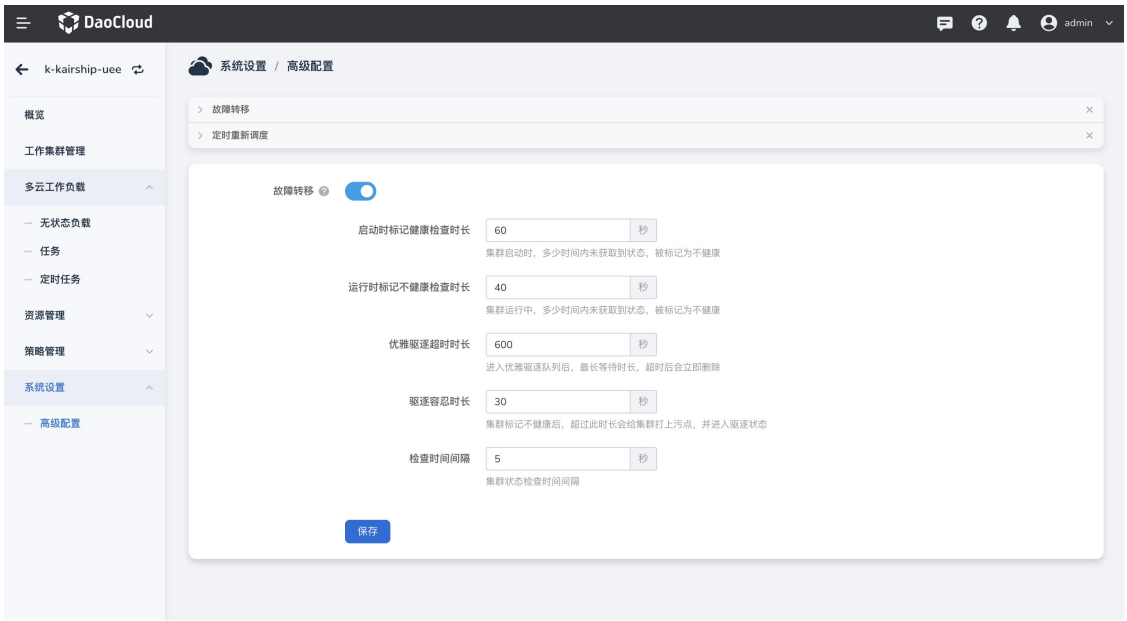
当某个集群发生故障时，该集群内的 Pod 副本会被自动迁移到其他可用集群，以保证服务稳定性。

前提条件

多云工作负载的调度策略只能选择聚合模式或者动态权重模式，此时故障转移功能才能生效。

开启故障转移

1. 进入多云编排模块，点击系统设置–>高级配置，故障转移可实现多个集群之间的副本调度，默认关闭，如有需要请开启。



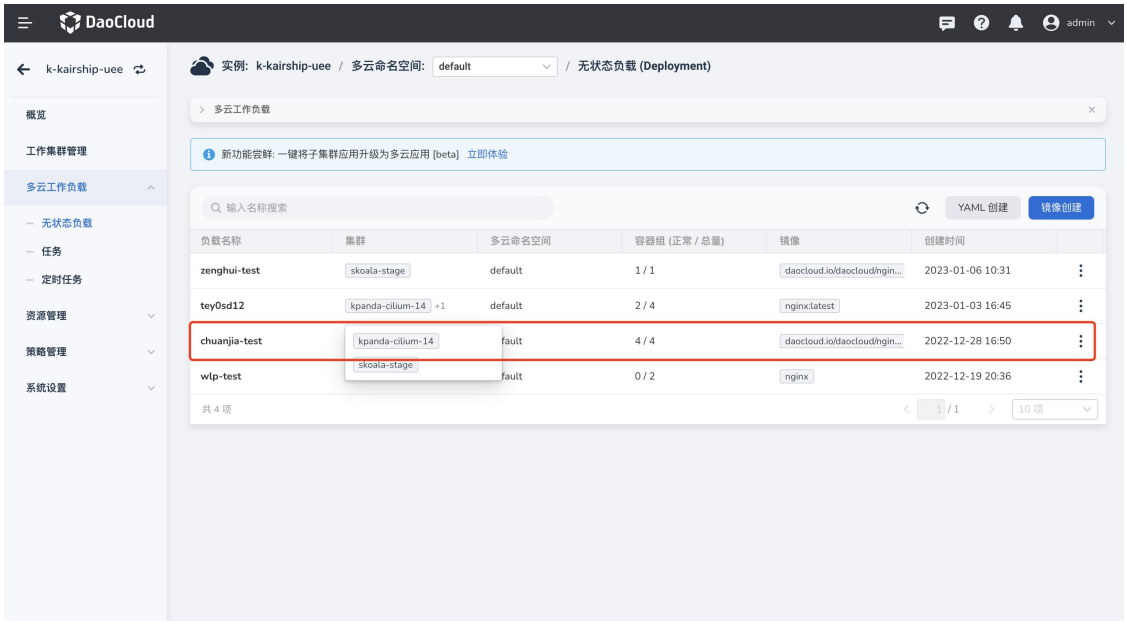
开启故障转移

2. 以下参数均针对集群，点击开启故障转移并保存。

参数	定义	描述	字段名 EN	字段名 ZH	默认值
ClusterMonitorPeriod	检查周期间隔	检查集群状态的时间间隔	Check Internal	检查时间间隔	60s
ClusterMonitorGracePeriod	运行中标记集群不健康检查时长	集群运行中，超过此配置时间未获取集群健康状态信息，将集群标记为不健康	The runtime marks the duration of an unhealthy check	运行时标记不健康检查时长	40s
ClusterStartupGracePeriod	启动时标记集群不健康的检查时长	集群启动时，超过此配置时间未获取集群健康状态信息，将集群标记为不健康	Mark health check duration at startup	启动时标记健康检查时长	600s
FailoverEvictionTimeout	驱逐容忍时长	集群被标记为不健康后，超过此时长会给集群打上污点，并进入驱逐状态 （集群会增加驱逐污点）	Eviction tolerance time	驱逐容忍时长	30s
ClusterTaintEvictionRetryFrequency	优雅驱逐超时时长	进入优雅驱逐队列后，最长等待时长，超时后会立即删除	Graceful ejection timeout duration	优雅驱逐超时时长	5s

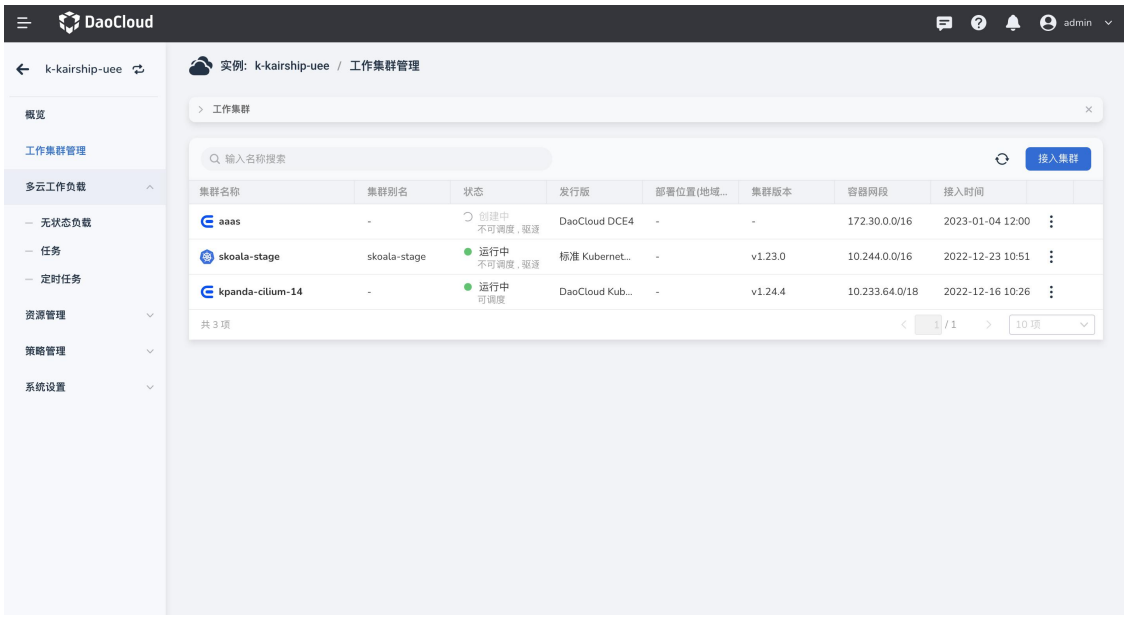
验证故障转移

1. 创建一个多云无状态负载，选择部署在多个集群上，调度策略选择聚合/动态权重模式.



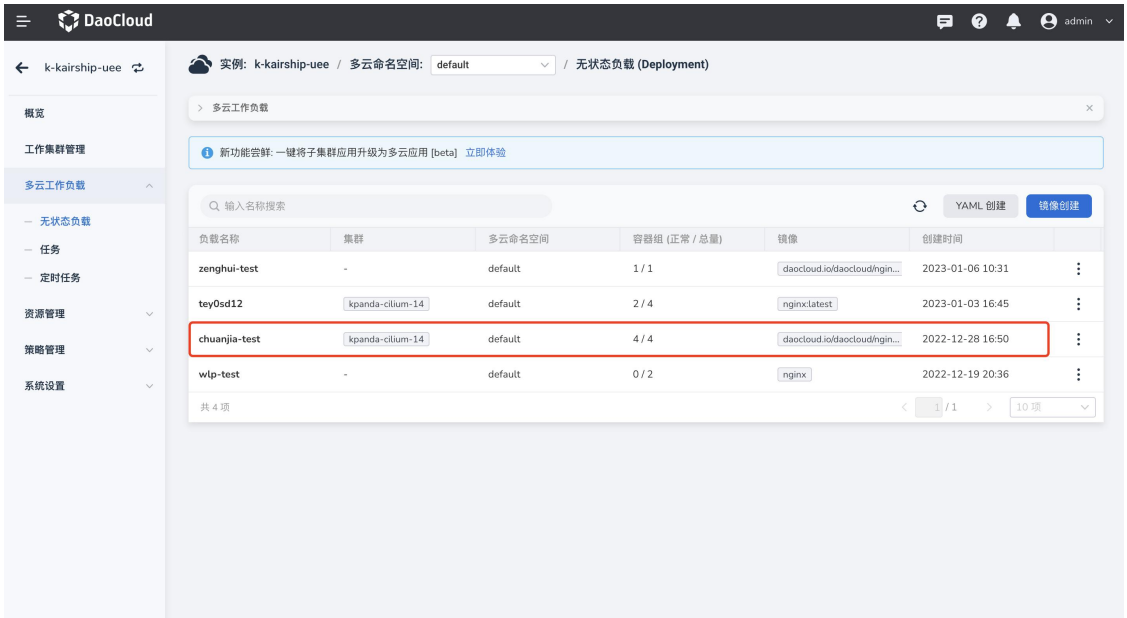
创建一个多云无状态负载

2. 若此时一个集群不健康并且在指定的时间范围内并未恢复，则将会为此集群打上污点，进入驱逐状态（此文档将手动为某一集群打上污点）



为集群打污点

3. 此时无状态负载的 Pod 将会根据剩余集群的资源等情况进行迁移。最终不健康（被打上污点）的集群内将不存在任何 Pod。



pod 迁移

优化故障转移延迟灵敏度

多云支持应用的跨集群自动故障转移，从而保障应用在多个集群部署的稳定性。影响故障转移的延时时效的主要有以下 2 个维度的指标，需要组合配置才能最终达到延迟灵敏度的效果。

1. 集群维度：标记集群为不健康的检查时长、集群驱逐容忍时长
2. 工作负载维度：集群污点容忍时长

Failover 特性介绍

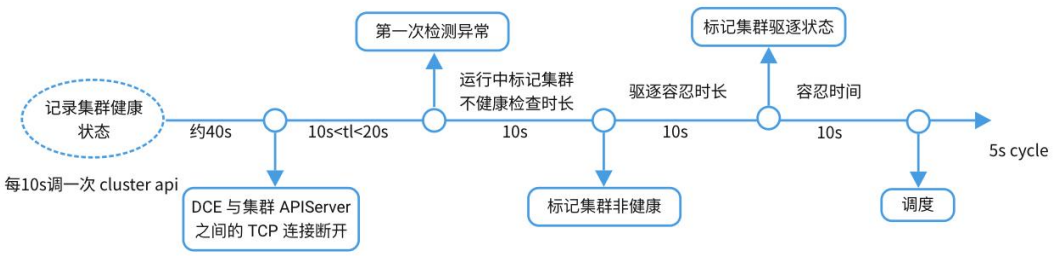
在 DCE 5.0 多云编排中启用了 failover 之后，提供了以下一些参数配置选项：

参数	定义	描述	字段名 EN	字段名 ZH	默认值
ClusterMonitorPeriod	检查周期间隔	检查集群状态的时间间隔	Check Internal	检查时间间隔	60s
ClusterMonitorGracePeriod	运行中标记集群不健康检查时长	集群运行中，超过此配置时间未获取集群健康状态信息，将集群标记为不健康	The runtime marks the duration of an unhealthy check	运行时标记不健康检查时长	40s
ClusterStartupGracePeriod	启动时标记集群不健康的检查时长	集群启动时，超过此配置时间未获取集群健康状态信息，将集群标记为不健康	Mark health check duration at startup	启动时标记健康检查时长	600s
FailoverEvictionTimeo	驱逐容忍时长	集群被标记为不健康后，超过此时长会给	Eviction tolerance time	驱逐容忍时长	30

参数	定义	描述	字段名 EN	字段名 ZH	默认值
ut		集群打上污点，并进入驱逐状态 （集群会增加驱逐污点）			s
ClusterTaintEvictionRetryFrequency	优雅驱逐超时时长	进入优雅驱逐队列后，最长等待时长，超时后会立即删除	Graceful ejection timeout duration	优雅驱逐超时时长	5s

工作负载被驱逐的时间线

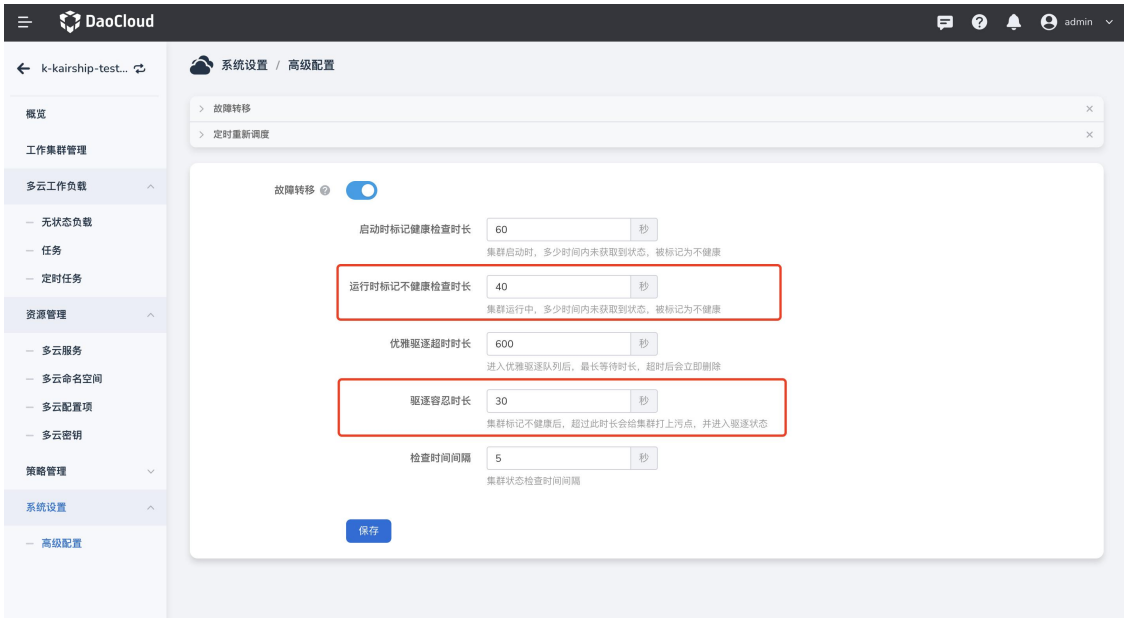
简单解释下图：我们规定 10s 调用一次集群的 API 用以记录集群的健康状态，当四次结果均为健康时，我们才认定集群是健康状态。 此时我们将 DCE 与集群 APIserver 之间的 TCP 断开 10s–20s 之内，若没有获取到集群的健康状态将认为集群异常， 指定时间内若集群没有恢复健康，将被标记为非健康状态，同时打上 NoSchedule 的污点，超过指定的驱逐容忍时长后，将被打上 NoExecute 的污点，最终被驱逐。



时间线

多云实例的优化配置

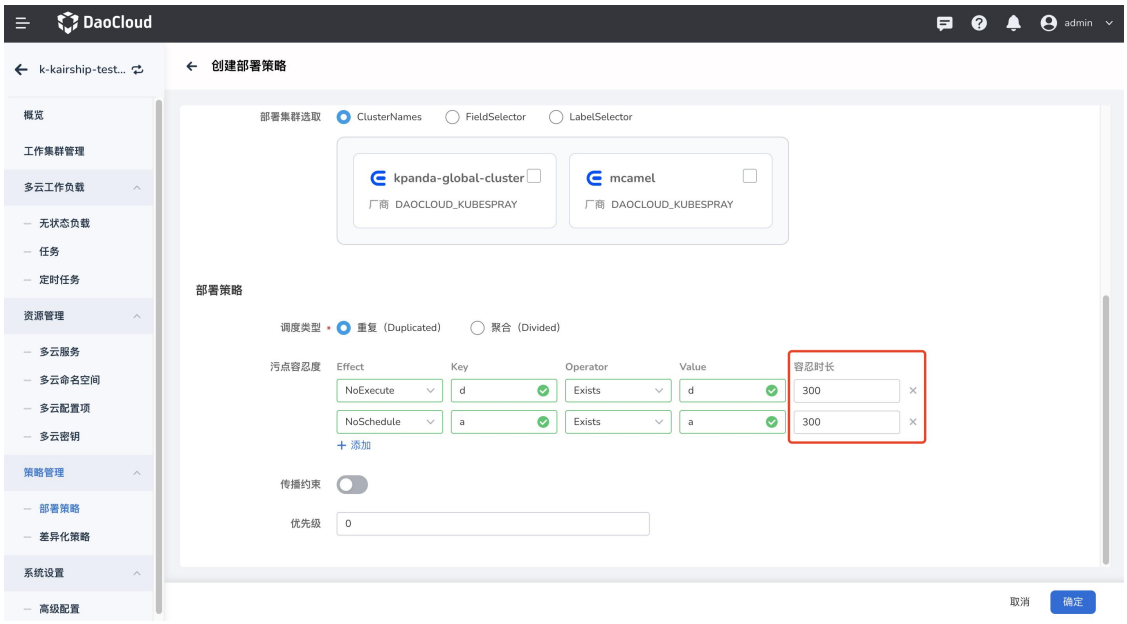
多云实例需要进入高级设置->故障转移部分，以下配置可参考上图填写参数信息。



故障转移

多云工作负载的配置优化

多云工作负载主要和其部署策略（PP）相关，需要在部署策略中修改对应的集群污点容忍时长。



工作负载

部署策略

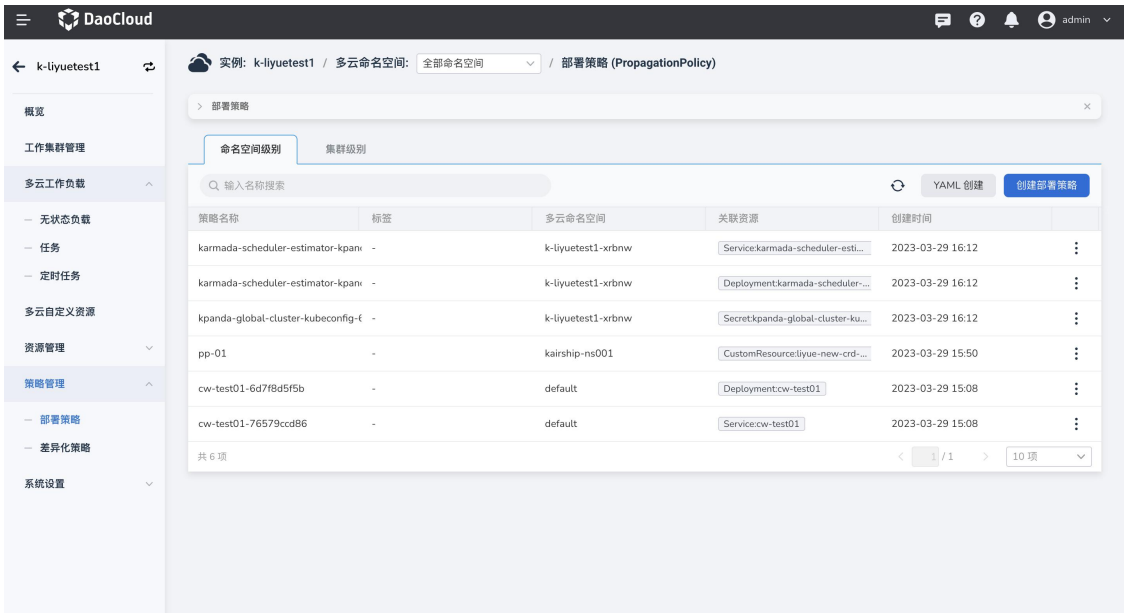
多云编排支持在界面上查看当前实例的部署策略列表及其关联的多云资源，支持以 YAML 和表单的方式创建和编辑部署策略信息，仅对空闲的部署策略提供删除按钮。

部署策略定义了在多云多集群中如何分发资源。部署策略（PropagationPolicy)分为命名空间级别和集群级别。

- 命名空间级别部署策略（PropagationPolicy ）表示将一组资源传播到一个或多个成员集群的策略， 只能在自己的命名空间传播资源。
- 集群级别部署策略（ClusterPropagationPolicy） 表示将一组资源传播到一个或多个成员集群的集群范围策略， 能够传播集群级别的资源和系统保留名称空间以外的任何名称空间中的资源。

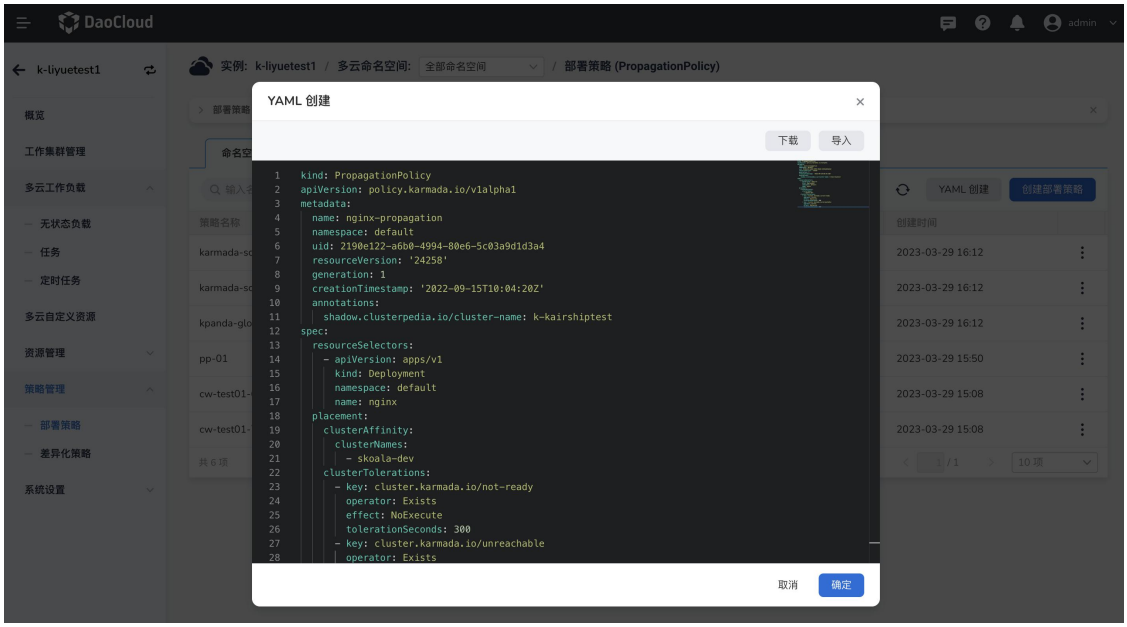
YAML 创建

1. 进入某一个多云实例后，在左侧导航栏中，点击策略管理 -> 部署策略->命名空间级别，点击右上角的 YAML 创建按钮。



命名空间级别部署策略

2. 在 YAML 创建页面中，输入正确的 YAML 语句后，点击确定。



yaml 创建

3. 返回部署策略列表，新创建的默认位于第一个。点击列表右侧的 ⋮，可以编辑 YAML 和执行删除操作。

YAML 示例

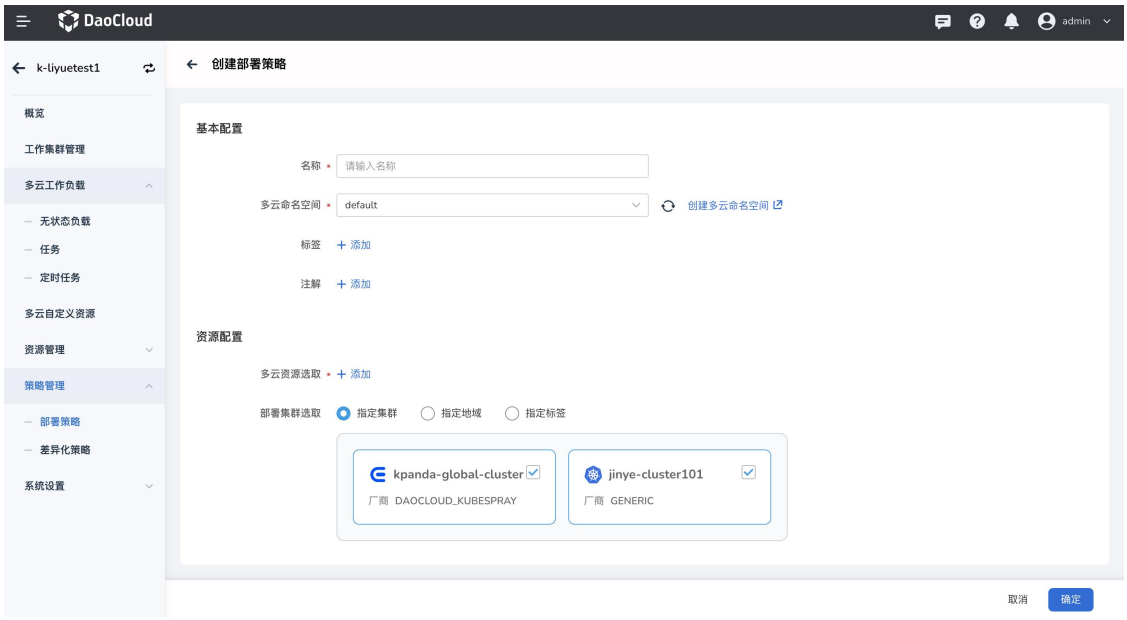
此处列出一个部署策略的 YAML 示例，您稍加修改就可以使用。

yaml title="YAML example" kind: PropagationPolicy apiVersion: policy.karmada.io/v1alpha1 metadata: name: nginx-propagation namespace: default uid: 2190e122-a6b0-4994-80e6-5c03a9d1d3a4 resourceVersion: '24258' generation: 1 creationTimestamp: '2022-09-15T10:04:20Z' annotations: shadow.clusterpedia.io/cluster-name: k-kairshiptest spec: resourceSelectors: - apiVersion: apps/v1 kind: Deployment namespace: default name: nginx placement: clusterAffinity: clusterNames: - skoala-dev clusterTolerations: - key: cluster.karmada.io/not-ready operator: Exists effect: NoExecute tolerationSeconds: 300 - key: cluster.karmada.io/unreachable operator: Exists effect: NoExecute tolerationSeconds: 300

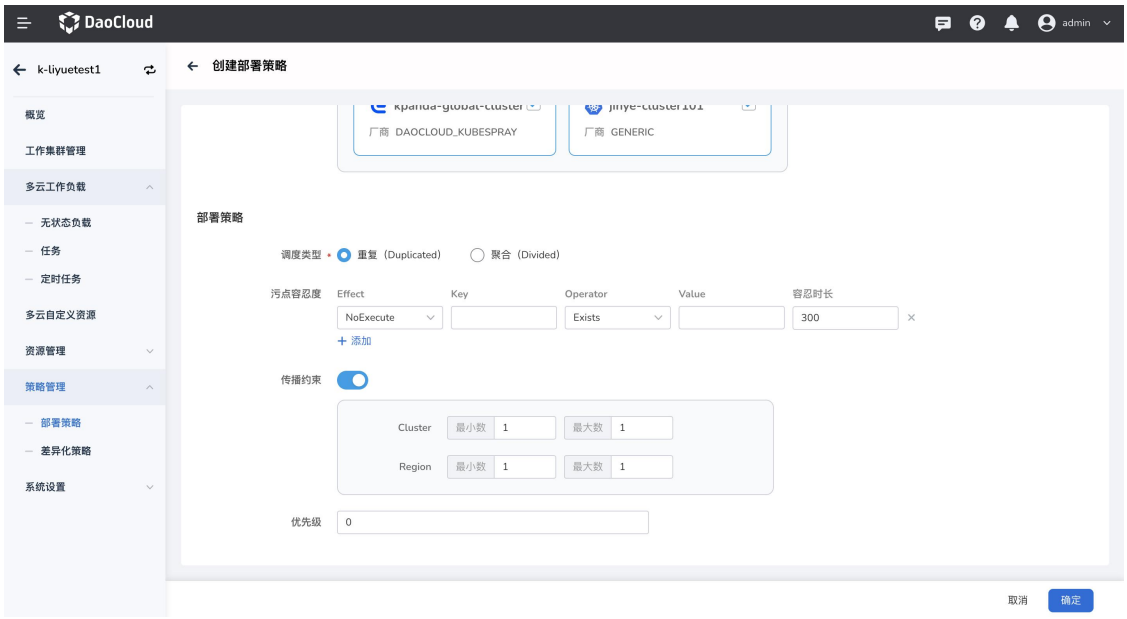
表单创建

1. 参照以下步骤创建一个部署策略。

    - 基本配置：填写名称、选择多云命名空间，增加标签注解
    - 配置资源：选择多云资源和目标部署集群
    - 部署策略：调度类型、污点容忍度、可以选择是否开启传播约束、支持填写优先级（创建资源时也可以同时创建部署策略，当在本页面也创建了部署策略时，可以根据优先级的高低来判断该资源使用哪个部署策略）

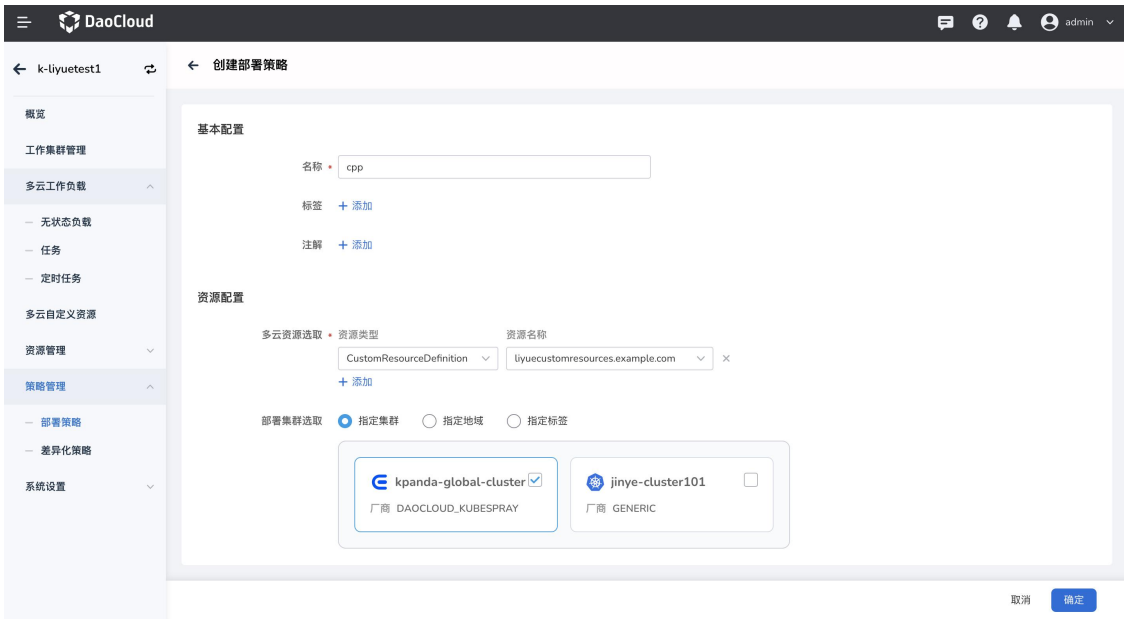


表单创建命名空间级别



表单创建

2. 表单创建集群级别的部署策略，不需要选择命名空间。



表单创建集群级别

!!! note

若要删除一个部署策略，需要先移除该策略相关的工作负载，删除之后该策略有关的信息都将被删除，请谨慎操作。

差异化策略

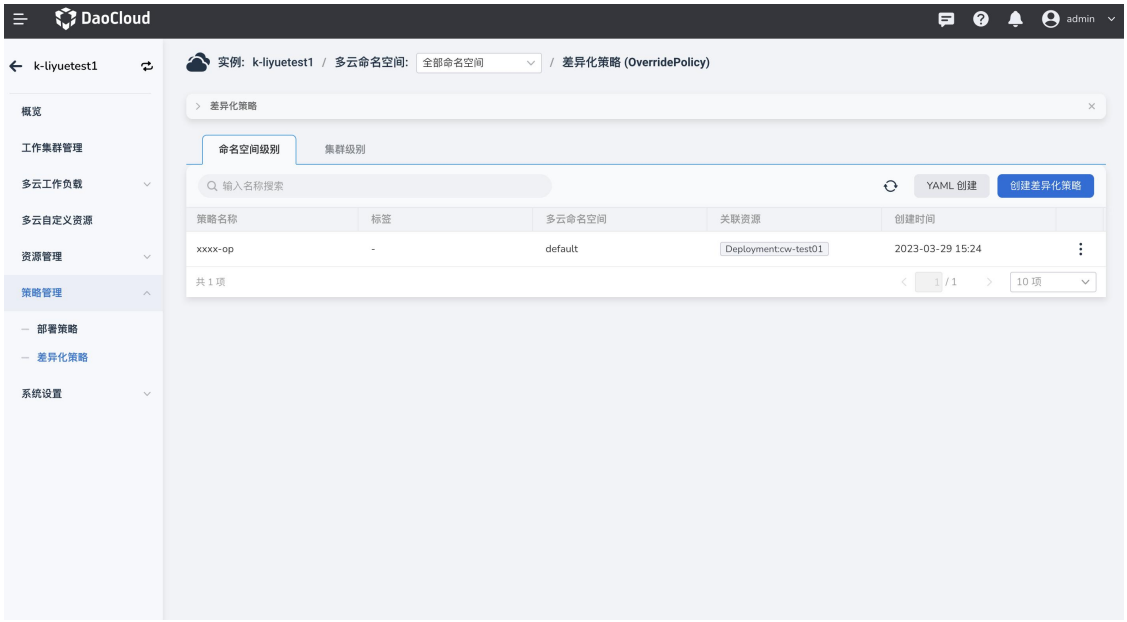
差异化策略（Override Policy）的作用是，定义在下发到不同集群中的配置，可以是不一样的，例如不同的集群所对应的镜像仓库的地址是不一样的，那就需要设置在不同集群中的工作负载的镜像地址不同，例如在不同的环境下，需要设置不同的环境变量等。差异化策略（Override Policy）的作用时机是在部署策略（Propagation Policy）之后，以及在真正在下发到集群之前。

多云编排支持差异化策略，支持在界面上查看当前实例的差异化策略列表及其关联的多云资源，并且可以通过表单和 YAML 两种方式创建和更新差异化策略的信息。差异化策略（Override Policy）分为命名空间级别和集群级别。

YAML 创建

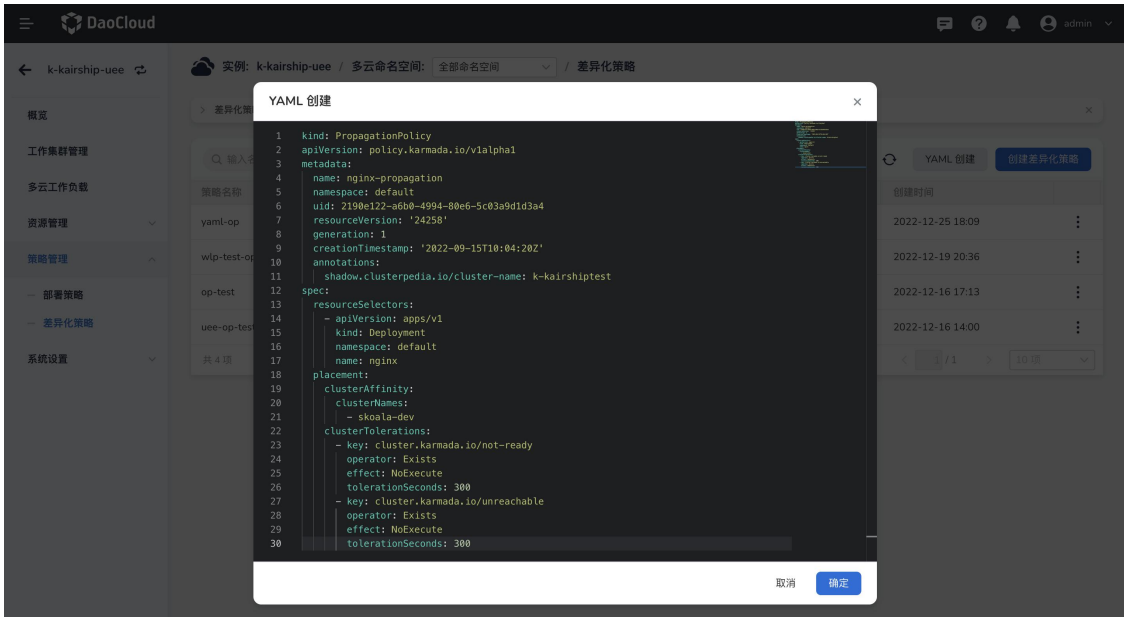
参照以下步骤使用 YAML 创建一个差异化策略。

1. 进入某一个多云实例后，在左侧导航栏中，点击策略管理 -> 差异化策略，点击 YAML 创建按钮。



命名空间级别

2. 在 YAML 创建页面中，输入正确的 YAML 语句后，点击确定。



yaml 创建

3. 返回部署策略列表，新创建的默认位于第一个。点击列表右侧的 ⋮，可以编辑 YAML 和执行删除操作。

!!! note

若要删除一个部署策略，需要先移除该策略相关的工作负载，删除之后该策略有关的信息都将被删除，请谨慎操作。

YAML 示例

此处列出一个差异化策略的 YAML 示例，您稍加修改就可以使用。

```yaml
kind: OverridePolicy
apiVersion: policy.karmada.io/v1alpha1
metadata:
  name: nginx-op
  namespace: default
  uid: 09f89bc4-c6bf-47b3-81bf-9e494b831aac
  resourceVersion: '856977'
  generation: 1
  creationTimestamp: '2022-09-21T10:30:40Z'
  labels:
    a: b
  annotations:
    shadow.clusterpedia.io/cluster-name: k-kairshiptest
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      namespace: default
      name: nginx
  overriders: {}
```

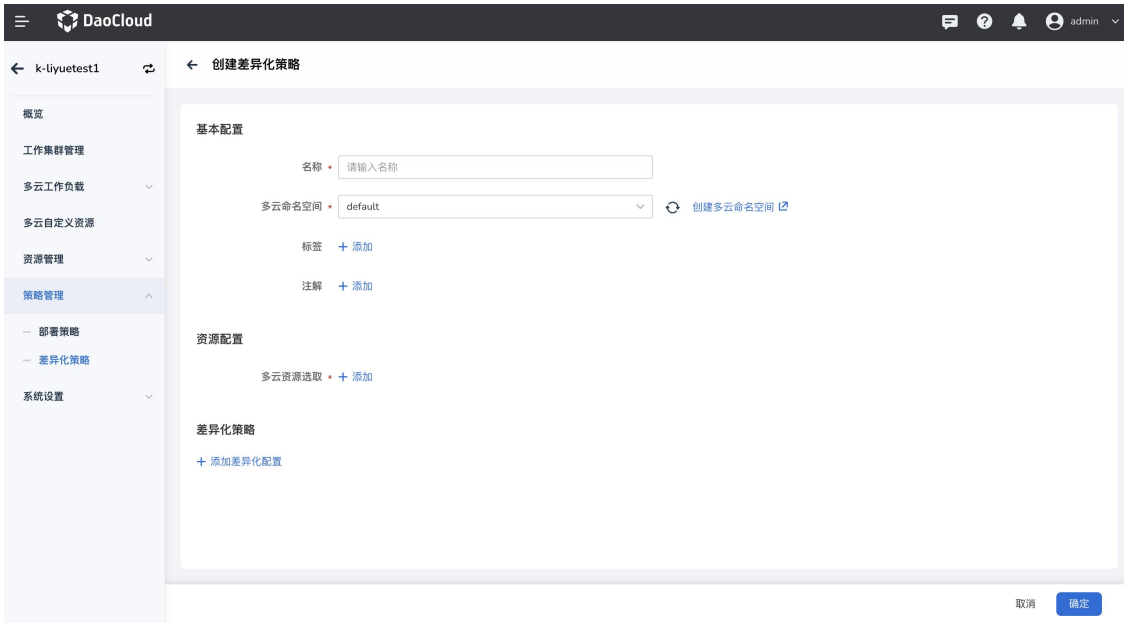
表单创建

参照以下步骤创建一个差异化策略。

1. 进入命名空间级别的菜单，点击创建差异化策略按钮。

2. 进入表单创建页面，创建一个完整的差异化策略需要配置基础配置、资源配置、差异化策略三部分。

   - 基本配置：包括名称、多云命名空间、标签和注解。其中名称和多云命名空间为必填项。
   - 资源配置：支持选取所选多云命名空间下的多云资源进行差异化配置，多云资源也支持多选。
   - 差异化策略：支持添加镜像/运行参数/运行命令/自定义等的差异化配置。



表单创建

3. 填写完成后点击确定则创建成功，支持对一条差异化策略进行 YAML 更新、表单更新、删除操作。

差异化策略详解

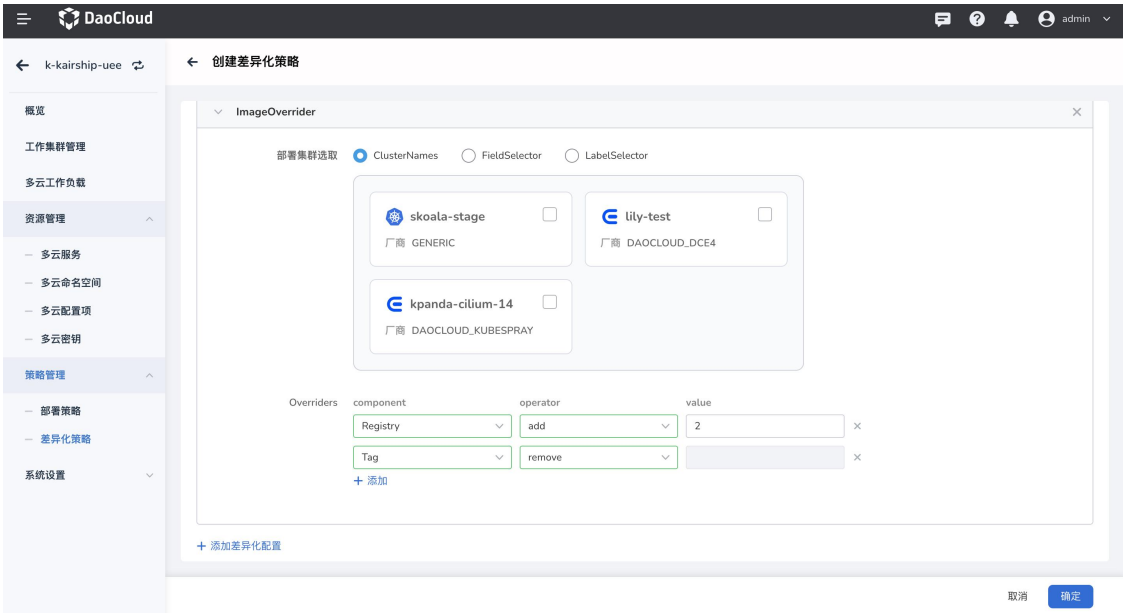
接下来将详细介绍差异化策略，共分为六种，且支持添加多条差异化配置：

1. 选择 ImageOverrider：镜像的差异化配置。

1. 选择部署集群

   支持 ClusterNames（集群名称）、FieldSelector（区域）、LabelSelector（标签）三种方式选择部署集群。

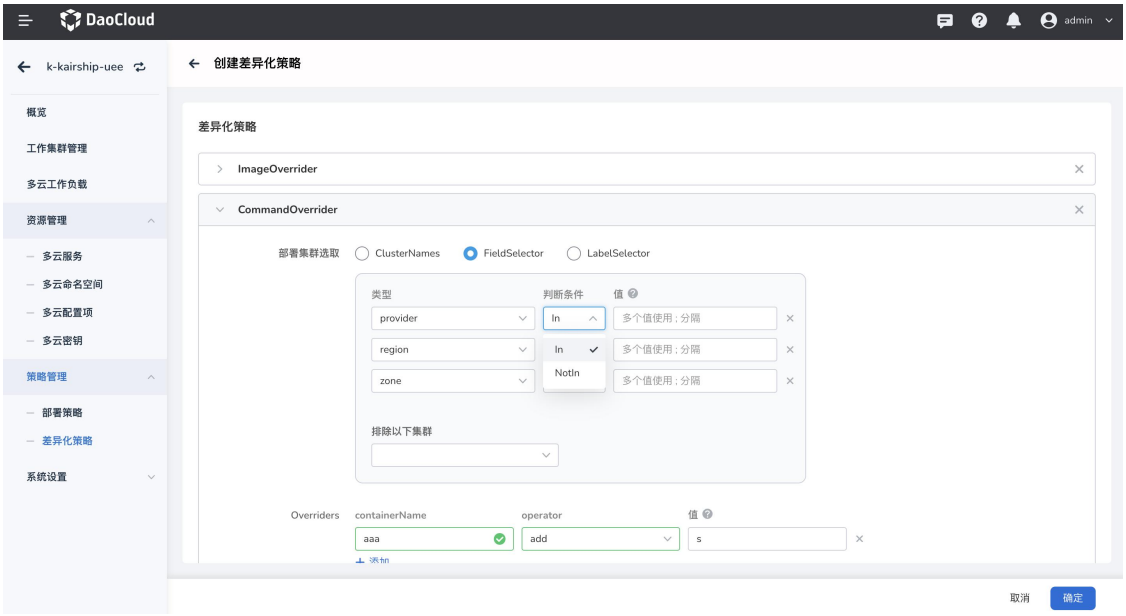
   === ClusterNames：直接选择指定的集群名称



   指定集群名称

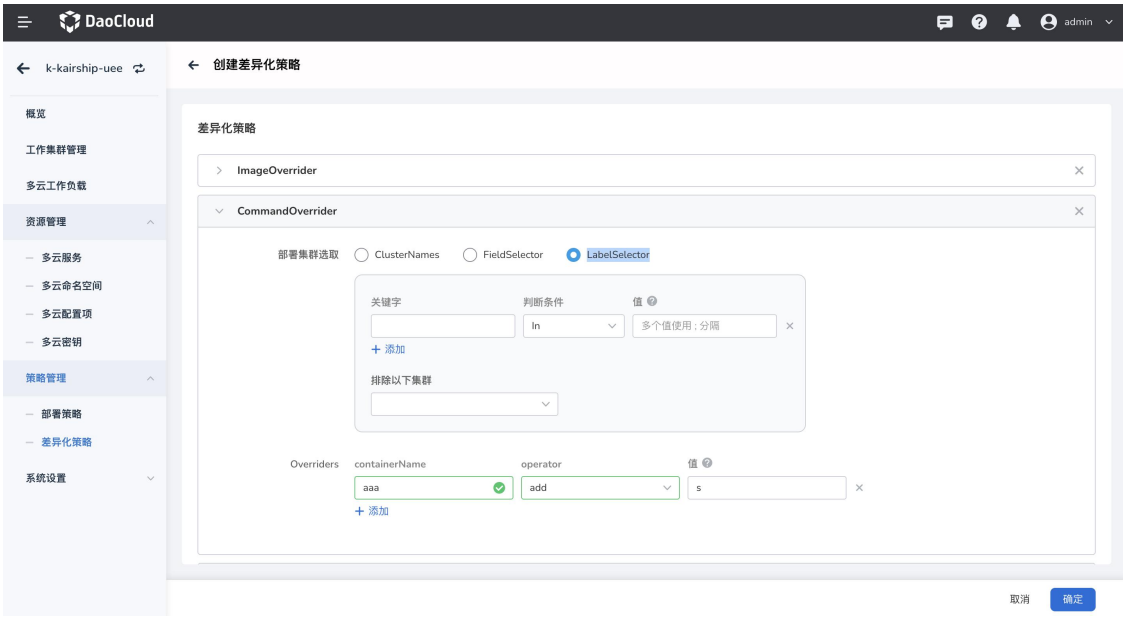
   === FieldSelector：通过字段来选择集群，包括：

   - provider（提供集群基础设施的服务商）
   - region（区域：是可用区的集合，指可以在其中托管资源的特定地理位置，例如北京、上海、深圳、成都等。）
   - zone（可用区：是区域内的部署区域，例如上海可用区 1、上海可用区 2……）
   - 还支持从已选集群中排除某指定集群。



从已选集群排除

=== LabelSelector：通过标签来选择集群，支持自定义标签，同时也支持排除指定集群。



标签

2. 对选中集群进行镜像的差异化配置

字段	必填	描述	示例
Component	是	镜像组成成分	Registry、Repository、Tag
Operator	是	对镜像进行的操作	add、remove、replace
Value	否	当 Operator 为 add 或 replace 时不能为空，默认为空；当 operator 为 remove 时不填。	

2. 当选择 ArgsOverrider：运行参数的差异化配置。
    1. 选择部署集群
    2. 对选中集群进行运行参数的差异化配置

字段	必填	描述	示例
ContainerName	是	容器名	
Operator	是	应用在 args 上的操作	add、remove
Value	否	应用在 args 上的值，当 operator 为 add 时该值 append 到 args；当 operator 为 remove 时，该值从 args 移除；如果该值为空，args 维持原状。	

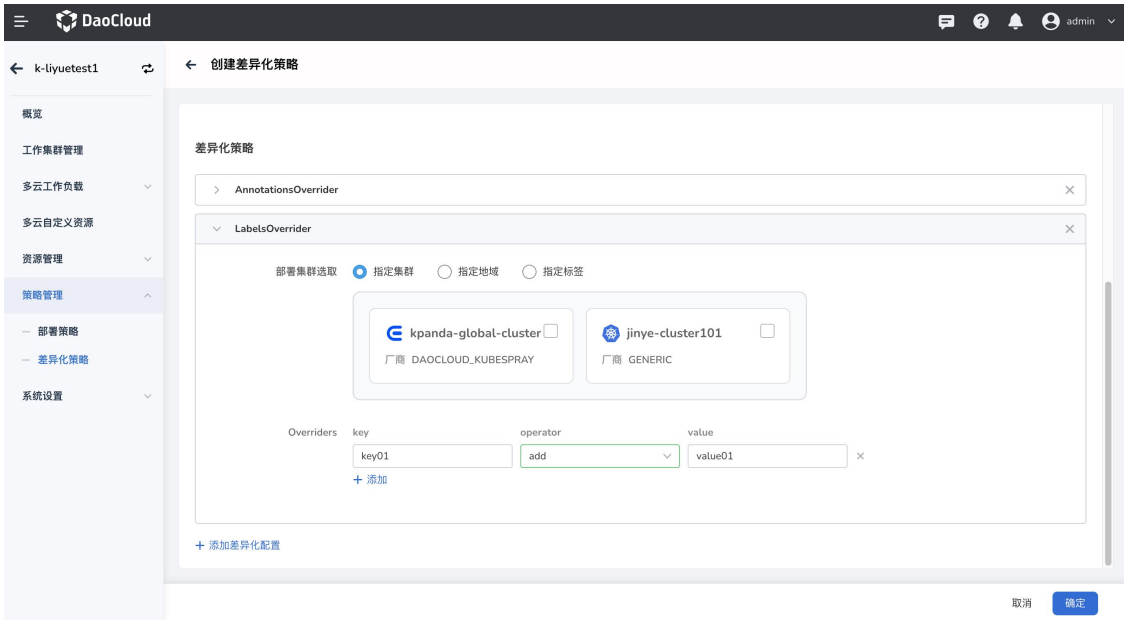
3. 当选择 CommandOverrider：运行命令的差异化配置。
    1. 选择部署集群
    2. 对选中集群进行运行命令的差异化配置

字段	必填	描述	示例
ContainerName	是	容器名	
Operator	是	应用在 command 上的操作	add、remove
Value	否	应用在 command 上的值，当 operator 为 add 时该值 append 到 command；当 operator 为 remove 时，该值从 command 移除；如果该值为空，command 维持原状。	可以添加单个或多个值，多值使用 ；划分

4. 当选择 PlainText：自定义的差异化配置。
    1. 选择部署集群
    2. 对选中集群进行运行命令的差异化配置

字段	必填	描述	示例
Path	是	目标字段的路径	
Operator	是	对目标字段操作类型	add、remove、replace
Value	否	应用在目标字段的值，当 Operator 为 remove 时，此字段必须为空	

5. 当选择 LabelsOverrider：标签的差异化配置。

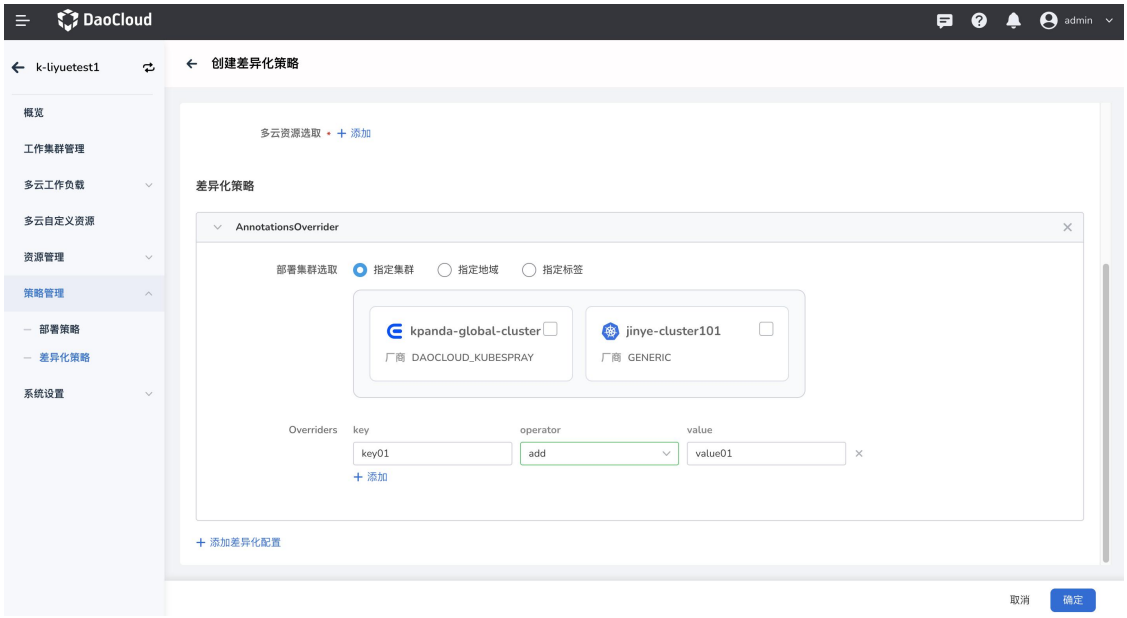


标签差异化

1. 选择部署集群
2. 对选中集群进行运行标签的差异化配置

字段	必填	描述	示例
Key	是	标签的键	
Operator	是	对目标字段操作类型	add、remove、replace
Value	否	标签的值	

6. 当选择 AnnotationsOverrider：注解的差异化配置。



注解差异化

1. 选择部署集群
2. 对选中集群进行运行注解的差异化配置

字段	必填	描述	示例
Key	是	注解的键	
Operator	是	对目标字段操作类型	add、remove、replace
Value	否	注解的值	

适用场景

DCE 5.0 集成社区最优秀的众多技术，内置依赖项数以万计，经海量调测后熔于一炉，铸就新一代 PaaS 服务平台，在各类使用场景中都游刃有余。

===“生产级容器管理”

用户在运维团队需要承接数十个至上百个集群运行维护任务，且集群网络规划需满足用户传统网络监管要求。

涉及模块：容器管理、容器网络、容器集群生命周期管理

方案优势：提供集群从部署、升级、证书变更、配置变更、回收等全集群生命周期管理管理能力。 尽可能复用当前企业环境中网络基础设施，针对不同的环境，可以实施的最佳方案有：MacVLAN 网络方案、SR-IOV 智能网卡加速方案、SpiderPool 云原生 IPAM 方案、Clilum eBPF 网络加速方案、Underlay 和 Overlay 协同网络方案。

通过自主开源的 Clusterpedia 统一控制平面管理所有集群及负载信息，兼容标准 Kubernetes 集群接入，突破 Kubernetes API 性能瓶颈，支持上千人同时使用。

===“多云编排”

用户中多集群、多云部署成为常态，希望能完成多云统一发布，并且拥有跨云容灾备份的能力

涉及模块：容器管理、多云编排、容器网络、边云协同、信创云

方案优势：创新性技术完成跨云容灾，跨云资源检索并发性能高，结合容器平台的能力适配边缘、信创场景。

===“融合微服务”

用户应用架构决定采用微服务架构或已经采用微服务，希望得到全方位的微服务框架等技术支持及运维兜底能力，或希望引入服务网格技术，并且在技术更迭的过程中实现平滑过度。

涉及模块：容器管理、容器网络、微服务引擎、服务网格、可观测性、应用工作台

方案优势：无缝融合以 SpringCloud、Dubbo 为代表的初代微服务与 Istio 服务网格代表的新一代微服务技术，完成从开发、部署、接入、对外、观测、运维全生命周期微服务管理能力。

===“云边协同”

用户按照云、边、端方案设计边缘协同方案，边缘端为通用算力平台，且边缘端有较强的算力需求。边缘端支持三种部署模式：边缘节点、边缘集群模式，边缘集群叠加边缘节点四层架构模式。

涉及模块：容器管理、容器网络、容器集群生命周期管理、边缘节点

方案优势：云端统一管控所有边缘节点、集群信息，在传统的云边端三层模式基础上，针对强边缘算力需求，增加边缘集群迭代边缘节点的方案，形成四层云边协同方案。

===“信创云原生”

用户有信创需求，对底层基础设施及操作系统有特定要求，例如处理器：Loongson 龙芯、海光、飞腾、鲲鹏、intel；操作系统：麒麟、统信 UOS、OpenEuler 等。

涉及模块：容器管理、容器网络、容器集群生命周期管理

方案优势：北向支持国产芯片及服务器，南向支持容器内信创操作系统及信创应用生态体系。

===“应用交付”

用户大规模采用云原生技术，并且期望规范化、流程化结合 DevOps 理念将云原生技术推广至更广泛的应用项目组。

涉及模块：容器管理、应用工作台、容器网络、镜像仓库、边云协同、信创云

方案优势：支持层级多租户体系，无缝适配用户组织架构规划资源分配。 CI/CD 流水线能力自动化完成应用构建、部署工作。创新性引入 GitOps、渐进式交付的能力体系，帮助应用进行更细致的交付管理能力。

===“云原生可观测”

用户对运行应用观测能力较弱，希望能轻量级或无改造接入就能完成观测接入，完成全方位的应用运行观测（日志、指标、链路）。

涉及模块：容器管理、可观测性、容器网络、边云协同、信创云

方案优势：观测数据统一汇总，一个控制面板就能查询到所有集群及负载观测数据，并且深入支持微服务架构、服务网格、网络 EBPF 观测能力。

===“数据服务”

用户应用架构依赖主流中间件能力，希望能统一运行维护中间件，并且得到较为专业的关于中间件规划、运维的支持能力。

涉及模块：容器管理、容器网络、容器本地存储、云原生中间件、边云协同、信创云

方案优势：专为有状态应用设计的云原生本地存储能力，统一平台管理云原生中间件，提供多租户、部署、观测、备份、运维的全生命周期的中间件管理能力，结合容器平台的能力适配边缘、信创场景。

===“应用商店”

用户希望获得专属场景开箱即用的云原生应用软件能力。

涉及模块：容器管理、应用商店

方案优势：收录来自生态伙伴十大领域的软件产品，面向企业实际业务需求提供完整软件堆栈，可轻松查找、测试、和部署在 DaoCloud Enterprise 上运行的消息中间件、数据中间件、低代码/无代码应用等。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验 DCE 5.0 场景化视频

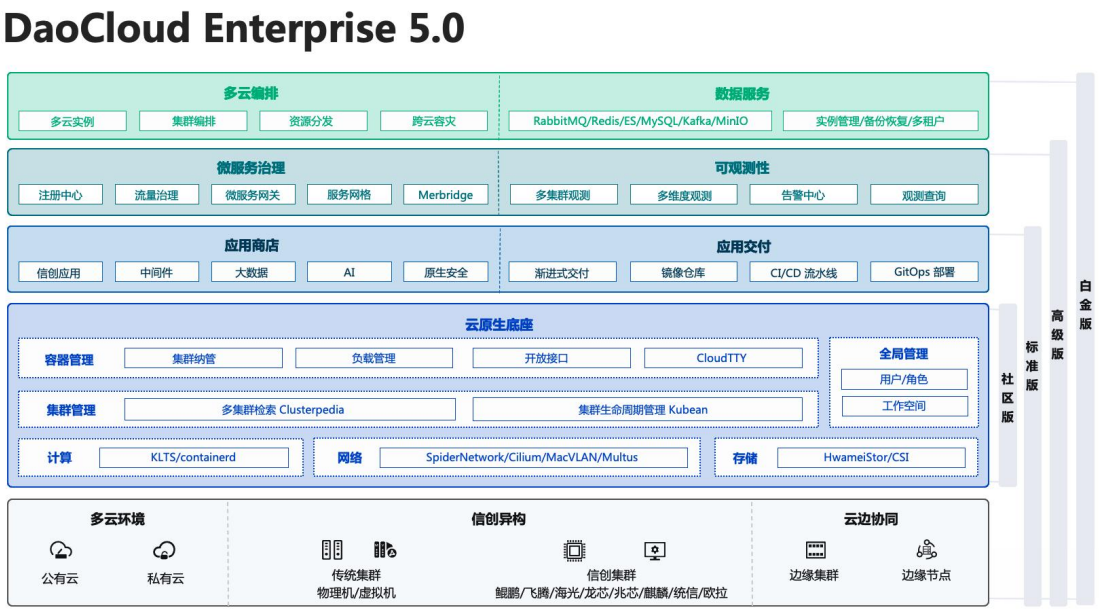
DaoCloud Enterprise 5.0

DaoCloud Enterprise 5.0 (DCE 5.0) 是一款高性能、可扩展的云原生操作系统。它能够在任何基础设施和任意环境中提供一致、稳定的体验，支持异构云、边缘云和多云编排。 DCE 5.0 集成了最新的服务网格和微服务技术，能够跟踪每一个流量的生发始终， 帮助您洞察集群、节点、应用和服务的详细指标，并通过动态仪表盘和拓扑大图可视化掌握应用健康状态。

DCE 5.0 原生支持 DevOps 开发运维模式，可以实现应用交付的全流程标准化和自动化，并集成各类精选数据库和中间件，使运维治理更加高效。 各个产品模块独立解耦，支持灵活升级，对业务没有影响，并且能够与众多云原生生态产品对接，提供完整的解决方案体系。 它经过了近千家行业客户的生产场景检验，构建了坚实、可靠的数字底座，帮助企业定义数字边界，释放云原生生产力。

*[DCE]: DaoCloud Enterprise 云原生操作系统的简称

- :fontawesome-solid-jet-fighter-up: **安装** 详尽的安装教程
- :octicons-container-16: **容器管理** 集群/节点/负载等基础设施
- :fontawesome-solid-user-group: **全局管理** 设置登录/访问权限/外观
- :material-monitor-dashboard: **可观测性** 一站式图形化仪表盘
- :material-microsoft-azure-devops: **应用工作台** CI/CD 流水线
- :material-cloud-check: **多云编排** 多云实例/负载/策略
- :material-engine: **微服务引擎** 微服务治理中心和网关
- :material-table-refresh: **服务网格** 非侵入式服务治理
- :material-middleware: **中间件** ES, Kafka, MinIO, MySQL 等
- :material-warehouse: **镜像仓库** 镜像集成和托管服务
- :material-dot-net: **网络** 多 CNI 融合方案
- :floppy_disk: **存储** 容器化存储综合方案



模块图

DCE 5.0 云原生操作系统提供了 9 大能力，自由搭配各种模块，可以应对海量应用场景。

这些模块就像乐高搭积木一样，糅合社区最优秀的几十种开源技术，经过众多辩证选型、攻坚克难、编码调试、海量测试，“十年磨一剑，一朝试锋芒”，全新搭建的新一代容器化平台能够满足企业上云的各类场景需求。

===“多云编排”

支持多云和混合云的统一集中管理，提供跨云资源检索及跨云的应用部署、发布和运维能力，实现多云应用高效管控，提供基于集群资源的应用弹性扩缩，实现全局负载均衡，具备故障恢复能力，有效解决多云应用灾备问题，助力企业构建多云、混合云的数字基础设施。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[多云编排](../kairship/intro/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)、信创异构

![多云](https://docs.daocloud.io/daocloud-docs-images/docs/images/01multi-cloud.png)

===“数据中间件服务”

专为有状态应用设计的云原生本地存储能力，满足中间件高 I/O 的存储需求，提升运维管理效率。精选各类数据库、分布式消息和日志检索等中间件，
提供多租户、部署、观测、备份、运维操作等全生命周期的中间件管理能力，实现数据服务的自助化申请、弹性扩展、高并发处理和稳定高可用。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)、[精选中间件](../middleware/index.md)

![数据](https://docs.daocloud.io/daocloud-docs-images/docs/images/02data.png)

===“微服务治理”

提供非侵入式流量治理功能，支持无感接入传统微服务、云原生微服务和开源微服务框架，实现企业现有微服务体系及新旧微服务体系的融合治理，
支持微服务从开发、部署、接入、观测、运维的全生命周期管理，提供高性能云原生微服务网关，保证微服务应用的连续可用性；引入自主开源的 eBPF 网格加速技术，全面提高流量转发效率。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[微服务引擎](../skoala/intro/index.md)、[服务网格](../mspider/intro/index.md)、[可观测性](../insight/intro/index.md)、[应用工作台](../amamba/intro/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)

![微服务](https://docs.daocloud.io/daocloud-docs-images/docs/images/03msgov.png)

===“可观测性”

基于日志、链路、指标、eBPF 等技术手段，全面采集服务数据，深入获取请求链路信息，动态观测、多维度掌控集群、节点、应用和服务的实时变化，
通过统一控制面实现所有集群及负载观测数据的查询，引入拓扑分析技术可视化掌握应用健康状态，实现秒级故障定位。

*[eBPF]: Extended Berkeley Packet Filter，这是 Linux 内核自带的组件，可以在内核中运行沙盒程序

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[可观测性](../insight/intro/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)

![观测](https://docs.daocloud.io/daocloud-docs-images/docs/images/04insight.png)

===“应用商店”

收录来自大数据、AI、中间件等十大领域生态伙伴的软件产品，实现生态技术、产品、运营服务等能力的整合，提供开箱即用的生态应用软件，面向企业实际业务需求，打造完整的解决方案体系。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)、应用商店、产品生态

===“应用交付”

通过一致性可推广的应用交付流程实现自助式上云，支持柔性租户体系，动态适配用户组织架构规划和实时资源分配，基于云原生化的 CI/CD 流水线，集成丰富的工具链并支持流水线高效并发执行流转，
自动化完成应用的构建、部署，创新性引入 Gitops、渐进式交付能力体系，实现应用更精细的管理运维。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[应用工作台](../amamba/intro/index.md)、[镜像仓库](../kangaroo/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)

![应用交付](https://docs.daocloud.io/daocloud-docs-images/docs/images/06appdeliv.png)

===“信创异构”

采用信创云原生技术架构，兼容国产芯片及服务器，支持信创操作系统及信创应用生态体系，屏蔽底层异构基础设施的复杂性，把传统操作系统从需要长期积累的软件生态兼容适配中解放出来，
实现混合异构集群的灵活调度，保证信创应用运行环境的稳定高可靠，助力信创进程进一步提速。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[集群生命周期管理](../community/kubean.md)、[信创中间件](../middleware/index.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)

![信创](https://docs.daocloud.io/daocloud-docs-images/docs/images/08xinchuan.png)

===“云边协同”

将云原生能力延伸至边缘，采用边缘集群、边缘节点模式，将数据中心算力下移，端设备算力上移，统一管控和调度离散、异构的计算资源，解决在海量边、端设备上统一完成大规模应用交付、运维、管控的诉求，实现云边的真正一体化。

涉及的模块：[云边协同](../kant/intro/index.md)、[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[集群生命周期管理](../community/kubean.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)、弱网集群

![云边](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/images/09cloud-edge.png)

===“云原生底座”

提供云原生计算、[网络](../network/intro/index.md)、存储等能力，兼容各种集群接入，支持集群从部署、版本升级、证书变更、配置变更、回收等全生命周期管理，突破 K8s API 性能瓶颈，实现企业超大规模用户并发使用多集群。
针对企业环境，提供场景化的网络方案，实现当前企业网络基础设施复用的最大化，降低企业使用云原生应用门槛。

涉及的模块：[全局管理](../ghippo/intro/index.md)、[容器管理](../kpanda/intro/index.md)、[集群生命周期管理](../community/kubean.md)、[云原生网络](../network/intro/index.md)、[云原生存储](../storage/index.md)

![云原生底座](https://docs.daocloud.io/daocloud-docs-images/docs/images/07base.png)

在成功获取许可证授权后，首次登录 DCE 5.0 将会呈现一个仪表盘。 该仪表盘以直观的方式展示了各种信息，包括正在运行的容器和 Pod、CPU 和内存使用情况、集群/节点/Pod 的健康状态、告警信息、资源利用情况以及功能一览等内容。



dashboard

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

K8s 认证服务商

DaoCloud 是经 CNCF 官方认证的 Kubernetes 提供商，即 Kubernetes Certified Service Provider (KCSP)，早在 2017 年就顺利通过了 Kubernetes 的官方认证，是国内最早涉足并得到 CNCF 官方认可的提供商。 我们常年积极参与 CNCF 相关社区贡献，引领社区技术潮流。每次 Kubernetes 新版本发布，都会率先适配增强新特性，并将其推向实际生产环境。

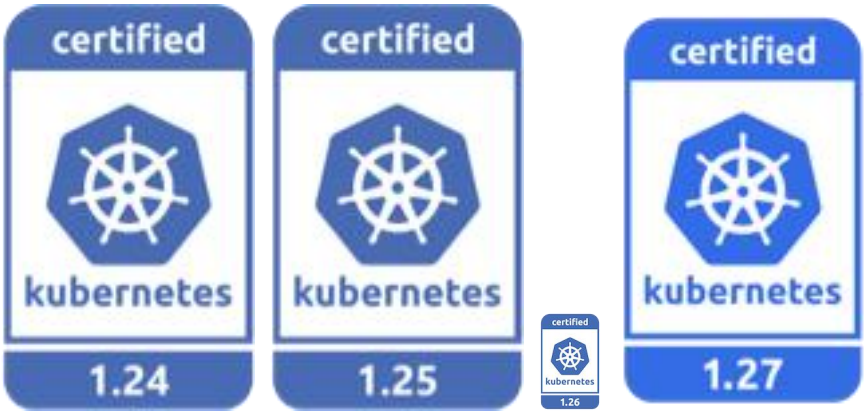
作为一家经过官方严格审核的 Kubernetes 服务提供商，DaoCloud 在帮助企业采用 Kubernetes 成功部署大规模集群方面有深厚的经验。

DaoCloud 作为一家经验丰富的 KCSP 意味着：

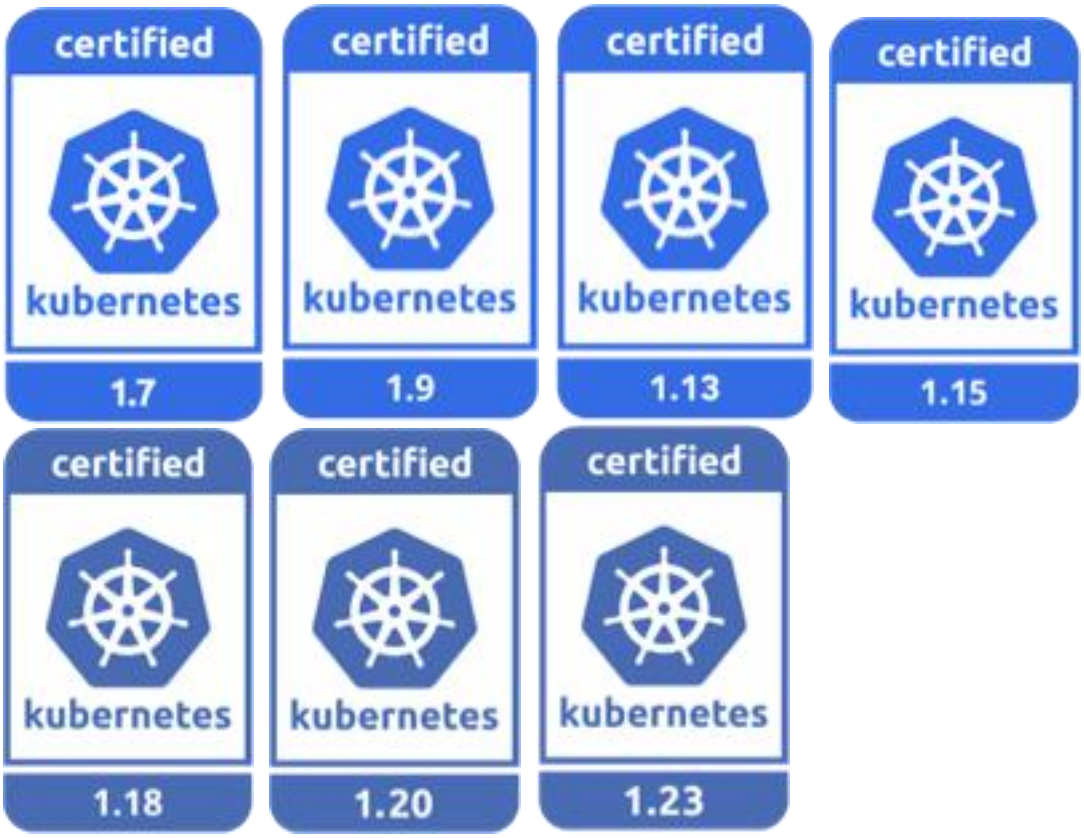
- DaoCloud 是 Kubernetes 官方推荐的资深合作伙伴
- DaoCloud 是经 CNCF 社区公认、有专家资质、能充分胜任 Kubernetes 集群部署专业工作的优良企业
- 最终用户选用 Kubernetes 部署集群时，DaoCloud 是 CNCF 云原生社区公认并推荐的品牌供应商
- 常年位列 KCSP 的 DaoCloud 掌握云原生领域领先技术，时刻引领云原生社区发展方向

目前 DaoCloud 经授权合规的 Kubernetes 版本包括但不限于：

K8s 官方维护的当前 4 个版本：



K8s 官方不再维护（但 DaoCloud KLTS 会持续维护）的历史版本：



有关 DaoCloud 开源贡献的更多信息，请参阅博客 DaoCloud 是 K8s 资深认证服务商。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

术语表

本页按字母顺序列出 DEC 5.0 常见的一些术语。

A

- Abstraction，抽象

  在当今计算的背景/上下文中，抽象是一种对服务的消费（消费对象可以是计算机程序或人类）， 在这个消费过程中会隐藏消费的细节，使整个系统更通用也更容易理解。

  电脑的操作系统就是一个很好的例子，它把计算机工作的所有细节都抽象出来了。 您不需要知道任何关于 CPU、内存以及程序如何被运行，只需在操作系统上安装好应用程序，系统就会处理所有运行相关的细节。 所有这些细节都隐藏在操作系统的“幕布”或抽象概念之后。

  系统通常有多个抽象层，这可以大大简化开发工作。 编程时开发人员只需构建与特定抽象层兼容的组件，而不必担心可能各种异构的底层细节。 如果某些组件能与抽象层一起工作，无论底层是什么样的，它都能在操作系统中良好运行。

- Addon，扩展

  扩展 DCE 功能的资源对象。 您可以通过容器管理 -> Helm 模板安装更多附加扩展组件。

- Admission Controller，准入控制器

  准入控制器是一段代码， 它会在请求通过认证和鉴权之后、对象被持久化之前拦截到达 API 服务器的请求。

准入控制器可以执行验证（Validating）和/或变更（Mutating）操作。 变更（mutating）控制器可以根据被其接受的请求更改相关对象；验证（validating）控制器则不行。

准入控制器限制创建、删除、修改对象的请求。 准入控制器也可以阻止自定义动作，例如通过 API 服务器代理连接到 Pod 的请求。 准入控制器不会（也不能）阻止读取（get、watch 或 list）对象的请求。

- Affinity，亲和性

  在 DCE 中亲和性是一组规则，它们为调度程序提供在何处放置 Pod 的提示信息。

  这些规则是使用 DCE Label 和 Pod 指定的选择算符定义的， 这些规则可以是必需的或首选的，这取决于您希望调度程序执行它们的严格程度。

  实际分为节点亲和性、Pod 间亲和性。

- Aggregation Layer，聚合层

  聚合层允许您在自己的集群上安装额外的 K8s 风格的 API。

  当配置了 K8s API 服务器来支持额外的 API 时，您就可以在 K8s API 中增加 APIService 对象来申领（Claim）一个 URL 路径。

- Alert Rule，告警规则

  在 Insight 中，这是基于资源状态创建的告警对象，可以自定义触发规则的条件以及通过何种方式发送通知的规则。

- Annotation，注解

  注解是以键值对的形式给资源对象附加随机的无法标识的元数据。

注解中的元数据可大可小，可以是结构化的也可以是非结构化的，并且能包含标签不允许使用的字符。 像工具和软件库这样的客户端可以检索这些元数据。

- API, Application Programming Interface

  API (即应用程序接口）是计算机程序间交互的一种方式。 就像人类可以通过网页与网站进行交互一样，API 允许计算机程序之间进行交互。 与人类的交互不同，API 可以限制对方可以问什么和不能问什么。对交互的限制有助于在程序之间创建稳定、实用的信息传输。 我们提供了完善的 OpenAPI 文档供第三方调用和开发。

- API-initiated eviction, API 发起的驱逐

  API 发起的驱逐是一个先调用 Eviction API 创建驱逐对象，再由该对象优雅地中止 Pod 的过程。

  您可以通过 kube-apiserver 的客户端，比如 kubectl drain 这样的命令，直接调用 Eviction API 发起驱逐。 当 Eviction 对象创建出来之后，该对象将驱动 API 服务器终止选定的 Pod。

  API 发起的驱逐取决于您配置的 PodDisruptionBudgets 和 terminationGracePeriodSeconds。

  API 发起的驱逐不同于节点压力引发的驱逐。

- API Group

  这是 K8s API 中的一组相关路径。

  通过更改 API 服务器的配置，可以启用或禁用 API Group。 您还可以禁用或启用指向特定资源的路径。 API 组使扩展 K8s API 更加的容易。 API 组在 REST 路径和序列化对象的 apiVersion 字段中指定。

- API 网关

API 网关是一种通过聚合多个应用程序的 API，并实现一站式管理的工具。 它允许组织将关键性功能移交到一个可集中管理的地方，例如身份验证和授权、限制应用程序之间的请求数量。 一个 API 网关则作为一个公共的接口，向 API 消费者（通常来自外部）提供服务。

- App Container，应用程序容器

  与 Init 容器比较而言，这是运行部分工作负载的容器。

  应用程序容器是在 Pod 中的容器（或 app 容器），在 Init 容器启动完毕后才开始启动。

  Init 容器使您可以分离对于工作负载整体而言很重要的初始化细节，一旦应用程序容器启动，Init 容器不需要继续运行。 如果 Pod 没有配置 Init 容器，则该 Pod 中的所有容器都是应用程序容器。

- App Architect，应用架构师

  应用架构师是负责应用高级设计的开发 Leader。

  应用架构师确保应用的实现允许它和周边组件进行可扩展的、可持续的交互。 周边组件包括数据库、日志基础设施和其他微服务。

- App Developer，应用开发者

  编写可以在 Kubernetes 集群上运行的应用的开发人员。 应用开发者专注于应用的某一部分。他们工作范围的大小有明显差异。

- App, Application，应用

  各种容器化服务运行所在的一层。

- Audit log，审计日志

  审计日志提供了对系统中对象所做更改的历史记录。

- Authorization，授权

授权指将用户完成具体工作所需的权限授予用户，授权通过系统角色或自定义角色的权限生效。 用户获得具体的权限后，可以对资源或服务进行操作。

- Autoscaling，自动扩缩

  自动扩缩，通常是指在计算资源方面，系统能够进行自动扩缩的能力。 自动扩缩系统可在需要时自动添置资源，通过扩缩来满足不断变化的用户需求。 自动扩缩的过程各不相同，可基于不同指标进行配置，例如内存或处理时间。 托管云服务相较于大多数本地部署环境，有更多的可选项和实施项，因此往往都搭配有自动扩缩功能。

  在此之前，基础设施和应用程序的架构设计会考虑到系统峰值的使用情况。这种架构意味着大部分资源没有得到充分利用，并且在面对不断变化的用户需求时缺乏弹性。 缺乏弹性则意味着低谷时的业务成本增加，而在高峰时又会由于需求过盛引起的服务中断而导致业务流失。

  通过利用云，虚拟化和容器化应用程序及其依赖项等手段，组织可以构建随用户需求而扩缩的应用程序。 他们可以监控应用程序的流量并自动扩缩，从而提供最佳的用户体验。 以 Netflix 每周五晚上的收视率增长为例，自动扩缩意味着动态添置更多资源：即增加服务器数量以支持更多视频播放需求，并在需求回落后同步缩减。

B, C

- Bare Metal Machine，裸机

  裸机是指一台物理计算机，更具体地说是一台服务器，它只有一个操作系统。 这样区分在现代计算环境中很重要，因为许多（甚至可以说大部分）服务器都是虚拟机。 物理服务器通常是一台相当大的计算机，内置强大的硬件。 直接在该物理硬件上安装操作系统并运行应用程序，无需虚拟化，称为在“裸机”上运行。

DaoCloud 提供了集成容器化管理平台的云原生一体机，实现软硬协同多架构融合体。

- Blue Green Deployment，蓝绿部署

  蓝绿部署是一种以最小的停机时间更新运行中的计算机系统的策略。 运营商维护两个环境，被称为“蓝”和“绿”。一个为生产流量服务（所有用户目前正在使用的版本），而另一个则被更新。 一旦在非活动（绿色）环境中的测试结束，生产流量就会被切换过来（通常通过使用负载平衡器）。请注意，蓝绿色的部署通常意味着一次性切换整个环境，包括许多服务。令人困惑的是，有时这个术语被用于系统内的单个服务。为了避免这种歧义，在提到单个组件时，最好使用“零停机部署”一词。

- Canary Deployment，金丝雀部署

  金丝雀部署是一种部署策略，开始时有两个环境：一个有实时流量，另一个包含没有实时流量的更新代码。 流量逐渐从应用程序的原始版本转移到更新版本。 它可以从移动 1% 的实时流量开始，然后是 10%，25%，以此类推，直到所有流量都通过更新的版本运行。 企业可以在生产中测试新版本的软件，获得反馈，诊断错误，并在必要时快速回滚到稳定版本。

  “金丝雀”一词是指“煤矿中的金丝雀”的做法，即把金丝雀带入煤矿以保证矿工的安全。 如果出现无味的有害气体，鸟就会死亡，而矿工们知道他们必须迅速撤离。 同样，如果更新后的代码出了问题，现场交通就会被“疏散”回原来的版本。

- cAdvisor

  cAdvisor (Container Advisor) 为用户提供运行中的容器资源用量和性能特征的相关信息。

  cAdvisor 是一个守护进程，负责收集、聚合、处理并输出运行中容器的信息。 具体而言，针对每个容器，该进程记录容器的资源隔离参数、历史资

源用量、完整历史资源用量和网络统计的直方图。 这些数据可以按容器或按机器层面输出。

- Certificate，证书

  证书是个安全加密文件，用来确认对 Kubernetes 集群访问的合法性。

  证书（Certificate）可以让 Kubernetes 集群中运行的应用程序安全的访问 Kubernetes API。 证书可以确认客户端是否被允许访问 API。

- cgroup, control group，控制组

  一组具有可选资源隔离、审计和限制的 Linux 进程。

  cgroup 是一个 Linux 内核特性，对一组进程的资源使用（CPU、内存、磁盘 I/O 和网络等）进行限制、审计和隔离。

- Chaos Engineering，混沌工程

  混沌工程或 CE 是在生产中对分布式系统进行实验的学科，以建立对系统承受动荡和意外情况时能力的信心。

  SRE 和 DevOps 实践侧重于提高产品弹性和可靠性的技术。 系统在故障容灾时确保服务质量的能力通常是对软件开发提出的要求。这里涉及到几个方面可能导致应用程序中断， 例如基础设施、平台或（基于微服务）的应用程序的其他部分。 高频地持续部署新功能到生产环境会增加服务中断和恶性事件发生的可能性，乃至于对业务产生重大影响。

  混沌工程是一种满足弹性要求的技术。它用于实现对基础架构、平台和应用程序等意外发生时的故障容灾。 混沌工程师使用混沌实验主动注入随机故障，以验证应用程序、基础架构或平台是否可以自我修复，并且故障不会对客户产生明显影响。 混沌实验旨在发现盲点（例如监控或自动扩缩技术）并在恶性事件发生期间增强团队之间的沟通。 这种方法有助于提高复杂系统的弹性和团队对其的信心，尤其是在生产环境。

- CIDR, Classless Inter-Domain Routing

  CIDR（无类域间路由）是一种描述 IP 地址块的符号，被广泛使用于各种网络配置中。

  在 Kubernetes 的上下文中，每个节点以 CIDR 形式（含起始地址和子网掩码）获得一个 IP 地址段， 从而能够为每个 Pod 分配一个独一无二的 IP 地址。 虽然其概念最初源自 IPv4，CIDR 已经被扩展为涵盖 IPv6。

- Client Server Architecture，主从式架构

  在主从式架构（客户端-服务器架构）中，构成应用程序的逻辑（或代码）会被分成两个或多个组件： 一个期望工作被完成的客户端（例如，运行在您的 web 浏览器中的 Gmail web 应用程序）， 以及一个或多个满足该请求的服务器（例如，运行在云中心的谷歌计算机上的“发送电子邮件”服务）。 在这个例子中，您所写的外发邮件是由客户端（运行在您的 web 浏览器中的 web 应用程序）发送到服务器（Gmail 的计算机，它将您的外发邮件转发给对应的收件人）。

  这与在一个地方完成所有工作的独立应用程序（如桌面应用程序）形成了对比。 例如，像 Microsoft Word 这样的文字处理程序可能会完全安装并运行在您的计算机上。

- Cloud Computing，云计算

  云计算是一种通过互联网按需提供计算资源（如 CPU、网络和磁盘功能）的模型。 云计算使用户能够在远程物理位置访问和使用计算能力。
  AWS、GCP、Azure、DigitalOcean 等云提供商都为第三方提供了在多个地理位置租用计算资源的能力。

  传统上，组织在尝试扩展计算能力的使用时面临两个主要问题。 他们要么获取、支持、设计和支付托管物理服务器和网络的设施，要么扩展和维护这些设施。 云计算允许组织将部分计算需求外包给另一个组织。

云提供商为组织提供按需租用计算资源并按使用付费的能力。这允许进行两项主要创新： 组织可以在不浪费时间计划和花费金钱或资源在新的物理基础设施上的情况下进行尝试，并且他们可以根据需要和按需扩缩。 云计算允许组织根据需要采用尽可能多或尽可能少的基础设施。

- Cloud Controller Manager，云控制器管理器

  一个 Kubernetes 控制平面组件，嵌入了特定于云平台的控制逻辑。 云控制器管理器允许您将您的集群连接到云提供商的 API 之上， 并将与该云平台交互的组件同与您的集群交互的组件分离开来。

  通过分离 Kubernetes 和底层云基础设置之间的互操作性逻辑， cloud-controller-manager 组件使云提供商能够以不同于 Kubernetes 主项目的步调发布新特征。

- Cloud-Native Apps，云原生应用程序

  云原生应用程序专门设计用于利用云计算中的创新。 这些应用程序可以轻松地与其各自的云架构集成，充分利用云的资源和可扩缩性功能。 它还指利用云计算驱动的基础设施创新的应用程序。 今天的云原生应用程序包括在云提供商的数据中心和本地云原生平台上运行的应用程序。

  传统上，本地环境以相当定制的方式提供计算资源。 每个数据中心都有与特定环境紧密耦合应用程序的服务，通常严重依赖于基础设施的手动配置，例如虚拟机和服务。 这反过来又将开发人员及其应用程序限制在该特定数据中心。 不是为云设计的应用程序无法利用云环境的弹性和扩缩能力。 例如，需要手动干预才能正确启动的应用程序无法自动扩缩，也无法在发生故障时自动重启。

  虽然云原生应用程序没有“一刀切”的路径，但它们确实有一些共性。 云原生应用程序具有弹性、可管理性，并由配套的云服务套件提供帮助。 各种云服务实现了高度的可观测性，使用户能够在问题升级之前检测和解决问

题。 结合强大的自动化，它们使工程师能够以最少的工作频繁且可预测地进行高影响力的更改。

- Cloud-Native Security，云原生安全

  云原生安全是一种将安全性构建到云原生应用程序中的方法。 它确保安全是从开发到生产的整个应用程序生命周期的一部分。 云原生安全旨在确保与传统安全模型相同的标准，同时适应云原生环境的特殊性，即快速的代码更改和高度短暂的基础设施。 云原生安全与称为测试左移的实践高度相关。

  传统的安全模型是根据许多不再有效的假设构建的。 云原生应用程序变化频繁，使用大量开源工具和库，通常运行在供应商控制的基础设施中，并且会受到快速的基础设施变化的影响。 代码审查、长质量保证周期、基于主机的漏洞扫描和最后一分钟的安全审查无法与云原生应用程序一起扩缩。

  云原生安全引入了一种新的工作方式，通过从传统的安全模型迁移到发布周期的每个步骤都涉及安全的模型来保护应用程序。 人工审核和检查在很大程度上被自动扫描所取代。 快速代码发布管道与在编译之前扫描代码以查找漏洞的工具集成在一起。 开源库从受信任的来源中提取并监控漏洞。 云原生安全模型不是减缓变化，而是通过频繁更新易受攻击的组件或确保定期更换基础设施来拥抱它。

- Cloud-Native Tech，云原生技术

  云原生技术，也称为云原生技术栈，是用于构建云原生应用程序的技术。 使组织能够在公共云、私有云和混合云等现代动态环境中构建和运行可扩缩的应用程序，他们坚持“云的承诺”并充分利用云计算的优势。 它们从头开始设计，以利用云计算和容器的功能，服务网格、微服务和不可变的基础设施就是这种方法的例证。

  云原生技术栈有许多不同的技术类别，可应对各种挑战。 如果您看看 CNCF 云原生全景图，您会看到它涉及到多少不同的领域。 但在高层次

上，它们解决了一组主要挑战：传统 IT 运营模式的缺点。挑战包括难以创建可扩缩、容错、自我修复的应用程序，以及资源利用效率低下等。

虽然每种技术都解决了一个非常具体的问题，但作为一个整体，云原生技术支持松散耦合的系统，这些系统具有弹性、可管理性和可观测性。 结合强大的自动化，它们使工程师能够以最少的工作频繁且可预测地进行高影响力的更改。 云原生系统的理想特性更容易通过云原生堆栈实现。

- Cloud Provider，云提供商

  一个提供云计算平台的商业机构或其他组织。 云提供商有时也称作云服务提供商（Cloud Service Provider, CSP）提供云计算平台或服务。

  很多云提供商提供托管的基础设施（也称作基础设施即服务或 IaaS）。 针对托管的基础设施，云提供商负责服务器、存储和网络，而用户（您） 负责管理其上运行的各层软件，例如运行一个 Kubernetes 集群。

  您也会看到 Kubernetes 被作为托管服务提供；有时也称作平台即服务或 PaaS。 针对托管的 Kubernetes，您的云提供商负责 Kubernetes 的控制平面以及节点及其所依赖的基础设施： 网络、存储以及其他一些诸如负载均衡器之类的元素。

- Cluster，集群

  集群是运行容器化应用程序的一组计算节点。通常，组成集群的计算节点彼此可以直接连接。集群通过规则或策略限制外部访问。 工作节点会托管 Pod，而 Pod 就是作为应用负载的组件。 控制平面管理集群中的工作节点和 Pod。 在生产环境中，控制平面通常跨多台计算机运行， 一个集群通常运行多个节点，提供容错性和高可用性。

  集群是一组计算机或应用程序，它们为一个共同的目标一起工作。 在云原生计算的背景下，这个术语最常被应用于 Kubernetes。 Kubernetes 集群是一组服务（或工作负载），它们在各自的容器中运行，通常在不同的机器上。 所有这些容器化服务的集合，通过网络连接，代表一个集群。

在单台计算机上运行的软件会出现单点故障——如果这台计算机崩溃了，或者有人不小心拔掉了电源线，那么一些关键的业务系统就可能被下线。 这就是为什么现代软件通常被构建为分布式应用，以集群的形式组合在一起。

集群式的分布式应用在多台机器上运行，消除了单点故障。但构建分布式系统真的很难，事实上，它本身就是一门计算机科学的学科。 对全球系统的需求和多年的试验和错误导致了一种新的技术栈的发展：云原生技术，这些新技术是使分布式系统的操作和创建更容易的构件。

- Cluster Architect，集群架构师

  集群架构师是指涉及一个或多个 Kubernetes 集群基础设施设计的人员。集群架构师通常更关心分布式系统的最佳实践，例如高可用性和安全性。

- Cluster Infrastructure，集群基础设施

  处于基础设施层，提供并维护虚拟机、网络、安全组及其他资源。

- Cluster Operations，集群运营

  Kubernetes 管理相关工作包括：日常管理运营和协调升级。

  集群运营工作的示例包括：

  - 部署新节点来扩容集群、执行软件升级、实施安全控制、
  - 添加或删除存储、配置集群网络、管理集群范围的可观测性和响应集群事件。

- Cluster Operator，集群运营人员

  配置、控制、监控集群的人员。 他们的主要职责是保证集群正常运行，可能需要进行周期性的维护和升级活动。

- CNCF, Cloud Native Computing Foundation，云原生计算基金会

隶属于 Linux 基金会，成立于 2015 年 12 月，是一个非营利性组织，致力于培育和维护一个厂商中立的开源生态系统来推广云原生技术，普及云原生应用。

- CNI, Container network interface，容器网络接口

  容器网络接口插件是遵循 appc/CNI 协议的一类网络插件。

  DCE 5.0 支持的 CNI 包括但不限于：

  - Calico
  - Cilium
  - Contour
  - F5networks
  - Ingress-nginx
  - Metallb
  - Multus-underlay
  - Spiderpool
  - Submariner
- ConfigMap，配置项

  ConfigMap 是一种 API 对象，用来将非机密性的数据保存到键值对中。使用时可以用作环境变量、命令行参数或者存储卷中的配置文件。

  ConfigMap 将您的环境配置信息和容器镜像解耦，便于应用配置的修改。
- Container Environment Variables，容器环境变量

  容器环境变量提供了 name=value 形式的、运行容器化应用所必须的一些重要信息。

容器环境变量为运行中的容器化应用提供必要的信息，同时还提供与容器重要资源相关的其他信息， 例如文件系统信息、容器自身的信息以及其他像服务端点（Service endpoints）这样的集群资源信息。

- Container Lifecycle Hooks，容器生命周期钩子

  生命周期钩子暴露容器管理生命周期中的事件，允许用户在事件发生时运行代码。

  针对容器暴露了两个钩子：

  - PostStart 在容器创建之后立即执行，
  - PreStop 在容器停止之前立即阻塞并被调用。

- Container Image，容器镜像

  容器镜像是一个不可改变的静态文件，包含创建容器的依赖性。 这些依赖可能包括一个可执行的二进制文件、系统库、系统工具、环境变量和其他必要的平台设置。 容器镜像是应用程序容器化的结果，通常存储在容器注册表中，在那里可以下载并使用容器运行时接口（CRI）作为一个孤立的进程运行。 容器镜像框架必须遵循开放容器倡议（OCI）定义的标准模式。

  传统上，应用服务器是按环境配置的，而应用则被部署到这些环境中。 环境之间的任何错误配置都是有问题的，常常导致停机或部署失败。 一个应用程序的环境需要是可重复的和定义明确的；否则，与环境有关的错误的机会就会增加。 当应用程序的环境定义不足或不准确时，应用程序的横向和纵向扩缩就会成为挑战。

  容器镜像将一个应用程序与它的任何运行时的依赖性捆绑在一起，例如一个应用程序服务器。 这提供了所有环境的一致性，包括开发人员的机器。 容器镜像可用于实例化所需的多个容器，允许更大的可扩缩性。

- Container，容器

容器是由计算机操作系统管理的具有资源和能力限制的运行进程。 容器进程可用的文件被打包为容器镜像。 容器在同一台机器上彼此相邻运行，但通常操作系统会阻止单独的容器进程相互交互。

在容器可用之前，需要单独的机器来运行应用程序。每台机器都需要自己的操作系统，它占用 CPU、内存和磁盘空间， 所有这些都是为了让单个应用程序运行。此外，操作系统的维护、升级和启动是另一个重要的工作来源。

容器共享相同的操作系统及其机器资源，分散了操作系统的资源开销并创建了物理机器的有效使用。 这种能力之所以成为可能，是因为容器之间的交互通常受到限制。 这允许更多的应用程序在同一台物理机器上运行。

但是，有一些限制。由于容器共享相同的操作系统，因此可以认为进程不如替代方法安全。 容器还需要对共享资源进行限制。 为了保证资源， 管理员必须约束和限制内存和 CPU 的使用，以使其他应用程序不会表现不佳。

- containerd

  强调简单性、健壮性和可移植性的一种容器运行时。

  containerd 是一种容器行时，能在 Linux 或者 Windows 后台运行。containerd 能取回、存储容器镜像，执行容器实例，提供网络访问等。

- Containerization，容器化

  容器化是将一个应用程序及其依赖关系捆绑到容器镜像中的过程。 容器构建过程需要遵守开放容器倡议（OCI）标准。 只要输出的是一个符合这个标准的容器镜像，使用哪种容器化工具并不重要。

  在容器盛行之前，企业依靠虚拟机（VMs）在一台裸机上协调多个应用程序。 虚拟机比容器大得多，需要一个管理程序来运行。由于这些较大的虚

拟机模板的存储、备份和传输，创建虚拟机模板也很慢。 此外，虚拟机可能会出现配置漂移，这违反了不变性的原则。

容器镜像是轻量级的（与传统的虚拟机不同），容器化过程需要一个带有依赖性列表的文件。 这个文件可以被版本控制，构建过程也可以自动化，允许一个组织在自动化过程中关注其他优先事项。 容器镜像由一个唯一的标识符来存储，该标识符与它的确切内容和配置相联系。 当容器被安排和重新安排时，它们总是被重置为其初始状态，从而消除了配置漂移。

- Container as a service (CaaS)，容器即服务

  容器即服务（CaaS）是一种云服务，有助于使用基于容器的抽象管理和部署应用程序。 这项服务可以部署在企业内部或云中。

  CaaS 供应商提供了一个框架或协调平台，使容器部署和管理的关键 IT 功能自动化。 它帮助开发者建立安全和可扩缩的容器化应用。 因为用户只购买他们需要的资源（调度能力、负载平衡等），他们可以节省资金并提高效率。 容器创造了一致的环境，以快速开发和交付可以在任何地方运行的云原生应用。

  如果没有 CaaS，软件开发团队需要部署、管理和监控容器运行的底层基础设施。

  当把容器化应用部署到 CaaS 平台时，用户可以通过日志聚合和监控工具获得对系统性能的可见性。 CaaS 还包括自动扩缩和协调管理的内置功能。 它使团队能够建立高可见性和高可用性的分布式系统。 此外，通过允许快速部署，CaaS 提高了团队的开发速度。在容器确保一致的部署目标的同时，CaaS 通过减少管理部署所需的 DevOps 资源，降低了工程运营成本。

- Continuous Delivery，持续交付

  持续交付，通常缩写为 CD，是一套实践，其中代码的变化被自动部署到验收环境中（或者，在持续部署的情况下，部署到生产中）。 CD 关键是包

括确保软件在部署前得到充分测试的程序，并提供一种在认为必要时回滚修改的方法。 持续集成（CI）是实现持续交付的第一步（也就是说，在测试和部署之前，变化必须干净地合并）。

部署可靠的更新在规模上成为一个问题。 理想情况下，我们会更频繁地部署，为终端用户提供更好的价值。 然而，手动操作会使每一个变化都转化为高额的交易成本。 历史上，为了避免这些成本，企业发布的频率较低，一次部署更多的变化，增加了出错的风险。

CD 策略创建了一个完全自动化的生产路径，使用各种部署策略测试和部署软件，如金丝雀部署或蓝绿部署发布。 这使得开发人员可以频繁地部署代码，让他们放心地认为新的修订版已经过测试。 通常情况下，CD 策略中使用基于主干的开发，而不是功能分支或拉动请求。

- Continuous Deployment，持续部署

  持续部署，通常缩写为 CD，通过将已经完成的软件直接部署到生产环境，比持续交付更进了一步。 持续部署（CD）与持续集成（CI）一起，通常被称为 CI/CD。 CI 流程测试给定应用程序的修改是否正确，CD 流程自动部署企业测试环境的代码更改到生产环境。

  发布新的软件版本是一个劳动密集且容易出错的过程。 这也是企业不想频繁发布新版本的原因，避免生产事故并减少工程师在正常工作时间之外需要随时响应的时间。 传统的软件部署模型使组织陷入了一个恶性循环，即发布软件的过程无法同时满足企业在稳定性和软件迭代速度方面的需求。

  通过自动化发布周期迫使企业更频繁地发布版本到生产环境，CD 为运维团队完成了 CI 为开发团队所做的事情。 具体来说，它迫使运维团队将生产部署中痛苦且容易出错的部分自动化，从而降低整体风险。 它还使企业能够更好地接受和适应生产环境变化，从而提高稳定性。

- Continuous Integration，持续集成

持续集成，通常缩写为 CI，是尽可能定期集成代码变化的做法。 CI 是持续交付（CD）的前提。 传统上，CI 过程从代码修改提交到源码控制系统（Git、Mercurial 或 Subversion）时开始，以准备被 CD 系统使用的测试工件结束。

软件系统通常是庞大而复杂的，有许多开发人员在维护和更新它们。 在系统的不同部分平行工作，这些开发人员可能会做出相互冲突的修改，并在无意中破坏对方的工作。 此外，由于多个开发人员在同一个项目上工作，任何日常工作，如测试和计算代码质量，都需要由每个开发人员重复进行，浪费了时间。

每当开发人员提交修改时，CI 软件都会自动检查代码修改是否合并得很干净。 使用 CI 服务器来运行代码质量检查、测试甚至部署，这几乎是一种普遍的做法。 因此，它成为团队内部质量控制的一个具体实施。 CI 允许软件团队把每一个代码提交变成具体的失败或可行的候选发布。

- Control Plane，控制平面

  控制平面是指容器编排层，它暴露 API 和接口来定义、部署容器和管理容器的生命周期。

  这个编排层是由多个不同的组件组成，例如以下（但不限于）几种：

  - etcd
  - API 服务器
  - 调度器
  - 控制器管理器
  - 云控制器管理器

  这些组件可以作为传统的操作系统服务（守护程序）或容器运行。运行这些组件的主机也被称为 Master。

- Controller，控制器

控制器通过 API 服务器监控集群的公共状态，并致力于将当前状态转变为期望的状态。

控制器（作为控制平面的一部分）通过 API 服务器监控您的集群中的公共状态。

其中一些控制器是运行在控制平面内部的，对 Kubernetes 来说，这些控制器提供核心控制操作。 比如部署控制器（deployment controller）、守护控制器（daemonset controller）、 命名空间控制器（namespace controller）、持久化数据卷控制器（persistent volume controller）等 都是运行在 kube-controller-manager 中的。

- Counter，计数器

  计数器是一种累计型的度量指标，它是一个 **只能递增** 的数值。计数器主要用于统计类似于服务请求数、任务完成数和错误出现次数这样的数据。

- Contour，网关控制节点

  Contour 作为微服务网关的控制面，被部署为控制节点，充当 Envoy 的后端管理服务能力。 Contour 提供便捷的网关配置，支持动态配置更新，多集群部署能力。 Contour 同时提供了 HTTPProxy CRD 用于增强 Kubernetes Ingress 的核心配置能力， Contour 以 Deployment 的方式部署，为保障生产服务稳定性，建议部署在多个副本。

- Control Plane，控制平面

  控制平面是一组系统服务，这些服务配置网格或者网格的子网来管理工作负载实例之间的通信。 单个网格中控制平面的所有实例共享相同的配置资源。

- CRD, CustomResourceDefinition，自定义资源定义

  通过定制化的代码给您的 Kubernetes API 服务器增加资源对象，而无需编译完整的定制 API 服务器。

当 Kubernetes 公开支持的 API 资源不能满足您的需要时，CRD 让您可以在自己的环境上扩展 Kubernetes API。

自定义资源定义是默认的 Kubernetes API 扩展，服务网格使用 Kubernetes CRD API 来进行配置。

- CRI-O

  专用于 Kubernetes 的轻量级容器运行时软件工具。该工具可让您通过 Kubernetes CRI 使用 OCI 容器运行时。

  CRI-O 是 CRI 的一种实现，使得您可以使用与开放容器倡议（Open Container Initiative；OCI） 运行时规范兼容的容器。

  部署 CRI-O 允许 Kubernetes 使用任何符合 OCI 要求的运行时作为容器运行时去运行 Pod，并从远程容器仓库获取 OCI 容器镜像。

- CRI, Container Runtime Interface，容器运行时接口

  CRI 是一组与节点上 kubelet 集成的容器运行时 API。 kubelet 和容器运行时之间通信的主要协议。

  CRI 定义了主要 gRPC 协议，用于集群组件 kubelet 和容器运行时。

- CR, Container Runtime，容器运行时

  容器运行时是负责运行容器的组件。

  Kubernetes 支持许多容器运行环境，例如 containerd、cri-o 以及 Kubernetes CRI 的其他任何实现。

- CronJob，周期调度任务

  管理定期运行的任务。

  与 **crontab** 文件中的一行命令类似，周期调度任务（CronJob）对象使用 cron 格式设置排期表。

- CSI, Container Storage Interface

  容器存储接口 （CSI）定义了存储系统暴露给容器的标准接口。

  CSI 允许存储驱动提供商为 Kubernetes 创建定制化的存储插件， 而无需将这些插件的代码添加到 Kubernetes 代码仓库（外部插件）。 要使用某个存储提供商的 CSI 驱动，您首先要将它部署到您的集群上。 然后您才能创建使用该 CSI 驱动的 Storage Class。

D

- DaemonSet，守护进程集

  确保 Pod 的副本在集群中的一组节点上运行。

  用来部署系统守护进程，例如日志搜集和监控代理，这些进程通常必须运行在每个节点。

- Data Center，数据中心

  数据中心是专门设计用于容纳计算机（通常是服务器）的专用建筑物或设施。 数据中心往往连接到高速互联网线路，尤其是专注于云计算的数据中心。 数据中心所在的建筑物还配备了即使在发生负面事件时也能维持服务的设备，例如在停电期间提供电力的发电机，以及处理计算机产生的废热的强大空调。

  与每个企业都必须在其所在的位置主管自己的服务器设备不同，数据中心允许公司和个人利用数据中心提供商的专业知识和规模效率。 这意味着不必担心管理电源、灭火技术、空调或高速互联网连接等问题。

  对于云计算，数据中心至关重要。由于可以根据可扩缩性配置资源和基础设施， 因此企业可以在数据中心租用云计算资源，而无需多虑预测的资源过多过少带来的问题。 由于数据中心遍布世界各地，这允许在地理上接近需求的地方提供资源，而无需实际运送和设置设备。

- Database as a service，数据库即服务

数据库即服务（DBaaS）是由云运营商（公共或私有）管理的服务，它支持应用程序，而无需应用程序团队执行传统的数据库管理功能。 DBaaS 允许应用程序开发人员利用数据库，而无需成为专家或聘请数据库管理员(DBA）来保持数据库最新状态。

传统上，在本地设置中，组织必须定期投资额外的存储和处理能力，以适应可能昂贵的数据库扩展。 此外，开发人员在 IT 基础架构团队的帮助下配置和配置数据库，从而降低了数据库驱动应用程序的部署速度。 加载和执行它们也需要更长的时间。

DBaaS 允许开发人员将所有管理/行政操作外包给基于云的服务提供商。服务提供商确保数据库顺利运行，包括配置管理、备份、补丁、升级、服务监控等，并通过用户友好的界面进行管理。 DBaaS 可帮助组织更快地开发企业级应用程序，同时最大限度地降低数据库成本。

- Data ID

  Nacos 中某个配置集的 ID。配置集是一组配置项的集合，通常表现为一个配置文件，其中包括系统各方面的配置。

- Data Plane，数据平面

  数据平面是网格的一部分，直接控制工作负载实例之间的通信。 服务网格的数据平面使用智能 Envoy 代理部署成边车去调节和控制服务网格中发送和接受的流量。

  提供诸如 CPU、内存、网络和存储的能力，以便容器可以运行并连接到网络。

- Debugging，调试

  调试是从计算机程序、软件或系统中查找并解决故障（或错误） 以获得预期结果的过程或活动。 故障是导致不正确或不符合预期结果的缺陷或问题。

软件开发是一项复杂的活动，在不引入故障的情况下编写代码几乎是不可能的。 这些故障导致代码在执行时可能无法按预期运行，或产生未定义的行为。 根据应用程序的重要性，故障可能会产生重大的负面影响 —— 经济上甚至是人类生活。 通常，应用程序代码必须在不同阶段或环境被测试。 应用程序越关键，测试就必须越准确。

当出现故障时，工程师通过调试（例如，查找和修复）应用程序以减少生产系统中的不良行为。 调试并不是一件容易的事，因为工程师必须追踪不良行为的根源。 它需要有关代码本身和运行时执行上下文的知识。 这是不同的调试技术和工具派上用场的地方。 例如，分析日志、链路信息和指标信息，用于直接在生产中进行调试。 开发人员可以使用交互式调试在运行时单步执行代码，同时分析相关的执行上下文。 一旦定位到故障的根源，他们通过发起修复错误的请求或者发布新的补丁来更正代码行为。

- Deployement，无状态负载

  管理多副本应用的一种 API 对象，通常通过运行没有本地状态的 Pod 来完成工作。

  每个副本表现为一个 Pod，Pod 分布在集群中的节点上。 对于确实需要本地状态的工作负载，请考虑使用 StatefulSet。

- Device Plugin，设备插件

  一种软件扩展，可以使 Pod 访问由特定厂商初始化或者安装的设备。

  设备插件在工作节点上运行并为 Pod 提供访问资源的能力， 例如：本地硬件这类资源需要特定于供应商的初始化或安装步骤。

  设备插件向 kubelet 公布资源，以便工作负载 Pod 访问 Pod 运行所在节点上的硬件功能特性。 您可以将设备插件部署为 DaemonSet，或者直接在每个目标节点上安装设备插件软件。

- DevOps，开发运维

开发运维是一种方法论，其中团队拥有从应用程序开发到生产操作的整个过程，因此 DevOps，它不仅仅是实施一套技术，还需要文化和流程的彻底转变。 DevOps 需要一组工程师来处理小组件（相对于整个功能），从而减少交接——一个常见的错误来源。

传统上，在具有紧密耦合单体应用程序的复杂组织中，工作通常分散在多个组之间。 这导致了多次交接和较长的交货时间。 每次组件或更新准备就绪时，都会将其放入队列中以供下一个团队使用。 因为个人只参与了项目的一小部分，这种方法导致缺乏所有权。 他们的目标是将工作交给下一个小组，而不是为客户提供正确的功能——明显的优先级错位。

到代码最终投入生产时，它经过了这么多开发人员，排了这么多队列，如果代码不起作用，很难追查问题的根源。DevOps 颠覆了这种方法。

让一个团队拥有应用程序的整个生命周期可以最大限度地减少交接，降低部署到生产中的风险，提高代码质量， 因为团队还负责代码在生产中的执行方式，并且由于更多的自主权和所有权而提高了员工满意度。

- Dependency topology，依赖拓扑

  以拓扑图的方式展示服务调用之间的依赖关系。

- Destination，目标服务

  目标服务是 envoy 代表一个源服务工作负载与之打交道的远程上游服务。这些上游服务可以有多个服务版本，envoy 根据路由选择对应的版本。

- Destination Rule，目标规则

  目标规则定义了在路由发生后应用于服务的流量策略。 这些规则指定负载均衡的配置、来自边车代理的连接池大小以及异常检测设置，从而实现从负载均衡池中检测和熔断不健康的主机。

- Diagnosis，诊断模式

诊断模式即对 Contour 进行功能调试，支持在 Contour 启动时附加对应的启动参数。

- Disruption，干扰

  导致 Pod 服务停止的事件。

  干扰（Disruption）是指导致一个或者多个 Pod 服务停止的事件。 干扰会影响依赖于受影响的 Pod 的资源，例如 Deployment。

  如果您作为一个集群操作人员，销毁了一个从属于某个应用的 Pod，Kubernetes 视之为主动干扰（Voluntary Disruption）。 如果由于节点故障或者影响更大区域故障的断电导致 Pod 离线，Kubernetes 视之为非主动干扰（Involuntary Disruption）。

- Distributed Apps，分布式应用

  分布式应用是一种应用，其功能被分解成多个较小的独立部分。 分布式应用通常由单独的微服务组成，处理更广泛应用中的不同问题。 在云原生环境中，各个组件通常作为容器在集群上运行。

  运行在单一计算机上的应用程序代表了一个单点故障。如果该计算机发生故障，应用程序就不可用。 分布式应用通常与单体式应用形成对比。一个单体应用可能更难扩展，因为各种组件不能独立扩展。 随着应用程序的增长，它们也会拖累开发人员的速度，因为更多的开发人员需要在一个不一定有明确边界的共享代码库上工作。

  当把一个应用程序拆分成不同的部分并在许多地方运行时，整个系统可以容忍更多的故障。 它还允许应用程序利用单个应用程序实例所不具备的扩缩功能，即水平扩缩的能力。 然而，这也是有代价的：增加了复杂性和操作开销。您现在正在运行很多应用组件，而不是一个应用。

- Distributed System，分布式系统

分布式系统是通过网络连接的自主计算元素的集合，在用户看来是一个单一的连贯系统。 一般被称为节点，这些组件可以是硬件设备（如电脑、手机）或软件进程。 节点被编程以实现一个共同的目标，为了协作，它们通过网络交换信息。

今天的许多现代应用程序都非常大，需要超级计算机来操作。想想 Gmail 或 Netflix。 没有一台计算机强大到足以承载整个应用程序。通过连接多台计算机，计算能力几乎变得无限大。 如果没有分布式计算，我们今天依赖的许多应用就不可能实现。

传统上，系统会纵向扩缩。这就是当您在一台单独的机器上添加更多的 CPU 或内存。 垂直扩缩很耗时，需要停机，而且很快就会达到极限。

分布式系统允许水平扩缩（例如，在需要时向系统添加更多节点）。 这可以是自动化的，允许系统处理工作负载或资源消耗的突然增加。

非分布式系统面临着故障的风险，因为如果一台机器发生故障，整个系统都会故障。 分布式系统可以设计成这样，即使一些机器发生故障，整个系统仍然可以继续工作，产生相同的结果。

- Docker

  Docker 是一种可以提供操作系统级别虚拟化（也称作容器）的软件技术。

  Docker 使用了 Linux 内核中的资源隔离特性（如 cgroup 和内核命名空间）以及支持联合文件系统（如 OverlayFS 和其他）， 允许多个相互独立的“容器”一起运行在同一 Linux 实例上，从而避免启动和维护虚拟机（Virtual Machines；VM）的开销。

- Dockershim

  Dockershim 是 Kubernetes v1.23 及之前版本中的一个组件，Kubernetes 系统组件通过它与 Docker Engine 通信。

  从 Kubernetes v1.24 起，Dockershim 已从 Kubernetes 中移除。

- Downward API

  将 Pod 和容器字段值暴露给容器中运行的代码的机制。 在不需要修改容器代码的前提下让容器拥有关于自身的信息是很有用的。 修改代码可能使容器直接耦合到 Kubernetes。

  Kubernetes Downward API 允许容器使用它们自己或它们在 Kubernetes 集群中所处环境的信息。 容器中的应用程序可以访问该信息，而不需要以 Kubernetes API 客户端的形式执行操作。

  有两种方法可以将 Pod 和容器字段暴露给正在运行的容器：

  - 使用环境变量
  - 使用 downwardAPI 卷

  这两种暴露 Pod 和容器字段的方式统称为 **Downward API**。

- Dynamic Volume Provisioning，动态卷供应

  允许用户请求自动创建存储卷。

  动态供应让集群管理员无需再预先供应存储。相反，它通过用户请求自动地供应存储。 动态卷供应是基于 API 对象 StorageClass 的，StorageClass 可以引用卷插件（Volume Plugin）提供的卷，也可以引用传递给卷插件的参数集。

E, F

- Edge Computing，边缘计算

  边缘计算是一种分布式系统方法，它将一些存储和计算容量从主数据中心转移到其他数据源。 采集的数据在本地（例如在工厂车间、商店或整个城市）进行计算，而不是发送到集中的数据中心进行处理和分析。 这些本地处理单元或设备代表系统的边缘，而数据中心是这些边缘的中心。 边缘计算的输出被发送回主数据中心进一步处理。 边缘计算的示例包括腕表等挂件或分析交通流量的计算机。

在过去十年中，我们看到了越来越多的边缘设备（例如手机、智能手表或传感器）。 在某些情况下，实时数据处理不再是可有可无，而是至关重要。 想一想自动驾驶的汽车。 现在想象一下，来自汽车传感器的数据必须先传输到数据中心进行处理，然后再发送回车辆让它能够做出合适的反应。 固有的网络延迟可能是致命的。 虽然这是一个极端的例子，但大多数用户都不想使用无法提供即时反馈的智能设备。

如上所述，要使边缘设备有用，这些边缘设备必须至少在本地进行部分处理和分析，才能向用户提供近乎实时的反馈。 这是通过将一些存储和处理资源从数据中心转移到生成数据的位置（边缘设备）来实现的。 已处理和未处理的数据随后被发送到数据中心进一步处理和存储。 简而言之，效率和速度是边缘计算的主要驱动力。

- EndpointSlice

  一种将网络端点与 Kubernetes 资源组合在一起的方法。

  一种将网络端点组合在一起的可扩缩、可扩展方式。 它们将被 kube-proxy 用于在每个节点上建立网络路由。

- Endpoint，端点

  端点负责记录与服务 Service 的选择算符相匹配的 Pod 的 IP 地址。

  端点可以手动配置到 Service 上，而不必指定选择算符标识。

  EndpointSlice 提供了一种可扩缩、可扩展的替代方案。

- Envoy

  Envoy 是在服务网格里使用的高性能代理，用于为所有服务网格里的服务调度进出的流量。

- Ephemeral Container，临时容器

  一种可以在 Pod 中临时运行的容器类型。

如果想要调查运行中有问题的 Pod，可以向该 Pod 添加一个临时容器并进行诊断。 临时容器没有资源或调度保证，因此不应该使用它们来运行任何部分的工作负载本身。 静态 Pod 不支持临时容器。

- etcd

  一致且高度可用的键值存储，用作 Kubernetes 的所有集群数据的后台数据库。

  如果您的 Kubernetes 集群使用 etcd 作为其后台数据库，请确保您针对这些数据有一份备份计划。

- Event，事件

  对集群中某处所发生事件的报告。通常用来表述系统中某种状态变更。

  事件的保留时间有限，随着时间推进，其触发方式和消息都可能发生变化。事件用户不应该对带有给定原因（反映下层触发源）的时间特征有任何依赖， 也不要寄希望于该原因所造成的事件会一直存在。

  事件应该被视为一种告知性质的、尽力而为的、补充性质的数据。

  在 Kubernetes 中，审计机制会生成一种不同类别的 Event 记录（API 组为 audit.k8s.io）。

- Event-Driven Architecture，事件驱动架构

  事件驱动架构是一种提倡事件的创建、处理和消费的软件架构。 事件是对应用程序状态的任何更改。 例如，在拼车应用上叫车代表一个事件。 这种架构创建了一个结构，在该结构中，事件可以从它们的源（请求乘车的应用程序）正确地路由到所需的接收器（附近可用司机的应用程序）。

  随着越来越多的数据变得实时，寻找可靠的方法来确保捕获事件并将其路由到必须处理事件请求的适当服务变得越来越具有挑战性。 处理事件的传统

方法通常无法保证消息被恰当地路由或发送或接收。 随着应用程序的扩展，编排事件变得更具挑战性。

事件驱动架构为所有事件建立了一个中心枢纽（例如，Kafka）。 然后定义事件生产者（源）和消费者（接收者），中心事件枢纽保证事件的流动。这种架构确保服务保持解耦，并且事件从生产者正确路由到消费者。 生产者通常通过 HTTP 协议接收传入事件，然后路由事件信息。

- Eviction，驱逐

  终止节点上一个或多个 Pod 的过程。

  驱逐的两种类型：

  - 节点压力驱逐
  - API 发起的驱逐

- Extensions，扩展组件

  扩展组件（Extensions）是扩展并与 Kubernetes 深度集成以支持新型硬件的软件组件。

  许多集群管理员会使用托管的 Kubernetes 或其某种发行包，这些集群预装了扩展组件。 因此，大多数 Kubernetes 用户将不需要安装扩展组件，需要编写新的扩展组件的用户就更少了。

- External Control Plane，外部控制平面

  外部控制平面可以从外部管理运行在自己的集群或者其他基础设施中的网格工作负载。 控制屏幕可以部署在一个集群中，但是不能部署在它所控制的网格的一部分集群中。 它的目的是将控制平面与网格的数据屏幕完全分离。

- Firewall，防火墙

防火墙是一个基于特定规则过滤网络流量的系统。防火墙可以是硬件、软件，或者是两者的组合。

默认情况下，在遵循网络的路由规则下，网络将会允许任何人进出。由于这种默认行为，保护网络安全是有挑战性的。 例如，在基于微服务的银行应用程序中，服务之间的沟通是通过其网络来相互传递高敏感性财务数据。假如没有防火墙，恶意行为者可能会渗透网络、拦截通信并且造成破坏。

防火墙使用预设规则来检查网络流量。所有流量都会被过滤，任何来自不可信或可疑来源的流量都会被阻止。 只有设置为被接受的流量才能进入。防火墙在安全和受控的内部可信网络间建立了一道屏障。

- Finalizer

  一个带有命名空间的键，告诉 Kubernetes 等到特定的条件被满足后，再完全删除被标记为删除的资源。

  Finalizer 是带有命名空间的键，告诉 Kubernetes 等到特定的条件被满足后， 再完全删除被标记为删除的资源。Finalizer 提醒控制器清理被删除的对象拥有的资源。

  当您告诉 Kubernetes 删除一个指定了 Finalizer 的对象时， Kubernetes API 通过填充 .metadata.deletionTimestamp 来标记要删除的对象， 并返回 202 状态码(HTTP “已接受”）使其进入只读状态。 此时控制平面或其他组件会采取 Finalizer 所定义的行动， 而目标对象仍然处于终止中（Terminating）的状态。 这些行动完成后，控制器会删除目标对象相关的 Finalizer。 当 metadata.finalizers 字段为空时，Kubernetes 认为删除已完成并删除对象。

  您可以使用 Finalizer 控制资源的垃圾回收。 例如，您可以定义一个 Finalizer，在删除目标资源前清理相关资源或基础设施。

- Folder，文件夹，层级

为了满足企业内各个部门的分支划分，DCE 引入了层级的概念，通常层级对应着不同的部门，每个层级可以包含一个或多个工作空间。

G

- Garbage Collection，垃圾回收

  Kubernetes 用于清理集群资源的各种机制的统称。

  Kubernetes 使用垃圾回收机制来清理资源，例如：

  - 未使用的容器和镜像
  - 失败的 Pod
  - 目标资源拥有的对象
  - 已完成的 Job
  - 过期或出错的资源
- Gateway node，网关工作节点

  工作节点主要运行 Envoy 开源应用，主要提供高性能的反向代理能力，支持负载均衡、路由、缓存、自定义路由等功能。 工作节点的数量和性能将直接影响网关的性能，建议根据需要部署足够的工作节点。

- Gateway，网关规则

  在服务网格中，网关规则（Gateway）定义了在网格南北向连接操作的负载均衡器， 用于建立入站和出站的 HTTP/TCP 访问连接。它描述了需要公开的一组端口、服务域名、协议类型、负载均衡器的 SNI 配置等信息。

- Gauge，计量器

  计量器是一个 **既可增又可减** 的度量指标值。计量器主要用于测量类似于温度、内存使用量这样的瞬时数据。

- Global rate limit，全局限流

可选择增加网关限流组件，通过限流组件可以支持更多的流量管控能力。但是，限流组件也会导致一定的资源消耗和性能损失，默认情况下不启用。请根据实际情况判断是否需要启用。

- Grafana

  Grafana 是一个开源的可视化平台，提供了多样的监控数据可视化面板。

- Group

  在全局管理中，这指的是由多个用户形成的组合，即用户组。

  在微服务引擎中，这是 Nacos 中的一组配置集。

H

- Heartbeat，心跳

  实例启动后每隔一段时间，内置的 Nacos 客户端会主动向 Nacos 服务器发起心跳包（HeartBeat），表示实例仍存活，避免被服务端剔除。 心跳包包含当前服务实例的名称、IP、端口、集群名称、权重等信息。

- Helm Chart, Helm 模板

  Helm Chart 是一组预先配置的 K8s 资源所构成的包，可以使用 Helm 工具对其进行管理。

  Chart 提供了一种可重现的用来创建和共享 K8s 应用的方法。 单个 Chart 可用来部署简单的系统（例如 memcached Pod）， 也可以用来部署复杂的系统（例如：HTTP 服务器、数据库、缓存等组件的完整 Web 应用堆栈）。

- Histogram，直方图

  直方图对观测结果（通常是请求持续时间或者响应大小这样的数据）进行采样，并在可配置的桶中对其进行统计。 有以下几种方式来产生直方图（假设度量指标为 <basename>）：

- 按桶计数，相当于 <basename>_bucket{le="<upper inclusive bound>"}
- 采样值总和，相当于<basename>_sum
- 采样值总数，相当于 <basename>_count ，也等同于把所有采样值放到一个桶里来计数 <basename>_bucket{le="+Inf"}

Histogram 可以理解为柱状图，典型的应用如：请求持续时间，响应大小。可以对观测结果采样，分组及统计。

- Horizontal Pod Autoscaler, HPA, Pod 水平自动扩缩器

  Pod 水平自动扩缩器是一种 API 资源，它根据目标 CPU 利用率或自定义度量目标扩缩 Pod 副本的数量。

  HPA 通常用于 ReplicationController、Deployment 或者 ReplicaSet 上。 HPA 不能用于不支持扩缩的对象，例如 DaemonSet。

- Horizontal Scaling，水平扩缩

  水平扩缩是一种通过添加更多节点来增加系统容量的技术，而不是向单个节点添加更多计算资源（后者称为垂直扩缩）。 假设我们有一个 4GB RAM 的系统，并且想要将其容量增加到 16GB RAM，水平扩缩意味着通过添加 4 x 4GB RAM 而不是切换到 16GB RAM 系统来实现。

  这种方法通过添加新实例或节点来提高应用程序的性能，以更好地均衡工作负载。 简而言之，它旨在减少服务器的负载，而不是扩大单个服务器的容量。

  随着对应用程序的需求增长超出该应用程序实例的当前容量，我们需要找到一种方法来扩缩（增加容量）系统。 我们可以向系统添加更多节点（水平扩缩）或向现有节点添加更多计算资源（垂直扩缩）。

  水平扩缩允许应用程序在底层集群设置的范围内进行扩缩。通过向系统添加更多实例，应用程序可以处理更多请求。 如果单个节点每秒可以处理

1,000 个请求，则每增加一个节点，每秒的请求总数应该会增加大约 1,000 个。 这使得应用程序可同时执行更多工作，而无需特别增加任何单个节点的容量。

- HostAliases，主机别名

  主机别名是一组 IP 地址和主机名的映射，用于注入到 Pod 内的 host 文件。

  [HostAliases](https://kubernetes.io/docs/reference/generated/kubernetes-api/{{< param “version” >}}/#hostalias-v1-core) 是一个包含主机名和 IP 地址的可选列表，配置后将被注入到 Pod 内的 hosts 文件中。该选项仅适用于没有配置 hostNetwork 的 Pod。

- IAM, Identity and access management，用户与访问控制

  在全局管理中，IAM 是用户与访问控制的简称，管理员被称为 IAM Admin，拥有该模块的最高权限。 被赋予 IAM Admin 的用户（用户组）将拥有用户与访问控制的全部且最高权限。

- Image，镜像

  镜像是保存的容器实例，它打包了应用运行所需的一组软件。

  镜像是软件打包的一种方式，可以将镜像存储在容器镜像仓库、拉取到本地系统并作为应用来运行。 镜像中包含的元数据指明了运行什么可执行程序、是由谁构建的以及其他信息。

- Immutable Infrastructure，不可变基础设施

  不可变基础设施指的是一旦部署就无法变更的计算机基础设施（例如虚拟机、容器、网络设备）。 具体实现方式可以是通过自动化进程覆写所有未经授权的变更，也可以通过某个系统从一开始就不允许变更。 容器是不可

变基础设施的一个很好的例子，因为如果想要持久变更容器，只能通过创建新版本的容器或从其镜像重新创建同样的容器。

通过预防或识别未经授权的变更，不可变的基础设施可以更轻松地识别和减轻安全风险。 操作这种系统变得更加直接，因为管理员可以在某个假设的前提下放心地操作。 毕竟管理员知道没有人做过误操作，也无需担心发生过自己忘记沟通的某些变更。 不可变基础设施与基础设施即代码密切相关， 其创建基础设施所需的一切自动化都存放在版本控制（例如 Git）中。 这种不可变更和版本控制的结合意味着对系统的每次授权变更都会有一个持久的审计日志。

- Infrastructure as a service，基础设施即服务

  基础设施即服务，或者 IaaS ，是一种云计算服务模型， 它提供物理或虚拟的计算、存储和网络资源，使用按需按量的计费模式。 云提供商拥有和管理软件和硬件设施，可供消费者在公共、私有或混合云部署和使用。

  在搭建传统的本地设施时，组织常常受困于如何保证资源的有效利用。 数据中心建立时必须考虑潜在的高峰需求，即使这样的需求只占 1%的使用时间。 而在低需求期间，这些计算资源是空闲的。 而且，如果工作负载超过预期需求，处理工作负载的计算资源则会出现短缺。 这种缺乏可扩缩性的使用方式，将导致成本增加和资源使用率降低。

  通过 IaaS ，组织可以避免为其应用程序购买和维护计算和数据中心资源。按需使用的基础设施允许他们根据需要租用计算资源，并推迟大型资本支出， 或 CAPEX，同时给予他们扩大或缩小规模的灵活性。

  IaaS 降低了试验或尝试新应用程序的初期成本，并提供了快速部署基础设施的工具。 云提供商是开发或测试环境的绝佳选择，它可以帮助开发人员低成本的进行试验和创新。

- Infrastructure as code，基础设施即代码

基础设施即代码是将基础设施的定义存储为一个或多个文件的做法。 这取代了通常通过 shell 脚本或其他配置工具手动配置基础架构即服务的传统模型。

云原生方式构建应用程序要求基础设施是一次性的和可复现的。 它还需要以自动化和可重复的方式按需扩缩，甚至可能无需人工干预。 手动配置则无法满足云原生应用对响应能力和灵活扩缩的诉求。 而且手动基础架构变更不可复现，很容易碰到扩缩瓶颈，并引入错误配置。

通过将服务器、负载均衡器和子网等数据中心资源表示为代码， 它允许基础架构团队对所有配置拥有单一的数据源，并允许他们在 CI/CD 通道中管理数据中心，实现版本控制和部署策略。

- Ingress，路由

  Ingress 是对集群中服务的外部访问进行管理的 API 对象，典型的访问方式是 HTTP。

  Ingress 可以提供负载均衡、SSL 终结和基于名称的虚拟托管。

- Init Container，Init 容器

  应用容器运行前必须先运行完成的一个或多个 Init 容器。

  Init 容器像常规应用容器一样，只有一点不同：Init 容器必须在应用容器启动前运行完成。 Init 容器的运行顺序：一个 Init 容器必须在下一个 Init 容器开始前运行完成。

- Istio

  Istio 是一个免费开源的服务网格，提供了一种统一的方式来集成微服务、管理流量、实施策略和汇总度量数据。

  添加 Istio 时不需要修改应用代码。它是基础设施的一层，介于服务和网络之间。 当它和服务的 Deployment 相结合时，就构成了通常所谓的服务网

格（Service Mesh）。 Istio 的控制面抽象掉了底层的集群管理平台，这一集群管理平台可以是 Kubernetes、Mesosphere 等。

J, K

- Job

  Job 是需要运行完成的确定性的或批量的任务。

  创建一个或多个 Pod 对象，并确保指定数量的 Pod 成功终止。 随着各 Pod 成功结束，Job 会跟踪记录成功完成的个数。

- Kops

  Kops 是一个命令行工具，可以帮助您创建、销毁、升级和维护生产级，高可用性的 Kubernetes 集群。 Kops 目前仅支持 AWS。对 GCE、VMware vSphere 及其他第三方 PaaS 平台的支持还处于 Alpha 阶段。

  您也可以将自己的集群作为一个构造块，使用 kubeadm 构造集群。 kops 是建立在 kubeadm 之上的。

- kube-apiserver, k8s API 服务器

  kube-apiserver 是提供 Kubernetes API 服务的控制面组件。 Kubernetes API 服务器验证并配置 API 对象的数据，这些对象包括 Pod、Service、replicationcontroller 等。 API 服务器为 REST 操作提供服务，并为集群的共享状态提供前端， 所有其他组件都通过该前端进行交互。

  API 服务器是 Kubernetes 控制平面的组件，该组件负责公开了 Kubernetes API，负责处理接受请求的工作。 API 服务器是 Kubernetes 控制平面的前端。

  Kubernetes API 服务器的主要实现是 kube-apiserver。 kube-apiserver 设计上考虑了水平扩缩，也就是说，它可通过部署多个实例来进行扩缩。 您可以运行 kube-apiserver 的多个实例，并在这些实例之间平衡流量。

- kube-controller-manager

  kube-controller-manager 是一个守护进程，是 Master 节点上运行控制器的组件，负责运行控制器进程。

  从逻辑上讲，每个控制器都是一个单独的进程， 但是为了降低复杂性，它们都被编译到同一个可执行文件，并在同一个进程中运行。

- kube-proxy

  kube-proxy 是集群中每个节点上运行的网络代理，是实现 Kubernetes Service 概念的组成部分。

  kube-proxy 维护节点上的一些网络规则，这些网络规则会允许从集群内部或外部的网络会话与 Pod 进行网络通信。

  如果操作系统提供了可用的数据包过滤层，则 kube-proxy 会通过它来实现网络规则。否则，kube-proxy 仅做流量转发。

- kube-scheduler

  这是一个控制平面组件，负责监视新创建的、未指定运行节点的 Pod，选择节点让 Pod 在上面运行。

  调度决策考虑的因素包括单个 Pod 及 Pods 集合的资源需求、软硬件及策略约束、亲和性及反亲和性规范、数据位置、工作负载间的干扰及最后时限。

- Kubeadm

  用来快速安装 Kubernetes 并搭建安全稳定的集群的工具。您可以使用 Kubeadm 安装控制面和工作节点组件。

- kubectl

kubectl 是用来和 Kubernetes 集群进行通信的命令行工具。 您可以使用 kubectl 创建、检视、更新和删除 Kubernetes 对象。

- kubelet

  一个在集群中每个节点上运行的代理。它保证容器都运行在 Pod 中。

  kubelet 接收一组通过各类机制提供给它的 PodSpecs，确保这些 PodSpecs 中描述的容器处于运行状态且健康。 kubelet 不会管理不是由 Kubernetes 创建的容器。

- Kubernetes, K8s

  Kubernetes，通常缩写为 K8s，是一种流行的现代基础设施自动化的开源工具。 它就像一个数据中心的操作系统，管理在分布式系统上运行的应用程序（就像您笔记本上的操作系统，管理您的应用程序）。

  Kubernetes 在集群的节点上调度容器。 它捆绑了几个基础设施结构，有时被称为“基元”，如应用程序的实例、负载平衡器、持久性存储等，以一种可以被组成应用程序的方式。

  Kubernetes 实现了自动化和可扩展性，使用户能够以可重复的方式声明性地部署应用程序。 Kubernetes 生态系统中的软件产品和项目利用这种自动化和可扩展性来扩展 Kubernetes API。 这使他们能够利用 Kubernetes 的自动化，并使他们的工具更容易被有经验的 Kubernetes 从业者所接受。

  长期以来，基础设施自动化和声明性配置管理一直是重要的概念，而且随着云计算的普及而变得更加紧迫。 随着对计算资源的需求增加，组织感到压力，要用更少的工程师提供更多的操作能力，需要新的技术和工作方法来满足这一需求。 此外，云计算的兴起与容器化相搭配，那些忙于自动化更多传统基础设施的组织需要一种机制来自动配置和部署其容器。

Kubernetes 以类似于传统的基础设施即代码工具的方式帮助实现自动化，但它的优势在于，与虚拟机或物理机相比，容器更能抵抗配置漂移。Kubernetes 的工作方式是声明式的，这意味着操作者不是提供关于如何做某事的指示，而是描述（通常是 YAML 文档）他们想要做什么；Kubernetes 将自行处理“如何”的问题。这导致 Kubernetes 与基础设施即代码极为兼容。

Kubernetes 还能自我修复。这意味着它确保集群的实际状态总是与操作者的期望状态相匹配。 如果 Kubernetes 检测到一个偏差，Kubernetes 控制器就会启动并修复它。 因此，虽然它使用的基础设施可能不断变化，但 Kubernetes 本身也在不断自动适应变化，并确保它与预期状态相匹配。

- Kubernetes API

  Kubernetes API 是通过 RESTful 接口提供 Kubernetes 功能服务并负责集群状态存储的应用程序。

  Kubernetes 资源和”意向记录”都是作为 API 对象储存的，并可以通过调用 RESTful 风格的 API 进行修改。 API 允许以声明方式管理配置。 用户可以直接和 Kubernetes API 交互，也可以通过 kubectl 这样的工具进行交互。 核心的 Kubernetes API 是很灵活的，可以扩展以支持定制资源。

L

- Label，标签

  用来为对象设置可标识的属性标记；这些标记对用户而言是有意义且重要的。

  标签是一些关联到 Pod 这类对象上的键值对。它们通常用来组织和选择对象子集。

- LimitRange

提供约束来限制命名空间中每个容器或 Pod 的资源消耗。

LimitRange 按照类型来限制命名空间中对象能够创建的数量，以及单个容器（Containers）或 Pod 可以请求/使用的计算资源量。

- Load Balancer，负载均衡器

  负载均衡器是一种在后端的一组服务器之间分配传入网络流量的方法。 该解决方案可以是基于软件或硬件的。

  这有助于解决与高可用性和分布式系统相关的问题。 在处理需要扩展到数十万用户的应用程序或服务时，通常需要将该应用程序分布在多个服务器上。 负载均衡器是路由流量的“交通警察”。

  负载均衡器将充当网络流量和客户端的前端。 它通常有多种方法来检查哪些服务器已启动并且处理请求的负载最低。

- Log，日志

  在 Insight 中，日志是集群或应用程序记录的事件列表。 系统运行过程中变化的一种抽象数据，其内容为指定对象的操作和其操作结果按时间的有序集合。

  应用程序和系统日志可以帮助您了解集群内部发生的情况。日志对于调试问题和监视集群活动非常有用。

- Loosely coupled architecture，松耦合架构

  松耦合架构（紧耦合架构的相反范式）是一种架构风格，其中应用程序的各个组件彼此独立构建。 每个组件，有时称为微服务，都是为了执行特定功能而构建的，以便被任意数量的其他服务使用。 这种模式的实现通常比紧耦合架构慢。但有许多好处，特别是随着应用程序的不断扩展。 松耦合的应用程序允许团队独立开发功能、部署和扩展，这允许组织快速迭代各个组件。 应用程序开发速度更快，团队结构可以围绕应用程序的能力构建，专注于他们的特定应用程序。

M

- Managed Control Plane，托管控制平面

  托管控制平面是一个为客户提供管理的控制平面。 托管控制平面降低了用户部署的复杂性，并通常保证一定水平的性能和可用性。

- Managed Mesh，托管网格

  由服务网格在所选集群创建并托管的控制平面。具备简单、低成本、高可用、无需单独运维管理 的特点。

- Managed Service，托管服务

  托管服务是一种软件产品，其运营和管理由第三方负责。 例如类似 Amazon RDS 的数据库即服务或类似 Datadog 的外部监控服务。

  软件的管理比较复杂，尤其是要考虑现代技术栈所包含的各种不同技术。而想要将管理做到面面俱到并招募能胜任此职的内部专家，要么成本过于高昂，要么会耗用工程师的宝贵时间。 您的团队应投入精力构建新功能，而不是处理可以通过外包就能轻松解决的运营任务。

  托管服务从一开始就处于使用就绪状态，运营开销非常小。 托管服务具备良好定义的、通常由 API 驱动的边界， 便于各个组织将超出其核心竞争力的任务有效外包出去。

- Manifest，清单

  一个或多个 Kubernetes API 对象的序列化规范。

  清单指定了在应用该清单时 kubernetes 将维护的对象的期望状态。每个配置文件可包含多个清单。

- Master cluster，主集群

  主集群是具有控制平面的集群。 一个网格可以有一个以上的主集群，以用于 HA 或需要低延迟的场景。 主集群可以充当工作集群的控制平面。

- Metric，指标

  对资源性能的数据描述或状态描述，指标由命名空间、维度、指标名称和单位组成。 采集目标暴露的、可以完整反映监控对象运行或者业务状态的一系列标签化数据。

- Metadata，元数据

  元数据是数据本身的描述信息，是关于数据的数据，例如服务版本、各种自定义标签等。 元数据分为服务级别的元数据、集群的元数据及实例的元数据。

- Microservice，微服务

  微服务是一种通过多个小型服务组合来构建单个应用的架构风格。 在微服务引擎中，微服务指将一个完整的应用根据业务功能拆分而得到的各个小型服务。

  微服务是一种利用云原生技术进行应用开发的现代方法。 虽然现代应用程序，如 Netflix，看起来是一个单一的应用程序，但它们实际上是一个较小的服务的集合——所有的服务都密切配合。 例如，一个允许您访问、搜索和预览视频的单一页面很可能是由较小的服务提供的，它们各自处理其中的一个方面（如搜索、认证和在浏览器中运行预览）。 简而言之，微服务指的是一种应用架构模式，通常与单体应用形成对比。

  微服务是对单体应用所带来的挑战的一种回应。一般来说，一个应用程序的不同部分需要分别进行扩缩。 例如，一个在线商店将有更多的产品视图而不是结账。这意味着您需要更多的产品视图功能的运行，而不是结账。 在一个单一的应用程序中，这些逻辑位不能被单独部署。 如果您不能单独扩展产品功能，您将不得不复制整个应用程序和所有其他您不需要的组件：这是一种低效的资源利用。 单机式应用程序也使开发人员容易屈服于设计陷阱。 因为所有的代码都在一个地方，所以更容易使这些代码高耦合，更难

执行关注点分离的原则。 单机通常要求开发人员了解整个代码库，然后才能有成效。

将功能分离成不同的微服务，使它们更容易独立部署、更新和扩展。 通过允许不同的团队专注于更大的应用中他们自己的一小部分，也让他们更容易在不对组织的其他部分产生负面影响的情况下对他们的应用进行工作。 虽然微服务解决了许多问题，但它们也产生了运营开销：您需要部署和跟踪的东西增加了一个数量级或更多。 许多云原生技术旨在使微服务更容易部署和管理。

- Microservice instance，微服务实例

  将同一个微服务部署在多个容器或虚拟机上，每个容器或虚拟机上运行的微服务副本就是一个微服务实例。 多个微服务实例可以同时运行。

- Microservice instance group，微服务实例分组

  将同一个微服务下的所有微服务实例按照需求进一步划分得到的分组。

- Monolithic Apps，单体应用

  单体应用在一个简单可部署的程序中包含所有的功能。 在制作一个应用程序时，这通常是最简单和最容易的开始。 然而，一旦应用程序的复杂性增加，单体式就会变得难以维护。 随着更多的开发人员在同一个代码库上工作，发生冲突性变化的可能性和开发人员之间的人际沟通的需要就会增加。

  将一个应用程序转变成微服务会增加其运营开销——有更多的东西需要测试、部署和保持运行。 在产品生命周期的早期，推迟这种复杂性并建立一个单体应用，直到产品被确定为成功，可能是有利的。

  精心设计的单体可以坚持精益原则，因为它是启动和运行应用程序的最简单方式。 当单体应用的商业价值被证明是成功的，它可以被分解成微服务。 在证明有价值之前，制作一个基于微服务的应用程序可能是过早地花费了工程努力。 如果应用程序没有产生任何价值，这些努力就会被浪费掉。

- Multitenancy，多租户模式

  多租户模式指的是通过单次软件安装为多个租户提供服务。 租户是一个用户、应用程序或一组用户/应用程序，租户们使用各自的数据集来操控同一个软件。 这些租户不共享数据（除非软件的所有者明确授权），甚至可能未意识到彼此的存在。

  租户可以是小到只有一个登录 ID 的独立用户（就像单机版软件）， 也可以是大到拥有数千个登录 ID 的整个公司，其中每个登录 ID 有自己的权限但又以多种方式相互关联。 多租户软件示例包括 Google Mail、Google Docs、Microsoft Office 365、Salesforce CRM 和 Dropbox， 以及更多归类为具有完全或部分多租户能力的软件。

  如果没有多租户模式，每个租户都需要专门安装一次软件。 这会增加资源利用和维护的工作量，最终会加剧软件成本。

  多租户软件为每个租户提供一个隔离（工作数据、设置、凭证列表等）的环境，同时为多个租户提供服务。 从租户的角度来看，每个租户都有其专用的软件安装实例，尽管实际上他们是在共享同一个软件。 具体实现的方式为：在服务器上运行一个软件，然后允许租户通过网络接口和/或 API 连接到该软件。 使用多租户软件时，各个租户可以共享同一个安装实例，彼此毫无影响，且能以预先定义和受控的方式使用该软件。 软件提供商由此达成的资源节省也可以转而让租户受益，显著降低每个用户的软件成本（想想基于 Web 的电子邮件或文档编辑器）。

- Mutual Transport Layer Security, mTLS，双向传输层安全性协议

  双向 TLS (mTLS) 是一种用于对两个服务之间发送的消息进行身份验证和加密的技术。 双向 TLS (mTLS) 是标准的传输层安全性协议 (TLS)，但不是仅验证一个连接的身份，而是验证双方。

微服务通过网络进行通信，就像您的 WiFi 网络一样，通过该网络传输的通信可能会被黑客入侵。 mTLS 确保没有未经授权的一方监听或冒充合法请求。

mTLS 确保客户端和服务器之间的双向流量是安全和可信的， 为进入网络或应用程序的用户提供了额外的安全层。 它还验证不遵循登录过程的客户端设备连接，例如物联网（IoT）设备。 mTLS 可以防止诸如路径上的攻击、欺骗攻击、凭证填充、暴力攻击等攻击。

N

- Nacos 集群节点角色

  这是微服务引擎中节点在 Raft 协议中的角色。Raft 是一种实现分布式共识的协议，即如何让多个节点达成一致。

  Leader 是所有请求的处理者，负责接收客户端发起的操作请求，将请求写入本地日志并向其他节点同步请求日志。 任何时候最多只能有一个 Leader。

  Follower 是 Leader 的跟随者，负责从 Leader 接收更新请求并将其写入本地日志，并在 Leader 通知可以提交日志时提交日志。

- Name，名称

  客户端提供的字符串，用来指代资源 URL 中的对象，如 /api/v1/pods/some-name。

  某一时刻，只能有一个给定类型的对象具有给定的名称。但是，如果删除该对象，则可以创建同名的新对象。

- Namespace，命名空间

  命名空间是 Kubernetes 用来支持隔离单个集群中的资源组的一种抽象。

命名空间用来组织集群中对象，并为集群资源划分提供了一种方法。 同一命名空间内的资源名称必须唯一，但跨命名空间时不作要求。 基于命名空间的作用域限定仅适用于命名空间作用域的对象（例如 Deployment、Services 等）， 而不适用于集群作用域的对象（例如 StorageClass、Node、PersistentVolume 等）。 在一些文档里命名空间也称为命名空间。

在微服务引擎中，命名空间指的是 Nacos 命名空间，主要用于实现租户层级的配置隔离，例如隔离开发环境、测试环境、生产环境的资源配置。

- Network Policy，网络策略

  网络策略是一种规范，规定了允许 Pod 组之间、Pod 与其他网络端点之间以怎样的方式进行通信。

  网络策略帮助您声明式地配置允许哪些 Pod 之间、哪些命名空间之间允许进行通信， 并具体配置了哪些端口号来执行各个策略。NetworkPolicy 资源使用标签来选择 Pod， 并定义了所选 Pod 可以接受什么样的流量。网络策略由网络提供商提供的并被 Kubernetes 支持的网络插件实现。 请注意，当没有控制器实现网络资源时，创建网络资源将不会生效。

- Node，节点

  节点是一台能与其他计算机（或节点）协同工作以完成一个共同任务的计算机。 以您的笔记本电脑、调制解调器或打印机为例。它们都通过您的 wifi 网络进行通信和协作，各自代表一个节点。 在云计算中，节点可以是一台物理机，也可以是一台虚拟机（即 VM），甚至可以是容器。

  虽然一个应用程序可以（很多应用程序确实是）运行在一台独立的机器上，但这样做存在一些风险。也就是说，底层系统的故障将会破坏应用程序。为了解决这个问题，开发人员开始创建分布式应用程序，其中每个进程都在自己的节点上运行。 因此，节点作为集群或节点组的一部分运行着应用程序或进程，而这些节点一起工作以实现共同的目标。

节点为您提供了一个可以分配给集群的独特计算单元（内存、CPU、网络）。 在云原生平台或应用程序中，一个节点代表一个可执行工作的单元。 理想情况下，单个节点是不作区分的，因为任何特定类型的节点与其相同类型的节点应该是不可区分的。

- Node Pressure Eviction，节点压力驱逐

  节点压力驱逐是 kubelet 主动使 Pod 失败以回收节点上的资源的过程。

  kubelet 监控集群节点上的 CPU、内存、磁盘空间和文件系统 inode 等资源。 当这些资源中的一个或多个达到特定消耗水平时， kubelet 可以主动使节点上的一个或多个 Pod 失效，以回收资源并防止饥饿。

  节点压力驱逐不用于 API 发起的驱逐。

- Notification，通知

  当资源存在异常而产生告警时，可将告警信息通过邮件、钉钉、企业微信、webhook 等方式发送给指定的用户。

O

- Object，对象

  Kubernetes 系统中的实体，代表了集群的部分状态。Kubernetes API 用这些实体表示集群的状态。

  Kubernetes 对象通常是一个“目标记录”。 一旦您创建了一个对象，Kubernetes 控制平面就会持续工作，确保该对象所代表的项目切实存在。创建一个对象相当于告知 Kubernetes 系统：您期望这部分集群负载看起来像什么；这也就是您集群的期望状态。

- Objective，目标

  Prometheus 抓取的采集目标。采集目标暴露自身状态，或者代理暴露监控对象的运行、业务指标。

- Observability，可观测性

  可观测性指的是从所观测的系统采集信号，持续生成并发现可执行的洞察力。 换言之，可观测性允许用户从某个系统的外部输出中洞察该系统的状态并采取（修正）措施。

  计算机系统的衡量机制为观测 CPU 时间、内存、磁盘空间等底层信号以及每秒 API 响应次数、每秒错误率、每秒处理的事务数等高级信号和业务信号。

  系统的可观测性对其运营和开发成本有重大影响。 可观测系统为操作人员提供了有意义的、可执行的数据，使他们能够达成有利的结果 （即更快的事件响应、更高的开发效率）以及更少的艰辛时刻和更短的停机时间。

  请注意，更多的信息并不一定能转化为可观测性更好的系统。 事实上，有时系统生成的大量信息会形成信息噪音，会使得鉴别有价值的健康信号变得更加困难。 可观测性需要在合适的时间为合适的消费者（一个人或一个软件）提供合适的数据，从而做出合适的决策。

- Operator

  Operator 是打包，部署和管理 Kubernetes 应用程序的一种方法。

- Operator Pattern, Operator 模式

  一种用于管理自定义资源的专用控制器。 Operator 模式是一种系统设计，将控制器关联到一个或多个自定义资源。

  除了使用作为 Kubernetes 自身一部分的内置控制器之外，您还可以通过将控制器添加到集群中来扩展 Kubernetes。

  如果正在运行的应用程序能够充当控制器并通过 API 访问的方式来执行任务操控那些在控制平面中定义的自定义资源，这就是一个 Operator 模式的示例。

- OverridePolicy，差异化策略

  差异化策略是定义多云资源对象分发到不同工作集群时的差异化配置策略，例如在不同的工作集群中，可以使用不同的镜像，增加不同的标签等。

  OverridePolicy 作为一个独立的策略 API 能够自动处理集群相关的配置，例如：

  - 根据子集群的地域分布自动为镜像添加不同的前缀
  - 根据您的云提供商使用不同的 StorageClass

P

- Permission，权限

  权限指是否允许用户对某种资源执行某种操作。 为了降低使用门槛，DCE 采用 RBAC 模型将权限聚合成一个个角色，管理员只需要将角色授权给用户，该用户就一次性得到了该角色下聚合的一组权限。

  默认情况下，管理员创建的 IAM 用户没有任何角色权限，需要对其单独授予角色或将其加入用户组并给用户组授予角色，才能使得用户获得对应的角色权限，这一过程称为授权。授权后，用户就可以基于被授予的角色权限对平台资源进行操作。

- Persistent Volume Claim, PVC，持久卷申领

  申领在持久卷中定义的存储资源，以便可以将其挂载为容器中的卷。

  指定存储的数量，如何访问存储（只读、读写或独占）以及如何回收存储（保留、回收或删除）。 存储本身的详细信息在 PersistentVolume 对象中。

- Persistent Volume, PV，持久卷

  持久卷是代表集群中一块存储空间的 API 对象。它是通用的、可插拔的、并且不受单个 Pod 生命周期约束的持久化资源。

持久卷提供了一个 API，该 API 对存储的供应方式细节进行抽象，令其与使用方式相分离。 在提前创建存储（静态供应）的场景中，PV 可以直接使用。 在按需提供存储（动态供应）的场景中，需要使用 PersistentVolumeClaims (PVC)。

- Platform as a service, PaaS, 平台即服务

  平台即服务（PaaS）是应用程序开发团队部署和运行其应用程序的外部平台。 Heroku、Cloud Foundry、App Engine 是 PaaS 产品的示例。

  要利用好微服务或分布式应用程序等云原生模式， 运维团队和开发人员需要能够免去大量运维工作，其中包括供应基础设施、处理服务发现和负载平衡以及扩展应用程序等任务。

  平台即服务（PaaS）以完全自动化的方式为应用程序开发人员提供通用基础设施工具。 它使开发人员可以了解基础设施并减少对基础设施的担忧，并将更多的时间和精力用于编写应用程序代码。 它还提供了一些监控和可观测性来帮助应用程序团队确保他们的应用程序是健康的。

- Pilot

  Pilot 是服务网格里的一个组件，它控制 Envoy 代理，负责服务发现、负载均衡和路由分发。

- Pod

  Pod 表示集群上正在运行的一组容器。Pod 是 Kubernetes 的原子对象，是 Kubernetes 部署的一个工作负载实例。

  通常创建 Pod 是为了运行单个主容器。 Pod 还可以运行可选的边车（sidecar）容器，以添加诸如日志记录之类的补充特性。 通常用 Deployment 来管理 Pod。

  Pod 中包含了一个或多个共享存储和网络的容器以及如何运行容器的规约。

- Pod Disruption Budget, PDB

  Pod Disruption Budget 是这样一种对象：它保证在主动干扰（voluntary disruptions）时，多实例应用的 Pod 不会少于一定的数量。

  PDB 无法防止非主动的干扰，但是会计入预算（budget）。

- Pod Disruption, Pod 干扰

  主动或非主动地终止节点上的 Pod 的过程。

  主动干扰是由应用程序所有者或集群管理员有意启动的。非主动干扰是无意的， 可能由不可避免的问题触发，如节点耗尽资源或意外删除。

- Pod Lifecycle, Pod 生命周期

  关于 Pod 在其生命周期中处于哪个阶段的更高层次概述。

  Pod 生命周期是关于 Pod 处于哪个阶段的概述。包含了下面 5 种可能的阶段：

  - Running
  - Pending
  - Succeeded
  - Failed
  - Unknown

  关于 Pod 的阶段的更高级描述请查阅 [PodStatus](https://kubernetes.io/docs/reference/generated/kubernetes-api/{{< param “version” >}}/#podstatus-v1-core) phase 字段。

- Pod Priority, Pod 优先级

  Pod 优先级表示一个 Pod 相对于其他 Pod 的重要性。

Pod 优先级允许用户为 Pod 设置高于或低于其他 Pod 的优先级，这对于生产集群工作负载而言是一个重要的特性。

- Pod Security Policy, Pod 安全策略

  为 Pod 的创建和更新操作启用细粒度的授权。

  Pod 安全策略是集群级别的资源，它控制着 Pod 规约中的安全性敏感的内容。 PodSecurityPolicy 对象定义了一组条件以及相关字段的默认值，Pod 运行时必须满足这些条件。Pod 安全策略控制实现上体现为一个可选的准入控制器。

  PodSecurityPolicy 已于 Kubernetes v1.21 起弃用，并在 v1.25 中删除。作为替代方案，请使用 Pod 安全准入或第三方准入插件。

- Policy as code, PaC, 策略即代码

  策略即代码是将一些策略的定义存储为一个或多个机器可读和可处理格式文件的做法。 这取代了在传统模型中，以人类可读的形式记录在单独文档中的策略。

  构建应用和基础设施通常受到某组织所定义的许多策略的约束， 例如禁止在源代码中存储 Secret、禁止以超级用户权限运行容器或禁止将某些数据存储在特定地理区域之外的安全策略。 对于开发人员和审查人员来说，按照策略文档手动检查应用和基础设施既耗时费力又容易出错。 手动检查策略无法满足云原生应用的响应要求和扩缩要求。

  通过使用策略即代码，可以自动检查系统属性和操作。 软件开发的最佳实践也适用于构建策略即代码，例如使用 Git 及相关工作流。

- Portability, 可移植性

  可移植性是一种软件特征，一种可重用性的形式，有助于避免“锁定”到某些操作环境， 例如云提供商、操作系统或供应商。

传统软件通常是为特定环境（例如 AWS 或 Linux）构建的。 而可移植软件可以在不同的操作环境中工作，无需大量返工。 如果应用适配新环境所需的工作量在合理范围内，则该应用被认为是可移植的。“移植”这个词意味着修改软件并使其适应在不同的计算机系统上工作。

- Preemption，抢占

  Kubernetes 中的抢占逻辑通过驱逐节点上的低优先级 Pod 来帮助悬决的 Pod 找到合适的节点。

  如果一个 Pod 无法调度，调度器会尝试抢占较低优先级的 Pod，以使得悬决的 Pod 有可能被调度。

- Prometheus

  Prometheus 是一套开源的监控、报警、时间序列数据库的组合。

- PromQL

  Prometheus 内置的数据查询语言，其提供对时间序列数据丰富的查询，聚合以及逻辑运算能力的支持。

- PropagationPolicy，部署策略

  在多云编排中，PropagationPolicy 是定义多云资源对象的分发策略，支持使用指定集群、指定标签等方式来规划将资源部署到哪些工作集群。

  PropagationPolicy 是一种独立的策略 API，可以根据分发要求来定义多集群调度方式。

  - 支持 1:n 的策略:工作负载，用户每次创建多云应用时无需重复指出调度的约束条件。
  - 采用默认的策略时，用户可以直接与 Kubernetes API 交互。

- Proxy，代理

  在计算机领域，代理指的是充当远程服务中介的服务器。

客户端与代理进行交互；代理将客户端的数据复制到实际服务器；实际服务器回复代理；代理将实际服务器的回复发送给客户端。

kube-proxy 是集群中每个节点上运行的网络代理，实现了部分 Kubernetes Service 概念。

您可以将 kube-proxy 作为普通的用户态代理服务运行。 如果您的操作系统支持，则可以在混合模式下运行 kube-proxy；该模式使用较少的系统资源即可达到相同的总体效果。

R

- RBAC, Role-Based Access Control，基于角色的访问控制

  管理授权决策，允许管理员通过 Kubernetes API 动态配置访问策略。

  RBAC 使用角色（包含权限规则）和角色绑定（将角色中定义的权限授予一个用户组）。

- Registration center，注册中心

  注册中心相当于微服务架构中的“通讯录”，负责记录服务和服务地址的映射关系。 微服务引擎支持的注册中心类型有：Eureka、Zookeeper、Nacos、Mesh、Kubernetes。

- Reliability，可靠性

  从云原生的角度来看，可靠性是指系统对故障的响应能力。 如果我们有一个可以在基础架构更改和单个组件发生故障时继续工作的分布式系统，那么它是可靠的。 另一方面，如果它很容易出现故障，并且需要操作人员手动干预以保持其运行，则它是不可靠的。 云原生应用的目标是构建内在可靠的系统。

- ReplicaSet

  ReplicaSet 是下一代副本控制器。

ReplicaSet 就像 ReplicationController 那样，确保一次运行指定数量的 Pod 副本。 ReplicaSet 支持新的基于集合的选择器需求（在标签的用户指南中有相关描述），而副本控制器只支持基于等值的选择器需求。

- Replication Controller，副本控制器

  一种管理多副本应用的（已弃用）的 API 对象。

  一种工作管理多副本应用的负载资源，能够确保特定个数的 Pod 实例处于运行状态。

  控制面确保所指定的个数的 Pods 处于运行状态，即使某些 Pod 会失效， 比如被您手动删除或者因为其他错误启动过多 Pod 时。

  ReplicationController 已被弃用。请参阅 Deployment 执行类似功能。

- Resource，资源

  资源泛指 DCE 平台上通过各个子模块创建的资源，是完成授权的具体数据。 通常资源描述一个或多个操作对象，每个子模块拥有其各自的资源和对应的资源定义详情，如集群、Namesapce、网关等。

  资源的拥有者是主账号 Super Admin。 Super Admin 具有在各子模块创建/管理/删除资源的权限。 普通用户在没有授权的情况下，不会自动拥有资源的访问权限，需要 Super Admin 进行授权。 工作空间支持跨子模块授权用户（用户组）对于资源的访问权限。

- Resource limit，资源限制值

  限制值是实例的可用资源上限。请求值小于限制值。

- Resource Quota，资源配额

  资源配额提供了限制每个命名空间的资源消耗总和的约束。

限制了命名空间中每种对象可以创建的数量，也限制了项目中可被资源对象利用的计算资源总数。

- Resource request，资源请求值

  请求值规定了为实例预先分配多少可用资源。

- Resource Template，资源模板

  在多云编排中采用了一种叫做联邦资源的模板， 这是基于 K8s 原生 API 定义的一种多云资源模板，便于集成使用 K8s 生态范围内的所有云原生工具。

  通过这种资源模板可以统一管理多云服务、 多云命名空间、 多云配置项和多云密钥。

- Role，角色

  角色是连接用户与权限的桥梁， 一个角色对应一组权限，不同角色具有不同的权限。向用户授予某角色，即授予该角色所包含的所有权限。 全局管理中有两种角色：

  - 预定义角色：由系统创建，用户只能使用不能修改，每个子模块都有一个管理员 Admin 角色。
  - 自定义角色：用户自主创建、更新和删除，自定义角色中的权限由用户自己维护。 同时因为全局管理汇聚了多个子模块，各子模块也拥有相应的管理员角色，例如：
    - IAM Admin：管理用户与访问控制，即管理用户/用户组以及授权
    - Workspace Admin：管理层级及工作空间的权限，仅此权限可以创建层级
    - Audit Admin：管理审计日志

- Rolling update，滚动更新

滚动更新指一次只更新一小部分副本、成功后再更新更多的副本、最终完成所有副本的更新。 滚动更新的最大的好处是零停机，整个更新过程始终有副本在运行，从而保证了业务的连续性。

- Routing Rule，路由规则

  在服务网格的虚拟服务中配置的路由规则，遵循服务网格定义了请求的路径。 使用路由规则，可以定义将寻址到虚拟服务主机的流量路由到指定目标的工作负载。 路由规则使您可以控制流量，以实现按百分比分配流量的分阶段等任务。

S

- Scalability，可扩缩性

  可扩缩性指的是一个系统能有多大的发展。这就是增加做任何系统应该做的事情的能力。 例如，Kubernetes 集群通过增加或减少容器化应用程序的数量来进行扩缩，但这种可扩缩性取决于几个因素。 它有多少节点，每个节点可以处理多少个容器，控制平面可以支持多少条记录和操作?

  可扩缩的系统使添加更多容量更容易。主要有两种扩缩方法。 一方面，有水平扩缩添加更多节点来处理增加的负载。 相比之下，在垂直扩缩中，单个节点的功能更强大，可以执行更多事务（例如，通过向单个机器添加更多内存或 CPU）。 可扩缩的系统能够轻松更改并满足用户需求。

- Secret

  Secret 用于存储敏感信息，如密码、 OAuth 令牌和 SSH 密钥。

  Secret 允许用户对如何使用敏感信息进行更多的控制，并减少信息意外暴露的风险。 默认情况下，Secret 值被编码为 base64 字符串并以非加密的形式存储， 但可以配置为静态加密（Encrypt at rest）。

  Pod 可以通过多种方式引用 Secret，例如在卷挂载中引用或作为环境变量引用。Secret 设计用于机密数据，而 ConfigMap 设计用于非机密数据。

- Security Context，安全上下文

  securityContext 字段定义 Pod 或容器的特权和访问控制设置，包括运行时 UID 和 GID。

  在一个 securityContext 字段中，您可以设置进程所属用户和用户组、权限相关设置。 您也可以设置安全策略（例如 SELinux、AppArmor、seccomp）。

  PodSpec.securityContext 字段配置会应用到一个 Pod 中的所有的容器。

- Selector，选择算符

  选择算符允许用户通过标签对一组资源对象进行筛选过滤。

  在查询资源列表时，选择算符可以通过标签对资源进行过滤筛选。

- Serverless

  Serverless 是一种云原生开发模型，允许开发人员构建和运行应用程序，而无需管理服务器。 Serverless 中仍有服务器，但它们被抽象出来，远离应用程序开发。 云提供商处理配置、维护和扩缩服务器基础架构的日常工作。 开发人员可以简单地将他们的代码打包在容器中进行部署。 部署后，Serverless 应用程序会响应需求并根据需要自动扩展和缩减。 公共云提供商的 Serverless 产品通常通过事件驱动的执行模型按需计量。 因此，当无服务器功能处于空闲状态时，它不会花费任何费用。

  在标准的基础设施即服务（IaaS）云计算模型下，用户预先购买容量单位，这意味着您需要向公共云提供商支付永远在线的服务器组件的费用来运行您的应用程序。 用户有责任在高需求时扩缩服务器容量，并在不在需要该容量时缩减容量。 即使在不使用应用程序时，运行应用程序所需的云基础设施也处于活动状态。

  相比之下，使用 Serverless 架构，应用程序仅在需要时启动。 当事件触发应用程序代码运行时，公共云提供商会为该代码动态分配资源。 当代码

执行完成后，用户就停止为资源付款。 除了成本和效率优势之外，Serverless 还使开发人员从与应用程序扩展和服务器配置相关的日常和琐碎任务中解放出来。 借助 Serverless，管理操作系统和文件系统、安全补丁、负载平衡、容量管理、扩缩、日志记录和监控等日常任务都被交给云服务提供商。

- Service，服务

  请注意，在 IT 中，服务有多种含义。 在这个定义中，我们将关注更传统的定义：微服务中的服务。 服务与微服务哪里有、是否有区别是细微的，不同的人可能有不同的看法。 在更高层次的定义中，我们将它们视为相同。具体请参考微服务的定义。

  服务所针对的 Pod 集合（通常）由选择算符确定。 如果有 Pod 被添加或被删除，则与选择算符匹配的 Pod 集合将发生变化。 服务确保可以将网络流量定向到该工作负载的当前 Pod 集合。

- ServiceAccount

  为在 Pod 中运行的进程提供标识。

  当 Pod 中的进程访问集群时，API 服务器将它们作为特定的服务帐户进行身份验证，例如 default。 创建 Pod 时，如果您没有指定服务账号，它将自动被赋予同一个{{< glossary_tooltip text="命名空间" term_id="namespace" >}}中的 default 服务账号。

- Service Catalog，服务目录

  服务目录是一种扩展 API，它能让 Kubernetes 集群中运行的应用易于使用外部托管的软件服务，例如云供应商提供的数据仓库服务。

  服务目录可以检索、供应并绑定外部托管服务（Managed Services），而无需知道那些服务具体是怎样创建和托管的。

- Service Discovery，服务发现

服务发现是查找组成服务各个实例的过程。 服务发现工具持续跟踪构成服务的各种节点或端点。

云原生架构是动态的和不确定的，这意味着它们不断在变化。 容器化的应用程序在其生命周期内可能会多次启动和停止。 每次这种情况发生时，它都会有一个新地址，任何应用程序想要找到它，都需要一个工具来提供新的地址信息。

服务发现持续跟踪网络中的应用程序，以便在需要时可以找到彼此。 它提供了一个公共的地方来查找和识别不同服务。 服务发现引擎是类似数据库的工具，用于存储当前有哪些服务以及如何找到它们。

- Service Entry，服务条目

  在服务网格中，服务条目用于将一个无法注册到服务注册表的网格内部服务（例如：vm）或网格外部服务器添加到服务网格抽象模型中。 添加服务条目后，Envoy 代理可以将流量发送到该服务，这个服务条目将和网格中的其他服务一样。

- Service Mesh，服务网格

  在微服务的理念里，应用程序被分解成多个较小的服务，通过网络进行通信。 就像您的 WIFI 网络一样，计算机网络本质上是不可靠的，可被黑客攻击的，而且往往很慢。 服务网格通过管理服务之间的流量（即通信），并在所有服务中统一添加可靠性、可观测性和安全功能来解决这一系列新的挑战。

  在转向微服务架构后，工程师们现在要处理数百个，甚至数千个单独的服务，都需要进行通信。 这意味着大量的流量在网络上来回传输。 除此之外，单个应用程序可能需要对通信进行加密，以支持监管要求，为运营团队提供通用指标，或提供对流量的详细洞察，以帮助诊断问题。 如果内置于单个应用程序中，这些功能中的每一个都会引起团队间的冲突，并减缓新功能的开发。

服务网格在集群的所有服务中统一增加了可靠性、可观测性和安全功能，而不需要改变代码。 在服务网格之前，这些功能必须被编码到每一个服务中，成为错误和技术债务的潜在来源。

- Service Name，服务名称

  服务名称是服务 Service 唯一的名字，是 Service 在服务网格里的唯一标识。服务名称是唯一的，不得重复。 一个服务有多个版本，但是服务名是与版本独立的。

- Service Operator

  Service Operator 是在服务网格里管理 Service 的代理， 通过操纵配置状态并通过各种仪表板监视服务的运行状况来管理这些服务。

- Service Proxy，服务代理

  服务代理拦截进出某项服务的流量，对其应用一些逻辑，然后将该流量转发给另一项服务。 它本质上是一个“中间人”，收集有关网络流量的信息，并决定是否对其应用规则。

  为了跟踪服务与服务之间的通信（又称网络流量），并可能对其进行转换或重定向，我们需要收集数据。 传统上，实现数据收集和网络流量管理的代码被嵌入每个应用程序中。

  服务代理允许我们将这种功能“外部化”。它不再需要生活在应用程序中。相反，它现在被嵌入到平台层中（您的应用程序运行的地方）。

  作为服务之间的守门员，代理提供对正在发生的通信类型的洞察力。 根据他们的洞察力，他们决定将一个特定的请求发送到哪里，甚至完全拒绝它。

  代理人收集关键数据，管理路由（在服务之间平均分配流量，或在某些服务中断时重新路由），加密连接，并缓存内容（减少资源消耗）。

- Service Registry，服务注册表

服务网格维护了一个内部服务注册表（service registry），包含在服务网格中运行的一组服务及其相应的服务 endpoints。服务网格使用服务注册表生成 Envoy 配置。

- Self Healing，自愈

  一个自愈系统无需任何人为干预就能从某些类型的故障中恢复。 像 DCE 5.0 这种系统自带一个“收敛”或“控制”循环，可以主动查看系统的实际状态并将其与运营商最初期望的状态进行比较。 如果有所差异（例如运行的应用程序实例数少于预期实例数），系统将自动采取修正措施（例如启动新的实例或 Pod）。

- Shuffle Sharding，混排切片

  一种将请求指派给队列的技术，其隔离性好过对队列个数哈希取模的方式。

  我们通常会关心不同的请求序列间的相互隔离问题，目的是为了确保密度较高的请求序列不会湮没密度较低的序列。 将请求放入不同队列的一种简单方法是对请求的某些特征值执行哈希函数，将结果对队列的个数取模，从而得到要使用的队列的索引。 这一哈希函数使用请求的与其序列相对应的特征作为其输入。例如在互联网上，这一特征通常指的是由源地址、目标地址、协议、源端口和目标端口所组成的五元组。

  这种简单的基于哈希的模式有一种特性，高密度的请求序列（流）会湮没那些被哈希到同一队列的其他低密度请求序列（流）。 为大量的序列提供较好的隔离性需要提供大量的队列，因此是有问题的。 混排切片是一种更为灵活的机制，能够更好地将低密度序列与高密度序列隔离。 混排切片的术语采用了对一叠扑克牌进行洗牌的类比，每个队列可类比成一张牌。 混排切片技术首先对请求的特定于所在序列的特征执行哈希计算，生成一个长度为十几个二进制位或更长的哈希值。 接下来，用该哈希值作为信息熵的来源，对一叠牌来混排，并对整个一手牌（队列）来洗牌。 最后，对所有处理过的队列进行检查，选择长度最短的已检查队列作为请求的目标队列。

在队列数量适中的时候，检查所有已处理的牌的计算量并不大，对于任一给定的低密度的请求序列而言，有相当的概率能够消除给定高密度序列的湮没效应。 当队列数量较大时，检查所有已处理队列的操作会比较耗时，低密度请求序列消除一组高密度请求序列的湮没效应的机会也随之降低。因此，选择队列数目时要颇为谨慎。

- Site Reliability Engineering, SRE，网站可靠性工程

  网站可靠性工程（SRE） 是一门结合运营和软件工程的学科。 后者特别适用于基础设施和运营问题。 这意味着，网站可靠性工程师不是构建产品功能，而是构建系统来运行应用程序。 与 DevOps 有相似之处，但 DevOps 专注于将代码投入生产环境， 而 SRE 是确保在生产环境中运行的代码正常工作。

  确保应用程序可靠运行需要多种功能， 从性能监控、警报、调试到故障排除。 没有这些，系统操作员只能对问题做出反应，而不是主动努力避免它们，因为停机只是时间问题。

  网站可靠性工程通过不断改进底层系统来最小化软件开发过程的成本、时间和工作量。 该系统持续测量和监控基础设施和应用程序组件。 当出现问题时，系统会提示网站可靠性工程师何时、何地以及如何修复它。 这种方法通过自动化任务来帮助创建高度可扩展和可靠的软件系统。

- Software as a service, SaaS，软件即服务

  软件即服务（SaaS）允许用户通过互联网连接或使用云服务。 常见的例子有电邮、日历和办公工具（例如 Gmail、AWS、GitHub、Slack）。 SaaS 以按需付费的方式提供完整的软件解决方案。 所有运维任务和应用数据由服务提供商处理。

  传统的商业软件被安装在独立的计算机上，需要管理员维护和更新。 例如：某组织可能在企业内部使用客户关系管理（CRM）软件。 该软件需要内部 IT 部门采购、安装、确保安全、维护和定期升级，为 IT 团队增加了

负担。 与许可证、安装和潜在附加硬件相关的前期成本可能令人望而却步。 也很难按需响应，很难随着业务增长或变化快速扩缩。

SaaS 应用无需内部 IT 部门付出任何特别努力即可工作。 这些应用由供应商安装、维护、升级和确保安全。 扩缩、可用性和容量问题由服务提供商处理，采用按需付费的模式。 对于想要使用企业级应用的各个组织而言，SaaS 是一种经济实惠的方式。

- Stateful Apps，有状态应用

  当我们说到有状态（和无状态）应用时，状态是指应用需要存储以便其按设计运行的任何数据。 例如，任何能记住您购物车的在线商店都是有状态应用。

  使用一个应用通常需要多个请求。 例如，使用网上银行时，您将通过输入密码（请求 #1）来验证自己的身份， 然后您可以将钱转给某个朋友（请求 #2），最后您需要查看转账详情（请求 #3）。 为了保证正常运行，每一步都必须记住前面的步骤，银行需要记住每个人的账号状态。 今天我们使用的大多数应用至少总有一部分是有状态的， 因为这些应用会存储诸如偏好和设置之类的东西以改善用户体验。

  有几种方法可以为有状态应用存储状态。 最简单的是将状态保存在内存中，而不是将其持久保存在任何其他地方。 这样做的问题是，每次应用必须重启时，所有状态都会丢失。 为了防止这种情况发生，状态被持久存储在本地（磁盘上）或数据库系统中。

- StatefulSet

  StatefulSet 用来管理某 Pod 集合的部署和扩缩，并为这些 Pod 提供持久存储和持久标识符。

  与 Deployment 类似，StatefulSet 管理基于相同容器规约的一组 Pod。但和 Deployment 不同的是，StatefulSet 为它们的每个 Pod 维护了一个

有粘性的 ID。 这些 Pod 是基于相同的规约来创建的，但是不能相互替换：无论怎么调度，每个 Pod 都有一个永久不变的 ID。

如果希望使用存储卷为工作负载提供持久存储，可以使用 StatefulSet 作为解决方案的一部分。 尽管 StatefulSet 中的单个 Pod 仍可能出现故障，但持久的 Pod 标识符使得将现有卷与替换已失败 Pod 的新 Pod 相匹配变得更加容易。

- Stateless Apps，无状态应用

  无状态应用不会在应用所在的服务器上保存任何客户端会话（状态）数据。每个会话都像第一次一样执行，响应不依赖于前一个会话的数据并提供使用打印服务、内容分发网络（CDN）或 Web 服务器的功能，以处理每个短期请求。 例如，有人在搜索引擎中搜索某个问题并按下了回车键。 如果搜索操作由于某种原因被中断或关闭，您必须重新开始一个新的，因为您之前的请求没有保存数据。

  无状态应用解决了弹性问题，因为集群中的不同 Pod 可以独立工作， 同时允许多个请求到达这些 Pod。 如果应用出现故障，您只需重启应用，就能恢复到初始状态，停机时间很短或没有停机时间。 因此，无状态应用的好处包括坚韧、弹性和高可用性。 然而，我们今天使用的大多数应用至少总有一部分是有状态的， 因为它们会存储诸如偏好和设置之类的东西以改善用户体验。

  归根结底，在无状态应用中集群唯一负责的是托管在其上的代码和其他静态内容。 就是这样，不更改数据库，没有写入，删除 Pod 时也没有留下文件。 无状态容器更易于部署，您无需担心如何将容器数据保存在持久存储卷上。 您也不必担心备份数据。

- Static Pod，静态 Pod

  静态 Pod 是指由特定节点上的 kubelet 守护进程直接管理的 Pod。 API 服务器不会察觉这种 Pod。静态 Pod 不支持临时容器。

- StorageClass

  StorageClass 是管理员用来描述可用的不同存储类型的一种方法。

  StorageClass 可以映射到服务质量等级（QoS）、备份策略、或者管理员任意定义的策略。 每个 StorageClass 对象包含的字段有 provisioner、parameters 和 reclaimPolicy。 动态制备该存储类别的持久卷时需要用到这些字段值。 通过设置 StorageClass 对象的名称，用户可以请求特定存储类别。

- sysctl

  用于获取和设置 Unix 内核参数的接口。sysctl 是一个半标准化的接口，用于读取或更改正在运行的 Unix 内核的属性。

  在类 Unix 系统上，sysctl 既是管理员用于查看和修改这些设置的工具的名称，也是该工具所调用的系统调用的名称。

  容器运行时和网络插件可能对 sysctl 的取值有一定的要求。

- Summary，汇总

  类似于直方图，汇总也对观测结果进行采样。除了可以统计采样值总和和总数，它还能够按分位数统计。有以下几种方式来产生汇总（假设度量指标为 <basename>）：

  - 按分位数，也就是采样值小于该分位数的个数占总数的比例小于 ϕ，相当于 <basename>{quantile="<ϕ>"}
  - 采样值总和，相当于 <basename>_sum
  - 采样值总数，相当于 <basename>_count

T

- Taint，污点

污点是一种核心对象，包含三个必需的属性：key、value 和 effect。污点会阻止在节点或节点组上调度 Pod。

污点配合容忍度一起工作，以确保不会将 Pod 调度到不适合的节点上。同一节点上可标记一个或多个污点。节点应该仅调度那些带着能与污点相匹配容忍度的 Pod。

- Temporary microservice instance，临时微服务实例

  不能持久化存储在 Nacos 服务端的微服务实例。 临时微服务实例需要通过上报心跳的方式进行包活，如果一段时间内没有上报心跳，就会被 Nacos 服务端摘除。

- Threshold，保护阈值

  阈值是一个 0 到 1 之间的浮点数。 当健康实例数占总服务实例数的比例小于该值时，无论实例是否健康，都会将这个实例返回给客户端。 这样可以防止流量压力把健康实例压垮并形成雪崩效应。

- Tightly Coupled Architecture，紧耦合架构

  紧耦合架构（松耦合架构的相反范式）是一种架构风格，其中许多应用程序组件相互依赖。 这意味着一个组件的更改可能会影响其他组件。 它通常比松耦合架构更容易实现，但会使系统更容易受到级联故障的影响。 它还意味着需要协调各个组件的部署，这可能会拖累开发人员的生产力。

  紧耦合应用程序架构是一种相当传统的应用程序构建方式。 在某些特定情况下，当我们不需要与微服务开发的所有最佳实践一致时，它将变得很有用。 这意味着更快、更简单地实现， 和单体应用很像，可以加快最初的开发周期。

- Toleration，容忍度

  一个核心对象，由三个必需的属性组成：key、value 和 effect。 容忍度允许将 Pod 调度到具有对应污点的节点或节点组上。

容忍度和污点共同作用可以确保不会将 Pod 调度在不适合的节点上。 在同一 Pod 上可以设置一个或者多个容忍度。 容忍度表示在包含对应污点的节点或节点组上调度 Pod 是允许的（但不必要）。

- Transport Layer Security, TLS, 传输层安全性协议

  传输层安全性协议（TLS）是一种旨在为网络通信提供更高安全性的协议。它确保通过互联网发送的数据安全交付，避免可能的数据监视和/或篡改。该协议广泛用于消息传递、电子邮件等应用程序中。

  如果没有 TLS，网页浏览习惯、电子邮件通信、在线聊天和电话会议等敏感信息在传输过程中很容易被他人追踪和篡改。 启用服务器和客户端应用程序对 TLS 的支持，可以确保它们之间传输的数据是加密的，并且第三方无法查看。

  TLS 使用多种编码技术，在通过网络传输数据时提供安全性。 TLS 允许客户端应用程序和服务器（如浏览器和银行站点）之间的加密连接。 它还允许客户端应用程序积极地识别他们正在调用的服务器，从而降低客户端与欺诈站点通信的风险。 这可以确保第三方无法查看和监控使用 TLS 在应用程序之间传输的数据，从而保护敏感隐私的信息，例如信用卡号、密码、位置等。

- Trace，链路

  记录单次请求范围内的处理信息，其中包括服务调用和处理时长等数据。一个 Trace 有一个唯一的 Trace ID ，并由多个 Span 组成。

U, V

- UID

  由 Kubernetes 系统生成、用来唯一标识对象的字符串。

  在 Kubernetes 集群的整个生命周期中创建的每个对象都有一个不同的 UID，它旨在区分类似实体的历史事件。

- User，用户

  用户是发起操作的主体，每个用户都有唯一的 ID，并被授予不同的角色。 默认创建的 IAM 用户没有任何权限，需要将其加入用户组，授予角色或策略，才能让用户获得对应的权限。

  用户以用户名登录 DCE，按照被授予的权限操作平台资源和服务。 所以用户是资源归属的主体，对其拥有的资源具有相应权限。

  用户可以在个人中心修改用户信息，设置密码、访问密钥和 UI 语言。

- User namespace，用户命名空间

  一种为非特权用户模拟超级用户特权的 Linux 内核功能特性。

  用来模拟 root 用户的内核功能特性，用来支持“Rootless 容器”。

  用户命名空间（User Namespace）是一种 Linux 内核功能特性，允许非 root 用户 模拟超级用户（“root”）的特权，例如用来运行容器却不必成为容器之外的超级用户。

  用户命名空间对于缓解因潜在的容器逃逸攻击而言是有效的。

  在用户命名空间语境中，命名空间是 Linux 内核的功能特性而不是 Kubernetes 意义上的命名空间概念。

- Version Control，版本控制

  源代码管理（或版本控制）是一种跟踪和管理文档更改的行为。 它是一个持续记录单个文件或一组文件变化的系统，以便您在以后可以回退到特定版本。

  版本控制系统致力于解决以下问题， 备份随时间变化的文档或代码库， 允许在多个用户存在交叉修改时解决冲突，并随时间存储更改日志。 处理关键业务的应用程序代码通常复杂且重要， 因此，跟踪谁更改了内容、什么时候更改的以及更改原因是非常重要的。 此外，许多（甚至可以说大部

分）应用程序是由多个开发人员修改的，并且不同开发人员引入的更改之间经常存在冲突。

版本控制可帮助开发人员快速行动并保持效率，同时存储更改记录并提供解决冲突的工具。 它可以将应用程序代码存储在代码仓库中并简化开发人员间的协作。 现代应用程序开发非常依赖版本控制系统，如 git，来存储他们的代码。

- Vertical Scaling，垂直扩缩

  垂直扩缩，也称为“向上和向下扩缩”，是一种通过在工作负载增加时向单个节点添加 CPU 和内存来增加系统容量的技术。 假设您有一台 4GB RAM 的计算机，并且想要将其容量增加到 16GB RAM，垂直扩缩就意味着切换到 16GB RAM 系统。 （请参阅水平扩缩了解不同的扩缩方法。）

  随着对应用程序的需求增长超出该应用程序实例的当前容量，我们需要找到一种方法来伸展（增加容量）系统。 我们可以向现有节点添加更多计算资源（垂直扩缩）或向系统添加更多节点(水平扩缩)。 可扩缩性有助于提高竞争力、效率、声誉和质量。

  垂直扩缩允许您在不更改应用程序代码的情况下调整服务器大小。 这与水平扩缩形成对比，在水平扩缩中，应用程序必须可以被复制来进行扩缩，而这可能需要代码更新。 垂直扩缩通过添加计算资源来增加现有应用程序的容量，允许应用程序处理更多请求并同时执行更多工作。

- Virtual Machine，虚拟机

  虚拟机（VM）是一台计算机及其操作系统，不受特定硬件的约束。 虚拟机依靠虚拟化将一台物理计算机分割成多个虚拟计算机。 这种分离使组织和基础设施供应商能够轻松地创建和销毁虚拟机，而不影响底层硬件。

  虚拟机利用了虚拟化的优势。 当裸机机器被束缚在一个单一的操作系统上时，该机器的资源的使用受到一定的限制。 另外，当一个操作系统被绑定

在一个单一的物理机上时，它的可用性直接与该硬件联系在一起。 如果物理机由于维护或硬件故障而脱机，操作系统也会脱机。

通过消除操作系统和单一物理机之间的直接关系，您解决了裸机的几个问题：配置时间、硬件利用率和弹性。

由于不需要购买、安装或配置新的硬件来支持它，新计算机的配置时间得到了极大的改善。 虚拟机通过在一台物理机上放置多个虚拟机，使您能够更好地利用现有的物理硬件资源。 不受特定物理机的约束，虚拟机也比物理机更有弹性。 当一台物理机需要下线时，在其上运行的虚拟机可以被转移到另一台机器上，几乎没有停机时间。

- Virtualization，虚拟化

  虚拟化，在云原生计算的背景下，是指将一台物理计算机，有时称为服务器，并允许它运行多个隔离的操作系统的过程。 这些隔离的操作系统及其专用的计算资源（CPU、内存和网络）被称为虚拟机或 VM。 当我们谈论虚拟机时，我们在谈论一个软件定义的计算机。 它看起来和行动都像一台真正的计算机，但与其他虚拟机共享硬件。 举个例子，您可以从 AWS 租赁一台“计算机”，该计算机实际上是一个虚拟机。

  虚拟化解决了许多问题，包括通过允许更多的应用程序在同一台物理机器上运行，同时为了安全起见仍然相互隔离，从而改善物理硬件的使用。

  在虚拟机上运行的应用程序没有意识到他们正在共享一台物理计算机。 虚拟化还允许数据中心的用户在几分钟内启动一台新的“计算机”（又称虚拟机），而不必担心在数据中心增加一台新计算机的物理限制。 虚拟机还使用户能够加快获得新的虚拟计算机的时间。

- Virtual Service，虚拟服务

  虚拟服务定义了一系列针对指定服务的流量路由规则。 每个路由规则都针对特定协议定义流量匹配规则。 如果流量符合这些特征，就会根据规则发送到服务注册表中的目标服务（或者目标服务的子集或版本）。

- Volume，卷

  包含可被 Pod 中容器访问的数据目录。

  每个 Kubernetes 卷在所处的 Pod 存续期间保持存续状态。 因此，卷的生命期会超出 Pod 中运行的容器，并且保证容器重启之后仍保留数据。

- Volume Plugin，卷插件

  卷插件可以让 Pod 集成存储。

  卷插件让您能给 Pod 附加和挂载存储卷。 卷插件既可以是 **in tree** 也可以是 **out of tree** 。 **in tree** 插件是 Kubernetes 代码库的一部分，并遵循其发布周期。 而 **Out of tree** 插件则是独立开发的。

W, Z

- Weight，权重

  权重为浮点数。权重越大，表示分配给该实例的流量越大。

- Workload，工作负载

  工作负载是在 Kubernetes 上运行的应用程序。

  代表不同类型或部分工作负载的各种核心对象包括 Deployment、StatefulSet、DaemonSet、Job、ReplicaSet。

  例如，具有 Web 服务器和数据库的工作负载可能在一个 StatefulSet 中运行数据库，而 Web 服务器运行在 Deployment。

- Workload Instance，工作负载实例

  在服务网格中，工作负载实例是工作负载的一个二进制实例化对象。 一个工作负载实例可以开放零个或多个服务 endpoint，也可以消费零个或多个服务。 工作负载实例具有许多属性：名称和命名空间、IP 地址、唯一的 ID、标签、主体等

- Workload Instance Principal，工作负载实例主体

  在服务网格中，工作负载实例主体是工作负载实例的可验证权限。服务网格的服务到服务身份验证用于生成工作负载实例主体。 默认情况下，工作负载实例主体与 SPIFFE ID 格式兼容。

- Workspace，工作空间

  工作空间是一种资源范畴，代表一种资源层级关系。 工作空间可以包含集群、命名空间、注册中心等资源。 通常一个工作空间对应一个项目，可以为每个工作空间分配不同的资源，指派不同的用户和用户组。

- Worker Cluster，工作集群

  工作集群是一个连接到集群外部控制平面的集群。 工作集群可以连接到主集群的控制平面，或连接到一个外部控制平面。

- Zero Trust Architecture，零信任架构

  零信任架构规定了一种完全消除信任的 IT 系统设计和实施方法。 其核心原则是“永不信任，永远验证”，设备或系统本身在与系统的其他组件进行通信时，总是先验证自己。 在今天的许多网络中，在企业网络中，内部的系统和设备可以自由地相互通信，因为它们在企业网络外围的信任边界内。零信任架构采取了相反的方法，虽然在网络边界内，但系统内的组件在进行任何通信之前首先必须通过验证。

  在传统的基于信任的方法中，存在于企业网络周边的系统和设备，其假设是，因为有信任，所以没有问题。 然而，零信任架构认识到，信任是一个弱点。 如果攻击者获得了对受信任设备的访问，根据对该设备的信任和访问程度，系统现在很容易受到攻击，因为攻击者在“受信任”的网络边界内，能够在整个系统内横向移动。 在零信任架构中，信任被移除，因此减少了攻击面，因为攻击者现在被迫在进入整个系统之前进行验证。

采用零信任架构带来的主要好处是增加安全，减少攻击面。 从您的企业系统中移除信任，现在增加了攻击者必须通过的安全门的数量和强度，以获得对系统的其他区域的访问。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

功能和优势

DaoCloud Enterprise 5.0 融海量特性于一体，提供一站式 PaaS 服务，目前支持的典型功能如下。

功能特性

===“多集群管理”

- 多集群的统一纳管，支持所有特定版本范围内任意 Kubernetes 集群纳入容器管理范围，实现云上云下多云&混合云容器云平台的统一管理。
- 集群的快速部署，通过 Web 界面快速部署企业级的 Kubernetes 集群，快速搭建企业级容器云台，适配物理机和虚拟机底层环境。
- 一键式集群升级，一键升级 Kubernetes 版本，统一管理系统组件升级。
- 集群高可用，内置集群容灾、备份能力，保障业务系统在主机故障、机房中断、自然灾害等情况下可恢复，提高生产环境的稳定性，降低业务中断风险。
- 集群的全生命周期管理，实现自建云原生集群的全生命周期管理。
- 开放式 API 能力，提供原生的 Kubernetes OpenAPI 能力。

===“云原生应用管理”

- 多形态云原生应用管理，实现不同类型的云原生应用的部署及全生命周期管理，支持云原生应用、Helm 应用、OAM 等，开放式多云应用兼容。
- 一站式部署，解耦底层 Kubernetes 平台，一站式部署和运维业务应用，实现应用的全生命周期管理。
- 应用负载的扩缩容，通过界面可实现应用负载的手动/自动扩缩容能力，自定义扩缩容策略，应对流量高峰。
- 应用的全生命周期，支持应用查看、更新、删除、回滚、时间查看以及升级等全生命周期管理。

- 跨集群负载统一管理能力。

===“持续性业务交付”

- 企业级云原生 CI/CD 能力，实现标准化的持续集成和交付能力，实现应用的敏捷迭代，快速满足业务新需求，提高应用的集成和发布效率。
- 持续交付能力云原生化，应用构建环境的容器化管理，自定义构建机器等资源，确保流水线级资源隔离，同时支持多种类型的应用部署，使得云原生应用程序的交付更加简单。
- 提升研发效能，在可视化的流水线构建环境中，预先封装声明性步骤，无需脚本就能创建复杂的流水线，操作简单。可根据资源大小自动创建/销毁环境，避免资源浪费。
- 流水线即代码，基于声明性（语法）的流水线易于学习，不同步骤的语法实现了标准化，其配置具有版本可控、模块化、可重用和声明性等特点。
- 综合集成社区流行的 DevOps 工具，单个流水线中的步骤可以在多操作系统、多体系结构节点上运行，支持在公有云、私有云或主机上独立部署，能很好地集成企业自有的系统和平台。 支持集成业界主流软件，如 Kubernetes、GitLab、SonarQube、Harbor 等。

===“多云应用分发”

- 统一多云编排控制面，统一管理多个多云实例，统一请求入口。
- 多云集群一键导入。
- 丰富的多云应用分发策略、覆盖策略等。
- 多云应用故障切换（Failover）功能。

===“统一的用户权限体系”

- 用户/用户组集中式管理，多功能模块统一用户/用户组管理，打通用户体系，降低管理运维成本。
- 多模型授权模型，满足不同企业规模、组织结构用户及授权模型，实现最短链路授权。

- 企业级访问控制，统一认证登录系统，通过联合认证接登录容器管理平台，实现单点登录。
- 提供工作空间/层级管理，高度契合企业业务层级，满足部门层级间的资源隔离及管理需求。
- 灵活的资源划分，基于资源粒度（命名空间、集群、注册中心等）进行资源划分及绑定，统一分配容器平台资源。

===“一站式微服务治理”

- 微服务注册中心集中纳管集群所有动态微服务。
- 微服务流量治理模块可快速与主流微服务框架集成。
- 注册配置中心能够集中管理和配置微服务相关资源。
- 微服务网关肩负管理微服务南北流量管控的重要作用。

===“可观测”

- 以应用为中心、开箱即用的新一代云原生可观测性运维平台，实现快速的故障定位和快捷的故障排查。
- 实时监控应用及资源，采集各项指标、日志及事件等数据分析应用健康状态，提供告警能力以及全面、清晰、多维度数据可视化能力。
- 开放兼容主流开源组件，并提供快捷故障定位及一键监控诊断的能力。
- 提供了指标、日志、链路三合一的统一采集，支持对指标、日志进行多维度的告警并提供简洁明了的可视化管理界面。

===“精选中间件”

- 支持多种主流中间件：Redis、MySQL、Elasticsearch、RabbitMQ、Kafka、MinIO 等。
- 支持实例的全生命周期的管理，满足多种高可用集群模式，可根据场景需求自定义副本和自定义配置。
- 支持在线的资源和存储的扩容。
- 支持实例资源和业务维度的统一监控。

- 可基于工作空间实现多租户需求的权限管理。

产品优势

===“简单易用”

- 一键式创建/接入容器集群，实现大规模集群跨云/跨集群的统一管理。图形化创建流水线，满足多场景需求。一站式应用分发，兼容多形态应用。全面打造一站式生产就绪云原生管理平台。
- 无需感知底层 Kubernetes 平台差异性，简化应用部署流程，降低应用上云难度，让用户聚焦应用开发，无需关心底层基础设置。同时为 **让用户像使用单集群一样来使用多集群** 奠定基础。

===“生产就绪”

- 应用高可用，支持应用的分布式部署，支持单点故障流量的自动切换。
- 简化应用部署流程，降低应用上云难度，让用户聚焦应用开发，无需关心底层基础设置。 同时为 **让用户像使用单集群一样来使用多集群** 奠定基础。
- 兼容传统微服务框架及云原生框架，传统微服务框架无需改造，无感接入，高度兼容 Zookeeper 等传统注册中心，以及云原生的 Kubernetes 和 Mesh 注册中心。 同时也支持 Spring Cloud、Dubbo 等主流的开源微服务框架以及 Envoy、Sentinel、Contour 等开源网关组件。
- 丰富的监控指标，实现应用的全方位监控，提前预警应用流量高峰，应用故障。

===“持续交付”

- 应用构建容器化，自定义流水线创建，简化交付流程。
- 基于容器技术，实现应用快速在不同环境中快速部署、验证和流转。
- 多形态应用兼容，包容多样性的部署及管理。

===“安全可靠”

- 自建集群默认为高可用部署模式，保障您的业务高可用。当节点出现故障或发生自然灾害时，应用可持续运行，保障生产环境的高可用，实现业务应用系统不中断。
- 跨地域应用高可用，支持跨地域部署不同的容器集群，并同时将业务部署在不同地域的多个云容器集群上，帮助应用实现多地域的流量分发。 当某个云容器集群发生故障、机房宕机或者自然灾害时，通过统一流量分发的机制，自动地将业务流量切换到其他云容器平台上，保障应用的高可用。
- 提供平台备份与容灾方案，保障平台数据及业务安全。
- 全方位的监控，三合一的监控系统，基于集中式的集群管理，实现集群、节点、组件、应用的多层面多维度的监控机制，实现快速定位故障、故障排查。
- 完善的用户权限体系，整合 Kubernetes RBAC 权限体系，支持为不同用户设置细粒度权限，实现资源隔离，保证业务数据安全性。

===“混合异构”

- 多云异构，实现云上云下、不同厂商容器云平台的异构管理能力。
- 提供高度自动化和具备弹性的异构多云支撑能力，适配 x86 以及信创云体系架构。
- 支持 x86、ARM 架构混合集群统一部署、统一管理，稳定支撑应用运行，保障应用间网络互通。

===“开放兼容”

- 基于原生的 Kubernetes 和 Docker 技术，全面兼容 Kubernetes API 和 Kubectl 命令。
- 提供了丰富的插件体系扩充云容器集群功能，例如 Multus、Cillum、Contour 等网络插件。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

免费申请「DCE 5.0 社区版」

DCE 5.0 社区版包含的模块为全局管理、容器管理、可观测性等模块。 建议下载并安装 DCE 5.0 之后，参照以下步骤进行激活。

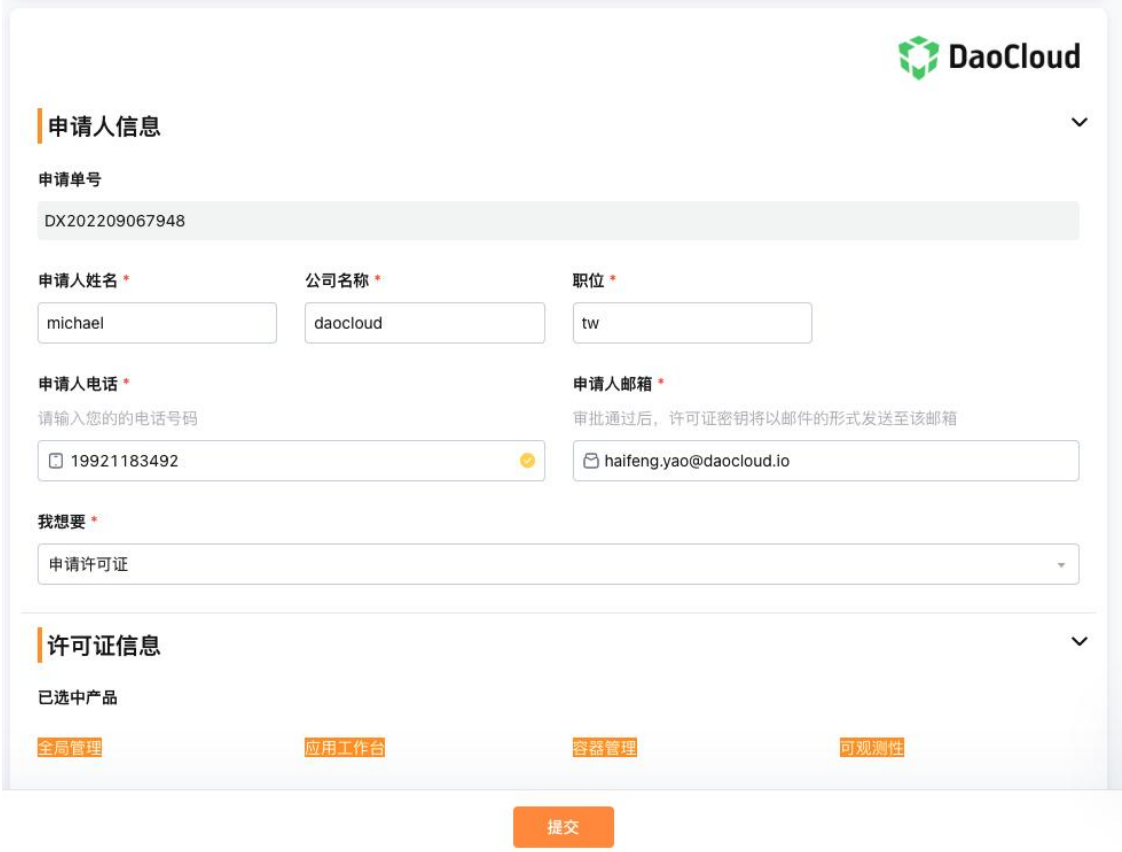


license

申请许可证

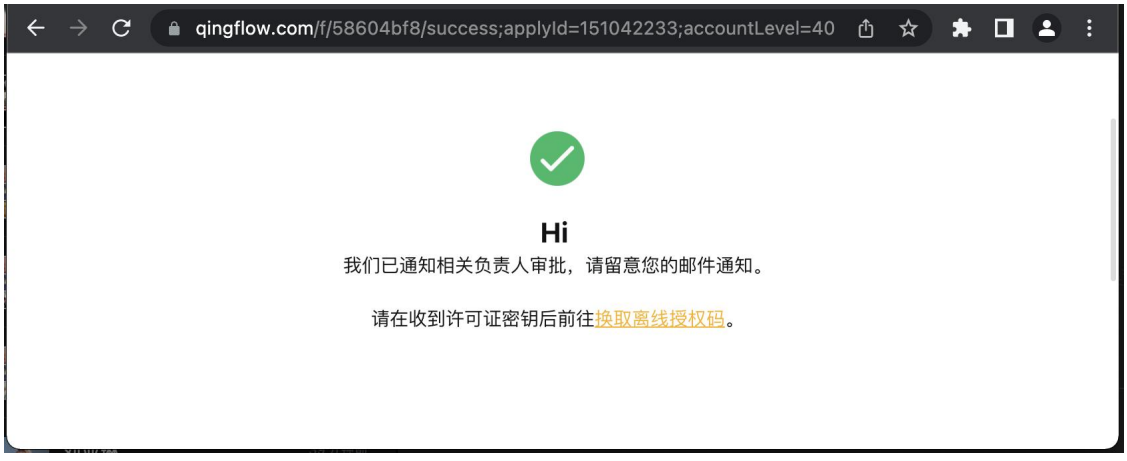
申请许可证的具体步骤如下：

1. 点击 申请许可证密钥 ，填写表单后点击提交。



license

2. 按屏幕提示，打开邮箱查看包含许可证密钥的邮件，然后点击 换取离线授权码 。



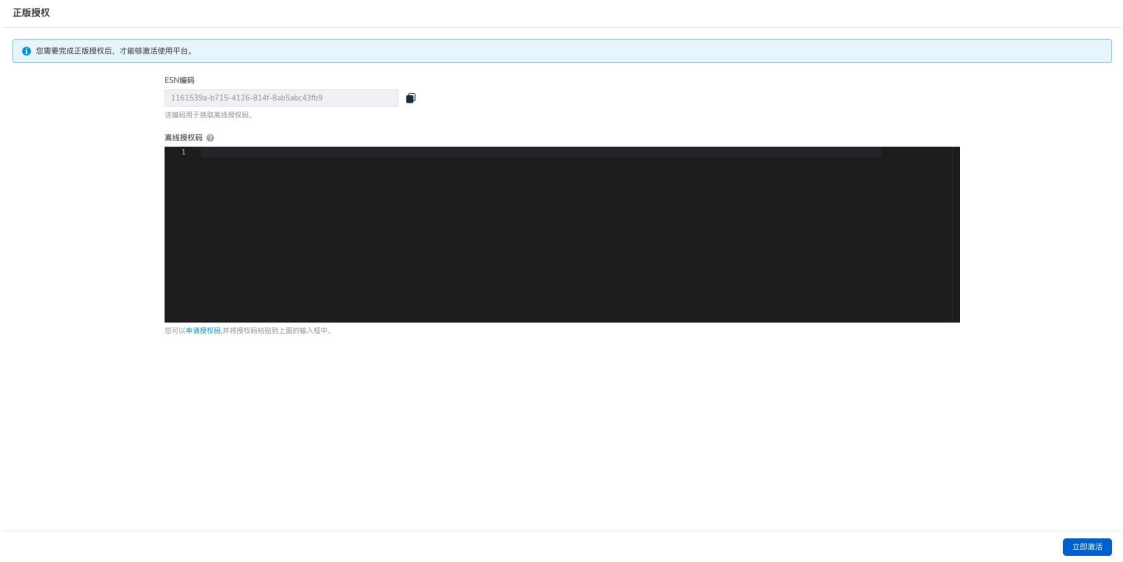
get-auth-code

3. 输入上一步的许可证密钥，输入设备独有的 ESN 后，点击获取离线授权码。



offline-auth-code

其中 ESN 是集群系统独有的设备编码。 具体获取方式为：打开 DCE，点击左侧导航栏全局管理 -> 平台设置 -> 正版授权，点击管理许可证按钮，打开管理许可证页面，复制 ESN 编码。



esn

4. 复制并粘贴上一步换取的离线授权码，点击立即激活。恭喜您！现在可以探索全新的 DCE 5.0 啦！

!!! info “�� 温馨提示”

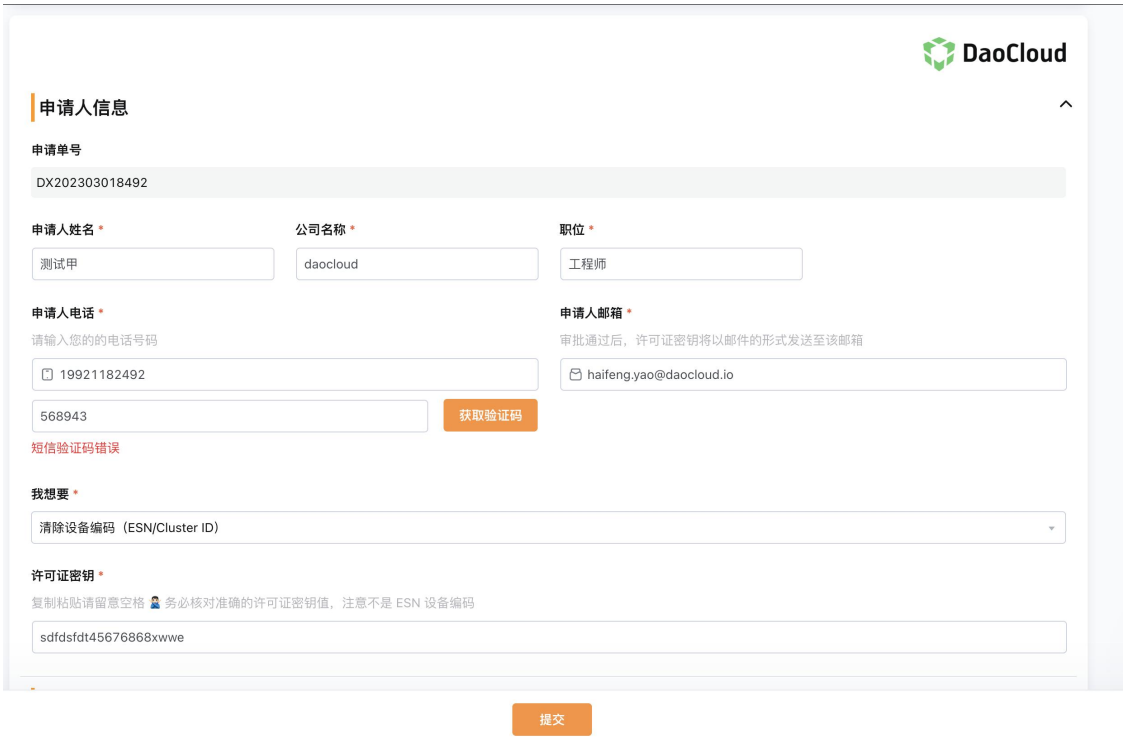
离线授权码与设备 ESN 绑定，请妥善保存好许可证密钥邮件。
您可以随时联系 DaoCloud 交付团队重新申请，我们已为您保存好所有许可密钥。

许可证续期

如果您的许可证因某些原因到期了，可参考以下步骤续期。

推荐方案

1. 首先清除设备编码，在表单中选择清除设备编码（ESN/Cluster ID），输入先前的许可证密钥，填写完成后点击提交。

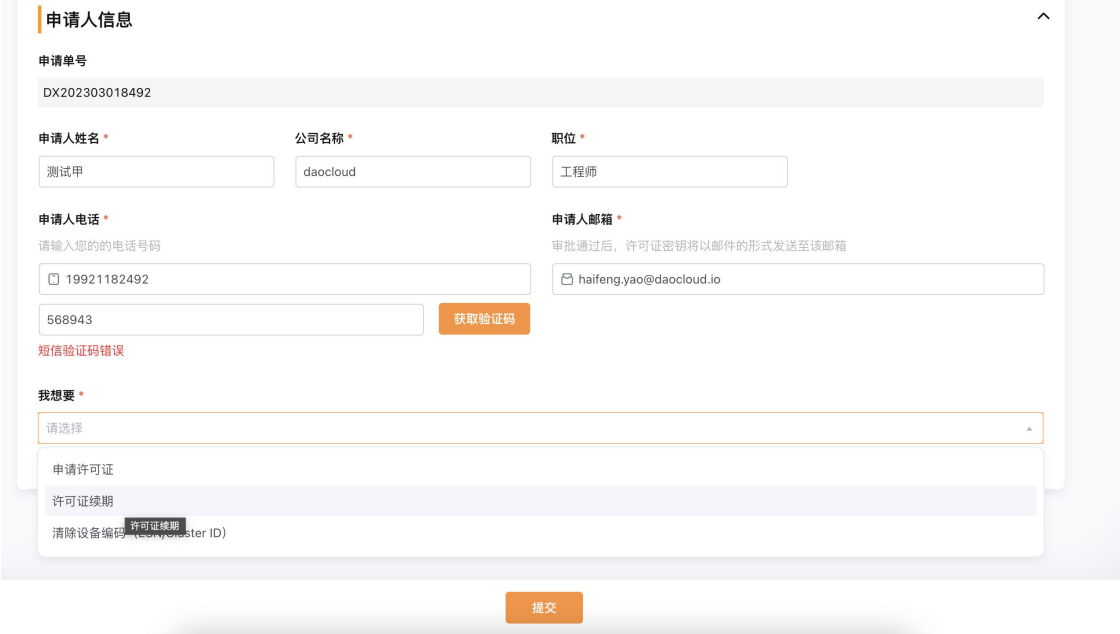


清除 ESN

2. 参照上一节重新申请许可证。

按模块延续许可证

这种方法需要对安装的每个模块延续许可证。 例如 DCE 5.0 社区版包含了容器管理、全局管理、可观测性 3 个模块，每个模块有 1 个许可证。 这样就需要在许可证页面，为每个模块选择许可证续期。



延期

参考链接

上述激活过程中常用到的几个页面汇总如下：

申请许可证密钥 查看许可证密钥 换取离线授权码 安装 DCE 5.0

DCE 5.0-20230731 商业版正式发布

本页说明了 DCE 5.0-20230731 商业版包含的最新特性、优化、缺陷修复等。DCE 5.0 每个产品模块都是独立解耦的，本次发布中所涉及的产品模块为 DCE 5.0 安装器默认包含的版本。 如需独立升级或替换各个模块，请联系 Daocloud 售后支持人员下载对应版本并升级。 另请参阅商业版部署要求。

功能模块版本

模块	版本	适配的 K8s 版本
安装器	v0.10.0	K8s 1.23 – 1.25
全局管理	v0.19.2	K8s 1.23 – 1.25
集群全生命周期管理	v0.8.0	K8s 1.25 – 1.27
容器管理	v0.20.0	K8s 1.18 – 1.25
网络	v0.8.1	K8s 1.18 – 1.25
Hwameistor	v0.11.2	K8s 1.18 – 1.25
VirtNest	v0.1.0	K8s 1.18 – 1.25
可观测性	v0.19.1	K8s 1.19 – 1.27

模块	版本	适配的 K8s 版本
应用工作台	v0.19.2	K8s 1.23 – 1.25
多云编排	v0.11.1	K8s 1.18 – 1.25
微服务引擎	v0.26.2	K8s 1.23 – 1.25
服务网格	v0.18.0	K8s 1.21 – 1.25
中间件 RabbitMQ	v0.12.3	K8s 1.23 – 1.25
中间件 MySQL	v0.10.3	K8s 1.23 – 1.25
中间件 ElasticSearch	v0.9.3	K8s 1.23 – 1.25
中间件 Redis	v0.10.0	K8s 1.23 – 1.25
中间件 MinIO	v0.7.3	K8s 1.23 – 1.25
中间件 PostgreSQL	v0.4.0	K8s 1.23 – 1.25
中间件 Kafka	v0.7.3	K8s 1.23 – 1.25
镜像仓库	v0.10.0	K8s 1.23 – 1.25

新功能和优化

详细的特性说明已体现在上表各版本的 Release Notes 中，这里简要说明一些新增的特性和优化。

安装器 v0.10.0

- **新增** 支持 Oracle Linux R8–U7 操作系统
- **新增** import–artifact 子命令支持根据 clusterConfig.yaml 配置文件中定义的外接服务来导入离线资源
- **优化** 重构、抽象 clusterConfig 检测层
- **优化** 优化前置依赖安装脚本的错误提示
- **优化** 在最小化安装过程中 ES 健康状态为 yellow 时允许继续安装

微服务引擎 v0.26.2

- **新增** 支持 API 级别的全局限流
- **新增** 支持域名级别的全局限流
- **新增** 支持 API 列表批量操作（上线/下线/删除）
- **新增** 支持云原生微服务治理通过插件方式扩展能力，包括：JWT 插件、Auth 插件、Wasm 插件、全局限流插件等

可观测性 v0.19.1

- **新增** Kubenetes 事件分析及查询功能
- **新增** 内置告警策略提供告警修复建议
- **新增** 支持自定义日志导出字段及格式
- **优化** 系统组件中获取 Elasticsearch 组件状态的逻辑
- **优化** 支持在安装参数中配置是否导入默认消息模板

服务网格 v0.18.0

- **新增** Istio 版本列表会包含推荐版本
- **新增** 服务诊断，支持自动修复和手工修复指南
- **新增** 支持网格控制面多集群高可用部署配置
- **优化** 命名空间边车默认过滤掉系统命名空间

- **优化** 网格权限优化，适配工作空间更新

多云编排 v0.11.1

- **新增** 支持在资源详情中查看事件信息
- **升级** Operator 至 Karmada 社区版本

容器管理 v0.20.0

- **新增** Helm 应用界面支持查看 Helm 操作日志
- **新增** 工作集群支持接入异构节点
- **新增** 创建集群支持批量导入节点
- **新增** 容器存储支持创建 NFS 类型的数据卷
- **新增** 新增 vGPU 支持，支持自动识别节点 CPU、支持新增为负载配置 CPU 配额
- **优化** 优化集群接入逻辑，当接入集群二次接入新管理平台时，需要预先清理旧管理平台的数据冗余才能被接入，关于集群接入的更多细节， 请参考卸载/解除接入集群
- **优化** 升级 Clusterpedia 到 v0.7.0
- **优化** 优化基于权限的页面交互，无权限用户将无法进入无资源权限的页面
- **优化** 接入节点支持配置内核调优等高级参数配置
- **优化** Insight 组件安装检测机制
- **修复** 修复 Helm 任务一直处在安装中、卸载中的问题
- **修复** 创建集群节点检查内核版本检测错误问题
- **修复** 创建集群插件无法自定义命名空间的问题
- **修复** 更新密钥默认增加 ca.crt 数据的问题

容器网络 v0.8.0

- **新增** Spiderpool CR 中新增 nodeName、multusName 字段，用于支持节点拓扑，能按需配置网络

- **新增** Spiderpool 提供了 SpiderMultusConfig CR，简化书写 JSON 格式的 Multus CNI 配置， 自动管理 Multus NetworkAttachmentDefinition CR
- **新增** Spiderpool 提供了 Coordinator 插件，解决 Underlay Pod 无法访问 ClusterIP、调谐 Pod 的路由、检测 Pod 的 IP 是否冲突、Pod 的网关是否可达等。参考 Coordinator 文档。
- **新增** IPVlan 的深度支持，适用于任何公有云环境
- **新增** 支持多个默认 IP 池，简化使用成本
- **新增** CNI 插件 Ifacer，用于自动创建子接口，参考 Ifacer 文档
- **新增** 通过 Pod 注解指定默认路由网卡
- **新增** 自动池的回收开关支持，可以自定义自动池是否删除
- **优化** 集群子网弹性 IP 的支持，可以很好地解决应用在滚动更新时，旧 Pod 未删除，新 Pod 没有可用 IP 的问题

集群生命周期管理 v0.8.0

- **新增** 集群操作记录保留条数配置
- **新增** Apache-2.0 开源 License 许可

应用工作台 v0.19.3

- **新增** Issues 界面列表支持跳转到 Jira 查看详情
- **新增** OAM 应用支持编辑 YAML、添加组件、编辑组件、编辑运维特征等功能
- **新增** 流水线支持 Sonarqube 配置、代码质量结果步骤

数据服务（中间件）

- **新增** Mcamel-Redis 访问白名单配置
- **新增** Mcamel-MongoDB 对 log 目录所在 PVC 的配置能力
- **新增** Mcamel-MongoDB 工作负载反亲和能力
- **新增** Mcamel-RabbitMQ 工作负载反亲和能力
- **新增** Mcamel-PostgreSQL 备份管理能力

- **新增** 数据服务下所有中间件实例 UI 界面权限访问限制
- **优化** Mcamel-Redis 创建实例对话框，添加默认反亲和标签值，简化配置过程
- **优化** Mcamel-Redis 数据恢复界面
- **优化** Mcamel-MySQL 数据恢复界面
- **优化** Mcamel-MongoDB 创建实例对话框增加反亲和默认标签，简化配置过程
- **优化** Mcamel-MongoDB 可以在非 Operator 所在命名空间创建 MongoDB 实例
- **优化** Mcamel-MongoDB 增加前端界面权限相关的展示信息
- **优化** Mcamel-PostgreSQL 监控图表，去除干扰元素并新增时间范围选择
- **修复** Mcamel-Redis 关闭节点亲和性失败
- **修复** Mcamel-PostgreSQL 监控图表部分 Panel 无法展示的问题

镜像仓库 v0.10.0

- **新增** 输出镜像仓库迁移/备份/恢复方案，已通过迁移 Release-ci 仓库验证
- **新增** 输出非安全镜像仓库登录最佳实践方案
- **新增** 创建托管 Harbor 支持使用内部中间件 Minio
- **新增** 适配人大金仓的 PostgreSQL 模式
- **优化** 创建托管 Harbor 时对 PostgreSQL、Redis 地址进行格式校验
- **优化** 无权限置灰+提示优化
- **优化** 集群解绑后的特殊情况处理

全局管理 v0.19.2

- **新增** 支持在工作空间-资源组绑定 Mesh / Mesh-namespace 资源
- **新增** 自定义角色权限点增加 Tips 提示
- **新增** 平台设置-安全策略-对单个用户的多重并发会话进行限制
- **新增** 平台设置-安全策略-能够对系统的最大并发会话连接数进行限制

- **新增** 删除工作空间时校验镜像仓库中是否还存在资源
- **优化** Ghippo 中无权限的按钮、菜单均置灰，且 Hover 时出现无权限提示

虚拟机容器 v0.1.0

- **新增** 支持通过集群展示虚拟机列表
- **新增** 支持通过容器镜像创建虚拟机
- **新增** 支持关机/启动、删除虚拟机
- **新增** 支持通过控制台访问（VNC）虚拟机
- **新增** 支持查看虚拟机详情

升级注意事项

- **微服务引擎**：微服务引擎版本大于 v0.24.2 版时，针对 v0.24.2 及之前版本存在不兼容更新，因为网关涉及开源组件仓库地址变更， 所以更新前需要手动删除旧有的 gateway-api-admission-xxxxx Job，然后进行正常升级更新操作
- **全局管理**：当从 v0.15.x (或更低版本）升级到 v0.19.0 (或更高版本）时，数据库连接 Helm 参数格式需要修改， 详情参照离线升级全局管理

兼容性

!!! note

全局管理：一旦将全局管理升级到 v0.19.0 及以上版本，必须将其他模块必须升级到相应版本，否则界面无法正常访问！

全局管理 v0.19.0 支持的各模块版本如下：

- 容器管理 v0.18.1 及以上版本
- 可观测性 v0.17.2 及以上版本
- 应用工作台 v0.17.3 及以上版本
- 多云编排 v0.9.1 及以上版本
- 微服务引擎 v0.22.0 及以上版本

- 服务网格 v0.16.2 及以上版本
- 中间件 mcamel-rabbitmq v0.11.0 及以上版本
- 中间件 mcamel-mysql v0.9.0 及以上版本
- 中间件 mcamel-elasticsearch v0.8.0 及以上版本
- 中间件 mcamel-redis v0.8.0 及以上版本
- 中间件 mcamel-MinIO v0.6.0 及以上版本
- 中间件 mcamel-Kafka v0.6.0 及以上版本
- 中间件 mcamel-PostgreSQL v0.2.0 及以上版本
- 镜像仓库 v0.8.0 及以上版本

已知问题

安装器 v0.10.0 的已知问题：

- Redhat v8.6 操作系统火种 kind 重启后 kubelet 服务无法启动，报错：failed to initialize top level QOS containers: root container [kubelet kubepods] doesn't exist
- 安装基于 TencentOS 3.1 的集群时，无法正确识别包管理器，如果需要 TencentOS 3.1 请使用安装器 v0.9.0

安装

- DCE 5.0 商业版本安装
- DCE 5.0 社区版本安装

下载站

- DCE 5.0 下载站

DCE 5.0-20230630 商业版正式发布

本页说明了 DCE 5.0–20230630 商业版包含的最新特性、优化、缺陷修复等。DCE 5.0 每个产品模块都是独立解耦的，本次发布中所涉及的产品模块为 DCE 5.0 安装器默认包含的版本。 如需独立升级或替换各个模块，请联系 Daocloud 售后支持人员下载对应版本并升级。 另请参阅商业版部署要求。

功能模块版本

模块	版本	适配的 K8s 版本
安装器	v0.9.0	K8s 1.23 – 1.25
全局管理	v0.18.0	K8s 1.23 – 1.25
集群全生命周期管理	v0.7.0	K8s 1.23 – 1.25
容器管理	v0.19.0	K8s 1.18 – 1.25
网络	v0.8.0	K8s 1.18 – 1.25
Hwameistor	v0.10.4	K8s 1.18 – 1.25
可观测性	v0.18.2	K8s 1.19 – 1.27
应用工作台	v0.18.1	K8s 1.23 – 1.25

模块	版本	适配的 K8s 版本
多云编排	v0.10.3	K8s 1.18 – 1.25
微服务引擎	v0.23.0	K8s 1.23 – 1.25
服务网格	v0.17.0	K8s 1.21 – 1.25
中间件 RabbitMQ	v0.11.0	K8s 1.23 – 1.25
中间件 MySQL	v0.10.0	K8s 1.23 – 1.25
中间件 ElasticSearch	v0.9.0	K8s 1.23 – 1.25
中间件 Redis	v0.9.0	K8s 1.23 – 1.25
中间件 MinIO	v0.7.0	K8s 1.23 – 1.25
中间件 PostgreSQL	v0.3.0	K8s 1.23 – 1.25
中间件 Kafka	v0.7.0	K8s 1.23 – 1.25
镜像仓库	v0.9.0	K8s 1.23 – 1.25

新功能和优化

详细的特性说明已体现在上表各版本的 Release Notes 中，这里简要说明一些新增的特性和优化。

安装器 v0.9.0

- **新增** 可修改 clusterConfig.yaml 配置暴露 火种 kind 地址及端口
- **新增** 安装器内置默认 k8s 版本升级到 v1.26.5
- **优化** 优化安装器 import-artifact 命令逻辑和功能
- **修复** 修复外接 http 的 Harbor 仓库同步镜像失败问题
- **修复** 修复 ES 组件无法在 OCP 启动的问题
- **修复** 修复中间件数据库在 arm64 环境高概率新建数据库失败的问题

可观测性 v0.18.2

- **新增** 日志告警
- **新增** 采集管理增加集群状态
- **新增** 静默规则列表增加静默条件
- **新增** 邮箱配置检测
- **新增** 支持 PostgreSQL 和 Kingbase 作为系统数据库
- **新增** Nvidia GPU 监控仪表盘
- **优化** 更新服务拓扑的图例
- **优化** 服务详情操作指标增加指标平均值并支持排序
- **优化** 链路查询支持 Span、延时、发生时间等排序
- **优化** 告警策略配置通知时下拉框增加搜索
- **优化** 告警模板支持格式化时区
- **优化** 升级 opentelemetry collector 的 Chart 版本从 v0.50.1 到 v0.59.3
- **优化** 升级 opentelemetry Operator 的 Chart 版本从 v0.26.1 到 v0.30.1
- **修复** 告警静默预览匹配告警时预览和实际生效静默的告警不相同的问题

- **修复** 系统组件中组件版本错误的问题
- **修复** 仪表盘中列表标题错误
- **修复** 通过告警列表创建静默，预览匹配告警数据无返回
- **修复** 部分环境中 Fluentbit 第一次无法启动的问题微服务引擎

微服务引擎 v0.24.1

- **新增** 为引擎组件添加增强健康检查的配置
- **新增** 为缓存逻辑添加标签缓存支持
- **新增** 审计日志集成
- **新增** 在持续集成中添加许可检查
- **新增** 将 Chart 中的敏感信息通过 Secret 存储
- **新增** 升级托管 Nacos 版本至 2.2.3 版
- **新增** Sentinel 版本至 0.10.5 版

服务网格 v0.17.0

- **新增** 审计日志能力
- **新增** 工作负载注入策略清除能力
- **新增** MCPC Controller 实现纳管服务自动注入能力
- **新增** K8s 各版本兼容性测试
- **优化** 网格创建时对托管集群检测加强，避免创建冲突网格

多云编排 v0.10.3

- **新增** 多云审计日志
- **新增** 资源新增分发状态展示
- **新增** Service 支持 LoadBalancer 类型

容器管理 v0.19.0

- **新增** 兼容 openAnolis / oracle linux 操作系统部署工作集群
- **新增** 离线环境创建集群支持自动添加 jfrog 的认证信息
- **新增** 创建工作负载新增环境变量规则校验
- **新增** 新增边缘负载和服务

- **新增** 新增双栈、系统内核作为节点前置检查项
- **新增** 创建工作负载新增 secretKey/configmapKey 作为配置项挂在在容器内
- **优化** 优化 Helm 仓库刷新机制
- **优化** 部分 I8N 英文翻译界面
- **修复** 在创建集群时，填写自定义参数，如果 value 为 0 或者 1，会被错误转换为 true 或者 false 的问题
- **修复** 修复在离线环境创建集群时，无法写入 containerd 账号密码配置的问题
- **修复** 修复对 1.26 及以上版本的集群进行升级时，由于 kubernetes 镜像仓库更改的原因，导致集群升级失败的问题
- **修复** 修复命名空间级用户无法使用 StorageClasses 创建 PV 相关问题
- **修复** 修复创建路由时指定命名空间不生效的问题
- **修复** 修复集群升级后，日期返回错误问题

集群生命周期管理 kubean v0.7.0

- **修复** 制作精简镜像包 python 无法启动的问题

应用工作台 v0.18.1

- **新增** 支持对原生应用及其子资源的更新与删除
- **新增** 支持 OAM 应用的创建、详情等基础功能
- **新增** 流水线权限支持了运行流水线权限点
- **修复** 目标集群没有安装原生应用的 CRD 时，原生应用列表页会报错

数据服务（中间件）

- **新增** 支持 MongoDB 的实例生命周期管理
- **新增** 支持 Elasticsearch 的反亲和性设置
- **新增** 支持 Redis 的监控采样时间间隔设置
- **新增** 支持 kafka 的监控采样时间间隔设置
- **新增** 支持 RabbitMQ 的负载均衡访问设置

- **新增** 支持 kafka 的负载均衡访问设置
- **优化** 优化中间件监控界面
- **优化** 优化 PVC 监控数据中立
- **优化** 优化可观测组件安装过程闭环
- **优化** 优化 Redis 纳管模式下更新配置的提示信息
- **优化** 优化备份页面结构
- **修复** 修复自定义角色中平台角色缺少工作空间权限关联的问题
- **修复** 修复 Redis 自动备份实际执行时间与配置时间相差 8 小事的问题

镜像仓库 v0.9.0

- **新增** 支持通过 UI 上传和下载 tar 包
- **新增** 支持审计镜像仓库重点操作
- **新增** 创建托管 harbor 支持使用中间件 PostgreSQL
- **修复** 修复 Sidecar 注入问题
- **修复** 修复镜像空间与工作空间的解绑问题

全局管理 v0.18.0

- **新增** 移动工作空间
- **新增** 接入管理 – Webhook（创建/编辑/删除/列表/查看）
- **新增** License 过期前提醒（提前 15 天）
- **新增** License 过期后功能不可用（可配置过期后是否可用）
- **新增** 用户在修改密码时需要输入旧密码
- **新增** 资源组支持绑定服务网格资源
- **新增** 支持外接 Postgres 数据库
- **新增** 运营管理 – 报表支持更多类型的货币
- **新增** istio-ingressgateway 和 istiod 支持高可用

升级注意事项

- 可观测性：当 **insight agent** 从 v0.16.x（或更低版本）升级到 v0.17.x 时， 由于 kube-prometheus-stack chart 版本从 41.9.1 升级至

45.28.1，其中使用的 CRD 存在部分字段的升级， 需要手动进行更新，请参考 Insight 升级注意事项。

- 微服务引擎：数据库初始化组件 sweet 已于 0.22.0 版废弃，将在 0.23.0 及之后版本彻底移除， 从 0.22.0 版开始数据表将自动同步更新，无需人工干预。
- 全局管理：当从 v0.15.x (或更低版本) 升级到 v0.17.0 (或更高版本) 时，数据库连接 helm 参数格式需要修改， 详情参照离线升级全局管理。

兼容性

!!! note

全局管理：一旦将全局管理升级到 v0.17.0 或 v0.16.0 及以上版本，必须将其他模块必须升级到相应版本，否则界面无法正常访问！**

全局管理 v0.17.0 支持的模块版本	全局管理 v0.16.0 支持的模块版本
容器管理 v0.18.1 及更高	容器管理 v0.17.0 及更高
可观测性 v0.17.2 及更高	可观测性 v0.16.0 及更高
应用工作台 v0.17.3 及更高	应用工作台 v0.16.0 及更高
多云编排 v0.9.1 及更高	多云编排 v0.8.0 及更高
微服务引擎 v0.22.0 及更高	微服务引擎 v0.21.1 及更高
服务网格 v0.16.2 及更高	服务网格 v0.15.0 及更高
中间件 RabbitMQ v0.11.0 及更高	中间件 RabbitMQ v0.10.1 及更高
中间件 MySQL v0.9.0 及更高	中间件 MySQL v0.8.1 及更高
中间件 ElasticSearch v0.8.0 及更高	中间件 ElasticSearch v0.7.1 及更高
中间件 Redis v0.8.0 及更高	中间件 Redis v0.7.1 及更高
中间件 MinIO v0.6.0 及更高	中间件 MinIO v0.5.1 及更高
中间件 Kafka v0.6.0 及更高	中间件 Kafka v0.5.1 及更高
中间件 PostgreSQL v0.2.0 及更高	中间件 PostgreSQL v0.1.0 及更高

全局管理 v0.17.0 支持的模块版本

镜像仓库 v0.8.0 及更高

全局管理 v0.16.0 支持的模块版本

镜像仓库 v0.7.0 及更高

安装

- DCE 5.0 商业版本安装
- DCE 5.0 社区版本安装

下载站

- DCE 5.0 下载站

DCE 5.0-20230531 商业版正式发布

本页说明了 DCE 5.0-20230531 商业版包含的最新特性、优化、缺陷修复等。DCE 5.0 每个产品模块都是独立解耦的，本次发布中所涉及的产品模块为 DCE 5.0 安装器默认包含的版本。 如需独立升级或替换各个模块，请联系 Daocloud 售后支持人员下载对应版本并升级。 另请参阅商业版部署要求。

功能模块版本

模块	版本	适配的 K8s 版本
安装器	v0.8.0	K8s 1.23 – 1.25
全局管理	v0.17.1	K8s 1.23 – 1.25
集群全生命周期管理	v0.5.4	K8s 1.23 – 1.25
容器管理	v0.18.1	K8s 1.18 – 1.25
网络	v0.7.0	K8s 1.18 – 1.25
Hwameistor	v0.10.2	K8s 1.18 – 1.25
可观测性	v0.17.2	K8s 1.19 – 1.27
应用工作台	v0.17.3	K8s 1.23 – 1.25

模块	版本	适配的 K8s 版本
多云编排	v0.9.1	K8s 1.18 – 1.25
微服务引擎	v0.22.0	K8s 1.23 – 1.25
服务网格	v0.16.2	K8s 1.21 – 1.25
中间件 RabbitMQ	v0.11.0	K8s 1.23 – 1.25
中间件 MySQL	v0.9.0	K8s 1.23 – 1.25
中间件 ElasticSearch	v0.8.0	K8s 1.23 – 1.25
中间件 Redis	v0.8.0	K8s 1.23 – 1.25
中间件 MinIO	v0.6.0	K8s 1.23 – 1.25
中间件 PostgreSQL	v0.2.0	K8s 1.23 – 1.25
中间件 Kafka	v0.6.0	K8s 1.23 – 1.25
镜像仓库	v0.8.0	K8s 1.23 – 1.25

新功能和优化

详细的特性说明已体现在上表各版本的 Release Notes 中，这里简要说明一些新增的特性和优化。

安装器 v0.8.0

- **新增** 支持外接 JFrog 镜像及 charts 仓库
- **新增** 支持 Yum 类型包管理的 Other Linux (OpenAnolis 8.8 GA)
- **新增** 支持混合架构部署方案

可观测性 v0.17.2

- **新增** 支持查看活动告警和历史告警详情
- **新增** 支持通过告警快速创建静默规则
- **新增** 告警支持短信通知
- **新增** 邮箱通知支持发送测试信息
- **新增** 邮箱通知的消息模板支持自定义邮箱主题
- **新增** 消息模板增加变量说明
- **新增** Insight Server 组件支持默认高可用

微服务引擎 v0.22.2

- **新增** 托管 Nacos 2.2.x 版本的支持
- **新增** 支持网关链路信息配置
- **新增** 支持网关通过插件方式扩展能力，包括 jwt、Auth、wasm、全局限流等
- **新增** OpenAPI 文档发布流程
- **新增** 支持云原生微服务基于端口的流量治理包括：负载均衡、离群检测、服务熔断等
- **新增** 支持基于域名级别的黑白名单功能

服务网格 v0.16.2

- **新增** 支持多工作负载边车操作
- **新增** 网格控制面组件状态展示

- **新增** 边车升级支持多工作负载能力

多云编排 v0.9.1

- **新增** Service 支持标签、注解表单编辑功能

容器管理 v0.18.1

- **新增** 巡检报告下载
- **新增** 对接高优先级操作全局审计日志
- **新增** 连接 Minio 的超时处理
- **优化** CloudShell 由用 ConfigMap 进行 KubeConfig 挂载改为用 Secret 进行 KubeConfig 挂载
- **优化** 创建备份策略集群下拉列表时，新增过滤并创建了备份策略的集群的开关
- **修复** etcdbrctl 镜像离线化
- **修复** 镜像选择器无法选择镜像
- **修复** 创建集群时的 repo 地址渲染

集群生命周期管理 kubean v0.5.4

- **新增** Flannel、kube-ovn 清单配置

应用工作台 v0.17.3

- **新增** 支持基于 nginx-ingress 的灰度发布策略
- **新增** API 支持应用的资源拓扑

数据服务（中间件）

- **新增** MySQL 可配置实例监控数据采集间隔时间
- **新增** Redis 可配置实例反亲和性
- **新增** 对 PostgreSQL 的支持

镜像仓库 v0.8.0

- **新增** 支持镜像回收功能，支持手动/定时/模拟回收
- **新增** 支持为镜像增加描述信息
- **优化** 生成登录指令优化，支持在生成指令页面切换镜像实例

- **优化** 镜像空间列表增加实例状态
- **优化** 创建托管 Harbor 时支持随机生成 nodeport 端口号

全局管理 v0.17.1

- **新增** 接入管理功能，支持将 DCE5.0 作为用户源接入其他平台
- **新增** 工作空间/文件夹授权支持展示角色的权限详情
- **新增** 审计日志支持下载为 Excel 和 CSV 格式
- **新增** 容器组计费报表、 命名空间计费报表、工作空间计费报表，所有报表支持下载为 Excel 和 CSV 格式
- **优化** 用户名/用户组名支持 . 和 @
- **优化** 平台默认语言改成 自动检测浏览器首选项
- **优化** 审计日志：两种日志（系统和用户）分开显示
- **优化** 禁止移除 admin 用户的 admin 角色或编辑 admin 的权限

升级注意事项

- 可观测性：当 **insight agent** 从 v0.16.x（或更低版本）升级到 v0.17.x 时， 由于 kube-prometheus-stack chart 版本从 41.9.1 升级至 45.28.1，其中使用的 CRD 存在部分字段的升级， 需要手动进行更新，请参考 Insight 升级注意事项。

- 微服务引擎：数据库初始化组件 sweet 已于 0.22.0 版废弃，将在 0.23.0 及之后版本彻底移除， 从 0.22.0 版开始数据表将自动同步更新，无需人工干预。

- 全局管理：当从 v0.15.x (或更低版本) 升级到 v0.17.0 (或更高版本) 时，数据库连接 helm 参数格式需要修改， 详情参照离线升级全局管理。

已知问题

- 镜像仓库：登录指令过期将导致服务不可用，请在使用登录指令时及时延长过期时间，该问题将在 hotfix 版本中修复。

- 安装器：离线场景下使用外置 OS Repo 仓库时，即 clusterConfig.yaml 中定义 osRepos.type=external， 部署 DCE5.0 成功后无法在容器管理中创建工作集群，解决方案请参考文档。

兼容性

!!! note

全局管理：一旦将全局管理升级到 v0.17.0 或 v0.16.0 及以上版本，必须将其他模块必须升级到相应版本，否则界面无法正常访问！**

全局管理 v0.17.0 支持的模块版本	全局管理 v0.16.0 支持的模块版本
容器管理 v0.18.1 及更高	容器管理 v0.17.0 及更高
可观测性 v0.17.2 及更高	可观测性 v0.16.0 及更高
应用工作台 v0.17.3 及更高	应用工作台 v0.16.0 及更高
多云编排 v0.9.1 及更高	多云编排 v0.8.0 及更高
微服务引擎 v0.22.0 及更高	微服务引擎 v0.21.1 及更高
服务网格 v0.16.2 及更高	服务网格 v0.15.0 及更高
中间件 RabbitMQ v0.11.0 及更高	中间件 RabbitMQ v0.10.1 及更高
中间件 MySQL v0.9.0 及更高	中间件 MySQL v0.8.1 及更高
中间件 ElasticSearch v0.8.0 及更高	中间件 ElasticSearch v0.7.1 及更高
中间件 Redis v0.8.0 及更高	中间件 Redis v0.7.1 及更高
中间件 MinIO v0.6.0 及更高	中间件 MinIO v0.5.1 及更高
中间件 Kafka v0.6.0 及更高	中间件 Kafka v0.5.1 及更高
中间件 PostgreSQL v0.2.0 及更高	中间件 PostgreSQL v0.1.0 及更高
镜像仓库 v0.8.0 及更高	镜像仓库 v0.7.0 及更高

开源项目

本页按字母顺序列出云原生开源项目的动态。

A, B

- Acorn：Acorn Labs 开源的 K8s 应用打包和部署框架

  Acorn 能够将所有应用程序的 Docker 镜像、配置和部署规范打包成一个 Acorn 镜像工件，这个工件可以推送到任何 OCI 镜像仓库。 因此，开发人员无需切换工具或技术栈就能够将在本地开发应用转移到生产中。Acorn 镜像通过在 Acornfile 中描述应用程序的配置来创建，无需 Kubernetes YAML 的样板文件。

- Aeraki Mesh 服务网格 v1.3.0 发布

  更新内容：支持 Istio 1.16.x 版本、支持多路复用、支持在 Gateway 上提供 MetaProtocol 七层路由能力、Dubbo 服务支持应用级服务治理、支持 Redis 流量管理。

- Aeraki Mesh 服务网格项目 v1.1.0 发布（CNCF 项目）

  该版本主要新特性：支持 Istio 1.12.x 版本；支持 bRPC（better RPC）协议，bRPC 是百度开源的工业级 RPC 框架；MetaProtocol 支持流式调用、支持采用 MetaData 设置负载均衡 Hash 策略以及在 Response 中回传服务器真实 IP 等。

- AlterShield：由蚂蚁集团开源的变更管控平台

  AlterShield 是一款集变更感知、变更防御、变更分析于一身的一站式管控平台。旨在通过定义变更标准协议，规范变更管控流程，使用户快速发现变更中的问题，并及时降低故障影响面。对于云原生场景，AlterShield 提供了 Operator，连接各类 CI/CD 工具到 OCMS（Open Change

Management Specification）SDK。同时，Operator 本身也提供了变更流速控制、异常回滚策略控制等能力。

- Antrea CNI 插件 v1.12.0 发布（CNCF 项目）

  版本特性：拓扑感知功能和节点 IP 地址管理功能从 Alpha 升级到 Beta，且默认启用；支持通过一个新的 Antrea 代理配置参数启用多播功能；在 AntreaProxy 中增加对 ExternalIP 的支持；为多集群增加 WireGuard 隧道模式的支持；为多集群服务增加对 EndpointSlice API 的支持。

- Antrea CNI 插件 v1.11.0 发布（CNCF 项目）

  版本特性：ClusterSet scoped 策略规则支持命名空间字段、L7 策略规则支持流量日志、支持在 TCP 网络包上处理 DNS 请求、AntreaProxy 的端点切片（EndpointSlice）功能升级为 Beta、AntreaProxy 支持处理终止过程中的端点（ProxyTerminatingEndpoint）、Egress 策略支持限制分配到一个节点上的 Egress IP 数量、多集群网关支持多种流量模式。

- Antrea CNI 插件 v1.10.0 发布（CNCF 项目）

  该版本主要新特性：增加 L7 网络策略功能、Antrea 的 CRD API 能够在任何 K8s 节点或 ExternalNode 上收集 support bundle 文件、增加对跨集群流量的网络策略的支持、在 Windows 上使用 containerd 作为运行时时，antrea-agent 可作为 DaemonSet 运行。

- Antrea CNI 插件 v1.9.0 发布（CNCF 项目）

  新增多个多集群功能，包括跨集群 Pod 间连接、节点控制器支持 Gateway 高可用、允许 Pod IP 作为多集群 service 的端点； 增加类似于 kube-proxy 的 service 健康检查；审计日志添加规则名称；一个 service 支持 800+ 端点。

- Antrea CNI 插件 v1.8.0 发布（CNCF 项目）

该版本主要新特性：增加 ExternalNode 功能、K8s 网络策略增加审计日志支持、支持利用 Antrea ClusterNetworkPolicy 控制 NodePort 服务的外部访问、允许对不同的网络端点进行逻辑分组、每次 Antrea release 都生成新的 Helm Chart 版本、支持拓扑感知 TopologyAwareHints、在 IPPool CRD 中添加状态字段。

- Antrea CNI 插件 v1.7.0 发布（CNCF 项目）

  该版本主要新特性：增加 TrafficControl 功能，控制 Pod 流量的传递；支持 IPsec 证书认证；丰富 Antrea-native 策略；丰富组播功能；增加多集群网关功能；丰富二级网络 IPAM 功能。

- APISIX 云原生网关 v3.2.0 发布

  版本特性：支持四层上的服务发现、新增一个能将 RESTful 请求转成 GraphQL 的插件、支持在每个日志插件上设置日志格式、新增插件支持 JSON 和 XML 之间的互相转换。

- APISIX 云原生网关 v3.1.0 发布

  该版本主要新特性：支持将插件的特定字段加密保存到 etcd 中、允许将敏感信息存储在外部安全服务中、实验性地支持基于 gRPC 的 etcd 配置同步、新增基于 Consul 的服务发现功能、增加了一个内置的调试器插件。

- APISIX 云原生 API 网关 v3.0 发布

  v3.0 新增 Consumer Group 用于管理多个消费者、支持配置 DNS 解析域名类型的顺序、新增 AI 平面用于智能分析和可视化配置与流量、全面支持 ARM64、新增 gRPC 客户端、实现数据面与控制面的完全分离、提供控制面的服务发现支持、新增 xRPC 框架、支持更多四层可观测性、集成 OpenAPI 规范、全面支持 Gateway API。

- APISIX 云原生 API 网关 v2.15.0 发布

该版本主要新特性：支持用户自定义插件的优先级、支持通过表达式决定插件是否需要执行、自定义错误响应、允许采集 Stream Route 上的指标。

- Apollo 分布式配置管理系统 v2.0.0 发布 （CNCF 项目）

  该版本主要新特性：主页上增加公共命名空间列表视图、灰度规则支持在 IP 不固定的情况下的标签匹配、启用命名空间的导出/导入功能、大多数列表都新增 DeletedAt 列支持唯一约束索引。

- Arbiter：时速云开源基于 K8s 构建的可扩展调度和弹性工具

  Arbiter 聚合各种类型的数据，用户可以基于这些数据管理、调度或扩展集群中的应用程序。它可以帮助 Kubernetes 用户了解和管理集群中部署的资源，进而提高企业应用程序的资源利用率和运行效率。

- Argo CD 持续部署工具 v2.8.0 发布（CNCF 项目）

  版本特性：添加 kubelogin 功能、允许用户在任意命名空间中使用 ApplicationSet 资源、为快照卷添加健康检查、新增插件生成器功能、支持从环境变量或命令行参数中指定监听地址、创建 job action、支持刷新 ExternalSecret。

- Argo CD GitOps 工具 v2.6.0 发布（CNCF 项目）

  版本特性：ApplicationSet 资源增加渐进式发布策略、允许用户为应用程序提供多个资源、允许多个 CRD 共享健康检查、支持反向代理扩展、argocd CLI 添加跨平台的文件加密工具 bcrypt 的支持。

- Argo 成为 CNCF 毕业项目

  Argo 让团队能够使用 GitOps，在 Kubernetes 上采用声明方式部署和运行云原生应用和工作流。 Argo 的采用率自其成为 CNCF 孵化项目以来增长了 250%，并于今年 7 月通过了第三方安全审计， 已经成为云原生技术堆栈和 GitOps 的重要组成部分。

- Argo CD GitOps 工具 v2.5.0 发布（CNCF 项目）

  该版本主要新特性：支持基于集群过滤应用程序、添加 Prometheus 健康检查、添加通知 API、支持 CLI 的自定义应用程序操作、增加对默认容器注释的支持、限制 redis 的出口规则、添加 Gitlab PR 生成器 webhook、新增 ApplicationSet Go 模板、新增 ArgoCD CLI 本地模板。

- Argo 2022 年项目安全审计结果发布

  Argo 团队、CNCF 与软件安全公司 Ada Logics 合作，对 Argo 的四个项目进行安全审计。 共发现 26 个问题：Argo CD 7 个，Argo Workflows 6 个，Argo Events 13 个。为 Argo CD 发布了七个 CVE，Argo Events 两个 CVE。 截止报告发布时，所有可利用的问题都已修复，并作为安全建议发布在 GitHub 上。
  审计报告全文:
  https://github.com/argoproj/argoproj/blob/master/docs/argo_security_audit_2022.pdf

- Argo CD 持续部署工具 v2.4.0 发布（CNCF 项目）

  该版本主要新特性：UI 添加 Web 终端——无需离开 Web 界面就能在运行的应用容器中启动一个 shell；引入对 Pod 日志和 Web 终端的访问控制；集成 OpenTelemetry Tracing。

- Backstage 开发者门户构建平台的安全审计报告和威胁模型发布（CNCF 项目）

  审计共发现 10 个漏洞。 在 Backstage v1.5 中，其中的 8 个漏洞都已完全修复。 尚待解决的 4 个问题中，3 个与速率限制和内部 DoS 问题相关。 威胁模型总结了操作员、开发人员和安全研究人员的主要安全考虑事项，涵盖 Backstage 典型设置中涉及的信任模型和角色、集成者的职责以及常见配置问题等。

- Backstage 开发者门户构建平台 v1.4.0 发布（CNCF 项目）

  该版本主要新特性：Search API 升级到 v1、新增后端系统（实验性）、大量命令行及授权后端符号弃用、增加了对 OpenAPI 规范中 $ref 处理的支持、增加 Apollo Explorer 支持（PoC）。

C

- Calico CNI 插件 v3.26.0 发布（CNCF 项目）

  版本特性：将 calico-node 和 calico-cni-plugin 服务账户分开、利用内核级路由过滤减少在系统的 CPU 使用率、支持 Windows Server 2022、支持 OpenStack Yoga。

- Calico CNI 插件 v3.25.0 发布（CNCF 项目）

  该版本主要新特性：优化 eBPF 数据平面，确保连接时间负载均衡（Connect Time Load Balancing）在规模更大的、快速变化的环境中工作；Felix 组件支持重写内部 readiness/liveness watchdog 的超时；Typha 组件支持优雅关闭。

- Calico CNI 插件 v3.24.0 发布（CNCF 项目）

  该版本主要新特性：支持 IPv6 网络 wireguard 加密、通过 API 暴露 IPAM 配置和 IPAM block 亲和性、operator API 新增字段、支持分割 IP 池、从 pod 安全策略过渡到 pod 安全标准。

- Carina 云原生本地存储项目 v0.11.0 发布

  该版本主要新特性：支持 Cgroup v2、移除 HTTP Server、升级 CSI 官方镜像版本、移除 ConfigMap 同步控制器、carina 镜像移动到单独命名空间、增加 carina e2e 测试用以替代原有的 e2e 测试代码（开发测试中）、优化 Storageclass 参数的 pvc 调度逻辑。

- Carina 本地存储项目 v0.10.0 发布（CNCF 项目）

该版本主要新特性：支持将裸盘挂载到容器内直接使用、velero 备份存储卷支持、新增 CRD 资源 NodeStorageResource 替代原将磁盘设备注册到 Node 节点的方式、使用 job 生成 webhook 证书，替代原有脚本生成证书方式等。

- cdk8s+：AWS 开源的 Kubernetes 开发框架正式可用

  cdk8s+ 允许用户使用熟悉的编程语言和面向对象的 API 来定义 Kubernetes 应用和可复用的抽象。 相较于去年发布的 beta 版本，正式版本新增功能包括：隔离 pod 网络，只允许指定的通信；改进在同一节点上运行多个 pod 的配置机制；集成 Datree 插件，使用第三方策略执行工具检查 Kubernetes 中的错误配置。

- Cert-manager 云原生证书管理项目 v1.12.0 发布（CNCF 项目）

  版本特性：将二进制文件和测试分离撑独立的 Go 模块、增加对 JSON 日志的支持、支持带有 Vault 的短暂服务账户令牌、支持 ingressClassName 字段、新增标志用于指定哪些资源需要被注入到 Kubernetes 对象中。

- Cert-manager 云原生证书管理项目 v1.11.0 发布（CNCF 项目）

  该版本主要新特性：支持使用 Azure Workload Identity Federation 进行认证、支持指定 cert-manager 在与 ACME 服务器通信时使用的信任存储、支持 gateway API v1beta1、启用针对 Kubernetes 1.26 的测试。

- Cert-manager 云原生证书管理项目 v1.10.0 发布（CNCF 项目）

  该版本主要新特性：使用 trivy 扫描本地构建的容器；如果目标 Secret 配置错误或在请求后创建，重新同步签署请求；增加从 Kubernetes Secret 加载 Vault CA Bundle 的选项；支持在 chart 部署的所有资源上添加相同的标签。

- Cert-manager 云原生证书管理项目升级成为 CNCF 孵化项目

Cert-manager 是 Kubernetes 的一个插件，用于自动管理和颁发各种来源的 TLS 证书，为云原生工作负载提供加密保护。近日，CNCF 技术监督委员会已经投票接受 cert-manager 成为 CNCF 孵化项目。

- Cert-manager 云原生证书管理项目 v1.9.0 发布（CNCF 项目）

  该版本主要新特性：支持使用 cert-manager certificate（alpha）、支持通过 Ingress 资源上的注释配置 ingress-shim 证书。

- Chaosblade 混沌工程项目 v1.7.0 发布（CNCF 项目）

  该版本主要新特性：添加 time travel 实验、添加插件 zookeeper 和 clickhouse、jvm 性能优化、支持 blade 服务器的 mTLS 认证。

- Chaos Mesh 混沌工程平台 v2.5.0 发布（CNCF 项目）

  该版本主要新特性：支持多集群混沌实验、HTTPChaos 增加 TLS 支持、允许在 Helm 模板中为 controller manager 和仪表盘配置 Pod 安全上下文、StressChaos 支持 cgroups v2。

- Chaos Mesh 混沌工程测试平台 v2.2.0 发布（CNCF 项目）

  该版本主要新特性：新增 StatusCheck 功能，用于检查应用程序的健康状况，当应用程序不健康时中止混沌实验；支持使用 Chaosctl 进行 outbound 强制恢复、在 dashboard 中新增基于流程图创建工作流的界面等。

- ChaosMeta：蚂蚁集团开源的云原生自动化混沌工程平台

  ChaosMeta 设计上是包含了完整混沌工程生命周期的一站式演练综合解决方案，涵盖了准入检查、流量注入、故障注入、故障度量、恢复度量和故障恢复等多个阶段。现阶段，ChaosMeta 已经对外开放了丰富的故障注入功能，不仅支持单机部署和 Kubernetes 云原生部署，还支持 Kubernetes 本身和 Operator 等云原生故障场景的实验。

- Cilium CNI 插件 v1.14.0 发布（CNCF 项目）

  版本特性：支持双向认证、支持以 DaemonSet 的形式部署 Envoy、支持对节点间的流量进行 WireGuard 加密并且可以在 WireGuard 上使用 Layer 7 策略、Cilium Mesh 可以在云端和异构工作负载之间提供一致的网络连接、支持通过 Layer 2 传输协议向本地网络广播外部 IP 地址、支持多个 IPAM 池、BIG TCP 支持 IPv4。

- Cilium CNI 插件 v1.13.0 发布（CNCF 项目）

  版本特性：支持 Gateway API v0.5.1、增加 IPv6 BIG TCP 支持、支持 LoadBalancer IP 地址管理、初步支持 SCTP、支持根据标签选择器对节点进行细粒度的配置、支持 k8s 1.26、支持通过 BGP 控制平面宣告 LoadBalancer 服务的功能、支持通过内置 Envoy 代理实现现有 Kubernetes service 的 L7 负载均衡、Ingress 资源可以共享 Kubernetes LoadBalancer 资源、datapath 支持 mTLS、支持 Service 内部流量策略、对所有镜像进行 cosign 签名并为每个镜像创建 SBOM。

- Cilium 发布安全审计报告和模糊测试审计报告

  安全审计和模糊测试共发现了 30 个问题，并未发现关键风险漏洞。 其中，两个是中风险问题，第一个问题是缺少易于访问的关于安全运行 Cilium 的文档，该问题 PR 正在处理中； 第二个是在配置错误的情况下可能无法解锁互斥锁 mutex，该问题已修复。其余都是低风险或信息性的问题。

- Cilium 2022 年度报告发布

  报告记录了 2022 年 Cilium 项目的贡献者增长、版本亮点、用户调查结果、生产落地情况、社区活动，以及 2023 年的发展方向。 在 2023 年，Cilium service mesh 将发展成熟；通过 eBPF 捕获的内核数据将帮助周边生态为终端用户建立更好的平台；供应链安全功能将得到加强.

- Cilium CNI 插件 v1.12.0 发布（CNCF 项目）

该版本主要新特性：推出 Cilium Service Mesh（多控制平面、边车/无边车、Envoy CRD)、集成 Ingress Controller、增加 K8s 服务拓扑感知提示、初始 NAT46/64 实现、支持 Pod 启用 BBR 拥堵控制并将带宽管理器移出 beta、支持在集群池 v2 IPAM 模式下动态分配 pod CIDR、支持设置服务后端状态、出口网关升级到稳定状态。

- Cloud Custodian 云资源管理工具 v0.9.20 发布（CNCF 项目）

  该版本主要新特性：增加 K8s admission controller 模式、添加角色和集群角色资源。

- Cloud Custodian 云资源管理工具升级为 CNCF 孵化项目

  Cloud Custodian 是一个云治理即代码工具，允许用户通过代码来管理和自动执行云安全、合规性、运营和成本优化的策略，包括编写管理 Kubernetes 资源的策略。与其他基于云的治理工具相比，提供了一个非常简单的 DSL 来编写策略及其跨云平台的一致性。

- Cloud Native App Initializer：阿里巴巴正式开源云原生应用脚手架

  云原生应用脚手架基于 Spring 开源的 Initializr 项目构建。 不仅能够帮助用户管理依赖，还能帮助用户生成测试或者可以直接使用的代码片段，用户创建完工程就可以进行测试，测试完就可以基于示例代码进行项目开发。因此，基于云原生应用脚手架构建项目仅需完成：新建工程和运行测试即可。

- Cluster API 声明式集群生命周期管理工具 v1.5.0 发布（CNCF 项目）

  版本特性：支持微服务预检查以提高集群稳定性、在 classy 集群中支持并发的 MachineDeployment 升级、在 clusterctl 中支持附加提供程序、改进规模化部署时的性能、通过 MachinePool Machine 提高 MachinePool 的可观测性、clusterctl 插件允许从 clusterctl 调用自定义代码、通过自定义 Kube State Metrics 配置可以收集更多指标。

- Cluster API 声明式集群生命周期管理工具 v1.3.0 发布（CNCF 项目）

  该版本主要新特性：支持自动更新由 Kubeadm 控制平面提供商提供的机器证书、可以从新的容器镜像注册中心 registry.k8s.io 发布和消费集群 API 镜像、允许在控制平面节点上创建没有污点的集群、clusterctl 现在可以管理 IPAM 和 RuntimeExtension 提供者。

- Clusternet 多云多集群管理项目 v0.16.0 发布（CNCF 项目）

  更新内容：支持将工作负载从未就绪的集群迁移到健康的备用集群上、支持为未就绪集群添加污点、可配置集群得分的百分比来进行调度、添加许可证扫描报告和状态、支持记录健康检查的指标数据。

- Clusternet 多云多集群管理项目 v0.15.0 发布（CNCF 项目）

  版本特性：将所有控制器从 clusternet-hub 中移出，放入一个名为“clusternet-controller-manager”的新组件中；增加多集群服务发现的功能门；为调度器迁移验证添加单元测试。

- Clusternet 分布式云原生多集群管理项目入选 CNCF 沙箱项目

  3 月 8 日，Clusternet 通过 CNCF 基金会 TOC 委员会的评定，成功入选 CNCF 沙箱项目。 Clusternet 通过组件化的、无侵入、轻量化的方式，将 Kubernetes 强大的单集群能力扩展至多集群，并且很好地兼容云原生生态。 未来，Clusternet 将探索更丰富多集群的落地场景，推动建立功能完备、API 标准化的多集群框架。

- Clusternet 多云多集群管理项目 v0.13.0 发布

  该版本主要新特性：增加从父集群到子集群 pod 的路由功能、添加调度器配置并支持自定义调度器插件、支持 discovery v1beta1、只为 k8s v1.21.0 及更高版本提供发现 endpointslice 的支持、使用阈值聚合工作节点标签、支持按集群 subgroup 进行调度、为在 capi 中运行的 clusternet-agent 更新 RBAC 规则。

- Constellation：Edgeless Systems 开源首个机密计算 Kubernetes

  Constellation 旨在提供一个端到端的机密 K8s 框架。 它将 K8 集群包裹在一个机密的上下文中，与底层云基础设施屏蔽开。 其支持运行和扩展所有容器；提供基于 Sigstore 的节点和工件证明；提供基于 Cilium 的网络方案等。

- Consul 服务发现框架 v1.13.0 发布（CNCF 项目）

  该版本主要新特性：移除对 Envoy 1.19 的支持；Cluster Peering 支持联合 Consul 集群用于服务网格和传统服务发现；允许在不修改目录的情况下，通过透明代理模式的终端网关路由出口流量。

- containerd 容器运行时 v1.7.0 发布（CNCF 项目）

  版本特性：新增 Sandbox API，简化更高级别的容器组的管理，为 shim 实现和客户端提供了新的扩展点；新增 Transfer Service，可用于在源和目标之间传输工件对象；支持扩展节点资源接口 NRI 的范围，实现常见的可插拔式运行时扩展；增加对 CDI 设备注入的支持；支持 cgroups blockio；为增强型重启管理器增加重启策略；初步支持 gRPC shim。

- containerd 完成模糊测试审计

  此次 containerd 审计共增加了 28 个模糊测试器（fuzzer），涵盖了广泛的容器运行时功能。 在此审计过程中，在 OCI 镜像导入程序中发现了一个漏洞：导入恶意镜像可能会导致节点的 DoS 攻击。 该问题已在 containerd 1.5.18 和 1.6.18 中修复。

- Contour Kubernetes ingress 控制器 v1.25.0 发布（CNCF 项目）

  版本特性：HTTPProxy 支持配置 IP 过滤器、支持将追踪数据导出到 OpenTelemetry、支持所有主机的外部授权、支持内部重定向。

- Contour Kubernetes ingress 控制器 v1.22.0 发布（CNCF 项目）

该版本主要新特性：支持 Gateway API v0.5.0、允许在单一的路由中为一组条件配置一个直接响应、可以为用户证书验证启用撤销检查、合并访问记录和 TLS 密码套件验证。

- Contour Kubernetes ingress 控制器 v1.21.0 发布 （CNCF 项目）

  该版本主要新特性：Contour 访问 leader election 资源的 RBAC 转移至命名空间角色；容器镜像现在只在 GitHub 容器注册中心（GHCR）上发布；新增 contour gateway-provisioner 命令和部署清单，用于动态配置 Gateways。

- CoreDNS DNS 服务器 v1.11.0 发布 （CNCF 项目）

  版本特性：支持通过 QUIC 协议（doq）接受 DNS 连接、支持对 CNAME 记录的目标进行重写、kubernetes 插件中移除对 Endpoint 和 Endpointslice v1beta 的支持。

- Cortex Prometheus 长期存储方案 v1.14 发布

  该版本主要新特性：移除对块存储的支持、实验性地支持垂直查询分片、启用 PromQL @修改器、可以使用 OTel 收集器将追踪信息发送到多个目的地、多项性能改进和问题修复。

- Cortex 分布式 Prometheus 服务 v1.13.0 发布（CNCF 项目）

  该版本主要新特性：新增元数据 API 查询器的 streaming 功能、为 compactor 提供实验性的 shuffle sharding 支持、修复 Distributor 和 Ruler 的内存泄漏、在分发请求时加入一个抖动来重置每个 pod 的初始时间。

- Crane 成本优化工具推出开源云原生应用碳排放计算优化器

  碳排放计算优化器基于运行在 Kubernetes 平台上的应用的实际资源消耗，计算对应服务器功耗，进而计算出应用运行所产生的碳排放量。 此

外，还支持提供 Pod 资源配置、workload 副本数、HPA 参数配置等的优化建议，以及优化后的功耗和碳排放推算结果。

- CRI-O 容器运行时 v1.28.0 发布（CNCF 项目）

  版本特性：允许用户禁用 Pod 的主机端口映射、新增指标以显示 Pod 和容器何时卡在创建的不同阶段、支持指定 kubelet 不要对某些镜像执行垃圾回收、添加版本升级自动化脚本、支持可配置指标输出器的命名空间、支持通过 CRI 的镜像策略验证错误。

- CRI-O：CNCF 宣布容器运行时项目 CRI-O 毕业

  CRI-O 为 Kubelet 提供了一个安全、高效和稳定的容器运行时接口实现，用于在生产 Kubernetes 环境中编排 Open Container Initiative（OCI）容器。未来，CRI-O 计划改进上游文档、自动化发布过程、增加节点上的 Pod 密度等等。该项目还正在努力将某些部分移动到 Rust 语言。

- CRI-O 容器运行时 v1.26.0 发布（CNCF 项目）

  该版本主要新特性：移除对 CRI v1alpha2 的支持（支持 v1）、增加对 OTLP 追踪支持、为 OTel 追踪添加日志和 GRPC 错误代码支持、支持 Evented PLEG、增加 seccomp 通知器功能、对没有基础设施的容器增加检查点和恢复容器的支持、允许完全更新运行时配置、允许在重载配置时添加其他运行时以及改变默认运行时。

- CRI-O 容器运行时 v1.25.0 发布（CNCF 项目）

  该版本主要新特性：支持设置运行时 pod 的最大日志文件大小、能够执行 Kubelet 的用户命名空间配置、新增注释用于配置容器的电源管理、 增加最小 checkpoint/restore 支持、通过 sigstore 签名来签署静态二进制包。

- CRI-O 容器运行时 v1.24.0 发布（CNCF 项目）

该版本主要新特性：默认阻止没有 CAP_SYS_ADMIN 的容器的 unshare 系统调用、使用任务集来生成新的 cri-o 运行命令、在 CRI-O HTTP API 中添加暂停和取消暂停功能。

- Crossplane 云原生控制平面构建框架完成模糊测试安全审计

  此次模糊测试共发现 4 个问题。其中一个问题是：允许部分不受信任的用户控制 crossplane-runtime 在某个状态下分配的内存量，这可能会造成由于资源耗尽而导致 DoS 攻击。该漏洞的修复版本 crossplane-runtime 0.19.2 和 Crossplane 1.11.2 已经发布。

- CubeFS 分布式存储系统 v3.3.0 发布（CNCF 项目）

  版本特性：添加对 ObjectNode 存储桶策略的支持、为 ObjectNode 添加跨域资源共享（CORS）支持、支持在执行重命名、删除、创建等操作时保证原子性、支持配置和动态调整 MP（多路复，Multiplexing）步长、添加对 UID 空间限制的支持、支持 autofs 挂载功能。

- CubeFS 分布式存储系统 v3.1.0 发布（CNCF 项目）

  该版本主要新特性：提供 QoS 服务以完善多租户隔离功能、优化混合云多级读缓存功能、支持两个副本的数据存储、卷支持配置 posixAcl 进行权限管理、为 datanode 添加数据分区总数限制。

- CubeFS 分布式存储系统从 CNCF 沙箱升级为孵化项目

  CubeFS 是国内首个具备完整的对象及文件存储能力的云原生存储产品。CubeFS 支持多副本和纠删码引擎，提供多租户、多 AZ 部署、跨区域复制等特性；适用于大数据、AI、容器平台、数据库及中间件存算分离等广泛场景。

D

- Dapr 分布式应用运行时 v1.11.0 发布（CNCF 项目）

版本特性：配置构件现在是一个 v1 版稳定的 API；服务调用支持调用非 Dapr 端点；支持在管理 API 中暂停、重启和清除工作流；引入密码学构建块；现在有两个可用的构建版本（包含所有的组件和只包含稳定组件）；Dapr 仪表板不再与控制平面一起安装；提供用于 Windows Server 2022 的容器镜像；应用程序健康检查升级为 stable。

- Dapr 分布式应用运行时 v1.10.0 发布（CNCF 项目）

  版本特性：新增 Dapr Workflows 用于跨多个应用建立长期运行的进程或数据流、支持批量发布和订阅信息、允许创建可用任何语言编写的可插拔组件 SDK、新增 Multi-App Run 功能改善本地开发、弹性策略升级为 stable、新增服务调用指标。

- Dapr 分布式应用运行时 v1.9.0 发布（CNCF 项目）

  该版本主要新特性：允许自定义可插拔组件、支持 OTel 协议、增加弹性观测指标、支持应用健康检查、支持设置默认弹性策略、允许使用任何中间件组件进行服务见调用、新增 pub/sub 命名空间消费者组、支持 Podman 容器运行时。

- DeepFlow 自动化云原生可观测平台的首个开源版本正式发布

  DeepFlow 是云杉网络研发的可观测性平台，基于 eBPF 等技术，实现自动同步资源、服务、K8s 自定义 Label 并作为标签统一注入到观测数据中。 它能够自动采集应用性能指标和追踪数据而无需插码，SmartEncoding 创新机制将标签存储的资源消耗降低 10 倍。 此外，还能集成广泛的数据源，并基于 SQL 提供北向接口。

- Devspace K8s 开发工具成为 CNCF 沙箱项目

  12 月 15 日，CNCF 宣布 DevSpace 正式成为 CNCF 沙箱项目。 通过 Devspace，可以直接在 Kubernetes 中构建、测试和调试应用；可以使用热重载进行开发； 统一团队内部以及跨开发、暂存和生产的部署工作流；自动执行镜像构建和部署的重复性任务。

- DevSpace K8s 开发工具 v6.0 发布

  该版本主要新特性：引入 pipeline 用于 管理 devspace.yaml 中的任务、新增导入功能将不同 devspace.yaml 文件合并在一起、新增 proxy command 在本地计算机上运行在容器中执行的命令。

- Dragonfly 镜像和文件分发系统 v2.1.0 发布 （CNCF 项目）

  版本特性：可视化控制台 v1.0 发布、新增虚拟网络拓扑探索功能、支持 Dragonfly 作为 JuiceFS 的后端存储、支持在 Manager 中控制 Scheduler 的功能、新增个人访问令牌功能、新增 Cluster 资源单位（Cluster 代表一个 P2P 集群）。

E

- Envoy v1.27.0 发布（CNCF 项目）

  版本特性：引入新的 golang 网络过滤器、新增用于在资源不足时拒绝请求的负载削减点（Load shed point）、支持 CONNECT-UDP、引入 Open Telemetry 兼容的统计数据收集器、新增用于打印 CEL 表达式的访问日志格式化器。

- Envoy v1.26.0 发布（CNCF 项目）

  版本特性：增加了对通用代理的跟踪支持、支持在 http 过滤器链的任何位置修改请求和响应头信息、支持在动态元数据中设置 JWT 认证失败状态代码和消息、增加过滤器状态输入、支持在 TLS 握手和过滤器匹配之前启用速率限制、支持上游访问日志中的路由信息、支持动态地禁用 TCP 隧道、增加负载均衡器 Maglev 扩展和环形哈希扩展。

- Envoy v1.23.0 发布（CNCF 项目）

  该版本主要新特性：在一个 SDS 请求中发送多个集群或监听器的 SDS 资源、通过 HTTP 过滤器的配置名称来获取过滤器配置、监听器过滤器统计

更新、dns_resolver 增加对多个地址的支持、为监听器过滤器添加动态监听器过滤器配置等。

- Envoy Gateway API 网关 v0.5 发布

  版本特性：添加数据平面代理遥测、支持直接配置 xDS、支持基于 IP 地址的不同速率限制、支持配置 EnvoyProxy Pod 标签、支持配置 EnvoyProxy 部署策略设置、卷和卷挂载、支持将 EnvoyProxy 配置为 NodePort 类型服务、添加 Pprof 调试支持。

- Envoy Gateway 基于 Envoy 的网关 v0.4 发布

  版本特性：支持通过 Helm 安装 Envoy Gateway；新增初始框架用于扩展 Envoy Gateway；支持基于 IP 子网的速率限制；支持自定义 Envoy 代理引导配置、Envoy 代理镜像和服务配置注释、资源和安全上下文设置等；支持 EDS（Endpoint Discovery Service)。

- Envoy Gateway v0.3 发布

  版本特性：支持扩展的 Gateway API 字段；支持 TCP 路由 API、UDP 路由 API 和 GRPC 路由 API；支持全局速率限制；支持请求认证。

- Envoy Gateway API 网关 v0.2 发布

  该版本主要新特性：支持 Kubernetes、支持 Gateway API 资源。

- Envoy Gateway 开源

  Envoy Gateway 正式成为 Envoy 代理家族的新成员，该项目旨在降低将 Envoy 作为 API 网关的使用门槛。 Envoy Gateway 可以被认为是 Envoy Proxy 核心的一个封装器。 它提供的功能包括：为网关用例提供简化的 API、开箱即用的控制器资源、控制平面资源、代理实例等的生命周期管理能力以及可扩展的 API 平面。

- Emissary Ingress 云原生 ingress 控制器和 API 网关 v2.3.0 发布（CNCF 项目）

  该版本主要新特性：当使用 lightstep 作为 driver 时，可以在 TracingService 的配置中设置 propagation_modes；支持在 Host 和 TLSContext 资源中设置 crl_secret，以比对证书撤销列表检查对等证书；优化与外部日志服务的通信等。

- eunomia-bpf：eBPF 轻量级开发框架正式开源

  eunomia-bpf 由各高校和龙蜥社区共同开发，旨在简化 eBPF 程序的开发、分发、运行。在 eunomia-bpf 中，只需编写内核态代码即可正确运行，在部署时不需要重新编译，并提供 JSON/WASM 的标准化分发方式。

F, G

- Falco 运行时安全项目 v0.35.0 发布（CNCF 项目）

  版本特性：引入指标快照选项和新的指标配置、支持使用 cosign 对发布的容器镜像进行签名、允许自定义 CA 证书和存储、支持管理 Talos 预构建驱动程序、Mesos 不再支持元数据增强功能。

- Falco 2023 年安全审计结果发布

  审计报告发现了一个中等严重程度的漏洞和若干低严重程度和信息严重程度的漏洞，不存在严重程度高的漏洞。所有问题已在 Falco 0.34.0 和 0.34.1 补丁版本中修复。

- Falco 运行时安全项目 v0.34.0 发布（CNCF 项目）

  版本特性：支持手动下载和应用相关的规则 application_rules.yaml、新检测规则使用 PTRACE 向进程注入代码、规则结果添加编译条件上下文、允许现代 bpf 探针为一个环形缓冲区分配一个以上的 CPU、添加 webserver 端点以检索内部版本号、在 systemd unit 中支持多个驱动。

- Falco 运行时安全项目 v0.32.0 发布（CNCF 项目）

  该版本主要新特性：新增配置项，当检测到规则或配置文件有变化时，触发 Falco 重启；支持检测权限过大的容器；支持检测共享主机 pid 和 IPC 命名空间的 pod 等。

- Finch：AWS 开源的容器开发客户端命令行工具

  Finch 可用于构建、运行和发布 Linux 容器。 它为 Lima、nerdctl、containerd 和 BuildKit 等开源工具提供了原生的、可扩展的 macOS 客户端安装程序，简化在 macOS 上使用 Containerd。 用户可以使用 Finch 在本地端创建和执行容器，并发布 OCI 容器镜像文件。

- Flagger 渐进式交付项目 v1.31.0 发布（CNCF 项目）

  版本特性：支持服务网格 Linkerd 2.12 及更高版本、删除 OSM e2e 测试。

- Flagger 渐进式交付项目 v1.22.0 发布（CNCF 项目）

  该版本主要新特性：支持以 KEDA ScaledObjects 替代 HPA、在参数表中添加命名空间参数、为 Canary.Service 添加可选的 appProtocol 字段。

- Fluent Bit 日志处理工具 v2.0.0 发布 （CNCF 项目）

  该版本主要新特性：增加对 Traces 的支持（与 Prometheus 和 OpenTelemetry 完全集成）、允许 input 插件在一个单独的线程中运行、所有需启用安全的网络传输层将使用 OpenSSL、input 插件新增原生 TLS 功能、支持将更多的插件类型与 Golang 和 WebAssembly 集成、支持检查流经管道的数据、引入收集和处理内部指标的新 input 插件。

- Fluentd 日志收集工具 v1.15.0 发布（CNCF 项目）

该版本主要新特性：支持设置日志收集的速率限制规则、支持处理 YAML 配置格式、允许设置重启 worker 的时间间隔。

- Flux 持续交付工具 v2.0 发布（CNCF 项目）

  版本特性：GitOps 相关的 API 升级为 v1；Flux 控制器增加水平扩展和分片功能；Git bootstrap 功能升级为稳定；Flux 项目供应链的构建、发布和证明部分暂时符合 SLSA Build Level 3；与 Kubernetes Workload Identity for AWS、Azure 和 Google Cloud 完全集成；警报功能优化。

- Flux 持续交付工具成为 CNCF 毕业项目

  Flux 是一套面向 Kubernetes 的持续渐进交付解决方案，支持开发者和基础设施团队的 GitOps 和渐进交付。自 2021 年 3 月成为 CNCF 孵化项目以来，Flux 及其子项目 Flagger 在用户群、集成、软件使用、用户参与度、贡献等方面增长了 200–500%。

- Flux 持续交付工具 v0.34.0 发布（CNCF 项目）

  该版本主要新特性：Flux 控制器的日志与 Kubernetes 的结构化日志保持一致、允许为非 TLS 容器仓库定义 OCI 源、从多租户集群的容器仓库处提取 OCI 工件时优先考虑静态凭证而非 OIDC 提供者。

- Gatekeeper 策略引擎 v3.13.0 发布 （CNCF 项目）

  版本特性：支持将审计信息发布到 PubSub 系统中、验证工作负载资源的 ExpansionTemplates 升级为 beta、添加了实验性的 VAP 驱动器用于验证和审查资源对象的合规性、添加了对外部数据提供者审计缓存的支持、支持获得有关准入、审计和 gator CLI 的可观测性统计数据。

- Gatekeeper 策略引擎 v3.10.0 发布 （CNCF 项目）

  该版本主要新特性：移除 Pod 安全策略并迁移到 Pod 安全准入、Mutation 功能升级为稳定、引入工作负载资源的验证（alpha）、性能改进。

- Gatekeeper 策略引擎 v3.9.0 发布（CNCF 项目）

  该版本主要新特性：增加约束模式验证测试、增加对外部数据提供商的 TLS 支持、增加 pod 安全上下文变量、支持验证子资源、允许在 gator 验证中跳过测试、为 gator 添加 dockerfile、添加 opencensus 和 stackdriver exporter。

H

- HoloInsight：由蚂蚁集团开源的智能可观测平台

  HoloInsight 是蚂蚁集团观测平台 AntMonitor 的开源版本，它主要聚焦基于日志的实时观测能力、业务指标监控以及时序智能和 AIOps。 同时，它也融入可观测领域有共识的其他数据类型，比如 Trace，Event 等。

- HAProxy Kubernetes Ingress Controller v1.8 发布

  该版本主要新特性：降低容器中所有进程的权限，不再默认运行 privileged 容器；暴露了一个用于查看 pprof 诊断数据的端点；支持收集控制器内部的 Prometheus 指标，如分配的内存量和花费的 CPU 时间；若入口规则不匹配，支持自定义设置后端端口。

- Harbor 容器镜像仓库 v2.8.0 发布（CNCF 项目）

  版本特性：支持 OCI distribution spec v1.1.0-rc1、支持通过 CloudEvents 格式发送 Webhook 负载、支持跳过任务扫描器的自动更新拉取时间、移除 helm chart 仓库服务器 ChartMuseum。

- Harbor 容器镜像仓库 v2.7.0 发布（CNCF 项目）

  该版本主要新特性：添加 Jobservice 仪表盘，以监测和控制 job 队列、调度器和 worker；支持按块复制镜像 blob；为工件列表添加 WASM 过滤器。

- Harbor 容器镜像仓库 v2.6.0 发布（CNCF 项目）

该版本主要新特性：引入缓存层，改善高并发情况下拉取工件的性能；增加 CVE 导出功能，允许项目所有者和成员导出由扫描仪生成的 CBR 数据；支持定期清理审计日志或按需运行，支持转发审计日志到远程系统日志终端；支持 WebAssembly 工件。

- HashiCorp Vault 私密信息管理工具 1.11 新增 Kubernetes Secret 引擎

  Kubernetes Secret 引擎可以动态生成 Kubernetes 服务账号令牌、服务账号、角色绑定和角色。创建的服务账号令牌有一个可配置的 TTL 值（Time To Live），当 lease 到期时，Vault 会自动删除创建的对象。对于每一个 lease，Vault 会创建一个连接到定义服务账号的令牌，服务账号令牌会返回给调用者。

- Helm 完成模糊测试安全审计

  本次审计共编写了 38 个模糊器，测试范围覆盖 chart 处理、版本存储和仓库发现等关键部分。 共发现 9 个问题（目前已修复 8 个），其中，4 个 空指针引用问题，4 个内存不足问题，1 个栈溢出问题。

- Helm 包管理工具 v3.10.0 发布（CNCF 项目）

  该版本主要新特性：支持通过命令行设置 json 值、允许在执行 helm list 的时候不输出表头、增加参数用于配置客户端节流限制、支持控制是否要跳过 kube-apiserver 的证书校验。

- Helm Dashboard：Komodor 开源 Helm 图形界面

  Helm Dashboard 可在本地运行并在浏览器中打开，可用于查看已安装的 Helm Chart，查看其修订历史和相应的 k8s 资源。此外，还可以执行简单的操作，如回滚到某个版本或升级到较新的版本。

- Helm v3.9.0 发布 （CNCF 项目）

  该版本主要新特性：新增字段以支持向 post renderer 传递参数、签名过程增加了更多检查、更新对 Kubernetes 1.24 的支持。

- Higress：阿里云开源的云原生网关

  Higress 遵循 Ingress/Gateway API 标准，将流量网关、微服务网关、安全网关三合一，并在此基础上扩展了服务管理插件、安全类插件和自定义插件，高度集成 K8s 和微服务生态，包括 Nacos 注册和配置、Sentinel 限流降级等能力，并支持规则变更毫秒级生效等热更新能力。

- Horizon：由网易云音乐开源的云原生应用部署平台

  Horizon 是一个基于 Kubernetes 的云原生持续部署平台，并且全面践行 GitOps。 平台团队可以自定义创建版本化的服务模板，为业务应用程序和中间件定义符合统一标准的部署和运维。 开发团队可以选择预先定义的模板，进行自动化的服务部署，确保基于 Kubernetes 的统一最佳实践。通过 Horizon GitOps 机制，确保任意变更（代码、配置、环境）持久化、可回滚、可审计。

- HwameiStor 云原生本地存储成为 CNCF 沙箱项目

  6 月 23 日，由「DaoCloud 道客」开源的云原生本地存储 HwameiStor 高票入选 CNCF 沙箱项目。HwameiStor 将系统节点上的所有本地磁盘进行统一管理，形成不同类型的本地存储资源池，并通过 CSI 标准接口提供本地数据卷服务，为有状态的云原生应用或组件提供数据持久化能力。

- iLogtail 可观测数据采集器的全部代码开源

  近日，阿里云正式发布完整功能的 iLogtail 社区版。 本次更新开源全部 C++ 核心代码，该版本在内核能力上首次对齐企业版。新增日志文件采集、容器文件采集、无锁化事件处理、多租户隔离、基于 Pipeline 的新版配置方式等诸多重要特性。

- Ingress-NGINX Controller v1.7.0 发布

版本特性：支持 golang 1.20、移除对 Kubernetes 1.23 的支持、集成 OpenTelemetry 模块。

- Istio 项目正式从 CNCF 毕业

  作为一个孵化项目进入 CNCF 不到一年的时间，Istio 就成为了 CNCF 的毕业项目，创下 CNCF 历史上最快毕业的项目记录。从 CNCF 毕业意味着，Istio 已经成为现代应用程序网络的关键组件，表明它是一个经过验证的成熟项目，可用于在生产中扩展关键应用程序。

- Istio 服务网格 v1.18 发布（CNCF 项目）

  版本特性：发布数据平面模式 —— Ambient Mesh（环境网格）；多项 Kubernetes Gateway API 改进，包括支持 v1beta1 版本、自动部署逻辑不再依赖 pod 注入；实现跨部署类型的并发配置一致性；Istioctl 增强。

- Istio 服务网格 v1.17 发布（CNCF 项目）

  版本特性：金丝雀升级的修订标签升级为 Beta、基于 Helm 安装 Istio 升级为 Beta、完全兼容最新版 Kubernetes Gateway API（0.6.1）、优化 IPv4/IPv6 双栈支持、增加对监听器过滤器补丁的支持、支持使用加解密技术 QuickAssist Technology (QAT) PrivateKeyProvider。

- Istio 公布 2022 年安全审计结果

  本次审计未发现关键问题，共发现 11 个安全问题，所有问题目前都已修复。其中，发现的唯一一个 CVE 漏洞是 Go 的请求走私漏洞，最常见问题与 Istio 通过网络获取文件有关。

- Istio v1.16 发布（CNCF 项目）

  该版本主要新特性：三个功能升级为 beta：外部授权功能、Kubernetes Gateway API 实现、基于路由的 JWT 声明；增加对 sidecar 和入口网关的 HBONE 协议的实验性支持、支持 MAGLEV 负载均衡算法、通过 Telemetry API 支持 OpenTelemetry tracing provider。

- Istio 正式成为 CNCF 孵化项目

  Istio 指导委员会于今年 4 月向 CNCF 呈递了移交项目的申请。经过近 5 个月的审核，现成为孵化项目。

- Istio 社区推出无 sidecar 代理的 Istio 数据平面模式 Ambient Mesh

  Ambient Mesh 将数据面的代理从应用 pod 中剥离出来独立部署，彻底解决 mesh 基础设施和应用部署耦合的问题。 通过引入零信任隧道（ztunnel）和 Waypoint proxy（路径点代理）实现零信任且减少了网格的资源占用，同时还可以与 sidecar 模式无缝互通，降低了用户的运维开销。

- Istio v1.14 发布

  该版本主要新特性：

  - 流量治理：支持向 Envoy 发送未就绪 endpoint；优化 egress 流量拦截功能；放宽设置 SNI 的限制条件；支持 filter 替换虚拟主机；在 Proxy Config 中增加 API runtimeValues，用于 Envoy 运行时配置。
  - 安全：支持通过 Envoy SDS API 进行 CA 集成、支持在 SDS 中使用 PrivateKeyProvider、支持工作负载的 TLS 配置 API。
  - Telemetry：新增 OpenTelemetry 访问日志、在日志中添加 WorkloadMode 选项。
  - 扩展：支持 WasmPlugin 通过 imagePullSecret 从私有仓库中拉取镜像。

J

- Jaeger 分布式追踪系统 v1.46.0 发布（CNCF 项目）

  版本特性：默认启用 OTLP 接收器、增加对 OpenTelemetry SpanMetrics 连接器的支持。

- Jaeger 分布式追踪系统 v1.36.0 发布（CNCF 项目）

  该版本主要新特性：支持报告 span size 指标、增加多租户支持。

- Jaeger v1.35.0 发布（CNCF 项目）

  该版本主要新特性：引入了通过 OpenTelemetry 协议（OTLP）接收 OpenTelemetry 追踪数据的能力、为 GRPC 服务器定义健康服务器、添加 flag 用于在 rollover 时启用/禁用依赖关系、为 Admin Server 添加 TLS 配置。

- Jaeger 分布式追踪系统 v1.34.0 发布（CNCF 项目）

  该版本主要新特性：为 hotrod 应用添加 kubernetes 实例、新增 streamingSpanWriterPlugin 以提高 grpc 插件的写入性能、MetricsReader 添加 metrics 仪表等。

- Jakarta EE 10 基于 Java 的框架发布，开启云原生 Java 时代

  Jakarta EE 10 引入了用于构建现代化和轻量级云原生 Java 应用的功能。 具体包括：新增配置文件规范，定义一个用于轻量级 Java 应用和微服务的多供应商平台；针对适合微服务开发的较小运行时，提供了规范子集，包括用于构建应用的新 CDI-Lite 规范；支持多态类型；将 UUID 标准化为基本类型并扩展查询语言和查询 API。

K

- k0smotron：Mirantis 开源 Kubernetes 控制平面项目 k0smotron

  k0smotron 本质上是一套 Kubernetes 控制器，能让你在单个 Kubernetes 集群中以 Pod 的形式运行和管理多个 Kubernetes 集群控制平面。实现真正的控制平面和工作平面分离，即在现有集群上运行的控制平面与工作平面没有直接的网络连接。此外，还支持将任何基础设施的工作者节点连接到集群控制平面。

- k8gb Kubernetes 全局负载均衡器 v0.10.0 发布（CNCF 项目）

该版本主要新特性：可以通过环境变量访问 LeaderElection、支持 OpenTelemetry 的 tracing 方案、支持创建 K8GB 指标的 Grafana 仪表盘样本、实现一致的轮询负载均衡策略。

- KapacityStack：由蚂蚁集团开源的云原生智能容量技术

  KapacityStack 提供了 IHPA（Intelligent Horizontal Pod Autoscaler）能力：支持在不同场景因地制宜地使用不同的算法进行扩缩容；支持在整个弹性过程中精细化地控制工作负载下每一个 Pod 的状态；在执行扩缩容时支持采用自定义灰度分批的变更策略；支持用户自定义的变更期稳定性检查；整个 IHPA 能力就拆分为了管控、决策、执行三大模块，任一模块都可以做替换或扩展。

- Karmada 多云多集群容器编排平台 v1.6.0 发布（CNCF 项目）

  版本特性：引入 FederatedHPA API，以解决跨集群扩展工作负载的要求；支持自动将不健康的应用程序迁移到其他可用的集群；引入新的声明式部署方式 Karmada operator；支持第三方资源解释器。

- Karmada 多云多集群容器编排平台 v1.5.0 发布（CNCF 项目）

  版本特性：支持多个调度组、默认调度器兼容与任何第三方调度器、内置解释器支持 StatefulSet、默认解释器支持 CronJob 聚合状态以及 Pod 中断预算（PodDisruptionBudget）。

- Karmada 多云多集群容器编排平台 v1.4.0 发布（CNCF 项目）

  该版本主要新特性：新增声明式资源解释器、支持声明分发策略/集群分发策略的优先级、通过指标和事件增强可观测性能力、故障切换/优雅驱逐 FeatureGate 升级为 Beta 且默认开启。

- Karmada 多云多集群容器编排平台 v1.3.0 发布（CNCF 项目）

该版本主要新特性：支持基于污点的优雅工作负载驱逐、引入多集群资源访问的全局代理、支持集群资源建模、新增基于 Bootstrap token 的集群注册方式、优化系统可扩展性等。

- Karmada 跨云多集群容器编排平台 v1.2.0 发布（CNCF 项目）

  该版本主要新特性：优化随时间变化的调度策略；支持跨区域部署工作负载；karmadactl 和 kubectl-karmada 支持更加丰富的命令；新增针对 Kubernetes 资源的分布式搜索和分析引擎（alpha）；实现自定义的资源状态收集。

- Karpenter 自动扩缩容工具 v0.19.0 发布

  该版本主要新特性：默认驱逐无控制器的 pod、将 AWS 设置从 CLI Args 迁移到 ConfigMap、支持 IPv6 自动发现、将 webhook 和控制器合并为一个二进制文件。

- Kata Container 容器安全项目 v3.1.0 发布

  版本特性：支持 AMD SEV-SNP 机密虚拟机；支持 EROFS 文件系统；完善 Docker/Moby 网络支持改进运行时（runtime-rs），包括增加对大页面（hugepages）的支持；增加 QEMU 日志记录功能；兼容 CRI-O 和 containerd 1.6.8；支持 Kubernetes 1.23.1-00。

- Kata Containers 安全容器运行时 v3.0.0 发布

  该版本主要新特性：新增 Rust 语言重写的容器运行时组件以及一个可选的集成虚拟机管理组件、支持主流的云原生生态组件（包括 Kubernetes、CRI-O、Containerd 以及 OCI 容器运行时标准等）、支持 cgroupv2、支持最新的 Linux 内核稳定版本。

- Kata Container 容器安全项目 v2.5.0 发布

  该版本主要新特性：支持 containerd shimv2 日志插件、支持 virtio-block 多队列、支持 QEMU 沙箱功能、支持 containerd 的核心调度、

kata-runtime iptables 子命令可在 guest 中操作 iptables、支持直接分配的卷。

- Katalyst 云原生资源管控系统 v0.3.0 发布

  版本特性：KCNR API 增加网络带宽资源的申请、调度、分配能力，并结合 EDT/TC 等限速方案提供网络带宽隔离能力；新增任务执行框架；新增异步执行框架；算法实现多 CPU Region 模式；混部能力增强。

- Katalyst：字节跳动开源的云原生资源管控系统

  Katalyst 的主要特点包括：完全孵化于超大规模混部实践，并在字节服务云原生化的进程中同步接管资源管控链路；搭载字节跳动内部的 Kubernetes 发行版 Enhanced Kubernetes 同步开源；系统基于插件化模式构建，用户可以在 Katalyst Framework 之上自定制各类调度、管控、策略、数据等模块插件等。

- KEDA：CNCF 宣布 K8s 自动缩放器 KEDA 毕业

  KEDA 是一个专门为 Kubernetes 设计的事件驱动自动缩放器。作为一个毕业项目，KEDA 团队计划提高项目的性能、多租户安装、监控和可观测性功能。并计划增加配置缩放行为和指标评估的能力，将碳和能源消耗考虑到缩放评估中，以及预测性自动缩放。

- KEDA 事件驱动自动伸缩器 v2.11.0 发布（CNCF 项目）

  更新内容：新增 Solr Scaler；支持暂停自动缩放；改进和扩展普罗米修斯指标；如果有多个带有 CPU 和内存扩展器的扩展器，能够将其扩展为零。

- KEDA 基于 Kubernetes 事件驱动自动缩放项目公布安全审计结果

  审计发现了 Redis Scalers 中的一个重大缺陷，该缺陷可能会影响系统的机密性、完整性或可用性。 这个问题与加密和绕过 TLS 有关，从而允许潜在的 MitM（中间人）攻击。 目前，该问题已修复。此外，基于审计结

果，KEDA 还更新了现有的安全工具链，引入了 semgrep 工具和 TLS 证书管理。

- KEDA 事件驱动自动伸缩器 v2.9.0 发布（CNCF 项目）

  该版本主要新特性：增加 CouchDB、Etcd 和 Loki 扩展器、引入 Grafana 仪表盘用于监控应用的自动缩放、在 KEDA Operator 中整合所有暴露的 Prometheus 指标、实验性的支持在轮询间隔期间为扩展器缓存指标值。

- KEDA 基于 Kubernetes 的事件驱动自动伸缩工具 v2.8.0 发布（CNCF 项目）

  该版本主要新特性：支持 NATS streaming、支持自定义 HPA 名称、支持在 ScaledJob 中指定最小 pod 副本数、允许 cpu/memory scaler 只对 pod 中的一个容器进行扩展。

- KEDA 基于 Kubernetes 事件驱动的扩缩容工具 v2.7.0 发布（CNCF 项目）

  该版本主要新特性：支持通过 ScaledObject 注释暂停自动缩放、新增基于 ARM 的容器镜像、支持以非 root 身份运行 KEDA 的默认安全模式、CPU、内存、Datadog 扩展器使用全局 metricType 代替 metadata.type 等。

- Kelemetry：字节跳动开源的面向 Kubernetes 控制面的全局追踪系统

  Kelemetry 从全局视角串联起多个 Kubernetes 组件的行为，追踪单个 Kubernetes 对象的完整生命周期以及不同对象之间的相互影响。通过可视化 K8s 系统内的事件链路，它使得 Kubernetes 系统更容易观测、更容易理解、更容易 Debug。

- Keptn 云原生应用程序生命周期编排项目 v0.19.0 发布（CNCF 项目）

该版本主要新特性：Helm-service 和 Jmeter-service 移到 keptn 贡献仓库、支持验证入站事件、引入签名的 Keptn Helm chart、支持通过 sigstore/cosign 签名所有发布/预发布的容器镜像。

- Keptn 云原生应用程序生命周期编排项目 v0.18.0 发布（CNCF 项目）

  该版本主要新特性：安装/卸载/升级命令不可用，改用 Helm 来操作 Keptn；在资源 API 中，尾部/将返回 404；弃用配置服务，所有 Keptn 的核心服务都依赖于资源服务。

- Keptn 云原生应用生命周期编排引擎升级为 CNCF 孵化项目

  Keptn 是基于事件的控制平面，使用声明式编程方法实现应用的连续交付和自动修复。Keptn 未来将支持 GitOps 和控制仓库的管理方式、RBAC、执行平面的远程管理等。

- Keptn 云原生应用持续交付和自动化操作工具 v0.16.0 发布（CNCF 项目）

  该版本主要新特性：resource-service 取代 configuration-service，以加快响应时间、支持在不停机的情况下升级 Keptn；v0.17 中，CLI 将删除安装/卸载/升级命令；支持直接向 Nats 发送事件；只有当服务连接到控制平面时，才会被视为准备就绪；允许在没有 distributor sideCar 的情况下运行 approval service。

- Keycloak 身份和访问管理项目 成为 CNCF 孵化项目

  Keycloak 提供对应用程序和 API 的集中身份验证和授权。Keycloak 可以与云原生生态系统很好地集成。 它支持在 Kubernetes 上运行，并可以使用 Operator Framework 构建的操作器进行安装。它还提供 Prometheus 指标，并与标准 Kubernetes 堆栈进行整合。 CNCF 许多项目直接集成了 Keycloak，用于身份验证和访问，包括 Argo、Envoy 和 Jaeger 等。

- Kindling 云原生可观测工具 v0.7.0 发布

版本特性：提供一个简易版视图来显示 Trace Profiling 数据、为 cpuevents 增加追踪功能、支持 NoAPM Java 应用的附属代理。

- Kindling 云原生可观测工具 v0.6.0 发布

  该版本主要新特性：在 cpu 事件中增加 tracing span 数据、增加 Trace Profiling 的调试工具、支持 RocketMQ 协议。

- Knative 基于 Kubernetes 的 serverless 架构方案 v1.11.0 发布（CNCF 项目）

  版本特性：域映射控制器逻辑已与 Serving 控制器合并、新增字段用于启用 Queue Proxy 时的资源请求和限制（仅适用于 CPU 和内存）、Activator 现在有一个单独的服务账号、支持通过服务级别的注释来配置 Queue Proxy 资源。

- Knative 模糊测试审计结果公布

  Knative 模糊测试报告中披露，此次模糊测试安全审计为 3 个 Knative 子项目编写了 29 个模糊测试器。这些模糊测试器发现了一个第三方依赖项中的问题，该问题目前已得到修复。

- Knative 基于 Kubernetes 的 serverless 架构方案 v1.8.0 发布（CNCF 项目）

  该版本主要新特性：修改默认域、升级 HPA 到 v2 版本、允许设置 seccompProfile 以启用使用受限的安全配置文件、最低 k8s 支持版本升至 v1.23、调和超时时间提高到 30 秒、默认启用 EmptyDir 卷功能参数。

- Koordinator 云原生混部系统 v1.0 发布

  该版本主要新特性：优化任务调度、优化 ElasticQuota 调度、支持细粒度的设备调度管理机制、支持根据节点的负载水位调整 BestEffort 类型 Pod 的 CPU 资源额度、支持使用 CPU Burst 来提高延迟敏感应用的性

能、实现基于内存安全阈值和资源满足的驱逐机制、精细化 CPU 调度、支持在不侵入 Kubernetes 已有的机制和代码前提下预留资源、简化自定义重调度策略的操作。

- Koordinator 云原生混部系统 v0.6.0 发布

  该版本主要新特性：完善 CPU 精细化编排策略、支持在不侵入 Kubernetes 已有机制和代码的前提下实现资源预留、支持 Pod 腾挪（PodMigrationJob）、发布了全新的 Descheduler 框架。

- Kruise Rollout 渐进式交付框架 v0.2.0 发布

  该版本主要新特性：支持 Gateway API、支持有状态应用的分批发布、新增了“Pod 批次打标”能力、集成到 KubeVela 之中实现 Helm Charts 金丝雀发布能力。

- KSOC Labs 发布第一个 Kubernetes 物料清单（KBOM）

  KBOM 是一款开源标准和命令行工具，可帮助安全团队快速分析集群配置并应对 CVE。该项目包含一份初始规范和实施方案，可跨云供应商、企业内部(on-premise）及自定义环境使用。安全和规范团队可通过使用 KBOM 对其 Kubernetes 集群（尤其是三方插件）获得更高的可见性，从而快速识别漏洞和维系。

- Kuasar：由华为云联合多家单位正式开源的多沙箱容器运行时

  Kuasar 在保留传统容器运行时功能的基础上，通过全面 Rust 化以及优化管理模型和框架等手段，进一步降低管理开销、简化调用链路，扩展对业界主流沙箱技术的支持。此外，通过支持多安全沙箱共节点部署，Kuasar 可以充分利用节点资源，实现降本增效。

- KubeAdmiral 字节跳动开源的基于 K8s 的新一代多集群编排调度引擎

  KubeAdmiral 是基于 Kubernetes Federation v2 迭代演进而来，旨在提供云原生多云多集群的管理和应用分发能力。KubeAdmiral 在其基础上做

了增强：兼容原生 Kubernetes API；提供更灵活的调度框架，支持丰富的调度分发策略；差异化策略；依赖调度/跟随调度；提供状态收集的框架，提供更灵活的状态收集。

- KubeClipper K8S 集群生命周期管理服务成为 CNCF 沙箱项目

  KubeClipper 基于 Kubeadm 封装，用户可以通过 Web 界面、API，或命令行工具（kcctl）来管理主机节点，可以快速创建和删除 K8S 集群，并可以对已存在的 K8S 集群进行纳管、升级、配置变更、应用部署，以及扩、缩容等操作。

- KubeClipper：九州云开源的 K8s 多集群全生命周期管理工具

  KubeClipper 基于 kubeadm 工具进行二次封装，提供在企业自有基础设施中快速部署 K8S 集群和持续化全生命周期管理（安装、卸载、升级、扩缩容、远程访问等）能力， 支持在线、代理、离线等多种部署方式，还提供了丰富可扩展的 CRI、CNI、CSI、以及各类 CRD 组件的管理服务。

- KubeEdge Sedna v0.6 & Ianvs 边云协同终身学习方案 v0.2 发布

  版本特性：支持非结构化数据场景下的开放世界边云协同终身学习；提供开源数据集、基线算法和评价指标的完整测试套件；针对机器人巡检、自动驾驶等场景，开发了新的未知任务识别和处理能力，包括新样本识别、训练数据生成、多模型联合推理等。

- KubeEdge 达到软件供应链 SLSA L3 等级

  在近期发布的 v1.13.0 版本中，KubeEdge 项目已达到 SLSA L3 等级（包括二进制和容器镜像构件），成为 CNCF 社区首个达到 SLSA L3 等级的项目。 这意味着，KubeEdge 可以端到端的从源码构建到发布流程进行安全加固，保障用户获取到的二进制或容器镜像产物不被恶意篡改。

- KubeEdge 云原生边缘计算平台 v1.12 发布（CNCF 项目）

该版本主要新特性：引入下一代云原生设备管理接口（DMI）、新版本的轻量级引擎 Edged 升级为 GA、EdgeMesh 新增高可用模式、支持边缘节点从云端升级、支持边缘 Kube-API 端点的授权、支持 GigE Device Mapper。

- KubeEdge 审计报告发布

  OSTIF（Open Source Technology Improvement Fund，开源技术改进基金会）完成了对 KubeEdge 的安全审计。审计发现了 12 个严重程度为中等的问题，建立了威胁模型，并集成到 OSS Fuzz。KubeEdge 安全团队已经在新发布的 v1.11.1、v1.10.2 和 v1.9.4 中对所有问题进行了修复。

- KubeEdge 云原生边缘计算平台 v1.11.0 发布（CNCF 项目）

  该版本主要新特性：新增节点组管理功能；提供边缘设备 Mapper 的 SDK，减少开发 Mapper 的工作量；正式支持容器化部署、离线安全等 Keadm 子命令；边缘节点代理 Edged 适用更多场景。

- Kubeflow：Kubernetes 机器学习平台 Kubeflow 升级成为 CNCF 孵化项目

  Kubeflow 是一个开源、社区驱动的项目，用于在 Kubernetes 上部署和管理机器学习（ML）堆栈。Kubeflow 社区积极开发和支持面向 Kubernetes 的 MLOps，为其用户开发和部署流行的框架，包括 TensorFlow、PyTorch、XGBoost、Apache MXNet 等分布式机器学习（ML）。

- KubeKey 集群部署工具 v3.0 发布

  该版本主要新特性：为 docker 构建和推送添加 GitHub 工作流、支持执行自定义设置脚本、添加 k3s 控制平面控制器和启动控制器、添加 k3s 容器运行时配置、添加 k3s e2e 测试支持、自定义 OpenEBS 基本路径、重构 KubeKey 项目、支持更多的 Kubernetes 和 k3s 版本。

- KubeKey 集群部署工具 v2.3.0 发布

  该版本主要新特性：添加 kubelet pod pid 限制、使用 Jenkins Pipeline 替代 GitHub Actions、在创建集群或添加节点时增加安全增强命令、删除集群或节点时清理 vip、支持 kube-vip BGP 模式、支持清理 CRI、支持 kube-vip、支持 k8s 最近发布的补丁版本。

- KubeKey 集群部署工具 v2.1.0 发布（KubeSphere 社区开源）

  该版本主要新特性：根据 OCI 标准实现了镜像的拉取、上传和归档保存等功能，使其在制作和使用 KubeKey 制品时不依赖额外的容器运行时；支持初始化操作系统命令（kk init os）使用制品进行离线本地源安装操作系统依赖；支持同一个 K8s 集群中同时包含 ARM64 节点和 AMD64 节点等。

- Kube-OVN CNI 插件 v1.12.0 发布（CNCF 项目）

  版本特性：优化底层调用 OVN 接口的方式；支持流量远程镜像；支持使用 IPSec 对集群内跨节点通信的流量进行加密；支持通过 kubectl-ko 插件一键收集所有 Kube-OVN 相关的组件日志、dmesg 信息、iptables 规则等和网络相关的详细信息；实现 Overlay 和 Underlay 网络的互通；新增 IPPool 的 CRD 资源；增加策略 NAT 功能；新增 OVN 原生的网关类型。

- Kube-OVN CNI 插件 v1.11.0 发布（CNCF 项目）

  该版本主要新特性：Underlay 和 Overlay 子网互通、新增 SR-IOV Network Operator 进行自动化网卡配置、支持自定义 VPC 内部负载均衡、新增 vpc-dns CRD、支持默认 VPC 下的 Load Balancer 类型 Service。

- Kube-OVN v1.10.0 发布（CNCF 项目）

该版本主要新特性：在子网新增 ACL 字段，用户可以按照自己的需求编写符合 OVN 流表规范的 ACL 规则； KubeVirt 场景下， VM instance 的地址分配策略采用和 StatefulSet 类似的策略，支持 DPDK、DHCP；集成 Submariner 用于多集群的互联；针对大规模环境，对控制平面性能进行了优化等。

- Kubernetes Cluster API v1.4.0 发布

  版本特性：支持在 KCP 控制平面部署失败时自动进行故障恢复、支持将某些标签从 Machine 同步到 Node、以将标签、注释等信息从 ClusterClass 传播到 KubeadmControlPlane/MachineDeployment 和 Machine 中、支持 ClusterClass 和 Managed Topologies 中的变量发现。

- Kubernetes v1.28 发布

  版本特性：支持控制平面和节点版本之间的偏差变化、节点非优雅关闭功能 GA、CRD 基于通用表达式语言（CEL）的验证规则 Beta、Kube APIServer 混合版本互操作代理 Alpha、 新增通用控制平面存储库 Alpha、设备插件 API 添加对 CDI 标准设备的支持 Alpha、原生支持 Sidecar 容器 Alpha、节点 Swap 内存支持 Beta、新增对 Windows 节点的支持 Beta。

- Kubernetes v1.27.0 发布

  版本特性：冻结 k8s.gcr.io 镜像仓库、SeccompDefault 升级为 GA、Job 可变调度指令升级为 GA、Pod 调度 Readiness 升级为 beta、允许通过 Kubernetes API 访问节点日志、引入新访问模式 ReadWriteOncePod 以将卷访问限制在集群中的单个 Pod 上、VolumeManager 重建升级为 beta。

- Kubernetes 1.24 的第三方安全审计结果发布

本次审计发现以下问题：在限制用户或网络权限方面存在问题，可能导致管理员混淆特定组件可用的权限；组件间身份验证存在漏洞，恶意用户能够获得集群管理员权限；日志和审计存在缺陷，攻击者可以在控制集群后利用这些缺陷来进行潜在活动；用户输入过滤存在漏洞，允许通过修改 etcd 数据存储的请求来绕过身份验证。

- Kubernetes v1.26 发布

  该版本主要新特性：允许用户跨命名空间从卷快照中配置卷、允许使用 CEL 进行 Admission Control、改进拓扑管理器中的多 numa 对齐、为 Pod 调度增加调度就绪的机制、移除 CRI v1alpha2、Auth API 允许获取 User 属性、减少 Kubelet 在 PLEG 中的 CPU 消耗。

- Kubernetes 发布 Kubernetes CVE 订阅列表

  Kubernetes 现在支持通过订阅一个自动更新的 JSON feed，跟踪 Kubernetes 的安全问题（也称为“CVE”，来自于不同产品和供应商的公共安全问题的数据库）。

- Kubernetes v1.25 发布（CNCF 项目）

  该版本主要新特性：升级到 Stable：Pod Security Admission 、临时容器、cgroups v2 支持、网络策略中的 endPort、本地临时存储容量隔离、核心 CSI Migration、CSI 临时卷升级；升级到 Beta：SeccompDefault、CRD 验证表达式语言；引入 KMS v2 API 等。

- Kubernetes v1.24.0 发布（CNCF 项目）

  该版本主要新特性：dockershim 组件从 kubelet 中删除；默认情况下关闭新的测试版 API；发布工件使用 cosign 进行签名，并且提供验证镜像签名的实验性支持；支持存储容量跟踪以显示当前可用的存储容量和卷扩展；gRPC 探针和 kubelet 凭证提供程序升级到 Beta；允许为服务的静态 IP 地址分配软保留一个范围等。

- Kubernetes Gateway API v0.5.0 发布

  该版本主要新特性：三个 API 升级为 beta：GatewayClass、Gateway 和 HTTPRoute；引入了 experimental 和 standard 版本渠道；通过指定端口号，路由可以连接到网关；支持 URL 重写和路径重定向。

- kube-rs Kubernetes 的 Rust 客户端 v0.79.0 发布（CNCF 项目）

  版本特性：增加对元数据 api 的支持以减少网络负载、暴露客户端的默认命名空间、允许在不消耗原始的 watcher stream 的情况下从 watcher 订阅事件、支持持久元数据 watch。

- KubeSkoop：由阿里云开源的 Kubernetes 网络监控工具

  KubeSkoop 支持功能包括：Pod 级别的网络监控，包括流量，应用层连接信息， socket 内存分配状态等；Pod 级别的网络异常状态的指标监控，例如 Pod 内进程对 socket 进行读写操作的等待时间超过 100ms 的次数等；Pod 级别的网络异常事件的现场，提供事件发生的详细信息的观测。

- KubeSphere 多租户容器平台 v3.4.0 发布（CNCF 项目）

  版本特性：添加 golangci-lint 工作流用于节点检查、集成 opensearch v1/v2、为 rulegroup 添加 admission webhook、添加 alerting v2beta1 api、支持使用标签选择器过滤工作区角色、为缓存添加动态选项、添加 helm 执行器通用包、生成 CRD 支持多个版本、支持 gitlab 身份提供者、支持节点和工作负载范围的全局规则、支持支持可插拔的通知功能。

- KubeSphere v3.3.0 发布（CNCF 项目）

  该版本主要新特性：

  - DevOps：后端支持独立部署，提供基于 GitOps 的持续部署方案，引入 Argo CD 作为 CD 的后端，可以实时统计持续部署的状态。

- 网络：集成负载均衡器 OpenELB，即使是在非公有云环境的 K8s 集群下，也可以对外暴露 LoadBalancer 服务。
- 多租户和多集群：集群应用可以通过一个 ConfigMap 获取到所在集群的名称、支持为集群设置集群成员和集群角色。
- 可观测性：添加容器进程/线程指标、优化磁盘使用率指标、支持在命名空间自定义监控中导入 Grafana 模板。
- 存储：支持 PVC 自动扩展策略、支持对卷快照内容和类型进行管理、支持为存储类型设置授权规则。
- 边缘：集成 KubeEdge。

- Kubespray Kubernetes 集群部署工具 v2.22.0 发布

  版本特性：同一镜像名称下支持多架构镜像、为 cert-manager 增加 DNS 配置、 kube-scheduler 配置中加入 kube-profile 配置、允许配置镜像垃圾收集、支持自定义 ssh 端口、支持启用 kube-vip 的控制平面负载均衡。

- Kubespray Kubernetes 集群部署工具 v2.20.0 发布

  该版本主要新特性：支持 Rocky Linux 8 和 Kylin Linux、在重置角色中添加“flush ip6tables”任务、支持 NTP 配置、添加 kubelet systemd 服务加固选项、为 CoreDNS/NodelocalDNS 添加重写插件、为 kubelet 添加 SeccompDefault admission 插件、为 k8s_nodes 增加 extra_groups 参数、新增 ingress nginx webhook、增加对节点和 pod pid 限制的支持、支持启用默认的 Pod 安全配置。

- KubeVela 混合多云环境应用交付平台 v1.8.0 发布（CNCF 项目）

  版本特性：支持通过多个分片对控制平面进行横向扩展、支持从现有的 KubeVela 定义中生成语言感知的 SDK、新增基于工作流的触发器 kube-trigger、允许应用开发人员以金丝雀发布的风格来协调应用交付过程。

- KubeVela 升级成为 CNCF 孵化项目

KubeVela 采用 Kubernetes 控制平面构建，使跨混合和多云环境部署和操作应用变得更容易、更快速、更可靠。 未来，KubeVela 社区计划通过交付工作流改善云资源创建和消费的用户体验，增强混合/多集群场景中整个 CI/CD 交付流程的安全， 支持允许用户轻松与第三方 API 集成的 KubeVela Dynamic API 等等。

- KubeVela 混合多云环境应用交付平台 v1.6.0 发布（CNCF 项目）

  该版本主要新特性：支持资源交付可视化、提供可观测基础设施搭建、面向应用的可观测、可观测即代码的能力、支持多环境流水线统一管理、支持应用间配置的共享并与第三方外部系统做配置集成。

- KubeVela 混合多云环境应用交付平台 v1.5.0 发布（CNCF 项目）

  该版本主要新特性：插件框架优化，提供创建脚手架、打包、推送到插件注册表等整个插件生命周期的管理；支持以 CUE 格式定义插件，并使用 CUE 参数来渲染部分插件；新增大量 vela cli 命令；VelaUX 支持管理由 CLI 创建的应用程序。

- KubeVela v1.4.0 发布（CNCF 项目）

  该版本主要新特性：支持多集群认证、使用 kubeconfig 的控制器自动登录、支持更多的授权方式；允许在 GC 策略中按资源类型选择资源、新增策略控制器为 VelaUX 和 CLI 参数生成 OpenAPI 模式；CLI 支持显示资源拓扑结构等。

- KubeVirt 虚拟机运行项目 v1.0 发布（CNCF 项目）

  版本特性：移除热插拔 VMI API、引入 CPU 热拔插、实验性支持 AMD 的安全加密虚拟化（SEV）、支持在偏好中设置 CPU 和内存最低资源需求、支持 VirtualMachine 对象上网络接口的热拔插、支持指定集群级别的虚拟机行为、增加多架构支持、 允许指定创建的虚拟机的内存、 允许在创建虚拟机时指定卷的启动顺序、克隆和导出虚拟机支持 RBAC 配置。

- KubeVirt 虚拟机管理插件 v0.58.0 发布（CNCF 项目）

  该版本主要新特性：在 cluster-deploy 中默认启用 DataVolume 垃圾回收、能够在启用受限的 Pod 安全标准时运行、添加 tls 配置、修复在有 containerd 的系统上带有 containerdisks 的虚拟机迁移失败问题。

- KubeVirt 虚拟机管理插件 v0.57.0 发布（CNCF 项目）

  该版本主要新特性：弃用 SR-IOV 热迁移功能门、弃用虚拟机实例预设资源、增加对虚拟机输出资源类型的支持、支持 DataVolume 垃圾回收、允许对配置虚拟机拓扑分布约束的支持。

- KubeVirt 虚拟机运行项目 v0.55.0 发布（CNCF 项目）

  该版本主要新特性：引入克隆 CRD、控制器和 API、引入弃用策略、增加 virt-launcher 的内存开销、启用内存转储到 VMSnapshot 中、支持监控 VMI 迁移对象从创建到某个特定阶段所需的时间、允许 VMI 从根目录迁移到非根目录。

- KubeVirt 虚拟机运行项目 v0.55.0 发布（CNCF 项目）

  该版本主要新特性：引入克隆 CRD、控制器和 API、引入弃用策略、增加 virt-launcher 的内存开销、启用内存转储到 VMSnapshot 中、支持监控 VMI 迁移对象从创建到某个特定阶段所需的时间、允许 VMI 从根目录迁移到非根目录。

- Kubewarden Kubernetes 策略引擎 v1.0.0 发布（CNCF 项目）

  该版本主要新特性：允许使用 Go、Rust 或 Swift 编写策略、支持使用 Kubewarden 策略取代每一个 Pod Security Policy、集成 OpenTelemetry、 利用 Sigstore 项目实现软件签名和验证。

- KubeWharf：字节跳动开源云原生项目集

KubeWharf 是一套以 Kubernetes 为基础构建的分布式操作系统，由一组云原生组件构成，专注于提高系统的可扩展性、资源利用率、可观测性、安全性等，支持大规模多租集群、在离线混部、存储等场景。 KubeWharf 第一批计划开源三个项目：高性能元信息存储系统 KubeBrain、kube-apiserver 七层网关 KubeGateway、轻量级多租户方案 KubeZoo。

- Kuma 服务网格项目 v2.2.0 发布（CNCF 项目）

  版本特性：支持 OpenTelemetry、支持使用 JSONPatch 定义 MeshProxyPatch 策略、支持重试指令和优先级、将底层 Envoy 版本升级到 1.25、新增策略以用于更精细地控制服务间的负载均衡、支持在 Kubernetes 集群中部署通用模式的全局控制平面（由 Postgres 支持）、支持为离线令牌签名和验证提供公钥。

- Kuma 服务网格 v2.0 发布（CNCF 项目）

  该版本主要新特性：在 CNI 和 init 容器配置中增加对 eBPF 的支持、新增 3 个“下一代”策略、优化用户界面、支持配置控制平面/API 服务器所支持的 TLS 版本和密码、允许配置多个 UID 使其被流量重定向忽略、允许在使用 iptables 进行流量重定向时开启日志功能。

- Kuma 服务网格项目 v1.8.0 发布（CNCF 项目）

  该版本主要新特性：跨网格网关支持多区域运行、网格网关/内置网关增加可观测性功能、重写 CNI、网格网关支持路径重写和报头添加/删除、支持过滤代理的指标、简化 TCP 流量日志实现、支持 Projected Service Account Tokens。

- Kurator 分布式云原生平台 v0.3.0 发布

  更新内容：新增模块 application manager，通过 GitOps 方式在整个 Fleet 中分发应用/配置；增加指标插件，支持多集群统一监控；新增策略管理器，提供一致的安全策略。

- Kurator 分布式云原生平台 v0.3.0 发布

  版本特性：在 Cluster API 的基础上增加了一个 CRD 集群，用户只需声明一个 API 对象即可管理 kubernetes 集群的生命周期；支持 kubernetes 集群升级；支持 kubernetes 集群扩缩容；支持在本地设置高可用 kubernetes 集群。

- Kurator 分布式云原生平台 v0.2.0 发布

  版本特性：支持基于 Thanos 的多集群监控及指标持久化存储；支持基于 Pixie 的实时 K8s 应用监控；新增组件 Cluster Operator，以云原生的方式来管理各种基础设施的 kubernetes 集群生命周期，包括公有云、混合云和本地数据中心。

- KusionStack：蚂蚁集团开源的可编程云原生协议栈

  KusionStack 通过自研的 DSL（KCL）沉淀运维模型（Kusion Model），将基础设施部分能力的使用方式从白屏转为代码化，同时结合 DevOps 工具链（Kusion CLI）实现配置快速验证和生效，以此提升基础设施的开放性和运维效率。

- Kusk Gateway 自助式 API 网关 v1.1.0 发布

  该版本主要新特性：支持指定服务或主机验证 authentication header、支持速率限制策略、通过在 OpenAPI 规范中注释简化 HTTP 缓存的复杂性、由 Envoy 本地处理所有 mocking。

- Kyma 云原生应用开发平台和运行时 v2.15.0 发布

  版本特性：计划将遥测组件重构成一个模块；在 Pod 重启和 webhook 修改的过程中，保持 webhook CA 捆绑稳定；新增 HTTP 请求的默认超时；简化了内部 Docker 注册中心的 Serverless 配置；支持 otel-collector 0.77.0。

- Kyverno 云原生策略引擎 v1.10.0 发布（CNCF 项目）

版本特性：将 Kyverno 分成 3 个独立的控制器/Deployment、支持集群内服务调用、支持验证 Notary v2 签名、更新生成和变更现有策略。

- Kyverno 云原生策略引擎 v1.8.0 发布（CNCF 项目）

  该版本主要新特性：新增 podSecurity 验证子规则，集成 Pod Security Admission 库；支持 YAML 清单签名验证；允许生成规则在单个规则中生成多个资源；支持 OpenTelemetry；支持测试生成策略；支持 Kubernetes 1.25。

- Kyverno 云原生策略引擎升级成为 CNCF 孵化项目

  Kyverno 是为 Kubernetes 打造的策略引擎，为 K8s 配置管理提供了自动化和安全性。 接下来项目计划添加一些特性，如 YAML 签名和验证、OpenTelemetry 支持、幂等（idempotent）自动生成的 pod 控制器策略、增强的 pod 安全标准集成、基于 OCI 的策略捆绑、集群内 API 调用等。

- Kyverno 云原生策略引擎 v1.7.0 发布（CNCF 项目）

  该版本主要新特性：支持通过动态客户端查询 mutate.target、允许 Kyverno jp 在 Yaml 文件上工作、优化镜像验证签名、在策略更新时 mutate 现有资源、允许用户在上下文中定义内联变量、禁用更新请求控制器的 leader election、在测试中支持 apiCall 以及 CLI 更新等。

L

- Lima Linux 虚拟机 v0.14.0 发布（CNCF 项目）

  该版本主要新特性：支持虚拟机共享文件系统 virtiofs、支持 Apple 的虚拟化框架 Virtualization.framework、支持 Containerd 命令行工具 nerdctl 1.1.0。

- Linkerd 服务网格 v2.14.0 发布（CNCF 项目）

版本特性：新增直接的 Pod 到 Pod 的多集群服务镜像功能、支持 Gateway API HTTPRoute 资源。

- Linkerd 服务网格项目 v2.13.0 发布（CNCF 项目）

  版本特性：引入客户端策略，包括动态路由和熔断器模式；支持调试基于 HTTPRoute 的策略；引入新的 init 容器 —— network-validator，确保本地 iptables 规则按预期工作。

- Linkerd 服务网格项目 v2.12.0 发布（CNCF 项目）

  该版本主要新特性：允许用户以完全零信任的方式定义和执行基于 HTTP 路由的授权策略；支持使用 Kubernetes Gateway API 进行配置；初始化容器增加对 iptables-nft 的支持。

- Litmus 混沌工程框架 v2.14.0 发布（CNCF 项目）

  该版本主要新特性：在 DNS 实验中添加对 containerd CRI 支持、支持在 service mesh 环境下执行 http-chaos 实验、在网络实验中增加源和目的端口支持、支持为混沌运行器的 pod 提供自定义标签、优化混沌结果中探针状态模式的描述。

- Litmus 混沌工程框架 v2.10.0 发布（CNCF 项目）

  该版本主要新特性：为 Kubernetes 应用增加 HTTP 混沌实验；引入 m-agent（机器代理），现在能够在非 k8s 对象上实施混沌；对混沌期间 应用状态检查失败时 节点警戒线实验的恢复 进行优化；在使用前端 nginx 时增加了对 Envoy 代理的支持；优化日志记录等 Litmusctl 更新。

- Longhorn 云原生分布式块存储 v1.5.0 发布（CNCF 项目）

  更新内容：推出基于 SPDK（存储性能开发工具包）的 V2 数据引擎、Cluster Autoscaler 功能升级成为 GA、实例管理器引擎和 Replica 合并、支持不同的卷备份压缩方法、自动化卷删除操作、通过 CSI

VolumeSnapshot 管理备份镜像、新增快照清理和删除 job、支持 CIFS 备份存储和 Azure 备份存储协议、新增 Kubernetes 升级节点耗尽策略。

- Longhorn 云原生分布式块存储 v1.4.0 发布（CNCF 项目）

  该版本主要新特性：支持 Kubernetes 1.25、ARM64 的支持升级为 GA、网络文件系统 NFS 的支持升级为 GA、支持卷快照校验、支持卷 Bit-rot 保护、提高卷复制的重建速度、支持通过删除旧快照来回收空间、支持在线卷扩展、允许用户创建一个停留在一致位置的副本卷、增加卷的 I/O 指标、支持备份和恢复 Longhorn 系统。

- Longhorn 云原生分布式块存储 v1.3.0 发布（CNCF 项目）

  该版本主要新特性：支持多网络 K8s 集群、兼容全托管 K8s 集群（EKS、GKE、AKS）、新增 Snapshot CRD、新增 Mutating & Validating admission webhooks、支持自动识别并清理无主/未使用的卷、引入 CSI 快照、支持通过 Kubernetes Cluster Autoscaler 进行集群扩展。

M

- Merbridge：由 DaoCloud 道客开源的服务网格加速器正式进入 CNCF 沙箱

  12 月 14 日，CNCF 基金会宣布 Merbridge 正式被纳入 CNCF 沙箱项目。Merbridge 是目前 CNCF 中唯一专注于使用 eBPF 加速服务网格的开源项目。 通过 Merbridge 只需要在 Istio 集群执行一条命令，即可直接使用 eBPF 代替 iptables，实现网络加速，提高服务性能。

- MetalLB Kubernetes 负载均衡器 v1.3.2 发布（CNCF 项目）

  该版本主要新特性：支持通过 CRD 进行配置（不再支持 ConfigMap）；可以在 L2 或 BGP 模式中广播地址，或者只分配 IP 不广播地址；为 L2 Announcement 和 BGP Announcement 增加节点选择器支持；新增

BGPPeer 选择器；支持使用 kustomize 进行更灵活的部署；增加 LoadBalancerClass 支持；支持多协议 BGP。

- Microcks API 模拟和测试项目成为 CNCF 沙箱项目

  Microcks 为不同 API 风格和协议 提供了一个事实标准，来加速和确保 API 的交付。Microcks 支持在创建 API contract 之前，使用“后端即服务”功能测试新的 API，并且支持 REST API、gRPC、GraphQL 和事件驱动的 API 等，无缝集成持续构建或流水线。

- MicroK8s 轻量级 Kubernetes 发行版 v1.25 发布

  该版本主要新特性：新增“严格限制”（strict confinement）的隔离级别以限制主机系统访问和实施更严格的安全态势、snap 大小减少了 25%、支持镜像侧加载（sideloading）、新增插件 kube-ovn 和 osm-edge。

- Mimir Prometheus 长期存储项目 v2.4.0 发布

  该版本主要新特性：引入查询调度器 query-scheduler，并支持 DNS-based 和 ring-based 两种服务发现机制；新增 API 端点暴露每个租户的 limit；增加新的 TLS 配置选项；允许限制最大范围查询长度。

- Mimir Prometheus 长期存储项目 v2.3.0 发布

  该版本主要新特性：支持摄取 OpenTelemetry 格式的指标、新增用于元数据查询的租户联盟、简化对象存储配置、支持导入历史数据、优化即时查询功能、默认启用查询分片。

- Mimir 新功能：将 Graphite、Datadog、Influx 和 Prometheus 的指标纳入统一的存储后端

  Mimir 是 Grafana Labs 在 Cortex 基础上开源的时序数据库。 Mimir 现在开源了三个代理，用于从 Graphite、Datadog 和 InfluxDB 摄取指标，并将这些指标存储在 Mimir 中，这为 Mimir 从任何系统摄取指标奠定了基础。 未来将支持本地摄取 OpenTelemetry 的 OTLP。

- MinIO 对象存储工具发布新功能：扩展版本库和正式支持 OPA（CNCF 项目）

  MinIO 对版本库进行了扩展，排除版本库中的某些前缀和文件夹，以提高 Spark S3A 连接器等应用的性能。此外，应外界广泛要求，MinIO 正式支持 OPA。

N

- Nacos 动态服务发现平台 v2.2.0 发布

  该版本主要新特性：支持批量注册和批量注销服务、支持 openAPI 2.0、增加多数据源插件、增加轨迹追踪插件、支持 Prometheus http 服务发现、支持 Ldaps 认证。

- Nacos 动态服务发现平台 v2.1.0 发布（CNCF 项目）

  该版本主要新特性：新增两个 SPI 插件：分别用于配置加解密和认证、支持集群 gRPC 客户端设置线程池大小、支持重置 raft 集群等。

- Narrows：由 VMware 开源的云原生安全检测器，能够在 Harbor 上增加容器安全的动态扫描

  Narrows 能够对 Kubernetes 集群和其中的工作负载进行运行时的安全态势评估，发现 Kubernetes 集群的错误配置，终止工作负载运行时中的攻击；对扫描报告进行汇总、聚合和分析并提供开放的 API 接口；与 Harbor 无缝集成，对于外部公共镜像仓库的镜像，可以自动同步到 Harbor 中，以生成安全数据。

- nerdctl Containerd 命令行工具 v1.3.0 发布

  版本特性：支持使用 notation 项目进行镜像签名和验证、支持 Port Windows 设备、新的项目维护者指南正式可用、修复不允许使用 systemd-homed 的无根模式的操作。

- nerdctl Containerd 命令行工具 v1.2.0

该版本特性：实验性支持读取 Kubernetes 容器日志、改进编译错误信息、允许在 Windows 上运行 Host Process 容器、添加 Windows HyperV 容器模式。

- NeuVector 容器安全平台 v5.2.0 发布

  版本特性：新增用于 CVE 查询的搜索 SaaS 服务、支持 NeuVector API 访问令牌、支持针对准入控制的镜像签名、增加对资源限制等自定义准入控制标准的支持、支持通过可插拔扫描仪接口从 Harbor 注册表调用 NeuVector 扫描仪、允许用户禁用网络保护、支持扫描 golang 依赖。

- NeuVector 容器安全平台 v5.0 发布

  该版本主要新特性：与 SUSE Rancher 集成，也能够与 Amazon EKS、Google GKE 和 Microsoft AKS 等其他企业级容器管理平台对接；支持 Web 应用程序防火墙检测；支持自动化容器保护；支持零漂移进程和文件保护以及对网络、进程/文件进行分割策略模式保护等。

- Nightingale 夜莺可观测性平台发布 V6 正式版

  版本特性：内置常见的中间件的监控仪表盘和告警规则；支持 LDAP、CAS、OIDC 等多种认证集成；内置支持告警自愈能力；支持灵活的告警规则、屏蔽规则、订阅规则、抑制规则；默认情况下，只需要一个二进制即可对接市面上常见的采集器；集成 ElasticSearch 数据源。

- Nightgale 夜莺监控 v5.10 发布

  该版本主要新特性：支持 recording rule 的管理、告警规则支持生效到多集群、仪表盘变量支持 TextBox、告警屏蔽支持更多操作符、更灵活的自定义告警内容。

- Notification Manager 多租户通知管理系统 2.0.0 发布

  Notification Manager 能够从 Kubernetes 接收告警、事件、审计，根据用户设置的模板生成通知消息并推送给用户。 新版本主要功能：新增路由

功能，用户通过设置路由规则，将指定的通知发送给指定的用户；新增静默功能，通过配置静默规则，在特定的时间段屏蔽特定的通知；支持动态更新模板等。

- Nydus 容器镜像加速服务 v2.2.0 发布（CNCF 项目）

  版本特性：启用镜像按需加载功能 erofs over fscache、支持 v6 镜像转换、合并子命令支持多个版本的镜像合并、支持将 Nydus 镜像层转换为 tar 文件、增加 BackendProxy 存储后端用于模拟注册表存储后端。

o

- OCM 多集群管理平台 v0.9 发布（CNCF 项目）

  该版本主要新特性：降低托管集群上的 worker agent 的权限、支持访问 kube-apiserver 及托管集群中的其他服务、支持使用 AddOn API 参考 AddOn 配置。

- OPA 通用策略引擎 v0.50.0 发布（CNCF 项目）

  版本特性：新增内置函数，用于验证 JSON Schema；package 范围内的 schema 注释可以跨模块应用；支持通过一个简单的命令用一个远程捆绑包启动 OPA；引入一个新的 EditTree 数据结构以改善 json.patch 的性能；支持通过状态 API 暴露决策日志错误；所有发布的 OPA 镜像现在都以非 root uid/gid 运行。

- OPA 通用策略引擎 v0.44.0 发布（CNCF 项目）

  该版本主要新特性：修复 3 个 CVE 漏洞、Set Element Addition 优化、内置 Set union 优化、在 OPA 评估命令中添加优化参数、允许将更多的捆绑程序编译到 WASM 中。

- OPA 通用策略引擎 v0.43.0 发布（CNCF 项目）

该版本主要新特性：Rego Object 插入的线性扩展优化、状态和日志插件现在可以接受 HTTP 2xx 状态代码、OPA bundle 命令现在支持 .yml 文件、存储系统修复、Rego 编译器和运行环境 bug 修复和优化。

- OPA 通用策略引擎 v0.41.0 发布（CNCF 项目）

  该版本主要新特性：新增一组内置函数用于验证、解析和查证 GraphQL query 和模式；内置函数声明支持通过元数据指定函数参数和返回值的名称和描述；支持根据 OCI 工件的摘要跳过捆绑重载；捆绑签名中删除空清单；单位解析和 token 更新等。

- OpenClusterManagement（OCM）多集群管理平台 v0.7 发布（CNCF 项目）

  该版本主要新特性：新增“DefaultClusterSet”功能，所有注册进 OCM 环境中的托管集群都会被默认注册进名叫“default”的 ClusterSet 中；支持基于 Taint / Toleration 的语义的多集群调度；部署架构调整为“Hosted 部署”模式，即托管集群内将不需要再部署其他的组件，所有的代理控制器均在远端执行。

- OpenCost 支持 FinOps 开放成本和使用规范（FOCUS）

  FinOps 开放成本和使用规范（FinOps Open Cost and Usage Specification，FOCUS）发布了其规范的第一个版本，以定义云成本、使用和计费数据的开放标准。OpenCost 项目宣布支持这个标准，并已经开始使用这个标准支持初始补丁。

- OpenEBS 云原生存储 v3.4.0 发布（CNCF 项目）

  版本特性：支持通过 OpenEBS helm chart 安装 Mayastor、支持在故障检测时按需切换 Mayastor 节点、支持使用 NDM 的 NVMe 虚拟路径检测、修复 LVM LocalPV helm chart 的拉取镜像密钥错误、在 NFS 服务器部署中添加后端卷 PVC 上下文作为标签。

- OpenEBS 云原生存储 v3.3.0 发布（CNCF 项目）

  该版本主要新特性：弃用 arch-specific 容器镜像、为 LocalPV Hostpath 强制执行带有 ext4 文件系统的主机路径配额、增强 NDM 功能、在 cstor 中添加日志以改善调试能力、增加速率限制器以减少 LocalPV LVM 中的日志泛滥问题。

- OpenFunction 函数即服务项目 v1.0.0 发布（CNCF 项目）

  版本特性：集成 wasmedge、支持从本地源代码构建、单个 Pod 支持多个函数、支持检测源代码或镜像的变化并重建/重新部署新构建的镜像。

- OpenKruise 升级为 CNCF 孵化项目

  OpenKruise 是一个扩展的组件套件，专注于应用程序自动化，如部署、升级、运维和可用性保护等方面。OpenKruise 通过其创新的 CRD 扩展帮助具有大型工作负载的组织采用和自动化 Kubernetes 和云原生部署，为 AI / ML 等领域的新用例打开了大门。

- OpenKruise 云原生应用自动化管理套件 v1.4.0 发布（CNCF 项目）

  版本特性：新增 JobSidecarTerminator 功能，可以在主容器退出后终止 sidecar 容器；新增字段用于立即重新创建容器；在 ImagePullJob 的过程中，支持在 PullImage CRI 接口中附加元数据；sidecarSet 控制器支持命名空间选择器；将 Kubernetes 的镜像地址引用从“k8s.gcr.io”更改为“registry.k8s.io”。

- OpenKruise 云原生应用自动化管理套件 v1.3.0 发布（CNCF 项目）

  该版本主要新特性：支持自定义探针并将结果返回给 Pod yaml、SidecarSet 支持在 kube-system 和 kube-public 命名空间下注入 pod、增加对上游 AdvancedCronJob 的 timezone 支持、WorkloadSpread 支持 StatefulSet。

- OpenKruise 云原生应用自动化管理套件 v1.2.0 发布（CNCF 项目）

该版本主要新特性：新增 CRD PersistentPodState 来持久化 Pod 的某些状态，如“固定 IP 调度”；CloneSet 针对百分比形式 partition 计算逻辑变化；在 lifecycle hook 阶段设置 Pod not-ready；支持保护 scale subresource 的任意自定义工作负载；新增大规模集群的性能优化方法等。

- OSM（Open Service Mesh）开放服务网格项目停止维护

  OSM 采用 Service Mesh Interface （SMI）作为其标准 API，从而允许与支持 SMI 的其他服务网格实现进行兼容，以简化服务网格操作体验。近日，OSM 维护团队宣布 OSM 不再发布新的版本，团队将转向与 Istio 社区共同合作，来推进 Istio 的发展。

- Open Service Mesh 服务网格 v1.2.0 发布（CNCF 项目）

  该版本主要新特性：支持自定义信任域（即证书通用名）、Envoy 更新到 v1.22，并且使用 envoyproxy/envoy-distroless 镜像、增加对 Kubernetes 1.23 和 1.24 的支持、支持限制 TCP 连接和 HTTP 请求的本地 per-instance 速率、修复 Statefulset 和 headless service。

- OpenShift Toolkit 1.0，简化云原生应用的开发

  OpenShift Toolkit 是一套 VS Code 和 IntelliJ 扩展。其功能包括：支持连接和配置 OpenShift；提供混合云支持，开发者可以连接到任何正在运行的 OpenShift 实例；可通过本地工作区、git 仓库或默认的 devfile 模板来开发应用程序；允许采用一键式策略，将仓库代码直接部署到 OpenShift；提供 Kubernetes 资源管理、无缝 Kube 配置上下文切换；多平 s 台支持。

- OpenTelemetry 可观测项目 v1.21.0 发布（CNCF 项目）

  版本特性：添加实验性的直方图建议 API、明确记录日志时使用的参数、将 OpenCensus 兼容性规范标记为稳定。

- OpenTelemetry v1.13.0 发布（CNCF 项目）

  该版本主要新特性：在设置 span 时 Context 是不可变的、支持实验性的配置 OTLP 指标输出器的默认直方图聚合、允许日志处理器修改日志记录、添加实验性的事件和日志 API 规范、在流程语义约定中添加网络指标、为 GraphQL 添加语义约定。

- OpenTelemetry Metrics 发布 RC 版本

  Java、.NET、Python 已经发布了 OpenTelemetry 指标的 RC 版本（JS 下周发布）。这意味着， specification、API、SDK 以及以创作、捕获、处理等方式与 metrics 交互的组件，现在拥有完整的 OTel metrics 功能集，并且随时可供使用。

- OpenTelemetry v1.11.0 发布 （CNCF 项目）

  该版本主要新特性：用更加了然的 bucket 直方图取代直方图、支持在 OpenMetrics 计数器上显示示例、增加数据库连接池指标的语义规范、允许所有 metrics 规范为同步或异步、添加 HTTP/3 等。

- Openyurt 云原生边缘计算项目 v1.3.0 发布（CNCF 项目）

  版本特性：重构 Openyurt 控制平面组件， 新组件 yurt-manager 用于管理多个组件的控制器和 webhook；允许用户为静态 Pod 定义 Pod 模板和升级模型；NodePort Service 支持节点池隔离。

- Openyurt 云原生边缘计算项目 v1.2.0 发布（CNCF 项目）

  新增节点池治理组件 Pool-Coordinator，用于降低云边网络带宽，提高云边网络故障时的边缘自治能力；使用 raven 组件来替代 yurt-tunnel 组件解决跨区域网络通信问题；改进证书管理器。

- Openyurt 云原生边缘计算项目 v1.1.0 发布（CNCF 项目）

该版本主要新特性：支持 DaemonSet 工作负载的 OTA/自动升级模式、支持 e2e 测试的自治功能验证、启用数据过滤功能、增加统一云计算边缘通信方案的建议、改进 health checker。

- Openyurt 云原生边缘计算项目 v1.0.0 发布（CNCF 项目）

  该版本主要新特性：NodePool API 版本升级到 v1beta1、使用 CodeCov 跟踪单元测试覆盖率、新增两份 OpenYurt 组件的性能测试报告。

- OpenKruise 云原生应用自动化管理套件 v1.2.0 发布（CNCF 项目）

  该版本主要新特性：新增 CRD PersistentPodState 来持久化 Pod 的某些状态，如“固定 IP 调度”；CloneSet 针对百分比形式 partition 计算逻辑变化；在 lifecycle hook 阶段设置 Pod not-ready；支持保护 scale subresource 的任意自定义工作负载；新增大规模集群的性能优化方法等。

- osm-edge：易衡科技 Flomesh 开源的边缘服务网格

  osm-edge 采用 osm 作为控制平面，可编程网关 Pipy 作为数据平面。支持 SMI 规范；支持 ingress、gateway API、跨集群服务发现的 fsm，提供“k8s 集群内+多集群”的“东西+南北”流量管理和服务治理能力。 其开发和测试环境采用 k3s、k8e 等，因此用户可以快速低成本地在 x86、arm、RISC-V、龙芯等硬件平台上部署低资源高性能的服务网格。

P

- Paralus Kubernetes 零信任访问服务成为 CNCF 沙箱项目

  Paralus 专为多集群环境设计，凭借即时服务帐户创建和细粒度的用户凭证管理，保证资源访问的安全性。 此外，Paralus 还融入了零信任原则，支持多个身份提供商、自定义角色等。

- Paralus：Rafay Systems 开源的业内首个 Kubernetes 零信任访问服务

Paralus 的主要功能有：能够统一处理多个集群的访问管理、允许与现有的 RBAC 策略和 SSO provider 集成、支持记录组织内用户执行的每一条 kubectl 命令、支持 OIDC、支持创建具有特定权限的自定义角色、允许动态撤销权限等。

- Phlare：Grafana 开源的大规模持续性能分析数据库

  Phlare 是一个水平可扩展、高可用、多租户的持续分析数据聚合系统，与 Grafana 完全集成，可以与指标、日志和追踪等观测指标相关联。安装只需一个二进制文件，无需其他依赖项。 Phlare 使用对象存储进行长期数据存储，并兼容多种对象存储实现。其原生多租户和隔离功能集允许多个独立团队或业务部门运行一个数据库。

- PipeCD 成为 CNCF 沙箱项目

  PipeCD 为多云应用提供了一个统一的持续交付解决方案，为任何应用程序提供一致的部署和操作体验.PipeCD 是一个 GitOps 工具，能够通过 Git 上的 pr 请求进行部署操作。

- Pisanix：SphereEx 开源面向 Database Mesh 的解决方案

  Pisanix 帮助用户轻松实现基于 Database Mesh 框架下 SQL 感知的流量治理、面向运行时的资源可编程、数据库可靠性工程等能力，助力用户云上数据库治理。Pisanix 目前已支持 MySQL 协议，主要包括 Pisa-Controller、Pisa-Proxy 和 Pisa-Daemon（即将推出）三个组件。

- Pixie Kubernetes 观测平台 发布 Plugin System

  Pixie Plugin System 允许用户将其 Pixie 数据导出到任何支持 OpenTelemetry 的服务中。这意味着用户可以利用外部数据存储长期保留数据，在现有工作流和仪表盘中无缝采用 Pixie，将 Pixie 数据与其他数据流结合。

- Podman 容器运行时项目 v4.5.0 发布

版本特性：支持对在 Pod 内运行的容器进行自动更新、支持使用 SQLite 数据库作为后端、增加对容器网络堆栈 Netavark 插件的支持、支持带有 Macvlan 和 Netavark 后台的 DHCP。

- Podman 容器引擎 v4.3.0 发布

  命令更新：支持改变容器的资源限制、删除 K8s YAML 创建的 pod 和容器、支持 K8s secret、支持从 URL 读取 YAML、支持 emptyDir 卷类型、支持 ConfigMap 中的二进制数据；支持重复的卷挂载。

- Podman 容器运行时项目 v4.2.0 发布

  该版本主要新特性：支持 GitLab Runner、新增命令用于创建现有 pod 的副本、新增命令用于同步数据库和任何卷插件之间的状态变化、pod 新增退出策略、自动清理未使用的缓存 Podman 虚拟机镜像、允许不同容器的多个 overlay 卷重复使用同一个 workdir 或 upperdir。

- Prometheus v2.46.0 发布（CNCF 项目）

  版本特性：添加 PromQL 格式和标签匹配器设置/删除命令、推送指标命令、为 K8s 服务发现的 Endpointslice 和 Endpoints Role 添加了更多标签、PodIP 的状态未设置时不会将 Pod 添加到目标组、支持验证远程写入处理程序中的指标名称和标签、支持将原生直方图存储在快照中。

- Prometheus v2.45.0 发布（CNCF 项目）

  更新内容：支持限制每个 TSDB 统计项的返回数量、在全局配置中增加限制、支持同时摄入 classic 直方图和 native 直方图、native 直方图支持限制存储桶的大小。

- Prometheus v2.44.0 发布（CNCF 项目）

  版本特性：每次发送的默认样本数提高到 2,000、支持处理原生直方图数据、在命令行中添加用于检查 Prometheus 服务器健康状态和可用性的功能、添加所有查询加载的样本总数指标。

- Prometheus v2.43.0 发布（CNCF 项目）

  版本特性：在一个单独的字符串中存储所有的标签和对应的值，以减小内存占用（只在使用 Go tag stringlabels 进行编译时启用）；在查询命令中提供 HTTP 客户端配置；添加从不同文件中导入抓取配置的选项；新增两个 HTTP 客户端配置参数；允许通过 API 动态设置查询的回溯时间。

- Prometheus v2.42.0 发布（CNCF 项目）

  该版本特性：支持原生直方图、更改原生直方图的 WAL 记录格式、支持选择 TSDB dump 时间序、支持 Float 直方图、为 Kubernetes 的 pod 对象添加容器 ID 作为元标签。

- Prometheus v2.40.0 发布（CNCF 项目）

  该版本主要新特性：增加对原生直方图的实验性支持、Kubernetes 发现客户端支持使用 protobuf 编码、改善排序速度、增加企业管理分区。

- Prometheus v2.39.0 发布（CNCF 项目）

  该版本主要新特性：增加对摄取无序样本的实验性支持、优化内存资源用量。

- Prometheus v2.37.0 发布（CNCF 项目）

  该版本主要新特性：允许 Kubernetes 服务发现从端点角色向目标添加节点标签、TSDB 内存优化、读取 WAL 时减少睡眠时间、优化签名创建、添加超时和 User-Agent header。

- Prometheus v2.37.0 发布（CNCF 项目）

  该版本主要新特性：允许 Kubernetes 服务发现从端点角色向目标添加节点标签、TSDB 内存优化、读取 WAL 时减少睡眠时间、优化签名创建、添加超时和 User-Agent header。

- Prometheus v2.36.0 发布（CNCF 项目）

该版本主要新特性：集成 Vultr、添加 Linode SD 故障计数指标和 prometheus_ready 指标、在模板 function 中添加 stripDomain、promtool 支持在查询标签值时使用匹配器、增加代理模式标识符。

R

- Rainbond 信创版本的云原生多云应用管理平台发布

  Rainbond 信创版支持部署和管理多个单架构集群。支持 amd64 集群、arm64 集群、amd64 & arm64 混合架构集群的一键部署和管理。兼容主流国产化 CPU，同时与多个国产化操作系统进行了适配。支持传统应用快速迁移到信创环境。云原生应用商店支持发布和安装 arm64 架构的应用。

- Rainbond 云原生多云应用管理平台 v5.14.0 发布

  版本特性：升级各语言源码构建包版本；支持一键删除应用及应用下相关资源；使用集群命令行创建的 Pod 有合理的回收机制；域名配置 https 证书时，增加搜索功能或优先匹配与域名相同的证书；支持配置日志存储路径。

- Rainbond 云原生多云应用管理平台 v5.12.0 发布

  版本特性：支持平台级插件和能力扩展、新增流水线插件、支持通过 OpenAPI 创建组件、优化 Helm 仓库安装应用逻辑。

- Rainbond 云原生多云应用管理平台 v5.9.0 发布

  该版本主要新特性：支持 Containerd 容器运行时、支持使用 grctl 命令更改 cluster API 地址、支持 K8s 1.24 和 1.25、支持 MiniKube 部署、支持自定义监控规则。

- Rainbond 云原生多云应用管理平台 v5.8.0 发布

  该版本主要新特性：支持一键导入 Kubernetes 集群中已存在的应用、支持直接通过 Jar、War 包或 Yaml 文件部署组件、支持创建

Job/CronJob 类型任务、支持扩展应用和组件支持的属性、支持 Dockerfile 使用私有镜像构建。

- Rook 云原生存储项目 v1.12.0 发布（CNCF 项目）

  版本特性：支持 Kubernetes v1.22+、默认使用 Ceph CSI v3.9、支持 Ceph Reef（v18）、增加用于配置存储桶的 Ceph COSI 驱动程序（实验性）、节点丢失后自动恢复 RBD (RWO) 卷、引入 Multus 网络验证工具、通过放弃容器特权来改进安全性、支持 RGW 作为 CephNFS 后端（实验性）。

- Rook 云原生存储项目 v1.11.0 发布（CNCF 项目）

  版本特性：支持 K8s v1.21 及以上版本、Ceph-CSI v3.8 成为默认部署的版本、移除对机器中断预算（MachineDisruptionBudgets）的支持、对象存储桶的通知和主题功能升至稳定状态、支持在具有重叠 CIDR 的多个集群间进行数据镜像、Ceph exporter 成为 Ceph 性能计数器（performance counter）的指标来源。

S

- Sealer 分布式应用交付工具 v0.9.0 发布（CNCF 项目）

  该版本主要新特性：支持通过 Clusterfile 配置标签、权限、角色、注册表、集群主机别名；支持 ipv4/ipv6 双栈；支持本地注册表的高可用模式；支持基于 buildah 的 OCI 标准；Kubefile 支持 Helm 包、k8s yaml 文件、shell 脚本等应用类型。

- Serverless-cd：Serverless Devs 发布基于 Serverless 架构的轻量级 CI/CD 框架——

  Serverless Devs 是业内首个支持主流 Serverless 服务/框架的云原生全生命周期管理的平台。Serverless-cd 基于 Serverless Devs 打造，完全遵循 Serverless 架构最佳实践，在规范和生态层面参考 Github Action

的实现。其采用 Master Worker 模型和事件驱动的架构，可用于快速构建企业内部应用管理 PaaS 平台。

- Serverless Devs 进入 CNCF 沙箱，成首个入选的 Serverless 工具项目

  Serverless Devs 由阿里云开源，致力于为开发者提供强大的工具链体系。借此，开发者不仅可以一键体验多云 Serverless 产品，极速部署 Serverless 项目，还可以在 Serverless 应用全生命周期进行项目管理，与其他工具/平台集成，提升研发运维效能。

- Skywalking 应用性能监控系统 v9.5.0 发布

  版本特性：新的拓扑结构布局图、支持对 Elasticsearch 服务器监控、支持持续分析功能、支持收集进程级相关指标、支持跨线程跟踪分析、支持监控 k8s StatefulSet 和 DaemonSet 的总数量指标、支持 Redis 和 RabbitMQ 监控。

- Skywalking 应用性能监控系统 v9.3.0 发布

  该版本主要新特性：增加指标关联功能、可使用 Sharding MySQL 作为数据库、可视化虚拟缓存和消息队列的性能、使用 Skywalking OpenTelemetry 接收器代替 prometheus-fetcher 插件来抓取 Prometheus 指标。

- Skywalking 应用性能监控系统 v9.2.0 发布

  该版本主要新特性：新增 K8s Pod 的 eBPF 网络分析功能、支持 MySQL 和 PostgreSQL 监控、关联事件组件和追踪、日志组件。

- Skywalking 应用性能监控系统 v9.1.0 发布（CNCF 项目）

  该版本主要新特性：更新 eBPF Profiling 任务到服务级别；为事件添加层字段，禁止报告没有层的事件；Zipkin 接收器机制改变，追踪不再流向 OAP 段；SQL 数据库相关更新；移除 InfluxDB 1.x 和 Apache IoTDB 0.X 的存储方案，添加 BanyanDB 存储方案（尚在开发中）。

- SPIFFE 与 SPIRE 项目正式成为 CNCF 毕业项目

  SPIFFE 为现代生产环境中的各种工作负载提供安全身份，消除了机密信息的共享需求，有望成为更高级别的平台中立安全控制方案的实现基础。SPIRE（SPIFFE 运行时环境）则负责在各种平台上实现 SPIFFE 规范，并强制发布身份的多因素证明。

- Spring Cloud Tencent：腾讯开源一站式微服务解决方案

  Spring Cloud Tencent 依托腾讯开源的一站式微服务解决方案 北极星（Polaris），主要提供微服务领域常见的服务注册与发现、配置中心、服务路由、限流熔断以及元数据链路透传能力。

- Strimzi 消息中间件 Kafka 的 K8s operator v0.31.0 发布（CNCF 项目）

  该版本主要新特性：支持 Kafka 3.2.1；可插拔的 Pod 安全配置文件，内置支持受限的 Kubernetes 安全配置文件；支持 leader 选举和运行多个 operator 副本；支持在 Strimzi 签发的证书中使用 IPv6 地址。

- SuperEdge 边缘容器管理系统 v0.8.0 发布（CNCF 项目）

  该版本主要新特性：将 edgeadm 从 SuperEdge 项目中分离出来、tunnel 支持 http_proxy 能力、Lite-apiserver 支持在边缘节点缓存一些关键资源（节点、service 等）和 ExternalName Service 转发。

- Sylva 云原生基础设施堆栈由 Linux 基金会欧洲分部推出，为电信服务奠定云原生的基础

  Sylva 利用容器网络功能（CNF）和 Kubernetes 等云原生平台来创建电信云技术栈，以减少电信和边缘服务的云基础设施的分散性。 Sylva 提供的技术栈由 5 个部分组成：网络性能（SRIOV、DPDK、指定 CNI 等）、分布式云（多集群 Kubernetes、裸机自动化）、能源效率、安全性（加固和合规性）以及开放性和标准化 API。

T

- TDengine 云原生时序数据库 v3.0 发布

  该版本主要新特性：支持 10 亿个设备采集的数据、100 个节点；支持存储与计算分离，引入计算节点，并重构了整个计算引擎；优化了对消息队列、流式计算和缓存的支持，引入事件驱动的流式计算；支持容器和 Kubernetes 部署。

- Tekton 云原生 CI/CD 框架 v0.50.0 发布

  版本特性：CSI 和计划卷 workspace 功能升级为稳定状态、单独 workspace 功能升级为 beta、支持还原 PVC 创建、添加事件配置映射、引入联合调度（coschedule）功能开关、支持在 PipelineRun 完成后自动删除由其卷声明模板创建的 PVC。

- Tekton：云原生 CI/CD 框架 v0.43.0 发布

  该版本主要新特性：支持解析 sidecar 日志以提取结果到任务运行的 CRD 中、默认启用 pipeline 中的自定义任务、增加对 PipelineRun 调节器的支持、允许用户为可信资源配置公钥、PodTemplate 可以用来更新全局的环境变量。

- Tekton 云原生 CI/CD 框架 v0.42.0 发布

  该版本主要新特性：支持配置 Webhook 端口号、支持为集群资源设置源值、新增一个和状态治理字段有关的 feature flag、支持记录远程资源的来源、在 reconciler 中添加验证功能。

- Tekton 云原生 CI/CD 框架 v0.40.0 发布

  该版本主要新特性：添加任务运行模板、支持在 pipelinerun 中传播 workspace 以简化规范编写、优化 git 远程解析支持、添加集群远程解析器、合并 podtempalte 和 affinity-assistant 的亲和力参数。

- Telepresence K8s 本地开发工具 v2.14.0 发布（CNCF 项目）

版本特性：DNS 配置新增排除字段和映射字段、支持排除环境变量、支持检测和报告客户端计算机上运行的其他 VPN 软件之间的路由冲突。

- Telepresence K8s 本地开发工具 v2.10.0 发布（CNCF 项目）

  该版本主要新特性：流量管理器支持被团队模式和单用户模式、在 Helm Chart 中添加拉取镜像的 secret、OSS Helm chart 将被推送到 telepresence 专有仓库（原先为 datawire Helm 仓库）。

- Tetragon 安全可观测和运行时增强平台 v0.10.0 发布

  版本特性：支持在 configmap 更改时重新创建 daemonset Pod、添加用于跟踪策略指标的事件、Pod 标签过滤器支持跟踪策略、支持加载 Linux 安全模块 LSM 和跟踪程序、 启用 bpf 对象的并行构建、为事件添加速率限制、添加套接字跟踪、为文件监控添加策略。

- Tetragon 安全可观测和运行时增强平台 v0.9.0 发布

  版本特性：增加加载 BPF 程序的日志功能、支持使用 cosign 进行容器镜像签名、支持基本的 Cgroups 跟踪功能、支持 pprof http、gRPC 支持使用 unix socket。

- Tetragon：Isovalent 开源基于 eBPF 的安全可观测性和运行时增强平台

  Tetragon 提供了基于 eBPF 的完全透明的安全可观测性能力以及实时的运行时增强能力。 Tetragon 基于 eBPF 的内核级收集器中直接内置了智能内核过滤能力和聚合逻辑，因此无需更改应用程序即可以非常低的开销实现深度的可观测性。 内嵌的运行时执行层不仅能够在系统调用层面进行访问控制， 而且能够检测到特权、Capabilities 和命名空间的提权逃逸，并实时自动阻止受影响进程的进一步执行。

- ThreatMapper 云原生安全观测平台 v1.4.0 发布

该版本主要新特性：新功能 ThreatGraph 能够结合网络等运行时环境来确定威胁扫描结果的优先级、支持对云资产进行无代理的云安全态势管理、集成云原生环境恶意程序扫描工具 YaraHunter。

- Trivy 容器漏洞扫描工具 v0.44.0 发布（CNCF 项目）

  版本特性：支持扫描本地仓库、将集成测试超时增加至 15 分钟、支持展示漏洞状态、支持策略捆绑的自定义 URL、支持为 Rego 策略自定义数据。

- Trivy 容器漏洞扫描工具 v0.41.0 发布（CNCF 项目）

  版本特性：支持使用 Vulnerability Exploitability Exchange（VEX）来过滤已检测到的漏洞、支持生成虚拟机镜像的 SBOM、支持嵌套 JAR 路径、支持使用自定义的 Docker socket。

- Trivy 容器漏洞扫描工具 v0.39.0 发布（CNCF 项目）

  版本特性：OCI 工件下载支持授权功能、支持在 OCI referrer 中发现 SBOM、支持 k8s 并行资源扫描、增加并发处理的 pipeline、增加节点容忍选项。

- Trivy 容器漏洞扫描工具 v0.35.0 发布（CNCF 项目）

  该版本主要新特性：为虚拟机镜像新增扫描器、多架构镜像支持 OS 通配符、支持按 digest 扫描镜像、增加慢速模式以降低 CPU 和内存利用率。

- Trivy 容器漏洞扫描工具 v0.33.0 发布（CNCF 项目）

  该版本主要新特性：禁用非 amd64 架构的 containerd 集成测试、重构 k8s 自定义报告、支持 non-packaged 二进制文件、修复 golang x/text 漏洞。

- Trivy 容器漏洞扫描工具 v0.32.0 发布（CNCF 项目）

该版本主要新特性：增加对 SPDX 的扫描、支持 Rust 二进制文件的依赖扫描、增加对 gradle.lockfile 和 conan.lock 文件的支持。

- Trivy 容器漏洞扫描工具 v0.31.0 发布（CNCF 项目）

  该版本主要新特性：支持扫描 SBOM 证明、为 dockerfile 的 misconf 处理程序添加测试、增加对 Cosign 漏洞证明的支持、允许用户为 token 定义一个已有的 secret、增加两个新漏洞。

- Trivy 容器漏洞扫描工具 v0.30.0 发布（CNCF 项目）

  该版本主要新特性：支持扫描许可证、推送 canary 构建镜像到注册表、Trivy k8s 支持扫描 single argument 资源、支持扫描 Cyclonedx 软件物料清单（SBOM）、增加 pnpm 支持。

- Trivy 容器漏洞扫描工具 v0.29.0 发布（CNCF 项目）

  该版本主要新特性：支持 RBAC 扫描、支持 K8s secret 扫描、增加对 WASM 模块的支持、增加对 containerd 的支持、支持显示慢速扫描建议、为 pom 文件中的变量评估增加了循环检查、新增 go mod tidy 检查。

- Trivy 容器漏洞扫描工具现在可以扫描几乎所有与云原生相关的东西（CNCF 项目）

  Aqua 称，Trivy 是当前最全面的漏洞和错误配置扫描工具， 可用于扫描云原生应用程序和基础设施，如源代码、存储库、镜像、工件注册表、基础设施即代码（IaC）模板和 Kubernetes 环境等。

V

- Vcluster 虚拟 Kubernetes 集群实现方案 v0.15.0 发布

  版本特性：允许用户在 vcluster 内访问所有主机集群服务、新增内置指标服务器、通用同步功能支持从主机集群导入集群范围资源。

- Vcluster 虚拟 Kubernetes 集群实现方案 v0.13.0 发布

  该版本主要新特性：增加日志和备份功能、增加具有外部数据存储的 k3s 的高可用性支持、vcluster 调度器开启时自动同步 CSI 资源。

- Velero 备份容灾工具 v1.11.0 发布（CNCF 项目）

  版本特性：增加插件进度监控功能、支持过滤备份时要跳过的卷、新增集群范围和命名空间范围的资源过滤器、增加用于设置 Velero 服务器与 k8s API 服务器的超时连接的参数、支持 JSON 格式的备份描述命令输出、用 controller-runtime 重构控制器、CSI 插件会通过检查 restorePVs 参数的设置来决定是否从快照中恢复数据。

- Velero 备份容灾工具 v1.10.0 发布（CNCF 项目）

  该版本主要新特性：引入统一存储库架构、集成跨平台备份工具 Kopia、重构文件系统备份、使用 Kubebuilder v3 对控制器进行重构、允许为卷快照位置添加凭证、增强 CSI 快照的稳健性、支持暂停备份计划、重命名部分模块和参数。

- Velero 备份容灾工具 v1.9.0 发布（CNCF 项目）

  该版本主要新特性：CSI 支持改进、使用 Kubebuilder v3 对控制器进行了重构、支持恢复所选资源的状态、支持在资源恢复过程中更新现有资源。

- Virtink：由 SmartX 开源的轻量 Kubernetes 原生虚拟化管理引擎

  不同于 KubeVirt，Virtink 并不支持遗留硬件设备的模拟以及桌面应用场景能力，而是使用 Cloud Hypervisor 作为底层虚拟化管理器，只支持现代化的云负载，可以在任何虚拟化 CPU 平台的 Kubernetes 中进行安装，以更安全轻量的方式支撑虚拟化负载。

- Vitess 云原生数据库系统 v14.0.0 发布（CNCF 项目）

该版本主要新特性：正式支持 online DDL、Gen4 成为默认的 planner、新增集群管理 API 和 UI——VTAdmin（Beta）、新增一个作为 Vitess 组件运行的 Orchestrator 分支——VTOrc（Beta）、支持跨多个 shard 和 keyspace 的聚合查询。

- Volcano 云原生批量计算项目 v1.8.0 发布（CNCF 项目）

  版本特性：添加 JobFlow 支持轻量级工作流编排、支持 vGPU 调度和隔离、支持 GPU 和用户定义资源的抢占能力、支持将 ElasticSearch 监控系统用于节点负载感知的调度和重新调度、添加 Kubernetes 默认调度器插件的启用和禁用开关、提供设备插件异常容错机制、为 Volcano 添加 Helm Chart。

- Volcano 云原生批量计算项目 v1.7.0 发布（CNCF 项目）

  该版本主要新特性：增加 Pytorch job 插件、支持为分布式高性能 AI 计算框架 Ray 提供批量调度、丰富调度策略以支持更多长期运行服务的应用场景、支持 Kubernetes v1.25、支持多架构镜像、支持实时查看队列的资源分配信息。

- Volcano 云原生批量计算项目 v1.6.0 发布（CNCF 项目）

  该版本主要新特性：支持基于真实节点负载的动态调度和重调度、支持弹性作业调度、新增 MPI job 插件、允许任务失败时不重试、支持查看 pod 请求的开销、支持在 pod group 入列过程中考虑资源配额、默认特权容器通过 admission webhook 的验证。

W

- WasmEdge WebAssembly 运行时 v0.13.0 发布（CNCF 项目）

  更新内容：统一 wasmedge CLI 工具、将 WasmEdge 扩展移植为插件、引入 wasi_logging 插件、编译时支持启用未定义行为检测器、引入了将数据包含到模块实例的新 API、支持异步调用 WASM 函数、引入统一的工具 API。

- WasmEdge WebAssembly 运行时 v0.12.0 发布（CNCF 项目）

  版本特性：引入了插件上下文和相关的 API、引入了多个 WASI 套接字 API 实现、增加 VM API、增加 wasm_bpf 插件。

- Wazero：Tetrate 开源的 Go 语言开发的 WebAssembly 运行时

  Wazero 可以让开发者用不同的编程语言编写代码，并在安全的沙箱环境中运行。Wazero 的特点包括：纯 Go，无依赖，支持跨平台和跨架构；遵循 WebAssembly 核心规范 1.0 和 2.0；支持 Go 的特性，如并发安全和上下文传递；提供了丰富的编程接口和命令行工具。

- werf CI/CD 的 CLI 工具 成为 CNCF 沙箱项目

  werf 可以与用户现有的任何 CI/CD 系统融合，为 Kubernetes 提供了应用全生命周期管理。 werf 依赖 Buildah 来构建容器镜像，兼容各种容器注册中心来管理镜像，使用 Helm chart 将应用程序部署到 Kubernetes。此外，其还支持自动构建缓存、基于内容的标签、Helm 中增强的资源跟踪功能、智能清理旧的/未使用的容器镜像等功能。

- Wolfi：Chainguard 发布首个保障软件供应链安全的 Linux 发行版，专为容器和云原生环境设计

  Wolfi 是一个为云原生设计的精简版 Linux 发行版，但其没有 Linux 内核，而是依靠环境（如容器运行时）来提供内核。主要功能：为所有软件包提供高质量构建时的 SBOM 作为标准；软件包是细粒度和相互独立的，以支持轻量级镜像；使用成熟可靠的 apk 包格式、完全声明性的、可重复的构建系统、支持 glibc 和 musl。

X, Z

- Xline 跨云元数据 KV 存储 进入 CNCF 沙箱项目

  Xline 是一个开源的分布式的 KV 存储，用来管理少量的关键性数据，并在跨云跨数据中心的场景下仍然保证高性能和数据强一致性。Xline 兼容

etcd 接口，提供 KV 接口，多版本并发控制，同时与 K8S 兼容，让用户使用和迁移更加流畅。

- Xline 由 DatenLord 开源：实现跨数据中心数据一致性管理

  Xline 旨在解决 etcd 无法完全满足跨云跨数据中心场景需求的问题。Xline 是一个分布式的 KV 存储，用来管理少量的关键性数据，并在跨云跨数据中心的场景下仍然保证高性能和数据强一致性。其兼容 etcd 接口，让用户使用和迁移更加流畅。

- Zadig 云原生持续交付工具 v1.12.0 发布

  该版本主要新特性：支持代码扫描、支持服务关联多个构建、K8s YAML 项目支持从现有 K8s 导入服务、支持从 Gitee 代码库中同步服务配置、支持服务配置变更后自动更新环境、支持全局构建模板、K8s Helm Chart 环境支持自测模式、支持集成多个 Jenkins 等。

商业产品

本页按字母顺序列出一些知名商业化产品的更新动态。

A

- Aqua 云原生安全平台引入基于 eBPF 技术的零日漏洞检测工具 Lightning Enforcer

  Aqua Lightning Enforcer 利用漂移和基于行为的自动化检测方法，检测零日漏洞等未知威胁，并提供事件管理、可疑行为告警以及开箱即用的 CNDR（云本地检测和响应）。 此外，eBPF 技术的应用避免了传统代理对工作负载的影响，降低了系统的资源开销。

- Aqua 推出首个端到端软件供应链安全解决方案，保护云原生应用供应链的安全

  该解决方案是 Aqua 云原生应用保护平台的一部分，提供的功能包括：代码扫描、CI/CD 态势管理、流水线安全管理、优化 SBOM 功能、评估开源代码的健康状况和口碑情况。

- ARMO Platform Kubernetes 安全平台 推出基于 eBPF 技术的漏洞分析和评估功能

  该功能可以针对 Kubernetes 平台上存在的漏洞进行分析和评估，识别和确定漏洞的相关性和优先级，为用户提供有针对性的修复建议。其中，相关性和优先级允许用户降低未使用的软件包和组件的漏洞的优先级，从而专注于解决对集群构成更大威胁的漏洞。

- AtomicJar 发布集成测试工具 Testcontainers Cloud

  Testcontainers Cloud 基于轻量级开源测试框架 Testcontainers 构建，通过 Docker 容器创建更真实的测试环境， 开发人员可以在将代码通过持

续集成（CI）平台转移到生产环境之前自己测试应用程序，这使得云原生应用开发者无需专门应用测试团队的帮助即可测试依赖关系。

- AWS 故障注入模拟器（FIS）支持在 EKS Pod 上开展混沌工程实验

  AWS FIS 是用于运行混沌工程实验，以测试和验证应用程序的弹性。FIS 新增了 7 个针对 EKS Pod 的故障注入动作。你可以对目标 Pod 和容器施加 CPU、内存或 I/O 压力，从而评估应用程序在负载情况下的性能。你也可以注入多种网络故障，包括增加网络流量延迟、丢弃全部或部分网络数据包。还能通过终止 Pod，以评估应用程序的弹性。

- AWS 容器应用监控服务 Amazon Managed Service for Prometheus 集成成本管理工具 Kubecost

  Amazon Managed Service for Prometheus with Kubecost 能够执行成本分配计算，提供跨多个 EKS 集群的统一 Kubernetes 成本视图，并为优化 Kubernetes 集群提供建议。

- AWS 应用配置工具 AWS AppConfig 推出一个用于容器运行时的代理，简化 feature flag 和运行时的配置

  AKS 等平台的用户可以使用 AWS AppConfig Agent 管理容器应用配置数据的检索和缓存，以及处理轮询和缓存逻辑等。 feature flag 和运行配置功能允许用户无需部署新代码即可发布新的功能。 此外，还具备配置安全功能，包括验证器、警报提醒，自动回滚配置等。

- AWS 推出本地集群，用于在混合云服务 Outposts 上本地运行 EKS 集群

  本地集群减少了因云端网络中断而导致的应用停机风险。在此期间，应用程序仍然可用，并且可以执行集群操作。当连接到云时，EKS 管理 Outposts 上 K8s 控制平面的可用性。本地集群运行的 Kubernetes 与云上 EKS 功能相同，支持自动部署最新的安全补丁。

- Azure 推出容器应用程序网关预览版

Azure 容器应用程序网关是一个应用程序负载均衡和动态流量管理产品，适用于在 Kubernetes 集群中运行的工作负载。它是 Azure 应用网关入口控制器（AGIC）的增强版，增强的功能包括：在反映 Kubernetes YAML 配置变化时，实现近乎实时的收敛时间；支持超过 1400 个后端 Pod 和 100 个监听；支持蓝绿部署策略和主动/主动或主动/被动路由。

- Azure 发布 AKS 的网络可观测性附加组件

  AKS 的网络可观测性附加组件是一个分布式监控解决方案，适用于 Linux 和 Windows 主机环境。这个组件通过收集实时数据，利用 Linux 的 eBPF、虚拟过滤平台和 Windows 中的主机网络服务来深入了解网络基础设施，并将这些数据提供给 Prometheus 和 Grafana 使用。

- Azure Kubernetes 发行版 AKS 5 月更新

  更新内容：支持自动化部署和集成 GitHub Actions；新增 AKS Fleet Manager 以及引入更新组、阶段和运行方式等功能帮助开发者有效管理多个 AKS 集群；支持 Windows 节点配置和 Gen 2 VM；支持专为容器工作负载设计的操作系统 Azure Linux；推出托管的 Prometheus 和 Grafana。

- Azure Container Storage 的公共预览版发布

  Azure Container Storage 是一种为容器构建的卷管理服务，提供了跨不同存储产品的一致管理体验，包括托管选项（由 Azure Elastic SAN 支持）、Azure 磁盘和容器服务上的临时磁盘。用户可以为生产规模的有状态应用轻松创建和管理块存储卷，并在 Kubernetes 上运行。

- Azure AKS 正式推出网络方案 Azure CNI Overlay

  Azure CNI Overlay 有助于简化 Azure 虚拟网络（VNet）子网中的集群节点和 Pod 管理工作，用户可将节点直接放置在 VNet 子网中，Pod 则可从一个单独的专用 CIDR 处获得 IP 地址。借此，AKS 集群管理、路由的配置以及集群的扩展工作都将得以大幅简化。

- Azure 推出面向 AKS 的 Kubernetes 应用方案

  Azure Kubernetes 应用方案旨在助力合作方在微软 Azure 市场中创建、发布和管理商业商品化的 Kubernetes 解决方案的“工具箱”。 Azure Kubernetes 应用方案支持利用现有 Helm Chart 打包，为 Kubernetes 应用方案创建捆绑包； 仅需 48 小时即可创建和发布方案；自动进行安全扫描；全方位赋能销售增量。

- Azure 混合云多云解决方案 Azure Arc 集成可观测平台 Datadog

  Azure Arc 与 Datadog 的集成提供了以下功能：监控连接状态和代理版本，确保 Azure Arc 启用的服务器、SQL 服务器和 Kubernetes 集群已连接且安全；将混合云和多云服务器、Kubernetes 集群和数据服务集成到一个统一仪表板中，实现跨环境的无缝管理；简化合规性的管理和跟踪，提供丰富的可视化效果和可操作告警。

- Azure Serverless 容器平台 Azure Container Apps 十一月更新

  更新内容：无需 Dockerfile 即从源代码构建容器镜像、可以使用 Azure pipeline 任务从 Azure DevOps 构建和部署容器应用、支持从 GitHub Actions 工作流中构建和部署容器应用、支持按 IP 限制 Azure 容器应用的入站流量。

- Azure Kubernetes 发行版 AKS 11 月更新

  更新内容：支持基于 Azure Arc 在数据中心和边缘节点上运行轻量级 AKS、集成事件路由服务 Event Grid、最大节点限制增加到 5000 个、支持通过 Azure Active Directory 进行工作负载身份管理、推出 Kubernetes 应用市场、新增 CNI 插件 Azure CNI Overlay mode、启用基于 AMD 的机密虚拟机节点池。

- Azure Container Apps (一款 Serverless 容器运行时）更新

更新内容：Dapr 组件支持通过托管身份验证后端服务提供者、支持 Dapr secrets API、集成监控服务 Azure Monitor。

- Azure Kubernetes 发行版 AKS 10 月更新

  更新内容：支持垂直 Pod 自动扩缩容（VPA）、支持下一代 Azure 磁盘存储 Premium SSD v2、新增镜像清理功能、支持开启 IPVS 负载均衡、简化数据库创建和身份管理、允许根据工作负载要求启/禁用 CSI 驱动、新增 Linux 发行版 Mariner 作为容器主机、Azure CNI Powered by Cilium 可用、支持 K8s 1.25、支持 Dapr 1.9。

- Azure Kubernetes 发行版 AKS 9 月更新

  更新内容包括：Azure 容器注册中心新增软删除功能，可恢复误删除的工件；允许中止 AKS 集群或代理池上的任何操作；支持多实例 GPU（MIG）；将 Azure 网络策略管理器（NPM）扩展到 Windows server 2022 的 AKS 集群；集成用于 AKS 私有集群的 API Server VNET。

- Azure 轻量级 Kubernetes 发行版 AKS Edge Essentials 正式可用

  AKS Edge Essentials 是 AKS 的本地 Kubernetes 实现，可以在资源有限的边缘设备上运行。它可以用来部署单节点和多节点的 K3s 和 K8s 集群，可以在现有的 Windows 设备上运行 Linux 和 Windows 工作负载。

- Azure 发布 AKS 舰队管理器（公开预览版）

  Azure Kubernetes 舰队管理器将多个群集地视为单个集群来管理。用户可通过创建一个 Fleet 资源来管理多个集群。用户可将现有 AKS 集群当作成员集群，加入到舰队中，然后用这些集群的元数据，编排多个集群场景，如 Kubernetes 资源传播和多集群负载均衡。

- 阿里云应用实时监控服务 ARMS 7 月产品更新

更新内容：新增对 JVM 系统参数、K8s Yaml 等参数信息的脱敏功能；支持 ZGC 垃圾回收器相关指标展示；应用告警支持 p99 等分位数、慢调用次数等规则指标配置；应用告警规则支持区间检测模式。

- 阿里云容器服务 Serverless 版（ACK Serverless）2.0 发布

  更新内容：对 10+ K8s 核心系统组件实现全托管；支持动态容量规划；支持自动化升级 K8s 版本，包括自定义运维窗口和升级频率策略以及智能化的风险检测等）；支持智能弹性预测 AHPA 升级；新增大数据计算解决方案，基于多可用区轮询 Spot ECI，抢占失败降级到按量实例，使总资源成本下降 20% 以上。

- 阿里云微服务引擎 MSE 6 月产品更新

  注册&配置中心提供迁移可用区功能、网关日志兼容 NginxIngressSLS 格式、供基于 ModSecurity 防护引擎实现的 WAF 插件、提供 OAuth 认证插件、RPC 服务的行为管理支持自定义。

- 阿里云容器服务 ACK 5 月动态

  更新内容：支持 Kubernetes 1.26、支持 Flexvolume 组件和 CSI 组件共存于一个集群内、Serverless 版 ACK 支持成本洞察、ASM 支持集成 KServe、ASM 网关支持 Serverless 形态、ASM 支持全局证书管理。

- 阿里云 Serverless 容器服务 ASK 更新

  更新内容：新增对十多个 K8s 核心组件的托管支持，提供动态容量规划能力；新增智能化风险识别能力，支持自动化升级 K8s 版本；提供基于容器镜像缓存的 AI 大镜像秒级拉取能力，启动时间降低 90%；增强智能弹性预测 AHPA 能力，相比人工配置，弹性准确率提升 80%；同时新增对于 GPU 的支持。

- 阿里云容器服务 ACK 4 月产品更新

更新内容：集群删除支持同步删除日志 project、用于托管高可用控制面的标准 Kubernetes 集群服务的 ACK 灵骏托管版集群正式发布、弹性调度可通过 ElasticResource 支持将 Pod 调度到 Arm 节点。

- 阿里云微服务引擎 MSE 4 月产品更新

  更新内容：Nacos 和服务治理结合，无侵入地实现 Java 应用的注册发现、全链路灰度等功能；云原生网关推出针对容器服务 Nginx Ingress 的迁移工具；微服务治理基础版支持无损上下线、金丝雀发布等功能；Nacos 支持 TLS 链路加密。

- 阿里云服务网格 ASM 2023 年 3 月产品动态

  更新内容：网关支持对接 WAF、支持配置 Ingress 资源、支持 Knative 服务的管理、网格拓扑支持 OIDC 方式登录、Sidecar 代理支持超卖模式、新增出口流量策略、支持配置全局默认的 HTTP 请求重试策略。

- 阿里云 ACK 2 月产品更新

  更新内容：节点池支持 Kubelet 参数自定义功能、AIOps 支持 Service 诊断功能、支持变更云盘类型、支持使用 cloud-auto 类型云盘作为动态存储卷。

- 阿里云服务网格 ASM 2023 年 2 月更新

  更新内容：支持以 Telemetry CRD 的方式定义管理日志、监控及链路追踪；支持网格全局及命名空间级别的细粒度配置；在 150 Pod 规模以上的集群中，网格拓扑加载速度提升；支持配置 sidecar 代理的环境变量、istio-proxy 线程数等；支持在多集群内配置集群内流量保持功能。

- 阿里云容器镜像服务 ACR 正式推出云原生制品中心

  云原生制品中心为容器开发者免费提供来源于阿里云官方、龙蜥社区的安全可信容器基础镜像。 包含应用容器化基础 OS 镜像、基础语言镜像、AI/

大数据相关镜像类别，覆盖 ARM、ARM 64、x64、x86-64 多种系统架构。

- 阿里云微服务引擎 MSE 2022 年 12 月更新

  更新内容：注册配置中心 zookeeper 支持导出快照和事务日志、云原生网关支持 HTTP/3、微服务洞察支持无损上下线过程中的日志采集以及关键节点信息展示。

- 阿里云服务网格 ASM 2022 年 12 月更新

  更新内容：支持托管式的按需推送 xDS 配置、支持自建 Istio 社区版本迁移到 ASM、ASM 网关支持配置自定义授权服务、启用服务等级目标 SLO。

- 阿里云分布式云容器平台 ACK One 更新

  更新内容：集成 ArgoCD 实现多集群的 GitOps 持续交付，优化多集群应用分发；统一配置报警规则，支持配置特定集群的差异化报警规则，支持自动同步报警规则到新关联集群。

- 阿里云容器服务 ACK 云栖大会更新

  更新内容：提供对 eRDMA 高性能容器网络的支持、基于新一代容器网络文件系统 CNFS 2.0 更好支持有状态应用的容器化、内建云原生混部系统 Koordinator 的产品化支持、发布 AIOps 套件和 FinOps 套件。

- 阿里云发布云原生技术中台 CNStack 2.0

  CNStack 2.0 支持不同厂商、不同架构、不同地域和 CPU/GPU 算力混合管理。提供应用开发、测试、运维全生命周期的一站式管理，场景覆盖容器服务、分布式应用、云边、DevOps。此外，平台还提供了完整的技术栈支持，包括内置的、开箱即用的产品组件和中间件以及原厂和合作伙伴提供的产品及组件。

- 阿里云微服务引擎 MSE 10 月更新

  更新内容：注册配置中心提供迁移工具及方案、云原生网关服务来源支持 Serverless 应用引擎、认证鉴权支持多个规则并存、路由及服务详情页面新增 QPS、错误、延迟等指标监控、全链路灰度及标签路由等能力支持 Consul 注册中心。

- 阿里云服务网格 ASM 9 月更新

  更新内容：应用服务治理支持预热功能；支持以试运行模式应用安全策路；优化 sidecar 代理配置；“请求认证”中新增多种 JWT 算法；支持通过外部授权接入 O1DC 协议服务；支持 K8s 1.24；AHPA 弹性预测支持基于 GPU 指标的预测；日志中心支持网络组件的日志和大盘展示；当 Pod 挂载 OSS 存储卷时，支持 I/O 可观测性。

- 阿里云容器服务 ACK 9 月更新

  更新内容：新增托管节点池、支持通过 Annotation 为 Service 配置网络配置型负载均衡 NLB 实例、为工作负载提供资源画像功能、支持使用容器网络文件系统对对象存储 OSS Bucket 进行生命周期管理。

- 阿里云服务网格 ASM 八月更新

  更新内容包括：兼容 Istio 1.14 系列版本、支持 Kubernetes 1.21 – 1.24 版本、增强网格诊断功能、支持第三方令牌在 ACK 集群的兼容、支持用户自定义 EnvoyFilter、无需修改代码即可实现同可用区优先路由。

B

- Backslash Security 推出同名云原生代码可视化工具

  Backslash 推出的可视化工具可以识别云原生代码中可能导致安全漏洞的工作流。 具体功能包括：通过上下文可视化仪表盘，自动发现和映射云原生应用的代码及其依赖关系；根据生产中的应用云状况，自动确定高风险代

码的优先次序；通过智能的自动风险识别，简化漏洞和风险的补救。 该工具目前支持 AWS 环境和 GitHub 代码库。

- BellSoft 推出 Alpaquita Cloud Native Platform，用于高效运行云原生 Java 应用

  Alpaquita Cloud Native Platform 利用适用于 Java 的 Linux 发行版、JVM 优化、多用途框架和原生镜像技术，提供了一个专为在容器中运行的 Java 应用而优化的运行时环境。该平台特性：云成本降低 20%，使用更小的 Pod，所需节点的数量减少；零迁移成本，不需要重构；效率提升 50%，实现低延迟、低成本、快启动。

- 博云容器中间件 BMM v2.1 更新发布

  更新内容：新增 RabbitMQ 及生命周期管理能力、首页新增中间件概览、支持存储扩容可视化、支持版本升级可视化、支持中间件操作审计、支持事件查询与告警。

- 博云容器云产品 BoCloud Container Platform v3.7 发布

  支持 x86 架构和 ARM 架构的混合管理、支持国产化数据库达梦、支持生产级 windows 容器、提供 ACK 和 TKE 公有云容器集群的统一资源纳管服务、支持同时发布多个原生 yaml 文件、支持原生 ingress 能力、将高性能虚拟机作为一种全新的资源类型进行独立管理、提供容器资源超分能力、完成容器云和微服务管理、服务网格、中间件等子产品的深度融合。

C

- Calico 云原生安全平台 Calico Enterprise 3.16 发布

  更新内容：egress 网关支持 Azure 和 AKS、支持通过 Tigera operator 部署 egress 网关、新增 Manager UI 用于启用和配置基于工作负载的 Web 应用防火墙、Kubernetes 工作负载的可视化范围扩展至 100 多个命名空间、允许来自不同命名空间的 pod egress 到不同的外部网络。

- Calico 云原生应用安全平台 Calico Enterprise 3.15 发布

  版本特性：支持将 Calico Enterprise 配置为符合 FIPS 140-2 的模式，FIPS 140-2 规定了应用程序和环境的加密模块需满足的安全标准；提供了一个基于工作负载的 IDS/IPS（入侵检测/入侵防御）系统，根据全球威胁情报源数据库检测并阻止不良工作负载的访问；自动化命名空间的安全策略建议；提供基于命名空间的仪表盘；支持出口网关 pod 的故障检测和故障转移。

- Canonical K8s 发行版 Charmed Kubernetes 1.26 发布

  版本特性：增加 Kube-OVN BGP 支持，实现从外部网络访问 pod IP；支持为 Kube-OVN 配置流量镜像；metallb-controller 和 metallb-speaker 可以部署到更多的架构上（AMD64、ARM、ARM64 等）；集成策略引擎 OPA gatekeeper。

- CAST AI Kubernetes 成本优化&自动化平台 1 月更新

  更新内容：新增 CAST AI kVisor 安全代理，用于评估安全漏洞和和分析集群配置的安全性；改进创建新节点时使用 AKS 镜像的方式；提供每天的 CPU 使用数据以及计费 CPU 数量；改进自动缩放功能。

- CAST AI Kubernetes 成本优化&自动化平台十一月更新

  更新内容：集群再平衡后可以配置最小节点数、为 EKS 用户增加使用存储优化节点进行自动缩放的支持、节点模板支持备选（Fallback）功能、增加对 EKS 和 GKE 集群的 ARM 节点支持。

- Chainguard 云原生供应链安全平台 Chainguard Enforce 产品更新

  更新内容：支持对容器镜像自动生成和摄取 SBOM、支持每日对云原生工作负载进行漏洞扫描并生成报告、新增用于过滤和搜索整个环境中的 SBOM 和漏洞的中央控制台、支持通过私人管理的签名基础架构提供无密钥签名。

- Chainguard 云原生供应链安全平台 Chainguard Enforce 更新

  更新内容：提供基于 Sigstore 的无密钥签名模式、增加一个开箱即用的安全策略库、支持用户通过其 Gitlab 账号登录、支持 CloudEvents、支持细粒度的 Kubernetes 工作负载对象的策略、增强大规模集群用户的稳定性。

- Cisco 发布全栈观测平台 FSO

  FSO 能够将来自应用、网络、基础设施、安全、云、可持续性和业务的数据汇集在一起，提供上下文和预测性的观测能力。FSO 利用 OpenTelemetry 来统一收集指标、事件、日志和跟踪，重点关注云原生应用的可观测性。其提供成本可观测性，提供对 Kubernetes 工作负载资源的优化措施，支持生成基于应用的业务风险评分等。

- CloudCasa 基于 Velero 的企业级 Kubernetes 数据保护方案 CloudCasa for Velero

  CloudCasa for Velero 为企业和服务提供商提供了在所有 Kubernetes 发行版以及混合云和公有云环境中对 Velero 备份进行多集群管理的能力。开发人员和平台工程师可以通过策略驱动的模板和引导式恢复功能，在控制台管理和监控基于 Velero 的备份。 Velero 用户可以订阅 CloudCasa 服务，并将其现有设置编入目录，以便在几分钟内进行集中管理。

D, E

- D2iQ 推出自然语言 AI 助手 DKP AI Navigator，用于简化 Kubernetes 管理

  DKP AI Navigator 是在 D2iQ 的内部知识库中训练出来的，知识库中包含了用户在生产中遇到的最棘手问题的解决方案。D2iQ 用户可以通过该工具，即时获得这些专业知识，并通过单一窗口管理 Kubernetes 集群。DKP AI Navigator 将在 DKP 2.6 中正式可用。

- D2iQ 企业级 K8s 平台 D2iQ Kubernetes Platform v2.5 发布

更新内容：支持将独立的 DKP Essential 集群扩展到 DKP 企业管理集群下进行集中管理、支持通过 Kube-bench 检查集群是否符合 CIS Kubernetes 基准、警报内容现在包括根因信息和解决方案建议、完全支持 Istio、支持 ARM64 机器、支持外部 Load Balancer。

- D2iQ 推出专为政府部门设计的 Kubernetes 平台 DKP Gov

  DKP Gov 基于 D2iQ Kubernetes 平台 (DKP)创建，旨在满足政府、军事和民用机构对创新技术的需求。 DKP Gov 支持物理和逻辑上隔离的节点，支持集中式多云平台、多节点 fleet 管理，符合联邦信息处理标准 (FIPS) 140-2 认证等。

- D2iQ Kubernetes Platform v2.4.0 发布

  该版本主要新特性：支持在本地和物理隔离环境中运行 GPU 节点、支持红帽企业 Linux 系统、Rook Ceph 替代 MinIO 成为平台的默认存储、集成 Trivy 等第三方扫描工具。

- D2iQ Kubernetes Platform v2.3 发布

  该版本主要新特性：支持 Kubernetes 1.23、支持多集群环境下的应用配置和部署、支持 Amazon EKS 和 GCP、支持多可用区、支持为每个集群配置自定义域名和证书、增加自助式故障排除能力。

- DaoCloud Enterprise 5.0 发布

  DaoCloud Enterprise 5.0 是自主开放、高性能、可扩展的新一代云原生操作系统。围绕云原生底座延伸能力场景，打通云边壁垒，跨越多云环境，解决信创基础设施兼容难题，提升应用交付效能，使能应用敏捷创新，丰富的数据服务赋能业务系统，融合微服务治理和全局可观测应对庞大业务系统运维管理挑战。

- Databend 发布云原生数据仓库 Databend Cloud

Databend Cloud 是基于开源云原生数仓项目 Databend 打造的弹性云数仓，由 3 层组成：存储层、计算层和元信息服务。底层基于对象存储，上层计算节点采用不同的规格，最上层是元数据和管理集群，实现多租户的隔离，保证用户的数据安全。

- Datadog 推出云安全平台 Cloud Security Management

  Cloud Security Management 将云安全态势管理、云工作负载安全、警报、事件管理和报告等功能集中在一个平台上，开发和安全团队能够借此识别错误配置、检测威胁并确保云原生应用的安全。

- Docker 发布技术预览版 WebAssembly 工具

  现在，Docker 允许用户使用容器和 Wasm 工件构建云原生应用，将 Wasm 视为 Linux 容器的补充性技术。 此外，Docker Engine 继续使用与整体生态系统相统一的 containerd 容器运行时，只不过把负责容器进程运行的 runC 替换成了 WasmEdge 运行时。

- Docker Desktop v4.12.0 发布

  该版本主要新特性：集成 containerd，使用 containerd 来管理和存储镜像；允许直接通过 Docker 仪表盘在运行的容器中执行命令。

- Dynatrace 一体化性能监控解决方案 Dynatrace Managed v1.270 发布

  更新内容：支持查看代码级漏洞的入口点信息、新增代码迭代过程中的安全事件（包括受影响节点的数量）、支持 init 容器、新增 OTLP 日志摄取端点、增加触发 Kubernetes 警报的条件。

- Dynatrace SaaS v1.250 云监控平台发布

  该版本主要新特性：支持检测 Go 应用中的第三方漏洞、在 Kubernetes 命名空间页中新增一个“Kubernetes services”列、支持通过 service、service name 或 service type 筛选 pod。

- Dynatrace 利用人工智能引擎 Davis AI 提供 Kubernetes 根因分析

  Davis AI 因果分析引擎为 Kubernetes 性能问题及其对业务连续性的影响提供根因分析。具体功能：自动化根因分析、支持实时跟踪 Kubernetes 的编排流程、将性能问题与 Pod 驱逐联系起来、支持根据工作负载部署清单的变化来识别与错误配置有关的性能下降问题的原因。

- Ermetic 云原生应用保护平台支持 Kubernetes 安全态势管理

  Ermetic 云原生应用保护平台的 Kubernetes 安全姿势管理功能支持自动发现和修复 Kubernetes 集群中的错误配置、合规性违规以及风险或过度特权。 它提供了所有 Kubernetes 集群内部资源的详细清单，能够持续执行安全态势评估和风险优先级排序，并提供补救方法和工作流集成。

F

- F5 发布中国版容器 Ingress 服务 CIS-C

  CIS-C 是一款将 Kubernetes 集群内服务通过 F5 BIG-IP 进行自动化发布的控制器软件。 帮助用户打通 Kubernetes 集群与外部入口，将 BIG-IP 应用交付能力集成到 Kubernetes 技术栈的云环境中。 它实现了多团队合作，用户可以灵活地、自动化地创建、变更应用或者服务入口策略。

- Fairwinds 的 Kubernetes 治理和安全平台 Fairwinds Insights 十一月更新

  更新内容：增加工作负载成本分配功能，允许查看一组工作负载的历史成本；优化成本页面，细化 Kubernetes 集群成本的分类；支持查看成本随时间变化的动态。

- Finout 推出无代理 Kubernetes 成本治理套件

  该无代理成本治理套件可以在跨各大云平台使用 Kubernetes 时自动检测和管理云浪费，预测支出并保持预算。其提供统一的 MegaBill 仪表盘，管理跨多个云平台的成本，并提供 Kubernetes 支出的上下文信息；支持设

置 Kubernetes 部署的预算和未来支出；支持识别跨 Kubernetes 和云服务的支出异常；提供成本优化实时建议。

- Fortinet 发布云原生防火墙服务 FortiGate CNF

  FortiGate CNF 是一个 SaaS 产品，通过与 AWS Gateway Load Balancer、AWS Firewall Manager 等服务的深度云原生集成，支持不同 AWS 环境下的一致安全策略，消除了网络安全的复杂性。此外，还支持下一代防火墙的安全检查功能，提供对应用层的深度可视性，可以保护多个账号、子网、虚拟私有云和可用性区域。

G

- GitLab 15.7 发布

  版本特性：支持将私有化部署的 GitLab 实例部署到 Jira 云、支持 SSH 密钥签署提交文件、支持显示每条流水线的多个代码质量扫描报告、允许在个人命名空间内共享 Kubernetes GitLab 代理的 CI/CD 访问权、支持在默认分支以外的地方进行 GitOps 部署。

- GitLab 15.5 发布

  该版本主要新特性：在合并请求流水线中运行安全扫描工具、支持 Kubernetes 容器扫描、支持从 GitHub 导入项目时导入更多的关系、新增预定义 DORA（评估 DevOps 效能水平的指标）可视化比较报告。

- GitLab 15.4 发布

  该版本主要新特性：改进 VSCode CI/CD 集成功能、支持 Gitee 项目导入、专业版支持测试管理、支持 DORA 指标用于评估企业 DevOps 效能、集成 Harbor 镜像仓库。

- Gloo Mesh v2.1 服务网格管理平台发布

该版本主要新特性：增加资源状态和调试页面、优化 Istio 生命周期管理、支持零信任访问策略、东西向流量的 TLS 终端问题修复、支持用同一个 Helm chart 安装代理和管理集群。

- Gloo Gateway 云原生 API 网关支持 GraphQL

  GraphQL for Gloo Gateway 无缝地将 GraphQL API 添加到 Gloo Gateway 中。用户可以借此对 API 进行联合 GraphQL 查询，而不需要为 GraphQL 建立额外的服务器以及解析器和解析器模式库；所有策略和请求都可以在 Envoy Proxy 过滤器中进行管理；同时支持声明式配置，与 CI/CD 和 GitOps 工作流完全兼容。

- Google Cloud GKE 引入 Network Function Optimizer，助力电信运营商采用云原生网络功能

  Network Function Optimizer on GKE 主要提供三个方面的能力： 通过 Kubernetes 原生的多网络实现云网络的现代化，为 Kubernetes 集群定义了一个网络目录，允许 Pod 根据连接性或性能需求，将一个接口附加到一个或多个网络； 基于软件的高性能数据平面加速，云原生网络功能（CNF）可以在任何 Pod 上进行调度，消除网络功能和底层网卡之间的依赖性； 流量定向能力，简化流量分类，支持将一组 Pod 定义为下一跳（nexthop）。

- Google Cloud Kubernetes 发行版 GKE 12 月更新

  更新内容：允许在 Autopilot 集群中运行基于 GPU 的工作负载；支持双栈集群；支持在多 GPU 工作负载上启用传输层插件 NCCL Fast Socket，提高英伟达集体通信库（NCCL）的性能；GKE 网关控制器支持全局外部 HTTP(S) 负载均衡器。

- Google Cloud Kubernetes 发行版 GKE 十一月更新

  更新内容：K8s 控制平面日志正式可用；支持使用弃用 insight 来识别 1.23 及更老版本的集群；Autopilot 集群支持紧凑型放置策略，以及当某

个节点有问题时，支持向 GKE 发出信号；单集群 GKE Gateway 正式可用；创建 LoadBalancer 服务时，谷歌云控制器会自动创建防火墙规则并应用到 GKE 节点上。

- Google 发布云原生文件存储服务 Filestore Multishares for GKE

  Filestore 是一个文件存储托管服务，Filestore 实例是一个网络附属存储（NAS）托管系统。 Filestore Multishares for GKE 为 GKE 实例提供了文件存储服务，其主要功能包括：支持区域性存储，防止区域故障； 一个 Filestore Enterprise 实例上可分配多达 10 个 share，每个 share 映射到 GKE 中唯一的持久卷；支持动态卷配置，可根据需要增加或减少 share 和实例的容量。

- Google 发布 Kubernetes Gateway API 的企业级实现 GKE Gateway Controller

  GKE Gateway Controller GKE 提供了一个可扩展的 API 来管理内外部的 HTTP(S) 负载均衡。 其主要功能包括：提供了一个多路由/租户共享的网关、默认提供两个 GatewayClass 即 Global external 和 Regional Internal 负载均衡、支持大规模的端到端加密、支持定制后端服务属性、提供高级流量管理等。

- Google 服务网格 Anthos Service Mesh 更新

  更新内容包括：支持使用 Fleet Feature API 自动配置 Anthos Service Mesh、支持通过托管数据平面自动升级数据平面代理。

- Google 混合云自动化配置工具 Anthos Config Management 更新

  更新内容包括：配置同步功能可作为 OCI 镜像同步存储在 Google Artifact Registry 或 Container Registry 中、Config Sync 支持从私有 Helm 仓库同步、支持用户提供的 CA 证书用于验证 Git 服务器的 HTTPS 连接、新增约束模板 K8sStorageClass。

- Grafana Cloud 观测平台集成 Cilium Enterprise，加强 Kubernetes 网络监控

  集成后，可以将 Cilium Enterprise 中部署的数据输送到 Grafana Cloud。 Grafana Cloud 提供四个预制的仪表盘：Cilium 总览、Hubble 总览、Cilium operator 和 Cilium Agent。还提供了 17 条告警规则，用于监测与 Cilium Agent 和 Kubernetes 集群状态相关的 Cilium 核心组件。

- Grafana Cloud 推出业内首个性能测试和分布式追踪的集成

  Grafana Cloud 引入负载测试项目 k6 和分布式追踪后端 Tempo（k6 x Tempo）的集成， 缩小性能测试的黑盒数据和系统内部白盒数据之间的差距。 该集成允许关联 k6 的测试运行数据和服务器端的追踪数据，从而进行根本原因分析；汇总收集的追踪数据以生成实时指标，帮助用户缩小搜索空间并快速发现异常行为。

- Gremlin 混沌工程平台 Gremlin 7 月更新

  更新内容：支持自动检测 Kubernetes 环境中的高优先级可靠性问题、推出 Failure Flags 的 Beta 版本（用于在 AWS Lambda 函数、无服务器工作负载和容器上运行混沌实验）、允许用户通过简单的注释将 Kubernetes Service 自动注册到 Gremlin 中、添加了同时管理多个服务的 Web 应用的支持。

H, I

- Harness 软件交付平台 Harness Platform 7 月产品更新

  更新内容：支持在运行流水线时灵活切换分支、支持 HPA、支持针对蓝绿部署和金丝雀发布执行策略的 Pod 中断预算。

- Harness 推出全托管 GitOps-as-a-Service

Harness GitOps-as-a-Service 基于 Argo CD 构建，集成 OPA，支持审计跟踪，提供细粒度的 RBAC，支持不同部署的统一单点登录，支持声明式设置，提供集中的 GitOps 控制中心用于管理和查看多个环境中的部署情况，支持与已有的 Argo CD 实例集成。

- 华为云云容器引擎 CCE Turbo 2023 上半年产品更新

  更新内容：支持 Kubernetes 1.25 版本；新增网络优先级控制、CPU 多核负载均衡隔离、CPU burst 能力；提供一键式原地无损自动升级；支持多容器共享 GPU；支持同节点池多系列、规格实例混合管理；支持节点 OS 批量自动更新升级。

- 华为云发布分布式云原生产品——容器 CCE Turbo 和分布式云原生 UCS (Ubiquitous Cloud Native Service)

  CCE Turbo 可通过计算、网络、调度全方位加速，实现极致弹性，如帮助客户以 3000 pod/min 的弹性轻松应对业务流量洪峰；UCS 提供跨云、跨地域的云原生应用管理，实现一致体验。

- 火山引擎云原生 PaaS 平台 veStack 敏捷版第二季度产品更新

  更新内容：新增地域管理功能；支持 Ingress-nginx 多实例管理；支持 kube-APIServer、kube-controller-manager 等核心组件自定义参数；支持控制面节点故障轮转；API 网关新增 Upstream 抽象、IP 黑白名单、按流量权重路由；微服务引擎新增泳道管理、支持通过同步任务将 Nacos 注册中心中的服务导入微服务引擎。

- 火山引擎容器服务 VKE 第二季度产品更新

  更新内容：支持 NodeLocal DNS 能力、支持 Kubernetes v1.24、新增 Prometheus 监控模块、支持 veLinux CentOS 兼容版镜像、新增 P2P 加速功能、支持 mGPU 多卡共享。

- 火山引擎推出托管 Prometheus 服务 VMP

火山引擎 VMP 是一套基于开源 Prometheus 监控引擎开发的开箱即用的产品方案。VMP 采用单 AZ 多副本、跨 AZ 高可用的方案，支持接入公有云 VKE 等产品，单条 query 扫描样本可多达 3 亿条数据。支持全面的 Kubernetes 集群监控场景、自定义监控场景以及开源生态指标观测场景。

- 火山引擎发布分布式云原生平台 DCP

  DCP 是面向多云多 Kubernetes 集群场景的企业级云原生统一管理平台，提供多云集群统一管理与运维、应用跨集群分发、统一流量管控等能力。DCP 将火山引擎容器集群、第三方集群、自建集群等统一纳管与运维；基于集群联邦等技术，提供跨集群、跨云的弹性调度能力；提供多集群服务发现、多集群统一入口访问等能力；可基于备份数据进行应用数据恢复及跨集群/云应用迁移。

- Isovalent 基于 eBPF 的云原生网络、安全和可观测平台 Isovalent Enterprise for Cilium 1.13 发布

  更新内容：新增 FQDN Ingress 网络策略，用户可以不在网络策略中硬编码 CIDR；支持 SRv6 L3VPN 测试版，用户能够通过 SRv6 将 Kubernetes 工作节点与其他服务和 Kubernetes 集群交叉连接；新增 Phantom Service，Service 中的 LoadBalancer IP 地址可以从 Cluster Mesh 的所有集群中访问；Cluster Mesh 支持重叠的 PodCIDR；支持同时启用端点路由和 BPF 主机路由。

J, K

- 骥步科技多云数据备份恢复产品 YS1000 v3.2 发布

  更新内容：增加集群的 etcd 备份支持；支持创建镜像备份仓库；支持可选应用镜像源的镜像备份；支持沙箱恢复，恢复后不影响原业务；支持备份恢复任务使用钩子程序进行灵活的附属功能配置；支持基于备份恢复的数据卷同步方式；持配置容灾实例时同时配置 ingress 映射。

- Kong 服务网格产品 Kong Mesh v2.3 发布

  更新内容：支持 Envoy 1.26、在 Kubernetes 中采用最小权限的安全默认配置、支持 TLS 网关监听器、允许细粒度控制网格的安装方式、支持只向全局控制平面发送配置更新中的变化、GitOps 工作流支持定制/禁用默认策略创建、支持 init 容器排序、支持 GAMMA 倡议。

- Kong 云原生 API 平台 Kong Enterprise 3.2 发布

  更新内容：当控制平面出现故障时，支持数据平面扩展；新增 Datadog 追踪插件，无需安装 Otel 收集器即可与 Datadog Agent 协同工作；支持基于时延的引导，允许 Kong 负载均衡器在代理上游服务时根据总响应时间选择“最快”的后端。

- Kong 服务网格产品 Kong Mesh v2.1 发布

  更新内容：完成所有下一代策略的实现，包括增加 MeshHTTPRoute、MeshCircuitBreaker、MeshFaultInjection、MeshOPA 等策略；在用户界面中增加了网关视图；支持在 eBPF 模式下配置端口。

- Kong Gateway v3.0 云原生 API 网关发布

  该版本主要新特性：提供联邦信息处理标准 FIPS 包以满足安全和合规性要求、支持选择插件的执行顺序、增加对 WebSocket 流量的本地支持、新增 OpenTelemetry 插件、引入新路由引擎用于处理复杂路由要求、吞吐量、延迟及内存消耗等性能显著优化。

- KSOC 推出业内首个实时 Kubernetes 安全态势管理平台

  Kubernetes 安全态势管理平台可以通过实时上下文以及当前和历史信息准确定位攻击活动，同时还可以根据集群的当前状态提供可操作的补救措施。具体功能包括：实时态势管理，发现基于事件的错误配置；汇总并找到 Kubernetes RBAC 中的过度权限；防止部署不合规的工作负载，减少潜在爆炸半径；扫描漏洞并为运行的容器生成 SBOM。

- Kyndryl 推出多云云原生应用迁移和优化方案 Cloud Native Services

  Kyndryl Cloud Native Services 提供了一个端到端的生命周期框架， 这个框架包括云原生应用迁移和优化所需的代码资产、工作流、专业基础知识、咨询管理服务， 以及与自动备份、补丁、KPI 监测、安全、告警和事件管理相关的综合服务。 该方案允许企业将其内部应用转移到云服务提供商的环境上，如 AWS、GCP 和 Azure。

L

- Lacework 云安全平台 Lacework Platform 2023 年 8 月更新

  更新内容：Agentless Workload Scanning（无代理工作负载扫描）支持扫描由 Containerd 管理的容器镜像、无代理工作负载扫描与 Google Cloud Platform (GCP) 集成、支持扫描主机上多个或次要的存储卷、支持已停止实例的扫描、引入了基于 Pub/Sub 的 (GCP) 审计日志集成、推出近实时告警解决方案。

- Lacework Platform 云安全平台十一月更新

  更新内容：平台扫描器支持多架构的容器镜像、提供攻击路径分析功能、无代理工作负载扫描功能普遍可用、在不活动的主机内核上发现的漏洞会自动被标记为例外、CIS GCP 1.3.0 基准报告和策略普遍可用、新增主机策略用于检测反向 shell 连接和加密劫持工件。

- Lacework Platform 云安全平台 9 月更新

  更新内容包括：支持无代理工作负载扫描、支持对 AWS 上的 Linux 主机进行主机漏洞评估、支持容器镜像漏洞评估。

- Logz.io 发布 K8s 全栈观测平台 Kubernetes360

  Kubernetes360 将日志、Prometheus 指标监控和 Jaeger 支持的分布式追踪统一在一个平台中，使 DevOps 团队能够以简单、高效和可操作的方式监控应用 SLO。

- 灵雀云发布企业级云原生 MLOps 解决方案 Alauda MLOps 以及其开源版本 kubeflow-chart

  Kubeflow-chart 使用 Helm Chart 方式定制了 Kubeflow 的安装方式，只需运行一个命令，即可轻松完成 Kubeflow 及其依赖组件。Alauda MLOps 在此基础上进行了强化，支持非侵入式的可视化工作流编排开发环境、可视化分布式、多租户和配额、虚拟 GPU 以及集成 MLFlow 等。

M

- Mirantis 发布轻量级 Kubernetes 发行版 k0s v1.27

  更新内容：兼容 Kubernetes 1.27；支持 containerd 插件，例如 WebAssembly(WASM) 和 gVisor 容器沙箱；k0s 将用自建的镜像来运行所有的系统组件；支持控制 Helm chart 的安装顺序。

- Mirantis 容器运行时 Container Runtime 23.0 发布

  更新内容：实验性地支持 CSI；增强 OCI 运行时支持，支持 Kata Containers 和 gVisor；执行健康检查所需的开销不再计入时间阈值，即使容器负载大，健康检查所需的时间也不会增加；默认使用镜像构建工具 BuildKit。

- Mirantis 私有云基础设施平台 Mirantis OpenStack for Kubernetes 22.5 发布

  版本特性：全面支持 OpenStack Yoga、引入了基于 OpenStack Manila 的共享文件系统、通过 MetalLB 启用 BGP 模式使 OpenStack 云更容易集成到 L3 网络。

- Mirantis 企业级容器平台 Mirantis Kubernetes Engine v3.6 发布

  该版本主要新特性：支持 Kubernetes 1.24、支持谷歌云平台 GCP、支持 cri-dockerd 取代 Dockershim、安全准入控制更新（提供 OPA gatekeeper 作为 PSP 的可选替代方案）。

N

- Nutanix 推出 Kubernetes 数据管理平台 Nutanix Data Services for Kubernetes (NDK)

  NDK 提供了一个应用感知的 Kubernetes 数据服务方案。NDK 为 Kubernetes 应用提供数据保护、恢复、迁移、克隆和复制数据管理功能；支持轻松将恢复时间目标（RTO）和恢复点目标（RPO）从几天缩短到几分钟；提供策略驱动的有状态应用管理；Kubernetes 和 IT 管理员可以通过设置一些规则和限制来管理基础设施，启用自助服务工作流程。

- NetApp Kubernetes 应用数据管理方案 Astra Control 23.07 发布

  更新内容：集成高可用性存储解决方案 ONTAP MetroCluster、支持将 Longhorn 作为存储提供者、Astra Trident 在面对大规模 Kubernetes 部署时进行了扩展性改进、Astra Trident 支持快照导入功能。

- Spot by NetApp 正式推出 Kubernetes 持续交付产品 Ocean CD

  Ocean CD 为 DevOps 和基础设施团队提供了开箱即用的流程，允许开发人员使用渐进式部署策略，如 Canary 部署、持续验证、故障策略和可观测性，实现通用和可靠的持续交付自动化。服务所有者能够将服务变更推广到生产中，而无需编写代码或重新制定部署策略。

- Spot by NetApp 推出 Kubernetes 应用持续交付产品 Ocean CD

  Ocean CD 是一个支持多集群的 SaaS 方案，以 Argo rollouts 为引擎，并在上面叠加许多管理功能。 Ocean CD 允许快速启用智能部署，如金丝雀、蓝绿部署或使用验证和失败策略的简单滚动更新；支持持续验证，根据金丝雀策略定义正确执行回滚和自动行动；提供一个开发者友好型的 UI 界面。

- NetApp Kubernetes 应用数据管理方案 Astra Control 更新

更新内容： 支持为多个命名空间设置一个应用数据保护策略； 扩大集群范围内资源的检测和保护范围； 基于标签和标签选择器允许每个命名空间有多个应用程序； K8s 应用现在可以在不同的项目、订阅或账号托管的集群间进行故障转移； 集成轻量目录访问协议 LDAP； 支持更多 Self-managed K8s 平台。

o

- Ocean 推出 Kubernetes 网络成本分析解决方案 Network Cost Analysis

  Network Cost Analysis（AWS EKS 集群上已提供 beta 支持）不仅支持显示 Kubernetes 应用使用的网络费用和带宽消耗，还支持预测未来支出和使用趋势。

- Ondat 云原生块存储平台 Ondat v2.10 发布

  更新内容：支持通过 storageoscluster 资源为大部分 Ondat pod 设置容器资源限制、支持实时计算平台 Red Hat Enterprise Linux for Real Time、operator 默认安装 CLI pod、允许在节点之间移动卷。

- Ondat v2.9 云原生块存储平台发布

  该版本主要新特性：支持在零停机的情况下调整卷的大小、通过存储池可以控制工作负载所使用的存储类型、允许定义卷的拓扑结构。

- Oracle Kubernetes 发行版 Container Engine for Kubernetes 支持虚拟节点池和虚拟节点

  虚拟节点提供了 serverless Kubernetes 的体验，用户无需管理、扩展、升级和排障节点基础设施，即可规模化运行容器化应用程序。虚拟节点提供了细粒度的 pod 级别弹性和按使用量计费的定价。用户可以通过在增强的集群中创建虚拟节点池以创建虚拟节点。

- OutSystems 发布云原生低代码开发解决方案 OutSystems Developer Cloud (ODC)

ODC 是用于构建云原生应用程序的高性能低代码解决方案。 ODC 将 Kubernetes、Linux 容器、微服务和 AWS 原生云服务与 DORA 高性能级 CI/CD、企业级安全性以及基于模型的可视化开发相结合。 支持大规模交易量和数据需求，通过高级可视化编程和人工智能提高开发人员生产力，实现高性能 CI/CD 实践。

P

- PerfectScale 推出同名 Kubernetes 成本管理 SaaS 平台

  PerfectScale 是业界首个专门为提高 K8s 环境的持久性和成本效益而建立的持续优化方案。 通过人工智能算法来评估使用模式和性能和成本指标，该平台提供了多集群、多云的可观测性，支持弹性和性能风险检测、资源浪费检测， 支持问题优先级排序，提供 GitOps 友好的的补救措施建议，支持预测系统变化对环境的影响等。

- Portworx K8s 数据备份产品 Portworx Backup v2.5 发布

  更新内容：支持将 Kubernetes 备份数据存储在 NFS v4 对象中；支持使用 IBM Cloud Block Storage VPC CSI Driver 5.0.0 或更高版本，在 IBM Cloud Kubernetes Cluster 集群上进行 CSI-native 备份和恢复。

- Portworx 云原生数据管理平台 Portworx Enterprise v3.0 发布

  更新内容：异步模式下的灾难恢复迁移时间缩短了 50%、新增用于蓝绿升级的许可证扩展、利用基于 NVMe 的存储为 AWS 用户提供低且一致的 I/O 延迟、可使用客户管理的密钥为 Oracle 云以及 AWS、GCP 和 Azure 的云驱动器进行加密、FlashArray 云驱动器新增 CSI 拓扑结构。

- Portworx Enterprise 2.12 云原生数据管理平台发布

  该版本主要新特性：本地用户可以启用 PX-Fast 功能，使用高性能存储作为 Kubernetes 的持久化存储；允许存储管理员使用自定义的 Kubernetes 对象为对象存储桶提供各种支持服务；自动生成 Vault 令

牌，在 Vault 中存储加密 secret 和云证书；允许 Kubernetes 资源迁移到目标集群之前对其进行修改。

- Portworx Backup v2.3 数据备份产品发布

  该版本主要新特性：支持与其他用户和组共享备份、支持根据特定的基础设施定制备份许可证、支持使用用户提供的或默认的密钥为不同类型的备份提供加密支持。

Q

- 青云发布分布式多租户云原生操作系统 KubeSphere 企业版 4.0

  KubeSphere 企业版 4.0 基于全新的云原生可扩展开放架构——KubeSphere LuBan 打造。KubeSphere LuBan 是一个分布式的云原生可扩展开放架构，为扩展组件提供可热插拔的微内核。4.0 具备简单轻量的系统架构，2 核 4G 即可轻松玩转；还支持集成 KubeSphere 功能组件或任何第三方业务组件，实现即插即用。

- 青云 Kubernetes 集群巡检 SaaS 服务正式发布

  Kubernetes 集群巡检 SaaS 服务通过即时或周期性检查 Kubernetes 多云环境中的集群节点、组件等配置是否符合最佳实践，帮助用户及时发现集群组件、服务、端口中的容器漏洞和暴露（CVE），分析 Kubernetes 中的运行风险，并推送报告，从而保证业务持续稳定运行，尽早地降低企业风险。

- 青云发布 KubeSphere 容器平台企业版 3.4.0

  更新内容：支持 Prometheus 内部 TLS 认证访问、内置集成 HPA 扩展插件 KEDA、新增企业空间级网关与租户级存储配额、提升资源别名展示权重、支持通过 UI 配置 Whizard 可观测中心。

- 青云企业云平台 v6.1 版本正式发布

更新内容；新增监控巡检功能；新增企业空间功能，涵盖组织管理、用户管理、配额管理、资源管理、流程审批等空间管理模块；新增对第三方存储的支持；提供 VMware vSphere 纳管工具；QKE 容器引擎支持裸金属服务器作为集群 Worker。

- 青云发布云原生虚拟化平台 KSV 1.6

  版本特性：统一管理与展示所有计算、存储和网络资源；支持 KSV 和 KubeSphere 融合部署，实现虚拟机和容器共存共管；基于 KubeVirt，支持以虚拟资源池的方式交付 CPU、存储等物理资源，并实现统一的管理、分配及调度，支持同时创建相互隔离的虚拟机环境；集成 Kube-OVN，实现了 Underlay & Overlay 的网络。

- 青云发布分布式多租户云原生操作系统 KubeSphere 企业版 3.3

  版本特性：新增可观测中心，汇总同步分散在各个集群的数据；重构集群监控页面，聚焦资源分配；优化租户监控，支持展示租户配额设置情况等；新增对微服务框架 Spring Cloud 的支持；集成 Argo CD 支持 GitOps。

- Quali 基础设施自动化方案 Torque 更新，简化 Kubernetes 基础设施管理

  更新内容包括：支持检测 Helm Chart 漂移、支持自动收集 Kubernetes 主机成本、收集的数据可以导入第三方审计工具、提供对环境定义的所有子组件的可见性。

- Qualys 容器安全工具 Qualys Container Security v1.22 发布

  更新内容：镜像漏洞报告支持显示与镜像相关的标签、容器漏洞报告支持显示镜像仓库信息和 Kubernetes 对象信息、支持扫描所有注册表中的所有镜像、软件成分分析（SCA）扫描增加对编程语言 PHP、Ruby 和 Rust 的支持。

- Qualys 容器安全工具 Qualys Container Security v1.21 发布

更新内容：支持为普通和 CI/CD 传感器提供传感器配置文件、支持在传感器配置文件中定义传感器配置、在自动注册表扫描时允许扫描所有镜像。

R

- Rafay Kubernetes 运维平台支持自动化管理 Fleet，简化 Kubernetes 多集群的管理

  自动化管理 Fleet 功能可简化跨数据中心、公有云和边缘环境的大规模 Kubernetes 集群的管理。具体包括：自动执行常用的 Fleet 管理操作，如升级集群、更新应用、扩展节点等；构建端到端自动化和自定义工作流；在后台自动执行集群的标准化策略和治理，减少管理开销；利用 Terraform 等现有工具启动整个 Fleet 的操作，并通过系统同步在 GitOps 资源库中反映 Fleet 配置的更新情况。

- Rafay Systems 推出 Environment Manager，优化 Kubernetes 开发者体验

  Environment Manager 通过为全栈环境提供自助服务功能，改善开发者体验。开发人员能够从平台团队策划、测试和持续管理的环境蓝图中配置现代应用堆栈。通过和 Rafay 的 Kubernetes 运维平台集成，Environment Manager 抽象了底层基础设施的复杂性，减少了配置和访问 Kubernetes 的环境所需的时间。

- Rafay Systems 发布 Kubernetes 成本管理服务

  成本管理服务是 Rafay Kubernetes 运维平台的一部分，可为位于公有云和内部数据中心的集群提供 Kubernetes 云成本优化。 其支持实时查看云计算支出；与 RBAC 预集成，提供基于角色的成本指标的可见性和访问权；可集中查看多个公有云账号和内部数据中心的集群；可根据资源消耗优化云预算。

- Rafay Systems 推出 Service Mesh Manager and Network Policy Manager，提供企业级的 K8s 流量管理和通信安全保障

Service Mesh Manager 基于 Istio 构建，为微服务团队提供集中的安全控制和流量管理策略配置。Network Policy Manager 基于 Cilium 构建，提供集中管理以及 pod 和命名空间通信的可见性，以隔离边界并减少集群的横向攻击面。

- Rainbond 云原生多云应用管理平台 Rainbond 企业版发布

  Rainbond 企业版拥有全局可观测中心，用于监视、调试和分析运行在 Kubernetes 集群和 rainbond 平台中的基础设施及应用服务。此外，还为 IT 业务系统向国产化信创环境中迁移提供了方案，支持全链路灰度发布和集群巡检，内置 WAF 防火墙和操作审计功能，支持私有化部署应用商店等。

- Red Hat 容器镜像仓库 Red Hat Quay v3.9.0 发布

  更新内容：增加容器镜像漏洞报告的覆盖范围、审计日志与外部日志管理系统 Splunk 集成、提供了一种可扩展的方式来跟踪注册表中大量租户的存储消耗、自动化异地备份功能。

- Red Hat 服务网格 OpenShift Service Mesh 2.4 发布

  更新内容：支持 Istio 1.16 和 Kiali 1.65、正式支持集群范围的部署拓扑架构、集成 Cert-Manager、正式支持外部授权、支持单栈 IPv6、新增用于 OpenShift 监控或外部 Prometheus 的 Prometheus 扩展 provider。

- Red Hat 发布混合云和多云环境服务连接产品——服务互连

  红帽服务互连基于开源项目 Skupper.io 创建，它能够在红帽平台（包括红帽企业 Linux、红帽 OpenShift 和非红帽环境（包括 Kubernetes 集群、公共云、虚拟机或裸机主机）之间实现应用和服务连接，而无需进行复杂的网络重新配置或提升安全权限。

- Red Hat 发布本地容器管理工具 Podman Desktop 发布 GA 版本 1.0

Podman Desktop 可在本地环境中安装、配置 Podman 并使 Podman 保持最新版本。提供了一个仪表板来与容器、图像、Pod 和卷进行交互，还可以使用 OCI 注册表和网络设置来配置环境。Podman Desktop 支持多种容器引擎，提供将 Pod 连接和部署到 Kubernetes 环境的功能。

- Red Hat OpenShift Container Platform 4.13 发布

  更新内容：cert–manager operator 提供应用证书生命周期管理；推出 Kubernetes 高级集群安全防护 4.0 版本；OpenShift Service on AWS 增加托管控制面板；推出自定义指标自动缩放 operator；推出 NUMA 感知调度与 NUMA 资源 operator；支持在 AWS、Azure、Google Cloud Platform 和 vSphere 上支持部署紧凑型三节点集群。

- Red Hat OpenShift 日志管理方案 Logging 5.6 发布

  更新内容：兼容 OpenShift 容器平台集群范围内的加密策略；支持通过 LokiStack 自定义资源声明租户、流和全局策略保留策略，并按优先级排序；新增日志转发输出选项 Splunk；Vector 取代 Fluentd 作为默认收集器。

- Red Hat 多集群管理平台 Advanced Cluster Management for Kubernetes 2.7 发布

  更新内容：支持根据依赖关系进行策略执行排序、策略生成器支持引用本地和远程定制配置、扩大边缘可管理的集群数至 3500 个、支持在 Arm 架构上创建集群、针对大规模环境的搜索组件正式可用、支持使用新的 Submariner LoadBalancer 模式简化集群部署、Submariner 支持无网络环境的集群。

- Red Hat 云原生 CI/CD 解决方案 OpenShift Pipelines 1.9 发布

  更新内容：Pipelines as Code 正式可用，支持在源代码库中定义 Tekton 模板；支持存储库 CRD 的并发限制；支持对管道中的 URL 进行认证；

新增 Resolvers 功能用以“解决”远程任务和管道的请求；新增 CLI 工具 opc。

- Red Hat OpenShift Container Platform 4.12 发布

  更新内容：使用 OVN-Kubernetes 网络插件作为默认网络插件；新增拓扑感知的生命周期管理器用于管理多个单节点 OpenShift 集群的部署和升级；支持通过 cgroup v2 优化资源分配管理；支持快速、低内存占用的 crun 容器运行时；针对断网环境优化基于代理的安装器；支持管理节点层面的防火墙配置；支持根据集群中的指标动态扩展默认的 Ingress 控制器；支持为 SR-IOV 设备配置多网络策略；支持 Serverless function 功能；新增 OpenShift Network Observability Operator 进行网络排障；新增 Security Profiles Operator 改善安全态势；支持将生产级 Kubernetes 部署到边缘设备上。

- Red Hat 容器镜像仓库 Red Hat Quay v3.8.0 发布

  该版本新特性：支持 IPv6 单栈和 IPv4/IPv6 双栈、用户必须拥有自签名的证书才能使用 SAN（Subject Alternative Name）、Quay 管理员可以使用存储配额来限制缓存的大小、新用户类型“受限用户”、超级用户经配置可以对系统中的所有内容拥有完全控制权。

- Red Hat OpenShift Service Mesh 2.3 发布

  该版本主要新特性：支持 Istio v1.14、支持通过注入 deployment 实例来创建和管理网关、新增集群范围的拓扑结构、增加 OpenShift 服务网格控制台 operator、支持 Istio 可视化工具 Kiali 1.57。

- Red Hat Openshift 日志管理服务 Logging 5.5 发布

  该版本主要新特性：支持把同一 pod 内不同容器的结构化日志转发给不同的索引、使用 Kubernetes 通用标签过滤带有 Elasticsearch 输出的日志、日志聚合 Loki Operator 和 观测数据收集 Vector collector 正式可用。

S

- Seal 应用统一部署管理平台 Seal AppManager v0.2 发布

  更新内容：支持服务配置变更历史的比较；支持批量以及跨环境克隆服务；优化服务和资源的操作交互；支持项目级别的环境/连接器管理；支持展示环境依赖图；支持克隆环境；支持多层级的变量配置；Kubernetes 高可用性安装部署、RBAC 和多租户管理。

- Sealos 云操作系统正式发布

  Sealos 是一款以 Kubernetes 为内核的云操作系统发行版。它以云原生的方式，抛弃了传统的云计算架构，使企业能够像使用个人电脑一样简单地使用云。借助 Sealos,你可以运行任何兼容 Kubernetes 的应用；30 秒在 Sealos 上跑个 nginx demo，自动伸缩；30 秒起各种数据库，业务系统内网直接连接数据库；直接启动你写的各种编程语言业务等等。

- ServiceNow 发布统一查询语言 Lightstep UQL，扩展 Kubernetes 应用的可见性

  Lightstep UQL 支持统一的“可观测性即代码”，通过单一查询语言简化从多个不同工具迁移到统一 Lightstep 平台的过程，还支持跨多个 Kubernetes 节点、服务器或 serverless 函数查询和关联指标、日志和跟踪。

- SmartX 携手灵雀云推出全栈云原生联合解决方案

  在该方案中，SmartX 基于超融合软件 SMTX OS 为客户提供生产就绪的 IaaS 层底座，并通过 Kubernetes 构建与管理服务工具 SMTX Kubernetes Service 帮助创建并管理生产级 Kubernetes 集群。灵雀云 ACP 容器云平台帮助客户实现业务应用从开发、测试到部署、运维的全生命周期平台化管理。

- SmartX 推出国内首款 K8s 原生的企业级分布式存储 IOMesh

IOMesh 对基础设施及 Kubernetes 平台无依赖，可充分利用 Kubernetes Worker 节点的硬件资源融合部署，支持小规模起步，可根据业务发展按需投资可为有状态应用提供 Kubernetes 原生、高性能、高可靠、高可扩展的持久化存储能力。

- SmartX 发布生产级 Kubernetes 构建与管理服务产品 SMTX Kubernetes Service （SKS） 1.0

  SKS 可基于 SmartX 超融合虚拟化自动创建多台虚拟机以构建高可用的 Kubernetes 集群，内置 SmartX 生产级分布式存储和 CSI 插件，可为有状态应用提供持久卷；通过 SmartX 网络与安全产品和 CNI 插件，实现虚拟机和容器的互联互通以及统一的网络安全策略管理。

- Solo.io 发布云原生应用网络平台 Gloo Platform v2.3

  更新内容：新增 Gloo Portal，帮助开发者对 API 进行分类、分享和管理；支持在 Istio 服务网格中处理 GraphQL 请求的流量，包括南北向和东西向的流量；支持 Kubernetes 1.25 和 Istio 1.17；商业化支持 Ambient Mesh。

- Solo.io 推出多云动态资源发现方案 Gloo Fabric

  Gloo Fabric 为 Gloo Platform 提供了多云发现、连接、安全和可观测性功能。Gloo Fabric 通过统一的 API 来配置和管理平台的所有功能，通过统一的管理控制平面来管理网络和安全策略，支持动态地发现网络和集群资源，并提供 API 管理和跨集群多租户隔离功能。此外，Gloo Fabric 还提供自动零信任安全和平台集成的可观测功能。

- Solo.io 发布云原生网关 Gloo Gateway 1.13

  版本特性：改进对 OpenCensus 和 OpenTelemetry 追踪扩展的支持；提供了一种内置的方法，可以在错误发生前捕获无效的速率限制配置；增加对 RBAC 的支持；增强控制平面的高可用性，支持多个副本的连续运行和零停机升级；集成集群调度工具 HashiCorp Nomad。

- Solo.io 发布服务网格和 API 平台 Gloo Platform

  Gloo Platform 是 Solo.io 的集大成之作，集成了三个产品： Gloo Gateway、Gloo Mesh 和 Gloo Network。利用 Kubernetes CR 和 GitOps 提供一个统一的操作模型，并将零信任安全、多租户、高级路由和可观测性等功能扩展到整个第三层到第七层堆栈。

- Solo.io 发布云原生网关 Gloo Gateway 和 CNI 插件 Gloo Network

  Gloo Gateway 基于 Envoy 构建，利用 Gloo Platform 的多租户和联邦特性，允许用户轻松管理多个开发团队的网关访问和进行多集群流量管理。Gloo Network 基于 Cilium 构建，将 Kubernetes CNI 层整合为 Gloo Platform 的组件。

- Spectro Cloud 云原生边缘计算平台 Palette Edge Platform v3.3 发布

  版本发布：支持从外部 OCI 注册表中加载镜像、安装程序现在可以包含预加载内容包（包括软件包和工件）；支持创建自定义 Edge 安装程序镜像；支持随机生产设备的 UUID 值。

- Spectro Cloud Kubernetes SaaS 管理平台 Palette v3.1 发布

  更新内容：Palette IaaS 集群支持自动扩缩容、提供符合联邦信息处理标准（FIPS）的 Kubernetes 版本、支持按标签过滤用户访问和查看集群详情视图、支持应用程序配置文件的版本管理、利用 Helm 和 OCI 注册表进行自定义包管理、允许暂停和恢复未使用的虚拟集群。

- Spectro Cloud 发布 Kubernetes SaaS 管理平台 Palette v2.8

  该版本主要新特性：支持利用嵌套集群为应用快速创建安全隔离的环境；利用基于 Web 的 Kubectl 允许用户通过终端部署应用；支持重复使用和共享具有许多附加组件和集成的大型配置文件。

- Spectro Cloud 云原生边缘计算平台 Palette Edge Platform 更新

更新内容包括：新增边缘 K8s 集群的防篡改功能、支持边缘优化的 K8s 发行版 Palette eXtended Kubernetes Edge、简化边缘设备部署。

- StormForge 发布 Kubernetes 生产环境优化方案 Optimize Live v2.0

  更新内容：通过一个 helm deployment 即可调整应用的大小；使用机器学习分析工作负载使用模式，提供 CPU 和内存建议；能够自动检测 HPA 的存在；提供一系列资源使用报告；简化资源建议的部署。

- StormForge K8s 优化平台与 Datadog 的云应用监控服务集成，简化 Kubernetes 的可观测性

  StormForge K8s 优化平台基于机器学习技术分析可观测数据并推荐资源设置（CPU、内存、副本），提高效率、规模和应用性能。平台现在可利用 Datadog 作为一个统一的观测平台，查看和应用优化建议，并确定应重新校准的应用参数，以减少集群规模，回收和重新分配资源。

- StormForge Kubernetes 集群性能优化方案 Optimize Live 利用机器学习优化 K8 自动扩缩容能力

  最新版本的 Optimize Live 提供了二维自动缩放能力，HPA 和 VPA 能够协同工作，既能调整 pod 的大小，又能水平设置利用率，最大程度优化自动缩放的效率。此外，还利用机器学习分析历史资源使用情况，以找到最佳的 HPA 目标利用率。

- SUSE 推出企业级容器管理平台 Rancher Prime

  Rancher Prime 是 Rancher 的一种分发版，核心功能代码均来自 Rancher 社区版，但更加重视安全方面的建设，并面向企业用户强化了相关功能和服务。 Rancher Prime 的特色功能在于：综合安全治理能力提升，提供可信的镜像仓库； 引入 UI 扩展功能；提供对阿里云、腾讯云以及华为云的托管集群的全生命周期支持； 引入对 openEuler Linux 的支持；对 ARM 体系的支持。

- SUSE 发布云原生边缘管理平台 SUSE Edge 2.0

  该版本主要新特性：简化边缘设备的添加和更新操作、支持通过统一的操作面板管理 Kubernetes 和底层操作系统、集成专为容器化和虚拟化工作负载打造的轻量级操作系统 SUSE Linux Enterprise Micro 5.3、为所有分布式环境提供安全防护。

- Sweet Security 推出云原生运行时保护平台

  Sweet Security 推出云运行时安全套件平台，用于保护云原生应用运行时的安全。该平台使用传感器将应用数据流传输到其服务器，该传感器通过运行一个框架来分析工作负载的异常行为。该传感器可在五分钟内完成部署，能在攻击发起前和攻击发生时提供对攻击的可观测性能力。

- 观测云可观测解决方案观测云产品更新

  更新内容：Span 列表页面新增瀑布图模式、新增定时报告功能、新增服务管理用于查看当前工作空间内不同服务的性能和业务数据及所有的关联分析等信息、支持在日志详情页直接查看上下文日志、新增当前服务的上下游调用关系表格、检测指标新增转换为 PromQL 查询、新增 Kubernetes 事件组件。

- 实时监测平台观测云更新

  更新内容包括：支持查看基础设施容器对应的 YAML 文件、新增日志查看器 DQL 搜索模式、优化应用性能监测、DataKit 支持将 k8s label 作为 tag 同步到 pod 的指标和日志中、支持将 k8s 中各类 yaml 信息采集到对应的对象数据上、Trace 采集支持自动提取一些关键 meta 信息。

- 时速云微服务治理平台 TMF v5.6.0 发布

  该版本主要新特性：支持独立部署，实现与底层平台的解耦；解耦微服务框架和性能监控能力两个模块；新增链路组件拓扑图；增加对无损流量上下线功能的支持；新增主备拓扑能力，拓扑图可视化展示主备关系。

- 数澈软件发布 SEAL 0.3：国内首个全链路软件供应链安全管理平台

  该版本新特性：支持集成任意 OCI 镜像仓库，扫描其中的容器镜像；支持集成任意 Kubernetes 集群，扫描其中的工作负载配置及镜像；支持扫描第三方软件物料清单文件；支持在任意 CI/CD 流水线中集成 SEAL 的安全扫描功能；能够聚合管理全链路各个阶段的资源提供全链路安全洞察；支持自动生成多策略修复建议、漏洞优先级排序、因时制宜处理安全问题。

T

- Tetrate 推出针对 Amazon EKS 设计的服务网格解决方案 TSE

  TSE 基于 Istio 和 Envoy 等开源服务网格组件构建，并针对 Amazon EKS 对 TSE 进行了简化安装、配置和操作的优化。 TSE 提供了 Istio 和 Envoy 之上的服务网格自动化。 它用于在 Amazon EKS 上安装和配置开源组件，与 AWS 服务集成，并为平台运营商提供管理控制台，以快速配置服务网格以实现安全、弹性和可观察性。

- Tetrate 应用连接平台 Tetrate Service Bridge v1.6.0 发布

  更新内容：增加安全域、服务安全设置等安全规则；增加东西向网关改善集群间的服务故障转移；用户界面优化，支持可视化和监控平台和服务活动；新增排障工具，无需集群的访问特权即可排障；支持集群内的多 Istio 环境；支持跨网关和服务代理的 WASM 扩展；Skywalking 的后端服务 OAP 代替 Zipkin，用于收集和查询 trace。

- Tigera 发布 Kubernetes 网络和容器威胁防御方案 Calico Runtime Threat Defense

  Calico Runtime Threat Defense 结合结合基于签名和行为的技术来检测已知威胁和零日威胁，能够检测 MITRE 最常见的容器和网络的攻击。 与传统的运行时威胁检测平台不同，Calico Runtime Threat Defense 无需编写复杂的规则即可持续监控和分析网络和容器行为，获取攻击指标（IOA）。

- Traefik Labs 推出业内首个 Kubernetes API 管理方案 Traefik Hub

  Traefik Hub 用于发布、保护和管理 API，支持 Traefik 和主要的第三方入口控制器，包括 NGINX、HAProxy、Ambassador 等。支持 GitOps，提供强大的安全功能，如基于角色的访问控制（RBAC），并支持认证和授权机制，如 OAuth 2.0 和 JWT。

- Traefik Labs 发布云原生 API 网关 Traefik Enterprise 2.10

  更新内容：允许 OIDC 中间件检查访问令牌中的要求、允许从 Kubernetes Secret 中加载敏感数据、支持限制 API 的访问速率。

- Traefik Labs 发布云原生网络平台 Traefik Hub 1.0

  Traefik Hub 1.0 允许用户可以使用 Traefik 或 Nginx 快速、安全地发布 Kubernetes 或 Docker 容器。Traefik Hub 提供了为 Kubernetes 集群联网所需的集中控制平面，而无需部署容器 sidecar 来运行网络软件。其支持通过安全加密隧道和直接私有连接进行容器联网，通过 JWT 或 OIDC 为服务添加访问控制，通过 GitOps 实现规模化的自动化等，并配有工作空间促进跨团队协作等。

- 腾讯云容器服务 TKE 2 月更新

  更新内容：集群节点升级入口支持 docker/containerd 组件小版本更新、上线 CFS-Turbo CSI 存储插件、集群日志采集规则支持对元数据的自定义配置、注册节点支持 GPU、支持通过云上控制台一键在用户 IDC 创建、管理和升级 Kubernetes 集群。

- 腾讯云云原生 API 网关更新

  更新内容：支持对接云函数 SCF 和微服务平台 TSF、支持定时弹性扩缩容和实例升降配、Kong Ingress 支持使用 Annotation 式配置、灰度能力产品化、支持对接云上 WAF 和云上 SSL、支持 mTLS。

- 腾讯云容器服务 TKE 三项能力升级

升级内容：节点管理方面，新增 HouseKeeper 运维范式，对原生节点、超级节点、注册节点等云上/云下资源进行纳管； 集群管理方面，支持一种集群管理任意节点，同时支持原生节点、超级节点、注册节点和边缘节点等多种节点； 应用管理方面，打造面向多云和边缘场景的应用管理平台，实现多集群统一分发、部署和管理。

- 腾讯云容器服务 TKE 9 月更新

更新内容：支持镜像仓库签名镜像的可信验证、kubelet 自定义参数功能全量开放、提供异常 Service/Ingress 事件信息错误码的说明、灰度上线超级节点上运行 Daemonset 能力、新增按照 Label 配置和管理 Pod 安全组的功能。

- 腾讯云 K8s 发行版 TKE 8 月更新

更新内容包括：新增 SecurityGroupPolicy 增强组件，支持为策略匹配的 Pod 绑定安全组；支持业务 Pod 使用 Service Account Token 访问 CVM、VPC 等云资源；支持在不重启 Pod 的情况下修改 CPU、内存的 Request/limit 值；支持节点/工作负载资源洞察能力；新增 Request 智能推荐功能；新增原生节点专用调度器；优化原生节点初始化流程。

- 天翼云发布云原生安全产品——红盾 1.0

红盾基于云原生底座将安全能力整合至统一的安全平台，面向云原生业务应用安全、网络安全、数据安全、云原生安全 4 大领域，打造一体化云安全可信运营体系和零信任架构。 其核心产品包括 Web 应用防火墙、DDoS 高防、网站安全监测、企业安全访问、天翼云数据安全管理平台等。

U, V

- Upbound 推出基于 Crossplane 的托管型控制平面管理服务（MCP）

用户可以可以通过 MCP 扩展到数千个 CRD，使用控制平面来管理其所需的所有云服务资源，从而允许用户运行数十个甚至数百个控制平面来为不同的环境、团队、业务和客户服务。 此外，MCP 与 Git 无缝集成，提供用

于管理和操作控制平面以及所有资源的控制台，支持安装在任何 Kubernetes 集群中，并支持在 GitOps 流程中管理基础设施和应用程序资源。

- Veeam 发布 Kubernetes 数据管理平台 Kasten K10 v6.0

  更新内容：提供一种存储可重复使用元数据转换的方法；支持多集群间共享许可证；支持应用指纹识别，即新部署的有状态应用程序可以自动映射到正确的蓝图；与 Veeam 数据平台集成，允许在 Veeam Backup & Replication V12 中集中查看和管理 Kubernetes 备份；支持即时恢复能力。

- Veeam 发布 Kubernetes 数据管理平台 Kasten K10 v5.5

  该版本主要新特性：新增备份窗口允许用户选择策略运行的时间间隔、支持自动安排底层备份工作的顺序、支持定义多个保护策略以设置备份频率和位置等参数、提供可视化 Helm 向导程序、支持 IPv6、集成 GitOps 流水线、新增存储类型、通过 OpenSSF 和 Azure Managed Identity 增强备份安全性。

- Venafi 推出云原生机器身份管理服务 TLS Protect for Kubernetes

  TLS Protect for Kubernetes 是 Venafi 机器身份管理平台 Control Plane for Machine Identities 的一部分， 帮助安全和平台团队在多云和多集群 Kubernetes 环境中管理云原生机器身份，如 TLS、mTLS 和 SPIFFE，增强机器身份管理的可观测性、控制和自动化。

- VMware Kubernetes DevOps 平台 VMware Tanzu Application Platform 1.6 发布

  更新内容：推出基于 Backstage 构建的内部开发人员门户 Tanzu Developer Portal、允许开发人员创建通过 Contour Ingress 对外暴露给公共互联网的服务器工作负载、集成容器镜像漏洞扫描器（beta）、支持通过 Tanzu Insight CLI 筛选 CVE（alpha）、支持提取工作负载构建的

SBOM、能够根据工作负载所使用的依赖关系中的上游变化进行自动构建、为供应链中的内部部署 Git 资源库提供自定义 CA 支持、支持基于 GitOps 的安装方式。

- VMware 威胁检测和响应产品 VMware Carbon Black XDR 增加对云原生应用的支持

  VMware Carbon Black 的云原生检测和响应（Cloud Native Detection and Response）功能使得用户能够在一个统一的平台内为容器和 Kubernetes 使用增强的威胁检测能力。具体包括：监控容器和 Kubernetes 环境中运行的进程；获得先前存在的容器中检测到的任何异常的历史数据；警报鉴别分类，分析事件来自哪个容器或 Kubernetes 节点。

- VMware 推出面向多云/混合云环境的 Kubernetes 资源统一管理产品 Tanzu Mission Control 自管理版本

  VMware Tanzu Mission Control 自管理版本（TMC Self-managed）将 TMC 作为本地部署的方式推向市场，使原先无法使用 TMC SaaS 服务的用户能够尝试使用该 Kubernetes 管理平台，或者原有 SaaS 用户在一些特殊情况下（比如断网）能够作为管理平台的高可用性备份。

- VMware 发布 API 网关 Spring Cloud Gateway for Kubernetes 2.0

  更新内容：支持使用 Redis 管理 API 网关实例之间的共享状态、使用 Zipkin 实现 Tracing、支持配置 API 网关实例从而执行上游应用程序的健康检查、支持使用元数据根据每个 API 路由配置跨源资源共享（CORS）、支持 Kubernetes 1.23+。

- VMware Tanzu 发布 Kubernetes DevOps 平台 VMware Tanzu Application Platform 1.5

  更新内容：完成支持 GitOps 模式的 Namespace Provisioner；平台工程师可以使用私有 Git 仓库来引用模板化的资源；支持基于 GitOps 的安装

流程；新增面板用于显示集群中工作负载的详细信息；新增 External Secrets Operator；支持分布式 API 网关 Spring Cloud Gateway for Kubernetes。

- VMware Tanzu 服务网格方案 Tanzu Service Mesh 3.0.3 发布

  更新内容：Tanzu Service Mesh CLI 支持 GitOps，可用于自动部署 Tanzu Service Mesh 环境到集群中；支持选择要注入 proxy sidecar 的命名空间；支持客户端集群和 Tanzu Service Mesh SaaS 之间的企业级代理通信。

- VMware Tanzu 多云多集群 Kubernetes 管理方案 Tanzu Mission Control 更新

  更新内容：支持 Pod 安全的 mutation 策略、支持 Tanzu Kubernetes Grid 2.1（包括 ClusterClass）、支持集群组的持续交付、支持从 Git 仓库中安装 Helm chart 到集群中。

- VMware Tanzu K8s DevOps 平台 Tanzu Application Platform v1.4 发布

  更新内容：支持 shared ingress issuer、新增命名空间配置器实现安全自动化的命名空间配置、新增 TAP 遥测报告以供查看 TAP 的使用情况、新增 Visual Studio 的 IDE 扩展 —— Tanzu Developer Tools for Visual Studio、支持 External Secrets Operator。

- VMware Tanzu Kubernetes Grid 2.1 发布

  该版本主要新特性：引入新的 Cluster API 功能 ClusterClass 以及 Carvel 工具，使用统一的、声明式的 API 创建和管理集群；支持公有云 Oracle Cloud 基础设施。

- VMware Tanzu 发布 Application Service Adapter for Tanzu Application Platform v1.0，旨在弥补 Cloud Foundry 和 Kubernetes 间的开发体验差距

  该版本主要新特性：支持在 Kubernetes 和 TAP 上提供一个无缝的 Cloud Foundry 推送工作流、使用 Contour 复制 Cloud Foundry 部署中的 goRouter 入口模式、使用本地 Kubernetes RBAC、重建了由本地 Kubernetes 命名空间支持的 Cloud Foundry 组织和空间结构、集成 Tanzu Build Service、集成 TAP 的端到端流水线供应链 Supply chain choreographer（实验性）。

- VMware Tanzu v2.0 应用容器化工具发布

  该版本主要新特性：集成自动化扫描工具 Cloud Suitability Analyzer（CSA）、支持 Windows 容器化、支持虚拟机容器化、支持 Linux 和 Windows 平台的 200 多个组件签名、提供命令行界面。

- VMware Tanzu Application Platform v1.3 (K8s DevOps 平台) 发布

  更新内容：支持在物理隔离环境下运行、集成供应链威胁扫描工具、新增一个统一的威胁监控大盘、支持 SBOM、支持 API 规范动态注册、集成 Jenkins CI/CD、支持自定义证书机构（CA）证书、新增运行时资源监测插件、增加对 Java 和 Python 函数工作负载的支持（beta）、在 OpenShift 上可用。

- VMware Tanzu Kubernetes Grid v1.6 发布

  该版本主要新特性：支持在 vSphere 7.0+ 上将工作负载集群部署到支持 GPU 的主机和边缘设备、使用 Multus 和 Whereabouts 实现多个 Pod 网络接口、支持使用 Amazon EBS CSI driver 和 Azure Disk CSI driver for Kubernetes 的 CSI 存储。

- VMware Spring Cloud Gateway for Kubernetes v1.2.0 发布

该版本主要新特性：分离客户端入口和上游应用服务通信的 TLS 配置、扩大自定义扩展云秘密管理的权限、增加熔断器状态指标、新增全局和每个 API 路由的响应缓存配置、增加 JSON 请求转换到 gRPC 上游服务的过滤器、增加 podOverrides 以配置 API 网关实例上的 K8s pod override、API 网关现在可以通过独立的 JAR 部署。

W, Z

- Weave GitOps 更新

  更新内容包括：优化可信交付（Trusted Delivery），增加策略即代码和渐进式交付功能；新增 Team Workspaces，优化多租户的工作方式；用户界面新增 ClickOps 功能，简化微服务部署；新增 GitOps Run 的技术预览功能，提供接近实时的迭代开发，一旦本地清单文件被保存，它就会立即与 Kubernetes 核对。

- 中国联通云原生架构全面升级

  升级亮点：支持多种国产芯片和国产操作系统；兼容 CNM、CNI 双标准规范；新增 DVR 分布式网关、VPC 互连、VPC 弹性网卡直通网络(基于 IPVLAN)、VPC 路由直通网络(兼容 Flannel/ Calico 插件)、策略路由、VPC 网络多出/入口、IPv4/6、四/七层负载分离等功能；单集群容量支持承载超 10W+ 容器；集成一整套可视化自动运维能力。

云原生社区要闻 - 2023

本页按月份列出 2023 年云原生社区比较热门的新闻、事件、活动等，供大家一站式查阅和检索。

2023-10

- 10 月 行业云平台系列平台运维能力评估开放报名（10 月截止）

  业云平台系列平台运维能力评估面向金融、电信、政务、交通、能源等行业云平台运维方展开评估，评估维度包括人员与资质、运维服务台、平台维护、安全管理、稳定运行五大能力及行业场景能力，从技术、人员、服务、流程多个角度进行能力度量。

- 10.27–10.28 FCon 全球金融科技大会启动

  2023 年 10 月 27–28 日，极客邦科技将于上海举办 FCon 全球金融科技大会。大会将汇聚中国互联网金融企业、银行、保险、证券等领域的懂技术、懂业务、懂产品的专家们，深入讨论数字化金融转型、区块链、大模型应用、AI 与大数据、支付创新、智能投顾、风险与合规技术等热门议题。

2023-09

- 9.30 2023 下半年可信云·云原生中间件系列评估正式启动（9.30 截止）

  云原生中间件成熟度评估依托云原生中间件的通用成熟度模型，从弹性、可观测性、可移植性、可维护性、高可用性、安全性、开放性七个维度衡量中间件的成熟度；云原生平台中间件管理能力评估主要从接入管理、生命周期治理、运维支撑、运营服务、高可用、兼容性、开放性、安全性八个部分对中间件管理能力进行考察。

- 9.26 ONE Summit Regional Day 上海站将于 9.26 举行

  今年的 ONE Summit Regional Day 与 KubeCon + CloudNativeCon + Open Source Summit China 2023 同场举办，主要聚焦人工智能、网络

接入、边缘计算、5G 和开源云技术发展等区域性交流活动。活动议题将在本月末公布。

- 9.23–9.24 共见开源新未来，开源创新汇（GOSIM2023）重磅来袭

  本届 GOSIM 大会设置 GOSIM 工作坊和 2023 GOSIM 开源创新会。9 月 23–24 日的 GOSIM 工作坊，面向开发者，将分享最新开源技术趋势和发展动向；9 月 26 日的 GOSIM 开源创新会，面向用户、战略师、CxO、架构师、开源专家等嘉宾，旨在推广草根开源项目，促进项目合作，并帮助项目成长、发展和成熟。

- 9.24 云原生社区首届社区峰会将于 9 月 24 日在上海举行

  云原生社区将于 2023 年 9 月 24 日，也就是 KubeCon China 2023 的前一天，在上海举办首届的云原生社区峰会。预计本次峰会的规模将达到 200 人左右。目前正在征集峰会的赞助商。讲师议题征集也将在不久后启动。

- 9.15 2023 下半年【可信云·云原生技术架构成熟度评估】报名开启（9.15 截止）

  云原生能力成熟度体系包括技术架构、业务应用、安全、中间件、大数据五个部分。云原生技术架构成熟度模型聚焦企业云原生底座能力，涵盖资源管理、运维保障、研发测试、应用服务四个能力域。评估结果分为成熟度 1–5 级，分别为初始级、基础级、全面级、优秀级以及卓越级。

- 9.15 第八届中国开源年会（COSCon’23）讲师征集令（9.15 截止）

  COSCon’23 将于 10 月 28~29 日在四川成都召开。本次大会将持续两天，计划的主题包括：人工智能、开源商业、云计算、大数据、区块链、物联网、开源教育、开源文化、开源治理、开源硬件、开源操作系统、编程语言、女性论坛、开源公益、Web 应用开发、开源百宝箱以及闪电演讲等方向。

- 9.10 关于发起“2023 边缘计算产业图谱”征集通知（9.10 截止）

  边缘计算产业图谱是每年下半年边缘计算领域自发传播最广的宣传活动，边缘计算社区现征集企业申报“2023 边缘计算产业图谱”，最终完整版图谱会收录在边缘计算社区主导编写的边缘计算行业报告中，并在 2023 全球边缘计算大会·上海站上发布。

- 9.9 EdgeX+OpenVINO 开发者生态大会将于 9.9 在深圳举行

  本次活动邀请到来自英特尔、联想研究院、Linux 基金会、阿里云、百度智能云等十数位边缘计算及人工智能领域的研发专家，分享边缘计算技术和产品在各领域的最新实践及应用、展示最新的边缘计算产品 Demo 等。

- 9.6–9.7 WasmCon 2023 会议议程已发布

  为期两天的 WasmCon 旨在为与 WebAssembly 相关的技术人员提供最新信息和趋势，包括用例、最佳实践、深入解析和新兴技术。

2023-08

- 8.27 亚马逊云科技上海 Community Day 社区嘉年华活动议程发布

  本次 Community Day 活动的主会场主题分享重点聚焦 AIGC 和云原生两大技术话题，还包括两个分会场：云原生 Codelab 区、数据安全合规分会场。目前，活动的合作招募、志愿者招募以及 Free Talk 快速演讲环节报名正在进行中。

- 8.23 关于申报“2023 年中国软件和信息技术服务业企业信用评价”的通知

  中国软件行业协定期向各地经信、工信主管部门推送信用评价良好的企业名录，AAA 级企业在申报国家有关部委的扶持资金、奖项荣誉时将被中国软协优先推荐。连续两次获评 AAA 级企业将被中国软协授予“中国软件诚信示范企业”称号，并将名单报送工信部信发司备案，获得重点宣传的机会。

- 8.20 CommunityOverCode Asia 云原生专场将于 8.20 在北京举行

CommunityOverCode Asia 2023（原 ApacheCon Asia）的云原生专题将介绍 Apache 项目（包括 SkyWalking、APISIX、Kafka、ShardingSphere 等）在云原生环境下的实践，以及云原生环境下的可观测性经验分享。

- 8.20 中国软件行业协会开展优秀工业软件案例征集活动（8.20 截止）

  中国软件行业协会面向社会开展“软件赋能制造业升级”优秀工业软件案例征集活动，对外公开征集 2023 年工业软件在推进制造业数字化升级的优秀应用实践案例。重点征集研发设计、生产制造、经营管理、运维服务等领域的工业软件优秀案例。

- 8.18–8.19 AI+软件研发数字(AiDD)峰会 将于 8.18–8.19 在北京举行

  本届峰会以“AI 驱动软件研发全面进入数字化时代”为主题，聚焦十二个主题论坛和一个企业专场，策划 60+ 精彩议题内容，包括知识工程、数字化研发、AI 原生开发、代码生成与重构、基于 AIGC 的测试生成等。

- 8.19 云原生社区成都站 Meetup 火热开启

  云原生社区 Meetup 成都站将邀请来自华为、Flomesh、蚂蚁集团的资深技术专家带来云原生高性能之 Rutt & C++ 领域的精彩分享。

- 8.16 CNCF 成立最终用户技术咨询委员会

  最终用户技术咨询委员会由来自最终用户成员的代表组成，将作为最终用户与各种 CNCF 治理机构之间的桥梁，包括治理委员会（Governing Board，GB）和技术监督委员会（Technical Oversight Committee，TOC）。

- 8.16 新加坡 2023 国际开源节云原生技术沙龙开启报名

  2023 年 8 月 16 日，字节跳动开源将联手 CNCF 基金会、云原生社区、2023 新加坡国际开源节等举办一场聚焦云原生开源项目的技术沙龙。

本次技术沙龙将分享当代企业需求背景下，如何满足企业上云、降本增效、环境隔离、多云架构的需求。

- 8.15 2023 LF AICON 中东峰会，面向全球征集分享议题（8.15 截止）

  2023 LF AICON 中东峰会由 LF AI&DATA 基金会主办，将于 10 月 8–9 日在沙特举行。拟定专题论坛包括 AIGC（ChatGPT）、MLOps、AI 安全、大语言模型、AutoML、隐私计算。

- 8.13 中国信通院《边缘计算产业发展研究报告（2023 年）》持续征集参编单位

  为更进一步对我国边缘计算产业发展进程、关键技术、应用实践、挑战机遇等进行详细梳理、总结实践经验，中国信通院正式启动《边缘计算产业发展研究报告（2023 年）》编制工作，并面向社会公开征集参编单位。

- 8.11–8.12 SECon 全球软件工程创新峰会将于 8 月 11–12 日在上海举行

  SECon 大会涵盖了一场主论坛和十五场垂直分论坛，分论坛邀请了来自腾讯、网易、阿里、百度和各大银行等一线专家带来 60 场深度技术演讲，演讲话题覆盖：云原生架构、大数据主流框架与实践、AIGC、平台工程、Java 创新与实践、数字化转型、金融级架构等。

- 8.10 中国信通院 2023 开源尖峰案例评选正式启动（8.10 截止）

  开源尖峰案例评选共分为：“开源人物”“开源项目”“开源社区”“开源企业”“开源技术创新（商业产品）”五个领域。评选结果将在 2023 年 9 月 21 日“OSCAR 开源产业大会”上公布。

- 8.7 CNCF 发布云原生安全白皮书中文版第二版

  《云原生安全白皮书》中文版第二版介绍了在云原生开发的不同阶段：“开发”、“分发”、“部署”和“运行时”，实施云原生安全的相关概念、工具和过程，以及云原生运行时环境的安全保护等。

- 8.5 议题征集 | KubeSphere Meetup 上海站将于 8 月 5 日开启

  本次 Meetup 将关注 AI + Serverless，但也欢迎与 KubeSphere 及云原生相关的议题。本次 Meetup 继续采用公开招募的方式，来招募讲师和议题。此次活动包括闪电演讲（10 min）、单人/双人主题演讲（40 min）两种形式。

- 8.2 全球云原生技术顶会 KubeCon China 2023 详细议程公布

  本次峰会将汇聚近百位来自全球的技术专家和行业领袖，带来上百场主题演讲，涵盖供应链安全、可观测性、安全性、平台工程、开源硬件、操作系统、数据+处理+存储、数据库、新兴和先进技术、服务网格、运维+性能、SDLC（软件开发生命周期）、网络+边缘计算、网络+边缘+电信、Open AI+数据等领域。

2023-07

- 7 月“云服务稳定运行能力标准体系”首评报名通道开启（7 月报名截止）

  评估面向云服务经营者开放云主机、云存储、云网络服务、消息中间件、容器服务、分布式缓存、内容分发 CDN、融合云 DNS 服务等 8 项云服务稳定运行能力展开首批评估。首评结果将于“2023 年可信云大会”公布。

- 7.31 中国电子信息行业联合会启动“2023 年软件和信息技术服务名牌企业”申报工作（7.31 截止）

  申报工作面向从事基础软件、工业软件、应用软件、新兴平台软件和嵌入式软件等领域的企业。申报企业要求在本年度内有突出贡献，如拥有关键核心技术和产品，有效解决了行业痛点难点等。最终结果将在 8 月 31 日在天津举办的第二十五届中国国际软件博览会上发布。

- 7.25 国内首个《云原生安全配置基线规范》标准正式发布

  《云原生安全配置基线规范》中的云原生安全配置基线要求是对云原生工具 Kubernetes 的安全配置基线的要求，主要包括对 Kubernetes 主节点

API Server、控制管理器、调度器、etcd 的安全配置要求，对工作节点 kube-proxy、kubelet、工作负载的安全配置要求，以及针对附加项的 CNI 和网络策略配置的要求。

- 7.25 DaoCloud 入选《混合云产业全景图（2023）》

  《混合云产业全景图（2023）》基于《混合云产业全景图（2021）》进行更新，靶向混合云网络、安全、服务提供、应用、管理、行业建设等多维厂商能力实现全景产业展示。其中 DaoCloud 道客 入选全景图的混合云 PaaS 板块。

- 7.25 中国信通院发布《云计算白皮书（2023 年）》

  这是中国信息通信研究院第九次发布云计算白皮书。本次白皮书聚焦过去一年多来云计算产业的新发展新变化，总结梳理国内外云计算政策、市场、技术、应用等方面的发展特点，并对未来发展进行展望。

- 7.24 中国信通院已开放信创兼容性评估申报

  中国信通院云计算与大数据研究所对于信息系统在自主基础软硬件环境下的兼容性应用等典型问题，基于兼容性、性能、可靠性、安全性等质量评估维度，发布服务器虚拟化、虚拟化云平台、容器平台、云管理平台、桌面云等多项兼容评估标准。目前已开放兼容性评估申报。

- 7.22 上海 DevOps 社区第 14 届 Meetup 线上活动报名正式开启

  上海 DevOps 社区第 14 届 Meetup 将于 7.22 在线上举行，邀请了来自华为云、顺丰科技分享有关研发组织效能提升的议题。

- 7.21-7.22 2023 ArchSummit 全球架构师峰会·深圳站演讲话题招募

  ArchSummit 全球架构师峰会是 InfoQ 中国团队推出的重点面向高端技术管理者、架构师的技术会议。本次深圳站峰会将设置微服务架构、可持续软件架构、ChatGPT + 大数据探索、金融业数字化转型探索、业务出海等专题。每个演讲时长 45 分钟，包括最后 5 分钟的 Q&A 提问环节。

- 7.20 IstioCon 2023 提案征集开启（7.20 截止）

  IstioCon 2023 将于 9.25–9.26 在上海举行。推荐议题内容包括：案例研究、新功能、Istio 配置及项目更新等。演讲类型包括单人演讲（30 分钟）、双人演讲（30 分钟）和小组讨论（30 分钟）三种形式。本次峰会支持现场演讲和线上演讲。

- 7.20 2023 年 NJSD 全球软件大会 &IAS 全国互联网架构峰会会议话题征集

  本届 NJSD 大会内容将于 10 月 20–21 日在南京举行，聚焦大模型训练以及生成式 AI 的应用开发，包括其中的技术实践和正在发生变化的软件工程。IAS 峰会将于 11 月 25 日在南京举行，主题为平台架构技术与工程实践。

- 7.18 《中国 DevOps 现状调查报告（2023）》正式发布

  调查报告显示：六成国内企业达到 DevOps 成熟度全面级，60% 受访企业已进行流水线建设，大部分项目的缺陷逃逸率集中在 2%~5.9% 之间，实践 DevSecOps 的企业首次突破七成。

- 7.16 2023 可信云大会·云原生技术与实践分论坛将于在 7.26 在北京举行

  2023 可信云大会将于 2023 年 7 月 25–26 日在北京国际会议中心举行。本届大会特设“云原生技术与实践分论坛”，论坛将呈现应用现代化建设方法论的分享，发布 2023 中国云原生中间件用户调查结果，以及云原生技术实践最佳路径等内容的探讨。

- 7.16 国内首届平台工程技术峰会将于 9 月 2 日在北京举行，演讲征集已开启（7.16 截止）

  国内首届平台工程技术峰会设置 10 场演讲，涵盖云原生、DevOps、平台工程以及研发效能等议题，将邀请技术专家到现场分享技术洞察和落地实

践。推荐的演讲议题包括：云原生技术的前沿趋势、平台工程的案例分享、研发效能提升实践、云原生赋能平台工程、DevOps 落地实践。

- 7.15 Harbor & Narrows & Velero 开源项目技术沙龙将于 7 月 15 日举行

  本次云原生技术沙龙由 VMware 主办，聚焦于 Harbor、Narrows、Velero 三个开源项目的技术分享活动。本次活动汇集了来自 VMware、蚂蚁集团、微软、骥步科技、雅克云的多位云原生领域技术专家现场分享全栈式的开源项目讲解和云原生应用的落地与实践。

- 7.15 第七届全球边缘计算大会将于 7.15 在深圳举行

  大会邀请了边缘计算领域比较活跃的行业意见领袖和甲方企业，分享超过 20 个议题，讨论边缘计算的落地实践、应用案例、边缘云、AI、开源、算力、芯片、音视频、出海、工业物联网、投资等内容。

- 7.15 IDC 启动云原生基础设施、低代码/无代码两大报告研究（研究合作申请 7.15 截止）

  云原生基础设施技术能力评估报告研究将深入评估市场上各种云原生基础设施解决方案的关键能力和特点，并洞察其在运行时、云原生存储、云原生网络、镜像、编排调度等维度的优势和挑战。低代码/无代码开发平台厂商评估将结合能力与战略两个维度对市场上主流低代码/无代码开发平台厂商进行全面评估，探索在云原生开发模式下的最优解。

- 7.15 云原生开源项目应用实践专场第三站报名开启

  7 月 15 日，云原生开源项目应用实践专场」第三站来到深圳。「DaoCloud 道客」携手腾讯云 | 腾源会、Microsoft、OpenCloudOS 社区、WasmEdge 社区、DCE 社区，分享最前沿的云原生技术、开源项目、解决方案、典型案例等。

- 7.14 金融级、电信级、泛行业类《行业云平台运营管理能力成熟度 第 1 部分：租户运营》首批评估开放报名（7.14 截止）

行业云平台运营管理能力成熟度 第 1 部分：租户运营》主要从企业资质、租户运营管理能力、租户技术支持三个维度，针对金融级和电信级场景，对行业云平台运营方对租户提供的运营能力进行考察。

- 7.12《IDC Perspective：中国云备份市场分析》和《IDC Perspective：中国云存储市场分析》两项研究正式启动

  云备份市场分析和云存储市场分析将从云平台用户的业务需求出发，对市场现状和发展趋势，以及主流厂商的产品和解决方案进行综合分析，为云备份和云存储市场总结核心观点，同时为技术买家提供行动建议，另外还会对行业最佳实践和主流厂商核心能力进行描述和展现。

- 7.10 《IDC 中国容器市场份额，2022》研究报告正式启动

  当下的中国容器市场空间有多大？不同玩家为市场带来了哪些变化？容器市场未来将如何演变？技术买家们如何看待容器市场？《IDC 中国容器市场份额，2022》报告将解读这些问题。

- 7.6–7.8 2023 世界人工智能大会将于 7 月 6 日–8 日 在上海举行

  本届大会聚焦科学前沿和产业发展，围绕技术、产业、人文三大话题，重点关注大模型、智能芯片、科学智能、机器人、类脑智能、元宇宙、自动驾驶、数据论坛、法治与安全、区块链等十大前沿风向。

- 7.7 中国信通院“云优化能力提升计划”第四批成员单位招募开启（7.7 报名截止）

  “云优化能力提升计划”成员单位招募成员单位可参与标准、研究报告、白皮书等团体文件研讨与编写，可获得阶段性工作文件和正式发布的文件，参与提升计划组织的各类沙龙、培训、产业大会等活动，且在各类活动中享有优先展示机会。第四批成员单位名单将在 7 月 25 日可信云大会正式公布。

- 7.3 中国信通院正式启动《分布式存储产业发展白皮书 2023》编写工作

《分布式存储产业发展白皮书 2023》将重点讨论分布式存储行业发展环境、分布式存储技术研究路径、分布式存储技术应用场景分析、面临的挑战与应对策略、行业发展趋势以及优秀应用案例分享。目前，参与白皮书编写工作的单位及专家征集已开启。经审核通过后，合格的单位和专家将有机会参与白皮书的编写和研讨工作。

2023-06

- 6.30–7.1 稀土开发者大会 2023 将于 6.30–7.1 在北京举行

  本届稀土开发者大会邀请了来自字节、腾讯、Intel、阿里、B 站、小红书等公司的行业专家做演讲分享，涵盖「大模型与 AIGC、前端未来、低代码技术实践和创新应用、云原生实践、前端工程实践、分布式云和边缘计算、可观测性探索、架构设计与演进、音视频技术前沿、研发效能与团队管理、出海业务探索」等 12 个分论坛。

- 6.30 《IDC Perspective: 中国容器安全市场洞察，2023》报告研究正式启动（6.30 申请截止）

  为了进一步展现 2023 年的中国容器安全产品市场，IDC 正式启动《IDC perspective：中国容器安全市场洞察，2023》报告，展现安全技术提供商在容器安全市场的表现，并探讨容器安全技术发展现状和未来趋势。

- 6.30 《IDC MarketScape：中国混合云基础架构厂商评估》研究正式启动（6.30 申请截止）

  IDC 将从不同层次技术视角出发，提供基于公有云、私有云/超融合、混合云/边缘计算部署形态的基础架构服务、基础架构平台、安全、监测与管理的战略和能力维度的评估内容，从而分析整个市场不同厂商市场地位和产品技术。IDC 将基于对该市场的分析，分别选择 8–10 家市场中主流厂商作为本次厂商评估的参与者。

- 6.30 2023 云边协同大会将于 6.30 在北京举行

第三届“云边协同大会”由中国信息通信研究院、中国通信标准化协会主办，以“云智物联，边筑算新”为主题，聚焦分布式云、边缘计算、AIoT 平台等领域前沿焦点。届时将发布《边缘计算产业发展研究报告（2023）》、《边缘计算产业全景图（2023）》等研究成果，以及《边缘计算技术创新与最佳实践先锋案例（2023）》等创新实践和“可信边缘计算”最新评测结果。

- 6.30 工业和信息化部办公厅关于组织开展 2023 年新一代信息技术典型产品、应用和服务案例遴选工作的通知

  典型产品名录征集工业软件优秀产品；典型应用案例征集工业互联网平台、区块链、企业上云用云等案例；典型服务模式征集校企人才实训服务、标准化工作站等案例。典型案例申报主体包括制造企业、信息技术企业、互联网企业、科研院所、高等院校或其联合体等。

- 6.30 中国信通院启动 2023 上半年智慧营销数字化服务能力评估报名（6.30 报名截止）

  智慧营销数字化服务能力评估包括六个模块：数字化场景、数字化治理、平台化产品、数字化服务运营、可信兼容生态和数字信任安全服务能力。其中，平台化产品能力重点考察是否具备客户洞察、营销策划、客户触达、客户转化、客户运营等智慧营销全链路环节能力。

- 6.28 《2022–2023 容器网络使用情况调研报告》发布

  报告显示，容器运行在公有云和虚拟机中的比例仍然最高，但同时运行在裸金属上的比例逐年增加，从 45% 提升至 50%。在开源容器网络方案中，Calico 的采用率最高。此外，在生产环境中使用 IPv6 或双栈容器网络的用户大幅增加，达到了 30.21%。

- 6.28 2023 年 CCF 中国软件大会专刊征稿火热进行中

  CCF 中国软件大会（CCF ChinaSoft）由中国计算机学会（CCF）主办，是中国软件科学与工程相关领域办会规格最高、参会人数最多、影响范围最

广、涵盖内容最全面的全国性学术会议。目前，《软件学报》、《计算机学报》等期刊的大会专刊征稿正在火热进行中。

- 6 月中下旬 中国信通院正式启动上半年一云多“芯”系列标准首批评估（6 月中下旬报名截止）

  本次评估的评估项目包括：一云多芯 IaaS/PaaS 平台能力评估、一云多芯 IaaS/PaaS 平台性能评估、一云多芯云管平台能力评估。评估结果将于 7 月可信云大会上发布。

- 6.27-6.28 亚马逊云科技中国峰会将于 6.27-6.28 在上海举行

  2023 亚马逊云科技中国峰会特设下一代云基础架构、高性能计算、云原生应用、云网络与云边缘四大技术论坛。在云原生应用论坛中，将介绍亚马逊云科技如何为大规模容器集群管理提供安全可靠和轻松扩展的体验，并将详细介绍亚马逊云科技 Serverless，容器等核心云原生应用开发服务。

- 6.25 上海市工业互联网专业服务商推荐目录申报工作（6.25 申报截止）

  申报工作聚焦先进材料、电子信息、生物医药、能源、时尚消费品、重大装备、汽车等重点产业，支持面向全国及长三角区域，提供云网服务、标识解析服务、咨询服务、工业数据服务、系统集成服务、工业信息安全服务、工业软件服务等第三方数字化转型专业服务的服务商。

- 6.25 可信云·云原生安全托管服务（MSS）能力首批评估正式启动（6.25 报名截止）

  云原生安全托管服务（MSS）能力评估包括服务运营能力、平台能力、服务保障能力、云数据接入能力、规范要求、人员能力六大基本部分，以及环境适配能力、本地接入能力两大可选部分，从服务的安全运营、事件管理、脆弱性管理等多个维度对云原生安全托管服务进行全面评估。

- 6.25 Linux 基金会正式启动 WasmCon 2023 议题征集（6.25 截止）

WasmCon 2023 是 Linux 基金会面向 WebAssembly 开发者和用户推出一个新的活动。该活动将于今年 9 月 6 日至 7 日在美国华盛顿举行。提交的议题主题可包括：Wasm 介绍、Wasm 性能优化、Wasm 安全、Wasm 和云、使用案例、调试 Wasm 应用程序、Wasm 工具和生态系统等等。

- 6.19 KubeDay 以色列站将于 6 月 19 日举行

  KubeDay 是 CNCF 主办的地区性云原生系列活动，主要针对社区发展活跃和对 Kubernetes 等云原生技术感兴趣的地区。 继去年 12 月在日本拉开帷幕后，KubeDay 第二站来到以色列特拉维夫。演讲提案申请现已开放（截至 4.30），演讲者需亲临现场。

- 6.17–6.18 RustChinaConf 2023 会议议程发布

  本次大会将于 6.17–6.18 在上海举行，共设两个主论坛（上午），六个分论坛（下午）。六个分论坛围绕 Rust 语言和高性能实践、数据库、嵌入式与操作系统、云原生、区块链、密码学和安全、WebAssembly 等内容。

- 6.18 KubeCon + CloudNativeCon 北美 2023 开启议题征集（6.18 截止）

  KubeCon + CloudNativeCon 北美 2023 将于 11.6–11.9 在芝加哥举行。本次峰会对云原生新手、应用交付和 CI/CD 结合、平台工程、安全、网络+边缘+电信等主题进行了更新和扩充。议题类型包括五类：单人演讲（30 min）、双人演讲（30 min）、小组讨论（30 min，3 至 5 名演讲者）、闪电演讲（5 min）以及教程（90 min）。

- 6.18 2023 年上海 KubeCon + CloudNativeCon 或 开源峰会开启议题征集（6.18 截止）

  本次峰会将于 9.25–9.27 在上海举行，提交的议题内容可围绕软件开发生命周期、平台工程、运维+性能、安全性、数据+处理+存储、网络+边缘+电信、服务网格等话题，议题类型包括四类：单人演讲（30 min）、双人

演讲（30 min）、小组讨论（30 min，3 至 5 名演讲者）、教程（90 min）。

- 6.18 2023 可信云技术/行业服务/用户最佳实践征集启动（6.18 报名截止）

  本次报名面向所有通过和正在参评可信云的服务商及用户企业，所有类别每家最多可以申报四个选项。征集范围分为技术最佳实践、行业服务最佳实践、用户案例最佳实践三类。

- 6.17 KCD 北京站将于 6.17 举行

  大会邀请了来自「DaoCloud 道客」、微软、阿里云、华为云、IBM、Intel、骥步科技、Flomesh、VMware 等企业的云原生技术专家探讨服务网格、供应链安全、可观测性、机器学习、Kubernetes、eBPF、多集群管理技术等内容。

- 6.15 & 6.23 2023 IDC 全球金融科技厂商排名及真实价值案例评选正式启动

  金融科技厂商排名评选分为金融科技厂商（Top 100）及综合行业 IT 服务厂商（Enterprise 25）两个类别，报名截止时间 6 月 23 日。金融行业真实价值案例评选该评选由厂商单独，或联合其行业用户共同申报，截止时间为 6 月 15 日。根据金融科技服务商为金融机构服务的案例实践，对八个金融服务类别进行评估并选出入围案例，并且根据这八个类别的整体评估，评选出整体数字化转型卓越奖。

- 6.15 2023 中国云原生中间件用户调查正式启动

  为了进一步调研产业现状，探明新周期下的云原生中间件的新特征、新变化、新需求、新趋势，中国信通院正式发起 2023 中国云原生中间件用户调查。调查截止时间为 2023 年 6 月 15 日。

- 6.15 2023 年度中国 DevOps 现状调查优秀实践案例征集正式启动（6.15 申报截止）

  本次 DevOps 优秀实践案例征集活动面向国内各企事业单位，每家单位最多申报 2 个案例，可由甲乙方单位联合申报。征集案例包含 11 个领域，可任选其一或跨领域申报：敏捷开发、持续交付、技术运营、持续测试、应用设计、安全及风险管理（DevSecOps）、研运效能度量、BizDevOps、系统和工具、平台工程、自主可控工具链。

- 6.10 EdgeX+OpenVINO 开发者生态大会将于 6.10 在上海举行

  本次大会将汇聚来自道客云、阿里云、英特尔、联想、VMware、中科创达、EMQ 等边缘计算和视觉推理两大领域的顶尖专家和创新企业。聚焦于边缘计算和视觉推理的最新进展和创新应用，探讨如何利用 EdgeX 和 OpenVINO 技术推动智能边缘的发展，从而实现更智能、更高效的边缘计算应用。

- 6.6 云原生产业发展论坛完整议程公布

  ICT 中国·2023 高层论坛–云原生产业发展论坛由工业和信息化部主办，中国信息通信研究院等承办的，将聚焦主题“云智原生新底座，数实融合新征程”。届时将发布一系列云原生技术服务评测及优秀案例结果、白皮书等，并对云原生产业发展展开解读。

- 6.6 ASF Conference Asia 2023 开启议题征集（6.6 截止）

  ASF Conference Asia 2023 将于 8.18–8.20 在北京举行，本次峰会将围绕 AI/机器学习、API/微服务、社区、云原生、数据存储/计算、性能工程等主题展开。

- 6.6 ICT 中国·2023 高层论坛–云原生产业高峰论坛将于 6 月 6 日在北京拉开帷幕

云原生产业高峰论坛由工业和信息化部主办，将邀请云原生领域领军智库、政府领导、院士专家、企业高层等重量级嘉宾共话云原生生态，共探云原生未来。

- 6.5 Gartner 发布首份《DevOps 平台魔力象限报告》

  除了公有云厂商（Microsoft、Amazon Web Services、Google Cloud Platform）和传统 DevOps 厂商（Gitlab、Atlassian），云原生厂商（Red Hat、VMware、灵雀云 Alauda）也在报告中崭露头角，这标志着云原生已成为构建下一代 DevOps 价值链平台的核心技术。



devops

- 6.5 IDC 发布《中国软件定义计算软件市场跟踪，2022》报告

IDC 定义的软件定义计算软件市场（主要包括虚拟化软件、云系统软件和容器基础架构软件三个子市场。在 2022 容器基础架构软件市场中，收入排名前五的厂商中有三家是提供公有云服务的厂商（华为、阿里巴巴和腾讯）。IBM/红帽由于在该市场拥有比较大的影响力和丰富的产品组合，其收入在该市场位列第三。博云的收入位列第五。

- 6.5 关于征集 2023 年度上海市软件行业协会团体标准立项申报的通知

  团体标准聚焦新产品、新技术、新业态和新模式，包括产品标准、技术标准、管理标准、服务标准等；围绕行业发展迫切需要和参与国内外市场竞争亟需为主要方向；鼓励成熟的企业标准项目升级转化为团体标准。团体标准申报为全年征集。

- 6.3 云原生社区重庆站 Meetup 将于 6 月 3 日举行

  此次 Meetup 将分享的内容有：如何使用开源软件 pipy 构建轻量级、高性能、高可靠性 web 应用程序；开源软件技术在企业数据解决方案中的应用；如何使用 OpenTelemetry 监控和跟踪云原生应用程序在多个集群中的性能和行为；ChatGPT 和 AIGC 技术在未来的潜力。

- 6.3 KubeSphere 社区云原生技术 Meetup 杭州站火热开启

  此次 Meetup 由 KubeSphere 社区联合极狐 GitLab 及 Curve 社区共同组织，包括 4 个分享以及 1 场 WorkShop，涉及云原生安全、云原生存储、服务网格等话题。

- 6.3 KCD 北京站开放议题征集（5.8 截止）

  本次 KCD 北京站活动由 CNCF 和来自「DaoCloud 道客」、Microsoft、Tetrate 的技术爱好者联合发起，包括闪电演讲（10min）、单人/双人演讲（40min）和圆桌讨论（30min）等三种分享形式。议题内容可包括 eBPF、WASM、Multi Cluster、可观测性、软件供应链安全、Serverless、DevOps、HPC、Streaming 等热点领域。

2023-05

- 5 月底 2023 年可信云首批性能专项测试启动（5 月底报名截止）

  可信云性能专项测试涵盖基础资源类、平台类、工具类共 3 大类，包括专有云、容器平台、信创云管平台、混合多云管理平台等 13 项云服务性能测试项目，同时可提供场景化、定制化的性能测试的能力。

- 5.31 中国信通院启动大规模边缘容器集群服务质量首批评测（5.31 截止）

  评测体系包括节点管理指标体系和应用分发指标体系。节点管理重点考察集群节点的管理性能、稳定性、效率等指标；应用分发重点考察应用副本创建不同阶段的功能和性能，以及对应用的综合管理能力。

- 5.30 《云管理产品与服务图谱（2023）》正式启动征集工作（5.30 申请截止）

  《云管理产品与服务图谱（2023）》征集范围包括解决方案和产品服务两大类。 其中，解决方案分为行业解决方案、智慧应用解决方案和上云通用解决方案；产品服务分为咨询规划、迁移实施、运行管理、增值开发、优化治理。

- 5 月底 中国信通院正式启动 2023 年上半年云咨询、云迁移、云管理和云容灾评估（5 月底报名截止）

  云咨询评估体系旨在评估在企业上云和数字化转型方面提供的服务能力；云迁移评估体系旨在评估云迁移平台能力和服务能力；云管理评估体系评估主要评估企业级一体化云管平台能力、一云多芯云管平台能力、多云管理平台方案；云容灾评估体系旨在评估云多活、云灾备、云双活容灾方案。

- 5 月底 中国信通院正式启动 2023 年上半年混合云、专有云评估（5 月底报名截止）

  可信云专有云评估旨在评估参评单位的专有云能力综合水平（包括自建模式和专用模式）、成熟度（通用）、行业能力（金融）以及专项能力（兼容性）。 混合云评估值在评估参评单位的基础平台能力（包括公有云部分、私有云部分、超融合平台、信创兼容）、专项能力（安全）、应用场景（数据分析、研发运营）以及综合能力。

- 5.26 SRE 研发运营系统可靠性与连续性评估报名正式开启

评估由中国信通院工程师在参评机构现场完成，主要分为研发过程的可靠性与连续性保障能力与技术运营过程的可靠性与连续性保障能力两大部分。评估结果依据能力成熟度分为初始级、稳健级、优化级、引领级、卓越级五个级别。

- 5.26 《云服务稳定运行能力标准体系 第五部分 云网络服务稳定性》首批测评正式开启

  评估面向云网络服务提供商，借助混沌工程技术，从网络产品服务稳定性与专项能力两个方面，综合评估服务商提供的云网络服务的稳定性等级。

- 5.19 中国信通院企业数字化发展共建共享平台开启“数字化业务安全生产推进中心”最新一批成员招募（5.19 招募截止）

  “数字化业务安全生产推进中心”的主要工作内容是提升数字化业务安全生产相关的风险识别预防、业务全局感知、实时监测预警、故障快速定位、事故应急处置能力， 规范各行业企业数字化业务安全生产制度流程的规范性、科学性。

- 5.18 全球分布式云大会 Distributed 创新企业榜单火热征集中

  Distributed 创新企业榜单以【分布式云十大技术趋势雷达图】为根基，全面围绕十大关键词：分布式云、分布式存储、分布式数据中心、分布式数据库、分布式算力、分布式网络、分布式创新，评选出在该领域有突出贡献和影响力的企业。申报企业需要提供 2~3 个实践案例，对产品或服务进行介绍。

- 5.18 标准参编单位征集 |《DevOps 系列标准第 13 部分：平台工程能力要求》（5.18 征集截止）

  《研发运营一体化（DevOps）能力成熟度模型 第 13 部分：平台工程能力要求》是首个针对 DevOps 平台工程能力成熟度的标准，此标准现开放征集参编单位及专家。

- 5.16 中国信通院启动“云原生产品目录 2.0”征集（5.16 截止）

  “云原生产品目录 2.0”的征集范围包括基础设施类、云原生中间件类、云原生数据类、云原生安全类、云原生融合服务类等。评选结果将于 6 月 6 日在 ICT 中国·高层论坛–云原生产业高峰论坛。

- 5.15 KubeSphere 社区开启 5.27 云原生 Meetup 杭州站议题征集（5.15 截止）

  本次云原生 Meetup 由 KubeSphere 社区与极狐 GitLab 共同组织，暂定于 5.27 下午举行，包括闪电演讲（10 min）、单人/双人演讲（40 min）两种形式。

- 5.15 《云原生应用引擎技术发展白皮书》重磅发布

  白皮书从云原生应用引擎的定义、产品形态、行业应用场景及未来趋势等多个维度，深度剖析了我国云原生应用引擎技术发展的现状和未来，以及解读如何通过应用引擎实现云原生“根”技术上的突破，并驱动云原生产业生态的建设。

- 5.12 中国信通院正式启动上半年面向云计算存储类的首批评估

  本次评估的评估项目包括：分布式存储平台建设成效评估、超融合平台建设成效评估、超融合平台在云原生及信创场景的落地成效评估。评估结果将于 7 月可信云大会上发布。

- 5.12 2023 可信边缘计算创新与实践案例暨产业全景图征集启动（5.12 申报截止）

  可信边缘计算技术创新案例范围包括物联网边缘计算、边缘云、边缘原生、边缘 AI、边缘数据、管理调度、边云协同、性能 / 稳定性等全栈技术领域。 最佳实践案例范围包括边缘计算在通信、政务、金融等各行业应用建设实践。全景图范围包括边缘软件及解决方案、边缘应用、边缘开源、产业服务等边缘产业生态。

- 5.8–5.9 cdCon + GitOpsCon 北美 2023 会议议程发布

  本届会议将呈现 60 多场演讲分享，演讲内容包括但不限于： DevOps 和 GitOps 的区别、软件交付平台的安全/合规性实践、静态配置、高级部署等等。

- 5.8–5.9 cdCon + GitOpsCon 2023 议题征集中（2.10 截止）

  cdCon + GitOpsCon 2023 由 CD 基金会和 CNCF 基金会主办，将于 5.8–5.9 在温哥华举行。 议题内容可包括：GitOps 入门、扩展和管理 GitOps、从生产部署中吸取的教训、先进的交付技术、云原生环境中的开源 GitOps 和 CD 技术实践、供应链安全等。

2023-04

- 3–4 月底 中国信通院正式启动 2023 年上半年『可信云·云基础设施与云软件系列评估』

  云基础设施与云软件评估体系包括五个模块： 虚拟化领域评估（虚拟化云平台评估、私有云建设成效评估等）、超融合领域的解决方案、建设成效及应用场景评估、分布式存储领域评估、其他评估（云桌面评估、云原生 API 网关等）。

- 4 月 云成本优化系列与云资源效益度量评估正式启动（4 月报名截止）

  成本优化系列标准评估旨在对成本优化服务、原生成本优化工具和混合多云成本优化工具进行评估。 资源效益度量评估面向云平台建设方、运营方、使用方，对云资源的使用情况与效果进行评估，参评企业可以在成本、性能、安全三类云资源效益模块中选择一项或多项进行评估。

- 4.22 可观测性峰会 2023 将在北京举行

  「可观测性峰会 2023」由云原生社区组织，旨在探讨和分享有关可观测性的最佳实践， 包括在云原生应用程序和基础架构中实现可观测性的最新技术和工具，如 Prometheus、SkyWalking、eBPF 等。

- 4.21–4.22 InfoQ ArchSummit 全球架构师峰会将在上海举行，议题征集中

  本次峰会已确定的专题方向包括：金融业数字化转型探索、大数据 + 架构的结合应用、ToB 软件质量保障、企业架构演进、数字化转型下的应用现代化和架构稳定性保障。 目前，峰会的议题征集仍在进行中，欢迎经过验证的、已经落地的实践案例来会分享。

- 4.8 和 4.22 第二届开源云原生开发者日将在上海 & 北京举行，讲师征集已启动

  第二届开源云原生开发者日由微软与英特尔联合发起，将探讨规模化云原生实践案例、最新的开源云原生技术、安全软件供应链和 GitOps 四大主题。 本次活动预计将分别在上海（4 月 8 日）和北京（4 月 22 日）举行。

- 4.21 边缘原生平台技术沙龙将在北京举行

  边缘原生平台技术沙龙由 VMware 主办，是以边缘为针对场景、借助云原生技术(Kubernetes)、实现多云/跨云服务和边缘计算结合的技术分享和讨论活动。 此次活动邀请了多位边缘计算领域技术专家现场分享边缘原生应用平台、基础设施、框架、用例等相关技术话题，活动上还将首次发布 ACE 边缘原生联盟启动仪式。

- 4.20 中国信通院启动 2023 年云原生优秀案例征集（4.20 申报截止）

  本次云原生优秀案例评选共有四个大类：云原生应用优秀案例、云原生技术创新解决方案和产品、云原生安全优秀案例以及云原生领域新锐企业。 云服务提供商可以与甲方客户进行联合申报，以甲方客户的应用案例为参评主体，云服务提供商以技术提供商作为联合申报单位进行申报。

- 4.20 中国信通院正式启动行业云平台管理系列评估

  行业云平台管理系列评估包括行业云平台管理服务（租户管理） 能力评估、行业云平台管理（平台管理）能力评估以及行业云平台使用效益成熟度

评估。首批评估面向金融级、电信级、通用类行业云平台管理服务（租户管理）能力。最终结果将在 7 月可信云大会上发布。

- 4.17 InfoQ 发布《中国开源生态图谱 2023》

  图谱共计收录 913 个中国开源项目，其中，前端、人工智能和云原生是中国开源最为活跃的领域，中间件、数据库、大数据和操作系统领域项目平均得分高于整体平均得分（基于代码价值和社区活跃），Top100 项目中超过半数项目在前端、数据库和人工智能三大领域。

- 4.15–16 全球开源技术峰会 GOTC 2023 将于 4 月 在上海举行，目前演讲议题投递通道已开放

  全球开源技术峰会由上海浦东软件园、 Linux 基金会亚太区、开源中国联合发起，会议聚焦云原生、元宇宙、eBPF、Web3.0、开源治理、开源教育培训、前端、基础软件、AI、IoT 等热门话题。

- 4.15 微服务 x 容器开源开发者 Meetup 将在成都举行

  本次微服务 x 容器开源开发者 Meetup 将围绕云原生领域当下热门开源项目的技术分享和企业实践展开， 邀请到 Dubbo、KubeVela、OpenSergo、OpenYurt、Seata、Higress 等云原生开源项目的核心维护者，分享前沿技术、社区最近动态、架构实践以及企业应用案例。

- 4.15 截止：2023 开放原子全球开源峰会议题开放征集中

  2023 开放原子全球开源峰会由开放原子开源基金会发起，以“开源赋能，普惠未来”为主题，将在 6 月中旬开幕。 本届峰会聚焦全球开源生态最新发展和前沿技术动态，覆盖开源治理、开源项目、开源前沿技术、开源行业实践等多个方向。 着力展示数据库、云原生、区块链、人工智能、智能驾驶、物联网、前端技术等领域新技术，以及各领域、各行业的最新应用成果。

- 4 月 15 日 2023 年云原生社区可观测性峰会将召开，讲师及赞助商招募进行中（3.10 截止）

  云原生社区将于 2023 年 4 月 15 号（星期六）在北京举办可观性峰会。议题围绕可观测性相关领域的开源项目及应用，包括但不限于：指标监控、分布式追踪、日志收集、技术创新、应用实践等。

- 4.12 CNCF 平台工作组发布云原生计算平台白皮书

  平台实现了基础设施和应用组件的快速整合，加速应用和服务的交付。 白皮书阐述了云原生平台的性质和优势，介绍了 CNCF 项目如何作为平台的基础与平台建设结合在一起，以及平台团队可能面临的挑战以及最佳实践等。

- 4.12 IDC 正式启动云原生系列研究

  本次研究将从云原生六大领域和业务场景切入，深度剖析云原生技术，对技术发展趋势、市场空间情况、重点厂商产品和解决方案、优秀实践等开展相关研究。

- 4 月 8 日 KubeSphere Meetup 深圳站已启动

  KubeSphere Meetup 深圳站将涵盖多 Kubernetes 集群流量管理、Jenkins 实践、KubeSphere 可插拔架构等话题。此外，本场 Meetup 目前还有一个议题空余，可投递分享云原生领域或者 KubeSphere 相关的实践经验。

- 4.8 2023 国内首场 KCD 落户大连，议题征集开启（3.20 截止）

  Kubernetes Community Days（KCD）大连站由 CNCF 和来自华为云、Flomesh、极狐 GitLab 的技术爱好者联合发起， 聚焦云原生生态的开源项目与技术实践等主题分享。 活动目前暂定将于 4 月 8 日举行，议题内容包括 CNCF 生态中的任何开源项目与云原生技术实践，不局限于 Kubernetes 本身。

- 4.8 KubeSphere Meetup 深圳站，议题火热征集中（3.27 截止）

  本次 Meetup 活动包括闪电演讲（10 min）、单人/双人演讲（40 min）两种形式。 议题内容要求必须与开源和云原生生态系统相关，更欢迎与 KubeSphere 相关的议题。

- 4.6 《金融电子化》杂志社发布《银行业私有云建设发展调研报告》

  报告显示，在参与调研的 129 家银行业金融机构中，有 53.5% 采用私有云，28.7% 采用容器云。 金融基础设施建设正在跨越虚拟化、私有云、容器云三个阶段。 平滑升级能力以及“一云多芯”能力、支持信创平台建设与应用迁移是私有云建设最受关注的前两项高价值技术。

- 4.6 “2023 云成本优化能力建设·企业分享会”将在北京举行

  本次分享会将发布 2023 年首批 MSP、原生成本优化工具、混合多云成本优化工具（首批）评估结果，公布云优化工作组最新成员单位名单。 此外，还将邀请多位具有云成本优化最佳实践的企业用户和服务专家分享云成本优化相关实战经验。

2023-03

- 3 月 由 DaoCloud 道客参与编写的《多云数据存储白皮书》正式发布

  《白皮书》系统地梳理了多云时代的驱动力和面临的数据挑战，提出了多云数据存储的应对措施和发展趋势。 白皮书强调，多云数据存储的发展重点在于，扩大数据共享应用，增强数据存储智能分级能力，使能数据跨云流动，通过面向容器的存储创新助力应用上云改造，发展新型存算架构实现云计算基础架构的扁平分层和灵活组装。

- 3.27–4.2 2023“工赋上海”创新大会系列活动将于 3 月 27 日–4 月 2 日 举行

  2023“工赋上海”创新大会系列活动由上海市工业互联网协会发起，活动主题为“数智生态 融合链接”。「DaoCloud 道客」创始人兼首席执行官–

陈齐彦受邀参加 3.27“工赋上海”创新大会圆桌论坛，以《数智融合，助力新型工业化发展》为主题，探讨工业数字化转型的最佳实践，并参与 3.29 工赋创始人峰会 TED 演讲，以《为工业数字化寻找全局最优解》为主题，分享在全局视角下，对工业数字化最优解的思考。

- 3.31 全球边缘计算大会 & 边缘计算开源专场论坛将在北京举行

  全球边缘计算大会围绕边缘计算展开，主要讨论边缘云、边缘智能、算力网络、开源和 6G &前沿网络。同期举行的边缘计算开源专场论坛邀请了 Edge X Foundry、LF Edge ekuiper、KubeEdge、Akraino 、Shifu、YoMo 等开源项目核心维护者，在大会上分享他们的经验和技术。

- 3.31 中国信通院正式启动 2023 上半年“系统稳定性保障体系”评估（3.31 报名截止）

  “系统稳定性保障”评估体系包括三大模块：平台能力评估（混沌工程、可观测性、全链路压测、应用多活评测）、建设成效评估（分布式系统稳定性、证券基金期货行业重要系統稳定性等） 以及云服务稳定安全运行能力评估（消息队列、云数据库、云主机、内容分发服务、容器集群、云存储、网络服务稳定性）。

- 3.29–3.30 KubeCon 欧洲 2023 预热活动——『KubeCrash Spring 2023』将于 3.29–3.30 在线上举行

  KubeCrash Spring 2023 邀请到了来自 Xbox、波音等专家以及 CNCF 项目维护者进行演讲分享。 第一天为零信任日，主要讨论集群的安全访问、集群内的通信安全、管理 TLS 证书和策略定义等；第二天为创新日，将关注游戏、ML/AI 和其他突破性技术。

- 3.29 InfoQ 研究中心发布白皮书《中国开源生态图谱 2022——云原生领域》

  白皮书显示，目前云原生领域国内开源项目已经超过 110 个，涉及 26 家项目发起机构， 主要可以分为 7 大技术板块，分别为微服务、中间件、

服务网格、容器技术和编排、平台 & 数据库 & 数据采集、框架 & 引擎以及功能系统。 其中，微服务、容器技术与编排、服务网格等云原生内核技术领域是云原生技术体系的核心。

- 3.27 LF Research 开启「聚焦全球 2023」开源趋势调查，公开征集赞助商和合作伙伴

  Linux Foundation Research 发起的「聚焦全球 2023」项目将调查对不同地区开源参与的看法，为更好地理解和支持世界各地的开源参与提供宝贵的见解。 包括：跨组织和行业的开源项目的范围、私营和公共部门参与开源的机遇和挑战、开源的价值主张、领先的开源技术和最佳实践。

- 3 月 25 日，论道魔都！云原生开源项目应用实践专场第二站报名开启

  3 月 25 日，由「DaoCloud 道客」、观测云、 叠境数字与 DCE 5.0 社区联合承办，微软、中国移动和云赛空间倾情参与的云原生开源项目应用实践专场将来到上海。 本次活动将围绕集群管理、云原生虚拟化技术、数字孪生、可观测性等热点话题，聚焦云原生开源项目，探讨技术突破和应用实践。

2023-02

- 2.25 论道原生·云原生开源项目应用实践专场北京站将于 2.25 举行

  活动由「DaoCloud 道客」、边无际、天谋科技和 DCE 5.0 社区联合承办，与云原生领域的技术专家、最终用户和开源项目贡献者现场对话， 从云原生技术视角出发，聚焦容器管理、云原生存储、低代码开发、AIoT 等热点领域，分享云原生开源项目的最新突破和应用实例。

- 2.21 OSS Compass 开源指南针正式发布：聚焦开源生态健康评估，不再单一看 star 数

  OSS Compass 是一个用于开源生态健康评估的 SaaS 平台，面向 GitHub、Gitee 等平台所有开源项目开放。 只需输入 GitHub 或 Gitee 托管平台上的仓库名称或社区名称，即可全面展示该仓库或项目的健康状

态。 目前，OSS Compass 构建了一个包括生产力、稳健性、创新力三个维度，涵盖 14 个指标模型在内的开源生态评估体系。

- 2.8 KubeCon + CloudNativeCon 欧洲 2023 会议议程发布

  KubeCon + CloudNativeCon 欧洲 2023 将于 4.18–21 线下线上同步举行。 本次峰会共计 303 场活动、主题演讲、闪电演讲和分组会议，其中的 91 场由 CNCF 项目的维护者主持。 会议主题囊括合规性、Pod 安全、环境可持续、混合云/多云、CNCF 项目等。

- 2.6 IDC 正式启动《中国政务云解决方案厂商评估，2023》

  《中国政务云解决方案厂商评估，2023》旨在通过对国内市场上核心政务云解决方案提供商的综合评估以及大量政府客户的访谈，帮助行业全面了解政务云市场与相关技术的最新发展情况。 评估结果将在 6.31 发布，以报告形式呈现，包括评估象限图、IDC 市场观察、对技术买家的建议、入选厂商介绍、评估标准等内容。

- 2.6 中国信通院 2023 年上半年可信云·云原生安全类评估正式启动（3.10 评估报名截止）

  可信云容器类·云原生安全类评估体系由 6 个方面组成： 云原生安全成熟度评估、云原生应用保护平台（CNAPP）能力评估、容器平台安全能力评估、容器安全解决方案评估、云原生 API 安全治理能力评估、Web 应用和 API 保护（WAAP）能力评估。 评估结果将于 6 月在 ICT 中国·高端论坛发布。

- 2.2 中国信通院 2023 年上半年可信云·容器类评估正式启动（3.10 评估报名截止）

  可信云容器类评估体系由 5 个方面组成：全栈容器云解决方案评估（包含通用容器、边缘容器和混合多云容器三类）、 组件级容器平台性能评估、面向金融场景的大规模容器平台性能评估、容器平台安全能力评估、容器安全解决方案评估。 本次评估结果将于 6 月在 ICT 中国·高端论坛发布。

- 2.1 CNCF 基金会将在第三季度推出 Kubernetes 和云安全（KCSA）认证

  KCSA 认证考试旨在为希望从事云原生安全的个人提供一个切入点。 考试考查应试者制定安全策略和程序、识别评估并减少安全风险和漏洞、协助事件响应和取证调查、测试和监控安全系统等云原生安全能力。 考试系统目前处于开发阶段，预计将在 11 月北美 KubeCon 召开前普遍推出。

- 2 月 1 日至 2 日 CNCF CloudNativeSecurityCon 会议即将举行，目前会议议程已发布

  CloudNativeSecurityCon 首个围绕云原生安全项目和企业安全挑战实践的独立会议。 会议共有 72 场演讲，演讲主题包括体系结构和策略、安全软件开发、供应链安全、身份和访问、取证等。

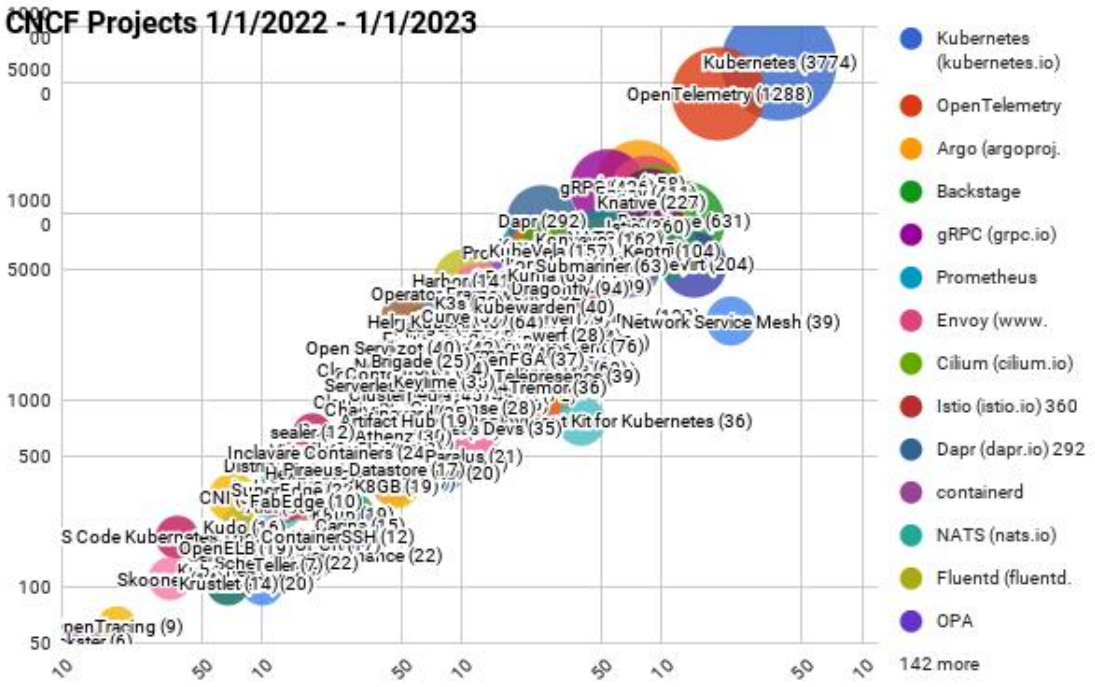
2023-01

- 1.22 开源社发布《2022 中国开源年度报告》

  2022 中国开源年度报告由四部分组成，分别是大事记篇、数据篇、商业化篇和问卷篇。 在 GitHub 数据排名 Top30 榜单中，中国顶尖开源项目在前端、数据库、云原生、人工智能领域表现突出。 此外，在开源软件商业全球化方面，搭建高活跃度的全球化开源社区、充分参与开源基金会等国际开源组织、全球化服务、全球化销售等都是可实施的路径。

- 1 月 11 日 2022 年 CNCF 和 Linux 基金会开源项目排名公布

  在 CNCF 开源项目中，Kubernetes 拥有最多的贡献者，OpenTelemetry 排名第二，而一年前没有入围的 Backstage 今年排名第四。 此外，GitOps 项目 Argo、服务网格技术 Envoy、Cilium 和 Istio 等也排名前列。 2022 年 CNCF 项目排名如下图（气泡面积与作者的数量成正比，y 轴为 pr 和 issue 的数量、x 轴是 commit 的数量）：



CNCF

- 1 月 9 日 中国信通院发布 2022 中国 Serverless 用户调查报告

  调查报告分为基本情况、函数即服务应用现状、Serverless 容器服务应用现状、Serverless 应用托管服务应用现状、总结与展望五个部分。 其中，八成用户已开始采纳 Serverless，但生产环境采纳率有待进一步提升；Serverless 距离大规模推广仍有一段距离，需要持续优化改造成本、用户习惯、启动性能等方面的问题。

- 1 月 9 日 由信通院牵头、「DaoCloud 道客」深度参与编制的《应用现代化建设指南》正式发布

  《应用现代化建设指南》系统性梳理了应用现代化的发展历程，给出了应用现代化的清晰定义，拆分了“五化两指引”建设方法论，呈现出完整的应用现代化建设路径。 「DaoCloud 道客」应用现代化部门参编专家主要贡献了应用现代化建设思路、应用架构现代化、应用现代化建设路径等内容。

- 1 月 9 日云原生产业联盟年会将在线上举行

云原生产业联盟（CNIA）是云计算开源产业联的子联盟，由国内顶尖云原生厂商、行业云原生典型用户联合组成。 会上将发布 3 大报告：《应用现代化建设指南》、《2022 中国云原生安全用户调查报告》、《2022 Serverless 用户调查报告》， 以及 5 大云原生领域的评测：存储、安全、大数据、混部、中间件。

- 1 月 6 日《混合云产业全景图（2022）》征集正式启动（1 月 31 日申报截止）

  中国信通院联合混合云产业联盟启动《混合云产业全景图（2022）》的编制工作。 《混合云产业全景图（2022）》基于《混合云产业全景图（2021）》进行更新，将根据混合云产业发展情况及技术融合情况，对全景图技术架构进行更新和完善。 通过审核的厂商将被纳入全景图并在 2023 年混合云大会上发布。

- 1 月 2 日 CNCF 即将开启 2023 年技术监督委员会（TOC）选举

  TOC 在 2023 年有一个终端用户任命的席位和三个管理委员会任命的席位空缺。 选举提名将于明年 1 月开始，候选人应该为 CNCF 社区内的高级工程师， 有必要的条件和时间可参与到 TOC 的工作中，并且能够在讨论中保持厂商或项目的中立性。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

云原生社区要闻 - 2022

本页按月份列出 2022 年云原生社区比较热门的新闻、事件、活动等，供大家一站式查阅和检索。

2022-12

- 12 月 30 日 CNCF 2022 年度报告发布

  CNCF 2022 年度报告从基金会发展趋势、活动举办、开源项目、社区生态等方面进行了总结。 其中，在 2022 年，CNCF TOC 接受了 35 个新项目；OpenTelemetry 成为 CNCF 生态中发展第二快的项目； Backstage 是增长最快的项目之一，解决了云原生开发者体验方面的痛点；Argo 和 Flux 等 GitOps 项目也在发展壮大。

- 12 月 29 日第一届混合云技术发展论坛将线上举行

  大会由中国信通院、中国通信标准化协会主办， 届时将发布《混合云产业全景图（2022 年）》框架、《混合云用户调查报告（2022 年）》、新增混合云标准等重要成果， 并邀请混合云领域的代表厂商就混合云未来的发展、技术结构及行业问题带来干货分享。

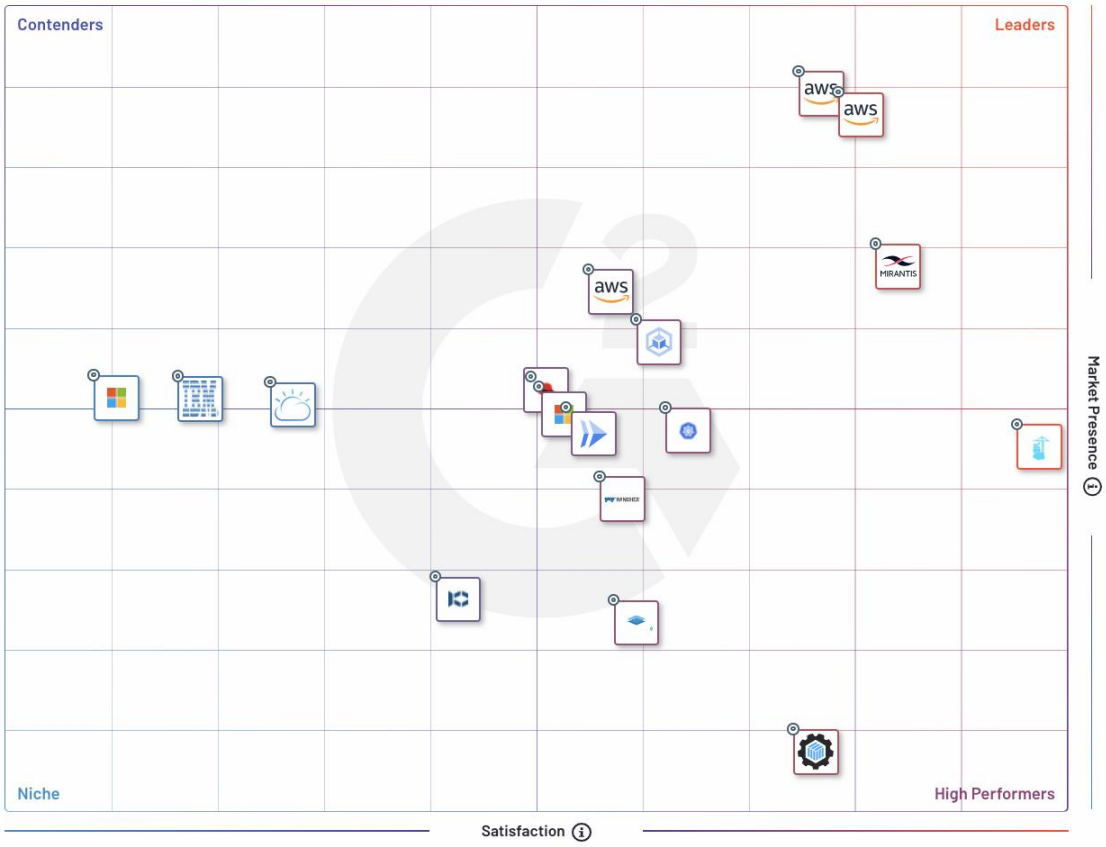
- 12 月 28 日 InfoQ 2022 中国技术力量年终榜单发布

  本次榜单分为【2022 年度十大开源新锐项目】、【2022 年度十大高价值技术团队】、【2022 年度十大云原生创新技术】三大榜单。KubeVela、RocketMQ、TDengine 等入选十大开源新锐项目。

- 12 月 22 日商业软件评测机构 G2 发布《容器管理平台评测报告》

  G2 根据 G2 上的评论以及从在线资料和社交平台上收集的数据，对厂商及其产品进行评估，形成如下的 G2 Grid。 其中，X 轴代表客户满意度，Y 轴代表市场份额。 取得领导者地位的容器管理平台产品为：AWS

Fargate、Amazon ECS、Amazon EKS 和 Mirantis Kubernetes Engine。



- Kasten 上线免费的云原生学习网站 KubeCampus.io

  KubeCampus.io 提供了和 Kubernetes 存储、安全、可观测性和应用一致性等相关的学习资源，如课程、白皮书、视频和网络研讨会。 注册用户还可以通过撰写博客或发表演讲进行贡献，或是与 Kubernetes 社区成员进行交流互动。 该网站还提供了实践实验室，用户可以在专家的指导下，体验从安装 Kubernetes 集群到构建应用的全过程。

- 12 月 21 日全球分布式云大会·深圳站议题征集中

  由全球分布式云联盟、众视 Tech 主办的 2022 全球分布式云大会 · 深圳站将于 12 月 21 日召开。

大会以“云筑数字新基 · 远见数智未来”为主题，将设立分布式主题报告会以及分布式算力、分布式数据库、云原生开源实践三大主题分论坛。

- 12 月 17 日中国 DevOps 社区峰会·武汉定档

  峰会设置 1 个主会场和 4 个分会场。 主会场分享最新 DevOps 成功实践案例，分会场包括精益敏捷专场、工程卓越专场、数字化转型专场、DevOps 流水线工作坊。

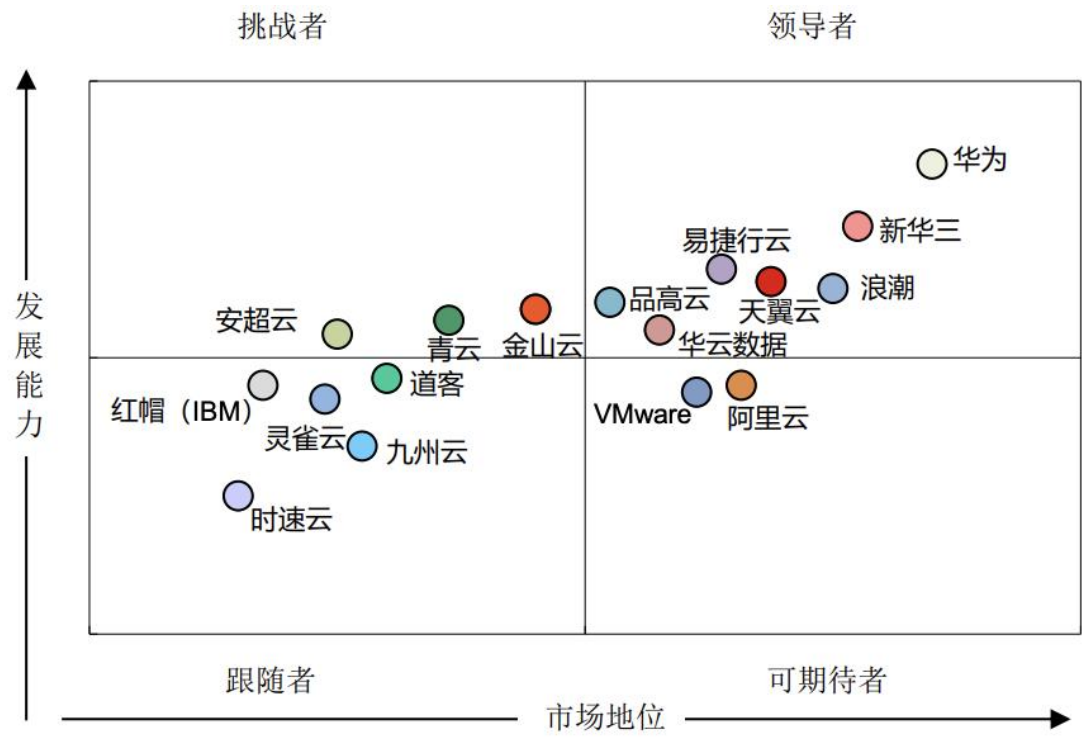
- 12 月 16 日 Linux 基金会亚太团队正式启动 OSPO Summit 2022，共话本地和全球的开源发展

  峰会定于 12 月 16 日在北京举行，将重点讨论 Upstream First 的成功实践和开源办公室（OSPO）的工作实践。

  此外，还将就降本增效、DevSecOps、开源文化、企业单位与开源共同体、许可、供应链、SBOM、项目共同体构建、成为开源世界的领导者等话题展开讨论。

- 12 月 14 日赛迪顾问发布《2022 中国私有云市场研究报告》

  报告对中国私有云和私有云系统平台的发展现状、市场规模、行业应用和产品结构、市场竞争及未来趋势等进行了全面解析。 2021 年，中国私有云系统平台市场规模达到 51.4 亿元，同比增长 45.2%。 其中，容器平台的市场规模为 12.6 亿元，占私有云系统平台规模的 24.5 %。 2021 私有云系统平台厂商竞争力象限分析图如下：



private

- 12 月 14 日第二届「云原生十大技术创新/先锋实践」评选正式启动

  评选由创原会发起，CNCF、中国信通院、华为云参与评选，旨在评选取得显著效果、具有一定复制性和参考性的云原生技术应用实践以及实现创新突破的云原生开源项目、创新产品、科学论文等。 获评项目将受邀参加云原生年度技术创新峰会，并获得媒体宣传机会等。

- 12 月 13 日寻找中国云原生实践先锋的前 1%！CNBPA 2022 云原生最佳实践评选启动

  在信通院云原生产业联盟和 CNCF 的指导下，云原生技术实践联盟面向全行业，征集利用云原生技术实现降本增效、结构升级、管理优化、业务创新等价值的案例。

申报案例所涉及技术领域包括但不限于容器、DevOps、微服务、Serverless、云原生存储、云原生安全、云原生网络、云原生数据库、云原生中间件等方向。

- 12 月 7 日 SODACON 2022 将在日本横滨举行

  SODACON 2022 以“从容器到多云”为主题，将讨论云原生、物联网、大数据、机器学习在存储管理领域的最新趋势、创新、挑战及解决方案。会议以线下及 ZOOM 线上的形式同步举行。

- 12 月 7 日 CNCF 发起地区性云原生系列活动 KubeDay，首场在日本拉开帷幕

  首次 KubeDay 活动 将于 12 月 7 日在日本横滨举行。演讲提案申请现已开放（截至 9 月 6 日 23:59），演讲者需亲临现场。

  CNCF 社区计划在 2023 年将这一活动推广到全球各地，主要针对社区发展活跃和对 Kubernetes 等云原生技术感兴趣的地区。 KubeDay 将分为初级和高级板块，当地和国际演讲者约各占一半。

  所有的主题演讲和会议的录制视频将于 12 月中旬在 CNCF 的 YouTube 频道上发布。

- 12 月 6 日 CNCF 云原生 Meetup 香港站将于 举行

  本次 Meetup 邀请到 CNCF 基金会中国区总监，以及来自甲骨文、字节跳动、极狐 GitLab、Aqua、F5 等公司的技术专家，围绕多云管理、可观测性/追踪、GitOps、云原生安全、云原生人才发展等话题展开讨论。

- 12 月 5 日 ICT 中国·云原生发展论坛将在北京拉开帷幕

  论坛由工业和信息化部主办，聚焦主题 软件架构的变革、新生产力的飞跃 。 届时将发布和应用现代化、金融行业湖仓、Serverless 相关的报告，以及公布云原生存储、大数据、混部、安全、弹性算力、中间件领域的评估结果。

- 12 月 3 日微服务 x 容器开源开发者 Meetup 在上海举行

  百花齐放逐梦云原生，开源分享赋能开发者。

  12 月 03 日上海站，微服务 x 容器开源开发者 Meetup 将围绕云原生领域当下 11 个热门开源项目的技术分享和企业实践展开， 邀请到 Spring Cloud Alibaba、Dubbo、OpenYurt、KubeVela、OpenSergo 等云原生领域传统&新锐开源项目核心维护者，为大家分享社区最近动态、架构实践及企业应用案例等精彩内容。

- 12 月 3 日云原生社区举行 Meetup 长沙站

  此次云原生 Meetup 长沙站由云原生社区主办，演讲者来自极狐 GitLab、中国移动、中国电信等公司， 演讲内容涵盖 DevSecOps、ClusterAPI、云桌面等技术和话题。

2022-11

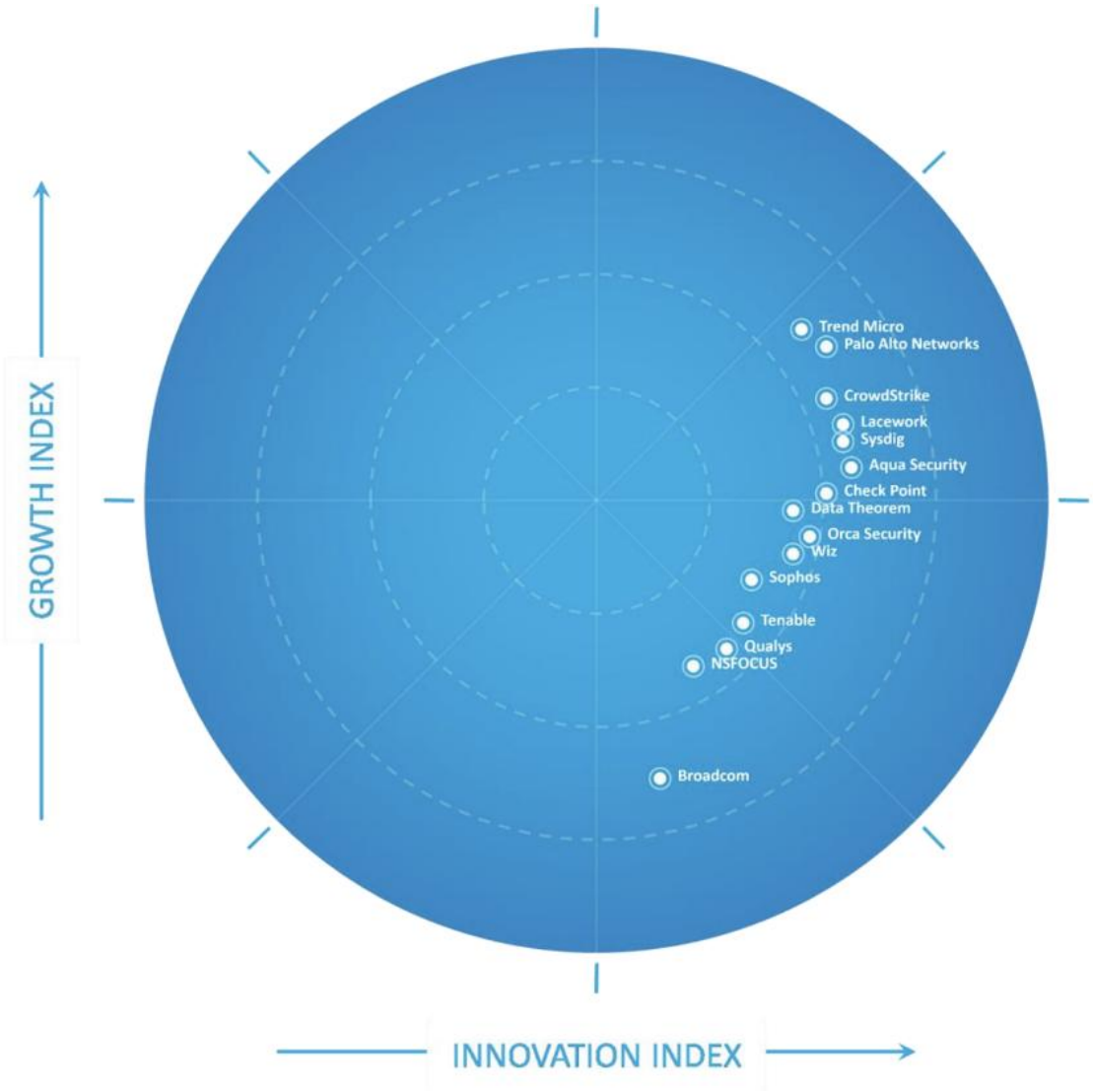
- 11 月 30 日全球分布式云大会年终盛典开启奖项申报

  全球分布式云大会年终盛典由全球分布式云联盟和众视 Tech 主办，旨在表彰在全球范围内分布式云领域的佼佼者， 具体奖项设置包括：

  - 分布式云原生企业 TOP3
  - 中国算力先锋企业 TOP3
  - 分布式操作系统 TOP3
  - 边缘云建设贡献奖
  - 分布式数据库奖等。

- 11 月 29 日国际咨询公司沙利文发布 2022 年云原生应用保护平台雷达报告

  沙利文根据业务增长和创新指标评价评选出 15 家头部厂商，包括 Aqua Security、Check Point、Lacework、 Sysdig 等。其中，Aqua 被评选为投资或合作首选的标杆企业。



protection

- 11 月 29 日 KubeSphere 社区推出播客节目 KubeSphere Talk

  KubeSphere Talk 是由 KubeSphere 发起的一档访谈、聊天的音频节目。 节目前期，主要邀请 KubeSphere 社区的 Ambassador、Contributor 等角色，围绕云原生、开源等热门话题展开讨论。 目前节目已在小宇宙、喜马拉雅、苹果播客三个平台上线，并已发布第一期节目。

- 11 月 23 日 KubeCon 北美 2022 展台热词揭晓，“安全”和“平台”成热门词汇

在对 290 个参展商展台进行分析后发现，“Kubernetes”和“Cloud”分列最常见的词汇前两名，继而是“安全”、“平台”和“数据”。 此外，“规模”被提及 16 次，“成本”被提及 15 次，“Serverless”只被提到 3 次。 由此可以看出，安全扩展和 Kubernetes Day 2 运维管理将继续成为 2023 年的关注焦点。

- 11 月 20 日 QCon 全球软件开发大会（北京站）2022 会议日程发布

  QCon 全球软件开发大会（北京站）2022 将于 11 月 20–21 日举行，会议设置了云原生微服务架构新趋势、云原生架构变革、 DevOps 流程与实践、大规模集群调度、云原生时代的可观测最佳实践、玩转 WebAssembly、业务安全合规等 20 余个专题。

- 11–18 Kubernetes Community Days 成都站重启

  Kubernetes Community Days 成都站由 CNCF 和 DaoCloud 道客、华为云等联合发起的， 聚焦 Istio、KubeEdge、中间件、分布式应用运行时、存储、多云管理等云原生开源项目与技术实践的主题分享。

- 11 月 6 日 OpenSSF 开源安全中国峰会嘉宾阵容完整发布

  OpenSSF 开源安全中国峰会由 OSTech 联合信通院，OpenSSF 基金会发起，会上将分享开源安全最佳实践、开源安全工具、开源软件供应链安全、软件开发安全教育、企业案例分享等内容。

- 11 月 16 日第一届云原生边缘计算学术研讨会议题征集中

  CNCF KubeEdge 社区将于 11 月 16 日举办第一届云原生边缘计算学术研讨会。 研讨会的议题征集已开启，议题可聚焦但不限于云原生边缘计算技术研究成果和基于 KubeEdge 平台的边缘计算动手实践演示。

- 11 月 15 日研究咨询机构爱分析发布《2022 信创厂商全景报告》，DaoCloud 入选厂商全景图信创云板块

报告选取数据库、操作系统、中间件、信创云、云桌面等 9 个特定市场的需求定义和代表厂商能力进行重点分析， 为企业厂商选型提供参考。其中，云平台的全面国产化，逐渐成为企业构建全栈自主可控 IT 架构的关键一环。

- 11 月 8 – 9 日 PromCon EU 在德国慕尼黑举行

  PromCon 由 Prometheus 开发者社区主办，旨在交流使用 Prometheus 的经验和最佳实践。PromCon EU 2022 为线下会议，参会方式：https://cvent.me/EDBynP。

- 11 月 8 日国际开源节 CNCF 云原生论坛嘉宾阵容揭晓

  2022 国际开源节 CNCF 云原生论坛将于 11 月 8 日在深圳举办。 来自 DaoCloud、阿里巴巴、OPPO、腾讯等厂商的专家围绕以下专题在专场论坛上共话云原生：

  - DevOps，CI/CD
  - 容器，多云管理
  - 云原生安全
  - 监视，可观测性
  - 虚拟化、原生应用案例
  - 云原生和其他技术整合(云原生+)

- 11 月 3 日杭州云栖大会定档

  今年云栖大会以 计算·进化·未来 为主题，将于 11.3–5 举行。大会设置两场主论坛，70 多场数字技术、产业和生态分论坛。 其中，云计算论坛将聚焦云原生前沿技术、云上安全和合规、专有云等内容。 目前，大会已开放参会门票免费申领。

- 11 月全球开源技术峰会 GOTC 在深圳开启

GOTC 由 Linux 基金会、开源中国和开源社区联合发起，聚焦 eBPF、Web 3.0、区块链、开源社区、开源商业化、开源教育培训、云原生、AI、工业软件、IoT 等热门话题。 此外，峰会还将邀请 Linux 基金会执行董事 、Apache 软件基金会董事、CNCF 总经理等数十位来自全球的开源重磅嘉宾出席进行演讲分享。

点击查阅赞助商方案 PDF

2022-10

- 10 月 31 日 KubeEdge 云原生边缘计算免费公开课发布

  KubeEdge 云原生边缘计算公开课联合 CNCF 基金会、KubeEdge 社区、高校、企业等产学研多方力量， 提供云原生边缘计算领域的行业发展概览、特性应用实操、典型案例剖析和源码解读等课程。

- 10 月 29 日云原生 DevOps 技术开放日议题征集中

  云原生 DevOps 技术开放日由云原生社区主办、极狐 GitLab、Flomesh 等合作举办，围绕云原生、DevOps、开源等话题。目前已公布部分嘉宾阵容，仍有少量议题席位开放。

- 10 月 27 日 Linux 基金会首个 Istio 免费在线自学课程

  Istio 免费在线自学课程——Istio 入门课程专为希望在 Linux、Docker 和 Kubernetes 知识基础上学习如何安装和配置服务网格的人士设计的。

  讲师为来自服务网格公司 Tetrate 的两位工程师。课程现已上线，用户可在注册课程后十周内免费访问课程，完成课程后，可以选择付费取得结业证书。

- 10 月 26–27 日全球分布式云大会·上海站报名开启

  2022 全球分布式云大会·上海站由全球分布式云联盟、众视 Tech 主办，以 万象智算 为主题，将关注分布式基础设施、分布式云服务、分布式云管理平台、容器应用平台、数据库云原生化、边缘云等内容。

- 10 月 24 至 28 日 KubeCon + CloudNativeCon North America 会议议程发布

  KubeCon + CloudNativeCon 北美将于 10 月 24 日至 28 日在美国底特律举行，共包括 189 场会议、主题演讲和闪电演讲。

  点击查阅线上与会方式

- 10 月 24 日 CNCF 宣布首批通过云原生网络功能认证计划的产品

  CNCF 宣布了第一批通过云原生网络功能（CNF）认证的产品，包括 F5、Juniper Networks、MATRIXX Software 和 PANTHEON.tech。

  CNF 认证计划于今年 5 月发布，利用 CNF 测试套件，帮助通信服务提供商和其他电信组织确定哪些网络设备提供商遵循云原生最佳实践。

- 10 月 21 日中国信通院启动云原生平台中间件管理能力首批评估

  评估主要从接入管理、生命周期治理、运维支撑、运营服务、高可用、兼容性、开放性、安全性八个部分对中间件管理能力进行考察。

- 10 月 19 日 Gartner 发布《2022 云基础设施和平台服务（CIPS）魔力象限》

  Gartner CIPS 魔力象限关注基础设施即服务（IaaS）和集成平台即服务（PaaS）两类产品。 今年的报告显示，AWS、Azure 和 Google Cloud 继续保持领先地位，入选领导者象限。而国内方面，阿里云、华为云和腾讯云入选，其中，华为云是首次入选。



quadrant

- 10 月 17 日艾瑞咨询发布研究报告《2022 年基础云服务行业发展洞察》

  报告显示，2021 年中国整体云服务市场增速回暖，市场规模达到 3280.2 亿元，增速为 45.4%。 公有云市场仍为整体云市场底座，行业云建设为非公有云提供增长机遇。 互联网用户仍是公有云领域的主要客群，但市场相对饱和；金融和政务则为非公有云行业的主要贡献，但增幅缓慢，工业制造和医疗贡献能力显著提升。

- 10 月 16 日全球开源技术峰会 GOTC 2023 议题征集中

  GOTC 2023 由 Linux 基金会、开源中国和开源社区联合发起的，将于 2023 年 2 月 25-26 日在深圳举行。 会议聚焦云原生、元宇宙、eBPF、Web3.0、开源治理、前端、基础软件、AI、IoT 等热门话题。

- 10 月 9 日下半年『可信云·云原生类评估』报名正式启动

  中国信通院组织编写了云原生相关的系列行业标准，并据此建立了可信云·云原生评估体系。 依据评估体系，信通院开启下半年云原生类评估测试工作，评估覆盖容器、微服务、Serverless、云原生安全、云原生数据以及云原生成熟度等多个方面，旨在规范云服务商与安全服务商产品质量， 助力云原生改造，为产品选型、评测、验收提供参考。

- 10 月 5-6 日 KubeCon 北美 2022 预热活动——『KubeCrash Fall 2022』免费开放注册

  KubeCrash 是由 CNCF 项目和其他开源项目的维护者提供的免费线上云原生技术分享活动。今年的秋季 KubeCrash 将于 10 月 5-6 日举行。首批演讲者和议题已公布，分享内容包括云原生成熟度模型、cert-manager、Telepresence、Linkerd、Polaris、Pulumi 等开源技术。

  注册地址：https://www.kubecrash.io/register

- 10 月 4 日谷歌发布《2022 DevOps 状况报告》

  报告深入探讨软件供应链安全这一主题。 报告显示，在 SLSA（软件工件供应链）和 SSDF（安全软件开发框架）的实践中，使用应用安全扫描作为 CI/CD 系统的一部分是最常见的做法。 同时，报告还关注了软件交付和运营绩效。此外，云计算的使用量持续增长，公共云的使用率为 76%（2021 年为 56%）。

2022-09

- 9 月底 eBPF Summit 2022 召开

峰会建议演讲应介绍如何使用 eBPF 来解决实际问题，如：观察、跟踪和调试应用程序，保护系统、CI/CD Pipeline、应用，解决网络或性能问题，运行时、内核实现和 eBPF 库，用高级语言编写 eBPF 程序，eBPF 项目，基于 eBPF 的可观测性和监测，来自终端用户的案例研究，未来展望等。 峰会接受提交 5–10 分钟的闪电演讲和 20 分钟的会议。

- 9 月 28–29 日 eBPF Summit 2022 会议议程发布

  eBPF Summit 2022 将于 9.28–29 日以线上方式举行，会议全程免费对外开放，届时来自 Datadog、Isovalent 等技术专家将代理有关 eBPF 的网络、安全和可观测等内容的演讲。

- 9 月 26 日 Linux 基金会开源软件学园人才激励计划正式启动

  人才激励计划旨在为 IT 行业的人才提供开源培训的机会，共包括开源萌新、云原生领航员、运维之星、大前端技术人才等 8 个类别，申请者可根据自身情况选择。 入选者可以在开源软件学园中任意选择一门课程进行免费学习，并免费参加一次基金会提供的任何认证考试。

- 9 月 25 日云原生社区 Meetup 广州站

  云原生社区 Meetup 广州站将于 9 月 25 日举行，『云原生社区』视频号和 B 站将同步直播。活动邀请来自极狐、Intel 等工程师分享有关云原生应用安全、负载均衡等内容。

- 9 月 24 日第六届全国互联网架构峰会（IAS2022）正式开放注册，DaoCloud 首席执行官陈齐彦将带来开场演讲

  IAS2022 由 NJSD 技术社区携领主办，将于 9.24 在南京举行。会议内容涵盖流量编程、微服务治理、Data Mesh、多态应用管理、DDD 架构设计、eBPF 性能加速等议题。目前已开放注册，视会场人数控制额满为止。

- 9 月 24 日服务网格峰会 Service Mesh Summit 2022 重新开启，在上海举行

Service Mesh Summit 2022 首届服务网格峰会由云原生社区主办，将于 9 月 24 日在上海举行。 届时，来自 Tetrate、阿里云、腾讯云等的技术专家将分享服务网络实践落地、Istio 自定义扩展实现和相关开源项目等。目前，已重新开放免费报名通道。

- 9 月 23 日艾瑞咨询发布《2022 年中国金融云行业研究报告》

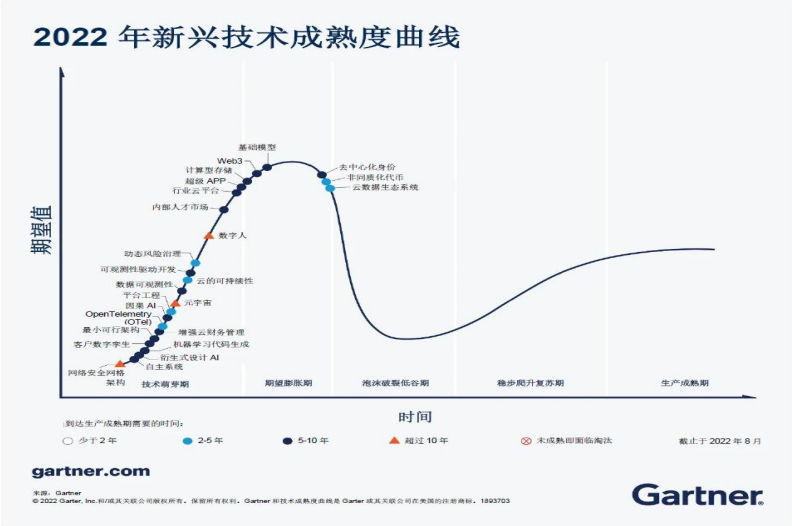
  报告指出，2021 年中国金融云市场规模达到 394 亿元，预计未来 4 年复合增长率为 28.6%；存量市场：自主研发、自主可控的国内技术路线成为金融机构的采用趋势。 增量市场：云原生成为重要战略技术趋势，多云战略部署及跨云生态连接成为行业共识。

- 9 月 23 日 DaoCloud 发布《云原生产业发展白皮书》

  白皮书从云原生的定义、起源与应用现状出发，梳理云原生产业的发展逻辑与产业全景图，分析云原生给中国信创产业带来的机遇，并结合国内领先的云原生实践，展望未来趋势，定义云时代的操作系统。

- 9 月 22 日 Gartner 发布 2022 新兴技术成熟度曲线

  2022 年新兴技术趋势主要专注于三个主题：发展/扩展沉浸式体验、加速人工智能自动化以及优化技术人员交付。 其中，优化技术人员交付部分，OpenTelemetry 等在未来 2–5 年将对商业及社会产生显著影响，而数据可观测性、可观测性驱动开发、行业云平台等需 5–10 年才会得到广泛采用。



hype

- 9 月 15–17 日国际开源节，云原生成为焦点

  2022 国际开源节由 OSTech 联合中国信通院、Linux 基金会发起，展会将于 9 月 15–17 日在深圳举办。 CNCF 云原生论坛作为其中重要专题技术论坛之一，将重点围绕“DevOps，CI/CD”、“容器，多云管理”、“云原生安全”、“监视，可观测性”、“虚拟化，原生应用案例”、“云原生和其他技术整合(云原生+)”等专题。

  合作申请：http://iosf-world.mikecrm.com/Dd9SsbA

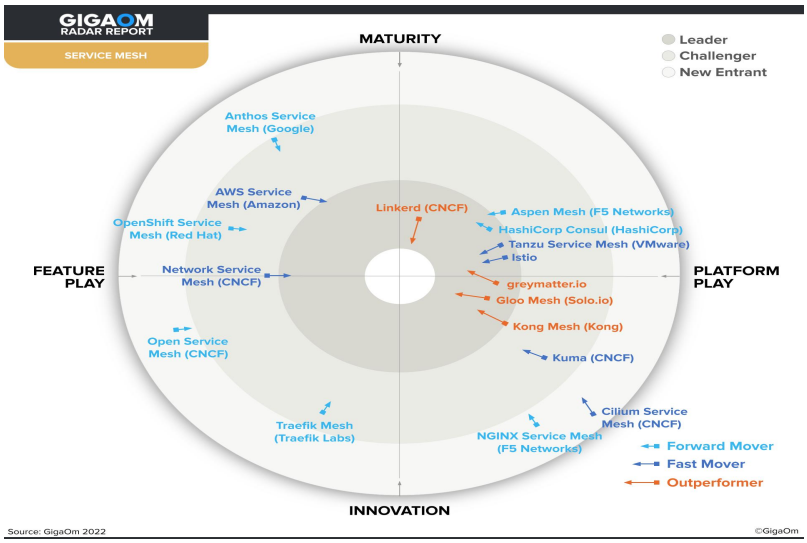
- 9 月 15 日 SUSE 成立 RFO SIG，建设面向 openEuler 的容器基础设施平台

  RFO 即 Rancher for openEuler，旨在将 Rancher 产品生态与 openEuler 深度结合，为欧拉开源社区构建容器工程基础设施，打造面向 openEuler 的 Rancher 衍生产品。

- 9 月 15 日国际分析机构 GigaOm 发布《服务网格雷达报告 v2.0》

  GigaOm 从专门基础设施层、sidecar 实践、控制平面配置以及控制平面遥测四个维度，对主流服务网格进行了评测。 其中，Gloo Mesh、

greymatter.io、Kong Mesh 和 Linkerd 被评定为表现优异者。值得关注是 Cilium Service Mesh，是首批既可以利用 sidecar 模式也可以选择无 sidecar 模式的服务网格之一。



radar

- 9 月 13–16 日 Linux 基金会公布 Open Source Summit 欧洲 2022 会议日程

  Open Source Summit 欧洲 2022 将于 9 月 13–16 日以线下（都柏林）+线上方式举行。峰会共包括 13 个分论坛，325+ 场演讲， 话题覆盖开源领域最重要和最前沿的话题：Linux 系统、供应链安全、OSPO、嵌入式物联网、云、容器等。

- 9 月 11 日 VMware 发布报告《Kubernetes 使用现状 2022》

  报告指出，Kubernetes 集群数量出现快速增长，29% 的公司拥有 50 个以上的集群；多云占据主导地位，本地部署和单个云失去优势；多集群、多云安全性是新的前沿问题。

  点击查阅完整版报告

- 9 月 9 日 Cisco 发布《2022 年全球混合云发展趋势报告》

报告显示，混合多云将成为常态，82% 的受访者使用混合多云架构来支撑其应用程序。 混合云发展面临安全，运营，成本三大挑战。边缘计算、基础设施自动化和可组合基础设施将成为未来混合云领域的三个技术热点。云原生技术加速发展、基础设施即代码的广泛应用、及开放文化的积极作用是未来混合云发展的三大趋势。

点击查阅完整版报告

- 9 月 8 日 CNCF Prometheus 职业资格认证考试升级为 GA

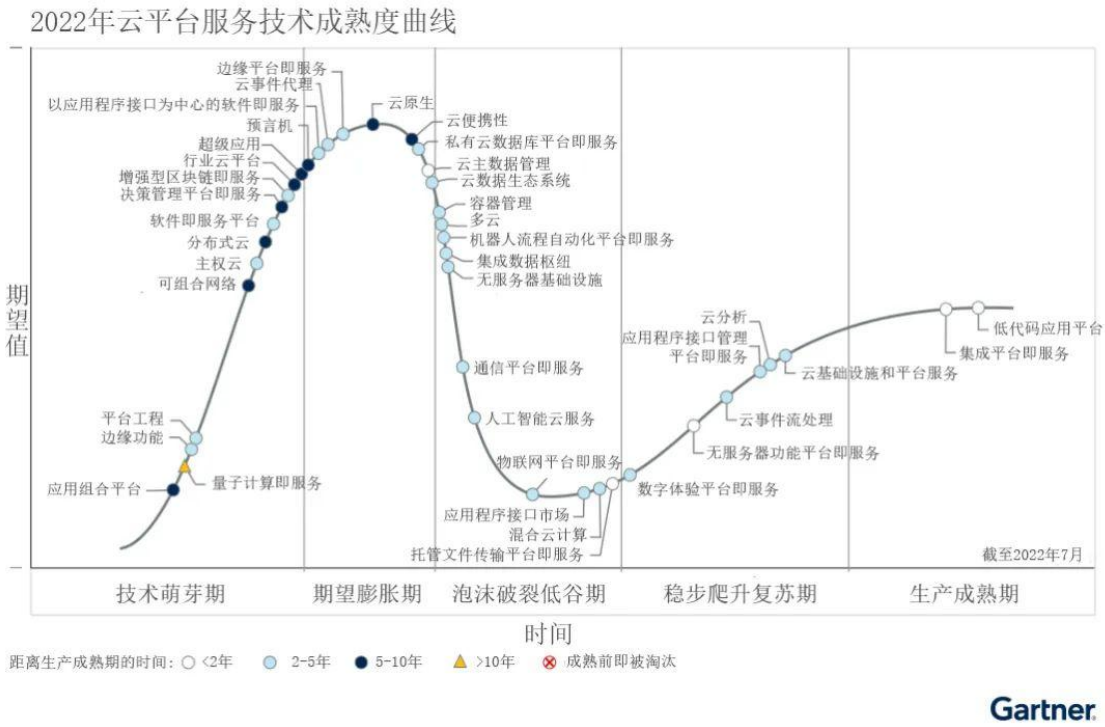
  在 Beta 测试版发布数月后，CNCF 宣布 Prometheus 职业资格认证（PCA）考试已升级为 GA。

  PCA 是一门在线认证的多项选择测试，专为对可观测性和监控感兴趣的工程师或应用开发人员设计，旨在测试考生使用 Prometheus 监控云原生应用和基础设施及通过可观测数据提高应用性能等相关基础知识的掌握情况。

2022-08

- 8 月 31 日 Gartner 发布 2022 年云平台服务技术成熟度曲线

  2022 年云平台服务技术成熟度曲线显示，云原生技术当前处在期望膨胀期，预计未来 5–10 年进入生产成熟期； 容器管理技术处在泡沫破裂低谷期，2–5 年进入成熟期；集成平台即服务（iPaaS）和低代码应用平台（LCAP）技术将在两年内达到生产成熟期（技术的现实意义已得到证明和接受）。



Gartner

- 8 月 27 日 KubeSphere 社区云原生技术交流 Meetup 杭州站报名开启

  云原生技术交流 Meetup 杭州站由 KubeSphere 社区与极狐 GitLab 社区联合组织。活动包括闪电演讲（10min）、单人/双人演讲（40min）两种形式， 议题内容可包括云原生领域的项目与云原生技术实践，活动日期定于 8 月 27 日。

- 8 月 23-24 日 Linux 基金会公布 Open Source Summit 拉美 2022 会议日程

  Open Source Summit 拉美 2022 将于 8 月 23-24 日在线上举行。峰会共包括 LinuxCon、 CloudOpen、ContainerCon 等 9 个分论坛，75+ 场演讲。

- 8 月 19 日智能云边开源峰会于线上召开

VMWare 携手 Intel、EMQ、Kyligence 等将于 8 月 19-20 日举办 2022 智能云边开源峰会 。会上多位来自开源领域的技术领导者将分享 人工智能、云原生、边缘计算 三大热门技术的趋势及洞察。

- 8 月 19 日全球分布式云大会拟在上海召开（内附报名方式）

  本次大会采取线上线下联动的方式，会议议程包括上午的分布式云主题报告会和下午的四个分论坛：数据库专题报告会、边缘云论坛、云游戏论坛和云计算开源论坛。

- 8 月 17 日业界首个产学研 eBPF 技术探索 SIG 成立

  在 Linux 内核专家的引领下，龙蜥社区成立 eBPF 技术探索 SIG，打造一个适合业界和学术界深入研究 eBPF 技术、落地效果的良好生态。 同时，也带来了两大开源项目 Coolbpf 和 LMP，其中 Coolbpf 着重于提升开发效率， 后续会从 Security 角度深入探索；LMP 通过机器学习模型等方案，从可视化平台的角度基于 eBPF 技术深入调试内核。

- 8 月中旬 2022“科创中国”开源创新榜评选全面开启

  2022“科创中国”开源创新榜由中国科学技术协会主办、开源创新联合体等承办，面向中国开源行业领域（云原生、人工智能、数据库、大数据、物联网、工业互联网、软件研发等），评选具有创新性、贡献度和影响力的产品、社区、活动、人物。

- 8 月 13 日论道原生云原生 Meetup 南京站报名开启

  论道原生 Meetup 南京站由道客、鹏云网络等主办，聚焦服务网格、云原生存储、云原生安全等话题及相关开源项目。 本次活动将于 8 月 13 日举行，采取线下 + 线上直播的形式。

- 8 月 10 日 CloudNativeDay 2022 于线上举行

  CloudNativeDay 2022 是由科技媒体公司 Techstrong 发起的云原生演讲活动。 演讲者分别来自 Google、IBM、Splunk、Oracle 等公司，聚焦

Kubernetes、供应链安全、可观测性、服务网格、WebAssembly、DevSecOps 等话题。

- 8 月 8 日 InfoQ 发布《中国开源发展研究分析 2022》

  报告显示，中国开源项目技术领域正在从优势领域数据库向操作系统 & 人工智能和云原生等方向进行升级聚焦。 非互联网行业中，对开源模式认知度最高的分别是制造、金融、政府 / 非营利机构。开发者（互联网 + 非互联网）整体关注的技术领域相对平均，排名前三的是数据库、云原生和工具。中国开源代码质量与社区活跃度呈正相关。

  点击阅读 PDF 报告

- 8 月 6 日云原生技术交流 Meetup 广州站议题征集开启

  今年的第二站云原生 Meetup 将于 8 月 6 日在广州举办。 本次 Meetup 由 KubeSphere 社区联合 SOFAStack 社区发起，主题仍然是「云原生技术交流」。 此次活动包括闪电演讲（10 min）、单人/双人演讲（40 min）两种形式，议题内容可包括云原生领域的项目与云原生技术实践。

- 8 月 2 日 CNCF 电信行业微调查：四分之三的受访者表示将采用云原生网络功能（CNF）

  微调查显示，四分之三的受访者正在将作为物理网络功能（PNF）或虚拟网络功能（VNF）运行的服务转换为基于容器和 Kubernetes 的云原生网络功能（CNF）。 而对于使用 Kubernetes 时面临最大的挑战，排名前三的分别是：网络（57.14%）、产品间的互操作性（42.86%），集群管理和复杂性（均为 33.33%）。

- 8 月 2 日中国 DevOps 现状调查报告（2022）

  调查由中国信通究院联合超 50 家企业共同发起。 最终报告显示：中国企业 DevOps 落地成熟度不断提升，近六成企业向全面级迈进；近九成企业

在不同程度上实践敏捷开发模式； 自动化代码扫描、单元测试在自动化测试中的占比首次超七成；实践 DevSecOps 的企业首次突破六成。

点击查阅报告

- 8 月 1 日 CNCF 开启 2022 年有关云原生普及趋势的调查

  今年，CNCF 与 Linux 基金会合作发起云原生调查，总结容器、Kubernetes 以及其他云原生技术的普及趋势，以及用户使用的工具、项目和面对的挑战。 调查是匿名的，截止时间为 8 月 1 日。

- 8 月开发者平台服务商 Humanitec 发布《2022 Kubernetes 基准研究》

  Humanitec 对 1164 个开发团队进行研究，评估团队 Kubernetes 的实践现状、技术方法以及团队设置和文化。 根据 Kubernetes Performance Score（KPS），分为 top/mid/low performer。开发人员的 self-service 程度是区分 top 和 low 的关键。 其中，所有服务和应用都实现容器化的占比：68% vs 22%；启用开发者自助服务占比：73% vs 33%（top vs low）。

2022-07

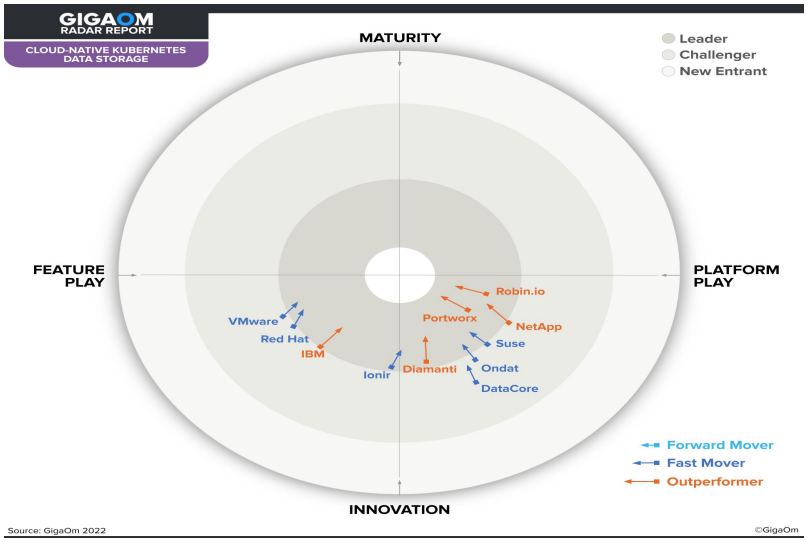
- 7 月 29 至 31 日 ApacheCon Asia 2022 将于 7 月底在线举行（内附演讲申请方式）

  ApacheCon Asia 大会将于 2022 年 7 月 29 日至 31 日在线举行。 大会将再次展示来自基金会的几十个项目相关的内容，围绕 API / 微服务、开源社区、应用集成、消息队列、中间件、可观察性、远程过程调用 RPC、WEB SERVER/TOMCAT 等十几个专题展开讨论。 演讲稿的征集已经开始，截止到 5 月 31 日。

- 7 月 28 日国际分析机构 GigaOm 发布 2022 云原生数据存储雷达报告

  如雷达图所示，右下象限的厂商将持久性存储视为其差异化竞争优势。 这一组的共同点是将存储平台与 Kubernetes 发行版和集群管理产品结合起

来，为 Kubernetes 用户提供一个高度集成的整体方案。 而左下象限的厂商更多面向特定功能、部署模式或个性化使用案例。



radar

- 7 月 27 日阿里云发布《云原生架构白皮书 2022 版》

  白皮书在阐述云原生架构定义的同时，也基于阿里云云原生实战经验总结了云原生架构应用所需的演进路径与设计规则。 相较于旧版本，2022 版新增数十家企业云原生实战经验合集，同时在云原生产品矩阵中，新增多个核心产品家族，如容器产品、微服务产品、云原生技术中台产品家族等。

  点击查阅白皮书

- 7 月 21 日中国信通院发布《云计算白皮书（2022 年）》

  今年的白皮书聚焦 新经济，上云用云新周期 。 新周期指的是用云原生改造应用架构，以算力服务为资源调度手段，实现高效、安全、精益上云用云的新阶段。 新周期下，云计算发展的五个重点：规模化建设、算力服务、稳定性、云安全和云成本优化。

  点击查阅白皮书阅读

- 7 月 21 日 KubeEdge 社区发布首篇云原生边缘计算威胁模型及安全防护分析报告

  KubeEdge 社区运用 ASTRIDE Low Level 威胁建模方法并结合 KubeEdge 安全审计报告，对 KubeEdge 进行系统的安全分析，认为 KubeEdge 潜在的攻击者包括三部分： 外部攻击者、内部攻击者和供应链攻击者。同时也列举出目前社区已有的消减机制和安全加固建议。

  点击查阅 GitHub 更多详情

- 7 月 14–15 日 DaoCloud 亮相首届 KubeEdge Summit，共赴一场云原生边缘计算盛会

  作为去年 KubeEdge 项目贡献榜排名第二的团队，DaoCloud 在上周圆满落幕的 KubeEdge Summit 2022 带来了十分精彩的技术分享， 内容涵盖面向边缘计算的网络安全技术、SIG Device-IOT 进展和未来规划、边缘计算系统一致性测试等话题。

- 7 月 14 – 15 日 2022 云原生边缘计算峰会议程发布

  首届云原生边缘计算峰会（KubeEdge Summit 2022）由 CNCF、KubeEdge 社区主办，将于 7 月 14–15 日以线上方式召开。峰会分为主论坛和技术前沿、生态创新、应用实践三个分论坛。

- 7 月 13 日云原生技术实践联盟联合云原生技术社区发布《（2021–2022）传统行业云原生技术落地调研——央国企篇》

  调研从 IT 预算、应用架构、技术选型、转型成效等方面对央国企的云原生数字化转型进行了解读。 其中，六成多受访央国企已经或考虑使用多云/混合云；更倾向采用一云多芯、双 CPU 架构；超 95% 考虑或正在使用容器、DevOps、微服务等云原生技术。

  点击查阅完整版报告

- 7 月 13 日 SMI 社区、Kubernetes SIG Network 发起 GAMMA 倡议，推动 Gateway API 和服务网格的融合

  SMI 允许对服务间的流量进行直接统一的管理，而 Gateway API 已经为 ingress 和 egress 用例实现了类似的目标。 因此，SMI（Service Mesh Interface）、SIG Network 共同发起 GAMMA 倡议（Gateway API Mesh Management and Administration），计划就 Gateway API 项目中的服务网格用例进行合作。

- 7 月 6 日灵雀云联合云原生技术社区发布一系列金融行业云原生实践报告
  - 《金融云原生技术实践案例汇编》：国内外头部金融企业的云原生技术实践案例。
  - 《金融行业云原生技术落地调研报告》：从全栈视角，探究金融企业目前高度关注的云原生基础设施、混合云、微服务架构、数据、安全等领域的技术问题和行业趋势。
  - 《2022 金融云原生落地实用指南》：聚焦金融企业云原生转型各个阶段的实际痛点及解决方案。从云原生架构、基础设施、安全、运维等角度出发，探讨金融企业云原生落地实践的成功路线。
- 7 月 4 日国际分析机构沙利文联合头豹研究院发布《云原生市场研究报告——容器》（2022）

  报告提出，当下云原生厂商容器产品的竞争关键在于“产品在相应场景中的能力，而产品能力的关键在于开放生态的建设”。 此外，报告基于应用、产品及生态三大维度对华为云、阿里云、腾讯云等主流容器厂商的核心竞争优势及综合壁垒进行了评价。

  点击查阅完整版报告 PDF

2022-06

- 6 月 25 日首届云原生边缘计算峰会，演讲议题火热征集中（内附议题提交方式）

CNCF KubeEdge 社区宣布，将于今年 6 月 25 日举办首届云原生边缘计算峰会（KubeEdge Summit 2022）。 会议地点暂定于北京，采取线下会议+线上直播的形式。议题聚焦 KubeEdge 技术干货分享、KubeEdge 社区生态发展和 KubeEdge 在传统与新领域的落地实践案例。 目前，大会议题征集通道已开放。

- 6 月 22 日 Rust For Linux 或将并入 Linux 5.20 内核中

  近日，Linux 操作系统之父 Linus Torvalds 表示，Linux 内核预计会包括 Rust 基础设施。 此外，Rust 可能会用于开发一些驱动程序（整个驱动子系统或文件系统），因此 Rust 并未“取代 C 语言”，而是“在一些有意义的地方扩展 Linux 内核的 C 代码”。 Linux 5.20 合并窗口将在 7 月底 Linux 5.19 稳定版发布后打开。

- 6 月 22 日中国信通院联合腾讯云发布业界首个《分布式云发展白皮书(2022)》

  白皮书首先明确分布式云标准定义、与混合云和边缘计算的区别与联系；然后介绍分布式云的关键技术和应用场景， 提出分布式云在基础技术、应用场景和标准方面面临的挑战和未来展望；最后深度解析多行业实践案例，为企业落地分布式云提供参考。

  白皮书阅读：https://share.weiyun.com/brailKaS

- 6 月 21 至 24 日 Linux 基金会公布 Open Source Summit North America 2022 会议日程安排

  近日，Linux 基金会公布了 2022 北美 Open Source Summit 的全部日程安排，大会将于 6 月 21 日至 24 日在美国德州和线上同步举行。 大会由 14 项活动组成，包括 LinuxCon、SupplyChainSecurityCon、CloudOpen、ContainerCon 等。

- 6 月 20 日软件咨询公司 Scott Logic 发布调查报告《The State of WebAssembly 2022》

报告的重点内容：Rust 的使用率和关注度持续攀升；Python 的使用量大幅提升；JavaScript 成为一种 WebAssembly 语言； Wasmtime 是使用最广泛的运行时；WebAssembly 在无服务器、容器化和插件环境方面的使用显著增加。

- 6 月 16 日 Linux 下一代架构基金会混沌工程特别兴趣小组成立

  由中国信通院牵头，携手谐云等小组成员申请在下一代架构基金会（NextArch Foundation）中设立混沌工程特别兴趣小组，将国内混沌工程相关研究与实践推向国际。 小组也同 CNCF TOC 紧密协作，推动混沌工程工具和建设的标准化。此外，谐云向小组贡献了首个开源项目：混沌工程故障库“太空舱”。

- 6 月 15 日 DevOps 工具链管理器 DevStream 进入 CNCF 沙箱项目

  DevStream 可以通过一个简单的配置文件，将软件研发生命周期中各环节的 DevOps 工具统一管理起来，完成各工具的快速安装部署、工具间整合、最佳实践配置等工作。

  项目地址：https://github.com/devstream-io/devstream

- 6 月 15 日服务网格项目 Aeraki Mesh 进入 CNCF 沙箱项目

  Aeraki Mesh 旨在解决目前的服务网格项目只满足 HTTP/gRPC 协定，不反对其余开源及公有协定的痛点，提供和 Istio API 完全兼容的管理服务，帮助用户以较小的迁移成本和维护代价，快速获得服务网格提供的流量管理和服务治理能力。

  项目地址：https://github.com/aeraki-mesh/aeraki

- 6 月 15 日网易开源的分布式存储系统 Curve 进入 CNCF 沙箱项目

  Curve 可以应用于各类云原生基础设施平台作为存储底座。和开源存储系统 Ceph 相比，Curve 的随机读写、延迟性能都更优，异常状态下的性能稳定性领先接近一倍半。 和基于容器的块存储开源软件 OpenEBS 相比，

Curve 提供块存储和文件存储能力，更有利于建设统一的数字化基础软件，运维管理成本要求也更低。

项目地址：https://github.com/opencurve/curve

- 6 月 15 日云原生安全实验室正式成立，国内首个云原生安全测试平台发布

  云原生安全实验室由中国信通院联合腾讯云等发起成立，成立后发布的一个主要成果是云原生安全测试平台。 该平台由开放的安全测试框架和独立的安全测试工具组成。

  共计集成 75 个安全能力测试工具，评估能力覆盖容器安全和研发运维安全，并能自动生成详细的评估报告和改进建议，支持 SaaS 接入和私有化部署。

- 6 月 14 日 CNCF Glossary 云原生词汇表中文版上线

  云原生词汇表是由 CNCF 商业价值小组委员会领导的一个项目，旨在以清晰通俗的语言解释云原生概念。

  中文版云原生词汇表：https://glossary.cncf.io/zh-cn/

  贡献方式：https://glossary.cncf.io/zh-cn/contribute/

- 6 月 14 日中国信通院发布《边缘计算市场和用户洞察报告（2022）》与《云边端一体化发展报告（2022）》

  《边缘计算市场和用户调查报告（2022）》指出，2021 年我国边缘计算市场规模达到 436.4 亿元，其中边缘软件与服务市场规模达 146.2 亿元。 边缘应用方面，用户现阶段呈现出在边缘侧部署类型多样化、技术应用以数据类为主。边缘原生、行业融合、开源协同成为发展趋势。

  《云边端一体化发展报告（2022）》指出，“云边协同”正在向“云边端一体化”演进，旨在屏蔽云、边、端分布式异构基础设施资源，提供统一视角

资源管理和使用。 发展趋势方面，一是云边端一体化驱动计算处理向边端扩散；二是云边端一体化构建新型数字基础设施操作系统。

边缘计算市场和用户：http://doc.opensourcecloud.cn/2022/0617-1.pdf

云边端一体化：http://doc.opensourcecloud.cn/2022/0617-2.pdf

- 6 月 14 日云边协同大会完整议程正式发布

  中国信通院联合中国通信标准化协会将于 6 月 14 日以线上方式召开“2022 云边协同大会”。 会上将发布分布式云评估成果、边缘计算行业应用评估成果、云边协同实践案例等，演讲聚焦云边协同、物联网、行业应用等话题。

- 6 月 11 日 CNCF、微软、英特尔联合发起开源云原生开发者日，聚焦微服务、持续集成交付和混沌工程（内附报名方式）

  由 CNCF、微软、英特尔联合发起，JFrog、Dapr 中文社区、KubeSphere 等鼎力协办的开源云原生开发者日定于 2022 年 6 月 11 日。 活动聚焦微服务、持续集成、交付和混沌工程等话题，探讨云原生的未来。目前，活动的议题正在征集中。

- 6 月 7 日 Gartner 发布 《2022 APM 和可观测性魔力象限》

  今年共 19 家厂商入选 APM 魔力象限，较去年增加 4 家。 其中，提供生产系统可观测性工具的厂商 Honeycomb 首次入围即进入领导者（LEADERS）象限。 云监控 SaaS 平台厂商 Datadog 也凭借其平台强大的产品组合排名大幅上升。 Cisco（AppDynamics）和 IBM (Instana) 的排名下降较明显。此外，阿里云凭借其 APM 的核心产品——应用实时监控服务（ARMS）成为国内唯一入选的厂商。



APM Quadrant

- 6 月 6 日 Kubernetes 成本管理公司 Kubecost 联合 AWS、谷歌等向 CNCF 提交成本管理开源项目—— OpenCost

  OpenCost 旨在实现成本跟踪、分配和量化的标准化，帮助 Kubernetes 团队了解其基础设施成本。 OpenCost 主要功能：按 Kubernetes 的各种对象实时分配成本、支持具有自定义定价表的企业内部 k8s 集群、支持分配集群内资源、支持将定价数据导出到 Prometheus 等。

项目地址：https://github.com/kubecost/opencost

- 6 月 3 日调查显示：服务网格已成主流，但对有关话题的阅读兴趣正在降低

  CNCF 最近发布的一项调查指出，服务网格已经成为云原生基础设施的一个关键组成部分，70% 的调查对象在生产或开发中采用服务网格。 同时，科技媒体 the New Stack 的一份调查显示，随着该技术的成熟，读者对其阅读兴趣正在降低。

  点击阅读《CNCF MicroSurvey of Service Meshes》完整版 PDF

- 6 月 2 日 Serverless 报告：AWS、Azure 和谷歌云平台中有超过 50% 的用户正在使用 Serverless

  Datadog 近日发布的《2022 年 Serverless 报告》发现 Serverless 正处于主流应用阶段。 随着产品成熟度的增加以及更多函数和 serverless 容器产品的发布，越来越多的组织将会采用 Serverless。 报告还发现，在 Google Cloud 上运行的 Datadog 客户中，有近 40% 使用 GCP 的无服务器容器产品 Google Cloud Run，比 2020 年 1 月增加了 4 倍。

  完整版报告：https://www.datadoghq.com/state-of-serverless/

- 6 月 2 日 NextArch 基金会旗下微服务标准化方案已开源：支持不同开发语言和技术框架

  微服务治理标准化的解决方案包含两个部分：一部分是服务治理的标准化功能定义和最佳实践，一部分是服务治理面和数据面的标准化协议和接口。标准化实现包含三个部分：服务治理控制平面、数据平面和开发框架。标准化解决方案分为八个部分： 标准化 Spec 建设、控制面支持（PolarisMesh）、数据面支持（PolarisMesh）、开发框架支持、go-zero、CloudWeGo/Kitex、GoFrame 和 Spring Cloud Tencent。

  相关项目地址:

- 服务治理标准化 Spec：https://github.com/nextarch/SIG-Microservice

- PolarisMesh / polaris：https://github.com/polarismesh

- go-zero：https://github.com/zeromicro/go-zero
- CloudWeGo/Kitex ：https://github.com/cloudwego/kitex
- GoFrame：https://github.com/gogf/gf
- Spring Cloud Tencent：https://github.com/Tencent/spring-cloud-tencent

- 6 月 1 日 Kube-OVN 团队发布 《2022 云原生网络趋势》

  2022 年云原生网络趋势存在以下特点：单一 CNI 很难满足所有场景需求，不同场景会出现特定网络插件； 不依赖特定 CNI 的网络辅助组件如 Submariner 或 Clusternet 会不断涌现； eBPF、智能网卡等一些新软硬件技术会与 K8s 不断融合，此外，云原生传统的技术也会不断增强整个云原生网络能力； 软硬件结合的场景可能会在不同领域、特定场景中发挥巨大的优势。

- 6 月 1 日 CNCF 发布 《kubernetes 2021 年度报告》

  报告记录了 2021 年关于 Kubernetes 社区发展状况的定量指标：过去一年有 62,000 人参与贡献、新增一个新的 SIG K8s Infra 等；

  也记录了社区 leader 和项目贡献者报告的定性总结： v1.21-1.24 正式 GA 或取得显著进展的功能、供应链安全性方面取得的成果、社区未来工作计划（包括建立“项目健康”指标、增加小组审查员、长期贡献等）以及不同 SIG 面临的困境等。

- 6 月云边协同创新实践案例火热征集中（内附申报方式）

中国信通院云计算与大数据研究所正式启动 2022 年分布式云与云边协同创新实践案例征集工作， 通过自主申报、专家评审、实际调研 / 考察等方式遴选一批典型创新案例并加以推广，评选结果将在今年 6 月的云边协同大会上正式发布。

- 5 月 31 日至 6 月 2 日 2022 国际开源节延期通知

  鉴于目前国内新冠疫情防控形势严峻，原定于 2022 年 5 月 31 日至 6 月 2 日在上海举办的 2022 国际开源节将延期举办，预计延期至 12 月举办，如有新的进展主办方将会发布公告。

2022-05

- Gartner 发布《监测和可观测性领域的优秀服务提供商》

  报告介绍了四家围绕可观测领域进行扩展的厂商：

  - Cribl：支持企业有效地管理由可观测性系统产生的大型数据集
  - Komodor：专注于捕捉和关联 Kubernetes 环境中的变化
  - Observe：支持通过关系和时间分析对事件数据进行探索
  - Rookout：在生产系统中提供可观测性，允许开发人员和 SRE 访问有关应用程序性能的实时信息

  点击阅读完整版报告

- OpenLogic 和 Open Source Initiative 发布《2021 全球开源趋势报告》

  报告对全球 2660 个企业用户进行了调研。报告主要内容：

  - 用户使用开源软件的首要原因是创新，其次是降低成本，补丁的安全性和可用性也排名靠前；
  - 采用开源软件的第一大障碍是缺乏测试、使用、集成和支持的内部技能；
  - 使用率总体增幅最高的是开源 DevOps 工具，云原生 CI/CD 工具的使用率显著提高；

- 最令人向往的新技术是容器和 Kubernetes，但目前只有 18% 的受访者使用 Kubernetes，39% 使用云，29% 采用容器化方案。

点击下载完整版报告 提取码：ny5a

- 5 月 23 日艾瑞咨询发布《2022 年中国云服务行业应用白皮书》

报告显示，云服务市场泛互联网行业占比最高，达 27%，其次是政务和金融行业。 报告侧重从供给侧和需求侧两个视角去看待云服务在各个垂直行业的差异化应用。 供给侧方面，研究厂商为满足不同行业客户需求所提供的针对性解决方案。而需求侧则强调云服务在垂直行业的应用形态。

点击阅读完整版报告

- 5 月 18 日 CNCF 启动云原生网络功能认证计划

云原生网络功能（Cloud Native Network Function，CNF）认证计划旨在帮助通信服务提供商和其他电信组织确定哪些网络设备提供商遵循云原生最佳实践。 CNF 认证程序使用 CNCF 的 CNF 测试套件运行，并支持在经过获得认证的 Kubernetes 环境中运行的任何产品。 CNF 测试套件是个开源测试套件（包括 Fluentd、Helm、Jaeger 和 Prometheus 等），用于电信公司验证他们遵循云原生原则和最佳实践的情况，如不可变基础设施和声明式 API。

- 5 月 18 日云原生成熟度模型 v2.0 出炉

云原生成熟度模型由 CNCF 社区 Cartografos 工作组推出，旨在帮助采用者和最终用户在 CNCF 景观和更广泛的云原生生态系统中导航。 成熟度模型包括 5 个级别，每个级别涵盖人员、流程、策略和技术。

v2.0 更新包括三个方面：

- 新增业务成果部分：该部分包括云原生相关的业务目标、定量和定性 KPI 等
- 流程更新：更新 GitOps 和 CI/CD 相关内容

- 技术更新：致力于提升安全性和集成基础设施即代码

点击查阅云原生成熟度模型表格

- 5 月 18 日 CNCF Security TAG 发布最新版本云原生安全白皮书

CNCF Security TAG（安全技术咨询小组）发布云原生安全白皮书 v2，以帮助社区了解保护云原生部署的最佳实践。 该白皮书旨在让组织及其技术领导，清楚地了解云原生安全、各生命周期的安全，以及确定 NIST SSDF 等标准适用性的考虑因素。

新版本增加的内容有：

- 安全默认值 Cloud Native 8：关于实施默认情况下安全的云原生应用的高级指南；
- SSDF 1.1 版映射：将 NIST SSDF 的实践和任务映射到云原生安全应用生命周期；
- ATT&CK 容器威胁矩阵：总结威胁矩阵如何提供白皮书指南的结构。

点击查阅云原生安全白皮书 v2

- 5 月 17 日 IDC 发布《中国公有云服务市场（2021 下半年）跟踪》报告

报告显示，2021 年下半年中国公有云服务整体市场规模（IaaS/PaaS/SaaS）达到 151.3 亿美元，其中 IaaS 市场同比增长 40.1%，PaaS 市场同比增速为 55.7%。 预计到 2026 年，中国公有云市场规模将达到 1057.6 亿美元，全球占比将从 2021 年的 6.7% 提升为 9.9%。

下半年中国公有云市场发展的主要特点有：

- 企业数字化转型加速推动云需求保持高增长态势
- 部分互联网行业增长放缓
- “双碳”政策和数据安全法规的推动

- 云服务商向关注可持续发展和行业生态能力建设迈进

- 5 月 14 日云原生 Meetup 成都站开启

  今年的次站 Meetup 将于 5 月 14 日 在成都开启。 此次 Meetup 由 KubeSphere 社区成都用户委员会联合 KubeSphere 社区及 SOFAStack 社区发起，主题为「云原生技术交流」。 本次活动议题围绕 DevOps、容器镜像、K8s 分级管理、云原生应用配置等话题。

- 5 月 13 日青云开源函数即服务项目 OpenFunction 加入 CNCF 沙箱项目

  近日，OpenFunction 通过 CNCF 技术监督委员会（TOC）的评定，成为 CNCF 沙箱项目。 OpenFunction 是一个函数即服务 FaaS 项目，旨在帮助开发者专注于业务逻辑，不必担心底层运行环境和基础设施，用户只需提交一段代码，就可以生成事件驱动、动态伸缩的 Serverless 工作负载。

- 5 月 4 日 Forrester 调查显示: 96% 的企业都曾面临云安全问题

  Forrester 在对 AWS 和公共云安全公司 Sonrai Security 的 154 名云安全从业者的调查中发现，96% 的企业都曾面临云安全问题，98% 的攻击与运行云应用的系统和服务身份相关。 此外，企业普遍面临的云安全问题还包括：云安全传统工具不能很好地与云整合、访问控制策略过于复杂、监管合规问题以及过度授权用户等。

- 5 月 2 日至 4 日 Linux Storage、Filesystem、Memory Management & BPF 峰会召开

  Linux 存储、文件系统、内存管理和 BPF 峰会聚焦 Linux 文件系统、存储和内存管理子系统的优化改进，这些改进将会在会议结束后 2–3 年内进入主线内核和 Linux 发行。

2022-04

- 2022 全球分布式云大会·北京站将于 4 月底在北京拉开帷幕（内附报名方式）

本次大会将围绕边缘计算发展全新趋势，围绕边缘云、5G MEC、边缘加速三大核心议题展开，邀请在华北地区布局的边缘云、边缘加速的相关企业参会，助力华北地区边缘计算的应用普及。

- 4 月 27 日青云开源的函数即服务项目 OpenFunction 成为 CNCF 沙箱项目

  4 月 27 日，OpenFunction 通过 CNCF 技术监督委员会（TOC）的投票，正式进入 CNCF 沙箱托管。 OpenFunction 帮助开发者专注于业务逻辑，而不必担心底层运行环境和基础设施。用户只需提交一段代码，就可以生成事件驱动的、动态伸缩的 Serverless 工作负载。

- 4 月 26 日 sealer 成为 CNCF Sandbox 项目，旨在构建分布式应用交付新标准

  4 月 26 日，CNCF TOC 例会投票，一致通过 sealer 开源项目成为 CNCF 沙箱项目。 sealer 项目诞生于阿里云智能，旨在解决分布式软件的打包、分发与运行的标准化问题。

  GitHub 地址：https://github.com/sealerio/sealer

- 4 月 25 到 29 日 IstioCon 2022 议程公布

  IstioCon 2022 议程已正式公布，本次活动为线上会议，时间为 4 月 25 到 29 日。本次会议有适应北京时间的中文演讲。

- 4 月 25 – 29 日 IstioCon 2022 开幕

  北京时间 4 月 25 – 29 日，第二届 IstioCon IstioCon 以线上会议的方式举行，来自华为、Google、IBM 等公司的 80+ 服务网格技术专家带来了 60+ 场技术分享。 大会聚焦社区新特性、生产落地案例、动手实战、社区生态发展等话题。

- 4 月 25 日 Istio 申请加入 CNCF

4 月 25 日，在 Istio 委员会的支持下，谷歌已经提交了 Istio 加入 CNCF 的申请提案。Istio 的加入会让云原生堆栈变得更加完善。

- 4 月 20 日云原生虚拟机运行项目 KubeVirt 升级为 CNCF 孵化项目

  CNCF 技术监督委员会已经投票接受 KubeVirt 作为 CNCF 的孵化项目。 KubeVirt 是 Redhat 开源的以容器方式运行虚拟机的项目，以 k8s add-on 方式，利用 k8s CRD 为增加资源类型 VirtualMachineInstance（VMI）， 使用容器的 image registry 去创建虚拟机并提供 VM 生命周期管理。

- 4 月 18 日 Carina 本地存储近日入选 CNCF 云原生全景图

  Carina 于 2021 年 10 月由博云正式开源，是一款专注于本地存储的解决方案。 Carina 基于 Kubernetes 及 LVM，提供了数据库与中间件等有状态应用在 Kubernetes 中运行所需的本地存储能力，减少了存储系统的运维压力。

  GitHub 地址：https://github.com/carina-io/carina

- 4 月 13 日艾瑞咨询发布 3 月云服务行业动态及热点研究月报

  月报基于 2022 年 1 – 3 月对云服务行业各赛道动态的观察，得出观点：SaaS 仍是最备受青睐的投资焦点 。 IaaS 赛道中，互联网类云厂商仍占据该市场主导。

  IaaS 与 SaaS 厂商均有参与 PaaS 平台开发与应用的行动。PaaS 细分赛道中，低代码成为热门。

- 4 月 10 日：2021 年云原生下的智能营销研究报告

  智能营销作为数字产业化的核心场景之一，在企业数字化转型中广泛落地。面向市场解决方案多元割裂、客户效果诉求提升等现状，云原生与智能营销的融合为企业提供了更优的解决方案。

点击下载完整报告

- 4 月 9 日云原生 Meetup 成都站延期

  由于疫情，KubeSphere 社区宣布原定于 4 月 9 日的成都 Meetup 将再次延期举办。最终举办时间，视疫情发展情况另行通知。在此期间，仍可以进行报名。

- 4 月 8 日申报启动！“分布式云全景生态图谱”优秀案例征集（内附申报方式）

  值 2022 全球分布式云大会召开之际，“分布式云全景生态图谱”计划第一阶段正式启动。 面向全行业征集边缘计算、云原生、数据库、分布式存储等领域的优秀案例。 入选企业将获得全球分布式云联盟组织的专访和专题报道。

- 4 月 7 日云原生批处理系统 Volcano 升级到 CNCF 孵化阶段

  Volcano 是一个云原生批处理系统，也是 CNCF 的第一个批处理计算项目。 4 月 7 日，CNCF 技术监督委员会（TOC）投票接受 Volcano 作为 CNCF 孵化项目。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

排障方案

本页列出云原生社区的一些故障排查方案，供运维人员参考。

- 如何解决 Prometheus 的存储容量问题?

  单机存储不好扩展是 Prometheus 一个广受诟病的问题。对此问题，文章给出了 3 种集群解决方案：Prometheus 联邦集群、远程存储方案、Prometheus 自身搭建集群，这这三种方案能够很好地解决 Prometheus 的存储问题。

- 云原生场景下，nettrace 怎么快速进行网络故障诊断?

  nettrace 是一款基于 eBPF 的集网络报文跟踪（故障定位）、网络故障诊断、网络异常监控于一体的网络工具集， 提供一种更加高效、易用的方法来解决复杂场景下的网络问题。

- Istio 运维实战：如何规避 503 UC 错误

  503 UC 是在 Istio/Envoy 使用过程中经常遇到的一个问题。 文章分析了由于 TCP 服务器 keepalive Timeout 导致的 503 UC 的原理及规避方法。

- K8s CNI Cilium 网络故障排查指南

  文章介绍了作者从 kubenet 升级到 cilium 时遇到的一个问题及排障过程。 问题是，在 kubenet 网络的节点上的 pod 无法访问 cilium 网络的节点上的 pod。 通过抓包测试，以及分析 cilium 的相关信息， 发现问题所在：cilium 如果没有管理所有 k8s 节点，剩余的节点就按照外部服务器的方式来处理。

- 如何高效地对 Kubernetes Service 进行排障

文章简要介绍了 service 的工作方式，在运行 service 时可能出现的各种 bug 以及如何解决这些 bug。 最后，还介绍了一个高效排障工具 Telepresence。 Telepresence 能够为 Kubernetes 建立一个远程开发环境，用户仍然可以使用自己的本地工具，如 IDE 和调试器， 并针对远程 K8 集群的微服务测试本地服务。

- K8s：彻底解决节点本地存储被撑爆的问题

  K8s 节点本地存储保存了镜像、可写层、日志、emptyDir 等内容。 除了容器镜像是系统机制控制的，其它都跟应用程序有关。 完全依赖开发者对应用使用的存储空间进行限制并不可靠，因此，K8s 提供了垃圾回收、日志总量限制、emptyDir 卷限制、临时数据总量限制来避免本地存储不够用的问题。

- K8s 有损发布问题探究

  应用发布过程其实是新 Pod 上线和旧 Pod 下线的过程，当流量路由规则的更新与应用 Pod 上下线配合出现问题时，就会出现流量损失。 总的来说，为了保证流量无损，需要从网关参数和 Pod 生命周期探针和钩子来保证流量路径和 Pod 上下线的默契配合。 文章从 K8s 的流量路径入手，分析有损发布产生的原因，并给出解决方案。

- K8s：Pod 的 13 种异常

  文章总结了 K8s Pod 的 13 种常见异常场景，给出各个场景的常见错误状态，分析其原因和排查思路。

- K8s: 你需要知道的一切

  电子书囊括了了解 Kubernetes 所需的所有定义，同时也包含了与 Kubernetes 相关的其他电子书、文章、教学视频等资源的介绍，以及 7 个最佳实践。

- Docker Volume 引发 K8s Terminating Pod 的问题

Terminating Pod 是业务容器化后遇到的一个典型问题，诱因不一。 文章记录了网易数帆-轻舟 Kubernetes 增强技术团队如何一步步排查，发现 Docker Volume 目录过多导致 Terminating Pod 问题的经历，并给出了解决方案。

- 安装 Kubernetes 或升级主机操作系统后出现网络问题？这可能是你的 iptables 版本有问题

  安装新版本的 Kubernetes 后，你的工作节点连接不上网络，突然间无法 ssh 访问甚至无法 ping 它们？ 这可能是因为 kube-router 1.25 的 iptables 版本和你安装的版本之间出现冲突。 归根结底是因为 iptables 1.8.8 和旧版本的规则格式不兼容。 对此，有三个解决方案：将 iptables 版本降到 1.8.7、清理 IPTables 链所有权（alpha）或是借助轻量级的 Kubernetes 发行版 k0s 解决。

- 利用 eBPF 进行 Kubernetes 集群磁盘 I/O 性能问题排查

  问题始于 eBay 工程师发现他们的 Kafka 服务在有些时候 follower 追不上 leader 的数据。 为解决问题，利用了一些 eBPF 工具，例如，biopattern 用于展示 disk I/O pattern，ebpf_exporter 用于数据收集和可视化。

- Kubernetes 网络排错骨灰级指南

  文章介绍 Kubernetes 集群中网络排查的思路，包括 Pod 常见网络异常分类、排查工具、排查思路及流程模型、CNI 网络异常排查步骤以及案例学习。

- Redis 缓存异常及处理方案总结

  Redis 在实际应用过程中，会存在缓存雪崩、缓存击穿和缓存穿透等异常情况，如果忽视这些情况可能会带来灾难性的后果，文章对这些缓存异常问题和相应处理方案进行了分析和总结。

- K8S Internals 系列：存储卷指标消失之谜

  这篇文章的创作灵感因 Grafana 不展示使用存储驱动创建的存储卷的容量指标而起，展示了该问题的排查思路及手段，进而勾勒出 Kubelet 对于存储卷指标收集的实现流程。

安全漏洞

本页列出云原生社区近年来出现的一些重大安全漏洞。

- 回顾 2022 Kubernetes 安全漏洞，我们可以从中学到什么

  文章对 2022 年 Kubernetes 的主要安全漏洞和解决方法进行了总结，例如 CRI-O 运行时容器逃逸漏洞、ArgoCD 绕过认证等。 介绍了一些防止漏洞的措施：实施安全配置文件，分配角色和权限时遵循最小权限原则，对 K8s 清单文件、代码库和集群进行持续扫描，定期更新集群上的软件包，使用容器沙盒项目等。

- Istio 高风险漏洞：拥有 localhost 访问权限的用户可以冒充任何工作负载的身份

  如果用户拥有 Istiod 控制平面的 localhost 访问权，他们可以冒充服务网格内的任何工作负载身份。 受影响的版本为 1.15.2。目前，已发布补丁版本 1.15.3。

- Istio 高风险漏洞：Golang Regex 库导致 DoS 攻击

  Istiod 存在请求处理错误漏洞，攻击者会在 Kubernetes validating 或 mutating webhook 服务曝光时，发送自定义或超大消息，导致控制平面崩溃。 目前，Istio 已发布补丁版本 1.15.2、1.14.5 和 1.13.9。低于 1.14.4、1.13.8 或 1.12.9 的版本会受此影响。

- CrowdStrike 发现针对 Docker 和 Kubernetes 基础设施的新型挖矿攻击

  该攻击通过容器逃逸技术和匿名矿池，使用一个模糊的域名来传递其有效负载，以对 Docker 和 Kubernetes 基础设施开展加密货币挖掘活动。 采用云安全保护平台能够有效保护云环境免受类似的挖矿活动的影响，防止错误配置和控制平面攻击。

- Kube-apiserver CVE 漏洞：聚合 API server 可能导致服务器请求伪造问题（SSRF）

  攻击者可能控制聚合 API Server，将客户端流量重定向到任何 URL。这可能导致客户端执行意外操作，或将客户端证书泄露给第三方。 此问题没有缓解措施。集群管理员应注意保护聚合 API Server，不允许不受信任方访问 mutate API Services。 受影响版本：kube-apiserver 1.25.0、1.24.0 – 1.24.4、1.23.0 – 1.23.10、1.22.0 – 1.22.14。已修复版本：1.25.1、1.24.5、1.23.11、1.22.14。

- Aqua 发现新型非法加密挖矿方式，能够利用容器消耗网络带宽

  近日，Aqua Security 发布了一则关于新型挖矿攻击的警报。较传统的挖矿攻击会造成 CPU 消耗的急剧增加，新型攻击利用容器消耗网络带宽，而 CPU 消耗增加并不显著。 因此，依靠 CPU 利用率来识别攻击的安全工具可能无法发现该威胁。 Aqua 建议，运行能够静态和动态分析容器的安全工具是最有效的抵御攻击的方式。

- Istio 漏洞：向 Envoy 发送格式不正确的头信息可能导致未知内存访问

  在受影响的版本（1.14.2 和 1.13.6）中，在某些配置中向 Envoy 发送的格式错误头信息可能导致未知行为或崩溃。该漏洞已在 1.12.8、1.13.5 和 1.14.1 版本中得到解决。此外，Istio 官方建议不要在生产环境中安装 1.14.2 或 1.13.6。

- CRI-O 漏洞：Kube API 访问可能会导致节点上的内存耗尽

  CRI-O 会读取 ExecSync 请求的输出结果。如果输出的数据量很大，有可能耗尽节点的内存或磁盘空间，对系统可用性造成威胁。目前，该文章尚未给出解决方案。

- Istio 控制平面存在请求处理漏洞

Istiod 存在请求处理漏洞，发送定制信息或超大信息的攻击者可能会导致控制平面进程崩溃。 当 Kubernetes 验证或变更 webhook 服务公开暴露时，受攻击的风险最大。此漏洞为 0day 漏洞。

知识分享

分享一些云原生相关的技术文章，愿大家携手共进。

- 排障方案
- 最佳实践
- 工具推荐
- 产品选型
- 前沿热点
- 安全漏洞
- 电子书
- 其他

前沿热点

本页列出云原生社区的一些前沿热点，便于大家了解潮流和方向。更多请参阅 OSS Insight。

- 国内首例社区双栈 Istio 方案落地经验，实现代码已开源

  在业界广泛应用的服务网格项目（以 Istio 和 Linkerd 为例）中，双栈技术的支持依然是缺失的。 为更好的实现服务网格与 Kubernetes 在双栈支持上的协同工作， 文章主要介绍了 Istio 在双栈技术中的实现方案以及该方案在移动云的落地实现场景。 但是，此方案只是实验性的分支，并没有完善单元测试和集成测试。

- 微服务进入深水区后该何去何从

  微服务进入深水区后，开发者和架构师更关心的是微服务安全、稳定性、成本优化、微服务治理标准化，以及促使云原生微服务架构逐步演进到多运行时微服务架构。 而对于多运行时架构存在的局限性，需要实现更加标准化与平台化的服务网格开发与运维能力，规范化 Sidecar 与运行时的定义，同时将运维平台变得更加标准易用。

- 2023 年云原生发展趋势的预测

  文章是 CNCF CTO Chris Aniszczyk 关于 2023 年云原生和技术领域的热点话题的预测， 包括：云原生 IDE 成为常态、FinOps 成为主流并左移、开源 SBOM 无处不在、GitOps 走向成熟并进入实质生产高峰期、OpenTelemetry 日益成熟、Backstage 开发者门户继续完善、WebAssembly 进入稳步爬升期等等。

- 探索基于隧道的 Kubernetes 跨集群通讯

文章介绍了基于 ssh 隧道的跨集群访问访问。 集群 A 中服务要访问集群 B 中的不同服务，即单隧道多服务的实现是通过在隧道两头增加一个隧道的代理，隧道左端监听多个端口，用来区分集群 A 中服务要访问的集群 B 中的不同服务。 并且将此信息告知隧道右端的代理，右端代理根据此信息来转发给对应的集群 B 中的服务。 目前该方案只是在 demo 的程度。

- 2023 年 eBPF 的六个发展趋势

  2023 年 eBPF 的六个发展趋势包括： 利用 eBPF 网络追踪建立高性能的 HTTP 监控、更深入的网络功能和 sidecar 优化、安全和恶意软件检测、云平台的大规模应用、电信领域的应用、更多基于 eBPF 的项目涌现。

- Kubernetes 之痛，平台工程可以解决

  构建平台工程所需的产品方法有助于组织了解尚未提供支持的 Kubernetes 领域。 开发者自助服务允许工程师自行调配和使用测试、保护和部署应用所需的技术，而无需等待运维提供资源或启动环境。 内部开发平台使开发和运维专注于各自的核心职责和优势，开发人员专注于编写代码，运维向上游转移到更关键的任务，如管理网络或物理硬件。

- 信创如何为业务带来价值?

  提到信创，大家第一反应就是自主可控。但要做好信创，需要从埋头于自主可控中抽出一部分精力，思考如何从信创中获取价值。 文章介绍了国内外技术替代的成功案例，技术替换在什么情况下能为业务带来价值，以及如何做能够让信创更好地为业务服务。

- 从 PingCAP 的 TiDB 数据库产品看国产技术出海模式

  文章介绍了 PingCAP 为实现技术出海，对 TiDB 在技术上、合规性、商业化方面所做的优化。 技术上，要实现降本增效、运维自动化、多租户管理、满足特定地区的场景需求； 合规性上，考虑数据安全、监管规则； 商业上，考虑计价模式、商业化策略等。 此外，TiDB 能够吸引海外客户还

有几个因素：TiDB 是开源的，有着活跃的开源社区； 客户数据存放在 AWS、GCP 等公有云上； 远程支持，不依赖当地技术团队，可以有效解决部分地区人力资源贵的问题。

- 探索 K8s 新功能 Container Checkpointing

  容器检查点功能（K8s 1.25 alpha）为某个正在运行的容器创建一个状态点的快照，并将其保存到磁盘中。 之后可以使用此检查点启动容器，恢复状态，或者将容器迁移到其他的机器上。 文章介绍了这个功能的工作原理、Podman 的检查点功能、CRIU 以及取证分析等应用场景。

- OCI 容器与 Wasm 初体验

  文章介绍如何通过配置，让 OCI 运行时运行 Linux 容器和与 WASI 兼容的工作负载。

- Service Mesh 的下一站是 Sidecarless 吗?

  尽管 Cilium、Linkerd、Istio 等几大开源社区都在 Sidecarless 领域进行了各自的探索和实践，但是各家在安全性，稳定性，管理成本，资源占用上是各有侧重点的，适应不同的业务场景。

- eBPF 程序摄像头——力争解决可观测性领域未来最有价值且最有挑战的难题

  当前可观测性用户很容易迷失在指标迷阵当中，不知该在什么时间查看何种指标，如何理解大规模细粒度的指标。 为解决此问题，Kindling 社区选择了基于 eBPF 的可观测性摄像头，按照 eBPF 粒度去获取程序执行过程当中的细粒度指标，帮助用户理解程序执行的真实过程，同时理解细粒度的指标是如何影响程序执行的。

- GitOps 是皇帝的新衣吗

  在采用 GitOps 前，我们需要了解清楚“什么是 GitOps？”，并问自己“我们使用这些工具为谁提供服务？我们试图解决什么问题？”文章针对

GitOps 的一些主要“卖点”（包括安全性、版本控制和环境历史、回滚、飘逸处理、单一真相来源等）提出了质疑，并介绍了 GitOps 带来的一些挑战。

- 蚂蚁规模化平台工程实践两年多，我们学到了什么

  平台工程是一门设计与构建工具链和工作流程的学科，可以为云原生时代的软件工程组织提供自助式服务功能。 文章基于可编程云原生协议栈 KusionStack 在蚂蚁平台工程及自动化中的实践，从平台工程、专用语言、分治、建模、自动化和协同文化等几个角度，阐述规模化平台工程实践中的收益和挑战。

- 可观测可回溯 | 持续性能分析 Continuous Profiling 实践解析

  持续性能分析（简称 CP）对于开发者的意义在于，在生产环节，永远知道代码的所有执行细节。 文章在介绍 CP 发展历史的基础上，分析了性能分析 profiling 的 2 个关键点：获取堆栈数据和生成火焰图，以及常见的 profiling tool。

- 下一代云原生边缘设备管理标准 DMI 的设计与实现

  DMI 整合设备管理接口，优化边缘计算场景下的设备管理能力； 同时定义了 EdgeCore 与 Mapper 之间统一的连接入口，并分别由 EdgeCore 和 Mapper 实现上行数据流和下行数据流的服务端和客户端，承载 DMI 具体功能。

- 使用 eBPF LSM 热修复 Linux 内核漏洞

  Linux Security Modules（LSM）是一个基于 hook 的框架，用于在 Linux 内核中实现安全策略和强制访问控制。 文章介绍了 eBPF LSM 的实现思路（核心内容是确定 hook 点），如何使用 unshare 将 user 映射到 root，以及如何通过在 eBPF 中实现程序来解决真实场景问题。 最后，对比了这个 LSM 程序的性能影响。

- 关于 Cilium Service Mesh，你需要知道的一切

  Cilium 在最新发布 v1.12 中正式推出 Cilium Service Mesh。 在 Cilium Service Mesh 中，除有多种不同的控制平面可供选择外，用户还能够灵活地选择运行有/无 sidecar 模型的服务网格。

- 使用 eBPF 准确定位服务网格的关键性能问题

  SkyWalking Rover 是 SkyWalking 生态系统中引入的 eBPF profiling 功能，可实现 CPU profiling、off-CPU profiling 等。 文章讨论了如何使用 eBPF 技术来改进 SkyWalking 中的剖析功能，并用于分析服务网格中的性能影响。

- coolbpf 技术实践，将 BPF 程序开发效率提升百倍

  coolbpf 是由龙蜥社区开源的一站式 BPF 开发编译平台，旨在解决 BPF 在不同系统平台的运行和生产效率提升问题。 coolbpf 的六大功能：本地编译服务，基础库封装；远程编译服务；高版本特性通过 kernel module 方式补齐到低版本；自动生成 BTF；各内核版本功能测试自动化；Python、Rust、Go、C 等高级语言支持。

- 阿里云容器服务负责人对云原生与软件供应链安全的思考

  文章作者介绍了目前容器化软件供应链安全的最新实践和工具链，包括容器镜像的 SBOM 支持，以及新一代镜像签名技术—— Sigstore 的无密钥签名 Keyless Signatures 模式。

- eBPF、Sidecar 和服务网格的未来

  Linkerd 团队认为，至少在可预见的未来，eBPF 服务网格仍然需要 sidecar 代理。 一方面，eBPF 的局限性意味着 L7 流量代理仍然需要用户空间网络代理来完成工作； 另一方面，同 sidecar 相比，基于主机的代理在操作、维护和安全性方面都更差。 而 Linkerd 未来将继续利用 eBPF 技术让 sidecar 代理变得更小更轻快。

- 通过模糊测试提高 CNCF 项目的安全性

  模糊测试是一种测试软件的技术，将“随机数据”传递到目标应用中，观察目标应用是否崩溃。 OSS-Fuzz 是一个开源项目，为持续模糊测试提供了一个自动化平台。截止目前，共有 18 个 CNCF 项目通过 OSS-Fuzz 进行持续模糊测试。
  点击了解 CNCF 模糊测试的更多细节

- 什么是持续分析?

  持续分析（continuous profiling）是一种分析程序复杂性的动态方法，通过了解系统在一段时间内的资源，定位、调试和修复与性能有关的问题。 CNCF 的 CTO 认为，持续分析是可观测性堆栈不可或缺的一部分。 文章介绍了与此相关的概念、行业动态以及开源持续性能分析平台。
  点击访问 Pyroscope 项目地址

- Envoy Gateway 计划提供一个标准化 Kubernetes 入口

  Envoy Gateway 计划使 Envoy 反向代理成为一个网络网关，不仅可以引导内部微服务流量，还可以管理进入网络的外部流量。 其根本目的是为 Kubernetes 建立一套标准化、简化的 API。

- 使用 WebAssembly 验证请求负载

  这篇文章介绍了如何使用 Wasm 插件来扩展 Istio——验证一个请求的有效负载。 当你需要添加 Envoy 或 Istio 不支持的自定义功能时，可以使用 Wasm 插件，如使用 Wasm 插件来添加自定义验证、认证、日志或管理配额。

- OPLG：新一代云原生可观测最佳实践

  OPLG 是指将 OpenTelemetry Traces、Prometheus Metrics、Loki Logs 通过 Grafana Dashboards 进行统一展示，满足企业级监控与分析的大部分场景。 基于 OPLG 体系可以快速构建一套覆盖云原生应用全栈

的统一可观测平台，全面监测基础设施、容器、中间件、应用及终端用户体验。

- Kubernetes 软件供应链安全必备四要素

  这篇文章介绍了 Kubernetes DevOps 团队理解软件供应链安全所需的四个要素：Artifacts、Metadata、Attestations 和 Policies（A-MAP）。 软件构建系统产生工件和元数据，验证构建完整性和软件组件安全属性需要证明和策略。

产品选型

本页列出一些基于云原生社区构建的商业化产品。

- Kubernetes 策略引擎：OPA vs. Kyverno vs. jsPolicy

  文章讨论了 Kubernetes 策略引擎的概念，并比较了三种不同的 Kubernetes 策略引擎：OPA、Kyverno 和 jsPolicy。 结论是：如果想要更直接和简单的方法，或者精通 JavaScript 和 TypeScript，推荐使用 jsPolicy。 如果更喜欢 YAML，并且希望继续直接使用 Kubernetes 资源，Kyverno 是一个不错的选择。

- 过去二十年里开源的 12 个开源监控工具大对比

  文章对 12 款典型的开源监控工具做了简要的介绍和分析，指出各自的优势和不足。 其中提到的工具包括分布式监控系统 Zabbix、时序数据库 VictoriaMetrics、Prometheus、云原生监控分析系统夜莺监控等。

- 云原生存储工具的选型和应用探讨

  文章逐步梳理了云原生存储的概念，并对 Longhorn、OpenEBS、Rook+Ceph 进行简要介绍和横向对比，最后选择了具有代表性的 Longhorn 演示其安装和使用。

- K8s CNI 插件选型和应用场景探讨

  文章介绍容器环境常见七个网络应用场景及对应场景的 Kubernetes CNI 插件功能实现。

- 云原生时代的 DevOps 平台设计之道（Rancher vs KubeSphere vs Rainbond）

文章重点介绍了 Rancher 、KubeSphere、Rainbond 三款云原生平台级产品各自不同的 DevOps 实现。 作者认为，DevOps 团队可以选择 Rancher + KubeSphere 或 Rancher + Rainbond 的组合。 Rancher 最擅长向下对接基础设施，管理集群的安全性与合规性，而向上为开发人员提供易用的云原生平台则交给 KubeSphere 或 Rainbond。

- 到底谁强？Grafana Mimir 和 VictoriaMetrics 性能测试

  文章比较 VictoriaMetrics 和 Grafana Mimir 集群在相同硬件上运行的工作负载的性能和资源使用情况。 在基准测试中，与 Mimir 相比，VictoriaMetrics 表现出更高的资源效率和性能。从操作上讲，VictoriaMetrics 扩展更为复杂，Mimir 可以很容易实现组件扩展。

- 一文读懂 Prometheus 长期存储主流方案

  由于 Prometheus 存在跨集群聚合、长时间存储等局限性，社区给出了多种扩展方案。 文章对包括 M3、VictoriaMetrics、Thanos、Cortex、Grafana Mimir 在内的 5 种主流 Prometheus 长期存储方案进行了多维度对比分析。

- eBPF 基础设施库的技术选型

  文章对几个 eBPF 基础设施库进行比较，如 libbcc、dropbox 的 goebpf、Cilium 的 ebpf 库、Calico 的底层库、falco 的 lib 库， 并说明开源可观测性工具 Kindling 选择 falco-libs 的原因。

- 在生产环境如何选择靠谱的 APM 系统

  文章从主流 APM 产品介绍出发（对比 Pinpoint、Jaeger、Skywalking、听云、腾讯云+阿里云 Arms 和 Datadog），通过生产环境中关注的几个重要维度，如产品体验、Agent 能力、报警+ DB 支持、云原生的支持能力、数据大屏等，给予 APM 选型方案建议。

电子书

本页列出一些源自云原生社区的电子书籍。

- 《云原生架构容器&微服务优秀案例集》

  案例集是相关客户通过阿里云云原生产品构建生产业务体系的经验分享及最佳实践总结。 案例集以容器微服务为中心共有 37 个企业案例，涵盖 7 大行业，覆盖 ACK、MSE、ACR、ASK、ACK One、ASM、可观测等 10+ 产品。

- 白皮书《云原生成熟度矩阵评估》中文版

  白皮书为正处于云原生转型的企业总结了一套云原生成熟度矩阵评估流程。从文化、产品、团队、流程、架构、维护、交付、配置、基础设施 9 个领域评估组织当前所处的状态。 同时，阐明 4 种经典的云原生转型误区，并总结出典型的云原生转型方案。
  点击阅读白皮书，提取码：kvrv

- 电子书《零信任的基石：使用 SPIFFE 为基础设施创建通用身份》

  这本书介绍了服务身份的 SPIFFE 标准，以及 SPIFFE 的参考实现 SPIRE。 深入阐述了如何设计一个 SPIRE 部署、与其他系统集成的方法、如何使用 SPIFFE 身份通知授权、与其他安全技术对比等。

- Kube-OVN v1.10 系列中文文档，99.9% 的问题都能在这里找到答案

  文档分为五个部分：Kube-OVN 快速入门、使用指南、运维指南、高级功能和技术参考。
  点击查阅文档

- 从构建到治理，业内首本微服务治理技术白皮书正式发布

阿里云云原生微服务团队近日发布《微服务治理技术白皮书》。 该白皮书聚焦微服务治理业务领域，囊括了技术原理、业务场景、解决方案、最佳实践等微服务落地的全流程。

其他

本页列出一些云原生社区的其他事项和文章。

- 云原生如何改变电信标准：自上而下与自下而上的对决

  文章介绍了云原生自下而上的开发原则、最佳实践和事实标准的逻辑，以及自上而下的、由委员会驱动的标准，如电信行业的标准。 而自上而下和自下而上的标准相遇会发生什么，当协议的管理者与云原生标准相遇时又会发生什么?

- 接管 K8s 部署运维，基础架构团队是否做好准备?

  Kubernetes 非常适合作为一种 Infrastructure as a Service（IaaS），既可以在公有云上提供，也可以在企业私有云上作为新一代的基础架构，为现代化应用提供可靠的基础设施。 因此，Kubernetes 的建设和运维也将更多地交由基础架构团队负责。 文章探讨了为什么基础架构团队适合负责 Kubernetes 的运维管理，以及这样的转变是否可行。

- 2022 云原生技术盘点与展望

  文章从底层基础技术和场景化的应用技术两方面，对 2022 年云原生的发展情况进行盘点：混部带来了效率的提升； Serverless 基于容器完成标准化；Service Mesh 进行新尝试，落地方式还在探索；降本增效大主题下，FinOps 理念快速发展等。

- K8s 原生支持的准入策略管理

  Kubernetes 1.26 提供了一个 alpha 版本的验证准入策略更新，即在验证准入策略时可以使用一种通用的表达式语言（CEL）来提供声明的、进程内的替代方法来验证 validating admission webhook。将 CEL 表达式嵌入到 Kubernetes 资源中，大大降低了 admission webhook 的复杂性。

- Containerd 深度剖析-NRI 篇

  NRI 是 containerd 中的 CRI 插件，提供一个容器运行时级别的插件框架来管理节点资源。 NRI 可用来解决批量计算、延迟敏感性服务的性能问题，以及满足服务 SLA/SLO、优先级等用户需求， 比如通过将容器的 CPU 分配同一个 numa node，来保证 numa 内的内存调用。 当然除了 numa，还有 CPU、L3 cache 等资源拓扑亲和性。

- Kubernetes 证书管理系列

  文章介绍了 Kubernetes 中的证书以及 Kubernetes 证书管理器在实际生产中的作用。然后以证书管理项目 cert-manager 为例，阐释其架构、组件、生态兼容等。

- 2022 年容器生态系统的 9 大趋势洞察

  Datadog 对客户运行的超 15 亿个容器进行了分析，总结出容器生态系统的主要趋势： 无服务器容器技术在公共云中的使用率持续上升、多云使用率和组织的容器数量正相关、Kubernetes Ingress 使用率正在上升、大多数主机使用超过 18 个月的 Kubernetes 版本、超过 30% 的运行 containerd 的主机使用不受支持的版本、NGINX、Redis 和 Postgres 是最受欢迎的容器镜像。

- Karmada 大规模测试报告发布，突破 100 集群和 50 万节点

  近日，Karmada 社区对 Karmada 开展了大规模测试工作。根据测试结果分析，以 Karmada 为核心的集群联邦可以稳定支持 100 个集群和 50 万个节点同时在线，管理超过 200 万个 Pod。 在使用场景方面，Push 模式适用于管理公有云上的 Kubernetes 集群，而 Pull 模式相对于 Push 模式涵盖了私有云和边缘相关的场景。 在性能和安全性方面，Pull 模式的整体性能要优于 Push 模式。

- toB 应用私有化交付技术发展历程和对比

在传统应用交付中，管理运行环境和操作系统差异是一个痛点。当前云原生应用交付使用容器和 kubernetes 相关技术解决了这个问题，但是这些技术的学习和使用门槛太高。 因而，抽象的应用模型成为下一代解决方案，例如，基于 OAM 的 KubeVela 应用交付和基于 RAM 的 Rainbond 应用交付。

- 国内外云厂商容器服务备份恢复方案调研

  文章对四家厂商的容器服务备份恢复方案（阿里云容器服务 ACK、腾讯云容器服务 TKE、华为云备份产品 CBR、谷歌云 Backup for GKE）进行了调研分析，比较各自的优缺点。

- 云原生网关 Apache APISIX 3.0 与 Kong 3.0 功能背后的趋势

  文章详细分析了两个 API 网关项目 APISIX 和 Kong 的最新版本，试图从更新细节洞察有价值的技术趋势。 Kong 3.0 开始逐渐倾向于企业版，侧重政府、金融业以及对安全合规更关注的大型企业。 而 APISIX 3.0 版本推出的所有功能都是开源的，在保证性能的同时，也在积极扩展周边生态。

- Kubernetes Node 组件指标梳理

  文章对 kubelet 自身指标、kube-proxy 指标、kube-state-metrics 指标以及 cadvisor 指标进行了完整梳理。

- Kubernetes 1.25 中的删除和主要变化

  v1.25 的删除和主要变化：删除 PodSecurityPolicy、核心 CSI 迁移功能升级为 GA、弃用 GlusterFS、Kubernetes 发布工件签名升级为 Beta、cgroup v2 支持升级为 GA、清理 IPTables 链所有权。

- 解读 Curve 资源占用之内存管理

文章结合分布式存储系统 Curve 说明内存布局，以及内存分配器的必要性和需要解决的问题， 并通过举例说明内存分配器的内存管理方法，最后介绍当前 Curve 内存分配器的选择及原因。

- Kubernetes 单机侧的驱逐策略总结

  文章总结用户空间和内核空间的驱逐流程和进程选择策略。 用户态中 Kubelet 通过驱逐来限制节点资源、pod 资源。 用户空间通过监控系统资源来触发驱逐流程，内核空间通过分配内存时触发驱逐流程。

- GitOps 成熟度检查清单

  文章介绍了一份 GitOps 检查清单，可用于评估团队 GitOps 的成熟度。清单共 16 项，包括六个维度：团队文化、git 管理、GitOps pipeline、Kubernetes、安全&策略。

- 为什么 Istio 要使用 SPIRE 做身份认证?

  Istio 1.14 中最值得关注的特性是新增对 SPIRE 的支持。 SPIFFE 统一了异构环境下的身份标准。Istio 利用 SPIRE 为每个工作负载提供一个唯一标识，服务网格中的工作负载在进行对等身份认证、请求身份认证和授权策略都会使用到服务标识，用于验证访问是否被允许。

- 弃用 Pod Security Policies 后，如何确保 Pod 安全

  Pod Security Policies（PSP） 已被弃用，未来将在 Kubernetes 1.25 中删除。 Pod Security Admission（PSA） 将取代 PSP 来实现 Pod 的安全特性，它为 pod 定义了三种隔离级别：privileged、baseline 和 restricted。此外，也可以采用外部准入控制器，如 OPA 和 Kyverno。

- CryptoMB – 加速 Istio TLS 握手

  文章介绍了 Envoy 1.20 和 Istio 1.14 中引入的一个新功能：通过 CryptoMB Private Key Provider 来加速 Istio TLS 握手。 CryptoMB 是

一个 Envoy 扩展，它利用 Intel multi-buffer 技术，使得服务网格可以卸载 TLS 握手，以处理更多连接，降低延迟并节省 CPU。

- 如何看待 Dapr、Layotto 这种多运行时架构

  文章分享了蚂蚁在落地多运行时架构 Layotto 之后的思考： Dapr API 的“可移植性”的必要性、多运行时架构的价值、与 Service Mesh、Event Mesh 的区别以及部署形态等。

- Kubernetes 1.24: 避免为 Services 分配 IP 地址时发生冲突

  Kubernetes 1.24 允许启用一个新的特性门控 ServiceIPStaticSubrange。启用此特性后，可以为 Service 使用不同的 IP 分配策略，减少冲突的风险。

- Harbor v2.5 远程复制：制品的签名如影随形

  Harbor v2.5 于上周发布，其中 Cosign 制品签名和验证解决方案是重要功能，解决了镜像等制品在远程复制中，其签名信息无法被复制到目标端的问题。 这篇文章对该功能进行了详细介绍。

最佳实践

本页列出云原生社区推崇的最佳实践，供产品和交付参考学习。

- 跨集群运行 StatefulSet

  跨集群可靠地运行 StatefulSet 可能需要解决网络和存储两个方面的问题。 文章基于一个示例，介绍了配置和管理跨集群的有状态应用服务的一个方法。

- 我们如何基于云原生项目发展开源社区?

  基于云原生项目发展开源社区，首先要知道哪些人会关心你的项目、谁是终端用户、利益相关者和贡献者? 他们做什么工作、在哪工作以及使用目标是什么? 发现更多的潜在利益相关者，为他们提供贡献途径，对提高项目的采用率来说至关重要。

- Dapr 集成 Flomesh 实现跨集群服务调用

  Flomesh 服务网格使用可编程代理 Pipy 为核心提供东西、南北向的流量管理。 通过基于 L7 的流量管理能力，突破计算环境间的网络隔离，建立一个虚拟的平面网络，使不同计算环境中应用可以互相通信。 文章介绍通过 Dapr 与 Flomesh 服务网格的集成进行跨集群的服务调用，实现“真正的”多集群互联互通。

- 云原生场景下，如何缓减容器隔离漏洞，监控内核关键路径?

  文章介绍了 OpenCloudOS 社区自研的两个方案：CgroupFS 和 SLI，用于缓减容器隔离漏洞，监控内核关键路径。 CgroupFS 方案提供一个内核态容器视角的虚拟机文件系统(/proc、/sys)，增强了容器资源视图隔离。 SLI 是容器级别的性能跟踪机制，从容器的角度对 CPU、内存资源的竞争情况进行跟踪、观测，从而为容器性能问题的定位、分析提供可靠的指标。

- 基于 Kubernetes 和 Istio 轻松完成 gRPC 到 REST 的转码

  文章介绍如何在无需修改大量代码的情况下使得 gRPC 服务与 REST 兼容。 该方案利用 Envoy 过滤器，建立一个转码器，让 RESTful JSON API 客户端通过 HTTP 发送请求，并被代理到一个 gRPC 服务。 转码器将 gRPC 服务方法的消息输出编码为 JSON，并将 HTTP 响应的 Content-Type 头设置为 application/json。

- kube-state-metrics 在大规模集群下的优化

  在小规模集群中，只需要保证 kube-state-metrics 高可用就可以在生产环境使用。 但是对于大规模的集群，只通过一个 KSM 实例来提供 metrics 指标是非常吃力的，还需要做很多优化： 例如，过滤不需要的指标和标签，通过分片降低 KSM 实例压力，使用 DaemonSet 方式单独针对 pod 指标进行部署。

- vivo 自研 Jenkins 资源调度系统设计与实践

  文章从目前业界实现 Jenkins 的高可用的实现方案入手，分析各方案的优缺点，引入 vivo 目前使用的 Jenkins 高可用方案——jenkins scheduler 系统。 该系统不采用原生的 Jenkins 部署方案，而是采用全 master 的方式，master 之间的关系、任务分配、离线、插件安装等由调度系统进行管理。 目前该系统已经投入生产环境运行。

- K8s 迁移 cgroup v2 之前要做的三件事

  随着 Kubernetes 1.25 关于 cgroup v2 特性的正式发布（GA），kubelet 容器资源管理能力得到加强。 文章在 cgroup v2 概念的基础上，介绍了采用 cgroup v2 之前在 Linux OS、K8s 生态、应用变更三方面要做的事。

- 阿里云云原生实战指南

该指南涵盖了全球云原生应用洞察与趋势、阿里云在云原生领域的最新产品与技术布局、阿里云 All in Serverless 的全新思考与投入、传音、新东方、小红书等企业的上云实战经验。

- 服务网格安全性和高可用性部署最佳实践

  文章介绍在多集群的基础设施中部署服务网格时的安全性和高可用性最佳实践，包括：控制平面应该如何部署在应用程序附近； 入口应该如何部署以促进安全性和敏捷性；如何使用 Envoy 促进跨集群负载均衡，以及网格内部如何使用证书。

- 大中型科技企业开源战略制定与落地

  目前，很多企业对于开源是什么，怎么用开源，如何参与开源，企业在开源方面如何做决策，如何进行开源治理，如何利用开源强化竞争力等方面的问题，仍存在着一些难点。 对此，文章介绍了什么是企业开源战略；企业为什么需要开源战略；企业开源战略包含的内容，以及企业开源战略制定和落地的实践经验。

- ServiceAccount Token 在不同 K8s 版本中的使用

  文章介绍了在不同 K8s 版本下，ServiceAccount Token 令牌的不同使用方式， 主要包括自动创建 Secret 和 Kubelet 通过 TokenRequest API 去申请 API 两种方式。

- 腾讯百万级别容器云平台实践揭秘

  文章介绍了腾讯容器云平台在线业务资源容器化部署遇到的问题、容器化对动态路由同步的挑战及各自的解决方案， 以及为解决 K8s 已有自愈机制存在容器销毁阶段卡住的问题，探索出的一种全新的容器销毁失败自愈机制。

- 在生产中大规模自动化 Istio CA 轮换

文章展示如何配置 Istio 以自动重新加载其 CA，以及如何配置 cert-manager 在 Istio 的中间 CA 到期前定期自动轮换，以提高在多个集群上管理 CA 的操作效率。

- Kubernetes 零信任安全的 mTLS 最佳实践

  文章介绍在 Kubernetes 中实现双向 TLS（mTLS）的三大最佳实践， 包括不要使用自签名证书、将 Istio 的信任根植于现有的公钥基础设施（PKI）中、使用中间证书。

- 微服务应用视角解读如何选择 K8s 的弹性策略

  对于集群资源弹性，K8s 社区给出了 Cluster Autoscaler（CA）和 Virtual Kubelet（VK）两种解决方案。 文章围绕微服务应用的形态与特点，剖析了 CA 与 VK 各自适用的场景，并总结了微服务架构下应用该如何选择集群资源弹性。

- GitOps 软件交付 pipeline 的组件介绍

  文章总结了开始使用 GitOps 所需的组件，包括容器平台、Git 仓库、容器镜像仓库、GitOps 代理和构建服务器。 以及在扩大规模时可以增加的一些组件，如容器配置自动化、策略引擎、服务网格、渐进式交付工具、事件监控等。

- 万字解读，如何从 0 到 1 构建 K8s 容器平台的 Nginx 负载均衡体系

  文章介绍了从 0 到 1 来构建大规模 Kubernetes 容器平台的 Nginx 负载均衡体系的实战经验， 包括体系的业务需求和运维需求、对 Nginx-Controller 组件的要求、架构设计、体验优化、核心考量点。

- 如何管理云原生应用程序的依赖关系

  文章介绍了一些云原生应用依赖关系管理的最佳实践，例如，使用 depcheck 来检查未使用的依赖关系，使用依赖关系检查脚本检测过期的依

赖关系， 以及利用自动化依赖关系管理开源工具 Mend Renovate 为所有类型的依赖关系更新自动创建拉取请求。

- K8s：利用 Mutating Admission Controller 简化应用的环境迁移

  Kubernetes 的 Mutating Admission Webhook 通常用于执行安全实践，确保资源遵循特定的策略或配置管理。 文章介绍了一个新的用例：简化应用的迁移，实现快速更新新环境的清单，同时保持旧环境正常运行。

- 阿里云容器攻防矩阵 & API 安全生命周期，如何构建金融安全云原生平台

  文章从攻击者视角介绍了容器平台（容器攻防矩阵）和 API 在金融科技领域的应用所带来的威胁面，分析威胁，并给出容器平台和 API 全生命周期的安全防护实践建议。

- 全球顶级开源公司是如何找到前 1000 名社区用户的?

  文章分享了 4 个顶级开源商业公司（HashiCorp、Confluent、Databricks 和 CockroachDB）的开源项目和开源社区的运作方式，如何找到前 1000 名社区用户及花了多长时间，以及增长衡量指标等。

- 容器云多集群环境下如何实践 DevOps

  文章介绍了如何通过 GitOps 来实现多集群 DevOps，并推荐了一种多集群 GitOps 工作流作为落地参考，最后介绍了在持续集成和持续交付中具体实践 GitOps 的主流方式。

- Kubernetes HPA 的三个误区与避坑指南

  Kubernetes 提供了水平弹性扩容能力（HPA），让应用可以随着实时指标进行扩/缩。 然而 HPA 的实际工作情况可能和我们预想的不一样，这里存在一些认知误区，例如，HPA 存在扩容死区、扩容与预期使用量不符、弹性行为总是滞后等。 对此，文章总结了一些注意事项，帮助在使用 HPA 时“有效避坑。

- 是时候思考 K8s 出向流量安全了（上）&（下）

  我们往往重视 Ingress 的能力，而忽视了 Egress 流量的安全管控。但无论从安全还是合规的角度，Egress 流量都应加强管控。 文章分析了进行 Egress 流量策略管控的原因、存在的挑战以及业界方案分析。方案包括六大类：基于 K8s Network Policy、CNI、Service Mesh、微分段技术以及 DNS interception。

- 如何保障云原生集群安全

  文章为容器集群安全交流论坛的内容汇总。 内容包括：云集群安全的分类以及预防及解决方案、容器云上集群划分原则、集群审计、节点安全保障、etcd 集群安全、设计网络策略保障集群资源的安全、网络隔离问题、按照安全等保级别针对 k8s 集群的标准化策略或配置等。

- Kubernetes 控制平面应该有多少个节点

  文章为确定 Kubernetes 控制平面的大小提供了参考。对于大多数用例，3 个节点足以。 5 个节点将提供更好的可用性，但在所需节点数量和硬件资源消耗方面成本更高。 此外，监控、备份 etcd 和控制平面节点配置是解决节点故障、预防意外的最佳实践方案。

- 利用 Istio 减少跨可用区流量开销

  跨可用区部署 Kubernetes 集群具备显著的可靠性优势，但需要一些额外的配置在本地保持流量。 Istio 的本地感知路由可以帮助减少延迟，并将云供应商对应用流量的跨区数据收费降到最低。

- 2 分钟测试 Kubernetes Pod 的安全

  从集群外部运行 Kyverno CLI，审计集群是否符合最新的 Kubernetes Pod 安全标准的要求，并为 pod 安全标准中定义的每个控制执行策略。执行审计不需要在集群中安装任何东西。

- Proxyless Mesh 在 Dubbo 中的实践

Dubbo Proxyless Mesh 直接实现 xDS 协议解析，实现 Dubbo 与 Control Plane 的直接通信，进而实现控制面对流量管控、服务治理、可观测性、安全等的统一管控，规避 Sidecar 模式带来的性能损耗与部署架构复杂性。

- 零信任策略下 K8s 安全监控最佳实践

  文章介绍如何在分布式容器化环境中借助 K8s 相关的安全数据源及采集技术，监控 K8s 集群，及时发现异常 API 访问事件、异常流量、异常配置、异常日志等行为，并且结合合理的告警策略建立更主动的安全防御体系。

- 基于 DevOps 流程的容器安全最佳实践

  文章分享了青藤云安全总结得出的适用于 DevOps 工作流程的 14 个容器安全实践，包括防御阶段的 6 个安全左移步骤、保护阶段的安全运行容器、检测阶段的异常行为告警以及响应阶段的事件响应与取证。

- 如何设计内部开发平台的“Golden Path”

  “Golden Path”是指构建和部署某个软件的预架构方法。 这种方法基于最佳实践的沉淀，能够加速典型应用的开发效率，是成功搭建内部开发平台（IDP）的先决条件。 文章介绍了“Golden Path”应该具备的特点以及一个简单相关成熟度模型，并展示了一个如何实施“Golden Path”的例子。

- 通过云原生的方式管理 GPU 资源

  文章介绍了如何使用云原生的方式管理 GPU 资源，包括统一 GPU 资源和框架，以及通过 qGPU 共享方式提高 GPU 使用率，降低了集群层面 GPU 资源的管理难度。

- 微服务全链路灰度新能力

  文章主要介绍阿里云 MSE 服务治理基于全链路灰度能力的应用场景与痛点延伸出来的两个新能力：运行时白屏化和配置灰度。 运行时白屏化用于

洞察全链路灰度的流量匹配以及运行的行为；配置灰度是指微服务应用中的配置项应该具备的灰度能力，以应对灰度应用对特殊配置值的诉求。

- Kubernetes 集群利用率提升实践

  文章分享了 Kubernetes 集群利用率提升的思路和实现方式，包括两级扩容机制、两级动态超卖、动态调度、动态驱逐。

- NGINX 现代应用参考架构（MARA）的 OpenTelemetry 集成进展报告

  MARA 是可用于在 Kubernetes 环境中构建和部署现代应用的生产就绪代码。 文章描述了 MARA 对多功能开源可观测性工具的需求，然后介绍了选择 OpenTelemetry 过程中的利弊权衡和设计决策， 以及采用了哪些技术将 OpenTelemetry 与使用 Python、Java 和 NGINX 构建的微服务应用相集成。

- Istio 外部控制面部署实践

  通过对外部 Istiod 部署模型进行调研与部署，作者对其架构进行了剖析并对其使用的部署 yaml 文件部分进行了解析，总结了在部署过程中可能会遇到一些问题。

- 监控 Cilium 的关键指标

  文章介绍了监控 Cilium 的关键指标：端点的健康状况、Kubernetes 网络状态、网络策略的有效性、Cilium 的 API 处理和速率限制的性能。

- 基于服务网格的云原生安全分层方法

  文章主要探讨如何利用服务网格优化 OSI 模型各层的现有安全工具和实践。 服务网格很适合作为最接近应用的一层，位于（并隐藏）底层安全层之上，为服务提供传输加密、应用身份和应用层访问控制。

- Observability：采样 Sampling 场景和落地案例（上篇）

文章先后介绍了采样的使用场景、采样在调用链系统架构的位置、主要方案（头部连贯采样、单元采样和尾部连贯采样）、主流开源和商业系统的采样实现、落地案例等。

- Observability 之采样 sampling 场景和落地案例（下篇）

  作者围绕一些落地案例，分享了一些关于可观测系统采样设计的思考。 先分析全量采样的必要性、适合全量采样的链路，然后分享了阿里鹰眼和字节跳动等业内完整采样方案以及 OpenTelemetry 生产环境采样案例。

- 如何扩展 K8s Descheduler 策略

  社区 Descheduler 的功能远不能满足公司内部生产环境的需要。 文章介绍了 LowNodeUtilization 和 HighNodeUtilization 策略的扩展。 在使用 Descheduler 迁移 Pod 时还需要对业务的稳定性做一些保障措施，完善组件功能后还需要对服务自身进行效果评估。

- Kubernetes 资源拓扑感知调度优化

  文章首先介绍了腾讯星辰算力的业务场景与精细化调度相关的知识，然后对 K8s 和 Volcano 社区进行调研，发现目前已有的方案都存在局限性。 最后通过痛点问题分析给出相应的解决方案。 优化后，原有测试任务的训练速度提升至原来的 3 倍，CPU 抢占的驱逐率大大降低至物理机水平。

- 如何正确配置 RBAC 防止容器逃逸

  文章介绍了在集群内利用危险的 RBAC 配置进行权限提升的思路，以此说明在容器逃逸后权限配置不当的影响。

- 如何优化 Prometheus 查询性能

  使用 Prometheus 时常会遇到的一个问题是：随着收集的指标数据量的增加，查询性能下降。 文章介绍了影响 Prometheus 查询性能的因素、Prometheus 提供的可识别缓慢/低效查询的选项以及优化查询性能的方法（如缩短查询时间范围、降低指标分辨率等）。

- vivo 容器集群监控系统架构与实践

  文章以 vivo 容器集群监控实践经验为基础，探讨了云原生监控体系架构如何构建，从监控高可用和数据转发层组件高可用两个方面介绍 vivo 容器集群监控架构的设计思路，最后总结了在实践过程中遇到的挑战和对策。

- 混合云的多活架构指南

  文章介绍了作业帮公司在混合云环境下的多活架构：多云间的网络是介于单云多可用区和中心–边缘之间的一种状态； 常态的应用间调用闭环在单云内部，仅需要单云内部独立注册发现； 某些场景下的跨云调用只需要集群间发现即可； 多云部署了全量服务，南北向流量就可以优先通过 DNS 进行调度。

- 2022 年，如何安全加固 Kubernetes?

  文章作者认为，美国国家安全局去年发布的《Kubernetes 加固指南》的摘要和后面主体内容存在很大差距。 为此，作者在总结该报告的基础上，提出了一些配置、应用任意许可、镜像扫描、安全测试等相关的额外见解。

- 混沌工程在微服务场景下的应用

  杭州每刻科技引入云原生混沌工程测试平台 Chaos Mesh ，以解决云财务场景下微服务健壮性测试过程中遇到的痛点。

- 百度云原生混部大规模落地实践之路

  混部技术，即将在线业务和离线任务混合部署到相同物理资源上，通过资源隔离、调度等控制手段，充分使用资源，同时保证服务的稳定性。 百度云原生混部系统主要分为三个部分：单机管理层、调度层、运营层。 单机层提供提供资源质量管理、内核级别的 QoS 隔离、资源视图上报和策略执行、eBPF 细粒度指标采集能力； 调度层负责资动态源视图的感知和提供最佳调度策略； 运营层提供资源画像、资源运营、水位设置、热点治理等功能。

- 确保 Kubernetes 安全合规的 6 个最佳实践

  文章介绍了，确保在容器化环境中持续合规而不影响生产力，应遵循的 6 个实践： 实现自动化、对 Kubernetes 本身进行保护、事先发现攻击、专注于零信任、利用 Kubernetes 内置安全措施验证云主机的安全性。

- 如何使用 Istio 授权策略执行 egress 流量

  文章以“服务网格内两个独立命名空间中的睡眠服务访问谷歌和雅虎的外部服务”为例，介绍了在执行 ingress 策略时，如何使用 Istio 的 Egress 网关以相似策略执行 egress 授权策略。

- 解决微服务架构下流量有损问题的实践和探索

  尽管可灰度、可观测和可滚回的安全生产三板斧可以最大限度的规避发布过程中由于应用自身代码问题对用户造成的影响，但对于高并发大流量情况下的短时间流量有损问题却仍然无法解决。 因此，文章围绕发布过程中如何解决流量有损问题、实现应用发布过程中的无损上下线效果相关内容展开方案介绍。

- 阿里巴巴的 Envoy Gateway 实践

  文章简单介绍了阿里巴巴探索下一代网关的历程：Envoy Gateway 1.0（孵化期）主要应用于东西向流量的 RPC 互通； 2.0（成长期）采用 Tengine + Envoy 两层网关，承担南北向网关流量； 3.0（成熟期）通过 Envoy 将流量网关 + 微服务网关合二为一，辅以硬件加速、内核调优等手段。

- 阿里云发布《云原生实战案例集》

  阿里云发布《云原生实战案例集》，通过技术瞭望、行业分析、实战案例解读为企业实践和落地云原生提供一定参考和借鉴。

- 阿里云发布企业云原生 IT 成本治理方案：五大能力加速企业 FinOps 进程

企业云原生 IT 成本治理存在业务单元与计费单元生命周期不匹配、动态资源交付与静态容量规划矛盾、企业 IT 成本治理模型与云原生架构适配等问题。 由此，阿里云容器服务推出了企业云原生 IT 成本治理方案，提供企业 IT 成本管理、成本可视化、成本优化等功能。

- 2022 年值得关注的 DevOps 趋势和最佳实践

  这篇文章介绍了 2022 年值得关注的 DevOps 趋势，其中包括微服务架构、DevSecOps 的应用、朝着完全无服务器的方向发展、弹性测试成为主流、GitOps 成为新常态以及发展混沌工程等。

- 分布式定时任务调度框架实践

  这篇文章介绍了任务调度框架使用的需求背景和痛点，对业界普遍使用的开源调度框架进行了探究实践，并分析了各自的优劣势。

- 拆分单独应用三问: What, When, How

  在微服务架构设计中，如何拆分单体是一件非常重要但令架构师十分头疼的事。这篇文章介绍了一些关于如何准备和执行单体应用程序拆分的思路和操作步骤。

- K8s 安全策略最佳实践

  文章开篇对 K8s 环境面临的安全风险及 K8s 提供的安全机制进行了介绍。此外，作者根据社区和自身团队的生产实践，总结了改善 K8s 安全状况的最佳实践。

- Prometheus 监控 Harbor 实战

  Harbor v2.2 及其更高版本支持对相关指标的采集和使用，这篇文章介绍了如何使用 Prometheus 轻松抓取 Harbor 实例的一些关键指标。

工具推荐

本页列出一些云原生社区常用的工具，以期提高 K8s 的用户体验。

- Kubernetes 解决“近邻干扰”（Noisy Neighbor）场景的探索

  当一个租户的性能由于另一个租户的活动而下降时，就会出现 noisy neighbor 问题。 虽然 Kubernetes 提供了 CPU 管理器、设备插件管理器以及拓扑管理器，协调资源分配和保证关键工作负载的最佳性能，但是没有根本上解决该问题。 对此，Intel 资源调配技术 RDT 支持通过为三种 Kubernetes QoS 级别配置相应的 RDT 类来限制非关键工作负载对共享资源的访问。

- FinOps 时代如何玩转应用资源配置

  Kubernetes 会按照应用程序申请的资源配额进行调度，因此如何合理的配置应用资源规格就成为提升集群利用率的关键。 文章分享如何基于 FinOps 开源项目 Crane 正确的配置应用资源，以及如何在企业内推进资源优化的实践。

- 帮助 Go 团队使用 OpenTelemetry 的新方法

  用 Go 实现 OTel 来向观测平台发送数据并不简单，Go instrumentation 需要大量的手动操作。 Helios 团队开发了一种新的 OTel Go instrumentation 方法，这种方法将 Go AST 和代理库的结合，易于实现和维护。 同时，对于最终用户而言，这种方法是非侵入性的，易于理解。

- 为什么在 Kubernetes 中调试应用的体验如此糟糕?

  对于开发者而言，想要的是：快速的内部开发循环，可以利用熟悉 IDE 工具来做本地调试；提前发现 bug，避免过早进入 Outer Loop；在内部环境，团队之间的协作可以互不干扰。 为解决上述痛点，文章介绍了三款工

具：Kubernetes 本地开发工具 Telepresence、云原生协同开发测试解决方案 KT-Connect 以及基于 IDE 的云原生应用开发工具 Nocalhost。

- 在 K8s 上构建端到端的无侵入开源可观测解决方案

  Odigos 是一个开源的可观测性控制平面，允许企业在几分钟内创建可观测性管道，集成众多第三方项目、开放标准，降低多个可观测性软件平台和开源软件解决方案的复杂性。 此外，还允许应用程序在几分钟内提供追踪、指标和日志，重要的是无需修改任何代码，完全无任何侵入性。 Odigos 能够自动检测集群中每个应用的编程语言，并进行自动检测。

- 为什么我们提倡以工作负载为中心而非基础设施为中心的开发模式?

  云原生开发人员经常被环境间的配置不一致所困扰。 Score 是一个开源项目，提供了一个以开发者为中心、与平台无关的工作负载规范， 能够声明式地描述其运行时的要求，消除了本地和远程环境之间的配置不一致性，提高开发者的生产力。

- 用 Bindle 轻松存储和分发云原生应用

  Bindle 是一个开源的包管理器，通常用于存储和分发 WebAssembly 应用和二进制文件。 Bindle 可以将多个不同的微服务打包在一个大应用程序中，并将其部署在任何需要的地方，也可以存储包含数十个不同二进制文件的应用程序。

- Kubernetes 跨集群流量调度实战

  文章演示了 Kubernetes 南北向流量管理方案 FSM 是如何实现服务的跨集群流量调度和负载均衡的，以及三种不同的全局流量策略的实现：仅本集群调度、故障转移、全局负载均衡。

- ChaosBlade：大规模 Kubernetes 集群故障注入的利器

  ChaosBlade 是一款遵循混沌工程原理和混沌实验模型的实验注入工具，帮助企业提升分布式系统的容错能力，并且为企业在上云或往云原生迁移过

程中提供业务连续性保障。 文章主要介绍 ChaosBlade 在 Kubernetes 中故障注入的底层实现原理、版本优化过程以及大规模应用演练测试。

- OpenFeature 和 Feature Flag 标准化如何造就高质量的持续交付

  Feature Flag 即功能开关或功能发布控制，是一种通过配置开关功能特性的技术，无需重新部署代码。 而 OpenFeature，是一个关于 Feature Flag 的开放标准，旨在使用云原生技术构建一个强大的 Feature Flag 生态系统，允许团队灵活地选择符合当前需求的 feature flag 方法，并在需求发生变化时切换到其他方法。

- Traffic Director: 在 GKE 上 使用 Envoy gateway 代理实现 TLS 路由

  Traffic Director 是谷歌托管的服务网格控制平面，用于解决微服务流量的治理问题。 文章分享了一个架构样本，在 GKE 集群上，使用 Traffic Director 配置 Envoy gateway 代理，使用 TLS 路由规则，将集群外的客户端流量路由到部署在集群上的工作负载。 此外，演示了如何利用 Envoy 代理作为入口网关，使南北流量进入服务网格，以及使用服务路由 API 来路由这些流量，最后还分享了一些故障排除技巧。

- Quarkus 的 Java 框架如何用于 serverless function 开发?

  Quarkus 解决了传统框架内存消耗大和容器环境的扩展问题。 通过 Quarkus，开发人员可以使用熟悉的技术构建云原生微服务和 serverless function。 文章介绍如何开始使用 Quarkus 进行 serverless function 开发、如何优化 function 并实现持续测试，以及制作跨 serverless 平台的可移植 function 等。

- GitLab + Jenkins + Harbor 工具链快速落地指南

  文章介绍了如何利用 DevOps 工具链管理器 DevStream 快速部署 DevOps 工具链（GitLab + Jenkins + Harbor）。

- Helm 部署高可用 Harbor 镜像仓库

文章介绍如何使用 Helm 包管理工具将 harbor 部署在 kubernetes 集群中并且实现高可用。

- 微软 Azure 在容器供应链安全的开源实践

  文章主要介绍 Azure 在容器供应链安全领域的开源实践， 例如：Microsoft/SBOM Tool 生成 SBOM 文件，Notary v2 对容器镜像等软件制品进行签名和验证，ORAS 扩展 OCI 赋能供应链安全，Ratify 帮助 Kubernetes 验证应用的部署安全。

- vcluster — 基于虚拟集群的多租户方案

  虚拟集群即 vcluster 是在常规的 Kubernetes 集群之上运行的一个功能齐全、轻量级、隔离性良好的 Kubernetes 集群。 其核心思想是提供运行在“真实”集群之上隔离的 Kubernetes 控制平面（例如 API Server）。 与完全独立的“真实“集群相比，虚拟集群没有自己的工作节点或者网络，工作负载实际上还是在底层宿主集群上调度。

- 5 个实用工具，提升 Kubernetes 生产力

  文章列出了五种与 Kubernetes 一起工作的强大工具， 分别是终端 UI K9s、清理 Kubernetes 集群的工具 Popeye、Kubernetes 集群部署检查工具 Kube-bench、上下文和命名空间快速切换工具 Kubectx、Kubens 和 fzf、日志聚合器 Stern、从 shell 快速检查文件 Bat。

- 利用 Open Cluster Management（OCM） 的 Placement 扩展多集群调度能力

  在 K8s 多集群管理项目 OCM 中，多集群调度能力由 Placement 控制器提供。 Placement 提供了一些默认的优先级排序器，用于分类和选择最合适的集群。在某些情况下，排序器需要更多的数据来计算集群的得分。因而，我们需要一种可扩展的方式来支持基于自定义得分的调度。

- 开源云原生安全治理平台 HummerRisk

HummerRisk 是一个开源的云安全治理平台，以非侵入的方式对云原生环境进行全面安全检测，核心解决三个方面的问题，底层的混合云安全合规，中层的 K8s 容器云安全和上层的软件安全。

- 使用 Config Sync 以 GitOps 的方式部署 OCI 工件和 Helm charts

  Config Sync 是一个开源工具，为 Kubernetes 集群提供 GitOps 持续交付。 它能够通过 GitOps 的方式来存储和部署 Kubernetes 清单，把 Kubernetes 清单打包成一个容器镜像，并且使用与容器镜像相同的认证和授权机制。

- 突破 Kubernetes 对自定义资源数量的限制

  Crossplane 社区发现了 Kubernetes 能够处理的 CRD 上限，并且帮助解决了这些问题。 其中，限制原因包括：限制性的客户端速率限制器、缓慢的客户端缓存、低效的 OpenAPI 模式计算、冗余高昂的成本、etcd 客户端。

- K8s DevOps 平台 TAP 基于 Knative 的云原生运行时

  TAP 的云原生应用运行时抽象层（CNR）的核心是 Knative。 TAP 提供了一个 Runtime 运行时层，既支持用户使用 K8S Deployment 和 Service,也可以使用 Knative Serving, Scale From/To Zero,Eventing 和 Streaming 等。

- 使用 Tekton 和 Kyverno 的基于策略的方法来保障 CI/CD 管道的安全

  Tekton 提供了一个强大的 CI/CD 框架，并且通过 Tekton Chains 等扩展确保构建工件的安全。 Kyverno 可用于管理策略，实现基于命名空间的隔离，并为 Tekton 管道生成安全资源。 此外，还可以签署 OCI 工件，如 Tekton 捆绑物。Tekton 和 Kyverno 的强大组合使软件构建和交付系统的安全和自动化达到了新的水平。

- K8S 运维开发调试神器 Telepresence 实践指南

Telepresence 可用于在本地运行单个服务，同时将该服务连接到远程 Kubernetes 集群。 Telepresence 提供了 3 个非常重要的功能：cluster 域名解析、cluster 流量代理和 cluster 流量拦截。

- 使用 Containerlab + kind 快速部署 Cilium BGP 环境

  kind 提供快速部署 K8s 能力，Containerlab 提供快速部署网络资源能力，二者结合可达到秒速部署跨网络 K8s 集群，可用于快速部署和销毁 Cilium BGP 环境。

- 可观测下一站：基于 Trace 的测试

  可观测性工具收集的分布式追踪数据是集成测试的理想选择。 基于 trace 的测试能够准确地指定要测试的事务，观测系统行为的结果，验证组件之间的依赖关系，主动测试潜在的问题。 Tracetest 是一个基于 trace 的测试工具，它利用 Opentelemetry 分布式追踪所捕获的数据，来生成强大的集成测试。

- RunD：高密高并发的轻量级 Serverless 安全容器运行时

  RunD 通过全栈的 host-to-guest 解决方案，解决了跨容器的重复数据、每个虚拟机的高内存占用和 host 端 cgroup 的高开销问题，提供高密度部署和高并发能力的支持。 使用 RunD 运行时，可以做到在一秒内启动超过 200 个安全容器，并且可以在一个 384GB 内存的节点上部署超过 2500 个安全容器。

- 在 Kubernetes 中实施零信任的难点及相关开源解决方案

  Kubernetes 的复杂性使得标准化应用零信任原则成为挑战。默认情况下，kubectl 不启用 RBAC，执行的命令不会被用户账号记录。通过防火墙访问资源很困难，监督多个集群也变得繁杂易错。 Paralus 是为多集群环境设计的资源访问管理、威胁识别响应方案，支持自定义角色、身份提供商（IdP）等，允许管理员创建具有不同权限的自定义规则。

- Kubernetes Gateway API 及其用例介绍

  文章介绍了 Gateway API 与 Ingress 的区别及其用例。Gateway API 是 Ingress 的演变，它通过扩展 API 定义来提供一些高级的功能，如 HTTP 和 TCP 路由、流量拆分、跨命名空间路由以及集成渐进式交付工具等。

- 如何进行 K8s 集群基准测试——Kube-burner 对 KubeVirt CRD 的扩展支持介绍

  Kube-burner 一个通过创建或删除大量对象进行 Kubernetes 集群压测的工具。 kube-burner 在默认情况下不支持第三方插件，如 KubeVirt CRD。 因此，Openshift 对 kube-burner 进行扩展以支持 KubeVirt CRD，通过创建/删除虚拟机实现集群基准测试。

- 在 Argo CD 中使用 Sops 增强 GitOps 安全性

  Sops 是一款加密文件的编辑器，支持 YAML、JSON、ENV 等格式，同时可以用 AWS KMS、GCP KMS、age 和 PGP 等进行加密。 Argo CD 可以与 Sops 集成，对私密信息进行加解密。

- 如何使用 KubeClarity 检测和管理软件物料清单以及容器镜像和文件系统漏洞

  KubeClarity 是一款专门用于检测和管理软件物料清单以及容器镜像和文件系统漏洞的工具，可以扫描运行时 K8s 集群和 CI/CD 管道，以增强软件供应链的安全性。

- 如何针对 Helm 做金丝雀发布?

  Helm 本身在设计时并不考虑灰度发布、也不针对工作负载做管理。 通过 KubeVela 的插件机制联合 fluxcd addon 和 kruise-rollout addon，无需对 Helm Chart 做任何改动，就可轻松完成 Helm 应用的金丝雀发布。

- 使用 Nocalhost 开发 Rainbond 上的微服务应用

Nocalhost 是基于 IDE 的云原生应用开发工具，Rainbond 是一款云原生多云应用管理平台。 Nocalhost 可以直接在 Kubernetes 中开发应用，Rainbond 可以快速部署微服务项目，无需编写 Yaml，Nocalhost 结合 Rainbond 加速微服务开发效率。

- 两个 OCI 镜像构建工具介绍 builders—— melange 和 apko

  apk 直接使用 Alpine 的包管理工具 APK 来构建镜像，不需要使用 Dockerfile，只需要提供声明式 YAML 清单。 melange 使用声明式 YAML 管道来构建 APK。

- 多集群场景下基于 Flux v2 的应用持续交付实践

  Flux v2 提供了一组可支持实现 GitOps 的工具，面向云原生应用持续交付提供了通用的解决方案。 文章主要从多集群场景下部署差异化配置的云原生应用出发，介绍了基于 Flux v2 的应用持续交付实践。

- 使用 Chain-bench 审计你的软件供应链是否符合 CIS 的要求

  互联网安全中心（CIS）最近发布了《软件供应链安全指南》，提供保护软件交付管道的最佳实践。 作为指南的发起者和主要贡献者，Aqua 团队开发了首个软件供应链审计开源工具—— Chain-bench，用以审计软件供应链是否符合 CIS 的基准，并实现审计过程自动化。

- Chainguard 版本的 下一代 Distroless 镜像

  Chainguard 正在构建下一代 distroless 镜像，使用一个工具链即可轻松地从现有的软件包中组成 distroless 镜像，并创建自定义软件包。 与谷歌的 Bazel 和基于 Debian 的系统不同的是，Chainguard 工具链以 apk（Alpine 软件包管理器）、apko（用来构建基于 Alpine 的 distroless 镜像）和 melange（使用声明式 pipeline 构建 apk 包）为核心，减少 distroless 镜像的复杂性。

- 通过 MetalLB 在 OpenYurt 边缘侧实现对 LoadBalancer 类型 service 的支持

  在云边协同场景下，由于边缘侧并不具备云端 SLB 服务的能力，边缘 Ingress 或边缘应用无法暴露 LoadBalancer 类型的服务供集群外访问。文章对此场景，探讨如何通过 MetalLB 在 OpenYurt 边缘侧实现对 LoadBalancer 类型 service 的支持。

- 在 Kubernetes 中实施热重载应用

  如果想在开发 Kubernetes 应用时直接在集群内测试应用，需要执行相当多的步骤。 文章介绍了在 Kubernetes 内部进行热重载的意义，以及如何使用 DevSpace 高效完成热重载——DevSpace 负责自动重建和重新部署应用，用户只需要保存应用文件。

- 使用 Tracetest 检测和修复性能问题

  Tracetest 是基于 OpenTelemetry Traces 的端到端测试工具。Tracetest 可以在用户或开发人员遇到 bug 之前发现代码中的异常情况。

- OPAL 介绍：实时动态授权

  OPAL 是开放策略代理（OPA）的一个管理层，它能够保持授权层的实时更新。 OPAL 对策略和策略数据的变化进行检测，并将实时更新推送给 agent。当应用状态发生变化时，OPAL 将确保服务始终与所需的授权数据和策略保持同步。

- 夜莺（Nightingale）—— Prometheus 的企业级版本

  夜莺是一款开源云原生监控分析系统，采用 All-In-One 的设计，集数据采集、可视化、监控告警、数据分析于一体，集成云原生生态，提供开箱即用的企业级监控分析和告警能力。 文章主要介绍了夜莺如何关联可观测性三指标、运维量化、告警噪音处理、产品定位以及 AIOps 应用等。

- 移植 eBPF 应用到 BumbleBee —— 开发 BPF CO-RE 程序最简单的方法

  BPF CO-RE（Compile Once – Run Everywhere）旨在解决 eBPF 程序可移植性的问题。 文章举例详细说明了如何将一个 BPF CO-RE libbpf 脚本移植到 Bumblebee 中，以解决 user space、distribution 和集成方面的挑战。

- 一键开启 Kubernetes 可观察性 —— 如何自动生成和存储 OpenTelemetry 追踪

  Tobs 是 Kubernetes 的可观测性技术栈，它可以在几分钟内在 Kubernetes 集群中部署一个完整的可观测性技术栈。 该栈包括 OpenTelemetry Operator、OpenTelemetry Collector、Promscale 和 Grafana。它还部署了其他几个工具，如 Prometheus，用于收集 Kubernetes 集群的指标，并将其发送到 Promscale。 Tobs 支持通过 OpenTelemetry Operator 用 OpenTelemetry trace 自动检测的 Python、Java 和 Node.js 服务。

- Categraf：夜莺监控默认数据采集 Agent 介绍

  Categraf 是夜莺监控的默认数据采集 Agent，主打开箱即用和 all-in-one，同时支持对 metrics、log、trace 的收集。 文章首先对比 categraf 和 telegraf、exporters、grafana-agent、datadog-agent 的异同，然后介绍了 catagraf 的安全、测试和配置。

- Crane-scheduler：基于真实负载进行调度

  原生 Kubernetes 调度器只能基于资源的 resource request 进行调度，然而 Pod 的真实资源使用率，往往与其所申请资源的 request / limit 差异很大。 Crane-scheduler 是成本优化开源项目 Crane 的调度插件，实现了基于真实负载的调度功能，作用于调度过程中的 Filter 与 Score 阶

段，并提供了一种灵活的调度策略配置方式，从而有效缓解 kubernetes 集群中各种资源的负载不均问题。

- 如何利用 Multus CNI 在二级网络上使用 Kubernetes 服务

  文章介绍了 OpenShift 4.10 开发者预览版如何利用 Multus CNI 在二级网络上使用 Kubernetes 服务，实现 Kubernetes 二级网络在功能上的平等。

- 使用 vmagent 代替 Prometheus 采集监控指标

  vmagent 是开源时序数据库 VictoriaMetrics（VM） 中的一个组件，它可以帮助我们从各种来源收集指标并将它们存储在 VM 或者任何其他支持 remote write 协议的 Prometheus 兼容的存储系统中。 vmagent 可以实现 Prometheus 的诸多功能，但指标抓取更灵活，支持拉取和推送指标，使用更少的内存、CPU、磁盘 I/O 以及网络带宽等。

- 深度解读：分布式系统韧性架构压舱石 OpenChaos

  OpenChaos 是 RocketMQ 团队针对分布式系统韧性能力而打造的新兴混沌工程工具。 OpenChaos 针对一些分布式系统特有属性，如 Pub/Sub 系统的投递语义与推送效率，调度编排系统的精准调度、弹性伸缩与冷启效率，streaming 系统的流批实时性等，设置专用的检测模型。 同时，内置可扩展的模型支持，以便验证在大规模数据请求以及各种故障冲击下的韧性表现，并为架构演进提供优化建议。

- Kubernetes 将大量采用 Sigstore 来保护开源生态系统

  去年推出的 Sigstore 是一项面向软件开发人员的免费签名服务，它通过由透明日志技术支持的加密软件签名，提高了软件供应链的安全性。Kubernetes 1.24 以及所有未来的版本采用 Sigstore 来签署工件和验证签名。

- Kubernetes 节点弹性伸缩开源组件 Amazon Karpenter 实践：部署 GPU 推理应用

  Karpenter 是亚马逊发布的针对 Kubernetes 集群节点弹性伸缩的开源组件。 该组件可以针对 Unscheduleable Pods 的需求，自动创建合适的新节点并加入集群中。 Karpenter 彻底抛弃了节点组的概念，利用 EC2 Fleet API 直接对节点进行管理。 文章以 GPU 推理的场景为例，详细阐述 Karpenter 的工作原理、配置过程以及测试效果。

- Kruise Rollout: 让所有应用负载都能使用渐进式交付

  Kruise Rollout 是 OpenKruise（阿里云开源的云原生应用自动化管理套件）针对渐进式交付抽象的定义模型，旨在解决应用交付领域的流量调度以及分批部署问题。 其能够配合应用流量和实际部署实例的金丝雀发布、蓝绿发布等，以及发布过程能够基于 Prometheus Metrics 自动化分批与暂停，兼容多种工作负载等。

- Prometheus 长期远程存储方案 VictoriaMetrics 入门实践

  VictoriaMetrics（简称 VM ）是一个支持高可用、经济高效且可扩展的开源监控解决方案和时间序列数据库，可用于 Prometheus 监控数据做长期远程存储。 除此之外，VM 的主要特点包括：对外支持 Prometheus 相关的 API、指标数据摄取和查询具备高性能和良好的可扩展性、高性能的数据压缩方式等。

- 声明式管理 Helm 版本的工具

  这篇文章介绍了 Kubernetes 生态中一些可用于声明式管理 Helm 版本的工具（如 CNCF 项目 Flux 和 Argo），并对这些工具进行了比较。

- 使用 Kubecost 和 Kyverno 进行云原生工作负载成本治理

这篇文章介绍了如何使用策略引擎 Kyverno 和 成本管理工具 Kubecost 对云原生工作负载进行成本治理，当 Kubecost 统计的某个 Kubernetes 工作负载的成本高于分配的值时，Kyverno 会创建违规/失败。

- Dubbo-go-Mesh 开启新一代 Go 微服务形态

  Dubbo-go-Mesh 是分布式 RPC 框架 Dubbo-go 在无代理服务网格场景下的跨生态服务框架实现方案。目前已支持兼容 Istio 的服务治理能力。支持接口级服务发现能力，兼容 Istio 生态的流量控制和管理能力，并且提供了脚手架和应用模板以提高 Go 应用开发效率。

- 如何通过二级调度器在 OpenShift 中引入自定义的调度器

  Kubernetes Scheduler 无法满足一些应用对调度器的特殊需求：如共同调度、拓扑感知调度、负载感知调度等。 由此，OpenShift 允许用户通过应用市场 Operator Hub 的二级调度器引入各自定制的调度器，并使用该调度器运行他们选择的工作负载。 而控制面组件仍使用 OpenShift 提供的默认调度器。

- 利用 eBPF 技术带来的可观测性的上帝视角：Kindling 开源项目介绍

  分布式追踪技术在实际落地过程中常会面临探针自动化覆盖依赖人工、难以覆盖多语言服务、APM trace 缺少内核可观测数据等痛点， 而 Kindling 基于 eBPF 技术构建的上帝视角带来了解决方案——关联内核可观测数据的 trace。

- 虚拟 Kubernetes 集群：多租户新模式

  一直以来，基于命名空间和集群的两种多租户模式都存在诸多弊端。 这篇文章引出一个较新的概念——虚拟集群。它结合了上述两种多租户方法的优点：多租户只使用一个命名空间，租户在虚拟集群内拥有完全控制权。

什么是 Falco

Falco 是一个云原生运行时安全工具，旨在检测应用程序中的异常活动，可用于监控 Kubernetes 应用程序和内部组件的运行时安全性。仅需为 Falco 撰写一套规则，即可持续监测并监控容器、应用、主机及网络的异常活动。

Falco 能检测到什么？

Falco 可对任何涉及 Linux 系统调用的行为进行检测和报警。Falco 的警报可以通过使用特定的系统调用、参数以及调用进程的属性来触发。例如，Falco 可以轻松检测到包括但不限于以下事件：

- Kubernetes 中的容器或 pod 内正在运行一个 shell 。
- 容器以特权模式运行，或从主机挂载敏感路径，如 /proc。
- 一个服务器进程正在生成一个意外类型的子进程。
- 意外读取一个敏感文件，如 /etc/shadow。
- 一个非设备文件被写到 /dev。
- 一个标准的系统二进制文件，如 ls，正在进行一个外向的网络连接。
- 在 Kubernetes 集群中启动一个有特权的 Pod。

关于 Falco 附带的更多默认规则，请参考 Rules 文档。

什么是 Falco 规则？

Falco 规则定义 Falco 应监视的行为及事件；可以在 Falco 规则文件或通用配置文件撰写规则。有关编写、管理和部署规则的更多信息，请参阅 Falco Rules。

什么是 Falco 警报？

警报是可配置的下游操作，可以像记录日志一样简单，也可以像 STDOUT 向客户端传递 gRPC 调用一样复杂。有关配置、理解和开发警报的更多信息，请参阅 Falco 警报。Falco 可以将警报发送至：

- 标准输出
- 一份文件
- 系统日志
- 生成的程序
- 一个 HTTP[s] 端点
- 通过 gRPC API 的客户端

Falco 由哪些部分组成？

Falco 由以下几个主要组件组成：

- 用户空间程序：CLI 工具，可用于与 Falco 交互。用户空间程序处理信号，解析来自 Falco 驱动的信息，并发送警报。
- 配置：定义 Falco 的运行方式、要断言的规则以及如何执行警报。有关详细信息，请参阅配置。
- Driver：一款遵循 Falco 驱动规范并发送系统调用信息流的软件。如果不安装驱动程序，将无法运行 Falco。目前，Falco 支持以下驱动程序：
  - 基于 C++ 库构建 libscap 的内核模块 libsinsp（默认）
  - 由相同模块构建的 BPF 探针
  - 用户空间检测

    有关详细信息，请参阅 Falco 驱动程序。
- 插件：可用于扩展 falco libraries/falco 可执行文件的功能，扩展方式是通过添加新的事件源和从事件中提取信息的新字段。 有关详细信息，请参阅插件。

云原生安全

DCE 5.0 针对容器、Pod、镜像、运行时、微服务提供了全面自动化的安全实现。 下表列出了一些已实现或正在实现中的安全特性。

安全特性	细目	描述
镜像安全	可信镜像分发	为实现镜像的安全传输，需具备密钥对和签名信息，保证传输安全。在传输镜像时具备选择密钥进行镜像签名能力。
运行时安全	事件关联分析	支持对运行时检测出的安全事件做关联与风险分析，增加攻击溯源能力，收敛告警，降低无效告警，提高事件响应效率。
–	容器诱饵仓库	具备容器诱饵仓库，常见诱饵包括但不限于：未授权访问漏洞、代码执行漏洞、本地文件读取漏洞、远程命令执行 RCE 漏洞等容器诱饵。
–	容器诱饵部署	支持自定义新增诱饵容器，可以自定义服务名称、服务位置等。
–	容器诱饵告警	支持对容器诱饵中可疑行为进行告警。
–	偏移检测	在扫描镜像同时，学习镜像中全部二进制文件信息，并形成“白名单”，容器上线后仅允许“白名单”中

安全特性	细目	描述
		二进制文件运行，确保容器内不能运行不授信（如非法下载）的可执行文件。
微隔离	隔离策略智能推荐	支持记录资源历史访问流量，并在对资源进行隔离策略配置时能够智能依据历史访问流量进行策略推荐。
–	租户隔离	支持对 Kubernetes 集群内租户进行隔离控制，具备对不同的租户设置不同的网络安全组的能力，支持租户级别的安全策略设置功能，通过不同安全组和设置的安全策略实现租户间网络的访问与隔离。
微服务安全	服务及 API 安全扫描	对集群内的服务及 API 支持自动扫描、手动扫描及周期性扫描的安全扫描方式，支持全部的传统 web 扫描项目包括 XSS 漏洞、SQL 注入、命令/代码注入、目录枚举、路径穿越、XML 实体注入、poc、文件上传、弱口令、jsonp、ssrf、任意跳转、CRLF 注入等风险，以及容器环境特有的项，针对发现的漏洞支持漏洞类型展示、url 展示、参数展示、危险级别展示、测试方法展示等。

Falco-exporter

Falco-exporter 是一个 Falco 输出事件的 Prometheus Metrics 导出器。

Falco-exporter 会部署为 Kubernetes 集群上的守护进程集。如果集群中已安装并运行 Prometheus，Prometheus 将自动发现 Falco-exporter 提供的指标。

安装 Falco-exporter

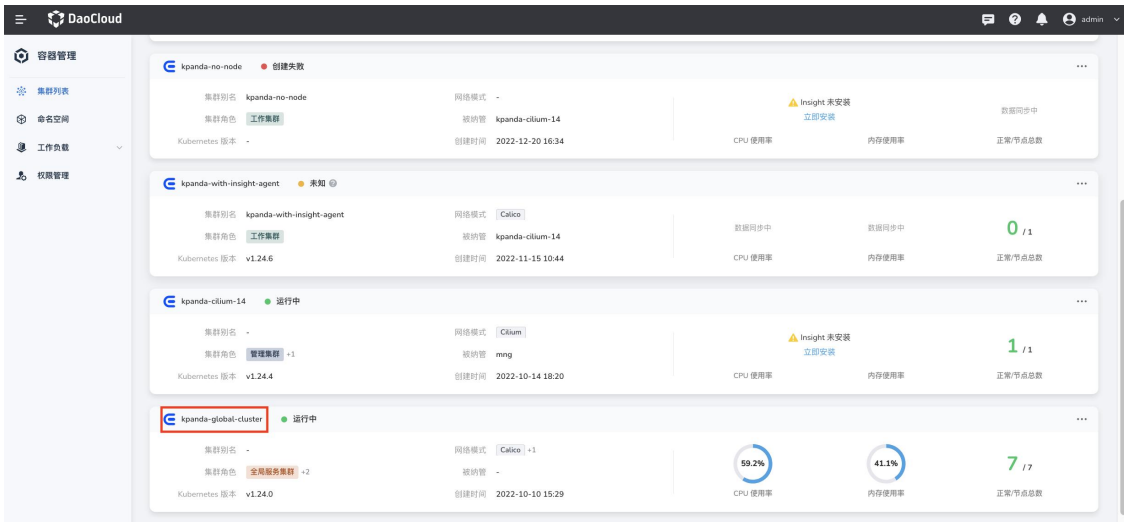
本页介绍如何安装 Falco-exporter 组件。

!!! note

在安装使用 Falco-exporter 之前，需要[安装](./falco-install.md)并运行 Falco，并启用 gRPC 输出（默认通过 Unix 套接字启用）。
关于启用 gRPC 输出的更多信息，可参阅[在 Falco Helm Chart 中启用 gRPC 输出](https://github.com/falcosecurity/charts/tree/master/falco#enabling-grpc)。

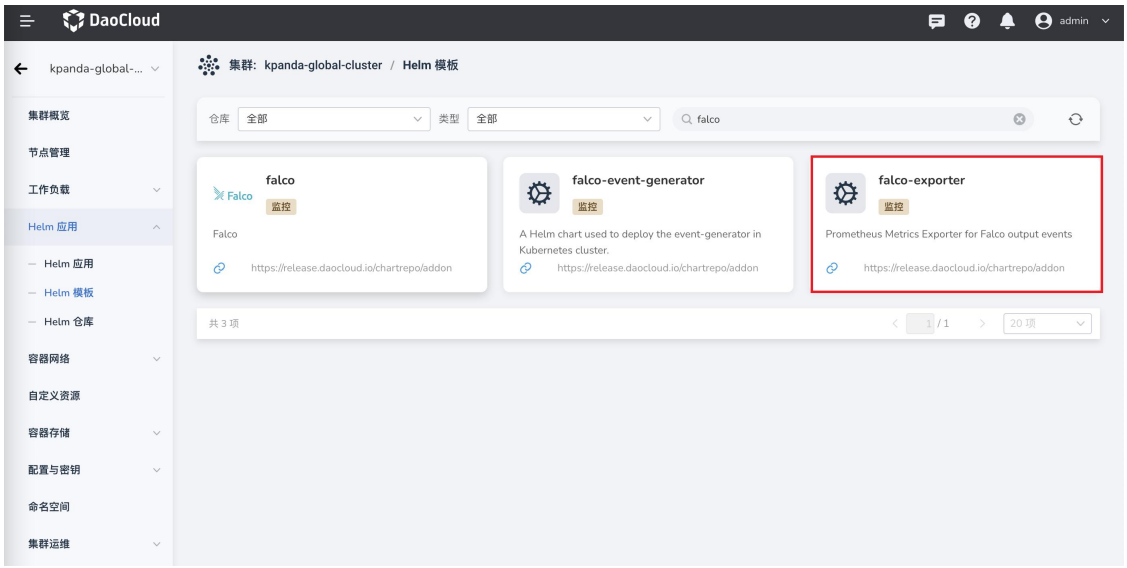
请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Falco-exporter。

1. 从左侧导航栏点击容器管理—>集群列表，然后找到准备安装 Falco-exporter 的集群名称。



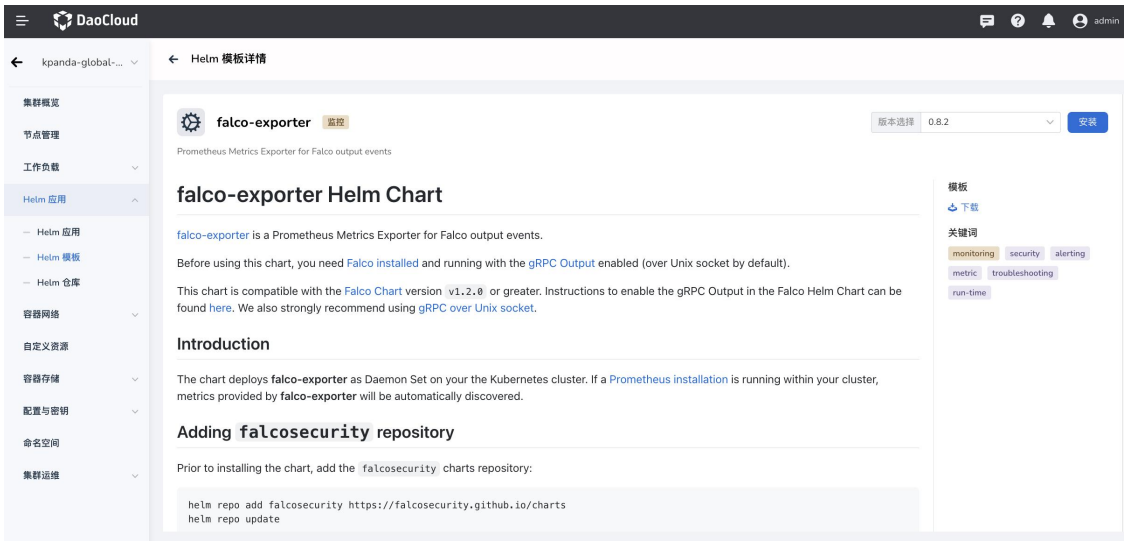
falco_cluster

2. 在左侧导航栏中选择 Helm 应用 –> Helm 模板，找到并点击 falco–exporter。



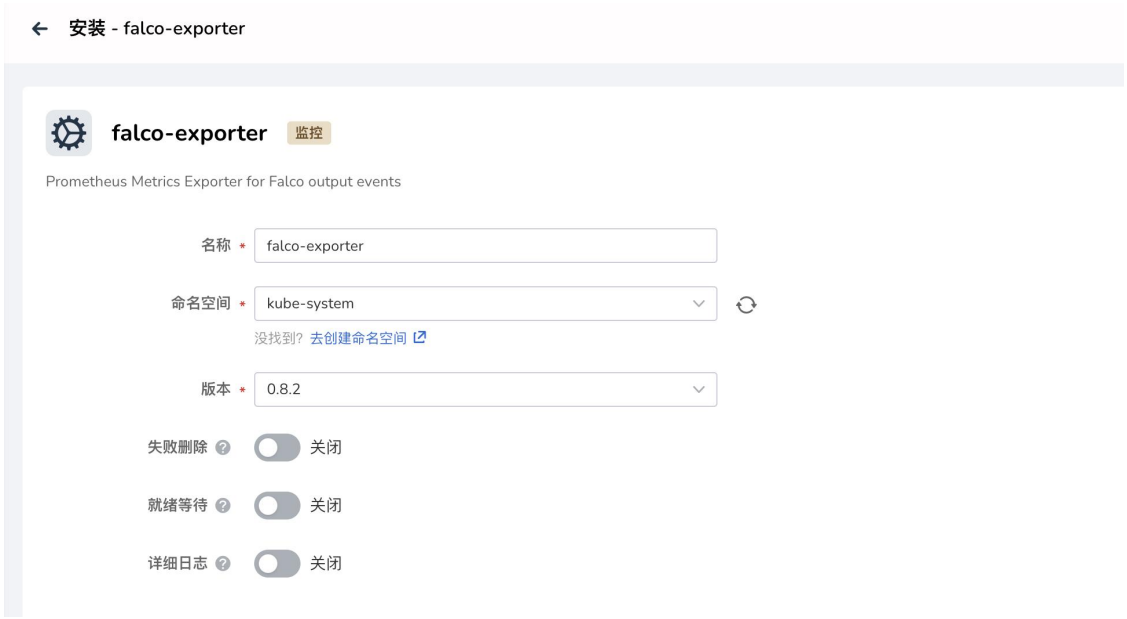
falco–exporter_helm–1

3. 在版本选择中选择希望安装的版本，点击安装。



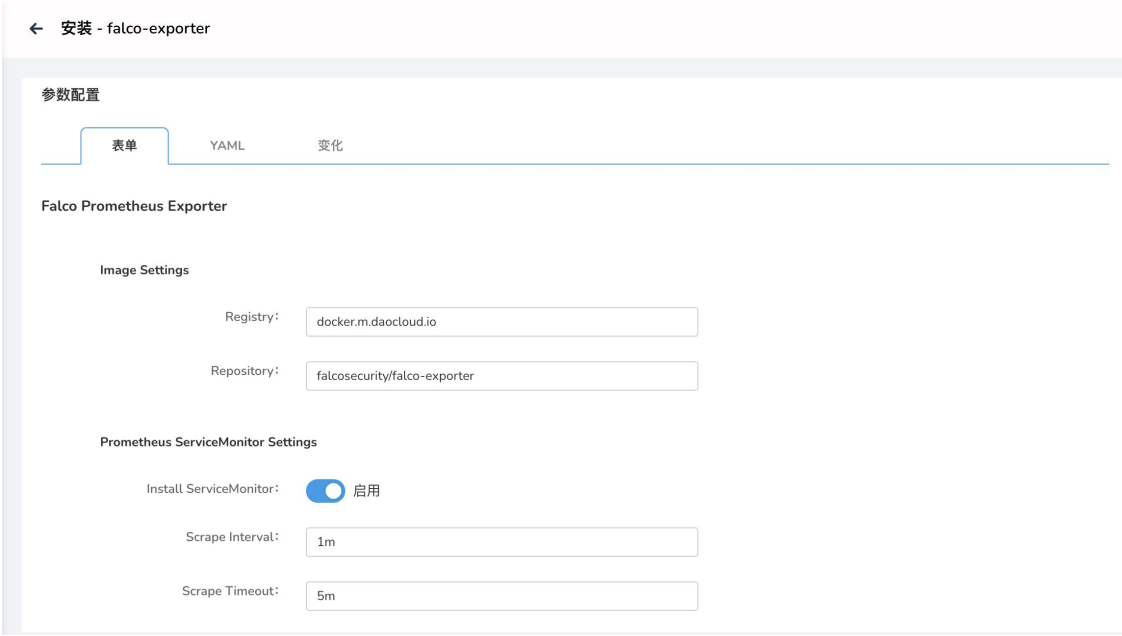
falco-exporter_helm-2

4. 在安装界面，填写所需的安装参数。



falco-exporter_helm-3

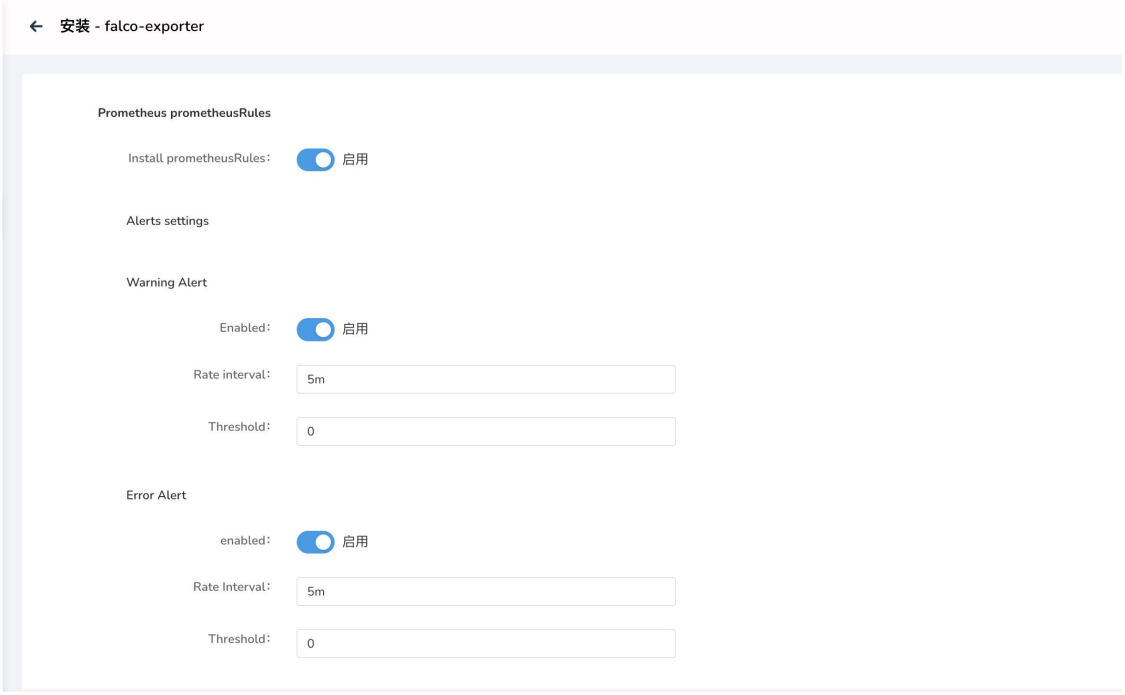
在如上界面中，填写应用名称、命名空间、版本等。



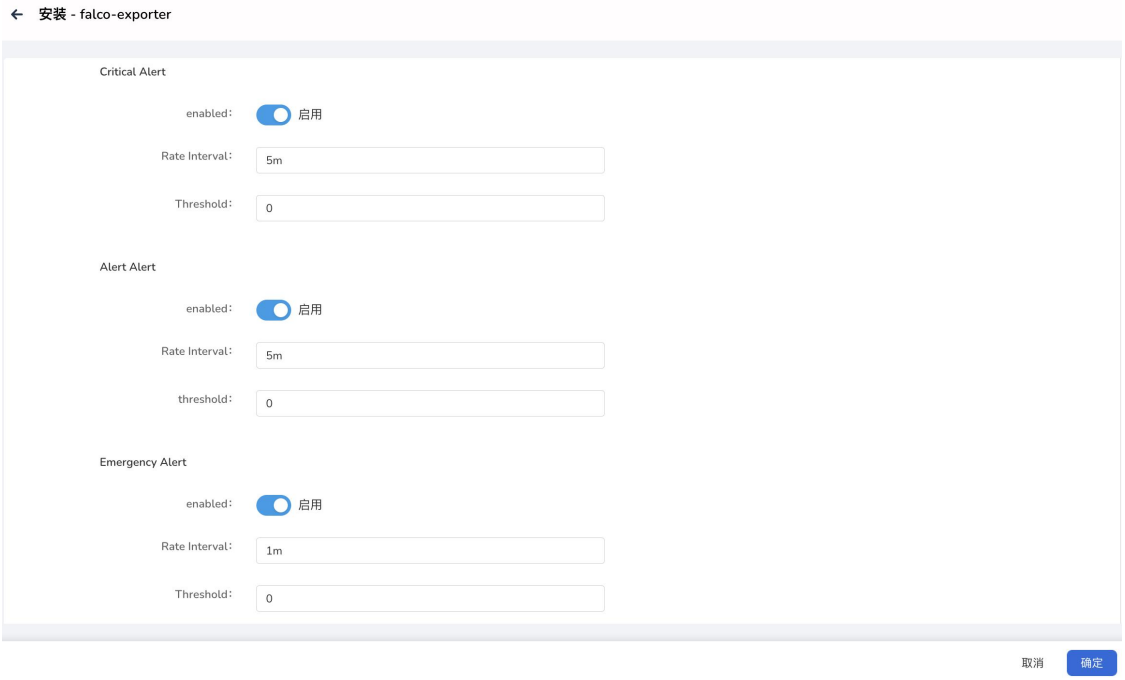
falco-exporter_helm-4

在如上界面中，填写以下参数:

- Falco Prometheus Exporter -> Image Settings -> Registry：设置 falco-exporter 镜像的仓库地址，已经默认填写可用的在线仓库。如果是私有化环境，可修改为私有仓库地址。
- Falco Prometheus Exporter -> Prometheus ServiceMonitor Settings -> Repository：设置 falco-exporter 镜像名。
- Falco Prometheus Exporter -> Prometheus ServiceMonitor Settings -> Install ServiceMonitor：安装 Prometheus Operator 服务监视器，默认开启。
- Falco Prometheus Exporter -> Prometheus ServiceMonitor Settings -> Scrape Interval：用户自定义的间隔；如果未指定，则使用 Prometheus 默认间隔。
- Falco Prometheus Exporter -> Prometheus ServiceMonitor Settings -> Scrape Timeout：用户自定义的抓取超时时间；如果未指定，则使用 Prometheus 默认的抓取超时时间。



falco-exporter_helm-4



falco-exporter_helm-4

在如上界面中，填写以下参数:

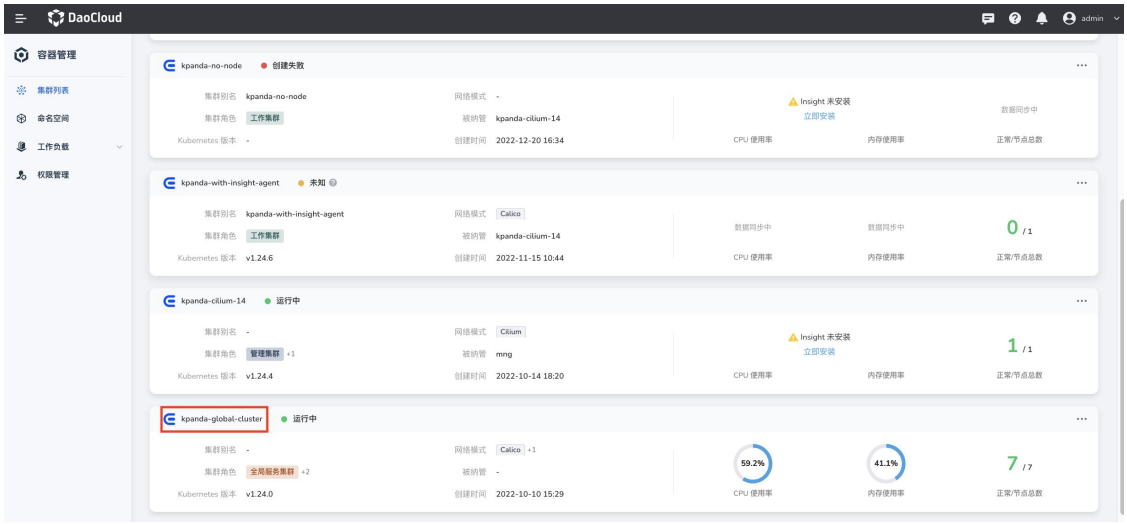
- Falco Prometheus Exporter -> Prometheus prometheusRules -> Install prometheusRules：创建 PrometheusRules，对优先事件发出警报，默认开启。
- Falco Prometheus Exporter -> Prometheus prometheusRules -> Alerts settings：警报设置，为不同级别的日志事件设置警报是否启用、警报的间隔时间、警报的阈值。

5. 点击右下角确定按钮即可完成安装。

安装 Falco

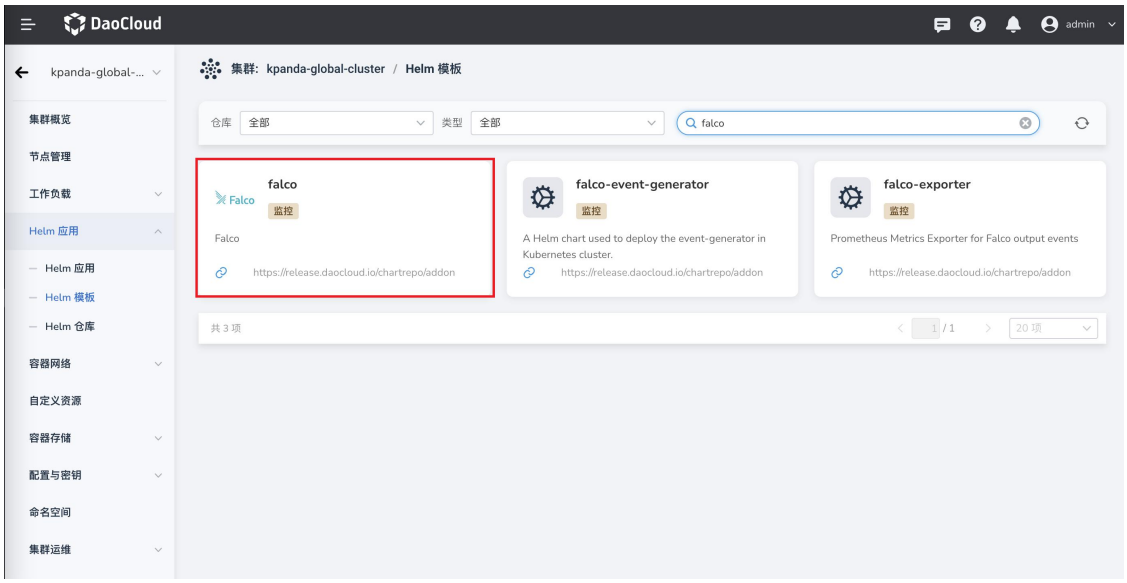
请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Falco。

1. 从左侧导航栏点击容器管理—>集群列表，然后找到准备安装 Falco 的集群名称。



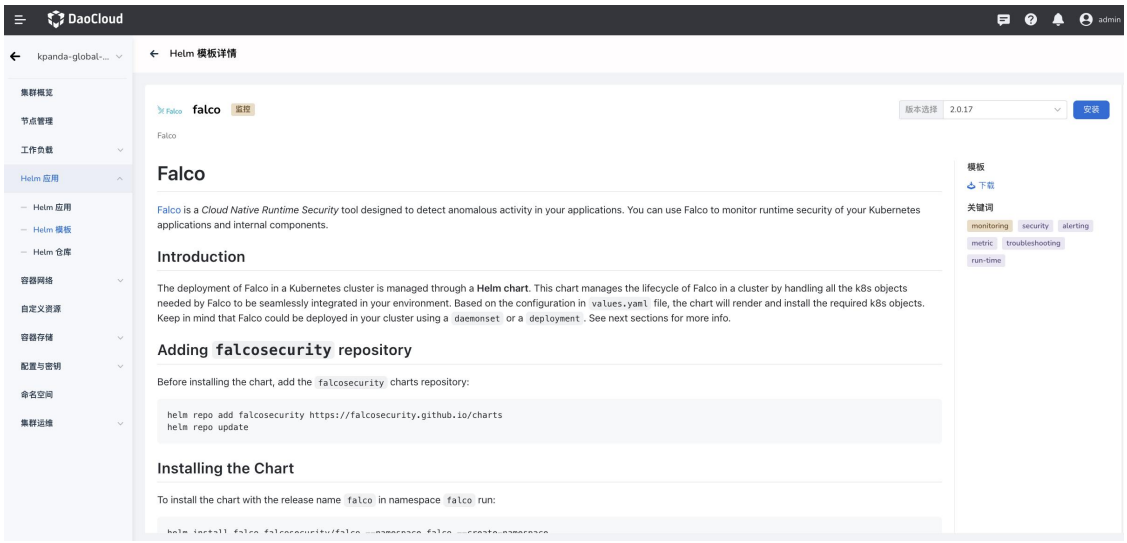
falco_cluster

2. 在左侧导航栏中选择 Helm 应用 –> Helm 模板，找到并点击 Falco。



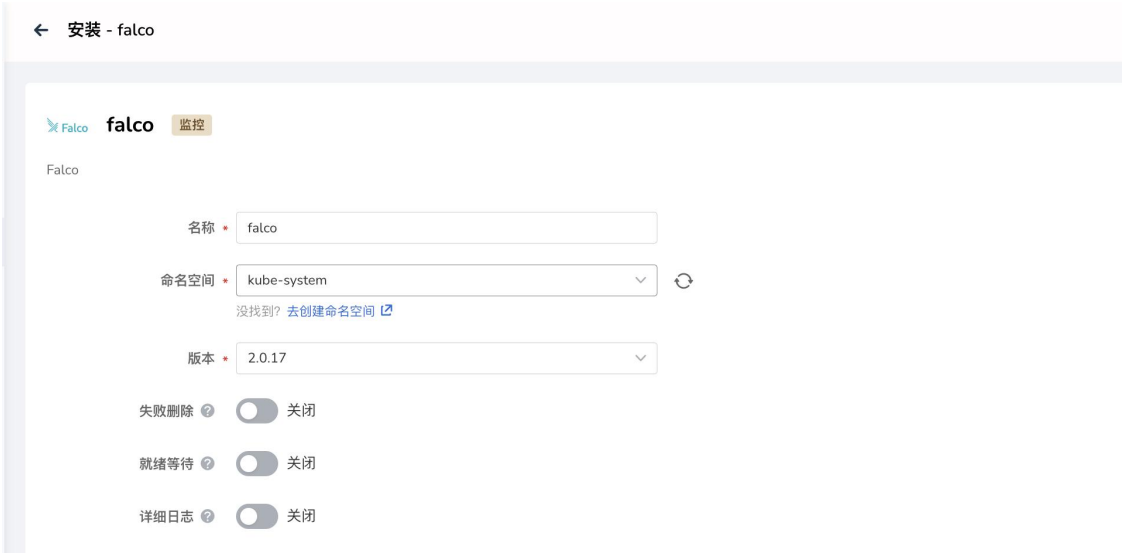
falco_helm-1

3. 在版本选择中选择希望安装的版本，点击安装。



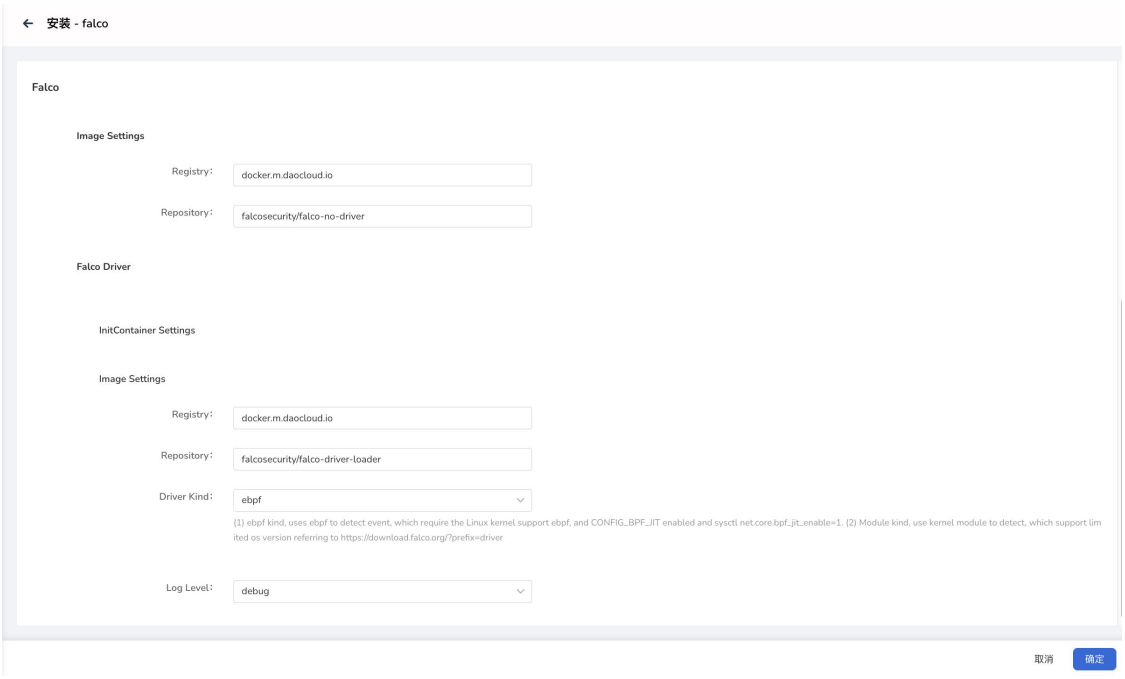
falco-helm-2

4. 在安装界面，填写所需的安装参数。



falco_helm-3

在如上界面中，填写应用名称、命名空间、版本等。



falco_helm-4

在如上界面中，填写以下参数：

- Falco -> Image Settings -> Registry：设置 Falco 镜像的仓库地址，已经默认填写可用的在线仓库。如果是私有化环境，可修改为私有仓库地址。
- Falco -> Image Settings -> Repository：设置 Falco 镜像名。
- Falco -> Falco Driver -> Image Settings -> Registry：设置 Falco Driver 镜像的仓库地址，已经默认填写可用的在线仓库。如果是私有化环境，可修改为私有仓库地址。
- Falco -> Falco Driver -> Image Settings -> Repository：设置 Falco Driver 镜像名。
- Falco -> Falco Driver -> Image Settings -> Driver Kind：设置 Driver Kind，提供以下两种选择：
    1. ebpf：使用 ebpf 来检测事件，这需要 Linux 内核支持 ebpf，并启用 CONFIG_BPF_JIT 和 sysctl net.core.bpf_jit_enable=1。
    2. module：使用内核模块检测，支持有限的操作系统版本，参考 module 支持系统版本。
- Falco -> Falco Driver -> Image Settings -> Log Level：要包含在日志中的最小日志级别。

  可选择值为：emergency, alert, critical, error, warning, notice, info, debug。
- Falco -> Falco Driver -> Image Settings -> Registry：设置 Falco Driver 镜像的仓库地址，已经默认填写可用的在线仓库。如果是私有化环境，可修改为私有仓库地址。
- Falco -> Falco Driver -> Image Settings -> Repository：设置 Falco Driver 镜像名。

- Falco -> Falco Driver -> Image Settings -> Driver Kind：设置 Driver Kind，提供以下两种选择：

    1. ebpf：使用 ebpf 来检测事件，这需要 Linux 内核支持 ebpf，并启用 CONFIG_BPF_JIT 和 sysctl net.core.bpf_jit_enable=1。
    2. module：使用内核模块检测，支持有限的操作系统版本，参考 module 支持系统版本。

- Falco -> Falco Driver -> Image Settings -> Log Level：要包含在日志中的最小日志级别。

  可选择值为：emergency、alert、critical、error、warning、notice、info、debug。

5. 点击右下角确定按钮即可完成安装。

体验社区版

欢迎安装体验!

DaoCloud Enterprise 社区版

DCE 社区版以云原生底座为核心，自研开源组件加持，提供云原生计算、网络、存储等能力，支持多云、多集群全生命周期统一纳管， 屏蔽底层基础设施复杂性，降低企业使用云原生应用门槛，提高应用开发效率。

解锁社区版三步曲

1st 安装教程传送门

!!! tip

1. 根据自身情况，选择安装教程
2. 安装成功后，即可生成登录 URL
3. 以默认账户和密码 admin/changeme 登录，申请免费许可证

在线安装

===“新手尝鲜”

适用范围: 云原生小白和爱好者，在机器资源不充裕，并且对 Kubernetes 集群的了解还不够深入的情况下使用

预计时间: 10–30 分钟

前情描述: 新手尝鲜安装模式能够帮助您零配置快速部署 DaoCloud Enterprise 社区版和 Kubernetes，
对于刚接触 DaoCloud Enterprise 社区版并想快速上手的用户，是绝佳的选择。需要注意的是，
该方式安装的特点仅支持通过节点端口的方式暴露 DaoCloud Enterprise 社区版来访问

服务。

:material-run: [开始零门槛在线安装](../install/community/kind/online.md)

===“已有 Kubernetes 集群”

适用范围: 适合已经成功搭建过标准的 Kubernetes 集群的技术人员，在机器资源充足并且能够搭建 Kubernetes 集群的情况下使用。

预计时间: 10-30 分钟

前情描述: 在已有的集群环境中，搭建一个 Kubernetes 集群即可开始安装。
该方式安装的特点是支持通过节点端口、VIP 的方式暴露 DaoCloud Enterprise 社区版来访问服务。

:material-run: [开始使用已有 K8s 集群在线安装](../install/community/k8s/online.md)

===“已有公有云 Kubernetes 集群”

适用范围: 适合已经在多种云上部署业务，面临多云管理问题的用户使用。

预计时间: 10-30 分钟

前情描述: 直接在公有云平台创建一个 Kubernetes 集群，其余安装操作与标准 Kubernetes 集群在线安装一致。
该方式安装的特点是支持通过公有云厂商提供的 CLB 方式暴露 DaoCloud Enterprise 社区版来访问服务，
且支持接入和管理市面上主流的 Kubernetes 发行版。

:material-run: [开始使用已有公有云 K8s 集群在线安装](../install/community/k8s/online.md)

离线安装

如果您的网络受限，可以选择使用 Kubernetes 集群离线安装。

适用范围: 适合安装环境为企业内网或其他网络需求

预计时间: 10–30 分钟

前情描述: 离线安装对网络没有要求，不过需要提前下载安装包

:material–run: 开始离线安装

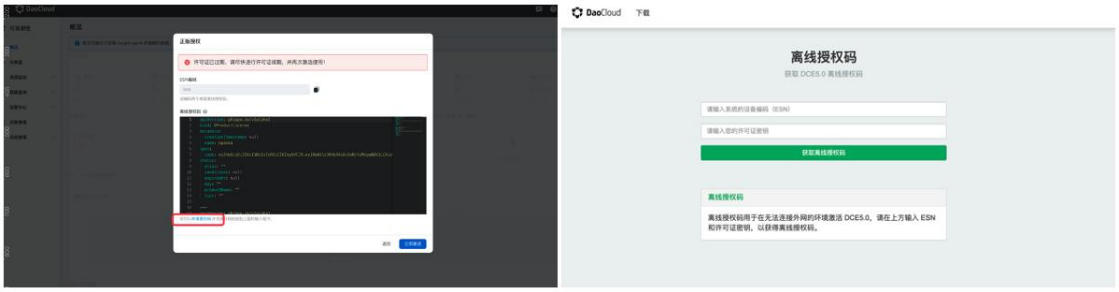
2nd 申请许可证密钥

1. 点击此处免费申请「DCE 社区版」许可证密钥，许可证密钥将发送至您的邮箱 （很多邮箱可能会拦截许可证密钥的邮件，如长时间未收到，可查看是否归为垃圾邮件）。



check mail

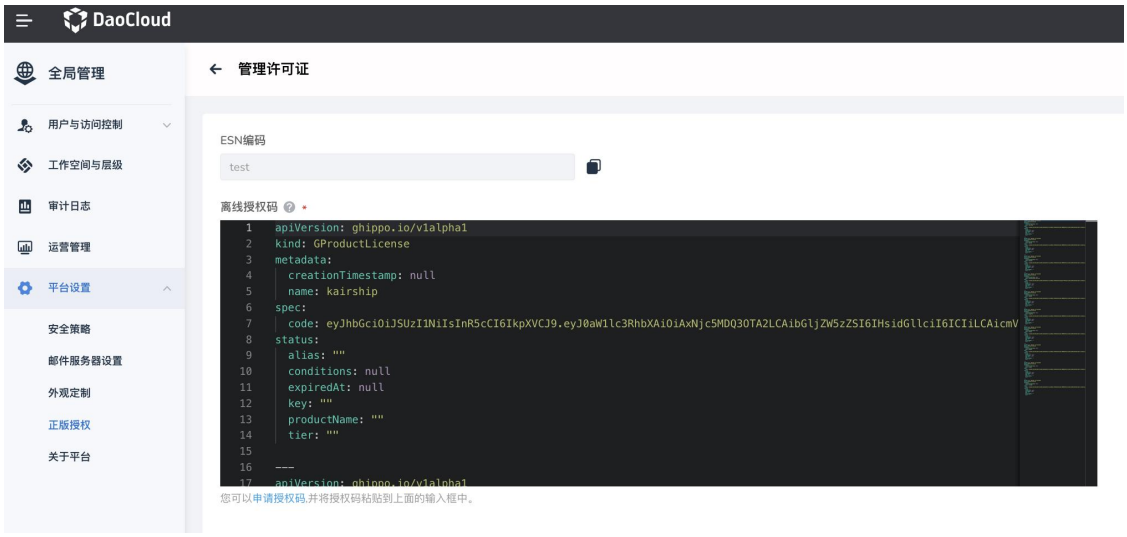
2. 在等待邮件的过程中，按屏幕提示，可先进入社区版登录页面（通过安装成功后获取的链接进入），点击申请授权码。



get code

3. 根据上图，您需要再获取 ESN（即集群系统独有的设备编码），具体获取方式为：

    1. 登录 DCE 社区版
    2. 点击左侧导航栏全局管理 -> 平台设置 -> 正版授权
    3. 点击管理许可证按钮，打开管理许可证页面，复制 ESN 编码。



esn

4. 收到许可证邮件后，回到获取离线授权码页面，输入设备独有的 ESN， 输入许可证密钥后，即可获得离线授权码。



get offline

5. 复制并粘贴上一步换取的离线授权码，再次回到 ESN 编码的获取页面，在下方代码块输入离线授权码，点击立即激活。恭喜您！现在可以探索全新的 DCE 社区版啦！

!!! note

由于在申请许可证时，选择的有效期过短，有些小伙伴会出现许可证过期的情况，这时，可点击[查看更多续期操作](../dce/license0.md#_1)。

3rd 开启云原生之旅

- 云原生技术实践公开课：适合零基础或初级云原生开发者学习，涵盖云原生发展背景、基础知识、操作实践等内容。
- 视频教程：涵盖容器管理、全局管理、可观测性这三大模块功能的使用演示，更有商业版功能演示教程，欢迎大家参阅。

进一步了解 DaoCloud Enterprise 社区版

什么是容器管理 什么是全局管理 什么是可观测性

网络面临的挑战

多云多集群下的网络连通性

多集群环境下，不仅要考虑各集群使用不同的网络 CNI，还需要实现各集群互联互通。 因此，需要解决 Cluster IP 互通、DNS 互通等问题。

多云多集群下的统一 IP 资源管理

多云多集群环境下要实现网络互通，集群与集群内的 IP 不能重复及冲突。因此，要基于多集群视角进行 IP 管理和规划，避免 IP 冲突、网段冲突（Underlay IP、Service IP 等）。

单一 CNI 难以满足多样化场景需求

面对多样化的网络场景，目前不同类型应用的网络需求不尽相同。单一的网络 CNI 很难满足多样化的场景需求。因此，在特定场景中需要使用特定 CNI 满足业务需求，在较复杂的场景中，还需要进行 CNI 组合。

固定并且可对外访问的 IP 需求旺盛，但缺乏灵活高效的 IPAM 机制

基于 Underlay CNI 实现固定并且可对外访问的 IP，但 Underlay CNI 中的 IP 资源相对紧缺，并且需要防火墙严格管控。因此，对于 Underlay IP 的使用，需要严格规划和分配。 当 IP 地址释放及分配失败时，需要及时回收，防止资源浪费。

网络安全策略和网络流量统一管理

需要统一不同集群的网络安全策略下发，以及对不同集群间的网络流量进行管控和加密。

软硬结合、网络加速等新技术结合带来的新挑战

为满足高性能、低延迟的网络需求，同 SR-IOV 等基于硬件的虚拟化加速网络方案以及 eBPF 等网络加速方案的结合也是云原生容器网络面临的一大机遇与挑战。

Spidernet Release Notes

本页列出 Spidernet 的 Release Notes，便于您了解各版本的演进路径和特性变化。

其中包含的 Spiderpool 是 DaoCloud 自研开源的 IPAM 模块，请参考 Spiderpool Release Notes。

2023-07-28

v0.8.1

适配 **Spiderpool 版本**：v0.6.0

新功能

- **新增** Spiderpool CR 中新增 nodeName、multusName 字段，用于支持节点拓扑，能按需配置网络
- **新增** Spiderpool 提供了 SpiderMultusConfig CR ，简化书写 JSON 格式的 Multus CNI 配置，自动管理 Multus NetworkAttachmentDefinition CR
- **新增** Spiderpool 提供了 Coordinator 插件，解决 Underlay Pod 无法访问 ClusterIP、调谐 Pod 的路由、检测 Pod 的 IP 是否冲突、Pod 的网关是否可达等。参考 Coordinator 文档
- **新增** IPVlan 的深度支持，适用于任何公有云环境
- **新增** 支持多个默认 IP 池，简化使用成本
- **新增** CNI 插件 Ifacer，用于自动创建子接口，参考 Ifacer 文档
- **新增**通过 Pod 注解指定默认路由网卡
- **新增**自动池的回收开关支持，可以自定义自动池是否删除
- **优化** 集群子网弹性 IP 的支持，可以很好地解决应用在滚动更新时，旧 Pod 未删除，新 Pod 没有可用 IP 的问题

2023-06-28

v0.8.0

适配 **Spiderpool 版本**：v0.5.0

优化

- **新增** spidernet 定义 Multus API
- **优化** spidernet e2e 稳定性
- **修复** spidernet goproduct Proxy Config
- **优化** spidernet 默认采用 2 副本

2023-05-28

v0.7.0

适配 **Spiderpool 版本**：v0.4.1

修复

- **修复** spidernet 子网根据 IP 排序问题。

2023-04-28

v0.6.0

适配 **Spiderpool 版本**：v0.4.1

修复

- **修复** 单栈时，查看工作负载出现指针错误
- **修复** 修复添加大量 IP 时 Controller 超时
- **修复** 亲和性填写可以填写中文
- **修复** 工作负载名称与命名空间不匹配
- **修复** 固定池显示问题
- **修复** 路由删除问题
- **修复** 分页数量显示问题
- **修复** 根据 Pod 获取 IP pool 问题
- **修复** 连续快速删除的缓存问题

新功能

- **新增** 创建 IP pool 时，填写 VLAN ID
- **新增** 显示 IP 总量与已用数
- **新增** e2e 自动使用最新版 DCE 组件

2023-03-28

v0.5.0

适配 **SpiderPool 版本**：v0.4.0

优化

- **优化** Spidernet API，适配 **Spiderpool v0.4.0** 新版 CRD

修复

- **修复** 有状态负载使用了自动 IP 池，IP 池创建成功，IP 分配成功，查询 IP 池为空问题。
- **修复** 子网管理–删除子网， 选择删除一个子网，系统提示成功，但是刷新后这个子网还在。
- **修复** 点击几次“容器网卡”，拿不到数据问题。
- **修复** Spidernet Chart 中的 Spidernet–UI Service Label 不正确问题。

2023-02-28

v0.4.4

优化

- **优化** 调整 CPU 内存 Request 值。

修复

- **修复** 有状态负载使用了自动 IP 池，IP 池创建成功，IP 分配成功，查询 IP 池为空问题。
- **修复** 子网管理–删除子网，选择删除一个子网，系统提示成功，但是刷新后这个子网还在。

- **修复** 点击几次“容器网卡”，拿不到数据问题。
- **修复** Spidernet Chart 中的 Spidernet-UI Service Label 不正确问题。

2022-11-30

v0.4.3

优化

- **优化** 资源使用量，降低 CPU,内存请求量。

v0.4.2

优化

- **优化** 资源使用量，降低 CPU,内存请求量。
- **优化** 在有 IP 占用的情况下,不能删除 Subnet 或 IPPool。
- **优化** 分页问题

v0.4.1

新功能

- **新增** 基于界面进行 IP 预留，IP 解除预留等功能。
- **新增** IP 池管理，可基于界面创建、编辑、删除。
- **新增** 工作负载使用容器多网卡功能，并可选择 IP 池 并固定。
- **新增** 应用固定 IP 池可用 IP /总 IP 数量查看。

优化

- **优化** 资源使用量，降低 CPU,内存请求量。

什么是云原生网络？

DCE 5.0 云原生网络基于多个开源技术构建，不仅提供单个 CNI 网络支持，也提供多个 CNI 网络的组合方案。具体方案如下：

方案一：Cilium + MacVLAN/SR-IOV/IPVLAN + SpiderPool + Multus

此方案适用于高内核版本（4.19.57+）的 Linux 操作系统。 方案以 Multus 为调度核心，搭配多 CNI，满足不同的网络场景需求，实现跨云跨集群的网络连通性。 同时还具备灵活的 IPAM 管理能力，基于 SpiderPool 加强 Underlay 网络的 IP 管理分配及 IP 回收能力。 不同 IP 池的使用满足了不同应用通信场景的需求。 此网络组合的主要功能如下：

1. 以 Multus 为调度核心，实现 Pod 多 CNI 的 IP 分配，支持应用的多态网络通信场景。基于开源方案实现本集群内跨 CNI 的 Pod 间通信。

   如果应用没有 Pod 多网卡以及不同网络形态需求，可以不安装 Multus。

2. 以 Spiderpool 作为 Underlay CNI 的 IPAM 管理组件，实现 IP 精细化管理、灵活的 IP 规划及分配。

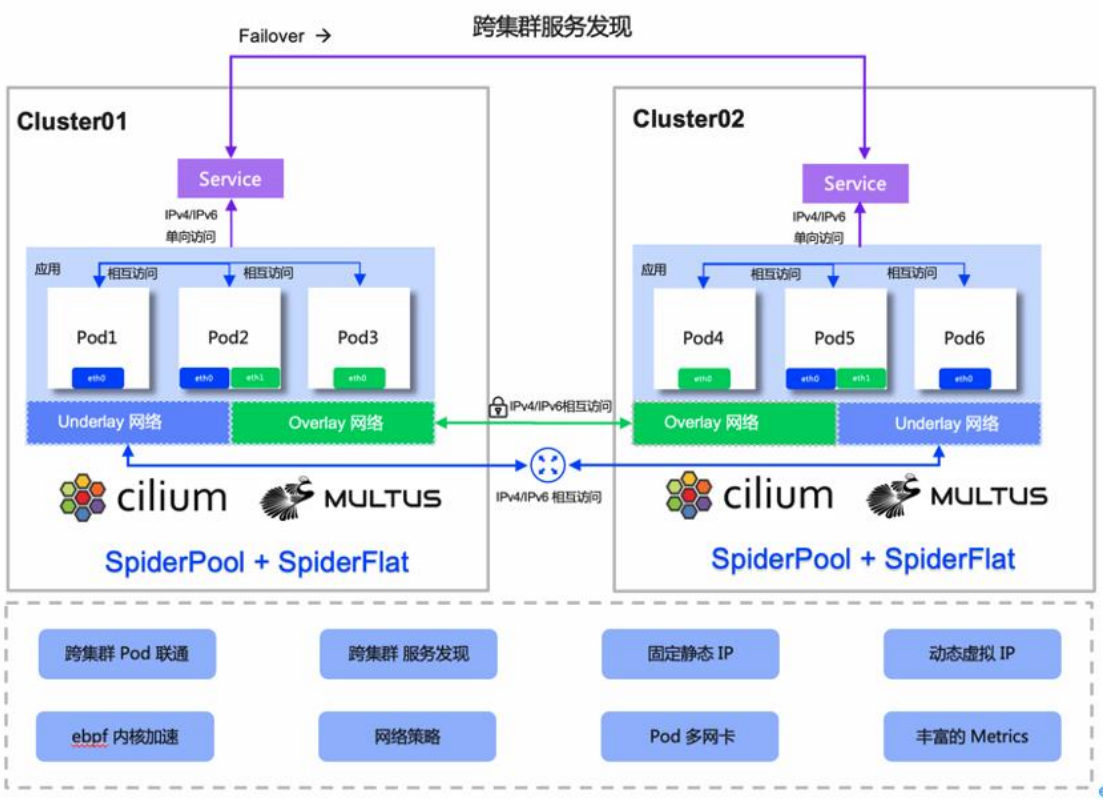
   如果在应用落地场景中未安装 Underlay CNI，可以不安装 SpiderPool。

3. Cilium 作为高性能 Overlay CNI，提供 eBPF 内核加速，实现跨集群 Pod 通信和跨集群 Service 通信，以及支持灵活的细粒度网络策略下发和丰富的流量观测能力。

   在此方案组合中，Cilium 为必备的网络 CNI。

4. 通过 MacVLAN / SRIOV / IPVLAN CNI 提供对外访问 IP，实现 Pod 二层对外通信能力。 搭配 Calico 动态虚拟网络，降低网络运维难度，并节省 IP 资源。

如果在应用落地场景中没有对外访问需求，可以不安装 Underlay CNI。



solution01

方案二：Calico + MacVLAN/SR-IOV/IPVLAN + SpiderPool + Multus

此方案适用于低内核版本的 Linux 操作系统，在用户有跨集群连通以及多 CNI 等需求时，可以采用此方案。

1. 以 Multus 为调度核心，实现多 CNI 的 IP 分配，支持应用的多态网络通信场景。基于开源方案实现本集群内跨 CNI 的 Pod 间通信。

   如果应用没有 Pod 多网卡以及不同网络形态需求，可以不安装 Multus。

2. 以 Spiderpool 作为 Underlay CNI 的 IPAM 管理组件，实现 IP 精细化管理、灵活的 IP 规划及分配。

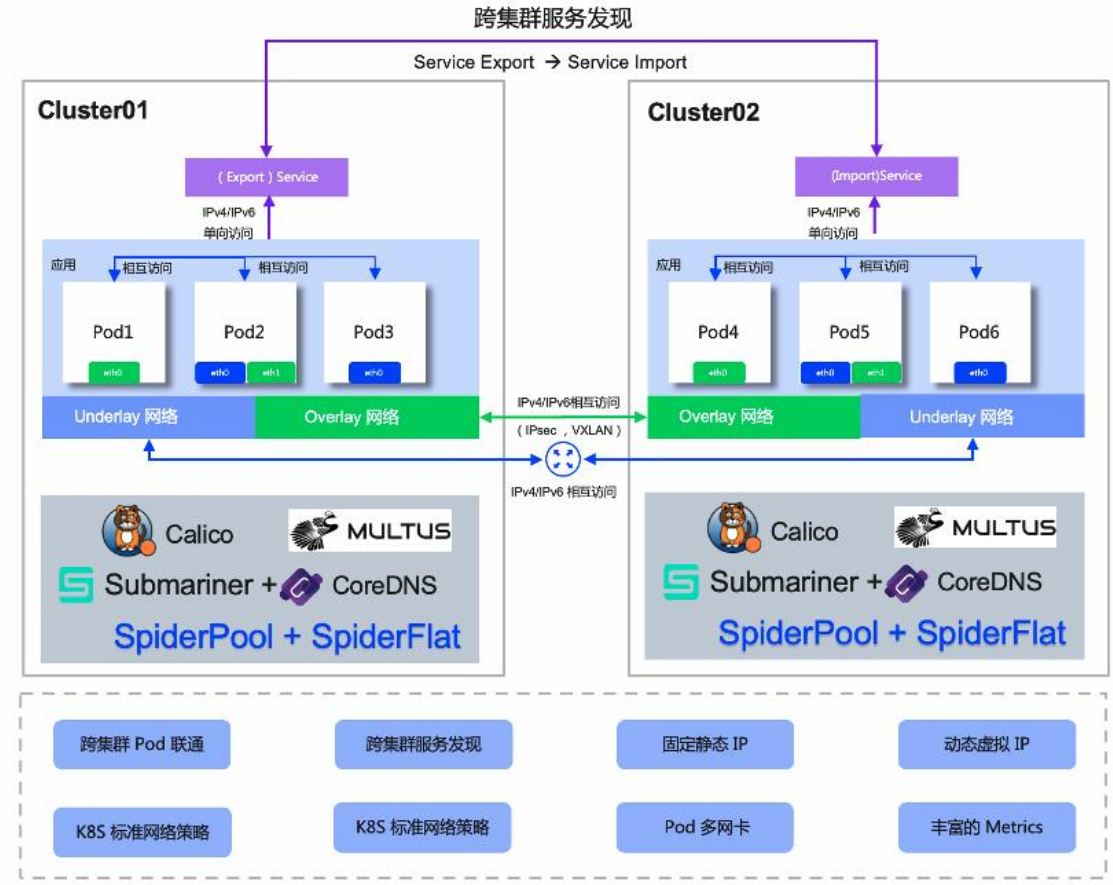
如果在应用落地场景中，未安装 Underlay CNI，可以不安装 SpiderPool。

3. 通过 MacVLAN / SRIOV / IPVLAN CNI 提供对外访问 IP，实现 Pod 二层对外通信能力。 搭配 Calico 动态虚拟网络，降低网络运维难度，节省 IP 资源。

   在此方案组合中，Calico 为必备的网络 CNI。如果在应用落地场景中，没有对外访问需求，可以不安装 Underlay CNI。

4. 通过 Submariner 组件打通跨集群 Pod 间的通信，结合 Submariner 及 Core DNS 服务发现，实现集群间的服务发现能力。

   Submariner 可根据需求选择安装。



solution02

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

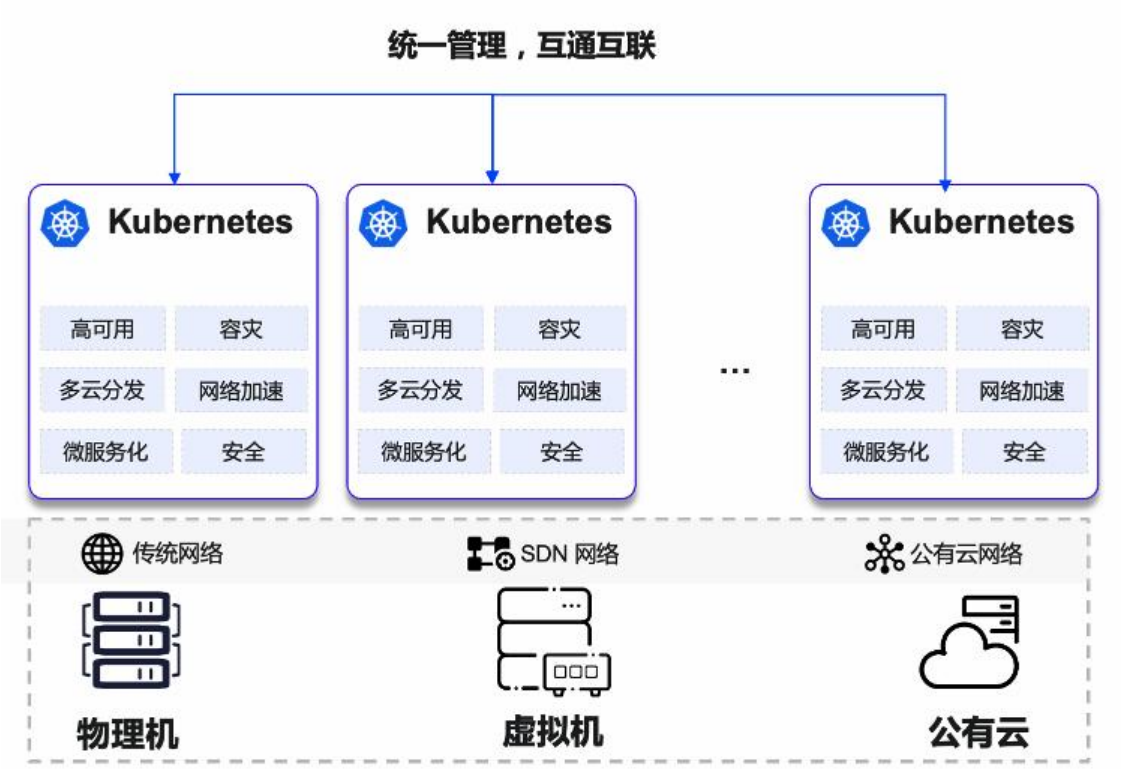
网络背景

多云混合云

云原生应用场景逐渐规模化，上云应用不断激增，场景越来越复杂。 为应对高可用、容灾、业务流量激增等场景，用户逐步采用多集群、跨云部署，而在方案的实施过程中，应用分发、应用联通以及平台管理等问题逐渐凸显。

其中，网络是整体方案落地的关键，主要面临的问题如下：

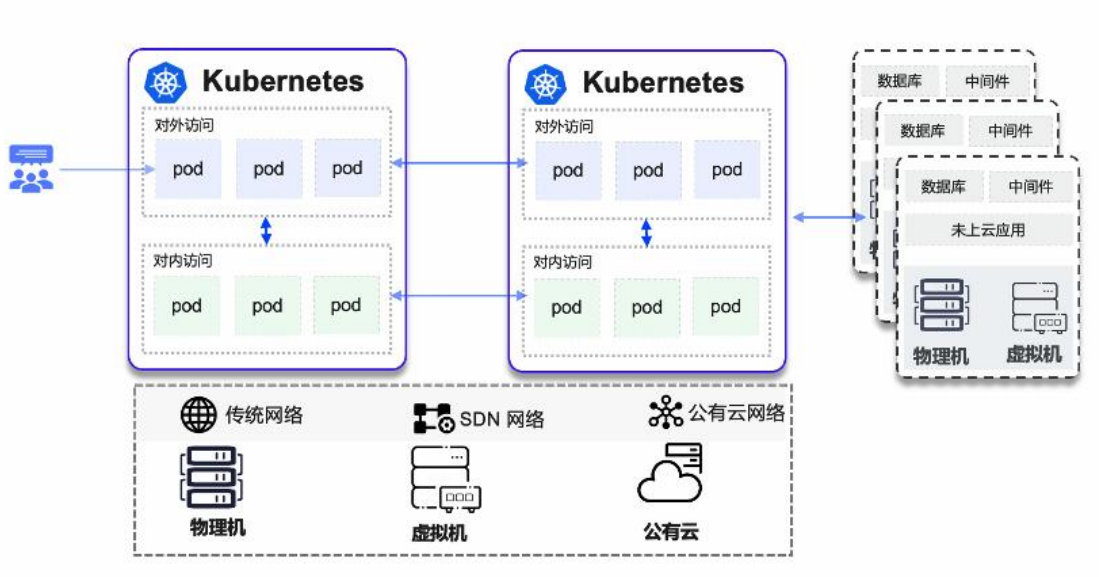
- 多集群管理需求增加，而且底层基础设施多样化，需要考虑不同的公有云、私有云以及跨地域环境。因此，在环境异构的场景下，需要异构集群网络的互通互联，并且具备公共服务跨集群服务发现的能力。
- 为保障高可用，微服务应用负载在不同集群中互为主备。在此场景中，同样需要保障网络的互联互通。



all-in-one

传统应用云原生化

- IP 管理：一些传统行业如制造、教育、能源在上云的过程中，应用常常未经微服务改造，很多应用仍需要通过某个固定 IP 进行访问。而且，此类 IP 也需要进行强管控，如严格的防火墙管控。因此，需要更灵活及更高效的 IP 管理能力。
- 集群内外互联互通：在传统应用微服务化过程中，用户会逐步将一部分应用容器化。部分容器化的应用需要具备可对外访问的 IP，实现集群内外互联互通。因此，此类应用同样也需要使用固定并且可对外访问的 IP。
- 网络类型多样化：基于对外访问 IP 的强管控需求，用户常在一个容器云平台中开放部分应用对外访问。而对于无需对外访问的应用，仍使用动态虚拟 IP，如部分应用使用 MacVLAN CNI 进行对外访问，部分使用 Calico CNI。因此，网络类型多样化同样也逐渐成为应用上云需求。



cloud-native

数据库、机器学习等应用上云

随着云原生领域的发展，数据库，机器学习等资源敏感性应用也开始运行在 Kubernetes 平台上。 此类应用对计算能力、网络性能和时延有较高的要求。 因此， SR-IOV (Single Root I/O Virtualization) 等基于硬件的虚拟化加速网络方案和 eBPF 等网络加速方案同 Kubernetes 融合也是一个必然的发展方向。

网络安全监管要求

容器和云原生平台帮助企业实现自动化应用部署，带来巨大的业务收益。 但是平台内部的 Pod 都是扁平的，横向的网络安全隔离缺少保障。 此外，随着云原生规模的扩张，以及微服务化的推进，东西向网络流量剧增。 尽管可以通过传统的防火墙和主机安全工具保护单体应用，但对于在主机之间甚至各种云之间动态部署的容器来说，必须对东西向流量以及内部流量攻击加强监控。 企业在构建容器云平台时，需要考虑更细粒度的网络策略，提升网络安全。

网卡规划

本页介绍在创建工作负载集群前，如何基于不同网络方案组合进行网卡规划。 请在集群部署之前完成集群部署要求及网卡规划。

仅 Overlay 方案

当集群内的应用 Pod 仅需要使用单一 CNI，并且不需要提供对外访问时，仅需部署组件 Calico/Cilium，网卡规划也较简单。具体网卡规划如下：

部署环境	基础设施	网卡数	网卡说明	用途
PoC 环境	物理机虚拟机 公有云	1 张物理网卡/ Bond 网卡	Access 模式接入，默认路由所在网卡 ==[3]==	承载 K8s 管理流量、Calico Pod 流量
生产环境	物理机虚拟机 公有云	1 或 2 张物理网卡/ Bond 网卡	**网卡 1**：Access 模式接入，默认路由所在网卡**网卡 2** ==[4]== (可选)：Access 模式接入	**网卡 1**：承载 K8s 管理流量、Calico Pod 流量 **网卡 2**：Hwameistor 数据副

[3] 建议所有节点网卡名称一致

[4] 建议所有节点网卡名称一致；如需使用 Hwameistor 本地存储并开启高可用功能，可根据实际场景额外规划网卡

部署环境	基础设施	网卡数	网卡说明	用途
				本跨节点同步流量

Overlay + Underlay 方案

当集群部署应用需要使用多 CNI 组合，并需要使用 Underlay CNI（Macvlan/SR-IOV）IP 提供对外通信时，可采用的网络组件组合为 Calico + SpiderPool + Multus + Macvlan/SR-IOV（同一集群可同时部署 Macvlan 和 SR-IOV CNI）。具体的网卡规划如下：

部署环境	基础设施	网卡数	网卡说明	用途
PoC 环境	物理机虚拟机	1 张物理网卡/ Bond 网卡	Access 模式接入，默认路由所在网卡==[5]==	承载 K8s 管理流量、Calico Pod 流量、Underlay CNI (Macvlan) Pod 流量
生产环境	物理机虚拟机	2 张物理网卡 / Bond 网卡	**网卡 1**：Access 模式接入，默认路由所在网卡**网卡 2** ==[6]==：Trunk 模式接入	**网卡 1**：承载 K8s 管理流量、Calico

[5] 建议所有节点网卡名称一致

[6] 建议所有节点网卡名称一致

部署环境	基础设施	网卡数	网卡说明	用途
生产环境	物理机虚拟机	3 张物理网卡 / Bond 网卡	**网卡 1**：Access 模式接入，默认路由所在网卡**网卡 2** ==[7]==：Trunk 模式接入**网卡 3**（可选）：Access 模式接入	Pod 流量 **网卡 2**：承载 Underlay CNI（Macvlan/SR-IOV）Pod 流量 **网卡 1**：承载 K8s 管理流量、Calico Pod 流量 **网卡 2**：承载 Underlay CNI（Macvlan/SR-IOV）Pod 流量 **网卡 3**：Hwameistor 数据副本跨节点

[7] 建议所有节点网卡名称一致；如需使用 Hwameistor 本地存储并开启高可用功能，可根据实际场景额外规划网卡

部署环境	基础设施	网卡数	网卡说明	用途
				同步流量

基于 eBPF 技术实施网络策略场景下的性能测试

有关测试表明，基于 eBPF 技术实施 Pod 间访问通信的安全策略，对性能带来了一定的提升，尤其随着安全策略数量增加，其提升效果更加明显。本页介绍基于 eBPF 技术实施网络策略场景下的性能测试结果。

测试对象

在测试中，对几种实施安全策略的方案进行了比较：

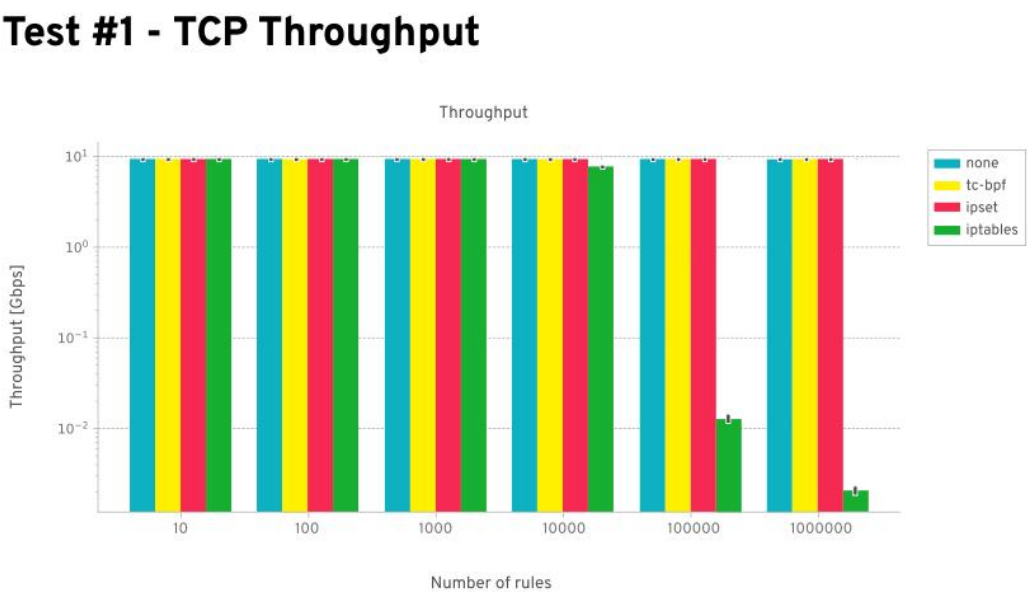
- 基于 iptables 进行数据包过滤，这种方案被 Calico 等项目所采用
- 基于 ipset 进行数据包过滤
- 基于 eBPF 技术进行数据包过滤

测试方案

在 2 个裸金属节点上，应用进行跨节点访问，在客户端 egress 方向实施安装过滤策略，最终，测试几个方案在过滤策略匹配过程中，对数据包的访问延时和吞吐量带来了多少的降低影响。

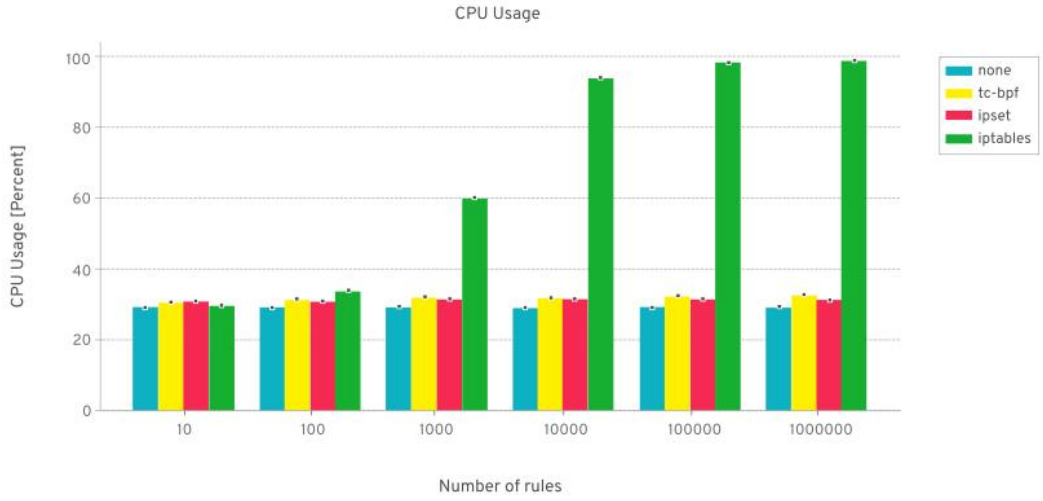
测试结果

- 在 TCP 吞吐量测试场景下，结果如下图所示。其中，当存在上万条过滤规则的规模下，iptables 方案给数据包的吞吐量带来了较大的降低：



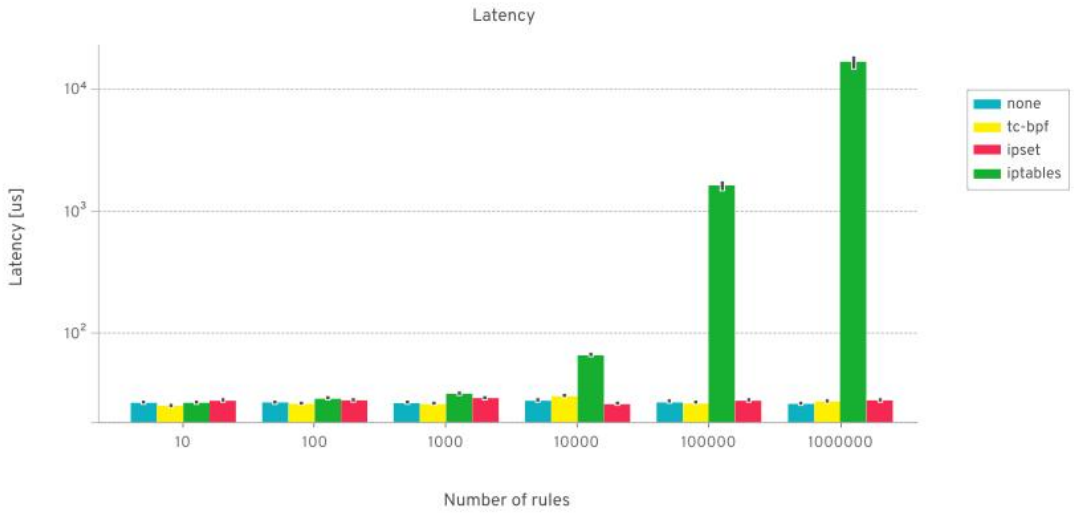
throughput

- 当存在上千条过滤规则的规模下，iptables 方案对主机的 CPU 产生了较大的开销：



cpu

- 当存在上千条过滤规则的规模下，iptables 方案对应用的访问延时带来了较大的负担



latency

结论

基于 eBPF 技术来实施网络安全策略时，即使安装了上万条规则，其对应用的通信性能也没有带来较大的干扰，比传统的基于 iptables 方案有更优的性能表现。

网络 CNI 性能测试报告

本测试报告覆盖 Cilium、Calico 及 Macvlan 三种 CNI 的性能测试。

测试背景

测试环境

型号	CPU	内存	网卡	系统	节点 IP
Dell PowerEdge R620	Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz * 2	4G 1333M * 4	Mellanox cx5 物理网卡带宽	Ubuntu 22.04	10.20.0.11
Dell PowerEdge R620	Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz * 2	4G 1333M * 4	Mellanox cx5	Ubuntu 22.04	10.20.0.12

测试 CNI 类型

本测试主要覆盖三种类型的 CNI：Cilium、Calico 及 Macvlan，并基于如下模式进行测试：

- Cilium
  - cilium (vxlan 模式)
- Calico
  - calico (vxlan 模式)

    - calico (underlay 模式)
    - callico (ipip 模式)
- Macvlan-Standalone

测试用例

本测试包括以下九种测试用例：

- 节点之间
- Pod 和 Pod 所在的节点之间
- Pod 和跨节点的节点之间
- Pod 和同节点的 Pod 之间
- Pod 和跨节点的 Pod 之间
- 节点和 Service 通信，Service 对应的 Endpoint 在此节点上
- 节点和 Service 通信，Service 对应的 Endpoint 在其他节点上
- Pod 和 Service 通信，Service 对应的 Endpoint 和此 Pod 在相同的节点
- Pod 和 Service 通信，Service 对应的 Endpoint 和此 Pod 在不同的节点

测试指标

- Pod 吞吐量（Pod 和节点之间 ）
- Service 吞吐量（Service 和节点之间）
- 时延-长连接(Pod 和节点之间)
- 时延-短连接(Pod 和节点之间)

测试工具及命令

本测试使用 netperf 工具进行性能测试，不同指标的测试命令如下：

- 吞吐量

netperf -t TCP_STREAM -H <serverIP> -p <serverPort> -l 60 -- -m 1024

iperf3 -c <serverIP> -i 1 -t 60

- 时延-长连接

  netperf -t TCP_RR -H <serverIP> -p <serverPort> -l 60 -- -r 1024 -O "MIN_LATENCY,MAX_LATENCY,P50_LATENCY,P90_LATENCY,P99_LATENCY,MEAN_LATENCY,STDDEV_LATENCY"

- 时延-短链接

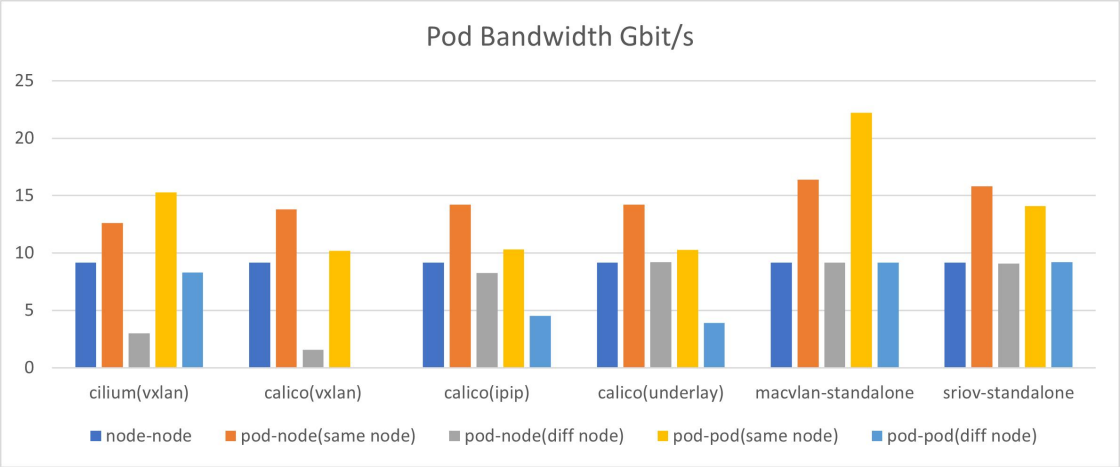
  netperf -t TCP_CRR -H <serverIP> -p <serverPort> -l 60 -- -r 1024 -O "MIN_LATENCY,MAX_LATENCY,P50_LATENCY,P90_LATENCY,P99_LATENCY,MEAN_LATENCY,STDDEV_LATENCY"

测试结果

针对四种测试指标，不同 CNI 模式在不同测试用例中的测试结果如下所示：

图表结论

- Pod 吞吐量

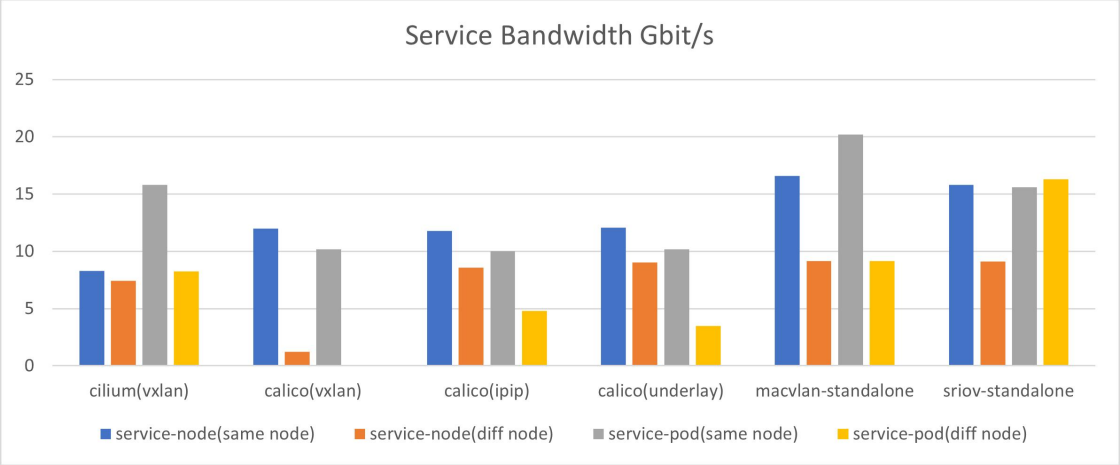


pod

测试用例 结论

测试用例	结论
Pod 和节点间，位于同节点	macvlan-standalone 吞吐量最大，cilium vxlan 最小
Pod 和节点间，位于不同节点	Calico-ipip、calico-underlay 和 macvlan-standalone 性能较好，calico-vxlan、cilium-vxlan 较差
Pod 和 Pod 间，位于同节点	macvlan 性能最好，cilium 次之，calico 三种模式表现相近，性能都较差
Pod 和 Pod 间，位于不同节点	macvlan 性能最好，cilium-vxlan 次之，calico 较差

- Service 吞吐量

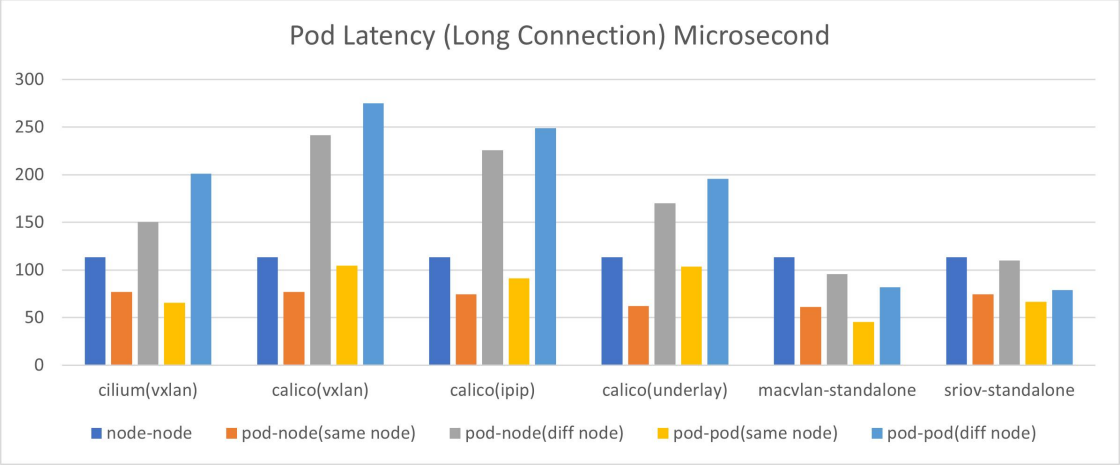


service

测试用例	结论
Service 和节点间，位于同节点	macvlan 性能最好，cilium 次之，calico 三种模式表现相近，性能都较差
Service 和节点间，位于不同节点	macvlan、calico-ipip、calico-underlay、cilium-vxlan 性能相近，calico-vxlan 较差

测试用例	结论
Service 和 Pod 间，位于同节点	macvlan 性能最好，cilium 次之，calico 三种模式表现相近，性能都较差
Service 和 Pod 间，位于不同节点	macvlan 和 cilium 性能较好，calico 较差

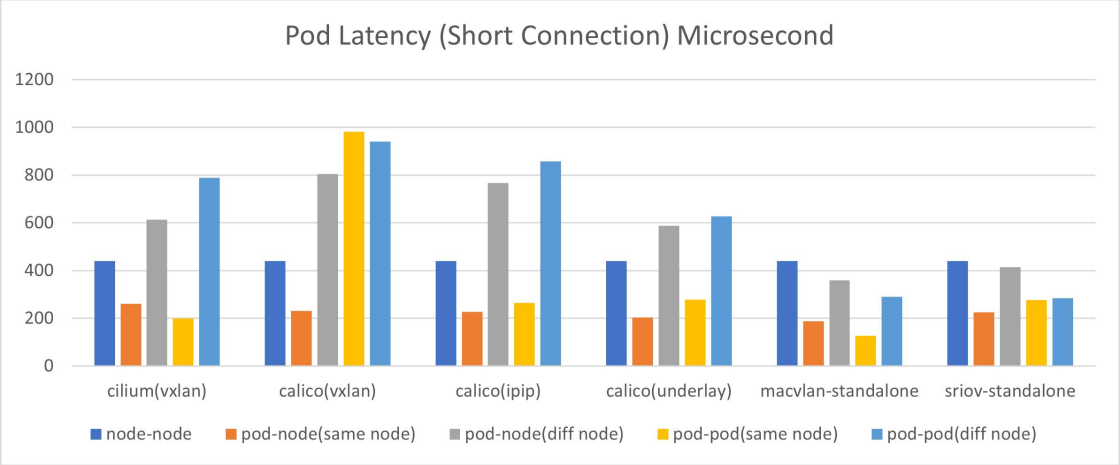
- Pod 长时延



long

测试用例	结论
Pod 和 节点间，位于相同节点	5 种 CNI 模式性能表现相近，macvlan 时延最小
Pod 和节点间，位于不同节点	macvlan 性能最好，cilium-vxlan 次之，calico-vxlan 时延最大
Pod 和 Pod 间，位于同节点	macvlan 性能最好，cilium 次之，calico 三种模式表现相近，性能相对较差
Pod 和 Pod 间，位于不同节点	macvlan 性能最好，calico-underlay 次之，calico-vxlan 最差

- Pod 短链接



short

测试用例	结论
Pod 和节点间，位于相同节点	5 种 cni 模式相差不大，macvlan 和 calico-underlay 时延最小
Pod 和节点间，位于不同节点	macvlan 性能最好，cilium-vxlan 次之，calico-vxlan 时延最大
Pod 和 Pod 间，位于同节点	macvlan 性能最好，cilium 次之，calico 三种模式表现相近，性能相对较差
Pod 和 Pod 间，位于不同节点	macvlan 性能最好，calico-underlay 次之，calico-vxlan 最差

详细数据

- Pod 相关吞吐量（Gbits/sec）

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
节点间	9.16	9.16	9.16	9.16	9.16	9.16

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan–standalone	sriov–standalone
Pod和节点间，位于相同节点	12.6	13.8	14.23	14.2	16.4	15.8
Pod和节点间，位于不同节点	3.02	1.57	8.24	9.2	9.15	9.09
Pod和Pod间，位于同节点	15.3	10.2	10.33	10.27	22.2	14.1
Pod和Pod间，	8.30	1.37	4.51	3.89	9.18	9.21

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
位于不同节点						

- Service 相关吞吐量（Gbits/sec）

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
Service 和节点间，位于同节点	8.30	12	11.77	12.07	16.6	15.8
Service 和节点间，位于不同节点	7.41	1.23	8.57	9.037	9.14	9.10
Service	15.8	10.2	10	10.17	20.2	15.6

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
和Pod间，位于同节点						
Service和Pod间，位于不同节点	8.27	1.3	4.81	3.49	9.15	16.3

- netpref 时延（长连接）（Microseconds）

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
节点间	113.27	113.27	113.27	113.27	113.27	113.27
Pod和节点间，位于	76.76	76.81	74.55	62.27	61.25	74.34

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan–standalone	sriov–standalone
相同节点						
Pod和节点间，位于不同节点	150.26	241.46	225.68	170.23	95.76	109.77
Pod和Pod间，位于同节点	65.62	104.82	91.44	103.71	45.47	66.74
Pod和Pod间，位于不同节点	201.27	275.22	248.72	195.61	81.74	78.83

- netpref 时延（短连接）（Microseconds）

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan-standalone	sriov-standalone
节点间	439.04	439.04	439.04	439.04	439.04	439.04
Pod和节点间，位于相同节点	259.92	230.65	226.26	203.63	187.41	225.66
Pod和节点间，位于不同节点	613.59	803.80	767.08	587.53	358.48	413.77
Pod和Pod间，位于同节点	198.99	982.92	265.07	277.75	127.48	275.52
Pod和	789.32	940.65	857.6	628.21	290.59	284.16

测试用例	cilium(vxlan)	calico(vxlan)	calico(ipip)	calico(underlay)	macvlan–standalone	sriov–standalone
Pod间，位于不同节点						

Spiderpool 性能测试

Spiderpool 是一个适用于 underlay 网络的高性能 IPAM CNI 插件，此文将对比其与市面上主流的 underlay IPAM CNI 插件（如 Whereabouts，Kube-OVN）以及被广泛使用的 overlay IPAM CNI 插件 calico-ipam 在 ”1000 Pod“ 场景下的性能表现。

背景

为什么要做 underlay IPAM CNI 插件的性能测试?

1. IPAM 分配 IP 地址的速度，很大程度上的决定了应用发布的速度。
2. 大规模的 Kubernetes 集群在故障恢复时，underlay IPAM 往往会成为性能瓶颈。
3. underlay 网络下，私有的 IPv4 地址有限。在有限的 IP 地址范围内，并发的创建 Pod 会涉及 IP 地址的抢占与冲突，能否快速的调节好有限的 IP 地址资源具有挑战。

环境

- Kubernetes: v1.25.4
- container runtime: containerd 1.6.12
- OS: CentOS Linux 8
- kernel: 4.18.0-348.7.1.el8_5.x86_64

Node	Role	CPU	Memory
master1	control-plane	4C	8Gi
master2	control-plane	4C	8Gi
master3	control-plane	4C	8Gi
worker4		3C	8Gi
worker5		3C	8Gi

Node	Role	CPU	Memory
worker6		3C	8Gi
worker7		3C	8Gi
worker8		3C	8Gi
worker9		3C	8Gi
worker10		3C	8Gi

测试对象

本次测试基于 0.3.1 版本的 CNI Specification，以 macvlan 搭配 Spiderpool 作为测试方案，并选择了开源社区中其它几种对接 underlay 网络的方案作为对比：

Main CNI	Main CNI 版本	IPAM CNI	IPAM CNI 版本	特点
macvlan	v1.1.1	Spiderpool	v0.4.0	集群中存在多个 IP 池，每个池中的 IP 地址都可以被集群中的任意一个节点上的 Pod 所使用，当集群中的多个 Pod 并发的从同一个池中分配 IP 地址时，存在竞争。支持托管 IP 池的全生命流程，使其同步的与工作负载创建、扩缩容、删除，弱化了过大的共享池所带来的并发或存储问题。
macvlan	v1.1.1	Whereabouts (CRD backend)	v0.6.1	各节点可以定义各自可用的 IP 池范围，若节点间存在重复定义的 IP 地址，那么这些 IP 地址上升为一种共享资源。

Main CNI	Main CNI 版本	IPAM CNI	IPAM CNI 版本	特点
Kube-OVN (underlay)	v1.11.3	Kube-OVN	v1.11.3	以子网来组织 IP 地址，每个 Namespace 可以归属于特定的子网， Namespace 下的 Pod 会自动从所属的子网中获取 IP 地址。子网也是一种集群资源，同一个子网的 IP 地址可以分布在任意一个节点上。
Calico	v3.23.3	calico-ipam (CRD backend)	v3.23.3	每个节点独享一个或多个 IP block，各节点上的 Pod 仅使用本地 IP block 中的 IP 地址，节点间无任何竞争与冲突，分配的效率非常高。

方案

测试期间，我们会遵循如下约定：

- IPv4/IPv6 双栈场景。
- 测试 underlay IPAM CNI 插件时，尽最大可能的确保可用的 IP 地址数量与 Pod 数量为 **1:1**。例如，接下来我们计划创建 1000 个 Pod，那么应当限制可用的 IPv4/IPv6 地址数量均为 1000 个。

具体的，我们会尝试以如下两种方式在上述 Kubenetes 集群上来启动总计 1000 个 Pod，并记录所有 Pod 均达到 Running 的耗时：

- 仅创建一个 Deployment，其副本数为 1000。
- 创建 100 个 Deployment，每个 Deployment 的副本数为 10。

接下来，我们会使用如下命令一次性的删除这 1000 个 Pod，记录被重建的 1000 个 Pod 再次全部达到 Running 的耗时：

```
kubectl get pod | grep "prefix" | awk '{print $1}' | xargs kubectl delete pod
```

最后，我们删除所有的 Deployment，记录所有 Pod 完全消失的耗时。

结果

单个 1000 副本的 Deployment

CNI	创建	重建	删除
macvlan + Spiderpool	2m35s	9m50s	1m50s
macvlan + Whereabouts	25m18s	失败	3m5s
Kube-OVN	3m55s	7m20s	2m13s
Calico + calico-ipam	1m56s	4m6s	1m36s

> 在测试 macvlan + Whereabouts 这个组合期间，创建的场景中 922 个 Pod 在 14m25s 内以较为均匀的速率达到 Running 状态，自此之后的 Pod 增长速率大幅降低，最终 1000 个 Pod 花了 25m18s 达到 Running 状态。至于重建的场景，在 55 个 Pod 达到 Running 状态后，Whereabouts 就基本不工作了，耗时类比于正无穷。

100 个 10 副本的 Deployment

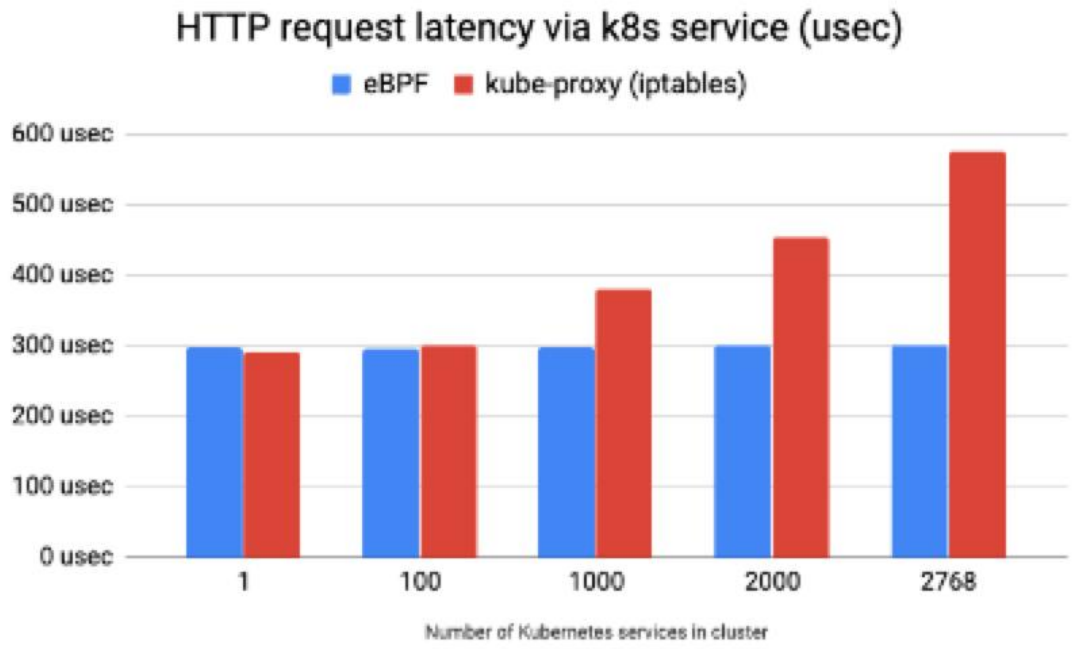
CNI	创建	重建	删除
macvlan + Spiderpool	1m37s	3m27s	1m22s
macvlan + Whereabouts	21m49s	失败	2m9s
Kube-OVN	4m6s	7m46s	2m8s
Calico + calico-ipam	1m57s	3m58s	1m35s

小结

虽然 Spiderpool 是一种适用于 underlay 网络的 IPAM CNI 插件，其相较于主流的 overlay IPAM CNI 插件，面临着更多的复杂的 IP 地址抢占与冲突的问题，但它在大多数场景下的性能表现亦不逊色于后者。

Cilium 对于 Pod 访问 Service 的加速

Cilium 在 Pod 的网络命名空间内，提前做了 Service 目的地址的 NAT 解析，而无需流量经过宿主机的 iptables 规则进行解析，因此提升了访问的性能。



cilium

在官方的测试报告中提到，在 Pod 间基于 HTTP 访问 Service 时，随着 Service 数量的增强，传统的 iptables 解析模式会使得访问延时越来越慢，而使用基于 Cilium 提供的加速方案，应用间的访问延时稳定不变。

Cilium 在 nodePort 场景下的访问加速测试

本页介绍 Cilium 在 nodePort 场景下的访问加速测试结果。

测试对象

在本测试中，对如下几个方案进行了 Kubernetes nodePort 的转发性能比较：

- 方案一：Cilium 的 eBPF XDP 加速 nodePort

  在 eBPF 的 XDP 模式下，nodePort 的转发规则能够 offload 运行到物理网卡上。该功能需要运行在裸金属机器上，需要物理硬件网卡的功能支持，当前，主流厂商的主流网卡是支持该功能的。该功能加速效果极好，硬件成本低，没有软件适配成本，是除了 DPDK 技术外的一个最佳选择

- 方案二：Cilium 的 eBPF TC 加速 nodePort

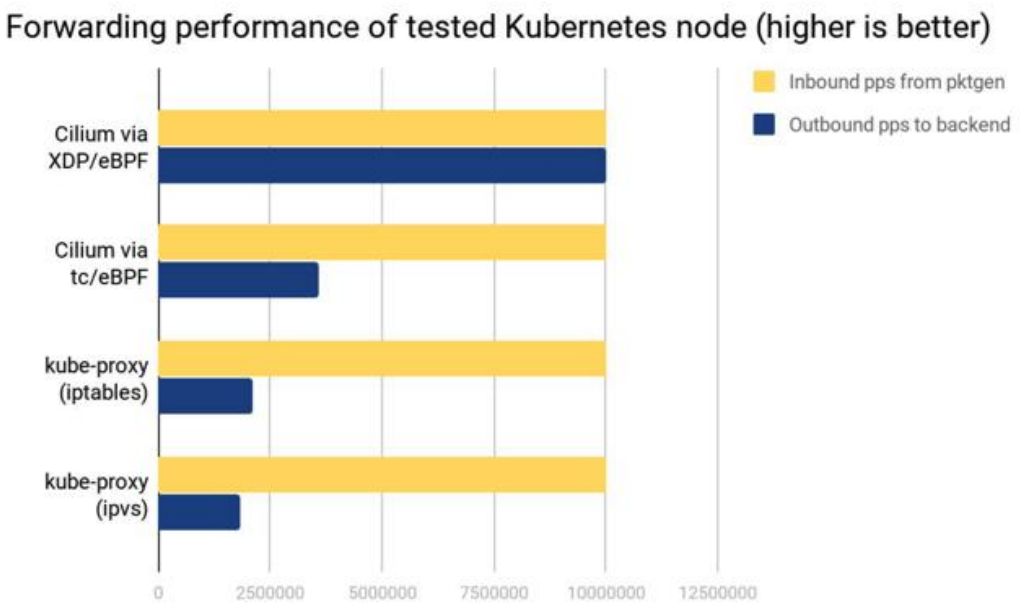
  在 eBPF 的 TC 模式下，nodePort 的转发规则在 linux 内核网络协议栈的最底层完成，加速效果欠缺于 eBPF XDP 加速，但是效果比传统的基于 iptables 转发方案还是要好很多。 该方案能运行在任意的机器上，包括裸金属机器和虚拟机，对硬件没有要求。

- 方案三：Kubernetes 传统的 kube-proxy 转发 nodePort

  kube-proxy 组件是 kubernetes 平台下完成 service 转发（包括 nodePort 转发）的一个组件，其原理是基于了 iptables 规则 或 ipvs 规则来实现，其转发性能普通。

测试一：测试 nodePort 的转发吞吐量

在裸金属机器测试环境下，使用发包工具 pktgen 产生了 10Mpps 的请求向数据，查看各个方案的 nodePort 转发吞吐量：



throughput1

从上图可以看出：

- Cilium 的 eBPF XDP 加速 nodePort

  能完全处理 10Mpps 的请求向数据，完成 nodePort 解析后，其能转发出 10Mpps 的请求向数据到后端。

- Cilium 的 eBPF TC 加速 nodePort

  对于 10Mpps 的请求向数据，该访问只能完成约 3.5Mpps 的请求。

- kube-proxy 基于 iptables 转发 nodePort

  对于 10Mpps 的请求向数据，该访问只能完成约 2.3Mpps 的请求

- kube-proxy 基于 ipvs 转发 nodePort

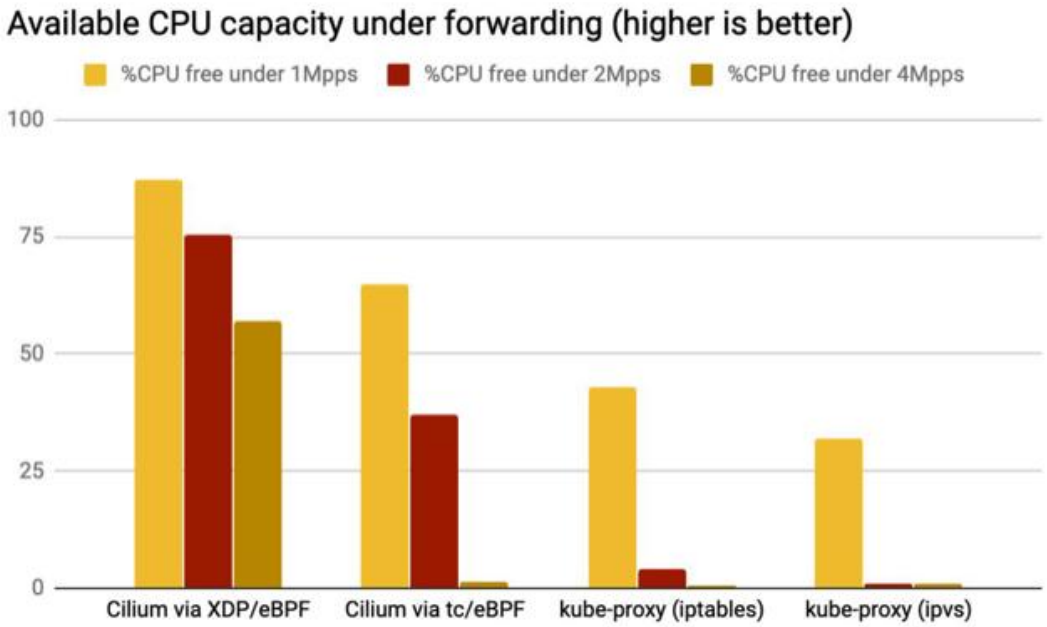
  对于 10Mpps 的请求向数据，该访问只能完成约 1.9Mpps 的请求。注：它相比 iptables 的优势要在 backend 数量很多的时候才能体现出来。

结论

Cilium 的 eBPF TC 和 eBPF XDP 加速 nodePort，其性能都远远超过了 Kubernetes 传统的 kube-proxy 组件性能，保障了集群北向访问入口的优秀网络转发性能。

测试二：测试 nodePort 转发的资源开销

在裸金属机器测试环境下，使用发包工具 pktgen 方便产生 1Mpps、2Mpps、4Mpps 的请求向数据，测试几种方案中实施 nodePort 转发主机 CPU 开销：



throughput2

从上图可以看出：

- Cilium 的 eBPF XDP 加速 nodePort 方案

  主机的 CPU 空闲率最高，意味着其转发 nodePort 请求所产生的 CPU 开销最小，即使在 4Mpps 的请求向压力下，主机还有大量的空闲 CPU 资源。

- Cilium 的 eBPF TC 加速 nodePort

主机的 CPU 空闲率比较高，意味着其转发 nodePort 请求所产生的 CPU 开销比较小，在 4Mpps 的请求向压力下，主机的 CPU 已经完全被耗尽。

- kube-proxy 基于 iptables 转发 nodePort

  主机的 CPU 空闲率比较低，意味着其转发 nodePort 请求所产生的 CPU 开销比较高，在 2Mpps 和 4Mpps 的请求向压力下，主机的 CPU 已经完全被耗尽。

- kube-proxy 基于 ipvs 转发 nodePort

  主机的 CPU 空闲率比较低，意味着其转发 nodePort 请求所产生的 CPU 开销比较高，在 2Mpps 和 4Mpps 的请求向压力下，主机的 CPU 已经完全被耗尽。

结论

Cilium 的 eBPF TC 和 eBPF XDP 加速 nodePort，其所消耗的主机 CPU 资源较低，从而能够使得这些 CPU 资源可用于保障主机上的其它业务。

Cilium 在 Pod 跨节点通信场景下的性能测试

本页介绍 Cilium 在 Pod 跨节点通信场景下的性能测试结果。

Cilium 官方提供的测试报告显示，Cilium 基于 eBPF 技术提升了跨节点 Pod 间的访问性能。该测试基于裸金属机器和 100G 网络链接下，对 Cilium 做了最大的配置调优，测试其极限的网络性能 对 Cilium 的调优，具体包括：

- 未配置任何隧道模式，使用 host routing 转发节点间的 Pod 数据
- 关闭了宿主机的 iptables 连接跟踪

测试对象

Configuration Name	Description
Baseline (Node to Node)	No Kubernetes
Cilium	Cilium 1.9.6, eBPF host-routing, kube-proxy replacement, No CT
Cilium (legacy host-routing)	Cilium 1.9.6, legacy host-routing, kube-proxy replacement, No CT
Calico	Calico 3.17.3, kube-proxy
Calico eBPF	Calico 3.17.3, eBPF datapath, No CT

cilium

测试环境配置

Item	Description
CPU	AMD Ryzen 9 3950x, AM4 platform, 3.5GHz, 16 cores / 32 threads
Mainboard	x570 Aorus Master, PCIe 4.0 ×16 support
Memory	HyperX Fury DDR4-3200 128GB, XMP clocked to 3.2GHz
Network Card	Intel E810-CQDA2, dual port, 100Gbit/s per port, PCIe 4.0 ×16
Kernel	Linux 5.10 LTS, see also Tuning Guide

env

测试工具及指标

测试使用 netperf 用来生成工作负载和收集指标。测试指标如下：

- 吞吐量（Throughput）

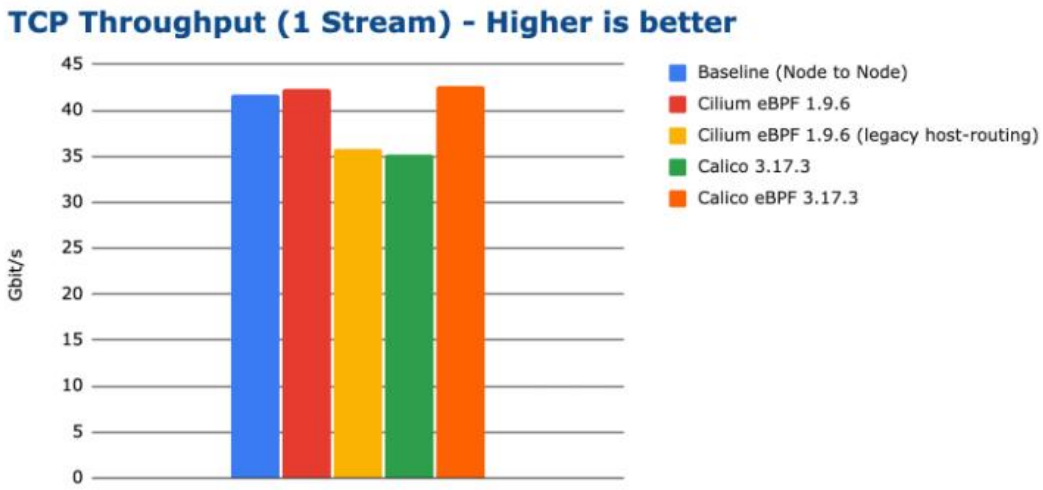
  通过单个 TCP 连接的最大传输速率和 32 个并发 TCP 连接的总传输速率。

- 请求/响应速率（Request/Response Rate, TCP_RR）

  通过单个 TCP 连接和 32 个并发 TCP 连接每秒可传输的请求/响应信息数量。

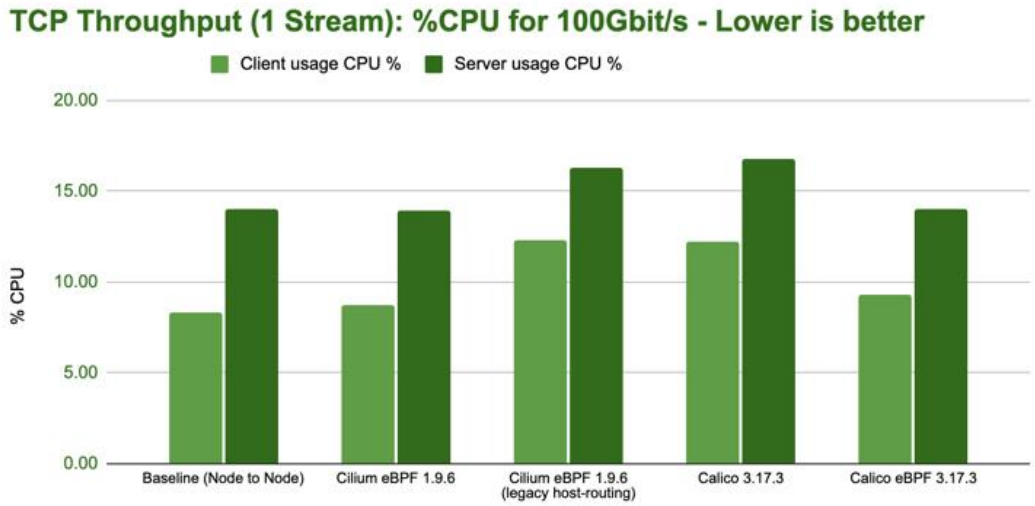
测试一

TCP 单流，测试两个跨节点 Pod 之间的 TCP 吞吐量：



single1

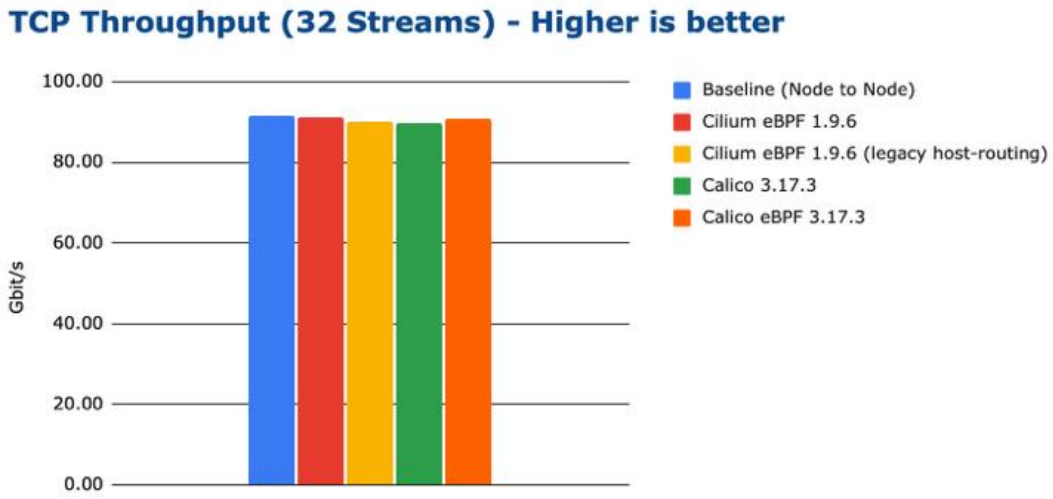
客户端和服务端的 CPU 开销：



single2

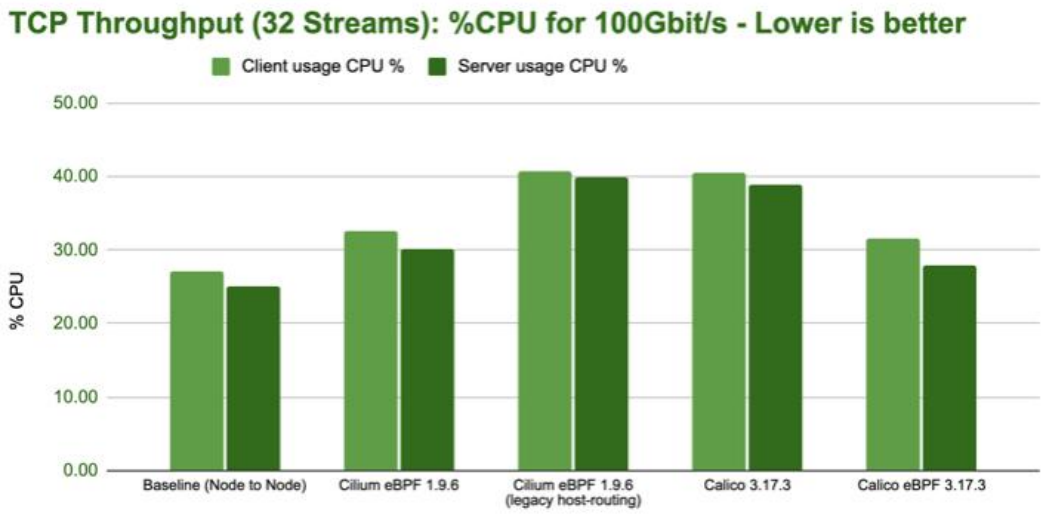
测试二

32 核 CPU，32 并发 TCP 链接，测试两个跨节点 Pod 之间的 TCP 吞吐量



multi

客户端和服务端的 CPU 开销：

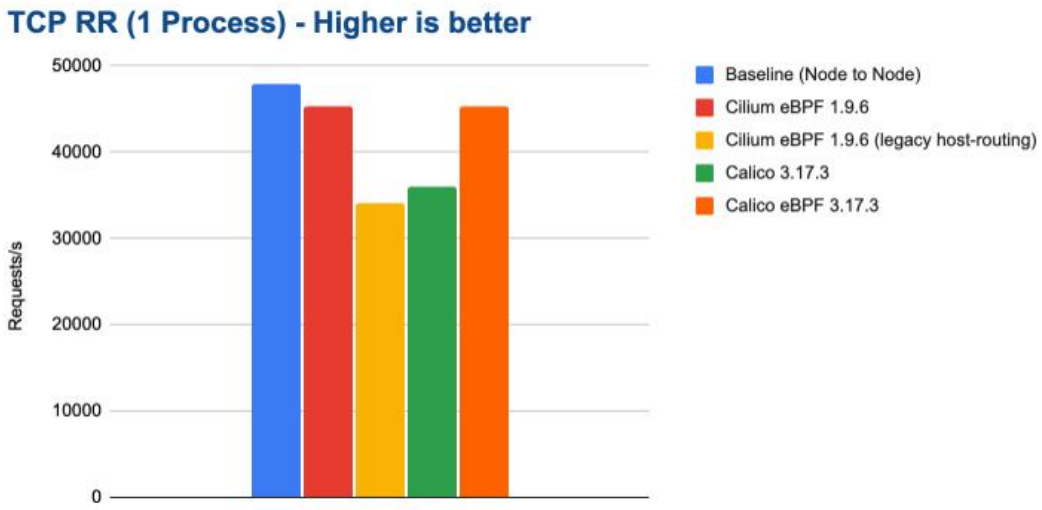


multi2

测试三

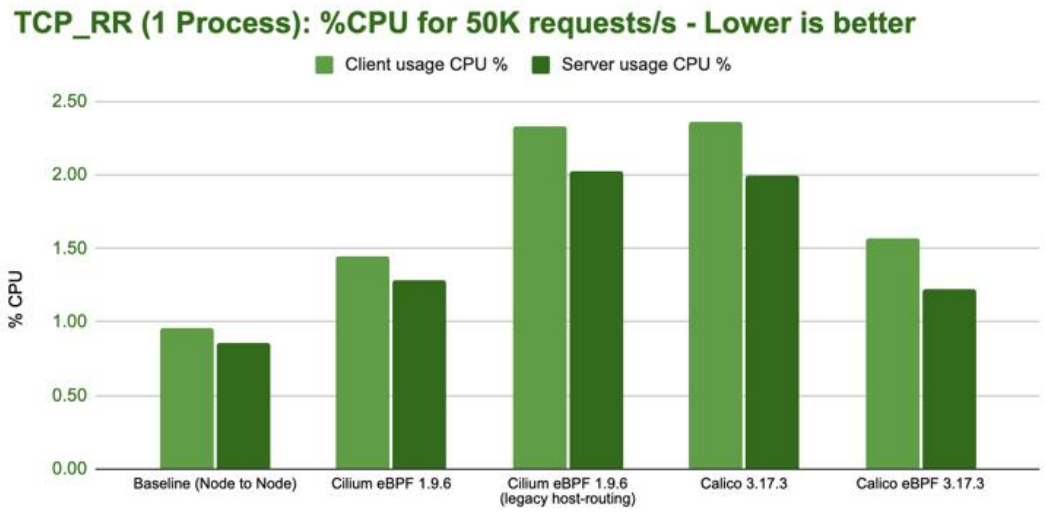
基于 TCP_RR 模式，模拟测试 7 层访问，其行为是在同一个 TCP 连接中，不断重复“发送 1 个请求，等待一个回复”

在 Pod 只有 1 核 CPU 的情况下，测试两个跨节点 Pod 之间的 TCP_RR 表现：



tcprr1

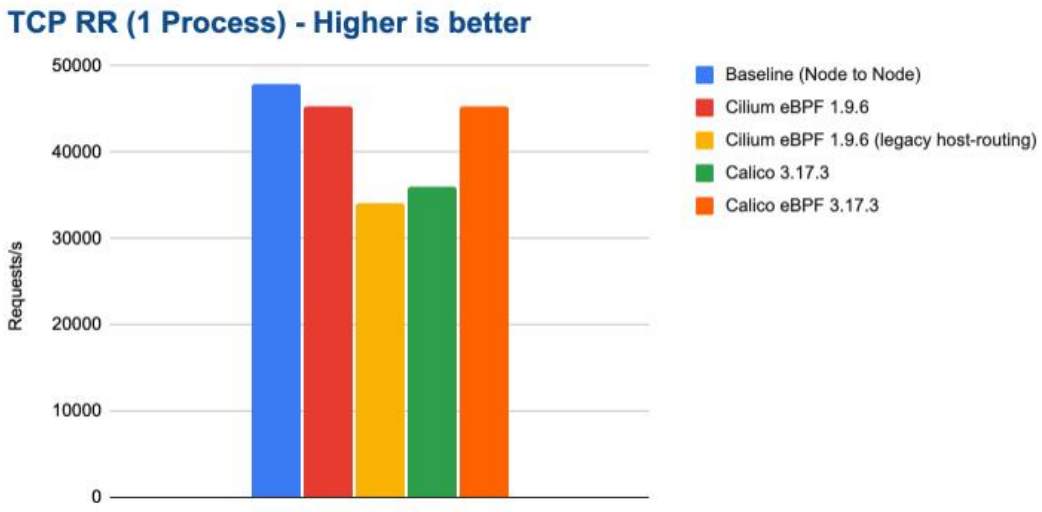
客户端和服务端的 CPU 开销：



tcprr2

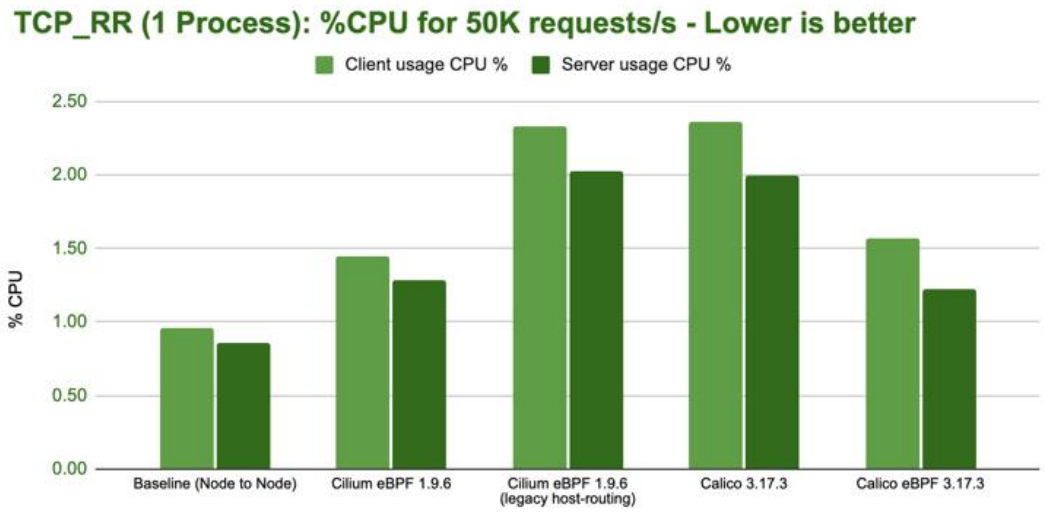
测试四

在 Pod 只有 32 核 CPU 的情况下，测试两个跨节点 Pod 之间的 TCP_RR 表现：



tcprr3

客户端和服务端的 CPU 开销：


TCP_RR (1 Process): %CPU for 50K requests/s - Lower is better
Client usage CPU %
Server usage CPU %
% CPU
2.50
2.00
1.50
1.00
0.50
0.00
Baseline (Node to Node)
Cilium eBPF 1.9.6
Cilium eBPF 1.9.6 (legacy host-routing)
Calico 3.17.3
Calico eBPF 3.17.3

tcprr4

Cilium 基于 eBPF 技术实施 Pod 带宽管控场景下的性能测试

Cilium 基于 eBPF 技术提升了 Pod 带宽的性能。本页介绍 Cilium 基于 eBPF 技术实施 Pod 带宽管控场景下的性能测试结果。

测试对象

- 在传统方案中，基于 Linux 相同的 TC 模块，使用 HTB/TBF qdisc 来实施 Pod 流量的整形管控
- 在 Cilium 中，使用 eBPF 技术实现了一种 Earlist Departure Time（EDT）限速模型，实其无锁（lockless）实施的特点，有着优秀的性能表现

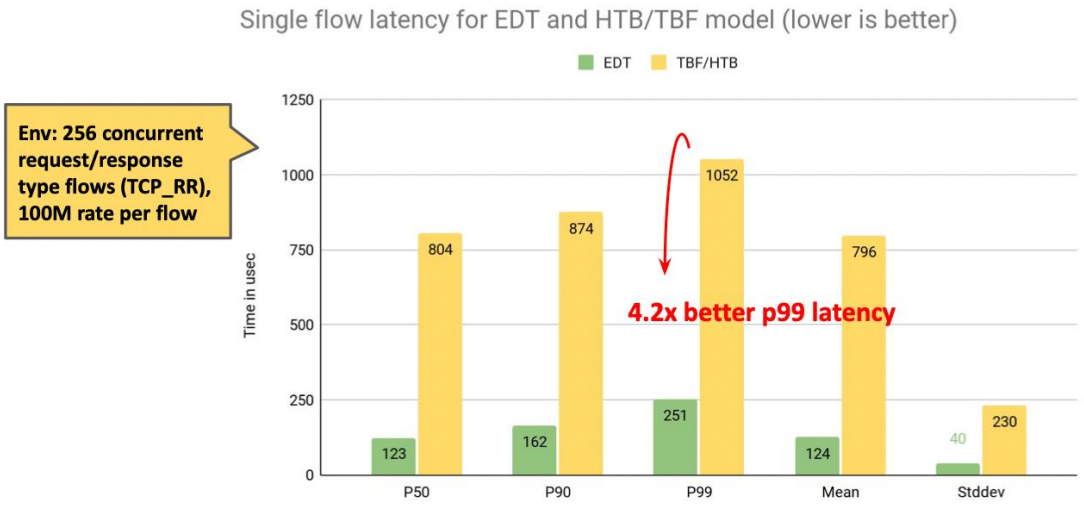
测试工具及测试环境

- 测试工具：netperf
- 测试环境：256 个并发会话的请求/响应类型流（TCP_RR），每个流的速率为 100M

测试结果

- Cilium 的 EDT Pod 带宽管控对数据包的延时影响极低，而传统的基于 HTB qdisc 的 TC 方案，对数据包通信的延时影响较大

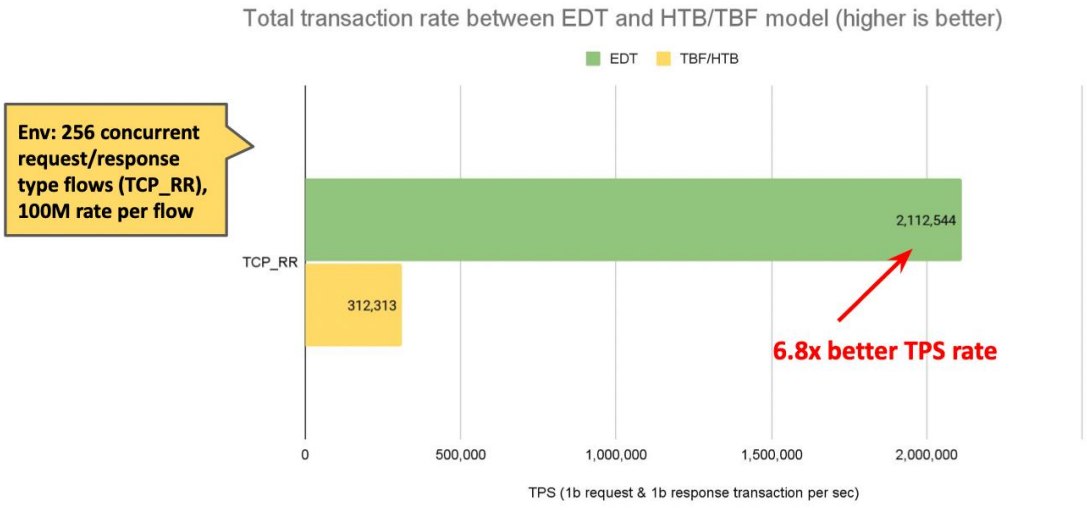
Comparison of Cilium's EDT implementation vs TBF



latency

- Cilium 的 EDT Pod 带宽管控对应用通信的 TPS 影响有限，而传统的基于 HTB qdisc 的 TC 方案，对应用通信的 TPS 影响较大

Comparison of Cilium's EDT implementation vs TBF



tps

网络组件

DCE 5.0 提供的网络方案对当前主流的开源网络组件进行优化组合，能够满足各种复杂场景的需求。



网络组件

目前，DCE 5.0 已支持的网络组件包括：

- Calico：基于 Linux kernel 技术实现的 Virtual Router，完成数据平面的转发。
- Cilium：基于 eBPF 内核构建的网络方案
- Contour：一个开源的 Kubernetes Ingress 控制器，使用 Envoy 作为数据面。
- f5networks：全面控制 F5 设备，将集群中 service 和 ingress 配置同步到 F5 硬件设备上，实现集群北向入口的负载均衡。
- Ingress-nginx：Kubernetes 社区托管的 Ingress 控制器，使用 nginx 作为反向代理和负载均衡。
- MetalLB：裸金属版的 Kubernetes 负载均衡器方案。

- Multus-underlay：基于 Multus、搭配 Macvlan + SRIOV-CNI 的多网卡方案。
- Spiderpool：自动化管理 IP 资源
- Submariner：跨集群的网络通讯方案

!!! note

以上所有 CNI 和 Ingress 等组件可以按需安装。

F5network 安装问题汇总

本页介绍安装 F5network 组件时遇到的一些问题及其解决办法。 有关 F5network 组件的安装步骤，请参考安装文档。

安装常见的一些问题汇总如下。

1. 安装 F5network 组件后 f5 ipam controller 服务无法拉起。

   如果本组件使用 4 层负载均衡模式，会安装 f5 ipam controller，而 f5 ipam controller 要求集群具备 storage 组件且提供 PVC 服务。 当没有 storage 组件时，会出现这个问题。可参考相关的存储组件安装手册进行安装。

2. Service 中指定了 cis.f5.com/ipamLabel: LabelName，但却无法分配到 IP。

   请检查集群中是否有其它 loadbalancer 组件。如果有，会导致 F5 IPAM 组件无法给 Service 分配 IP 地址，请卸载其它 loadbalancer 组件。

3. F5 端没有 Pool Member

   请检查 Service 是否具有 Endpoint。如果没有，将会导致 F5 端没有 Pool Member。

4. F5 流量无法转发到集群节点的 nodePort 上。

   请执行 kubectl describe pod <f5Name>-f5-bigip-ctlr-<xxx> -n <namespace>，检查是否设置了 —node-label-selector。 如果设置了，但是相关 node 却没有对应的 label，会导致 F5 不知道如何转发。

使用

本页从 4 层和 7 层负载均衡角度讲述如何使用 f5network

4 层负载均衡

当组件安装为 4 层负载均衡模式时，可为集群中的 service 创建 F5 的负载均衡服务，具体用法请参考 F5 官方文档。

以下给出简单的例子：

1. 当以 nodePort 转发方式工作时，必须为应用创建一个 LoadBalancer 类型的 service。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: http-server
  annotations:
    cis.f5.com/ipamLabel: LabelName
    cis.f5.com/health: '{"interval": 10, "timeout": 31}'
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
    name: http
  selector:
    app: http-server
```

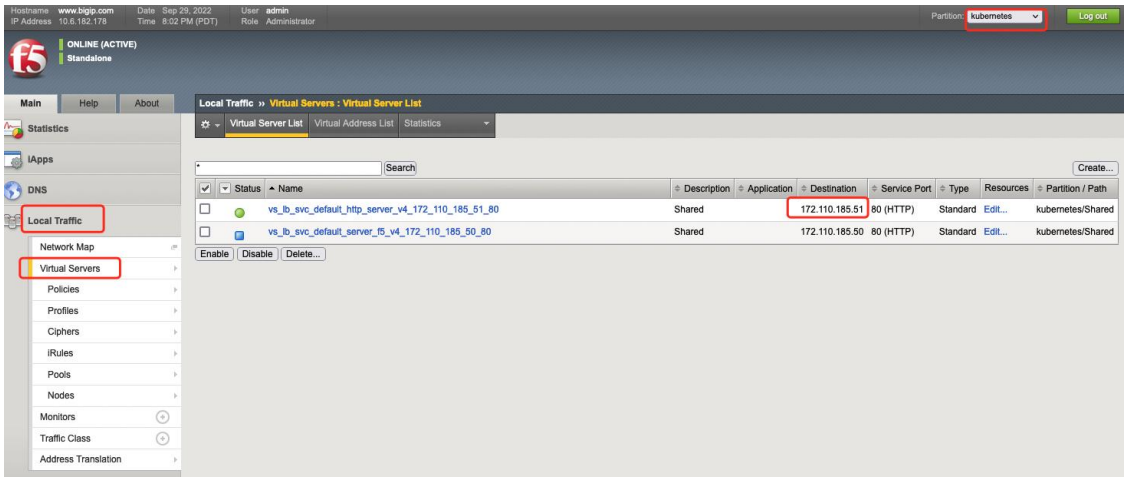
注意如上 YAML 中的 annotation：

1. 必须打上“cis.f5.com/ipamLabel”，注意替换该 key 的值，请设置为安装时填写”BIGIP L4 IP Pool”的 LabelName，否则该转发规则不会分配到 VIP。
2. “cis.f5.com/health”是可选的。
3. 更多的 annotation 支持，请参考 F5 官方文档。

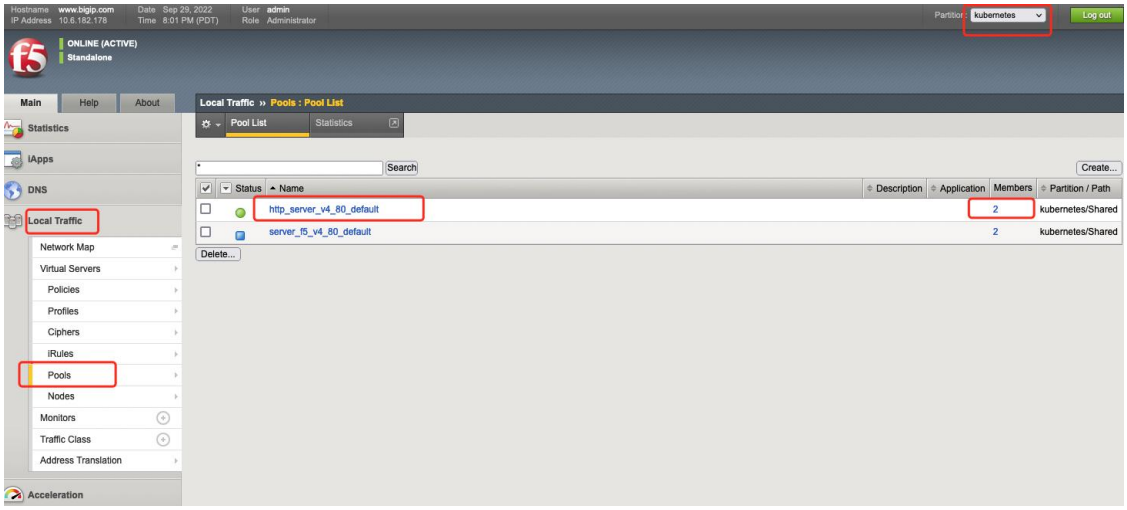
2. 创建 service 后，即可观测到分配到的 EXTERNAL-IP：

```
$ kubectl get service -o wide
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE    SELECTOR
http-server   LoadBalancer   172.21.25.10   172.110.185.51   80:31548/TCP   3d14h  app=http-server
```

3. 登录到 F5 Web UI，切换到所使用的 partition，即可观测到下发的转发规则。



f5network usage1



f5network usage1

4. 在集群外发，访问 F5 分配到的 VIP，即可访问到服务。

7 层负载均衡 http

当组件安装为 7 层负载均衡模式时，可为集群中的 ingress 创建 F5 的负载均衡服务，具体用法请参考 F5 官方文档。

以下给出简单的例子：

1. 确认 F5 的 ingressClass。

```
$ kubectl get ingressClass
NAME    CONTROLLER                  PARAMETERS    AGE
f5      f5.com/cntr-ingress-svcs    <none>        41h
```

2. 当本组件以 nodePort 转发方式工作时，必须为应用创建一个 nodePort 类型的 service 和 ingress 对象。

```
apiVersion: v1
kind: Service
metadata:
  name: http-server
spec:
```

```
  type: nodePort
  ports:
  - port: 80
    targetPort: 80
    name: http
  selector:
    app: http-server
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: http-server
spec:
  ingressClassName: f5
  rules:
  - http:
      paths:
        - path: /http-server
          pathType: Prefix
          backend:
            service:
              name: http-server
              port:
                number: 80
```

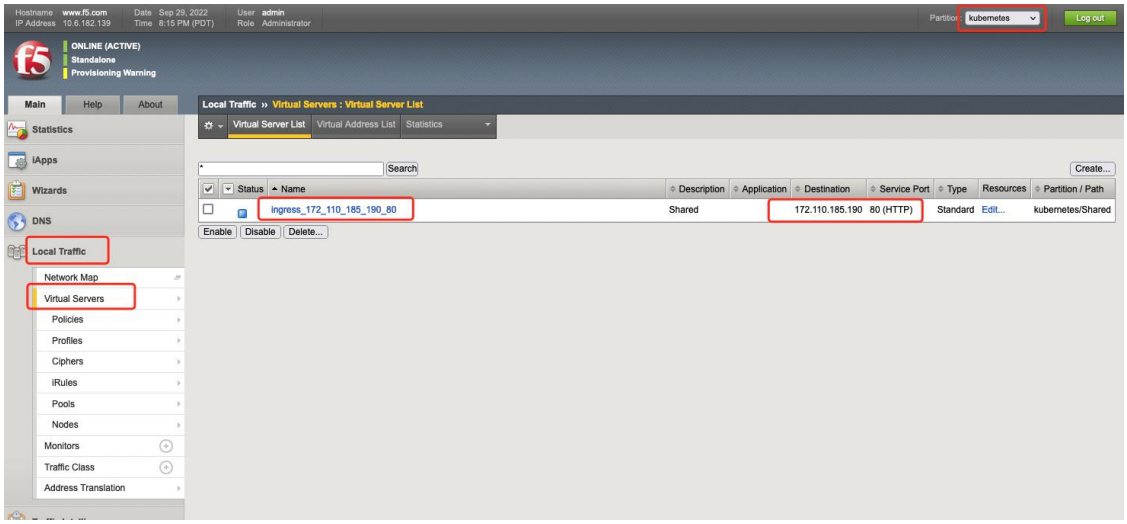
在如上 ingress yaml 中：

1. ingressClassName 使用 F5 的 ingressClass。
2. ingress 还支持更多 annotation，请参考 F5 官方文档。

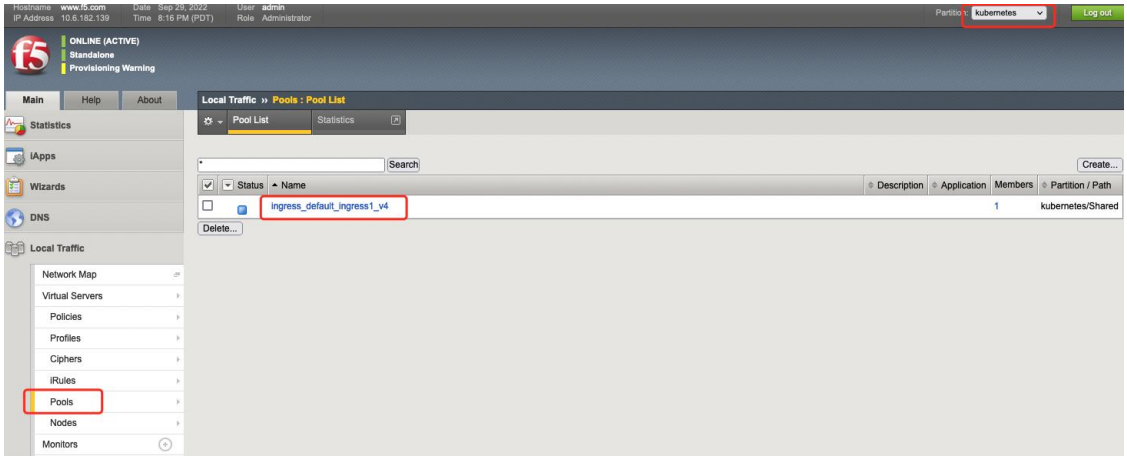
3. 创建 ingress 后，可观测到 ingress 对象分配到的 ADDRESS。

```
# kubectl get ingress
NAME          CLASS   HOSTS   ADDRESS           PORTS   AGE
http-server   f5      *       172.110.185.190   80      40h
```

4. 登录到 F5 Web UI，切换到所使用的 partition，即可观测到下发的转发规则。



f5network usage3



f5network usage4

5. 在集群外发，访问 F5 分配到 URL http://VIP/http-server，即可访问到服务。

7 层负载均衡 https

1. 创建一组 TLS 证书。

2. 创建 secret（注意：secert 的 key 必须是 tls.key 和 tls.crt）。

```
kubectl  delete secret ingress1-ssl
kubectl create secret generic ingress1-ssl \
    --from-file=./tls.crt --from-file=./tls.key
```

3. 创建 ingress 对象。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: $NAME
  annotations:
    # http 重定向到 https 来访问
    ingress.kubernetes.io/ssl-redirect: "true"
    ingress.kubernetes.io/allow-http: "false"
spec:
  ingressClassName: ${INGRESS_CLASS}
  tls:
  - hosts:
      - test.example.com
      secretName: ingress1-ssl
  rules:
  ....
```

安装

本章节介绍如何安装 F5network 组件

安装须知

AS3 是 F5 设备上的一个插件，其提供了 F5 设备远程网络可配置的接口，AS3 版本和 F5 设备的软件版本是有版本匹配要求的。

k8s bigip ctlr 是 K8S 集群上的 F5 控制面组件，该组件与 K8S 之间是有版本匹配要求的，且 k8s bigip ctlr 与 AS3 也有版本匹配要求。

因此，务必了解 F5 设备的软件版本和 K8S 集群的软件版本，才能正确掌握安装什么版本的 AS3 和 k8s-bigip-ctlr。

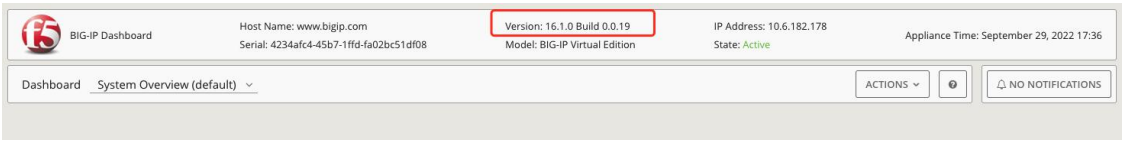
Connector	Version(s)	Platform	Version(s)	BIG-IP version(s)	AS3
CIS for Kubernetes	v1.14.1, v2.0.x-v2.10.0	Kubernetes	v1.13-v1.24	v12.x-v16.x	v3.13-v3.38

f5network version

获取最新的版本匹配关系，可参考 F5 官方文档的适配矩阵

F5 设备安装 AS3 服务

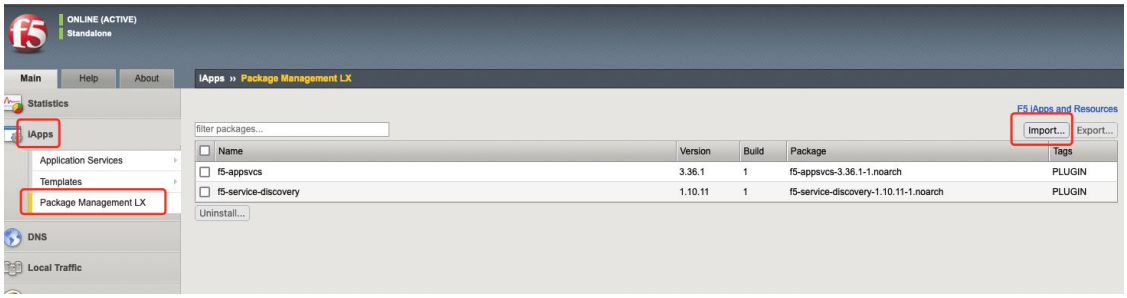
1. 登录 F5 设备的管理 Web UI，点击导航栏的 Statics -> Dashboard，获取 F5 设备的版本号。



f5 bigip version

2. 根据安装需知中的版本匹配关系，下载匹配版本的 AS3 RPM 包到本地电脑 https://github.com/F5Networks/f5-appsvcs-extension/releases。

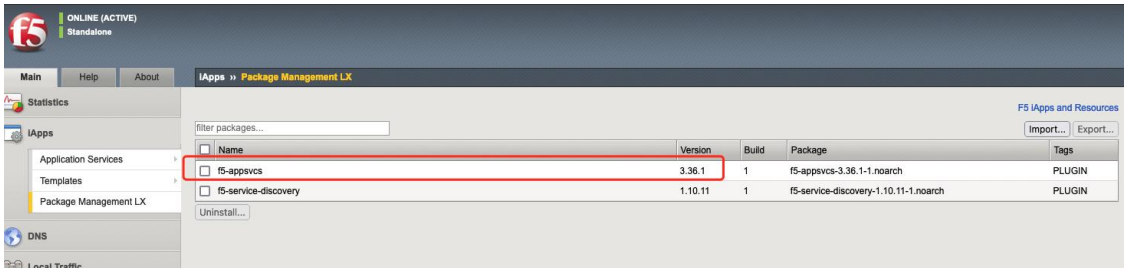
3. 登录 F5 设备的管理 Web UI，进入 iApps –> Package Management Lx 界面，点击右上角的 import。



f5 as3

4. 在 import package 界面中，点击浏览，选择本地电脑上的 AS3，完成 upload。

5. 完成安装后，可查看安装结果。

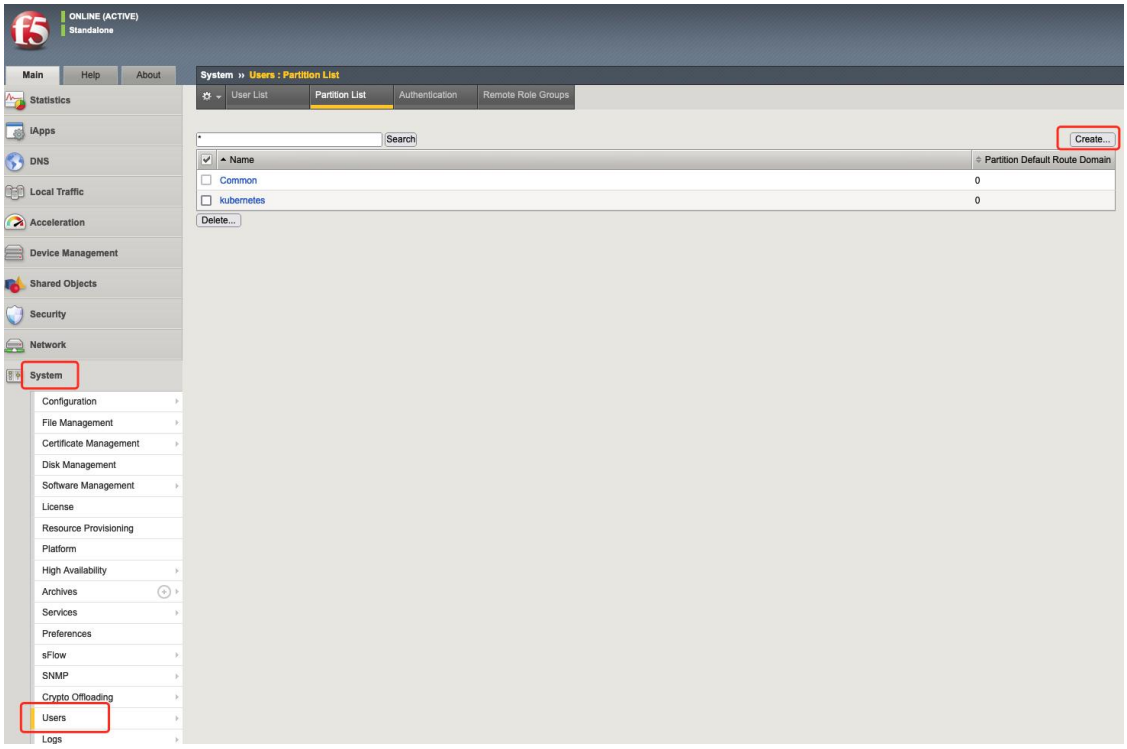


f5 as3

F5 设备创建 partition

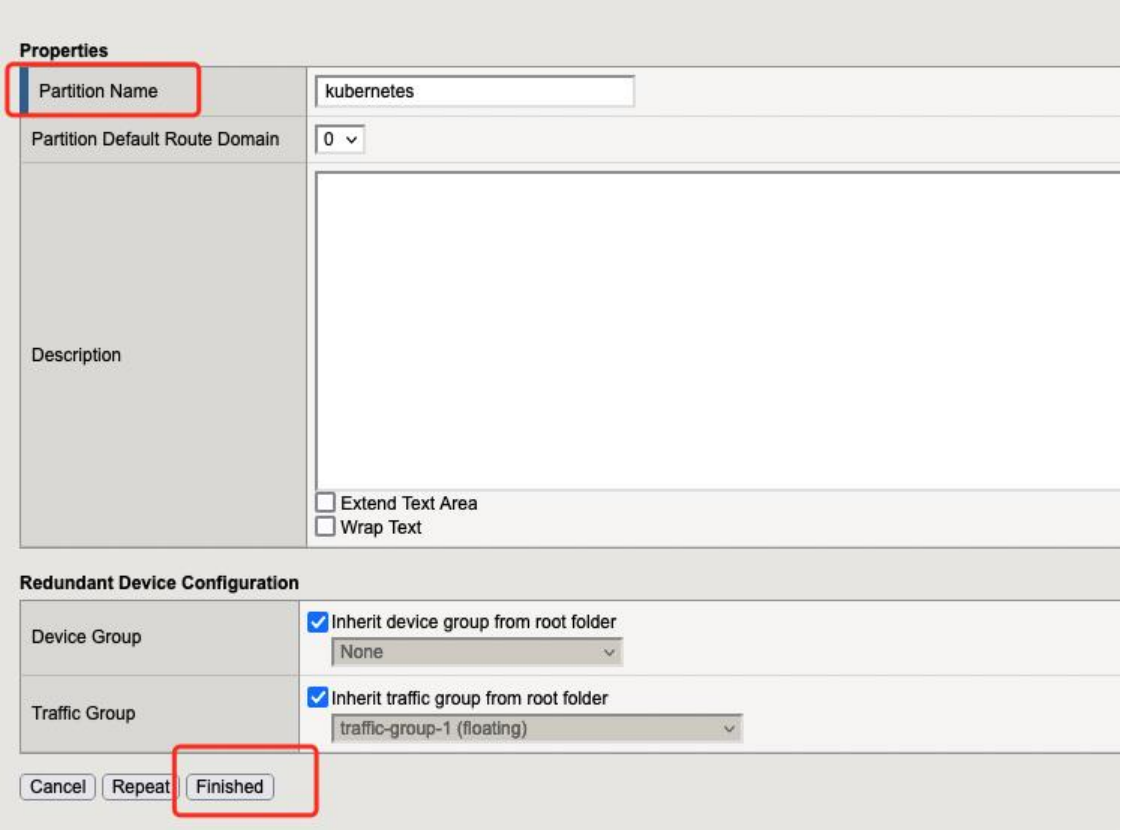
可在 F5 设备上创建一个独立的 partition，供 k8s bigip ctlr 组件对接存储转发规则。具体步骤如下：

1. 登录 F5 设备的管理 Web UI，进入 System –> Users –> Partition List 界面，点击右上角的 create。



f5 partition1

2. 在创建界面中，填写 Partion Name 即可，其它按照默认值不需要修改，点击 Finished 完成创建。



f5 partition2

集群安装 storage 存储组件（可选）

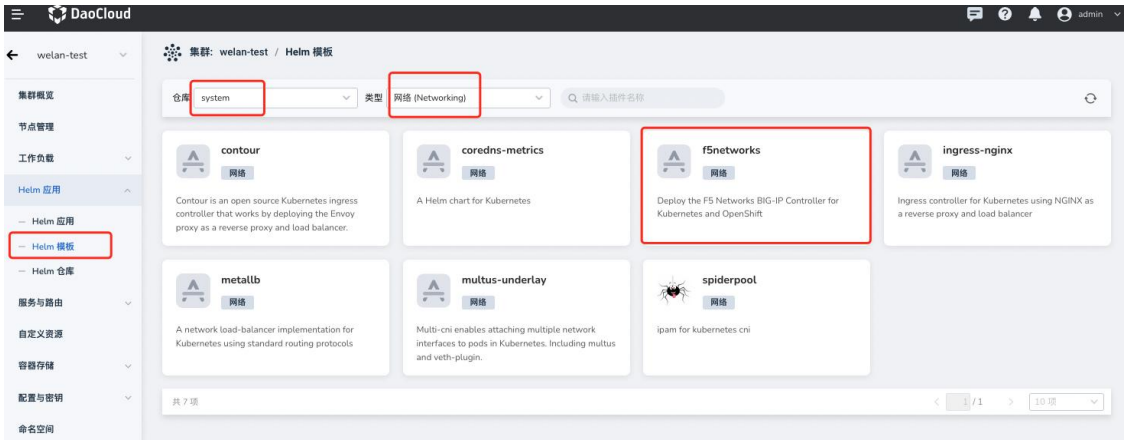
如果希望本组件安装在 4 层负载均衡模式下，要求安装 f5 ipam controller， 而 f5 ipam controller 要求集群具备 storage 组件提供 PVC 服务。可参考相关的存储组件安装手册。

如果希望本组件安装在 7 层负载均衡模式下，并不需要要求安装 f5 ipam controller，因此可忽略存储组件的安装。

集群安装组件步骤

1. 拥有一个 DCE 集群，登录 global 集群的 Web UI 管理界面，在导航的容器管理 -> 集群列表中，登录希望安装本组件的集群。

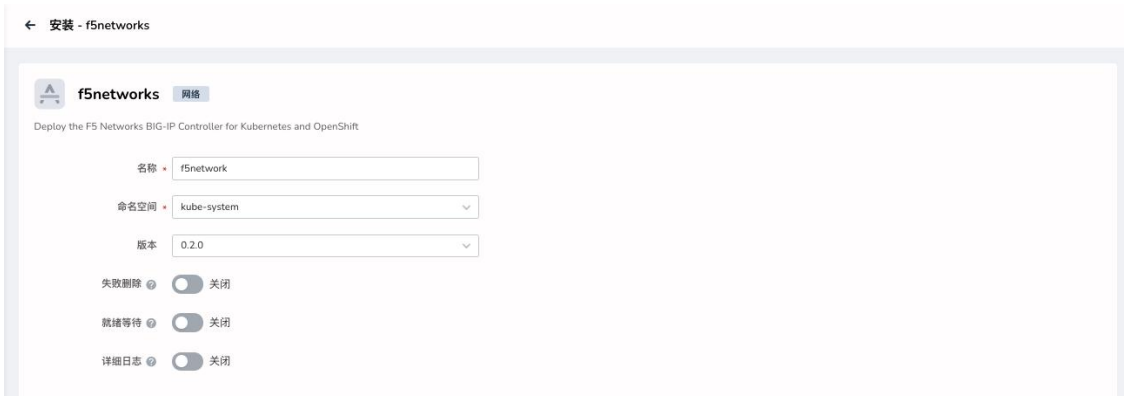
2. 在 Helm 应用 -> Helm 模板中，选择 system 仓库和网络组件，点击安装 f5network。



f5network helm

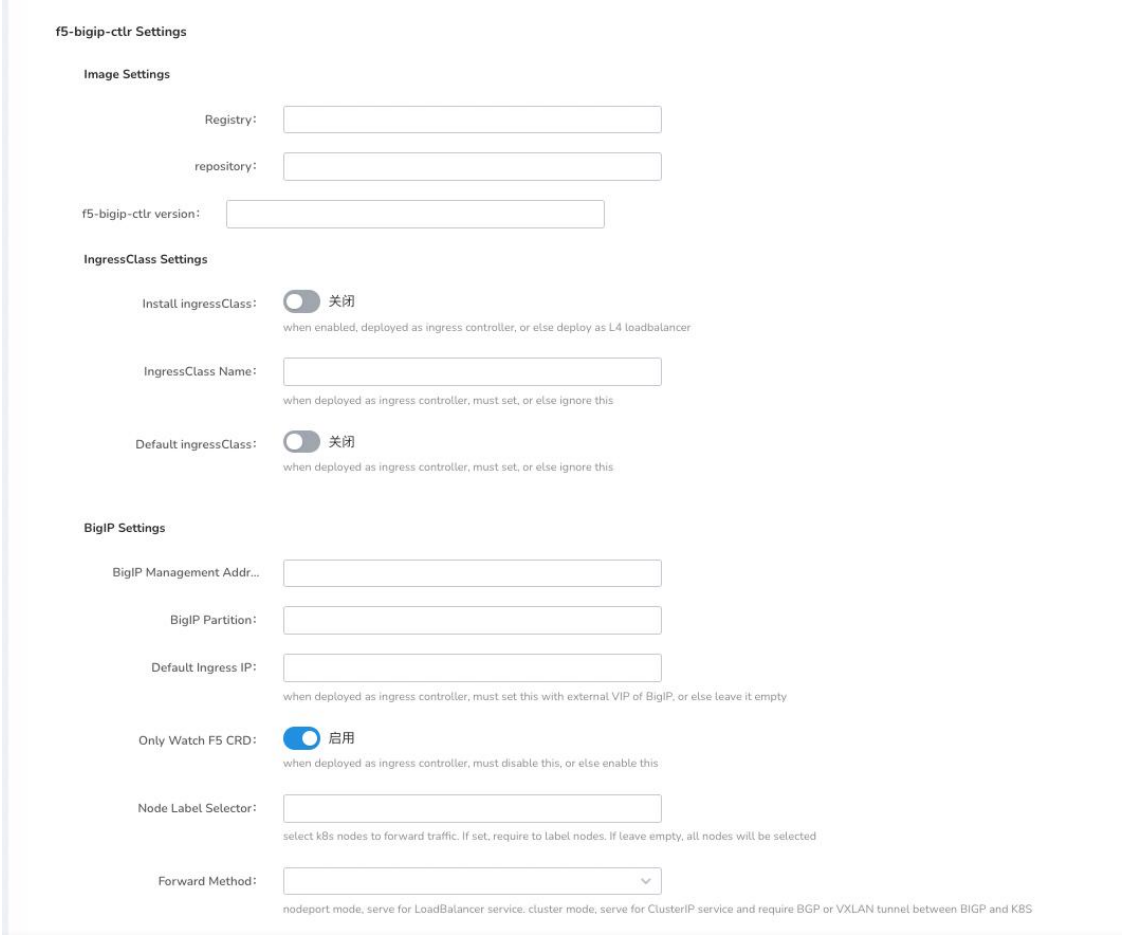
3. 在版本选择中选择希望安装的版本，点击安装。

4. 在安装参数界面，填写如下信息。



f5network install1

在如上界面中，填写安装名称、命名空间、版本。



f5network install2

如上界面的参数说明:

- f5-bigip-ctlr Settings -> Registry：设置 f5-bigip-ctlr 镜像的仓库地址，默认已经填写了可用的在线仓库，如果是私有化环境，可修改为私有仓库地址。
- f5-bigip-ctlr Settings -> repository：设置 f5-bigip-ctlr 镜像名。
- f5-bigip-ctlr version：设置 f5-bigip-ctlr 的镜像版本，默认值已经设置了最新的版本号。如果进行修改，注意安装需知中提到的版本适配问题。

- Install ingressClass：是否安装 ingressClass。如果希望使用 7 层转发模式，务必开启本选项，如果希望使用 4 层转发模式，请关闭本选项。

- IngressClass Name：设置 ingressClass 的名字。如果希望使用 7 层转发模式，务必设置本选项，如果希望使用 4 层转发模式，请忽略本选项。

- Default ingressClass：是否设置安装的 ingressClass 作为集群的默认 ingressClass。如果希望使用 7 层转发模式，务必开启本选项，如果希望使用 4 层转发模式，可忽略本选项。

- BigIP Management Addr：F5 设备的 WEB UI 登录地址。

- BigIP Partition：使用 F5 设备的上的 Partition 名字，即在 F5 设备创建 partition 步骤中创建的 Partition。

  若多个集群共用对接同一个 F5 设备，不同集群最好使用独立的 Partition。

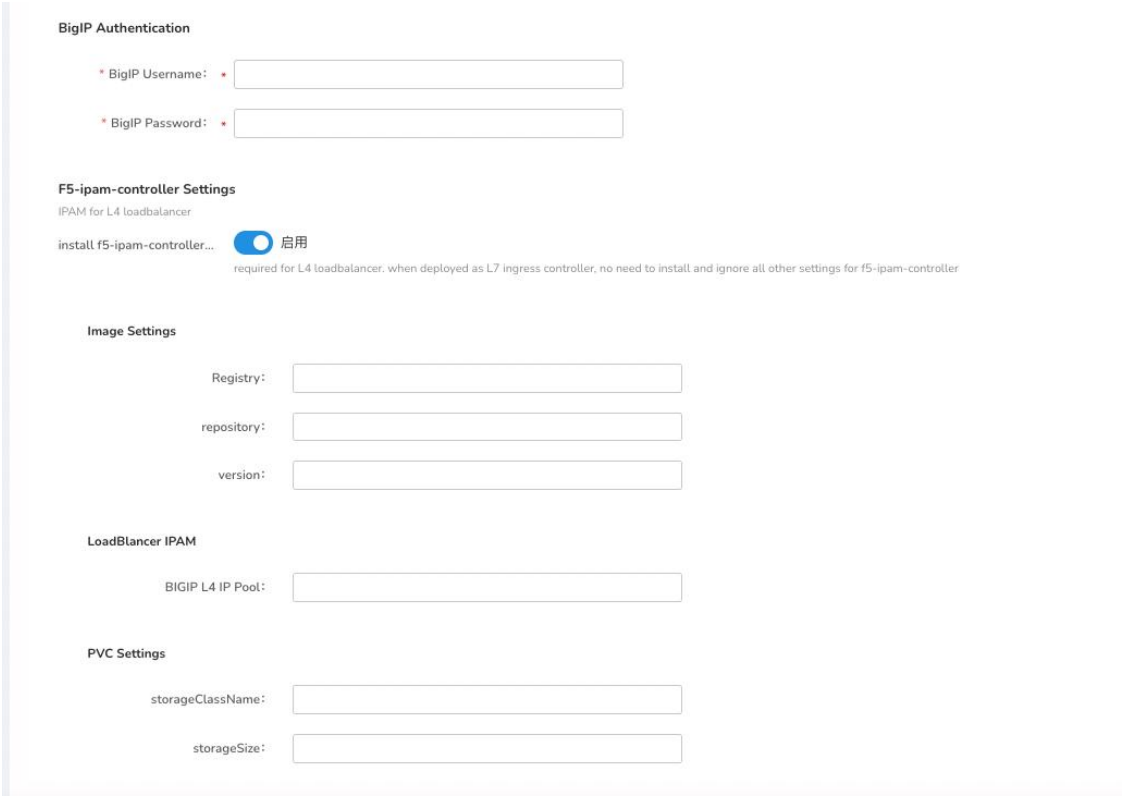
- Default Ingress IP：当本组件安装在 7 层负载均衡模式下，本值设置了 F5 上的 ingress 入口 VIP，注意该 IP 应该是 F5 external interface 子网的 IP 地址。 当本组件安装在 4 层负载均衡模式下，忽略本值。

  若多个集群共用对接同一个 F5 设备，不同集群要使用独立的 IP。

- Only Watch F5 CRD：当打开本选项，本组件只会监控自己的 CRD，适用于工作在 4 层负载均衡模式；否则，会监控全部的 K8S 资源，适用于工作在 7 层负载均衡模式。

- Node Label Selector：设置 node label selector，被选择的 node 会作为 nodePort 转发模式下的入口节点，如果不设置本值，F5 会把流量转发到集群所有节点的 nodePort 上。

- Forward Method：设置 F5 转发流量的模式、nodePort 和 cluster 模式。模式解释，可参考介绍。



f5network install2

在如上界面中:

- BigIP Username：F5 设备的登录账号。
- BigIP Password：F5 设备的登录密码。
- install f5-ipam-controller：是否安装 f5-ipam-controller 组件。当希望工作在 7 层负载均衡模式下，请关闭本开关，当本希望工作在 4 层负载均衡模式下，请打开本开关。
- F5-ipam-controller Settings -> Registry：设置 f5-ipam-controller 镜像的仓库地址，默认已经填写了可用的在线仓库，如果是私有化环

境，可修改为私有仓库地址。 当关闭了 install f5-ipam-controller，可忽略本选项。

- F5-ipam-controller Settings -> repository：设置 f5-ipam-controller 镜像名。 当关闭了 install f5-ipam-controller，可忽略本选项。

- F5-ipam-controller version：设置 F5-ipam-controller 的镜像版本，默认值已经设置了最新的版本号。 当关闭了 install f5-ipam-controller，可忽略本选项。

- BIGIP L4 IP Pool：设置 F5 的 4 层负载均衡的 VIP 地址池，这些 IP 应该是 F5 external interface 子网的 IP 地址。 本值的格式类似是 { LabelName: 172.16.1.1-172.16.1.5}， 其中 LabelName 将会用于创建应用 service 时的 annotation 上。

  !!! note

  创建应用 service 时，只要给 service 打上了 annotation `cis.f5.com/ipamLabel: LabelName`，就会被本组件分配 VIP，最终生效到 F5 设备上。
  当关闭了 `install f5-ipam-controller`，可忽略本选项。

  若多个集群共用对接同一个 F5 设备，不同集群要使用独立的 IP 池。

- storageClassName：设置 storageClass Name，f5-ipam-controller 组件将用于常见 PVC 对象。 当关闭了 install f5-ipam-controller，可忽略本选项。

- storageSize：设置存储池大小，f5-ipam-controller 组件将用于常见 PVC 对象。 当关闭了 install f5-ipam-controller，可忽略本选项。

cluster 转发模式下的配置（可选）

当组件安装为 cluster 转发模式时，需要在 F5 和 K8S 集群之间配置 VXLAN 隧道，或者配置 BGP 邻居。

K8S 的 CNI 组件往往可配置 BGP 及 F5 组件 BPG 邻居，实现网络打通。 例如在 Calico 场景下，具体可参考 F5 官方文档。

关于本组件的更多介绍，可参考 F5 官方文档。

什么是 F5network

F5network 组件整合了 F5 官方项目 f5 ipam controller 和 k8s bigip ctlr， 完成对 F5 设备的控制，将集群中 service 和 ingress 配置同步到 F5 硬件设备上，实现集群北向入口的负载均衡。

其中，k8s bigip ctlr 组件负责监控 service 或 ingress 对象，实现对 F5 硬件设备的控制面规则下发； 当工作在 4 层负载均衡模式下时，f5 ipam controller 组件主要负责 F5 硬件的入口 VIP 分配。

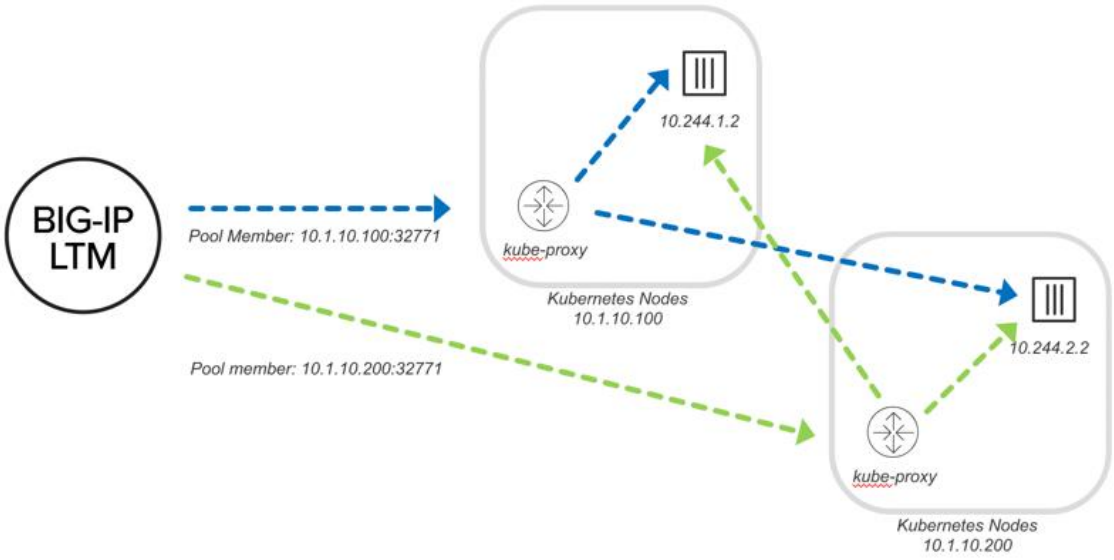
转发模式

F5 设备有两种模式来转发流量到集群（关于转发模式，更多信息可参考官方说明 ）。

1. NodePort 转发模式：F5 把流量转发到集群节点的 nodePort 上，该模式即可工作在“4 层转发”和“7 层转发”。

   优势：不需要在集群和 F5 设备之间做特殊处理，只要 F5 设备能够访问集群节点即可，通用性更强。

   要求：集群应用的 loadBalancer service 对象必须分配了 nodePort。

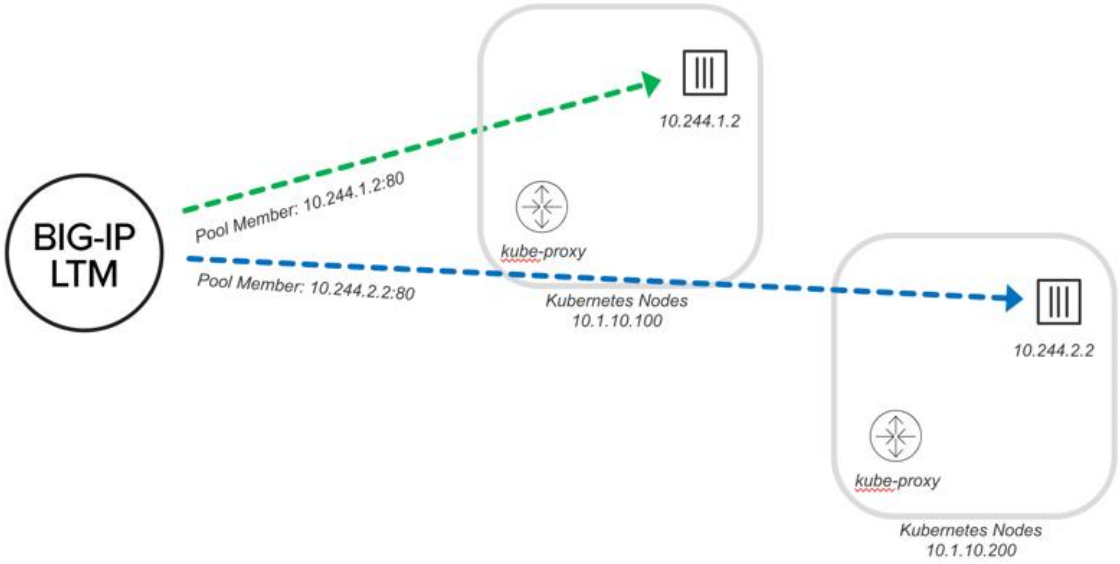


nodeport

2. Cluster 转发模式：F5 把流量直接转发到 Pod IP 上，该模式即可工作在“4 层转发”和“7 层转发”。

   优势：数据包转发不经过 kube proxy 的 NodePort，直接转发到 Pod 上，转发方式更加高效，延时更低。

   要求：集群通过 BPG 协议把 Pod 路由转发到网络中的路由器和 F5 设备上，或者集群中的节点同 F5 设备间建立 VXLAN 隧道。



cluster

功能说明

1. 4 层负载均衡

   该负载均衡模式下，配合“NodePort 转发模式”，可为 LoadBalancer 服务（须具备 nodePort）创建 F5 负载均衡； 也可配合“Cluster 转发模式”为 LoadBalancer 服务（可不分配 nodePort）创建 F5 负载均衡。其中，f5 ipam controller 组件维护一个可配置的 VIP 池，为每个 service 独立分配一个独享的 EXTERNAL IP。

   该模式下，务必安装 f5 ipam controller ，为每个 service 分配一个 VIP。

2. 7 层负载均衡

   该负载均衡模式下，工作为 ingress controller。可配合“NodePort 转发模式”，要求 ingress 匹配的 service 是 nodePort 类型； 可配合“Cluster 转发模式”，ingress 匹配的 service 是 clusterIP 类型即可。

   该模式下，不需要安装 f5 ipam controller，所有 ingress 共享一个 VIP。

!!! note

本组件不能同时工作在“4 层负载均衡”和“7 层负载均衡”下，只能二选一。

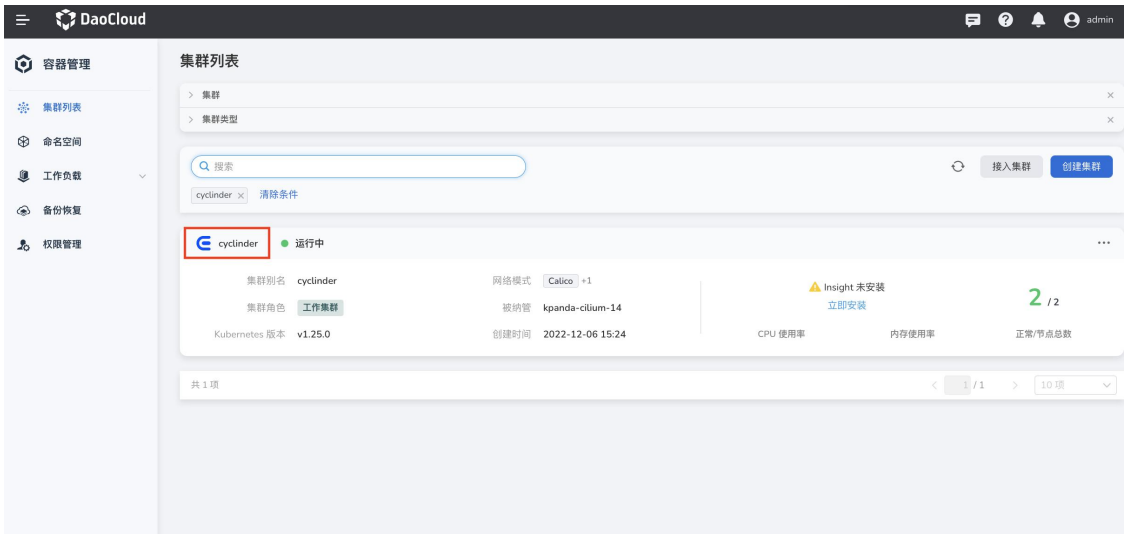
更多信息，可参考 [F5 官方文档](https://clouddocs.f5.com/containers/latest/userguide/)。

安装 Contour

本页介绍如何安装 Contour。

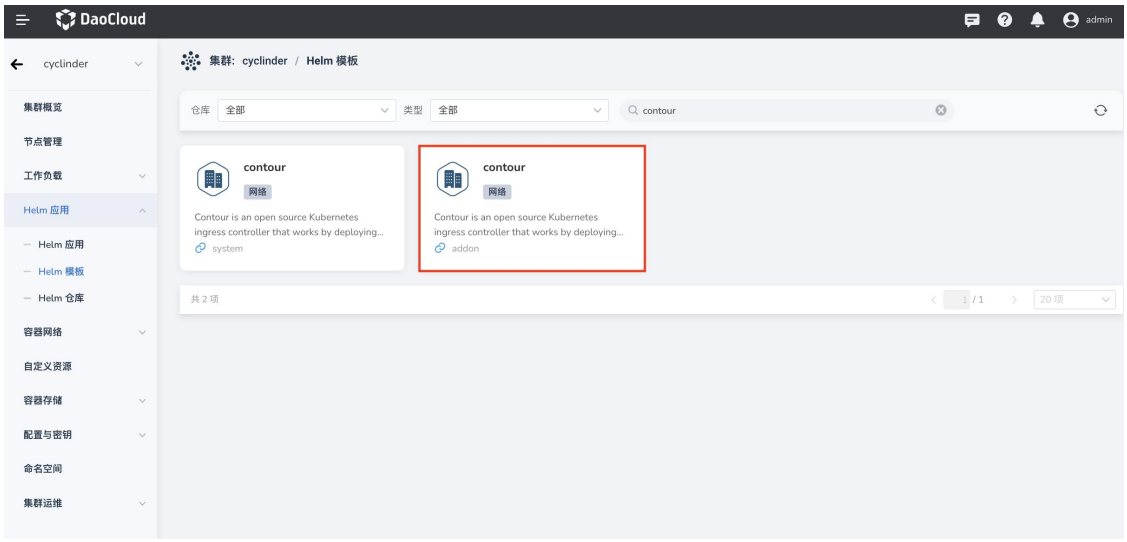
请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Contour。

1. 在左侧导航栏点击容器管理—>集群列表，然后找到准备安装 Contour 的集群名称。



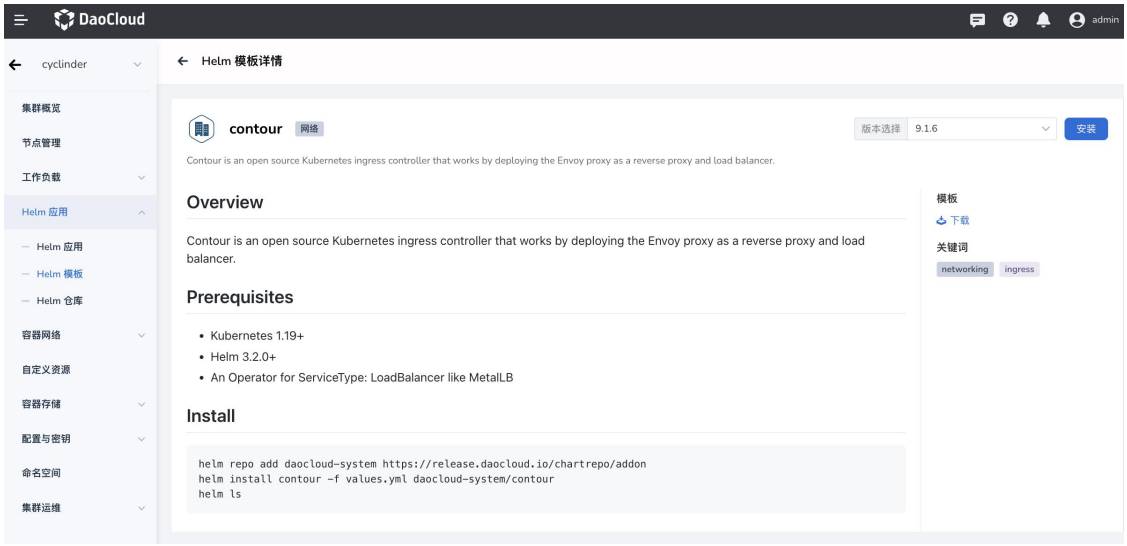
contour-cluster

2. 在左侧导航栏中选择 Helm 应用 -> Helm 模板，找到并点击 contour。



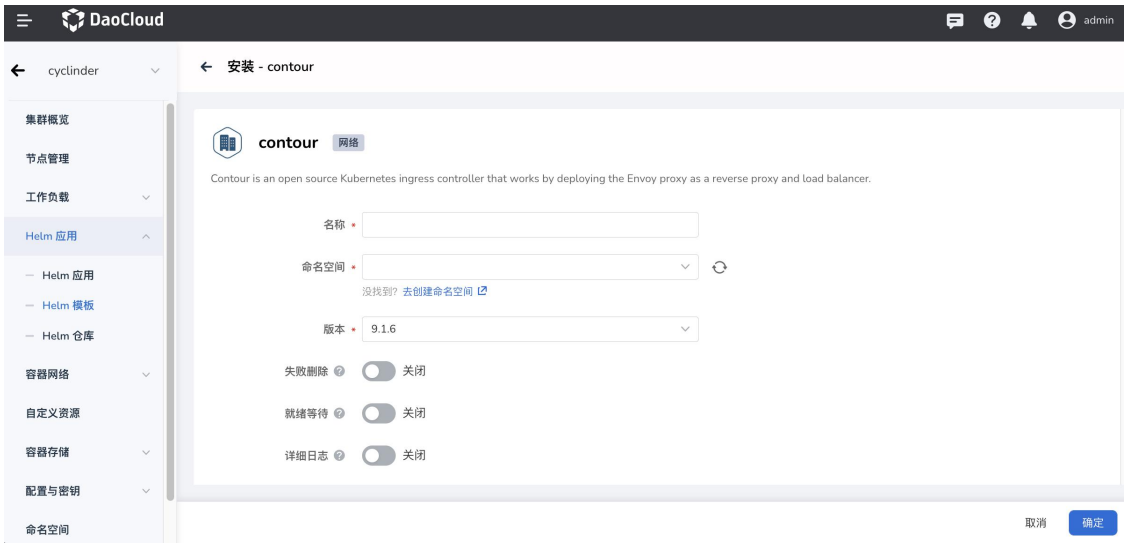
contour-helm

3. 在版本选择中选择希望安装的版本，点击安装。



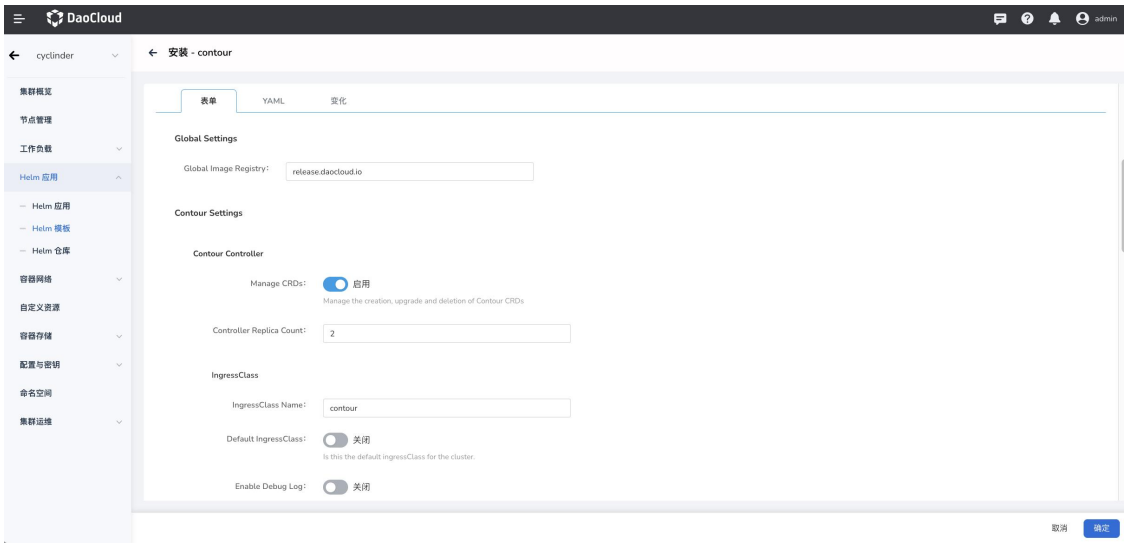
contour-version

4. 在安装界面，填写所需的安装参数。



contour-install-1

在如上界面中，填写应用名称、命名空间、版本等。

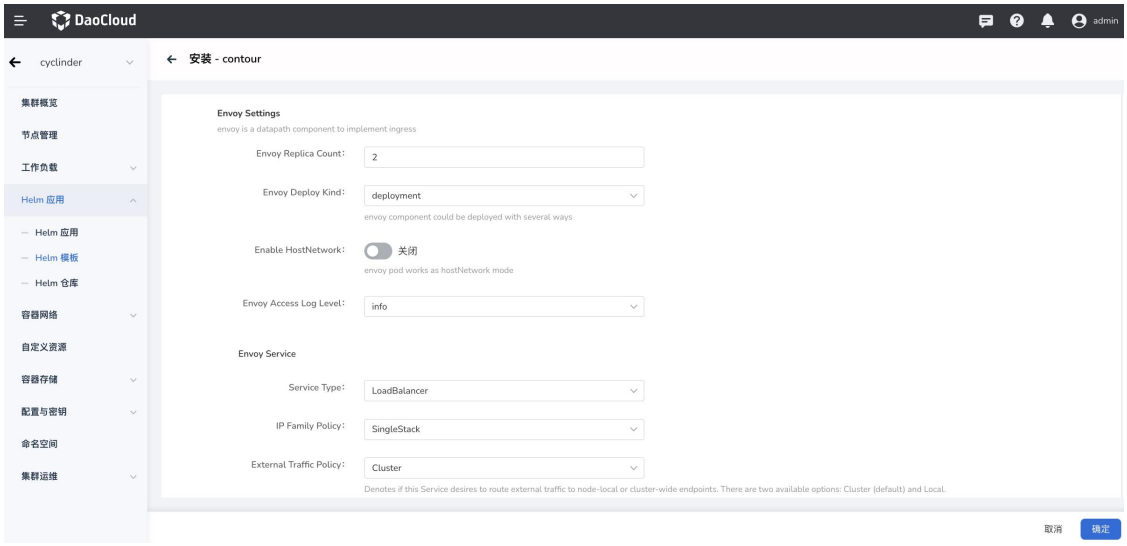


contour-install-2

在如上界面中，填写以下参数：

- contour —> Global Settings —> Global Image Registry：统一设置镜像仓库地址。
- contour —> Contour Settings —> Contour Controller —> Manage CRDs：创建 Contour 的 CRD。

- contour —> Contour Settings —> Contour Controller —> Controller Replica Count：配置 Contour 控制面的副本数量。
- contour —> Contour Settings —> IngressClass —> IngressClass Name：配置 Ingress Class 名称。如果集群部署多套 Ingress 的时候，可以使用此 Class 进行区分，这个字段会在创建 Ingres CR 的时候设置。
- contour —> Contour Settings —> IngressClass —> Default IngressClass：设置默认 Ingress。
- contour —> Contour Settings —> IngressClass —> Enable Debug Log：设置控制面 Debug 级别日志输出。

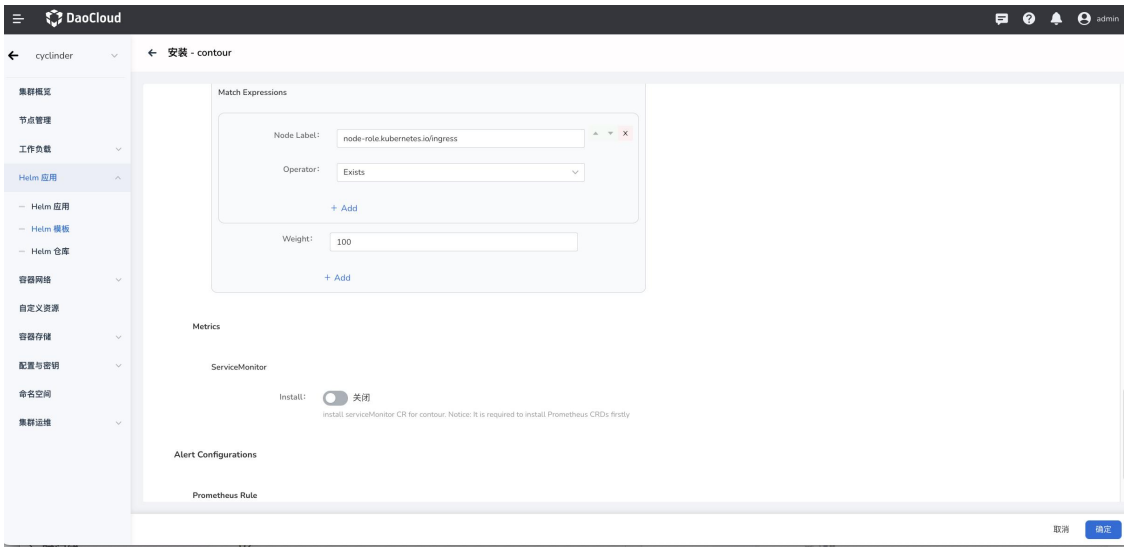


contour-install-3

在如上界面中，填写以下参数：

- contour —> Contour Settings —> Envoy Settings —> Envoy Replica Count：配置数据面 Envoy 的副本数量。
- contour —> Contour Settings —> Envoy Settings —> Envoy Deploy Kind：配置 Envoy 的部署类型，可以选择部署为 Deployment 或者是 DaemonSet。

- contour —> Contour Settings —> Envoy Settings —> Enable HostNetwork：使用 Host 网络，默认关闭。如没有特殊需求不推荐开启此选项。
- contour —> Contour Settings —> Envoy Settings —> Envoy Access Log Level：配置 Envoy 访问日志等级。
- contour —> Contour Settings —> Envoy Settings —> Envoy Service —> Service Type：配置 Service 类型。
- contour —> Contour Settings —> Envoy Settings —> Envoy Service —> IP Family Policy：IP 单双栈设置，可以根据需求开启。



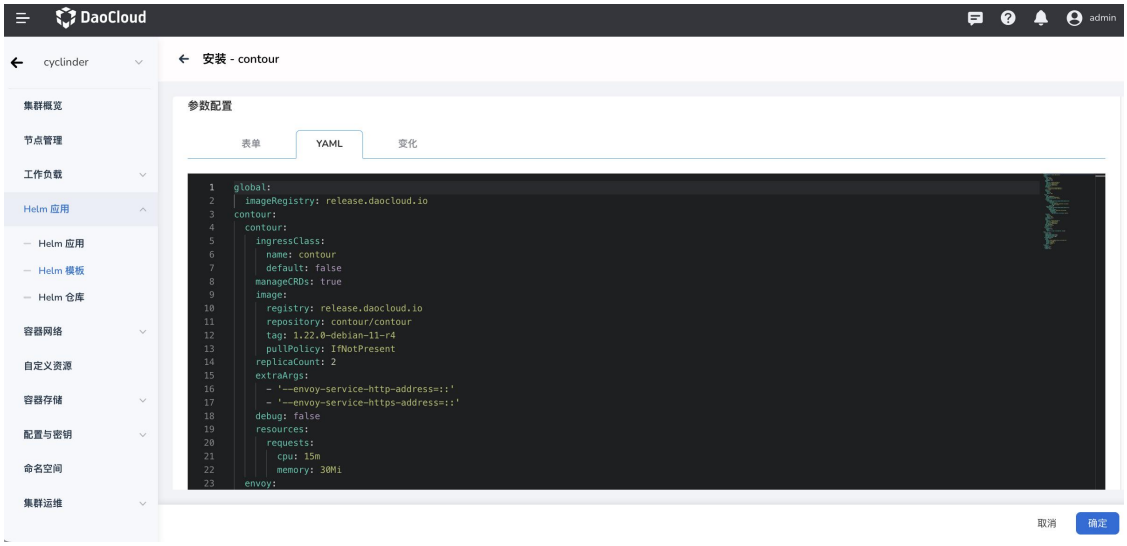
contour-install-4

在如上界面中，填写以下参数：

- contour —> Contour Settings —> Envoy Settings —> Envoy Node Affinity —> Match Expressions：通过软亲和性指定调度规则。
- contour —> Contour Settings —> Envoy Settings —> Envoy Node Affinity —> Match Expressions —> Weight：软亲和调度规则的权重。
- contour —> Contour Settings —> Metrics —> ServiceMonitor：集群需要已部署 Prometheus Operator。

- contour —> Contour Settings —> Alert Configurations —> Prometheus Rule：若开启，则创建包含告警规则 PrometheusRule CR。要求集群已安装 Prometheus Operator，或者部署 Insight 组件。

5. 点击 Tab 选项卡中 YAML 通过 YAML 方式进行高级配置。然后，点击右下角确定按钮即可完成创建。



contour-yaml

创建完成后，创建 Ingress 路由，可通过 平台级负载均衡 选择 Contour 实例。详情参考： 创建路由

什么是 Contour

Contour 是一个开源的 Kubernetes Ingress 控制器，使用 Envoy 作为数据面。Contour 支持动态配置更新和开箱即用的多团队入口代理，同时保持轻量级配置文件。

Contour vs Nginx Ingress

- 数据面与控制面分离
- 管理方便
- 性能优势
- 更易于使用的声明式语法
- 更全面的监控指标
- Gateway API 的支持

通过 Metallb + istio-ingressgateway 获取客户端源 IP

背景

通过 Metallb ARP 模式，用户可以在全局管理—>审计日志中查看操作者的真实 IP，而不是被 SNAT 后的 IP 地址。主要的关键步骤是设置 Service 的 spec.externalTrafficPolicy 为 Local 模式。

此方式同样适配 Istio 高可用模式下，获取客户端源 IP 。

操作步骤

1. 配置 Metallb 宣告上述节点作为 LB IPs 的下一跳。

```
[root@demo-dev-master-01 ~]# kubectl get l2advertisements.metallb.io -n
 metallb-system default-l2advertisement -o yaml
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  annotations:
    helm.sh/hook: post-install
    helm.sh/resource-policy: keep
  creationTimestamp: "2022-11-14T06:04:35Z"
  generation: 2
  labels:
    app.kubernetes.io/instance: metallb
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: metallb
    app.kubernetes.io/version: 0.13.5
    helm.sh/chart: metallb-0.13.5
  name: default-l2advertisement
  namespace: metallb-system
  resourceVersion: "133681854"
```

```
  uid: c5301f5b-fb08-40ae-8a22-2b03e129a092
spec:
  ipAddressPoolSelectors:
  - matchLabels:
      l2.ipaddress-pool.metallb.io: default-pool
  ipAddressPools:
  - default-pool
```

通过配置 spec.nodeSelectors 来实现绑定。

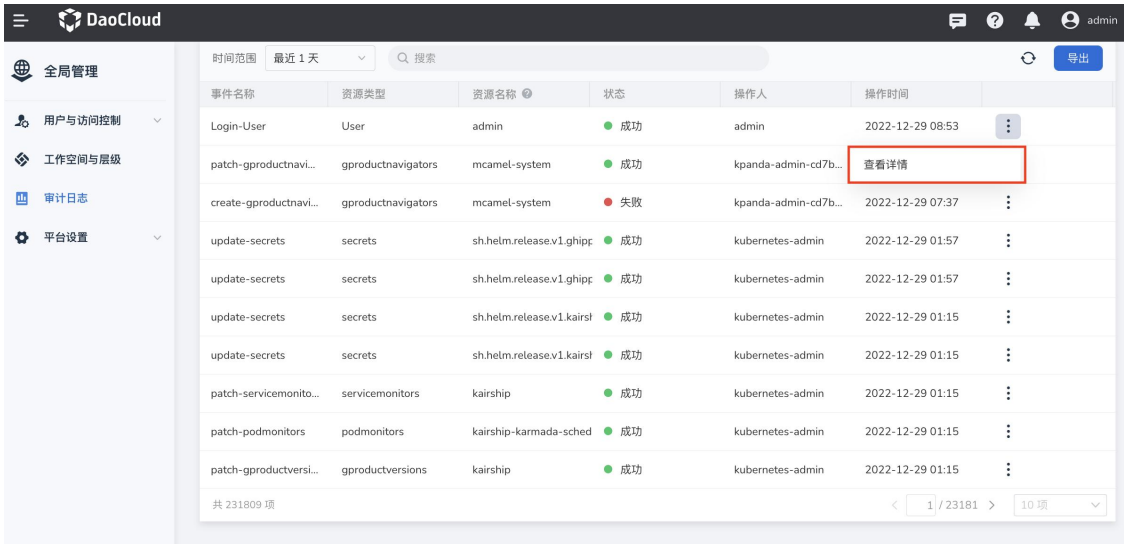
2. 修改名为 istio-ingressgateway 的 Service 中的字段
   spec.externalTrafficPolicy = Local，此模式可以保留真实源 IP:

```
[root@demo-dev-master-01 ~]# kubectl get svc -n istio-system istio-ingr
essgateway -o yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    meta.helm.sh/release-name: istio-ingressgateway
    meta.helm.sh/release-namespace: istio-system
  creationTimestamp: "2022-11-25T08:27:35Z"
  labels:
    app: istio-ingressgateway
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: istio-ingressgateway
    app.kubernetes.io/version: 1.15.0
    helm.sh/chart: gateway-1.15.0
    istio: ingressgateway
  name: istio-ingressgateway
  namespace: istio-system
  resourceVersion: "198389187"
  uid: 9308a4fa-88b2-48ff-9ccf-a9fa4d6c6bcf
```

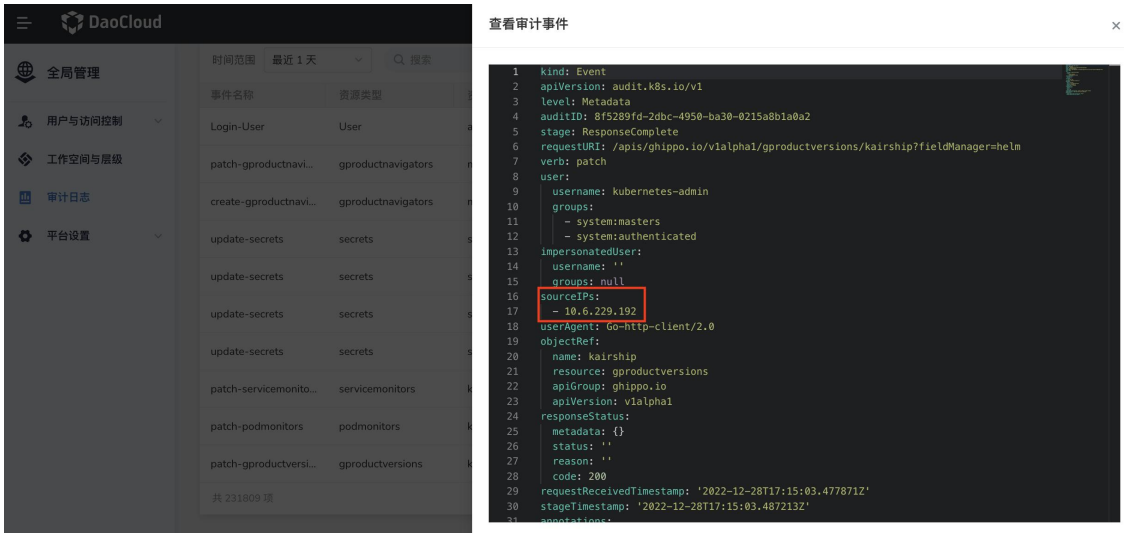
```
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.233.32.214
  clusterIPs:
  - 10.233.32.214
  externalTrafficPolicy: Local
  healthCheckNodePort: 32109
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: status-port
    nodePort: 32082
    port: 15021
    protocol: TCP
    targetPort: 15021
  - name: http2
    nodePort: 30421
    port: 80
    protocol: TCP
    targetPort: 8080
  - name: https
    nodePort: 30483
    port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  sessionAffinity: None
  type: LoadBalancer
```

```
status:
  loadBalancer:
    ingress:
    - ip: 10.6.229.180
```

3. 在全局管理—>审计日志页面，在任意事件后点击查看详情，查看获取到的客户端源 IP：



source–ip–1



source–ip–2

IPPool 使用说明

集群默认地址池

如果创建 LoadBalancer Service 的时候没有通过 Annotations: metallb.universe.tf/address-pool 指定地址池或通过 Annotation: metallb.universe.tf/loadBalancerIPs 指定 IP，那么会从现存的地址池中设置 autoAssign=true 的池中分配地址。

!!! note

创建的地址池必须跟 `Metallb` 组件在同一个 namespace，否则 `Metallb` 无法识别。

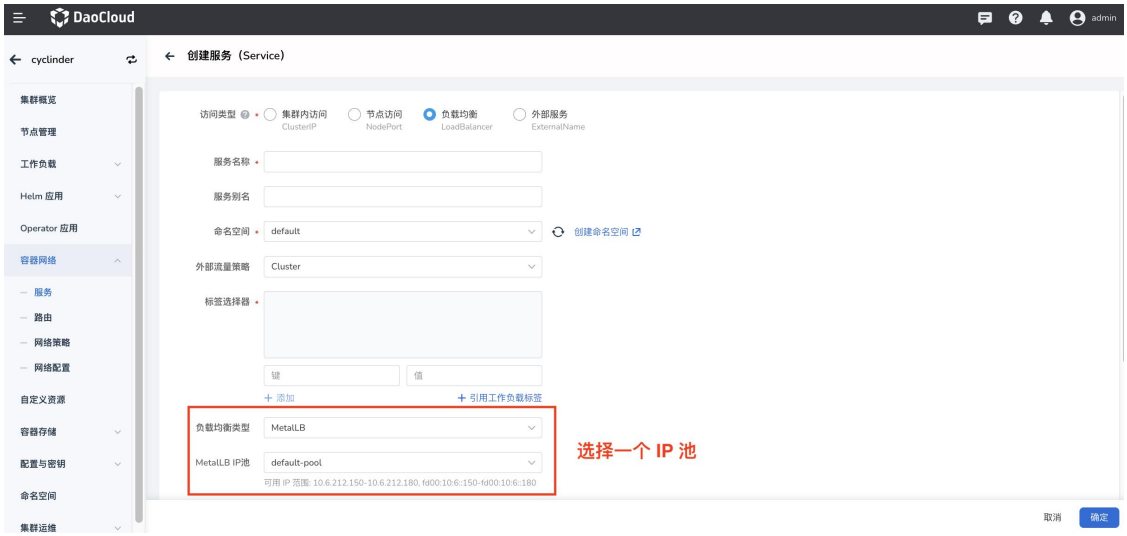
指定地址池

创建 LoadBalancer Service 的时候可以通过 Annotations: metallb.universe.tf/address-pool 指定地址池：

```
apiVersion: v1
kind: Service
metadata:
  name: metallb-ippool3
  labels:
    name: metallb-ippool3
  annotations:
    metallb.universe.tf/address-pool: default   # default 必须和 metallb 组件在同一个 namespace 下
spec:
  type: LoadBalancer
  ...
```

创建 LoadBalancer Service ,详情请参考：创建服务(Service)

MetalLB IP 池：部署 MetalLB 时设置的 IP 池，其中的 IP 为真实可用的物理 IP 地址。 负载均衡 IP：负载均衡 IP 为 IP 池中随机分配或手动指定 地址，用于对外暴露服务。

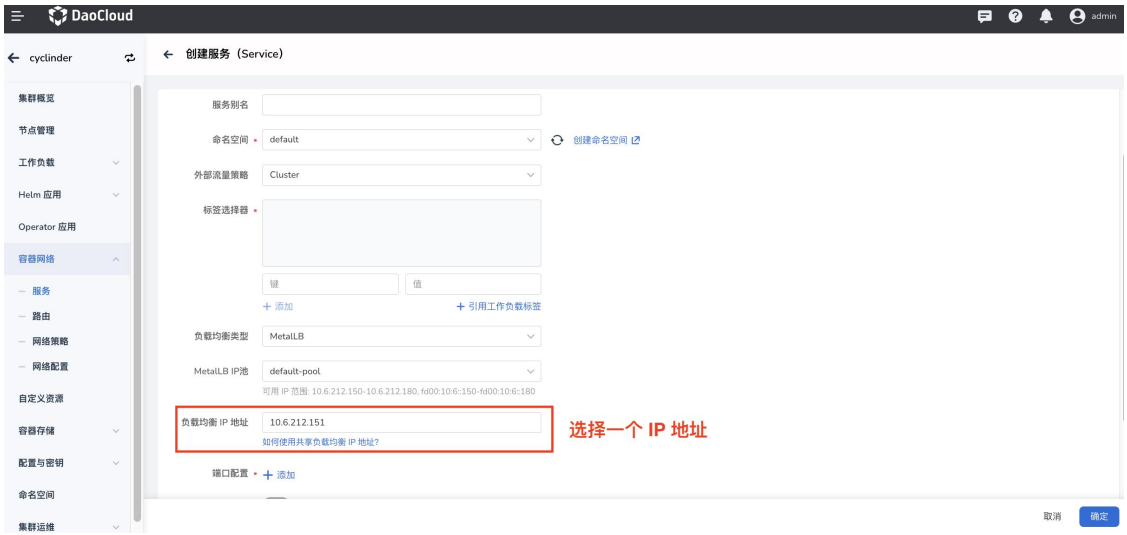


metallb ip pool

指定 IP 地址

创建 LoadBalancer Service 的时候可以通过 Annotations: metallb.universe.tf/loadBalancerIPs 指定 IP：

```
apiVersion: v1
kind: Service
metadata:
  name: metallb-ippool3
  labels:
    name: metallb-ippool3
  annotations:
    metallb.universe.tf/loadBalancerIPs: 172.16.13.210   # 该 IP 地址必须存在于现存的地址池中
spec:
  type: LoadBalancer
  ...
```



specify ip

共享 IP 地址

在 k8s v1.20 之前，LoadBalancer Service 不支持配置多种协议（v1.24 支持，已成为 Beta 功能），参考 #issue 23880。

Metallb 通过创建不同的 Service 并共享 Service IP，间接的支持这个功能。

创建两个 Service：

```yaml
apiVersion: v1
kind: Service
metadata:
  name: dns-service-tcp
  namespace: default
  annotations:
    metallb.universe.tf/allow-shared-ip: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  loadBalancerIP: 1.2.3.4
  ports:
    - name: dnstcp
      protocol: TCP
```

```
    port: 53
    targetPort: 53
  selector:
    app: dns
---
apiVersion: v1
kind: Service
metadata:
  name: dns-service-udp
  namespace: default
  annotations:
    metallb.universe.tf/allow-shared-ip: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  loadBalancerIP: 1.2.3.4
  ports:
    - name: dnsudp
      protocol: UDP
      port: 53
      targetPort: 53
  selector:
    app: dns
```

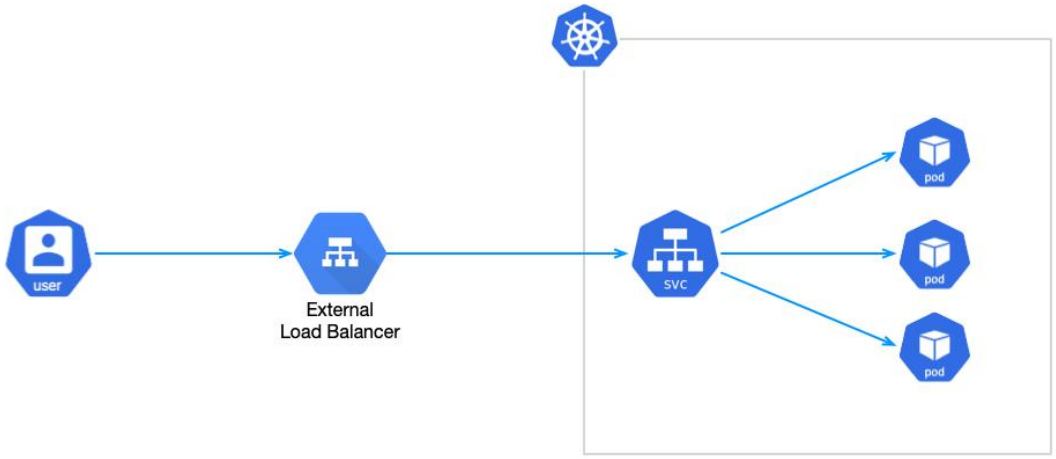
!!! note

只要 Annotations（`metallb.universe.tf/allow-shared-ip`）key 和 value 相同，那么不同的 `LoadBalancer Service`就会具有相同的 IP 地址（ipv4/ipv6）。
当然也可以通过 Annotations（`metallb.universe.tf/loadBalancerIPs`）指定 ipv4/ipv6 地址，或者通过 `.spec.loadBalancerIP` 指定（只支持 ipv4）。
创建后编辑 Annotation 没有效果。

共享 IP 另一个作用是 LoadBalancer IP 地址不足，需要多个 Service 共享同一个 IP，但注意不同 Service 的协议和端口应该是不同的，否则无法区分连接。

Metallb

在 Kubernetes 中，对于 LoadBancer 类型，需要使用云提供商的负载均衡器向外部暴露服务，外部负载均衡器可以将流量路由到自动创建的 NodePort 服务和 ClusterIP 服务上。 所以对于 LoadBancer 类型的 Service，必须要有 Cloud Provider 加持才能实现。 也就是说在裸机的 K8s 集群无法使用 LoadBancer 类型的服务。否则，您会发现 LoadBancer 的服务一直处于 Pending 状态。



metallb

Metallb 是一款开源软件，它采用标准的路由协议（ARP 或 BGP）实现了裸机 K8s 集群的负载均衡功能。

L2 模式 (ARP)

L2 模式下，Metallb 会通过 memberlist 选举出一个 Leader 节点，此节点负责向本地网络宣告 LoadBalancerIP。 从网络的角度来看，这台机器似乎有多个 IP 地址，它会响应来自 LoadBancerIP 的 ARP 请求。 L2 模式最大的优势是它不需要依赖譬如路由器等硬件的依赖便可工作。

- 优势：通用型，不需要额外的硬件支持

- 缺点：单节点的带宽限制、稍缓慢的故障转移（10s 左右）

L3 模式（BGP）

在 BGP 模式下，集群中的每个节点都会与路由器建立 BGP Peer，并使用该会话向集群外部通告集群服务的 LoadBalancerIP。 BGP Router 基于每个不同的连接选择一个下一跳（即集群某个节点，这不同于 L2 模式下所有流量先到达某个 Leader 节点）。

- 优势：负载均衡性更好
- 缺点：
  - 当某个节点故障，所有 BGP 会话将会中断
  - Calico BGP 模式无法和 MetaLB L3 模式并存，会存在冲突，详情请参考：https://metallb.universe.tf/configuration/calico/

L2&BGP 模式说明

MetalLB L2 模式

L2 模式下，Metallb 将会通过 ARP（for ipv4）、NDP（for ipv6）宣告 LoadBalancerIP 的地址。 在 Metallb < v0.13.2 之前，只能通过 configMap 来配置 Metallb。 在 v0.13.2 之后，通过 CRD 资源的方式配置 Metallb，另外 configMap 的方式已被弃用。

在 Layer2 模式下，当创建服务时，Metallb（speaker 组件）会为这个服务选举出集群中某个节点，作为这个服务对外暴露的主机。 当对 Service 的 externalIP 发出请求时，此节点会代替这个 externalIP 回复 arp 请求。 所以对 Service 发出的请求，会首先到达集群中这个节点，然后再经过这个节点上的 kube-proxy 组件，最后将流量导向这个 service 某个具体的端点（endpoint）。

服务选举节点的逻辑主要有三点：

1. 首先过滤掉未 ready 的节点以及 endpoint 未 ready 所在的节点
2. 如果该服务的 endpoint 分布在同一个节点，那么筛选此节点作为该服务 IP 的 arp 响应者
3. 如果该服务的 endpoint 分布在不同的节点，那么通过 sha256 计算 节点 + # + externalIP 后，按照字典顺序取第一个

这样，MetalLB 就会为每个 Service 选择一个节点作为暴露的主机。 metallb 会将这单个 Service 的流量，全部导向某个节点，所以这个节点可能会成为限制性能的瓶颈。 Service 的带宽限制也会取决于单个节点的带宽，这也是使用 ARP 或 NDP 最主要的限制。

此外，当此节点发生故障时，MetalLB 需要为服务重新选择一个新的节点。 然后 Metallb 会给客户端发送一个”免费”的 arp，告知客户端需要更新他们的 Mac 地

址缓存。 在客户端更新缓存前，流量仍会转发到故障节点。因此从某种程度来看：故障转移的时间，依赖于客户端更新 Mac 地址缓存的速度。

使用

- 创建 IP 池

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: demo-pool
  namespace: metallb-system
  labels:
    ipaddresspool: demo
spec:
  addresses:
  - 192.168.10.0/24
  - 192.168.9.1-192.168.9.5
  - fc00:f853:0ccd:e799::/124
  autoAssign: true
  avoidBuggyIPs: false
```

- addresses：IP 地址列表，每一个列表成员可以是一个 CIDR，可以是一个地址范围（如 192.168.9.1 – 192.168.9.5），也可以是不同 ipFamily、Metallb 会从其中分配 IP 给 LoadBalancer 服务
- autoAssign：是否自动分配 IP 地址，默认为 true。在某些情况下（IP 地址不足或公有 IP），不希望池中的 IP 被轻易地分配，可设置为 false。可以通过在 service 中设置 annotations: metallb.universe.tf/address-pool: pool-name。或者在 spec.LoadBalancerIP 字段设置 IP（注意这种方式已被 k8s 标记为遗弃）。
- avoidBuggyIPs：是否避免使用池中以 .0 或 .255 地址，默认为 false。
- 配置 LoadBalancerIP 通告规则（L2）

通过 L2Advertisement 绑定地址池，这样告诉 Metallb 这些地址应该由 ARP 或 NDP 通告出去。

```yaml
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: demo
  namespace: metallb-system
spec:
  ipAddressPools:
  - demo-pool
  ipAddressPoolSelectors:
  - matchLabels:
      ipaddresspool: demo
  nodeSelectors:
  - matchLabels:
      kubernetes.io/hostname: kind-control-plane
```

- ipAddressPools：可选，通过 name 筛选地址池，如 ipAddressPools 和 ipAddressPoolSelectors 同时未指定，则作用于所有地址池。
- ipAddressPoolSelectors：可选，通过 labels 筛选地址池，如 ipAddressPools 和 ipAddressPoolSelectors 同时未指定，则作用于所有地址池。
- nodeSelectors：可选，用于筛选哪些节点作为 loadBalancerIP 的下一跳，默认所有节点。
- 创建 LoadBalancer Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: metallb1-cluster
```

```
  labels:
    name: metallb
      #annotations:
      #metallb.universe.tf/address-pool: lan
spec:
  type: LoadBalancer
  allocateLoadBalancerNodePorts: false
  ports:
  - port: 18081
    targetPort: 8080
    protocol: TCP
  selector:
    app: metallb-cluster
```

只需要指定 spec.type=LoadBalancer，这样 Metallb 就会自然接管此 Service 的生命周期。

!!! note

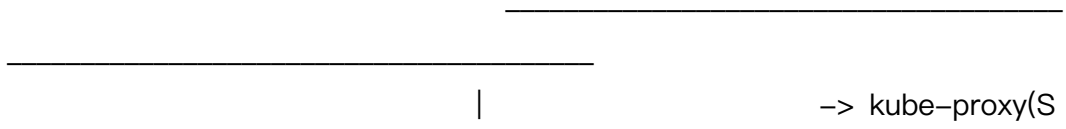
如果想让 Service 从指定的地址池中分配地址，通过 `annotations: metallb.universe.tf/address-pool: <pool-name>` 指定。或者通过 `service.spec.loadBalancerIP` 字段指定 IP（需要保证存在于一个池中，不推荐这种方式）。

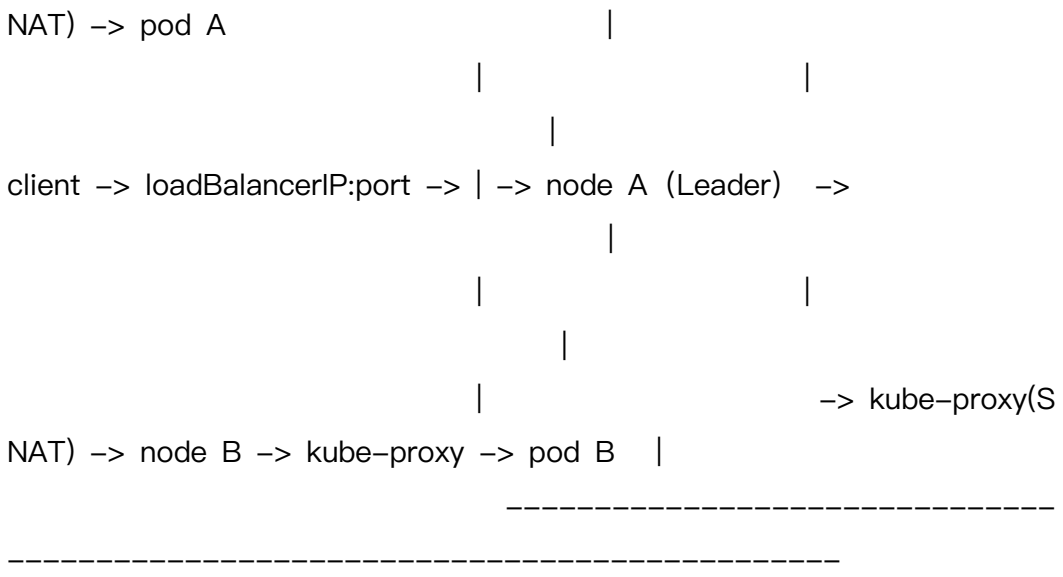
如果存在多种负载均衡器，可通过 `service.spec.loadBalancerClass` 字段指定。在部署 `Metallb` 时，可通过 `--lb-class` flag 进行配置。

负载均衡性

- 当 Service.spec.externalTrafficPolicy=cluster

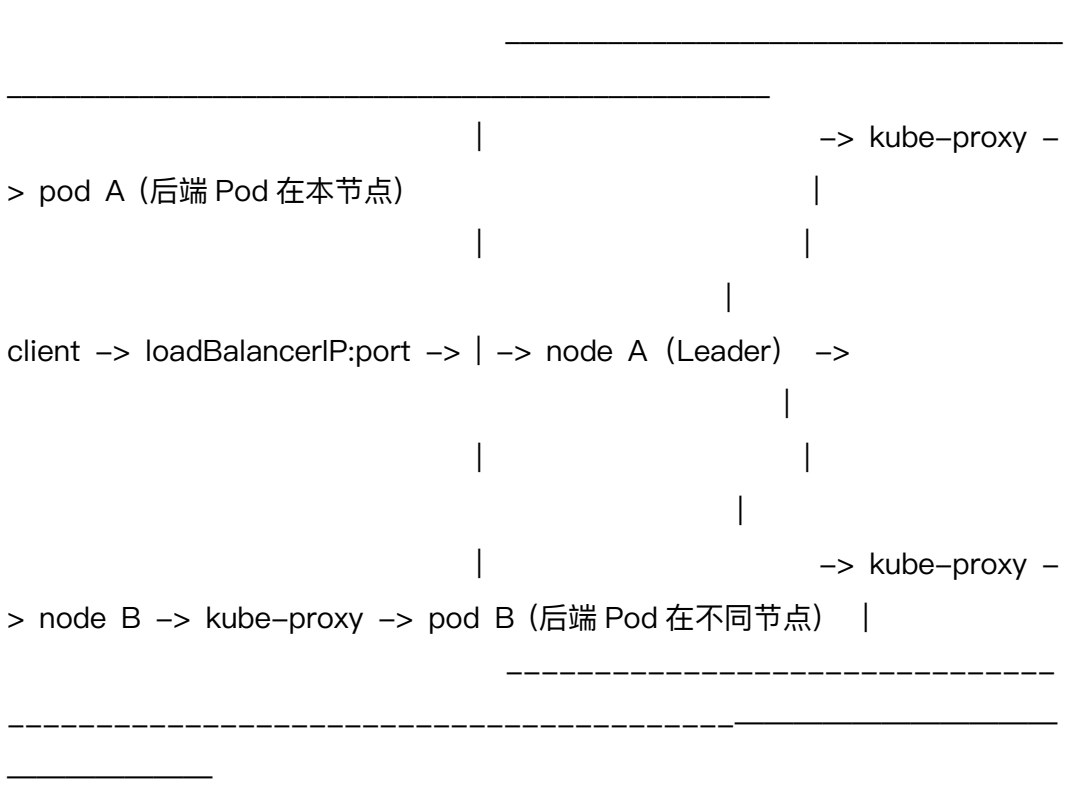
这种模式下，具有良好的负载均衡性，但流量可能经历多跳，这会隐藏客户端源 IP。





- 当 Service.spec.externalTrafficPolicy=local

  这种模式下，会保留客户端源 IP，但负载均衡性较差，流量会一直到某一个后端 Pod。



MetalLB BGP 模式(L3)

Layer2 模式局限在一个二层网络中，流向 Service 的流量都会先转发到某一个特定的节点，这并不算真正意义上的负载均衡。 BGP 模式不局限于一个二层网络，集群中每个节点都会跟 BGP Router 建立 BGP 会话，宣告 Service 的 ExternalIP 的下一跳为集群节点本身。 这样外部流量就可以通过 BGP Router 接入到集群内部，BGP Router 每次接收到目的是 LoadBalancer IP 地址的新流量时，它都会创建一个到节点的新连接。 但选择哪一个节点，每个路由器厂商都有一个特定的算法来实现。所以从这个角度来看，这具有良好的负载均衡性。

使用

- 创建 IP 池

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: bgp-pool
  namespace: metallb-system
  labels:
    ipaddresspool: demo
spec:
  addresses:
  - 192.168.10.0/24
  autoAssign: true
  avoidBuggyIPs: false
```

- 配置 LoadBalancerIP 通告规则（L3）

  !!! note

  BGP 模式需要硬件支持运行 BGP 协议。若无，可使用如 `frr`、`bird` 等软件代替。

  推荐使用 frr 进行安装:

```
# ubuntu
apt install frr
# centos
yum install frr
```

frr 配置 BGP：

```
router bgp 7675  # bgp as number
bgp router-id 172.16.1.1  # route-id 常常是接口 IP
no bgp ebgp-requires-policy # 关闭 ebpf filter !!!
neighbor 172.16.1.11 remote-as 7776  # 配置 ebgp -> neighbor 1, 172.16.1.11 为集群一节点
neighbor 172.16.1.11 description master1 # description
neighbor 172.16.2.21 remote-as 7776  # 节点 2
neighbor 172.16.2.21 description woker1
```

Metallb 配置：

- 配置 BGPAdvertisement

  此 CRD 主要用于指定需要通过 BGP 宣告的地址池，同 L2 模式，可通过池名称或者 labelSelector 筛选。同时可配置 BGP 一些属性：

```yaml
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: local
  namespace: metallb-system
spec:
  ipAddressPools:
  - bgp-pool
  aggregationLength: 32
```

  - aggregationLength：路由后缀聚合长度，默认为 32，意味这 BGP 通告的路由的掩码为 32，值调小可聚合路由条数

- aggregationLengthV6：同上，用于 ipv6，默认为 128
- ipAddressPools：[]string，选择需要 BGP 通告的地址池
- ipAddressPoolSelectors：通过 label 筛选地址池
- nodeSelectors：通过 node label 筛选 loadBalancerIP 的下一跳节点，默认为全部节点
- peers：[]string，BGPPeer 对象的名称，用于声明此 BGPAdvertisement 作用于哪些 BGP 会话
- communities：参考 BGP communities，可以直接配置，也可以指定 communities CRD 的名称

- 配置 BGP Peer

  BGP Peer 用于配置 BGP 会话的配置，包括对端 BGP AS 及 IP 等。

```yaml
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: test
  namespace: metallb-system
spec:
  myASN: 7776
  peerASN: 7675
  peerAddress: 172.16.1.1
  routerID: 172.16.1.11
```

  - myASN：本端 ASN，范围为 1-64511(public AS)、64512-65535(private AS)
  - peerASN：对端 ASN，范围同上。如果二者相等，则为 iBGP；否则为 eBGP
  - peerAddress：对端路由器 IP 地址
  - sourceAddress：指定本段建立 BGP 会话的地址，默认从本节点网卡自动选择

- nodeSelectors：根据 node label 指定哪些节点需要跟 BGP Router 建立会话

- 创建 LoadBalancer 类型的 Service

```
$ kubectl get svc | grep LoadBalancer
metallb-demo    LoadBalancer   172.31.63.207   10.254.254.1   18081:30531/TCP   3h38m
```

验证

在 BGP Router 上可以看到通过 BGP 学习到的路由：

```
$ vtysh

Hello, this is FRRouting (version 8.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/100] via 10.0.2.2, eth0, src 10.0.2.15, 03:52:17
C>* 10.0.2.0/24 [0/100] is directly connected, eth0, 03:52:17
K>* 10.0.2.2/32 [0/100] is directly connected, eth0, 03:52:17
B>* 10.254.254.1/32 [20/0] via 172.16.1.11, eth1, weight 1, 03:32:16
  *                        via 172.16.2.21, eth2, weight 1, 03:32:16
C>* 172.16.1.0/24 is directly connected, eth1, 03:52:17
```

可以看到通往 LoadBalancerIP 的下一跳分别是集群节点 1 和节点 2，在 BGP Router 执行连通性测试：

root@router:~# curl 10.254.254.1:18081
{"pod_name":"metallb-demo","pod_ip":"172.20.166.20","host_name":"worker1","client_ip":"172.20.161.0"}

FRR Mode

目前 Metallb BGP 模式有两种 Backend 实现：Native BGP 和 FRR BGP。

FRR BGP 目前是实验阶段，对比 Native BGP，FRR BGP 主要有以下几个优点：

- BFD 协议支持（提高故障反应能力，缩短故障时间）
- 支持 IPV6 BGP
- 支持 ECMP

安装 Multus-underlay

本页介绍如何安装 Multus-underlay 组件。

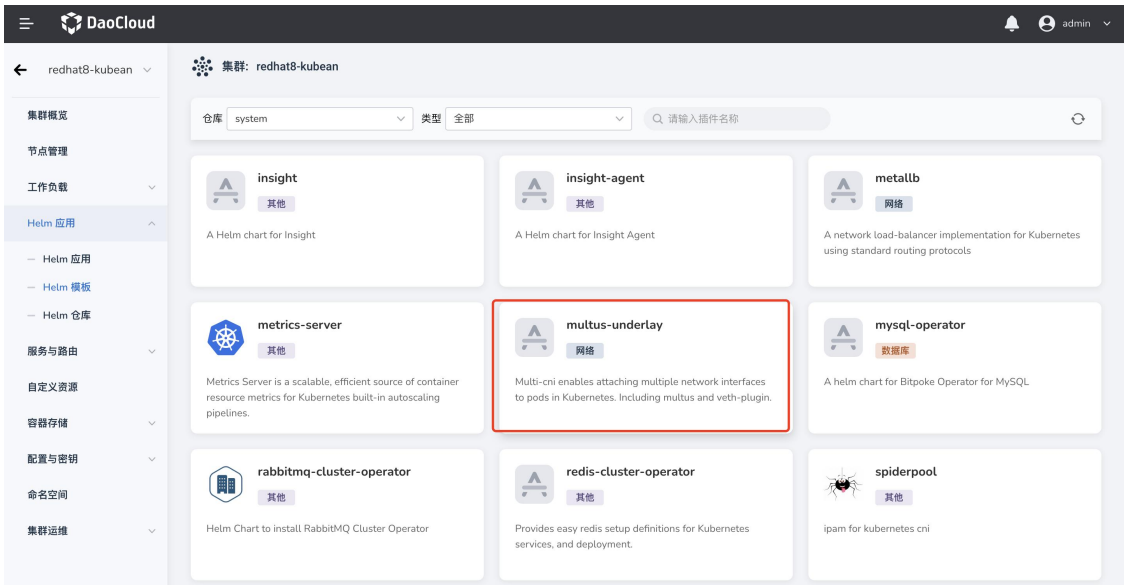
注意事项

- 默认 CNI：安装 Multus-underlay 之前，需要确认当前集群是否存在默认 CNI，比如 Calico 或者 Cilium，否则 Multus 可能会无法工作。
- Spiderpool：Multus-underlay 依赖 Spiderpool 作为 ipam。 安装 Spiderpool 请参考 Install Spiderpool。
- 如需安装 SRIOV-CNI，需要确认节点是否为物理主机且节点拥有支持 SRIOV 的物理网卡。 如果节点为 VM 虚拟机或者没有支持 SR-IOV 的网卡，那么 SR-IOV 将无法工作。 详情参考 sriov-device-plugin。
- 不建议同时安装 MacVLAN 和 SR-IOV。

安装步骤

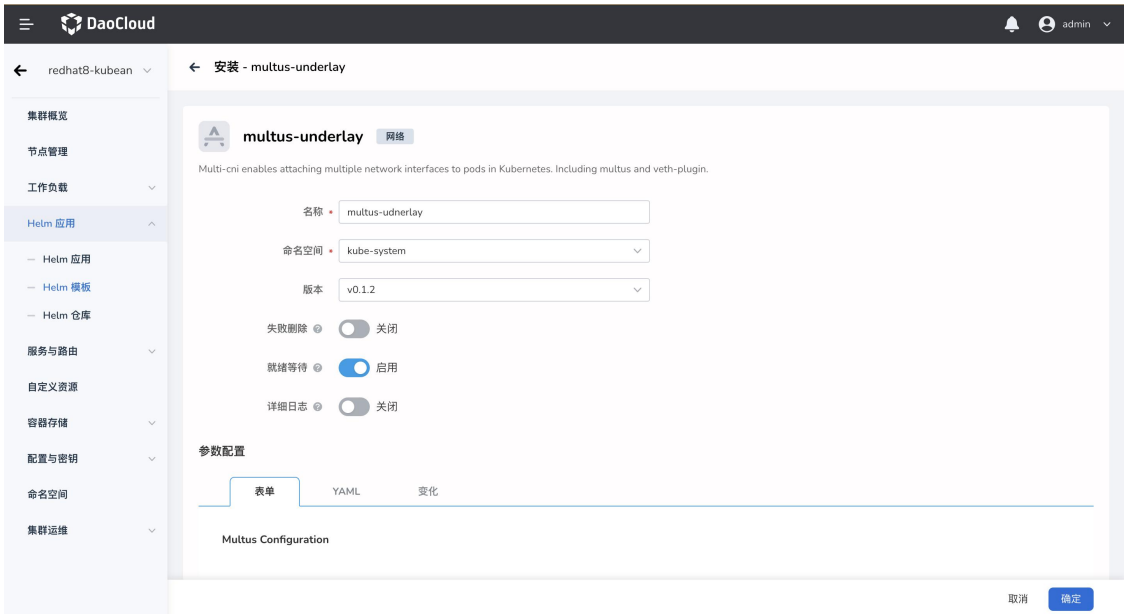
请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Multus-underlay。

1. 在左侧导航栏点击容器管理—>集群列表，然后找到准备安装 Multus-underlay 的集群名称。然后，在左侧导航栏中选择 Helm 应用 -> Helm 模板，找到并点击 multus-underlay。



helm repo

2. 进入安装界面，填写基础配置信息。命名空间选择 kube-system，并开启就绪等待:



helm install-1

3. 为 Multus 配置默认 CNI:

安装 Multus 之前，必须要先安装一种 CNI 作为默认 CNI。注意：保证选择的 Value 必须与集群目前安装的默认 CNI 保持一致。

!!! note

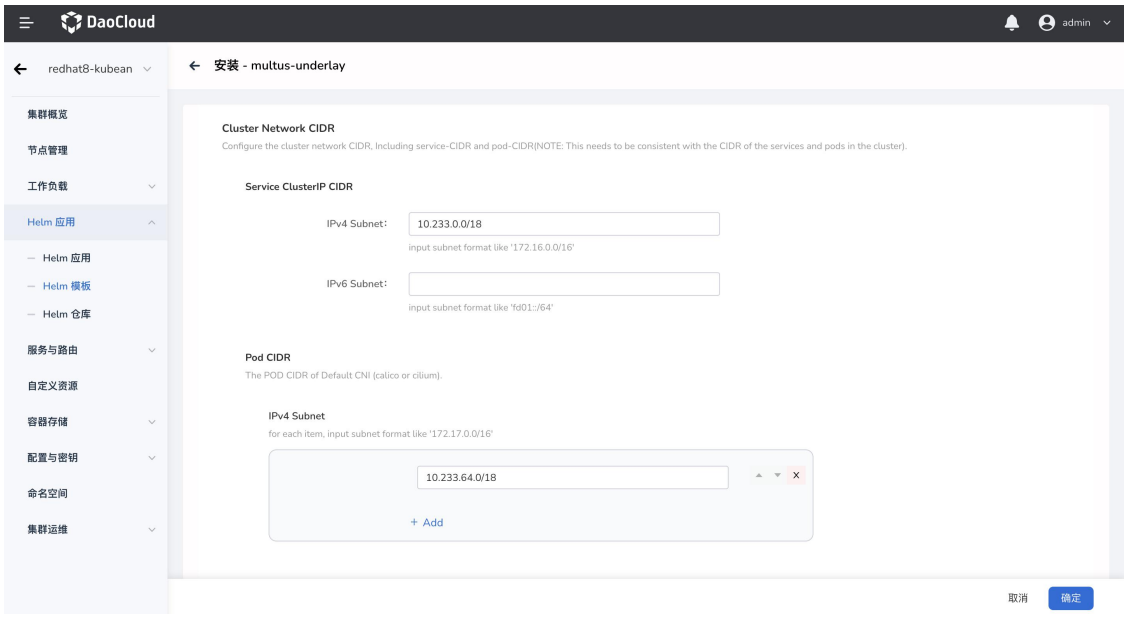
> 如果当前通过 kubean 安装的集群，那么 value 为 calico 或者 cilium 中二选一。
> 或通过查看主机：`/etc/cni/net.d/` 路径，按照字典顺序第一个 CNI 配置文件的 `name` key 所对应的 Value 值就为默认 CNI。比如：

```shell
root@master:~# ls /etc/cni/net.d/
10-calico.conflist  calico-kubeconfig
root@master:~# cat /etc/cni/net.d/10-calico.conflist
{
  "name": "k8s-pod-network",
  "cniVersion": "0.3.1",
...
```

> `name` 的值如果为 `k8s-pod-network`，那么这里就应该选中 `k8s-pod-network`。
> ![Default CNI](https://docs.daocloud.io/daocloud-docs-images/docs/network/images/install2.png)
> 如果当前集群是接入的第三方、calico 为 CNI 的集群，那么这里应该选择为 `k8s-pod-network`. 同样，也可以通过查看主机上`/etc/cni/net.d`文件确认。

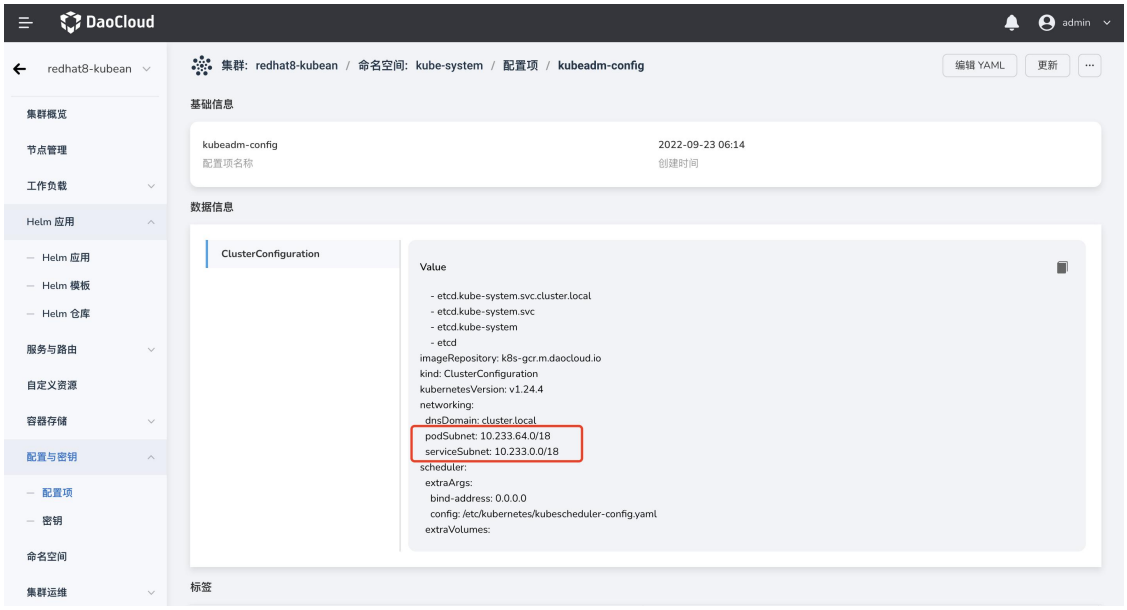
4. 配置目前集群 Service 和 Pod 的 CIDR:

此步骤的目的是告知 Meta-Plugins 集群的 CIDR，Meta-Plugins 会创建对应的路由规则，解决 Underlay CNI 的集群东西向通信问题。



Cluster CIDR

可通过查看 configMap: kube-system/kubeadm-config 获取目前集群中 Service 和 Pod 的 CIDR：



kubeadm-config

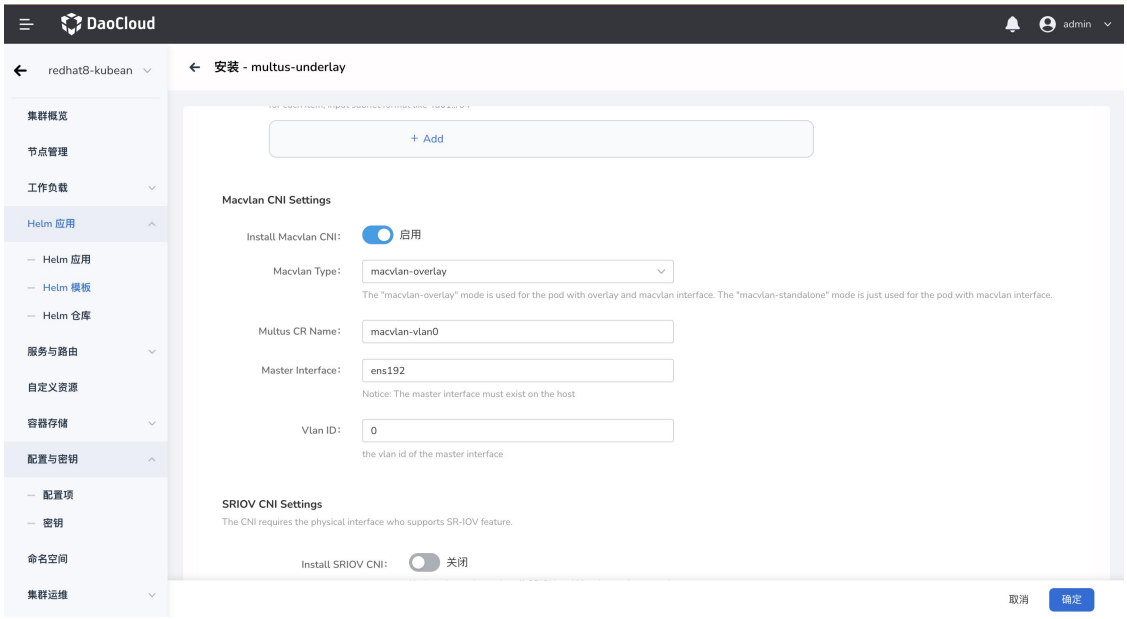
!!! note

如果为双栈集群，也需要填写 IPv6 的地址。

如果 CNI 为 Calico 且有多个 IP 池，Pod CIDR 可配置多个。

5. 安装 MacVLAN（可选，默认安装）：

此步骤会根据配置创建 MacVLAN 对应的 Multus CRD 实例:



macvlan

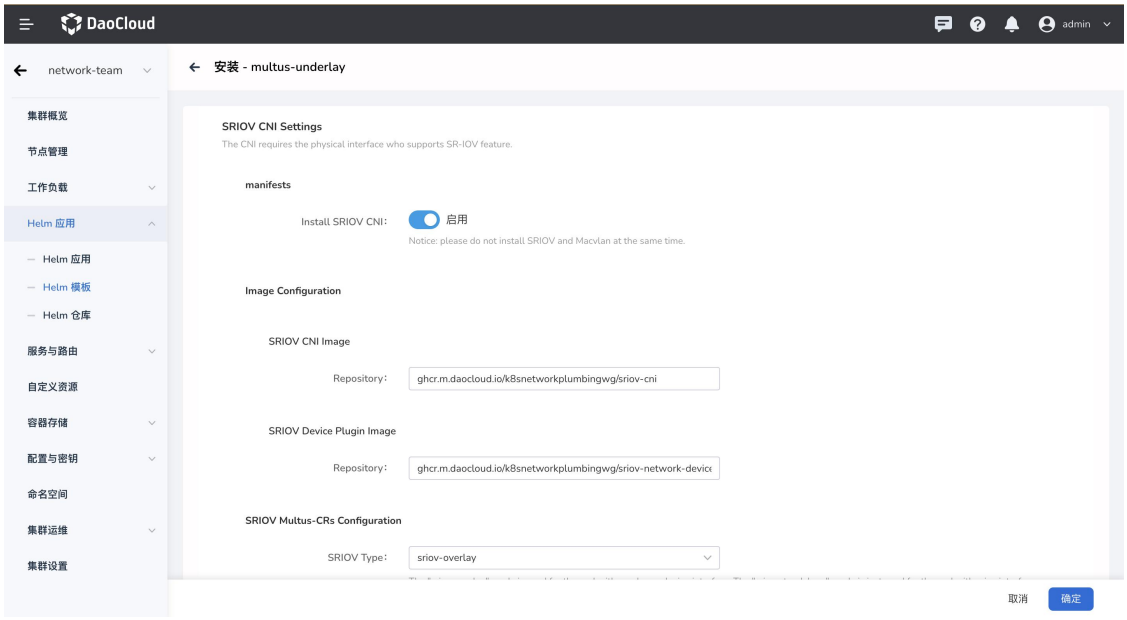
- Install Macvlan CNI：true/false，是否创建 MacVLAN 的 Multus CRD 实例。
- Macvlan Type：macvlan-overlay/macvlan-standalone，安装 MacVLAN CRD 实例的类型。
  - macvlan-overlay：此类型下，MacVLAN 会与默认 CNI 搭配使用（比如 Calico），这样会在 Pod 中插入两张网卡。 分别是默认 CNI 和 MacVLAN 的网卡，前者用于解决 Pod 与集群东西向通信问题；后者用于 Pod 集群南北向通信。

- macvlan-standalone：此类型下，Pod 中只会插入一张 MacVLAN 的网卡，只由其完成与集群东西向和南北向的通信问题。

- Multus CR Name：Multus CRD 实例的名称。
- Master Interface：MacVLAN 主接口的名称。注意：配置的主接口必须存在于主机上，否则 MacVLAN 无法工作。
- Vlan ID：可选，MacVLAN 主接口的 Vlan tag。

6. 安装 SR-IOV（可选，默认不安装）：

   配置 SR-IOV Multus CRD：



sriov_install

- Install SRIOV CNI：是否安装 SR-IOV，默认不安装。
- SRIOV Type：安装 SR-IOV 的 Multus CRD 实例的类型，有以下几种：
  - sriov-overlay：此类型下，SR-IOV 会与默认 CNI 搭配使用（比如 Calico），这样会在 Pod 中插入两张网卡。 分别是

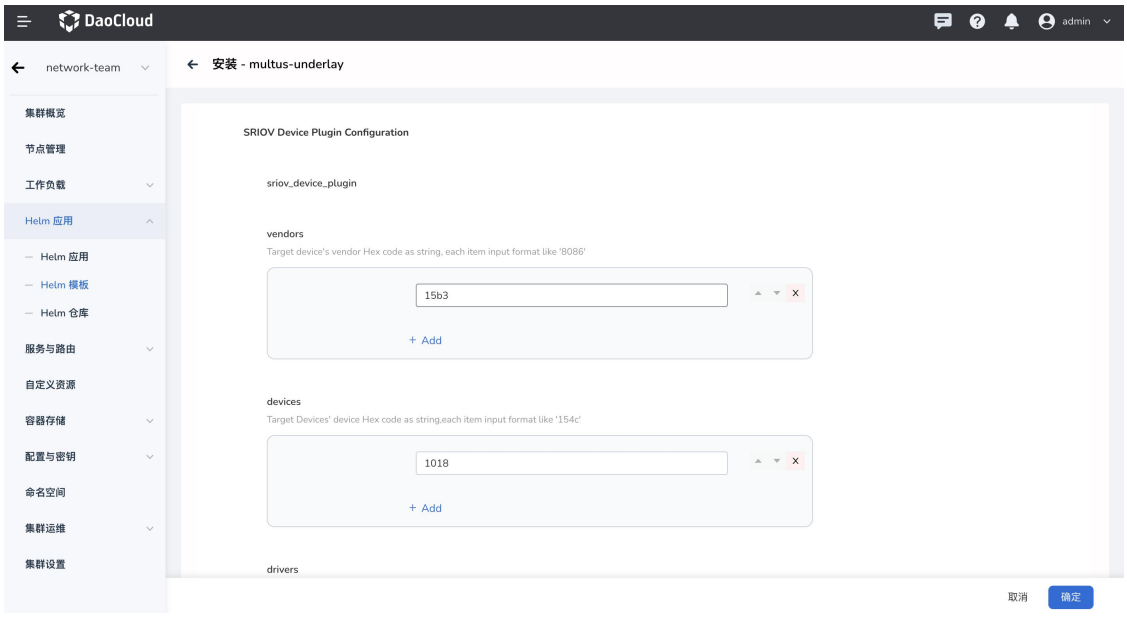
默认 CNI 和 SR-IOV 的网卡，前者用于解决 Pod 与集群东西向通信问题；后者用于 Pod 集群南北向通信。

- sriov-standalone：此类型下，Pod 中只会插入一张 SR-IOV 的网卡，只由其完成与集群东西向和南北向的通信问题。

- SRIOV CR Name：Multus CRD 实例的名称。
- Vlan ID：可选，SR-IOV PF 的 Vlan tag。
- SRIOV Device Plugin Configuration：用于发现主机上的 SR-IOV PF 和 VF device，筛选方式可以为：vendors、devices、drivers、pfNames。 具体参考 sriov-device-plugin-readme.md。

配置 SR-IOV Net-Device Plugin：

- vendors：PCI 设备厂商号，如‘8086’代表 Intel
- devices：PCI 设备型号，如‘154c’
- drivers：PCI 设备驱动，如‘mlx5_core’
- pfNames：PF 设备的名称列表



sriov-net-device

!!! note

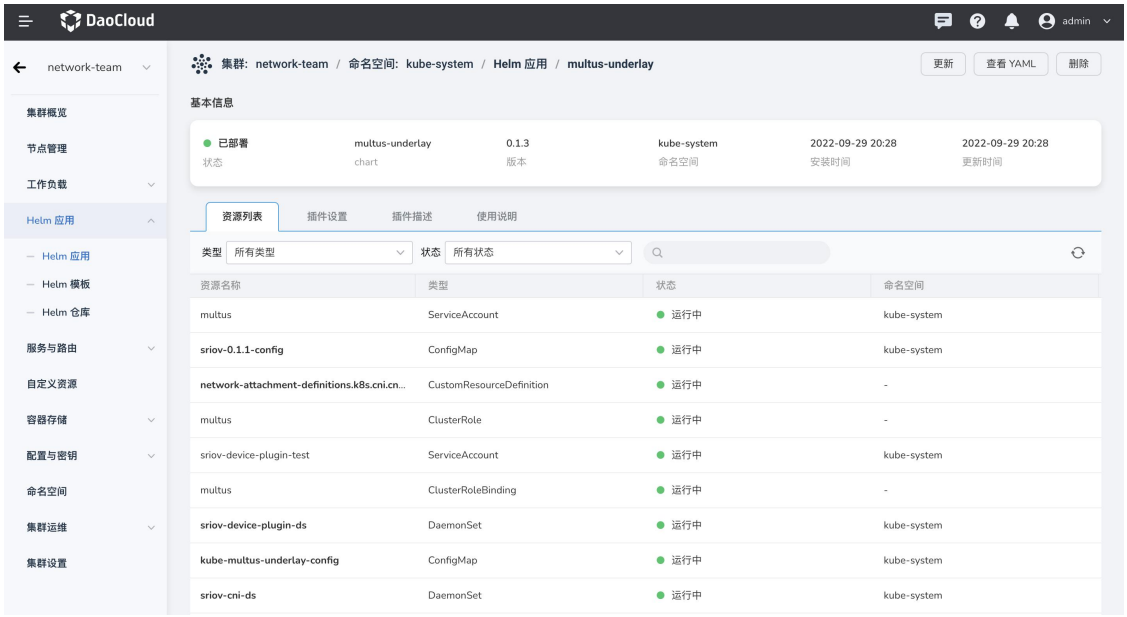
不建议同时启用 MacVLAN 和 SR-IOV。另外启用 SR-IOV 需要硬件支持，安装前确认物理主机的网卡是否支持 SR-IOV。

7. 配置完成，点击安装。

验证

1. 检查各组件是否正常 Running：

   包括 Multus、Meta-plugins、SR-IOV CNI（如果启用）、SRIOV-Device-Plugins（如果启用）。



install_finished

2. 创建工作负载，以 MacVLAN 为例：

   - 如果 type 为 macvlan-overlay，那么需要在 Pod 的 Annotations 中插入以下的注解：

     ```
     annotations:
       k8s.v1.cni.cncf.io/networks: kube-system/macvlan-vlan0
     ```

     k8s.v1.cni.cncf.io/networks：表示会在 Pod 中除默认 CNI 之外再插入一张 MacVLAN 网卡。

- 如果 type 为 macvlan-standalone，那么需要在 Pod 的 Annotations 中插入以下的注解：

```
annotations:
  v1.multus-cni.io/default-network: kube-system/macvlan-vlan0
```

v1.multus-cni.io/default-network：修改 Pod 的默认网卡。如果不指定，将通过集群默认 CNI 为 Pod 分配第一张网卡。

以 type 为 macvlan-overlay 为例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: macvlan-overlay
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app.kubernetes.io/instance: dao2048-macvlan-overlay
      app.kubernetes.io/name: dao2048-macvlan-overlay
  template:
    metadata:
      annotations:
        ipam.spidernet.io/ippool: |-
          {
                "interface": "net1",  # 1. 指定 Pod 第二张网卡(net1)从哪一个 IPPool 池中分配 IP.
                "ipv4": [
                  "172-81-0-1"
                ],
                "ipv6": [
```

```
            "172-81-0-1-v6"
        ]
    }
    k8s.v1.cni.cncf.io/networks: kube-system/macvlan-vlan0   # 2. 设置 Pod 第二张网卡.
    ...
```

ipam.spidernet.io/ippool：指定从哪一个 IP 池为 MacVLAN 网卡分配 IP 地址。 如果不指定，将会从默认池中分配。更多 Spiderpool 使用说明请参考 Spiderpool。

k8s.v1.cni.cncf.io/networks：通过指定 MacVLAN Multus CRD，为 Pod 再分配一张 MacVLAN 网卡（net1）。

创建成功:

```
root@master:~# kubectl get po  -o wide | grep overlay
macvlan-overlay-b968998d8-mbtbp              1/1     Running   0          54s     10.250.186.234   master   <none>           <none>
macvlan-overlay-589d6ddc68-kk798             1/1     Running   0          43s     10.253.255.73    172-17-8-120   <none>           <none>
```

3. 测试连通性。

可以看到 Pod 的第一张网卡仍然由 Calico 分配，第二张网卡由 MacVLAN 分配：

```
root@master:~# kubectl exec -it macvlan-overlay-589d6ddc68-kk798 sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNK
```

```
NOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
4: eth0@if503978: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN>
mtu 1430 qdisc noqueue state UP
    link/ether ba:b0:dc:d9:91:1c brd ff:ff:ff:ff:ff:ff
    inet 10.253.255.73/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fd00:10:244::eaa:bafd/128 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::b8b0:dcff:fed9:911c/64 scope link
       valid_lft forever preferred_lft forever
5: net1@if7: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 150
0 qdisc noqueue state UP
    link/ether 4e:31:cb:b5:68:13 brd ff:ff:ff:ff:ff:ff
    inet 172.17.8.193/21 brd 172.17.15.255 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fd00:1033::172:17:8:14b/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::4c31:cbff:feb5:6813/64 scope link
       valid_lft forever preferred_lft forever
```

测试

MacVLAN 网卡的 IP 地址段从宿主机分配，所以在宿主机网络路由可达的情况下，可以直接访问该 Pod 的 MacVLAN 网卡，测试步骤如下：

1. 在集群节点中访问 10.253.255.73 和 172.17.8.193。

```
# ping pod eth0
root@master:~# ping 10.253.255.73
PING 10.253.255.73 (10.253.255.73) 56(84) bytes of data.
64 bytes from 10.253.255.73: icmp_seq=1 ttl=64 time=0.170 ms
64 bytes from 10.253.255.73: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 10.253.255.73: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.253.255.73: icmp_seq=4 ttl=64 time=0.117 ms
64 bytes from 10.253.255.73: icmp_seq=5 ttl=64 time=0.073 ms
^C
--- 10.253.255.73 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4076ms
rtt min/avg/max/mdev = 0.073/0.105/0.170/0.035 ms

# ping pod net1
root@master:~# ping 172.17.8.193
PING 172.17.8.193 (172.17.8.193) 56(84) bytes of data.
64 bytes from 172.17.8.193: icmp_seq=1 ttl=64 time=0.115 ms
64 bytes from 172.17.8.193: icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from 172.17.8.193: icmp_seq=3 ttl=64 time=0.091 ms
64 bytes from 172.17.8.193: icmp_seq=4 ttl=64 time=0.103 ms
^C
--- 172.17.8.193 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 0.068/0.094/0.115/0.017 ms
```

2. 在集群外访问 172.17.8.193。

```
$ ping 172.17.8.193
PING 172.17.8.193 (172.17.8.193): 56 data bytes
64 bytes from 172.17.8.193: icmp_seq=0 ttl=62 time=37.668 ms
64 bytes from 172.17.8.193: icmp_seq=1 ttl=62 time=44.025 ms
64 bytes from 172.17.8.193: icmp_seq=2 ttl=62 time=39.606 ms
^C64 bytes from 172.17.8.193: icmp_seq=3 ttl=62 time=40.599 ms
```

```
--- 172.17.8.193 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 37.668/40.474/44.025/2.305 ms
```

3. 访问集群中的 Calico Pod。

```
root@master:~# kubectl get po  -o wide | grep nginx
nginx-8f458dc5b-kshpw                    1/1     Running   0          2
d3h   10.250.186.210   172-17-8-110   <none>           <none>
nginx-8f458dc5b-wgkpg                    1/1     Running   0          4
6h    10.253.255.72    172-17-8-120   <none>           <none>

root@master:~# kubectl exec -it macvlan-overlay-b968998d8-mbtbp  sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in
 a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ # ping 10.250.186.210
PING 10.250.186.210 (10.250.186.210): 56 data bytes
64 bytes from 10.250.186.210: seq=0 ttl=63 time=0.247 ms
64 bytes from 10.250.186.210: seq=1 ttl=63 time=0.151 ms
^C
--- 10.250.186.210 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.151/0.199/0.247 ms
/ # ping 10.253.255.72
PING 10.253.255.72 (10.253.255.72): 56 data bytes
64 bytes from 10.253.255.72: seq=0 ttl=62 time=0.596 ms
64 bytes from 10.253.255.72: seq=1 ttl=62 time=0.428 ms
64 bytes from 10.253.255.72: seq=2 ttl=62 time=0.387 ms
^C
--- 10.253.255.72 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.387/0.470/0.596 ms
```

4. 访问 ClusterIP。

```
root@master:~# kubectl get svc | grep kubernetes
Kubernetes                    ClusterIP    172.96.0.1                  <none>
  443/TCP                                                    78d
root@172-17-8-110:~# kubectl get svc | grep nginx
nginx-172-17-8-110-v4      NodePort      172.96.53.151              <none>
      80:31696/TCP,5201:32137/TCP,12865:31253/TCP    78d
nginx-172-17-8-110-v6      NodePort      2001:4860:fd00::85db    <none>
       80:32452/TCP,5201:32658/TCP,12865:32231/TCP    78d
root@master:~# kubectl exec -it macvlan-overlay-5c87d74c46-fzkln sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in
 a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ # curl 172.96.53.151
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed an
d
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
```

```
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

卸载

卸载 Multus-underlay 还需要删除每个节点上的 Multus 配置文件：

```
root@master:~# rm /etc/cni/net.d/00-multus.conf
root@master:~# rm -rf /etc/cni/net.d/multus.d
```

Macvlan

Macvlan 是 Linux 的一种网卡虚拟化的解决方案，它可以将一张物理网卡虚拟为多张虚拟网卡。 借助 Mutus，可以为 Pod 分配一张或者多张 Macvlan 网卡，从而实现 Pod 借助 macvlan 网卡与外部通讯。

安装

在 Kubernetes 中，Macvlan 只是存放于每个节点 /opt/cni/bin 下的一个二进制文件，不存在单独的安装方式。 默认情况下，安装集群时将包括 macvlan 在内的多个插件复制到每个节点的 /opt/cni/bin 下。 如果在节点的 /opt/cni/bin 下未发现 macvlan 的二进制文件， 那么需要手动下载 cni-plugins， 并解压到各个节点上。而安装 multus-underlay 时，仅创建属于 Macvlan 的 Multus network-attachment-definition CRD 对象。

说明

Multus + Macvlan 一般会有两种使用场景：

- macvlan-standalone

  类型为 macvlan-standalone，意味着 Pod 的第一张网卡（eth0）为 macvlan 分配的网卡，通过在 Pod 的 annotations 中插入如下字段：

  ```
  annotations:
    v1.multus-cni.io/default-network: kube-system/macvlan-standalone-vlan0
  ```

  注意：macvlan-standalone 只与 macvlan-standalone 类型搭配，不能与 macvlan-overlay 搭配。你可以通过下面的方式为 Pod 插入多张 macvlan 网卡：

```
annotations:
  v1.multus-cni.io/default-network: kube-system/macvlan-standalone-vlan0
  k8s.v1.cni.cncf.io/networks: kube-system/macvlan-standalone-vlan0
```

- macvlan-overlay

  此类型意味着 macvlan 与 overlay 类型的 CNI 搭配（如 calico 或 cilium），macvlan 不作为 Pod 的缺省 CNI，即不会是 Pod 的第一张网卡（eth0）。 所以 macvlan-overlay 类型的 Pod 要与 overlay 类型的 Pod 能够正常通讯。你可以通过下面的方式为 Pod 多分配一张网卡：

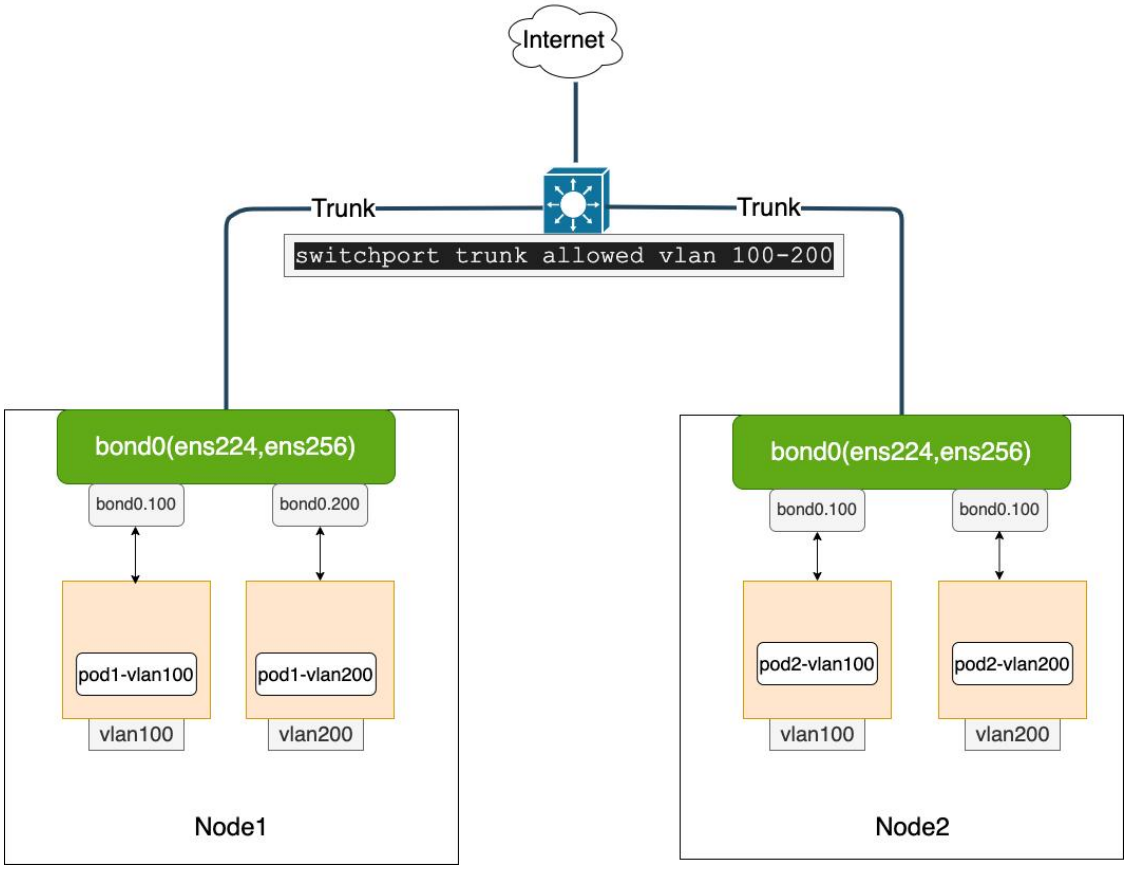
  ```
  annotations:
    k8s.v1.cni.cncf.io/networks: kube-system/macvlan-overlay-vlan0
  ```

  !!! caution

  `v1.multus-cni.io/default-network` 的值不能为 macvlan-overlay 类型的 CRD，即 macvlan-overlay 不能作为 Pod 的第一张网卡。

其他

使用 macvlan 一个比较常见的网络场景：



如图所示，将主机上两个物理接口（ens224、ens256）组成一个 bond0，然后根据 bond0 创建两个 VLAN 子接口，分别是 bond0.100 和 bond0.200。 然后将 bond0（也就是 ens224 和 ens256）与交换机 trunk 相连。并且在交换机上配置允许 vlan100 和 vlan200 通过。

然后创建两个不同 vlan 的 macvlan multus 实例，它们的 master 接口分别为 bond0.100 和 bond0.200。 这样使用不同 macvlan multus 实例创建的 Pod 也就属于不同的 vlan 了。但它们都可以通过交换机做到同 vlan 或不同 vlan 之间的通讯。

注意：它们的网管应该指向交换机对应的 vlanif IP 地址。

这是一个比较常见、稍复杂的网络拓扑。总结：

- 在主机上创建 bond 和 vlan 接口
- 配置交换机

- 创建 multus 的 CRD 实例
- 创建不同的 Spiderpool IP 池
- 在 Pod 的 annotations 中指定对应的实例和选择对应的 spiderpool IP 池

在主机上创建 bond 和 vlan 等接口，可以参考 nmstat 用法。

SR-IOV CNI 配置

环境要求

使用 SR-IOV CNI 需要先确认节点是否为物理主机并且节点拥有支持 SR-IOV 的物理网卡。 如果节点为 VM 虚拟机或者没有支持 SR-IOV 的网卡，那么 SR-IOV 将无法工作。 可通过下面的方式检查节点是否存在支持 SR-IOV 功能的网卡。

检查是否支持 SR-IOV

通过 ip link show 获取所有网卡：

```
root@172-17-8-120:~# ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether b8:ca:3a:67:e5:fc brd ff:ff:ff:ff:ff:ff
    altname enp1s0f0
3: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode DEFAULT group default qlen 1000
    link/ether b8:ca:3a:67:e5:fd brd ff:ff:ff:ff:ff:ff
    altname enp1s0f1
4: eno3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode DEFAULT group default qlen 1000
    link/ether b8:ca:3a:67:e5:fe brd ff:ff:ff:ff:ff:ff
    altname enp1s0f2
5: eno4: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode DEFAULT group default qlen 1000
    link/ether b8:ca:3a:67:e5:ff brd ff:ff:ff:ff:ff:ff
```

```
    altname enp1s0f3
6: enp4s0f0np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 04:3f:72:d0:d2:86 brd ff:ff:ff:ff:ff:ff
7: enp4s0f1np1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 04:3f:72:d0:d2:87 brd ff:ff:ff:ff:ff:ff
8: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:60:cb:04:10 brd ff:ff:ff:ff:ff:ff
```

过滤常见的虚拟网卡（如 docker0、cali*、vlan 子接口等），以 enp4s0f0np0 为例，确认其是否支持 SR-IOV：

```
root@172-17-8-120:~# ethtool -i enp4s0f0np0
driver: mlx5_core      # 网卡驱动
version: 5.15.0-52-generic
firmware-version: 16.27.6008 (LNV0000000033)
expansion-rom-version:
bus-info: 0000:04:00.0   # PCI 设备号
supports-statistics: yes
supports-test: yes
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: yes
```

通过 bus-info 查询其 PCI 设备详细信息：

```
root@172-17-8-120:~# lspci -s 0000:04:00.0 -v | grep SR-IOV
    Capabilities: [180] Single Root I/O Virtualization (SR-IOV)
```

如果输出有上面此行，说明此网卡支持 SR-IOV。获取此网卡的 vendor 和 device：

```
root@172-17-8-120:~# lspci -s 0000:04:00.0 -n
04:00.0 0200: 15b3:1017
```

其中，

- 15b3：表示此 PCI 设备的厂商号，如 15b3 表示 Mellanox
- 1017：表示此 PCI 设备的设备型号，如 1017 表示 Mellanox MT27800 Family [ConnectX-5] 系列网卡

  可通过 https://devicehunt.com/all-pci-vendors 查询所有 PCI 设备信息。

配置 VF（虚拟功能）

通过下面的方式为支持 SR-IOV 的网卡配置 VF：

```
root@172-17-8-120:~# echo 8 > /sys/class/net/enp4s0f0np0/device/sriov_numvfs
```

确认 VF 配置成功：

```
root@172-17-8-120:~# cat /sys/class/net/enp4s0f0np0/device/sriov_numvfs
8
root@172-17-8-120:~# ip l show enp4s0f0np0
6: enp4s0f0np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 04:3f:72:d0:d2:86 brd ff:ff:ff:ff:ff:ff
    vf 0     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state auto, trust off, query_rss off
    vf 1     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state auto, trust off, query_rss off
    vf 2     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state auto, trust off, query_rss off
    vf 3     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state auto, trust off, query_rss off
    vf 4     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, l
```

```
ink-state auto, trust off, query_rss off
    vf 5     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, l
ink-state auto, trust off, query_rss off
    vf 6     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, l
ink-state auto, trust off, query_rss off
    vf 7     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, l
ink-state auto, trust off, query_rss off
```

输出上图内容，表示配置成功。

安装 SR-IOV CNI

通过安装 Multus-underlay 来安装 SR-IOV CNI，具体安装流程参考安装。 注意，安装时需正确配置 sriov-device-plugin resource 资源，包括 vendor、device 等信息。 否则 SRIOV-Device-Plugin 无法找到正确的 VF。

配置 SRIOV-Device-Plugin

安装完 SR-IOV CNI 之后，通过下面的方式查看 SR-IOV CNI 是否发现了主机上的 VF：

```
root@172-17-8-110:~# kubectl describe nodes 172-17-8-110
...
Allocatable:
  cpu:                                24
  ephemeral-storage:                  881675818368
  hugepages-1Gi:                      0
  hugepages-2Mi:                      0
  intel.com/sriov-netdevice:          8          # 此行表示 SR-IOV CNI 成功的发现了该主
机上的 VFs
  memory:                             16250260Ki
  pods:                               110
```

使用

使用 SR-IOV CNI 创建工作负载需要注意下面三个方面：

- 确认 SR-IOV multus network-attachment-definition 对象中存在 sriov-device-plugin 资源：

  root@172-17-8-110:~# kubectl get network-attachment-definitions.k8s.cni.cncf.io -n kube-system sriov-vlan0 -o yaml
  apiVersion: k8s.cni.cncf.io/v1
  kind: NetworkAttachmentDefinition
  metadata:
    annotations:
      k8s.v1.cni.cncf.io/resourceName: intel.com/sriov_netdevice
    name: sriov-vlan0
    namespace: kube-system

  resourceName 必须与注册到 kubelet 中的资源名称保持一致（即和 Node 对象中的名称保持一致）。

  创建 network-attachment-definition 对象后，再更新 annotations: k8s.v1.cni.cncf.io/resourceName 不会生效！

- 创建工作负载时，需要在 Pod 的 annotations 字段中通过 multus 的注解绑定指定的 network-attachment-definition 对象。

  - 如果 type 为 sriov-overlay，那么需要在 Pod 的 Annotations 中插入以下的注解：

    ```
    annotations:
      k8s.v1.cni.cncf.io/networks: kube-system/sriov-overlay-vlan0
    ```

    k8s.v1.cni.cncf.io/networks：表示会在 Pod 中除默认 CNI 之外再插入一张 SR-IOV CNI 网卡。

  - 如果 type 为 sriov-standalone，那么需要在 Pod 的 Annotations 中插入以下的注解：

```
    annotations:
      v1.multus-cni.io/default-network: kube-system/sriov-standalone
-vlan0
```

v1.multus-cni.io/default-network：修改 Pod 的默认网卡。如果不指定，将通过集群默认 CNI 为 Pod 分配第一张网卡。

- 在 Pod 的 Resource 字段中给予容器 SR-IOV 资源配额:

```
...
  containers:
    resources:
      requests:
        intel.com/sriov-netdevice: '1'
      limits:
        intel.com/sriov-netdevice: '1'
...
```

名称需要保持一致，否则 Pod 会因为请求不到 VF 而创建失败。

什么是 Multus-underlay

Multus-underlay 基于 Multus 并搭配一些 Underlay 类型的 CNI 插件（如 MacVLAN 或 SR-IOV CNI 等），可为 Pod 插入多张网卡。 此外，Multus-underlay 还解决了 Underlay CNI 的一些通信问题。 比如当 MacVLAN 作为 CNI 时，它无法与集群 ClusterIP 直接通信，并且它也无法直接与主机 MacVLAN Master 接口通信的（这是 Linux MacVLAN 技术的限制）。

解决的问题

- Underlay CNI 类型的 Pod 无法访问集群 ClusterIP
- Underlay CNI 类型的 Pod 无法通过健康检测

包含的组件

Multus-underlay 包含下面几个组件：

- Multus：调度多个 CNI 插件，按需为 Pod 插入一张或者多张网卡
- Meta-plugins：包括 Veth 和 Router 两个插件，解决 Underlay CNI 的一些通信问题
- SR-IOV CNI（可选）：可选安装 SR-IOV CNI，但需要硬件支持

Meta-plugins

Meta-plugins 包含两个 Meta 插件。 分别是 Veth 和 Router，它们以 CNI-Chain 的方式被 CRI 调用。在 MacVLAN/SR-IOV 类的插件调用完成之后再调用， 通过在 Pod 的 NetNs 中设置一些规则，解决各种通信问题。

Veth-Plugin

Veth 插件与 ptp 有些类似，通过在 Pod NetNs 中添加一对 Veth-Peer 设备，并劫持来自主机、集群中的流量从 Veth 设备通过。 而集群南北向流量仍然从 eth0 通过。下面是 veth 插件搭配 MacVLAN 的 multus CRD 实例的配置示例：

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-standalone
  namespace: kube-system
spec:
  config: |-
    {
        "cniVersion": "0.3.1",
        "name": "macvlan-standalone",
        "plugins": [
            {
                "type": "macvlan",
                "master": "enp4s0f0np0",  # macvlan 主接口
                "mode": "bridge",
                "ipam": {
                    "type": "spiderpool", # ipam 使用 spiderpool
                    "log_level": "DEBUG",
                    "log_file_path": "/var/log/spidernet/spiderpool.log",
                    "log_file_max_size": 100,
                    "log_file_max_age": 30,
                    "log_file_max_count": 10
                }
            },{
                "type": "veth", # veth 以 cni-chain 的方式调用
                "service_hijack_subnet": ["172.96.0.0/18","2001:4860:fd00::/108"],
 # 集群 service 的网段，包括 IPv4 和 IPv6
                "overlay_hijack_subnet": ["10.240.0.0/12","fd00:10:244::/96"],  #
集群 pod 的网段集合
                "additional_hijack_subnet":[], # 可自定义的网段,访问此集合的网段
的数据包将会先从 veth 设备送往主机，再由主机进行转发。
                "rp_filter": {
```

```
                "set_host": true,
                "value": 0
            },
            "migrate_route": -1, # 取值范围`-1,0,1`，默认为 -1，表示是否将新增网卡的默认路由移动到一个新的 route table 中去。-1 表示通过网卡名自动迁移(eth0 < net1 < net2)，0 为不迁移，-1 表示强制迁移。
            "log_options": {  # 日志写入文件
              "log_level": "debug",
              "log_file": "/var/log/meta-plugins/veth.log"
            }
        }
    ]
}
```

Router-Plugin

Router 插件通过在 Pod Netns 中设置一些路由规则，使来自主机、集群内的数据包从 Pod 的 eth0（默认 CNI 创建的网卡）转发，而来自于集群外的数据包从 MacVLAN/SR-IOV 创建的网卡转发。 下面是 Router 插件搭配 MacVLAN 的 Multus CRD 实例的配置示例：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-vlan0-overlay
  namespace: kube-system
spec:
  config: |-
    {
        "cniVersion": "0.3.1",
        "name": "macvlan-overlay",  # macvlan 主接口
        "plugins": [
            {
```

```
            "type": "macvlan",
            "master": "enp4s0f0np0",
            "mode": "bridge",
            "ipam": {
                "type": "spiderpool",  # ipam 使用 spiderpool
                "log_level": "DEBUG",
                "log_file_path": "/var/log/spidernet/spiderpool.log",
                "log_file_max_size": 100,
                "log_file_max_age": 30,
                "log_file_max_count": 10
            }
        },{
            "type": "router", # router 插件 以 cni-chain 的方式调用
            "overlay_interface": "eth0",
            "migrate_route": -1,  # 取值范围`-1,0,1`，默认为 -1，表示是否将新增网卡的默认路由移动到一个新的 route table 中去。-1 表示通过网卡名自动迁移(eth0 < net1 < net2)，0 为不迁移，1 表示强制迁移。
            "skip_call": false,
            "service_hijack_subnet": ["172.96.0.0/18","2001:4860:fd00::/108"], # 集群 service 的网段，包括 IPv4 和 IPv6
            "overlay_hijack_subnet": ["10.240.0.0/12","fd00:10:244::/96"],  # 集群 pod 的网段集合
            "additional_hijack_subnet":[], # 可自定义的网段。访问此集合的网段的数据包将会先从 eth0 设备送往主机，再由主机进行转发。
            "rp_filter": {
                "set_host": true,
                "value": 0
            },
            "log_options": {
              "log_level": "debug",
              "log_file": "/var/log/meta-plugins/router.log"
            }
```

```
            }
        ]
    }
```

nmstate

随着混合云的出现，节点网络设置变得更加具有挑战性。不同的环境有不同的网络要求。 容器网络接口（CNI）标准实现了不同的解决方案，它解决了集群中 Pod 的通讯问题，包括为其设置 IP 和创建路由等。

然而，在所有这些情况下，节点必须在 Pod 被安排之前设置好网络。 在一个动态的、异质的集群中设置网络，具有动态的网络需求，这本身就是一个挑战。

nmstate 这个项目旨在通过 k8s CRD 的方式配置节点上的网络，它可以一定程度上简化网络配置。

限制

nmstate 依赖 NetworkManager，所以不是所有的 Linux 发行版都支持，比如 ubuntu 等不支持。并且 NetworkManager 的版本必须 >= 1.20

可通过下面的方式检查 NetworkManager 的版本：

```
[root@master ~]# /usr/sbin/NetworkManager --version
1.22.8-4.el8
```

更多内容请查阅 nmstate Helm chart 的 README 文件.

安装

```
helm repo add daocloud https://daocloud.github.io/network-charts-repackage/
helm install nmstate -n nmstate daocloud/nmstate --create-namespace
```

更多安装细节请参考 Helm chart README 文件.

在安装完成后，创建 CR 实例，触发 nmstate controller 工作:

```
cat <<EOF | kubectl create -f -
apiVersion: nmstate.io/v1
kind: NMState
```

```
metadata:
  name: nmstate
EOF
```

当安装完成之后，可通过下面的方式查看是否正常工作：

```
[root@10-6-185-30 ~]# kubectl get nns
NAME           AGE
10-6-185-30    3d4h
10-6-185-40    3d4h
```

每一个 nns 对象都是 nmstate 搜集该节点的所有网络信息，包括所有接口、IP 地址、路由等。

使用

本节展示一些示例。

- 为某个节点的网卡配置 IP 地址

```
cat << EOF | kubectl apply -f -
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: static-ip
spec:
  nodeSelector:
    kubernetes.io/hostname: node02
  desiredState:
    interfaces:
    - name: eth1
      type: ethernet
      state: up
      ipv4:
        address:
```

```
      - ip: 10.244.0.2
        prefix-length: 24
      dhcp: false
      enabled: true
```

其中，

- nodeSelector：表示此网络配置将会在哪个节点上生效
- desiredState：表示期望的配置，包括网卡、IP 等等

- 创建 vlan 子接口

```
cat << EOF | kubectl apply -f -
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan
spec:
  desiredState:
    interfaces:
    - name: eth1.102
      type: vlan
      state: up
      vlan:
        id: 102
        base-iface: eth1
      ipv4:
        dhcp: true
        enabled: true
```

这将会在集群所有节点创建一个名为 eth1.102 的 vlan 子接口。

- 创建 bond 接口

```
cat << EOF | kubectl apply -f -
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond-vlan30
spec:
  nodeSelector:
    kubernetes.io/hostname: 10-6-185-30
  desiredState:
    interfaces:
    - name: bond0
      type: bond
      state: up
      ipv4:
        dhcp: false
        address:
        - ip: 172.39.0.30
          prefix-length: 16
        enabled: true
      ipv6:
        address:
        - ip: fc00:172:39::30
          prefix-length: 64
        enabled: true
      link-aggregation:
        mode: active-backup
        port:
        - ens161
        - ens256
    - name: bond0.144
      type: vlan
      state: up
```

```
    ipv4:
      address:
      - ip: 172.144.185.30
        prefix-length: 16
      dhcp: false
      enabled: true
    ipv6:
      address:
      - ip: fd00:144::172:144:185:30
        prefix-length: 64
      enabled: true
    vlan:
      base-iface: bond0
      id: 144
  - name: bond0.145
    type: vlan
    state: up
    ipv4:
      address:
      - ip: 172.145.185.30
        prefix-length: 16
      dhcp: false
      enabled: true
    ipv6:
      address:
      - ip: fd00:145::172:144:185:30
        prefix-length: 64
      enabled: true
    vlan:
      base-iface: bond0
      id: 145
```

创建这个对象之后，它会做这几件事：

- 基于 ens161 和 ens256，创建一个 bond0 接口，模式为主备，并配置 IP 地址
- 基于 bond0，分别创建 vlan 子接口：bond0.144 和 bond0.145。并配置 IP 地址

对象创建后，可通过下面的方式检查是否配置完成及成功：

```
[root@10-6-185-30 ]# kubectl get nncp
NAME                STATUS         REASON
bond-vlan30     Available     SuccessfullyConfigured
```

这说明已经配置成功，再检查接口：

```
[root@10-6-185-30 ]# ip a s bond0.144
90: bond0.144@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdi
sc noqueue state UP group default qlen 1000
    link/ether 00:50:56:b4:e9:41 brd ff:ff:ff:ff:ff:ff
    inet 172.144.185.30/16 brd 172.144.255.255 scope global noprefixroute bond0.
144
       valid_lft forever preferred_lft forever
    inet6 fd00:144::172:144:185:30/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:feb4:e941/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

更多 example 使用例子请参考: https://github.com/nmstate/kubernetes-nmstate/tree/main/docs/examples

DPDK

本文主要介绍如何在 DCE 5.0 中快速创建第一个 DPDK 应用。

前置依赖

- 安装 Multus-underlay，并启用安装 SR-IOV 组件，参考安装
- 需要硬件支持：拥有支持 SR-IOV 系列的网卡并设置虚拟功能（VF），参考 SR-IOV
- 需要切换网卡驱动为用户态驱动

```
# 下载 dpdk 源码
root@master:~/cyclinder/sriov/# wget https://fast.dpdk.org/rel/dpdk-22.07.tar.xz && cd dpdk-22.07/usertools
root@master:~/cyclinder/sriov/dpdk-22.07/usertools# ./dpdk-devbind.py --status
root@172-17-8-120:~/cyclinder/sriov/dpdk-22.07/usertools# ./dpdk-devbind.py --status
Network devices using kernel driver
===================================
0000:01:00.0 'I350 Gigabit Network Connection 1521' if=eno1 drv=igb unused=vfio-pci
0000:01:00.1 'I350 Gigabit Network Connection 1521' if=eno2 drv=igb unused=vfio-pci
0000:01:00.2 'I350 Gigabit Network Connection 1521' if=eno3 drv=igb unused=vfio-pci
0000:01:00.3 'I350 Gigabit Network Connection 1521' if=eno4 drv=igb unused=vfio-pci
0000:04:00.0 'MT27800 Family [ConnectX-5] 1017' if=enp4s0f0np0 drv=mlx5_core unused=vfio-pci *Active*
0000:04:00.1 'MT27800 Family [ConnectX-5] 1017' if=enp4s0f1np1 drv=mlx5_core unused=vfio-pci *Active*
0000:04:00.2 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v
```

0 drv=mlx5_core unused=vfio-pci
0000:04:00.3 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v1 drv=mlx5_core unused=vfio-pci
0000:04:00.4 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v2 drv=mlx5_core unused=vfio-pci
0000:04:00.5 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v3 drv=mlx5_core unused=vfio-pci
0000:04:00.6 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v4 drv=mlx5_core unused=vfio-pci
0000:04:00.7 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v5 drv=mlx5_core unused=vfio-pci
0000:04:01.1 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v6 drv=mlx5_core unused=vfio-pci

以 0000:04:00.2 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v0 drv=mlx5_core unused=vfio-pci 为例：

- 0000:04:00.2：该 VF PCI 地址
- if=enp4s0f0v0：该 VF 网卡名称
- drv=mlx5_core：当前网卡驱动
- unused=vfio-pci：可切换的网卡驱动

DPDK 支持的用户态驱动有三种：

- vfio-pci：在启用 IoMMU 情况下，推荐使用此驱动，性能安全性最好
- igb-uio：适用性较 uio_pci_generic 更强，支持 SR-IOV VF，但需手动编译 module 并加载到内核
- uio_pci_generic：内核原生驱动，不兼容 SR-IOV VF，但支持在 VM 上使用

切换网卡驱动为 vfio-pci：

root@172-17-8-120:~/cyclinder/sriov/dpdk-22.07/usertools# ./dpdk-devbind.py --bind=vfio-pci 0000:04:01.1

查看绑定结果:

root@172-17-8-120:~/cyclinder/sriov/dpdk-22.07/usertools# ./dpdk-devbind.py --status

Network devices using DPDK-compatible driver
==
0000:04:01.1 'MT27800 Family [ConnectX-5 Virtual Function] 1018' drv=vfio-pci unused=mlx5_core

Network devices using kernel driver
===================================
0000:01:00.0 'I350 Gigabit Network Connection 1521' if=eno1 drv=igb unused=vfio-pci
0000:01:00.1 'I350 Gigabit Network Connection 1521' if=eno2 drv=igb unused=vfio-pci
0000:01:00.2 'I350 Gigabit Network Connection 1521' if=eno3 drv=igb unused=vfio-pci
0000:01:00.3 'I350 Gigabit Network Connection 1521' if=eno4 drv=igb unused=vfio-pci
0000:04:00.0 'MT27800 Family [ConnectX-5] 1017' if=enp4s0f0np0 drv=mlx5_core unused=vfio-pci *Active*
0000:04:00.1 'MT27800 Family [ConnectX-5] 1017' if=enp4s0f1np1 drv=mlx5_core unused=vfio-pci *Active*
0000:04:00.2 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v0 drv=mlx5_core unused=vfio-pci
0000:04:00.3 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v1 drv=mlx5_core unused=vfio-pci
0000:04:00.4 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v2 drv=mlx5_core unused=vfio-pci

0000:04:00.5 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v3 drv=mlx5_core unused=vfio-pci
0000:04:00.6 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v4 drv=mlx5_core unused=vfio-pci
0000:04:00.7 'MT27800 Family [ConnectX-5 Virtual Function] 1018' if=enp4s0f0v5 drv=mlx5_core unused=vfio-pci

0000:04:01.1：已经变为 vfio-pci 驱动

- 设置大页内存和开启 IoMMU（vfio-pci 驱动依赖 IOMMU 技术）：

  编辑 /etc/default/grub，在 GRUB_CMDLINE_LINUX 中加入以下内容：

  ```
  GRUB_CMDLINE_LINUX='default_hugepagesz=1GB hugepagesz=1GB hugepages=6 isolcpus=1-3 intel_iommu=on iommu=pt'
  update-grab && reboot
  ```

  !!! note

  更新上述配置，需要重启系统，重启系统前最好备份。
  如果不能更新配置，驱动需要切换为 igb-uio 驱动，需手动 build && insmod && modprobe，具体参考 https://github.com/atsgen/dpdk-kmod

配置 SRIOV-Device-Plugin

- 更新 SRIOV-Device-plugin 的 configmap：新建资源池 sriov_netdevice_dpdk，让其能够找到支持 dpdk 的 VF：

  ```
  kubectl edit cm -n kube-system sriov-0.1.1-config
  apiVersion: v1
  data:
    config.json: |-
      {
        "resourceList":
        [{
          "resourceName": "sriov_netdevice",
  ```

```
        "resourcePrefix": "intel.com",
        "selectors": {
          "device": ["1018"],
          "vendors": ["15b3"],
          "drivers": ["mlx5_core"],
          "pfNames": []
        }
      },{
        "resourceName": "sriov_netdevice_dpdk",
        "resourcePrefix": "intel.com",
        "selectors": {
          "drivers": ["vfio-pci"]
        }
      }]
    }
```

新增 sriov_netdevice_dpdk。注意 selectors 中 driver 指定 vfio-pci 后将重启 sriov-device-plugin。

```
kubectl delete po -n kube-system -l app=sriov-dp
```

等待重启完成，查看 Node 是否加载 sriov_netdevice_dpdk 资源：

```
kubectl describe nodes 172-17-8-120
...
Allocatable:
  cpu:                                24
  ephemeral-storage:                  881675818368
  hugepages-1Gi:                      6Gi
  hugepages-2Mi:                      0
  intel.com/sriov_netdevice:          6
  intel.com/sriov_netdevice_dpdk:     1   # 这里显示表示已经可用了
```

- 创建 Multus DPDK CRD：

```
cat EOF | kubectl apply -f -
> apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    helm.sh/hook: post-install
    helm.sh/resource-policy: keep
    k8s.v1.cni.cncf.io/resourceName: intel.com/sriov_netdevice_dpdk
    v1.multus-underlay-cni.io/coexist-types: '["default"]'
    v1.multus-underlay-cni.io/default-cni: "false"
    v1.multus-underlay-cni.io/instance-type: sriov_dpdk
    v1.multus-underlay-cni.io/underlay-cni: "true"
    v1.multus-underlay-cni.io/vlanId: "0"
  name: sriov-dpdk-vlan0
  namespace: kube-system
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "sriov-dpdk",
      "type": "sriov",
      "vlan": 0
    }
> EOF
```

创建 DPDK 测试 Pod

```
cat << EOF | kubectl apply -f -
> apiVersion: v1
kind: Pod
metadata:
  name: dpdk-demo
  annotations:
```

```
    k8s.v1.cni.cncf.io/networks: kube-system/sriov-dpdk-vlan0
spec:
  containers:
  - name: sriov-dpdk
    image: docker.io/bmcfall/dpdk-app-centos
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /etc/podnetinfo
      name: podnetinfo
      readOnly: false
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      requests:
        memory: 1Gi
        #cpu: "4"
        intel.com/sriov_netdevice_dpdk: '1'
      limits:
        hugepages-1Gi: 2Gi
        #cpu: "4"
        intel.com/sriov_netdevice_dpdk: '1'
    # Uncomment to control which DPDK App is running in container.
    # If not provided, l3fwd is default.
    #   Options: l2fwd l3fwd testpmd
    env:
    - name: DPDK_SAMPLE_APP
      value: "testpmd"
    #
    # Uncomment to debug DPDK App or to run manually to change
    # DPDK command line options.
    command: ["sleep", "infinity"]
```

```
  volumes:
  - name: podnetinfo
    downwardAPI:
      items:
        - path: "labels"
          fieldRef:
            fieldPath: metadata.labels
        - path: "annotations"
          fieldRef:
            fieldPath: metadata.annotations
  - name: hugepage
    emptyDir:
      medium: HugePages
> EOF
```

等待 Pod Running，然后进入 Pod 中：

```
root@172-17-8-120:~# kubectl exec -it sriov-pod-2 sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a futu
re version. Use kubectl exec [POD] -- [COMMAND] instead.
sh-4.4# dpdk-app
ENTER dpdk-app:
 argc=1
 dpdk-app
E1031 08:17:36.431877      116 resource.go:31] Error getting cpuset info: open /p
roc/116/root/sys/fs/cgroup/cpuset/cpuset.cpus: no such file or directory
E1031 08:17:36.432266      116 netutil_c_api.go:119] netlib.GetCPUInfo() err: open
 /proc/116/root/sys/fs/cgroup/cpuset/cpuset.cpus: no such file or directory
Couldn't get CPU info, err code: 1
  Interface[0]:
    IfName=""  Name="kube-system/k8s-pod-network"  Type=SR-IOV
    MAC=""  IP="10.244.5.197"  IP="fd00:10:244:0:eb50:e529:8533:7884"
    PCIAddress=0000:04:01.1
```

```
  Interface[1]:
    IfName="net1"   Name="kube-system/sriov-dpdk-vlan0"   Type=SR-IOV
    MAC=""


 myArgc=14
 dpdk-app -n 4 -l 1 --master-lcore 1 -w 0000:04:01.1 -- -p 0x1 -P --config
="(0,0,1)" --parse-ptype
```

dpdk-app 会打印出当前 Pod 的相关信息，包括 eth0 的 IP、MAC 和 type 等。 其中值得注意: net1 网卡没有任何 IP 和 MAC 等网络信息，这符合 DPDK 的特性，不需要内核网络协议栈也能工作。

创建子网及 IP 池

本页介绍在创建工作负载使用 IP 池之前，如何创建子网及 IP 池。

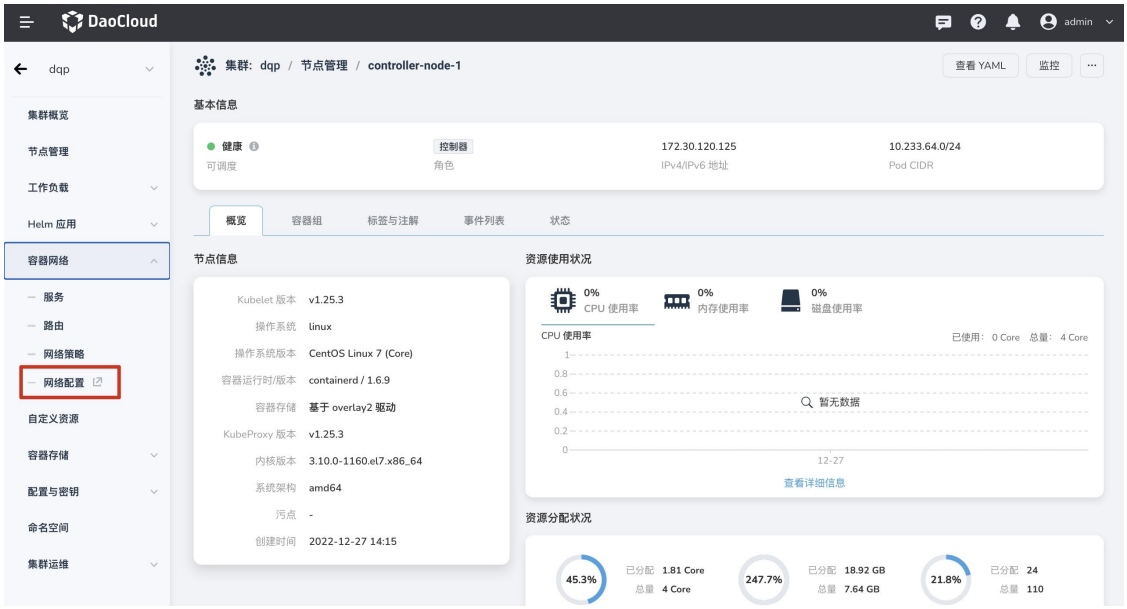
- 如部署 SpiderPool 组件时已创建子网及 IP 池，可直接使用 IP 池。
- 如需要创建新的子网及 IP 池可参考此文档。

前提条件

- SpiderPool 已成功部署。
- Multus 搭配 Macvlan/SR-IOV 已成功部署。
- 创建子网及 IP 池前建议做好子网和 IP 规划，并充分了解 IP 池的使用方式。

界面操作

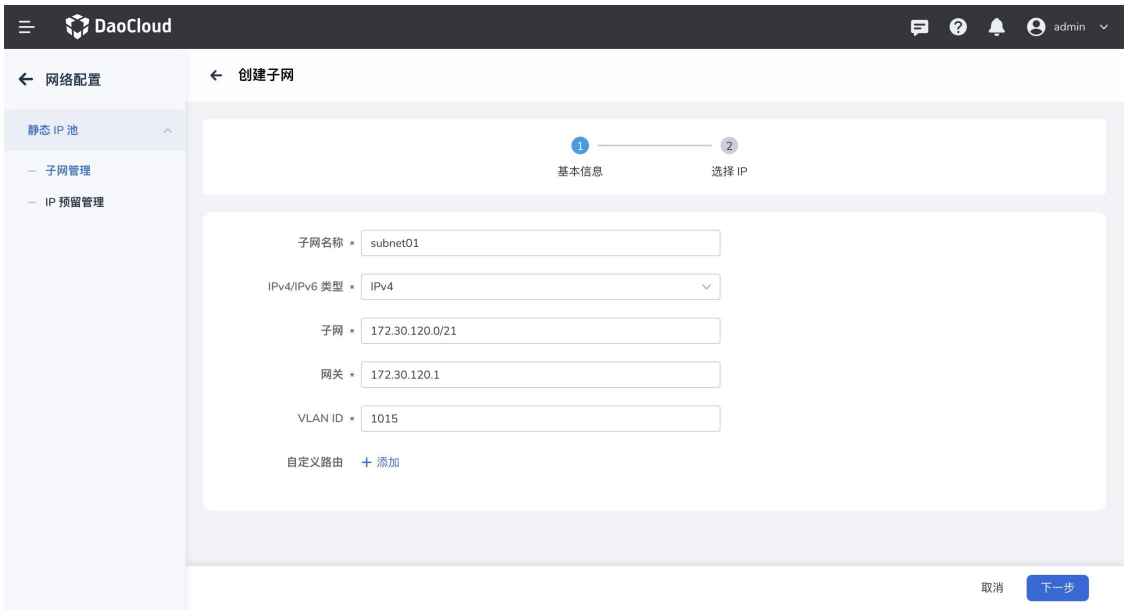
1. 登录 DCE UI 后，在左侧导航栏点击 容器管理 —> 集群列表，找到对应集群。然后在左侧导航栏点击容器网络 —> 网络配置。



网络配置

2. 进入网络配置，确认待使用子网是否已创建。

- 如待使用子网已默认创建，可直接创建 IP 池。
- 如待使用子网没有默认创建，可进入页面点击创建子网。

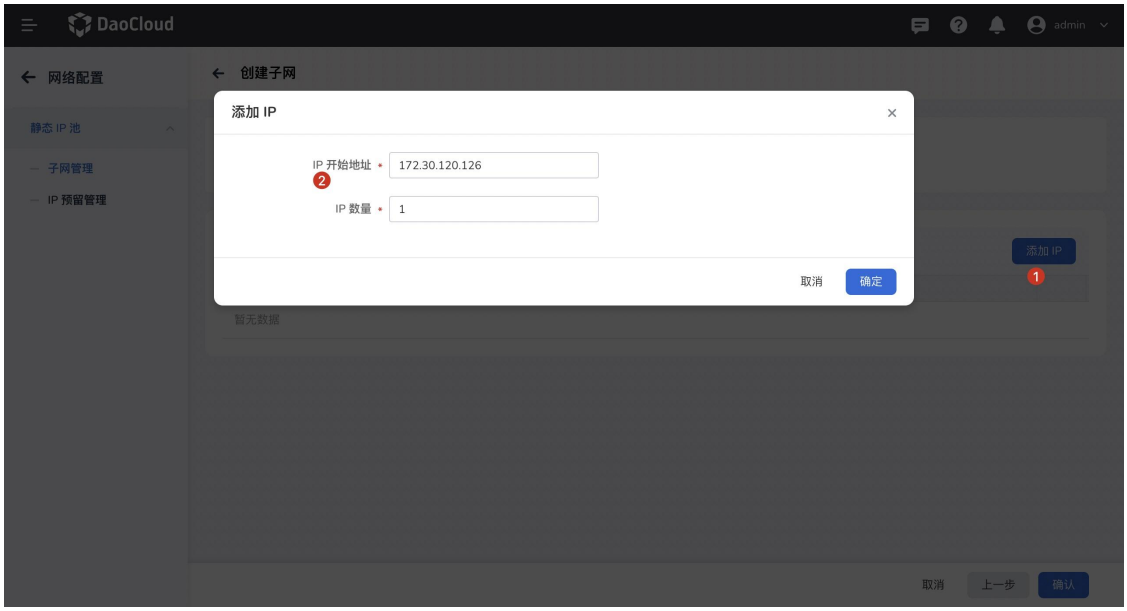


创建子网

参数说明：

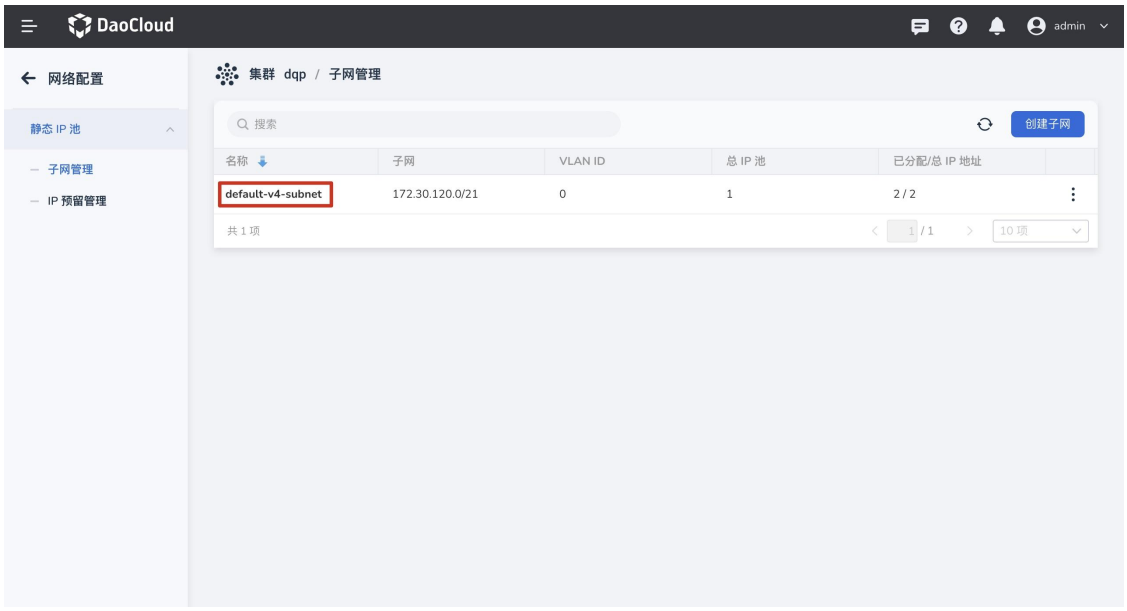
- IPv4/IPv6 类型：待创建子网的子网类型。
- 子网：已经规划好的子网段。如配合 Macvlan 使用，请提前同网络管理员确认对应网络接口/子接口所对应的网段。
- 网关：子网对应网关，请提前同网络同事确认。
- VLAN ID：子网所对应的 VLAN ID。

3. 点击下一步进入 IP 选择，输入待使用的 IP 段（输入上述子网内 IP）。点击确定，完成子网创建。



创建子网

4. 点击待使用的子网名称，进入子网详情页面。



子网详情

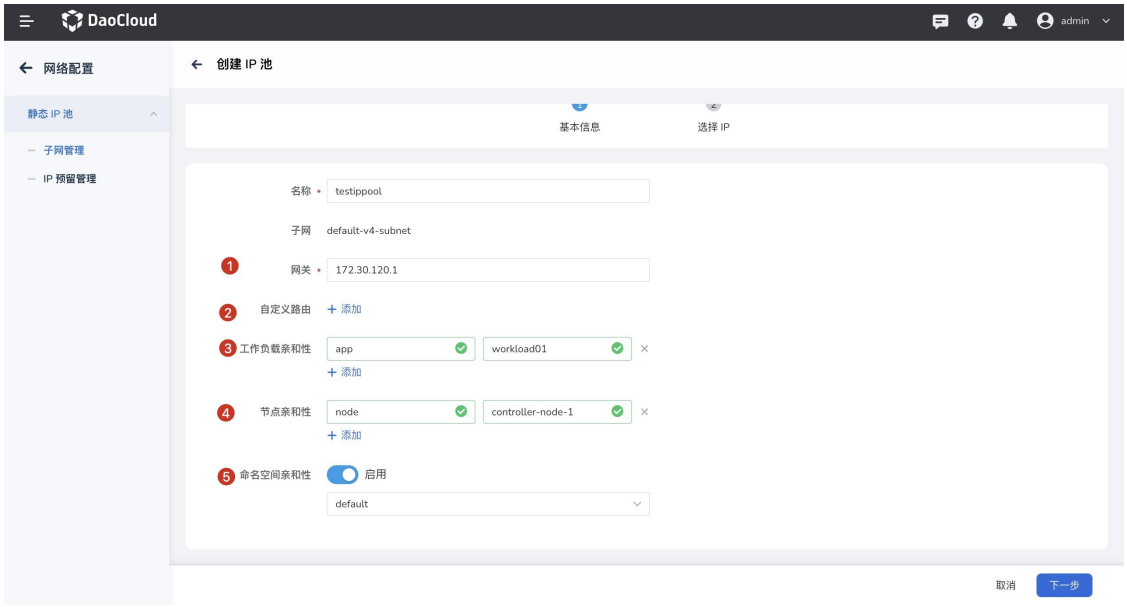
5. 在子网详情页，点击创建 IP 池。

!!! note

创建 IP 池为可选步骤，可根据需要进行创建。

– 如需要对 IP 进行严管控，可提前完成 IP 池创建。
– 如粗粒度管控 IP 资源，可不用提前创建 IP 池。

进入创建页面，输入如下参数：



创建 IP 池

- 网关：网关默认继承子网网关，可修改。
- 自定义路由：当用户有特殊路由需求时，可基于 IP 池粒度的进行自定义。
- Multus 实例：可以指定从哪个 IP 池为 Pods 分配 IP 地址，或者定义当 Pod 连接到特定网络时应使用哪个 IP 池。
- 工作负载亲和性：工作负载标签（如 app: workload01）。IP 池创建后，仅可被对应的工作负载选择，实现固定 IP 池效果。
- 节点亲和性：节点标签（如 node:controller–1）。IP 池创建后，工作负载 Pod 需要调度到对应节点才能使用创建的 IP 池。

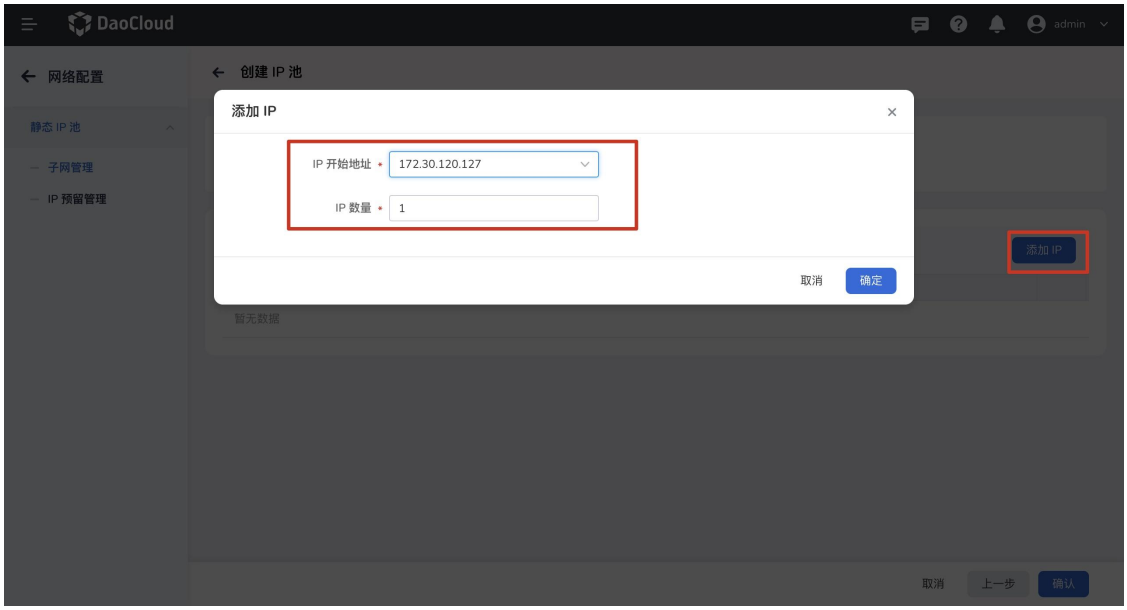
- 命名空间亲和性：开启后可选择对应命名空间。选择后，只有对应命名空间中的工作负载可使用创建的 IP 池。

!!! note

如果创建时不添加任何亲和性，创建后的 IP 池为`共享 IP 池`。

6. 点击 IP 池名称，然后点击添加 IP 选择 IP 开始地址以及加入 IP 池中的 IP 数量，点击确定，完成 IP 添加，再次点击完成 IP 池创建。

   获取 IP 规则：从 IP 开始地址依次获取输入对应数量的 IP，IP 段如不是连续的 IP，则跳过中间断档 IP，依次往后获取。



添加 IP

7. 创建完成后工作负载即可使用 IP 池。

YAML 创建

也可以直接通过 YAML 来创建子网和 IPPool。

YAML 创建子网

```
apiVersion: spiderpool.spidernet.io/v1
kind: SpiderSubnet
```

```
metadata:
  name: default-v4-subnet
spec:
  gateway: 172.30.120.1
  ipVersion: 4
  ips:
  - 172.30.120.126-172.30.120.127   # 已规划子网 IP，可输入 IP 段
      # 如 72.30.120.126-172.30.120.127 段或单个 IP 如：172.30.120.126
  subnet: 172.30.120.0/21
  vlan: 0
```

YAML 创建 IPPool

```
apiVersion: spiderpool.spidernet.io/v1
kind: SpiderIPPool
metadata:
  name: standard-ipv4-ippool
spec:
  ipVersion: 4
  subnet: 172.30.120.0/21
  ips:
  - 172.30.120.126-172.30.120.127        # 已添加至子网内的 IP，  可输入 IP 段
      # 如 172.30.120.126-172.30.120.127 段或单个 IP 如：172.30.120.126
```

阿里云环境运行

本页主要介绍如何使用 Spiderpool 在阿里云环境运行，并如何实现一套完整的 Underlay 解决方案

背景

当前公有云厂商众多，如：阿里云、华为云、腾讯云、AWS 等，但当前开源社区的主流 CNI 插件难以以 Underlay 网络方式运行其上，只能使用每个公有云厂商的专有 CNI 插件，没有统一的公有云 Underlay 解决方案。尤其是在混合云场景下，统一的 CNI 方案能够便于多云管理。Spiderpool 是一种适用于任意的公有云环境中的 Underlay 网络解决方案。

Spiderpool 针对阿里云存在的局限性提供的解决方案

Spiderpool 有节点拓扑、解决 MAC 地址合法性、对接基于 spec.externalTrafficPolicy 为 Local 模式的 service 等功能。Spiderpool 能基于 IPVlan Underlay CNI 运行在公有云环境上，它的实现原理如下：

1. 公有云下使用 Underlay 网络，但公有云的每个云服务器的每张网卡只能分配有限的 IP 地址，当应用运行在某个云服务器上时，需要同步获取到 VPC 网络中分配给该云服务器不同网卡的合法 IP 地址，才能实现通信。根据上述分配 IP 的特点，Spiderpool 的 CRD：SpiderIPPool 可以设置 nodeName，multusName 实现节点拓扑的功能，通过 IP 池与节点、IPvlan Multus 配置的亲和性，能最大化的利用与管理节点可用的 IP 地址，给应用分配到合法的 IP 地址，让应用在 VPC 网络内自由通信，包括 Pod 与 Pod 通信，Pod 与云服务器通信等。

2. 公有云的 VPC 网络中，处于网络安全管控和数据包转发的原理，当网络数据报文中出现 VPC 网络未知的 MAC 和 IP 地址时，它无法得到正确的转发。例如，基于 Macvlan 和 OVS 原理的 Underlay CNI 插件，Pod

网卡中的 MAC 地址是新生成的，会导致 Pod 无法通信。针对该问题，Spiderpool 可搭配 IPVlan CNI 进行解决。IPVlan 基于三层网络，无需依赖二层广播，并且不会重新生成 Mac 地址，与父接口保持一致，因此通过 IPvlan 可以解决公有云中关于 MAC 地址合法性的问题。

3. 在 service 将 .spec.externalTrafficPolicy 设置为 Local，可以保留客户端源 IP ，但公有云自建集群在这种模式下使用平台的 Loadbalancer 组件进行 nodePort 转发时，会出现访问不通。针对该问题 Spiderpool 提供了 coordinator 插件，该插件通过 iptables 在数据包中打标记，确认从 veth0 进入的数据的回复包仍从 veth0 转发，进而解决在该模式下 nodeport 访问不通的问题。

实施要求

- 使用 IPVlan 做集群 CNI 时，系统内核版本必须大于 4.2。
- 已安装 Helm。

实施步骤

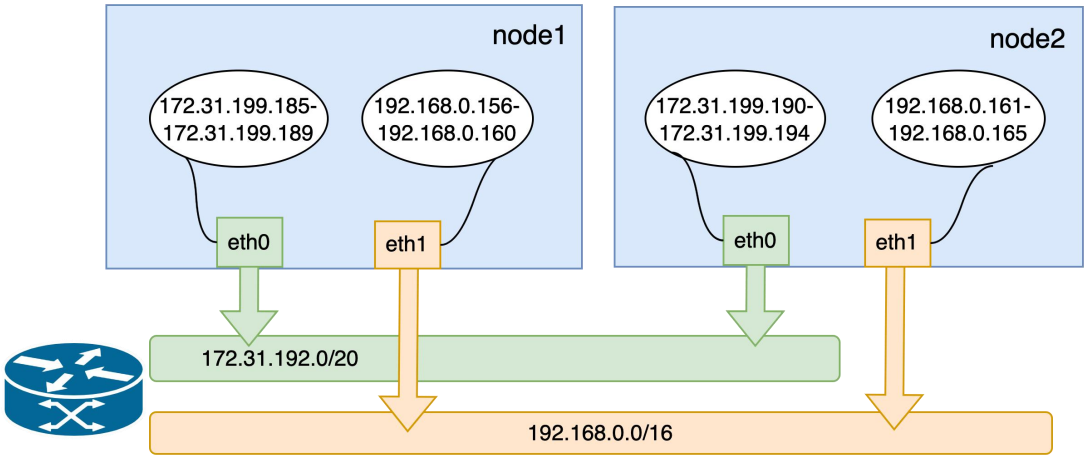
准备阿里云环境

- 准备一个阿里云环境，给虚拟机分配 2 个网卡，每张网卡均分配一些辅助私网 IP，如图：



alicloud-web-network

- 使用上述配置的虚拟机，搭建一套 Kubernetes 集群，节点的可用 IP 及集群网络拓扑图如下：



网络拓扑

安装 Spiderpool

通过 helm 安装 Spiderpool。

```
helm repo add spiderpool https://spidernet-io.github.io/spiderpool

helm repo update spiderpool

helm install spiderpool spiderpool/spiderpool --namespace kube-system --set ipam.enableStatefulSet=false --set multus.multusCNI.defaultCniCRName="ipvlan-eth0"
```

!!! note

- 如果您使用的是中国大陆的云厂商服务器，可以指定参数 `--set global.imageRegistryOverride=ghcr.m.daocloud.io` ，以帮助您更快的拉取镜像。

- Spiderpool 可以为控制器类型为：`Statefulset` 的应用副本固定 IP 地址。在公有云的 Underlay 网络场景中，云主机只能使用限定的 IP 地址，当 StatefulSet 类型的应用副本漂移到其他节点，但由于原固定的 IP 在其他节点是非法不可用的，新的 Pod 将出现网络不可用的问题。对此场景，将 `ipam.enableStatefulSet` 设置为 `false`，禁用该功能。

- 通过 `multus.multusCNI.defaultCniCRName` 指定集群的 Multus clusterNetwork，clusterNetwork 是 Multus 插件的一个特定字段，用于指定 Pod 的默认网络接口。

安装 CNI 配置

Spiderpool 为简化书写 JSON 格式的 Multus CNI 配置，它提供了 SpiderMultusConfig CR 来自动管理 Multus NetworkAttachmentDefinition CR。如下是创建 IPvlan SpiderMultusConfig 配置的示例：

```
IPVLAN_MASTER_INTERFACE0="eth0"
IPVLAN_MULTUS_NAME0="ipvlan-$IPVLAN_MASTER_INTERFACE0"
IPVLAN_MASTER_INTERFACE1="eth1"
IPVLAN_MULTUS_NAME1="ipvlan-$IPVLAN_MASTER_INTERFACE1"

cat <<EOF | kubectl apply -f -
```

```
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderMultusConfig
metadata:
  name: ${IPVLAN_MULTUS_NAME0}
  namespace: kube-system
spec:
  cniType: ipvlan
  coordinator:
    mode: underlay
    tunePodRoutes: true
    podCIDRType: cluster
  enableCoordinator: true
  ipvlan:
    master:
    - ${IPVLAN_MASTER_INTERFACE0}
---
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderMultusConfig
metadata:
  name: ${IPVLAN_MULTUS_NAME1}
  namespace: kube-system
spec:
  cniType: ipvlan
  coordinator:
    mode: underlay
    tunePodRoutes: true
    podCIDRType: cluster
  enableCoordinator: true
  ipvlan:
    master:
    - ${IPVLAN_MASTER_INTERFACE1}
EOF
```

在本文示例中，使用如上配置，创建如下的两个 IPvlan SpiderMultusConfig，将基于它们自动生成的 Multus NetworkAttachmentDefinition CR，它们分别对应了宿主机的 eth0 与 eth1 网卡。

```
~# kubectl get spidermultusconfigs.spiderpool.spidernet.io -n kube-system
NAME              AGE
ipvlan-eth0     10m
ipvlan-eth1     10m

~# kubectl get network-attachment-definitions.k8s.cni.cncf.io -n kube-system
NAME              AGE
ipvlan-eth0     10m
ipvlan-eth1     10m
```

创建 IPPool

Spiderpool 的 CRD：SpiderIPPool 提供了 nodeName、multusName 与 ips 字段：

- nodeName：当 nodeName 不为空时，Pod 在某个节点上启动，并尝试从 SpiderIPPool 分配 IP 地址，若 Pod 所在节点符合该 nodeName，则能从该 SpiderIPPool 中成功分配出 IP，若 Pod 所在节点不符合 nodeName，则无法从该 SpiderIPPool 中分配出 IP。当 nodeName 为空时，Spiderpool 对 Pod 不实施任何分配限制。
- multusName：Spiderpool 通过该字段与 Multus CNI 深度结合以应对多网卡场景。当 multusName 不为空时，SpiderIPPool 会使用对应的 Multus CR 实例为 Pod 配置网络，若 multusName 对应的 Multus CR 不存在，那么 Spiderpool 将无法为 Pod 指定 Multus CR。当 multusName 为空时，Spiderpool 对 Pod 所使用的 Multus CR 不作限制。
- spec.ips：该字段的值必须设置。由于阿里云限制了节点可使用的 IP 地址，故该值的范围必须在 nodeName 对应主机的辅助私网 IP 范围内，您可以从阿里云的弹性网卡界面获取。

依据如上所述，使用如下的 Yaml，为每个节点的每张网卡(eth0、eth1)分别创建了一个 SpiderIPPool，它们将为不同节点上的 Pod 提供 IP 地址。

```
~# cat <<EOF | kubectl apply -f -
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: master-172
spec:
  default: true
  ips:
  - 172.31.199.185-172.31.199.189
  subnet: 172.31.192.0/20
  gateway: 172.31.207.253
  nodeName:
  - master
  multusName:
  - kube-system/ipvlan-eth0
---
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: master-192
spec:
  default: true
  ips:
  - 192.168.0.156-192.168.0.160
  subnet: 192.168.0.0/24
  gateway: 192.168.0.253
  nodeName:
  - master
  multusName:
```

```
  - kube-system/ipvlan-eth1
---
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: worker-172
spec:
  default: true
  ips:
  - 172.31.199.190-172.31.199.194
  subnet: 172.31.192.0/20
  gateway: 172.31.207.253
  nodeName:
  - worker
  multusName:
  - kube-system/ipvlan-eth0
---
apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: worker-192
spec:
  default: true
  ips:
  - 192.168.0.161-192.168.0.165
  subnet: 192.168.0.0/24
  gateway: 192.168.0.253
  nodeName:
  - worker
  multusName:
  - kube-system/ipvlan-eth1
EOF
```

创建应用

以下的示例 Yaml 中，会创建 2 组 DaemonSet 应用和 1 个 type 为 ClusterIP 的 service ，其中：

- v1.multus-cni.io/default-network：用于指定应用所使用的子网，示例中的应用分别使用了不同的子网。

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: test-app-1
  name: test-app-1
  namespace: default
spec:
  selector:
    matchLabels:
      app: test-app-1
  template:
    metadata:
      labels:
        app: test-app-1
      annotations:
        v1.multus-cni.io/default-network: kube-system/ipvlan-eth0
    spec:
      containers:
      - image: docker.io/centos/tools
        imagePullPolicy: IfNotPresent
        name: test-app-1
        ports:
        - name: http
          containerPort: 80
```

```
            protocol: TCP
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: test-app-2
  name: test-app-2
  namespace: default
spec:
  selector:
    matchLabels:
      app: test-app-2
  template:
    metadata:
      labels:
        app: test-app-2
      annotations:
        v1.multus-cni.io/default-network: kube-system/ipvlan-eth1
    spec:
      containers:
      - image: docker.io/centos/tools
        imagePullPolicy: IfNotPresent
        name: test-app-2
        ports:
        - name: http
          containerPort: 80
          protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: test-svc
  labels:
    app: test-app-1
spec:
  type: ClusterIP
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: test-app-1
EOF
```

查看 Pod 运行状态：

```
~# kubectl get po -owide
NAME                              READY   STATUS    RESTARTS   AGE   IP
              NODE       NOMINATED NODE   READINESS GATES
test-app-1-b7765b8d8-422sb        1/1     Running   0          16s   172.31.199.
187   master     <none>           <none>
test-app-1-b7765b8d8-qjgpj        1/1     Running   0          16s   172.31.199.1
93    worker     <none>           <none>
test-app-2-7c56876fc6-7brhf       1/1     Running   0          12s   192.168.0.16
0     master     <none>           <none>
test-app-2-7c56876fc6-zlxxt       1/1     Running   0          12s   192.168.0.16
1     worker     <none>           <none>
```

Spiderpool 自动为应用分配 IP 地址，应用的 IP 均在期望的 IP 池内：

```
~# kubectl get spiderippool
NAME            VERSION   SUBNET              ALLOCATED-IP-COUNT   TOTA
L-IP-COUNT   DEFAULT
master-172      4         172.31.192.0/20     1                    5
  true
```

```
master-192    4    192.168.0.0/24    1    5
  true
worker-172    4    172.31.192.0/20   1    5
 true
worker-192    4    192.168.0.0/24    1    5
  true
```

测试集群东西向连通性

- 测试 Pod 与宿主机的通讯情况：

```
~# kubectl get nodes -owide
NAME     STATUS   ROLES           AGE    VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION                CONTAINER-RUNTIME
master   Ready    control-plane   2d12h  v1.27.3   172.31.199.183   <none>        CentOS Linux 7 (Core)   6.4.0-1.el7.elrepo.x86_64     containerd://1.7.1
worker   Ready    <none>          2d12h  v1.27.3   172.31.199.184   <none>        CentOS Linux 7 (Core)   6.4.0-1.el7.elrepo.x86_64     containerd://1.7.1

~# kubectl exec -ti test-app-1-b7765b8d8-422sb -- ping 172.31.199.183 -c 2
PING 172.31.199.183 (172.31.199.183): 56 data bytes
64 bytes from 172.31.199.183: seq=0 ttl=64 time=0.088 ms
64 bytes from 172.31.199.183: seq=1 ttl=64 time=0.054 ms

--- 172.31.199.183 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.054/0.071/0.088 ms
```

- 测试 Pod 与跨节点、跨子网 Pod 的通讯情况

```
~# kubectl exec -ti test-app-1-b7765b8d8-422sb -- ping 172.31.199.193
 -c 2
PING 172.31.199.193 (172.31.199.193): 56 data bytes
64 bytes from 172.31.199.193: seq=0 ttl=64 time=0.460 ms
64 bytes from 172.31.199.193: seq=1 ttl=64 time=0.210 ms

--- 172.31.199.193 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.210/0.335/0.460 ms

~# kubectl exec -ti test-app-1-b7765b8d8-422sb -- ping 192.168.0.161
-c 2
PING 192.168.0.161 (192.168.0.161): 56 data bytes
64 bytes from 192.168.0.161: seq=0 ttl=64 time=0.408 ms
64 bytes from 192.168.0.161: seq=1 ttl=64 time=0.194 ms

--- 192.168.0.161 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.194/0.301/0.408 ms
```

- 测试 Pod 与 ClusterIP 的通讯情况：

```
~# kubectl get svc test-svc
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
test-svc   ClusterIP   10.233.23.194   <none>        80/TCP    26s

~# kubectl exec -ti test-app-2-7c56876fc6-7brhf -- curl 10.233.23.194
-I
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 21 Jul 2023 06:45:56 GMT
```

Content-Type: text/html

Content-Length: 4086

Last-Modified: Fri, 21 Jul 2023 06:38:41 GMT

Connection: keep-alive

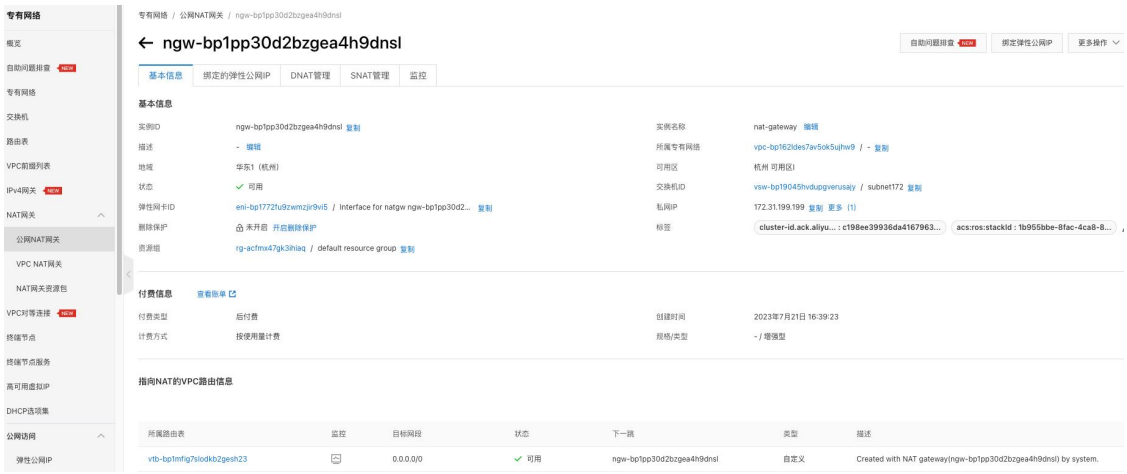
ETag: "64ba27f1-ff6"

Accept-Ranges: bytes

测试集群南北向连通性

集群内的 Pod 流量出口访问

- 阿里云的 NAT 网关能实现为 VPC 环境下构建一个公网或私网流量的出入口。通过 NAT 网关，实现集群的流量出口访问。参考 NAT 网关文档 创建 NAT 网关，如图：



alicloud-natgateway

- 测试集群内 Pod 的流量出口访问

```
~# kubectl exec -ti test-app-2-7c56876fc6-7brhf -- curl www.baidu.com -I
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Connection: keep-alive
```

```
Content-Length: 277
Content-Type: text/html
Date: Fri, 21 Jul 2023 08:42:17 GMT
Etag: "575e1f60-115"
Last-Modified: Mon, 13 Jun 2016 02:50:08 GMT
Pragma: no-cache
Server: bfe/1.0.8.18
```

负载均衡流量入口访问

部署 Cloud Controller Manager

CCM（Cloud Controller Manager）是阿里云提供的一个用于 Kubernetes 与阿里云基础产品进行对接的组件，本文中通过该组件结合阿里云基础设施完成负载均衡的流量入口访问。参考下列步骤与 CCM 文档 完成 CCM 的部署。

1. 集群节点配置 providerID

   务必在集群中的每个节点上，分别执行如下命令，从而获取每个节点各自的 providerID。http://100.100.100.200/latest/meta-data 是阿里云 CLI 提供获取实例元数据的 API 入口，在下列示例中无需修改它。更多用法可参考实例元数据

   ```
   ~# META_EP=http://100.100.100.200/latest/meta-data
   ~# provider_id=`curl -s $META_EP/region-id`.`curl -s $META_EP/instance-id`
   ```

   在集群的 master 节点通过 kubectl patch 命令为集群中的 每个节点 补充各自的 providerID，该步骤必须被执行，否则对应节点的 CCM Pod 将无法运行。

   ```
   ~# kubectl get nodes
   ~# kubectl patch node ${NODE_NAME} -p '{"spec":{"providerID": "${provider_id}"}}'
   ```

2. 创建阿里云的 RAM 用户，并授权。

RAM 用户是 RAM 中的一种实体身份，代表需要访问阿里云的人员或应用程序。通过参阅 RAM 访问控制创建 RAM 用户，并授于需要访问资源的权限。

为确保后续步骤中所使用的 RAM 用户具备足够的权限，请与本文保持一致，给予 RAM 用户 AdministratorAccess 和 AliyunSLBFullAccess 权限。

3. 获取 RAM 用户的 AccessKey & AccessKeySecret

   登录 RAM 用户，访问 用户中心，获取对应 RAM 用的 AccessKey & AccessKeySecret。

4. 创建 CCM 的 Cloud ConfigMap。

   将步骤 3 获取的 AccessKey & AccessKeySecret，参考下列方式写入环境变量。

```
~# export ACCESS_KEY_ID=LTAI**********
~# export ACCESS_KEY_SECRET=HAeS**********
```

   执行如下命令，完成创建 cloud-config。

```
accessKeyIDBase64=`echo -n "$ACCESS_KEY_ID" |base64 -w 0`
accessKeySecretBase64=`echo -n "$ACCESS_KEY_SECRET"|base64 -w 0
`


cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: cloud-config
  namespace: kube-system
data:
  cloud-config.conf: |-
    {
```

```
        "Global": {
            "accessKeyID": "$accessKeyIDBase64",
            "accessKeySecret": "$accessKeySecretBase64"
        }
    }
EOF
```

5. 获取 Yaml ，并通过 kubectl apply –f cloud–controller–manager.yaml 方式安装 CCM，本文中安装的版本为 v2.5.0

   – 使用如下命令，获取 cloud–controller–manager.yaml，并替换其中 <<cluster_cidr>> 为您真实集群的 cluster cidr 。

```
~# wget https://raw.githubusercontent.com/spidernet–io/spiderpool/main/docs/example/alicloud/cloud–controller–manager.yaml
~# kubectl apply –f cloud–controller–manager.yaml
```

6. 检查 CCM 安装完成。

```
~# kubectl get po –n kube–system | grep cloud–controller–manager
NAME                                READY   STATUS    RESTARTS   AGE
cloud–controller–manager–72vzr      1/1     Running   0          27s
cloud–controller–manager–k7jpn      1/1     Running   0          27s
```

为应用创建 Loadbalancer 负载均衡访问入口

如下的 Yaml 将创建 spec.type 为 LoadBalancer 的 2 组 service，一组为 tcp（四层负载均衡），一组为 http （七层负载均衡）。

- service.beta.kubernetes.io/alibaba–cloud–loadbalancer–protocol–port：CCM 提供的创建七层负载均衡注解。可以通过它自定义暴露端口。更多用法参考 CCM 使用文档 。

- .spec.externalTrafficPolicy：表示此 Service 是否希望将外部流量路由到节点本地或集群范围的端点。它有两个可用选项：Cluster（默认）和 Local。将.spec.externalTrafficPolicy 设置为 Local，可以保留客户端源 IP。

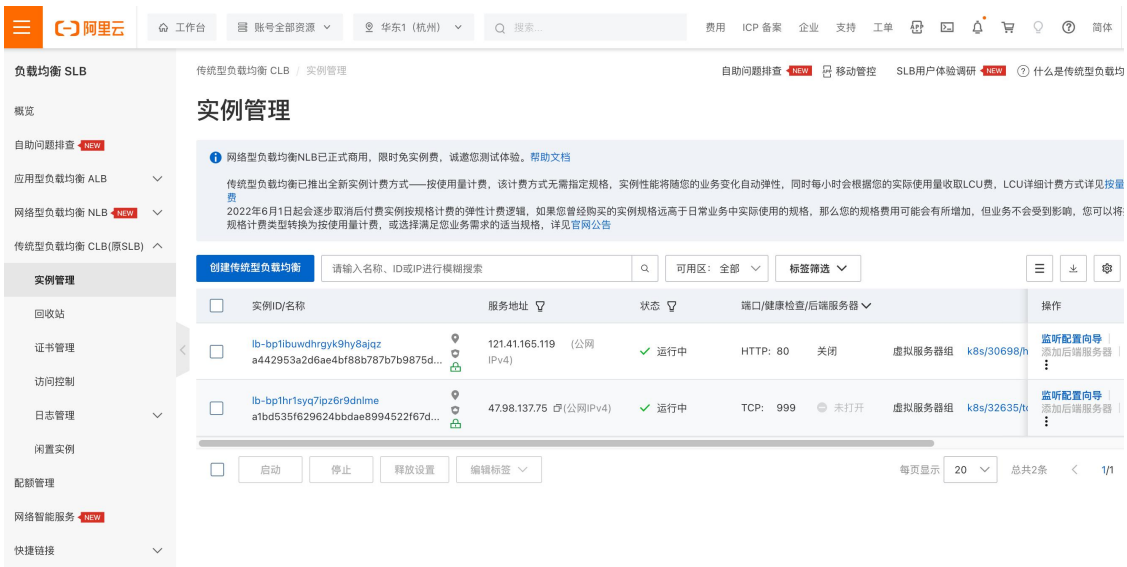
```
~# cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: tcp-service
  namespace: default
spec:
  externalTrafficPolicy: Local
  ports:
  - name: tcp
    port: 999
    protocol: TCP
    targetPort: 80
  selector:
    app: test-app-2
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-protocol-port: "http:80
"
  name: http-service
  namespace: default
spec:
  externalTrafficPolicy: Local
```

```
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: test-app-2
  type: LoadBalancer
EOF
```

创建完成后，您可以查看到如下内容：

```
~# kubectl get svc |grep service
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)
          AGE
http-service        LoadBalancer   10.233.1.108    121.41.165.119   80:30698/TCP
 11s
tcp-service         LoadBalancer   10.233.4.245    47.98.137.75     999:32635/TCP
    15s
```

CCM 将自动在 IaaS 层创建四层与七层的负载均衡器，可以通过阿里云界面进行查看，如下：



alicloud-loadbalancer

验证负载均衡流量入口访问

在公网的机器上，通过负载均衡器的公网 IP + 端口实现流量入口访问

```
# 访问四层负载均衡
$ curl 47.98.137.75:999 -I
HTTP/1.1 200 OK
Server: nginx/1.25.1
Date: Sun, 30 Jul 2023 09:12:46 GMT
Content-Type: text/html
Content-Length: 615
Last-Modified: Tue, 13 Jun 2023 15:08:10 GMT
Connection: keep-alive
ETag: "6488865a-267"
Accept-Ranges: bytes

# 访问七层负载均衡
$ curl 121.41.165.119:80 -I
HTTP/1.1 200 OK
Date: Sun, 30 Jul 2023 09:13:17 GMT
Content-Type: text/html
Content-Length: 615
Connection: keep-alive
Last-Modified: Tue, 13 Jun 2023 15:08:10 GMT
ETag: "6488865a-267"
Accept-Ranges: bytes
```

总结

Spiderpool 能够运行在阿里云集群中，并且可以保证集群的东西向与南北向流量均正常。

工作负载使用 IP 池

本章节主要介绍结合 Multus 及 Underlay CNI 插件，为工作负载 Pod 配置多网卡，并通过 Spiderpool 进行 Underlay 网络的 IP 的分配和固定。主要介绍内容如下：

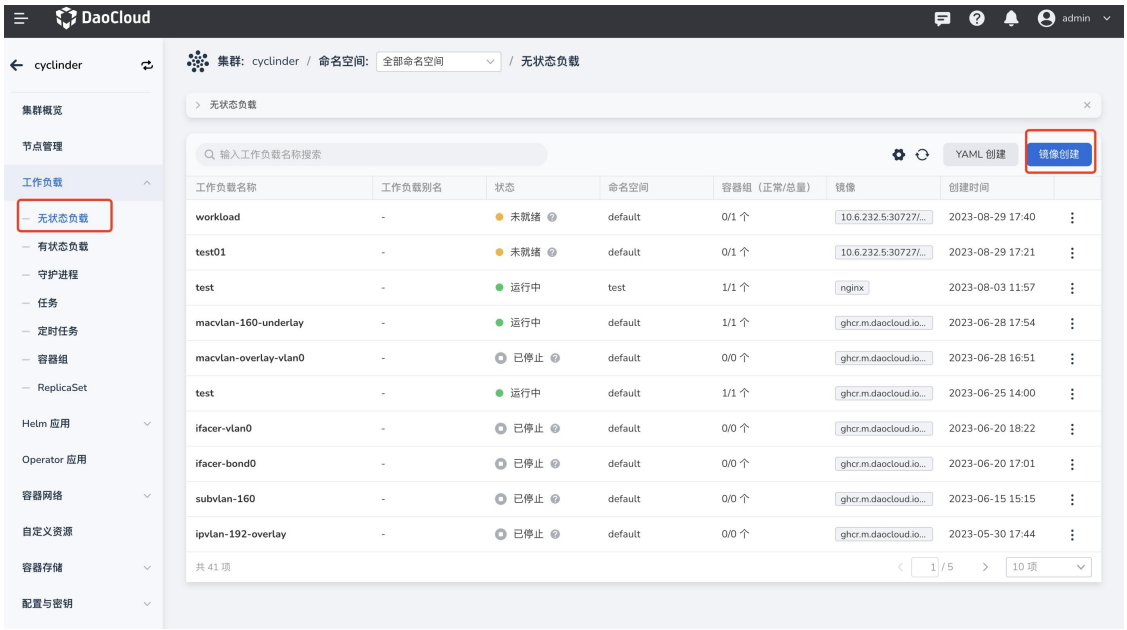
1. Pod 设置多容器网卡
2. 工作负载使用 IP 池
3. 工作负载使用固定 IP 池
4. 工作负载使用自动创建固定 IP 池
5. 工作负载使用默认 IP 池

前提条件

1. SpiderPool 已成功部署。
2. Multus 搭配 Macvlan/SR-IOV 已成功部署。
3. 如使用手动选择 IP 池，请提前完成创建 IP 子网和 IP 池。如使用自动创建固定 IP 池，请提前完成创建 IP 池。
4. 如果使用默认 IP 池，请提前完成创建 IP 子网和 IP 池。并在容器网络 Multus CNI 配置中，配置好带有默认 IP 池的网卡。

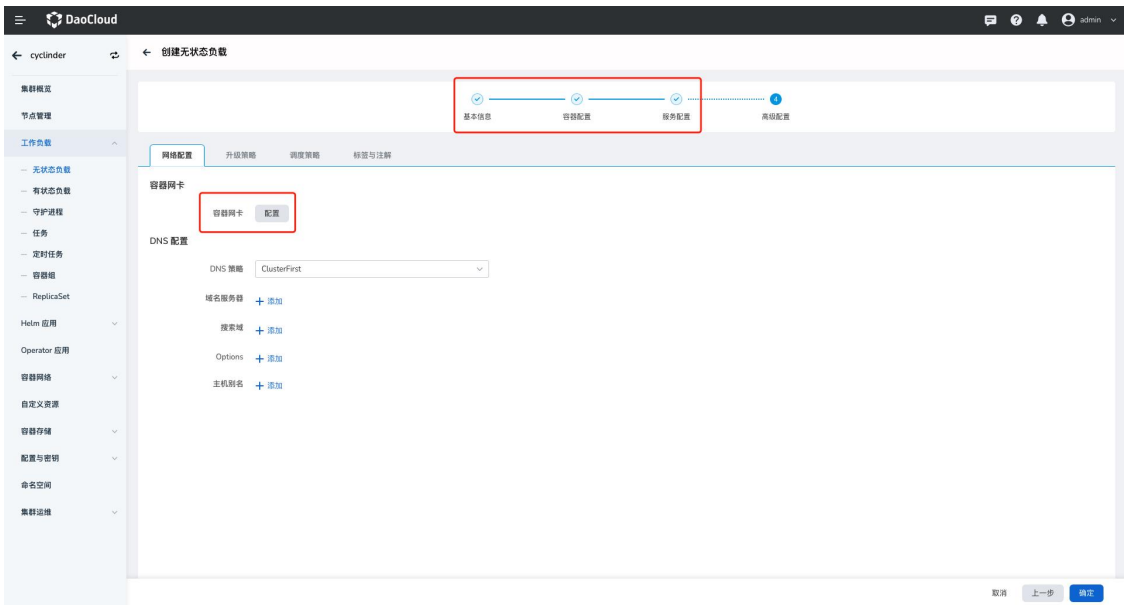
界面操作

1. 登录平台 UI，在左侧导航栏点击容器管理->集群列表，找到对应集群。然后，在左侧导航栏选择无状态负载，点击镜像创建。



镜像创建

2. 在创建无状态负载页面，完成基本信息、容器配置、服务配置页面的信息输入。然后，进入高级配置，点击配置容器网卡。



容器网卡

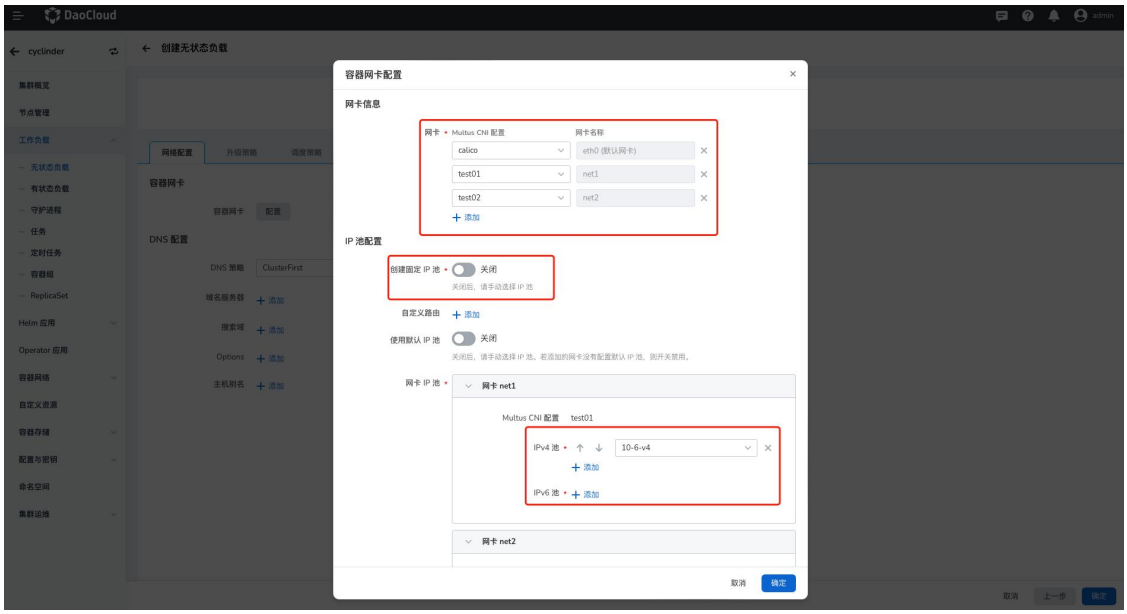
3. 进入容器网卡配置页面，完成以下参数的配置：

- 网卡信息：若创建的应用容器需要使用多张网卡（如一张东西向通信，一张南北向通信），可以添加多网卡。
    - eth0（默认网卡）：默认为 Overlay CNI，Calico/Cilium。
    - net1：可选择 Underlay CNI 配置，如 Macvlan/SR-IOV，本文示例为 Macvlan。
- IP 池配置：Underlay CNI IP 分配的规则。
    - 创建固定 IP 池： 开启后，只需要为新增的容器网卡（net1、net2、net3）选择对应子网，工作负载在部署时会自动创建固定 IP 池，部署后容器网卡仅能使用此 IP 池中的地址。
    - 弹性 IP：开启后，IP 池中的 IP 数量会根据设置的弹性 IP 数量变动。最大可用 IP 数等于 Pod 副本数 + 弹性 IP 数量。Pod 扩容时 IP 池也随之进行扩容。
    - 自定义路由：当应用创建有特殊路由需求时，可添加自定义路由。
    - 网卡 IP 池：选择对应网卡待使用的子网或对应 IP 池。
    - 使用默认 IP 池：开启后，会为新增的容器网卡（eth0、net1、net2）全部选择好默认的 IP 池。

工作负载使用 IP 池有如下两种方式，两种方式的使用场景及流程差异可参考：IP 池的使用说明

手动选择已有的 IP 池

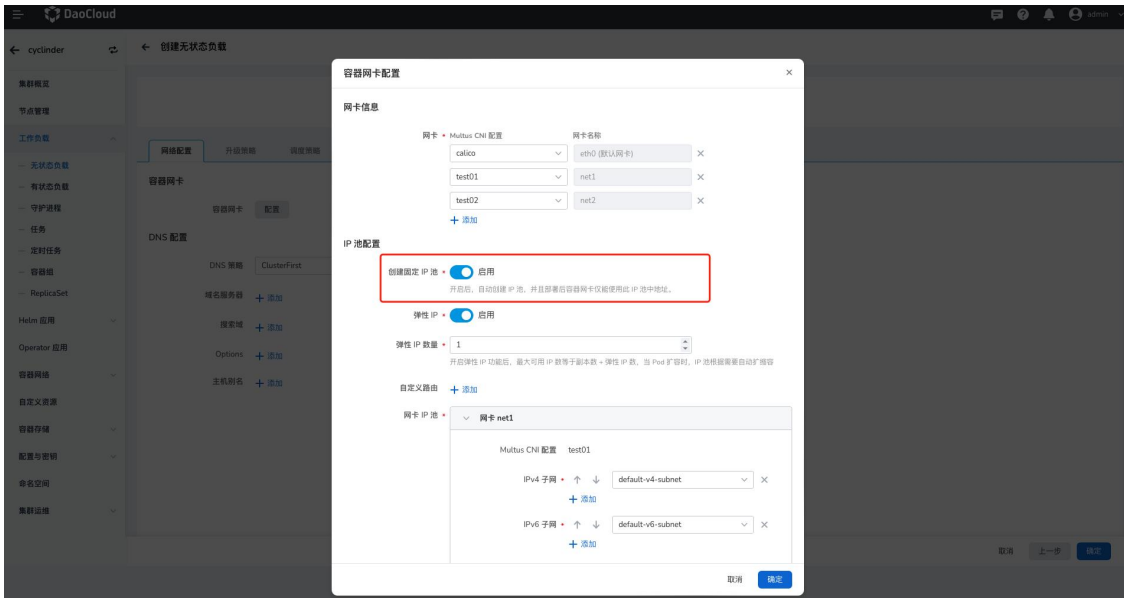
手动选择 IP 池需要提前创建 IP 池，可选择 IP 池范围为：共享 IP 池，添加了当前应用亲和性的 IP 池，添加了当前命名空间亲和性的 IP 池。



手动选择

自动创建固定 IP 池

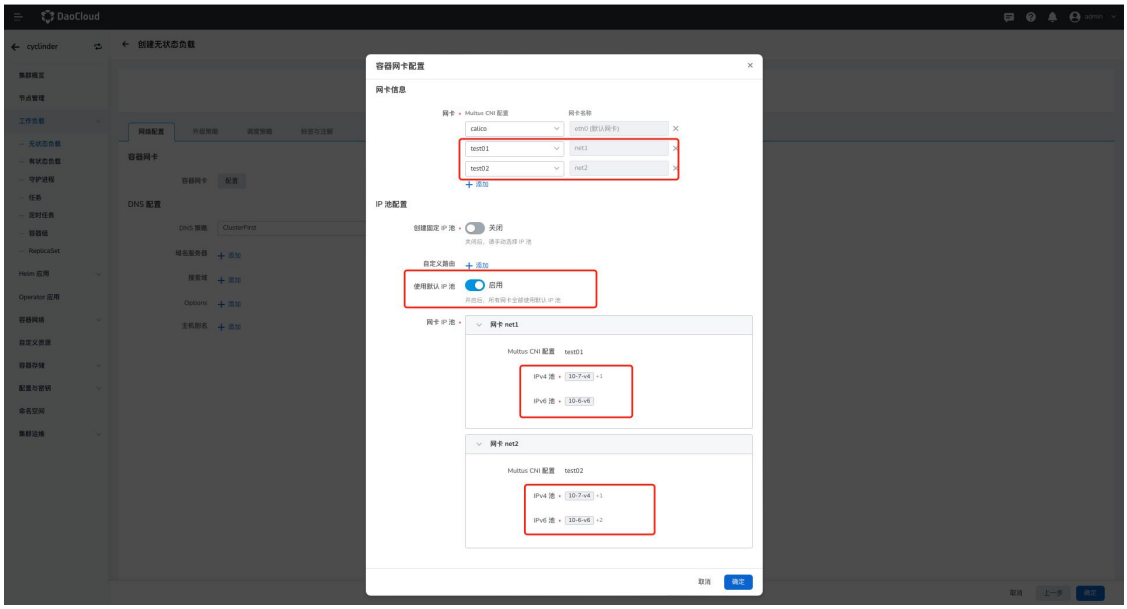
仅需要选择对应的子网，即可自动创建固定 IP 池。



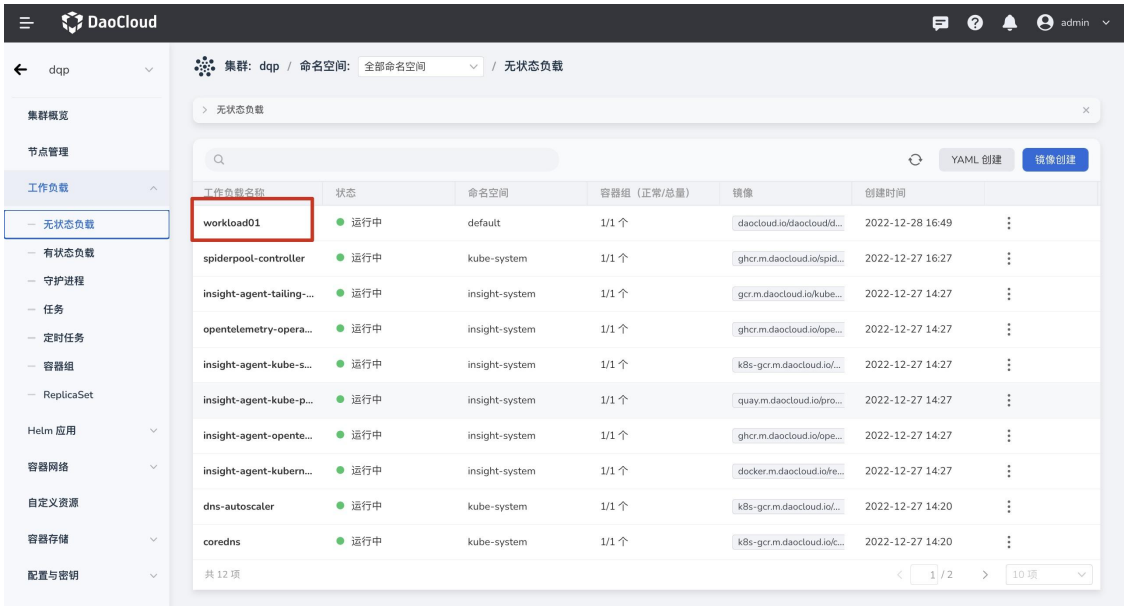
自动创建

使用默认 IP 池

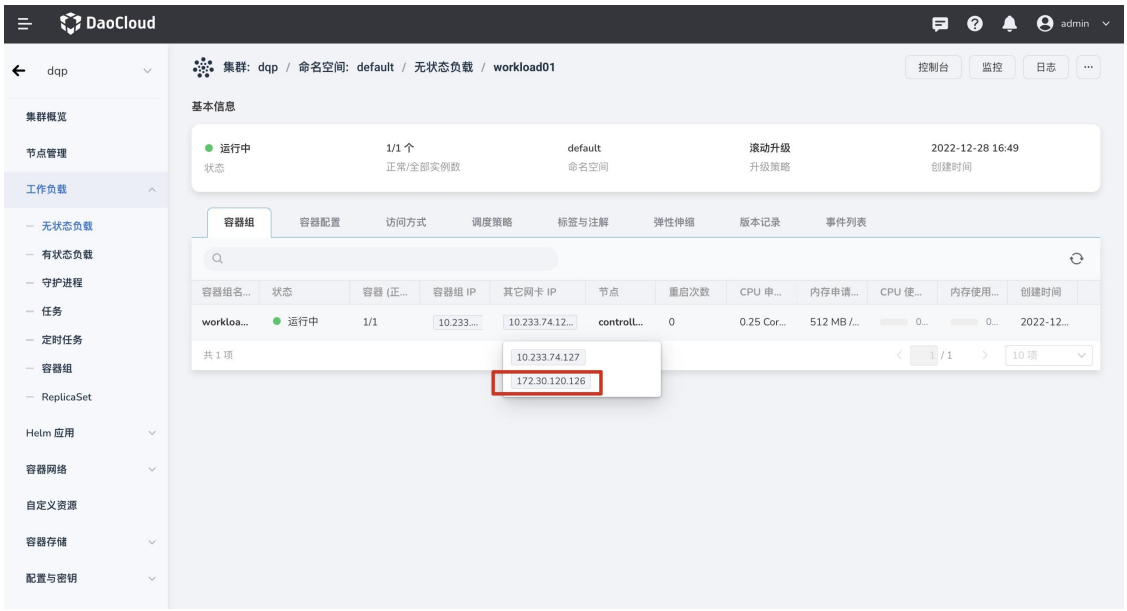
提前创建好 IP 池，并在 Multus CNI 配置中，选择带有默认 IP 池的网卡，即可使用默认 IP 池功能。



4. 创建完工作负载后，可点击对应工作负载 workload01 查看工作负载 Pod 使用的 IP。



工作负载 IP



工作负载 IP

YAML 使用

1. 使用 Pod 注解 ipam.spidernet.io/ippool 选择从 IP 池 testingippool 分配 IP，创建以下 Deployment：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workload01
spec:
  replicas: 3
  selector:
    matchLabels:
      app: workload01
  template:
    metadata:
      annotations:
        ipam.spidernet.io/ippool: |-
          {
```

```
            "ipv4": ["testingippool"]
          }
      labels:
        app: workload01
    spec:
      containers:
      - name: workload01
        image: busybox
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh", "-c", "trap : TERM INT; sleep infinity & wait"]
```

2. 由 Deployment workload01 所控制的 Pod 均从 IP 池 testingippool 分配 IP 地址且成功运行。

```
kubectl get se
NAME                                                INTERFACE   IPV4POOL
          IPV4                  IPV6POOL   IPV6   NODE                CRE
ATETION TIME
workload01-6967dcd8df-8b6zp   eth0        standard-ipv4-ippool   172.
18.41.47/24                          spider-worker   7s
standard-ippool-deploy-6967dcd8df-cvq79   eth0        standard-ipv4-i
ppool   172.18.41.50/24                          spider-worker   7s
standard-ippool-deploy-6967dcd8df-s58x9   eth0        standard-ipv4-i
ppool   172.18.41.41/24                          spider-worker   7s
```

第三方工作负载使用 IP 池

本章节主要介绍结合 Multus 及 Underlay CNI 插件，为自定义工作负载（本文采用 OpenKruise 控制器 创建的工作负载 CloneSet） Pod 配置 IP Pool，通过 Spiderpool 进行 Underlay 网络的 IP 的分配和固定。

前提条件

1. SpiderPool 已成功部署。
2. Multus 搭配 Macvlan/SR-IOV 已成功部署。
3. 如使用手动选择 IP 池，请提前完成创建 IP 子网和 IP 池。本示例中需要使用固定 IP 池，请提前完成创建 固定 IP 池。

操作步骤

使用已创建固定 IP 池

1. 部署自定义资源 CloneSet，并在 Annotation 中指定默认网络类型，VLAN ID ，子网接口，IP 池信息

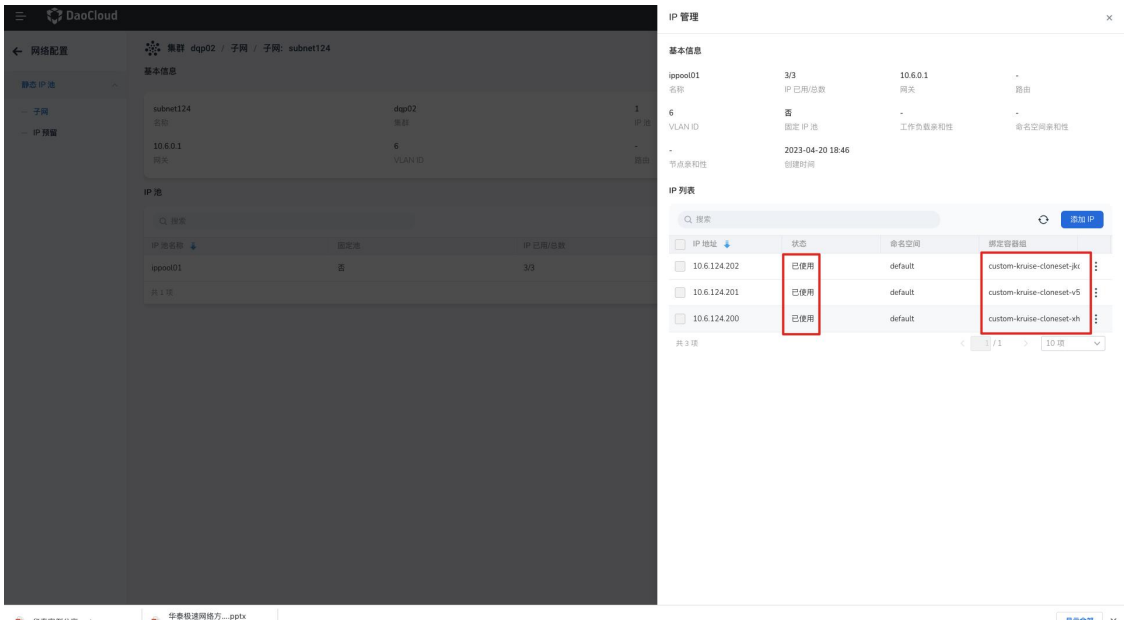
```
v1.multus-cni.io/default-network: kube-system/calico
k8s.v1.cni.cncf.io/networks: kube-system/vlan6
ipam.spidernet.io/ippools: '[{"interface":"net1","ipv4":["ippool01"]}]'

apiVersion: apps.kruise.io/v1alpha1
kind: CloneSet
metadata:
  name: custom-kruise-cloneset
spec:
  replicas: 1
  selector:
    matchLabels:
      app: custom-kruise-cloneset
  template:
```

```
    metadata:
      annotations:
        v1.multus-cni.io/default-network: "kube-system/calico"
        k8s.v1.cni.cncf.io/networks: "kube-system/vlan6"
        ipam.spidernet.io/ippool: |-
          {
            "interface":"net1", "ipv4":["ippool01"]
          }
      labels:
        app: custom-kruise-cloneset
    spec:
      containers:
      - name: custom-kruise-cloneset
        image: busybox
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh", "-c", "trap : TERM INT; sleep infinity & wa
it"]
```

2. 部署后，查看 CloneSet 状态

```
kubectl get pods -A|grep kruise-clone
default              custom-kruise-cloneset-jkqtx
                1/1     Running             0                   44h
default              custom-kruise-cloneset-v5qz7
                1/1     Running          0                   44h
default              custom-kruise-cloneset-xhtq6
                1/1     Running
0                    44h
```

3. 进入容器平台->选择对应集群->点击容器网络，然后找到对应子网->进入子网详情， 查看 IP 使用情况：



自动创建固定 IP 池

1. 如需要使用子网自动创建固定 IP 池，请在创建 自定义工作负载时，添加如下 Annotation:

```
apiVersion: apps.kruise.io/v1alpha1
kind: CloneSet
metadata:
  name: custom-kruise-cloneset03
spec:
  replicas: 3
  selector:
    matchLabels:
      app: custom-kruise-cloneset03
  template:
    metadata:
      annotations:
        v1.multus-cni.io/default-network: "kube-system/calico" # 指定默认容器网卡
        k8s.v1.cni.cncf.io/networks: "kube-system/vlan6" #指定 Multus CR
```

D 实例（NetworkAttachmentDefinition）

```
        ipam.spidernet.io/subnet: |-     #指定使用固定 IP 池的网卡及待使用子网
          {"interface":"net1","ipv4": ["subnet124"]}
        ipam.spidernet.io/ippool-ip-number: "1" #指定弹性 IP 数量，可使用 IP 数=弹性 IP 数+Replica 数
      labels:
        app: custom-kruise-cloneset03
    spec:
      containers:
      - name: custom-kruise-cloneset03
        image: busybox
        imagePullPolicy: IfNotPresent
        command: ["/bin/sh", "-c", "trap : TERM INT; sleep infinity & wait"]
```

2. 部署后查看 Clonset 状态：

```
kubectl get pods -A|grep kruise-clone03
```

3. 查看 IP Pool IP 状态：

```
kubectl get sp -oyaml | grep kruise
      ipam.spidernet.io/application: apps.kruise.io/v1alpha1:CloneSet:default:custom-kruise-cloneset03
    name: auto-custom-kruise-cloneset03-v4-net1-f3114156804d
    allocatedIPs: '{"10.6.124.200":{"interface":"net1","pod":"default/custom-kruise-cloneset-r7xjd","podUid":"43942169-3c43-4a81-aaae-60ba1dc9d07e"},"10.6.124.201":{"interface":"net1","pod":"default/custom-kruise-cloneset-sp5t6","podUid":"4980a045-7ee9-467a-a6a3-b259411963cb"},"10.6.124.202":{"interface":"net1","pod":"default/custom-kruise-cloneset-j94tl","podUid":"72b13a85-5275-44b1-8491-323f9fff3571"}}'
```

配置参考

安装 Spiderpool 之后，请在 /etc/cni/net.d/ 目录编辑 CNI 配置文件。

以下是 MacVLAN CNI 的配置示例。

```
{
  "cniVersion": "0.3.1",
  "type": "macvlan",
  "mode": "bridge",
  "master": "eth0",
  "name": "macvlan-cni-default",
  "ipam": {
    "type": "spiderpool"
  }
}
```

常用命令

本页说明如何在 CLI 命令行使用 spiderpoolctl 程序进行调试。

- spiderpoolctl gc

  触发对 spiderpool-controller 的垃圾回收请求。

  --address string [optional] address for spider-controller (default to service address)

- spiderpoolctl ip show

  显示正在使用此 IP 的 Pod。

  --ip string [required] ip

- spiderpoolctl ip release

  释放一个 IP。

  --ip string [optional] ip
  --force [optional] force release ip

- spiderpoolctl ip set

  设定一个 Pod 要使用的 IP。这将更新 ippool 和工作负载端点资源。

  --ip string [required] ip
  --pod string [required] pod name
  --namespace string [required] pod namespace
  --containerid string [required] pod container id
  --node string [required] the node name who the pod locates

--interface string [required] pod interface who taking effect the ip

什么是 Spiderpool

Spiderpool 是一个 IP 地址管理（IPAM）插件，可为容器云平台分配 IP 地址。大部分的 Overlay CNI 都具备符合功能特性的 IPAM 组件，SpiderPool 的主要设计目标是搭配 Underlay CNI （例如 MacVLAN CNI、 VLAN CNI、 IPVLAN CNI 等）进行 IP 的精细化管理。

Spiderpool 适用于任何能够对接第三方 IPAM 的 CNI 插件，尤其适合于一些缺失 IPAM 的 CNI， 包括 SR-IOV、 MacVLAN、 IPVLAN、 OVS-CNI 等。而一些 overlay CNI 自带了 IPAM 组件，对 Spiderpool 需求相对较低。

目前 SpiderPool 支持的能力如下：

功能	描述
多途径的 IP Pool 的使用方式	1. IP Pool 可通过 Node Selector 同命名空间、节点和应用进行关联使用。 2. 应用部署时可在 Annotation 中指定 IP Pool，按照优先级依次获取 IP Pool 中的 IP 资源。
IP Pool 的节点亲和性	同一集群内接入的节点可能属于不同数据中心，或者属于不同子网，因而同一个应用不同副本进行调度时，需要分配不同子网下的 IP 地址。IP Pool 的节点亲和性支持此场景。
IP Pool 的命	基于 IP Pool 的命名空间亲和性，同一 IP Pool 可以同时共享给多个 Namespace 使用。

功能	描述
名空间亲和性	
备用 IP Pool	当 IP Pool 中 IP 地址分配完毕后，并且对应的子网已无可用 IP，可新建子网和 IP Pool，并指定给应用使用，防止应用扩容失败。
应用固定 IP	自动化创建应用固定 IP 池并选择固定 IP 范围。
应用使用默认 IP 池	默认 IP 池，供那些没有特定 IP 需求的工作负载使用。
防止 IP 地址分配冲突的机制	1. IP Pool 中 IP 地址错开，IP Pool 间地址不重叠。 2. 严格管控 IP Pool 的增删改查，规避 IP 重叠。 3. 预留 IP 机制，可将已被集群外部节点使用的 IP 冻结，防止 IP 冲突。
防止 IP 地址泄露的回收机制	在 Pod 故障、重启、重建等场景下，清理 IP 资源被一些“僵尸 Pod”占用的垃圾数据，规避可用 IP 减少问题。 在 overlay IPAM 场景下，因为 CIDR 范围很大，所以该问题并不突出。而在 underlay 场景下，IP 资源有限，且部分应用有固定 IP 地址范围的需求，该问题就会影响应用的健康运行。
双栈支持	支持 ipv4-only、ipv6-only 和 dual-stack。

功能	描述
Statefulset 支持	Statefulset Pod 在重启、重建场景下，持续获取到相同的 IP 地址。
Pod 多网卡支持	配合 Multus，实现对多网卡场景下的 IP 分配支持。
预留 IP	预留 IP 机制，可将已被集群外部节点使用的 IP 冻结，防止 IP 冲突。
多层级路由定制	支持子网 CIDR、IP Pool、应用级别的自定义路由，路由优先级由低至高。
丰富的 Metrics	提供丰富的监控指标，保障集群 IP 资源监控。

IP 池的使用说明

本页介绍在使用 Spiderpool 进行 IP 分配以及管理时，IP 池的不同使用场景。使用前请确保 SpiderPool 已正确安装。

工作负载使用固定 IP

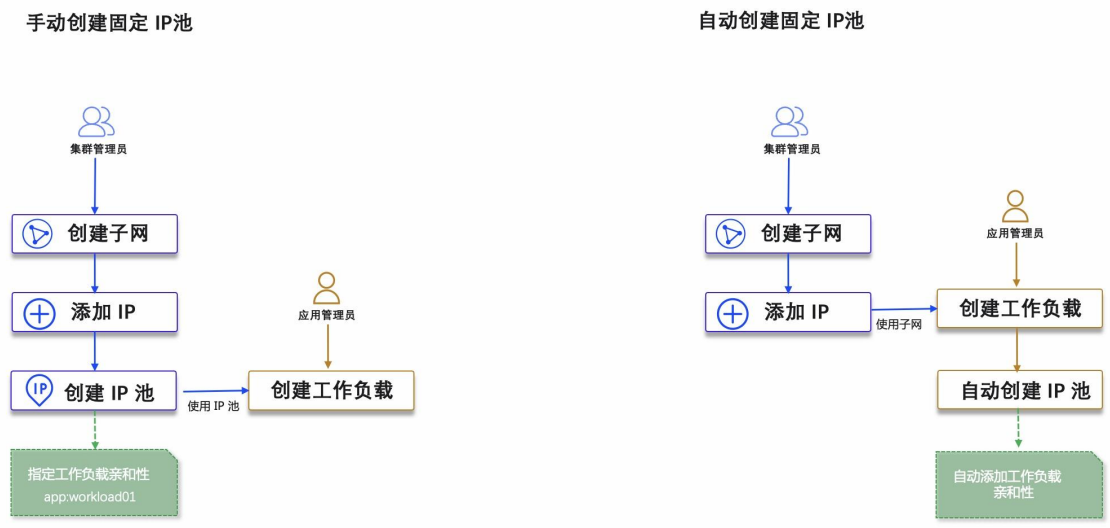
工作负载使用固定 IP 有 2 种使用方式：

- 手动创建固定 IP 池，并指定待使用 IP Pool 的工作负载亲和性，创建工作负载时选择对应的固定 IP Pool。

  适用场景：此方式适用于 IP 强管控场景，需要提前申请开通防火墙进行 IP 放行，放行后工作负载才能使用对应的固定 IP 。操作方式可参考：创建子网及 IP 池 及工作负载使用 IP 池。

- 自动创建固定 IP 池，创建子网并添加待使用 IP 后，应用管理员直接基于创建后的子网自动创建固定 IP Pool，创建后的 IP Pool 仅供此应用负载独享。

  适用场景：此方式适用于 IP 粗粒度管理场景，可以基于较大范围 IP（如：10.6.124.10~10.6.124.200）范围进行防火墙放行，放行后工作负载可基于此 IP 段自动获取 IP 并创建对应的固定 IP 池。操作方式可参考创建子网及 IP 池及工作负载使用 IP 池。



fixedippool

工作负载使用默认 IP 池

默认 IP 池是预先定义的、具有特定 IP 地址范围的 IP 资源集合。管理员需要手动设置和配置默认 IP 池，由于 IP 地址是预先分配的，所以这种方式更具确定性。这些 IP 地址可以为特定的工作负载或应用进行分配。

适用场景：此方式适用于 IP 强管控场景，如安全性要求高或需要与外部系统（如防火墙）进行配合的场景。操作方式可参考：创建子网及 IP 池 及工作负载使用 IP 池。

IP 池的节点亲和性

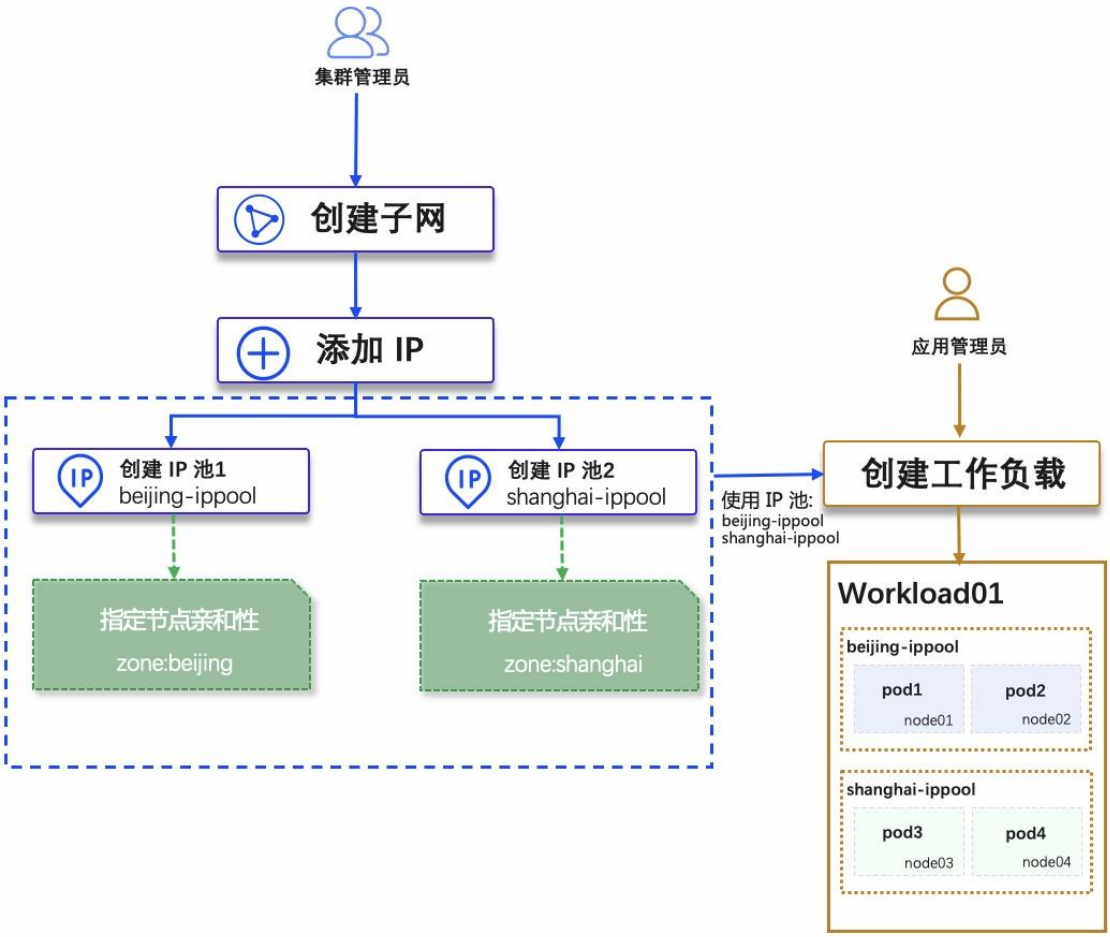
适用场景：

当集群节点为跨子网或跨数据中心时，不同的 Node 上可用的子网不相同，例如：

- 同一数据中心内，集群接入的节点分属不同子网
- 单个集群中，节点跨越了不同的数据中心

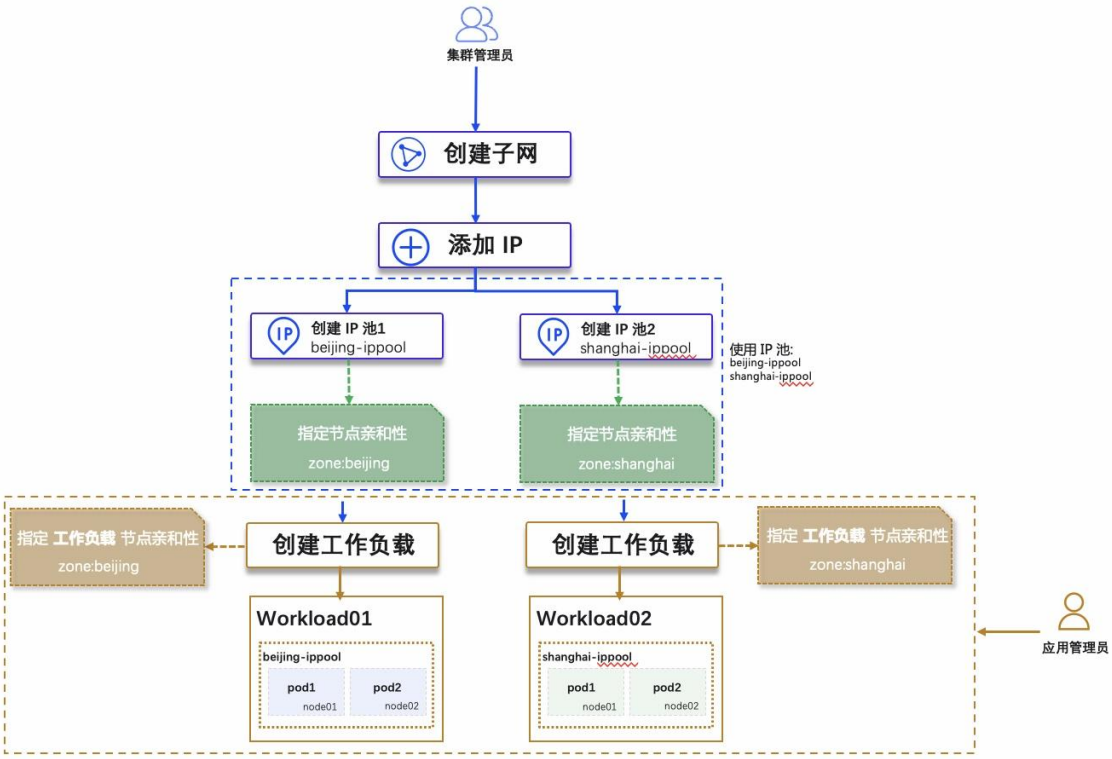
同时工作负载创建时：

- 同一工作负载需要调度至 **跨子网** 或 **跨数据中心** 节点上，使用流程如下：



nodeaffinity01

- 不同工作负载需要调度至 **不同节点** 上，并使用 **不同子网** ，如：SR-IOV 同 Macvlan CNI 混用场景，具体使用流程如下：



nodeaffinity02

IP 池的命名空间亲和性

适用场景:

如果创建的 IP 池仅指定给某一个命名空间使用，该命名空间内的工作负载创建时，均可使用此 IP 池内的 IP，并添加命名空间亲和性的 IP 池为指定命名空间内共享。操作方式可参考创建子网及 IP 池。

FAQ

1. 问：添加了命名空间亲和性，同时添加了工作负载亲和性，或者节点亲和性，最后效果是什么样?

   答：效果为多重亲和性叠加，需要符合所有亲和性，此 IP Pool 才能被使用。

2. 问：添加了命名空间亲和性的 IP 池，是否可基于 命名空间 IP 池细分后，再分配给应用进行固定 IP 池?

如：ippool01 属于子网 10.6.124.0/24，ippool01 中有 100（10.6.124.10~109）个 IP，并添加了命名空间亲和性：kubernetes.io/metadata.name: default，是否可以将：

1. 10.6.124.10~19 分配给 default 命名空间下的应用负载 workload01 使用并 **固定** 。
2. 10.6.124.20~29 分配给 default 命名空间下的应用负载 workload02 使用并 **固定** 。
3. …

答：无法做到，可以做到给某 **1 个** 应用负载使用并 **固定** ，使用方式为：在 IP Pool 上同时添加对应工作负载亲和性。

Calico 网络排障

本页列出通用的 Calico 网络排障步骤。

确认 IP 池地址是否用完

确认 IP 池的 Block 是否被节点已经分完：

- 计算总 IP 数

  IPS = 2 ^ (32 – mask)

  如果掩码为 20，则总 IP 数= 2 ^ 12 = 4096

- 计算有多少个 Block

  查看 IP 池中 BlockSize。默认为 26，则一个 Block 拥有 2 ^ (32 – 26) = 64 个地址，Block 数= 4096 / 64 = 64。

- 确认 Block 是否被分完

  在 Calico 中，应确保每个节点至少拥有一个 Block。 所以如果节点大于 64，则可能需要考虑新增 IP 池或者修改原 IP 池大小，否则可能造成无法正常访问某些节点的 Pod。 请参考 ippool.md。

确认隧道接口是否工作正常

以 vxlan 模式为例:

- 查看 vxlan.calico 状态

  ```
  $ ip a show vxlan.calico
  6: vxlan.calico: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdis
  c noqueue state UNKNOWN group default
      link/ether 66:07:01:14:06:90 brd ff:ff:ff:ff:ff:ff
  ```

```
    inet 10.244.0.0/32 scope global vxlan.calico
    valid_lft forever preferred_lft forever
```

注意其 state 应为 UNKNOWN。

- 查看隧道 IP 是否正常

  正常情况下，隧道 IP 地址应是本节点所属 Block 的 IP 范围内，如果不是则可能出现通信问题，大多一般是地址不够造成的。

- 查看路由是否正常

  calico 在每个节点下发的路由是以其节点 Block 范围为单位的，也就是一个 Block 对应一条静态路由:

  ```
  $ ip r
  default via 172.18.0.1 dev eth0
  ...
  10.244.0.48/28 via 10.244.0.48 dev vxlan.calico onlink
  ...
  ```

  如上 10.244.0.48/28 via 10.244.0.48 dev vxlan.calico onlink，类似这样的路由可能有多条，分别对应不同节点、不同 Block 的地址。

  确认目标 Pod IP 是否存在本节点如上静态路由的目标地址: 如 10.244.0.48/28。 如果不存在，路由更新有问题。

  如果存在，再次检查此静态路由的下一跳：10.244.0.48（即对端节点的 vxlan 隧道 IP）是否和目标地址所对应的 Block 处于同个节点上。 如果不是，可能造成通信问题。如果是，检查本端的邻居表:

  ```
  $ ip n | grep 10.244.0.48
  10.244.0.48 dev vxlan.calico lladdr 66:46:b3:ce:05:5d PERMANENT
  ```

到对应节点确认：对端节点的 vxlan 隧道 IP 的 Mac 是否为：66:46:b3:ce:05:5d。 如果不是，则邻居表学习错误，可删除本段 arp 缓存，使其重新学习。

tcpdump 抓包确认

通过 tcpdump 分别在 veth 虚拟网卡、隧道网卡、主机网卡或 Pod 内部抓包，观察报文在哪一个阶段丢弃。

iptables 规则查看

如果路由转发无异常，可能被某些异常的 iptables 规则过滤掉了。

TODO: calico iptables 规则梳理

如果上述仍未解决：保存现场，打包日志，联系研发。

calicoctl

资源操作

- get
- create
- patch
- delete
- apply
- replace

ipam

- 查看目前分配的 block 及使用情况

```
$ calicoctl ipam show --show-blocks
+----------+-------------------+-----------+------------+------------------+
| GROUPING |       CIDR        | IPS TOTAL | IPS IN USE |      IPS FREE    |
```

```
+----------+-------------------+-----------+-----------
+------------------+
| IP Pool  | 10.244.0.0/26     |        64 | 12 (19%)  | 52 (81%)
         |
| Block    | 10.244.0.0/28     |        16 | 1 (6%)    | 15 (94%)
         |
| Block    | 10.244.0.16/28    |        16 | 3 (19%)   | 13 (81%)
       |
| Block    | 10.244.0.32/28    |        16 | 7 (44%)   | 9 (56%)
         |
| Block    | 10.244.0.48/28    |        16 | 1 (6%)    | 15 (94%)
         |
| IP Pool  | fd1a:29a5:aa4e::/48 | 1.2089e+24 | 0 (0%)    | 1.2089e+24
(100%) |
+----------+-------------------+-----------+-----------
+------------------+
```

- 检查某个 IP 是否被使用

  ```
  $ calicoctl ipam show --ip=10.244.0.17
  IP 10.244.0.17 is in use
  Attributes:
  namespace: local-path-storage
  node: kind-control-plane
  pod: local-path-provisioner-547f784dff-jjmp6
  timestamp: 2022-05-30 16:02:17.551768878 +0000 UTC
  $ calicoctl ipam show --ip=10.244.0.100
  10.244.0.100 is not assigned
  ```

- 释放某个 IP

  如果确认某个 IP 没有被分配，Calico 没有正确地回收此 IP，可手动释放：

calicoctl ipam release --ip=10.244.0.100

NOTE：一定要确认此 IP 并未有 Pod 在使用。

node

- 查看 node 状态

  calicoctl node status

- node 状态检查

  calicoctl node diags

更多命令详情：calicoctl --help

安装参数配置

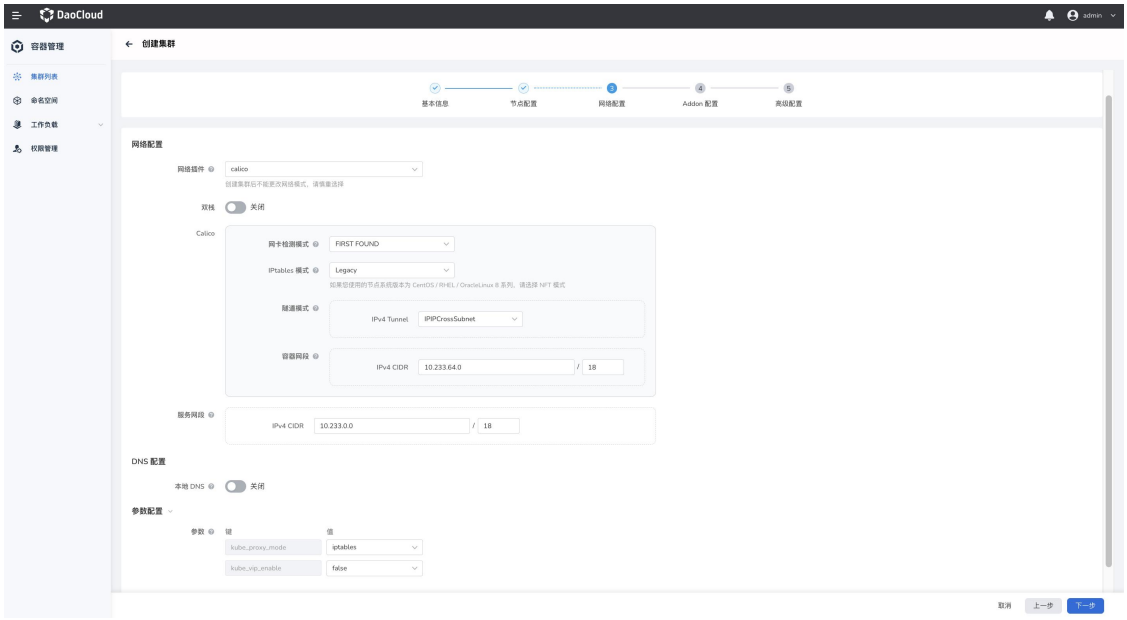
本页说明用 Kubespray 安装 Calico 时各项参数的配置。

前提条件

在 DCE 5.0 中安装 Calico，需要在创建集群—>网络配置页面下，网络插件选择 calico。关于创建集群，请参阅创建工作集群。

参数配置

进入创建集群第三步网络配置，配置如下参数：



calico-install

- 双栈：是否开启双栈，默认关闭。enable_dual_stack_networks：开启后，将为 pod 和 service 提供 IPv4 和 IPv6 网络。
- 网卡检测模式： Calico 网卡检测模式，选择后会通过此网卡进行流量通信。

  - FIRST FOUND：默认模式，枚举所有的节点 IP（忽略一些常见的本地网卡，如 docker0，kube-ipvs0 等），并选择第一个地址。
  - KUBERNETES INTERNAL IP：选择 Node 对象 Status.addresses 字段的第一个内部地址。
  - INTERFACE REGEX：通过网卡名称或者正则表达式指定网卡。
- IPtables 模式：Calico 依赖 IPtables 实现 SNAT 以及网络策略，并且需要和主机的 IPtables 模式保持一致，否则可能会造成通讯问题。支持 Legacy,NFT,Auto 三种模式，默认为 Legacy 模式。如果您使用的节点系统版本为 CentOS 8 / RHEL 8 / OracleLinux 8 系列，请选择 NFT 模式。
- 隧道模式：Calico 支持两种数据包封装模式: IPIP 和 VXLAN，并都支持 Always 和 CrossSubnet 模式。
  - VXLAN Always：无论节点是否处于同一网段，所有跨节点数据包都将使用 VXLAN 被封装，这满足绝大部分的使用场景。
  - VXLAN CrossSubnet：只有当节点处于不同网段时，跨节点的数据包才会被封装。
  - IPIP CrossSubnet：只有当节点处于不同网段时，跨节点的数据包才会被封装。
  - IPIP Always：无论节点是否处于同一网段，所有跨节点数据包都将使用 IPIP 被封装，这满足绝大部分的使用场景。
- IPv4 容器网段：kube_pods_subnet，默认同 ipv4_pools 保持一致。集群下容器使用的网段，决定了集群下容器的数量上限。创建后不可修改。默认值为 10.233.64.0/18。
- IPv6 容器网段：kube_pods_subnet_ipv6，默认同 ipv6_pools 保持一致。集群下容器使用的网段，决定了集群下容器的数量上限。创建后不可修改。开启双栈后需要输入此网段信息。
- 服务网段：同一集群下容器互相访问时使用的 Service 资源的网段决定了 Service 资源的上限。 创建后不可修改。默认为 10.244.0.0/18 。

以下介绍使用 Kubespray 安装 Calico 时的各项参数配置：

!!! note

如果用户需要为 Calico 配置更多功能，可通过 Kubespray 安装 Calico。关于使用 Kubespray 安装 Calico 时的各项参数配置，请在`高级配置`—>`自定义参数`下根据需要添加并填写。
![calico-install](https://docs.daocloud.io/daocloud-docs-images/docs/network/images/calico-arg.png)

建议根据实际情况开启，可作为安装配置项。

- calico_pool_blocksize：默认的 ippool(ipv4) 的 blockSize。

  默认该字段未定义，由全局变量 kube_pods_subnet 控制，默认为 24。

  使用默认配置即可，但必须根据集群规模来设定，至少每个节点拥有一个 block，可在安装时暴露配置项。

- calico_pool_blocksize_ipv6：默认的 ippool(ipv6) 的 blockSize。

  由 calico_pool_blocksize_ipv6 定义，默认为 116。

  使用默认配置即可，但必须根据集群规模来设定，至少每个节点拥有一个 block，可在安装时暴露配置项。

- calico_felix_prometheusmetricsenabled：是否暴露 felix metrics。

  默认关闭，可根据需求决定是否打开。

  可在安装时暴露配置项，默认 false。

- nat_outgoing：跨 ippool 池访问，是否需要 snat。

  默认开启，推荐开启。

  可在安装时暴露配置项，默认 true。

- calico_allow_ip_forwarding：是否在容器命名空间内开启 ip_forwarding。

默认 false，推荐 false。

不建议作为基础配置项，可作为高级配置项。

- calico_datastore：Calico 数据存储引擎、kdd 或 etcd。

  默认 kdd，推荐 kdd。

IPPool

IPPool 表示 Calico 期望从中给 Pod 分配 IP 的地址集合。通过 Kubespray 拉起 Calico 之后，会分别为 IPv4、IPv6 创建一个默认的地址池： default-ipv4-ippool 和 default-ipv6-ippool。

运行命令：

calicoctl get ippools default-ipv4-ippool -o yaml

输出为：

```
apiVersion: crd.projectcalico.org/v1
kind: IPPool
metadata:
  annotations:
    projectcalico.org/metadata: '{"uid":"637e72f1-d4ff-433c-92c5-cafe3aef5753",
"creationTimestamp":"2022-05-12T14:34:03Z"}'
  creationTimestamp: "2022-05-12T14:34:03Z"
  generation: 1
  name: default-ipv4-pool
  resourceVersion: "689"
  uid: c94af437-5a37-46c8-b521-641724b86ff8
spec:
  allowedUses:
    - Workload
    - Tunnel
  blockSize: 26
  cidr: 10.244.0.0/18
  ipipMode: Never
  natOutgoing: true
```

```
  nodeSelector: all()
  vxlanMode: Always
```

BlockSize

在 Calico IPAM 中，IPPool 被细分为 Block，这些 Block 与集群中的特定节点相关联。 集群中的每个节点可以有一个或多个与之相关的 Block。当集群中的节点或 Pod 的数量增加或减少时，Calico 会根据需要自动创建和销毁 Block。

Block 的存在使 Calico 可以有效地聚合分配给同一节点上的 Pod 地址，这将会减少路由表的大小。 默认情况下，Calico 将会从节点相关的 Block 中分配 IP，并在必要时创建新的 Block。 Calico 也支持从与该节点不相关的 Block 中分配 IP 地址。默认情况下，Calico 创建的 Block 可容纳 64 个地址（掩码为 /26），该地址数量支持自定义设置。

!!! note

在集群节点数量较多且 `IPPool Cidr` 不足的情况，我们往往需要根据集群规模提前规划好 `IPPool` 和 `BlockSize` 的大小。
否则可能会出现某些节点无法分配到 `Block` 的情况。

- `BlockSize` 默认为 26，即每个 block 拥有 2^(32-26) = 64 个地址。可由 `calico node env`: `CALICO_IPV4POOL_BLOCK_SIZE` 控制 (IPv4: 20-32;IPv6:116-128)。

- Calico 要求 `BlockSize` 必须大于或者等于 `IPPool` 的 CIDR 的掩码，但在实际环境中应确保每个节点至少有一个 Block。
  所以 Block 的个数应该大于等于节点的个数. 即 2^(`BLOCK_SIZE-IPPool_MASK`) >= NUM(nodes)。

指定多个默认池

随着集群的扩张或 Pod 数量的增加，默认地址池的地址数量可能不足。 这会导致 Pod 可能无 IP 可用或者某些节点没有可分配的 Block。 我们可以通过修改 Calico 的配置文件，使其在 IP 地址不够的时候，选择其他的 IPPool。

创建新的 IPPool 池

运行命令：

```
cat << EOF | calicoctl apply -f -
```

输出为：

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: extra-ippool
spec:
  cidr: 192.168.0.0/20
  blockSize: 26
  vxlanMode: Always
  natOutgoing: true
EOF
```

其中，

- cidr：可由实际环境决定 IP 地址范围
- blockSize：默认为 26，根据实际集群规模决定。缩小 blockSize 意味着每个 Block 中的地址变多，但 Block 总的数量会减少。 这适用于节点数不多但每节点上 Pod 比较多的场景。增大 blockSize 意味着每个 Block 中的地址变少，但 Block 总的数量会增加。 这适用于节点数较多的场景。但总的来说，只要 IPPool 的 CIDR 足够大，可不调整 blockSize（保持默认即可）。
- vxlanMode：采用 vxlan 模式用于跨子网通信
- natOutgoing：跨 IPPool 通信是否需要 snat

IPPool 细粒度控制

默认情况下，IPPool 是集群全局共享的。但是。也可以将 IPPool 指定分配给特定的节点、租户、Pod。

节点过滤

在 IPPool 中根据 nodeSelector 字段去匹配特定的节点，只有特定节点才可以从此 IPPool 中去分配 IP。

- 给节点打上 label

  kubectl label nodes node1 type=test

- 在 IPPool 中配置 nodeSelector

  运行命令：

  cat << EOF | calicoctl apply -f -

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: extra-ippool
spec:
  cidr: 192.168.0.0/20
  blockSize: 26
  vxlanMode: Always
  natOutgoing: true
  nodeSelector: type=="test"
EOF
```

但此 IPPool 不会影响到该节点已经创建的 Pod，如果要更新其 Pod 从该 IPPool 分配地址，则需要 recreate pod。 进一步了解高级的 selector 语法。

租户过滤

可通过在 namespace 中打上特定的 annotation，使该 namespace 下的 Pod 会从此 label 对应的 ippool 分配 IP。

如果要为 namespace 添加 annotation，则编辑 namespace，在 annotation 中添加如下的 key-value 对：

kubectl annotate namespace test-ns "cni.projectcalico.org/ipv4pools"='["extra-ippool"]'

value 为 ippool 的 name 列表。如果是 ipv6，那么 key 为：cni.projectcalico.org/ipv6pools。

!!! note

此操作只能保证此 namespace 下的 Pod 会从 `extra-ippools` 中分配 IP。
但其他 namespace 的 Pod 仍然可以从 `extra-ippools` 分配 IP。

Pod 过滤

可通过在 Pod 的 Annotation 中指定 ippool，使 Pod 从该 ippool 中分配地址：

kubectl annotate pod test-pod "cni.projectcalico.org/ipv4pools"='["extra-ippool"]'

改变 BlockSize

> 在安装 Calico 之前，应事先定义好 blockSize。因为安装之后，BlockSize 的值不能编辑。 因此，建议在安装前更改 IPPool block 的大小，以尽量减少对 Pod 连接的中断。

改变 BlockSize 步骤：

1. 创建临时的 ippool

```
apiVersion: projectcalico.org/v3
kind: IPPool
```

```
metadata:
  name: temporary-pool
spec:
  cidr: 10.0.0.0/16
  ipipMode: Always
  natOutgoing: true
```

!!! note

注意不要跟现有的 `ippool` 子网冲突。

2. 将 default-ipv4-ippool 设置为 disable

calicoctl patch ippool default-ipv4-ippool -p '{"spec": {"disabled": true}}'

其中，default-ipv4-ippool 为将要修改的 ippool 的名称。

3. 检查 ippool 的状态

运行命令：

```
$ calicoctl get ippool -o wide
NAME                  CIDR             NAT    IPIPMODE   DISABLED
default-ipv4-ippool   192.168.0.0/16   true   Always     true
temporary-pool        10.0.0.0/16      true   Always     false
```

default-ipv4-ippool 已 DISABLED，新创建的 Pod 不会从此 ippool 中分配地址。

4. 删除先前的所有 Pod

!!! note

此步骤需要删除全部存在于 `default-ipv4-ippool` 下的 Pod，所以会造成 Pod 的连通性暂时中断，请在合适的时机进行此操作。

删除 default namespace 下的所有 Pod：

kubectl delete po --all

等待 Pod 重建完成，Pod 会使用 temporary-pool 的地址。

5. 删除 default-ipv4-ippool

   calicoctl delete ippool default-ipv4-ippool

6. 重新创建 default-ipv4-ippool 并修改 cidr 或者 blockSize

   运行命令：

   calicoctl create -f -<<EOF

   ```
   apiVersion: projectcalico.org/v3
   kind: IPPool
   metadata:
     name: default-ipv4-ippool
   spec:
     blockSize: 27    # change blockSize from 26 to 27
     cidr: 192.0.0.0/16
     ipipMode: Always
     natOutgoing: true
   EOF
   ```

   !!! note

   可根据实际情况修改 `cidr` 和 `blockSize`。

7. DISABLE temporary-pool

   calicoctl patch ippool temporary-pool -p '{"spec": {"disabled": true}}'

8. 重新创建所有 Pod

   删除 default namespace 下的所有 Pod：

   kubectl delete po --all

等待 Pod 重建完成，Pod 会使用 default-ipv4-ippool 的地址。

9. 删除 temporary-pool

迁移 IPPool

原有地址池的 IP 地址不足，需要迁移到新的 IPPOOL 中。

!!! note

如果在创建和验证新的 IP 池之前删除旧的 IP 池，现有 Pod 的连接将会受到影响。当 Pod 被删除时，业务会可能中断。

迁移 IPPool 步骤：

1. 创建新的 ippool

!!! note

建议新 `IPPool` 处于 Kubernetes 集群的 CIDR 中。如果从 Kubernetes 集群 CIDR 外部分配 Pod IP，某些流量可能会不必要地应用 NAT，导致意外行为。

2. Disable 旧 ippool

   ```
   calicoctl patch ippool default-ipv4-ippool -p '{"spec": {"disabled": true}}'  # 假设 default-ipv4-ippool 是旧 IPPool
   ```

3. 重新创建旧 IP 池下的所有 Pod

   目的是让所有 Pod 从新的 IPPool 中分配地址

4. 验证

   新启动 Pod，观察是否从新池中分配 IP 并测试连通性

5. 删除旧 ippool

Q&A

- 不同的 IPPool 是否支持子网重叠?

不支持，在创建 ippool 时，Calico 会校验 ippool 的 cidr 是否与已存在的 ippool 存在子网重叠的情况。

- IPPool 创建后 BlockSize 是否还能修改?

  不能修改。ippool 创建之后，已经根据 CIDR 和 BlockSize 创建好集群所有的 block，手动修改 BlockSize 是没有效果的。

什么是 Calico

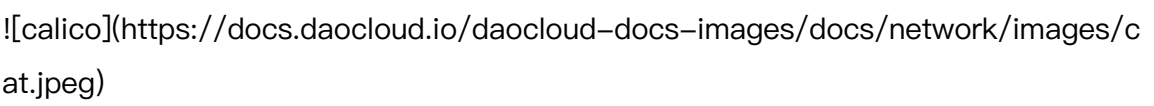
Calico 创建和管理一个扁平的三层网络（不需要 overlay），每个容器会分配一个可路由的 IP。 由于通信时不需要解包和封包，网络性能损耗小，易于排查，且易于水平扩展。

小规模部署时可以通过 BGP client 直接互联，大规模下可通过指定的 BGP Route Reflector 来完成，这样保证所有的数据流量都是通过 IP 路由的方式完成互联的。

Calico 基于 iptables 提供了丰富而灵活的网络 Policy，保证通过各个节点上的 ACL 来提供 Workload 的多租户隔离、安全组以及其他可达性限制等功能。

!!! info

Calico 原意为”有斑点的“花猫，也叫三色猫，所以 Calico 的 logo 是一只三色猫。



Calico 组件

Calico 由以下组件组成，在部署 Calico 的时候部分组件是可选的。

- 什么是 Calico
  - Calico 组件
    - Calico API Server
    - Felix
    - BIRD
    - confd
    - Dikastes
    - CNI

- 数据存储插件
  - Kubernetes API datastore
  - etcd
- IPAM
- kube-controller
- Typha
- calicoctl
- 云编排器插件

Calico API Server

可以使用 kubectl 直接管理 Calico。

Felix

Felix 以 agent 代理的形式在每台机器端点上运行。对路由和 ACL 以及主机进行编程，为该主机上的端点提供所需的连接。

根据具体的编排器环境，Felix 负责：

- 接口管理

  将有关接口的信息编入内核，以便内核能够正确处理来自该端点的流量。特别是，确保主机响应来自每个工作负载的 ARP 请求，提供主机的 MAC，并为它所管理的接口启用 IP 转发。 它还监控接口，以确保编程在适当的时候应用。

- 路由编程

  将其主机上的端点的路由编程到 Linux 内核的 FIB（转发信息库）。 这可以确保到达主机上的以这些端点为目的地的数据包被相应地转发。

- ACL 编程

在 Linux 内核中编程 ACL，以确保只有有效的流量可以在端点之间发送，并且端点不能规避 Calico 的安全措施。

- 状态报告

  提供网络健康数据。特别是在配置其主机时报告错误和问题。这些数据被写入数据存储，以便对网络的其他组件和运营商可见。

BIRD

BGP Internet Routing Daemon，简称 BIRD。从 Felix 获取路由，并分发到网络上的 BGP peer，用于主机间的路由。在每个 Felix 代理的节点上运行。

BGP 客户端负责：

- 路由分配

  当 Felix 将路由插入 Linux 内核的 FIB 时，BGP 客户端将它们分配给部署中的其他节点。这确保了部署中的有效流量路由。

- BGP 路由反射器的配置

  BGP 路由反射器通常是为大型部署而配置的，而不是一个标准的 BGP 客户端。BGP 路由反射器作为连接 BGP 客户端的一个中心点。（标准 BGP 要求每个 BGP 客户端在网格拓扑结构中与其他每个 BGP 客户端连接，这很难维护。）

  为了实现冗余，可以无缝部署多个 BGP 路由反射器。BGP 路由反射器只参与网络的控制，没有终端数据通过它们。 当 Calico BGP 客户端将其 FIB 中的路由通告给路由反射器时，路由反射器将这些路由通告给部署中的其他节点。

confd

开源的、轻量级的配置管理工具。监控 Calico 数据存储对 BGP 配置和全局默认的日志变更，如 AS 号、日志级别和 IPAM 信息。

confd 根据存储中的数据更新，动态生成 BIRD 配置文件。当配置文件发生变化时，confd 会触发 BIRD 加载新的文件。

Dikastes

执行 Istio 服务网格的网络策略。作为 Istio Envoy 的一个 Sidecar 代理，在集群上运行。

Dikastes 是可选的。Calico 在 Linux 内核（使用 iptables，在三、四层）和三到七层使用 Envoy 的 Sidecar 代理 Dikastes 为工作负载执行网络策略，对请求进行加密认证。 使用多个执行点建立基于不同标准的远程端点的身份。即使安全威胁绕过 Envoy 代理，影响工作负载 Pod，主机 Linux 内核的执行也能保护工作负载。

CNI

为 Kubernetes 集群提供 Calico 网络。

将 API 暴露给 Kubernetes 的 Calico 二进制文件为 CNI 插件，该插件必须安装在 Kubernetes 集群的每个节点上。 Calico CNI 插件允许使用符合 CNI 网络规范的编排调度器使用 Calico 网络。

数据存储插件

通过减少每个节点对数据存储的影响来扩大规模。

Kubernetes API datastore

在 Calico 中使用 Kubernetes API 数据存储（kdd）的优点是：

- 管理简单，不需要额外的数据存储
- 使用 Kubernetes RBAC 来控制对 Calico 资源的访问
- 使用 Kubernetes 审计日志来生成 Calico 资源变化的审计日志

etcd

etcd 是一个一致的、高可用的分布式键值存储，为 Calico 网络提供数据存储，并用于组件之间的通信。 etcd 仅支持保护非集群主机（Calico v3.1 及更新版本）。etcd 的优点是：

- 支持在非 Kubernetes 平台上运行 Calico。
- 解耦 Kubernetes 和 Calico 资源，例如允许独立地扩展数据存储。
- 支持 Calico 集群中包含多个 Kubernetes 集群，例如让 Calico 主机保护的裸机服务器与 Kubernetes 集群互通。

IPAM

使用 Calico 的 IP 池资源将 IP 地址分配给集群中的 Pod。

一般情况下，IPAM 为 Calico 的默认安装插件。

kube-controller

监控 Kubernetes 的 API，并根据集群状态执行行动。

tigera/kube-controllers 容器包括以下控制器：

- Policy 控制器
- Namespace 控制器
- ServiceAccount 控制器
- WorkloadEndpoint 控制器
- Node 控制器

Typha

通过减少每个节点对数据存储的影响来扩大规模。作为数据存储和 Felix 实例之间的一个守护程序运行。Typha 为默认安装的插件，但默认不进行配置。

Typha 代表 Felix 和 confd 等所有客户端维护一个单一的数据存储连接。 可用于缓存数据存储的状态，并复制事件，以便将事件推广到更多的监听器。 一个 Typha 实例支持数百个 Felix 实例，可以大幅降低数据存储的负载。 由于

Typha 可以过滤掉与 Felix 无关的更新，因而减少了 Felix 的 CPU 使用。 在一个大规模（100 多个节点）的 Kubernetes 集群中，这是至关重要的，因为 API 服务器产生的更新数量随着节点数量的增加而增加。

calicoctl

Calicoctl 命令行作为二进制或容器需要单独安装，支持在任何可以通过网络访问 Calico 数据存储的主机上使用。

云编排器插件

将管理网络的编排器 API 转换成 Calico 的数据模型和数据存储。

对于云供应商，Calico 为每个云编排平台提供了一个单独的插件。 这使得用户可以使用各自的编排器来管理 Calico 网络。 当需要时，编排器插件会将 Calico 网络的反馈信息提供给编排器。 例如，提供关于 Felix liveness 的信息，并在网络设置失败时将特定端点标记为失败。

监控指标

本页说明如何在 Calico 中开启 calico_prometheus_metrics，接入 Prometheus 监控指标。

开启组件 metrics

通过 kubespray 部署时可根据 calico_felix_prometheusmetricsenabled 参数决定是否打开，默认为 false，或者通过以下方式手动开启：

1. 开启 calico_felix_prometheusmetricsenabled：

   calicoctl patch felixconfiguration default --patch '{"spec":{"prometheusMetricsEnabled": true}}'

   或者

   kubectl patch felixconfiguration default --type merge --patch '{"spec":{"prometheusMetricsEnabled": true}}'

2. 开启 calico_kube_controller_metrics：

   calicoctl patch kubecontrollersconfiguration default --patch '{"spec":{"prometheusMetricsPort": 9095}}'

   或者

   kubectl patch kubecontrollersconfiguration default --type=merge --patch '{"spec":{"prometheusMetricsPort": 9095}}'

创建各自组件的 metrics service

calico_felix_svc：

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: calico-node-metrics
  namespace: kube-system
  labels:
    app: calico-node
    role: metrics
spec:
  clusterIP: None
  selector:
    k8s-app: calico-node
  ports:
  - port: 9091
    name: metrics
    targetPort: 9091

calico_:

apiVersion: v1
kind: Service
metadata:
  name: calico-kube-controllers-metrics
  namespace: kube-system
  labels:
    app: calico-kube-controllers
    role: metrics
spec:
  clusterIP: None
  selector:
    k8s-app: calico-kube-controllers
  ports:
  - port: 9095
    name: metrics
    targetPort: 9095
```

创建 ServiceMonitor 对象

calico_felix_svc：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: calico-node
  namespace: kube-system
  labels:
    app: calico-node
spec:
  endpoints:
  - interval: 30s
    port: metrics
  selector:
    matchLabels:
      app: calico-node
      role: metrics
```

calico-kube-controllers-metrics：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: calico-kube-controller
  namespace: kube-system
  labels:
    app: calico-kube-controller
spec:
  endpoints:
  - interval: 30s
    port: metrics
  selector:
```

```
    matchLabels:
      app: calico-kube-controllers
      role: metrics
```

重要指标列表

序号	指标	说明	价值指数	其他说明	关联问题
1	felix_ipset_errors	执行 ipset-restore 失败次数	**	可考虑采集	次数+1不一定造成问题
2	felix_iptables_restore_calls	执行 iptables-restore 次数	*****	采集	NA
3	felix_iptables_restore_errors	执行 iptables-restore 失败次数	*****	采集	restore失败可能造

序号	指标	说明	价值指数	其他说明	关联问题成Pod访问失败。出现restore失败的原因可能是xtables_lock

序号	指标	说明	价值指数	其他说明	关联问题竞争失败，请检查主机上iptables数量是否过多
4	felix_iptables_save_calls	执行 iptables-save 的次数	****	可考虑采集	NA
5	felix_iptables_save_e	执行 iptables-save 失败的次数	*****	采集	

序号	指标	说明	价值指数	其他说明	关联问题
	rrors				
6	felix_log_errors	日志报告 error 的次数	*****	建议采集	
7	ipam_allocations_per_node	每个节点上 IP 分配的数量	****	建议采集	
8	ipam_blocks_per_node	每个节点上分配的 Block 数量	*****	采集	

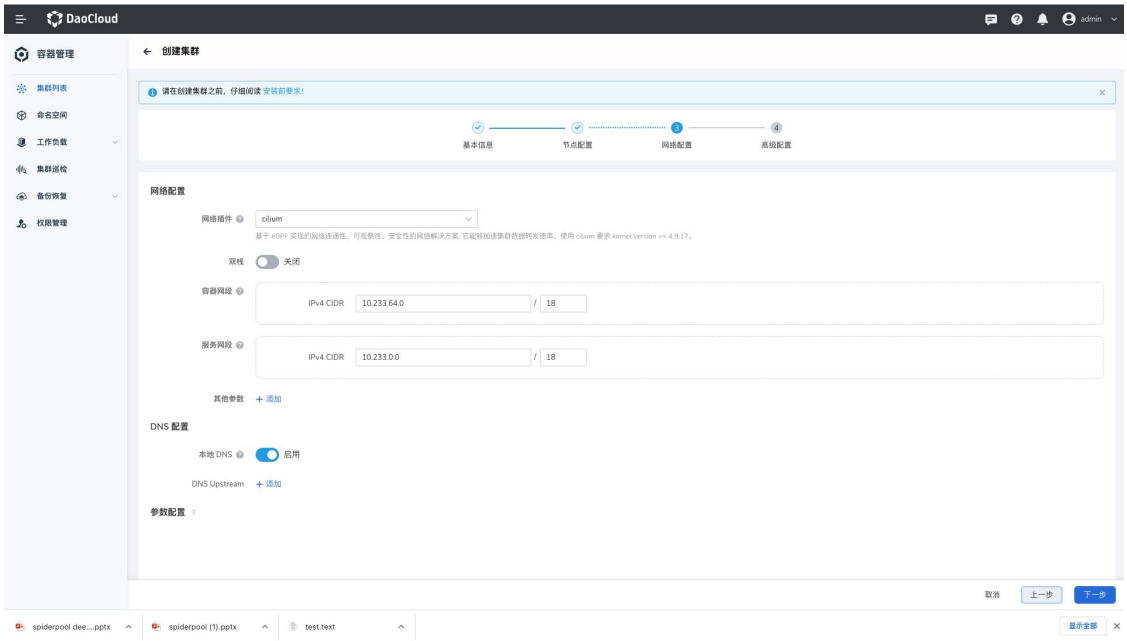
安装参数配置

本页说明使用 Kubean 安装 Cilium 时的各项参数配置，以及主要功能的开启情况和相关说明。

前提条件

1. 请确认 操作系统 Kernel 版本号 >= 4.9.17，推荐 5.10+。

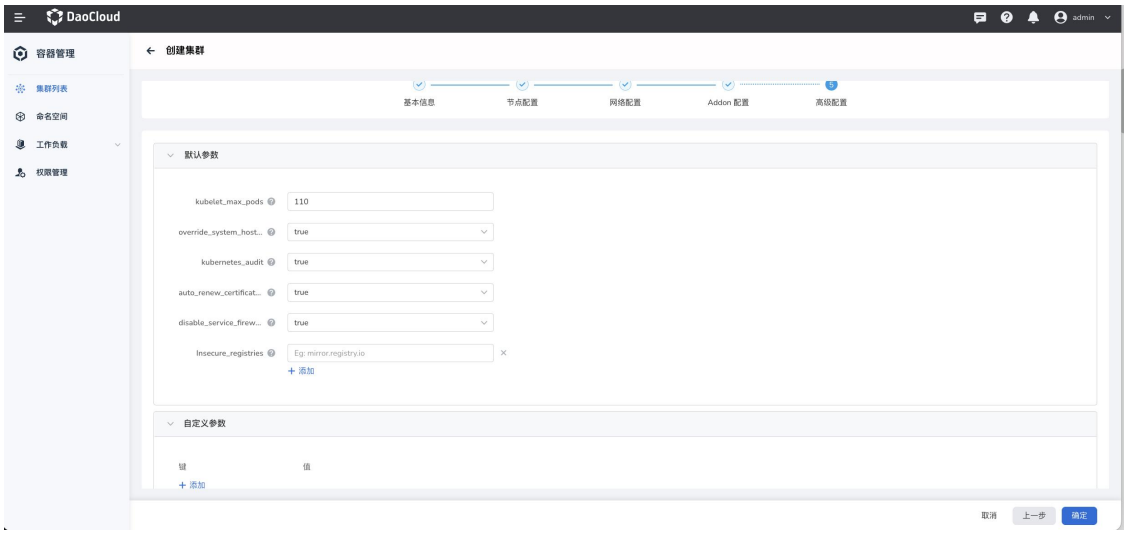
2. DCE 5.0 安装时，选择的网络 CNI 为 Cilium 在 DCE 5.0 中安装 Cilium，需要在创建集群—>网络配置页面下，网络插件选择 cilium。关于创建集群，请参阅创建工作集群。



网络设置

高级参数

如果用户需要为 Cilium 配置更多功能，可通过 Kubean 安装 Cilium。关于使用 Kubean 安装 Cilium 时的各项参数配置，请在高级配置—>自定义参数下根据需要添加并填写。



cilium-arg

以下介绍使用 Kubean 安装 Cilium 时的各项参数配置：

- Cilium 数据模式

  默认使用 VXLAN 隧道模式，通过以下参数进行设置：

  cilium_tunnel_mode: vxlan

  支持的值有“vxlan”、“geneve”及“disabled”，其中“disabled”表示使用路由模式。

- IPAM 模式 IPAM 负责分配和管理网络端点（容器或其它）的 IP 地址。Cilium 支持多种 IPAM 模式。默认使用“Cluster Scope”模式，可以通过以下参数进行设置：

  cilium_ipam_mode: cluster-pool

  支持的值有“cluster-pool”、“kubernetes”及各大公有云定制的模式。

  1. kubernetes: 使用 Kubernetes 自带的 host-scope IPAM。地址分配委托给每个节点进行，per-node 的 Pod CIDR 存放在 v1.Node 中。

2. cluster-pool:默认的 IPAM 模式，它分配 per-node 的 Pod CIDR，并在每个节点上使用 host-scope 的分配器来分配 IP 地址。

   此模式和 kubernetes 类似，区别在于后者在 v1.Node 资源中存储 per-node 的 Pod CIDR。

- IPV4 及 IPV6

  默认使用 IPV4，可以通过以下参数进行设置，如通过界面开启双栈开关，默认 IPv6 参数自动打开：

  ```
  cilium_enable_ipv4: true
  cilium_enable_ipv6: false  # true 开启 IPV6
  ```

- 集群名称

  Cilium 集群的默认名称为“default”，可以通过以下参数进行设置：

  ```
  cilium_cluster_name: default
  ```

- 身份模式

  cilium id 存储结构，crd or kvstore，通常情况下直接使用 CRD 存储元信息会是一个更为便利的选择，不过在大规模集群下，拆分一组独立的 ETCD 给 cilium 单独使用会是一个更高效的选择。

  默认使用“crd”模式，可以通过以下参数进行设置：

  ```
  cilium_identity_allocation_mode: crd
  ```

- 资源限制

  默认的值为：

  ```
  cilium_memory_limit: 500M
  cilium_cpu_limit: 500m
  ```

```
cilium_memory_requests: 64M
cilium_cpu_requests: 100m
```

用户可以根据自身集群情况进行相应的调整。

- Cilium DaemonSet 再次准备就绪的时间

  可以通过以下参数进行设置：

  ```
  cilium_rolling_restart_wait_retries_count: 30
  cilium_rolling_restart_wait_retries_delay_seconds: 10
  ```

- 监控聚合级别

  默认使用“medium”，可以通过以下参数进行设置：

  ```
  cilium_monitor_aggregation: medium
  ```

  支持的值有“none”、“low”、“medium”及“maximum”。

  第一次监控 TCP 标志时，会生成监视器通知，但是只有在聚合等级为“medium”或更高级别时生效。

  可以通过以下参数进行设置：

  ```
  cilium_monitor_aggregation_flags: "all"
  ```

- 替换 kube-proxy

  默认不进行替换，可以在界面上开启或关闭，也可以通过以下参数进行设置：

  ```
  cilium_kube_proxy_replacement: disabled
  ```

  支持的值有“disabled”、“strict”、“probe”及“partial”。

- 当流量离开集群时是否做 SNAT

Cilium 默认情况下会对离开集群的 IPv4、IPv6（若已开启）流量做 SNAT。可以通过以下参数进行设置：

```
cilium_enable_ipv4_masquerade: true
cilium_enable_ipv6_masquerade: true
```

做 SNAT 的方式默认使用“iptables”，还支持“eBPF”模式。“eBPF”模式更高效，但“eBPF”不支持 IPv6。

可通过以下参数进行设置：

```
cilium_enable_bpf_masquerade: false
```

- Hubble

  默认安装并启用 Hubble。默认暴露的指标有：

  ```
  - dns
  - drop
  - tcp
  - flow
  - icmp
  - http
  ```

  可通过以下相关参数进行设置：

  ```
  cilium_hubble_install: true            # 安装 hubble
  cilium_enable_hubble: true              # 启用 hubble
  cilium_enable_hubble_metrics: true   # 开启 metrics
  cilium_hubble_metrics: {}               # metrics 内容
  cilium_hubble_tls_generate: true      # 自动更新 hubble-relay 证书
  ```

- cgroup2 文件系统的自动挂载

  默认启用 cgroup2 文件系统的自动挂载功能，默认挂载路径为“/run/cilium/cgroupv2”。

可以通过以下参数进行设置：

```
cilium_cgroup_auto_mount: true
cilium_cgroup_host_root: "/run/cilium/cgroupv2"
```

- 绕过主机命名空间中的 netfilter

  默认绕过主机命名空间中的 netfilter，可以通过以下参数进行设置：

  ```
  cilium_enable_host_legacy_routing: true
  ```

- 启用远程节点身份

  默认启用远程节点身份，可以通过以下参数进行设置：

  ```
  cilium_enable_remote_node_identity: true
  ```

- 自定义参数

  Kubespray 支持的 Cilium 参数有限。如果要开启 Kubespray 不支持的配置，可通过以下参数进行设置：

  ```
  cilium_config_extra_vars:
    enable-endpoint-routes: true
  ```

- Kubespray 支持的参数

  Kubean 默认调用 Kubespray 来安装集群，参阅以下文档：

  - 通过 Kubespray 安装 Cilium
  - Cilium 默认参数

什么是 Cilium

Cilium 是一款开源软件，为在 Linux 容器管理平台（如 Kubernetes）上部署的应用服务透明地提供网络和 API 连接，并保障这些网络和连接的安全。

Cilium 的底层技术利用 Linux 内核的新技术 eBPF，可以在 Linux 系统中动态实现安全性、可见性和网络控制逻辑。 Cilium 基于 eBPF 提供了多集群路由、替代 kube-proxy 实现负载均衡、透明加密以及网络和服务安全等诸多功能。 除了提供传统的网络安全之外，eBPF 的灵活性还支持应用协议和 DNS 请求/响应安全。 同时，Cilium 与 Envoy 紧密集成，提供了基于 Go 的扩展框架。 因为 eBPF 运行在 Linux 内核中，所以无需对应用程序代码或容器配置进行任何更改就可以应用所有 Cilium 功能。

基于微服务的应用程序分为小型独立服务，这些服务使用 HTTP、gRPC、Kafka 等轻量级协议通过 API 相互通信。 但是，现有的 Linux 网络安全机制（例如 iptables）仅在网络和传输层（即 IP 地址和端口）上运行，并且缺乏对微服务层的可见性。

Cilium 为 Linux 容器框架（如 Docker 和 Kubernetes）带来了 API 感知网络安全过滤。 通过 eBPF 技术，Cilium 提供了一种基于容器 / 容器标识来定义和实施网络层和应用层安全策略的方法。

注：Cilium 中文意思是“纤毛“，它十分细小而又无处不在。

Cilium 网络通信加速

介绍

本章节介绍如何配置 Cilium 通信加速能力，本章节提供 2 种方式配置，可根据需要进行选择

前提条件

请确认操作系统 Kernel 版本号 >= 4.9.17，推荐 5.10+。查看并安装升级最新的 Linux 内核版本，您可以按照如下命令进行操作：

1. 查看当前内核版本：

   ```
   uname -r
   ```

2. 安装 ELRepo 存储库，ELRepo 存储库中提供的最新 Linux 内核版本：

   ```
   rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
   rpm -Uvh https://www.elrepo.org/elrepo-release-7.el7.elrepo.noarch.rpm
   ```

3. 安装最新的 Linux 内核版本：

   ```
   yum --enablerepo=elrepo-kernel install kernel-ml
   ```

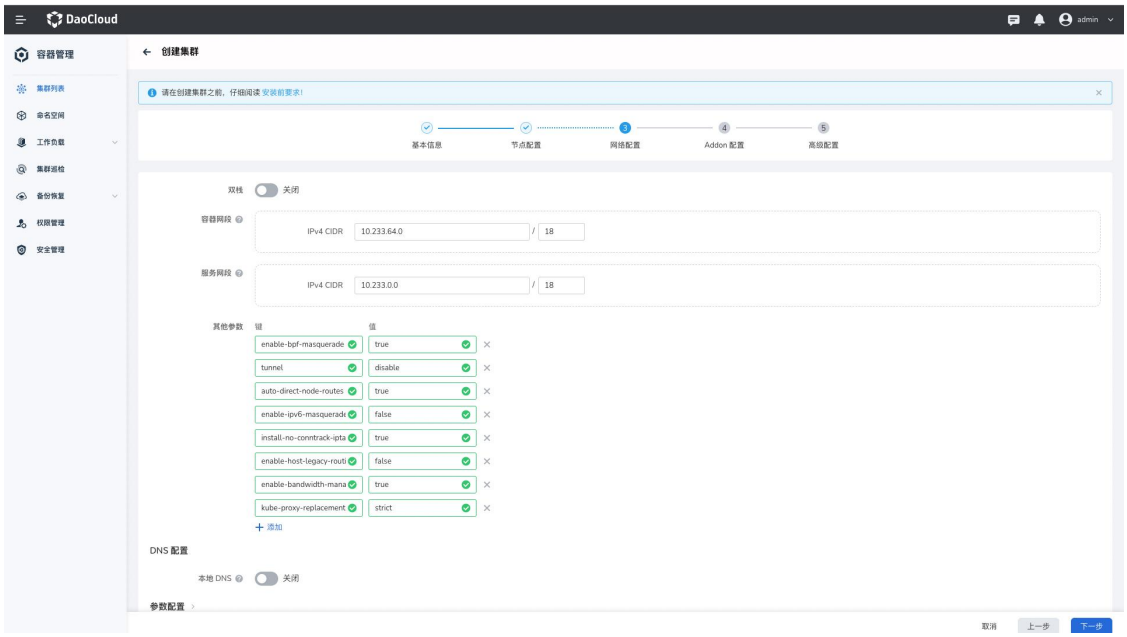
   kernel-ml 是最新文档版内核，您可以根据需要选择其他版本。

4. 更新 GRUB 配置，以便在启动时使用新内核版本：

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

方式一：创建集群时配置

1. 点击容器管理一>集群列表，点击创建集群，输入集群基本信息，节点配置信息后，进入网络配置进行如下配置。



- 集群的网络插件选择 cilium

- 添加其他参数 如下：

  auto-direct-node-routes: "true" #路由模式下必须设置为 True，否则无法路由跨节点流量
  enable-bpf-masquerade: "true" #设置该参数为 true 之后，如果使用了 masquerading 特性，那么会基于 eBPF 来替代 iptables 的实现，需要内核 5.10 及之后版本，否则就算开启也会降级到 iptables 实现
  enable-ipv6-masquerade: "false" #对于 POD 访问外界的流量做源地址转换，如果使用 tunnel 模式需要开启，如果是通过 BGP 打通了物理网络则禁用
  install-no-conntrack-iptables-rules: "true" # 为 pod 下发 no-ct 的 iptables 规则，提升性能
  enable-host-legacy-routing: "false" # 关闭可以主机处理数据包时绕过其内核协议栈，加快数据转发。默认情况下打开，但如果主机内核不支持，则回退到传统行为

enable-bandwidth-manager: "true # 打开 bandwidth-manager，提高 tcp、udp 的性能

kube-proxy-replacement: strict # 启用 kube-proxy replacement 功能，需要删除 kube-proxy 组件

tunnel: disabled # 关闭隧道模式

enable-bbr: "true" #（可选） bbr 网络阻塞控制，要求内核大于 5.18

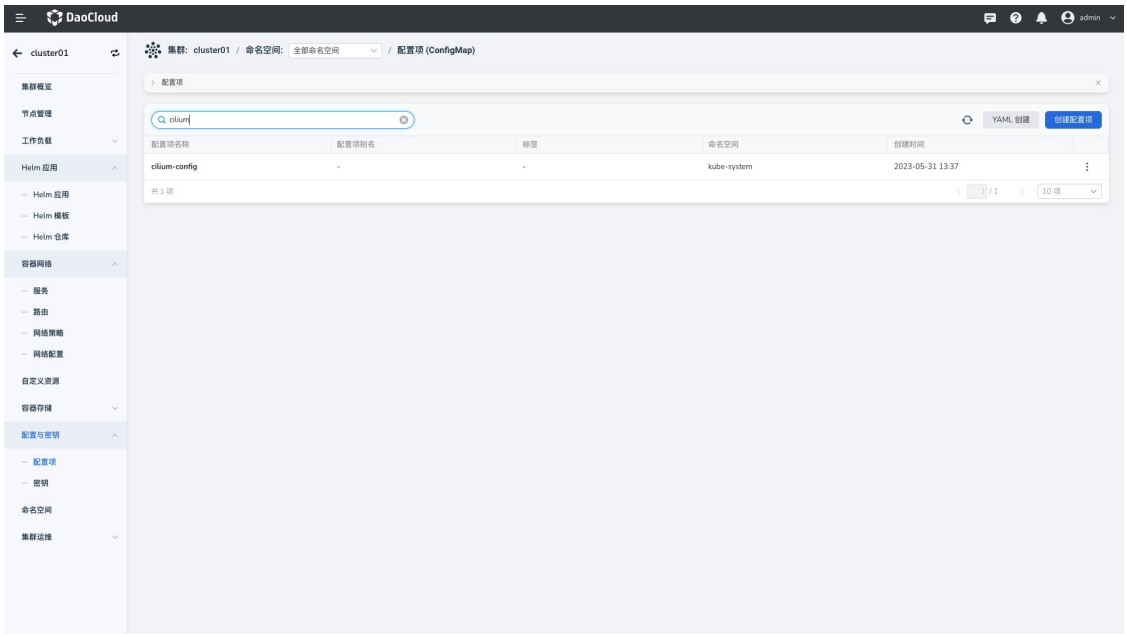
- 其他均使用默认配置项

2. 点击创建集群完成创建。

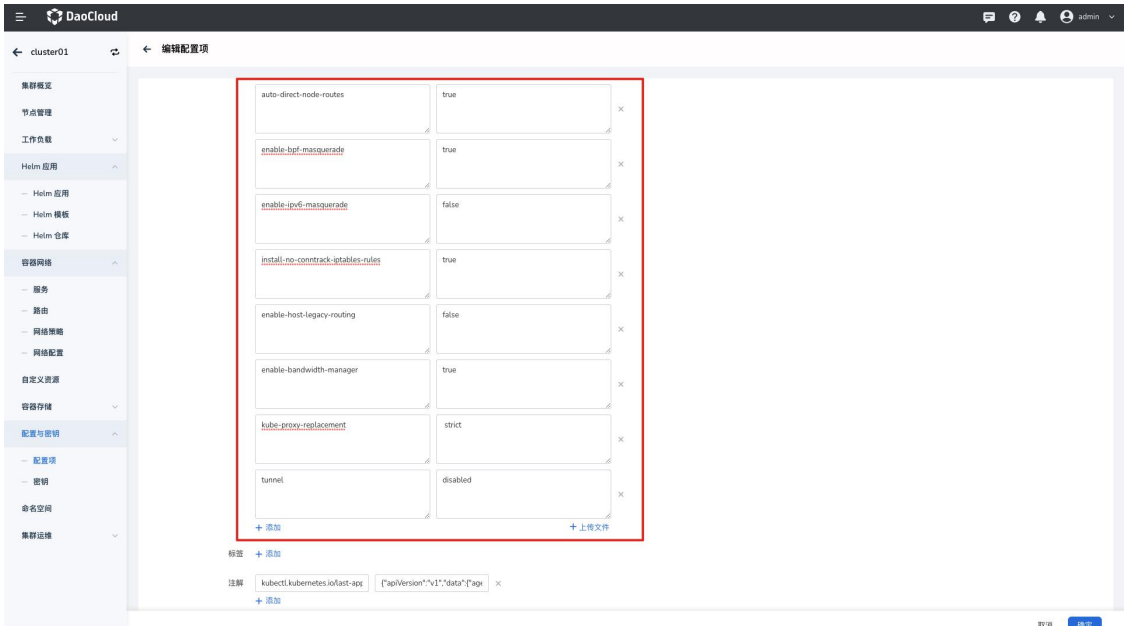
方式二：修改 Cilium Config 配置

如集群已经创建，需要开启加速参数，可修改 Cilium-Config 文件。

点击容器管理->集群列表，点击已创建的集群名称，进入详情后，点击配置与秘钥，选择配置项，找到 cilium-config 点击编辑，输入上述加速参数：



cilium-sppedup02



speed-up03

参数编写完成后，点击完成，应用 Pod 通信即可进行加速。

应用跨集群通信

介绍

随着微服务进程发展，为了满足应用隔离、高可用/容灾以及运维管理的需求，许多企业选择部署多个 Kubernetes（K8s）集群。然而，这种多集群部署带来了一个问题：一些应用依赖于其他 K8s 集群中的微服务，需要实现集群间的通信。具体而言，需要从一个集群的 Pod 去访问另外一个集群的 Pod 或者 Service。

前提条件

请确认操作系统 Kernel 版本号 >= 4.9.17，推荐 5.10+。查看并安装升级最新的 Linux 内核版本，您可以按照如下命令进行操作：

1. 查看当前内核版本：

   ```
   uname -r
   ```

2. 安装 ELRepo 存储库，ELRepo 存储库中提供的最新 Linux 内核版本：

   ```
   rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
   rpm -Uvh https://www.elrepo.org/elrepo-release-7.el7.elrepo.noarch.rpm
   ```

3. 安装最新的 Linux 内核版本：

   ```
   yum --enablerepo=elrepo-kernel install kernel-ml
   ```

   kernel-ml 是最新文档版内核，您可以根据需要选择其他版本。

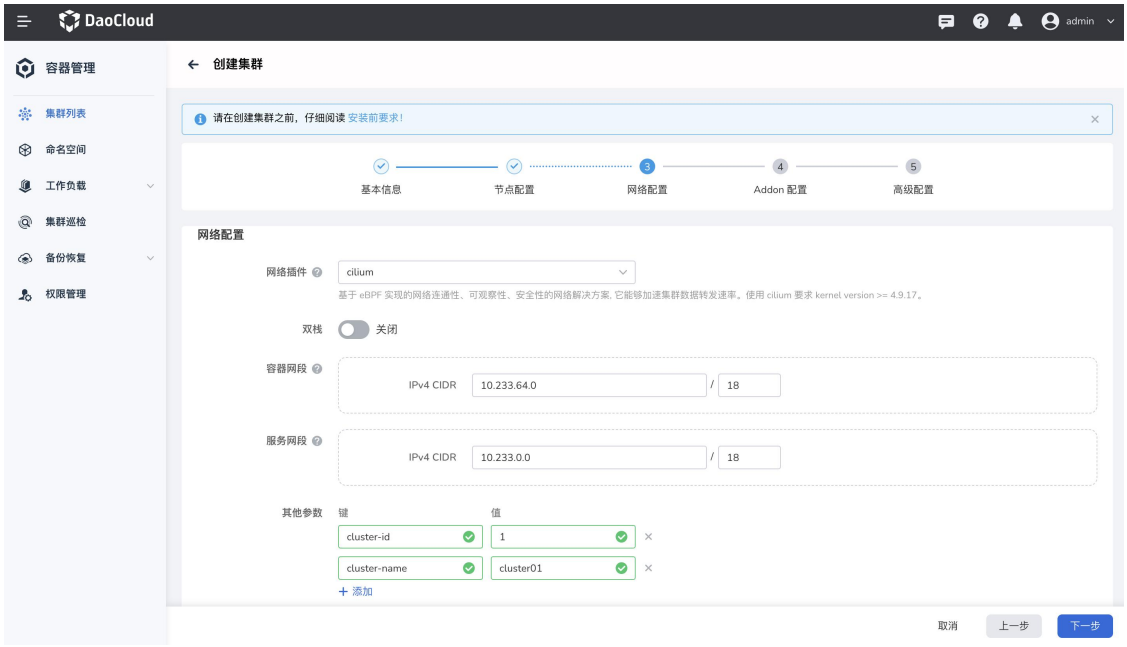
4. 更新 GRUB 配置，以便在启动时使用新内核版本：

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

创建集群

关于创建集群的更多信息，可参考创建集群

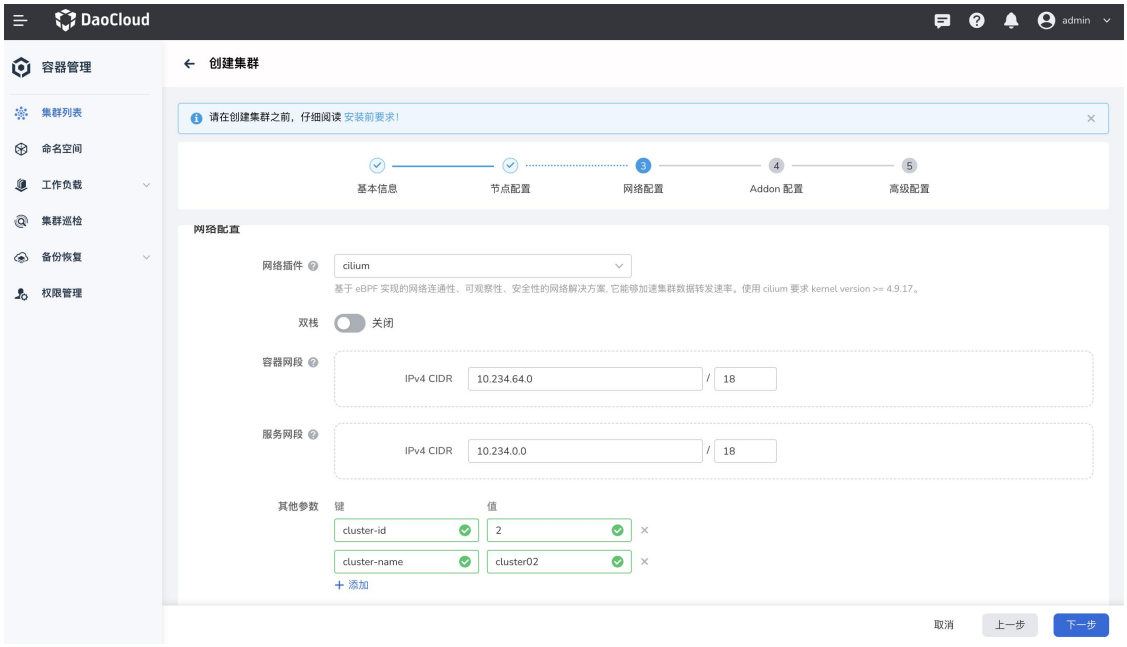
1. 创建两个名称不同的集群分别为 cluster01 和 cluster02。



创建集群 1

- 集群 cluster01 的网络插件选择 cilium
- 添加两个参数 cluster-id 和 cluster-name
- 其他均使用默认配置项

2. 以同样的步骤创建集群 cluster02。

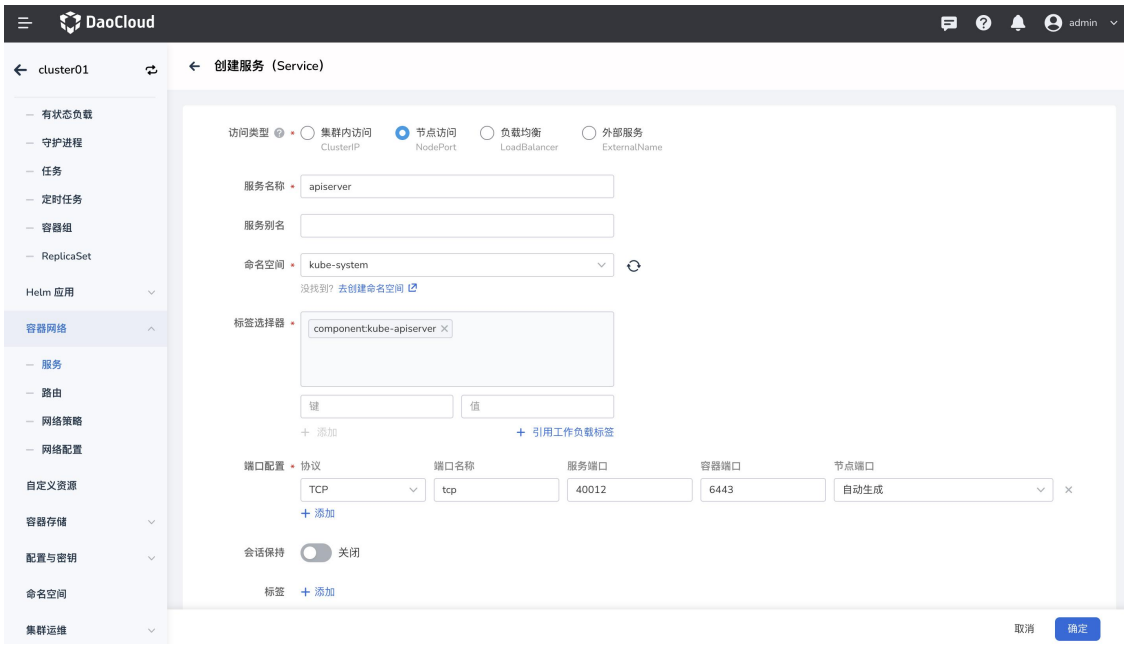


创建集群 2

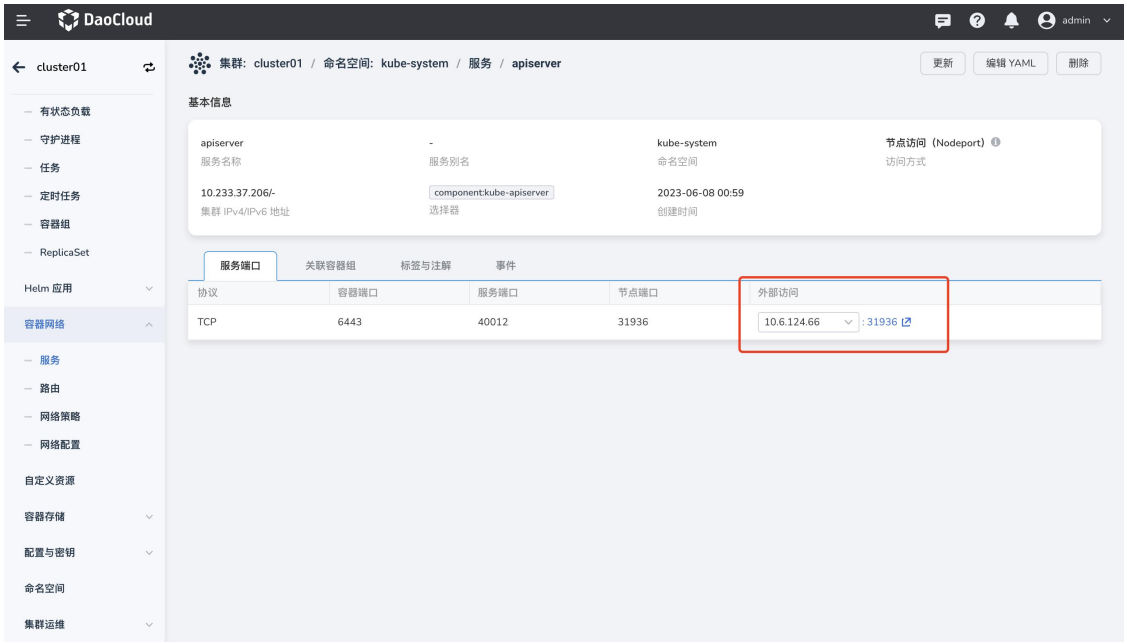
两个集群使用的容器网段和服务网段一定不能冲突。两个参数的值不能冲突，便于标识集群确保唯一性，避免跨集群通信时出现冲突。

为 API Server 创建 Service

1. 集群创建成功后，在两个集群上分别创建一个 Service，用于将该集群的 API server 对外暴露。



创建 service

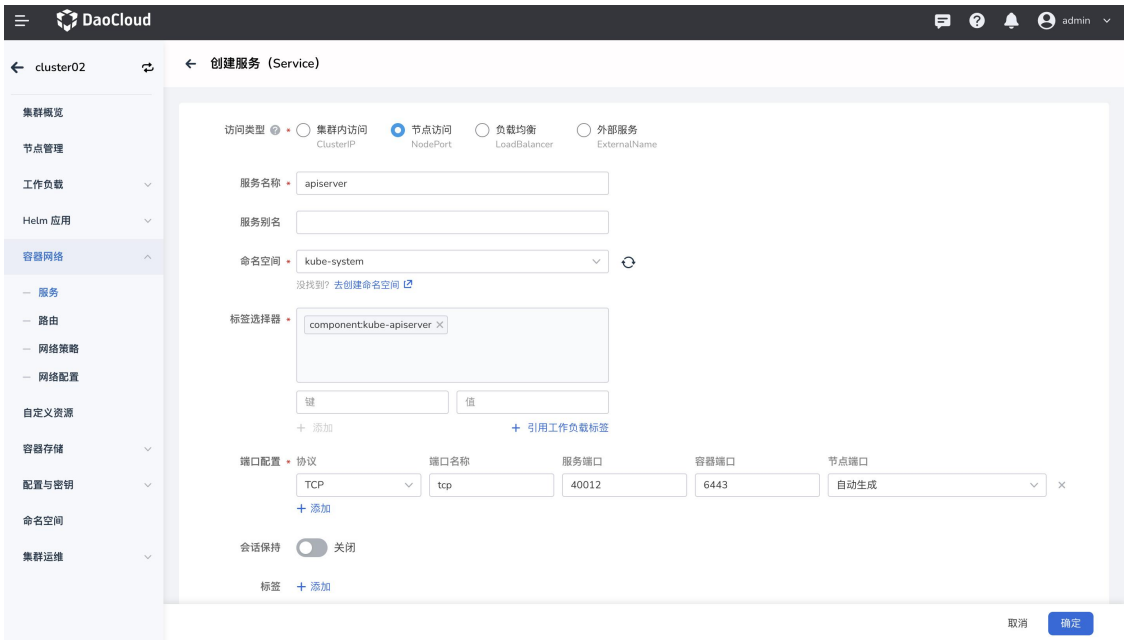


创建 service

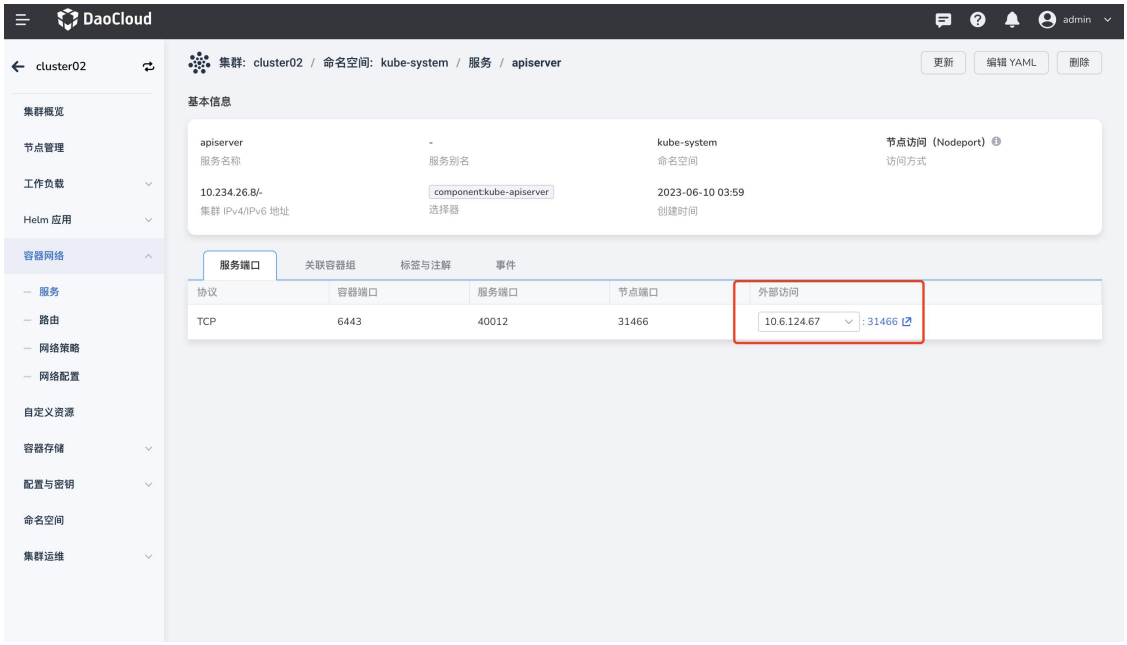
- 集群 cluster01 访问类型选择 NodePort，便于外部访问
- 命名空间选择 kube-system，即 API Server 所在命名空间

- 标签选择器筛选 API Server 组件，可返回查看 API Server 的选择器
- 配置 Service 的访问端口，容器端口为 6443
- 获取该 Service 的外部访问链接

2. 再以同样方式在集群 cluster02 上为 API Server 创建 Service。



创建 service



创建 service

修改集群配置

通过 vi 命令开始编辑 集群 cluster01 和集群 cluster02 的 kubeconfig 文件:

```bash
vi $HOME/.kube/config
```

1. 在两个集群 cluster01 和 cluster02 里分别添加新的 cluster、context、user 信息。

    - 在 clusters 下面添加新的 cluster 信息：两个集群原有的 CA 颁发机构不变；新的 server 地址改为上述创建的 API Server Service 地址；name 改为两个集群本身的名称：cluster01 和 cluster02。

      API Server Service 的地址可以从 DCE5.0 的页面查看或复制，需要使用 https 协议。

- 在 contexts 下面添加新的 context 信息：将 context 中集群的 name 、user、cluster 三个字段的值均修改为两个集群本身的名称：cluster01 和 cluster02 。
- 在 users 下面添加新的 user 信息：两个集群 cluster01 和 cluster02 分别复制本身原有的证书信息，将 name 改为两个集群本身的名称：cluster01 和 cluster02。

2. 在对端集群中分别互相添加已经创建好的 cluster 、context、user 信息。

   如下为操作过程中的 yaml 示例：

```
clusters:
- cluster: #添加本集群 cluster01 `cluster`信息
    certificate-authority-data: {{cluster01}}
    server: https://{{https://10.6.124.66:31936}}
  name: {{cluster01 }}
- cluster: #添加对端集群 cluster02`cluster`信息
    certificate-authority-data: {{cluster02}}
    server: https://{{https://10.6.124.67:31466}}
  name: {{cluster02}}

contexts:
- context: #添加本集群 cluster01 `context` 信息
    cluster: {{cluster01 name}}
    user: {{cluster01 name}}
  name: {{cluster01 name}}
- context: #添加对端集群 cluster02 `context`信息
    cluster: {{cluster02 name}}
    user: {{cluster02 name}}
  name: {{cluster02 name}}
current-context: kubernetes-admin@cluster.local
```

```
users:
- name: {{cluster01}} #添加本集群 cluster01 `user`信息
  user:
    client-certificate-data: {{cluster01 certificate-data}}
    client-key-data: {{cluster01 key-data}}
- name: {{cluster02}} #添加对端集群 cluster02 `user`信息
  user:
    client-certificate-data: {{cluster02 certificate-data}}
    client-key-data: {{cluster02 key-data}}
```

配置集群连通性

执行如下命令验证集群的连通性：

1. 在集群 cluster01 上输入：

   ```
   cilium clustermesh enable --create-ca --context cluster01 --service-type NodePort
   ```

2. 集群 cluster02 开启 clustermesh 执行如下命令：

   ```
   cilium clustermesh enable --create-ca --context cluster02 --service-type NodePort
   ```

3. 在集群 cluster01 上建立连接：

   ```
   cilium clustermesh connect --context cluster01 --destination-context cluster02
   ```

4. 集群 cluster01 出现 connected cluster1 and cluster2!，集群 cluster02 出现 ClusterMesh enabled! 说明两个集群通了。

```
[root@controller-node-1 ~]# cilium clustermesh connect --context cluster01 --destination-context cluster02
✨ Extracting access information of cluster cluster02...
🔑 Extracting secrets from cluster cluster02...
⚠️ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ️ Found ClusterMesh service IPs: [10.6.124.67]
✨ Extracting access information of cluster cluster01...
🔑 Extracting secrets from cluster cluster01...
⚠️ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ️ Found ClusterMesh service IPs: [10.6.124.66]
✨ Connecting cluster cluster01 -> cluster02...
🔑 Patching existing secret cilium-clustermesh...
✨ Patching DaemonSet with IP aliases cilium-clustermesh...
✨ Connecting cluster cluster02 -> cluster01...
🔑 Patching existing secret cilium-clustermesh...
✨ Patching DaemonSet with IP aliases cilium-clustermesh...
✅ Connected cluster cluster01 and cluster02!
[root@controller-node-1 ~]#
```

联通

```
✅ ClusterMesh is already enabled
[root@controller-node-1 ~]# vi $HOME/.kube/config
[root@controller-node-1 ~]# cilium clustermesh connect --context cluster02 --destination-context cluster01
✨ Extracting access information of cluster cluster01...
🔑 Extracting secrets from cluster cluster01...
⚠️ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ️ Found ClusterMesh service IPs: [10.6.124.66]
✨ Extracting access information of cluster cluster02...
🔑 Extracting secrets from cluster cluster02...
⚠️ Service type NodePort detected! Service may fail when nodes are removed from the cluster!
ℹ️ Found ClusterMesh service IPs: [10.6.124.67]
✨ Connecting cluster cluster02 -> cluster01...
🔑 Patching existing secret cilium-clustermesh...
✨ Patching DaemonSet with IP aliases cilium-clustermesh...
✨ Connecting cluster cluster01 -> cluster02...
🔑 Patching existing secret cilium-clustermesh...
✨ Patching DaemonSet with IP aliases cilium-clustermesh...
✅ Connected cluster cluster02 and cluster01!
[root@controller-node-1 ~]#
```

联通

创建演示应用

1. 使用 cilium 官方文档中提供的 rebel-base 应用，复制如下 yaml 文件：

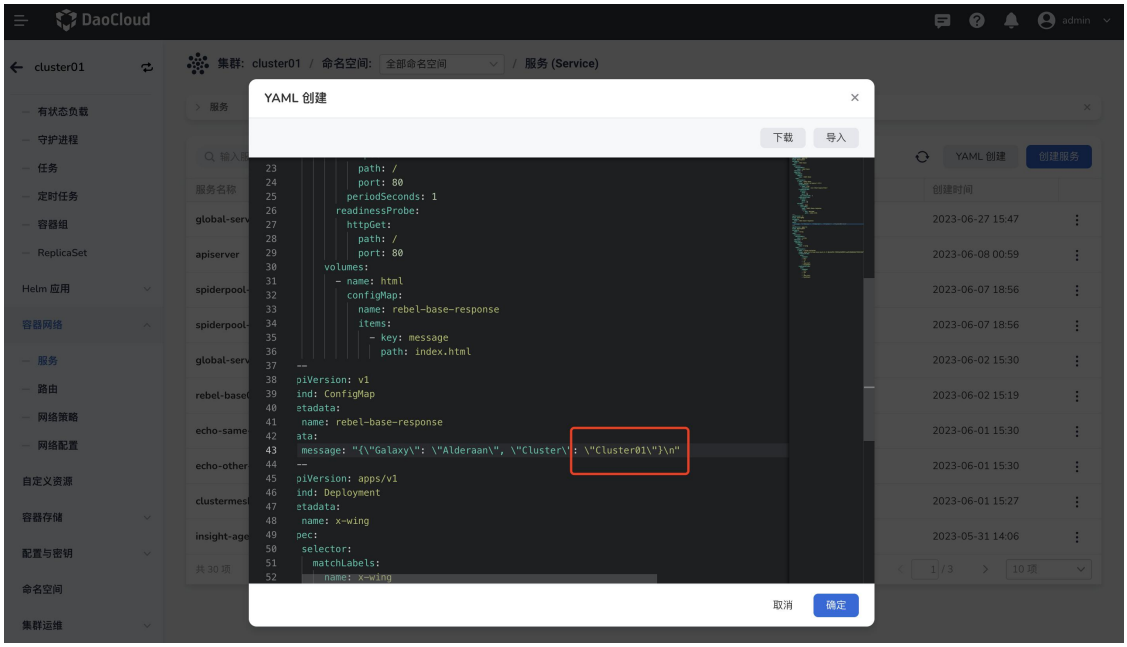
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rebel-base
spec:
  selector:
    matchLabels:
      name: rebel-base
  replicas: 2
  template:
```

```yaml
    metadata:
      labels:
        name: rebel-base
    spec:
      containers:
      - name: rebel-base
        image: docker.io/nginx:1.15.8
        volumeMounts:
          - name: html
            mountPath: /usr/share/nginx/html/
        livenessProbe:
          httpGet:
            path: /
            port: 80
          periodSeconds: 1
        readinessProbe:
          httpGet:
            path: /
            port: 80
      volumes:
        - name: html
          configMap:
            name: rebel-base-response
            items:
              - key: message
                path: index.html
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: rebel-base-response
data:
```

```
  message: "{\"Galaxy\": \"Alderaan\", \"Cluster\": \"Cluster-1\"}\n" #将 Cluster-1 修改为集群一的名称
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: x-wing
spec:
  selector:
    matchLabels:
      name: x-wing
  replicas: 2
  template:
    metadata:
      labels:
        name: x-wing
    spec:
      containers:
      - name: x-wing-container
        image: quay.io/cilium/json-mock:v1.3.3@sha256:f26044a2b8085fc aa8146b6b8bb73556134d7ec3d5782c6a04a058c945924ca0
        livenessProbe:
          exec:
            command:
            - curl
            - -sS
            - -o
            - /dev/null
            - localhost
        readinessProbe:
          exec:
            command:
```

- curl
- -sS
- -o
- /dev/null
- localhost

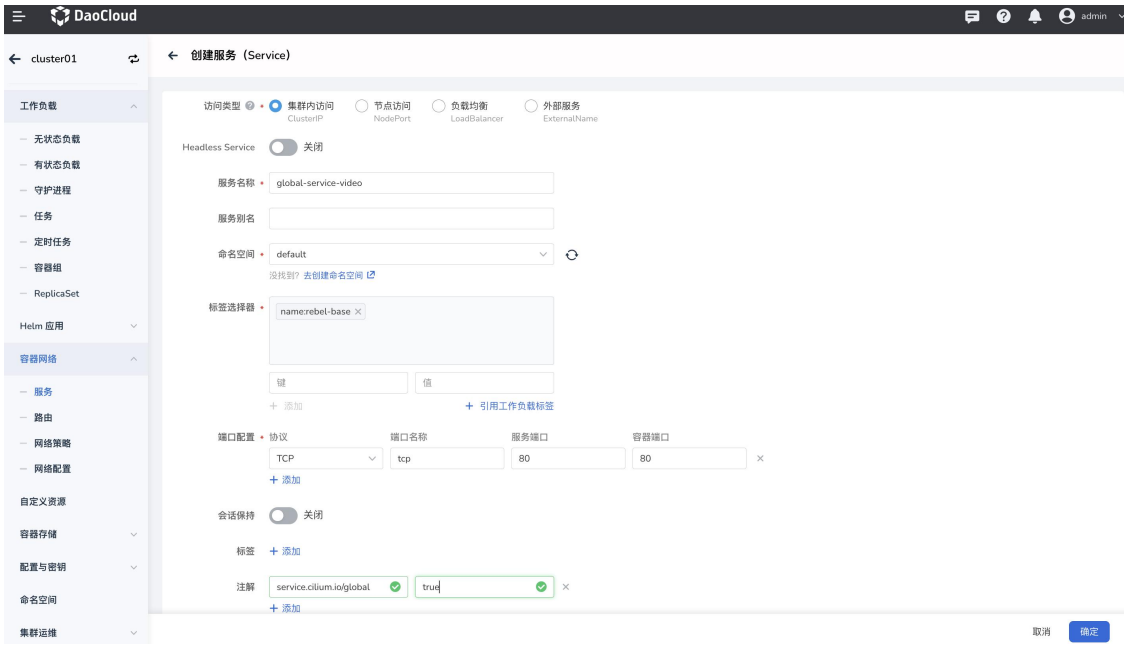
2. 在 DCE 5.0 中通过 yaml 文件快速分别创建两个集群 cluster01 和 cluster02 的应用。



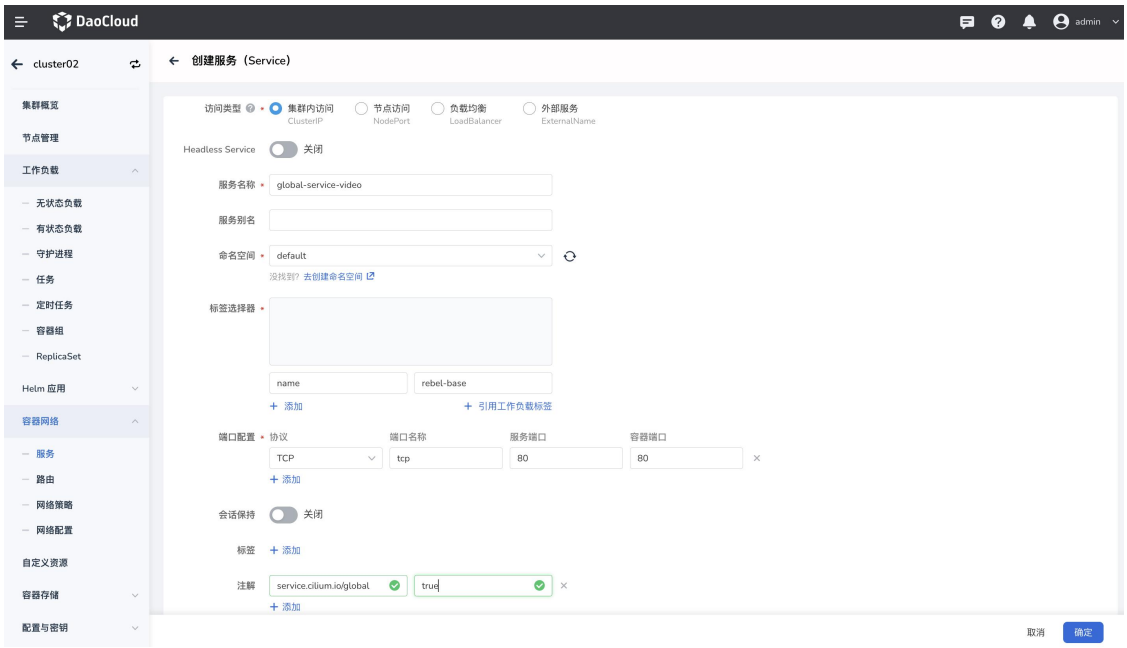
创建应用

分别修改 ConfigMap 的内容，使得访问集群 cluster01 和集群 cluster02 中的 Service 时，返回的数据分别带有 cluster01 和 cluster02 名称的标签。可在 rebel-base 应用中查看容器组标签。

3. 在两个集群 cluster01 和 cluster02 中分别创建一个 global service video 的 Service，指向的是已创建的 rebel-base 应用。



创建 service 应用



创建 service 应用

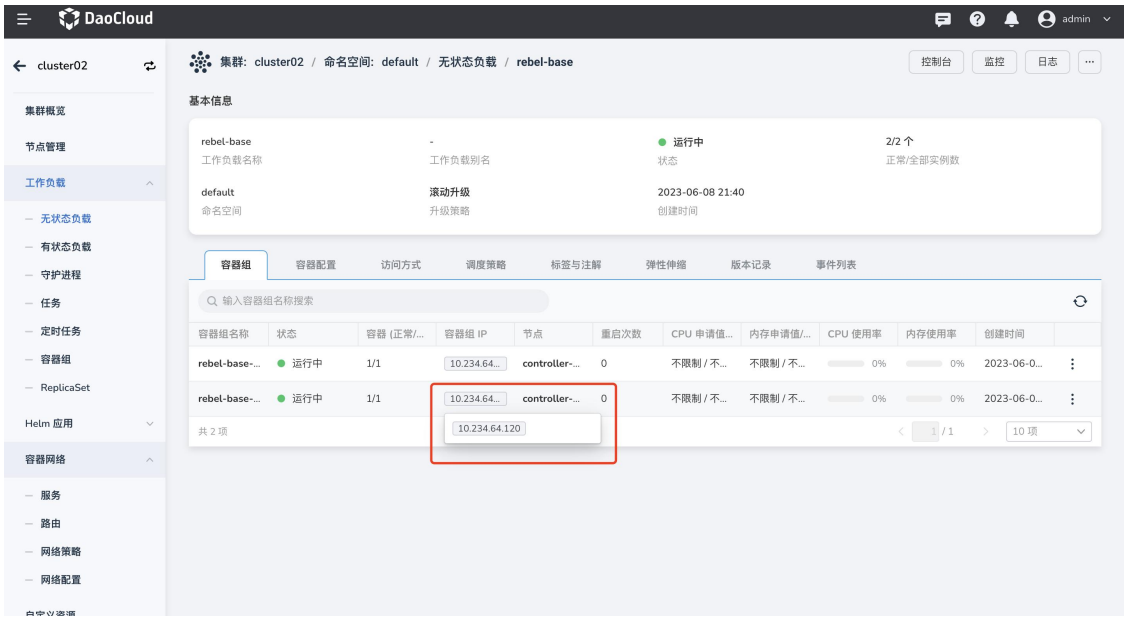
- Service 类型为 ClusterIP
- 添加应用的容器组标签筛选对应的应用
- 配置端口

- 添加注解，使当前的 Service 在全局生效

在创建集群 cluster02 的 Service 时，两个集群的 service name 必须相同，并位于相同的命名空间，拥有相同的端口名称和相同的 global 注解。

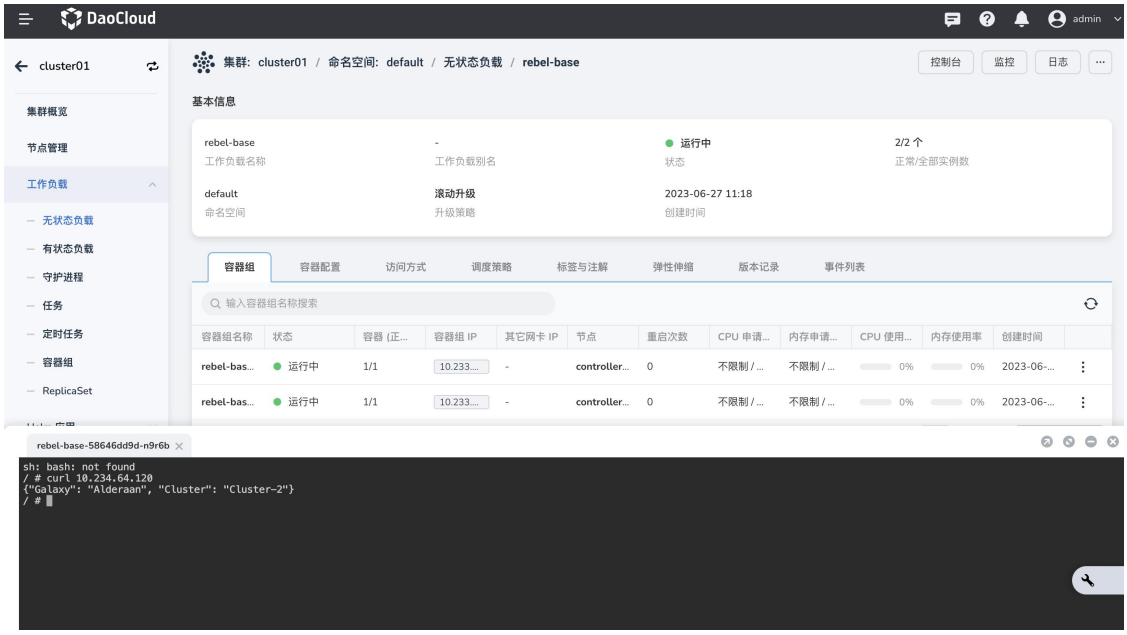
跨集群通信

1. 先查看集群 cluster02 中应用的 Pod IP。



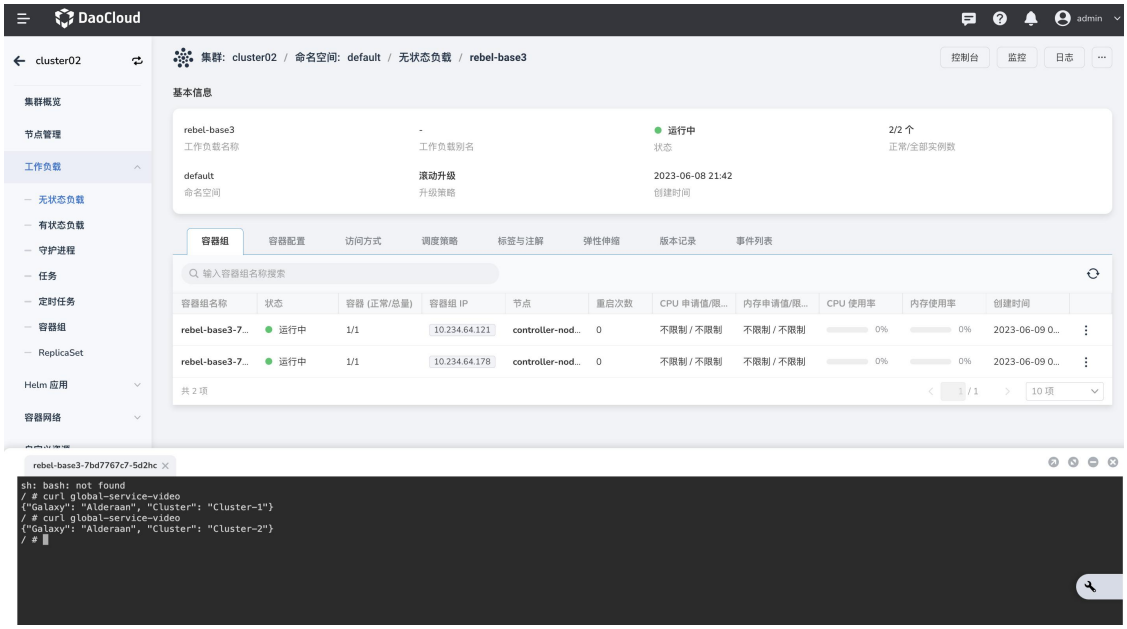
查看 pod ip

2. 进入集群 cluster01 详情，点击应用 rebel-base Pod 控制台，curl 集群 cluster02 应用 rebel-base 的 Pod IP，成功返回 cluster02 信息，说明两个集群中的 Pod 可以相互通信。



pod 通信

3. 查看集群 cluster01 的 Service 名称，进入集群 cluster02 的应用 rebel-base Pod 控制台，curl 对应的 cluster01 的 Service 名称，有些返回内容来自 cluster01，说明两个集群中的 Pod 和 Service 也可以互相通信。



Pod 和 Service 通信

Kubespray 未支持的 Cilium 功能说明

本页说明 Kubespray 不支持的 Cilium 功能。

> 由于 Cilium 功能繁多，本页仅简要介绍几个功能特性。

Egress Gateway

Cilium 通过 CiliumEgressGatewayPolicy 来定义哪些流量离开集群时可以指定相应的节点以及出集群的源 IP。

> 需要注意的是，Cilium 并不自行维护出口的源 IP，且当前只支持 IPv4。

当前 Egress Gateway 与 L7 策略并不兼容。也就是说，当 Egress Gateway 的策略与 L7 策略同时命中一个端点时，Egress Gateway 策略将失效。

开启 Egress Gateway

```
enable-bpf-masquerade: true
enable-ipv4-egress-gateway: true
enable-l7-proxy: false
kube-proxy-replacement: strict
```

使用 Kubean 安装 Cilium 时，可以通过“cilium_config_extra_vars”进行配置。参阅 Egress Gateway 文档。

Cluster Mesh

Cilium 支持 Cluster Mesh 功能，可以将多个 Cilium 集群连接在一起。 该功能打通了各个集群中 Pod 的连通性，支持定义全局 SVC，在多个集群之间进行负载均衡。

> 该功能不支持在创建集群时开启，需在完成集群创建后单独开启。

设置 Cluster Mesh

开启前置条件

- 所有集群的 Cilium 网络模式一样，都是隧道模式或者路由模式。
- 设置 Cilium 集群名称 --set cluster.name 及 ID --set cluster.id，且集群名称与 ID 是唯一的，ID 范围在 1-255。由于 ID 会写入身份中，因此，如果创建集群后修改集群 ID (cm 中对应为 cluster-name、cluster-id)，则需要重启所有的 Pod。
- 所有集群和所有节点的 PodCIDR 范围不冲突
- 所有集群节点间的网络必须是连通的
- 确保在防火墙在中开启与集群间通信的相关端口
- 所有集群都安装了 Cilium CLI
- 所有集群都需要准备一个能够被外部使用的 context
- 集群名称不能使用大写，否则生成的域名是非法的

如果使用路由模式，额外的要求如下：

- 每个集群的 native-routing-cidr 都应该包括所有集群的全部 Pod CIDR 的范围
- 所有节点和集群间的 Pod 必须能直接通信，其中包括三层和四层是连通的。

开启时需选择的服务类型

在某些情况下，可能无法自动识别 SVC 类型，需要手动指定 clustermesh-apiserver 的类型，可以指定为：

- LoadBalance：推荐使用该模式。但前提是集群能支持 LB，否则会一直 pending，等待 EXTERNAL-IP 的分配。
- NodePort：存在一个弊端，如果用于访问的节点出现故障，则需要重新连接到另外一个节点，这可能会造成网络中断。如果所有节点都出现故障，则需要重新连接集群以提取新 IP。
- ClusterIP：需要 ClusterIP 能跨集群路由。

开启 clustermesh

- 在第一个集群开启 clustermesh

  使用 --create-ca 参数在第一个集群开启 clustermesh，同时创建 ca 证书，用于 hubble-rely。将创建好的 Secret ca 证书导出到其他集群。

  ```
  # 开启 clustermesh，同时创建 ca 证书
  cilium clustermesh enable --create-ca --context x1 --service-type NodePort

  # 导出 ca 证书
  kubectl -n kube-system secret cilium-ca -oyaml > cilium-ca.yaml

  # 在其他集群导入 ca
  kubectl apply -f cilium-ca.yaml
  ```

- 在其他集群开启 clustermesh

  ```
  # 根据需要指定 --service-type 类型
  cilium  clustermesh enable --context x2 --service-type NodePort
  ```

连接集群

只需要在一个集群中执行连接到其他集群的操作。

```
cilium  clustermesh connect --context x1 --destination-context x2
```

负载均衡和服务发现

在 SVC 中添加注解，让其成为全局的 SVC，可被其他集群发现或访问。同时还可指定本集群的服务能否被其他集群访问及服务的负载均衡方式。

- io.cilium/global-service: “true/false”：将 SVC 定义为全局的 SVC，可被其他集群发现
- io.cilium/shared-service: “true/false”：有一个同名的全局 SVC，但本集群的 SVC 值设置为 false 时，不可被其他集群发现或访问

- io.cilium/service-affinity: “none/local/remote/”：SVC 负载均衡的方式。默认为 none，表示在所有集群中负载均衡。local 表示优先负载均衡到本地集群。remote 表示优先负载均衡到其他集群。

数据存储

开启 clustermesh 时，会启动一个 clustermesh APIServer 的 Pod，用于集群间数据同步。同时还会启动一个 ETCD 用于数据存储。参照以下命令查看数据：

```
# 进入 clustermesh APIServer Pod，配置 ETCD 相关相关证书
alias etcdctl='etcdctl --cacert=/var/lib/etcd-secrets/ca.crt --cert=/var/lib/etcd-secrets/tls.crt --key=/var/lib/etcd-secrets/tls.key '

# 身份存储路径
etcdctl get --prefix cilium/state/identities/v1

# 已使用 IP 存储路径
etcdctl get --prefix cilium/state/ip/v1/<NS>

# 节点
etcdctl get --prefix cilium/state/nodes/v1

# SVC
etcdctl get --prefix cilium/state/services/v1/<clusterName>/<NS>
```

请参阅 Cluster Mesh 文档。

Service Mesh

当前 Cilium 并不支持直接通过修改某些参数来直接开启 Service Mesh。 只支持通过 Cilium CLI 或 Helm 来开启。所以使用 Kubean 或 Kubespray 安装的集群，无法通过配置参数的方式来启用该功能。

请参阅 Service Mesh 文档。

带宽管理

当 Kubespray <= v2.20.0 时，只能使用“cilium_config_extra_vars”方式将“enable-bandwidth-manager”变量设置为 true 来开启。 之后的版本可通过“cilium_enable_bandwidth_manager”直接开启。

请参阅 Bandwidth Manager 文档。

替换 kube-proxy

Kubespray 支持使用参数“cilium_kube_proxy_replacement”启用该功能。 但是相关的高级功能并不支持参数配置。Cilium 默认关闭这些高级功能，开启方式较复杂。这里对部分高级功能做一个简述。

> 后续高级配置涉及到的参数，均为 Helm 参数。

Maglev 哈希一致性

开启 Maglev 哈希一致性：

```
--set loadBalancer.algorithm=maglev    # 开启
```

针对外部流量，根据五元组做哈希计算得到后端 Pod 地址，相同的五元组计算的结果都是一致的，所以不需要在各节点之间同步状态。 需要注意的是该策略只对外部流量生效，由于内部请求直接到后端，所以不受 Maglev 限制。该策略也可以兼容 Cilium 的 XDP 加速技术。

该算法有两个参数可调整：

- maglev.tableSize：指定每个单一服务的 Maglev 查询表的大小。 Maglev 建议表的大小（M）要远远大于预期的最大后端数量（N）。 在实践中，这意味着 M 应该大于 100*N，以保证在后端变化时，重新分配的差异最多只有 1%。 M 必须是一个素数。Cilium 使用默认的 M 大小为 16381。 以下的 M 大小作为 maglev.tableSize Helm 选项被支持。 支持的值有 251、509、1021、2039、4093、8191、16381、32749、65521、131071

- maglev.hashSeed：建议设置 maglev.hashSeed 选项，促使 Cilium 不依赖固定的内置种子。 种子是一个 base64 编码的 12 字节的随机数，可运行以下命令生成一次：

  ```
  head -c12 /dev/urandom | base64 -w0
  ```

  集群中的每个 Cilium 代理必须使用相同的哈希种子，Maglev 才能工作。

  具体设置方式为：

  ```
  --set maglev.tableSize=65521 \
  --set maglev.hashSeed=$SEED \
  ```

!!! note

与 loadBalancer.algorithm=random 的默认值相比，启用 Maglev 将导致 Cilium 管理的每个节点上的内存消耗更高。这是因为随机不需要额外的查询表，但是，随机的后端选择不一致。

直接 SVC 返回 (DSR)

启用 DSR 模式：

```
--set tunnel=disabled \
--set autoDirectNodeRoutes=true \
--set loadBalancer.mode=dsr \
```

针对外部流量。必须运行在路由模式下，可以保留源 IP。 当流量到达 LB 或者 NodePort 的节点时，转给后端 EP 时不做 SNAT，应答流量也不再经过 LB 或者流量进来的节点，而是直接返回给客户端。 所以这需要 Pod 能够与外部路由联通，Cilium 不能使用隧道模式。因此，流量返回时就少了一跳，起到了加速的作用，而且保留了源 IP。

由于一个 Pod 可以被多个 SVC 使用，所以返回的 SVC IP 及端口信息需告知 EP。 Cilium 将此信息编码在 Cilium 特定的 IPv4 选项或 IPv6 目标选项扩展报头中，代价是 MTU 值变小。 对于 TCP 服务，Cilium 只对 SYN 数据包的

SVC IP/端口进行编码，后续的数据报头中并不会携带这些信息。所以要关闭源/目的检测功能。

此外，由于来去的路径不一致，所以会出现路由不对称的现象，有一些 iptables 规则会将这些流量丢弃。

混合 DSR 和 SNAT 模式

配置混合模式：

```
--set tunnel=disabled \
--set autoDirectNodeRoutes=true \
--set loadBalancer.mode=hybrid \
```

在混合模式中，对 TCP 执行 DSR，对 UDP 执行 SNAT。 这样可以避免手动修改 MTU，又可以减少 TCP 的跳数。

loadBalancer.mode 默认为 snat，同时也支持 dsr、hybrid 模式。

XDP 加速

开启 XDP 加速：

```
--set loadBalancer.acceleration=native \
```

Cilium 可以对 NodePort、loadBalancer 及对外可访问的 SVC 都提供 XDP 加速支持。XDP 加速需要底层驱动支持。 该功能支持 loadBalance 的 DSR、SNAT 及 Hybrid 模式。由于 XDP 加速阶段很早，所以使用 tcpdump 抓取不到数据包。

> 只有网卡驱动支持 XDP 时，该功能才能使用。 如果 Cilium 自动检测使用多个网卡来暴露 NodePort，或者指定了多个 device，则所有的网卡驱动都要支持 XDP。

查看设备使用的驱动：

```
$ethtool -i eth0 | grep driver
driver: vmxnet3       # 网卡驱动
```

当前支持的驱动列表可查阅 LoadBalancer & NodePort XDP Acceleration。

在 Pod 命名空间中绕过 Socket LoadBalancer

在 kube-proxy-free 环境中绕过 Socket LB 的配置：

```
--set tunnel=disabled \
--set autoDirectNodeRoutes=true \
--set socketLB.hostNamespaceOnly=true
```

Cilium 在 Pod 中默认访问的是 SVC IP，因此在 Pod 中就会做后端选举，直接连接到后端地址。 应用层看到的还是连接的 SVC IP，但是底层其实是对应的后端地址。 如果需要依赖 SVC IP 做负载时，此功能会失效，可以将通过以上配置将该功能关闭。

开启拓扑感知提示

开启拓扑感知提示：

```
--set loadBalancer.serviceTopology=true \
```

Cilium kube-proxy 也实现了 K8s 服务 Topology Aware Hints 功能，可以让请求更偏向于同一区域的后端端点。

邻居发现

Cilium 1.11 版本后，已经将邻居发现的库删除，完全依赖于 Linux 内核来实现邻居发现。 在 5.16 及以上版本的内核中，通过“managed”功能实现，并用“extern_learn”来标记 arp 记录，以防被内核垃圾回收。 对于低版本的内核，通过 cilium-agent 定期将新节点的 IP 地址写入 Linux 内核中以进行动态解析。 默认是 30s，可通过以下参数进行设置：

```
--set --arping-refresh-period=30s \
```

外部访问 clusterIP

允许外部访问 ClusterIP Service：

```
--set bpf.lbExternalClusterIP=true \
```

Cilium 默认不允许外部访问 ClusterIP SVC。可以通过 bpf.lbExternalClusterIP=true 启用。但需要自行打通相关的路由。

更多内容请参阅替换 kube-proxy 高级配置。

IngressClass

IngressClass 代表 Ingress 实例 的类，当创建 Ingress 规则时，可在 Ingress 的 spec 中引用。主要的适用场景如下：

适用场景

- 同一个集群中，有内部 Ingress 和外部 Ingress 需求
- 同一个集群同一个命名空间中，不同团队部署不同的应用，使用不同 Ingress 实例
- 同一个集群中，同一团队部署不同应用对 Ingress 实例资源配比有要求
    - 例如某些业务需要独享 4C 4G 的数据面网关资源

前提条件

- 已经完成 Ingress nginx 实例部署，并设置了 IngressClassName。
- 已获取对应的 IngressClassName。

使用操作

通过 YAML 创建 Ingress 并指定 IngressClass

通过 YAML 创建 Ingress 时，如需要指定 ingressClass 时，需要通过 ingressClassName 指定。 注解 kubernetes.io/ingress.class 已经废弃。

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dao-2048
spec:
  ingressClassName: contour # 指定 ingress class 名称
  rules:
  - http:
      paths:
```

```
      - path: /
        pathType: Prefix
        backend:
          service:
            name: dao-2048
            port:
              number: 80
```

通过界面创建 Ingress 并指定 IngressClass

如通过图形化创建路由（Ingress ）时，可直接在界面输入对应的 IngressClassName。

默认 IngressClass

每个集群都可以有一个默认的 IngressClass。存在默认 IngressClass 时（ 创建 Ingress 实例时，开启 DefaultIngressClass ），创建 Ingress 时可以不指定 ingressClassName 字段。

ingressclass.kubernetes.io/is-default-class 注解可以用来标明一个 IngressClass 作为默认类。一个集群最多只能设置一个。 当某个 IngressClass 资源将此注解设置为 true 时，没有指定 class 的新 Ingress 资源将被分配到此默认类。

QA

问：不同租户如何使用不同 Ingress 负载流量而且还不用指定 ingressClassName?

答：可以通过指定 --watch-namespace 的方式，不同的实例 watch 不同的 namespace。 ingress-nginx 可以通过 helm 安装时指定 controller.scope.enabled=true 和 --set controller.scope.namespace=$NAMESPACE， 更多信息可以参考 scope。

Ingress 使用范围（Scope）

IngressClass Scope 用于指定 Ingress 实例的使用范围为集群级、命名空间级和工作空间级。

适用场景

- 同一个集群中，共享同一个 Ingress 实例，可设置平台级 Ingress 实例。
- 单一命名空间独享一个 Ingress 实例实现负载隔离，可设置命名空间级 Ingress 实例。
- 工作空间独享一个 Ingress 实例实现负载隔离，可设置租户级 Ingress 实例，工作空间中对应当前集群下的命名空间，其中 Pod 都可以接收到由该负载均衡分发的请求。

如果在同一个集群内，同一命名空间内有不同应用需要使用不同 Ingress 实例的需求，请参考 IngressClass。

平台级 Ingress 实例

创建 Ingress 实例时，如果启用 Ingress Scope，创建后的 IngressClass 资源在以下两种情况下， Ingress 实例的使用范围都为平台级：

1. 仅设置了 parameters 但是未设置 .spec.parameters.scope
2. 设置了 .spec.parameters.scope 为 cluster

```yaml
#示例
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb-1
spec:
  controller: example.com/ingress-controller
  parameters:
```

```
    scope: Cluster # 指定 Ingress 实例范围为 Cluster
    apiGroup: k8s.example.net
    kind: ClusterIngressParameter #指定 Ingress 实例 Kind 为 ClusterIngressParameter
    name: external-config-1
```

命名空间级 Ingress 实例

当创建 Ingress 实例时，如果启用 Ingress Scope，IngressClass 设置了 .spec.parameters，并且设置 .spec.parameters.scope 为 Namespace，那么 Ingress 实例的 Ingress Class 指向为命名空间级，需要指定待使用的命名空间。

命名空间级的 Ingress 实例，相当于管理员将 Ingress 的使用权限下发给到某个命名空间，可以实现资源隔离。

```
#示例
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb-2
spec:
  controller: example.com/ingress-controller
  parameters:
    scope: Namespace # 指定 Ingress 实例范围为 Namespace
    apiGroup: k8s.example.com
    kind: IngressParameter # 指定 Ingress 实例 Kind 为 IngressParameter
    namespace: default # 指定待使用的 Namespace
    name: external-config
```

租户级 Ingress 实例

当创建 Ingress 实例时，如果启用 Ingress Scope，IngressClass 设置了 .spec.parameters，并且设置 .spec.parameters.scope 为 namespaceSelector，并输入的 Label 为 workspace.ghippo.io/id: '1235'(其中 12345 为指定的工作空间

workspace ID)，那么 Ingress 实例的 Ingress Class 指向为租户级，适用范围为 workspace01 中所有在当前集群的命名空间。

租户级的 Ingress 实例，相当于管理员将 Ingress 的使用权限下发给到某个工作空间，从而实现租户资源隔离。

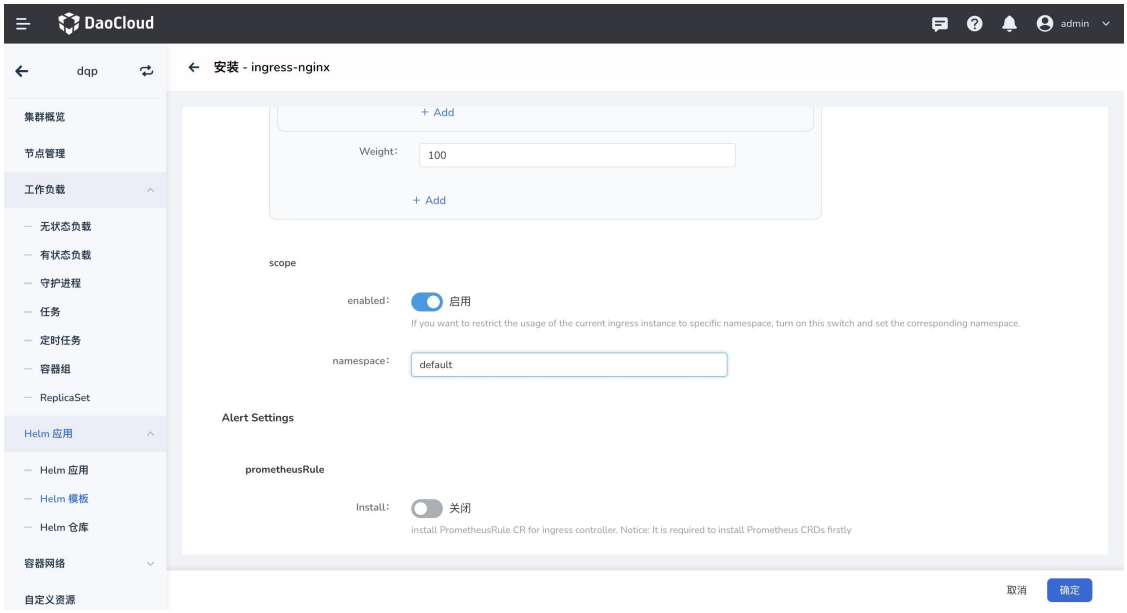
```yaml
#示例
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: external-lb-2
spec:
  controller: example.com/ingress-controller
  parameters:
    scope: Namespace # 指定 Ingress 实例范围为 Namespace
    apiGroup: k8s.example.com
    kind: IngressParameter # 指定 Ingress 实例 Kind 为 IngressParameter
    namespaceSelector: workspace.ghippo.io/id: '1235' # 指定待使用的工作空间 ID
    name: external-config
```

如何部署平台/命名空间级 Ingress 实例?

可以通过指定 --watch-namespace 的方式，不同的实例 watch 不同的命名空间。 若 ingress-nginx 实例通过 Helm 安装，需通过指定 controller.scope.enabled=true 和 --set controller.scope.namespace=$NAMESPACE 开启并设置平台/命名空间级 Ingress。

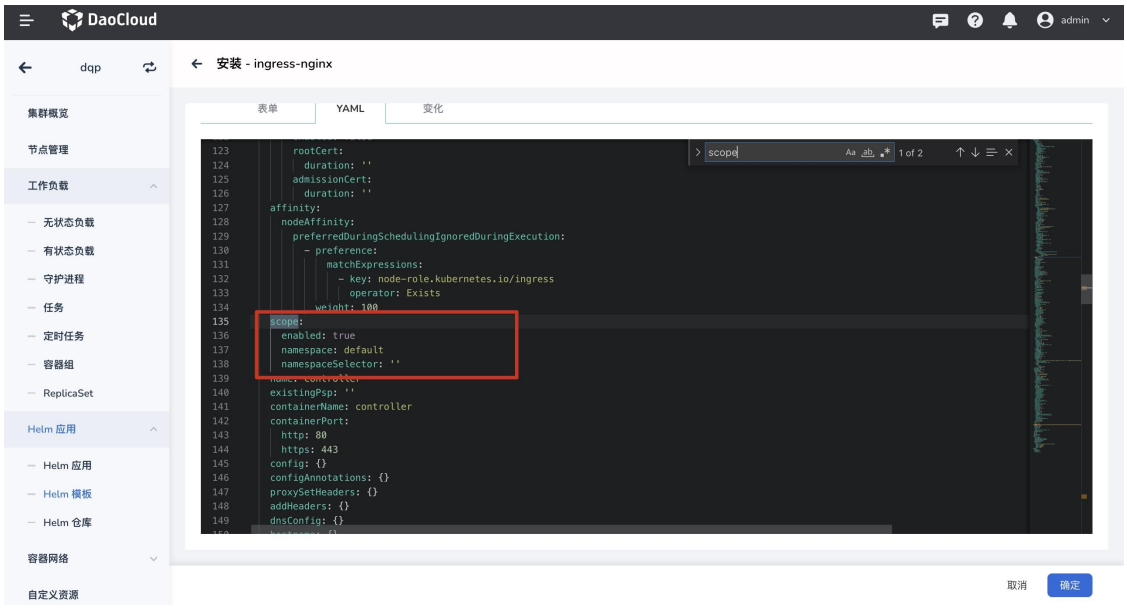
1. 平台级 Ingress 实例：创建 Ingress 实例时， scope 关闭，则创建的 Ingress 实例为 平台级。
2. 命名空间 Ingress 实例：创建 Ingresss 实例时，scope 关闭，并在 namespace 中指定对应的命名空间，则创建的 Ingress 为 命名空间级。

如下示例，创建的 Ingress-nginx 为 Default 独享：



scope01

对应的 value.yaml 中的配置信息：

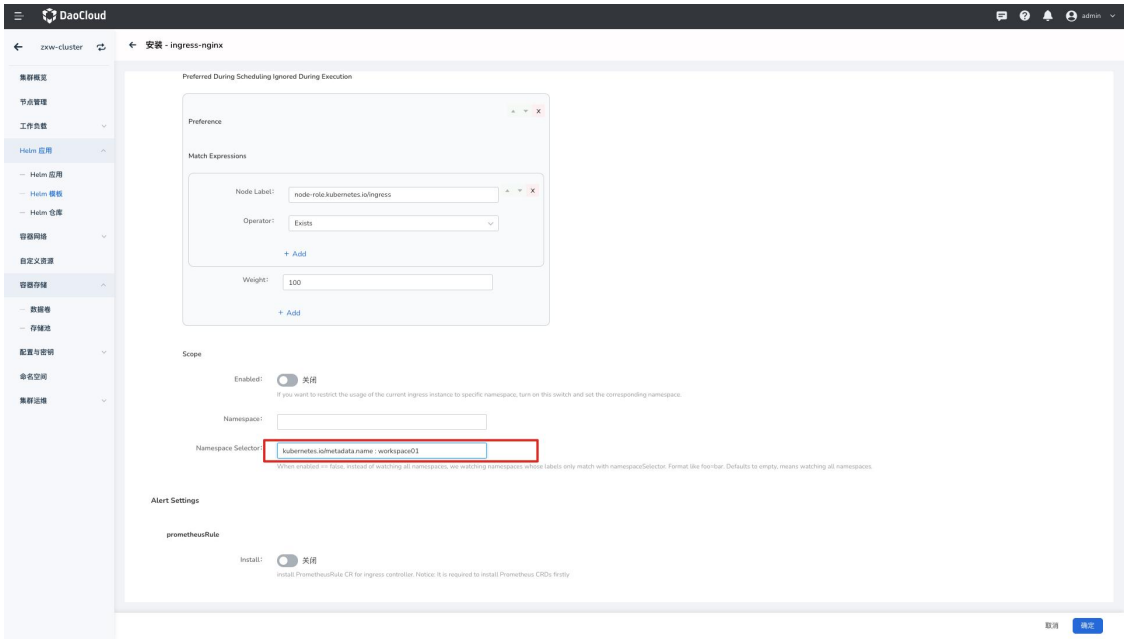


scope02

如何部署租户级 Ingress 实例？

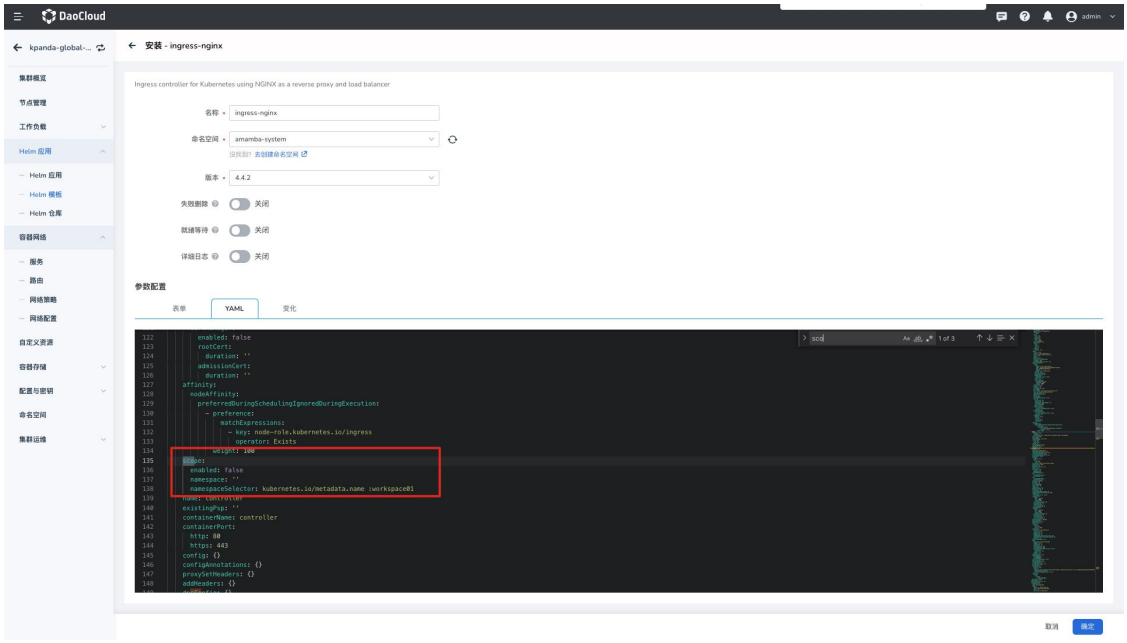
部署负载均衡器时，指定给某个工作空间使用后，此工作空间中对应当前集群下的命名空间，其中 Pod 都可以接收到由该负载均衡分发的请求。

部署 Ingress-Ngnix 时 指定 在 Namespace Selector 中输入 kubernetes.io/metadata.name :workspace01,创建后的 Ingress 实例为 工作空间 workspace01 独享。



工作空间 Ingress

对应的 value.yaml 中的配置信息：



workspaceingress02

Ingress 实例部署后，可在对应的命名空间中创建 Ingress 规则，并选择对应实例的 Ingress Class 进行使用。

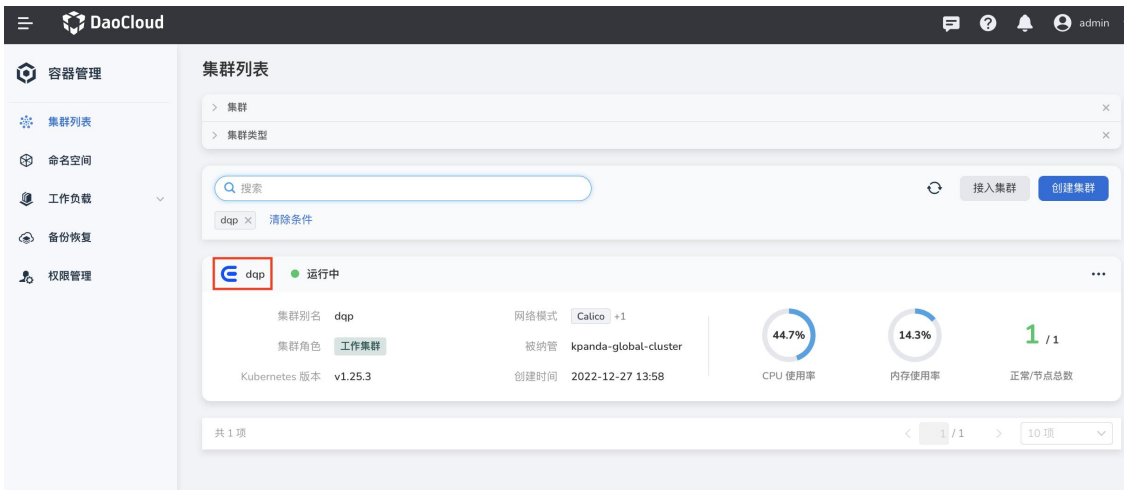
更多信息可以参考 scope。

安装 ingress-nginx

本页介绍如何安装 ingress-nginx 组件。

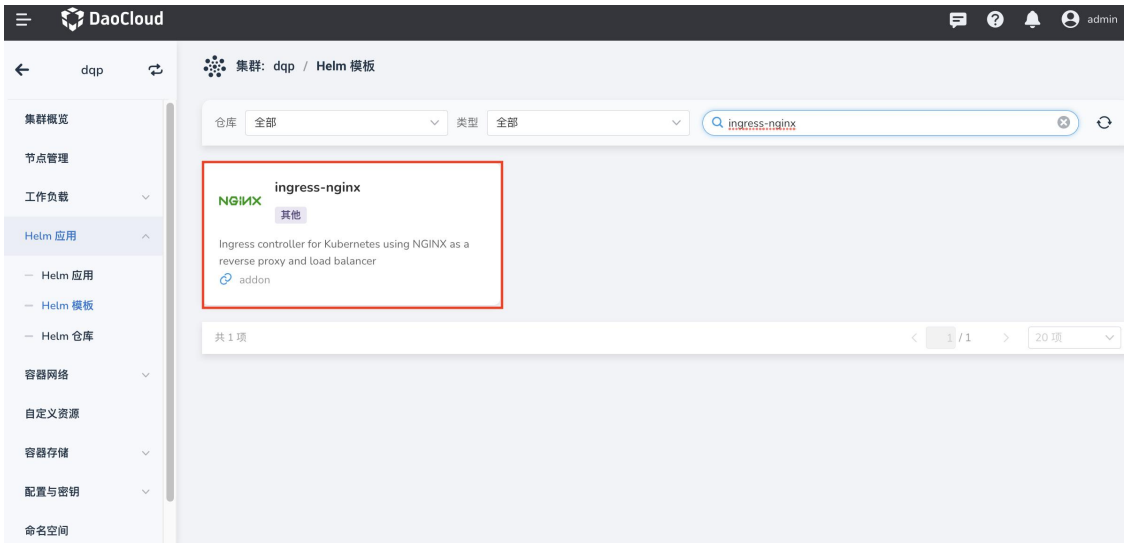
请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 ingress-nginx。

1. 在左侧导航栏点击容器管理—>集群列表，然后找到准备安装 ingress-nginx 的集群名称。



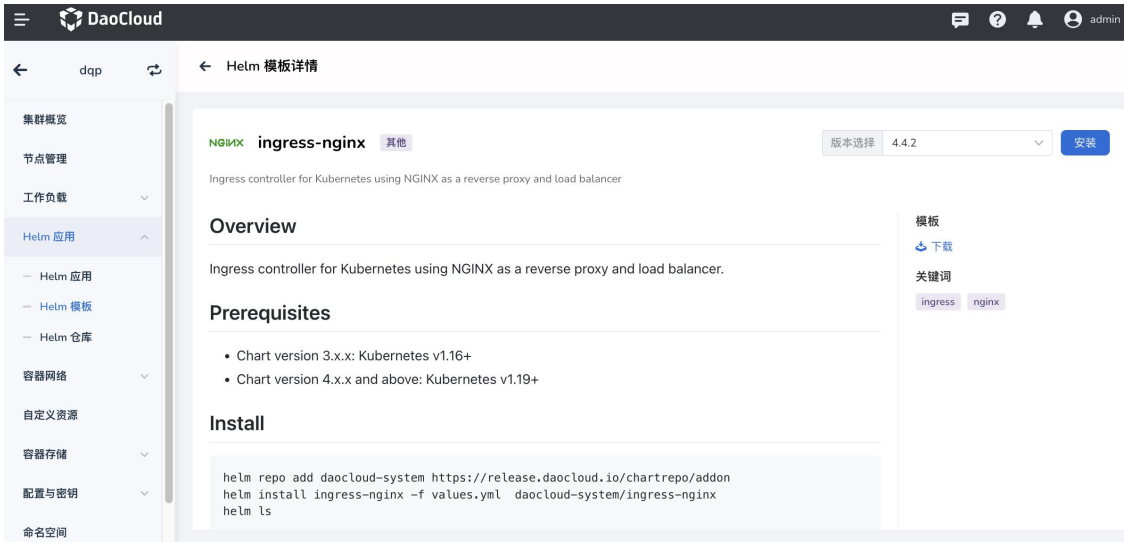
cluster

2. 在左侧导航栏中选择 Helm 应用 -> Helm 模板，找到并点击 ingress-nginx。



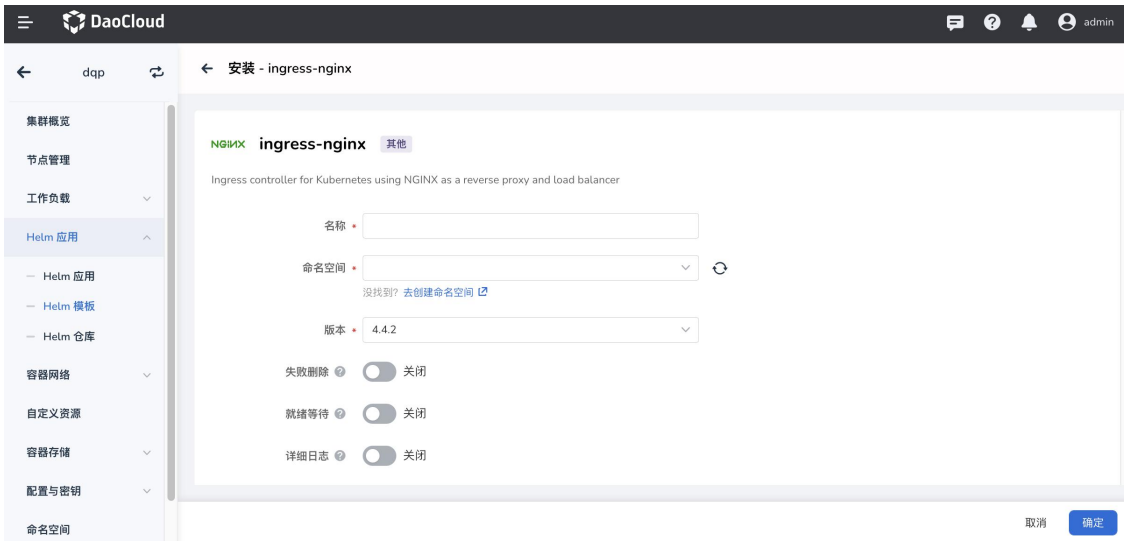
helm

3. 在版本选择中选择希望安装的版本，点击安装。



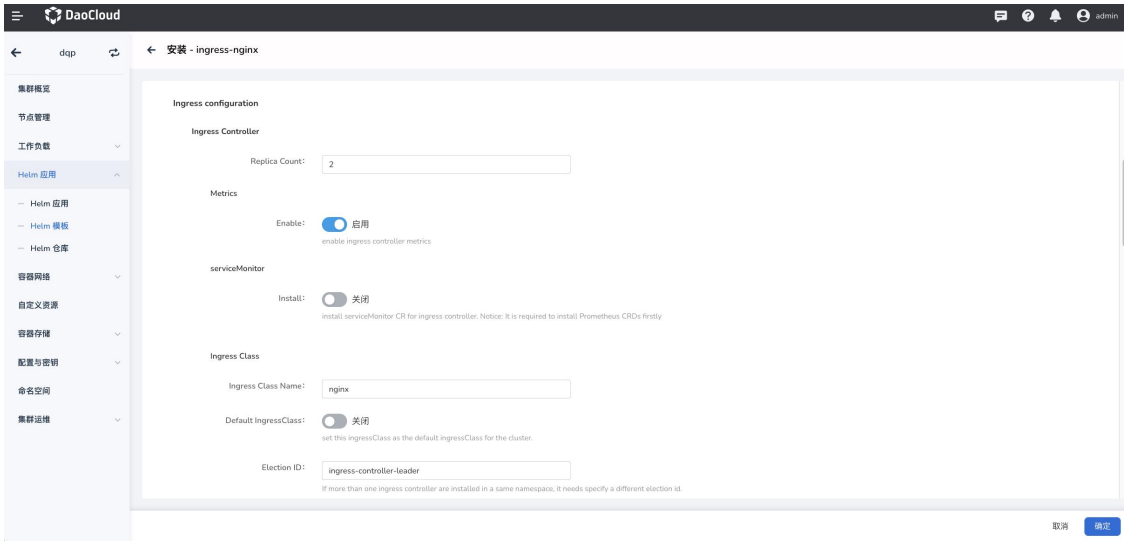
version

4. 在安装界面，填写所需的安装参数。



parameter1

在以上界面中，输入部署后的应用名称、命名空间以及部署的选项。

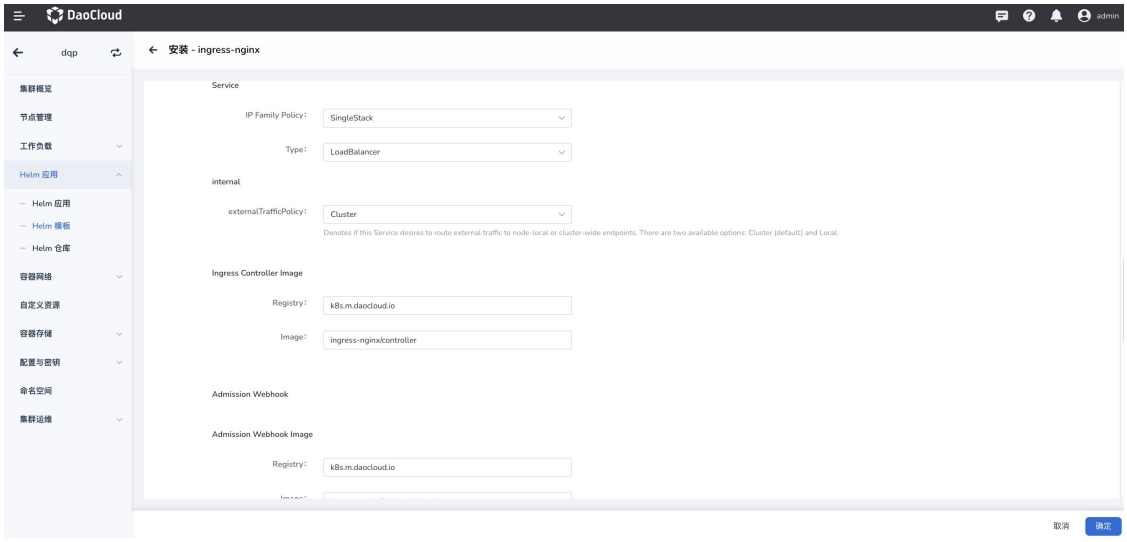


parameter2

上图中的各项参数说明：

- Ingress Controller –> Replica Count：配置副本数量，推荐配置为 2，以提高可用性。对于 POC 体验可以配置为 1，以减少资源占用。

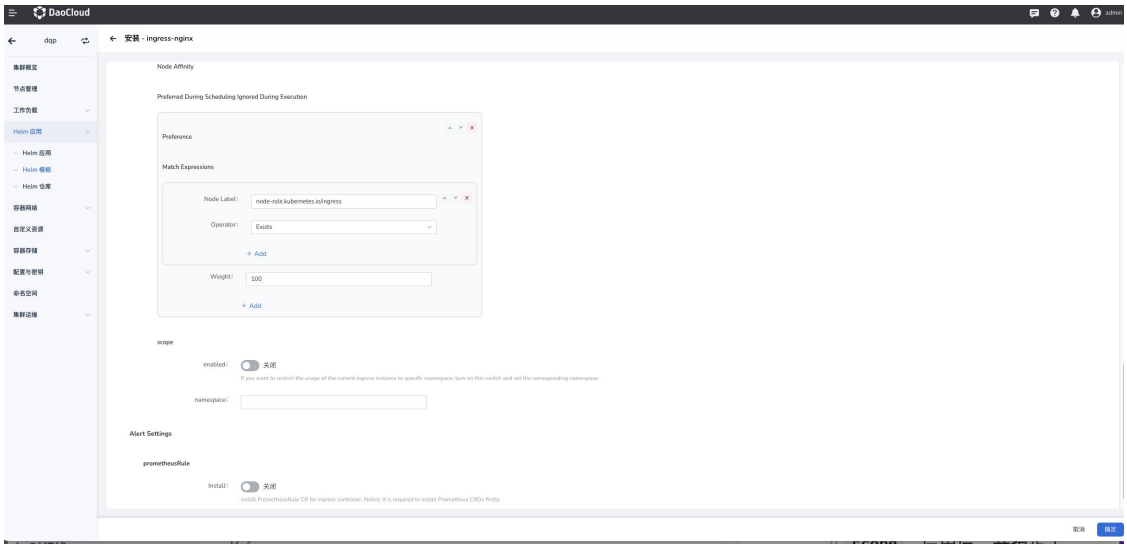
- Ingress Controller –> Metrics：开启后，Controller Pod 会暴露 metrics 接口，监控的 Agent 可以采集指标数据，通过监控告警提高业务和服务的可靠性。
- Ingress Controller –> ServiceMonitor：要求集群内已部署 Insight 组件，或者 Prometheus Operator。开启后会在后端创建对应的 ServiceMonitor CR。
- Ingress Controller –> Ingress Class –> Ingress Class Name：配置 Ingress Class 名称。Ingress CR 则可以通过 ingressClass 字段指定相同的 Class 名称，以选择使用此套 Ingress 实例来生效访问路由。当集群中拥有多套 Ingress 时，甚至在同一租户拥有多套 Ingress 的时候，通过 Ingress Class 来区分 Ingress 非常方便。详细使用方式请参考：如何使用 Ingress Class
- Ingress Controller –> Ingress Class –> Default IngressClass：设置 Ingress Class 为默认 Class。开启该选项后创建 Ingress CR 时无需显式指定 ingressClass 字段。Kubernetes 自动更新 Ingress 的字段为默认 Class。同一个集群只能有一个默认 Ingress Class。
- Ingress Controller –> Ingress Class –> Election ID：同一个租户部署多套 Ingress 时，需要保证此名称不会重复。



parameter3

上图中的各项参数说明：

- Ingress Controller –> Service –> IP Family Policy：Service 的 IP 单双栈设置，可以根据业务需求开启。
- Ingress Controller –> Service –> Type：配置 Service 类型。如果设置为 LoadBalancer，需要集群中安装相应的 LB 组件，例如 MetalLB。
- Ingress Controller –> Ingress Controller Image：配置镜像仓库和镜像名称。
- Ingress Controller –> Admission Webhook Image：自定义 Webhook Pod 的镜像仓库和镜像名称。



parameter4

上图中的各项参数说明：

- Ingress Controller –> Node Affinity –> Preferred During Scheduling Ignored During Execution：通过软亲和性指定调度规则。
- Ingress Controller –> scope：该选项启用，并指定了 Namespace 后，创建的 Ingress 实例的使用范围仅限于指定的命名空间。默认关闭，关闭时为集群级 Ingress。更多详情请参考：Ingress 使用范围（Scope）。

- Alert Settings -> PrometheusRule：配置创建告警规则 Prometheus CR。该选项依赖集群部署的 Prometheus Operator。

5. 对于更高级的配置可以通过点击 Tab 选项卡中 YAML 以通过 YAML 方式进行配置。 点击右下角确定按钮即可完成创建。

镜像流量

流量镜像是验证服务功能和性能的有力工具，可以通过复制实时流量并发送到镜像服务，无需造数据且不会影响线上访问。

配置项	描述	默认值
nginx.ingress.kubernetes.io/mirror-target	指定流量目标地址。支持 Service 和外部地址，例如设置为 https://test.env.com/$request_uri，$request_uri 可以选择将原始请求的 URI 添加到目标 URL 的末尾。	""
nginx.ingress.kubernetes.io/mirror-request-body	是否镜像请求流量的 Body。	on
nginx.ingress.kubernetes.io/mirror-host	指定用于转发请求 Host 信息。	""

日志配置

本页说明 Ingress Nginx 全局设置中用于配置日志的选项，例如日志输出路径，输出格式，日志等级等。

配置项	描述	默认配置
access-log-path	用于设置访问日志的输出位置。建议将日志输出到 stdout，以便在容器平台上进行方便的日志收集。	默认为 /var/log/nginx/access.log，其是 /dev/stdout 的符号链接。
error-log-path	用于设置错误日志的输出位置。建议将日志输出到 stderr，以便在容器平台上进行方便的日志收集。	默认为 /var/log/nginx/error.log，其是 /dev/stderr 的符号链接。
disable-access-log	关闭访问日志输出。	N/A
http-access-log-path 和 stream-access-log-path	分别控制 http 和 stream 的日志。	默认为：“”，若无指定，则使用 access-log-path 。
enable-access-log-for-default-backend	启用默认后端的日志访问。	默认值为 false。
error-log-level	错误日志等级，支持设置为 debug, info, notice, warn, error, crit, alert, emerg。这些日志级别是按照严重程度的增加顺序排列的，例如级别 crit 将导致 crit, alert 和	默认为 notice。

配置项	描述	默认配置
	emerg 消息被记录下来。	
log-format-escape-none	设置是否完全禁止变量中的字符转义参数（true）或由 log-format-escape-json 控制（false） 设置 nginx 日志格式。	N/A
log-format-escape-json	在输出格式为 JSON 的场景，对渲染的值进行转义。	默认为 false。
log-format-stream	设置 nginx stream 日志输出格式。	N/A
skip-access-log-urls	过滤掉出现在 NGINX 访问日志中的 URL 列表。最佳实践是过滤 /health。这种健康检查 URL 很有用，因为它们会使阅读日志变得复杂。	默认为空。
enable-syslog	启用 syslog 功能访问日志和错误日志。	默认为 false。
log-format-upstream	设置 nginx 日志格式，可使用的字段可以参考 Embedded Variables。以下是一个设置为 json 输出的例子。	

```
log-format-upstream: '{"time": "$time_iso8601", "remote_addr": "$proxy_protocol_addr", "x_forwarded_for": "$proxy_add_x_forwarded_for", "request_id": "$req_id", "remote_user": "$remote_user", "bytes_sent": $bytes_sent, "request_time": $request_time, "status": $status, "vhost": "$host", "request_proto": "$server_protocol",
```

```
  "path": "$uri", "request_query": "$args", "request_length": $request_length, "dur
ation": $request_time,"method": "$request_method", "http_referrer": "$http_referer
",
  "http_user_agent": "$http_user_agent" }'
```

跨域

什么是跨域？

跨域是指在浏览器中，不同域名下的资源之间的请求会遇到跨域限制。

配置跨域示例

在 Kubernetes 安装 nginx-ingress 后，可以通过配置 Ingress CR 的 Nginx 配置来处理跨域问题。具体方法如下：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/enable-cors: "true"
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: example-service
            port:
              name: http
```

重新部署 Ingress CR：

```
kubectl apply -f example-ingress.yaml
```

这样，nginx-ingress 就会在每个请求的响应头中添加 Access-Control-Allow-Origin: *，实现跨域。

高级配置

- 配置跨域接受哪些方法

  通过 nginx.ingress.kubernetes.io/cors-allow-methods 控制跨域接受哪些方法，默认为：GET, PUT, POST, DELETE, PATCH, OPTIONS。

- 配置预检请求的缓存时长

  nginx.ingress.kubernetes.io/cors-max-age 用于配置浏览器在多长时间内，不再向服务器发送预检请求（Preflight Request）。

  预检请求是浏览器在发送跨域请求前，先向服务器发送的一个询问请求，询问服务器是否允许该跨域请求。

  通过配置配置该选项，可以减少对服务器的询问次数，从而提高网页性能。

- 配置是否允许在跨域期间发送凭证

  nginx.ingress.kubernetes.io/cors-allow-credentials 用于配置是否允许浏览器发送凭证（Credentials）。

  Credentials 包括诸如 Cookie、HTTP Authentication 或 Client-side SSL certificates 等数据。如果允许浏览器发送 Credentials，那么必须在服务器响应头中添加 Access-Control-Allow-Credentials: true。

- 配置服务器允许请求的来源

  nginx.ingress.kubernetes.io/cors-allow-origin 用于配置服务器允许请求的来源（origin）。

  当浏览器发送一个请求时，会发送一个 origin 头，告知服务器请求来源。服务器可以通过检查请求头中的 origin 字段，判断是否允许跨域请求。

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/cors-allow-origin: "https://example.com,https://www.example.com"
...
```

以上配置表示，只有来自 https://example.com 和 https://www.example.com 的请求会被允许。

上传限制

全局配置

当客户端请求 Body 大小超过限制时，将向客户端返回一个状态码 413 的 Request Entity Too Large 错误。 全局设置可以通过参数 client_max_body_size 进行配置，默认为 1m。

```
proxy-body-size: 1m
```

特定资源限制

要为特定 Ingress 资源实现这一限制，可以通过添加注解 nginx.ingress.kubernetes.io/proxy-body-size 的方式实现。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: 8m
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: host-a.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
```

```
name: service-a
port:
  name: http
```

介绍

ingress-nginx 是 Kubernetes 官方社区托管的 Ingress 控制器，使用 Nginx 作为反向代理和负载均衡，经过大量的生产实践，稳定可靠。

- 安装
- IngressClass
- IngressClass Scope
- 跨域配置
- 负载均衡
- 超时配置
- 日志配置
- 上传限制
- 镜像流量

超时配置

本页介绍 Ingress Nginx 的全局超时配置和特定的 Ingress 资源可用的超时配置。

全局超时配置

可通过以下配置项进行 Ingress Nginx 的全局超时配置：

配置项	描述	默认值
proxy-connect-timeout	设置与代理服务器建立连接的超时。	默认 5s，但通常不能超过 75s
proxy-read-timeout	设置从代理服务器读取响应的超时。该超时仅在两个连续的读取操作之间设置，而不是为整个响应的传输设置。	60s
proxy-send-timeout	设置向代理服务器传输请求的超时。该超时只在两个连续的写操作之间设置，而不是为整个请求的传输设置。	60s
proxy-stream-next-upstream-timeout	限制允许将连接传递到下一个服务器的时间。	默认为 600s，设置为 0 值则关闭此限制。
proxy-stream-timeout	设置客户端或代理服务器连接上两个连续的读或写操作之间的超时。如果在这个时间内没有传输数据，连接就会关闭	600s

配置项	描述	默认值
upstream-keepalive-timeout	设置一个超时时间，在这个时间内，与上游服务器的空闲保持连接将保持开放。	60
worker-shutdown-timeout	设置优雅停机的超时时间。	240s
proxy-protocol-header-timeout	设置接收代理协议头文件的超时值。默认的 5 秒可以防止 TLS 直通处理程序无限期地等待一个中断的连接。	5s
ssl-session-timeout	设置 SSL 会话缓存中的会话参数的有效时间。会话过期时间是相对于创建时间而言的。每个会话缓存会占用大约 0.25MB 空间。	10m
client-body-timeout	定义读取客户端请求正文的超时。	60s
client-header-timeout	定义读取客户端请求头的超时。	60s

特定的资源自定义超时配置

以下是特定的资源自定义超时配置，以及相关的参数配置，例如重试次数等：

配置项	描述
nginx.ingress.kubernetes.io/proxy-connect-timeout	设置代理连接超时时间。
nginx.ingress.kubernetes.io/proxy-send-timeout	设置代理发送超时时间。
nginx.ingress.kubernetes.io/proxy-read-	设置代理读取超时时间。

配置项	描述
timeout	
nginx.ingress.kubernetes.io/proxy-next-upstream	配置重试策略或者重试条件，可以使用多个组合用空格分隔，例如设置为 http_500 http_502，支持下列策略：
nginx.ingress.kubernetes.io/proxy-next-upstream-tries	如果满足重试条件时，可用的重试次数。
nginx.ingress.kubernetes.io/proxy-request-buffering	是否启用请求缓冲功能。设置为 on 启用请求缓冲，off 则禁用请求缓冲。如果启用了请求缓冲功能，则会在接收到完整的请求数据之后才将其转发到后端工作负载，否则可能会在接收到部分请求数据时就开始转发请求。HTTP/1.1 Chunked 编码的请求不受此参数限制，始终会进行缓冲。如果禁用了，发送过程中出现发送错误，就不会选择下一个工作负载重试。

负载均衡与流量代理配置指南

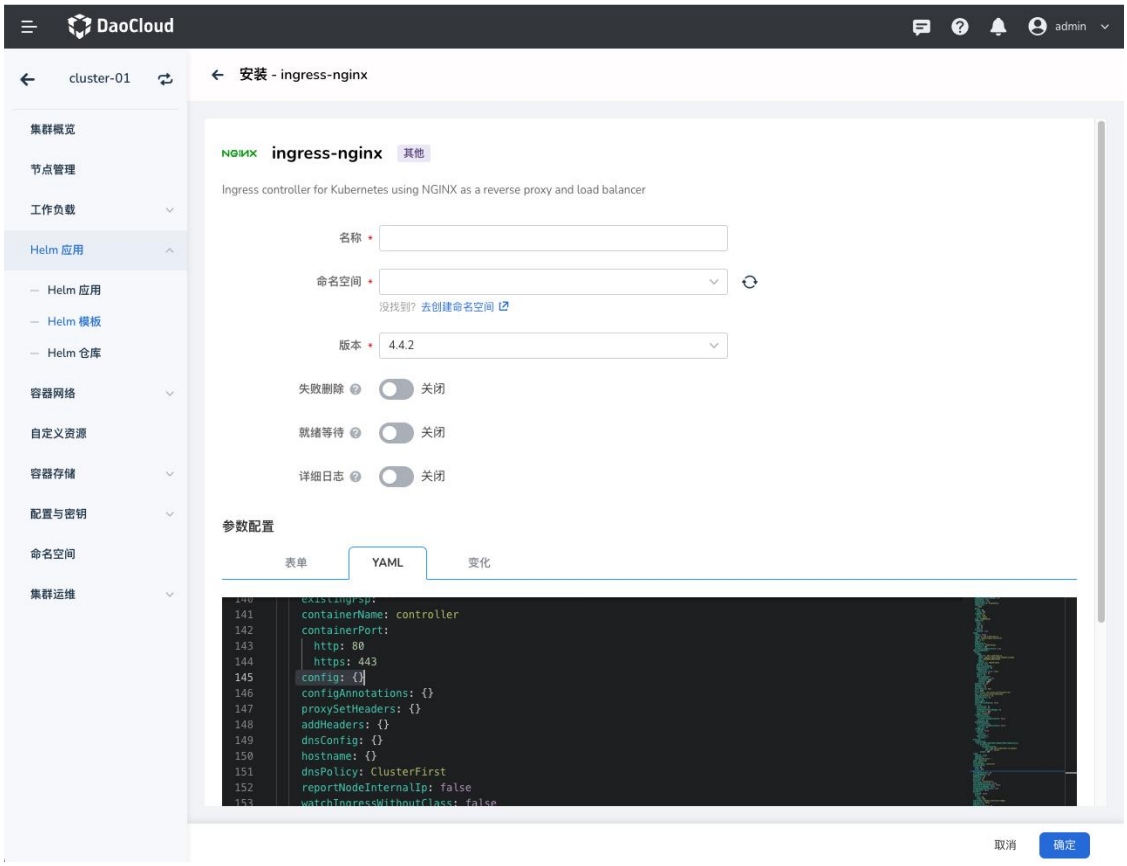
本页介绍负载均衡和流量代理的配置方法，包括全局配置、基于域名、路径、Header、Cookie、请求哈希等不同负载均衡策略。

负载均衡全局配置

Ingress Nginx 全局配置可以指定默认负载均衡算法，支持 round_robin 和 ewma 两种，默认为 round_robin。 主要区别在于如何选择后端工作负载：round_robin 算法会按照事先定义的顺序循环选择，将请求平均地分配到每个后端工作负载。如果后端工作负载的性能差异较大，可能会导致负载不均衡。ewma 算法可以将请求发送到加权平均负载最低的工作负载上，加权负载指数会随着请求的到来而逐渐变化，使得负载均衡更加均衡。

可以参考安装章节，在 Helm 安装配置 .Values.yaml 指定如下内容：

```yaml
ingress-nginx:
  controller:
    config:
      load-balance: ewma
```



set-load-balancer

基于域名的流量负载

域名是一个字符串，用于标识互联网上的网站或资源，如 www.example.com。它是人类可读的， 而 IP 地址（如 10.6.0.1）是机器可读的。域名通过 DNS 服务器映射到对应的 IP 地址，使人们可以通过浏览器访问特定网站。

Ingress Nginx 支持转发不同域名的流量。通过将域名系统映射到 Ingress Nginx 的 VIP，实现域名到 IP 的映射。

通过使用以下配置定义，可以将不同域名的流量转发到相应的后端 Service。例如，将域名 A 的流量转发到 Service A，域名 B 的流量转发到 Service B。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: host-a.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-a
            port:
              name: http
  - host: host-b.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-b
            port:
              name: http
```

基于请求路径的负载

URL 请求路径（URL Request Path）是在 URL 中指定的从域名或 IP 地址到达特定页面或资源的路径。 它是以 / 开头的字符串。 例如：https://www.example.com/page-hello-world 其中 /page-hello-world 是 URL 请求路径。

Ingress Nginx 支持使用将不同的 URL 请求路径指向不同的 Service。

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /path-a
        pathType: Prefix
        backend:
          service:
            name: service-a
            port:
              name: http
      - path: /path-b
        pathType: Prefix
        backend:
          service:
            name: service-b
            port:
              name: http
```

基于 Header 请求头的负载

以下是使用 nginx.ingress.kubernetes.io/canary-by-header 注解的示例。 可以在请求中添加版本头以将流量路由到应用程序的稳定版本或金丝雀版本。

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
```

```yaml
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-header: "version"
    nginx.ingress.kubernetes.io/canary-by-header-value: "v2"
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-service-v2
                port:
                  name: http
```

在这个示例中，对 example.com 的流量将根据 Header 请求头 version 的值进行分割。 对于 Header 请求头值为 v2 的流量，请求将会路由到 example-service-v2。

基于 Cookie 会话的负载

基于 Cookie 的负载均衡策略，通过使用 Cookie 将客户端与后端服务绑定在一起，这种方法可以确保每个客户端的请求始终由相同的后端服务处理，从而使得服务变得更加稳定。

Cookie 负载均衡的工作原理如下：

1. 当客户端发送第一次请求时，Ingress Nginx 将使用负载均衡算法选择一个后端服务来处理请求。

2. Ingress Nginx 将一个名为 example-cookie-name 的 Cookie 发送回客户端。这个 Cookie 包含了后端服务的标识信息。
3. 客户端再次发送请求时，会带上这个 Cookie。Ingress Nginx 根据 Cookie 中的标识信息，将请求转发到对应的后端服务。

以下是配置示例：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/session-cookie-name: "example-cookie-name"
    nginx.ingress.kubernetes.io/session-cookie-expires: "172800"
    nginx.ingress.kubernetes.io/session-cookie-max-age: "172800"
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-service
                port:
                  name: http
```

基于请求哈希的负载均衡

使用 nginx.ingress.kubernetes.io/upstream-hash-by 注解，可以指定客户端于服务器映射的哈希值。 例如可以使用 $binary_remote_addr 或 $http_x_forwarded_for 完成基于客户端 IP 的映射。

有时候集群中 Ingress Nginx 在多个负载均衡后面代理流量，这时候根据 $binary_remote_addr 获取的不是真实的客户端地址，而是其前置负载均衡器的 IP 地址。这时候可以使用 $http_x_forwarded_for 获取真实 IP。

当然你也可以设置为 $request_uri，让其基于请求路径负载均衡。

上面的几种方式是映射到单个上游服务器，你可以添加注解 nginx.ingress.kubernetes.io/upstream-hash-by-subset: "true" 开启分组功能，这时候会对上游工作负载进行分组，流量到达分组后再随机分配给组中的工作负载，你可以使用 nginx.ingress.kubernetes.io/upstream-hash-by-subset-size 指定每个分组工作负载的数量。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginxhello
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginxhello
  template:
    metadata:
      labels:
        app: nginxhello
    spec:
```

```yaml
      containers:
        - name: nginxhello
          image: registry.k8s.io/e2e-test-images/echoserver:2.3
          ports:
            - containerPort: 8080
          env:
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
---
kind: Service
apiVersion: v1
metadata:
  name: nginxhello
  labels:
    app: nginxhello
spec:
  selector:
    app: nginxhello
```

```yaml
  ports:
    - name: http
      port: 80
      targetPort: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/upstream-hash-by: "$binary_remote_addr"
  name: nginxhello-ingress
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: foo.bar.com
    http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: nginxhello
              port:
                number: 80
```

使用说明

本页介绍如何使用 Submariner 实现跨集群通讯。

!!! note

当集群的 CNI 为 Calico 时，由于 Calico 插入的 IPTables 规则优先级高于 Submariner 插入的 IPTables 规则，导致跨集群通讯出现问题。
所以我们必须手动配置 Calico 的 IPPool 来规避这个问题。

比如：

Cluster	Pod CIDR	Service CIDR
A	10.244.64.0/18	10.244.0.0/18
B	10.245.64.0/18	10.245.0.0/18

在 Cluster A：

```
cat > clusterb-cidr.yaml <<EOF
  apiVersion: projectcalico.org/v3
  kind: IPPool
  metadata:
    name: clusterB-pod-cidr
  spec:
    cidr: 10.245.64.0/18
    natOutgoing: false
    disabled: true
EOF


cat > podwestcluster.yaml <<EOF
    apiVersion: projectcalico.org/v3
    kind: IPPool
```

```
    metadata:
      name: clusterB-service-cidr
    spec:
      cidr: 10.245.0.0/18
      natOutgoing: false
      disabled: true
EOF
```

在 Cluster B 执行同样操作：

```
cat > clusterb-cidr.yaml <<EOF
  apiVersion: projectcalico.org/v3
  kind: IPPool
  metadata:
    name: clusterB-pod-cidr
  spec:
    cidr: 10.244.64.0/18
    natOutgoing: false
    disabled: true
EOF


cat > podwestcluster.yaml <<EOF
  apiVersion: projectcalico.org/v3
  kind: IPPool
  metadata:
    name: clusterB-service-cidr
  spec:
    cidr: 10.244.0.0/18
    natOutgoing: false
    disabled: true
EOF
```

应设置 natOutgoing 为 false。

使用

使用 Subctl 工具导出 Service，验证并排查问题。

1. 下载 Subctl

```
root@controller:~# curl -Ls https://get.submariner.io | bash
root@controller:~# export PATH=$PATH:~/.local/bin
root@controller:~# echo export PATH=\$PATH:~/.local/bin >> ~/.profile
root@controller:~# subctl version
version: v0.13.4
```

2. 验证 Submariner 是否就绪：

```
root@controller:~# subctl show all
Cluster "cluster.local"
√ Detecting broker(s)
√ No brokers found

√ Showing Connections
GATEWAY                CLUSTER    REMOTE IP      NAT   CABLE DRIVER
SUBNETS            STATUS       RTT avg.
controller-node-1   cluster2   10.6.212.10   no    libreswan       244.1.10.0/24
   connected   985.269µs

√ Showing Endpoints
CLUSTER ID                    ENDPOINT IP       PUBLIC IP          CAB
LE DRIVER          TYPE
cluster1                      10.6.212.101      47.243.194.240   libreswan
           local
cluster2                      10.6.212.10       47.243.194.240   libreswan
            remote

√ Showing Gateways
```

NODE HA STATUS SUMMARY

controller active All connections (1) are established

√ Showing Network details

Discovered network details via Submariner:

Network plugin: calico

Service CIDRs: [10.233.0.0/18]

Cluster CIDRs: [10.233.64.0/18]

Global CIDR: 244.0.10.0/24

√ Showing versions

COMPONENT REPOSITORY VERSION

submariner quay.m.daocloud.io/submariner 0.14.0

submariner-operator quay.m.daocloud.io/submariner 0.14.0

service-discovery quay.m.daocloud.io/submariner 0.14.0

3. 如果出现跨集群通讯失败，可以使用以下命令排查：

root@controller:~# subctl diagnose all

Cluster "cluster.local"

√ Checking Submariner support for the Kubernetes version

√ Kubernetes version "v1.25.0" is supported

√ Checking Submariner support for the CNI network plugin

√ The detected CNI network plugin ("calico") is supported

√ Trying to detect the Calico ConfigMap

√ Calico CNI detected, checking if the Submariner IPPool pre-requisites

are configured

√ Checking gateway connections
√ All connections are established

√ Globalnet deployment detected – checking if globalnet CIDRs overlap
√ Clusters do not have overlapping globalnet CIDRs
Pod "submariner-gateway-x2x48" has restarted 14 times
Pod "submariner-globalnet-fv7x9" has restarted 9 times

Starting with Kubernetes 1.23, the Pod Security admission controller expects namespaces to have security labels. Without these, you will see warnings in subctl's output. subctl should work fine, but you can avoid the warnings and ensure correct behavior by adding at least one of these labels to the namespace "submariner-operator":
pod-security.kubernetes.io/warn=privileged
pod-security.kubernetes.io/enforce=privileged
pod-security.kubernetes.io/audit=privileged

√ Checking Submariner support for the kube-proxy mode
√ The kube-proxy mode is supported

Starting with Kubernetes 1.23, the Pod Security admission controller expects namespaces to have security labels. Without these, you will see warnings in subctl's output. subctl should work fine, but you can avoid the warnings and ensure correct behavior by adding at least one of these labels to the namespace "submariner-operator":
pod-security.kubernetes.io/enforce=privileged
pod-security.kubernetes.io/audit=privileged
pod-security.kubernetes.io/warn=privileged

⋱ Checking the firewall configuration to determine if the metrics port (8080) is allowed Starting with Kubernetes 1.23, the Pod Security admission controller expects namespaces to have security labels. Without these, you will see warnings in subctl's output. subctl should work fine, but you can avoid the warnings and ensure correct behavior by adding at least one of these labels to the namespace "submariner-operator":

pod-security.kubernetes.io/audit=privileged

pod-security.kubernetes.io/warn=privileged

pod-security.kubernetes.io/enforce=privileged

√ Checking the firewall configuration to determine if the metrics port (8080) is allowed

√ The firewall configuration allows metrics to be retrieved from Gateway nodes

Starting with Kubernetes 1.23, the Pod Security admission controller expects namespaces to have security labels. Without these, you will see warnings in subctl's output. subctl should work fine, but you can avoid the warnings and ensure correct behavior by adding at least one of these labels to the namespace "submariner-operator":

pod-security.kubernetes.io/enforce=privileged

pod-security.kubernetes.io/audit=privileged

pod-security.kubernetes.io/warn=privileged

⋱ Checking the firewall configuration to determine if intra-cluster VXLAN traffic is allowed Starting with Kubernetes 1.23, the Pod Security admission controller expects namespaces to have security labels. Without these, you will see warnings in subctl's output. subctl should work fine, but you can avoid the warnings and ensure correct behavior by adding at least one of these labels to the namespace "submariner-operator":

pod-security.kubernetes.io/warn=privileged

pod-security.kubernetes.io/enforce=privileged

pod-security.kubernetes.io/audit=privileged

√ Checking Globalnet configuration
√ Globalnet is properly configured and functioning

Skipping inter-cluster firewall check as it requires two kubeconfigs. Please run "subctl diagnose firewall inter-cluster" command manually.

如果上面方式不能解决，通过以下方式收集当前环境所有信息：

root@controller:~# subctl gather

- 跨集群的服务发现：

  如果你想使用跨集群的服务发现，你需要手动导出 Service 到其他集群，可以参考下面的方式：

  在 ClusterA 导出这个服务：

```
root@controller:~# kubectl get svc
NAME                                                TYPE                CLUSTE
R-IP        EXTERNAL-IP     PORT(S)          AGE
kubernetes                                          ClusterIP        10.233.0.1
    <none>            443/TCP          81d
        ClusterIP        10.233.35.64     <none>            80/TCP            5
8d
test                                                ClusterIP        10.233.21.14
3    <none>            80/TCP           79d
root@controller:~# subctl export service test
```

  上面命令将会导出 default 命令空间下一个名为 test 的 Service，导出完成之后查看状态：

```
root@controller:~# kubectl get serviceexports.multicluster.x-k8s.io
NAME    AGE
```

```
test     2d2h
root@controller:~# kubectl get serviceexports.multicluster.x-k8s.io test -o yaml
apiVersion: multicluster.x-k8s.io/v1alpha1
kind: ServiceExport
metadata:
  creationTimestamp: "2023-02-24T03:58:00Z"
  generation: 1
  name: test
  namespace: default
  resourceVersion: "20068131"
  uid: 05fa2aab-e6b1-491b-92bb-4f2561c92a1b
status:
  conditions:
  - lastTransitionTime: "2023-02-24T03:55:12Z"
    message: ""
    reason: ""
    status: "True"
    type: Valid
  - lastTransitionTime: "2023-02-24T03:55:12Z"
    message: ServiceImport was successfully synced to the broker
    reason: ""
    status: "True"
    type: Synced
```

显示导出成功，即可在 ClusterB 访问这个 Service：

```
root@controller-node-1:~# kubectl  exec -it dao2048-5745d9b5d7-bsjjl sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ # nslookup test.default.svc.clusterset.local
Server:         10.233.0.3
```

```
Address:        10.233.0.3:53

Name:      test.default.svc.clusterset.local
Address:  10.233.21.143
```

安装 Submariner

本页介绍如何安装 Submariner。

前置条件

- 至少准备两个集群，可将 submariner-k8s-broker 部署在其中一个集群中，也可以单独部署在其他集群，submariner-operator 部署在每个 join 的子集群中。子集群和 submariner-k8s-broke 部署的集群之间应能够正常通讯
- 不同子集群的子网最好不要重叠，否则需要启用 Globalnet 功能，而 Globalnet 功能有一定的限制
- 支持的 CNI 列表为:
  - OpenShift-SDN
  - Weave
  - Flannel
  - Canal
  - Calico(存在兼容性问题，需要手动规避，参考使用说明)
  - OVN(要求 OVN NorthBound DB 的版本 > 6.1.0+)
- Kube-proxy 的模式必须为 IPtables，IPvs 模式目前尚不支持
- 集群内部需要允许 Vxlan 的流量，集群外部需要放通 udp/4500 端口 > Submariner 现在处于比较早期的阶段，出现 Bug 比较常见。

安装步骤

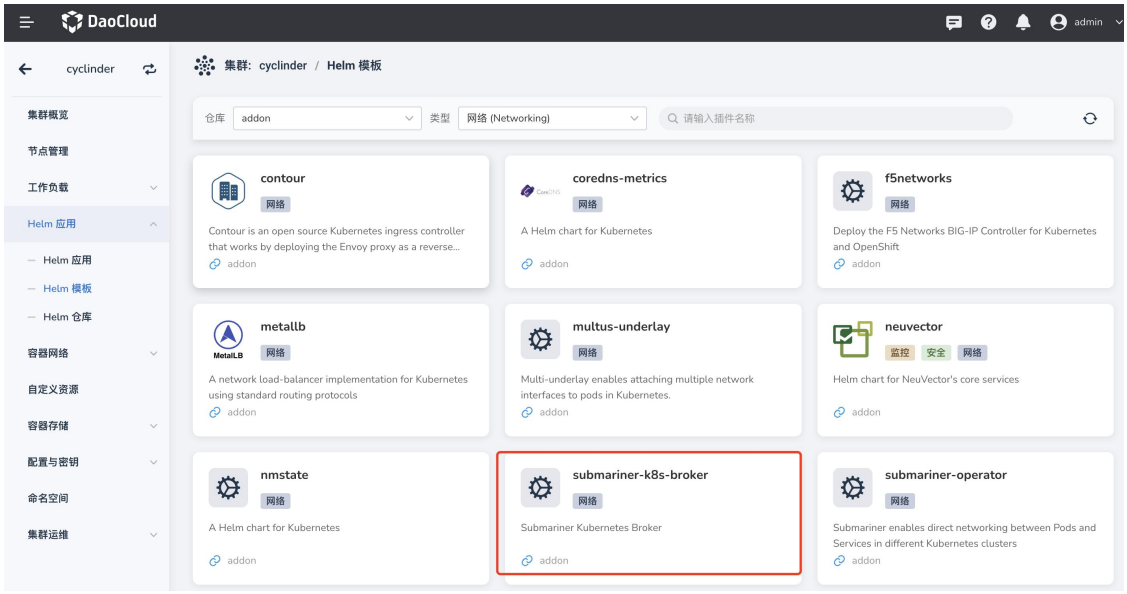
!!! note

`submariner-k8s-broker` 可以部署在一个单独的集群，也可以部署在 Join 的子集群中。本文以 `submariner-k8s-broker` 部署在一个单独的集群为例，展示安装过程。

请确认您的集群已成功接入容器管理平台，然后执行以下步骤首先安装 submariner-k8s-broker:

1. 在左侧导航栏点击 容器管理 —> 集群列表，然后找到准备安装 submariner-k8s-broker 的集群名称。

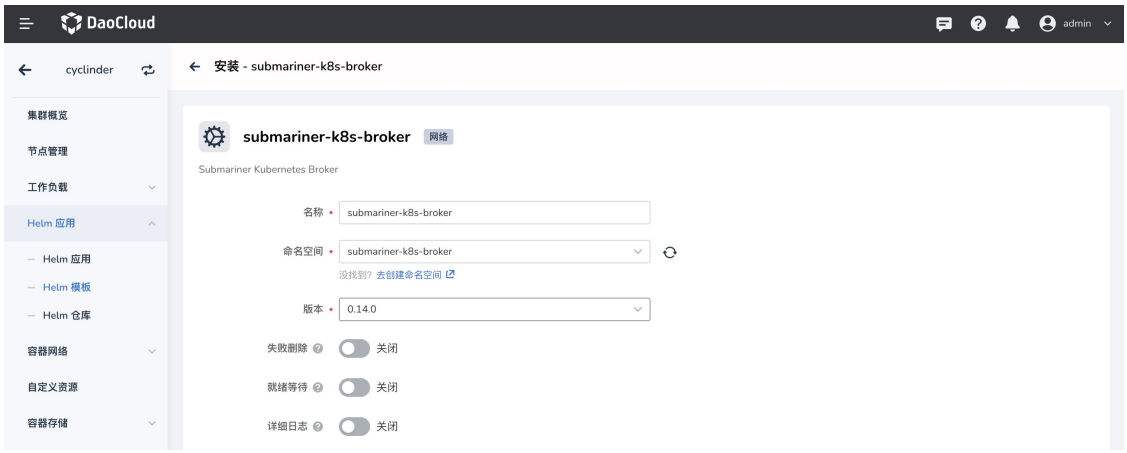
2. 在左侧导航栏中选择 Helm 应用 -> Helm 模板，找到并点击 submariner-k8s-broker。



helm

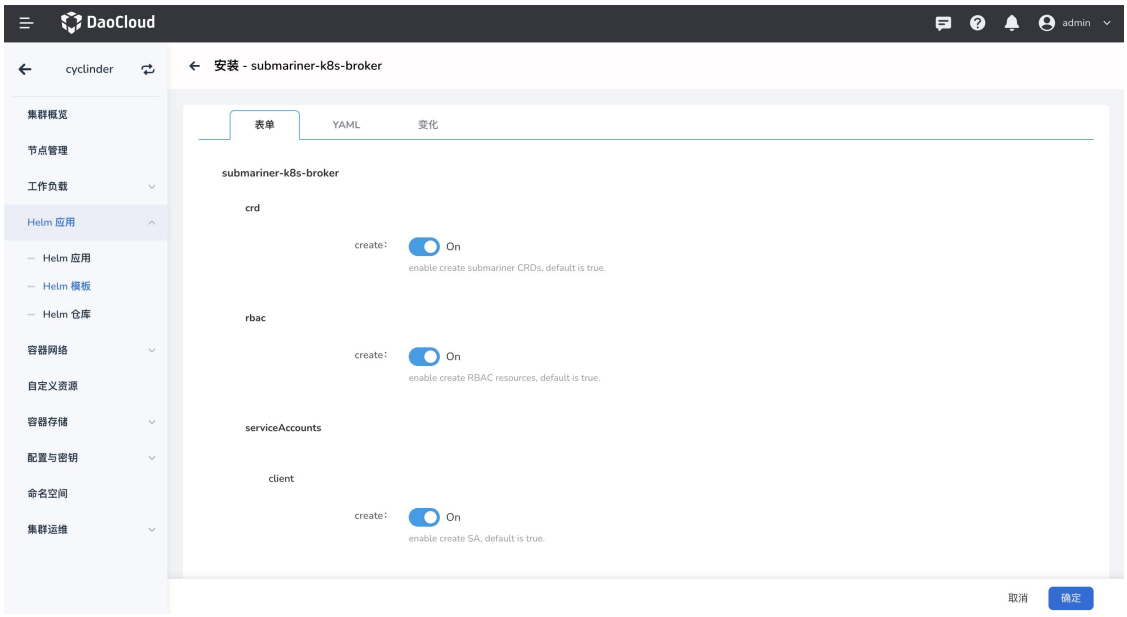
3. 在版本选择中选择希望安装的版本，点击安装。

4. 推荐安装到 submariner-k8s-broker 命名空间下：



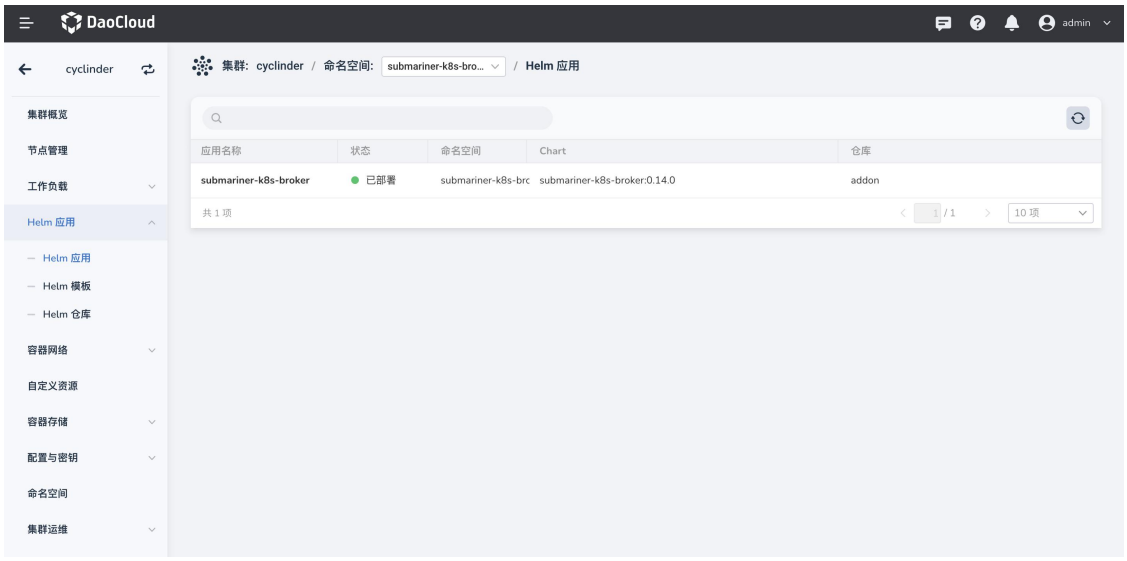
broker-ns

5. 下图所示配置无需更改，保持默认参数即可：



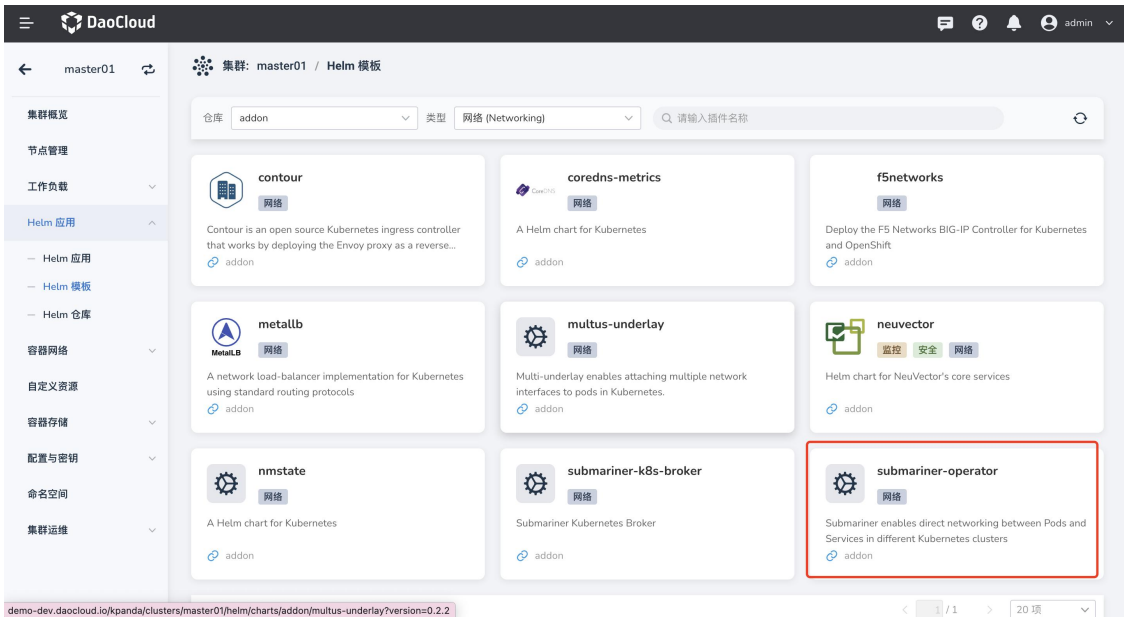
config

6. 在 Broker 集群成功安装 submariner-k8s-broker：



broker

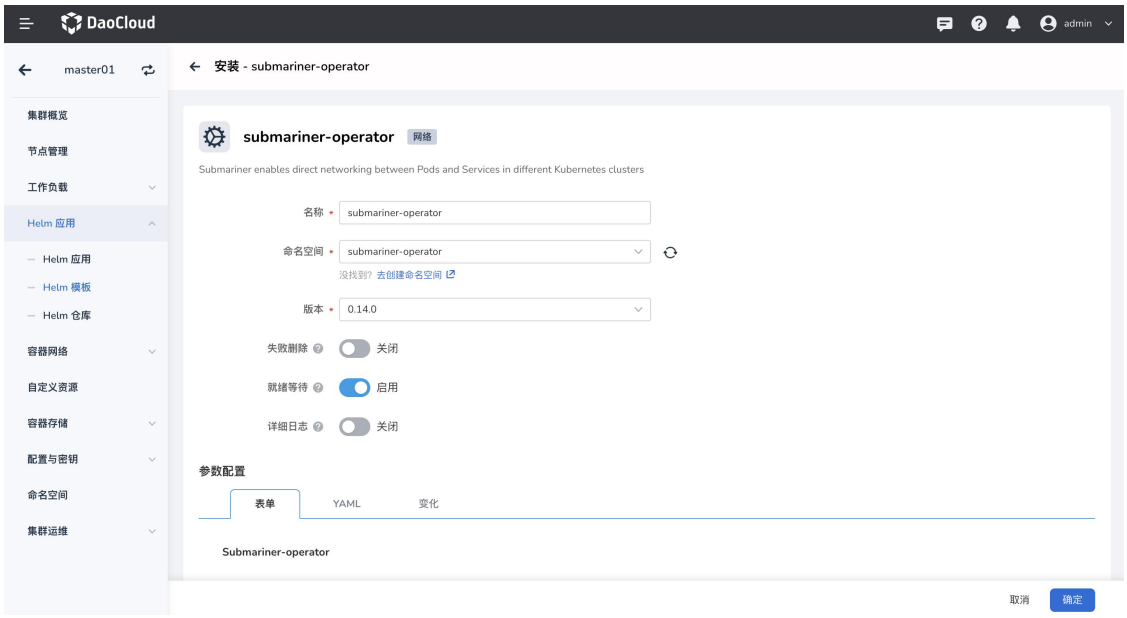
7. 切换到其中子集群：master01，安装 submariner-operator：



operator

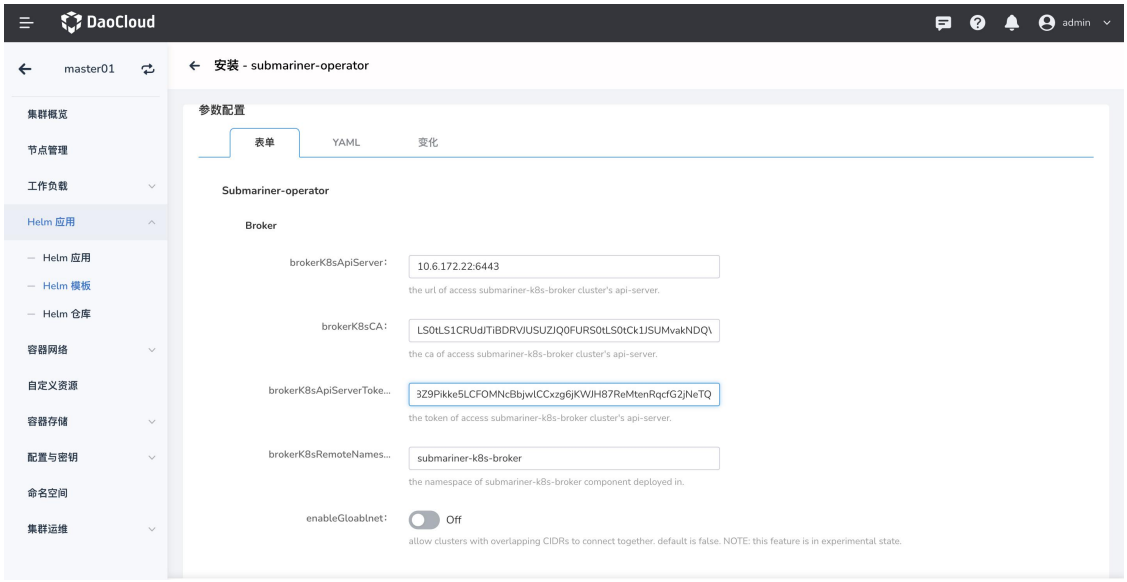
8. 在版本选择中选择希望安装的版本，点击安装。

9. 推荐安装到 submariner-operator 命名空间下，开启就绪等待：



operator-ns

10. 配置 submariner-operator 连接 Broker 集群的配置:



operator-broker

上面的参数说明:

- Broker —> brokerK8sApiServer：Broker 集群 API-Server 的地址，可以通过以下方式获取：

```
# 在安装 submariner-k8s-broker 的集群上
[root@broker ~]# kubectl -n default get endpoints kubernetes -o jsonpath="{.subsets[0].addresses[0].ip}:{.subsets[0].ports[?(@.name=='https')].port}"
10.6.172.22:6443
```

- Broker —> brokerK8sCA：访问 Broker 集群 API-Server 的客户端证书，可以通过以下方式获取：

```
# 在安装 submariner-k8s-broker 的集群上
[root@broker ~]# kubectl -n submariner-k8s-broker get secrets -o jsonpath="{.items[?(@.metadata.annotations['kubernetes\.io/service-account\.name']=='submariner-k8s-broker-client')].data['ca\.crt']}"
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJek1ESXlNakV3TkRjek5Wb1hEVE16TURJeE9URXdORGN6TlZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTVFLCmRVUHZoeDNOcFRXeGU0STZNV1l1VFU0VThJeVhBeWF6ZXJRRnhZK3l5Q3F1NnhFOUUwTDRSbUdxS0pyTFRSVWYKQi85YmowcWRaNFRJd2FTTmN3cGVqanYxYllTaVVKV0NsT0dXbTdLWUd4UnhCMEZFbVlCaGovT29HWGJhaEw5eQpqR1A1VjFsYTMxYlhONllSY0duR3hXMCt0QmE0RndhMGlUV3owd2R5dFE4cVpuMXdLUC9QMjNKZ0o1N1pwN3hOCmlLOHV6OENGRjZmTnkyNVJYeEV2cGRQdzEwcnJDM0Npem1mcjNaZHl0cGVFdk44U2V6STByMVNtVHBxSTM4UlEKZzVKdTFvV1NRcENEZjBPV1N5WUxxRno1NmU5dStPc0tyVXFzQkVsUkdKNWhvUGppVW9VN3I2N3lmSGl4TmdTcgpEM1l4aklSQzZlTE5OTHE5UG8wQ0F3RUFBYU5aTUZjd0RnWURWUjBQQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZDclJxaGJRNnV1aWFlOVJJd3RsdGd0bHhaRzhNQlVHQTFVZEVRUU8KTUF5Q0NtdDFZbVZ5Ym1WMFpYTXdEUVlKS29aSWh2Y05BUUVMQlFBRGdnRUJBSU5QUmtZMEpKSk5jNjFlVUpSOQo2bm9LTFl0cUN6VUxqRDVOZ3BmSy9GK0NvNk45Y0k5c0c1V1FVQ3Z
```

MUVc5MUlFRlhFencvSEswUFBJTy8zVFplCjVQS05TUjR5d2hwSmtyeF
RveDEvWWRqK2RWNzNGUnFKTFVuQkw1T3U4SWlKR2RNR0VKc040
dnd5amk5TGt3WGMKcC9aUUVieWhLdmRvRWdOaDlhUzMrQzRJSVF
nNmVqZTgvYVh6NTVaL2RoN2NZQUhCZWFQYmVwWU1kNklwc3VhT
wo5Smx3TmJaSWczNHlzL3hQYjhCNUJOSkp2MXI3eE1YblFlaXc1MFN
WaGdpR0JXMFdOdFo4NDVGREY2NHFzVFBsCjhLSjZOQkdjRUVuTHp
DSXNhVUwxUHk4bk90K3NWWFNlUlhXV2VlWm84R0krWGpXQjZWL3
FscnllSmp6Yy9hUU4KYWdBPQotLS0tLUVORCBDRVJUSUZJQ0FURS
0tLS0tCg==

一 Broker —> brokerK8sApiServerToken：访问 Broker 集群 API-Server 的 Token，可以通过以下方式获取：

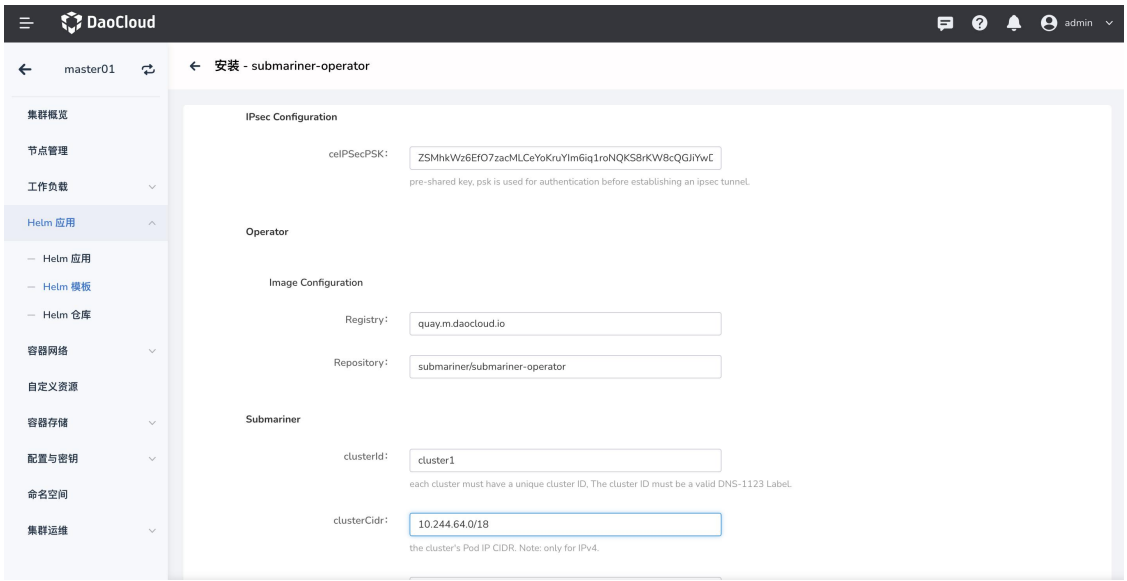
```
# 在安装 submariner-k8s-broker 的集群上
[root@broker ~]# kubectl -n submariner-k8s-broker get secrets -o
 jsonpath="{.items[?(@.metadata.annotations['kubernetes\.io/service-
account\.name']=='submariner-k8s-broker-client')].data.token}" | ba
se64 --decode
eyJhbGciOiJSUzI1NiIsImtpZCI6Ik1kUWpXaIlwUVV0RmtTcjJXdElvMW1W
WHdZbU5Md2pRN0tFeVZSbUpINTgifQ.eyJpc3MiOiJrdWJlcm5ldGVzL
3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3
VudC9uYW1lc3BhY2UiOiJzdWJtYXJpbmVyLWs4cy1icm9rZXIiLCJrdWJ
lcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlY3JldC5uYW1lIjoic3Vib
WFyaW5lci1rOHMtYnJva2VyLWNsaWVudC10b2tlbiIsImt1YmVybmV0ZX
MuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUi
OiJzdWJtYXJpbmVyLWs4cy1icm9rZXItY2xpZW50Iiwia3ViZXJuZXRlcy
5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQudWlkIjoiYz
M0MzQ0MTItOTEyNi00NmFjLTk5ODUtZDgwYWI3MDAxOTdkIiwic3ViIj
oic3lzdGVtOnNlcnZpY2VhY2NvdW50OnN1Ym1hcmluZXItazhzLWJyb2tl
cjpzdWJtYXJpbmVyLWs4cy1icm9rZXItY2xpZW50In0.quC7a3hTctbgaR
KHmzAvlP16EspwtTzWzirgj0o2d9XYfVe6bPX29Wg4XHh3ZzetaMYmvj
_toukQJcQ6bO1CG7xv4mOFFkLF2ECrQNPGKYa5A2LHgQCFiWteWjU
7wcisW3lugMTC6a5OrbK4optvVjseLwqI-BClm8nsgjGocBTrv1qqDp-4
```

aiPkLtgcZRV2bsgE3yLsyc7Mhuncs7gDml2NuBXoYRGfXtAjd6XlbnXd5Tp5sAV_k92xruNqKDpzQZl32K6l7lt1WNQvtGOTYHa9sks0gxgUmQu8Z9Pikke5LCFOMNcBbjwlCCxzg6jKWJH87ReMtenRqcfG2jNeTQ

注意需要执行 decode

- Broker —> brokerK8sRemoteNamespace： submariner-k8s-broker 组件的命名空间，默认为 submariner-k8s-broker(参考步骤 4)。
- Broker —> enableGloablnet：是否启用 Globalnet 功能。如果子集群间子网重叠，可以启用。

11. 配置 submariner-operator：



operator-install1

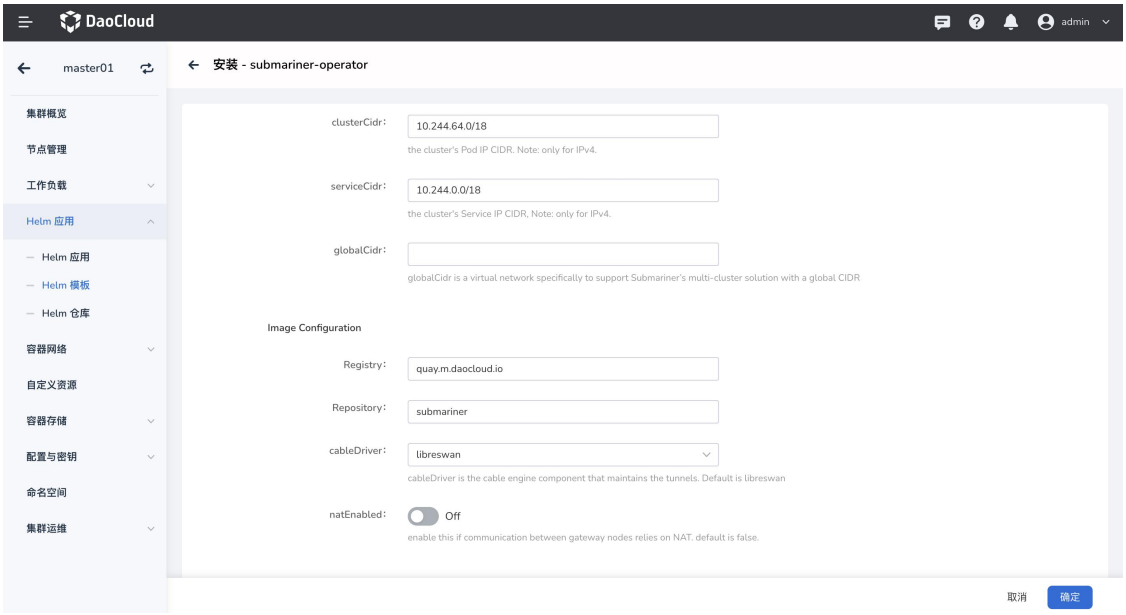
以上配置说明:

- IPsec Configuration —> ceIPSecPSK：建立 IPsec 隧道时所需要的预共享密钥。如果对安全性有较高要求，可以使用下面的方式获取：

  [root@broker ~]# LC_CTYPE=C tr -dc 'a-zA-Z0-9' < /dev/urandom | fold -w 64 | head -n 1

ZSMhkWz6EfO7zacMLCeYoKruYlm6iq1roNQKS8rKW8cQGJiYwDaYQNrzbbdP2azl

- Submariner —> clusterId：用于标识该子集群，填写规范需要满足 DNS-1123 Label。
- Submariner —> clusterCidr：子集群 Pod 的 CIDR。

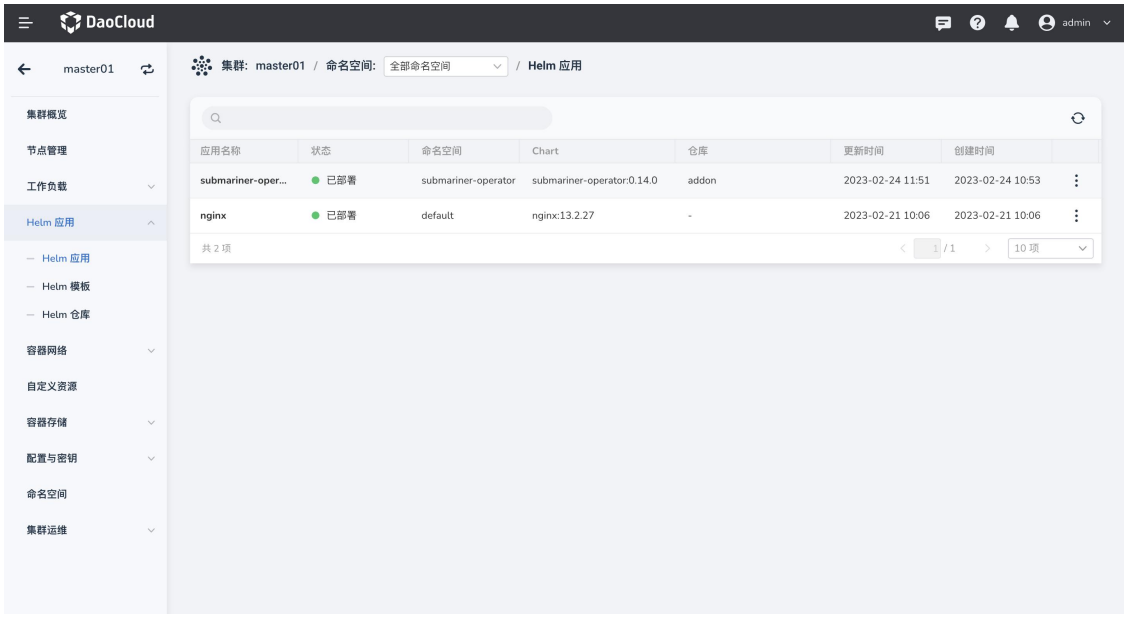


operator-install2

- Submariner —> serviceCidr：子集群 Service 的 CIDR。
- Submariner —> globalCidr：开启 globalnet 功能。

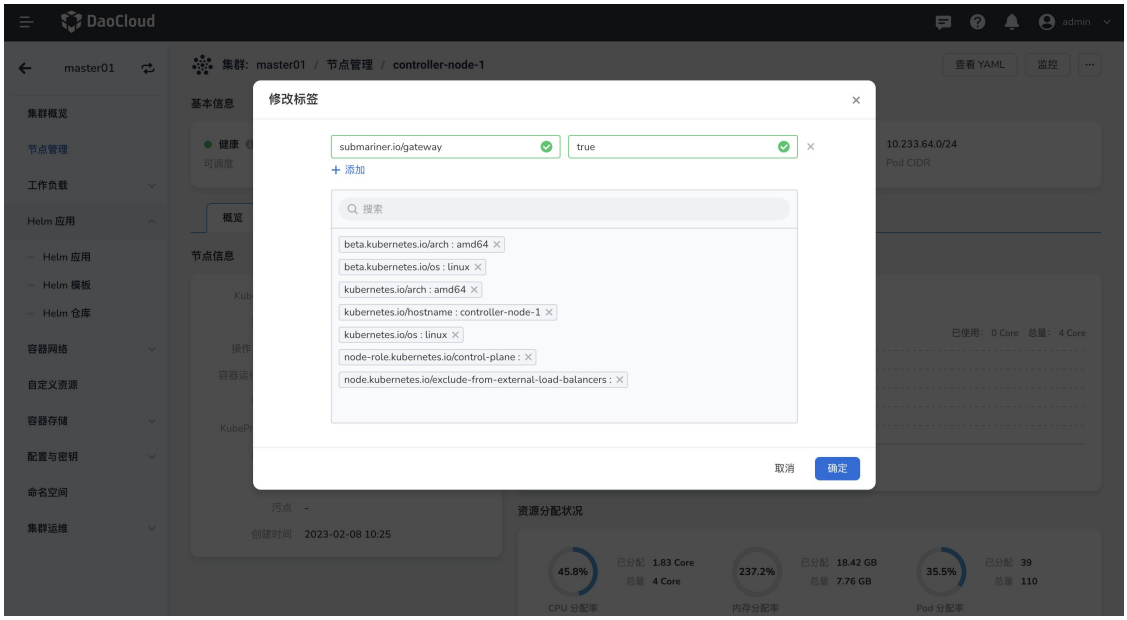
  注意：如果不使用 globalnet 功能，请不要配置该字段。
- Image Configuration —> cableDriver：隧道模式，支持 libreswan(默认)、wireguard、vxlan。
- Image Configuration —> natEnabled：是否打开网关节点之间的 NAT 功能。如果不同子集群的网关节点需要跨公网才能通讯，则开启。

12. 在子集群 master01 安装完成：



opearator-install-3

13. 安装完成后，需要手动设置子集群 master01 其中某个节点为网关节点，需要添加标签“submariner.io/gateway: true”：



gateway

14. 查看组件是否正常 running：

```
root@controller-node-1:~# kubectl  get po -n submariner-operator
NAME                                                READY   ST
ATUS      RESTARTS        AGE
helm-operation-upgrade-submariner-operator-q79sp-9bkzl   0/1     Co
mpleted   0               31h
submariner-gateway-jpclp                            1/1     Runnin
g         18 (30m ago)    31h
submariner-lighthouse-agent-589676874b-dn75z        1/1     Run
ning      0               31h
submariner-lighthouse-coredns-5d675b4897-tm752      1/1     Run
ning      0               32h
submariner-lighthouse-coredns-5d675b4897-wxr8z      1/1     Run
ning      0               32h
submariner-metrics-proxy-2ptmn                      2/2     Runn
ing       0               31h
submariner-operator-75ccdf484-6862g                 1/1     Run
ning      0               32h
submariner-routeagent-kh4hp                         1/1     Runni
ng        0               31h
```

15. 子集群 master01 join 完成，按照同样方式在另一个子集群安装 submariner。

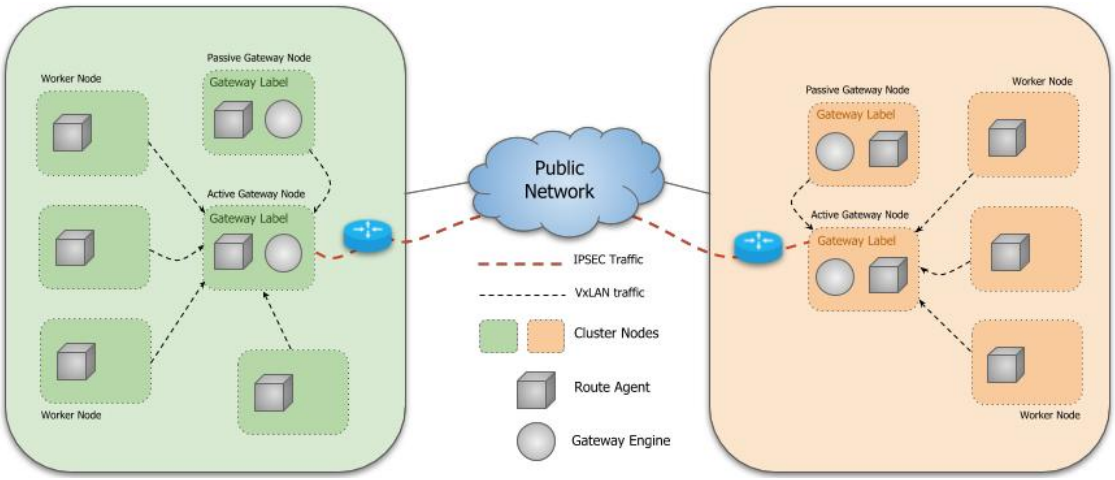
注意事项

- 安装完之后需要至少给一个节点打上标签：“submariner.io/gateway: true”。只有该标签存在, Gateway 组件才会被安装。
- 如果集群 CNI 为 Calico，需要做一些额外的操作，来解决与 Calico 的兼容性问题，请参考使用说明。

什么是 Submariner

随着 Kubernetes 不断的发展，多集群也逐渐变得流行起来。多集群提高应用的冗余、规模和故障隔离等能力，但多集群连通是一个比较大的问题。 Submariner 是一款开源的多集群网络解决方案，它以一种安全的方式实现了跨集群间 Pod 与 Service 连通性，并且通过 Lighthouse 组件实现 KMCS 以提供跨集群的服务发现能力。

架构图:



submariner

它包括下面几个 **重要组件** ：

- Broker：没有实际的 Pod 和 Service，只提供子集群访问 Broker 集群 API-Server 的凭证。基于此，使子集群之间可以交换 Metadata 信息，用于发现对方。
- Gateway Engine：建立和维护集群间的隧道，打通跨集群的网络通讯。
- Route Agent：在 Gateway 节点和工作节点之间建立 Vxlan 隧道，使工作节点上的跨集群流量先转发到 Gateway 节点，再通过跨集群的隧道从 Gateway 节点发送至对端。

- Service Discover：包括 Lighthouse-agent 和 Lighthouse-dns-server 组件，实现 KMCS API，提供跨集群的服务发现。

可选组件：

- Globalnet Controller：支持重叠子网的集群间互连。

重要 CRD 列表：

```
[root@master1]# kubectl get crd | grep -iE 'submariner|.multicluster'
brokers.submariner.io                                 2023-02-22T13:56:30Z
clusterglobalegressips.submariner.io                  2023-02-22T13:56:37Z
clusters.submariner.io                                2023-02-22T13:56:37Z
endpoints.submariner.io                               2023-02-22T13:56:37Z
gateways.submariner.io                                2023-02-22T13:56:37Z
globalegressips.submariner.io                         2023-02-22T13:56:37Z
globalingressips.submariner.io                        2023-02-22T13:56:37Z
servicediscoveries.submariner.io                      2023-02-22T13:56:30Z
serviceexports.multicluster.x-k8s.io                  2023-02-22T11:32:29Z
serviceimports.multicluster.x-k8s.io                  2023-02-22T11:32:29Z
submariners.submariner.io                             2023-02-22T13:56:30Z
```

- submariners.submariner.io：用于 submariner-operator 组件创建所有 Submariner 组件
- clusters.submariner.io：存储每个子集群的信息，包括其 Pod、Service 的子网信息
- endpoints.submariner.io：每个子集群网关节点的基本信息，包括私有/公有 IP/隧道模式/状态等
- serviceexports.multicluster.x-k8s.io：导出每一个 Service，对应一个 serviceexports 对象，用于服务发现
- serviceimports.multicluster.x-k8s.io：对于每一个 serviceexports 对象，Lighthouse-agent 创建对应的 serviceimports 对象，供其他集群消费

- clusterglobalegressips.submariner.io：全局的 CIDR，用于启用 Globalnet 时解决子集群子网重叠的问题

下一步：

- 安装 Submariner
- 使用和 debug

微服务引擎 Release Notes

本页列出微服务引擎的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-08-25

V0.27.0

新功能

- **新增** 各模块功能的权限依赖关系
- **新增** 托管资源异常信息详情 API 字段
- **新增** 获取用户权限点 API 接口
- **新增** 网关域名创建时的重复检查 API
- **新增** 流量泳道中服务添加失败原因 API 信息
- **新增** 托管 Nacos 灰度发布相关接口
- **新增** 网关支持单机器多 Envoy 实例解决网格为网关注入边车失败的问题
- **新增** 获取端口限流规则列表接口
- **新增** 网关 API 匹配支持正则表达式和完整路径匹配模式
- **新增** Sentinel 控制台认证相关接口
- **新增** 网关 API 导出导入接口
- **新增** 网关概览中的相关统计数据接口

修复

- **修复** 网关服务接入时权限与预定义权限不符的问题
- **修复** 服务治理状态表达不正确的问题
- **修复** 托管 Nacos 配置灰度发布返回异常问题
- **修复** 托管 Nacos 中服务实例列表报错问题
- **修复** 流量泳道服务版本重复的问题
- **修复** 流量泳道中服务列表异常问题

- **修复** 流量泳道服务删除的问题
- **修复** 当错误原因未空时泳道列表空指针异常的问题
- **修复** Skoala-init Chart 表单模式配置不生效的问题

优化

- **优化** 依要求将删除泳道服务从 delete 方法变为 put。
- **优化** 移除多余权限依赖并完善内部权限依赖关系
- **优化** 流量泳道适配界面展示需求
- **优化** 云原生微服务插件接口
- **优化** 添加泳道服务的方式由批量改为单个
- **优化** 前端组件 Deployment 端口由 80 修改为 8080
- **优化** 升级 Ghippo SDK 到 v0.20.0-dev2

!!! note

重要：微服务引擎版本大于 `v0.24.2` 版时，针对 `v0.24.2` 及之前版本存在不兼容更新，因为网关涉及开源组件仓库地址变更，所以更新前需要手动删除旧有的 `gateway-api-admission-xxxxx` Job，然后进行正常升级更新操作。

2023-08-03

v0.26.2

修复

- **修复** Consul 注册中心接入时检测不通的问题
- **修复** 网关由于插件应用异常重启问题
- **修复** 网关验证插件配置异常问题
- **修复** Sentinel 集群流控规则编辑异常的问题

!!! note

重要：微服务引擎版本大于 `v0.24.2` 版时，针对 `v0.24.2` 及之前版本存在不兼容更新，因为网关涉及开源组件仓库地址变更，所以更新前需要手动删除旧有的 `gateway-api-admission-xxxxx` Job，然后进行正常升级更新操作。

2023-07-26

v0.26.1

修复

- **修复** 修复 Agent 组件版本错误问题

2023-07-25

v0.26.0

新功能

- **新增** Sentinel 限流规则详情接口
- **新增** 云原生微服务治理相关 API
- **新增** 云原生微服务流量泳道相关 API
- **新增** 适配可观测模块 0.19 版本
- **新增** 总体概览添加审计日志展示 API
- **新增** 网关概览页面添加资源展示 API
- **新增** 添加网关级别全局验证插件能力相关 API

修复

- **修复** Nacos 端口修改不生效的问题
- **修复** Sentinel 集群流控规则保存不生效的问题
- **修复** 删除 Nacos 中非空服务时接口异常的问题
- **修复** 网关监控数据中重复数据的问题
- **修复** 云原生微服务使用插件相关 API 的问题
- **修复** 网关域名命名规则的相关问题
- **修复** 云原生微服务流量泳道的版本错误问题

优化

- **优化** 人大金仓数据库的驱动名由 kb_v8r6 改为 kingbase

2023-07-19

v0.25.0

新功能

- **新增** 将网关运行时的网格注入标识修改被标签方式标记
- **新增** 与中间件关联操作的权限级联选择能力
- **新增** 云原生微服务限流规则详情 API
- **新增** 云原生微服务治理相关 API
- **新增** 为安装托管资源时集群就绪检查添加对应组件版本信息
- **新增** 为各类搜索 API 添加模糊搜索能力

修复

- **修复** 流水线外部镜像扫描地址不对的问题
- **修复** 与中间件关联权限依赖未涵盖目录级别的问题

优化

- **优化** 网关及相关资源变更的连锁校验
- **优化** 网关监听端口由集群 IP 改为容器端口

2023-07-06

v0.24.2

修复

- **修复** 页面体验优化及问题修复

2023-07-05

v0.24.1

修复

- **修复** Skoala-init Chart 表单中版本不是最新的问题
- **修复** 页面体验优化及问题

2023-06-30

v0.24.0

新功能

- **新增** 云原生微服务插件相关 API
- **新增** 与中间件关联操作的权限级联选择能力

修复

- **修复** Insight 集成数据异常的问题
- **修复** 网关状态筛选结果异常的问题

优化

- **优化** 网关及相关资源变更的连锁校验

2023-06-26

v0.23.0

新功能

- **新增** 网关 API 的批量上下线接口
- **新增** 为引擎组件添加增强健康检查的配置
- **新增** 为缓存逻辑添加标签缓存支持
- **新增** 网关创建和更新时网格边车强制注入选项
- **新增** 网关列表按照状态筛选支持
- **新增** 审计日志集成
- **新增** 在持续集成中添加许可检查
- **新增** 将 Chart 中的敏感信息通过 Secret 存储
- **新增** 升级托管 Nacos 版本至 2.2.3 版
- **新增** Sentinel 版本至 0.10.5 版

修复

- **修复** Insight 集成数据指标于原始数据对齐
- **修复** 与 mSpider 模块对接时未继承权限的问题
- **修复** Nacos 实例列表治理开启状态缺失的问题

- **修复** Sentinel 集群流控名称的问题
- **修复** 整体概览查询数据不是按照工作空间统计的问题
- **修复** 网关未开启 HTTPS 但域名可以开启 HTTPS 的问题
- **修复** Skoala-init Chart 表单数据默认值问题

优化

- **优化** Sentinel Grafana 的使用体验和问题
- **优化** 托管资源告警逻辑只展示相关资源告警条目
- **优化** 重构全局管理模块配置结构
- **优化** 托管 Nacos 的信息读取通过 Clusterpedia 完成
- **优化** Insight 对接至 0.17.3 版
- **优化** 数据库中立相关配置的灵活性

2023-05-31

v0.22.2

修复

- **修复** Sentinel 集群流控 API 的问题
- **修复** Sentinel 规则模糊查询 API 的问题

优化

- **优化** 连接数据库的默认值提高容错性

2023-05-29

v0.22.1

修复

- **修复** 插件 CRD 位置不对的问题
- **修复** OpenAPI 发布流程问题

优化

- **优化** 为 Hive 组件的数据库配置设置默认值

2023-05-26

v0.22.0

新功能

- **新增** 托管 Nacos 2.2.x 版本的支持
- **新增** 网关链路功能支持
- **新增** 云原生微服务路径重写 API
- **新增** 云原生微服务超时 API
- **新增** 云原生微服务故障注入 API
- **新增** 云原生微服务重试等 API
- **新增** 云原生微服务使用 WASM 插件功能
- **新增** Skoala-init Chart 添加 JSON Schema
- **新增** OpenAPI 文档发布流程
- **新增** Hive 支持数据库中立

修复

- **修复** 托管 Nacos 相关 API 有几率出现空指针的问题
- **修复** 托管 Nacos 端口展示不正确的问题
- **修复** Grafana 内的 HTTPProxy 拼写错误
- **修复** Nacos 数据库初始化 SQL 脚本问题
- **修复** 网关组件自定义权限对接问题
- **修复** 调用 Ghippo 空指针问题
- **修复** Nacos 服务元数据接口异常问题
- **修复** 更新域名的错误
- **修复** 调用 Kpanda API 频繁的问题
- **修复** 接入 Insight 获取数据不准确的问题
- **修复** WASM 插件查询出现错误
- **修复** 更新 API 路由服务后 API 出现错误
- **修复** Sentinel Token 服务器资源问题

- **修复** Zookeeper 链接未关闭问题

优化

- **优化** Sentinel 监控面板结构和数据逻辑
- **优化** 网关域名管理中移除 virtualhost 自定义资源
- **优化** Sentinel 集群流控的 API 逻辑
- **优化** Ghippo 升级至 0.17.0-rc2
- **优化** 数据库初始化组件 sweet 已于 0.22.0 版废弃，将在 0.23.0 及之后版本彻底移除，从 0.22.0 版开始数据表将自动同步更新，无需人工干预

2023-05-07

v0.21.2

修复

- **修复** Sentinel 监控面板问题
- **修复** 网关被注入网格边车的问题
- **修复** 传统微服务开启网格治理的注册中心地址格式问题
- **修复** 托管注册中心选择中间件实例未按照集群筛选的问题
- **修复** 注册中心统计不正确的问题

优化

- **优化** 更新网关组件到社区最新测试版

2023-04-26

v0.21.1

修复

- **修复** 云原生微服务分页问题

优化

- **优化** 为 Nacos 添加禁用注入 Istio 边车的配置
- **优化** 升级 Insight 版本至 0.16.0 正式版
- **优化** 组件连接数据库重试机制

- **优化** 配合 Istio 治理能力修改 Nacos 的 9848 端口名为 GRPC

2023-04-25

v0.21.0

新功能

- **新增** 网关接入内部及外部地址分离展示
- **新增** 云原生微服务治理能力相关 API
- **新增** 告警消息列表 API
- **新增** 网关使用插件相关 API
- **新增** 网关各类插件的逻辑 API

修复

- **修复** 网关更新时 Envoy 配置不更新的问题
- **修复** 网关只能添加单个端口的问题
- **修复** Insight 集成 JVM 查询的问题
- **修复** 云原生微服务治理 API 的问题
- **修复** Sentinel 规则无法存取的问题
- **修复** 链接不到数据库时某些 API 调用会造成程序崩溃的问题
- **修复** 资源状态 API 的问题
- **修复** 云原生微服务治理 API 时间单位问题
- **修复** 域名相关格式校验问题
- **修复** 插件一些字段命名错误的问题

优化

- **优化** Insight 集成到 0.16.0
- **优化** 部署模板终端服务名称添加模块名前缀
- **优化** virtualhost crd
- **优化** httpproxy crd
- **优化** skoalaplugin crd

2023-04-21

v0.20.0

新功能

- **新增** Sentinel 门户版本
- **新增** 网关域名级别黑白名单支持
- **新增** 原生服务治理列表 API
- **新增** 原生服务治理编辑 API
- **新增** 可观测 JVM 监控集成
- **新增** 网关资源工作负载状态展示
- **新增** 网关负载策略选择配置

修复

- **修复** Contour 镜像版本
- **修复** 自定义角色功能点及 API 映射
- **修复** 网关概览 API 排序和条目

优化

- **优化** 插件中心相关 API
- **优化** 配置文件结构
- **优化** 配置参数由直接读取改为配置包实现
- **优化** 管理组件整体包结构
- **优化** 管理组件
- **优化** Contour 升级到 v1.24.3-ipfilter-tracing
- **优化** Envoy 升级到 v1.25.4

2023-04-10

v0.19.4

修复

- **修复** 托管 Nacos 的启动问题

2023-04-10

v0.19.3

修复

- **修复** 前端问题

2023-04-04

v0.19.2

修复

- **修复** Nacos 及 Sentinel 默认验证账号问题
- **修复** 概览网关 API 排序问题

2023-04-04

v0.19.1

修复

- **修复** CVE-2022-31045 漏洞
- **修复** 插件中心 API 问题
- **修复** 网关重启的问题
- **修复** 插件更新时版本不能成功更新的问题
- **修复** Nacos 及 Sentinel 默认验证账号问题
- **修复** 概览内微服务网关网关 API 排序问题
- **修复** Nacos 支持版本回滚至 2.0.4

2023-03-24

v0.19.0

新功能

- **新增** 自定义权限点及 API 的实现
- **新增** 注册中心概览相关 API
- **新增** 网关黑白名单相关 API
- **新增** 概览中网关健康度相关 API

- **新增** Nacos 支持版本至 2.1.2
- **新增** 获取 Nacos 及网关版本信息 API
- **新增** 概览中注册配置中心统计收集器
- **新增** 概览中注册配置中心统计 API
- **新增** 云原生微服务服务列表 API
- **新增** 云原生微服务服务导入相关 API
- **新增** 插件中心自定义资源设计
- **新增** 插件中心插件管理相关 API
- **新增** 网关前置流量拦截配置 API
- **新增** 级联资源操作添加事务（类似）机制处理包
- **新增** 资源重启功能

修复

- **修复** Nacos Operator 中初始化 Nacos 的数据库脚本问题
- **修复** Sentinel 相关数据概览 API 的问题
- **修复** 网关相关数据的概览 API 的问题
- **修复** 网关生命周期管理减少网关异常重启问题
- **修复** 概览 API 路径大小写问题
- **修复** Nacos 2.1.2 无法创建集群的问题
- **修复** 网关前置流量拦截修改不生效的问题
- **修复** 网关黑白名单 API 的问题
- **修复** Nacos GRPC 端口名字对集成 Istio 产生的问题
- **修复** 每日构建中的外部镜像安全扫描

优化

- **优化** CI 流程并简化不必要的任务
- **优化** 全部资源的更新操作都采用 retry 机制
- **优化** 网关相关功能重构

2023-02-25

v0.18.0

新功能

- **新增** 添加注册中心配置中心分离 API
- **新增** 添加概览相关逻辑及 API

修复

- **修复** gateway-api 镜像版本问题
- **修复** 负载均衡模式网关的 IP 池加载问题
- **修复** 健康检查相关问题

2023-02-22

v0.17.1

新功能

- **新增** 网关 NodePort 支持
- **新增** 网关 LoadBalancer 支持
- **新增** Sentinel 规则统计 API
- **新增** Sentinel 治理的服务列表 API
- **新增** 网关 API 的 Cookie 重写策略
- **新增** 概览数据定时任务
- **新增** 定时收集异常 Sentinel 任务
- **新增** Sentinel 集群流控详情 API
- **新增** 网关接入服务列表端口选择
- **新增** 网关服务健康检查策略
- **新增** 网关 API 中对健康策略的支持
- **新增** Sentinel 统计相关 API
- **新增** 支持 chart 离线化的 CI 流程
- **新增** 在每日构建中增加外部镜像安全扫描能力
- **新增** 发布自动更新 chart 中镜像版本

修复

- **修复** Nacos Namespace 创建异常的问题
- **修复** Nacos 持久化存储修改异常的问题
- **修复** Nacos 生命周期管理资源校验问题
- **修复** 网关监控面板数据展示问题
- **修复** Ghippo 链接 GRPC 地址缺失问题
- **修复** Sentinel 获取集群流控 API 的问题
- **修复** 托管 Nacos 资源状态不更新的问题
- **修复** Sentinel 适配 Nacos public 字符串问题
- **修复** Sentinel 获取资源 API 没有聚合不同实例问题
- **修复** Sentinel 系统规则不生效问题
- **修复** 网关服务注册中心类型分页错误的问题
- **修复** 创建服务端口错误的问题
- **修复** 数据库初始化组建的问题
- **修复** 使用 Helm 命令替代 Argocd 部署 Alpha 环境
- **修复** 基础镜像 CVE 问题并升级至 3.17.2
- **修复** 发布过程 Chart 更新问题

优化

- **优化** 升级 gateway-api 到 v0.6.0
- **优化** 待更新资源获取由 clusterpedia 改为 client-go
- **优化** Sentinel 应用监控模板
- **优化** 将离线 chart 构建 CI 步骤独立
- **优化** Contour 升级到 v1.24.1
- **优化** envoy 升级到 v1.25.1
- **优化** 通过 Chart 能力使 Skoala Init 安装时固定命名空间

2022-12-30

v0.16.1

修复

- **修复** 构建镜像时重复创建 builder 的问题

优化

- **优化** Sentinel 应用监控面板细节

2022-12-29

v0.16.0

修复

- **修复** Sentinel 调用具备认证开启 Nacos 接口的问题
- **修复** nacos-operator 频繁修改服务资源的问题

优化

- **优化** 添加 Sentinel 服务的 Grafana 监控面板
- **优化** 升级 Insight 为最新版本支持通过集群名查询监控数据

2022-12-28

v0.15.2

新功能

- **新增** 网关 API 对于认证服务器的支持
- **新增** 托管注册中心服务接入 API
- **新增** Sentinel 集群流控相关 API

修复

- **修复** Sentinel 规则拼接错误的问题
- **修复** Sentinel 仪表盘名称的问题
- **修复** 管理组件 Chart 对于生产换环境的 Service IP 问题
- **修复** Nacos 控制器处理逻辑的问题
- **修复** 与集群管理集成的 egress 地址问题

优化

- **优化** 托管 Nacos 监控仪表盘问题
- **优化** nacos-operator 数据库初始化的文件获取地址
- **优化** 更新 Sentinel 镜像至 v0.6.0

2022-12-22

v0.14.0

新功能

- **新增** Init Chart 所需镜像的离线支持
- **新增** 获取托管 Nacos 的令牌

修复

- **修复** Skoala Chart 中 Values 命名问题
- **修复** CI 流程中的镜像问题

优化

- **优化** 设置默认日志输出到控制台
- **优化** 升级 nacos-operator 到社区版本
- **优化** 更新 Nacos 自定义资源的认证开启支持
- **优化** 设置默认组件日志级别

2022-12-21

v0.13.0

新功能

- **新增** 对接中间件 MySQL 和 Redis 的相关 API
- **新增** 网关 JWT 验证支持的 API
- **新增** 网关域名校验逻辑
- **新增** Sentinel 资源列表 API
- **新增** 网关查询注册中心服务的接口
- **新增** 版本发布后推送 Init Chart 至 addon 仓库
- **新增** 版本发布时完成 gitlab release 操作

- **新增** 动态更改日志级别

修复

- **修复** 全局限流规则更新时不生效的问题
- **修复** Envoy Log Level 未设置问题
- **修复** 更新网关时异常未判断的问题
- **修复** 托管 Nacos 的数据库初始化问题

优化

- **优化** 注册中心列表按更新时间降序排列
- **优化** 统一网关 JWT 相关的字段名称
- **优化** 网关域名列表增加是否开启 JWT 的字段
- **优化** Sentinel 服务名连接符的逻辑
- **优化** 升级 Contour 到 1.23 版
- **优化** 升级 Envoy 到 1.24 版
- **优化** 升级 k8s.io/相关组件到 0.25 版
- **优化** 将 go-replayers 组件回归社区版本
- **优化** 将 go-helm-client 组件回归社区版本
- **优化** 升级 Contour 到 1.23.1 版
- **优化** 修改 Agent 组件为强制不注入网格边车
- **优化** 将 Nacos 镜像默认配置回归社区版本
- **优化** 移除 Nacos 镜像相关 CI 流程

2022-12-13

v0.12.2

新功能

- **新增** 添加 Sentinel 自身监控的 Grafana 模板支持
- **新增** 添加自定义配置网关索引的配置信息

修复

- **修复** 微服务集成可观测组件的状态问题
- **修复** 注册中心开启网格插件能力的治理状态问题
- **修复** 网关日志索引问题
- **修复** 前置依赖检查接口的问题
- **修复** Sentinel 与 Nacos 默认命名空间匹配的逻辑问题
- **修复** 连接容器管理模块的端口异常情况的逻辑

功能总览

「DaoCloud 道客」微服务引擎是面向业界主流微服务生态的一站式微服务管理平台，主要提供微服务治理中心和微服务网关两个维度的功能，具体包括服务注册发现、配置管理、流量治理、服务级别的链路追踪、API 管理、域名管理、监控告警等，覆盖了微服务生命周期中的各种管理场景。微服务引擎具有很强的兼容性，不仅可以无缝对接 DCE 5.0 的其他组件，也可以完美兼容 Spring Cloud、Dubbo 等开源生态，帮助您更便捷地使用开源微服务技术构建自己的微服务体系。

微服务注册与发现

统一纳管传统微服务和云原生微服务，实现从传统微服务生态向云原生微服务生态的平稳过渡，助力企业走向云原生化。

- 支持创建 Nacos 托管中心，进而管理微服务命名空间、治理微服务流量、管理微服务配置、链路追踪与监控等。
- 支持接入 Eureka、Zookeeper、Nacos 三类传统的微服务注册中心。
- 支持接入 Kubernetes 和 Service Mesh 两类云原生微服务注册中心。

微服务流量治理

在流量治理层面，采用线上流量治理方案，可以快速与主流开源微服务框架集成，用 Sentinel 和 Mesh 解决不同生产情况下的痛点。

- 支持通过 Sentinel 使用流控、熔断降级、热点、系统、授权、集群流控等规则治理传统微服务的东西向流量。
- 支持通过 Service Mesh 通过虚拟服务、目标规则、网关规则在网格中治理微服务流量。

微服务配置中心

Nacos 托管注册中心可作为微服务的配置管理器，可以从不同项目中抽取通用配置事先统一管理，也可以为同一项目应用多个不同配置，实现差异化管理。

- 基于微服务命名空间和分组（Group）对配置文件进行隔离。
- 结合 @RefreshScope 注解动态更新配置项。
- 管理配置文件的历史版本，支持版本差异对比并一键回滚到特定版本。
- 支持查询当前配置的监听者以及 MD5 校验值。
- 提供示例代码，便于新手快速使用客户端编程消费该配置，降低新手使用门槛。

微服务网关

微服务网关肩负管理微服务南北流量管控的重要作用，提供 API 管理、接口限流、多种策略安全认证、黑白名单、路由转发、MockAPI 等能力，同时提供企业级高性能和高扩展的云服务能力。

- 多网关管理：原生支持对容器管理模块中的多集群、多命名空间的网关实例进行管理，支持网关实例的全生命周期管理。
- API 策略管理：通过图形化界面进行 API 的增删改查，配置 API 策略，例如负载均衡、路径改写、超时配置、重试机制、请求头/响应头重写、WebSocket、本地限流、健康检查等，**同时保障原生 API 的能力不受影响**。
- 插件管理：提供丰富的插件功能，支持安全、流量管控、缓存等插件，支持一键开启/停用插件。
- 监控告警：微服务网关在部署时会自动配置监控、告警等功能。每个网关都自带完善的资源监控和网关业务监控。

适用场景

「DaoCloud 道客」微服务引擎是面向业界主流微服务生态的一站式微服务管理平台，主要提供微服务治理中心和微服务网关两个维度的功能，具体包括服务注册发现、配置管理、流量治理、服务级别的链路追踪、API 管理、域名管理、监控告警等，覆盖了微服务生命周期中的各种管理场景。

基于所提供的各项功能，微服务引擎可用于微服务的注册与发现、配置管理、微服务流量治理，以及微服务网关管理等场景。典型的应用场景如下：

一站式管理大量异构微服务

随着应用服务的爆发式增长，微服务的数量越来越多，使用的架构也可能各不相同，传统的微服务和云原生微服务并存。各个微服务之间互相调用，互相依赖，牵一发而动全身，容易出现级联效应，造成系统雪崩。随着微服务系统的运维难度和成本越来越高，用户需要一个能够同时管理传统微服务和云原生微服务、监控服务信息、追踪服务级别的链路调用情况、统一管理微服务配置、提供微服务网关的**一站式**产品。微服务引擎完全覆盖了这些场景需求，能够很好地满足用户需要。

从传统到云原生的转变和过渡

云原生在弹性扩展、屏蔽底层差异、故障处理等方面具有显著优势，一些企业在云原生浪潮的影响下，希望采取稳态模式逐渐从传统的微服务架构转向云原生微服务架构，有些企业希望采取敏态模式实现快速转变。无论采取稳态还是敏态模式，微服务引擎都是很好的选择，因为其支持传统微服务和云原生微服务的统一纳管，支持 Sentinel 和 Service Mesh 两种流量治理模式，分别适用于传统微服务和云原生微服务的东西向流量治理。

微服务网关的可视化与高效能

很多开源的网关产品仅支持命令行操作，使用门槛高、难度大。某些情况下，多个微服务共用一个网关，网关资源开销大，而且可能逐渐成为整个系统的资源瓶颈。

使用「DaoCloud 道客」微服务引擎在 Contour 的基础上增加界面操作能力，大大降低了使用门槛和维护成本，可以实现多网关管理，在不同集群和不同命名空间下轻松创建多个网关实例。这些网关实例之间互相隔离，因而具有更高的可用性和稳定性。此外，还可以使用网关路由的级联功能，实现微服务的蓝绿部署动态切换。

产品优势

「DaoCloud 道客」提供的微服务引擎包含微服务治理中心和微服务网关两个模块，提供微服务的注册发现、流量治理、配置管理、网关 API 等一组简单实用且高性能的微服务治理能力，帮助从传统微服务架构稳定升级到云原生微服务架构。

相比其他同类产品，「DaoCloud 道客」提供的微服务引擎具有以下优势：

- 一站式治理

  提供服务注册发现、配置管理、流量治理、链路追踪、指标监控、网关管理、API 管理、域名管理、监控告警、网关策略等功能，满足微服务生命周期中各个阶段的管理需求，实现一站式治理。

- 无感迁移

  完全兼容 Nacos 开源注册中心和 Envoy、contour 开源网关，支持在零代码改造的前提下完成从自建服务到「DaoCloud 道客」微服务引擎的迁移。传统微服务无需作任何改动，即可通过注册中心接入到微服务引擎，进而实现流量治理、配置管理、链路追踪、指标监控等功能。

- 平稳过渡

  统一管理传统微服务和云原生微服务，支持接入传统注册中心（Zookeeper、Eureka、Nacos）和云原生注册中心（Kubernetes、Service Mesh） 类型的注册中心，实现从传统微服务生态向云原生微服务生态的平稳过渡，助力企业走向云原生化。

- 开放兼容

  微服务引擎支持传统和云原生的注册中心，也支持 Spring Cloud、Dubbo 等主流的开源微服务框架，以及 Envoy、Sentinel、Contour 等开源网关

组件。此外，还可以和 DCE 5.0 的多云编排、数据服务中间件、服务网格、应用工作台等模块自由组合，实现更多定制化、精细化的功能。

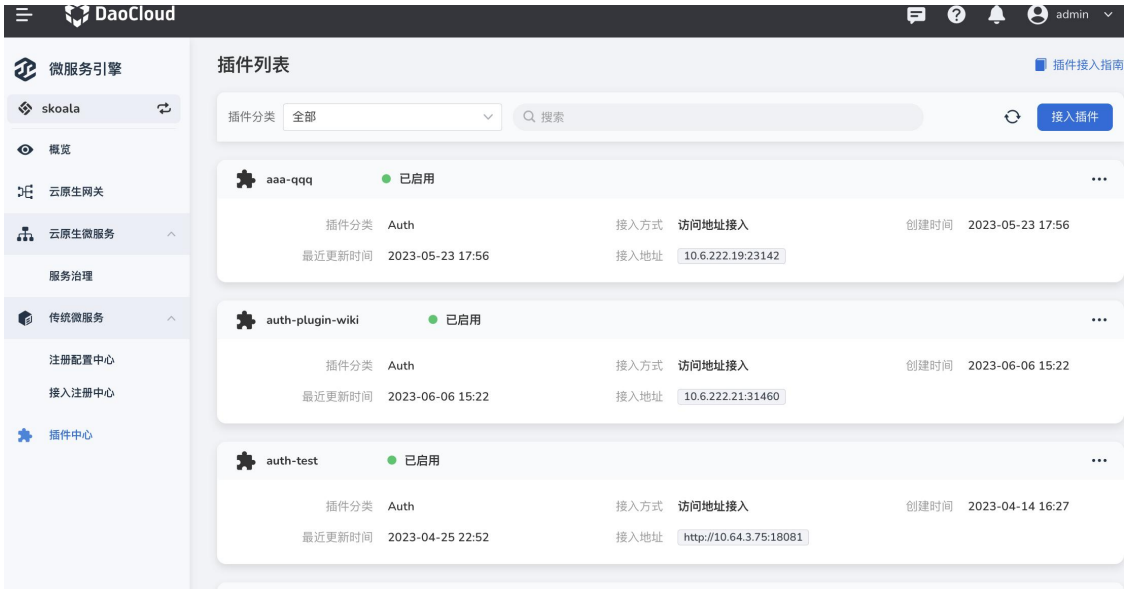
- 可视化界面与数据

  通过功能封装对外暴露简单易用的交互页面，支持通过简洁直观的 UI 界面进行所有操作，降低操作难度，真正实现点点鼠标就可以完成微服务整个生命周期中的各类管理操作。

插件中心介绍

微服务引擎支持接入 Auth、JWT、全局限流、Wasm 插件。

- Auth：接入该插件后，才能在创建域名时配置安全认证策略。
- JWT：接入该插件后，才能在创建域名时配置安全认证策略。
- 全局限流：接入该插件后，才能在创建云原生网关实例时启用全局限流插件。
- Wasm：接入该插件后，才能在云原生微服务治理页面

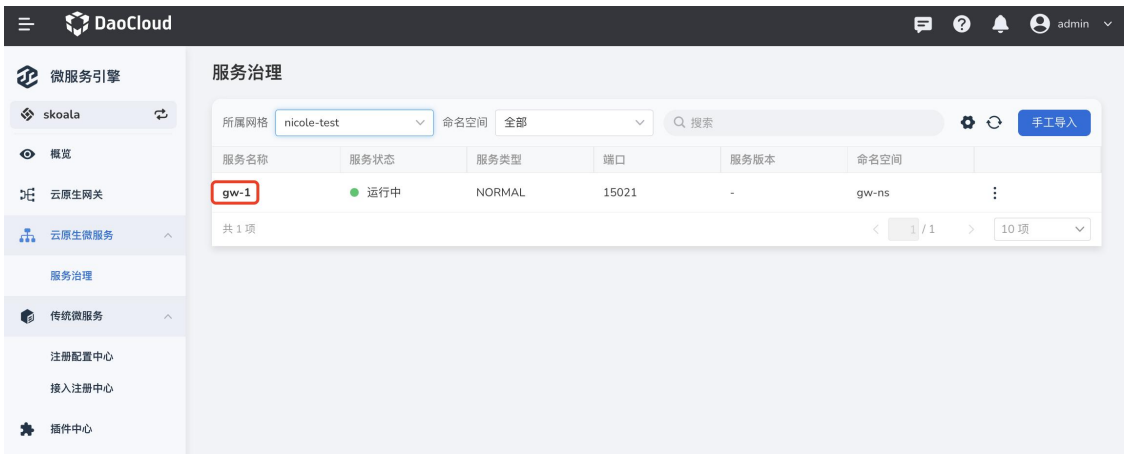


服务治理

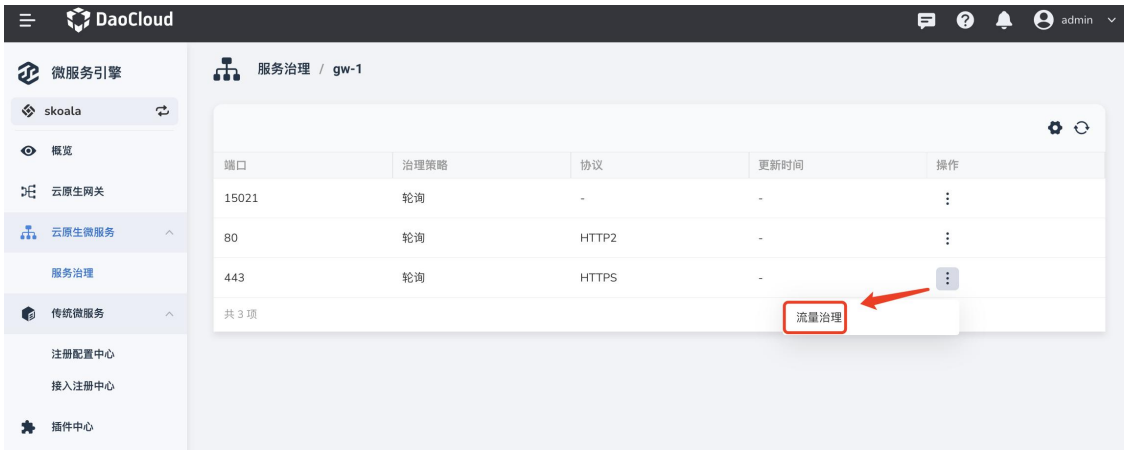
服务治理指基于 Service Mesh 对 DCE 5.0 服务网格中的服务进行东西向流量治理。

将网格服务导入云原生微服务引擎之后，就可以针对服务暴露的不同端口设置不同的东西向流量策略。

1. 点击服务名称



2. 选择目标端口



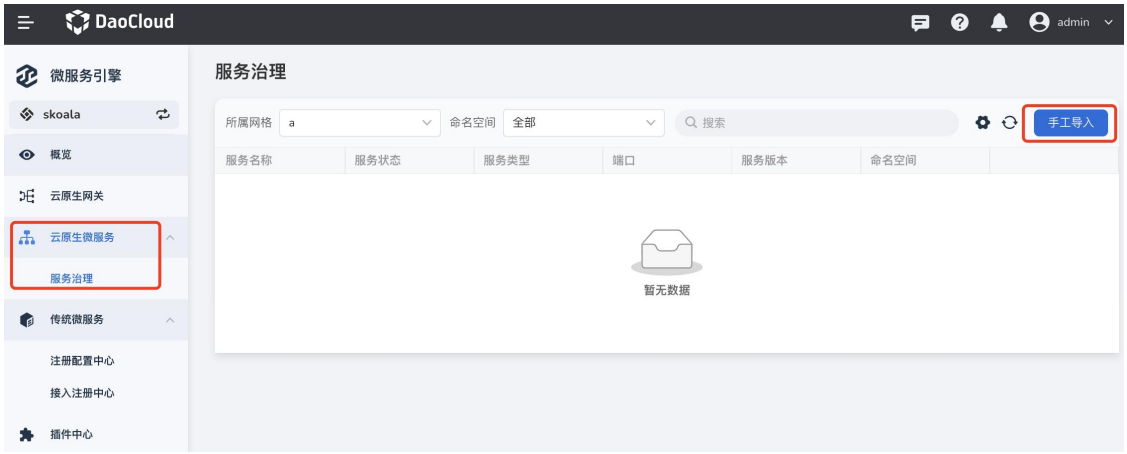
3. 根据需要配置流量策略，最后点击确定即可

导入/移除服务

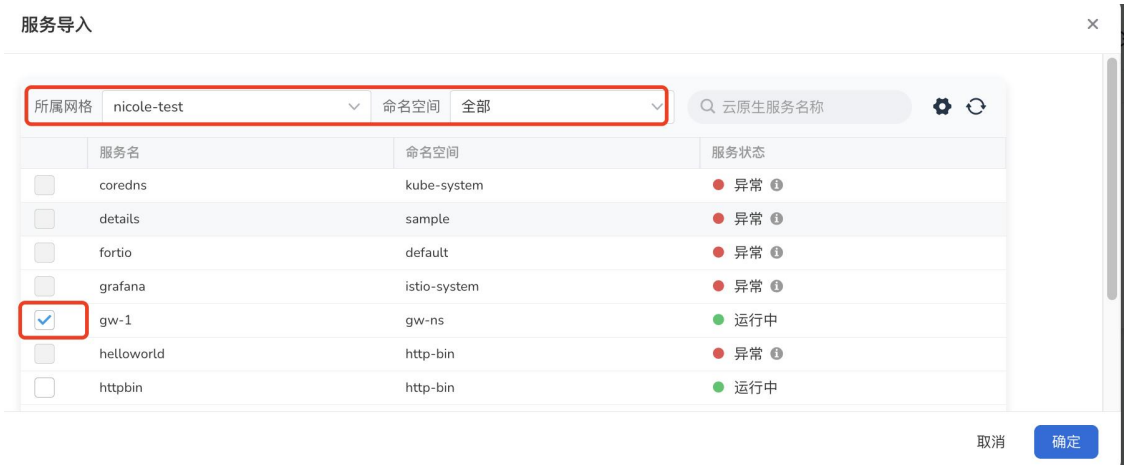
开始治理云原生微服务之前，首先需要将服务导入微服务引擎。目前仅支持从 DCE 5.0 服务网格模块下的网格中导入服务。

导入服务

1. 进入微服务引擎模块，在左侧点击云原生微服务–>服务治理，然后在右上角点击手工导入

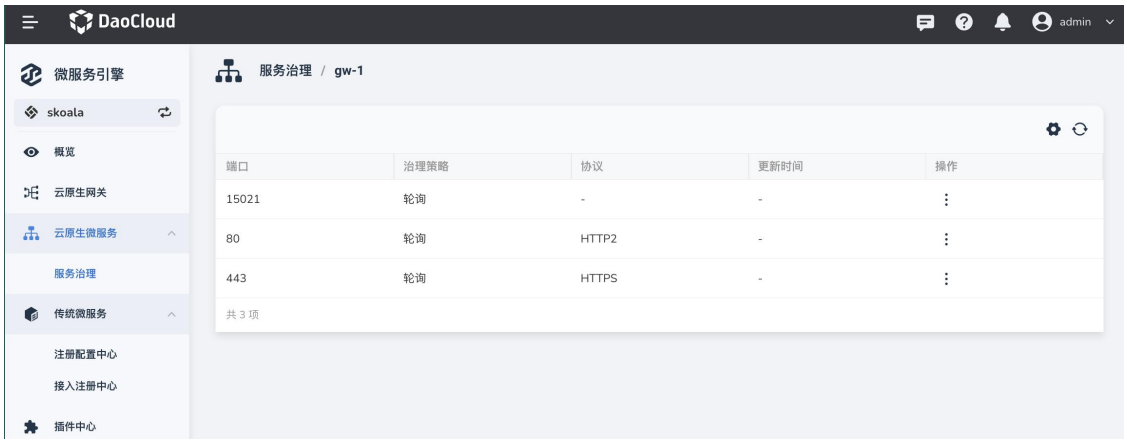


2. 筛选目标服务所在的服务网格和命名空间，勾选该服务，然后点击确定即可。



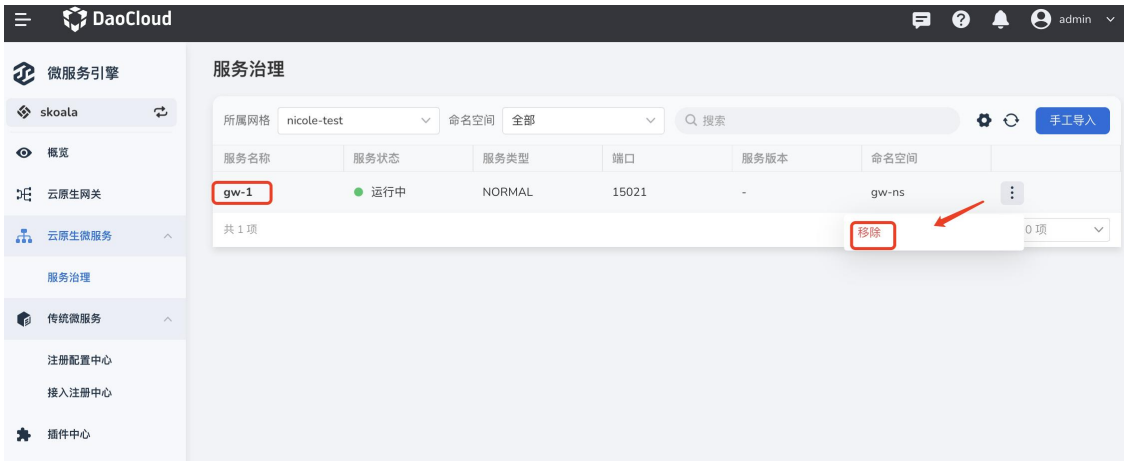
查看服务

查看已经导入的所有云原生微服务，点击服务名称进一步查看服务暴露的端口和协议。



移除服务

在右侧操作栏移除不需要的微服务。



网关压力测试报告

本文介绍 DCE 5.0 云原生网关在不同场景下的性能表现，便于您根据需求为网关配置合适的资源。

测试环境

开始测试之前，需要部署 DCE 5.0，下载安装测试工具，并准备好用于测试的压力机。

对象	角色	说明
DCE 5.0 云原生网关	测试对象	采用一主一从的模式，部署位置：172.

对象	角色	说明
Locust	测试工具	30.120.211 采用 1+4 的主从分布式运行模式，四台压测机的资源

对象	角色	说明
Nginx	用于测试网关性能的演示服务	配置均为8核8G通过DCE 5.0云原生网关访问，访问地址为

对象	角色	说明：http://172.30.120.211:30296/
contour	DCE 5.0 云原生网关的控制面	版本为 1.23.1
envoy	DCE 5.0 云原生网关的数据面	版本为 1.24.0
全局管理	DCE 5.0 云原生网关依赖的组件	版本为

对象	角色	说明
容器管理	DCE 5.0 云原生网关依赖的组件	0.12.1 版本为 0.13.1
微服务引擎	DCE 5.0 云原生网关依赖的组件	版本为 0.15.1

性能指标

- 吞吐量（RPS）：每秒处理的请求数。结合 CPU 使用率，判断 DCE 5.0 云原生网关在特定资源配置下每秒可以处理的并发请求数。吞吐量越高，说明网关性能越好
- CPU 使用率：测试处理特定数量的并发请求时，DCE 5.0 云原生网关实例的 CPU 使用情况。当 CPU 使用量达到 90% 以上时，认为 CPU 接近满载，此时的吞吐量（RPS）是当前配置能够正常处理的最大并发请求数。

测试脚本

- 在 Locust Web 机器执行如下命令，收集压测结果

  ```
  docker run -p 8089:8089 --network=host -v $PWD:/mnt/locust locustio/locust -f /mnt/locust/gateway-external-nginx.py --master
  ```

- 在 Locust 压测机执行如下命令，模拟用户访问，执行压力测试

```
docker run -p 8089:8089 --network=host -v $PWD:/mnt/locust locustio/locust -f /mnt/locust/gateway-external-nginx.py --worker --master-host=172.30.120.210
```

- 压测脚本 gateway-external-nginx.py

```
from locust import task
from locust.contrib.fasthttp import FastHttpUser

class ShellCard(FastHttpUser):

    host = "http://172.30.120.211:30296" # 被测服务的访问地址
    @task
    def test(self):
        header = {"Host": "external.nginx"}
        self.client.get("/", headers=header)
```

测试 nginx 吞吐量：三副本，不限制资源

测试结果

并发用户数

每秒吞吐量

CPU 使用量

分析

4

4300

58%——70%

资源未充分利用，理论上仍有能力处理更多请求。

8

5700

77%——83%

仍有少量资源空闲，可以尝试增加并发请求数，测试吞吐量上限。

12

6700

90%——95%

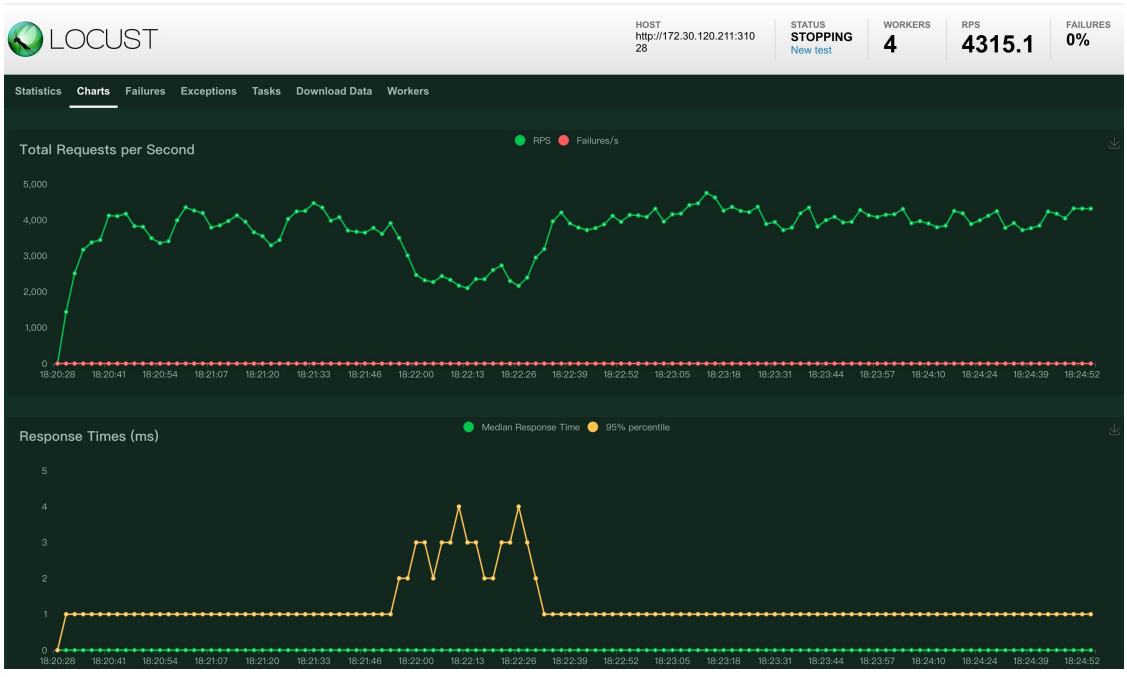
只有极少量资源空闲，CPU 接近满载，视为达到吞吐量上限。

!!! success

综上，当服务部署三副本且不限制资源用量时，DCE 5.0 云原生网关能够处理的最大并发请求数大约为 6000 到 7000，较同类产品而言可谓性能优异。

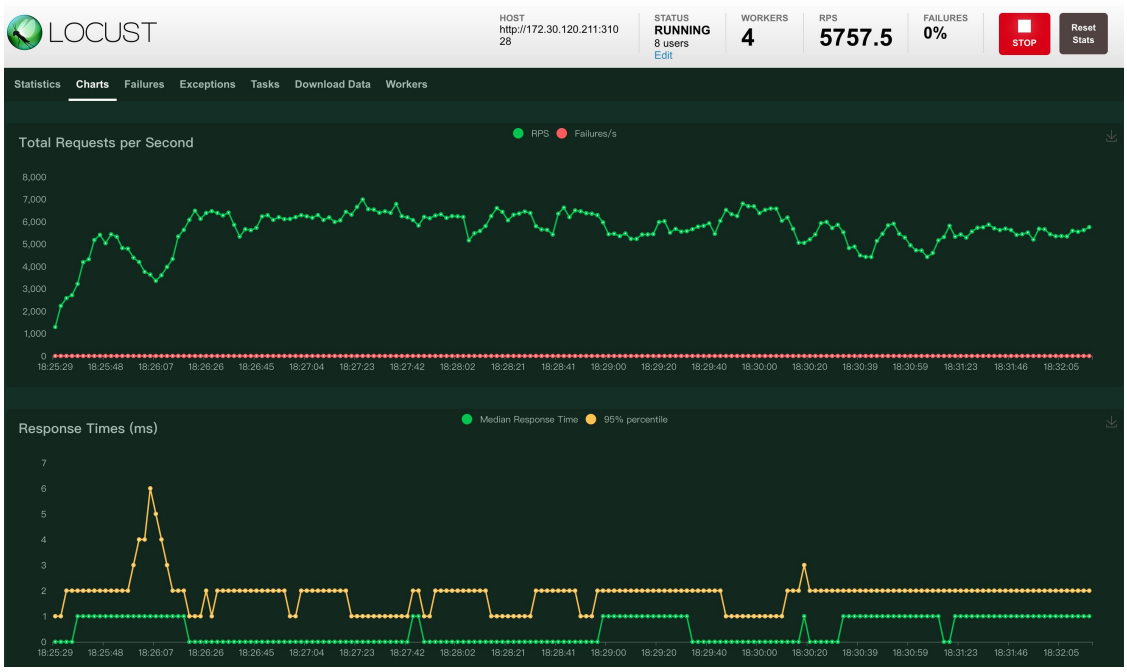
测试过程截图

- 并发用户数为 4





- 并发用户数为 8





- 并发用户数为 12





Statistics | Charts | Failures | Exceptions | Tasks | Download Data | Workers

Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_1629371e3652405d9bee4946db9ea88e	running	3	95.2%	45.27 MB
localhost.localdomain_464d51bdeb4349babceb164f3cc069d5	running	3	94%	42.17 MB
localhost.localdomain_4abded22990d4349b723f4b2bcf60b74	running	3	95.6%	43 MB
localhost.localdomain_63f64f990c72454cab5bcf9d5f32a556	running	3	90%	42.69 MB

探究 contour 资源配置对 envoy 的性能影响

DCE 5.0 云原生网关是在开源项目 contour 和 envoy 的基础上进一步研发优化而来。contour 充当网关的控制平面，envoy 充当数据平面。

创建 DCE 5.0 云原生网关时，系统要求必须为网关配置不低于 1 核 1 G 的资源。因此，本次测试中 contour 的资源限制最低为 1 核 1 G。

此外，为了更好地体现 contour 的资源配置的影响，将 envoy 的资源限制设置为 6 核 3 G，保证 envoy 自身始终具有较高的性能，不会因为自身资源不足而影响测试结果。

为保证压测机器资源负载正常，默认 Locust users 为 8。

测试结果

contour 资源规格

每秒吞吐量

CPU 使用量

分析

1 核 1 G

3700

53%——69%

contour 的资源配置从 1 核 1 G 到 2 核 1 G，再到 3 核 2 G，但网关每秒吞吐量一直维持在 3700 左右，上下变动幅度只有 100。CPU 使用率也维持在 50%——70%左右，整体变化非常小。

2 核 1 G

3600

55%——72%

3 核 2 G

3800

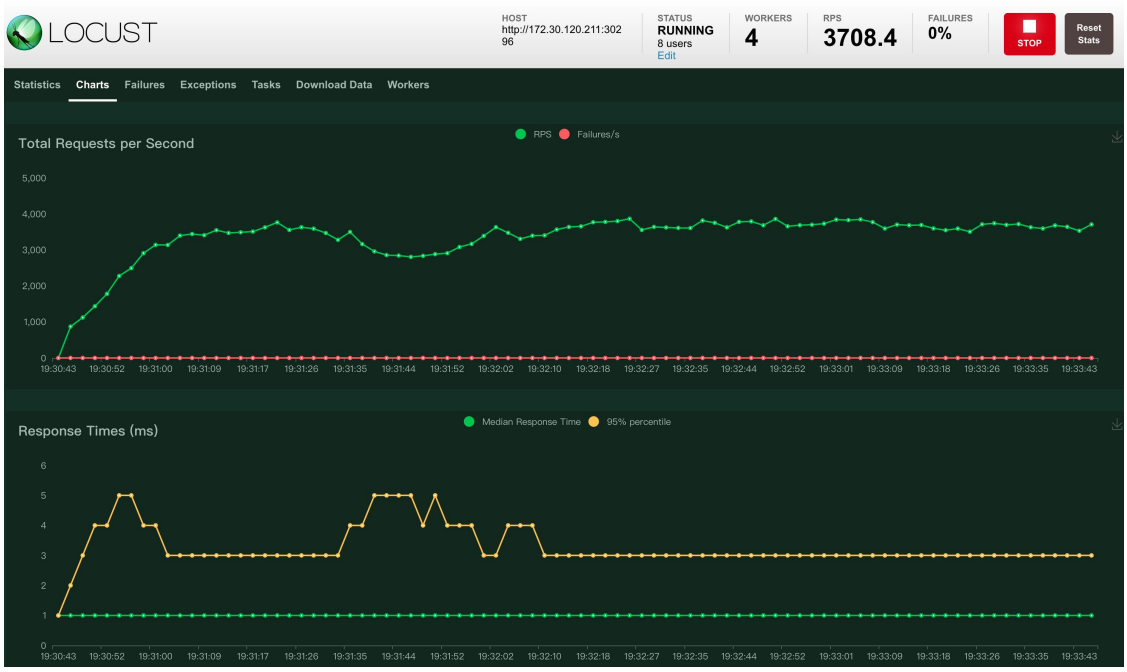
57%——69%

!!! success

这说明 contour 的资源配置对 envoy 的性能几乎没有影响。

测试过程截图

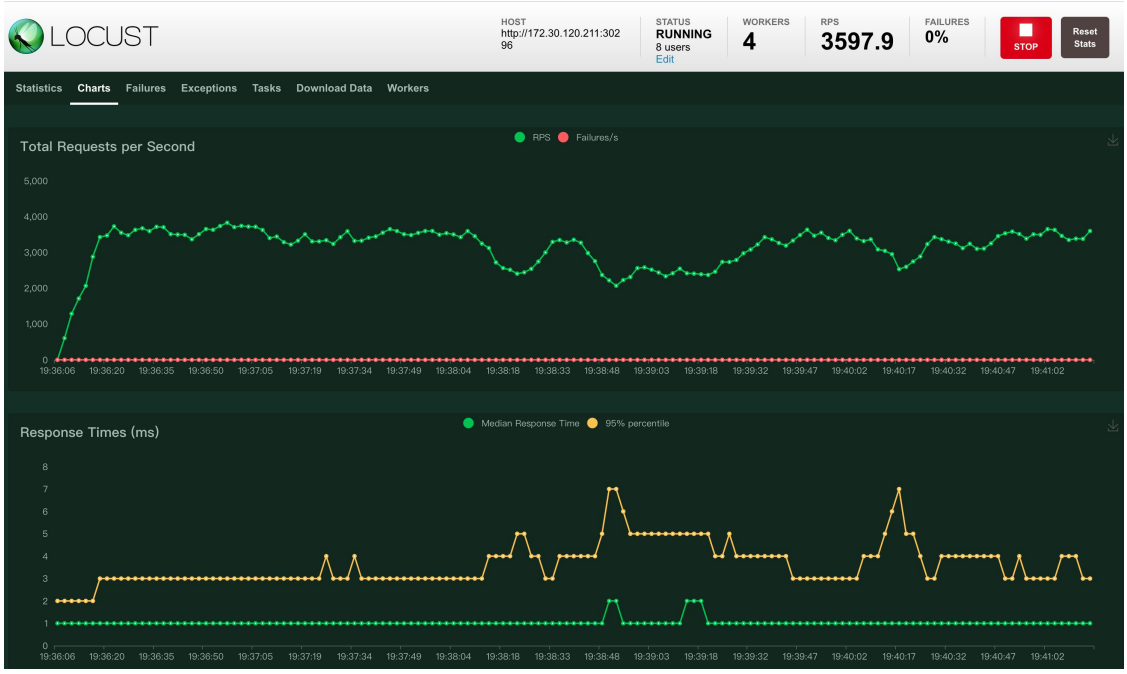
- contour 资源 1 核 1 G

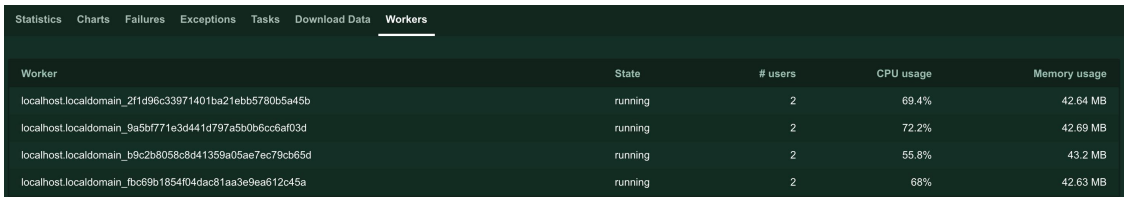




Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	2	64.4%	42.4 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	2	60.6%	40.52 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	2	69.6%	42.65 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	2	53.6%	42.4 MB

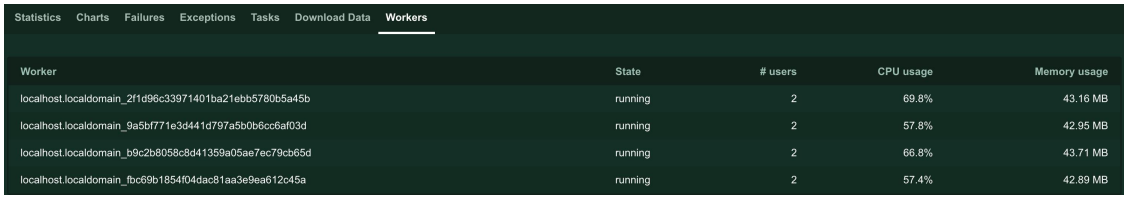
- contour 资源 2 核 1 G





- contour 资源 3 核 2 G





探究 envoy 资源配置对吞吐量的影响

envoy 固定为 1 副本，contour 配置为 1 核 1 G，被测试服务 nginx 配置为 3 副本，不限制资源用量。

测试结果

envoy 资源规格

并发用户数

每秒吞吐量

CPU 使用量

分析

1 核 1 G

4

1016

18%——22%

当 envoy 的资源配置不变时，即使并发用户数成倍增加，网关吞吐量也变化不大。

8

1181

19%——20%

16

1090

19%——22%

2 核 1 G

4

2103

28%——41%

并发用户数为 4 时，当资源配置从 1 核增加到 2 核，吞吐量也增加了 1000 左右。

8

2284

38%——47%

3 核 1 G

8

3355

59%——70%

并发用户数为 8 时，当资源配置从 1 核增加到 3 核，吞吐量也增加了 2000 左右。

12

3552

52%——59%

4 核 2 G

8

3497

58%——80%

并发用户数为 12 时，当资源配置从 3 核增加到 5 核，吞吐量也增加了 1000 左右。

12

4250

78%——86%

5 核 2 G

8

3573

60%——81%

12

4698

68%——78%

6 核 2 G

12

4574

78%——85%

当配置为 6 核 2 G 时，吞吐量为 5400 左右，CPU 使用量也到达了 90% 以上，接近满载。

16

5401

90%以上

!!! success

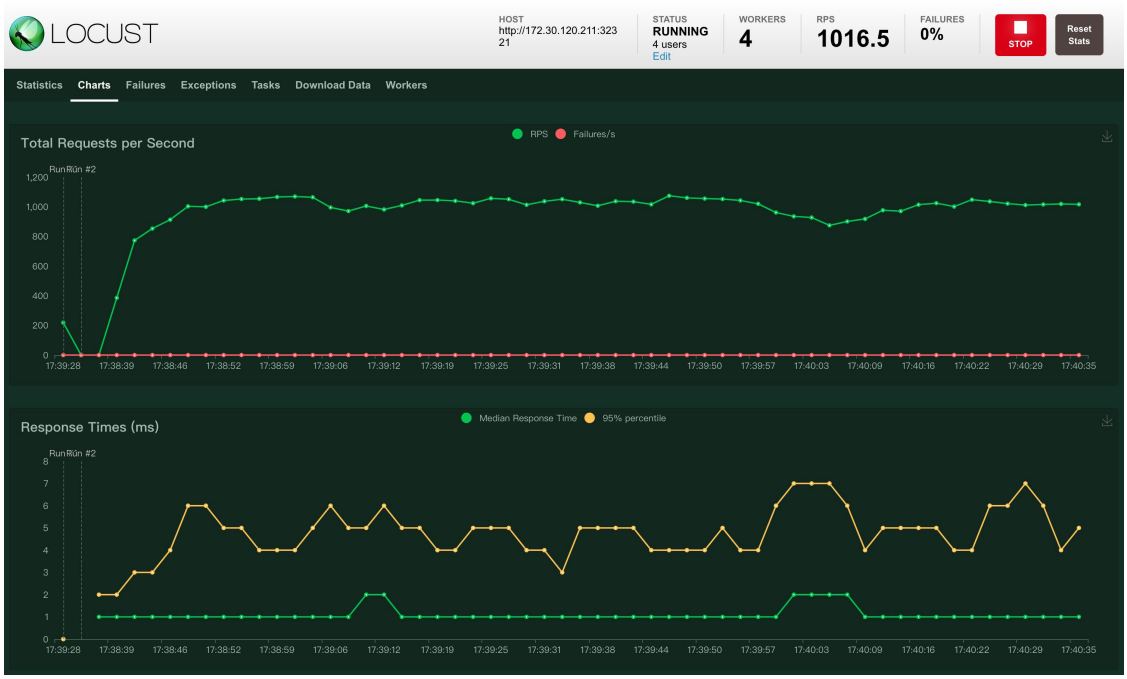
综上：

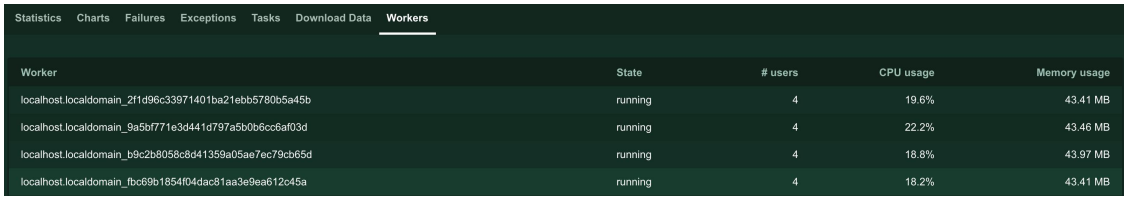
- envoy 的 CPU 配置对吞吐量起决定性因素。
- 在当前压测资源下，通过 envoy 访问 nginx 的吞吐量能够达到直接访问 nginx 的吞吐量的 80% 以上。

测试过程截图

当 envoy 配置为 1 核 1G

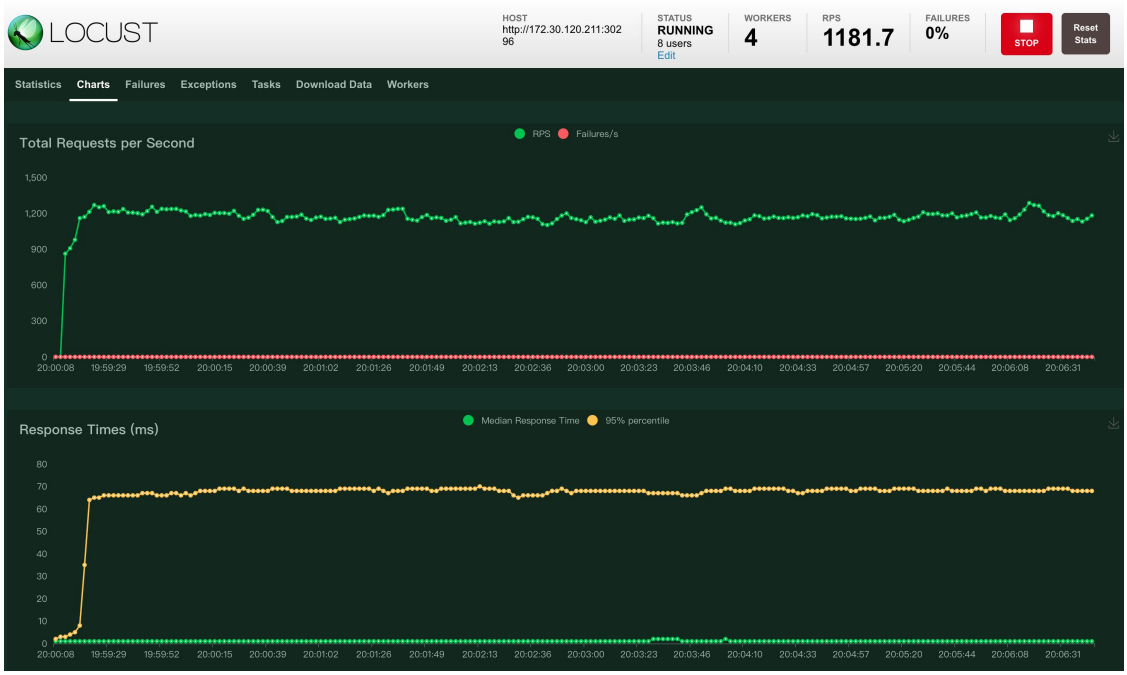
- 并发用户数为 4

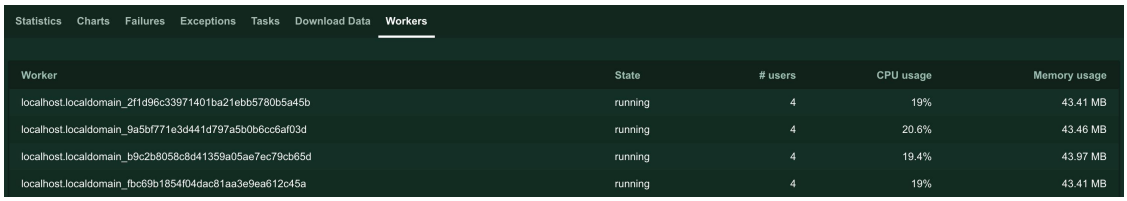




Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	4	19.6%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	4	22.2%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	4	18.8%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	4	18.2%	43.41 MB

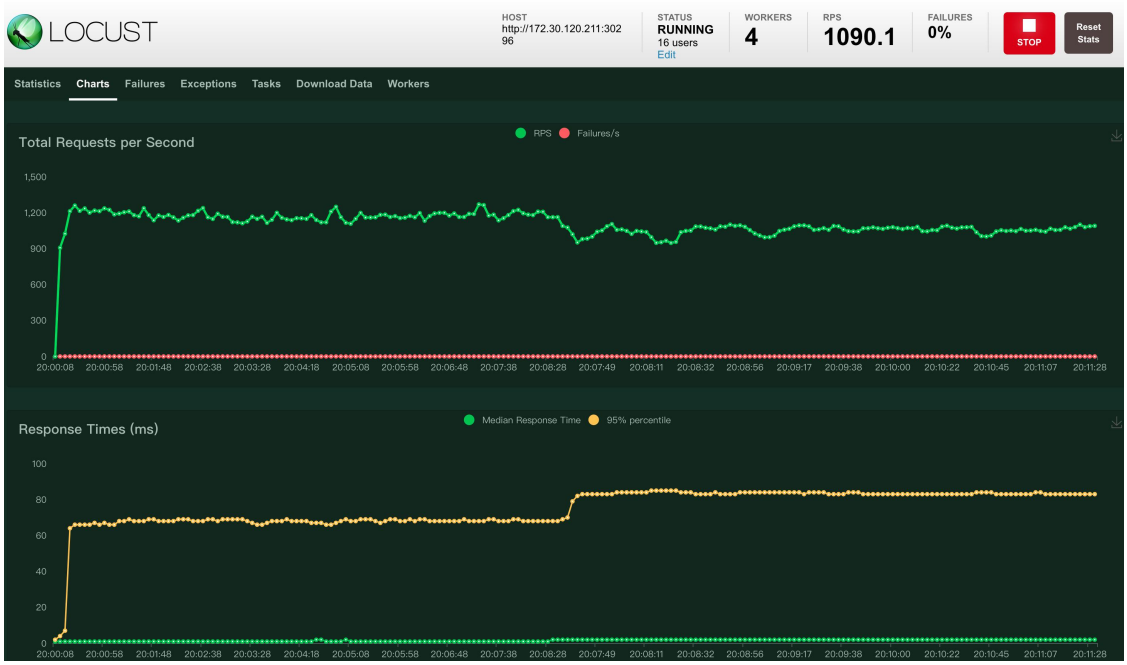
- 并发用户数为 8





Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	4	19%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	4	20.6%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	4	19.4%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	4	19%	43.41 MB

- 并发用户数为 16

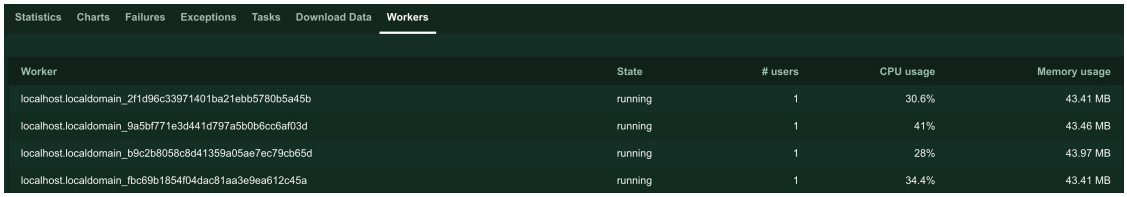




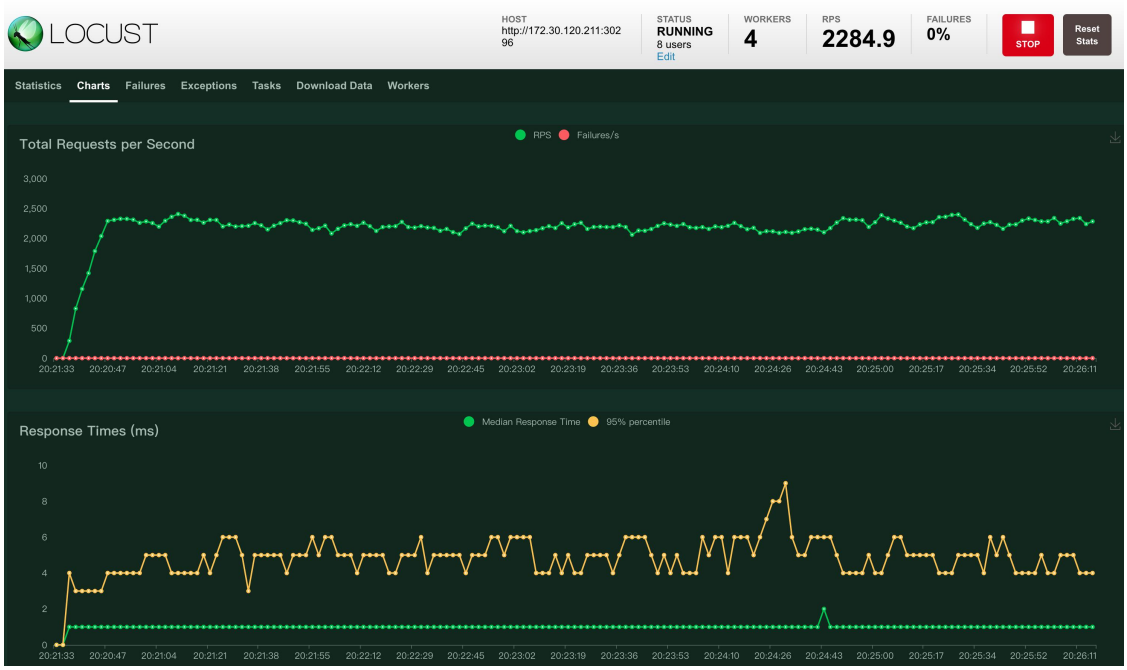
当 envoy 配置为 2 核 1G

- 并发用户数为 4





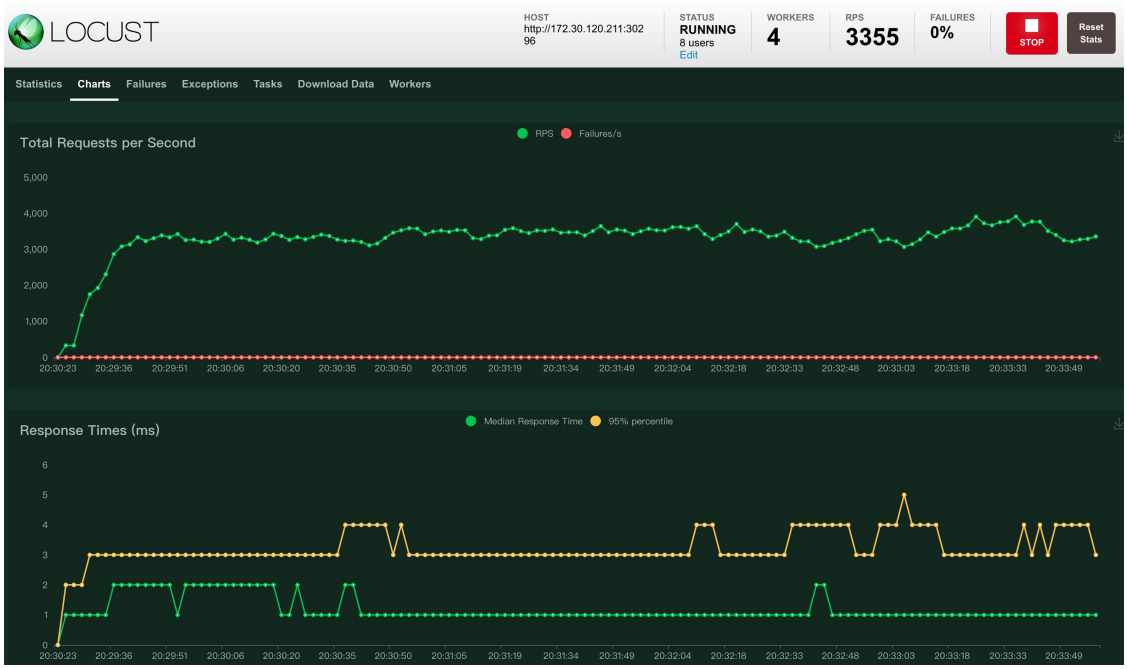
- 并发用户数为 8

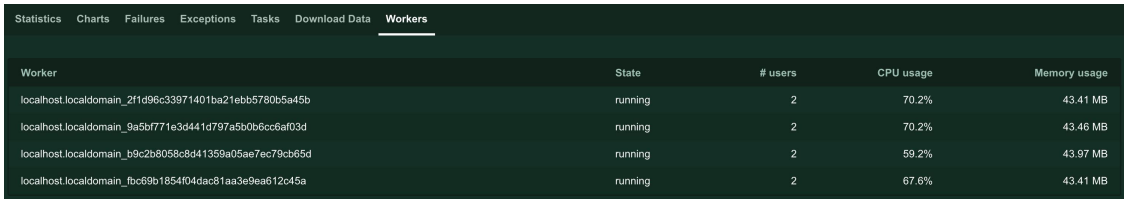




当 envoy 配置为 3 核 1G

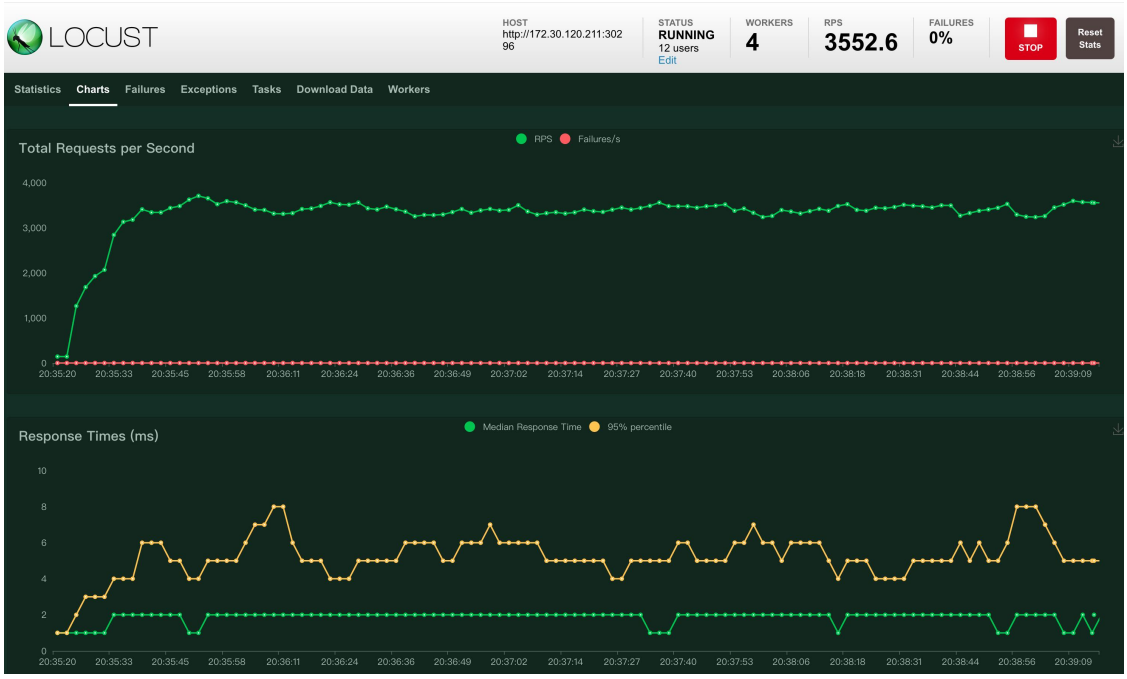
- 并发用户数为 8

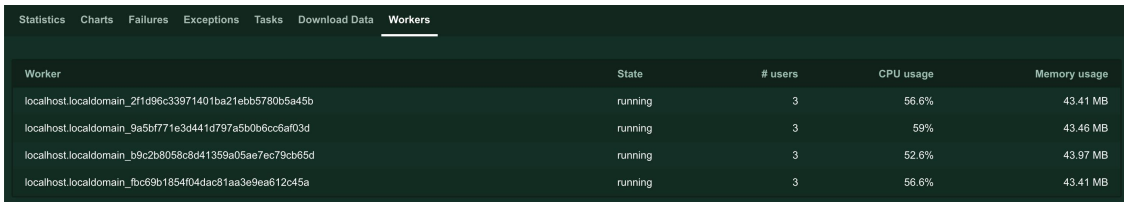




Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	2	70.2%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	2	70.2%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	2	59.2%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	2	67.6%	43.41 MB

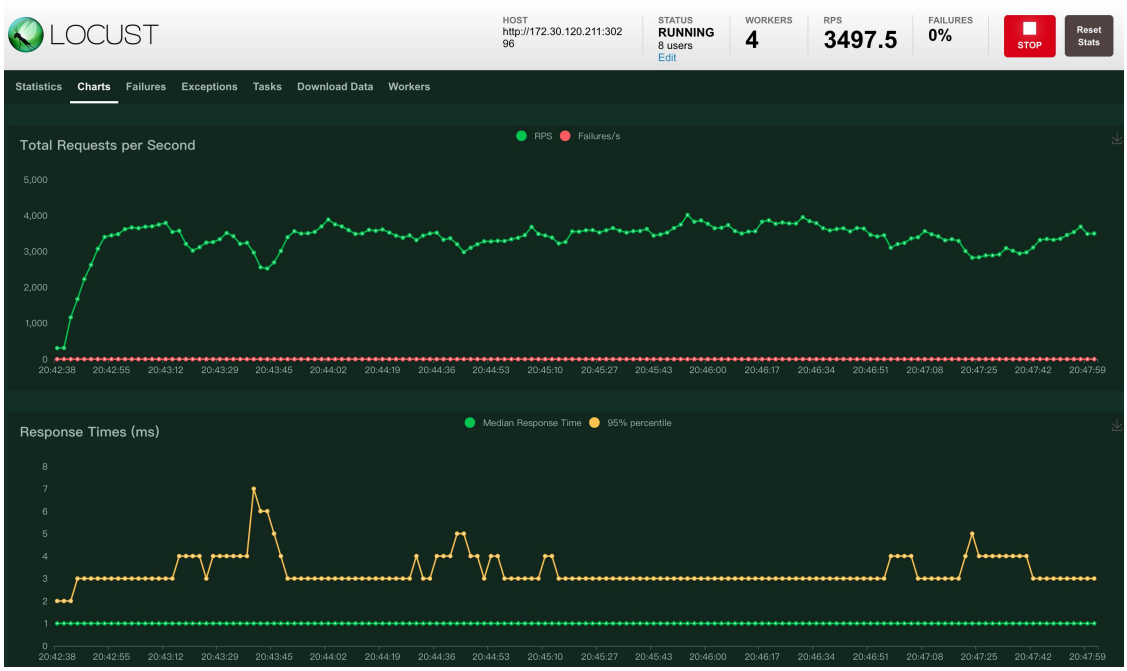
- 并发用户数为 12





当 envoy 配置为 4 核 2G

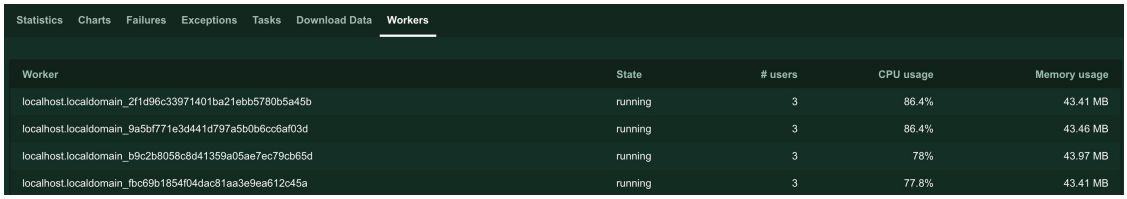
- 并发用户数为 8





- 并发用户数为 12

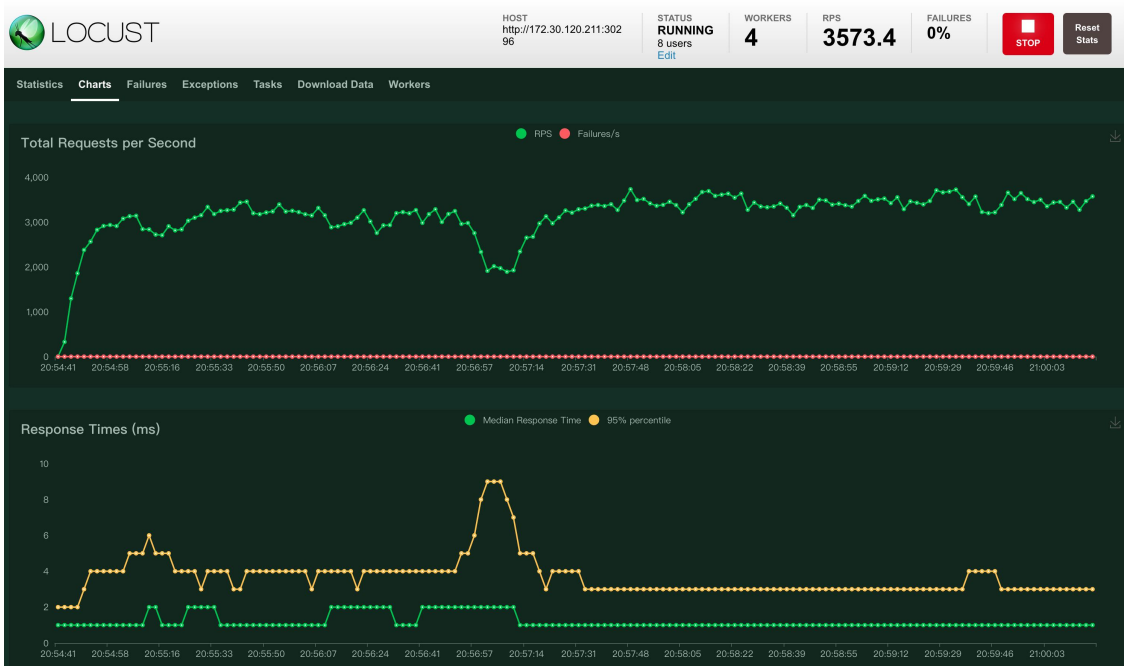


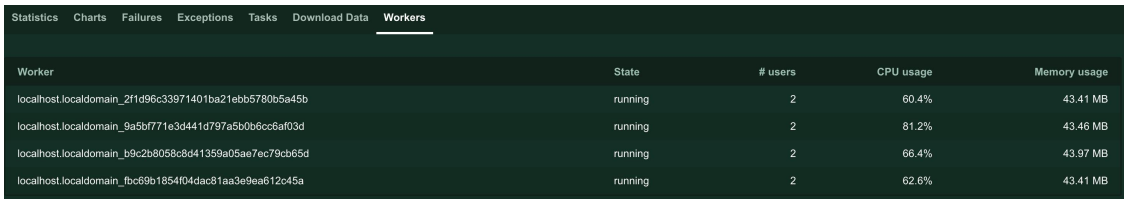


Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	3	86.4%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	3	86.4%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	3	78%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	3	77.8%	43.41 MB

当 envoy 配置为 5 核 2G

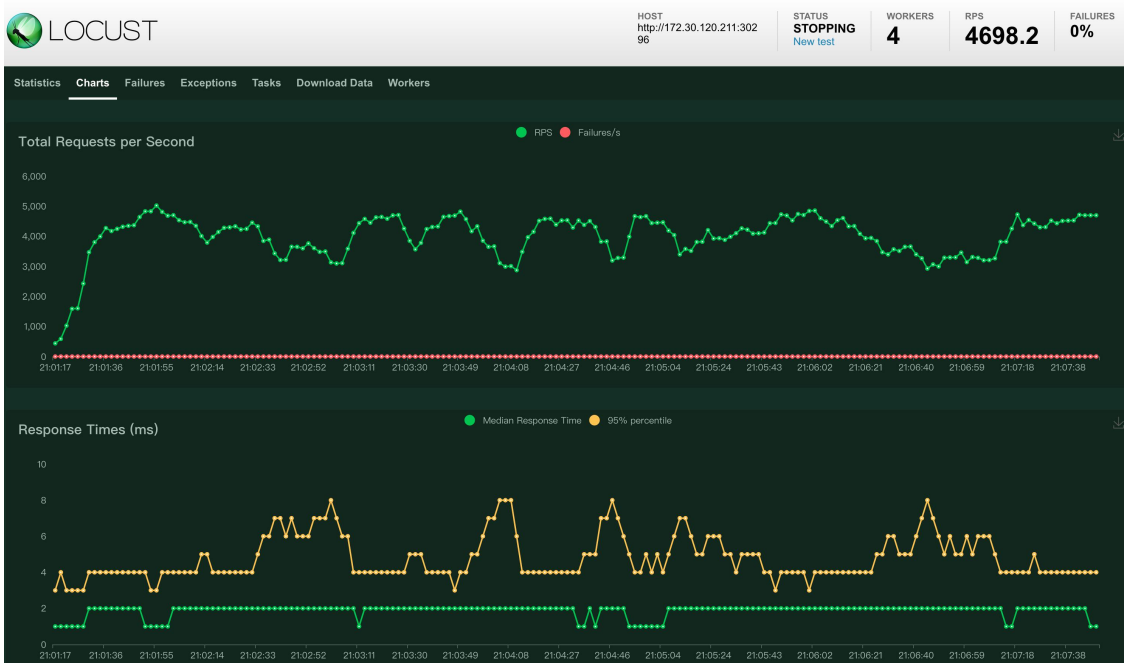
- 并发用户数为 8

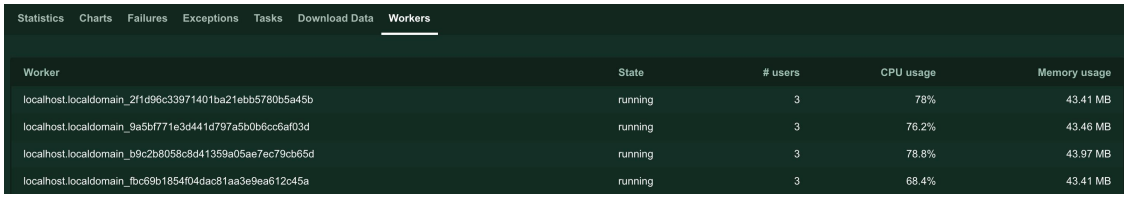




Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	2	60.4%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	2	81.2%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	2	66.4%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	2	62.6%	43.41 MB

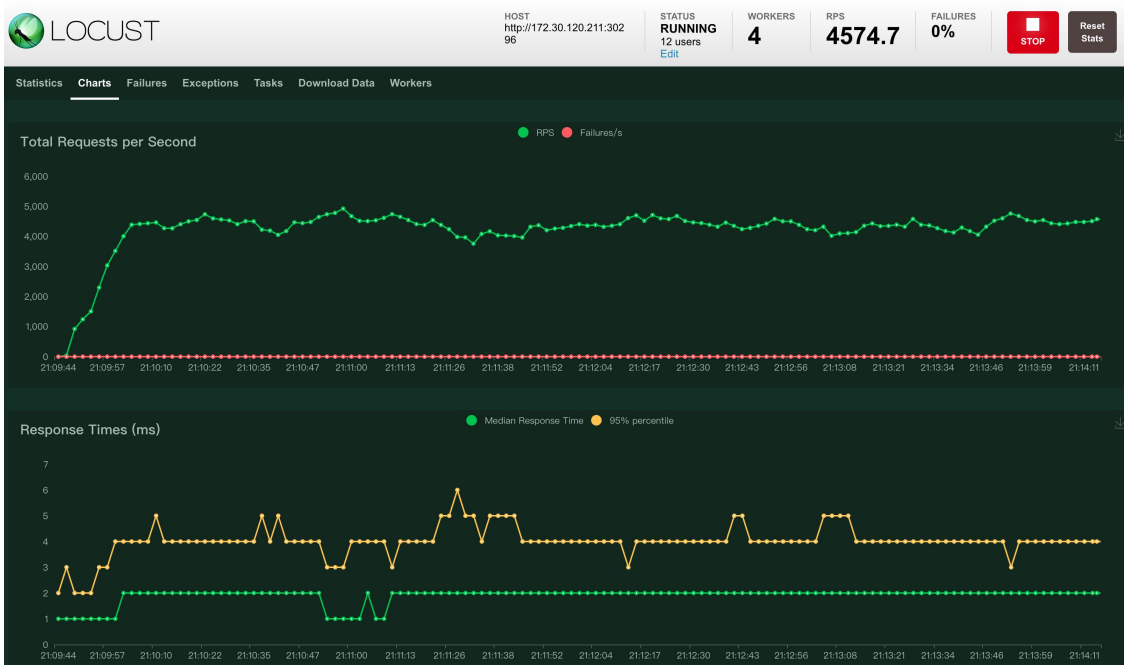
- 并发用户数为 12





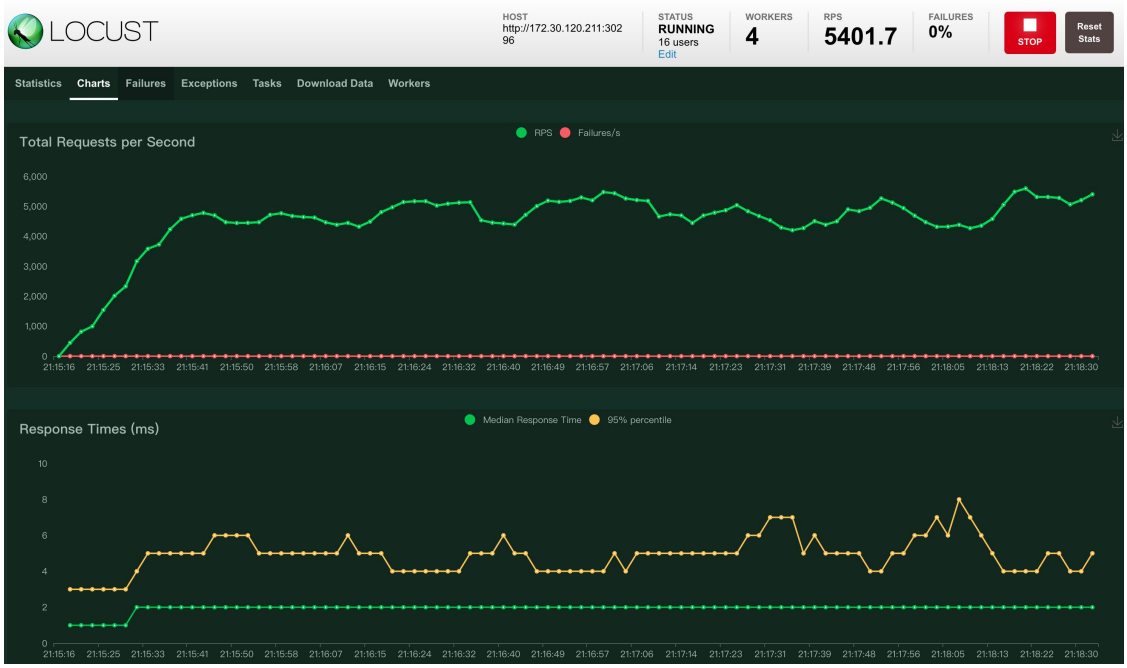
当 envoy 配置为 6 核 2G

- 并发用户数为 12





- 并发用户数为 16





Worker	State	# users	CPU usage	Memory usage
localhost.localdomain_2f1d96c33971401ba21ebb5780b5a45b	running	4	92.6%	43.41 MB
localhost.localdomain_9a5bf771e3d441d797a5b0b6cc6af03d	running	4	91.8%	43.46 MB
localhost.localdomain_b9c2b8058c8d41359a05ae7ec79cb65d	running	4	93.6%	43.97 MB
localhost.localdomain_fbc69b1854f04dac81aa3e9ea612c45a	running	4	93.8%	43.41 MB

Nacos 压力测试报告

本文通过 JVM 虚拟机测试托管 Nacos 注册配置中心在不同资源规格下的性能表现，便于您根据实际的业务场景为 Nacos 提供合适的资源配额。

性能指标

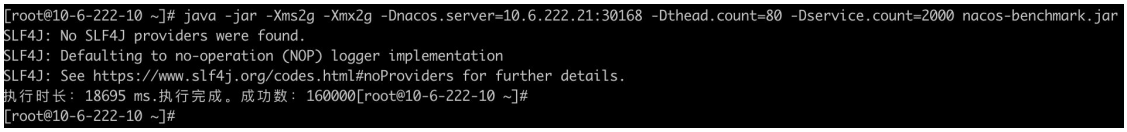
TPS：通常指服务器每秒处理的事务数（Transactions Per Second）。在本文中具体指 Nacos 服务器每秒可以处理多少个服务注册请求。

测试命令

java –jar –Xms2g –Xmx2g –Dnacos.server=10.6.222.21:30168 –Dthead.count=100 –Dservice.count=500 nacos–client–1.0–SNAPSHOT–jar–with–dependencies.jar

- –Xms2g：JVM 的初始内存
- –Xmx2g：JVM 可使用的最大内存
- –Dnacos.server：Nacos 注册配置中心的访问地址
- –Dthead.count：线程数，对应测试结果中的并发数
- –Dservice.count：每个线程的请求数，对应测试结果中的服务数
- nacos–client–1.0–SNAPSHOT–jar–with–dependencies.jar：压测使用的 Jar 包，实际测试过程中在本地将其重命名为 nacos–benchmark.jar

测试过程截图



process

测试结果

规格

节点数

并发数

服务数

执行时间(毫秒)

TPS

1 Core 2 GiB

1

30

1000

17086

1755.8

50

1000

23315

2144.5

100

500

26194

1908.4

2 Core 4 GiB

1

30

1000

8794

3409.1

50

1000

15460

3225.8

100

500

15699

3184.7

4 Core 8 GiB

1

50

1000

11843

4221.9

80

1000

14510

5513.4

100

500

9501

5262.6

8 Core 16 GiB

1

50

2000

14391

6948.8

80

2000

18695

8558.4

100

1000

11686

8557.2

探究 JVM 资源配置对 Nacos 的性能影响

为探究 JVM 的资源配置对托管 Nacos 注册配置中心的性能影响，在 Nacos 申请资源为 2 Core 4 GiB 时，为 JVM 也配置相同的资源，即 2 Core 4 GiB。

这种情况下得出的测试结果如下：

规格

节点数

并发数

服务数

执行时间(毫秒)

TPS

2 Core 4 GiB

1

30

1000

11003

2727.3

50

1000

15006

3333.3

100

500

13963

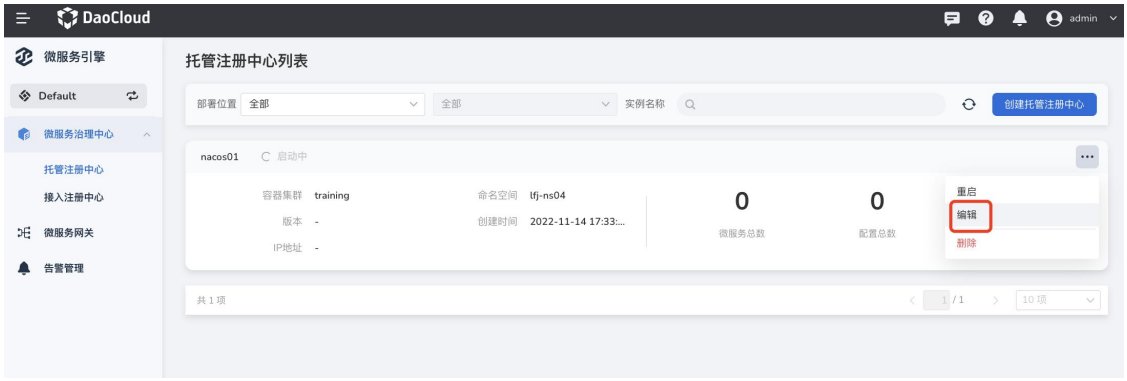
3580.9

!!! success

TPS 与调整前的数据不相上下。这说明，调整 JVM 对 Nacos 的性能基本没有影响。

更新托管注册中心

1. 在接入注册中心列表页选择需要更新的注册中心，在右侧点击 ⋯ 并选择编辑。

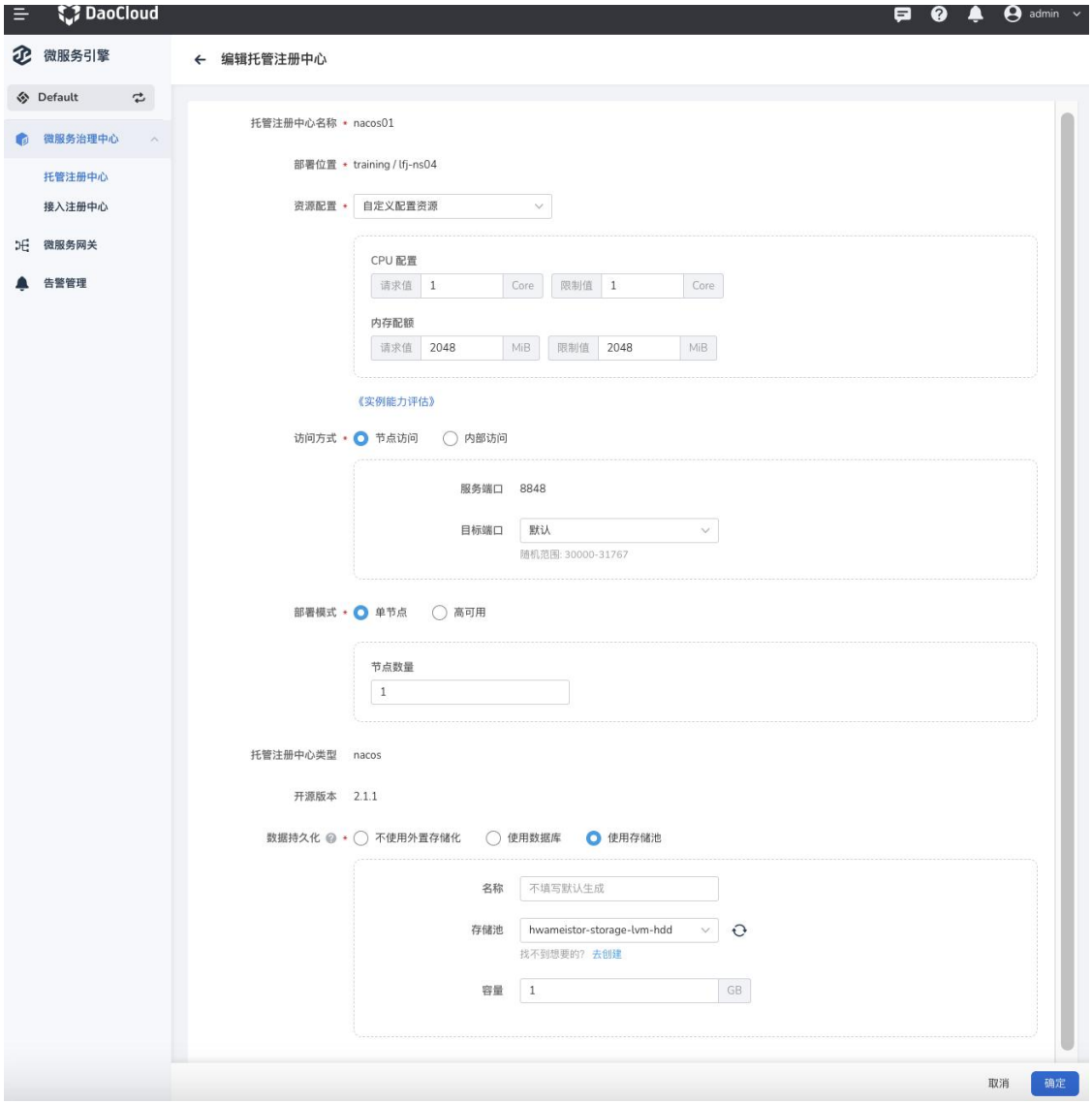


进入更新页面

2. 更新注册中心的配置，然后在页面底部点击确定。

!!! warning

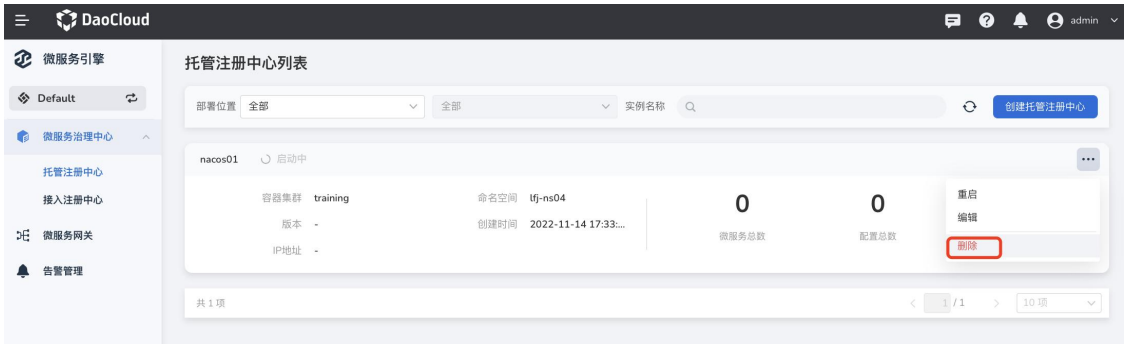
- 修改存储池/数据库，变动存储位置后，之前的数据不会随之迁移！
- 托管注册中心名称和部署位置不可编辑。


DaoCloud
admin
微服务引擎
Default
微服务治理中心
托管注册中心
接入注册中心
微服务网关
告警管理
编辑托管注册中心
托管注册中心名称 * nacos01
部署位置 * training / lfj-ns04
资源配置 * 自定义配置资源
CPU 配置
请求值 1 Core 限制值 1 Core
内存配额
请求值 2048 MiB 限制值 2048 MiB
《实例能力评估》
访问方式 * 节点访问 内部访问
服务端口 8848
目标端口 默认
随机范围: 30000-31767
部署模式 * 单节点 高可用
节点数量
1
托管注册中心类型 nacos
开源版本 2.1.1
数据持久化 * 不使用外置存储化 使用数据库 使用存储池
名称 不填写默认生成
存储池 hwameistor-storage-lvm-hdd
找不到想要的? 去创建
容量 1 GB
取消
确定

更新配置

删除注册中心

1. 在托管注册中心列表页选择需要删除的注册中心，在右侧点击 ⋯ 并选择删除。



进入删除页面

2. 输入注册中心的名称，点击移除。



确认名称

!!! note

接入型的注册中心仅支持`移除`操作，而托管型的注册中心仅支持`删除`操作。二者的区别在于：

– 移除：只是将注册中心从 DCE 5.0 的微服务引擎中移除，不会删除原有的注册中心和数据，后续还可以再次接入该注册中心。

– 删除：删除注册中心及其中的所有数据，后续无法再次使用该注册中心，需要重新创建新的注册中心。

创建托管注册中心

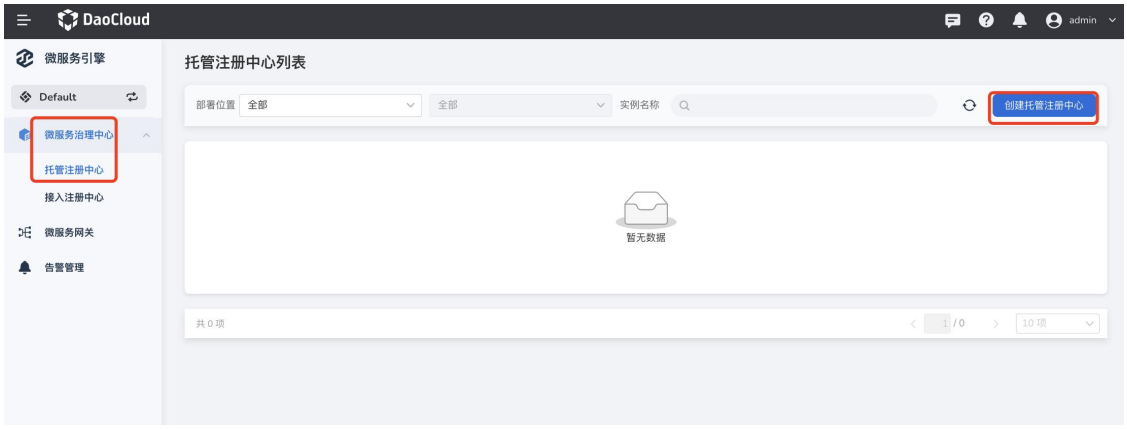
微服务引擎支持托管 Nacos 注册中心，即在微服务治理中心从零创建一个 Nacos 类型的全新注册中心，并且可以通过微服务治理中心全面管理该注册中心。相对于接入型注册中心而言，托管型注册中心支持更多操作，包括查看注册中心实例的基础信息、微服务命名空间管理、微服务列表、微服务配置列表、监控告警、日志查看、插件中心等。

!!! note

– 需要事先在目标集群的 **skoala-system** 命名空间中安装 skoala-init 组件，具体步骤可参考[管理 Helm 应用](../../../kpanda/user-guide/helm/helm-app.md)。
– 如果所选集群中没有 **skoala-system** 命名空间，可参考[创建命名空间](../../../kpanda/user-guide/namespaces/createns.md)创建一个名为 **skoala-system** 的命名空间。

创建托管注册中心的步骤如下。

1. 在左侧导航栏点击微服务治理中心->托管注册中心，然后在页面右上角点击创建托管注册中心，进入托管注册中心实例的创建页面。

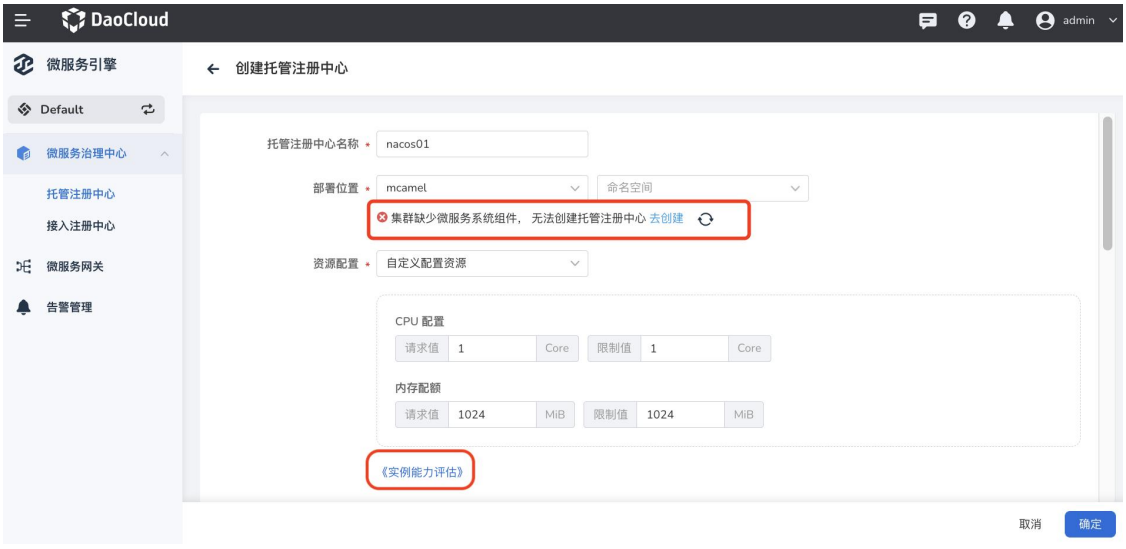


进入创建注册中心页面

2. 填写配置信息。

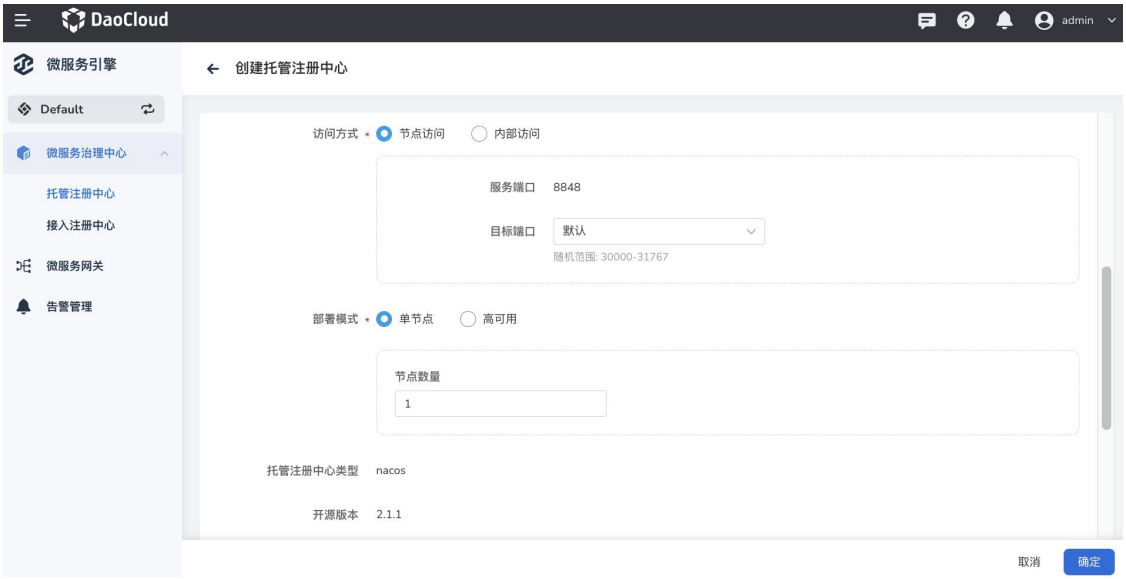
需要注意的是：

- 注册中心名称：支持输入字母、数字和分隔符（-），注册中心创建之后不可更改名称。
- 部署位置：系统会自动校验所选集群下的 **skoala-system** 命名空间中是否安装了 skoala-init 组件。
  - 如未安装，则无法创建注册中心。可以根据页面提示去安装该组件。
  - 注册中心创建之后不可更改部署位置。
- 资源配置：可直接选择 1 核 2G、2 核 4G 等配置，也可以自定义配置资源限额。
  - 1 核 2G 指 CPU 的请求值和限制值分别为 2 核，内存的请求值和限制值分别为 2 G，以此类推。
  - 点击《实例能力评估》可以查看在 2 Core 4 GiB、4 Core 8 GiB 和 8 Core 16 GiB 等主流规格下的吞吐量（TPS）。



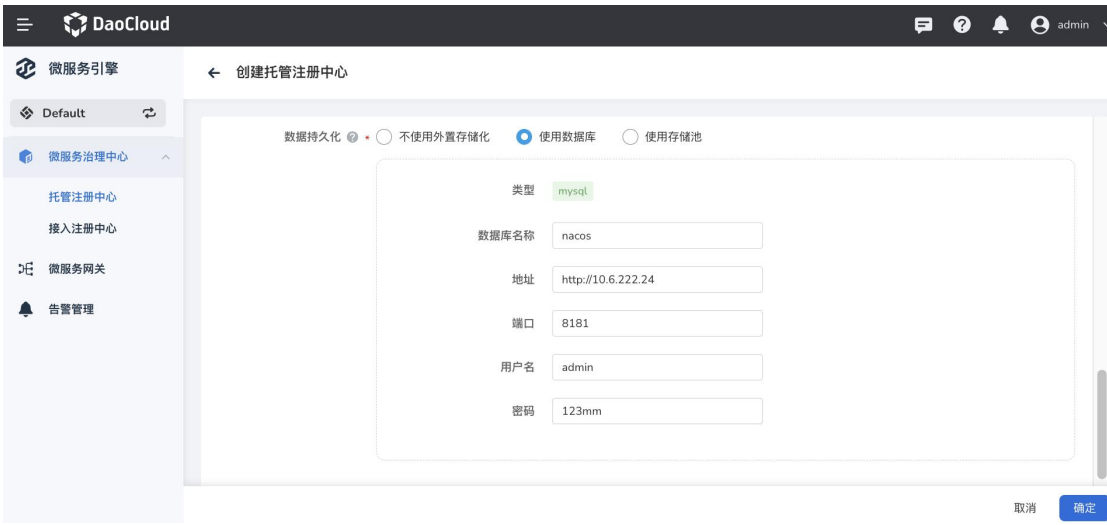
进入创建注册中心页面

- 访问方式：选择节点访问可通过 **服务端口+目标端口** 的方式从外部访问注册中心，选择内部访问则只能在所在的集群范围内通过服务端口访问注册中心。默认的服务端口为 8848。
- 部署模式：选择高可用模式时，节点数量不少于 3 个。生产环境下建议使用高可用模式。



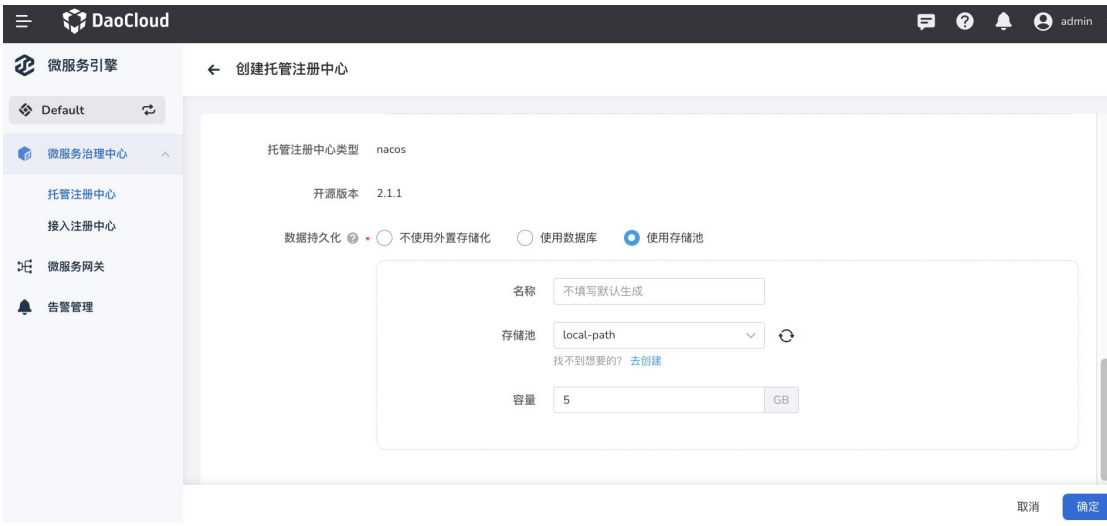
进入创建注册中心页面

- 数据持久化：建议使用外置存储化。
  - 不使用外置存储化：数据存储在 Nacos 所在的 Pod 的文件系统里。Pod 重启之后数据会遗失，因此建议使用外部存储。
  - 使用数据库：填写数据库的名称、地址、端口、用户名和密码



进入创建注册中心页面

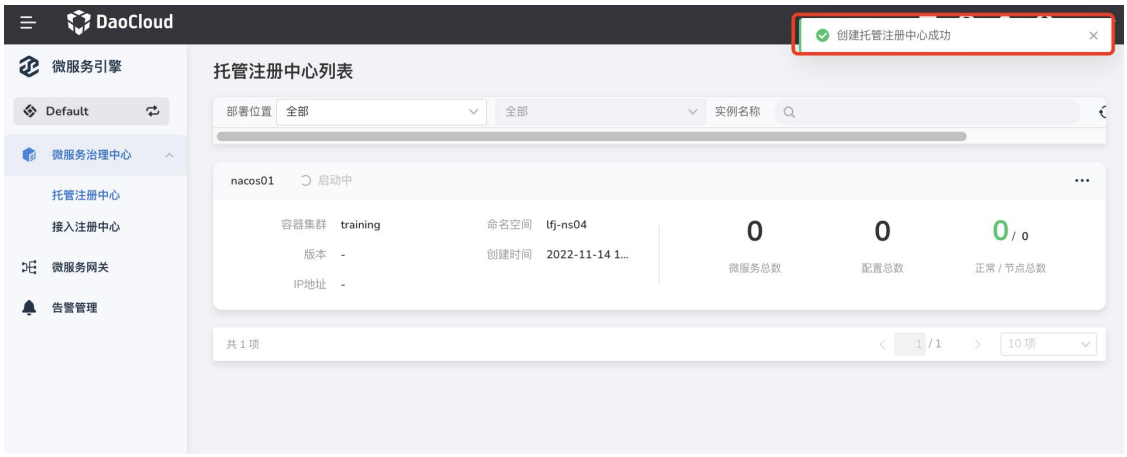
- 使用存储池：选择注册中心所在集群中已有的存储池并配置存储容量。



进入创建注册中心页面

3. 在页面底部点击确定。

   如果操作正确，页面右上角会弹出创建成功的消息，托管注册中心列表页会展示新建的注册中心实例。



创建成功

!!! info

– 新建注册中心需要一段时间进行初始化，其间处于“启动中”状态。初始化完成后进入“运行中”状态。

– 点击注册中心的名称可以查看所在集群/命名空间、运行状态、资源配额、服务端口、存储配置、节点列表等基础信息。

微服务命名空间

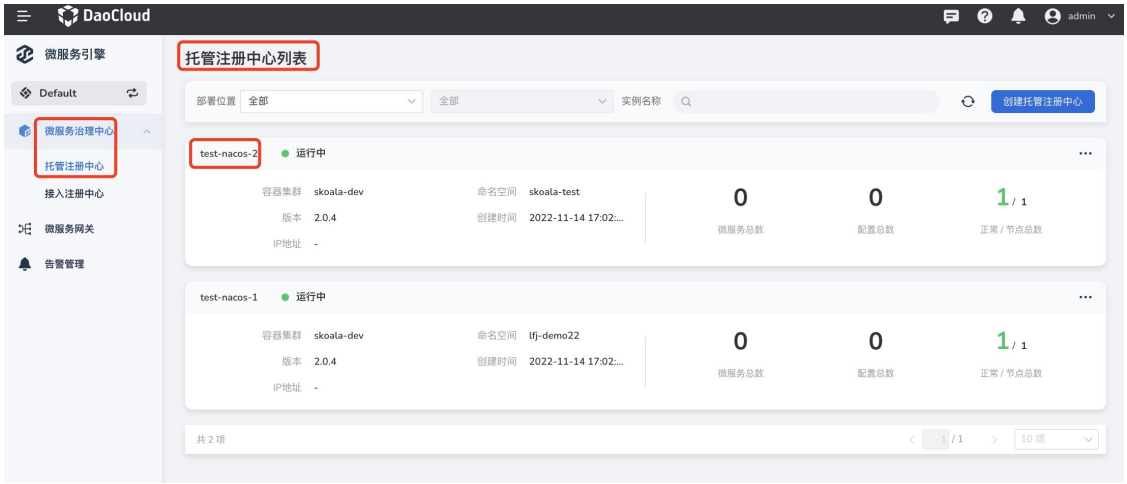
微服务命名空间可用于隔离生产、开发、测试等不同环境下的服务、配置等资源。微服务引擎模块中的命名空间指的是微服务命名空间，即 Nacos 语境中的命名空间，并非 Kubernetes 场景下的命名空间。

!!! note

– 创建托管型注册中心实例时，系统会自动创建一个名为 **public** 的默认命名空间。该命名空间不可编辑、不可删除，属于系统自带的保留命名空间。
– 不同命名空间下的服务和配置严格隔离，不能互相引用。例如，A 命名空间下的服务不能引用 B 命名空间下的配置。

创建微服务命名空间

1. 进入微服务引擎 –> 微服务治理中心 –> 托管注册中心模块，点击目标注册中心的名称。



命名空间

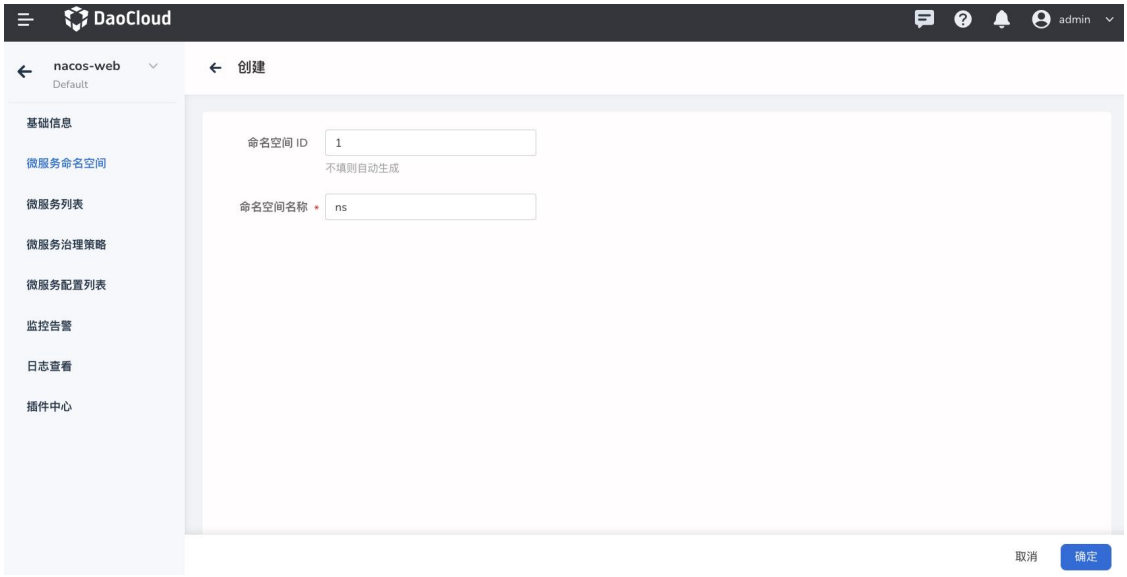
2. 在左侧导航栏点击微服务命名空间，然后在右上角点击创建。



创建命名空间

3. 填写命名空间的 ID，名称和描述信息，在页面右下角点击确定。

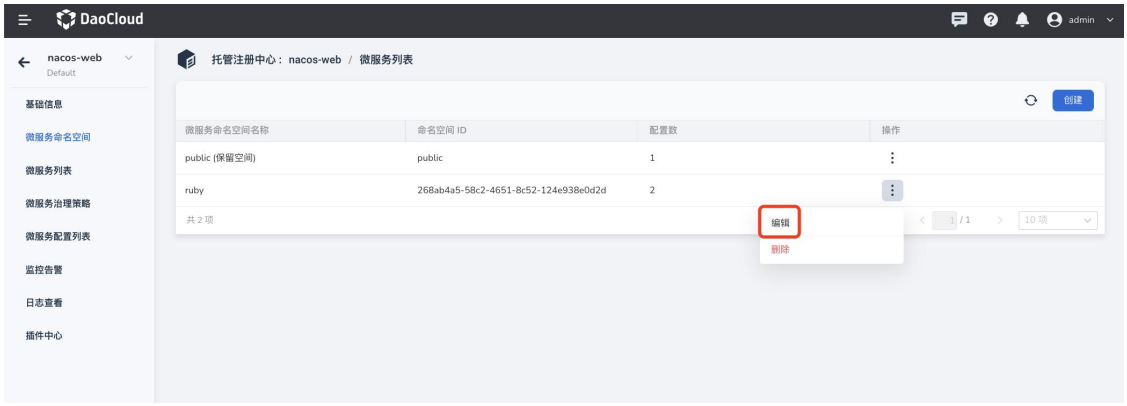
   如果不填写 ID，系统会自动生成一个 ID。**命名空间 ID 在创建之后不可更改。**



创建命名空间

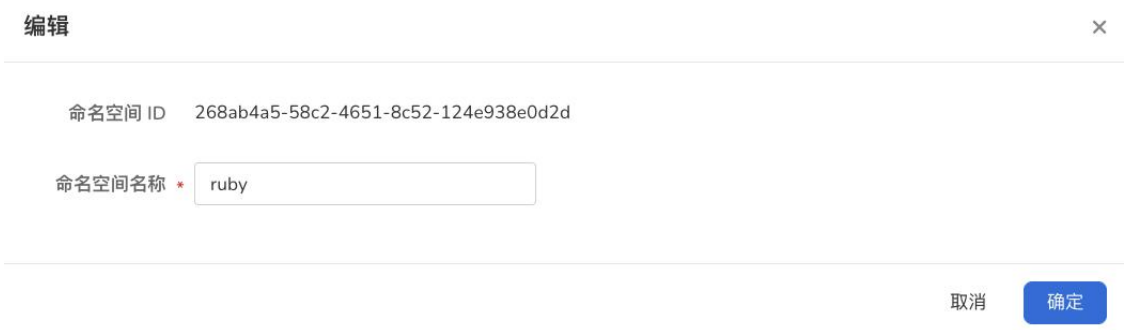
更新微服务命名空间

1. 在对应命名空间的右侧操作栏下点击编辑，进入更新页面。



更新命名空间

2. 修改命名空间的名称、描述信息，点击确定。



更新命名空间

删除微服务命名空间

在对应命名空间的右侧操作栏下点击删除，然后在弹框中点击立即删除。

!!! note

删除命名空间之前，必须清理该命名空间下的所有资源，否则无法删除命名空间。


DaoCloud
admin
nacos-web
Default
基础信息
微服务命名空间
微服务列表
微服务治理策略
微服务配置列表
监控告警
日志查看
插件中心
托管注册中心： nacos-web / 微服务列表
创建
微服务命名空间名称
命名空间 ID
配置数
操作
public (保留空间)
public
1
ruby
268ab4a5-58c2-4651-8c52-124e938e0d2d
2
共 2 项
1 / 1
10 项
编辑
删除

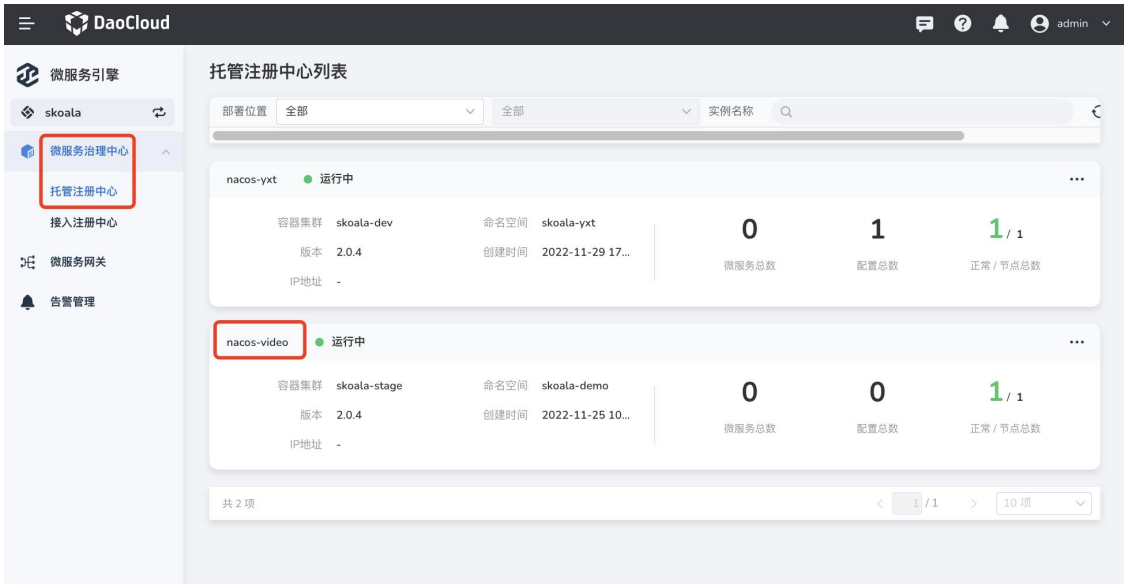
删除命名空间

微服务配置列表

微服务引擎支持基于微服务命名空间对微服务进行分组管理，以便灵活地根据自己的需要按照环境或者应用、模块等分组管理微服务以及 Spring 的大量配置。配置列表主要提供了微服务配置的增删改查、查看历史版本、回滚、订阅者查询等核心管理能力。

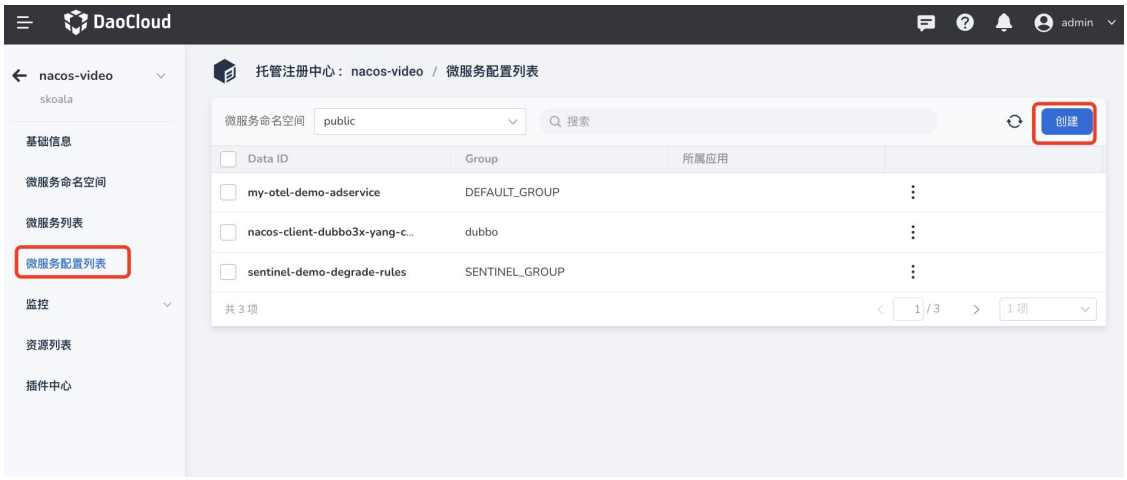
新建配置

1. 进入微服务引擎 –> 微服务治理中心 –> 托管注册中心模块，点击目标注册中心的名称。



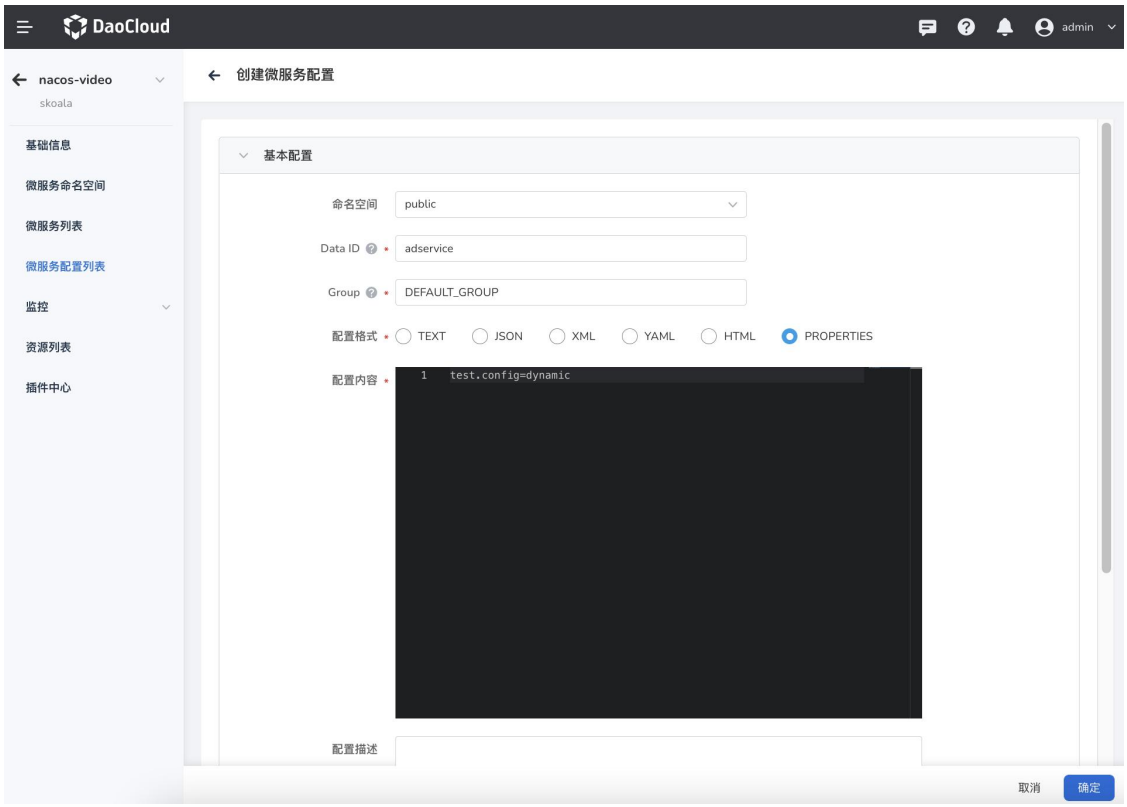
点击注册中心名称

2. 在左侧导航栏点击微服务配置列表，然后在右上角点击创建。



创建第一步

3. 填写各项配置信息



填写配置

- 命名空间：选择当前配置所属的微服务命名空间,默认为 public

不同命名空间下的服务和配置严格隔离，不能互相引用。例如，A 命名空间下的服务不能引用 B 命名空间下的配置。

- Data ID：相当于当前配置文件的名称。同一分组下的 Data ID 不能重复。

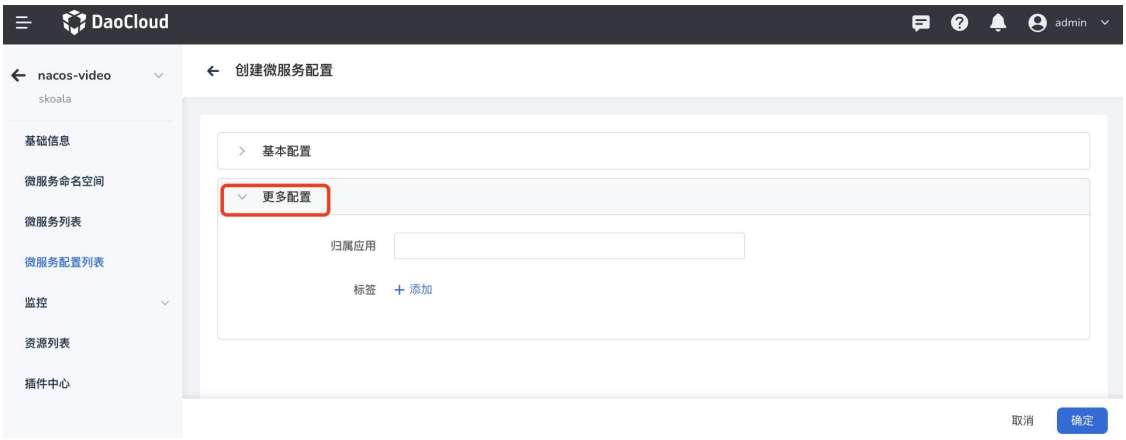
  完整格式为：${prefix}-${spring.profiles.active}.${file-extension}。

  - prefix 默认为 spring.application.name 的值，也可以通过配置项 spring.cloud.nacos.config.prefix 来配置。
  - spring.profiles.active 即为当前环境对应的 profile，详情可以参考 Spring Boot 文档。 注意：当 spring.profiles.active 为空时，对应的连接符 - 也将不存在，Data ID 的拼接格式变成 ${prefix}.${file-extension}。
  - file-exetension 为配置内容的数据格式，可以通过配置项 spring.cloud.nacos.config.file-extension 来配置。
- Group：选择当前配置所属的分组，默认为 DEFAULT_GROUP。

通常用命名空间隔离环境，用分组区分项目。

- 配置格式: 设置当前配置文件的格式，
- 配置内容：输入服务源代码中具有 @Value 注解的配置项。

  - 支持格式校验，如果配置内容存在格式错误，系统会自动提示错误。
  - 服务源代码中具有 @RefreshScope 注解的配置项支持动态更新。
- 更多配置->归属应用（选填）：选择当前配置所属的应用。

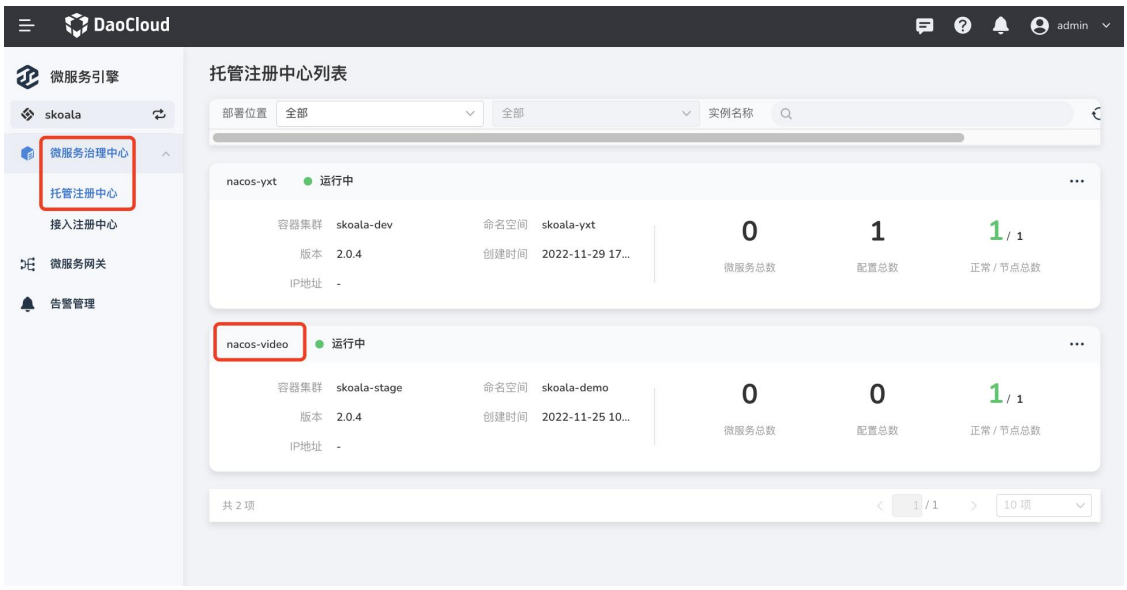


填写配置

4. 在页面右下角点击确定。

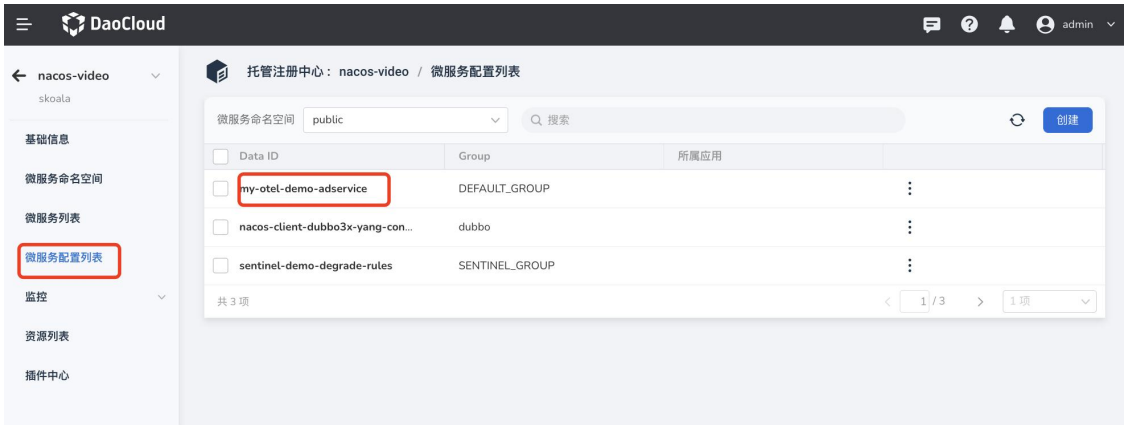
查看配置

1. 进入微服务引擎 –> 微服务治理中心 –> 托管注册中心模块，点击目标注册中心的名称。



点击注册中心名称

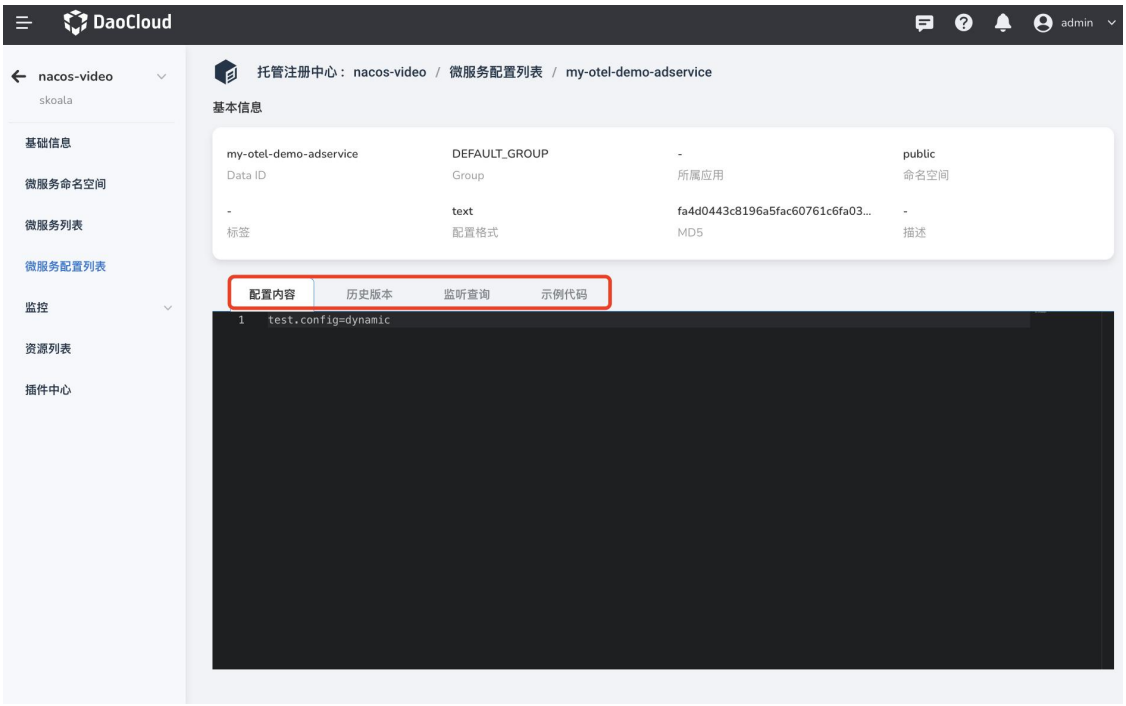
2. 在左侧导航栏点击微服务配置列表，点击目标配置的 Data ID。



点击名称

3. 查看该配置的基本信息、配置内容、历史版本、监听者、示例代码等信息。

    - 新手可以借助示例代码快速使用客户端编程消费该配置，降低新手使用门槛。
    - 支持查询当前配置的监听者以及 MD5 校验值，便于了解正在使用此配置的微服务，以及配置变更是否成功推送到了客户端。

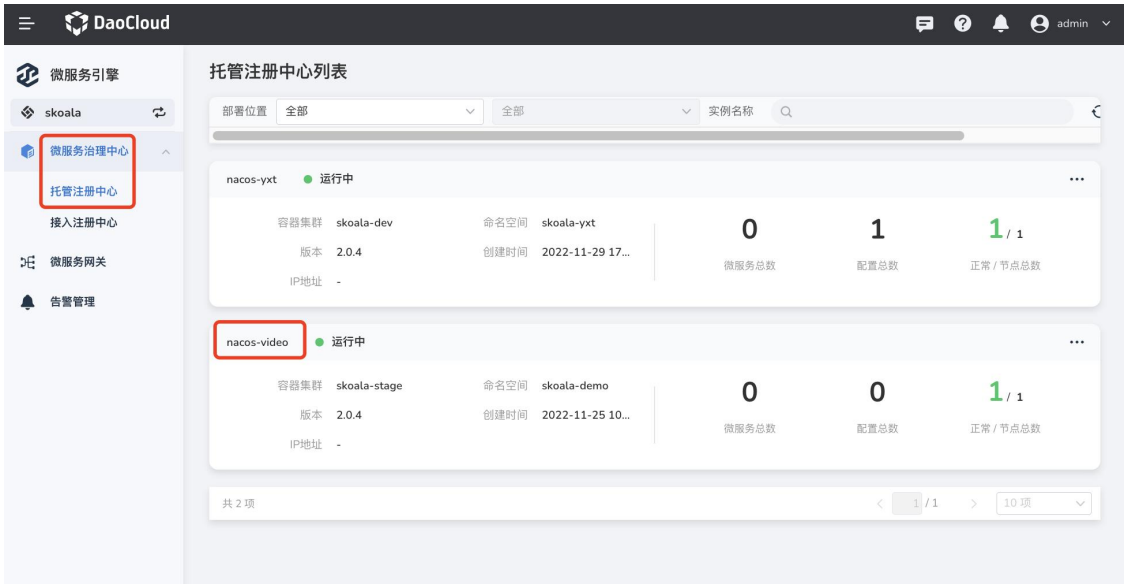


查看

历史版本回滚

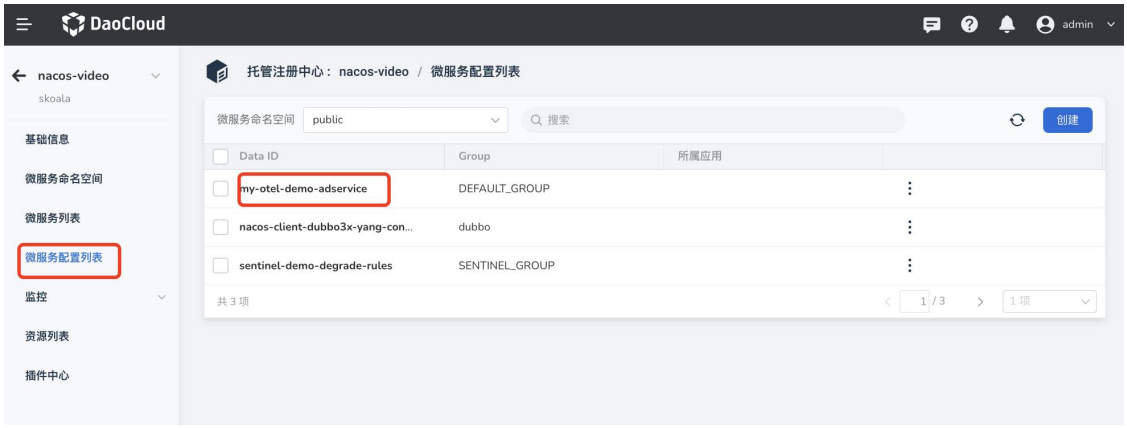
微服务配置列表会记录配置文件的历史版本，支持一键回滚到特定的历史版本，便于用户在改错配置时快速恢复，降低微服务系统中的配置可用性风险。回滚时支持版本差异对比，便于对比当前版本和目标回滚版本的差异，校验修改内容，降低改错带来的风险。

1. 进入微服务引擎 –> 微服务治理中心 –> 托管注册中心模块，点击目标注册中心的名称。



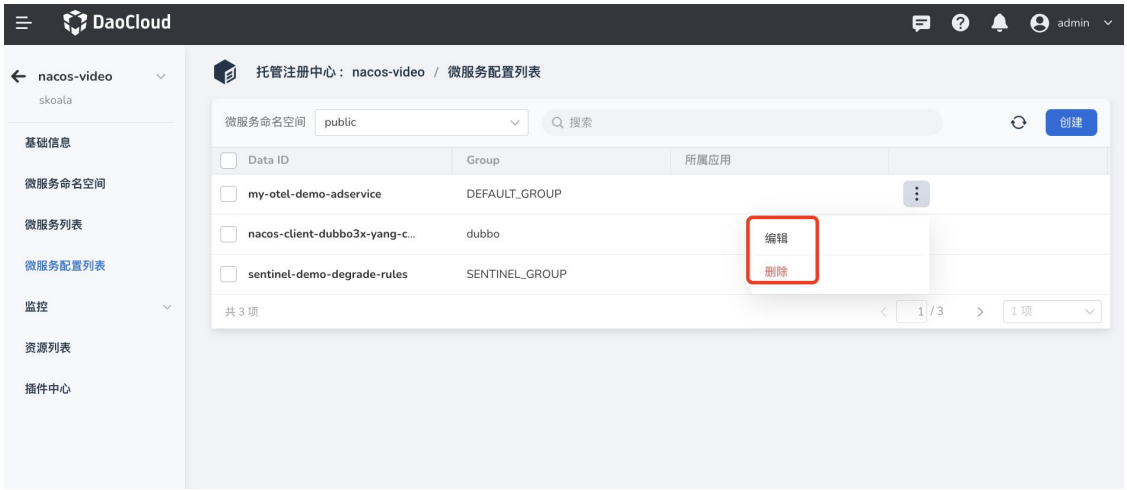
点击注册中心名称

2. 在左侧导航栏点击微服务配置列表，点击目标配置的 Data ID。



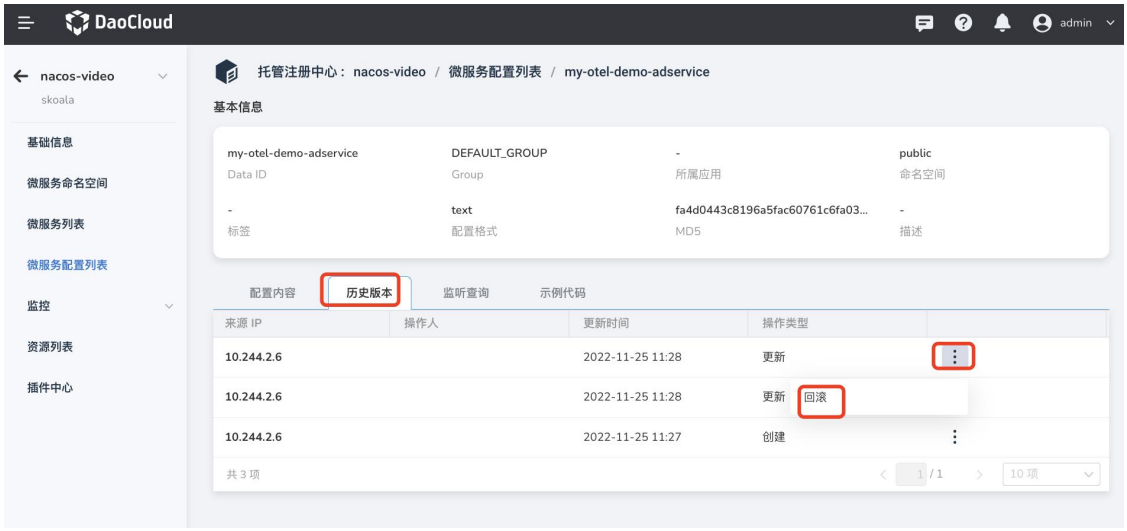
点击名称

3. 点击历史版本页签，找到对应的记录，在记录右侧点击　　并选择回滚。



点击名称

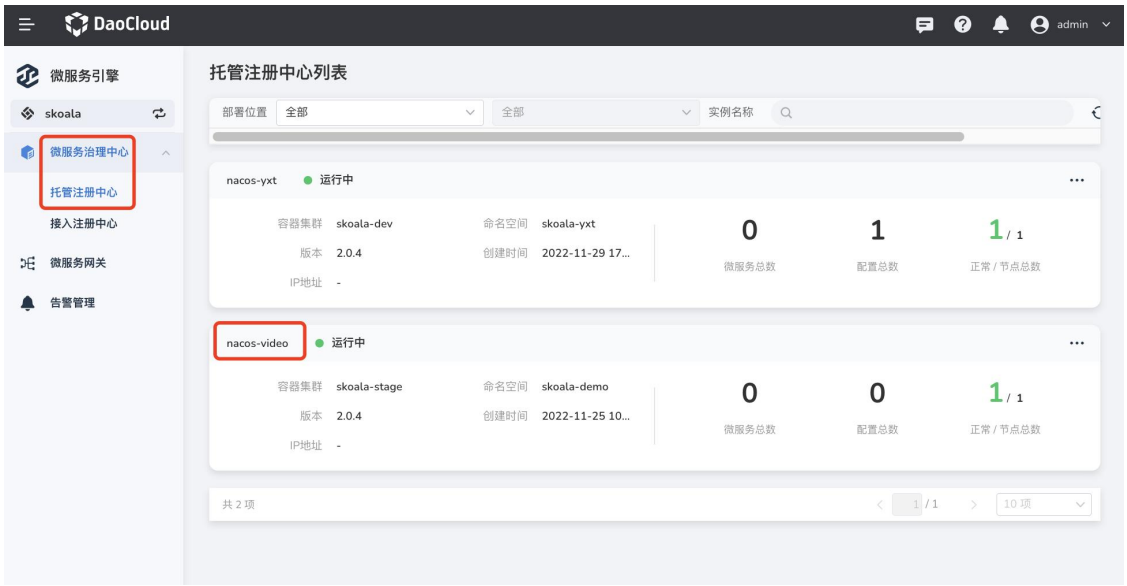
4. 对比版本差异并确认，然后在右下角点击回滚。



回滚

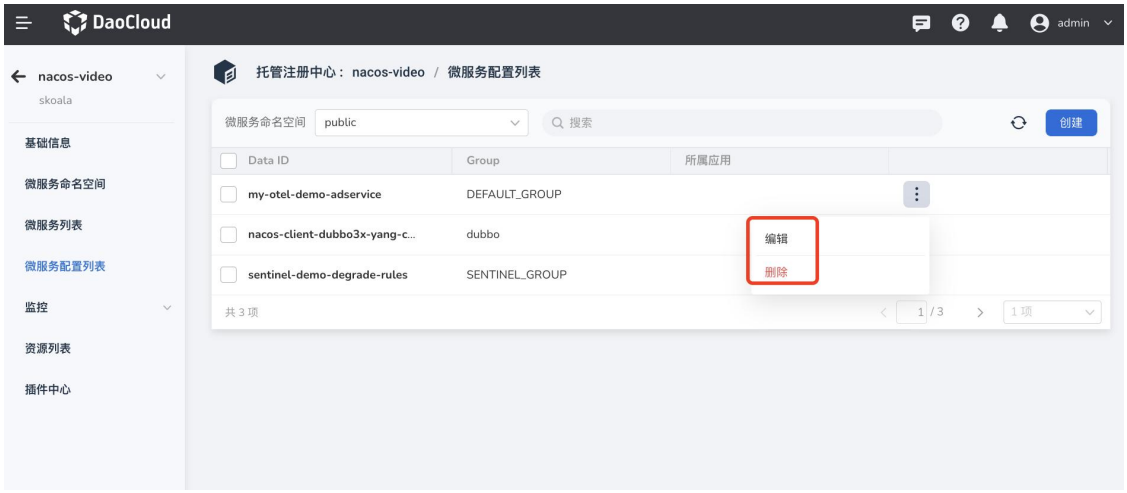
更新/删除配置

1. 进入微服务引擎 –> 微服务治理中心 –> 托管注册中心模块，点击目标注册中心的名称。



点击注册中心名称

2. 在左侧导航栏点击微服务配置列表，在目标配置右侧点击　　按钮根据需要选择更新或删除。

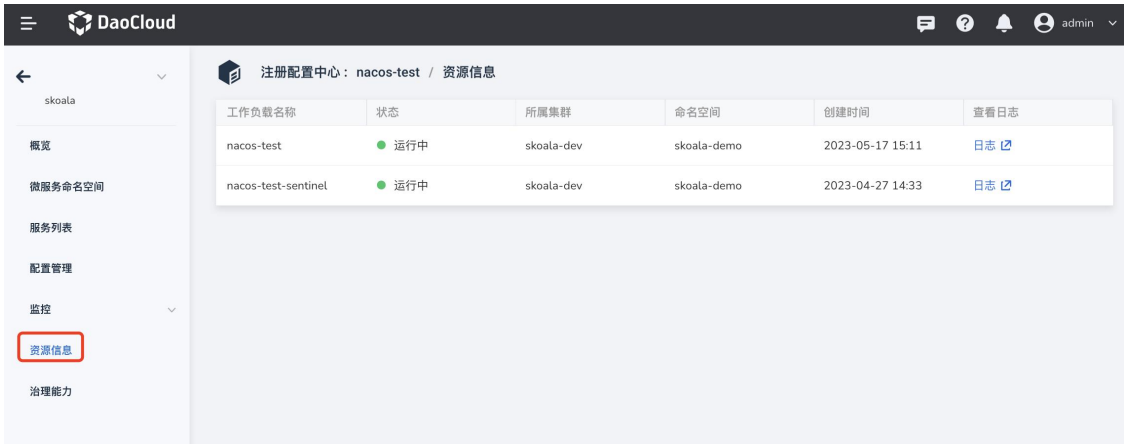

DaoCloud
admin
nacos-video
skoala
基础信息
微服务命名空间
微服务列表
微服务配置列表
监控
资源列表
插件中心
托管注册中心： nacos-video / 微服务配置列表
微服务命名空间
public
搜索
创建
Data ID
Group
所属应用
my-otel-demo-adservice
DEFAULT_GROUP
nacos-client-dubbo3x-yang-c...
dubbo
sentinel-demo-degrade-rules
SENTINEL_GROUP
编辑
删除
共 3 项
1 / 3
1 项

更新/删除

资源信息

微服务引擎支持查看托管注册中心自身实例的运行状况、部署位置及运行日志。

- 与当前注册配置中心同名的工作负载就是该注册配置中心自身。
- 如果启用了 Mesh 或 Sentinel 治理能力，则会出现另外一个实例，代表对应的服务网格实例或 Sentinel 实例。



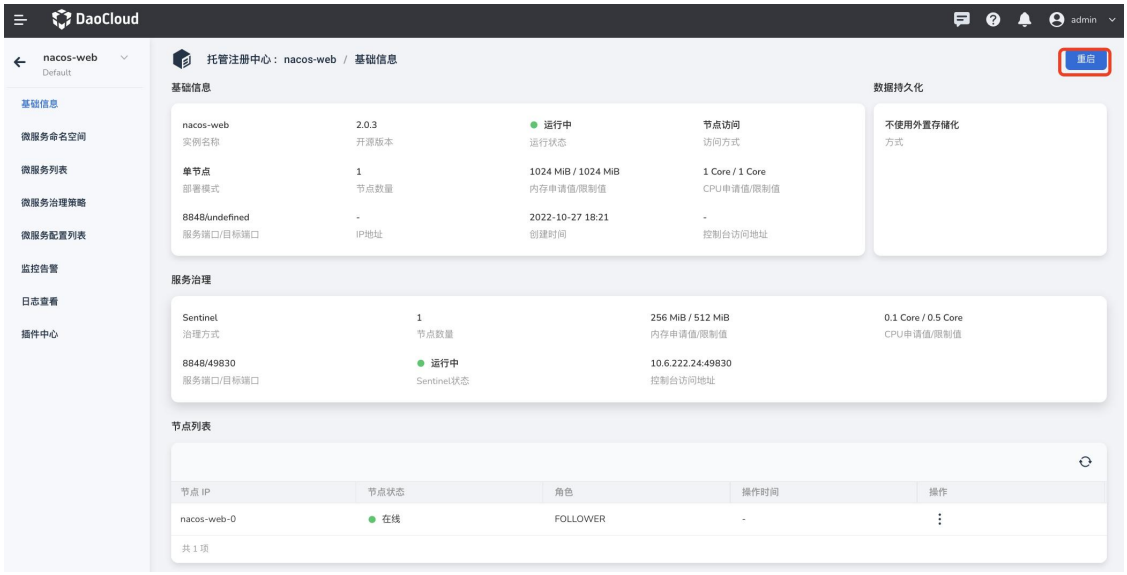
查看注册中心详情

在托管注册中心列表页找到需要查看详情的注册中心实例，点击实例名称进入基础信息页面。

在基础信息页面可以查看注册中心的基础信息、服务治理信息（需要开启服务治理功能）、节点列表、数据持久化信息等。“运行状态”是注册中心实例信息的一部分，用于反映注册中心实例的状态。

相关操作：

- 重启注册中心实例：在页面右上角点击重启可以对整个托管注册中心实例进行重启。



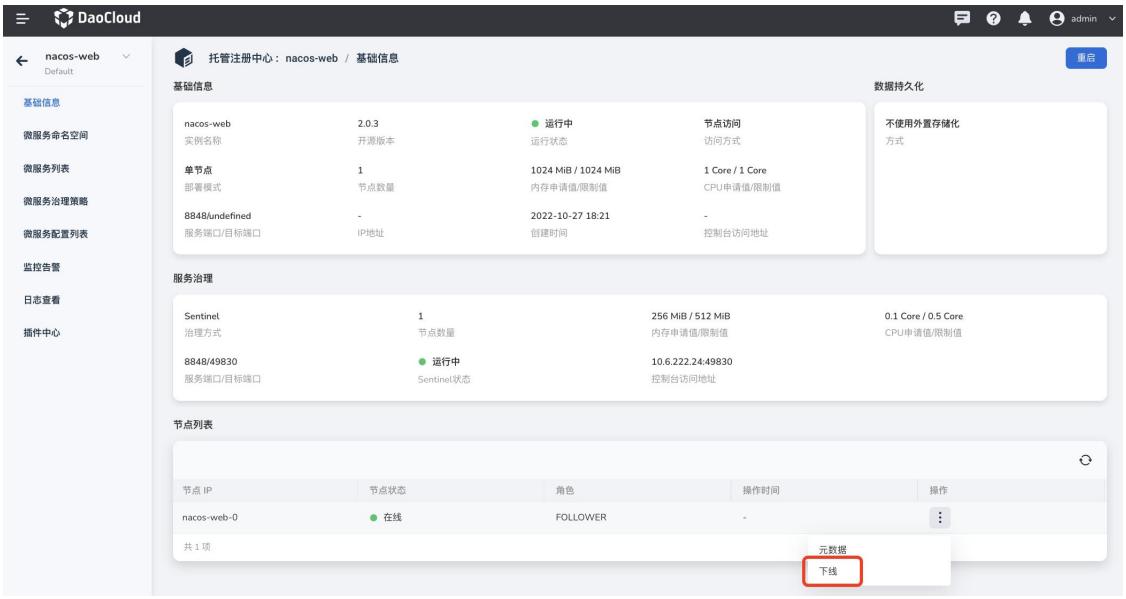
全量重启

- 节点上下线：在节点列表中选择目标节点，修改节点状态。
    - 下线

      点击操作列的下线，修改节点实例状态为“下线”

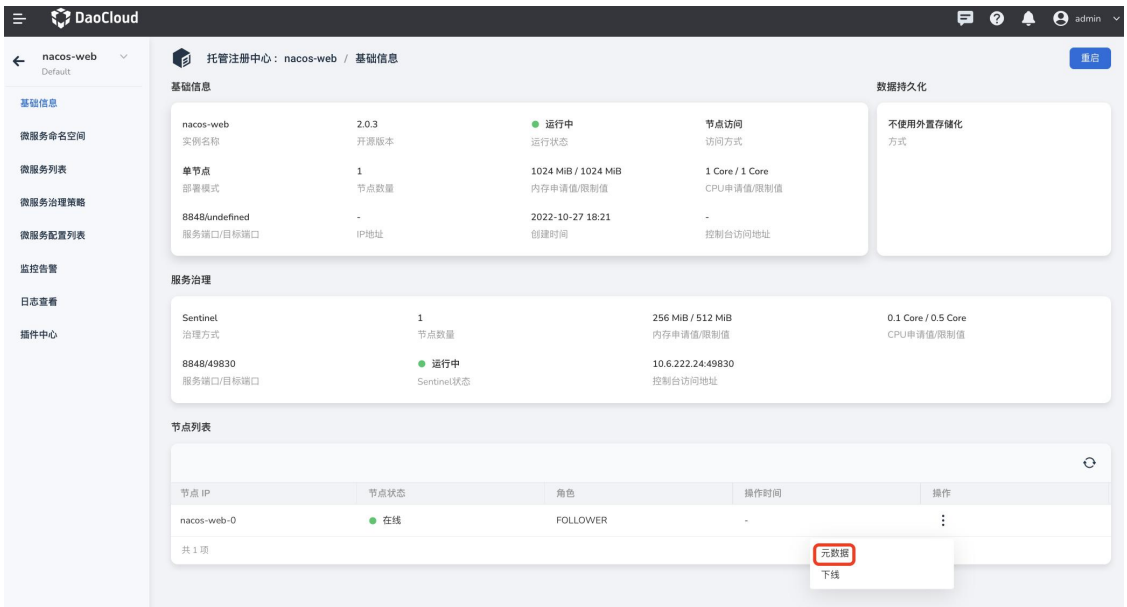
– 上线

点击操作列的上线，修改节点实例状态为“上线”



单节点重启

- 查看节点元数据



查看元数据

组件监控

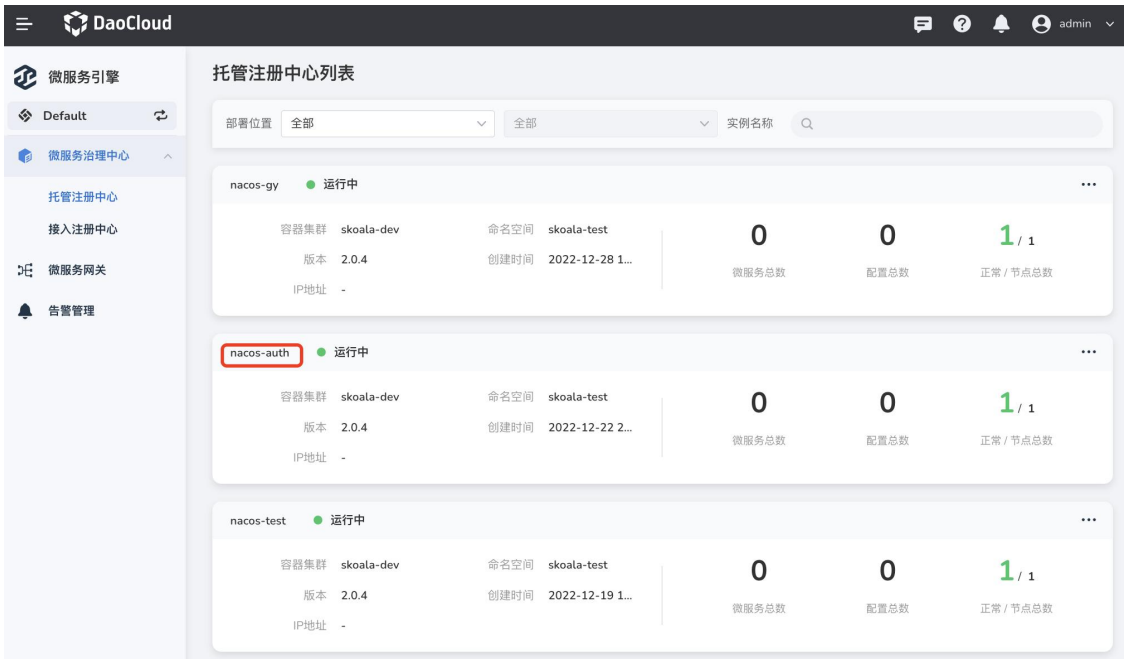
!!! note

集群中需要[安装 insight-agent 组件](../../../../insight/quickstart/install/install-agent.md)后才能使用监控功能。

微服务引擎通过内置 Grafana 提供全方位的监控功能，覆盖各个微服务、系统组件、以及服务调用链路等监控对象。其中，组件监控功能可以提供对 Nacos 和 Sentinel 的监控信息，包括节点数、服务数、CPU/内存用量、JVM 线程数、http 请求总耗时等多个维度的信息。

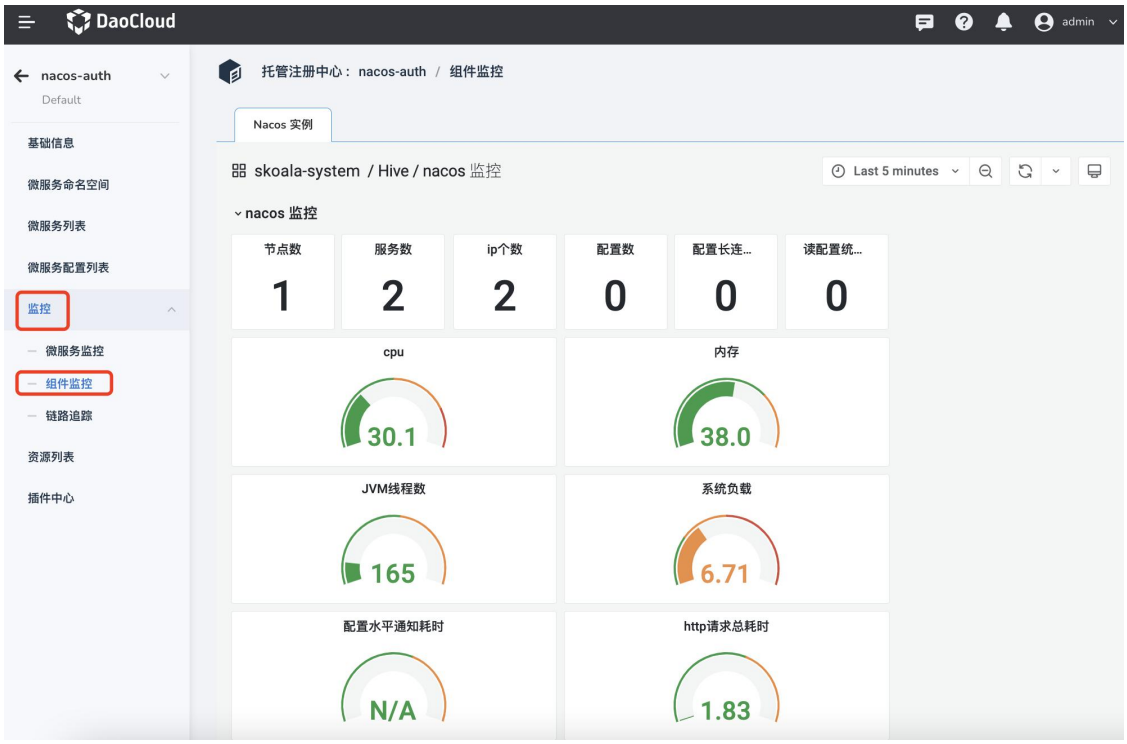
查看组件监控的步骤如下：

1. 进入微服务引擎模块，点击目标注册中心的名称。



点击名称

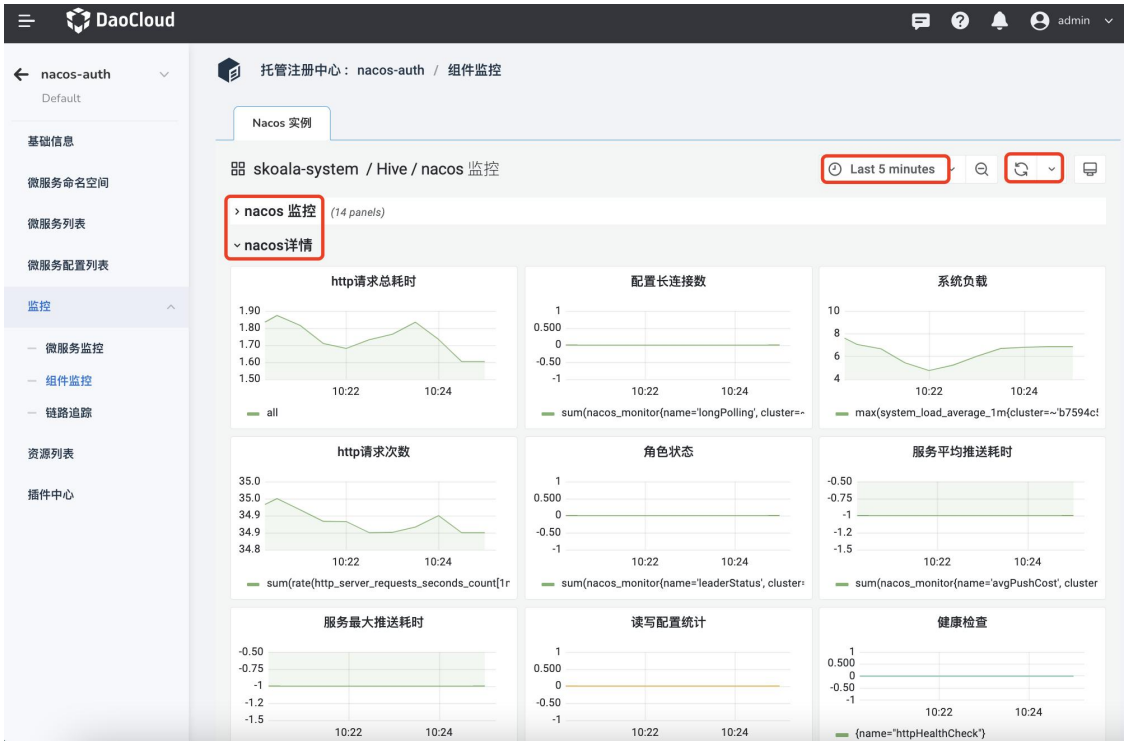
2. 在左侧导航栏点击监控->组件监控。



导航栏

3. 点击 Naocs 实例页签即可查看 Nacos 实例的监控信息。

   下拉页面可查看请求耗时、请求次数等统计详情。在页面右上角可切换统计时间窗口和刷新周期。


DaoCloud
admin
nacos-auth
Default
基础信息
微服务命名空间
微服务列表
微服务配置列表
监控
微服务监控
组件监控
链路追踪
资源列表
插件中心
托管注册中心： nacos-auth / 组件监控
Nacos 实例
skoala-system / Hive / nacos 监控
Last 5 minutes
nacos 监控
(14 panels)
nacos详情
http请求总耗时
配置长连接数
系统负载
http请求次数
角色状态
服务平均推送耗时
服务最大推送耗时
读写配置统计
健康检查
all
10:22
10:24

查看信息

链路追踪

DCE 5.0 支持服务级别的链路追踪与查询，查看各个微服务的链路时延分布情况，在出现故障时可以快速定位故障。

!!! note

集群中需要[安装 insight-agent 组件](../../../../insight/quickstart/install/install-agent.md)后才能使用监控功能。

更多详情，可参考可观测性->链路查询。

微服务监控

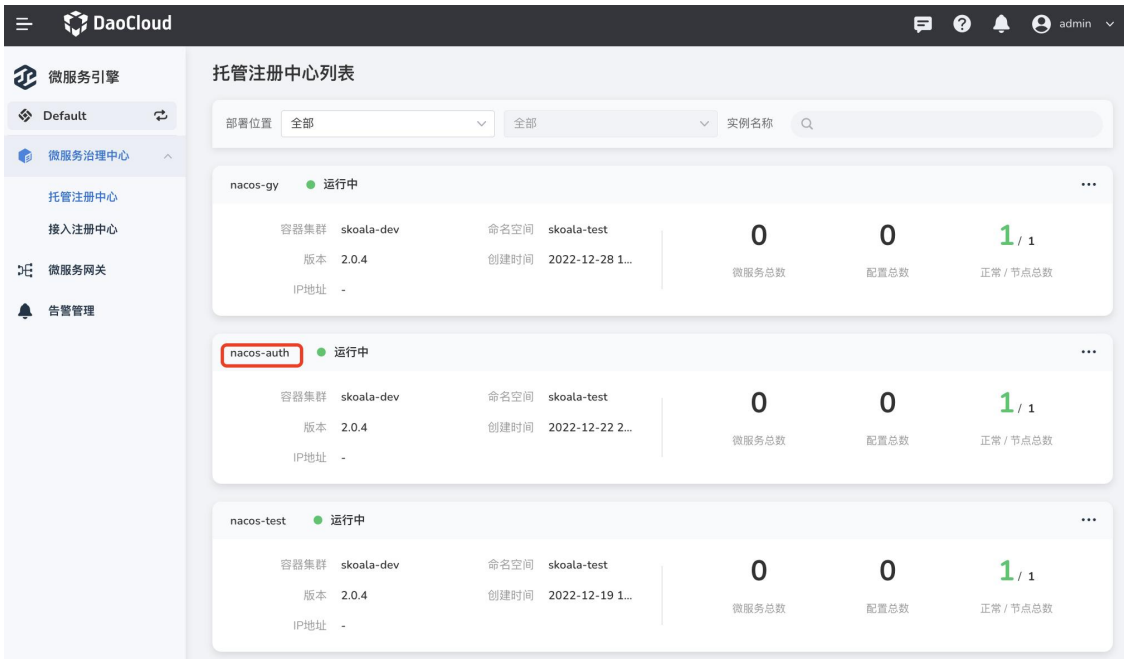
微服务引擎通过内置 Grafana 提供全方位的监控功能，覆盖各个微服务、系统组件、以及服务调用链路等监控对象。其中，微服务监控功能可以提供对托管注册中心下各个微服务的监控信息，包括服务响应时间、并发量、异常 QPS、阻塞 QPS、通过 QPS、成功 QPS 等多个维度的信息。

!!! note

集群中需要[安装 insight-agent 组件](../../../../insight/quickstart/install/install-agent.md)后才能使用监控功能。

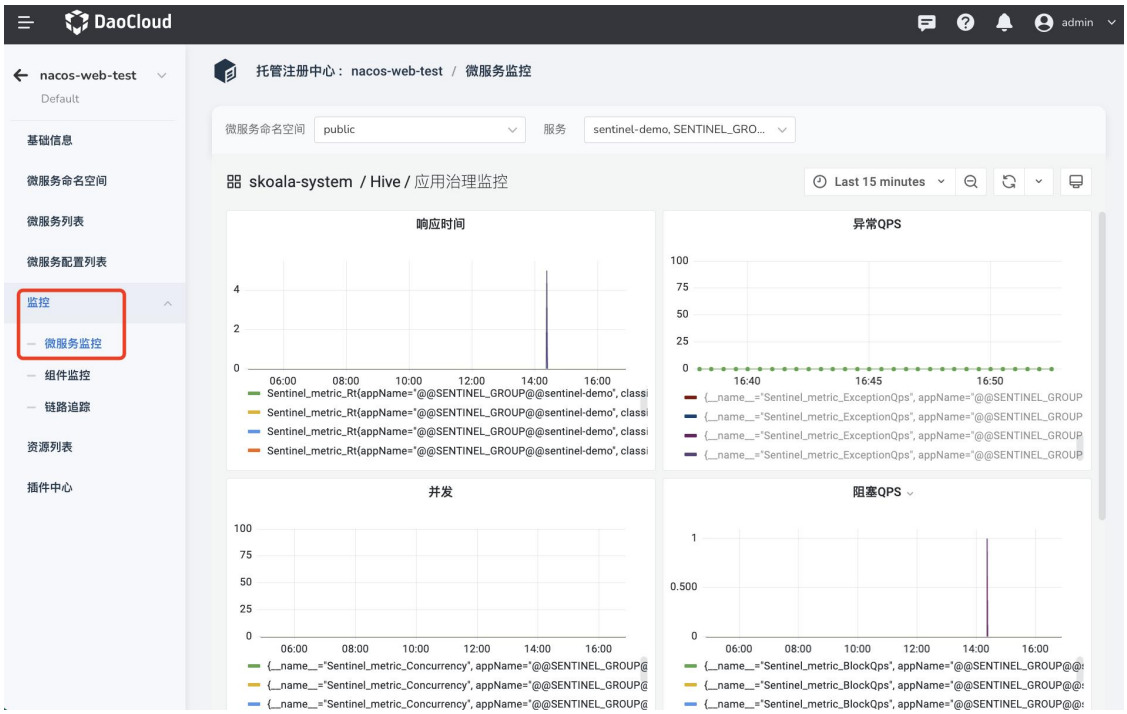
查看微服务监控的步骤如下：

1. 进入微服务引擎模块，点击目标注册中心的名称。



点击名称

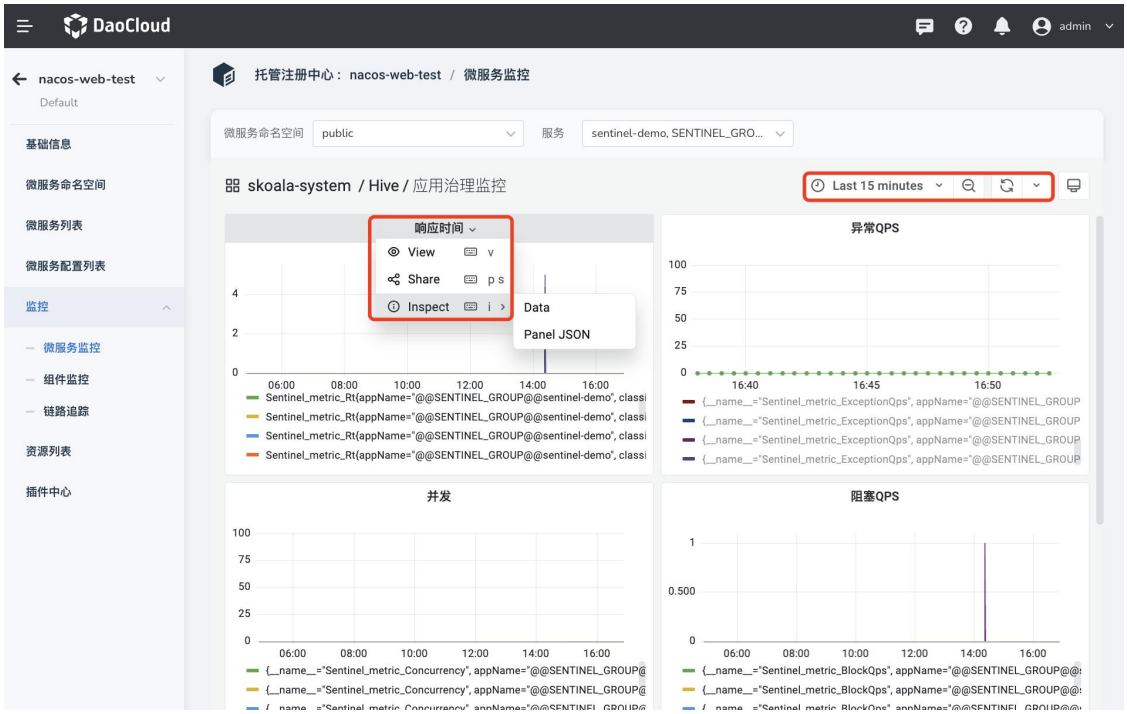
2. 在左侧导航栏点击监控->微服务监控。



导航栏

3. 选择对应的微服务命名空间即可查看该命名空间下的微服务监控数据。

   点击仪表盘名称可以进一步查看或分享该监控信息。在页面右上角可切换统计时间窗口和刷新周期。


DaoCloud
admin
nacos-web-test
Default
基础信息
微服务命名空间
微服务列表
微服务配置列表
监控
微服务监控
组件监控
链路追踪
资源列表
插件中心
托管注册中心：nacos-web-test / 微服务监控
微服务命名空间
public
服务
sentinel-demo, SENTINEL_GRO...
skoala-system / Hive / 应用治理监控
Last 15 minutes
响应时间
View v
Share p s
Inspect i
Data
Panel JSON
异常QPS
并发
阻塞QPS

查看信息

启用 Sentinel 治理插件

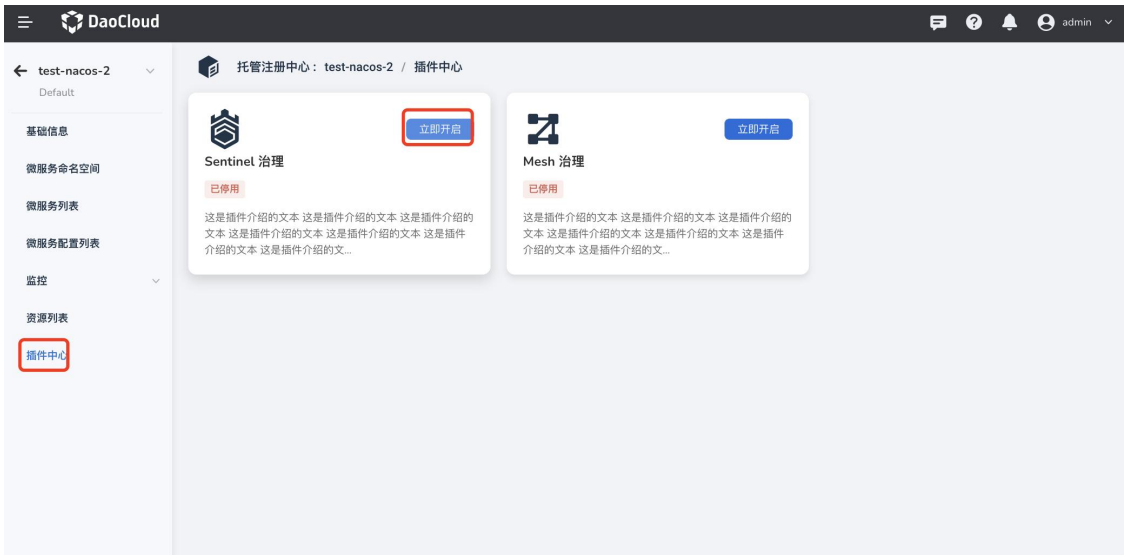
Sentinel 是面向分布式、多语言异构化服务架构的流量治理组件，主要以流量为切入点，从流量路由、流量控制、流量整形、熔断降级、系统自适应过载保护、热点流量防护等多个维度来帮助开发者保障微服务的稳定性。

操作步骤

1. 在托管注册中心列表页面点击目标注册中心的名称，然后。

   进入插件中心

2. 在左侧导航栏点击插件中心，在 Sentinel 治理卡片上点击立即开启。



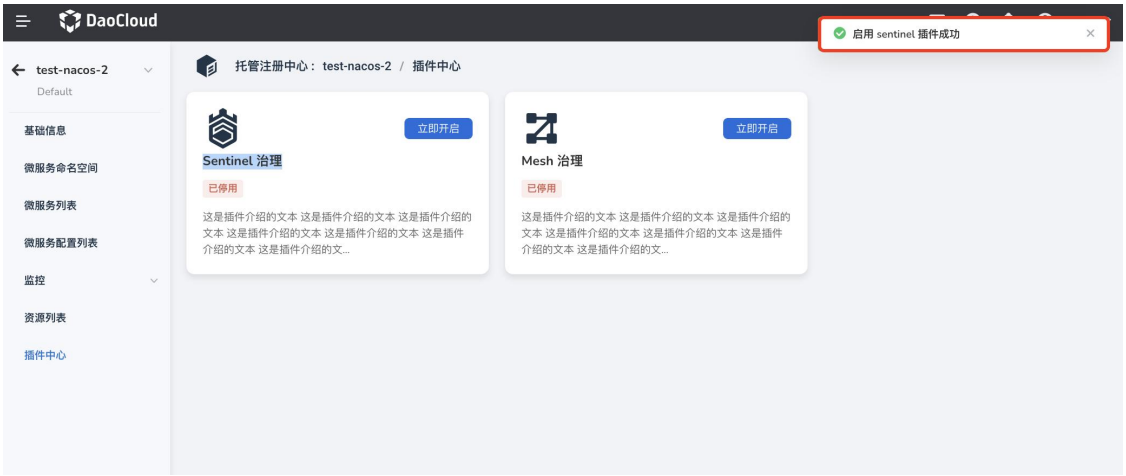
开启插件

3. 填写各项配置信息，然后在弹框底部点击确定。



配置

4. 如果满足前提条件并且配置正确，页面右上角会弹出”启用 Sentinel 插件成功”的消息。



配置

插件中心

插件中心提供 Sentinel 治理和 Mesh 治理两种插件，支持通过用户界面实现可视化配置。安装插件后可以扩展微服务治理能力，满足不同场景下的业务诉求。

!!! info

同一个注册中心实例不能同时开启这两种插件，但可以根据不同的场景进行切换。



插件中心

Sentinel 治理

Sentinel 插件主要适用于传统微服务的治理场景，支持流控规则、熔断规则、热点规则、系统规则、授权规则等多种治理规则。

开启 Sentinel 治理插件后会创建一个 Sentinel 实例，需要为其设置资源配额、选择部署模式（单节点/高可用）和访问方式。

Mesh 治理

Mesh 插件主要适用于云原生微服务的治理场景，提供虚拟服务、目标规则、网关规则、对等认证、请求身份认证、授权策略等治理规则。

开启 Mesh 治理插件需绑定一个网格实例，将微服务加入到网格中，并根据服务网格的要求配置边车等资源。

启用 Mesh 治理插件

操作步骤

1. 在托管注册中心列表页面点击目标注册中心的名称，然后。

   进入插件中心

2. 在左侧导航栏点击插件中心，在 Mesh 治理卡片上点击立即开启。



开启插件

3. 选择想要绑定的服务网格，然后在弹框底部点击确定。

   如果找不到想要的服务网格，可以去服务网格模块创建一个网格。



配置

4. 如果满足前提条件并且配置正确，页面右上角会弹出”启用 Mesh 插件成功”的消息。


DaoCloud
启用 mesh 插件成功
test-nacos-2
Default
基础信息
微服务命名空间
微服务列表
微服务配置列表
监控
资源列表
插件中心
托管注册中心： test-nacos-2 / 插件中心
立即开启
Sentinel 治理
已停用
这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文...
立即开启
Mesh 治理
已停用
这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文本 这是插件介绍的文...

配置

创建熔断规则

熔断规则可以监控应用内部或者下游服务的响应时间或异常比例。当达到指定的阈值时降低下游依赖的优先级，并且在指定的时间范围内不会调用优先级低的不稳定资源，避免应用受到影响，从而保障应用高可用性。一个熔断时长结束后，熔断器进入探测恢复状态：如果接下来的一个请求的响应时间小于所设定的慢调用最大响应时间，则结束熔断，否则该资源会被再次被熔断

创建熔断规则的方式如下：

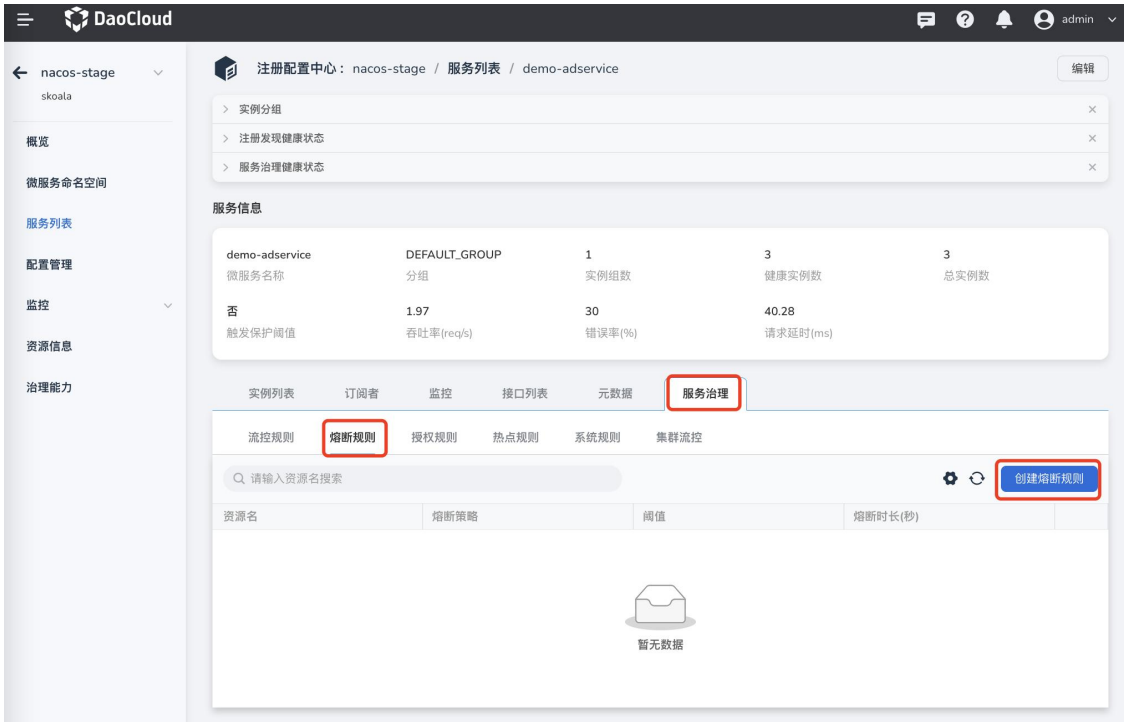
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

   注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。



   微服务列表

2. 选择熔断规则，然后在右侧点击创建熔断规则。



微服务列表

3. 参考下列说明填写规则配置，并在右下角点击确定。

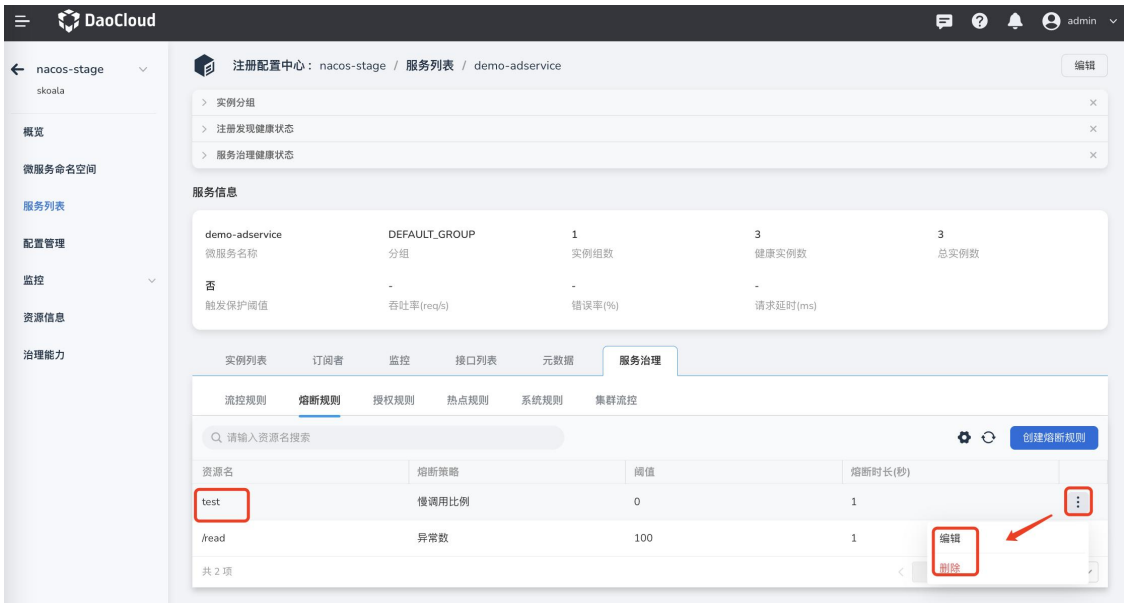
   - 资源名：需要控制哪个资源的流量就填写哪个资源的名称，例如当前服务下的某个 API 接口、函数、变量等。
   - 慢调用比例：如果在一个单位统计时长内，如果慢调用比例大于所设定的比例阈值，则在该熔断时长内的剩余时间里，对资源进行熔断。
   - 异常比例：在单位统计时长内，如果异常请求的比例大于阈值，则在剩余时间内对资源进行熔断。
   - 异常数：在单位统计时长内，如果收到的请求数达到所设定的阈值，则在剩余时间内对资源进行熔断。
   - 最大 RT：请求的最大响应时间，单位为毫秒，**响应时间超过此值的请求被视为慢调用请求**。

- 比例阈值：在一个统计时长内，慢调用请求（适用于慢调用比例策略）/异常请求（适用于异常比例策略）在所有请求中占的比例。阈值范围是 [0.0, 1.0]，代表 0% – 100%。
- 熔断时长：熔断持续的时间，单位为秒。在熔断时长内，所有的请求都会快速失败。
- 最小请求数：在一个统计时长内，触发熔断的最小请求数。**若当前统计时长内的请求数小于此值，即使达到熔断条件也不会触发熔断**。
- 统计时长：统计的时间窗口长度，单位为毫秒，取值范围为 1 秒 —— 120 分钟。



微服务列表

4. 创建完成后可以在熔断规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。


DaoCloud
admin
nacos-stage
skoala
概览
微服务命名空间
服务列表
配置管理
监控
资源信息
治理能力
注册配置中心： nacos-stage / 服务列表 / demo-adservice
编辑
实例分组
注册发现健康状态
服务治理健康状态
服务信息
demo-adservice
微服务名称
DEFAULT_GROUP
分组
1
实例组数
3
健康实例数
3
总实例数
否
触发保护阈值
-
吞吐率(req/s)
-
错误率(%)
-
请求延时(ms)
实例列表
订阅者
监控
接口列表
元数据
服务治理
流控规则
熔断规则
授权规则
热点规则
系统规则
集群流控
请输入资源名搜索
创建熔断规则
资源名
熔断策略
阈值
熔断时长(秒)
test
慢调用比例
0
1
/read
异常数
100
1
编辑
删除
共 2 项

流控规则列表

规则介绍

微服务引擎支持通过服务网格或 Sentinel 治理东西向流量。

Sentinel 治理规则

- 流控规则

  流控规则的原理是监控应用或服务流量的 QPS 指标，当指标达到阈值时根据预先设定的规则对请求流量进行控制，防止应用因短时间内无法处理过多流量而崩溃。使用流控规则后，系统可以在接下来的空闲期间逐渐处理堆积的请求，当指标重新恢复到阈值以下后，恢复正常的流量请求控制。

- 熔断降级

  在分布式系统中，各个服务通常需要调用其他的内部或外部服务才能正常运行，如果被调用的服务不够稳定，那么级联效应会导致调用者自身的响应时间也变长，产生线程堆积甚至导致服务不可用。为了避免这种情况出现，需要通过熔断机制根据预设的规则切断不稳定的调用链路，或者对下游服务进行降级，保护系统的整体可用性。

- 热点规则

  热点指经常被访问的数据。设置热点规则时，需要配置热点参数（即需要统计访问量的目标参数），然后系统会统计对该热点参数的请求量，当达到一定的阈值后，包含该热点参数的资源就是被限制调用。热点规则适用于统计被频繁访问的资源，达到一定的阈值后限制对该资源的访问。

- 系统规则

  系统规则是指，由 Sentinel 综合系统容量、CPU 使用率、平均响应时间、入口 QPS 等数据，从整体维度出发自动选择流控规则控制请求流

量。和其他规则不同的是，系统规则针对的是应用级别的入口流量，即仅对进入应用的流量生效，而其他规则通常是针对资源维度进行控制。

- 授权规则

  授权规则允许基于请求来源进行流量治理，例如仅放行白名单中的调用方发起的请求，不放行黑名单中的调用方发起的请求。

流控规则

Mesh 治理规则

流量治理为用户提供了三种资源配置，虚拟服务、目标规则、网关规则。 通过配置相应规则可以实现路由、重定向、熔断、分流等多项流量治理功能。用户可以通过向导或 YAML 形式创建、编辑治理策略。

- 虚拟服务主要用于对请求流量的路由定制规则，并可以对数据流做出分流、重定向、超时返回等处理。
- 目标规则则更关注流量本身的治理，为请求流量提供更强大的负载均衡、连接存活探寻、熔断等功能。
- 网关规则为 Istio 网关提供服务在网关的暴露方式。

有关这三项治理规则的详细说明，可参考文档流量治理。

虚拟服务 目标规则 网关规则

微服务列表

微服务列表页面列出了当前注册中心实例下的所有微服务，可以查看微服务的分组、健康状况、保护阈值、请求状况、链路追踪、治理状况等。在列表中点击微服务名称后，还可以进一步查看微服务实例列表、监控信息、接口列表、元数据等。



微服务列表

分组：指微服务的配置文件分组。将配置文件和和微服务实例互相分离，便于为不同的服务或组件应用相同的配置。

保护阈值：保护阈值是 0 到 1 之间的一个浮点数，表示集群中健康实例占比的最小值。如果实际健康实例的占比小于或等于该阈值时，就会触发阈值保护，即无论实例（Instance）是否健康，都会将这个实例（Instance）返回给客户端。阈值保护主要是为了防止故障实例过多时所有流量全部流入剩余实例，造成流量压力将剩余实例被压垮形成雪崩效应。

链路追踪：通过应用工作台创建服务时，可以选择否接入 OpenTelemetry 链路追踪组件。如果服务是通过其他途经注册到注册中心的，则可以参考使用 OTel SDK 为应用程序暴露指标手动接入该组件。建议接入，以便获取完整的可观测信息。

是否可以治理：判断该微服务是否满足治理条件，例如是否开启了治理插件、是否进行了容器化、是否被网格纳管、是否通过应用工作台创建的等一系列要求。

创建热点规则

热点即经常被访问的数据。热点规则适用于对高频访问的资源进行流量控制，防止短时间内流量过大影响系统稳定性。例如，在以下场景中需要统计某个热点数据中访问频次最高的数据，并对其流量控制。

- 针对一段时间内最频繁购买的商品 ID 进行限制，防止击穿缓存而导致大量请求到数据库的情形。
- 针对一段时间内频繁访问的用户 ID 进行限制，防止恶意刷单。

在 DCE 5.0 中，触发热点规则后的流控效果是直接失败，即超过设定的阈值时，对资源的请求会直接失败，而不是排队等待。

创建热点规则的方式如下：

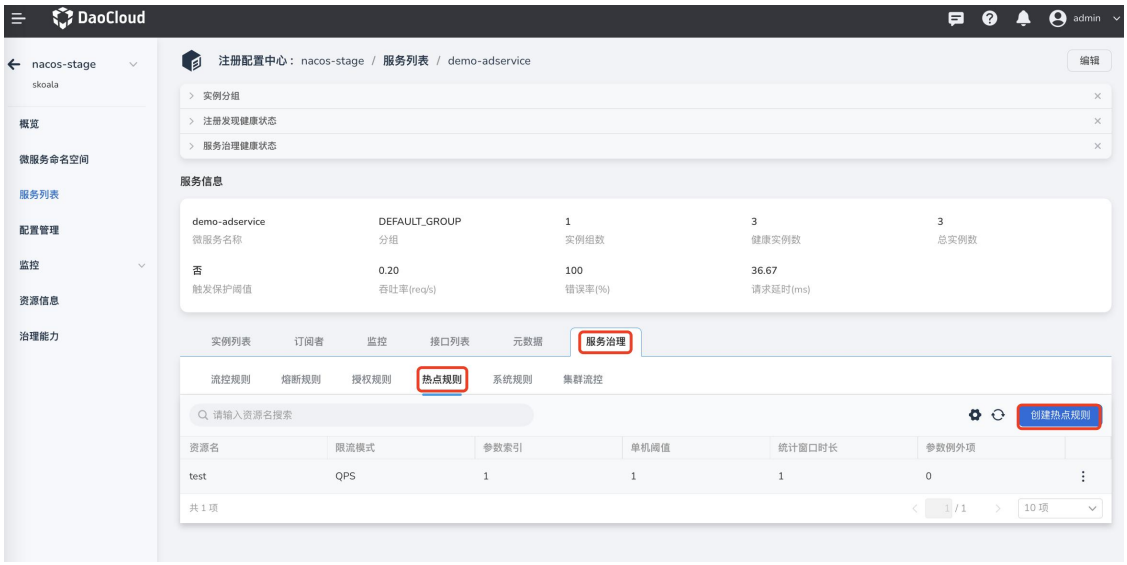
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

   注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。



   微服务列表

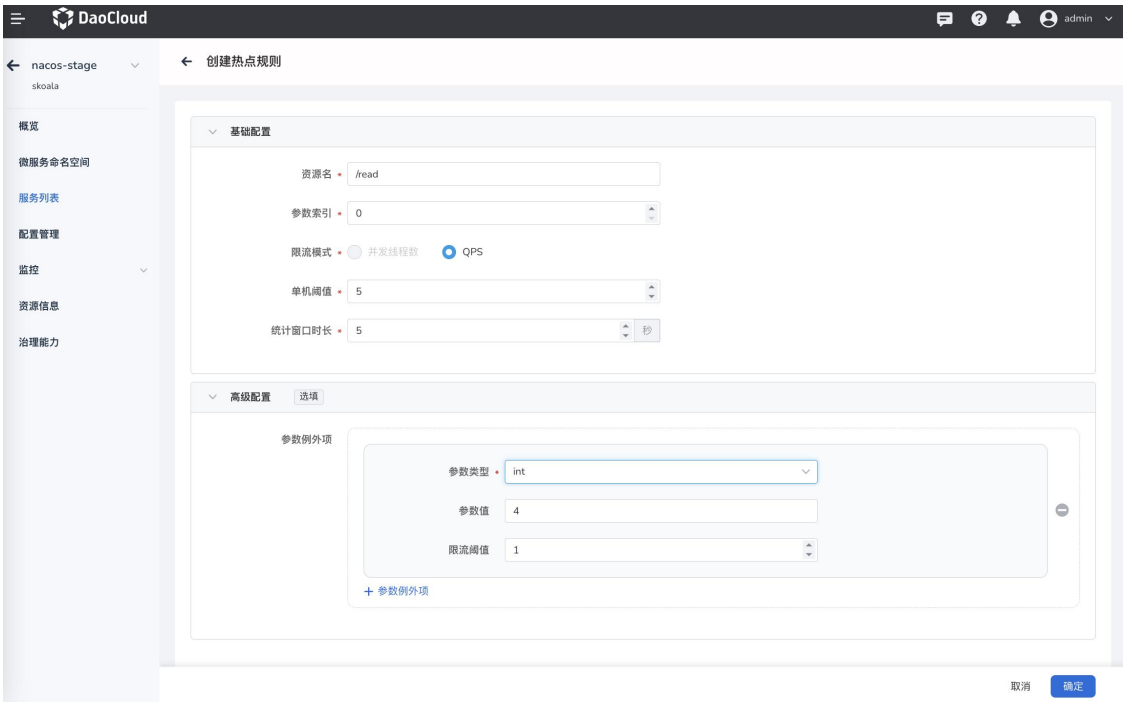
2. 选择热点规则，然后在右侧点击创建热点规则。



微服务列表

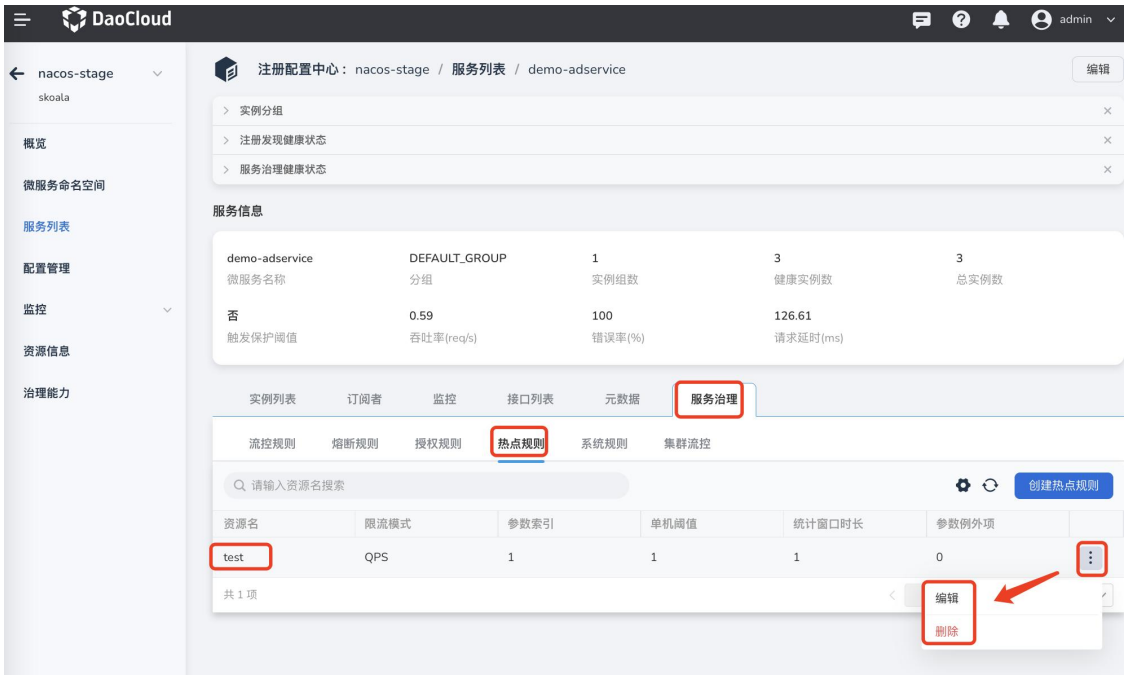
3. 参考下列说明填写规则配置，并在右下角点击确定。

   - 资源名：需要控制哪个资源的流量就填写哪个资源的名称，例如当前服务下的某个 API 接口、函数、变量等。
   - 参数索引：热点参数的索引位置，对应 WithArgs(args ...interface{}) 中的参数索引位置，从 0 开始。
   - 限流模式：选择基于调用资源的线程数或者 QPS 进行流量控制。
   - 单机阈值：作用于每个热点参数的阈值，例如 QPS 超过多少时对热点参数进行流量控制。
   - 统计窗口时长：线程数或者 QPS 的统计周期，例如统计窗口时长为 10s 且 QPS 阈值为 5 时，表示 10s 内对每个热点参数的访问不超过 5 次。
   - 参数例外项：当某个参数值单独设置阈值，不受上方单机限阈值的限制。



微服务列表

4. 创建完成后可以在热点规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。


DaoCloud
admin
nacos-stage
skoala
概览
微服务命名空间
服务列表
配置管理
监控
资源信息
治理能力
注册配置中心： nacos-stage / 服务列表 / demo-adservice
编辑
实例分组
注册发现健康状态
服务治理健康状态
服务信息
demo-adservice
微服务名称
DEFAULT_GROUP
分组
1
实例组数
3
健康实例数
3
总实例数
否
触发保护阈值
0.59
吞吐率(req/s)
100
错误率(%)
126.61
请求延时(ms)
实例列表
订阅者
监控
接口列表
元数据
服务治理
流控规则
熔断规则
授权规则
热点规则
系统规则
集群流控
请输入资源名搜索
创建热点规则
资源名
限流模式
参数索引
单机阈值
统计窗口时长
参数例外项
test
QPS
1
1
1
0
共 1 项
编辑
删除

热点规则列表

创建授权规则

授权规则开业根据资源的请求来源限制资源是否通过。若配置白名单，则只有请求来源位于白名单内时才可通过；若配置黑名单，则请求来源位于黑名单时不通过，其余的请求通过。

创建流控规则的方式如下：

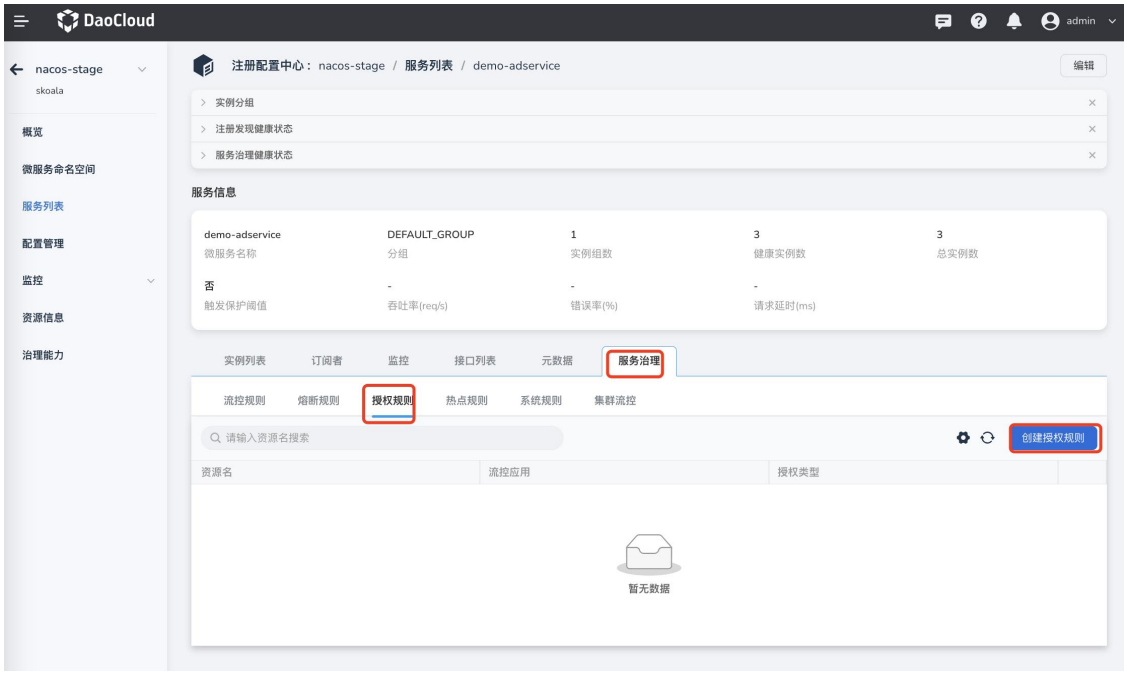
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

   注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。



   微服务列表

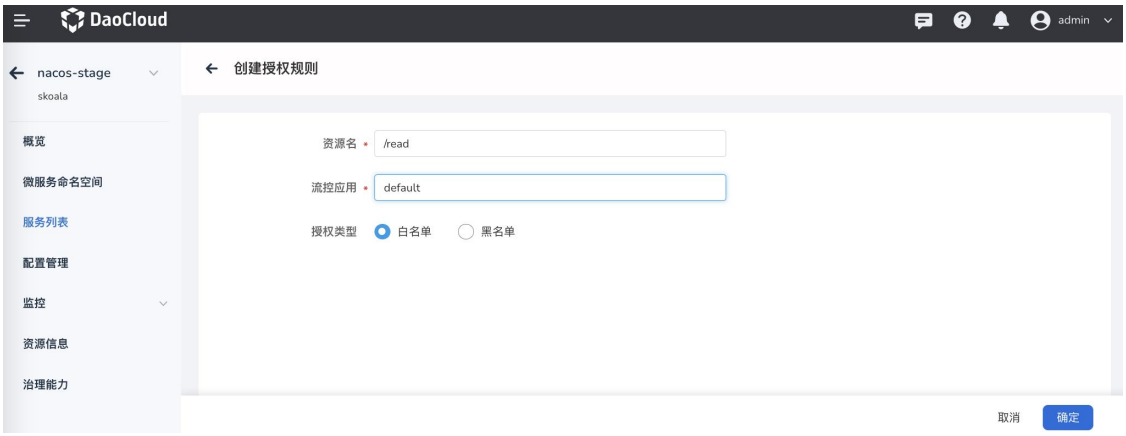
2. 选择授权规则，然后在右侧点击创建授权规则。



微服务列表

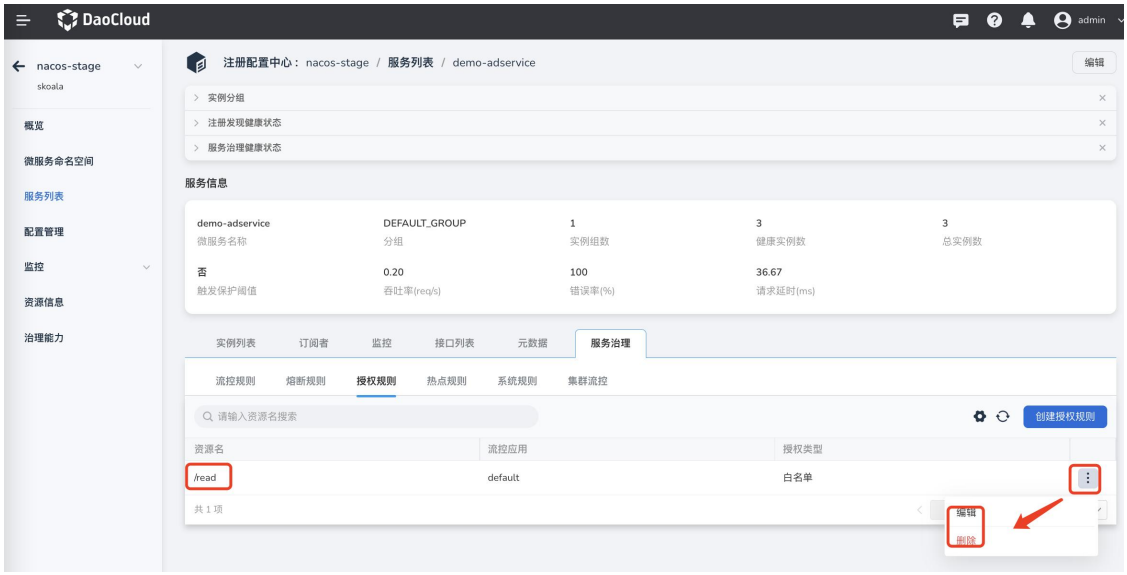
3. 参考下列说明填写规则配置，并在右下角点击确定。

   - 资源名：需要控制哪个资源的流量就填写哪个资源的名称，例如当前服务下的某个 API 接口、函数、变量等。
   - 流控应用：资源的请求来源，请求来源用 ，分隔，如 appA,appB
   - 授权类型——白名单：只有请求来源位于白名单内时才允许访问该资源。
   - 授权类型——黑名单：如果请求来源位于黑名单时，不允许其访问该资源。对于不在黑名单上的请求来源，不做限制。



微服务列表

4. 创建完成后可以在授权规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。



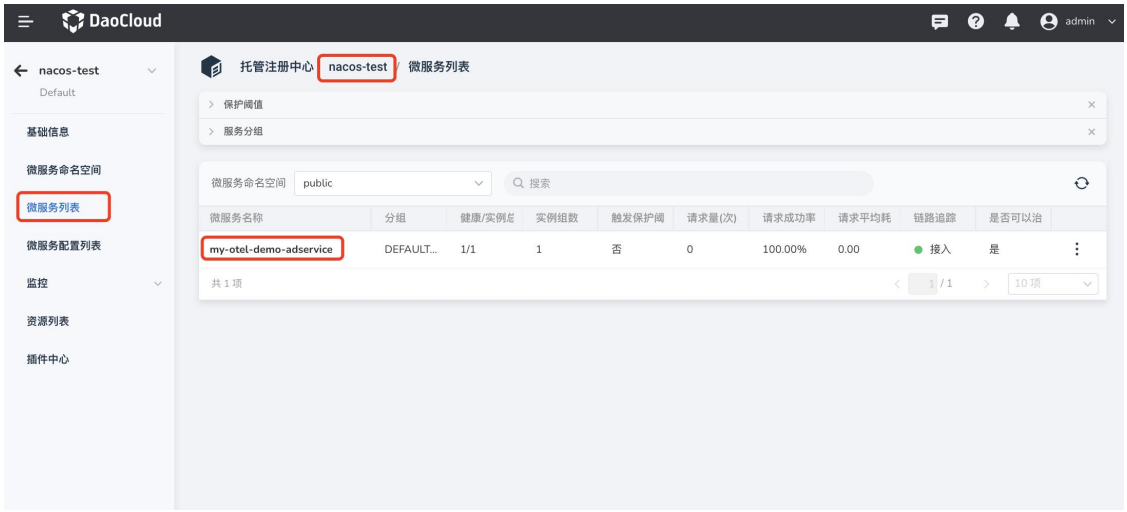
流控规则列表

查看微服务详情

在服务列表页面点击微服务名称可以查看服务详情，进一步查看实例列表、订阅者、监控、接口列表、元数据、服务治理等信息。

实例列表

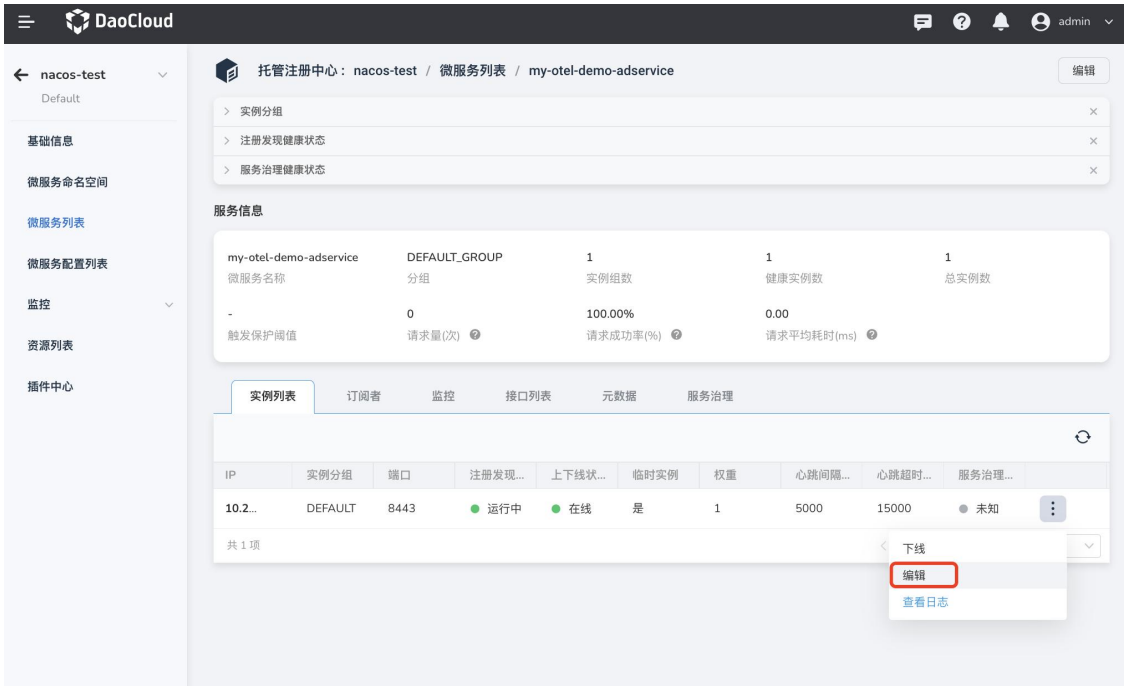
首先需要进入目标注册中心，在左侧导航栏点击微服务列表，点击目标微服务的名称，进入微服务详情页面后方可执行后续操作。



进入微服务列表

- 服务流量权重的调整

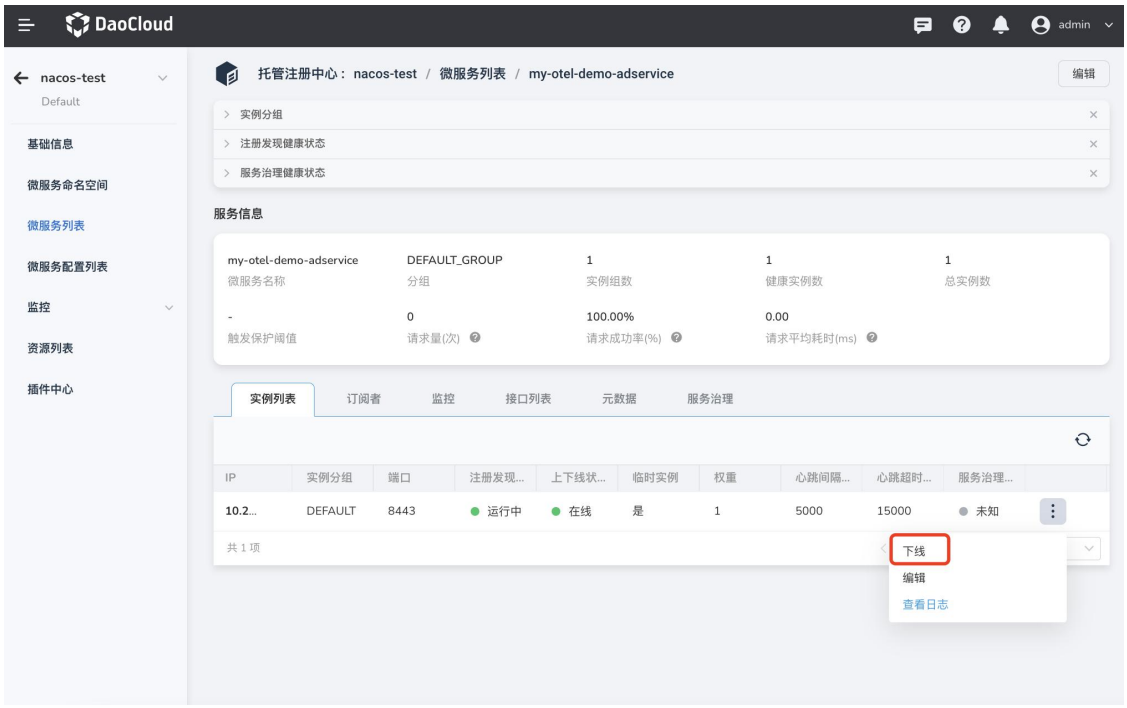
  提供流量权重控制的能力，同时开放服务流量的阈值保护，帮助用户更好的地保护服务服务提供者集群不被意外打垮。可以点击实例的编辑按钮，修改实例的权重。如果想增加实例的流量，可以将权重调大，如果不想实例接收流量，则可以将权重设为 0。



调整权重

- 实例上下线

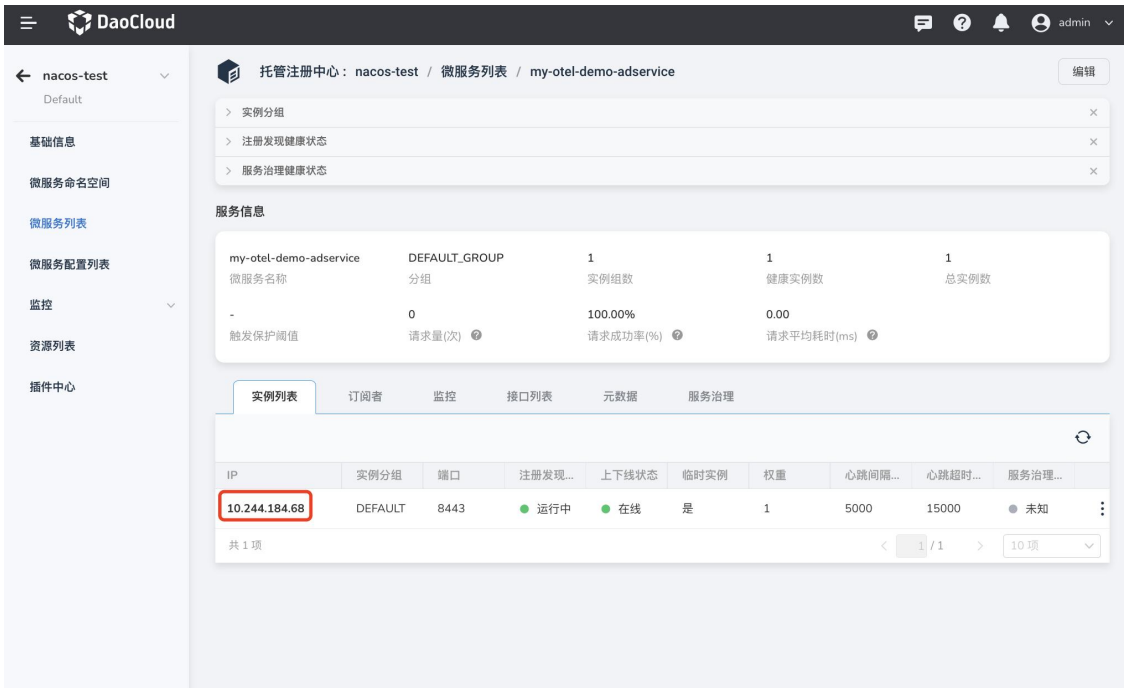
提供服务实例的上下线操作，在实例列表的操作列，可以点击实例的上线或者下线按钮。实例下线后状态会变为离线且不会被服务发现。



上下线

- 实例详情

  点击实例名称，可进入实例详情，查看实例的监控和元数据。

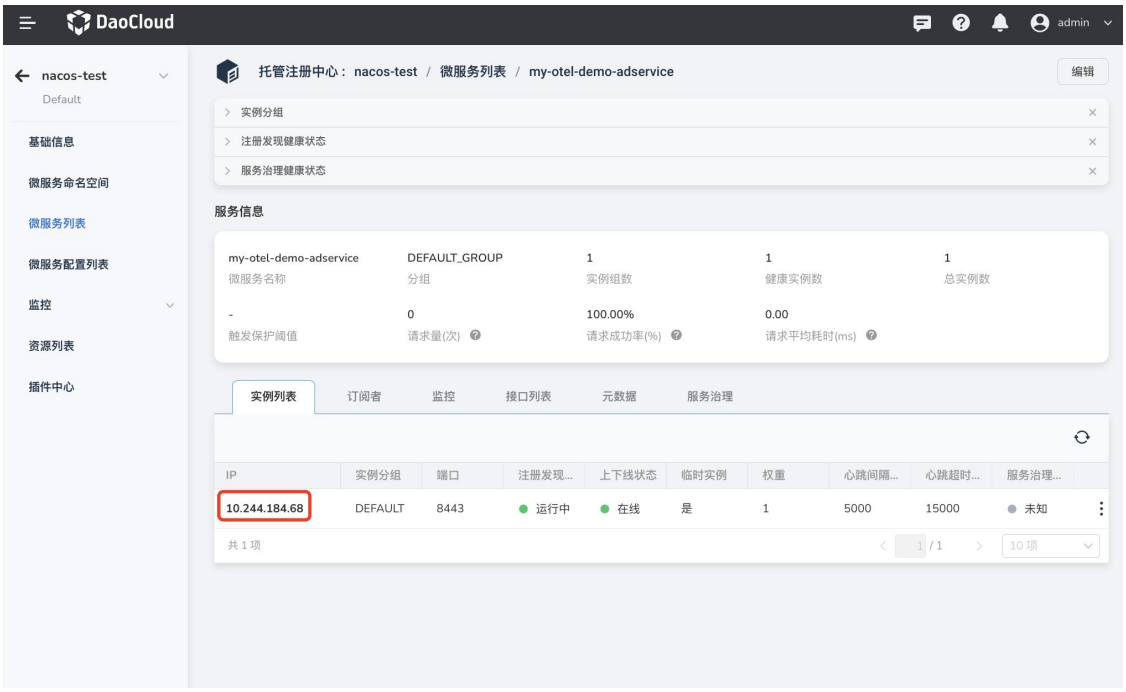


实例详情

一　实例监控

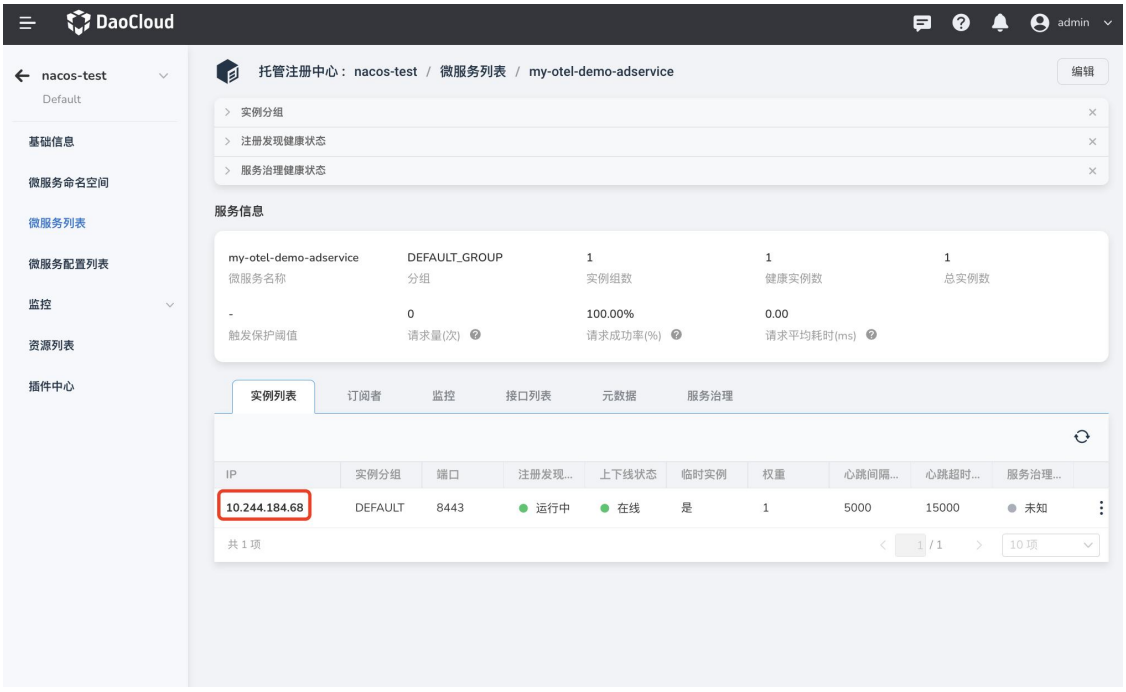
实例监控功能用于监控服务实例的状态，例如服务实例的请求数，错误率、响应耗时、请求率、CPU 指标、内存指标、读写速率、接收发送速率等指标的时序曲线。

响应耗时中 p95 代表线上 95% 的请求耗时都小于某个时间。



实例监控

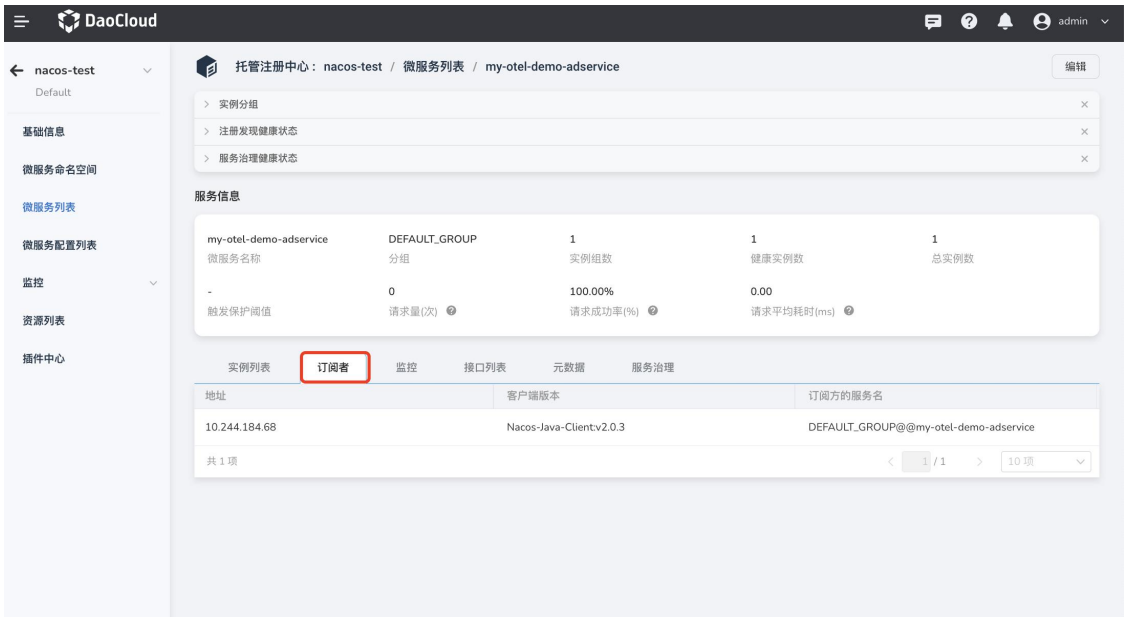
一 实例元数据



实例元数据

订阅者

进入订阅者列表页面，可查看到订阅者的信息，包括 IP 和端口、客户端版本和应用名。

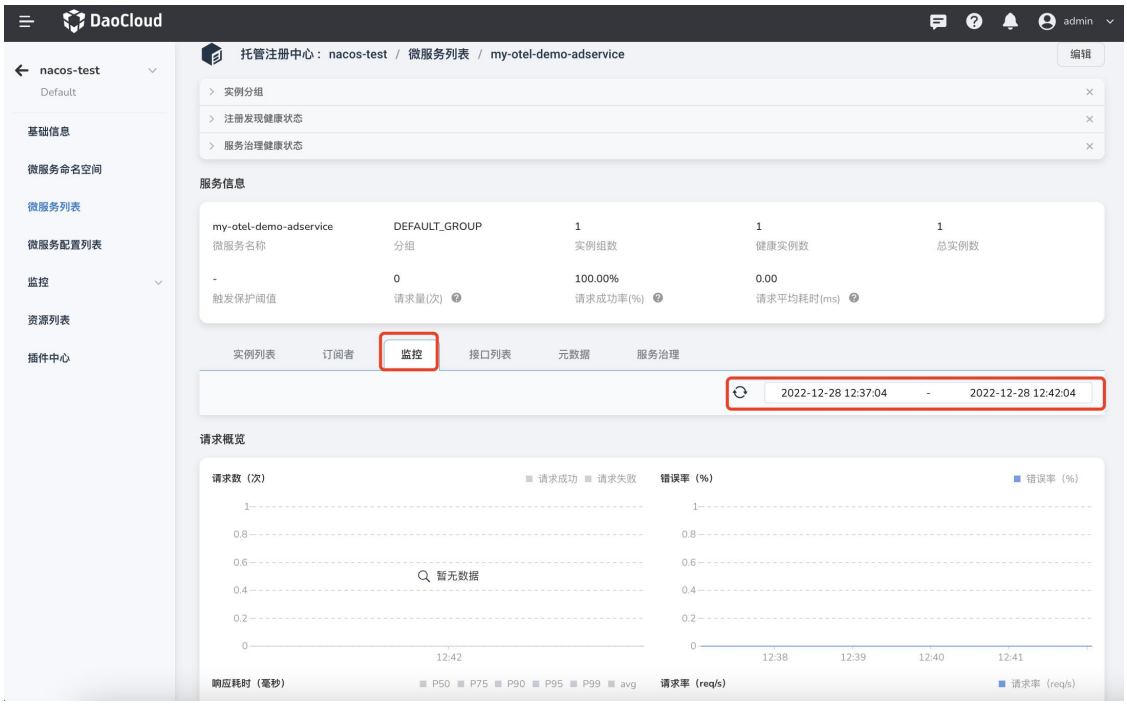


订阅者

服务监控

服务监控用于查看特定时间范围内，某个命名空间下微服务的运行状态，并且根据微服务的监控指标（请求数、错误率、响应耗时、请求率）初步判断是否出现异常。

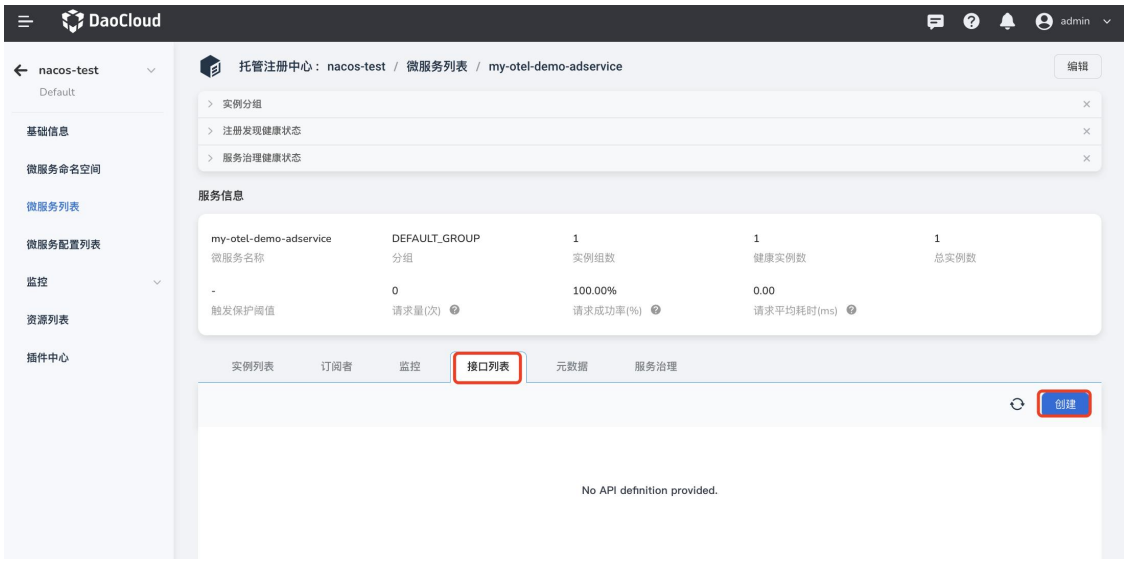
在监控图表的右上角可以更改监控数据的时间范围。



监控

接口列表

接口列表显示服务对外暴露的 API 列表。点击右侧的创建可以手动录入 API，

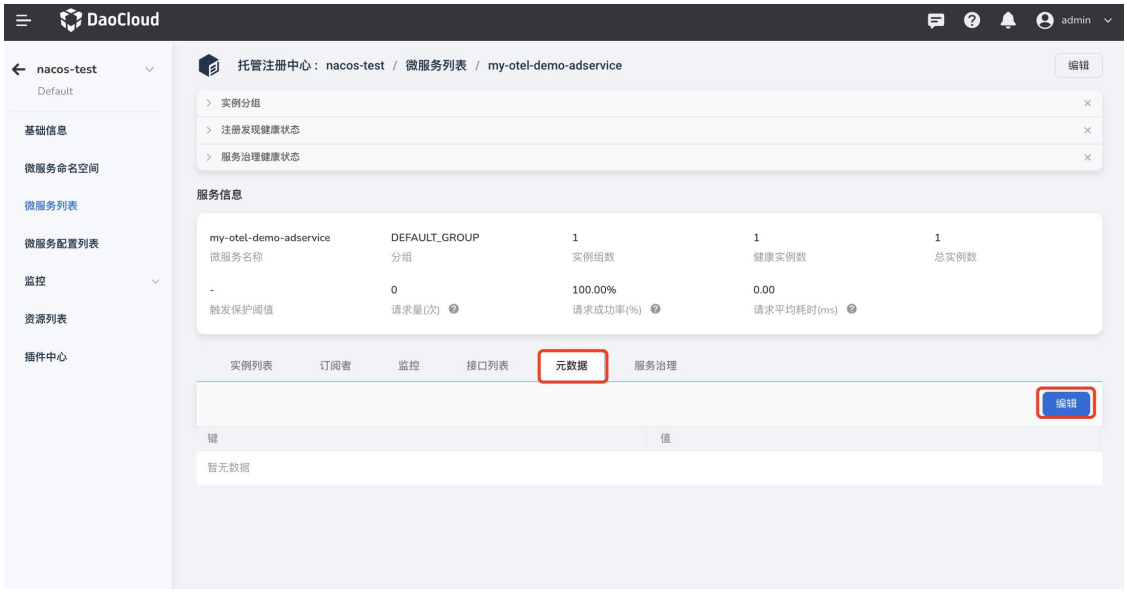


接口列表

微服务元数据

提供多个维度的服务元数据，帮助用户存储自定义的信息。这些信息都是以 Key-Value 的数据结构存储，在控制台上会以 k1=v1,k2=v2 这样的格式展示。

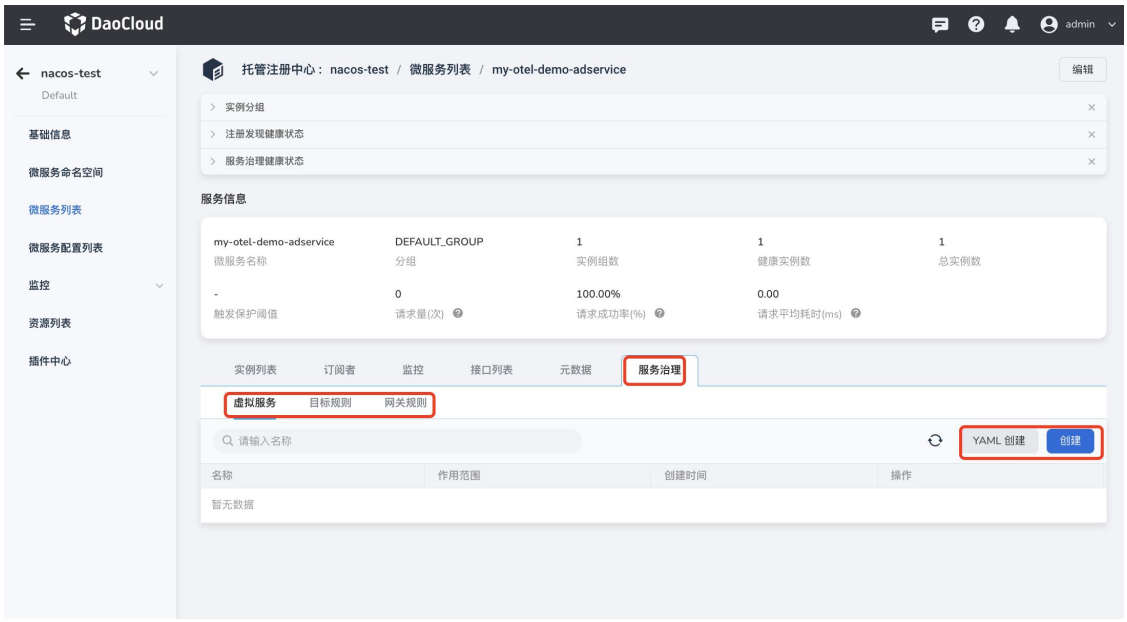
在右侧点击编辑可以修改元数据。



接口列表

微服务治理

开启微服务治理插件后，可以通过 YAML 文件或页面表单为服务创建虚拟服务、目标规则、网关规则等三种治理规则。有关微服务治理的更多说明，可参考流量治理。


DaoCloud
admin
nacos-test
Default
基础信息
微服务命名空间
微服务列表
微服务配置列表
监控
资源列表
插件中心
托管注册中心： nacos-test / 微服务列表 / my-otel-demo-adservice
编辑
实例分组
注册发现健康状态
服务治理健康状态
服务信息
my-otel-demo-adservice
微服务名称
DEFAULT_GROUP
分组
1
实例组数
1
健康实例数
1
总实例数
-
触发保护阈值
0
请求量(次)
100.00%
请求成功率(%)
0.00
请求平均耗时(ms)
实例列表
订阅者
监控
接口列表
元数据
服务治理
虚拟服务
目标规则
网关规则
请输入名称
YAML 创建
创建
名称
作用范围
创建时间
操作
暂无数据

接口列表

创建系统规则

系统规则从整体维度对应用入口流量进行控制，结合应用的 Load、总体平均 RT、入口 QPS 和线程数等监控指标，让系统的入口流量和系统的负载达到一个平衡，让系统尽可能跑在最大吞吐量的同时保证系统整体的稳定性。

系统规则是针对应用整体维度的，而不是资源维度的，并且仅对入口流量生效。

创建系统规则的方式如下：

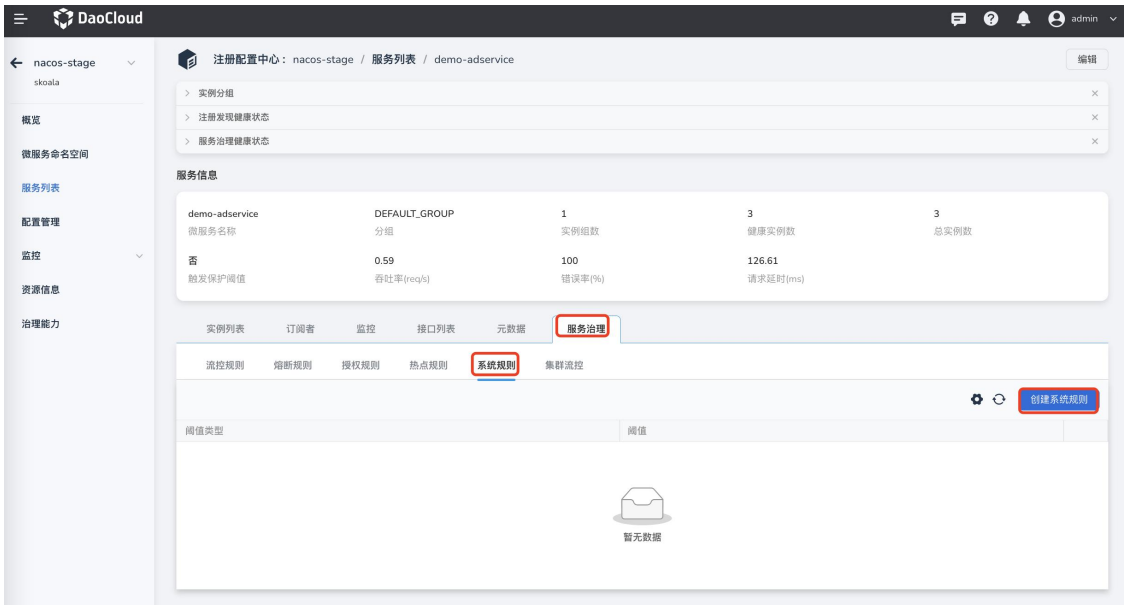
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

   注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。



微服务列表

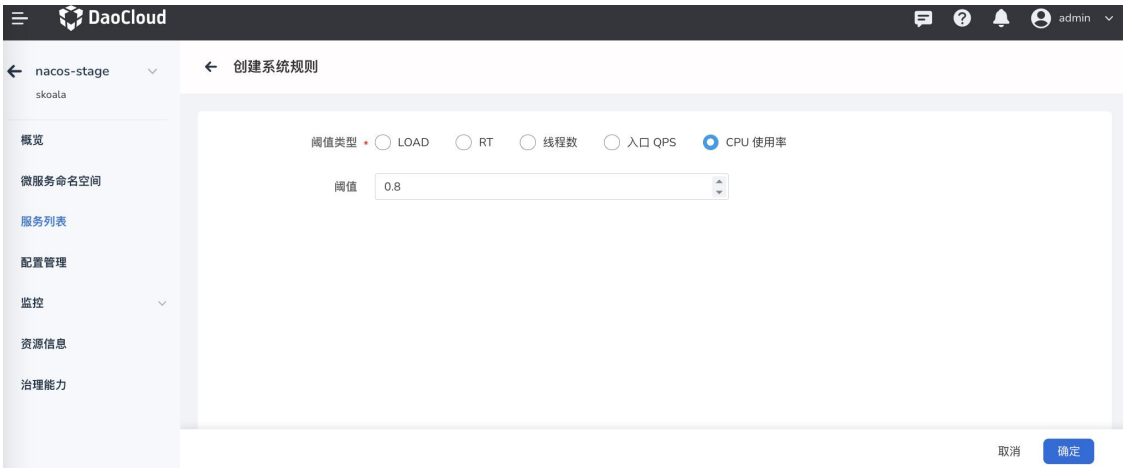
2. 选择系统规则，然后在右侧点击创建系统规则。



微服务列表

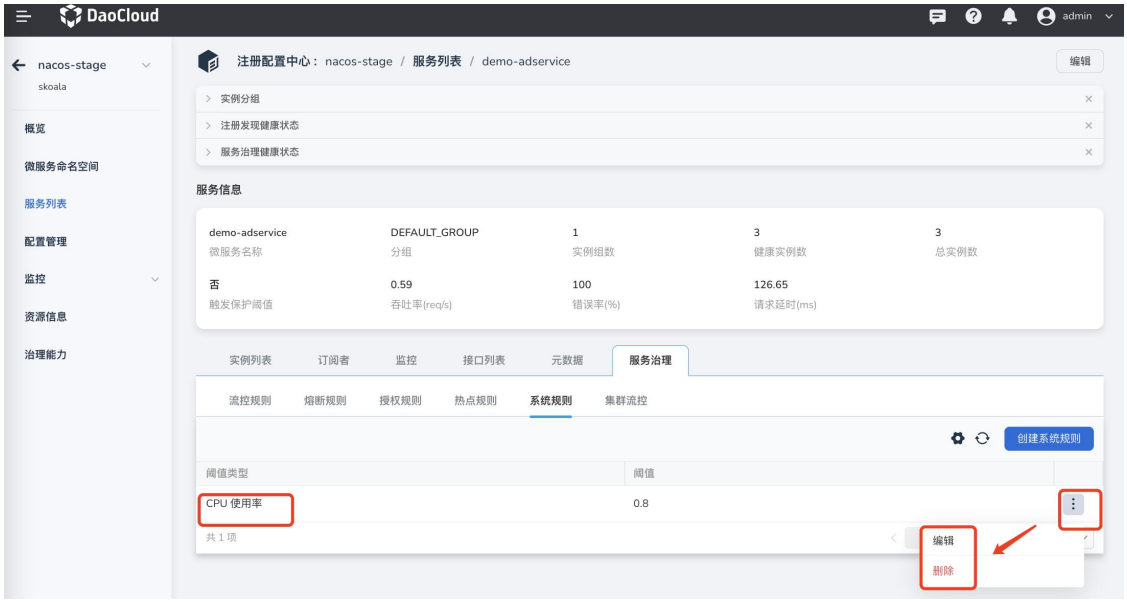
3. 参考下列说明填写规则配置，并在右下角点击确定。

   - 资源名：需要控制哪个资源的流量就填写哪个资源的名称。
   - Load：当系统 load1 （1 分钟的平均负载） 超过阈值，且系统当前的并发线程数超过系统容量时触发系统保护。系统容量由系统的 maxQps * minRt 计算得出。设定参考值一般是 CPU cores * 2.5。
   - RT：当单台机器上所有入口流量的平均响应时间达到阈值（单位：毫秒），触发系统流控规则。
   - 线程数：当单台机器上所有入口流量的并发线程数达到阈值时，触发系统流控规则。
   - 入口 QPS：当单台机器上所有入口流量的 QPS 达到阈值时，触发系统流控规则。
   - CPU 使用率：当系统 CPU 使用率超过阈值时（取值范围 0.0–1.0），触发系统流控规则。



微服务列表

4. 创建完成后可以在系统规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。



流控规则列表

创建集群流控规则

流控规则、熔断规则、授权规则、热点规则和系统规则只能统计本地的资源调用信息，使用于单机应用。但是在如今的云原生时代下，很多应用都是部署在集群中，可能分布在不同的机器上。在分布式场景下，上述规则存在一些缺点。例如，路由到集群中每台机器的流量不均衡，可能尚未达到整个集群的阈值总和时，某一台机器就因此超出阈值而停止服务。

集群流控模式正是为了解决这种问题而设计的。在集群流控模式下，需要理解两个概念：

- TokenServer：统计集群中所有实例的 QPS 并进行加总求和，将总值与集群整体阈值（单机阈值✖ 机器数量）进行比较。如果尚未达到集群整体的阈值，则向客户端发送 Token。
- TokenClient：即应用分布在不同机器上的各个实例。客户端会向 TokenServer 请求 Token。如果能成功获得 Token，说明尚未达到整体阈值，该实例可以继续处理请求。如果不能获得 Token，表示已经达到了整体的阈值，对该实例的请求会直接失败。

!!! note

– TokenSever 需要配置在 Nacos 的 `public` 命名空间，分组（Group）应为 `SENTINEL_GROUP`。
– TokenSever 是单点的，一旦 TokenSever 宕机，集群流控模式就会退化成单机限流模式。

创建集群流控规则的方式如下：

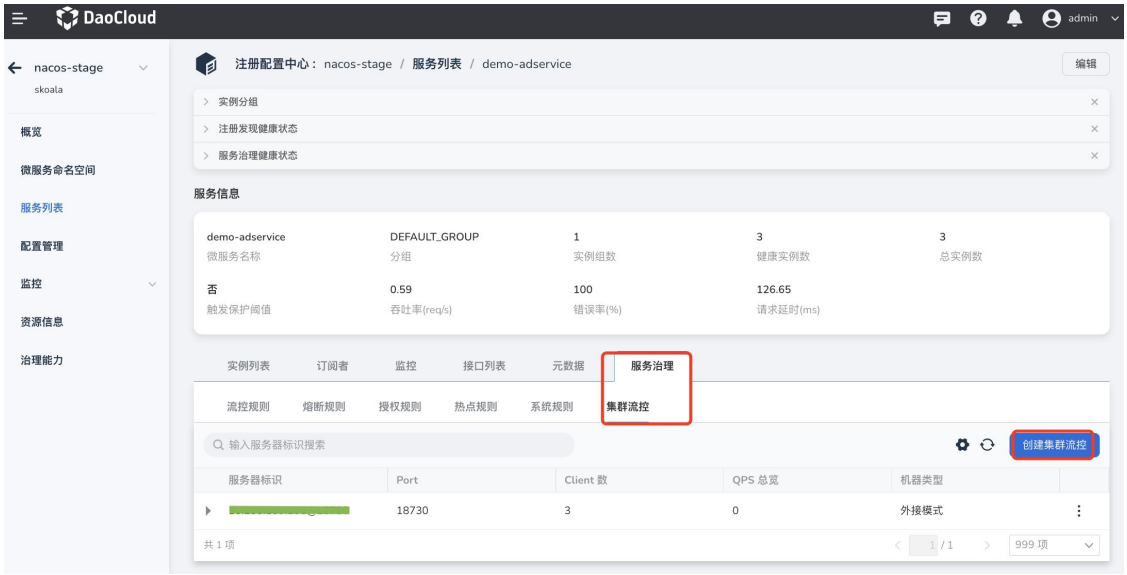
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。



微服务列表

2. 选择集群流控，然后在右侧点击创建集群流控。

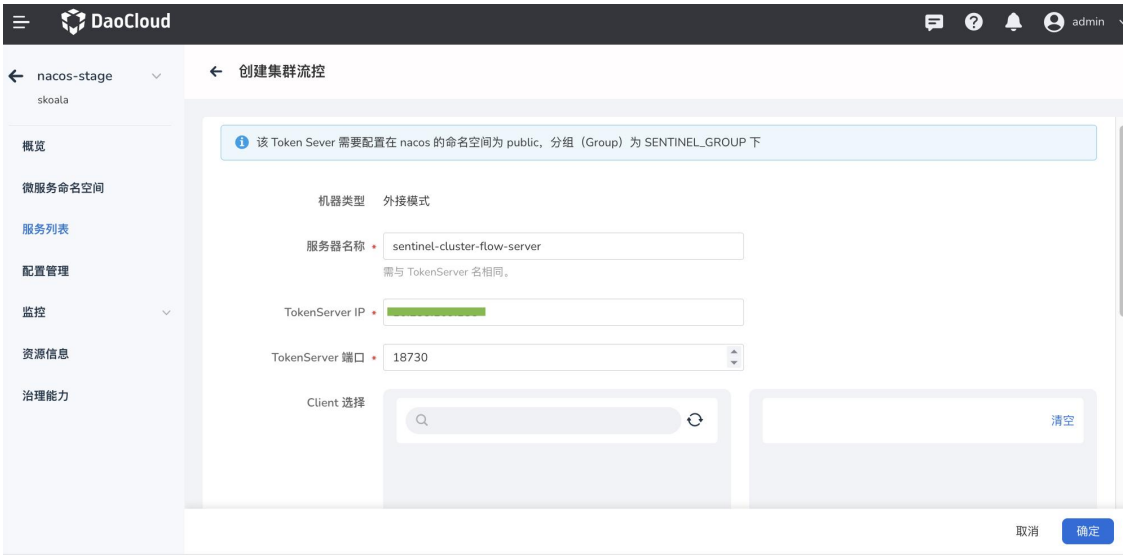


微服务列表

3. 参考下列说明填写规则配置，并在右下角点击确定。

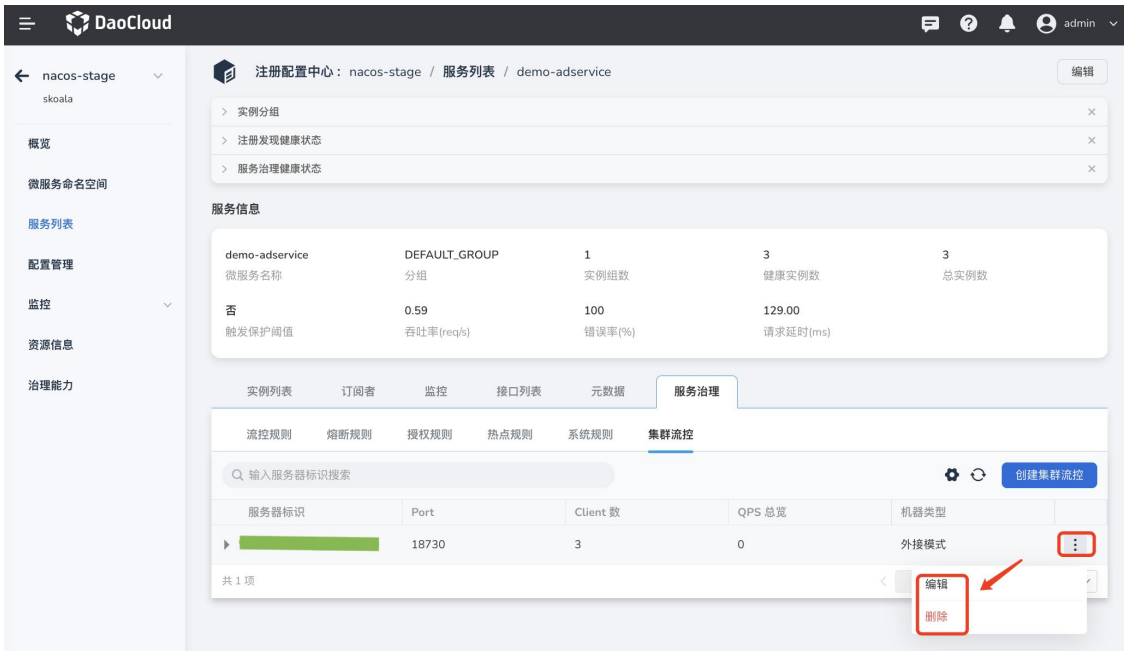
   - 服务器名称：Token 服务器的名称
   - TokenServer IP：Token 服务器的 IP 地址
   - TokenServer 端口：Token 服务器的端口号

- Client 选择：集群流控客户端，用于向所属 TokenServer 通信请求 Token。



微服务列表

4. 创建完成后可以在系统规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。



流控规则列表

创建流控规则

流控规则的原理是监控应用或服务流量的 QPS 指标，当指标达到阈值时根据预先设定的规则对请求流量进行控制，防止应用因短时间内无法处理过多流量而崩溃。使用流控规则后，系统可以在接下来的空闲期间逐渐处理堆积的请求，当指标重新恢复到阈值以下后，恢复正常的流量请求控制。

视频教程：Sentinel 流控规则

创建流控规则的方式如下：

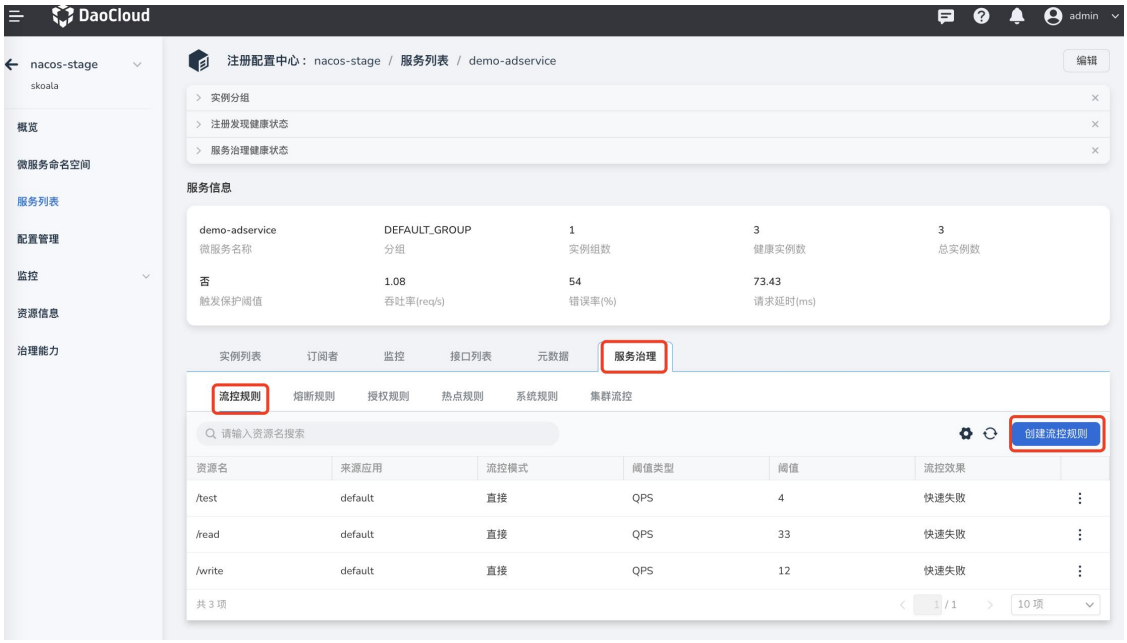
1. 点击目标托管注册中心的名称，然后在左侧导航栏点击微服务列表，在最右侧点击更多按钮选择治理。

   注意需要治理的微服务在是否可以治理一栏应该显示为是，才能进行后续步骤。

   微服务列表

2. 选择流控规则，然后在右侧点击创建流控规则。

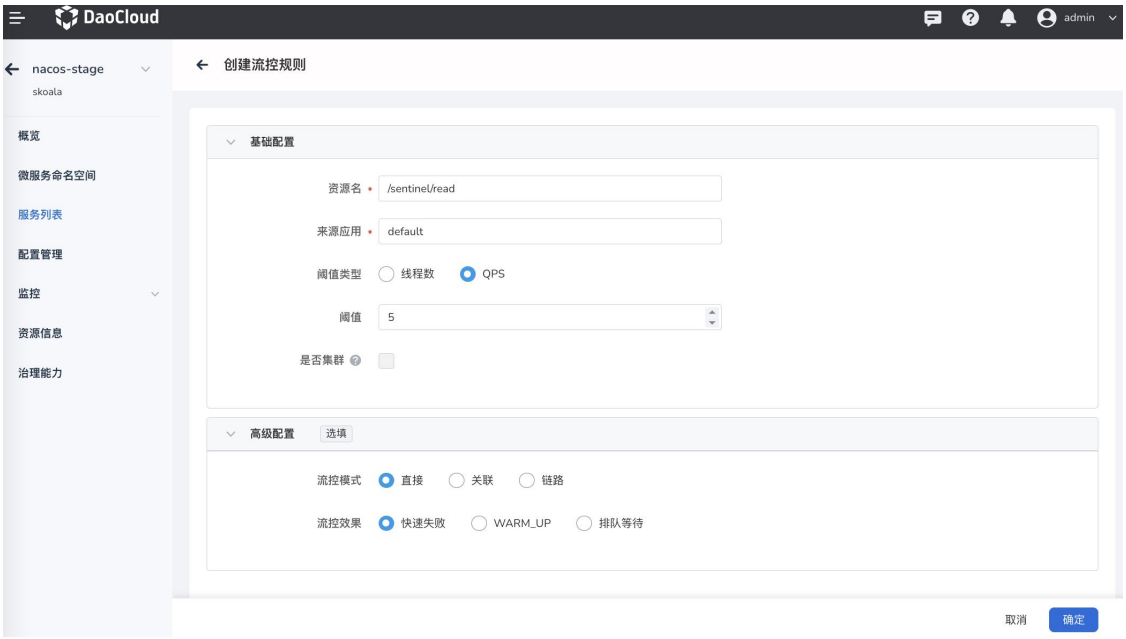


微服务列表

3. 参考下列说明填写规则配置，并在右下角点击确定。

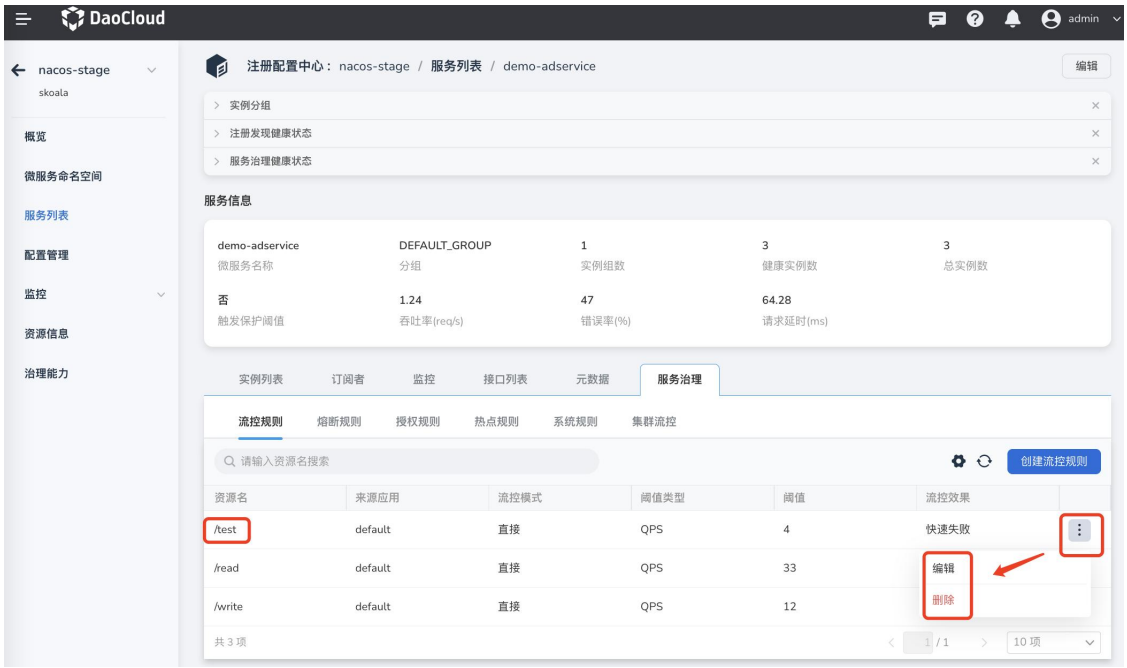
   - 资源名：需要控制哪个资源的流量就填写哪个资源的名称，例如当前服务下的某个 API 接口、函数、变量等。
   - 来源应用：Sentinel 可以针对调用者进行限流，只有当请求来自特定的调用者时才启用流控规则，来自其他调用者的请求不受此流控规则的约束。默认为 default，表示不区分来源，限制所有的调用者发来的请求。
   - 阈值类型——QPS：QPS 指每秒钟的请求数量。当资源每秒内响应的请求达到所设置的阈值时，启用流控规则开始限制服务接收的请求。
   - 阈值类型——线程数：调用该接口的线程数达到一定阈值时进行限流。
   - 是否集群：是否将所设置的规则应用于服务所在的集群范围。

- 流控模式——直接：直接对上方资源名字段填写的资源进行流量控制。
- 流控模式——关联：在 **同时调用** 资源及其关联资源时，如果对 **关联资源** 的调用达到上方设定的阈值，则对原资源进行流量控制。关联模式适用于在资源优先级不同的情况下，**对优先级较低的资源实施限流，以保护优先级更高的资源**。
- 流控模式——链路：如果上方资源名字段填写的资源的 QPS 超过设定的阈值时，对入口资源进行流量控制。入口资源即该调用链路入口的上下文名称。
- 流控效果——快速失败：当资源响应的请求达到所设置的阈值时，直接拒绝后续的调用请求。阈值类型必须需设成 QPS，否则无效。
- 流控效果——WARM_UP：在设定的预热时间内逐渐将 QPS 增加到设定的阈值，之后一直维持在该阈值。阈值类型必须需设成 QPS，否则无效。这样可以缓慢增加通过的流量，逐渐预热系统，防止空闲系统突然涌入大量流量而被压垮。
- 流控效果——排队等待：QPS 超过阈值时后续请求排队等待响应，超过所设定的超时时间（单位：毫秒）后请求失败。阈值类型必须需设成 QPS，否则无效。适用于处理间隔性突发流量，在某一秒有大量请求涌入，而接下来的几秒则处于空闲状态，例如消息队列。



微服务列表

4. 创建完成后可以在流控规则列表中查看新建的规则。在右侧点击更多按钮可以更新规则或者删除该规则。



流控规则列表

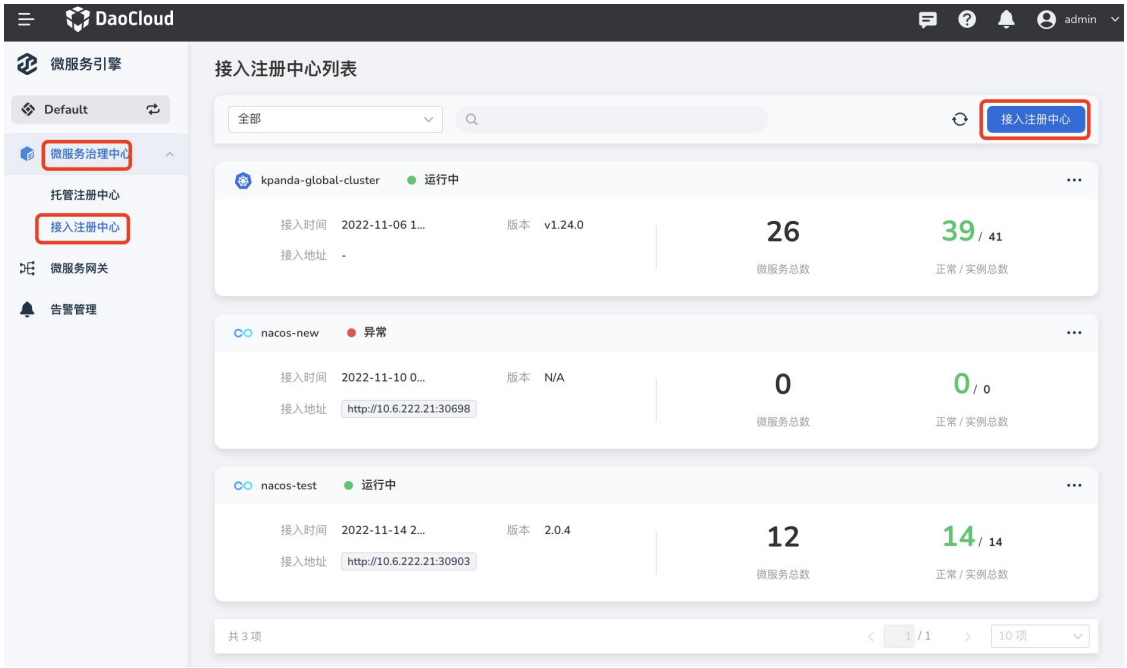
接入注册中心

注册中心支持接入 Nacos 注册中心、Eureka 注册中心、Zookeeper 注册中心、Kubernetes 注册中心、Mesh 注册中心。

相对于托管型注册中心而言，接入型注册中心只支持一些基础操作，例如查看基本信息、监控信息等。如需体验更高级更全面的管理操作，需要创建托管注册中心。

接入注册中心的步骤如下：

1. 在左侧导航栏点击微服务治理中心->接入注册中心，然后在页面右上角点击接入注册中心。



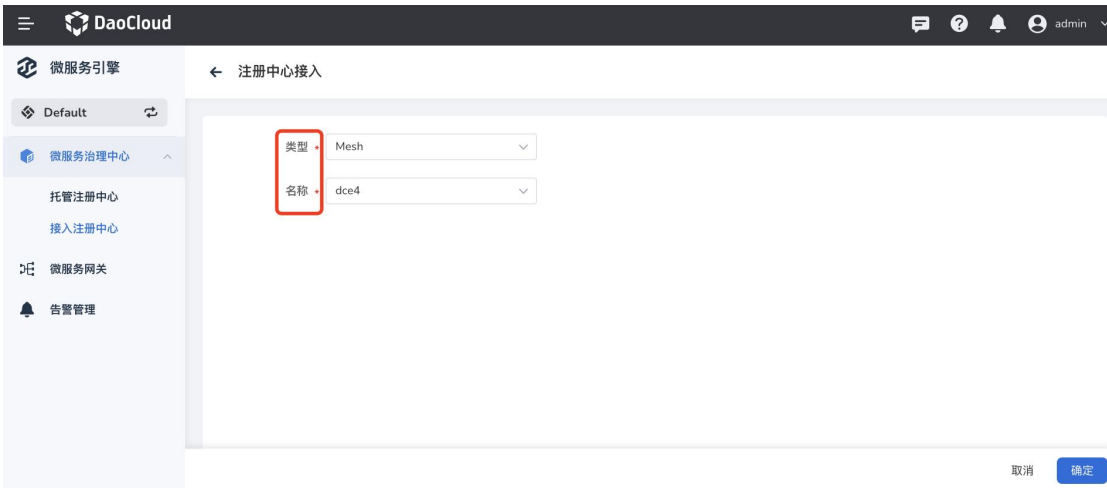
进入接入注册中心页面

2. 填写配置信息，然后在页面底部点击确定。

   接入不同类型的注册中心需要填写不同的配置信息。

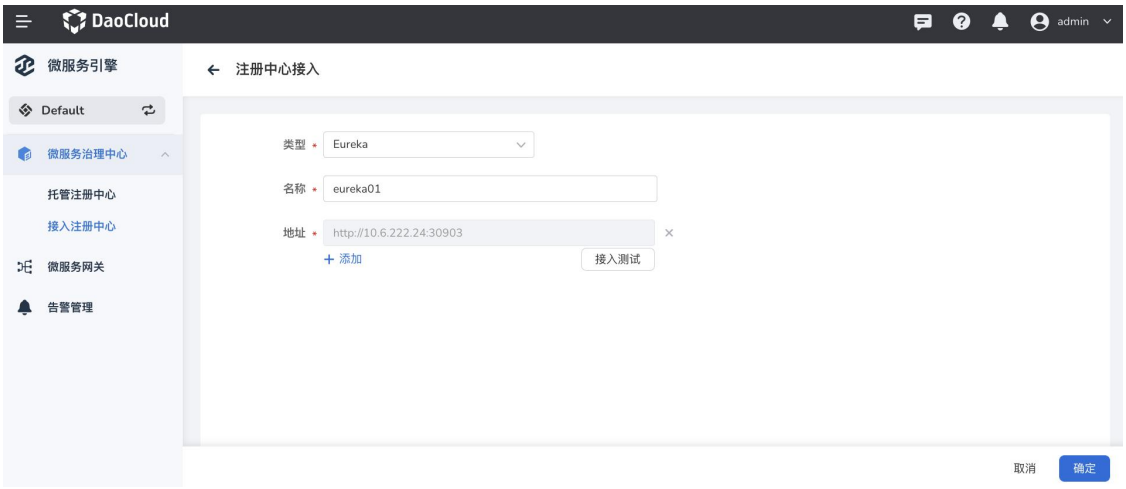
- Kubernetes/Mesh 注册中心：直接选择想要接入的集群或网格服务。

    - 如果找不到想要添加的 Kubernetes 集群，可以去容器管理模块接入集群或创建集群。
    - 如果找不到想要添加的网格服务，可以去网格服务模块创建网格。

    接入 Mesh/Kubernetes

- Nacos/Zookeeper/Eureka 注册中心：填写注册中心的名称和地址，点击确认。

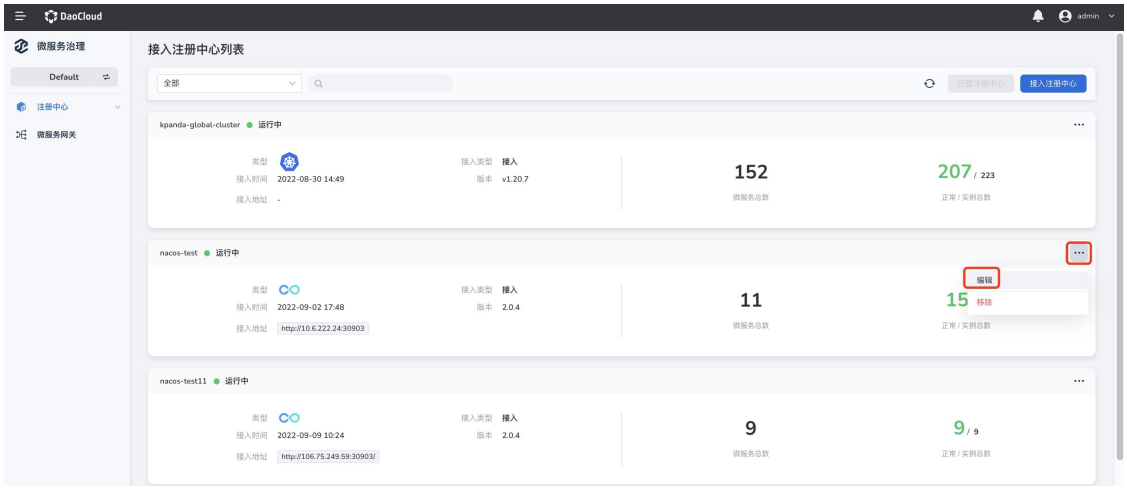


接入 Nacos/Zookeeper/Eureka

更新注册中心配置

微服务引擎目前仅支持更新 Nacos/Zookeeper/Eureka 注册中心的配置。

1. 在接入注册中心列表页选择需要更新的注册中心，在右侧点击 ⋯ 并选择编辑。



进入更新页面

2. 增加或删除注册中心的地址，然后在页面底部点击确定。



进入更新页面

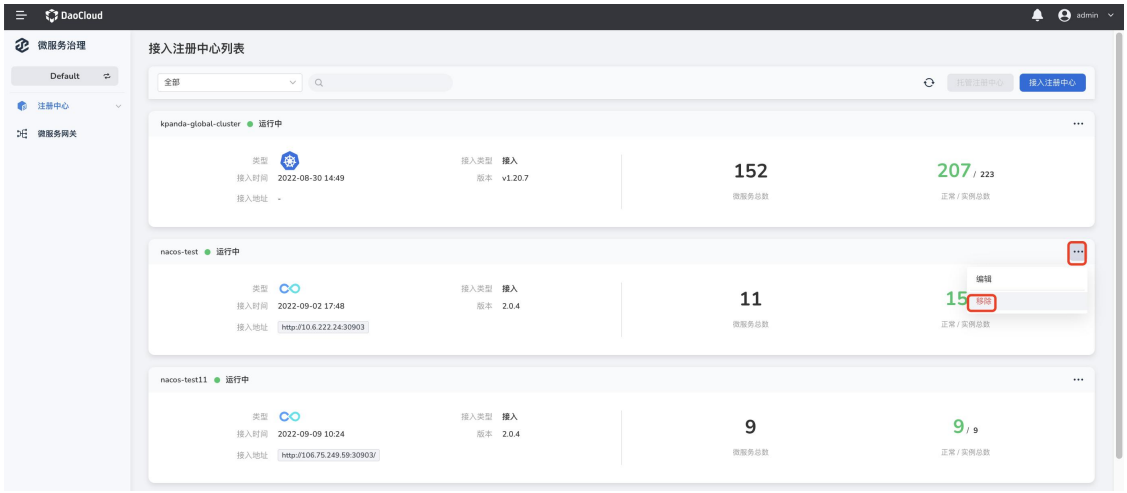
!!! note

如需更新 Kubernetes/Mesh 注册中心：

– 可以先[移除已经接入的注册中心](remove-registry.md)，然后再重新接入其他的注册中心。
– 也可以去容器管理模块[更新对应的集群](../../../kpanda/user-guide/clusters/upgrade-cluster.md)，或者去服务网格模块[更新对应的网格服务](../../../mspider/user-guide/service-mesh/README.md)。

移除注册中心

1. 在接入注册中心列表页选择需要移除的注册中心，在右侧点击 ⋯ 并选择移除。



进入移除页面

2. 输入注册中心的名称，点击移除。



进入移除页面

!!! note

接入型的注册中心仅支持`移除`操作，而托管型的注册中心仅支持`删除`操作。二者的区别在于：

– 移除：只是将注册中心从 DCE 5.0 的微服务引擎中移除，不会删除原有的注册中心和数据，后续还可以再次接入该注册中心。
– 删除：删除注册中心及其中的所有数据，后续无法再次使用该注册中心，需要重新创建新的注册中心。

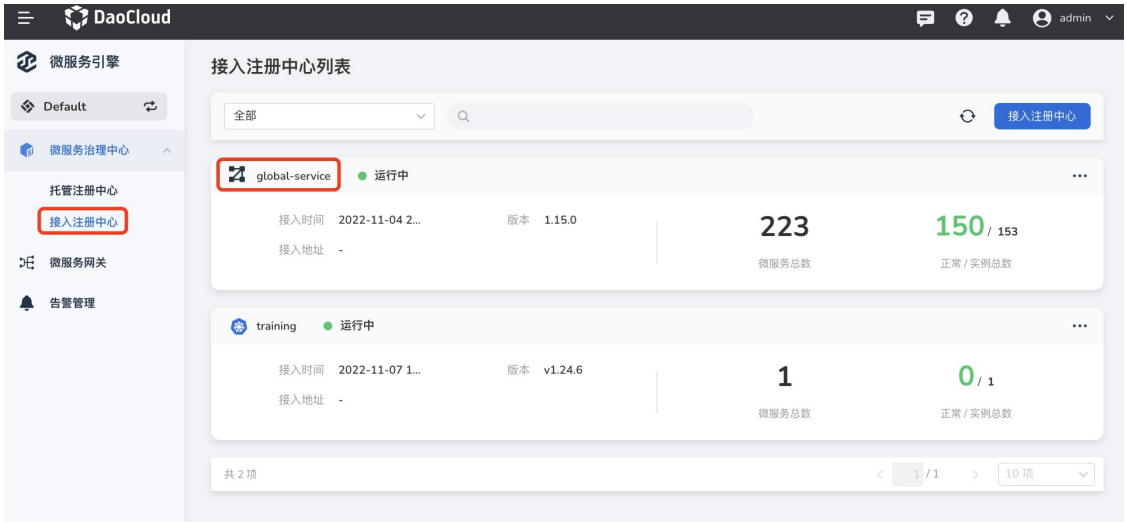
微服务管理

接入注册中心之后，可以通过注册中心对其中的微服务进行管理。微服务管理主要指查看注册中心下的微服务，

!!! note

接入型注册中心仅支持基础的管理操作。对于更复杂的管理场景，建议创建[托管注册中心](../hosted/create-registry.md)以便执行更多高级操作。

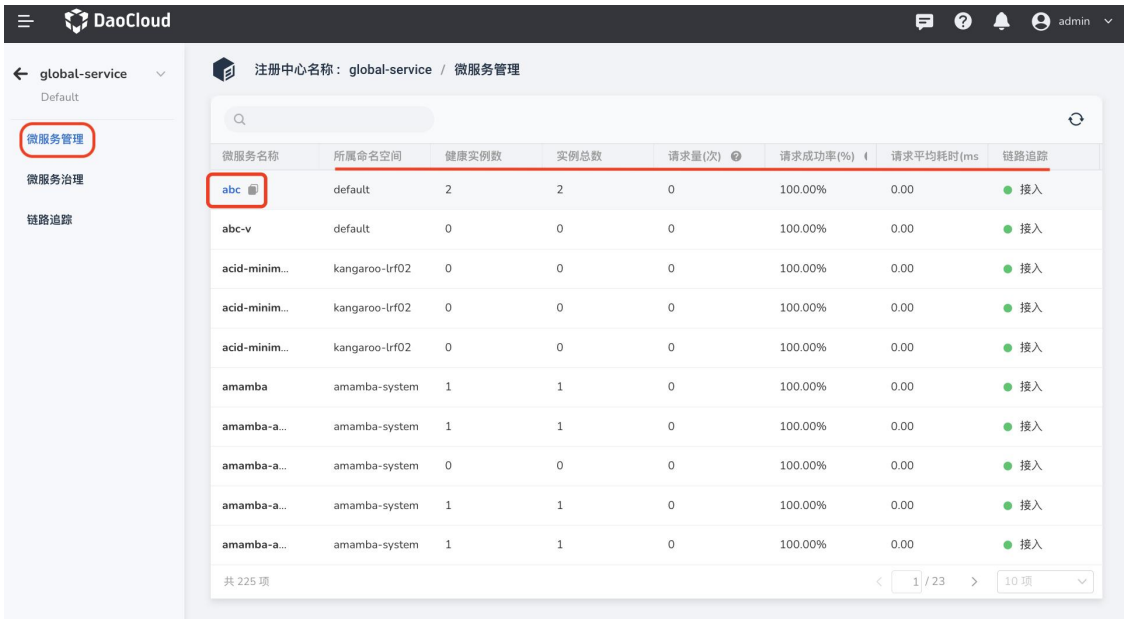
1. 在接入注册中心列表页面点击目标注册中心的名称。



点击注册中心名称

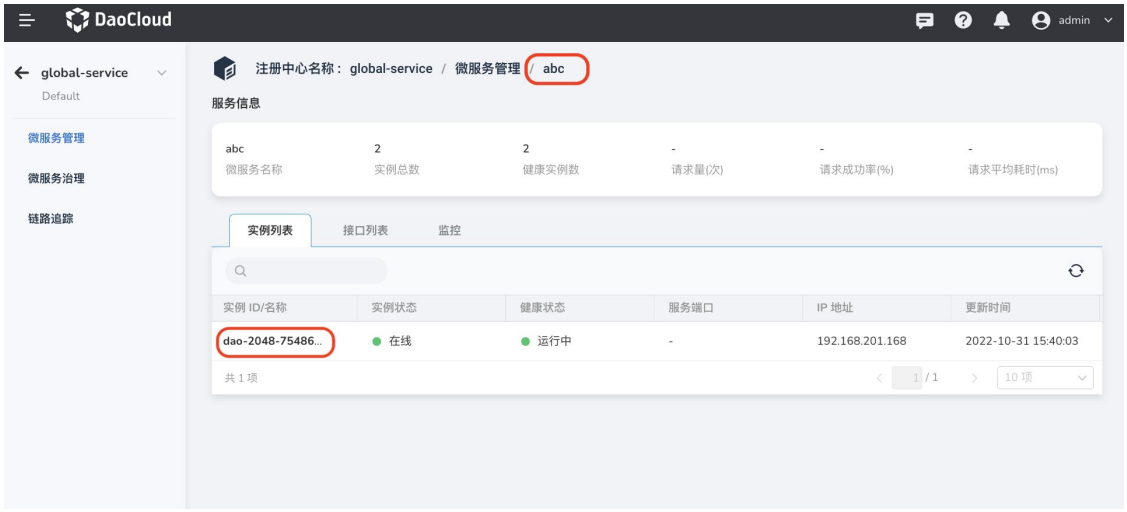
2. 在左侧导航栏点击微服务管理，查看微服务列表和基本信息。

   在当前页面，可以复制微服务的名称，可以查看当前注册中心下的所有微服务，以及各个微服务的所属命名空间、实例情况、请求统计数据等。



点击注册中心名称

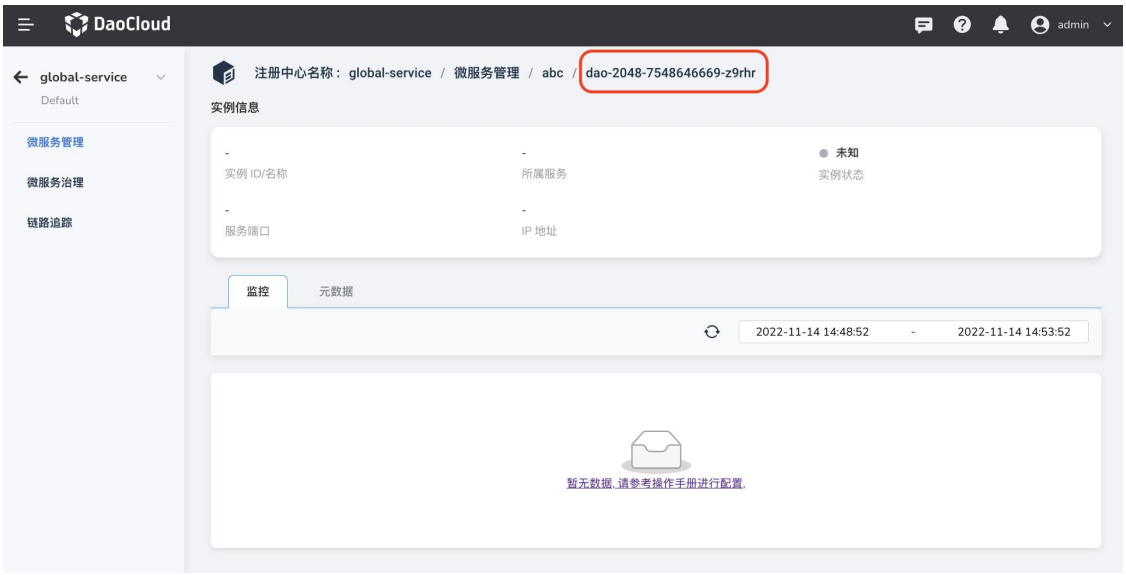
3. 点击微服务的名称，查看微服务的实例列表、接口列表、监控信息等。



点击注册中心名称

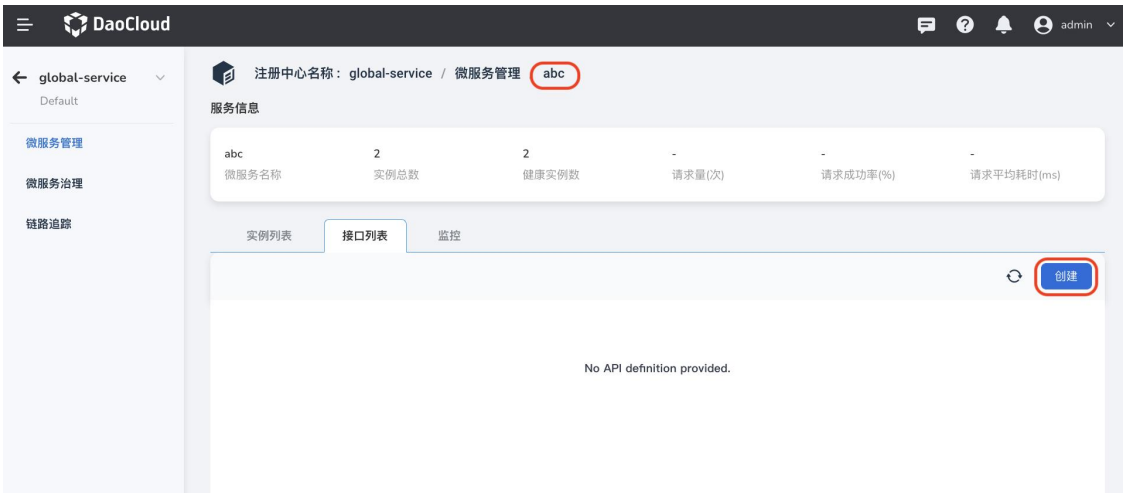
- 实例列表：查看实例状态、IP 地址、服务端口等。

  点击实例名称，还可以进一步查看该实例的监控信息和元数据。



实例详情

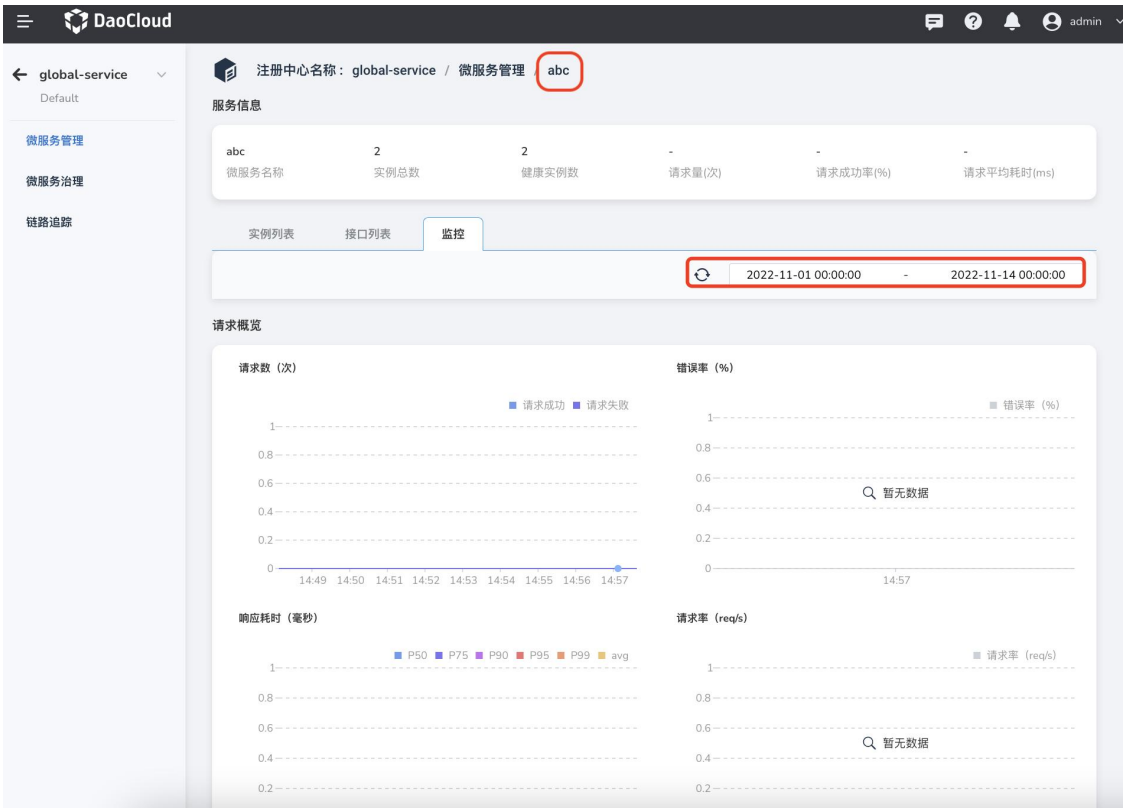
- 接口列表：查看微服务已经具有的接口，或者创建新的接口。



实例详情

- 监控信息：查看微服务的监控信息，包括请求数、错误率、响应耗时、请求率等。

  支持自定义时间范围。


DaoCloud
admin
global-service
Default
微服务管理
微服务治理
链路追踪
注册中心名称： global-service / 微服务管理 / abc
服务信息
abc
微服务名称
2
实例总数
2
健康实例数
-
请求量(次)
-
请求成功率(%)
-
请求平均耗时(ms)
实例列表
接口列表
监控
2022-11-01 00:00:00 - 2022-11-14 00:00:00
请求概览
请求数（次）
请求成功 请求失败
1
0.8
0.6
0.4
0.2
0
14:49 14:50 14:51 14:52 14:53 14:54 14:55 14:56 14:57
错误率（%）
错误率（%）
暂无数据
14:57
响应耗时（毫秒）
P50 P75 P90 P95 P99 avg
请求率（req/s）
请求率（req/s）
暂无数据

实例详情

链路追踪

DCE 5.0 支持服务级别的链路追踪与查询，查看各个微服务的链路时延分布情况，在出现故障时可以快速定位故障。

更多详情，可参考可观测性->链路查询。

在线升级微服务引擎

开始升级操作之前，了解一下微服务引擎的部署架构有助于更好地理解后续的升级过程。

微服务引擎由两个组件构成：

- skoala 组件安装在控制面集群，用于在 DCE 5.0 的一级导航栏中加载微服务引擎模块
- skoala-init 组件安装在工作集群，用于提供微服务引擎的核心功能，例如创建注册中心、网关实例等

!!! note

- 升级微服务引擎时需要同时升级这两个组件，否则会存在版本不兼容的情况。
- 有关微服务引擎的版本更新情况，可参考 [release notes](../intro/release-notes.md)。

升级 skoala 组件

由于 skoala 组件安装在控制面集群中，所以需要在控制面集群中执行下列操作。

1. 执行如下命令备份原有数据

   ```
   helm -n skoala-system get values skoala > skoala.yaml
   ```

2. 添加微服务引擎的 Helm 仓库

   ```
   helm repo add skoala https://release.daocloud.io/chartrepo/skoala
   ```

3. 更新微服务引擎的 Helm 仓库

   ```
   helm repo update
   ```

4. 执行 helm upgrade 命令

```
helm --kubeconfig /tmp/deploy-kube-config upgrade --install --create-namespace -n skoala-system skoala skoala/skoala --version=0.19.2 --set hive.image.tag=v0.19.2 --set sesame.image.tag=v0.19.2 -f skoala.yaml
```

需要将 version、hive.image.tag、sesame.image.tag 三个参数的值调整为您需要升级到的微服务引擎的版本号。

升级 skoala-init 组件

由于 skoala-init 组件安装在工作集群中，所以需要在每个工作集群中各执行一次下列操作。

1. 备份原有参数

   ```
   helm -n skoala-system get values skoala-init > skoala-init.yaml
   ```

2. 添加微服务引擎的 Helm 仓库

   ```
   helm repo add skoala https://release.daocloud.io/chartrepo/skoala
   ```

3. 添加微服务引擎的 Helm 仓库

   ```
   helm repo update
   ```

4. 执行 helm upgrade 命令

   ```
   helm --kubeconfig /tmp/deploy-kube-config upgrade --install --create-namespace -n skoala-system skoala-init skoala/skoala-init --version=0.19.2 --set nacos-operator.image.tag=v0.19.2 --set skoala-agent.image.tag=v0.19.2 --set sentinel-operator.image.tag=v0.19.2 -f skoala-init.yaml
   ```

   !!! note

   - 需要将 `version`、`nacos-operator.image.tag`、`skoala-agent.image.tag`、`sentinel-operator.image.tag` 四个参数的值调整为您需要升级到的微服务引擎的版本号。

5. 根据自身需要，手动删除所需更新的其他 crd 文件，重新运行对应的文件。

   ```
   kubectl apply -f xxx.yaml
   ```

在线安装微服务引擎

如需安装微服务引擎，推荐通过 DCE 5.0 商业版 的安装包进行安装；通过商业版可以一次性同时安装 DCE 的所有模块。

本教程旨在补充需要手工 **单独在线安装** 微服务引擎模块的场景。下文出现的 skoala 是微服务引擎的内部开发代号，代指微服务引擎。

!!! note

本文提供了在线安装的方式，如果已部署了离线商业版，建议参考[离线升级微服务引擎](offline-upgrade.md)离线安装或升级微服务引擎。

使用商业版安装包安装

通过商业版安装微服务引擎时，需要注意商业版的版本号（点击查看最新版本）。需要针对不同版本执行不同操作。

商业版的 **版本号 ⩾ v0.3.29** 时，默认会安装微服务引擎，但仍旧建议检查 mainfest.yaml 文件进行确认 components/skoala/enable 的值是否为 true，以及是否指定了 Helm 的版本。

> 商业版中默认安装的是经过测试的最新版本。如无特殊情况，不建议修改默认的 Helm 版本。

??? note “如果商业版 ⩽ v0.3.28，点击查看对应操作”

此注释仅适用于商业版 ⩽ v0.3.28；大多数情况下您的版本都会大于此版本。

执行安装命令时，默认不会安装微服务引擎。需要对照下面的配置修改 `mainfest.yaml` 以允许安装微服务引擎。

修改文件：

```bash
./dce5-installer install-app -m /sample/manifest.yaml
```

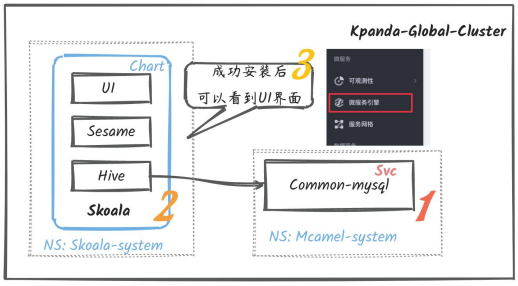
修改后的内容：

```yaml
...
components:
  skoala:
    enable: true
    helmVersion: v0.12.2 # 应该获取当前最新的版本号
    variables:
...
```

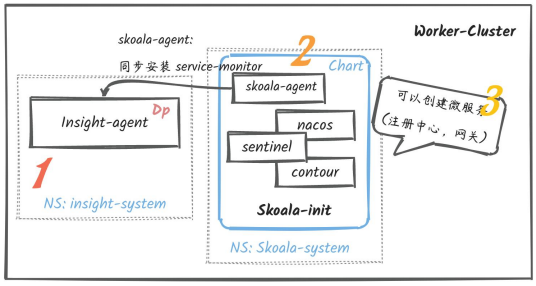
手动安装前检测

微服务引擎由两个组件构成，在代码中分别命名为 skoala 和 skoala-init。安装微服务引擎时需要同时安装这两个组件。

微服务引擎部署结构



Skoala 系统模块，依赖 common-mysql



Skoala 应用模块，部署在工作集群，依赖 Insight-agent

image

左侧蓝色框内的 chart 即 skoala 组件，需要安装在控制面集群，即 DCE 5.0 的全局集群 kpanda-global-clsuter，详情可参考 DCE 5.0 的部署架构。安装 skoala 组件之后即可以在 DCE 5.0 的一级导航栏中看到微服务引擎模块。另外

需要注意：安装 skoala 之前需要安装好其依赖的 common-mysql 组件用于存储资源。

右侧蓝色框内的 chart 即 skoala-init 组件，需要安装在工作集群。安装 skoala-init 组件之后即可以使用微服务引擎的各项功能，例如创建注册中心、网关实例等。另外需要注意，skoala-init 组件依赖 DCE 5.0 可观测模块的 insight-agent 组件提供指标监控和链路追踪等功能。如您需要使用该项功能，则需要事先安装好 insight-agent 组件，具体步骤可参考安装组件 insight-agent。

检测微服务引擎是否已安装

查看 skoala-system 命名空间中是否有以下对应的资源。如果没有任何资源，说明目前尚未安装微服务引擎。

```
~ kubectl -n skoala-system get pods
NAME                                READY   STATUS    RESTARTS      AGE
hive-8548cd9b59-948j2               2/2     Running   2 (3h48m ago)   3h48m
sesame-5955c878c6-jz8cd             2/2     Running   0               3h48m
ui-7c9f5b7b67-9rpzc                 2/2     Running   0               3h48m

~ helm -n skoala-system list
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
skoala  skoala-system   3               2022-12-16 11:17:35.187799553 +0800 CST deployed        skoala-0.13.0           0.13.0
```

检测依赖的存储组件

安装微服务引擎时需要用到 common-mysql 组件来存储配置，所以要确保该组件已经存在。此外，还需要查看 common-mysql 命名空间中是否有名为 skoala 的数据库。

```
~ kubectl -n mcamel-system get statefulset
NAME                                    READY   AGE
mcamel-common-mysql-cluster-mysql       2/2     7d23h
```

建议使用如下参数为微服务引擎配置数据库信息：

- host: mcamel-common-mysql-cluster-mysql-master.mcamel-system.svc.cluster.local
- port: 3306
- database : skoala
- user: skoala
- password:

检测依赖的监控组件

微服务引擎依赖 DCE 5.0 可观测性模块的能力。如您需要监控微服务的各项指标、追踪链路，则需要在集群中安装对应的 insight-agent，具体说明可参考。



image

!!! note

– 如果安装 `skoala-init` 之前没有事先安装 `insight-agent`，则不会安装 `service-monitor`。
– 如果需要安装 `service-monitor`，请先安装 `insight-agent`，然后再安装 `skoala-init`。

手动安装过程

一切就绪之后，就可以开始正式安装微服务引擎了。具体的流程如下：

!!! note

– 如果安装的是 skoala-release/skoala v0.17.1 以下的版本，则需要手动初始化数据库表。

– 如果安装的是 skoala-release/skoala v0.17.1 或更高版本，系统会自动初始化数据库表，无需手动进行。

??? note “如果初始化失败，请检查 skoala 数据库内是否有下方 3 张数据表以及对应的 SQL 是否全部生效。”

```sql
mysql> desc api;
+-----------------+-----------------+------+-----+-------------
------+-----------------------------------------------+
| Field           | Type            | Null | Key | Default           | Extra
                                    |
+-----------------+-----------------+------+-----+-------------
------+-----------------------------------------------+
| id              | bigint unsigned | NO   | PRI | NULL              | auto_incr
ement                                 |
| is_hosted       | tinyint         | YES  |     | 0                 |
                                    |
| registry        | varchar(50)     | NO   | MUL | NULL              |
                                      |
| service_name    | varchar(200)    | NO   |     | NULL              |
```

```
                                                  |
| nacos_namespace   | varchar(200)      | NO    |       | NULL                  |
                                                  |
| nacos_group_name | varchar(200)      | NO    |       | NULL                  |
                                                  |
| data_type           | varchar(100)      | NO    |       | NULL                  |
                                                  |
| detail              | mediumtext        | NO    |       | NULL                  |
                                                  |
| deleted_at          | timestamp         | YES   |       | NULL                  |
                                                  |
| created_at          | timestamp         | NO    |       | CURRENT_TIMESTAMP | DE
FAULT_GENERATED                                   |
| updated_at          | timestamp         | NO    |       | CURRENT_TIMESTAMP | DE
FAULT_GENERATED on update CURRENT_TIMESTAMP |
+-----------------+-----------------+------+-----+-------------
------+-----------------------------------------------+

mysql> desc book;
+-------------+-----------------+------+-----+-----------------
--+-----------------------------------------------+
| Field         | Type              | Null | Key | Default             | Extra
                                  |
+-------------+-----------------+------+-----+-----------------
--+-----------------------------------------------+
| id            | bigint unsigned  | NO    | PRI | NULL                | auto_increme
nt                                  |
| uid           | varchar(32)       | YES   | UNI | NULL                |
                                    |
| name          | varchar(50)       | NO    | UNI | NULL                |
                                    |
| author        | varchar(32)       | NO    |       | NULL                |
```

```
                                        |
| status        | int                  | YES  |      | 1                    |
                                        |
| isPublished | tinyint unsigned | NO   |      | 1                    |
                                        |
| publishedAt | timestamp            | YES  |      | NULL                 |
                                        |
| deleted_at  | timestamp            | YES  |      | NULL                 |
                                        |
| createdAt   | timestamp            | NO   |      | CURRENT_TIMESTAMP | DEFAU
LT_GENERATED                                  |
| updatedAt   | timestamp            | NO   |      | CURRENT_TIMESTAMP | DEFAU
LT_GENERATED on update CURRENT_TIMESTAMP |
+-------------+------------------+------+-----+-----------------
--+-----------------------------------------------+
10 rows in set (0.00 sec)

mysql> desc registry;
+--------------+-----------------+------+-----+-----------------
--+-----------------------------------------------+
| Field         | Type              | Null | Key | Default             | Extra
                                        |
+--------------+-----------------+------+-----+-----------------
--+-----------------------------------------------+
| id            | bigint unsigned | NO   | PRI | NULL                | auto_increme
nt                                      |
| uid           | varchar(32)      | YES  | UNI | NULL                |
                                        |
| workspace_id | varchar(50)      | NO   |      | default             |
                                        |
| ext_id        | varchar(50)      | YES  |      | NULL                |
                                        |
```

name	varchar(50)	NO	MUL	NULL
type	varchar(50)	NO		NULL
addresses	varchar(1000)	NO		NULL
namespaces	varchar(2000)	NO		NULL
is_hosted	tinyint	NO		0
deleted_at	timestamp	YES		NULL
created_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TIMESTAMP

```
+--------------+-----------------+------+-----+-----------------
--+-----------------------------------------------+
12 rows in set (0.00 sec)
```

配置 skoala helm repo

配置好 skoala 仓库，即可查看和获取到 skoala 的应用 chart

```
~ helm repo add skoala-release https://release.daocloud.io/chartrepo/skoala
~ helm repo update
```

需要事先安装 Helm

注意：添加 Skoala-release 仓库之后，通常需要关注的有 2 个 Chart：

- Skoala 是 微服务引擎 的控制端的服务，
  - 安装完成后，可以在 DCE 5.0 平台看到微服务引擎的入口
  - 包含 3 个组件 skoala-ui、hive、sesame

    - 需要安装在全局管理集群
- Skoala-init 是 微服务引擎 所有的组件 Operator
    - 仅安装到工作集群即可
    - 包含组件有：skoala-agent, nacos, contour, sentinel
    - 未安装时，创建注册中心和网关时会提示缺少组件

默认情况下，安装完成 skoala 到 kpanda-global-cluster(全局管理集群)，就可以在侧边栏看到对应的微服务引擎的入口了。

查看微服务引擎最新版本

在全局管理集群，查看 Skoala 的最新版本，直接通过 helm 命令获取版本信息；

```
~ helm repo update skoala-release
~ helm search repo skoala-release/skoala --versions
NAME                            CHART VERSION   APP VERSION DESCRIPTION
skoala-release/skoala           0.13.0          0.13.0      The helm chart for Sko
ala
skoala-release/skoala           0.12.2          0.12.2      The helm chart for Sko
ala
skoala-release/skoala           0.12.1          0.12.1      The helm chart for Skoal
a
skoala-release/skoala           0.12.0          0.12.0      The helm chart for Sko
ala
......
```

在工作集群，查看 skoala-init 的最新版本，直接通过 helm repo 来更新获取最新的

```
~ helm repo update skoala-release
~ helm search repo skoala-release/skoala-init --versions
NAME                            CHART VERSION   APP VERSION DESCRIPTION
skoala-release/skoala-init      0.13.0          0.13.0      A Helm Chart for Skoala
```

```
 init, it includes Skoal...
skoala-release/skoala-init   0.12.2          0.12.2        A Helm Chart for Skoala
 init, it includes Skoal...
skoala-release/skoala-init   0.12.1          0.12.1        A Helm Chart for Skoala i
nit, it includes Skoal...
skoala-release/skoala-init   0.12.0          0.12.0        A Helm Chart for Skoala
 init, it includes Skoal...
......
```

执行部署（同样适用于升级）

直接执行命令即可，注意对应的版本号

```
~ helm upgrade --install skoala --create-namespace -n skoala-system --clean
up-on-fail \
    --set ui.image.tag=v0.17.0 \
    --set hive.configMap.database[0].driver="mysql" \
    --set hive.configMap.database[0].dsn="skoala:xxx@tcp(mcamel-common-mys
ql-cluster-mysql-master.mcamel-system.svc.cluster.local:3306)/skoala?charset=ut
f8&parseTime=true&loc=Local&timeout=10s" \
    skoala-release/skoala \
    --version 0.24.2
```

查看 Pod 是否启动成功

```
~ kubectl -n skoala-system get pods
NAME                                  READY   STATUS      RESTARTS
  AGE
hive-8548cd9b59-948j2                 2/2     Running     2 (3h48m ago)
3h48m
sesame-5955c878c6-jz8cd               2/2     Running     0                3
h48m
skoala-ui-7c9f5b7b67-9rpzc            2/2     Running     0                3h
48m
```

安装 skoala-init 到工作集群

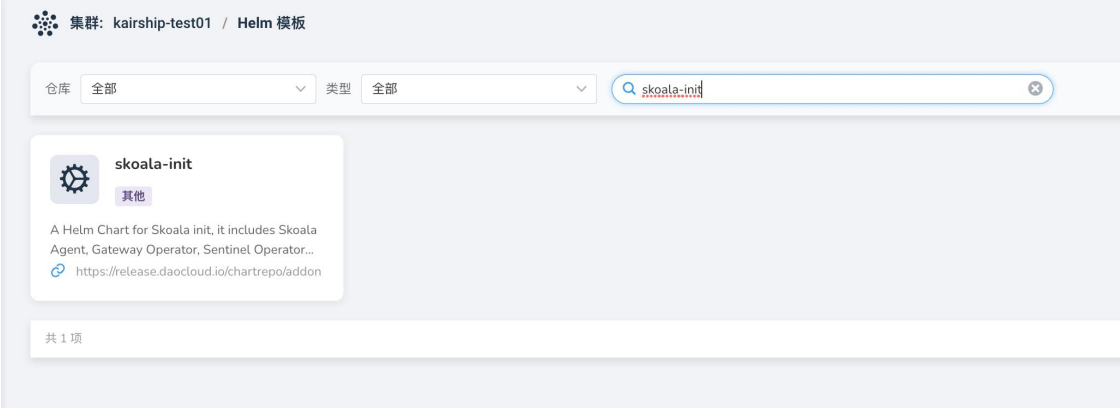
由于 Skoala 涉及的组件较多，我们将这些组件打包到同一个 Chart 内，也就是 skoala-init，所以我们应该在用到微服务引擎的工作集群安装好 skoala-init。此安装命令也可用于更新该组件。

```
~ helm search repo skoala-release/skoala-init --versions
NAME                            CHART VERSION   APP VERSION DESCRIPTION
skoala-release/skoala-init      0.24.2          0.24.2      A Helm Chart for Skoal
a init, it includes Skoal...
skoala-release/skoala-init      0.24.1          0.24.1      A Helm Chart for Skoala
 init, it includes Skoal...
skoala-release/skoala-init      0.24.0          0.24.0      A Helm Chart for Skoal
a init, it includes Skoal...
skoala-release/skoala-init      0.23.0          0.23.0      A Helm Chart for Skoal
a init, it includes Skoal...
......
```

确认所有 Pod 启动成功：

```
~ helm upgrade --install skoala-init --create-namespace -n skoala-system --c
leanup-on-fail \
    skoala-release/skoala-init \
    --version 0.24.2
```

除了通过终端安装，也可以在 容器管理->Helm 应用 内找到 skoala-init 进行安装。



image

更新微服务引擎

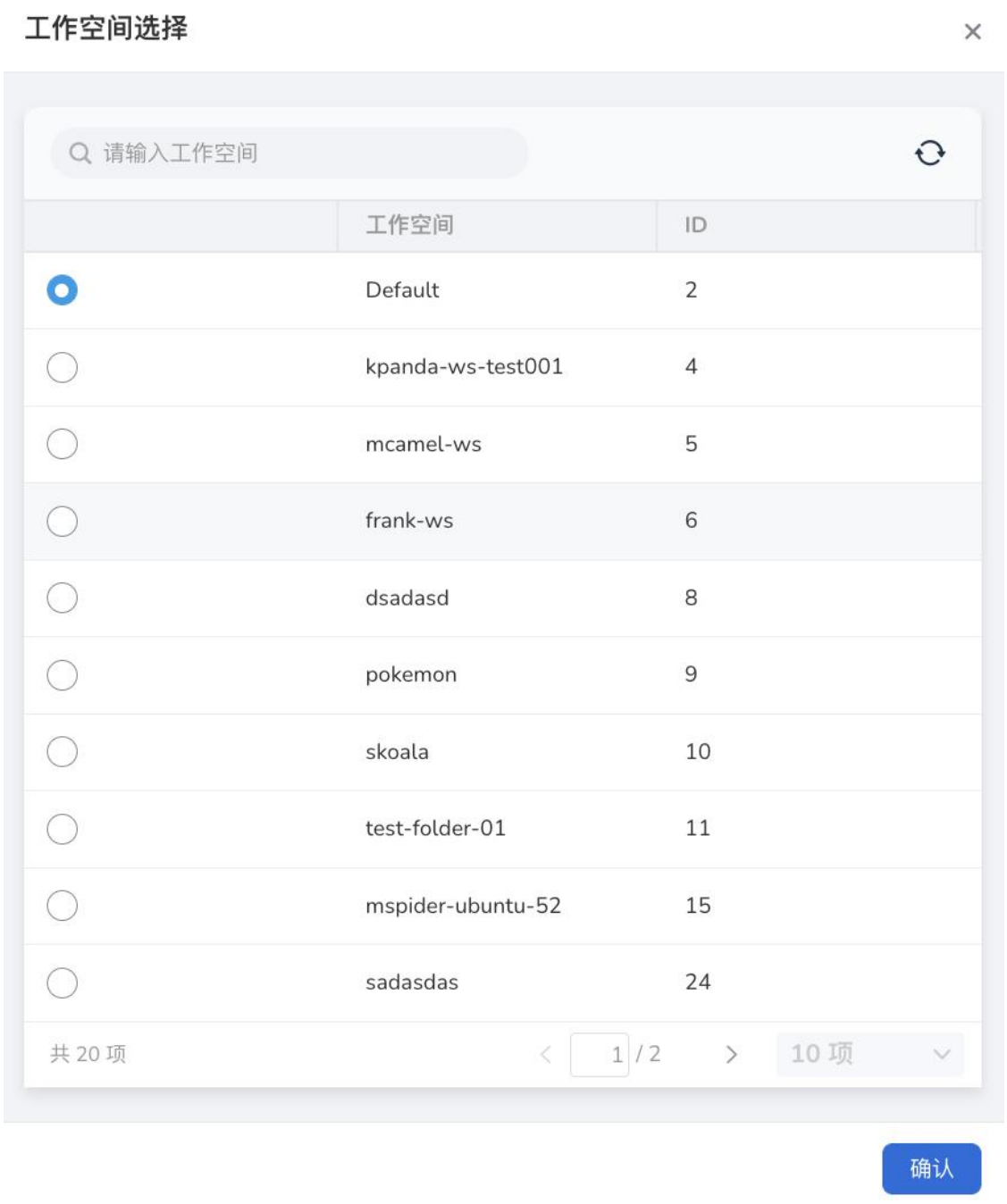
支持离线升级和在线升级两种方式，具体可参考离线升级或在线升级

卸载微服务引擎

```
~ helm uninstall skoala-init -n skoala-system

~ helm uninstall skoala -n skoala-system
```

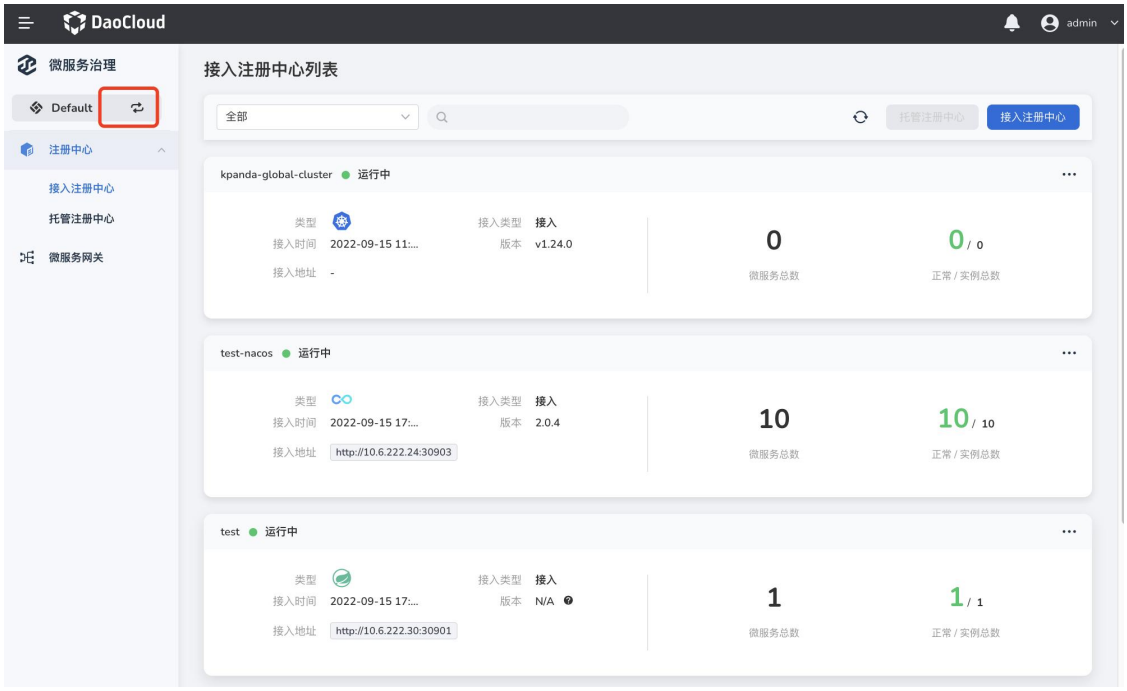
选择工作空间

首次进入微服务治理模块时，首先必须选择一个工作空间。



选择工作空间

如需更改当前所在的工作空间，可以在左侧边栏点击更换图标重新选择工作空间。



更改工作空间

!!! note

如果当前没有可选的工作空间，需要先联系超级管理员创建一个工作空间。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

离线升级微服务引擎

DCE 5.0 的各个模块松散耦合，支持独立安装、升级各个模块。此文档适用于通过离线方式安装微服务引擎之后进行的升级。

同步镜像

将镜像下载到本地节点之后，需要通过 chart-syncer 或容器运行时将最新版镜像同步到您的镜像仓库。推荐使用 chart-syncer 同步镜像，因为该方法更加高效便捷。

chart-syncer 同步镜像

1. 使用如下内容创建 load-image.yaml 作为 chart-syncer 的配置文件

   load-image.yaml 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并参考如下说明修改各项配置。有关 chart-syncer 配置文件的详细解释，可参考其官方文档。

   ===“已安装 chart repo”

   若当前环境已安装 chart repo，则可以使用如下配置直接同步镜像。

   ```yaml
   source:
     intermediateBundlesPath: skoala-offline # (1)
   target:
     containerRegistry: 10.16.23.145 # (2)
     containerRepository: release.daocloud.io/skoala # (3)
     repo:
       kind: HARBOR # (4)
       url: http://10.16.23.145/chartrepo/release.daocloud.io # (5)
       auth:
   ```

```
      username: "admin" # (6)
      password: "Harbor12345" # (7)
  containers:
    auth:
      username: "admin" # (8)
      password: "Harbor12345" # (9)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

=== “未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 `tgz` 文件并存放在指定路径。

```yaml
source:
  intermediateBundlesPath: skoala-offline # (1)
target:
  containerRegistry: 10.16.23.145 # (2)
  containerRepository: release.daocloud.io/skoala # (3)
  repo:
    kind: LOCAL
    path: ./local-repo # (4)
```

```
  containers:
    auth:
      username: "admin" # (5)
      password: "Harbor12345" # (6)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker/containerd 同步镜像

1. 解压 tar 压缩包。

```
tar xvf skoala.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```shell
docker load -i images.tar
```

=== “containerd”

```shell
ctr image import images.tar
```

!!! note – 需要在每个节点上都通过 Docker 或 containerd 加载镜像。 – 加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

开始升级

镜像同步完成之后，就可以开始升级微服务引擎了。

=== “通过 helm repo 升级”

1. 检查微服务引擎 helm 仓库是否存在。

   ```shell
   helm repo list | grep skoala
   ```

   若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

   ```none
   Error: no repositories to show
   ```

2. 添加微服务引擎的 helm 仓库。

   ```shell
   helm repo add skoala-release http://{harbor url}/chartrepo/{project}
   ```

3. 更新微服务引擎的 helm 仓库。

```shell
helm repo update skoala-release # (1)
```

1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

4. 选择您想安装的微服务引擎版本（建议安装最新版本）。

```shell
helm search repo skoala-release/skoala --versions
```

```none
[root@master ~]# helm search repo skoala-release/skoala --versions
NAME                        CHART VERSION   APP VERSION   DESCRIPTION
skoala-release/skoala   0.14.0          v0.14.0       A Helm chart for Skoal
a
...
```

5. 备份 `--set` 参数。

在升级微服务引擎版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values skoala -n skoala-system -o yaml > bak.yaml
```

6. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade skoala skoala-release/skoala \
-n skoala-system \
-f ./bak.yaml \
--set global.imageRegistry=$imageRegistry
--version 0.14.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

在升级微服务引擎版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values skoala -n skoala-system -o yaml > bak.yaml
```

2. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```

```shell
helm upgrade skoala . \
-n skoala-system \
-f ./bak.yaml \
--set global.imageRegistry=$imageRegistry
```
```

服务接入 Nacos SDK

本文介绍不同框架的传统微服务如何接入 Nacos 的原生 SDK。

JAVA（无框架）

1. 在 pom.xml 文件中添加依赖项。目前的最新版本为 2.2.2

```
<dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-client</artifactId>
    <version>${latest.version}</version>
</dependency>
```

2. 在服务中添加服务注册和服务发现的代码：

```
//添加配置变量 serverAddr: Nacos 的地址，e.g,: 192.168.0.0:8848. namespace: Nacos 中的命名空间
Properties properties = new Properties();
properties.setProperty("serverAddr", System.getProperty("serverAddr"));
properties.setProperty("namespace", System.getProperty("namespace"));

NamingService naming = NamingFactory.createNamingService(properties);
//注册实例: 注册时带上服务的 IP 和端口
naming.registerInstance("sentinel-demo", "11.11.11.11", 8888, "DEFAULT");

System.out.println(naming.getAllInstances("sentinel-demo"));

naming.deregisterInstance("sentinel-demo", "11.11.11.11", 8888, "DEFAULT");

System.out.println(naming.getAllInstances("sentinel-demo"));
//添加对服务的订阅，在变更时获取事件通知
```

```
naming.subscribe("sentinel-demo", new EventListener() {
    @Override
    public void onEvent(Event event) {
        System.out.println(((NamingEvent)event).getServiceName());
        System.out.println(((NamingEvent)event).getInstances());
        }
    });
```

3. 如需添加 Nacos 的更多特性，可参考更多使用方式

JAVA (SpringBoot) 框架

1. 在 pom.xml 文件中添加依赖项

```
<dependency>
    <groupId>com.alibaba.boot</groupId>
    <artifactId>nacos-config-spring-boot-starter</artifactId>
    <version>${latest.version}</version>
</dependency>
```

!!! note

- 版本 [0.2.x.RELEASE](https://mvnrepository.com/artifact/com.alibaba.boot/nacos-config-spring-boot-starter) 对应的是 Spring Boot 2.x 版本。
- 版本 [0.1.x.RELEASE](https://mvnrepository.com/artifact/com.alibaba.boot/nacos-config-spring-boot-starter) 对应的是 Spring Boot 1.x 版本。

2. 在项目中添加 bootstrap.yaml 配置文件

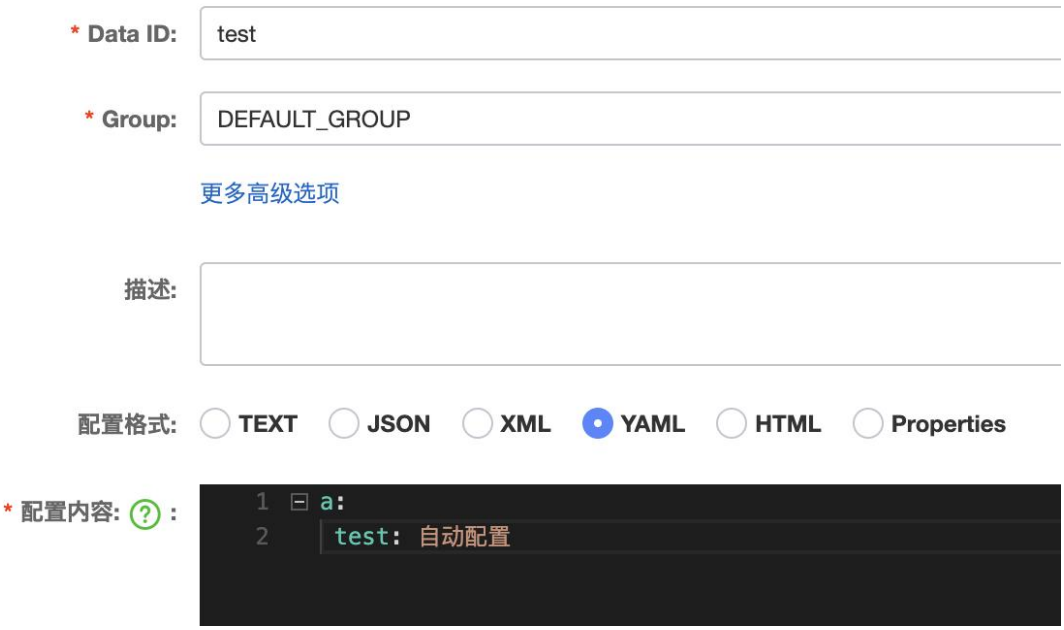
```
spring:
  application:
    name: demo
nacos:
  config:
    data-id: test # Nacos 配置的 data-id
    server-addr: 127.0.0.1:8848 # Nacos 服务器地址
```

```
    group: DEFAULT_GROUP # 配置文件 Group
    namespace: public # 命名空间 ID
    type: yaml # Nacos 配置文件类型
    auto-refresh: true # 是否启用动态刷新配置
  discovery:
    server-addr: 127.0.0.1:8848 # Nacos 服务器地址
    group: DEFAULT_GROUP # 注册应用的 Group
    namespace: public # Nacos 的命名空间
```

3. 服务注册功能无需改动代码，直接启动项目就能在服务列表看到启动的服务。

4. 添加服务配置的代码

    1. 登录 Nacos 控制台添加配置文件。



screenshot

    2. 然后在控制器代码中添加如下代码：

```java
@RestController
@RequestMapping("config")
public class Controller {
    @NacosValue(value = "${a.test}", autoRefreshed = true)
    private String name;

    @GetMapping("get")
    public String get() {
        return this.name;
    }
}
```

5. 如需添加 Nacos 的更多特性，可参考更多使用方式

JAVA (SpringCloud) 框架

1. 在 pom.xml 文件中添加依赖项。

   版本对应关系参考：版本说明

```xml
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
    <version>${latest.version}</version>
</dependency>

<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    <version>${latest.version}</version>
</dependency>
```

2. 在项目中添加 bootstrap.yaml 配置文件

```yaml
spring:
  application:
    name: demo
---
spring:
  cloud:
    nacos:
      config:
        enabled: true
        server-addr: 127.0.0.1:8848 # nacos 服务器地址
        name: test # nacos 配置的 data-id
        group: DEFAULT_GROUP # 配置文件 Group
        namespace: public # 命名空间 ID
        file-extension: yaml # 配置文件后缀
      discovery:
        enabled: true
        server-addr: 127.0.0.1:8848 # nacos 服务器地址
        namespace: public # 命名空间 ID
        group: DEFAULT_GROUP # 应用分组名
```

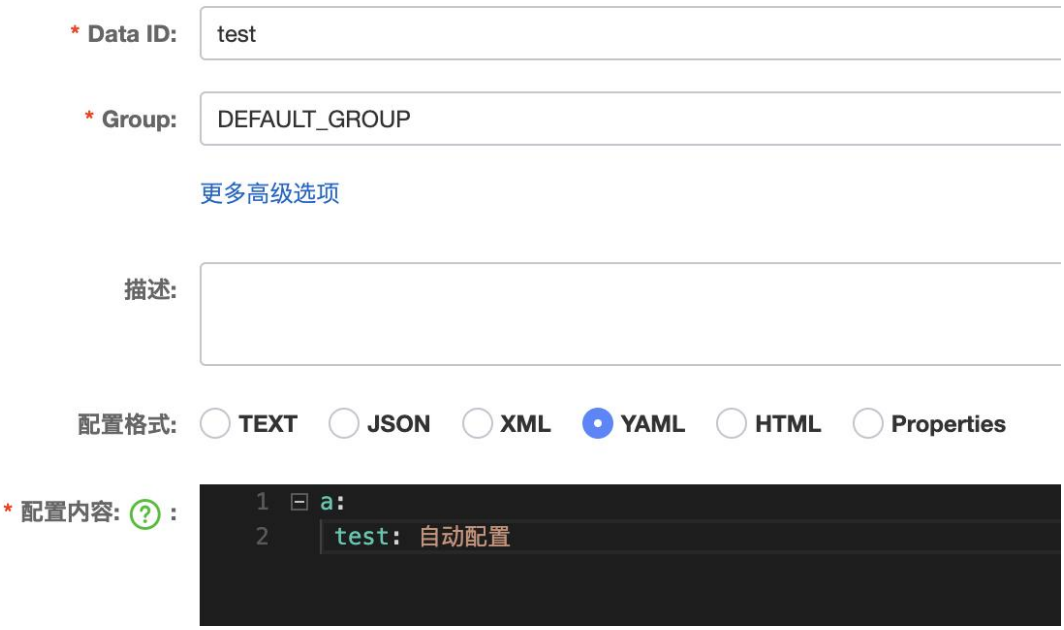
3. 在服务添加 Nacos 服务注册的代码

   在启动类上添加 @EnableDiscoveryClient 注解开启服务注册

```java
@SpringBootApplication
@EnableDiscoveryClient
public class NacosProviderApplication {

  public static void main(String[] args) {
    SpringApplication.run(NacosProviderApplication.class, args);
  }
}
```

4. 添加动态配置的代码

    1. 在 Nacos 控制台添加配置文件



screenshot

    2. 在控制器中添加 @RefreshScope 和 @Value 注解

    通过 springcloud 的 @RefreshScope 注解可以实现自动配置

```
@RestController
@RequestMapping("config")
@RefreshScope
public class Controller {
    @Value(value = "${a.test}")
    private String name;

    @GetMapping("get")
    public String get() {
        return this.name;
```

```
        }
    }
```

5. 如需添加 Nacos 的更多特性，可参考更多使用方式

Go 框架

在 Go 框架微服务中添加 Nacos SDK 时，需要满足以下两个前提条件：

- Go 版本 1.15 以上
- Nacos 版本 2.x 以上

具体操作步骤如下：

1. 获取依赖

```
$ go get -u github.com/nacos-group/nacos-sdk-go/v2
```

2. 在服务中添加服务注册的代码

```
//create ServerConfig,配置 Nacos 服务器的地址
sc := []constant.ServerConfig{
  *constant.NewServerConfig("127.0.0.1", 8848, constant.WithContextPath("
/nacos")),
}

//create ClientConfig, 配置客户端的链接配置
cc := *constant.NewClientConfig(
  constant.WithNamespaceId("public"), //命名空间 ID
  constant.WithTimeoutMs(5000), // 超时时间
  constant.WithLogDir("/tmp/nacos/log"), //日志地址
  constant.WithCacheDir("/tmp/nacos/cache"), // Nacos 服务缓存的地址
  constant.WithLogLevel("debug"), // 日志级别
)

// 通过 ServerConfig 和 ClientConfig 创建 nacosclient 链接
```

```
client, err := clients.NewNamingClient(
  vo.NacosClientParam{
    ClientConfig:  &cc,
    ServerConfigs: sc,
  },
)

//Register 注册服务
registerServiceInstance(client, vo.RegisterInstanceParam{
  Ip:          "10.0.0.10",
  Port:        8848,
  ServiceName: "demo.go",
  GroupName:   "group-a",
  ClusterName: "cluster-a",
  Weight:      10,
  Enable:      true,
  Healthy:     true,
  Ephemeral:   true,
  Metadata:    map[string]string{"idc": "shanghai"},
})

//DeRegister 注销服务
deRegisterServiceInstance(client, vo.DeregisterInstanceParam{
  Ip:          "10.0.0.10",
  Port:        8848,
  ServiceName: "demo.go",
  GroupName:   "group-a",
  Cluster:     "cluster-a",
  Ephemeral:   true, //must be true
})
```

3. 配置加载

```
//create ServerConfig,配置 Nacos 服务器的地址
sc := []constant.ServerConfig{
  *constant.NewServerConfig("127.0.0.1", 8848, constant.WithContextPath("
/nacos")),
}

//create ClientConfig, 配置客户端的链接配置
cc := *constant.NewClientConfig(
  constant.WithNamespaceId("public"), //命名空间 ID
  constant.WithTimeoutMs(5000), // 超时时间
  constant.WithLogDir("/tmp/nacos/log"), //日志地址
  constant.WithCacheDir("/tmp/nacos/cache"), // Nacos 服务缓存的地址
  constant.WithLogLevel("debug"), // 日志级别
)

// 通过 ServerConfig 和 ClientConfig 创建 nacosclient 链接
client, err := clients.NewNamingClient(
  vo.NacosClientParam{
    ClientConfig:  &cc,
    ServerConfigs: sc,
  },
)

//get config 获取配置
content, err := client.GetConfig(vo.ConfigParam{
  DataId: "test-data",
  Group:  "test-group",
})
fmt.Println("GetConfig,config :" + content)

//Listen config change,key=dataId+group+namespaceId. 监听配置改变
```

```
err = client.ListenConfig(vo.ConfigParam{
    DataId: "test-data",
    Group:  "test-group",
    OnChange: func(namespace, group, dataId, data string) {
        fmt.Println("config changed group:" + group + ", dataId:" + dataId +
", content:" + data)
    },
})
```

4. 如需添加 Nacos 的更多特性，可参考更多使用方式

Python 框架

1. 获取依赖

```
pip install nacos-sdk-python
```

2. 代码接入

```
import nacos

# Nacos server 地址
SERVER_ADDRESSES = "127.0.0.1:8848"
# 命名空间 ID
NAMESPACE = "public"

client = nacos.NacosClient(SERVER_ADDRESSES, namespace=NAMESPAC
E)

# 注册服务
client.add_naming_instance("test.service1", "1.0.0.7", 8080, "testCluster2",
0.2, "{}", False, True)
# 发送心跳
client.send_heartbeat("test.service", "1.0.0.7", 8080, "testCluster2", 0.1, "{}")
```

```
# get config
data_id = "dev-config"
group = "DEFAULT_GROUP"

# 全局服务配置
server_config = json.loads(client.get_config(data_id, group))

# 当服务配置发生变化
def config_update(data):
    global server_config
    server_config = json.loads(data['content'])
    print('new data->', server_config)

# 监听服务配置变化
client.add_config_watcher(data_id, group, config_update)
```

3. 如需添加 Nacos 的更多特性，可参考更多使用方式

Node.js 框架

1. 添加依赖

   需要使用 2.x 以上的版本

   ```
   npm install nacos --save
   ```

2. 添加服务发现的代码逻辑

```
'use strict';

const NacosNamingClient = require('nacos').NacosNamingClient;
const logger = console;

const client = new NacosNamingClient({
```

```
  logger,
  serverList: '127.0.0.1:8848', // Nacos 服务地址
  namespace: 'public', //命名空间 ID
});
await client.ready();

const serviceName = 'nodejs.test.domain';

// 注册服务
await client.registerInstance(serviceName, {
  ip: '1.1.1.1',
  port: 8080,
});
await client.registerInstance(serviceName, {
  ip: '2.2.2.2',
  port: 8080,
});

// 订阅服务
client.subscribe(serviceName, hosts => {
  console.log(hosts);
});

// 注销服务
await client.deregisterInstance(serviceName, {
  ip: '1.1.1.1',
  port: 8080,
});
```

3. 添加动态配置的代码逻辑

```
import {NacosConfigClient} from 'nacos';    // ts
const NacosConfigClient = require('nacos').NacosConfigClient; // js


const configClient = new NacosConfigClient({
  serverAddr: '127.0.0.1:8848',
  namespace: 'public', //命名空间 ID
});

// 获取配置文件
const content= await configClient.getConfig('test', 'DEFAULT_GROUP'); //
dataID: test,group: DEFAULT_GROUP
console.log('getConfig = ',content);

// 监听配置变化
configClient.subscribe({
  dataId: 'test',
  group: 'DEFAULT_GROUP',
}, content => {
  console.log(content);
});
```

4. 如需添加 Nacos 的更多特性，可参考更多使用方式

C++ 框架

1. 下载依赖

   下载工程源代码并执行下述命令:

   ```
   cd nacos-sdk-cpp
   cmake .
   make
   ```

   执行命令后会产生一个 libnacos-cli.so 和一个 nacos-cli.out 文件

2. 运行 make install 将 libnacos-cli 安装到 lib 目录

3. 在服务代码中添加服务注册的代码逻辑

```
#include <iostream>
#include <unistd.h>
#include "Nacos.h"

using namespace std;
using namespace nacos;

int main() {
    Properties configProps;
    configProps[PropertyKeyConst::SERVER_ADDR] = "127.0.0.1"; // nacos 服务地址
    configProps[PropertyKeyConst::NAMESPACE] = "public"; // 命名空间 ID
    NacosServiceFactory *factory = new NacosServiceFactory(configProps);
    ResourceGuard <NacosServiceFactory> _guardFactory(factory);
    NamingService *namingSvc = factory->CreateNamingService();
    ResourceGuard <NamingService> _serviceFactory(namingSvc);
    Instance instance;
    instance.clusterName = "DEFAULT";
    instance.ip = "127.0.0.1";
    instance.port = 2333;
    instance.instanceId = "1";
    instance.ephemeral = true;

    //模拟注册 5 个服务
    try {
        for (int i = 0; i < 5; i++) {
```

```
            NacosString serviceName = "TestNamingService" + NacosStri
ngOps::valueOf(i);
            instance.port = 2000 + i;
            namingSvc->registerInstance(serviceName, instance);
        }
    }
    catch (NacosException &e) {
        cout << "encounter exception while registering service instance, r
aison:" << e.what() << endl;
        return -1;
    }
    sleep(30);
    //注销服务
    try {
        for (int i = 0; i < 5; i++) {
            NacosString serviceName = "TestNamingService" + NacosStri
ngOps::valueOf(i);

            namingSvc->deregisterInstance(serviceName, "127.0.0.1", 2000
 + i);
            sleep(1);
        }
    }
    catch (NacosException &e) {
        cout << "encounter exception while registering service instance, r
aison:" << e.what() << endl;
        return -1;
    }
    sleep(30);

    return 0;
}
```

4. 添加动态配置的代码逻辑

```
#include <iostream>
#include "Nacos.h"

using namespace std;
using namespace nacos;

class MyListener : public Listener {
private:
    int num;
public:
    MyListener(int num) {
        this->num = num;
    }

    void receiveConfigInfo(const NacosString &configInfo) {
        cout << "==========" << endl;
        cout << "Watcher" << num << endl;
        cout << "Watched Key UPDATED:" << configInfo << endl;
        cout << "==========" << endl;
    }
};

int main() {
    Properties props;
    props[PropertyKeyConst::SERVER_ADDR] = "127.0.0.1:8848"; //nacos
地址
    props[PropertyKeyConst::NAMESPACE] = "public"; // 命名空间 ID
    NacosServiceFactory *factory = new NacosServiceFactory(props);
    ResourceGuard <NacosServiceFactory> _guardFactory(factory);
    ConfigService *n = factory->CreateConfigService();
```

```
    ResourceGuard <ConfigService> _serviceFactory(n);

    MyListener *theListener = new MyListener(1);//You don't need to free it, since it will be deleted by the function removeListener
    n->addListener("dqid", "DEFAULT_GROUP", theListener);//dataID为"dqid"并且group为"DEFAULT_GROUP"的配置改变都将监听到

    cout << "Input a character to continue" << endl;
    getchar();
    cout << "remove listener" << endl;
    n->removeListener("dqid", NULLSTR, theListener);//取消监听
    getchar();

    return 0;
}
```

5. 如需添加 Nacos 的更多特性，可参考更多使用方式

服务接入 Sentinel 规范

为了正常使用 DCE 5.0 微服务引擎提供的 Sentinel 流量治理和查看 Sentinel 数据监控，需要将应用接入 Sentinel 控制台，并且传递应用参数时需要满足一定规范。

微服务中引入 Sentinel 相关 SDK

pom.xml 文件中加入相关引用，常见的 SDK 如下：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>

<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
</dependency>

<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-core</artifactId>
    <version>2.0.0-alpha</version>
</dependency>
```

bootstrap.yml 配置文件中注意如下设置

project.name 参数的格式应为：{{nacos_namespace_id}}@@{{nacos_group}}@@{{appName}}。

```
---
project:
```

```
  # 服务注册在 sentinel 中的名称，建议与 nacos 注册服务名相同
  name: ${spring.cloud.nacos.discovery.namespace}@@${spring.cloud.nacos.discovery.group}@@${spring.application.name}
```

需要注意的是：

- 符合此规范时，Sentinel 的治理规则会被推送到对应命名空间下，对应配置分组下的配置中心。
- 如果不符合此规范，比如只有 appName 或者 {{nacos_group}}@@{{appName}}，则所有治理规则都会被推送到 public 命名空间下的 SENTINEL_GROUP 配置中心。
- 第一部分 {{nacos_namespace_id}} 指的是 Nacos 命名空间的 **ID**，而非命名空间的名称。
- Nacos 的 public 命名空间对应的 ID 是空字符“”。
- 如果想把应用接入 public 命名空间，必须使用空字符串，例如 @@A@@appA。

Sentinel 接入配置

```
---
spring:
  cloud:
    sentinel:
      enabled: false
      # 是否开启预加载，设置为 true 时实例启动后自动在 sentinel dashboard 中展示，设置为 false，需要实例有流量后该实例才会出现在 sentinel dashboard 中
      eager: true
      transport:
        # 配置 sentinel dashboard 地址
        dashboard: 10.6.222.24:31165
      # 以下配置为规则存放在 nacos 配置中心的相关参数
      datasource:
```

```yaml
flow:
  nacos:
    server-addr: ${spring.cloud.nacos.config.server-addr}
    dataId: ${spring.application.name}-flow-rules
    groupId: ${spring.cloud.nacos.discovery.group}
    namespace: ${spring.cloud.nacos.discovery.namespace}
    ruleType: flow
degrade:
  nacos:
    server-addr: ${spring.cloud.nacos.config.server-addr}
    dataId: ${spring.application.name}-degrade-rules
    groupId: ${spring.cloud.nacos.discovery.group}
    namespace: ${spring.cloud.nacos.discovery.namespace}
    rule-type: degrade
system:
  nacos:
    server-addr: ${spring.cloud.nacos.config.server-addr}
    dataId: ${spring.application.name}-system-rules
    groupId: ${spring.cloud.nacos.discovery.group}
    namespace: ${spring.cloud.nacos.discovery.namespace}
    rule-type: system
authority:
  nacos:
    server-addr: ${spring.cloud.nacos.config.server-addr}
    dataId: ${spring.application.name}-authority-rules
    groupId: ${spring.cloud.nacos.discovery.group}
    namespace: ${spring.cloud.nacos.discovery.namespace}
    rule-type: authority
param-flow:
  nacos:
    server-addr: ${spring.cloud.nacos.config.server-addr}
    dataId: ${spring.application.name}-param-flow-rules
```

```
            groupId: ${spring.cloud.nacos.discovery.group}
            namespace: ${spring.cloud.nacos.discovery.namespace}
            rule-type: param-flow
```

!!! note

为了正常显示 Sentinel 监控数据，应使用 Sentinel 官方 SDK v1.8.6 **以上** 的版本。如未能显示监控数据，可查看 Sentinel 的版本是否符合要求。

服务接入 Sentinel 监控

本文介绍如何为传统观念微服务接入 Sentinel 监控功能。

1. 添加依赖项

   sentinel metric exporter SDK 的版本需要 >= v2.0.0-alpha

   ```
   <dependency>
     <groupId>com.alibaba.csp</groupId>
     <artifactId>sentinel-metric-exporter</artifactId>
     <version>v2.0.0-alpha</version>
   </dependency>
   ```

   如需了解相关原因，可参考：
   https://github.com/alibaba/Sentinel/pull/2976

2. 启动服务时添加 javaagent 参数，且 JMX 端口固定为 12345

   -javaagent:/jmx_prometheus_javaagent-0.17.0.jar=12345:/prometheus-jmx-config.yaml

   有关 JMX 的详细说明，可参考使用 JMX Exporter 暴露 JVM 监控指标。

3. 为服务创建 Kubernetes Service。重点包括以下参数：

   - labels 字段：固定为 skoala.io/type: sentinel
   - ports 字段：固定为 name: jmx-metrics，port: 12345，targetPort: 12345

     ```
     apiVersion: v1
     kind: Service
     ```

```yaml
    metadata:
      labels:
        skoala.io/type: sentinel
      name: sentinel-demo
      namespace: skoala-jia
    spec:
      ports:
      - name: jmx-metrics
        port: 12345
        protocol: TCP
        targetPort: 12345
      selector:
        app.kubernetes.io/name: sentinel-demo
```

!!! note “如需了解相关原因，可参考 ServiceMonitor CR 的定义”

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    release: insight-agent
    operator.insight.io/managed-by: insight
  name: sentinel-service-monitor
spec:
  endpoints:
    - port: jmx-metrics
      scheme: http
  jobLabel: jobLabel
  namespaceSelector:
    any: true
  selector:
    matchLabels:
```

```
        skoala.io/type: sentinel
```

服务接入 Sentinel 集群流控

为服务接入 Sentinel 集群流控时，需要从服务器和客户端两方面调整配置。

集群流控服务器

DCE 5.0 不提供托管的 Sentinel 服务器，所以用户需自己启动 Sentinel 官方提供的集群流控服务器，且该服务器需要满足以下条件：

- 连接 Nacos，以便从 Nacos 读取持久化配置
- 为 Sentinel 服务器命名，以便作为配置前缀来区分多个集群流控服务器的 NamespaceSet 和 ServerTransport 配置

```
ReadableDataSource<String, Set<String>> namespaceDs = new NacosDataSource<>(nacosAddress, DEFAULT_GROUP,
                    serverName + NAMESPACESET_POSTFIX,
                    source -> JSON.parseObject(source, new TypeReference<Set<String>>() {}));
        ClusterServerConfigManager.registerNamespaceSetProperty(namespaceDs.getProperty());
        ReadableDataSource<String, ServerTransportConfig> transportConfigDs = new NacosDataSource<>(nacosAddress, DEFAULT_GROUP,
                    serverName + SERVERTRANSPORT_POSTFIX,
                    source -> JSON.parseObject(source, new TypeReference<ServerTransportConfig>() {}));
        ClusterServerConfigManager.registerServerTransportProperty(transportConfigDs.getProperty());
```

- 相关的代码实现，可参考 sentinel-cluster-flow-control-java

集群流控客户端

由于 Sentinel 官方提供的客户端 sentinel-cluster-client-default SDK 只适配内存模式使用。而 DCE 5.0 也为集群流控客户端配置做了持久化，所以官方原始客户端 SDK 已无效。

1. 首先，用户需引入如下代码：

```
public class SentinelClusterClientInitFunc implements InitFunc {

    private String nacosAddress;
    private Properties properties = new Properties();
    private String groupId;
    private String dataId;

    @Override
    public void init() throws Exception {
        getNacosAddr();

        parseAppName();

        initDynamicRuleProperty();

        initClientConfigProperty();

        initClientServerAssignProperty();

        initStateProperty();
    }

    private void getNacosAddr() {
        nacosAddress = System.getProperty("nacos.address");
```

```
        if (StringUtil.isBlank(nacosAddress)){
            throw new RuntimeException("nacos address start param must be set");
        }
        System.out.printf("nacos address: %s\n", nacosAddress);
    }

    private void parseAppName() {
        String[] apps = APPNAME.split("@@");
        System.out.printf("app name: %s\n", APPNAME);
        if (apps.length != 3) {
            throw new RuntimeException("app name format must be set like this: {{namespaceId}}@@{{groupName}}@@{{appName}}");
        } else if (StringUtil.isBlank(apps[1])){
            throw new RuntimeException("group name cannot be empty");
        }
        properties.put(PropertyKeyConst.NAMESPACE, apps[0]);
        properties.put(PropertyKeyConst.SERVER_ADDR, nacosAddress);
        properties.put(PropertyKeyConst.USERNAME, DEFAULT_NACOS_USERNAME);
        properties.put(PropertyKeyConst.PASSWORD, DEFAULT_NACOS_PASSWORD);
        groupId = apps[1];
        dataId = apps[2];
    }

    private void initDynamicRuleProperty() {
        ReadableDataSource<String, List<FlowRule>> ruleSource = new NacosDataSource<>(properties, groupId,
            dataId + FLOW_POSTFIX, source -> JSON.parseObject(source, new TypeReference<List<FlowRule>>() {}));
        FlowRuleManager.register2Property(ruleSource.getProperty());
```

```
        ReadableDataSource<String, List<ParamFlowRule>> paramRuleSour
ce = new NacosDataSource<>(properties, groupId,
                dataId +  PARAM_FLOW_POSTFIX, source -> JSON.parse
Object(source, new TypeReference<List<ParamFlowRule>>() {}));
        ParamFlowRuleManager.register2Property(paramRuleSource.getProp
erty());
    }

    private void initClientConfigProperty() {
        ReadableDataSource<String, ClusterClientConfig> clientConfigDs =
new NacosDataSource<>(properties, groupId,
                dataId + CLUSTER_CLIENT_POSTFIX, source -> JSON.par
seObject(source, new TypeReference<ClusterClientConfig>() {}));
        ClusterClientConfigManager.registerClientConfigProperty(clientConfi
gDs.getProperty());
    }

    private void initClientServerAssignProperty() {
//        Cluster map format:
//        [
//            {
//                "machineId": "10.64.0.81@8720",
//                 "ip": "10.64.0.81",
//                 "port": 18730,
//                 "clientSet": ["10.64.0.81@8721", "10.64.0.81@8722"]
//            }
//        ]
        ReadableDataSource<String, ClusterClientAssignConfig> clientAssig
nDs = new NacosDataSource<>(properties, groupId,
                dataId +  CLUSTER_MAP_POSTFIX, source -> {
            List<ClusterGroupDto> groupList = JSON.parseObject(source,
```

```
new TypeReference<List<ClusterGroupDto>>() {});
                return Optional.ofNullable(groupList)
                    .flatMap(this::extractClientAssignment)
                    .orElse(null);
            });
            ClusterClientConfigManager.registerServerAssignProperty(clientAssig
nDs.getProperty());
    }

    private void initStateProperty() {
        ReadableDataSource<String, Integer> clusterModeDs = new Nacos
DataSource<>(properties, groupId,
                    dataId + CLUSTER_MAP_POSTFIX, source -> {
                List<ClusterGroupDto> groupList = JSON.parseObject(source,
new TypeReference<List<ClusterGroupDto>>() {});
                return Optional.ofNullable(groupList)
                    .map(this::extractMode)
                    .orElse(ClusterStateManager.CLUSTER_NOT_STARTED);
            });
            ClusterStateManager.registerProperty(clusterModeDs.getProperty());
    }

    private int extractMode(List<ClusterGroupDto> groupList) {
        if (groupList.stream().anyMatch(this::machineEqual)) {
            return ClusterStateManager.CLUSTER_SERVER;
        }

        boolean canBeClient = groupList.stream()
            .flatMap(e -> e.getClientSet().stream())
            .filter(Objects::nonNull)
            .anyMatch(e -> e.equals(getCurrentMachineId()));
        return canBeClient ? ClusterStateManager.CLUSTER_CLIENT : Clus
```

```
terStateManager.CLUSTER_NOT_STARTED;
    }

    private Optional<ClusterClientAssignConfig> extractClientAssignment(Lis
t<ClusterGroupDto> groupList) {
        if (groupList.stream().anyMatch(this::machineEqual)) {
            return Optional.empty();
        }
        for (ClusterGroupDto group : groupList) {
            if (group.getClientSet().contains(getCurrentMachineId())) {
                String ip = group.getIp();
                Integer port = group.getPort();
                return Optional.of(new ClusterClientAssignConfig(ip, port));
            }
        }
        return Optional.empty();
    }

    private boolean machineEqual(ClusterGroupDto group) {
        return getCurrentMachineId().equals(group.getMachineId());
    }

    private String getCurrentMachineId() {
        return HostNameUtil.getIp() + SEPARATOR + TransportConfig.getR
untimePort();
    }

    private static final String SEPARATOR = "@";
}
```

2. 接着在 Spring 容器中通过如下方式加载：

```
@PostConstruct
public void initSentinelClusterFlow() throws Exception{
    new SentinelClusterClientInitFunc().init();
}
```

网关对接服务网格

本文介绍如何对接 DCE 5.0 云原生网关和服务网格，使得可以通过网关访问服务网格下接入的服务，并且可以正常使用网格的所有流量治理能力，例如虚拟服务、目标规则等。

现状

目前 DCE 5.0 云原生网关采用 contour 作为控制面，无法同步 Istio 的策略。通过云原生网关访问网格服务时，网关 API 直接连接到网格服务，由于没有同步 Istio 的策略，所以无法应用虚拟服务、目标服务等规则，导致网格能力缺失。

对接思路

让云原生网关通过 Istio 的 Sidecar 访问网格服务，从而应用 Istio 的规则。

操作步骤

1. 创建云原生网关时，开启注入 Sidecar。

2. 为 Pod 设置 Annotation 注解 traffic.sidecar.istio.io/includeInboundPorts: ""。

   这样可以不让 Istio 处理入口流量，减少性能损耗。

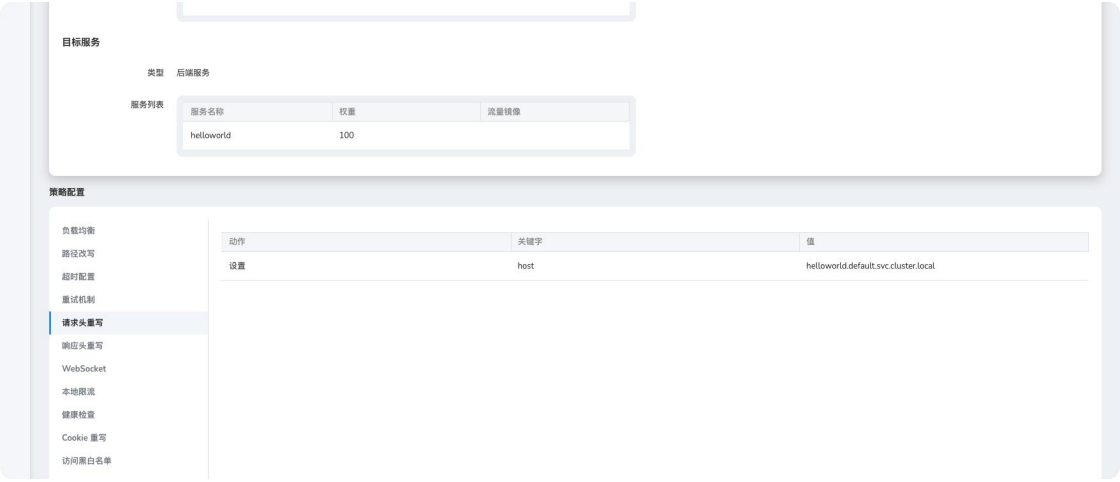
3. 在 Envoy 的启动参数中增加 --base-id 10，允许同一个命名空间中存在两个 envoy 实例，防止冲突。

```
    name: envoy-admin
  - mountPath: /tmp
    name: envoy-access-log
- args:
  - -c
  - /config/envoy.json
  - --service-cluster $(CONTOUR_NAMESPACE)
  - --service-node $(ENVOY_POD_NAME)
  - --log-level info
  - --base-id 10
  command:
  - envoy
  env:
  - name: CONTOUR_NAMESPACE
```

参数示意图

4. 在网关下添加 API 时，设置 Host 请求头。

   这样可以让网关在多集群场景下，通过 Sidecar 访问其他集群的服务，实现服务的跨集群负载均衡。

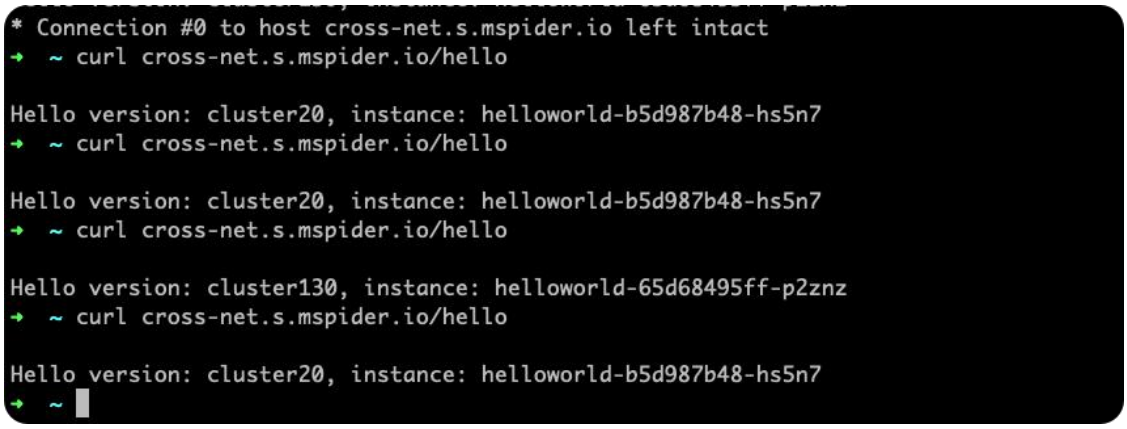


参数示意图

5. 在服务网格模块为网关 Sidecar 配置资源，不低于云原生网关的默认资源配置 1 核 1 G。

效果展示

完成上述配置之后，通过网关域名访问服务。可以看到每次请求的是不同集群中的服务，说明网关可以跨集群访问服务。



参数示意图

微服务网关的 JWT 功能

微服务引擎网关支持 JWT 验证。下面介绍如何使用此功能。

前提条件

- 创建一个集群或接入一个集群
- 创建一个网关
- 准备一个 Token 和用于验证 Token 的 JWKS 应用。如果尚且没有 JWKS 应用，可以参考创建 JWKS 应用创建一个。

操作步骤

1. 参考创建域名创建协议为 https 的域名，在域名的安全策略中启用 JWT 认证。
    - JWKS 名称：唯一的 JWKS 的名称，用于标识具体的 JWT 策略，必填
    - JWKS 服务器地址：返回 JWKS 内容的 JWT 服务的完整 FQDN 地址，必填
    - Token 透传：是否将 JWT 的 Token 信息发送到后端服务
    - Issuer：Token 颁发者认证，不填则不进行校验
    - Audiences：Token 的受众，不填则不进行校验
    - Token 缓存时长：JWKS 内存的缓存时间，在缓存有效期内不会重复请求 JWKS 服务器地址
    - 验证超时时长：JWKS 服务器的响应超时时间，超过超时时间获取 JWKS 失败



基础配置

2. 参考添加 API 创建 API 并启用 JWT 认证安全策略。



基础配置

3. 带上 Token 访问验证，如果访问成功说明 JWT 策略配置成功



基础配置

创建 JWKS 应用

如果当前环境中尚没有 JWKS 应用，可以参考以下流程部署一个应用。

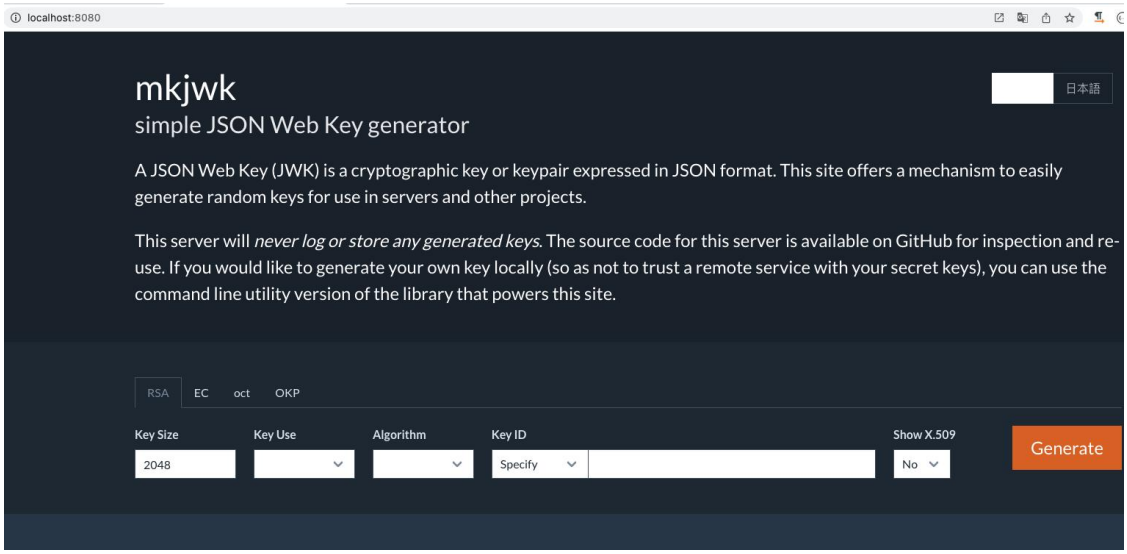
1. 将 JWKS 生成器代码下载到本地。

   git clone https://github.com/projectsesame/jwks-generator

2. 在本地运行 JWKS 生成器。

   mvn package –DskipTests && java –jar target/ROOT.war

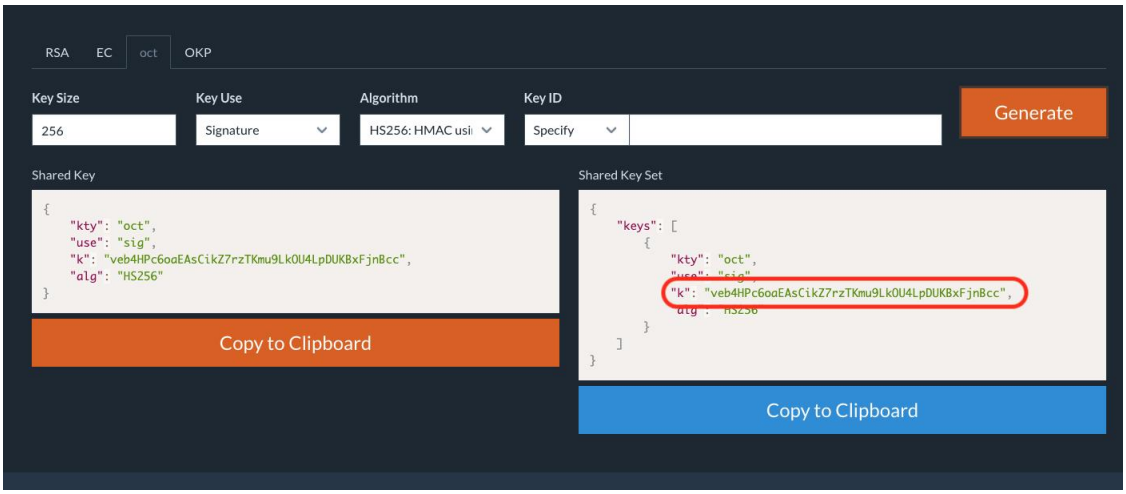
   访问 http://localhost:8080，如果出现下方界面说明 JWKS 生成器已经在本地成功运行起来。



   基础配置

3. 参考下方说明填写信息，点击 Generate 生成 JWKS 内容。

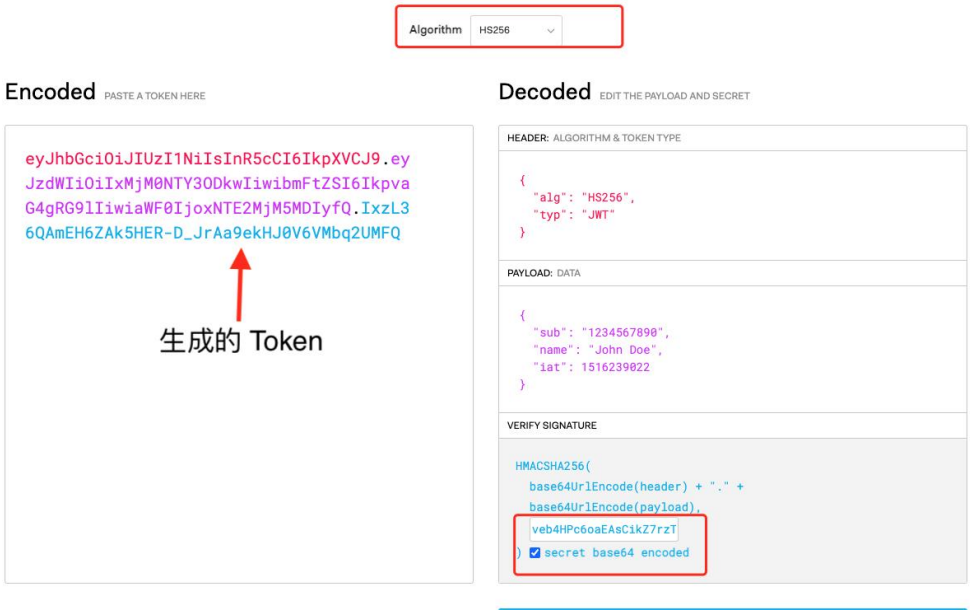
   - KeySize：生成 secret 的大小，输入 256
   - KeyUse：用途，选择签名
   - Algorithm：算法，选择 HS256
   - KeyID：可选项，JWKS 有多个值时的匹配参数



基础配置

4. 复制上图中 k 字段的取值，访问 https://jwt.io，生成 Token。

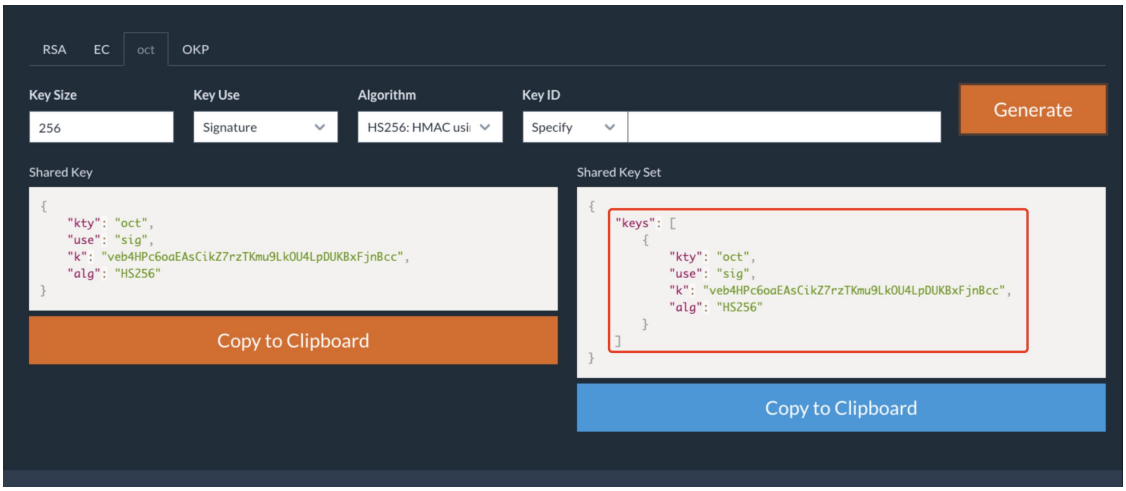
   - 算法选择 HS256
   - 将复制好的 k 值粘贴到 secret 里面，并勾选 secret base64 encoded



基础配置

5. 基于 YAML 模板创建 YAML 文件，然后使用 kubectl apply 命令安装 JWKS 应用

   - 将 namespace 修改为网关所在的命名空间，在本例中使用 envoy-yang
   - 将 jwks.json 修改为上述第三步生成的 JWKS 内容



基础配置

??? note “本例中配置的 YAML 文件如下所示”

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: remote-jwks-go
  name: remote-jwks-go
  namespace: envoy-yang
spec:
  selector:
    matchLabels:
      app: remote-jwks-go
```

```
  template:
    metadata:
      labels:
        app: remote-jwks-go
    spec:
      containers:
        - args:
            - jwks
            - -c
            - /app/jwks.json
          command:
            - main
          image: release-ci.daocloud.io/skoala/demo/remote-jwks-go:0.1.
0
          imagePullPolicy: IfNotPresent
          name: remote-jwks-go
          ports:
            - containerPort: 8080
              name: http
              protocol: TCP
          volumeMounts:
            - name: config
              mountPath: /app/jwks.json
              subPath: jwks.json
      volumes:
        - name: config
          configMap:
            name: jwks-config
      restartPolicy: Always
      securityContext:
        runAsNonRoot: true
        runAsUser: 65534
```

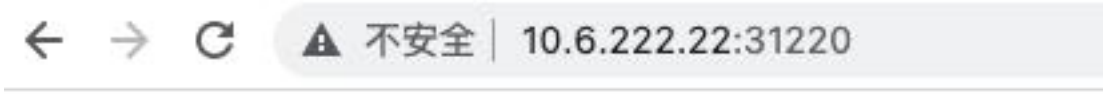
```
          runAsGroup: 65534

---
apiVersion: v1
kind: Service
metadata:
  name: remote-jwks-go
  namespace: envoy-yang
  labels:
    app: remote-jwks-go
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: remote-jwks-go

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: jwks-config
  namespace: envoy-yang
  labels:
    app: remote-jwks-go
data:
  jwks.json: |+
    {
      "keys": [
```

```
            {
            "kty": "oct",
            "use": "sig",
            "k": "veb4HPc6oaEAsCikZ7rzTKmu9LkOU4LpDUKBxFjnBcc",
            "alg": "HS256"
        }
          ]
        }
```

6. 访问应用的 8080 端口，出现 success 说明应用安装成功。

   JWKS 地址应为 网关访问地址/jwks 构成，例如 http://13.5.245.34:31456/jwks

   在微服务引擎下的网关概览页面可以查看网关的访问地址。



success

基础配置

示例应用体验微服务治理

微服务引擎属于 DCE 5.0 高级版功能，其中包含注册中心、配置中心、微服务治理（传统微服务、云原生微服务）、云原生网关等功能。本文将通过示例应用带您体验其中的微服务治理功能。

此次最佳实践包含的全部流程如下：

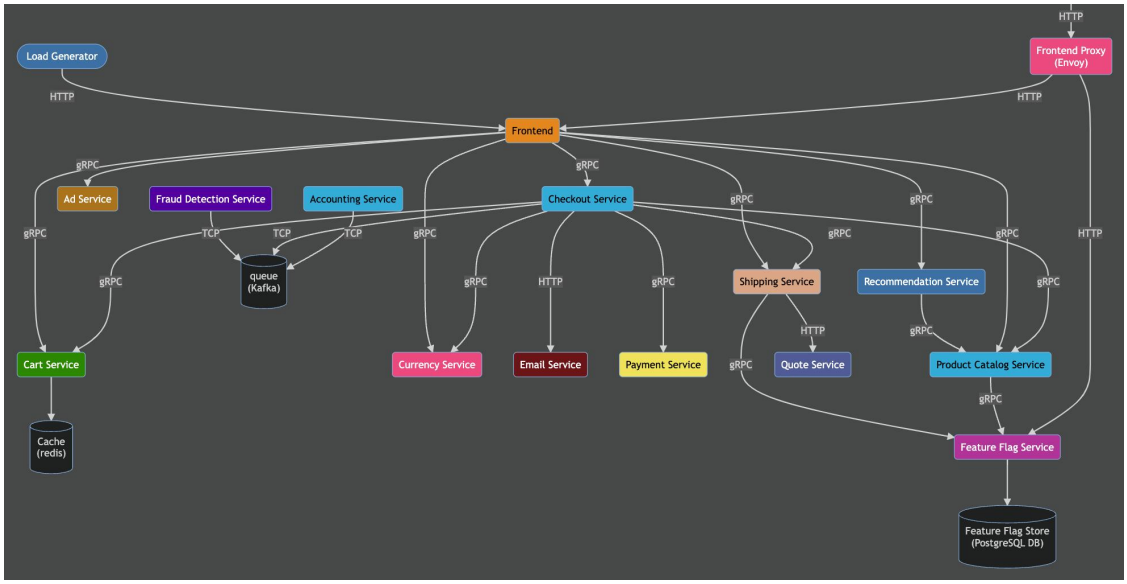
1. 在应用工作台部署示例应用，并启用微服务治理功能
2. 在微服务引擎中启用传统微服务治理插件
3. 在微服务引擎中配置对应的治理规则
4. 在微服务引擎暴露 API 并访问应用

示例应用介绍

本次实践使用的示例应用基于 OpenTelemetry 的标准演示应用，由 DaoCloud 大微服务团队根据 DCE 5.0 的功能加以优化，以便更好地体现云原生以及可观测能力，呈现微服务治理效果。

示例应用已经在 Github 开源，访问该应用的 Github 仓库地址可以获取详细信息。

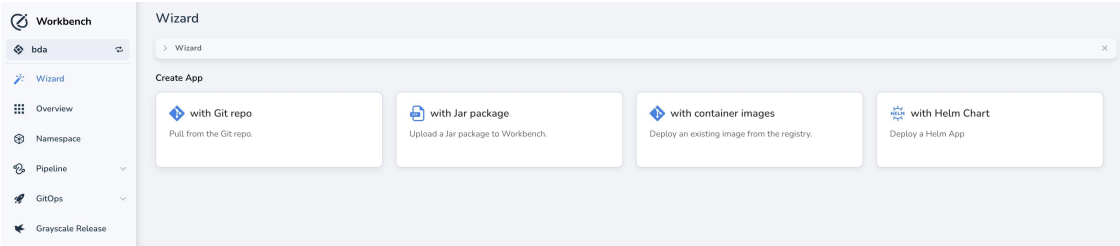
示例应用的架构图如下：



image

应用部署

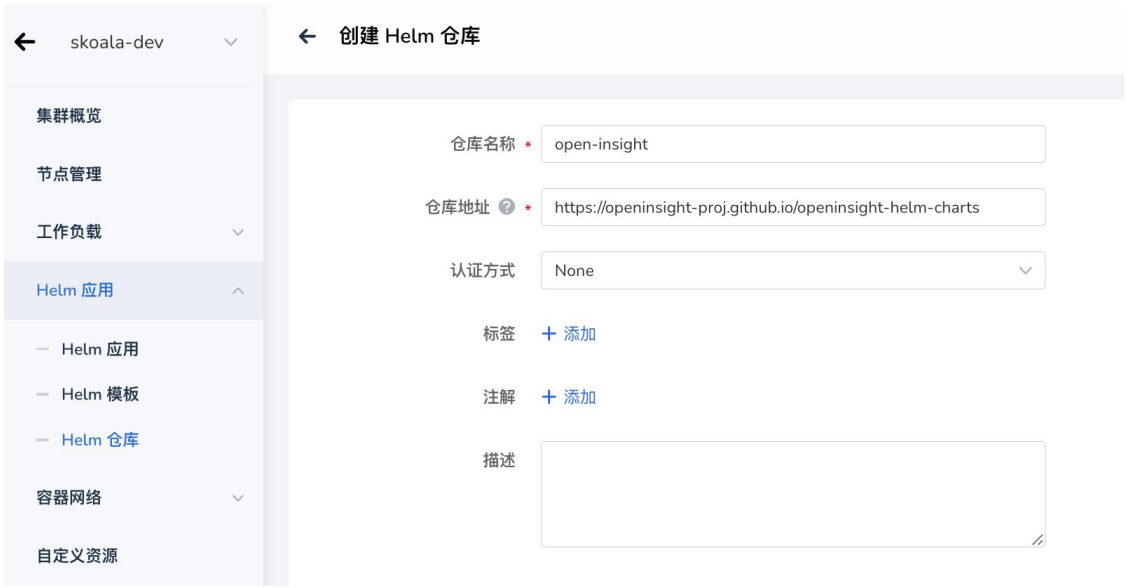
应用工作台是 DCE 5.0 的应用管理模块，支持创建/维护多种类型的应用、GitOps 和灰度发布等功能，可以快速将应用部署到任何集群。应用工作台支持基于 Git 仓、Jar 包、容器镜像、Helm 模板部署应用。本次实践基于 Helm 模板部署示例应用。



image

部署应用之前需要满足如下的前提条件：

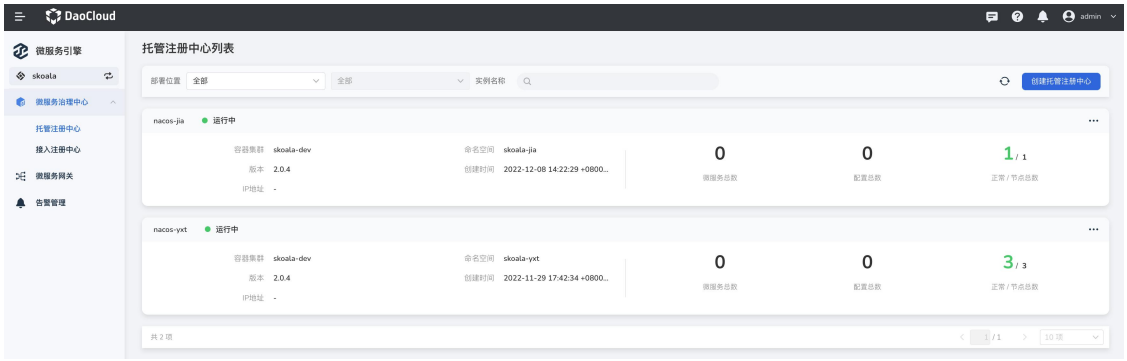
- 在容器管理中添加 Helm 仓库：



image

- 在微服务引擎中创建 Nacos 注册中心实例

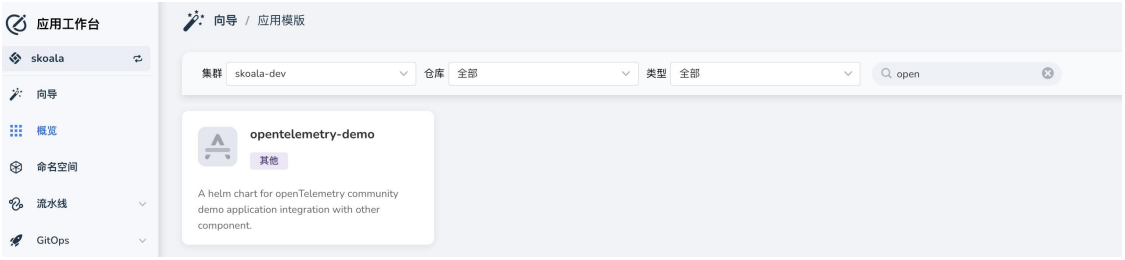
注意记录注册中心的地址信息，后续安装应用时需要用到。



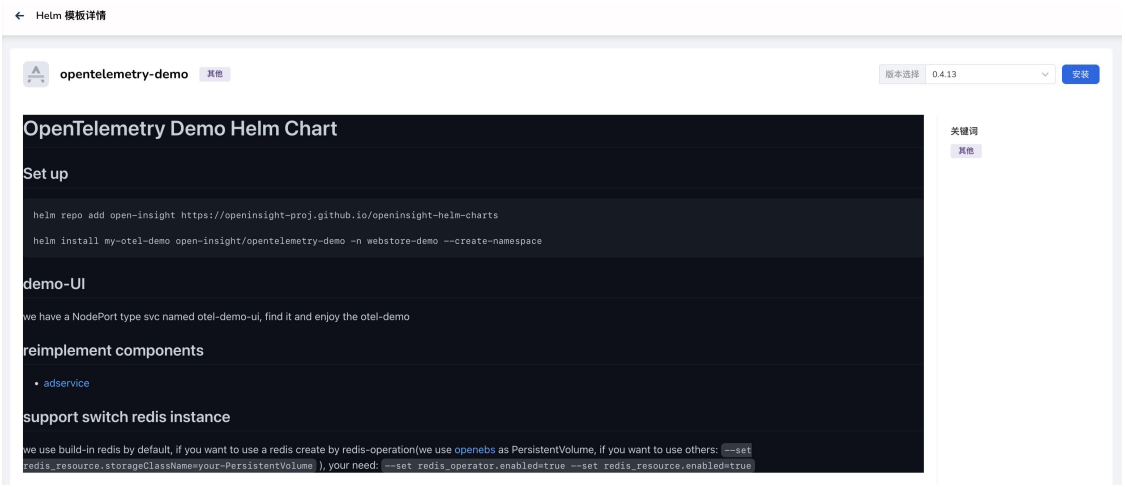
image

基于 Helm 模板部署

1. 在应用工作台–>向导–>基于 Helm 模板中，找到 opentelemetry–demo 应用，点击应用卡片进行安装

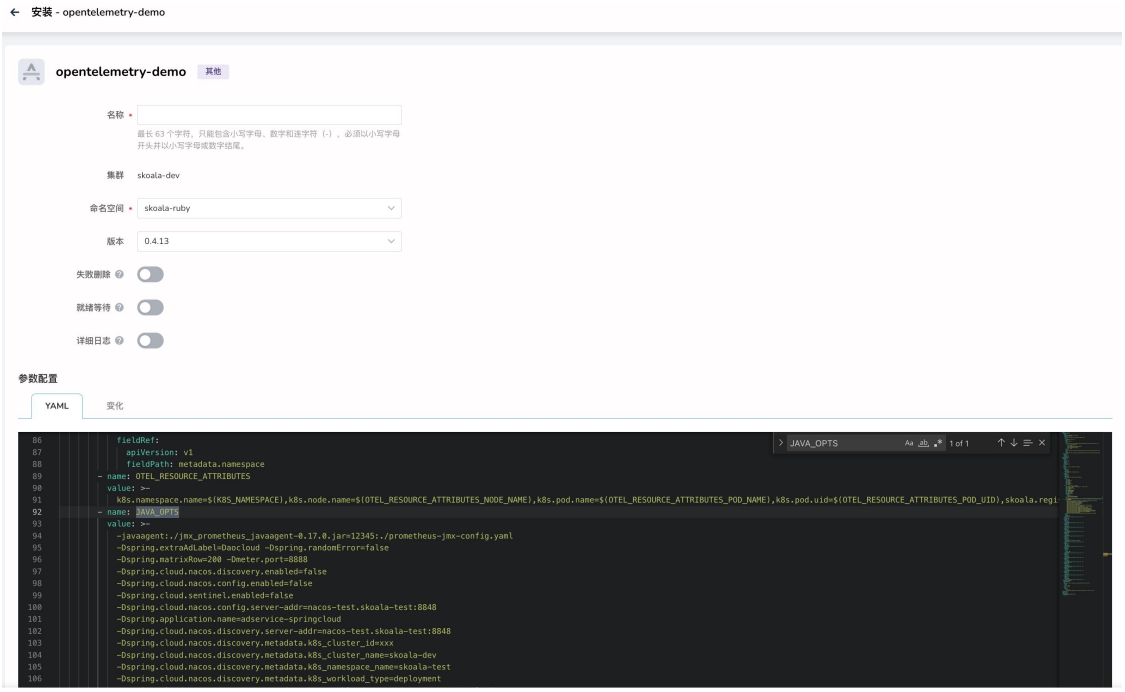


image



image

2. 在 Helm 的安装界面，注意确认部署位置是否正确，然后按照下方要求更新 JAVA_OPTS 部分的参数配置。



image

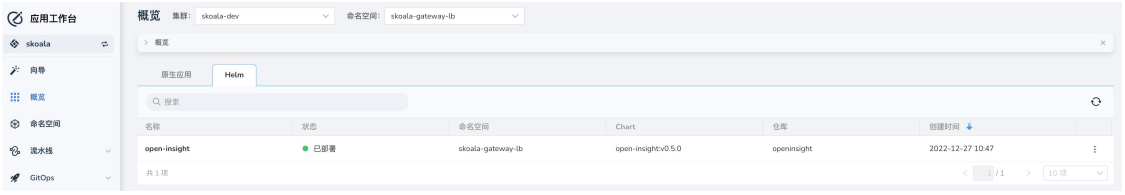
根据上方记录的注册中心地址，将下方带有注释的参数更新如下：

```
-javaagent:./jmx_prometheus_javaagent-0.17.0.jar=12345:./prometheus-jmx
-config.yaml
    -Dspring.extraAdLabel=Daocloud  -Dspring.randomError=false
    -Dspring.matrixRow=200  -Dmeter.port=8888
    -Dspring.cloud.nacos.discovery.enabled=true          # 修改，以启用 Nac
os 服务注册发现
    -Dspring.cloud.nacos.config.enabled=true             # 修改，以启用 Nac
os 配置管理能力
    -Dspring.cloud.nacos.config.server-addr=nacos-test.skoala-test:8848
          # 修改，以配置 Nacos 注册中心地址
    -Dspring.application.name=adservice-springcloud
    -Dspring.cloud.nacos.discovery.server-addr=nacos-test.skoala-test:884
8          # 修改，以配置 Nacos 注册中心地址
    -Dspring.cloud.nacos.discovery.metadata.k8s_cluster_id=xxx
          # 修改，以配置 Nacos 注册中心所在集群 ID
```

-Dspring.cloud.nacos.discovery.metadata.k8s_cluster_name=skoala-dev
修改，以配置 Nacos 注册中心所在集群名称
-Dspring.cloud.nacos.discovery.metadata.k8s_namespace_name=skoala-test # 修改，以配置 Nacos 注册中心所在命名空间
-Dspring.cloud.nacos.discovery.metadata.k8s_workload_type=deployment
-Dspring.cloud.nacos.discovery.metadata.k8s_workload_name=adservice-springcloud
-Dspring.cloud.nacos.discovery.metadata.k8s_service_name=adservice-springcloud
-Dspring.cloud.nacos.discovery.metadata.k8s_pod_name=${HOSTNAME}
-Dspring.cloud.sentinel.enabled=**false** # 修改，以启用 Sentinel
-Dspring.cloud.sentinel.transport.dashboard=nacos-test-sentinel.skoala-test:8080 # 修改，以配置 Sentinel 控制台地址

获取集群 ID、集群名称、命名空间名称的方法可参考: kubectl get cluster <clusername> -o json | jq .metadata.uid

3. 应用创建成功后，会显示在应用工作台的 Helm 应用列表。



image

Java 项目自行开发调试

如果采用其他部署方式，配置注册中心地址的方法可能有所不同。Java 项目在开发时需要集成 Nacos 的 SDK，而 DCE 5.0 提供的注册中心完全兼容开源 Nacos，所以可以直接使用开源 Nacos 的 SDK。具体操作步骤可参考基于 Jar 包部署 Java 应用。

使用 java -jar 启动项目时，添加对应的环境变量配置即可

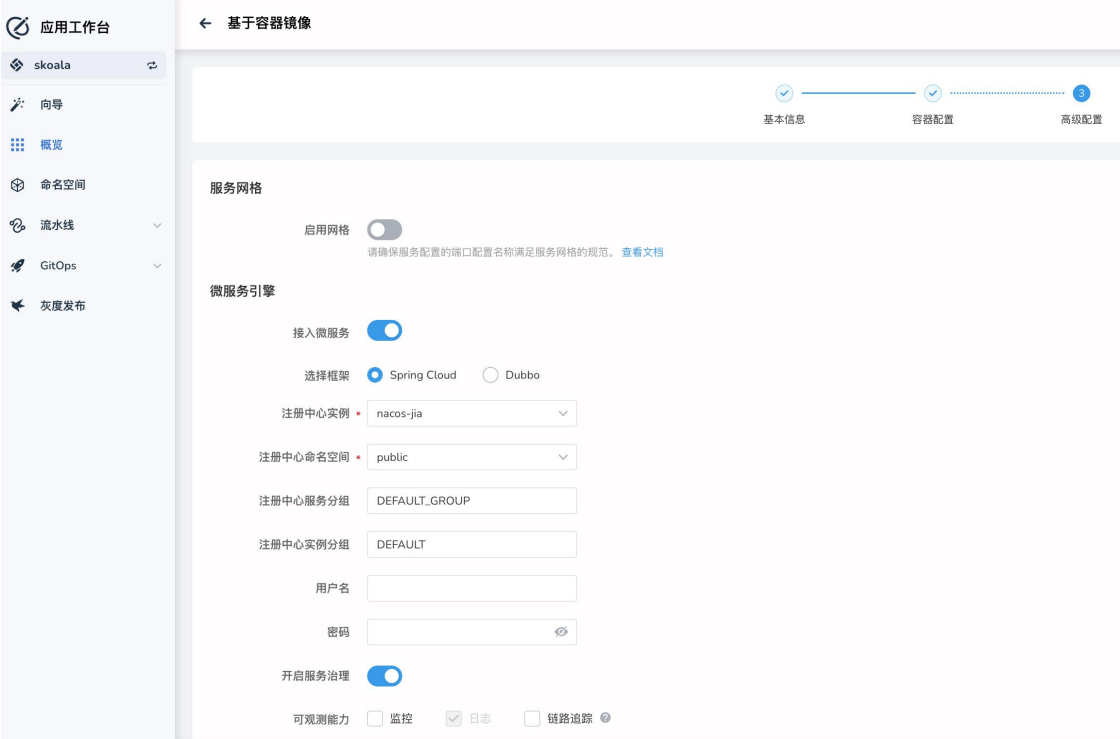
```
      -Dspring.cloud.nacos.discovery.enabled=false          # 启用 Nacos 服务注册
发现
      -Dspring.cloud.nacos.config.enabled=false             # 启用 Nacos 配置管理
能力
      -Dspring.cloud.sentinel.enabled=false                 # 启用 Sentinel
      -Dspring.cloud.nacos.config.server-addr=nacos-test.skoala-test:8848
    # 配置 Nacos 注册中心地址
      -Dspring.application.name=adservice-springcloud
      -Dspring.cloud.nacos.discovery.server-addr=nacos-test.skoala-test:8848
    # 配置 Nacos 注册中心地址
      -Dspring.cloud.nacos.discovery.metadata.k8s_cluster_id=xxx
  # 配置 Nacos 注册中心所在集群 ID
      -Dspring.cloud.nacos.discovery.metadata.k8s_cluster_name=skoala-dev
      # 配置 Nacos 注册中心所在集群名称
      -Dspring.cloud.nacos.discovery.metadata.k8s_namespace_name=skoala-test
        # 配置 Nacos 注册中心所在命名空间
      -Dspring.cloud.nacos.discovery.metadata.k8s_workload_type=deployment
      -Dspring.cloud.nacos.discovery.metadata.k8s_workload_name=adservice-sprin
gcloud
      -Dspring.cloud.nacos.discovery.metadata.k8s_service_name=adservice-springcl
oud
      -Dspring.cloud.nacos.discovery.metadata.k8s_pod_name=${HOSTNAME}
```

!!! note

上面的 `metadata` 信息不能缺失，否则注册中心中呈现的服务会缺失这部分信息。

使用容器镜像部署

如果选择基于容器镜像部署应用，可以直接在用户界面配置中开启微服务治理并选取对应的注册中心模块，操作更简便。具体步骤可参考基于 Git 仓构建微服务应用。



image

启用传统微服务治理

开始使用微服务治理功能之前，需要在对应注册中心下的插件中心开启对应的治理插件。插件中心提供 Sentinel 治理和 Mesh 治理两种插件，支持通过用户界面实现可视化配置。安装插件后可以扩展微服务治理能力，满足不同场景下的业务诉求。

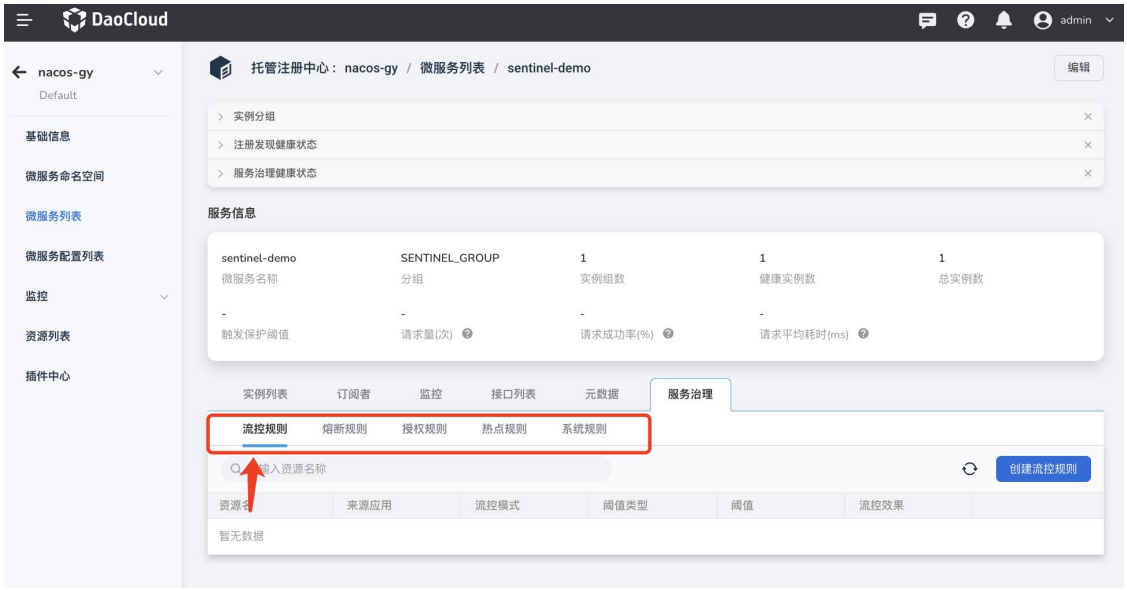
本次实践采用传统微服务治理，即开启 Sentinel 治理插件。如需了解详细步骤，可参考启用 Sentinel 治理插件。



image

配置对应的治理规则

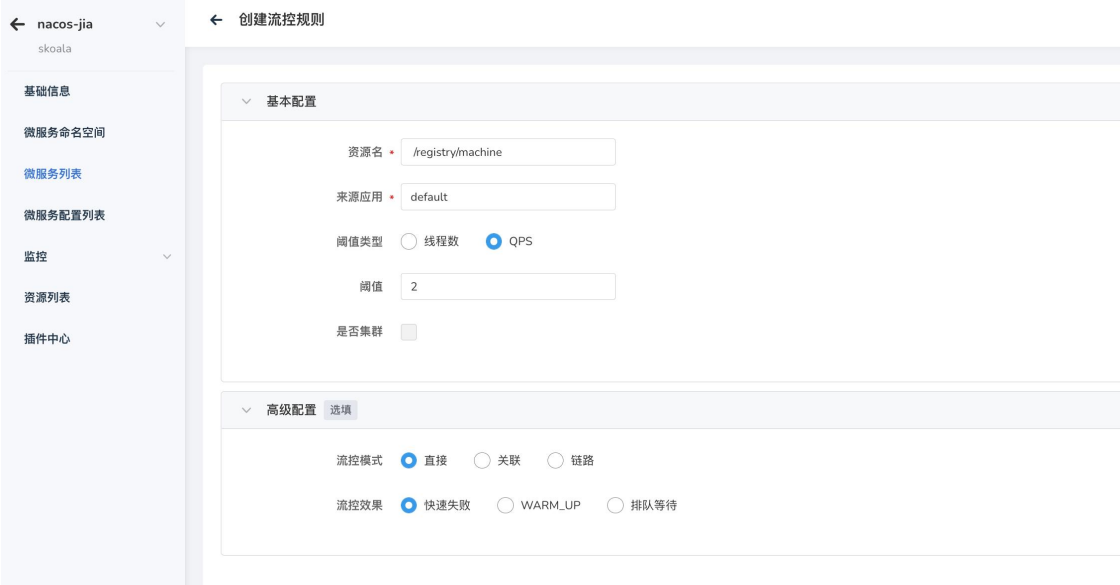
应用部署成功后，可以在之前准备注册中心下的微服务列表中查看对应的服务。微服务列表提供流控规则、熔断降级、热点规则、系统规则、授权规则等流量治理规则。本次实践以流控规则为例进行演示。



image

配置流控策略

这里限流策略示例，我们通过简单的配置即可为服务增加对应的限流策略。



image

测试流控策略

通过访问服务地址，我们可以看到在 1 分钟内请求次数大于 2 次之后，后续请求都会被拦截；在超过 1 分钟后自动恢复。

暴露 API 并访问应用

微服务应用部署完成后，需要通过 API 网关将应用入口开放给外部访问，完成这一步才是完成的服务使用体验。为了暴露服务 API，需要创建云原生网关，将服务接入该网关，并创建对应的 API 路由。

创建云原生网关

首先需要创建一个云原生网关，具体操作步骤可以参考： 创建云原生网关

!!! note

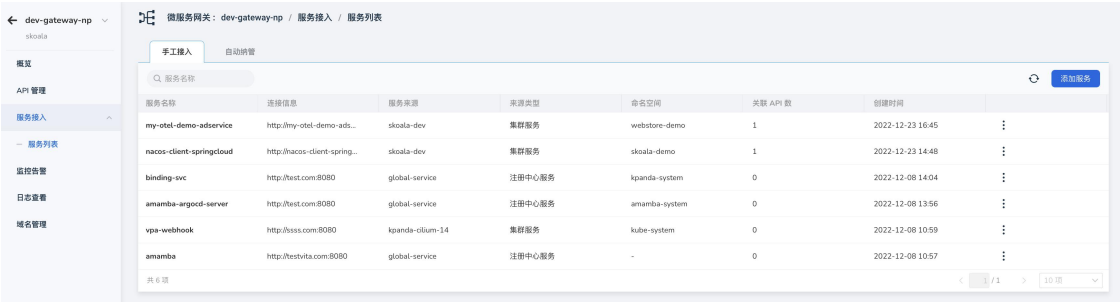
创建网关时，应该将网关部署在示例应用所在的集群，并且该网关需要管辖示例应用所在的命名空间。

![image](https://docs.daocloud.io/daocloud-docs-images/docs/skoala/images/gatewaylist.png)

接入服务

基于 DCE 5.0 的特性，云原生网关可以自动发现所管辖命名空间内的服务，所以无需手动接入服务。

本次演示采用 Nacos 注册中心的服务，很大程度上扩宽了网关可接入的服务数量，可以在服务接入中选择接入 Nacos 注册中心的服务。



image

!!! info

当服务不在网关所管辖的命名空间内，或者想要接入注册中心或者其他外部服务时（使用域名/IP），可以采用手工接入服务的方式。

创建 API 路由

参考文档 添加 API 创建对应的 API 路由。

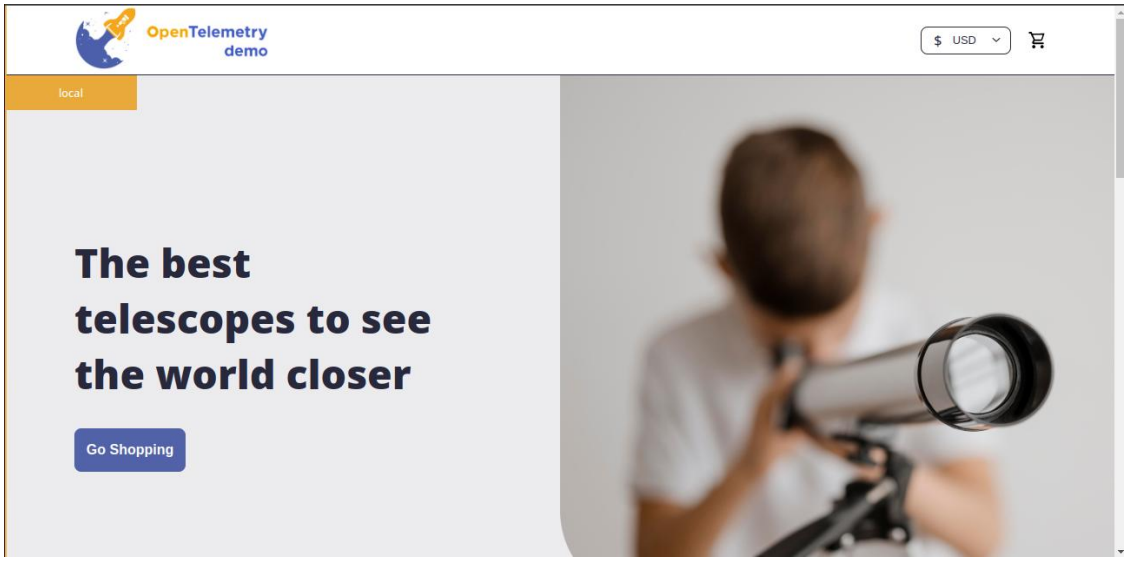


image

访问应用

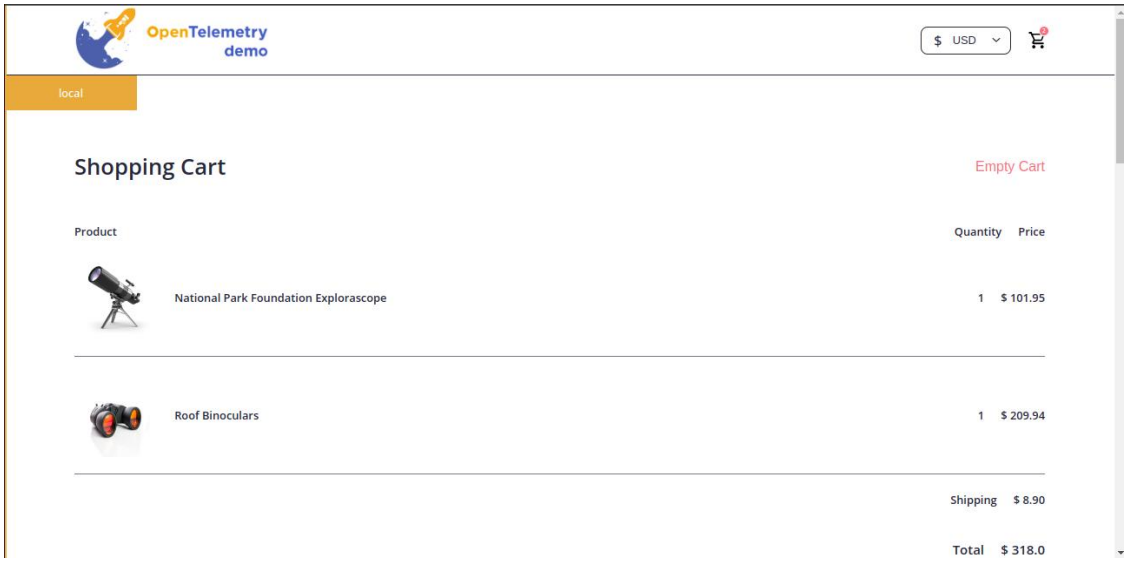
当网关 API 创建完成后，使用创建 API 时配置的 **域名** 与 **外部 API 路径** 即可成功访问到应用页面，如下图所示。

示例应用的首页:



image

示例应用的订单确认页面:



image

结语

以上就是整个微服务引擎模块的体验之旅。在整个 DCE 5.0 的能力支持下我们顺利完成了应用部署、启用微服务治理、配置并测试微服务治理策略、通过云原生网关开放 API、实际访问应用等操作。

更多能力

当我们的应用部署成功之后，对于后续的应用维护过程中，实际非常依赖 DCE 5.0 提供的可观测能力。接下来，我们会补充对应可观测能力实践。

- 查看应用部署后的拓扑结构
- 查看应用的日志内容
- 查看云网关网关 API 的访问日志

微服务网关接入认证服务器

微服务网关支持接入第三方认证服务器。

前提条件

- 创建一个集群或接入一个集群
- 创建一个网关

配置认证服务器

使用默认的认证服务器

1. 将认证服务器的代码模板克隆到本地。

   git clone https://github.com/projectsesame/envoy-authz-java

2. 直接使用 all-in-one-contour.yaml 以及 all-in-one-contour.yaml 下的默认镜像。

   默认镜像如下：

   - release.daocloud.io/skoala/demo/envoy-authz-java:0.1.0
   - release-ci.daocloud.io/skoala/demo/envoy-authz-java:0.1.0

3. 模板为简单的路径判断，当访问路径为 / 时通过认证，其余路径为拒绝访问。

使用自定义的认证服务器

1. 将认证服务器的代码模板克隆到本地。

   git clone https://github.com/projectsesame/envoy-authz-java

   该项目分为两个子模块：

   - API 模块是 envoy 的 protobuf 文件的定义（无需修改）

- authz-grpc-server 模块是认证服务器的认证逻辑处理地址（在这里填写认证逻辑）
- release.daocloud.io/skoala/demo/envoy-authz-java:0.1.0

2. 使用如下命令编译 API 模块，解决类找不到的问题

   mvn clean package

3. 成功编译之后，在 check 方法中编写自定义的认证逻辑。

   - check 方法在 envoy-authz-java/authz-grpc-server/src/main/java/envoy/projectsesame/io/authzgrpcserver/AuthzService.java

   - 模板为简单的路径判断，当访问路径为 / 时通过认证，其余路径为拒绝访问。

4. 代码编写完成之后，使用 Docker 打包镜像。

   代码模板仓库中已存在 Dockerfile 文件，可以直接使用该模板构建镜像。

5. 将镜像地址填入 all-in-one-contour.yaml 文件中的 Deployment 下的 spec/template/spec/containers/image 字段。

64 lines (62 sloc) | 1.24 KB

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: envoy-authz-java
  name: envoy-authz-java
  namespace: envoy-yang
spec:
  selector:
    matchLabels:
      app: envoy-authz-java
  template:
    metadata:
      labels:
        app: envoy-authz-java
    spec:
      containers:
        - image: release-ci.daocloud.io/skoala/demo/envoy-authz-java:0.1.0
          imagePullPolicy: IfNotPresent
          name: envoy-authz-java
          ports:
            - containerPort: 18081
              name: grpc
              protocol: TCP
      restartPolicy: Always
      securityContext:
        runAsNonRoot: true
        runAsUser: 65534
        runAsGroup: 65534
```

填写镜像

接入认证服务器

1. 在网关所在的集群内创建以下资源。使用 kubectl apply 命令基于 all-in-one-contour.yaml 文件可以一次性快速创建下述三项资源。

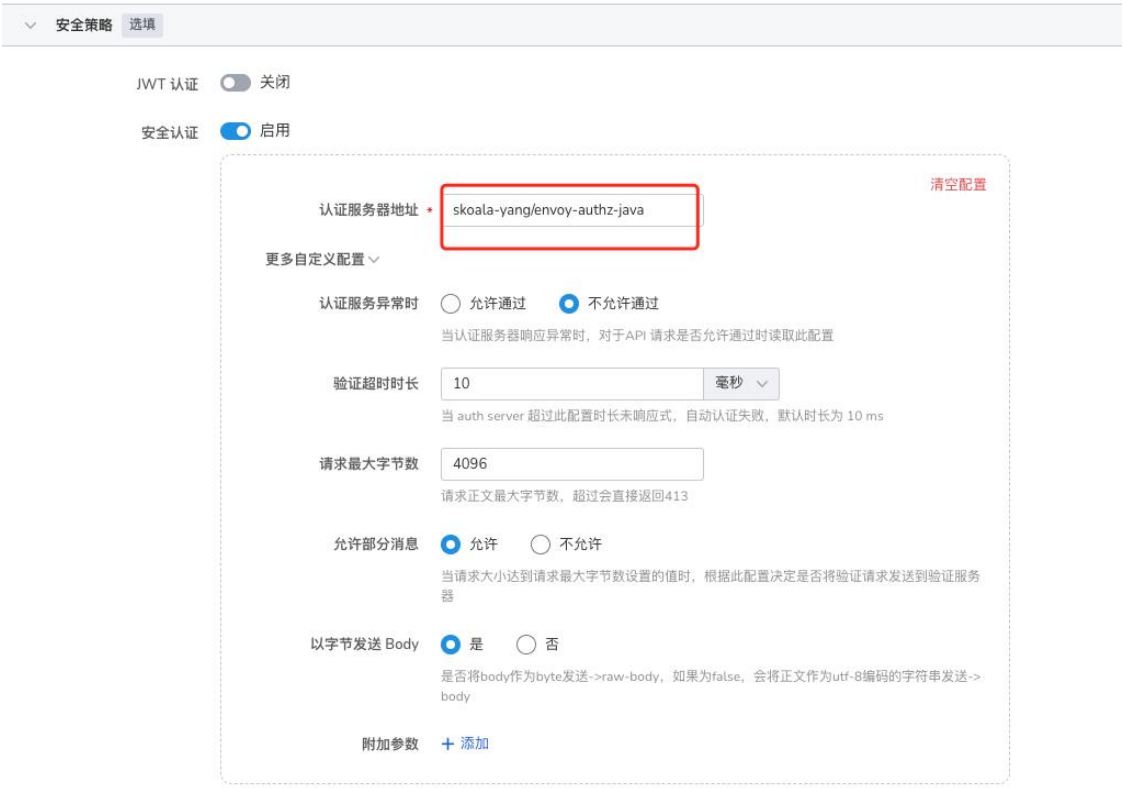
   - 认证服务器的 Deployment
   - 认证服务器的 Service
   - 认证服务器的 ExtensionService

2. 在网关下创建一个使用 https 协议的域名，填写基础信息。



基础配置

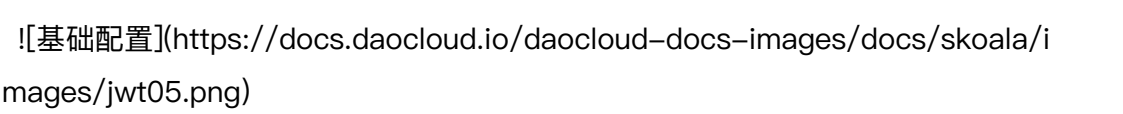
3. 填写该域名的安全配置，指定认证服务器的地址。认证服务器地址格式为namespace/name。



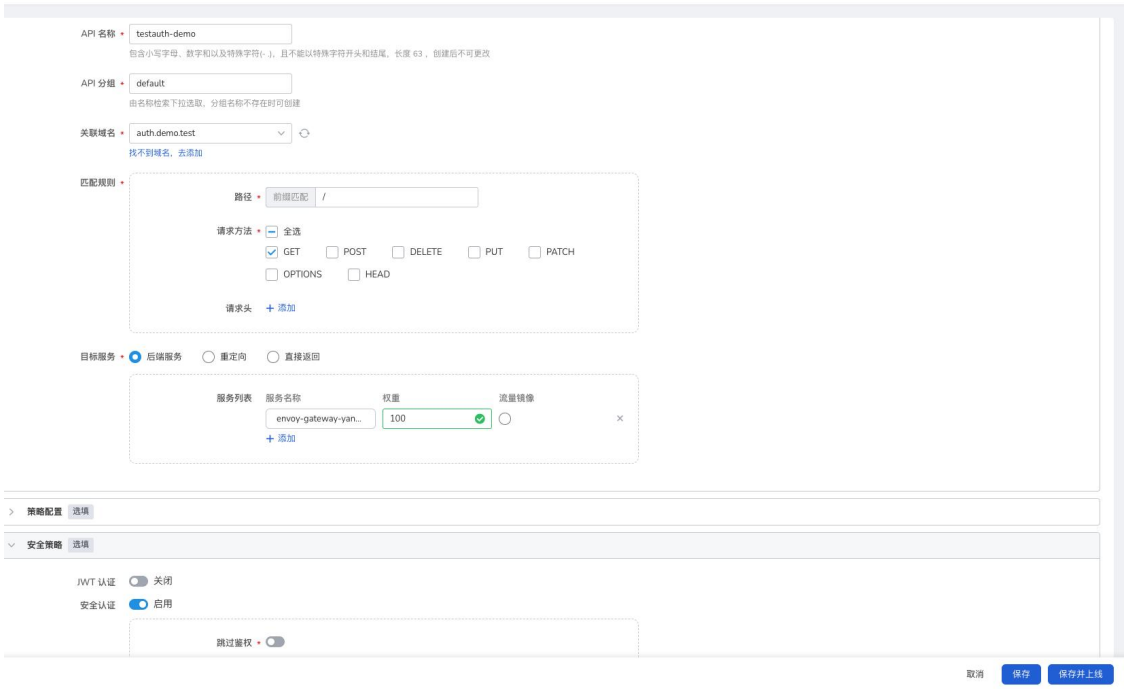
基础配置

!!! note

认证服务器的 `namespace/name` 指的是 [all-in-one-contour.yaml](https://github.com/projectsesame/envoy-authz-java/blob/main/all-in-one-contour.yaml) 文件中 ExtensionService 下的 `metadata` 部分的 `namespace` 和 `name` 字段的取值。



4. 在网关下创建一个 API，关联域名填写刚才新创建的域名，匹配路径为 /，并开启安全认证，并将 API 上线。



基础配置

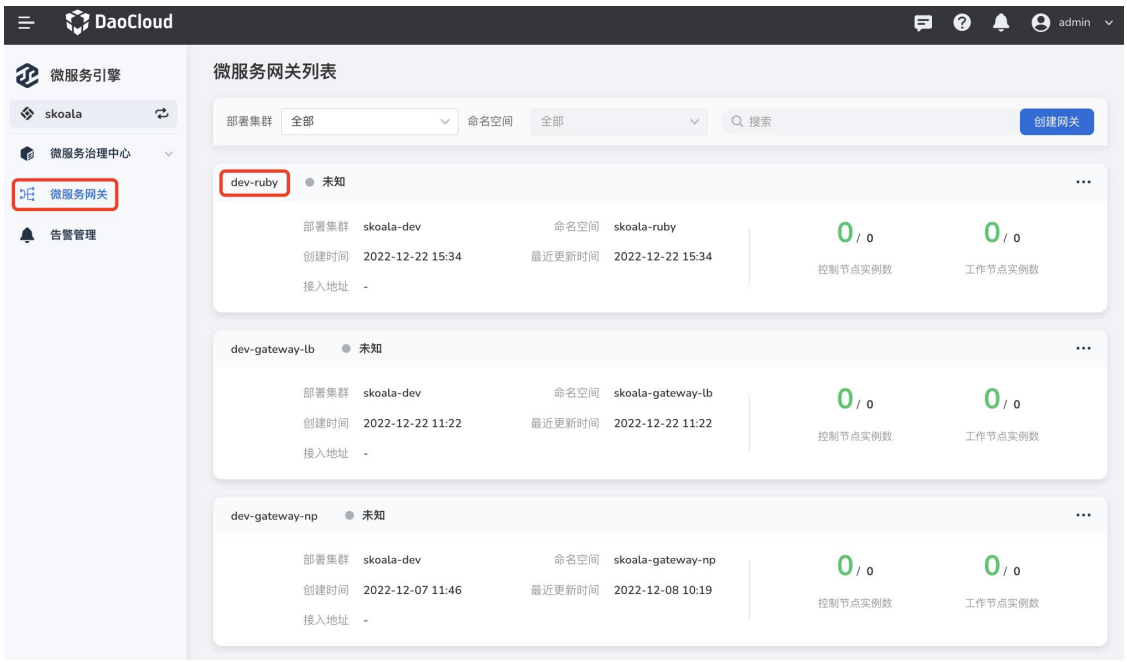
5. 现在即可通过认证服务器访问该 API 了。

监控告警

微服务网关通过内置 Grafana 看板监控网关的资源使用情况、Envoy 的监控详情、服务监控详情、APIServe、HTTPProxy 等资源的运行信息。

查看监控告警信息的步骤如下所示：

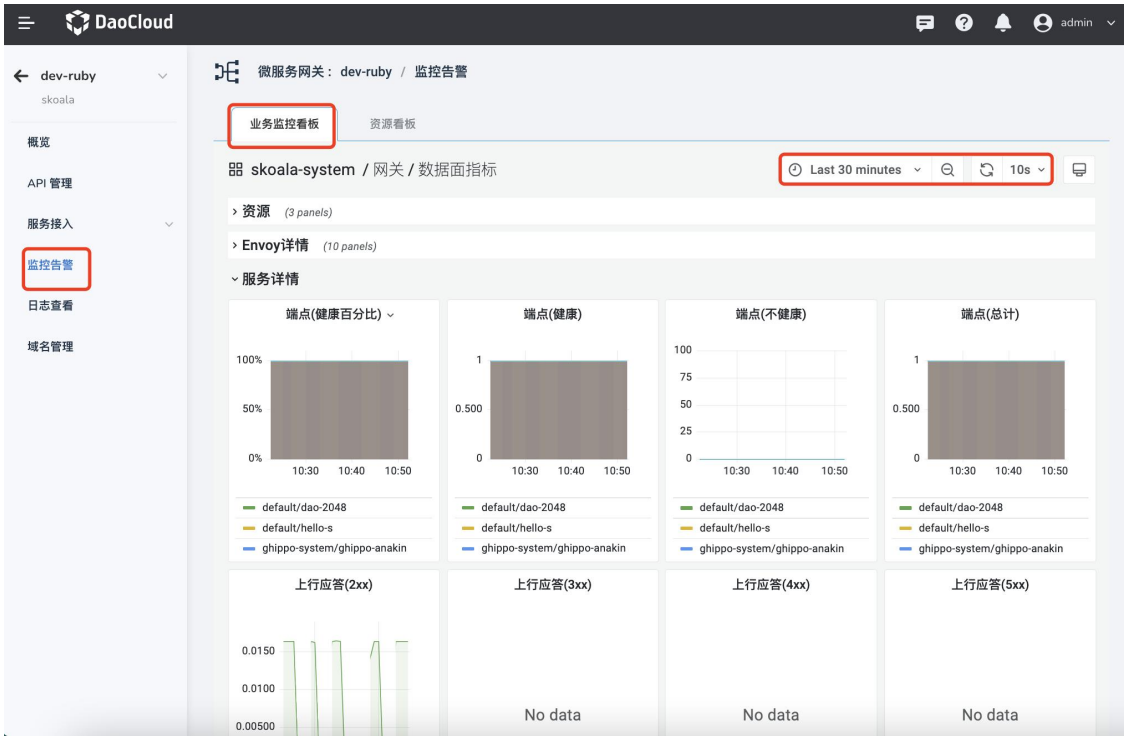
1. 进入微服务引擎模块，在左侧导航栏点击微服务网关，点击目标网关的名称。



点击名称

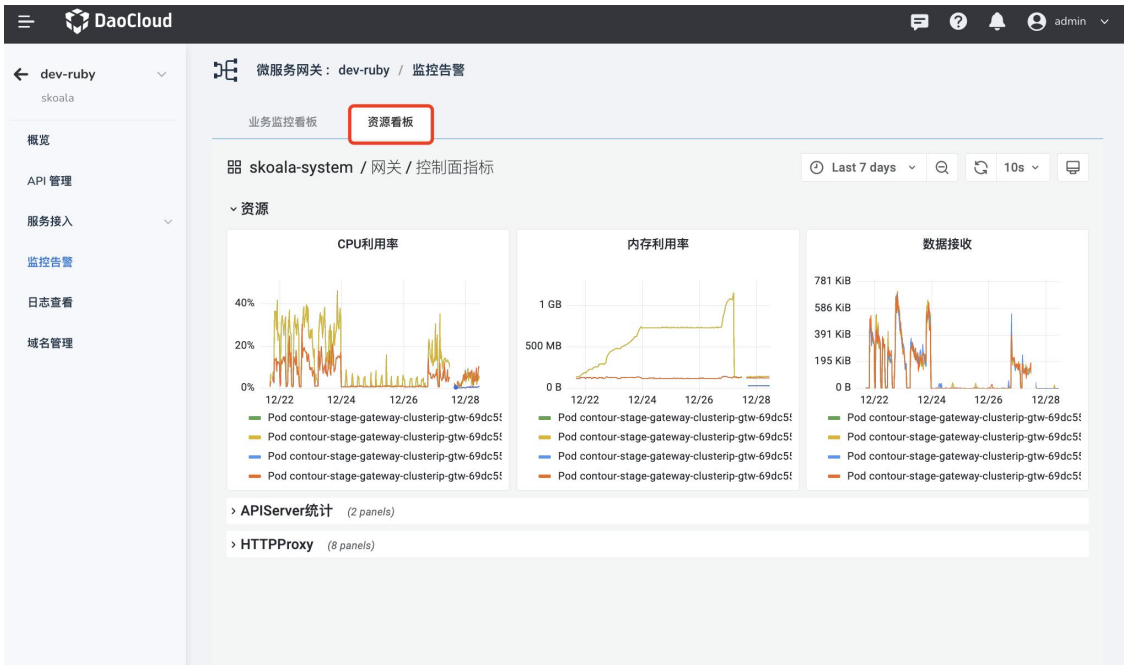
2. 在左侧导航栏点击监控告警，点击业务监控看版页签可查看业务监控信息。

   在页面右上角可选择监控时间窗口和刷新周期。



点击名称

3. 点击资源看版页签可查看资源监控信息。



点击名称

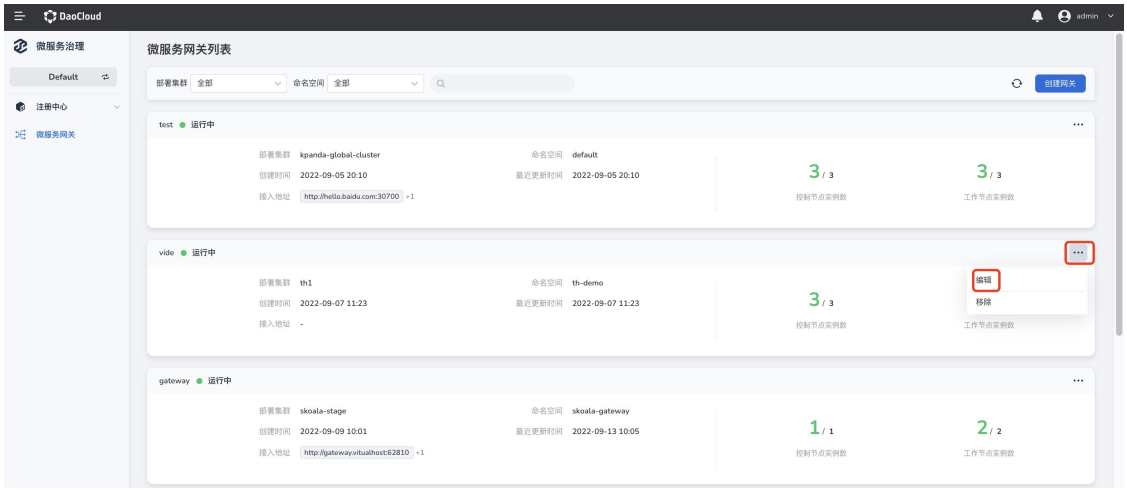
更新微服务网关

微服务网关支持多租户实例的高可用架构，兼容多种模式微服务的统一网关接入能力。本页介绍如何更新微服务网关实例。

更新网关配置

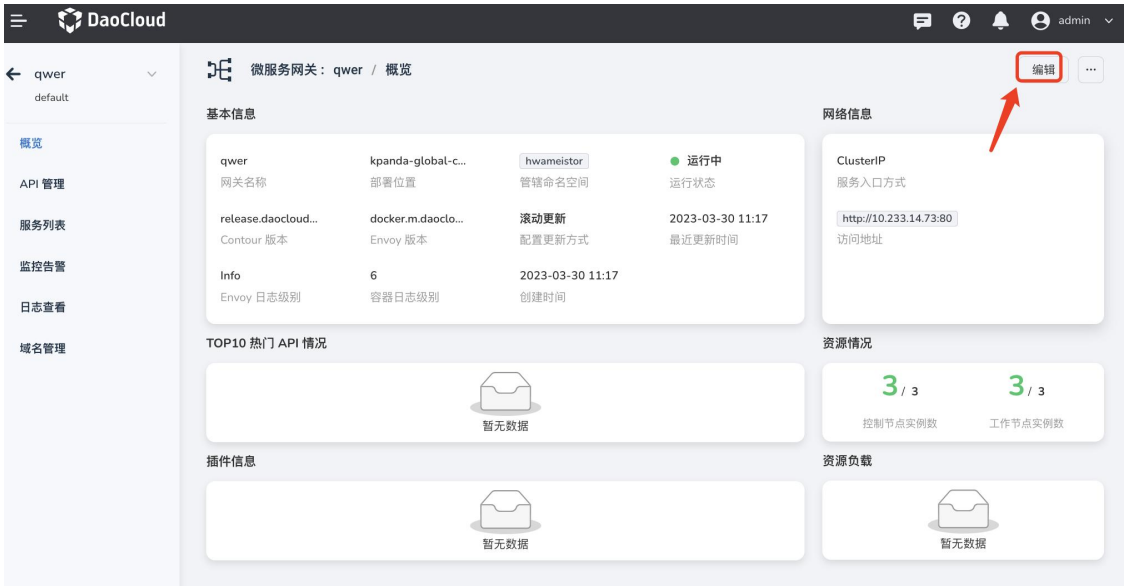
有两种方式可以更新网关配置。

- 在微服务网关列表页选择需要更新的网关实例，在实例右侧点击 ⋯ 并选择编辑。



更新网关

- 点击网关名称进入进入概览页面后，在右上角点击编辑。


DaoCloud
admin
qwer
default
概览
API 管理
服务列表
监控告警
日志查看
域名管理
微服务网关： qwer / 概览
编辑
基本信息
qwer
网关名称
kpanda-global-c...
部署位置
hwameistor
管辖命名空间
运行中
运行状态
release.daocloud...
Contour 版本
docker.m.daoclo...
Envoy 版本
滚动更新
配置更新方式
2023-03-30 11:17
最近更新时间
Info
Envoy 日志级别
6
容器日志级别
2023-03-30 11:17
创建时间
网络信息
ClusterIP
服务入口方式
http://10.233.14.73:80
访问地址
TOP10 热门 API 情况
暂无数据
资源情况
3 / 3
控制节点实例数
3 / 3
工作节点实例数
插件信息
暂无数据
资源负载
暂无数据

更新网关

删除微服务网关

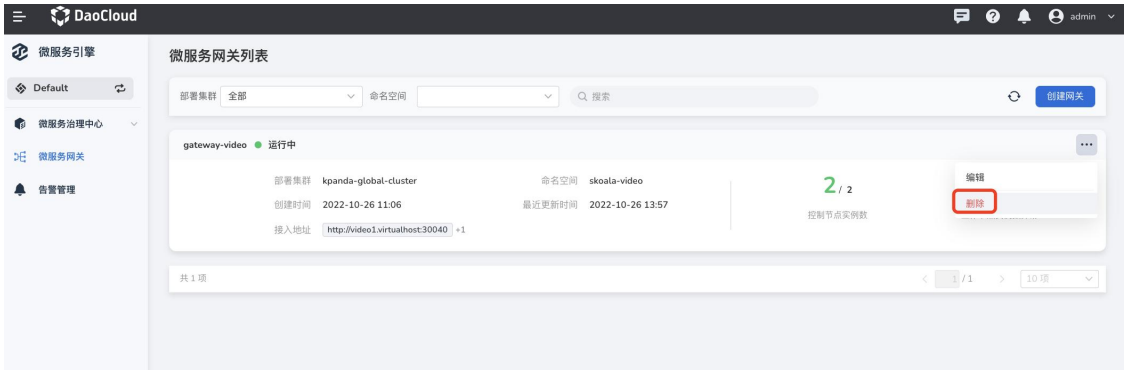
微服务网关支持多租户实例的高可用架构，兼容多种模式微服务的统一网关接入能力。本页介绍如何删除微服务网关实例。

删除网关同样也有两种方式。为保证服务不受影响，删除网关之前需要释放到网关中全部路由的 API。

!!!danger

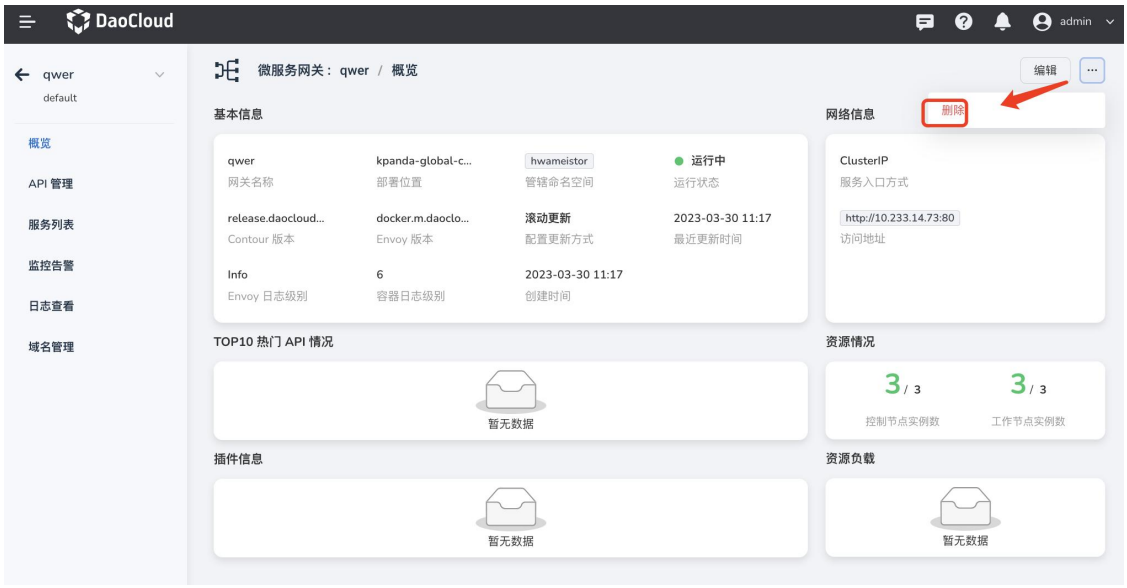
网关删除后不可恢复，请谨慎操作。

- 在微服务网关列表页选择需要移除的网关实例，在实例右侧点击 ⋯ 并选择删除。



移除网关

- 点击网关名称进入进入概览页面后，在右上角 ⋯ 并选择删除。


DaoCloud
admin
qwer
default
概览
API 管理
服务列表
监控告警
日志查看
域名管理
微服务网关： qwer / 概览
编辑
删除
基本信息
qwer
网关名称
kpanda-global-c...
部署位置
hwameistor
管辖命名空间
运行中
运行状态
release.daocloud...
Contour 版本
docker.m.daoclo...
Envoy 版本
滚动更新
配置更新方式
2023-03-30 11:17
最近更新时间
Info
Envoy 日志级别
6
容器日志级别
2023-03-30 11:17
创建时间
网络信息
ClusterIP
服务入口方式
http://10.233.14.73:80
访问地址
TOP10 热门 API 情况
暂无数据
资源情况
3 / 3
控制节点实例数
3 / 3
工作节点实例数
插件信息
暂无数据
资源负载
暂无数据

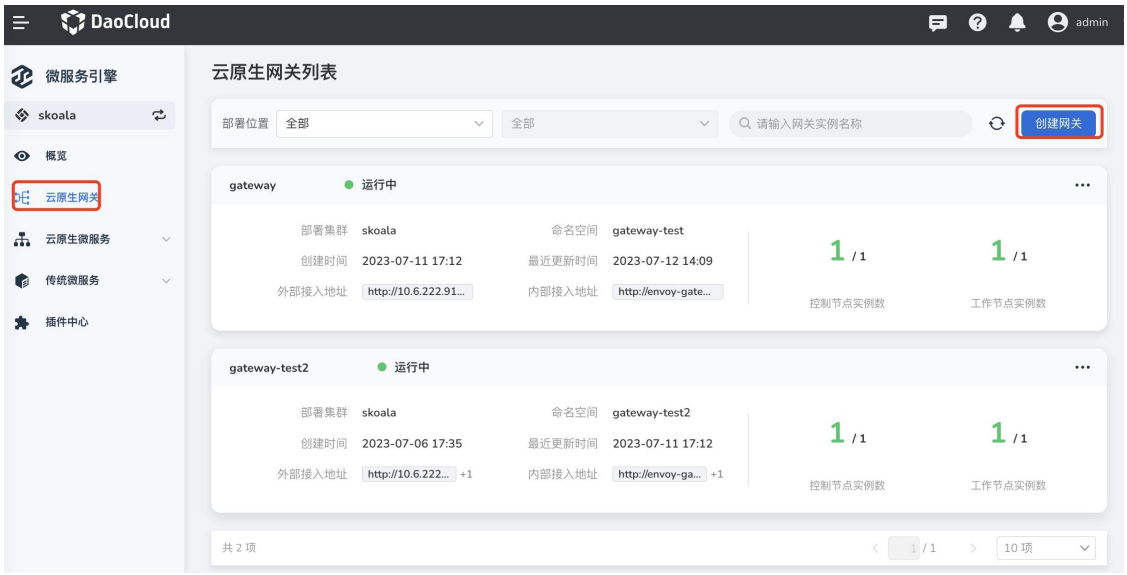
移除网关

创建微服务网关

微服务网关支持多租户实例的高可用架构，兼容多种模式微服务的统一网关接入能力。本页介绍如何创建微服务网关实例。

创建微服务网关的步骤如下：

1. 在左侧导航栏点击云原生网关，然后在右上角点击创建网关。



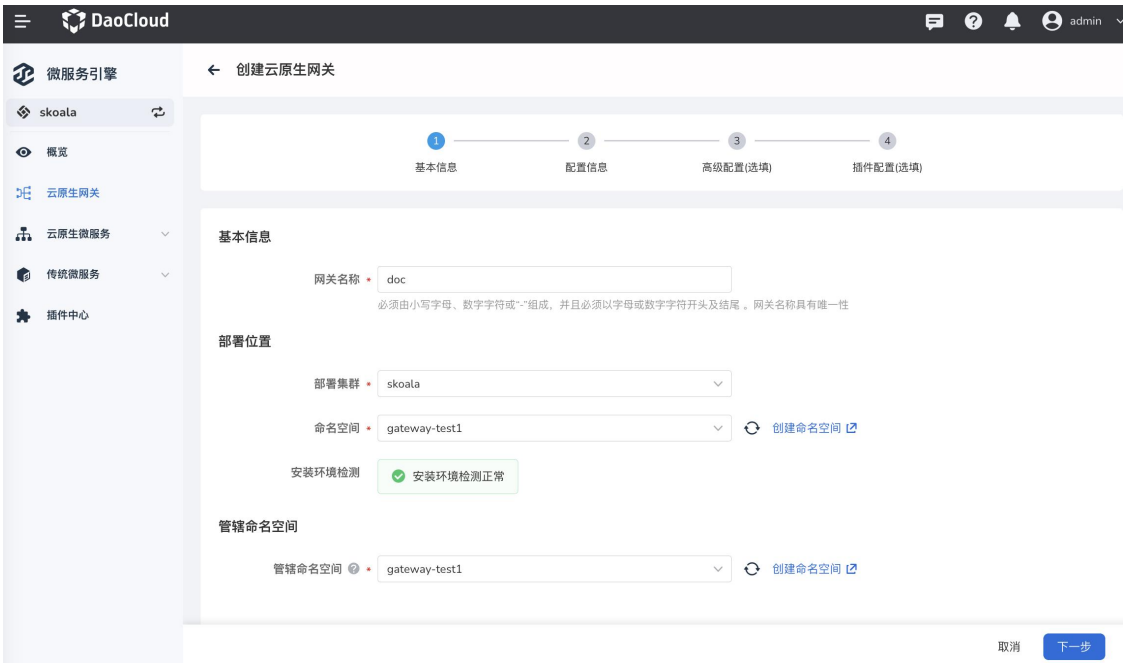
进入创建页面

2. 参考以下说明填写基本信息

   - 网关名称：长度不超过 63 个字符，支持字母、数字、下划线。名称在网关创建完成后不可更改。
   - 部署集群：选择将网关部署在哪个集群。

     如果可选列表中没有出现目标集群，可以去容器管理模块中接入或创建集群并通过全局管理模块将该集群或其中的某个命名空间绑定到当前工作空间。

- 命名空间：选择将网关部署在哪个命名空间。一个命名空间中只能部署一个网关。
- 安装环境检测：选择集群和命名空间后，系统会自动检测安装环境。未通过检测时，页面会给出原因和操作建议，根据页面提示操作即可。
- 管辖命名空间：设置新建的网关可以管辖哪些命名空间。默认管辖网关所在的命名空间。支持同时管辖多个命名空间。同一个命名空间不能被两个网关同时管辖。



填写基本配置

3. 参考以下说明填写配置信息

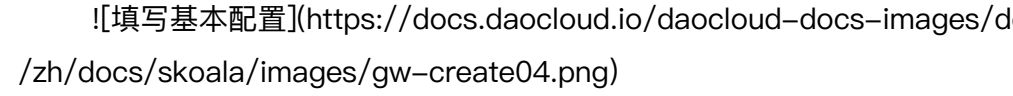
===“服务配置”

- 集群内部访问：只能在同一集群的内部访问服务。

![填写基本配置](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-create03.png)

– 节点访问：通过节点的 IP 和静态端口访问服务，支持从集群外部访问服务。

外部流量策略：`Cluster` 指流量可以转发到集群中其他节点上的 Pod；`Local`指流量只能转发到本节点上的 Pod。



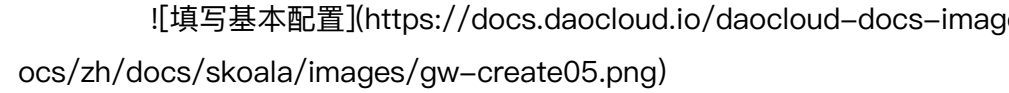
– 负载均衡器：使用云服务提供商的负载均衡器使得服务可以公开访问

– 外部流量策略：`Cluster` 指流量可以转发到集群中其他节点上的 Pod；`Local`指流量只能转发到本节点上的 Pod

– 负载均衡类型：MetalLB 或其他

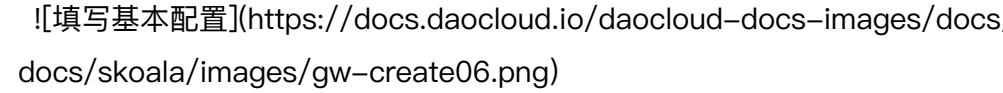
– MetalLB IP 池：支持自动选择或指定 IP 池子

– 负载均衡器 IP 地址：支持自动选择或指定 IP



===“资源配置”

为当前网关配置多少控制节点和工作节点。单副本存在不稳定性，需谨慎选择。



===“高级配置”

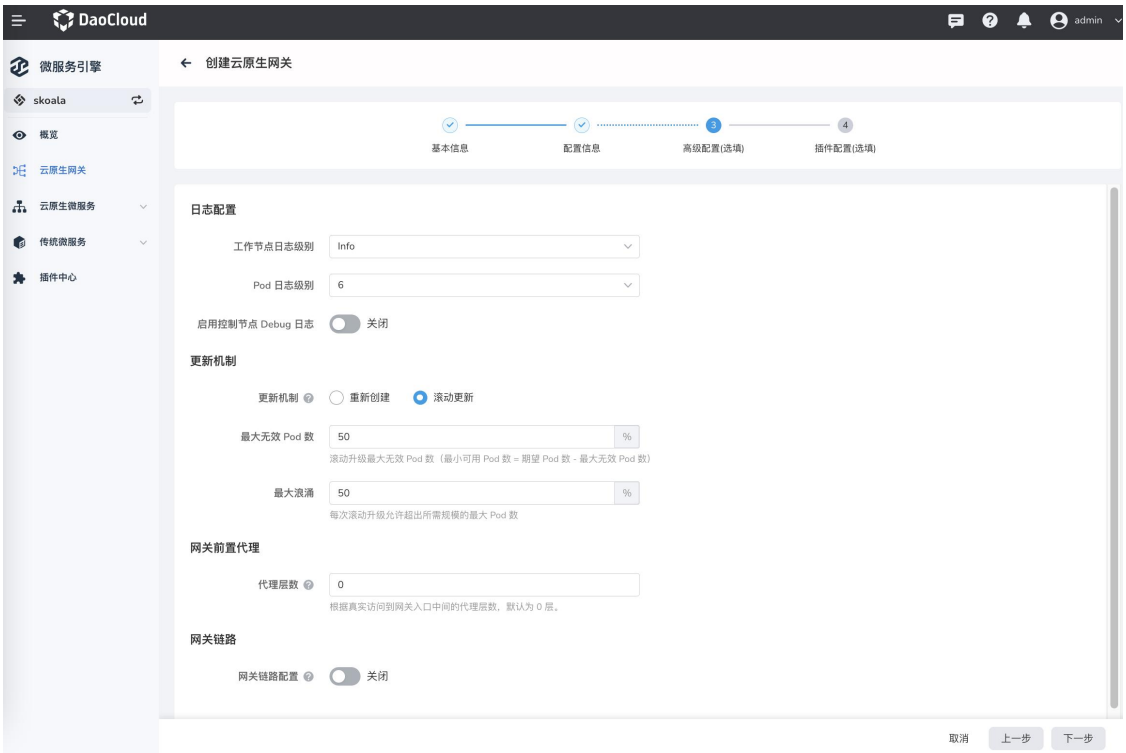
– 控制节点：设置为控制节点（contour）配置多少 CPU 和内存资源

– 工作节点：设置为工作节点 （envoy）配置多少 CPU 和内存资源

![填写高级配置](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-create07.png)

4. 参考以下信息填写高级配置

   - 日志配置：设置工作节点 （envoy）的日志级别和 Pod 的日志级别
   - 更新机制：重新创建指删除原来的网关并新建一个网关，滚动更新指不删除网关，而是滚动式地更新网关相关的 Pod
   - 网关前置代理层数：请求从客户端到网关中途需要经过几个代理端点。需要根据实际情况填写。例如客户端–Nginx–网关的代理层数为 1，因为中间只经过 1 个 Nginx 代理端点。
   - 网关链路：启用后，可以根据通过网关进行的请求生成链路信息并发送给可观测模块进行数据采集。

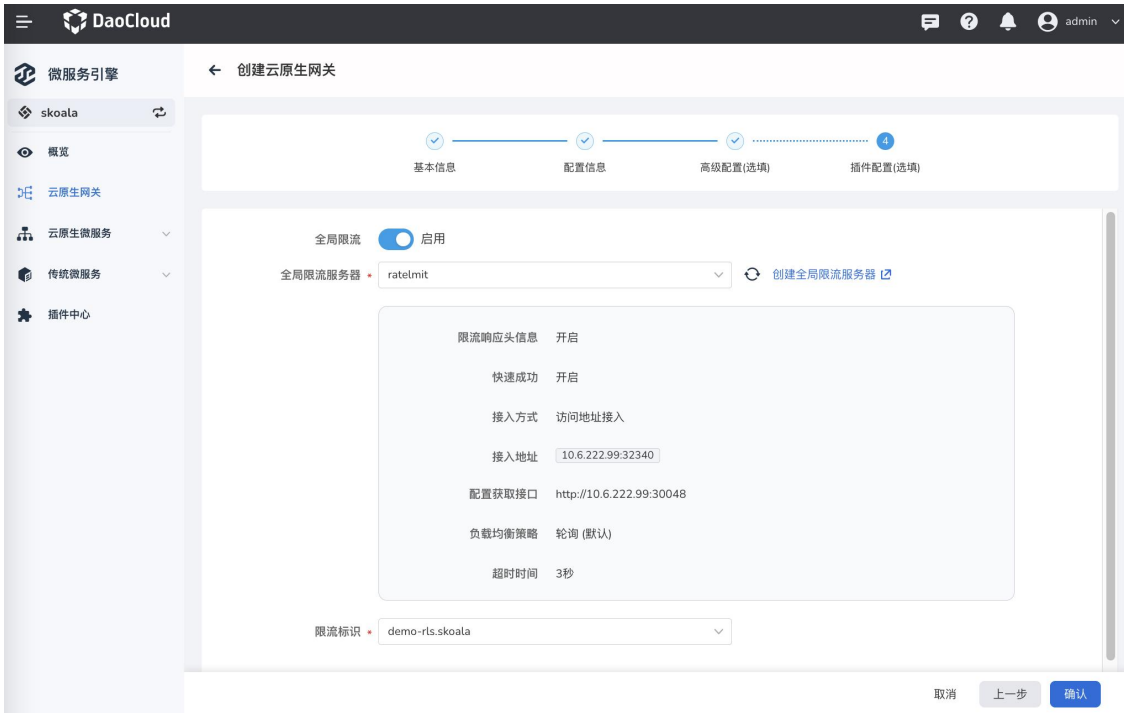


填写高级配置

5. 参考以下信息填写插件配置（选填），最后在页面右下角点击确认即可。

选择是否启用全局限流插件。

需要事先在插件中心接入插件，或在当前页面点击蓝色文字跳转到相应页面接入创建。



填写高级配置

!!! note

系统会自动返回云原生网关列表页面，在右侧可以执行[更新网关](update-gateway.md)或[删除网关](delete-gateway.md)的操作。

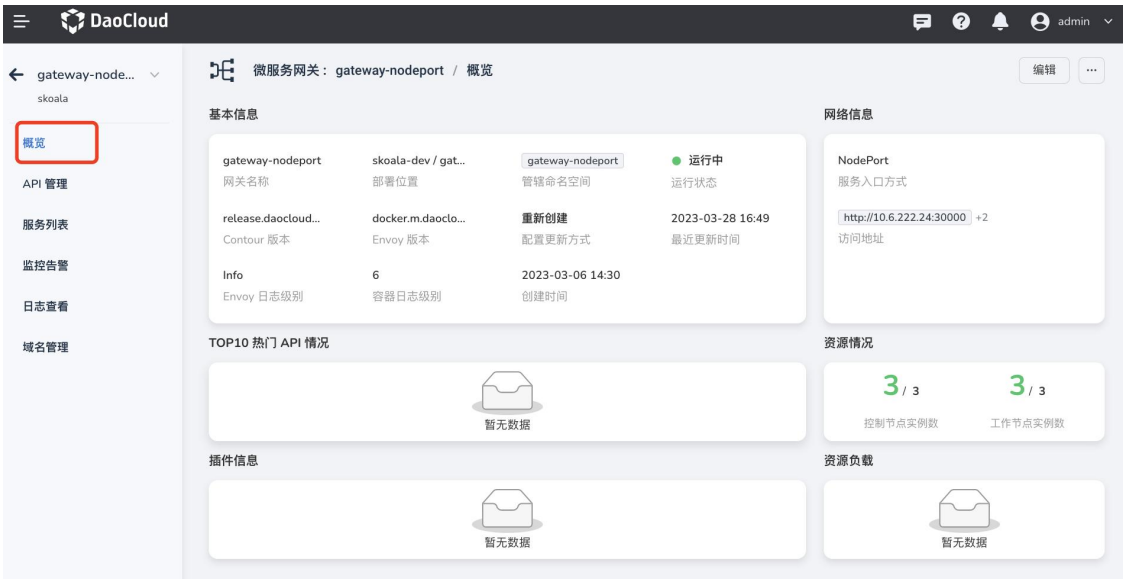
![确认所填信息](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-create10.png)

网关概览

可以在概览页面查看网关的详细信息，包括名称、部署位置、网关运行状态、服务入口方式、控制节点/工作节点的健康状态、API、插件等信息。

进入网关详情页面

在网关列表页面，选择目标网关的名称，即可进入网关概览页面`。



概览页面

网关详情

网关详情分为：基本信息、网络信息、TOP10 热门 API 情况、资源情况、资源负载、插件信息等部分。

部分数据的说明如下：

- 控制/工作节点实例数：显示节点实例的总数和健康状况。/左侧的数字表示当前在线的实例数，右侧的数字表示节点实例总数。

    - 如果在线节点/左侧数字等于全部应在线节点/右侧数字，则显示为绿色。
    - 如果在线节点/左侧数字小于全部应在线节点/右侧数字，则显示为红色。
- Top 10 APIs：按照 API 响应代码 2xx、4xx、5xx 的数量由高到低排列的前 10 个 API，默认按照响应代码 200 的数量降序排列。可查看过去 15 分钟、过去 1 小时、过去 24 小时的数据。
- 资源负载：以折线图的形式显示网关控制节点和工作节点在过去 1 小时内或过去 3 小时内的 CPU 使用情况和 Memory 使用情况。
- 插件信息：显示当前网关的插件启停情况以及其他的可用插件。

相关操作

除了查看网关详情，您还可以通过网关详情页面进行更新网关配置、删除网关、诊断网关等操作。

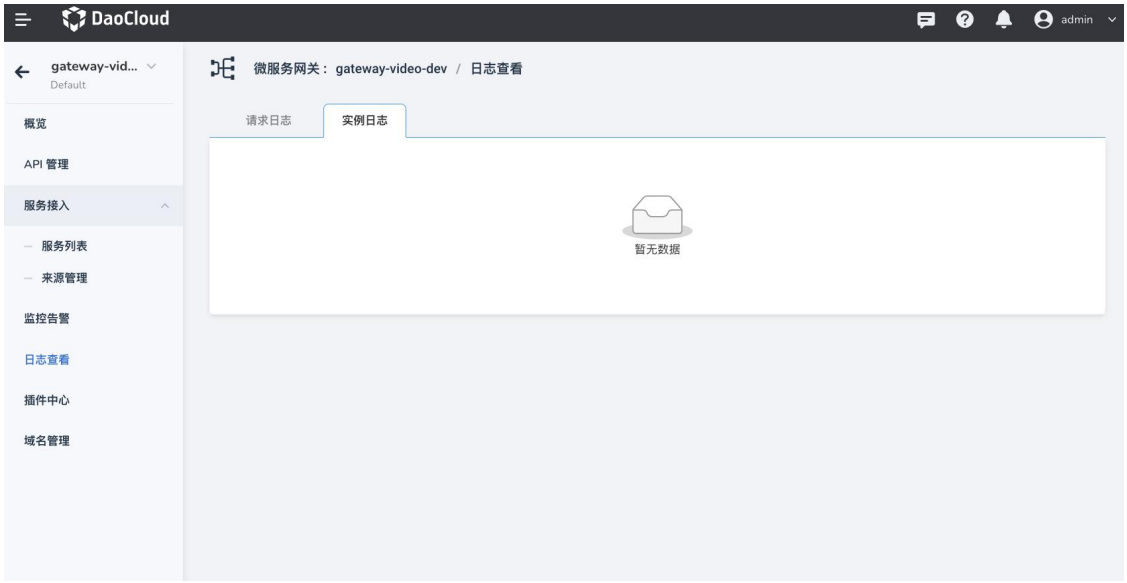
- 更新网关：在页面顶部点击编辑可以跳转到更新网关配置的页面，具体步骤参见更新网关配置。
- 删除网关：在页面顶部点击 ··· 后选择删除可以跳转到删除网关的页面，具体步骤参见删除网关。

实例日志

微服务网关支持查看请求日志和实例日志。本页介绍如何查看实例日志以及查看日志时的相关操作。

查看方式

点击目标网关的名称，进入网关概览页面，然后在左侧导航栏点击日志查看–>实例日志。



相关操作

- 筛选日志：支持筛选实例仅查看某个容器组的日志，也可以参考 KQL 语法直接输入关键字查找特定内容。
- 限定时间范围：日志时间范围可以选择：近 5 分钟的日志、近 1 小时、近 12 小时、近 7 天的日志，或者自定义时间范围。
- 自动刷新：在查看日志时，开启自动刷新可以查看实时日志。

!!! info

如需查看更多日志或下载日志，可前往[可观测性模块](../../../insight/intro/index.md)通过[日志查询](../../../insight/user-guide/data-query/log.md)功能查询或下载特定集群、命名空间、工作负载、容器组的日志。

请求日志

微服务网关支持查看请求日志和实例日志。本页介绍如何查看实例日志以及查看日志时的相关操作。

查看方式

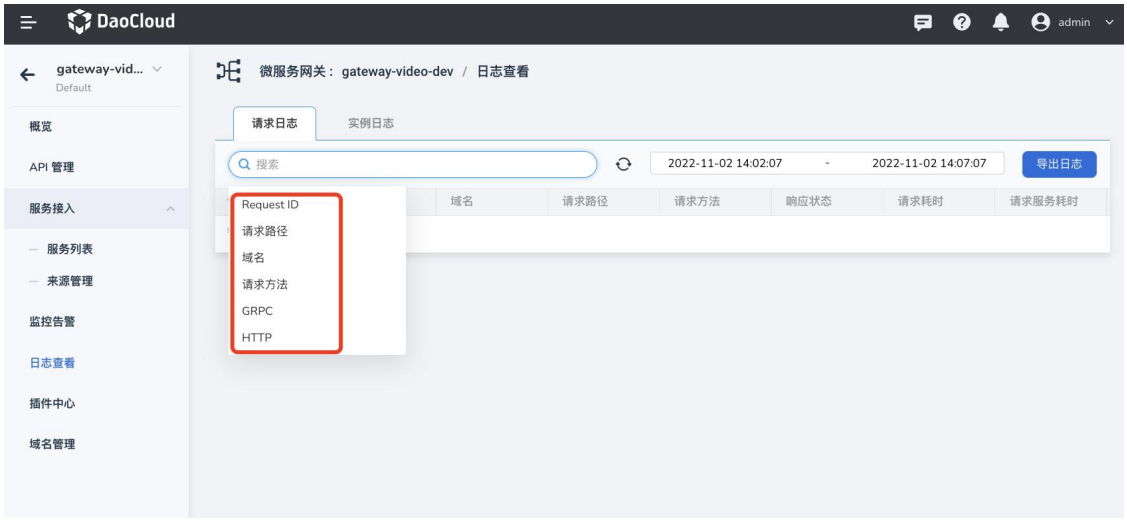
点击目标网关的名称，进入网关概览页面，然后在左侧导航栏点击日志查看–>请求日志。



查看请求日志的路径

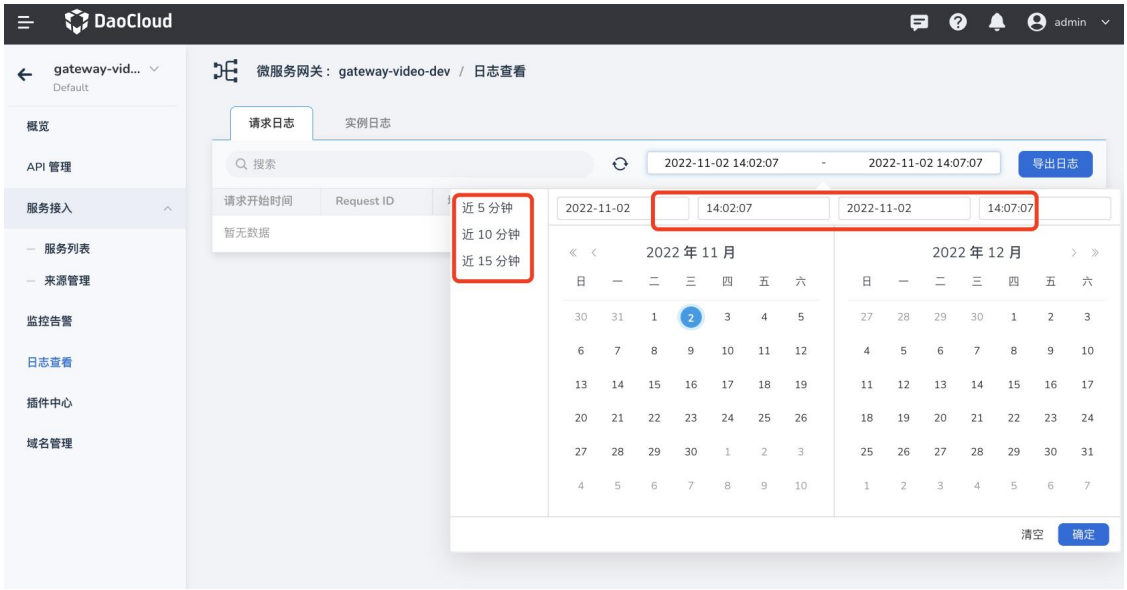
相关操作

- 筛选日志：支持通过 Request ID、请求路径、域名、请求方法、HTTP、GRPC 等条件筛选日志，支持按照请求开始时间、请求耗时、请求服务耗时对日志进行排序。



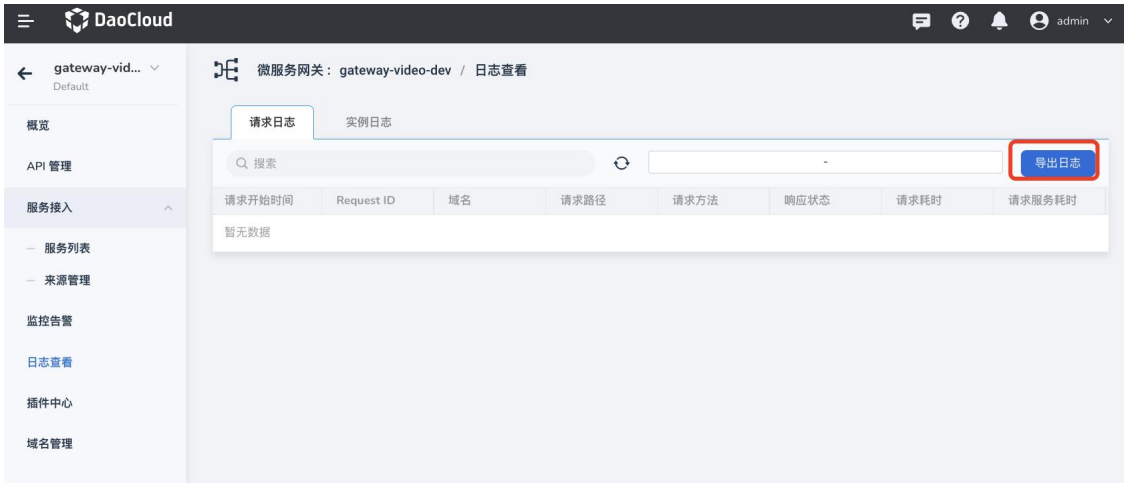
筛选日志

- 限定时间范围：可选择近 5 分钟、10 分钟、15 分钟的日志，或者自定义时间范围。



限定时间

- 导出日志：支持将日志文件导出到本地。


DaoCloud
admin
gateway-vid...
Default
概览
API 管理
服务接入
服务列表
来源管理
监控告警
日志查看
插件中心
域名管理
微服务网关： gateway-video-dev / 日志查看
请求日志
实例日志
搜索
导出日志
请求开始时间
Request ID
域名
请求路径
请求方法
响应状态
请求耗时
请求服务耗时
暂无数据

导出日志

移除 API

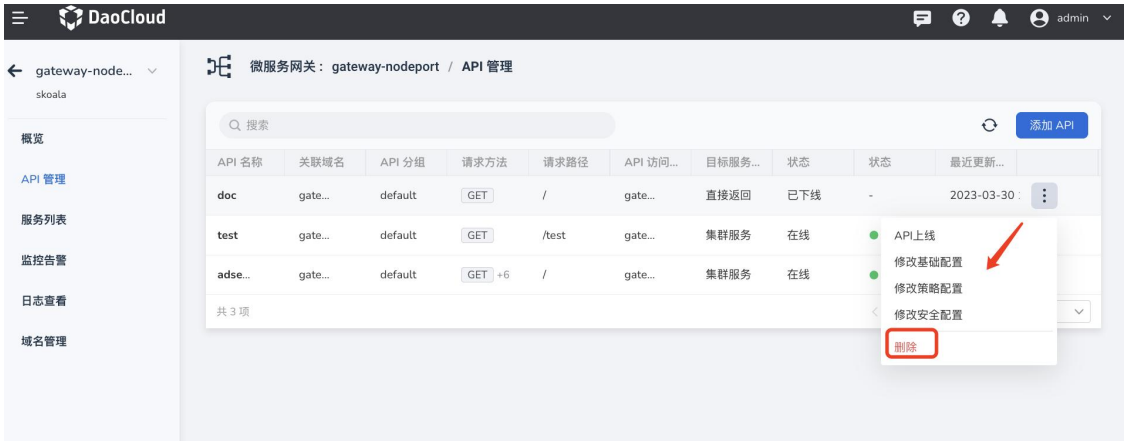
微服务网关支持对网关实例的 API 进行全生命周期管理，包括 API 的添加、更新和移除。本页介绍如何移除 API。

可以通过两种方式移除 API。

!!!danger

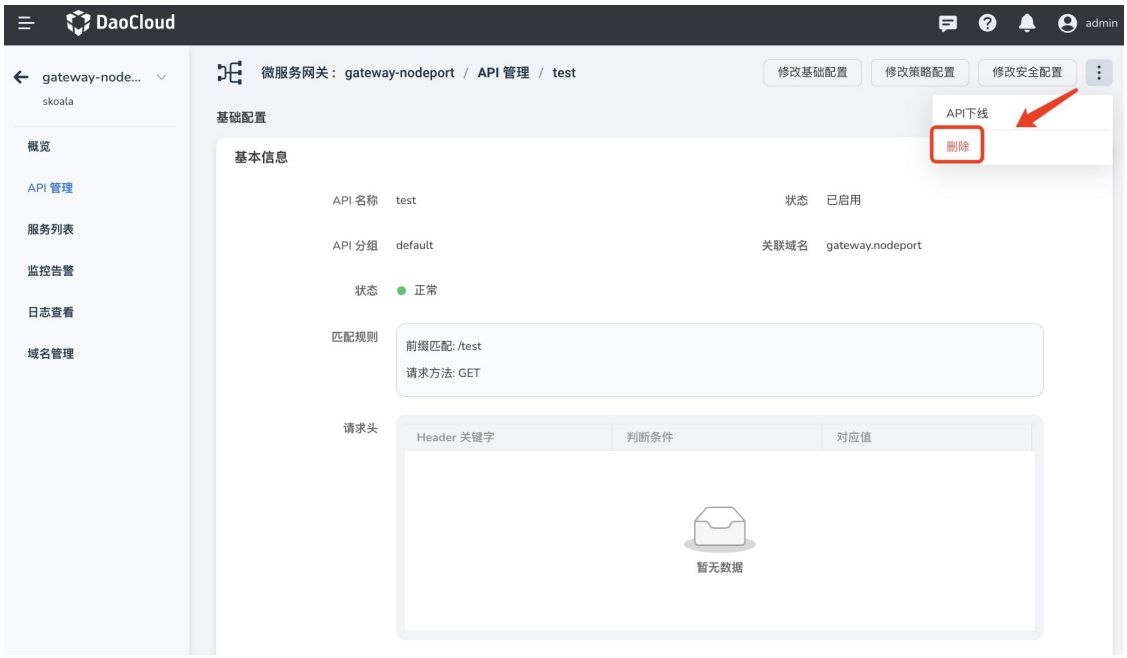
注意：删除操作是不可逆的。无论 API 是否处于在线状态，删除后均立即失效并且不可恢复。

- 在 API 管理页面找到需要删除的 API，在该 API 的右侧点击 并选择移除。



在列表页删除

- 点击 API 名称进入 API 详情页，在页面右上角点击 操并选择移除。


DaoCloud
admin
gateway-node...
skoala
概览
API 管理
服务列表
监控告警
日志查看
域名管理
微服务网关： gateway-nodeport / API 管理 / test
修改基础配置
修改策略配置
修改安全配置
API下线
删除
基础配置
基本信息
API 名称 test
状态 已启用
API 分组 default
关联域名 gateway.nodeport
状态 正常
匹配规则
前缀匹配: /test
请求方法: GET
请求头
Header 关键字
判断条件
对应值
暂无数据

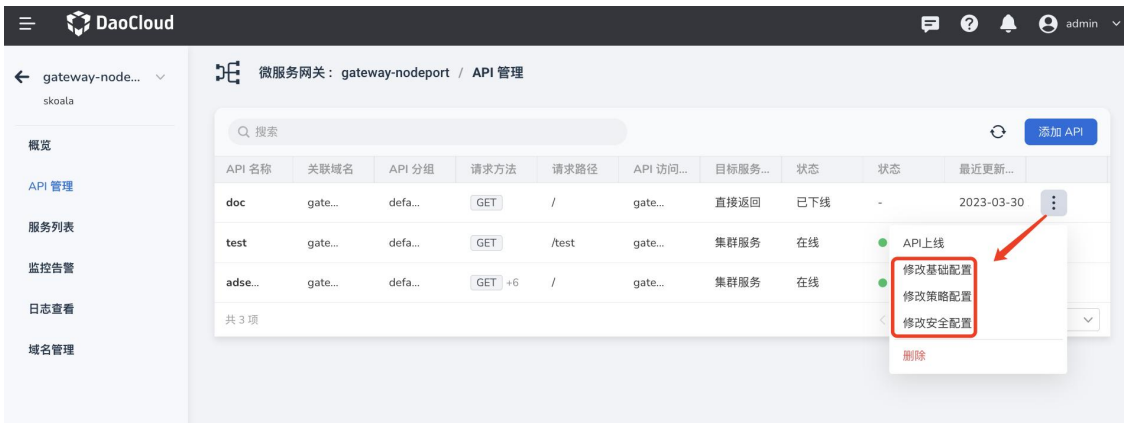
在详情页删除

更新 API

微服务网关支持对网关实例的 API 进行全生命周期管理，包括 API 的添加、更新和删除。本页介绍如何更新 API。

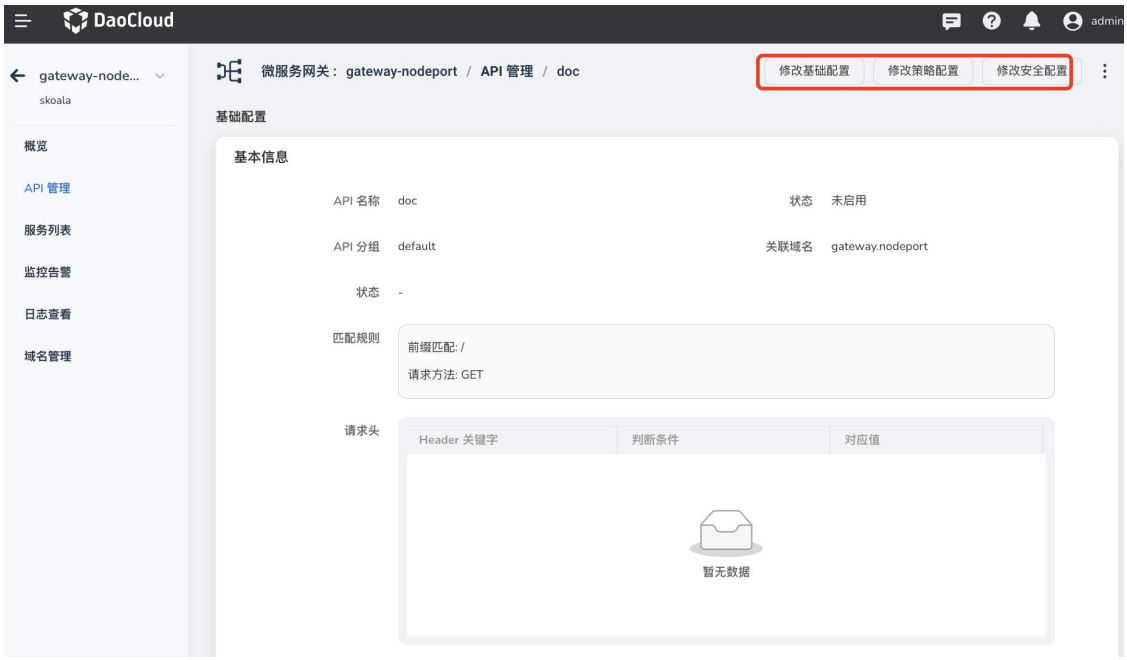
可以通过两种方式更新 API 的基础配置、策略配置和安全配置。

- 在 API 管理页面找到需要更新的 API，在该 API 的右侧点击 选择修改基础配置、修改策略配置或修改安全配置。



在列表页更新基础信息

- 点击 API 名称进入 API 详情页，在页面右上角修改基础配置、修改策略配置或修改安全配置。


DaoCloud
admin
gateway-node...
skoala
概览
API 管理
服务列表
监控告警
日志查看
域名管理
微服务网关： gateway-nodeport / API 管理 / doc
修改基础配置
修改策略配置
修改安全配置
基础配置
基本信息
API 名称 doc
状态 未启用
API 分组 default
关联域名 gateway.nodeport
状态 -
匹配规则
前缀匹配: /
请求方法: GET
请求头
Header 关键字
判断条件
对应值
暂无数据

在详情页更新

配置 API 策略

DCE 5.0 微服务网关支持九种 API 策略：负载均衡、路径改写、超时配置、重试机制、请求头重写、响应头重写、Websocket、本地限流、健康检查。可以单独使用某一种策略，也可以组合使用多种策略达到最佳实践。

有两种方式可以配置 API 策略：

- 在创建 API 的过程中设置策略，参考添加 API。
- 在 API 创建完成后通过更新 API 策略配置进行调整。

视频教程：

- API 策略的高级配置（1）
- API 策略的高级配置（2）

每一项策略的配置说明如下：

负载均衡

当 API 的目标后端服务为多实例服务时，可以通过负载均衡策略控制流量分发，根据业务场景调整不同服务的实例接收到的流量。

- 随机

  默认的负载均衡策略。选择随机规则时，网关会将请求随机分发给后端服务的任意实例。在流量较小时，部分后端服务可能会负载较多。效果参考下图：

```
☁  pythonProject  python3 lb_count.py
指向10.244.1.135的流量为： 89
指向10.244.2.207的流量为： 107
指向10.244.3.193的流量为： 104
```

负载均衡

- 轮询

  向后端服务的所有实例轮流分发请求，各个服务实例接收到的请求数基本相近。此规则可以在流量较小时保障流量的平均分配。效果参考下图：

```
☁  pythonProject  python3 lb_count.py
指向10.244.1.135的流量为： 101
指向10.244.2.207的流量为： 99
指向10.244.3.193的流量为： 100
```

负载均衡

- 权重

  根据 API 目标后端服务的权重分发流量，权重数值越大，优先级越高，承担的流量也相对较多。服务权重的配置入口见下图：



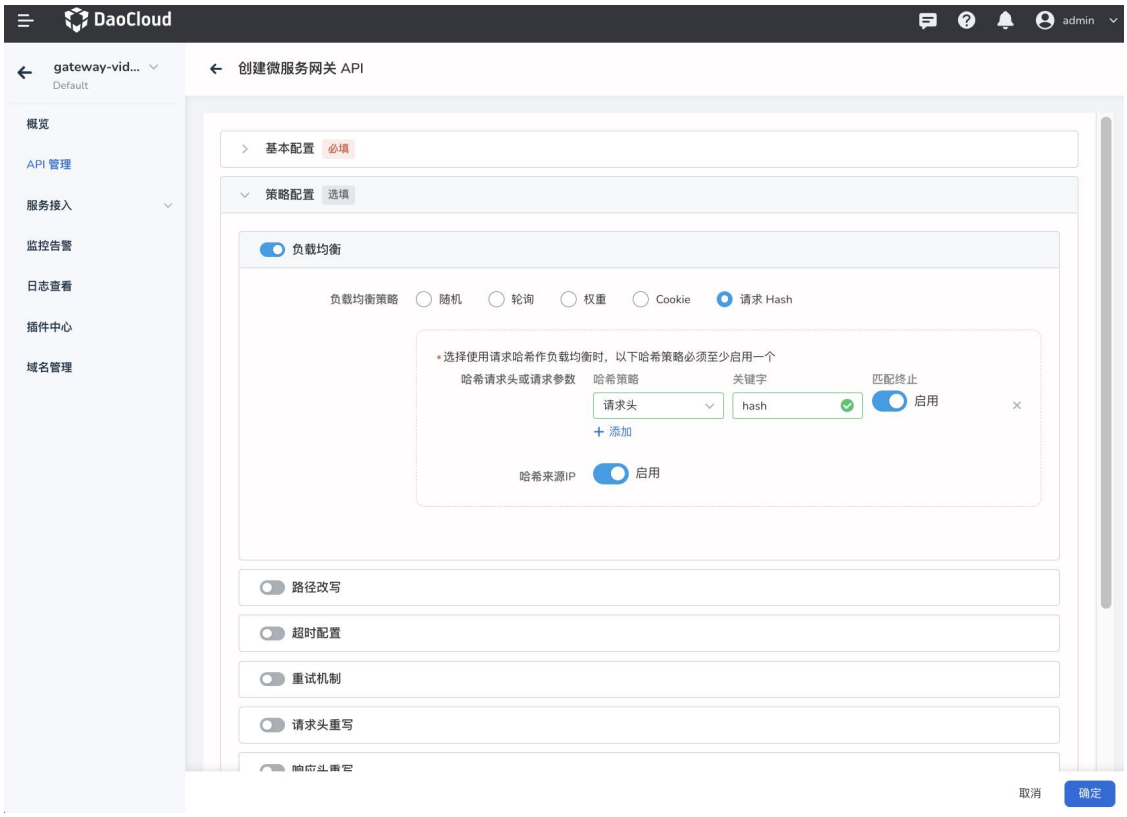
负载均衡

- Cookie

  将来源请求头中属于相同 Cookie 的流量分发到固定的后端服务实例，前提是后端服务能够根据 Cookie 做出不同的响应处理。

- 请求 Hash

  选择请求 Hash 时，可以通过一些高级策略来进负载均衡分配。当前支持的 Hash 策略为：IP、请求参数。

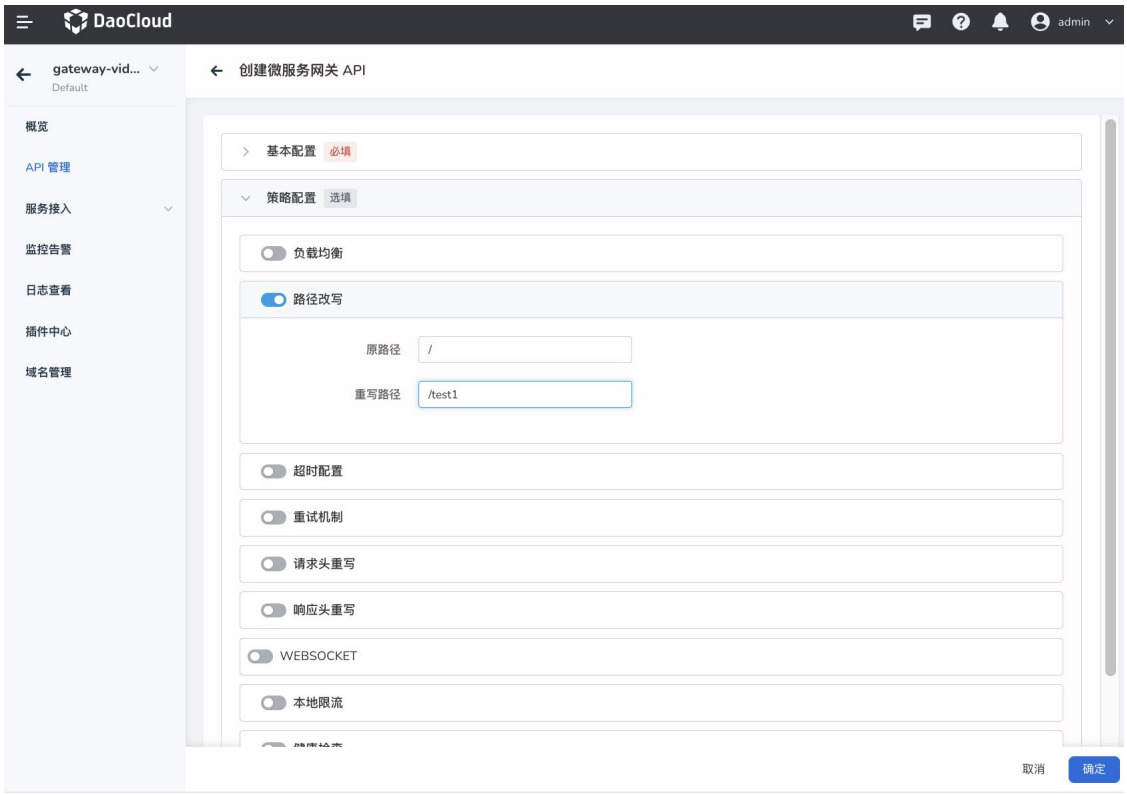


负载均衡

路径改写

如果对外暴露的 API 路径与后端服务提供的路径不一致，可以改写 API 路径使其与后端服务的路径一致，确保服务的正常访问。 启用路径改写后，网关会将外部请求流量转发到重写后的路径。

注意：**需要确保重写的路径是真实存在的，并且路径正确，以“/”开头。**

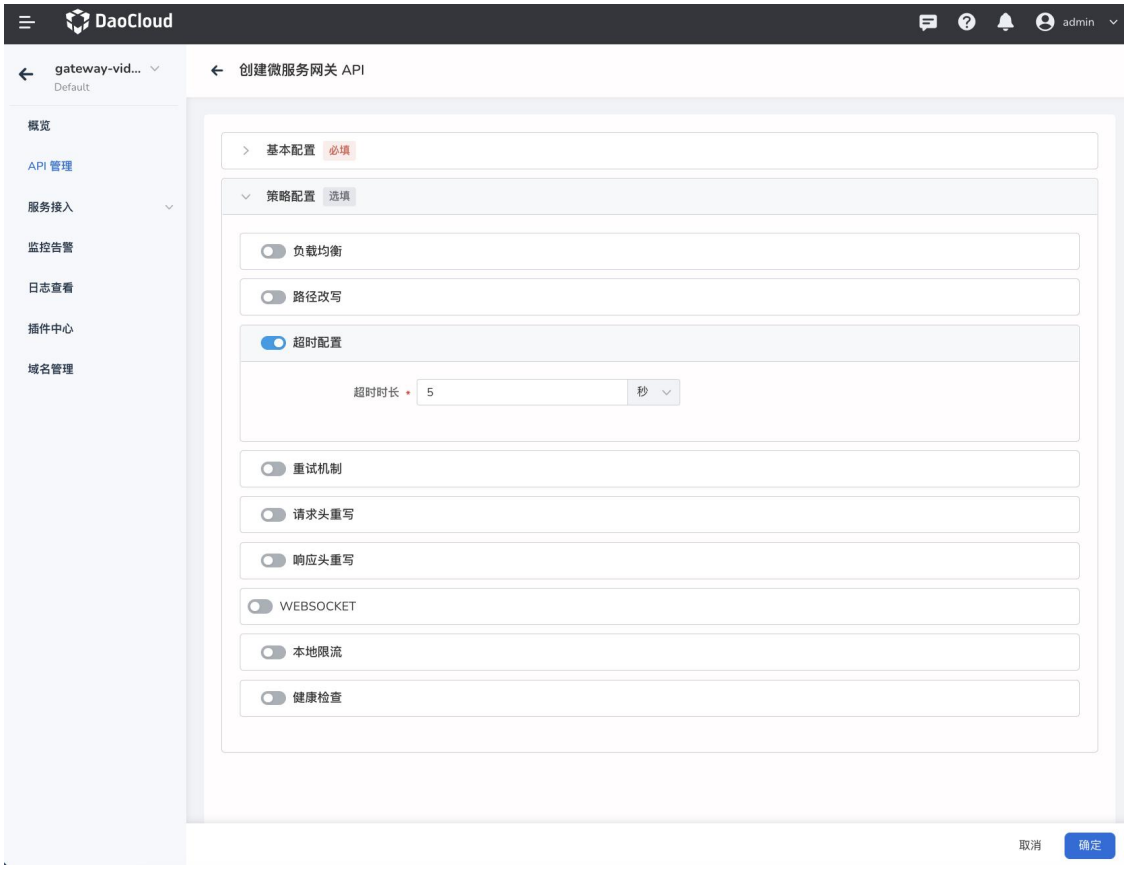


路径改写

超时配置

设置请求响应的最大时长，如果超出所设置的最大时长，则直接请求失败。超时时长支持数值类型为 >=1 的整数值，时间单位为“秒（s）”。

超时配置默认处于关闭状态，开启后必须配置超时时长。开启超时配置有助于减少异常处理导致的阻塞问题。



超时

重试机制

微服务网关的 API 支持配置非常丰富的重试机制。启用重试机制后，网关会在请求失败时自动重新尝试访问。达到重试超时时间之后自动触发再次重试，当重试次数达到配置的最大重试次数时停止重试。重试机制默认处于关闭状态，开启后必须配置重试次数和重试超时时长。

支持通过自定义配置选择不同的重试条件，自定义重试状态码等。

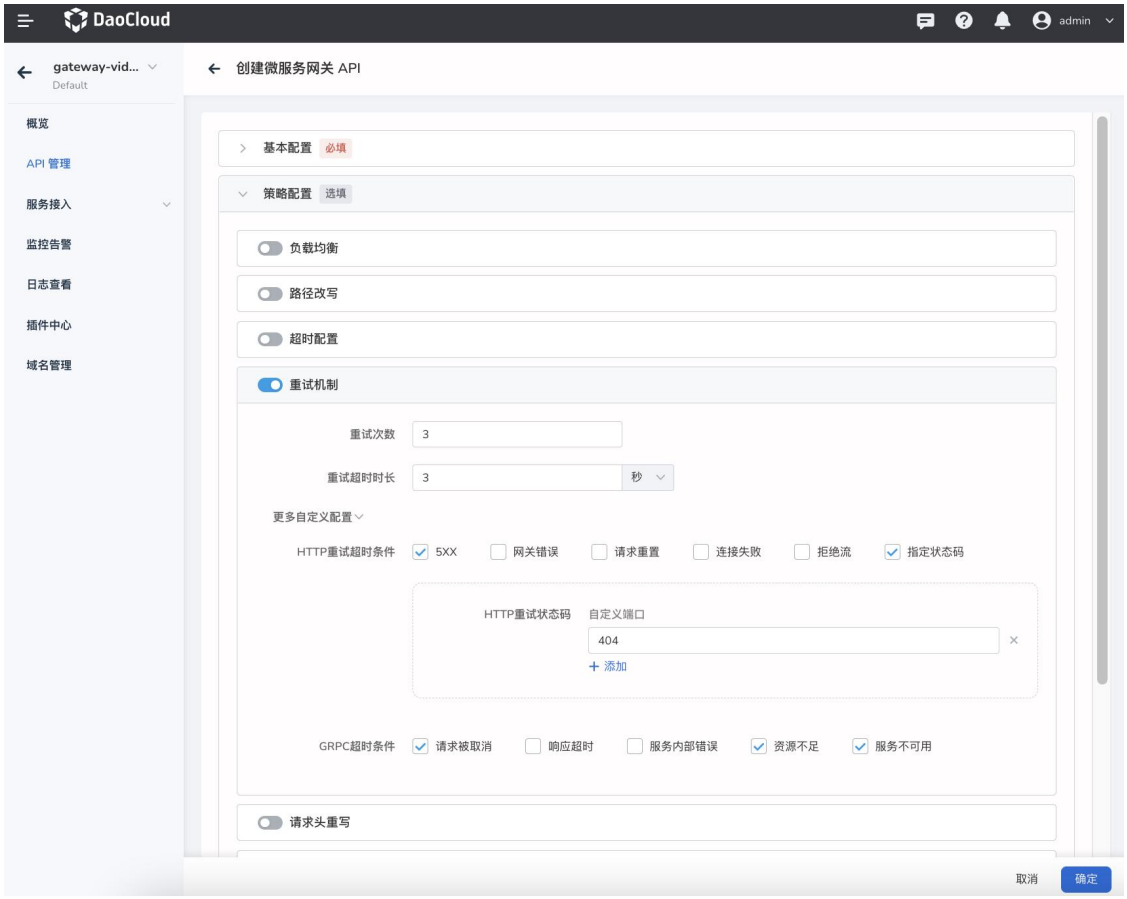
HTTP 重试

- 5XX 响应错误：后端服务响应 HTTP status_code 大于 500 时进行重试。
- 网关错误：当响应结果为网关错误提示时，自动进行重试。
- 请求重置：当响应结果为请求重置消息时，自动进行重试。

- 连接失败：当响应结果为网络连接失败的返回时，自动进行重试。
- 拒绝流：当响应结果为后端服务将请求标记为拒绝处理时，自动进行重试。
- 指定状态码：当后端服务响应 HTTP status_code 为特定状态码时自动进行重试，支持配置特定的状态码。

GRPC 重试

- 请求被取消：当响应结果为 GRPC 请求被后端服务取消时自动进行重试。
- 响应超时： 当后端服务响应超时，自动进行重试。
- 服务内部错误：当响应结果为服务内部错误时，自动进行重试。
- 资源不足：当响应结果为资源不足时，自动进行重试。
- 服务不可用时：当响应结果为后端不可用时，自动进行重试。

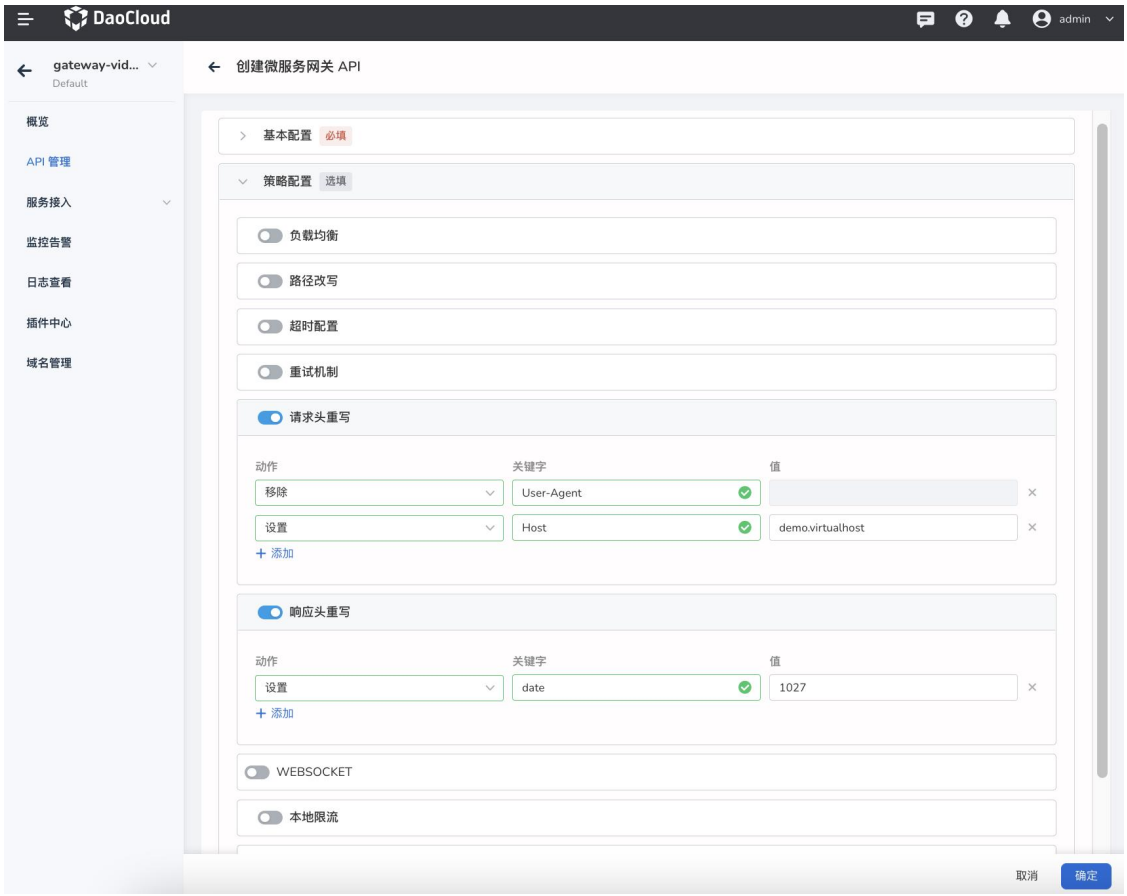


重试

请求头/响应头改写

支持添加、修改、删除请求头和响应头及其对应的值。

- 增加请求头/响应头：使用设置动作，填写新的关键字和新值。
- 修改请求头/响应头：使用设置动作，填写已有的关键字并赋予新值。
- 移除请求头/响应头，使用移除动作，只填写需要移除的关键字即可，无需填写对应的值。

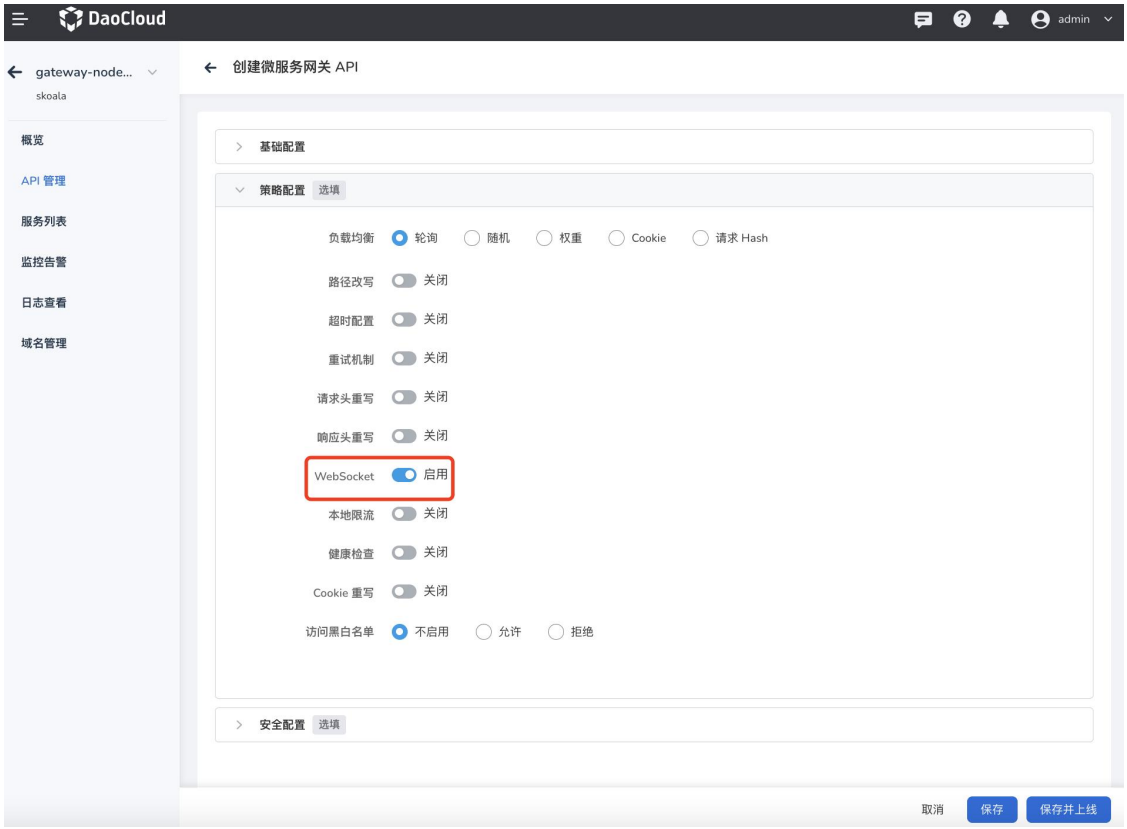


header 改写

Websocket

WebSocket 是一种在单个 TCP 连接上进行全双工通信的协议。WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

启用 Websocket 之后支持通过 Websocket 协议访问 API 的后端服务。



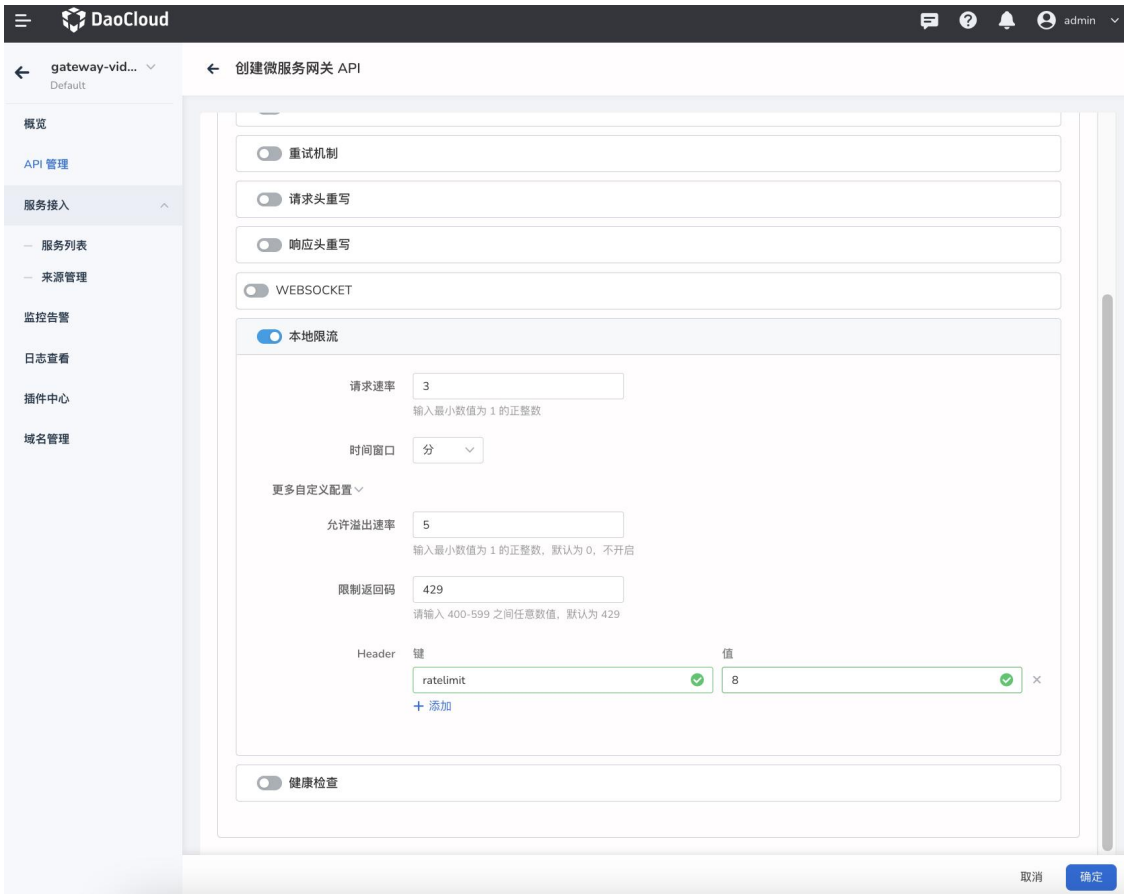
websocket

本地限流

微服务网关支持丰富的限流能力，支持在 API 层级启用本地限流能力。

- 请求速率：时间窗口（秒/分/时）内允许的最大请求速率，例如每分钟最多允许 3 次请求。支持输入 >=1 的整数。
- 允许溢出速率：达到预设的请求速率时，仍旧允许额外处理一部分请求，适用于业务突增的流量高峰时段。支持输入 >=1 的整数。
- 限制返回码：默认返回码为 429，表示请求次数过多。可参考 envoy 官方文档了解本地限流支持的状态码。
- Header 关键字：默认为空，可根据需求自行设置。

下图中的配置表示：每分钟最多允许请求 8 次 (3+5)，第 9 次访问时会返回 429 状态码，提示访问次数过多。每次请求成功后返回的响应内容都会带上 ratelimit：8 响应头。



本地限流

!!! info

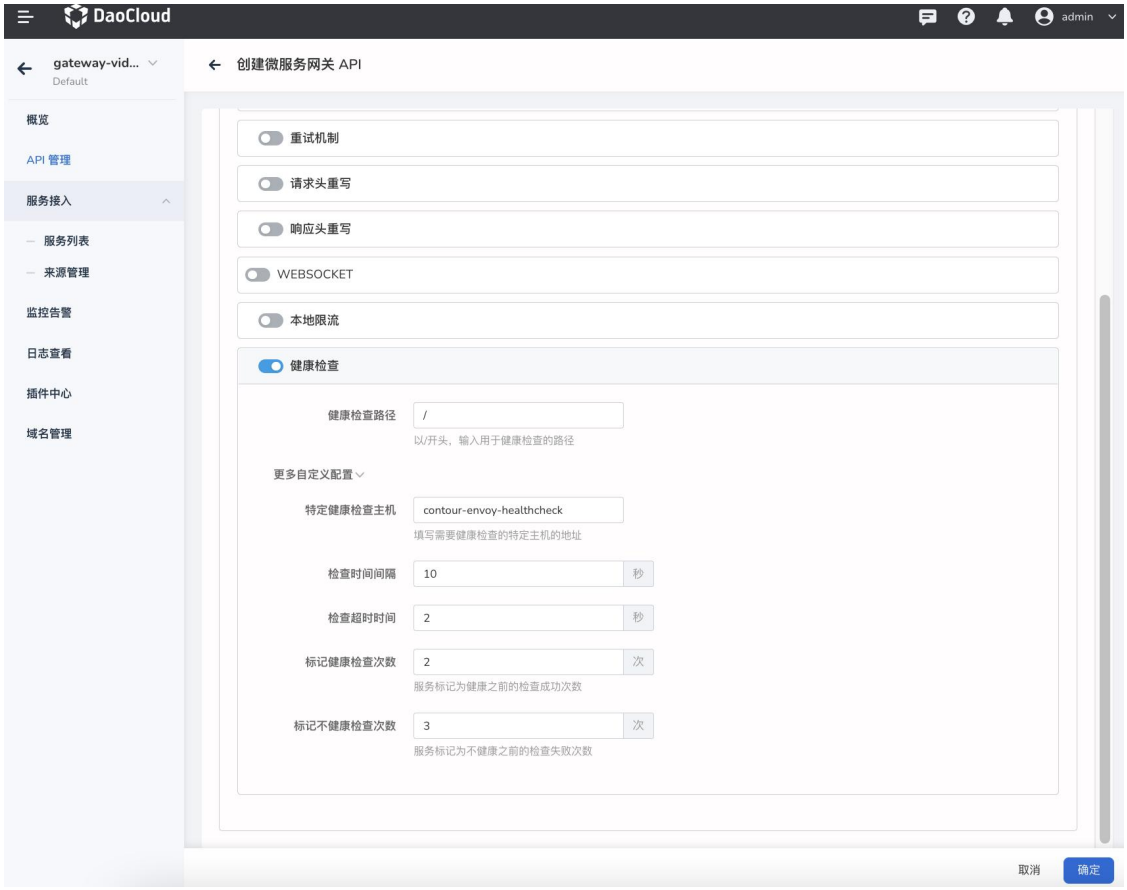
除了在 API 层级的本地限流能力之外，还可以通过[配置域名策略](../domain/domain-policy.md)针对整个域名进行限流处理。当 API 与域名同时配置限流策略时，以 API 层级的限流策略为准。

健康检查

通过设置健康检查地址，可以有效保证当后端服务异常时，网关自动进行负载均衡调整。对不健康的后端服务进行标记，停止向该服务分发流量。当后端服务恢复并通过设定的健康检查条件后，自动恢复流量分发。

- 健康检查路径：以“/”开头，并且全部后端服务的所有实例都应提供相同的健康检查接口。

- 特定健康检查主机：配置主机地址后，仅对该主机进行健康检查。
- 检查时间间隔：每次健康检查的时间间隔，时间单位为“秒”，例如每隔 10 秒进行一次健康检查。
- 检查超时时间：健康检查的最大超时时长，当健康检查超过配置的时长时，直接标记健康检查失败。
- 标记健康检查次数：连续检查 N 次并且每次结果都是健康时，才将服务实例标记为健康状态；当服务实例被标记为健康状态后，请求流量将会自动分发到该服务实例。
- 标记不健康检查次数：连续检查 N 次并且每次结果都是不健康时，就将服务实例标记为不健康状态，当服务实例被标记为不健康时，停止向该实例分发请求流量。



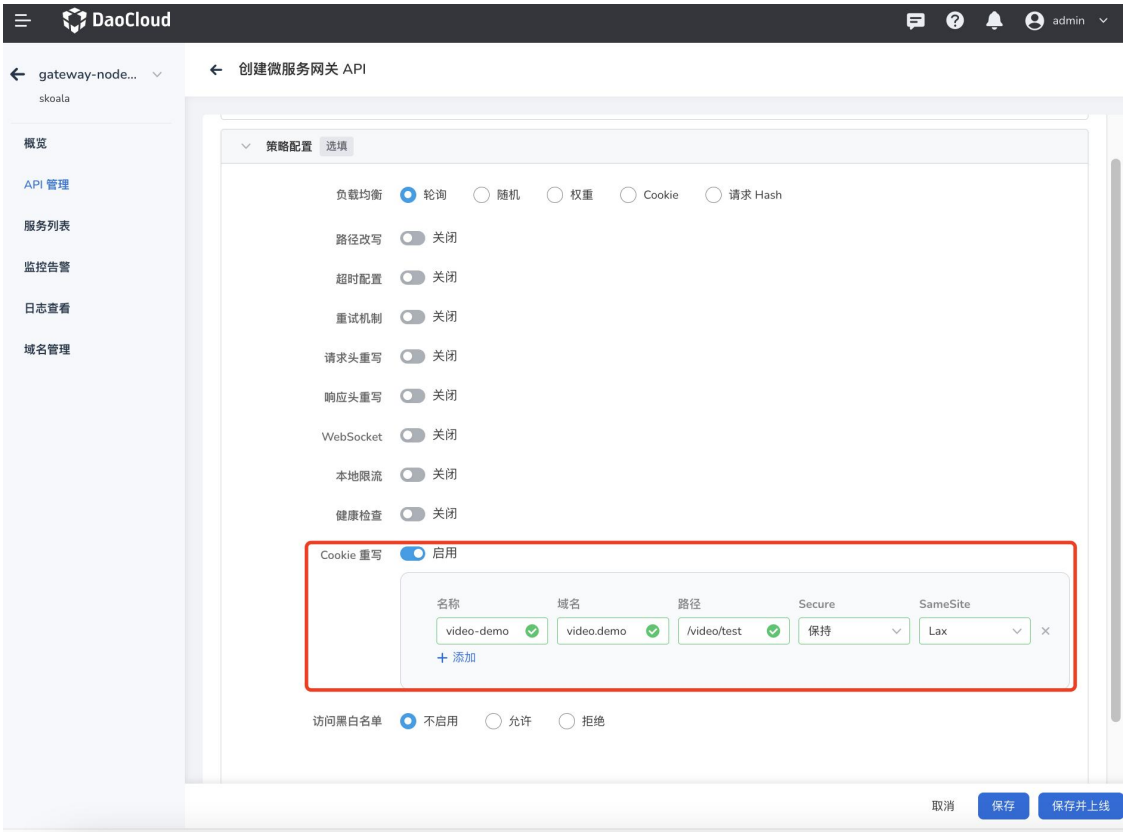
健康检查

Cookie 重写

参考下方说明配置 cookie 重写策略：

- 名称：必须填写当前已经存在的 cookie 名称
- 域名：重新定义 cookie 的域名
- 路径：重新定义 cookie 的路径
- Secure：保持指启用安全模式，禁用指禁用安全模式。在安全模式下，请求必须为安全连接（HTTPS），cookie 才会被保存下来。如果使用 HTTP 协议下，cookie 将无效
- Samesite：是否在跨域时发送 cookie

- Strict：跨域请求严禁携带本站 cookie
- Lax：大多数情况禁止，但是导航到目标网址的 Get 请求除外。
- None：跨域请求允许携带本站 cookie，前提是 Secure 必须设置为保持，即只能在 HTTPS 协议下使用



cookie 重写

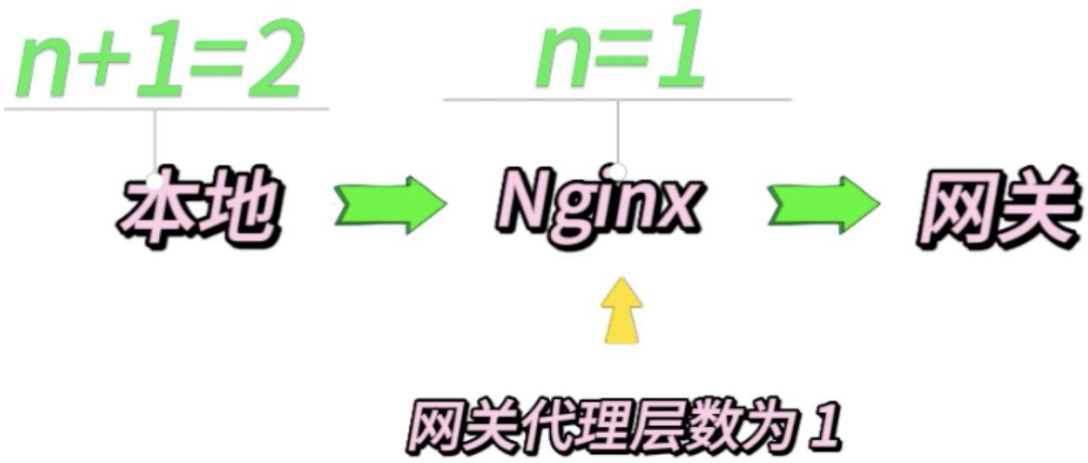
访问黑白名单

启用访问黑白名单后，可以仅允许白名单上的 IP 请求通过网关，拒绝其他所有来源的请求；或者拒绝黑名单上的 IP 请求通过网关，允许其他所有来源的请求。

- 网关前置代理层数：请求从客户端到网关中途需要经过几个代理端点。例如 **客户端–Nginx–网关** 的代理层数为 1，因为中间只经过 1 个 Nginx 代理端点。

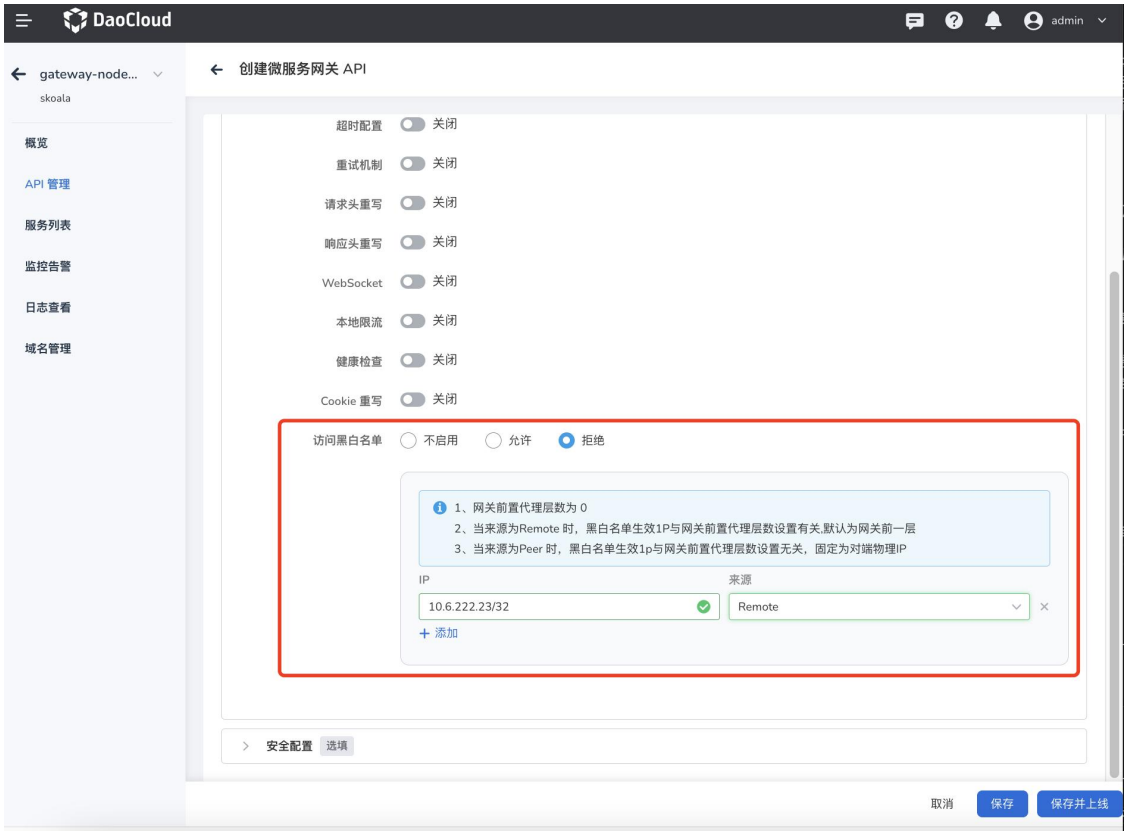
创建/更新网关时，可以在网关的高级配置部分设置代理层数，需要按照实际情况填写。

- Remote：IP 来源为 Remote 时，黑白名单是否生效取决于网关前置代理层数。当代理层数为 n 时，生效的是从网关开始向前第 n+1 个端点的 IP。例如 **客户端–Nginx–网关** 前置代理层数为 1，则仅对网关向前第 2 个端点的 IP 生效，即客户端的 IP。如果填写 Nginx 的 IP，黑白名单不会生效。



黑白名单

- Peer：IP 来源为 Peer 时，无论网关前置代理层数是多少，黑白名单都仅对网关的 **直接** 对端 IP 生效。例如客户端–...–Nginx–网关，无论客户端和 Nginx 中间有多少个代理端点，黑白名单都仅对最后一个 Nginx 的 IP 生效。


DaoCloud
admin
gateway-node...
skoala
概览
API 管理
服务列表
监控告警
日志查看
域名管理
创建微服务网关 API
超时配置 关闭
重试机制 关闭
请求头重写 关闭
响应头重写 关闭
WebSocket 关闭
本地限流 关闭
健康检查 关闭
Cookie 重写 关闭
访问黑白名单 不启用 允许 拒绝
1、网关前置代理层数为 0
2、当来源为Remote 时，黑白名单生效1P与网关前置代理层数设置有关,默认为网关前一层
3、当来源为Peer 时，黑白名单生效1p与网关前置代理层数设置无关，固定为对端物理IP
IP
来源
10.6.222.23/32
Remote
添加
安全配置 选填
取消
保存
保存并上线

黑白名单

添加 API

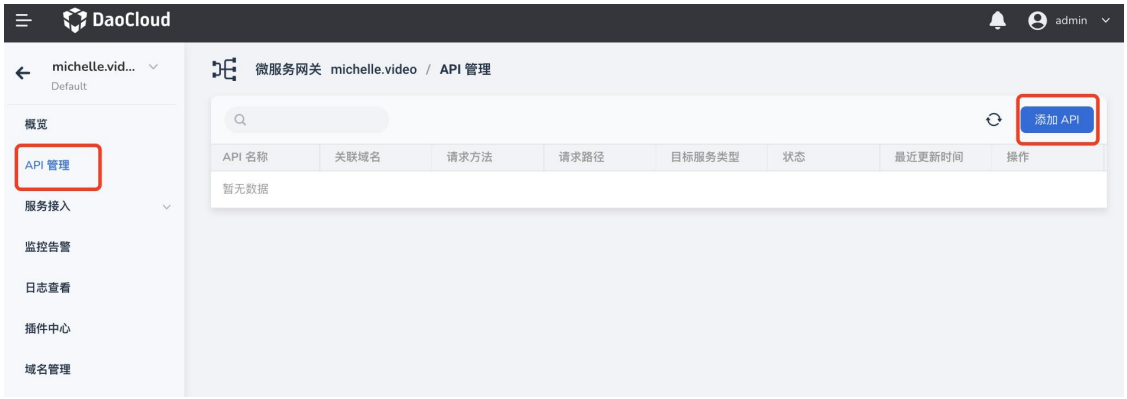
微服务网关支持对网关实例的 API 进行全生命周期管理，包括 API 的添加、更新和删除。本页介绍如何添加 API。

前提条件

- 有可选的域名，可参考域名管理创建域名。
- 如果 API 的目标服务为后端服务，则需要确保有可选的后端服务，可参考通过手动或自动方式接入服务。

创建 API 的步骤如下：

1. 点击网关名称进入网关概览页面，然后在左侧导航栏点击 API 管理，在页面右上角点击 添加 API。



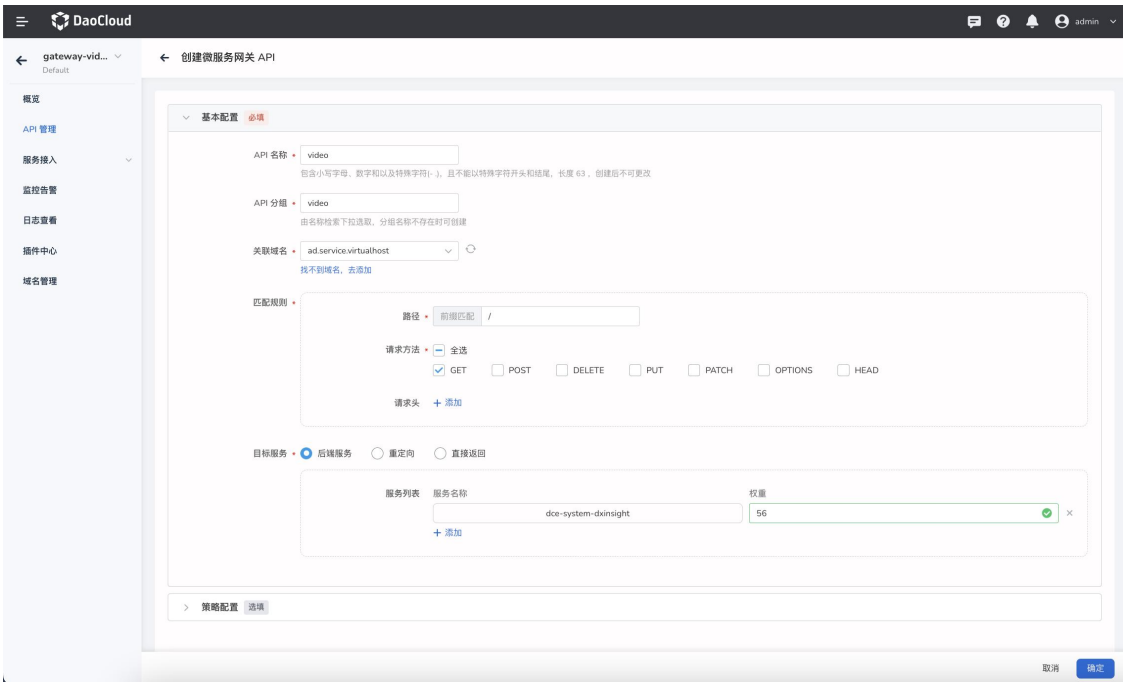
进入添加页面

2. 参考下方说明填写基本配置。

   配置分为基本配置和策略配置两部分。填写基本配置信息时需要注意：

   - API 名称：包含小写字母、数字和以及特殊字符(– .)，不能以特殊字符开头和结尾。

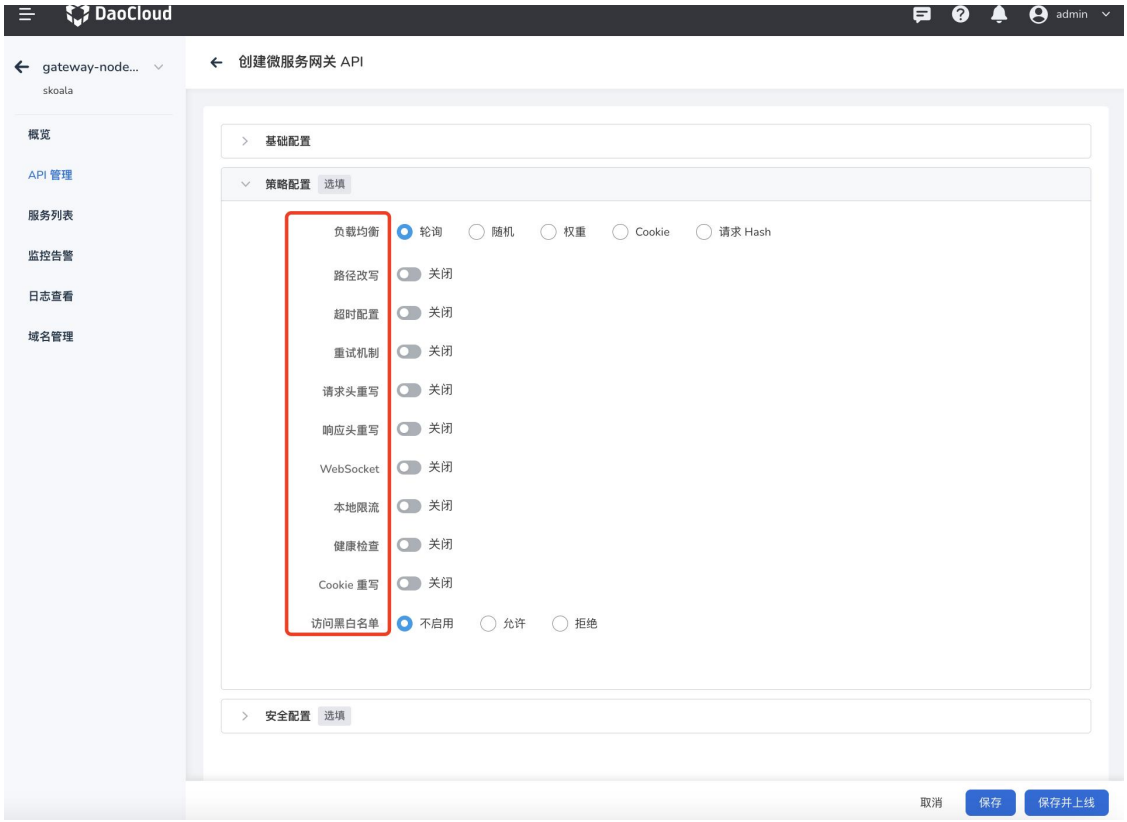
- API 分组：选择 API 所属的分组组名。如果输入不存在的分组名称，则自动创建一个新的分组。
- 关联域名：填写关联域名后，可以通过域名 + 端口号的方式访问 API。找不到域名时可以添加新域名，可参考添加域名。
- 匹配规则：只允许符合该规则的请求通过。如果设置了多条规则，需要同时满足所有规则才能放行。如果添加了请求头，则需要在访问该 API 时添加相应的请求头。
- 请求方法：选择 HTTP 协议的请求方式。有关各种请求方式的详细说明，可参考 W3C 的官方文档方式定义。
- 目标服务：选择将请求直接发送到后端服务，或者重定向到其他服务，或者直接返回 HTTP 状态码。
- 如果选择后端服务，则需要配置权重。权重越大，网关向其分发的流量就越多。



配置信息

3. 参考下方说明填写策略配置（选填）。

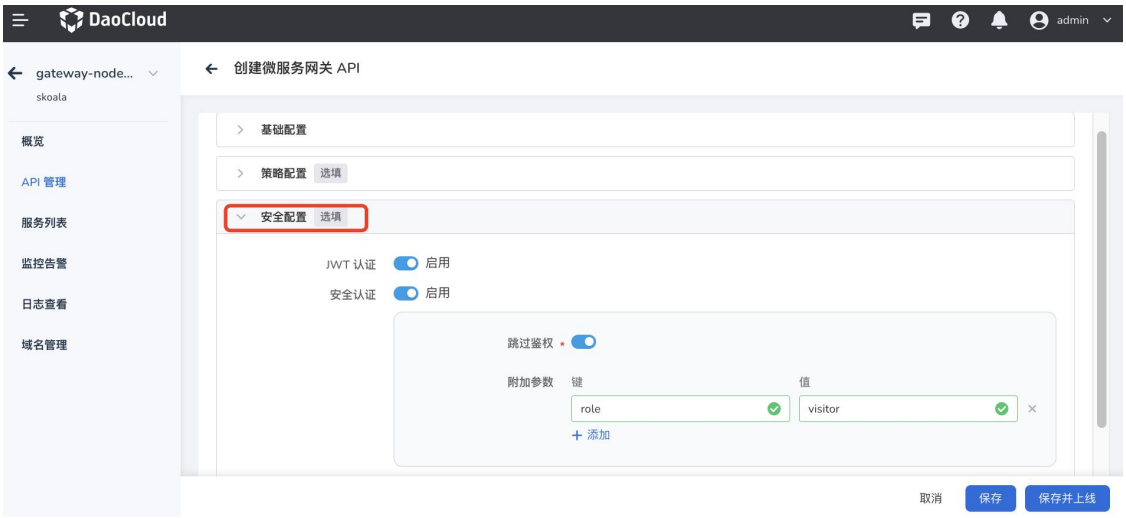
   支持 11 种 API 策略：负载均衡、路径改写、超时配置、重试机制、请求头重写、响应头重写、Websocket、本地限流、健康检查、cookie 重写、访问黑白名单。有关各项策略的配置说明，可参考 API 策略配置。



   配置策略

4. 参考下方说明填写安全配置（选填）。

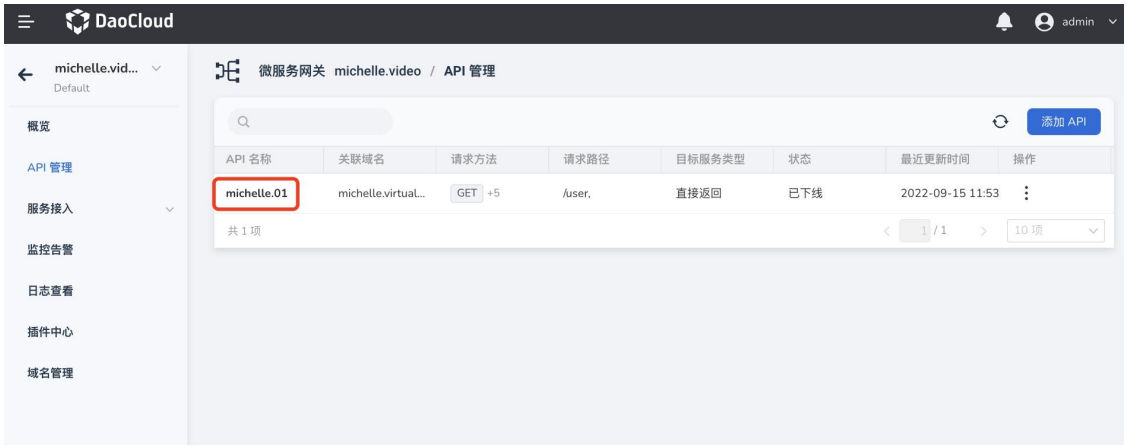
   - JWT 认证：开启或禁用 JWT 认证
   - 安全认证：开启或禁用安全认证



配置策略

5. 在页面右下角点击保存（不上线）。如果点击保存并上线则可以直接上线 API。

   点击确定后，如果所有配置都正常，右上角会弹出创建网关 API 成功的提示信息。可以在 API 管理页面查看新建的 API。

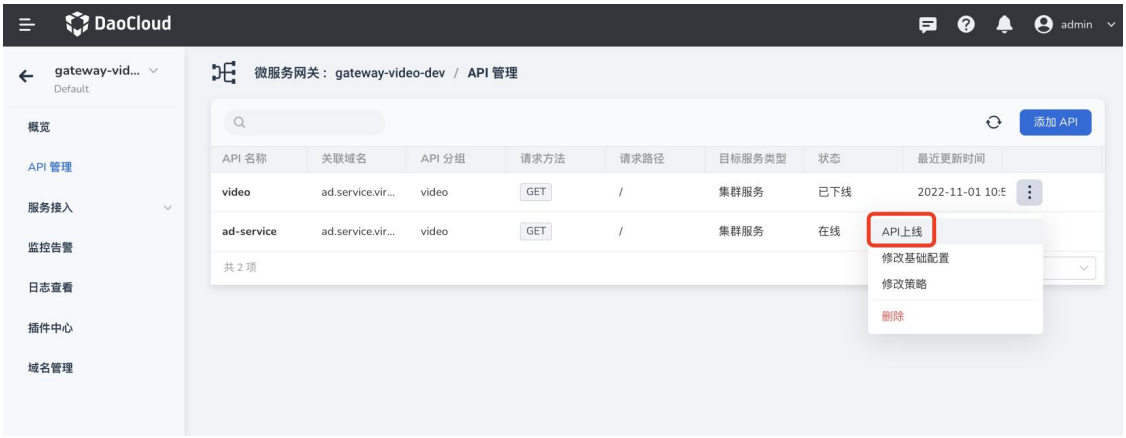


创建成功

6. API 上线

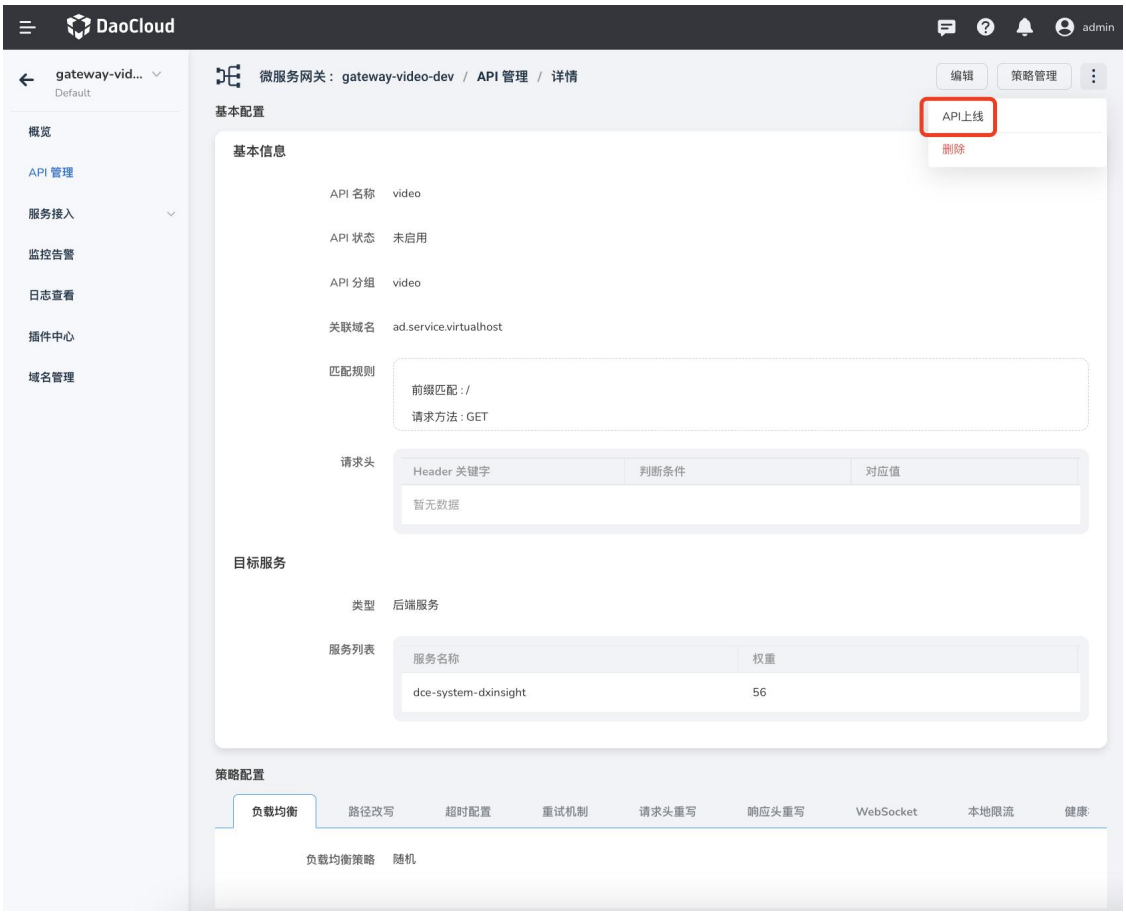
   API 创建成功后，**默认处于下线状态，此时无法访问**。需要将 API 调整为上线，才能正常访问。API 上线有两种方式。

- 在 API 在 API 管理页面找到需要更新的 API，在该 API 的右侧点击 选择 API 上线。



API 上线

- 点击 API 名称进入 API 详情页，在页面右上角点击 并选择 API 上线。



API 上线

!!! info

点击 API 名称进入 API 详情，可查看 API 的详细配置信息，例如上下线状态、域名、匹配规则、目标服务、策略配置等。

![API 上线](https://docs.daocloud.io/daocloud-docs-images/docs/skoala/ms-gateway/api/imgs/online1.png)

自动纳管服务

添加成功的服务会出现在服务列表页面，添加 API 时也可以选择列表中的服务作为目标后端服务。微服务网关支持通过手动接入和自动纳管两种方式添加服务。本页介绍如何自动纳管服务。

网关实例创建成功之后，服务来源中的服务会被自动添加到该网关实例的服务列表中，无需手动添加。

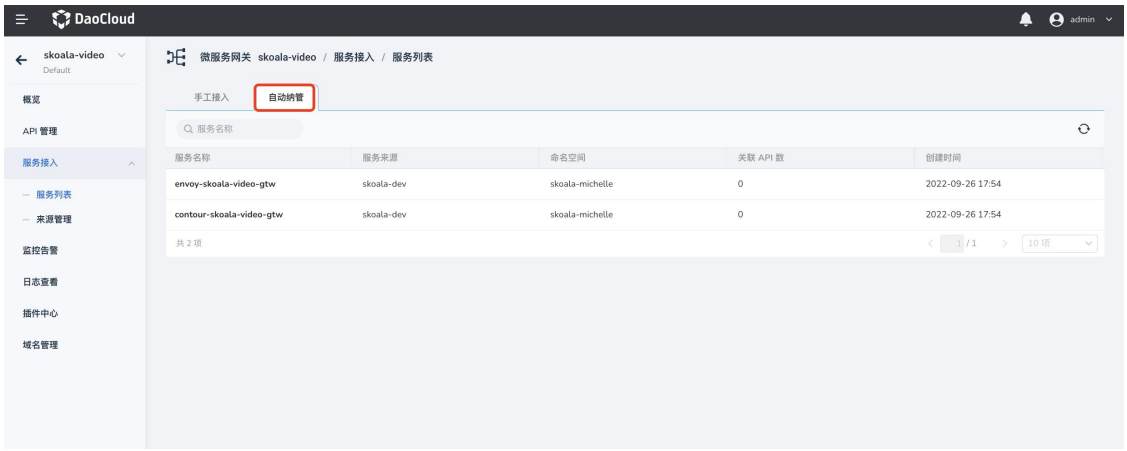
查看自动纳管的服务

1. 在微服务网关列表页面点击目标网关的名称，进入网关概览页后，在左侧导航栏点击服务接入—>服务列表。



服务列表

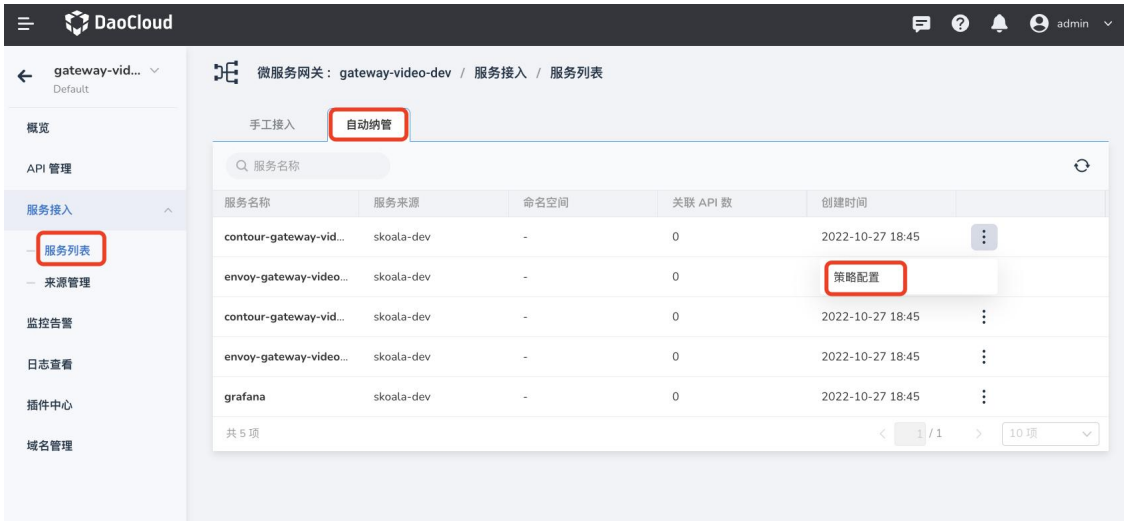
2. 在服务列表页面点击自动纳管。



自动发现服务

配置服务策略

1. 在服务列表–>自动纳管页面找到目标服务，在右侧点击　　选择策略配置。

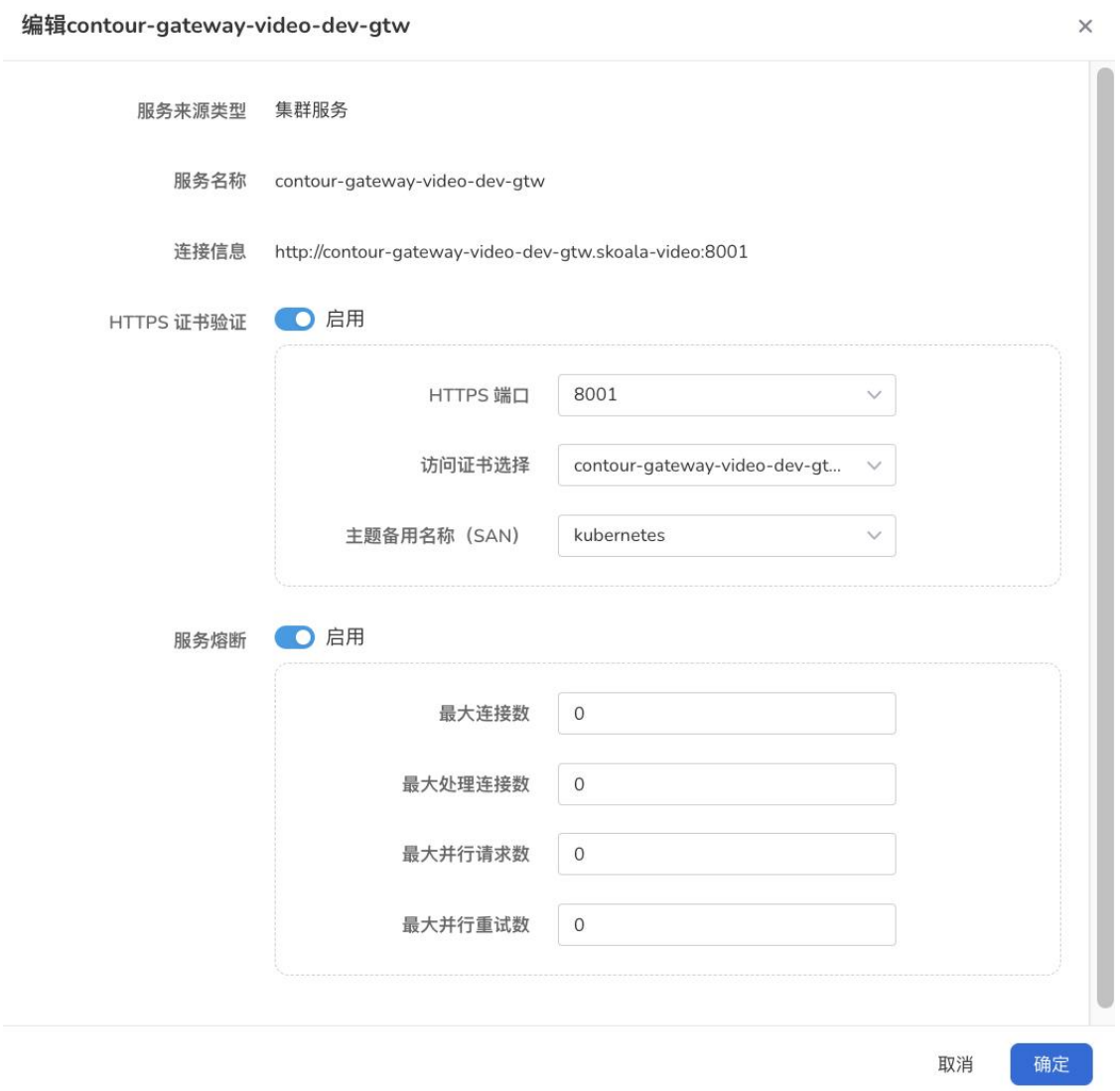


策略配置

2. 按需调整服务策略配置，在弹框右下角点击确定。

   - HTTPS 证书验证：开启后，必须通过证书校验才能成功访问该服务。
   - 服务熔断：当最大连接数、最大处理连接数、最大并行请求数、最大并行重试数 **任何一个** 指标达到设定的阈值时，自动切断对该服务

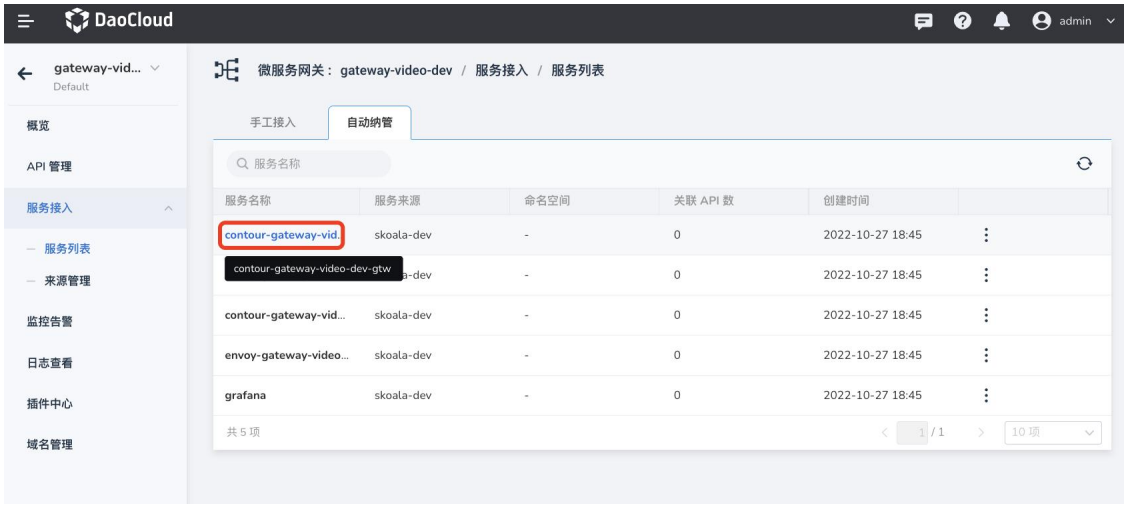
的调用，保护系统整体的可用性。当指标降到设定的阈值之后，自动恢复对该服务的调用。



策略配置

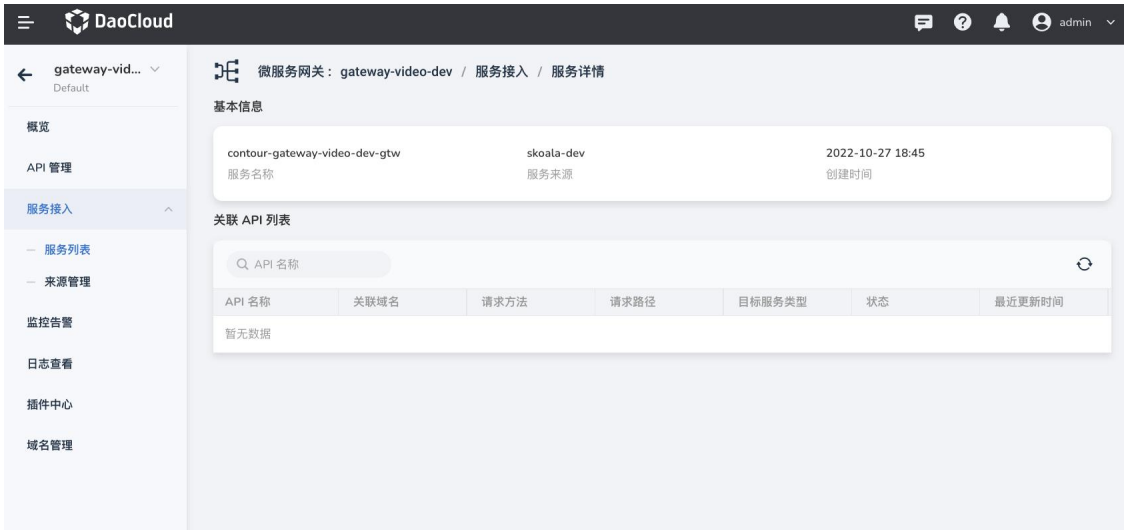
查看服务详情

1. 在服务列表–>自动纳管页面找到目标服务，点击服务名称。



服务详情

2. 查看服务名称、来源、关联的 API 等信息。支持安装最近更新时间进行排序。



服务详情

!!! info

对于自动纳管的服务，仅支持上述操作，不支持更新、删除服务的操作。

手动管理服务

添加成功的服务会出现在服务列表页面，添加 API 时也可以选择列表中的服务作为目标后端服务。微服务网关支持通过手动接入和自动发现两种方式添加服务。本页介绍如何手动接入服务。

接入服务

1. 在云原生网关列表页面点击目标网关的名称，然后在左侧导航栏点击服务列表，接着在右上角点击添加服务。



服务列表

2. 选择服务来源，配置服务连接信息，点击确定。

   ===“集群服务/网格服务”

   选择目标服务所在的集群和命名空间，填写访问协议、地址以及端口。

   ![添加集群服务](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-service01.png)

   对于集群服务的访问方式，可在`容器管理`->`容器网络`->`服务`中点击服务名称进行查看：

![获取服务访问地址](https://docs.daocloud.io/daocloud-docs-images/docs/skoala/ms-gateway/service/imgs/service-access.png)

===“接入注册中心服务/注册配置中心服务”

选择目标服务所在的注册中心，填写访问协议、地址和端口。

![添加注册中心服务](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-service04.png)

===“外部服务”

填写服务名称、访问协议、地址、端口。

![添加外部服务](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/skoala/images/gw-service02.png)

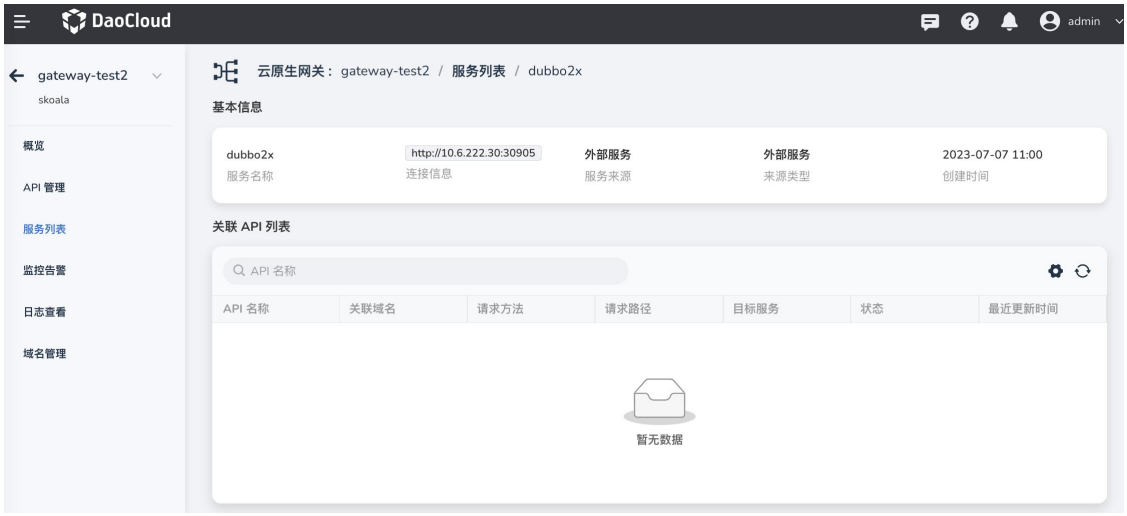
查看服务详情

1. 在服务列表页面点击目标服务的名称，进入服务详情页面。



服务详情

2. 查看服务来源、连接信息、关联 API 等信息。



服务详情

更新服务

更新基础配置

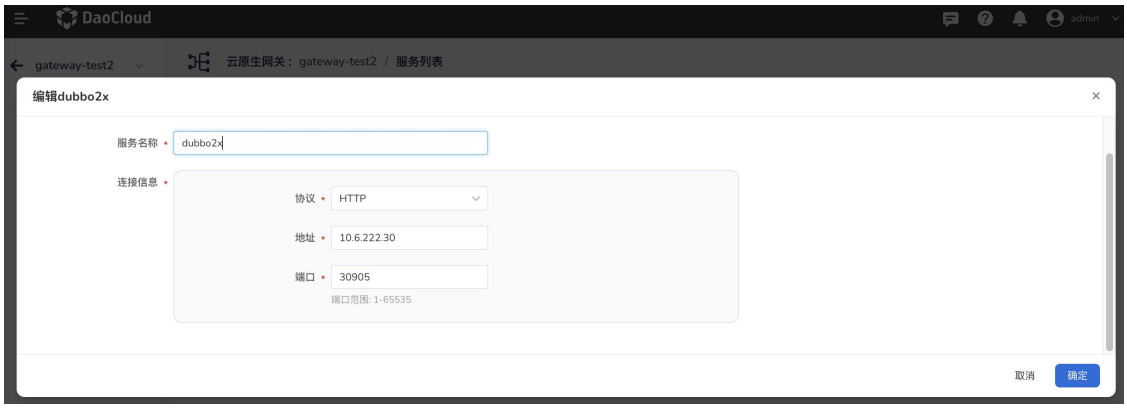
更新基础配置指修改服务的名称、协议、地址和端口等连接信息。

1. 在服务列表页面找到需要更新的服务，在服务右侧点击　，选择修改基础配置。



更新服务

2. 更新基本信息，点击确定。



更新服务

!!! danger

如果更新基础配置时选择了其他服务或修改了外部服务的连接信息，那么原来的服务会被删除，相当于添加了一个新的服务。但原服务关联的 API 会被自动关联到新的服务。

更新策略配置

1. 在服务列表页面找到需要更新的服务，在服务右侧点击 ，选择修改策略配置。



更新服务

2. 更新策略配置，点击确定。



更新服务

删除服务

在服务列表页面找到需要删除的服务，在服务右侧点击　，选择删除。



删除服务

删除服务之前，需要确保没有 API 正在使用该服务。如果该服务正在被某个 API 使用，需要先根据页面提示，点击 API 管理删除关联的 API 之后才能删除该服务。



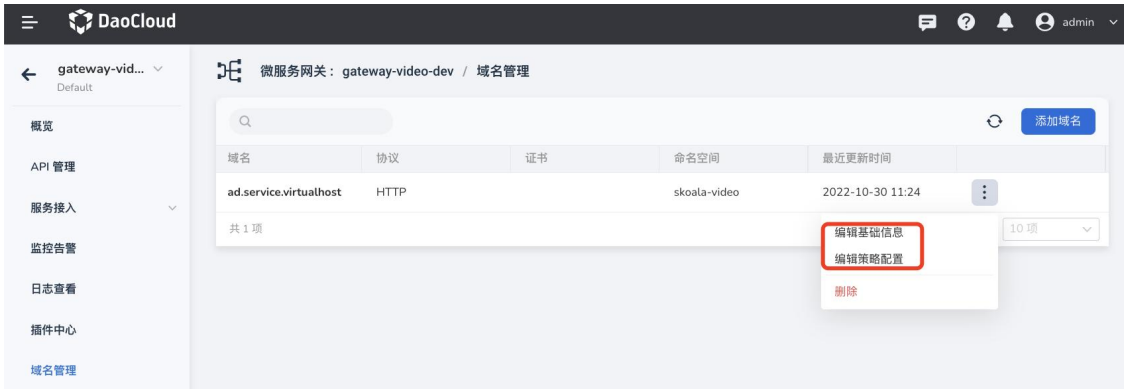
删除服务

更新域名

微服务网关支持对统一托管的域名进行全生命周期管理，包括添加、更新、删除域名域名。通过域名管理，可以将一个域名应用到网关内的多个 API，并且可以配置域名层级的网关策略。

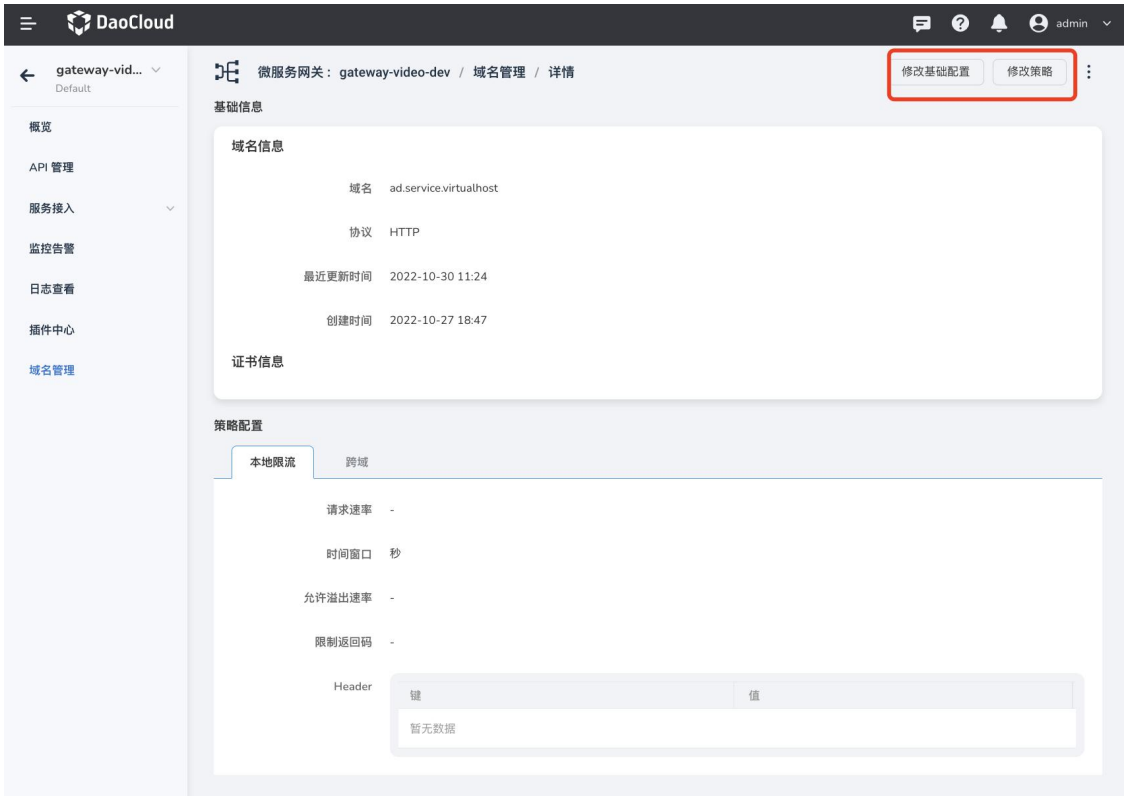
可以通过两种方式修改域名的基础信息和策略配置。

- 在 域名管理页面找到需要更新的域名，在右侧点击 选择编辑基础信息或编辑策略配置。



在列表页更新基础信息

- 点击域名名称进入域名详情页，在页面右上角点击修改基础配置更新基本信息，点击修改策略更新策略。


DaoCloud
admin
gateway-vid...
Default
概览
API 管理
服务接入
监控告警
日志查看
插件中心
域名管理
微服务网关： gateway-video-dev / 域名管理 / 详情
修改基础配置
修改策略
基础信息
域名信息
域名 ad.service.virtualhost
协议 HTTP
最近更新时间 2022-10-30 11:24
创建时间 2022-10-27 18:47
证书信息
策略配置
本地限流
跨域
请求速率 -
时间窗口 秒
允许溢出速率 -
限制返回码 -
Header
键
值
暂无数据

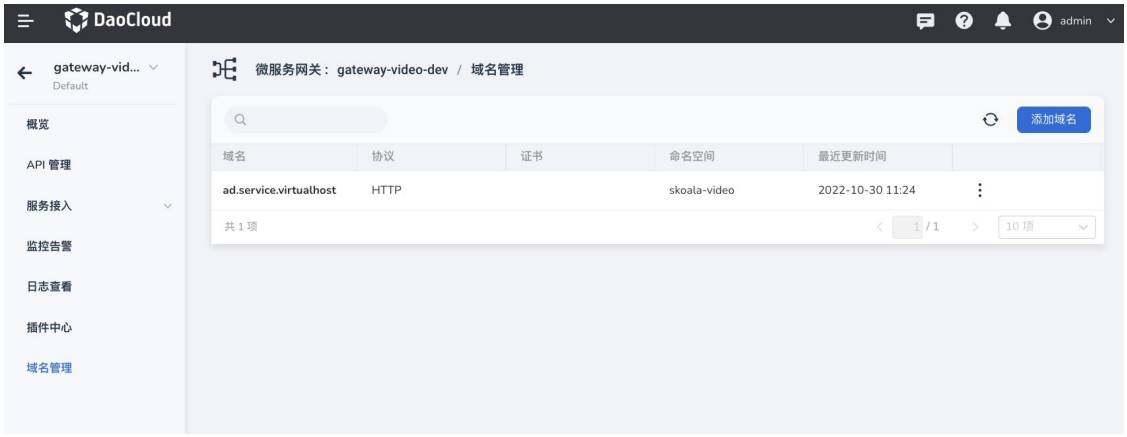
在详情页更新

添加域名

微服务网关支持对统一托管的域名进行全生命周期管理，包括添加、更新、删除域名域名。通过域名管理，可以将一个域名应用到网关内的多个 API，并且可以配置域名层级的网关策略。本页介绍如何添加域名。

添加域名的步骤如下：

1. 点击目标网关的名称进入网关概览页面，然后在左侧导航栏点击域名管理，在页面右上角点击添加域名。



添加域名

2. 填写配置信息

   域名的配置信息分为基础信息（必填）和策略配置（可选）两部分。

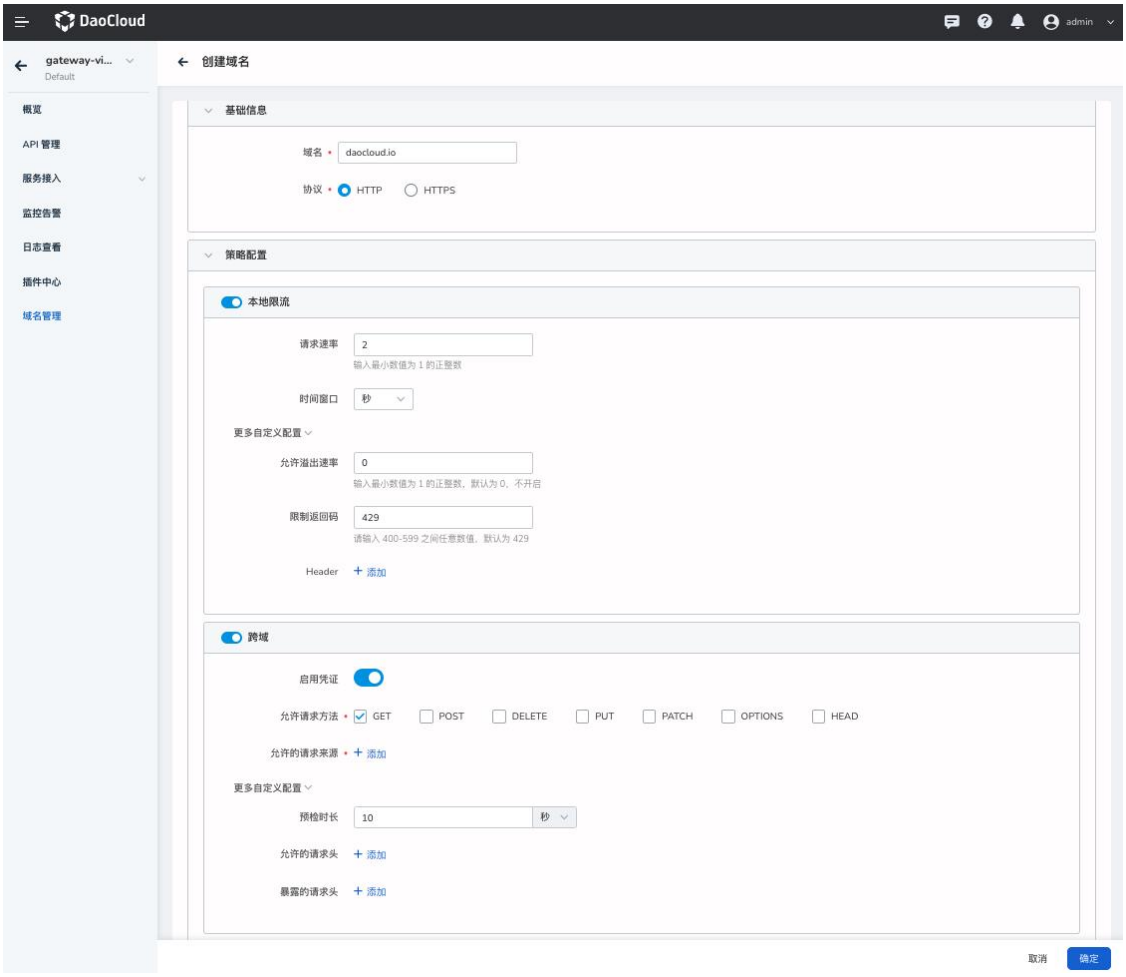
   - 域名：域名创建后不可以修改。
   - 协议：默认选择 HTTP。如果选择 HTTPS，需要提供对应的 HTTPS 证书。

     目前仅支持选取已经存在的证书，自动签发证书和手动上传证书功能正在开发。



https

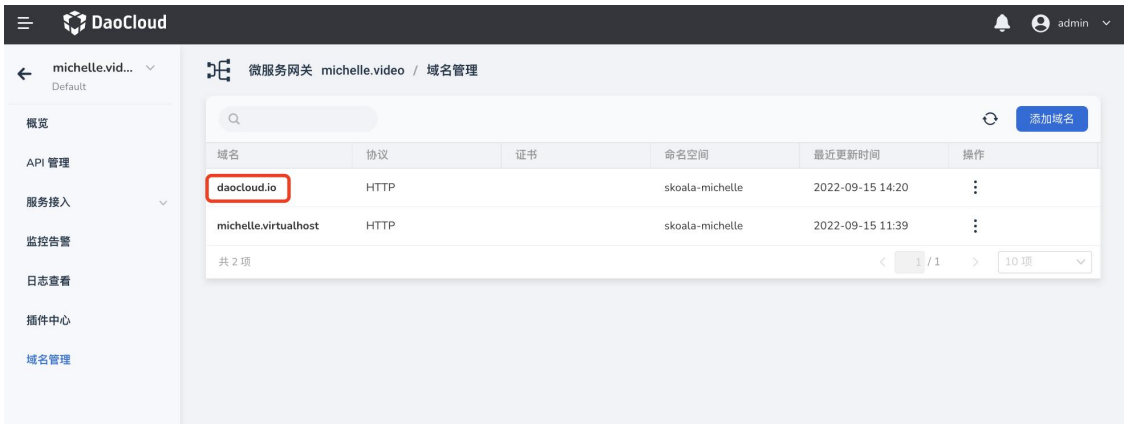
- 本地限流：参考本地限流
- 跨域：参考跨域



填写配置

3. 在页面右下角点击确定

   点击确定后，将自动跳转到域名管理页面，可以在域名列表中看到刚才新建的域名。


DaoCloud
admin
michelle.vid...
Default
概览
API 管理
服务接入
监控告警
日志查看
插件中心
域名管理
微服务网关 michelle.video / 域名管理
添加域名
域名
协议
证书
命名空间
最近更新时间
操作
daocloud.io
HTTP
skoala-michelle
2022-09-15 14:20
michelle.virtualhost
HTTP
skoala-michelle
2022-09-15 11:39
共 2 项
1 / 1
10 项

添加成功

删除域名

微服务网关支持对统一托管的域名进行全生命周期管理，包括添加、更新、删除域名域名。通过域名管理，可以将一个域名应用到网关内的多个 API，并且可以配置域名层级的网关策略。

!!! danger

– 正在被 API 使用的域名无法删除，需要先删除相关的 API 然后才能删除域名。
– 域名删除后无法恢复。

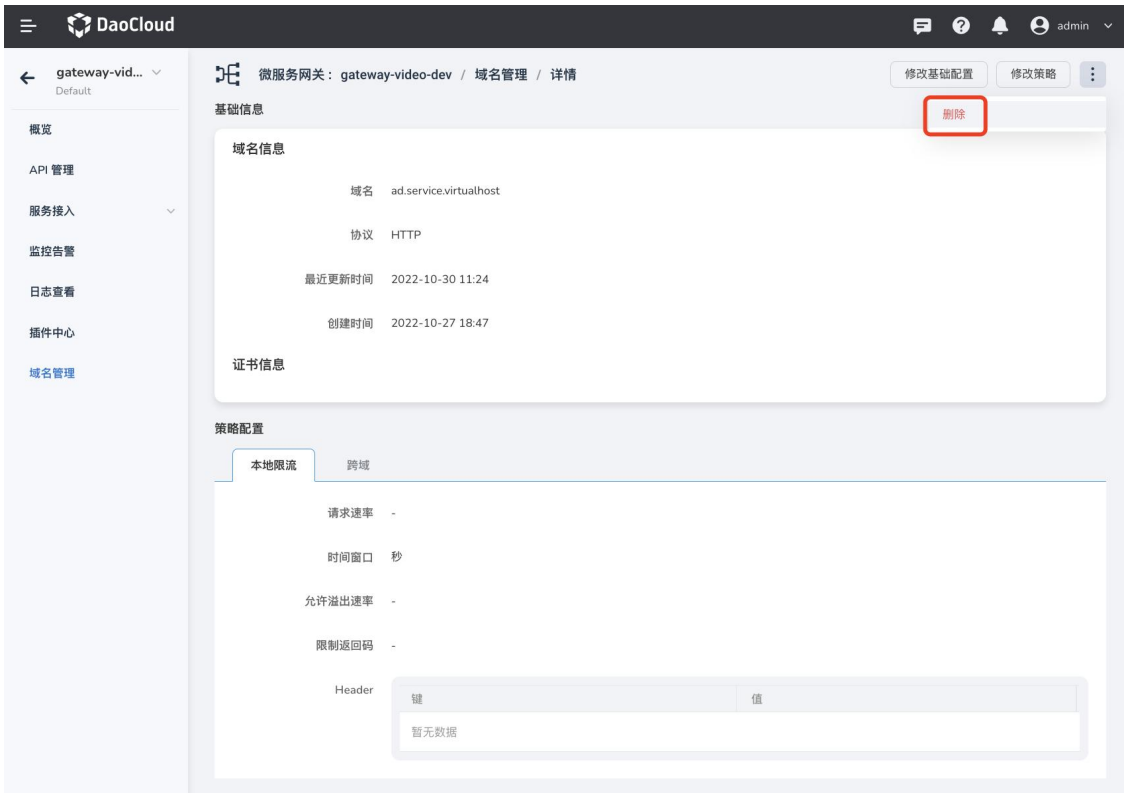
可以通过两种方式删除域名。

- 在域名管理页面找到需要删除的域名，在点击　　并选择删除。



在列表页删除

- 点击域名名称进入域名的详情页，在页面右上角点击　　操并选择删除。



在详情页删除

如果域名正在被某个 API 使用，需要页面提示点击查看服务详情去删除对应的 API。



在详情页删除

配置域名策略

微服务网关提供了域名级的策略配置能力。配置域名级的策略之后，无需对同一域名下的多个 API 重复配置策略。目前支持两种域名级的策略：跨域和本地限流。

有两种方式可以配置域名策略：

- 在创建域名的过程中设置策略，参考添加域名。
- 在域名创建完成后通过修改域名进行调整。

对跨域和本地限流策略的详细说明如下：

本地限流

为域名配置本地限流之后，该配置会自动应用到使用此域名的所有 API。

有关本地限流的详细配置说明，可参考本地限流。

!!! note

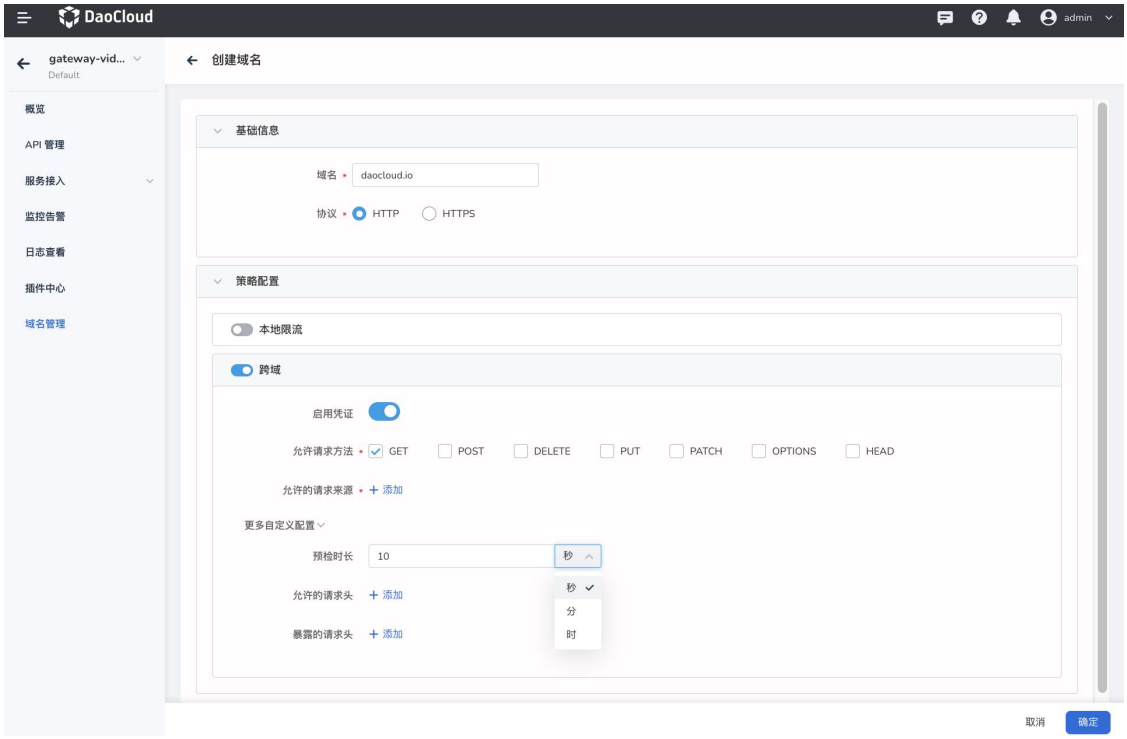
如果 API 层级与域名层级的限流策略冲突时，以 API 层级的限流策略为准。

跨域

填写配置时需要注意：

- 启用凭证：开启后，需要对跨域请求进行凭证检查。检查通过之后，才能开始处理跨域请求。
- 允许的请求方法：选择 HTTP 协议的请求方式。有关各种请求方式的详细说明，可参考 W3C 的官方文档方式定义。
- 允许的请求来源：限定多个特定的请求来源，通常使用 IP。
- 预检时长：在处理跨域请求之前检查凭证、请求方法等事项所用的时间，时间单位为秒、分、时。

- 允许的请求头：限定特定的 HTTP 请求头关键字。添加关键字后，需在请求头中加上对应的关键字才能正常访问目标服务。
- 暴露的请求头：控制暴露的请求头关键字，可以配置多项。



跨域

产品优势

云边协同提供六大核心优势能力，为用户解决不同类型的场景业务需求，更高效协同云边算力资源和数据。



benefits

- 边缘自治
  - 边缘侧自为一套完整的体系，具有不依赖云端的离线处理能力
  - 弱网/无网环境下，边缘节点具备自我管理能力，保障应用业务的连续性
- 智能运维
  - 边缘节点全生命周期监控
  - 边缘应用全生命周期监控
- 可插拔设备接入
  - 设备侧支持 Modbus 等多种协议终端接入，并提供标准接口 DMI 对接设备能力
  - 支持通过指定格式文件将设备批量导入
- 全流程安全保证
  - 提供平台、文件夹、工作空间多级权限管理，资源隔离
  - 支持云边双向数据加密传输，比如 TLS 加密
  - 提供审计日志功能
- 数据协同
  - 提供云边消息路由，支持时序消息的双向传输
  - 提供云边数据通道，支持非结构化数据的双向传输
- 边缘智能
  - 支持模型以容器模式在边缘节点运行
  - 支持模型的批量部署和升级

基本概念

- 边缘单元

  指容器运行所需计算资源的集合，包括云端 Master 和边端工作节点 Node。 边缘单元跟 K8S 中的集群是同一概念，但组网方式有所不同，Master 节点部署在云端，一个云端 master (主备多台)对应一个边缘集群。

- 边缘节点

  是容器集群组成的基本元素，既可以是云主机，也可以是物理机，用于运行容器化应用的载体，边缘应用将以 Pod 的形式在节点上运行。

- 批量注册节点

  相同类型的边缘节点能够预安装软件，在节点开机联网后能自动纳管到平台中。 批量注册节点与边缘节点满足一对多的关系，提高管理效率，也节约了运维成本。

- 边缘节点组

  将节点按照特定属性抽象成节点组概念，以节点组的维度对不同边缘区域下的节点进行统一管理和运维。

- 终端设备

  终端设备可以小到传感器、控制器，大到智能摄像机或工控机床；终端设备可以连接到边缘节点，支持通过 Modbus 协议接入并进行统一管理。

- 工作负载

是管理应用副本的一种 API 对象，具体表现为没有本地状态的 Pod，这些 Pod 之间完全独立、功能相同，能够滚动更新，其实例的数量可以灵活扩缩。

- 批量工作负载

  将相同配置或差异化较小的工作负载定义部署到节点组，是一个任务或批量部署动作。

- 配置项

  是以键/值对形式保存的非机密性配置信息，Pod 可以将其用作环境变量、命令行参数或者数据卷中的配置文件。

- 密钥

  是以键/值对形式保存的、包含密码、令牌、密钥等少量敏感信息的对象，将敏感信息和容器镜像解耦，不需要在应用程序代码中包含机密数据。

- 消息端点

  发送或接收消息的一方，可以是终端设备、云服务等。

- 消息路由

  消息转发的路径。

边缘节点接入要求

边缘节点需要满足下表的规格要求。

操作系统 OS

- x86_64 架构

  Ubuntu LTS (Xenial Xerus)、Ubuntu LTS (Bionic Beaver) 、CentOS、EulerOS、RHEL、银河麒麟、中兴新支点、中标麒麟、openEuler

- armv7i（arm32）架构

  Raspbian GNU/Linux (stretch)

- aarch64（arm64）架构

  Ubuntu LTS (Bionic Beaver)、CentOS、EulerOS、openEuler

内存

边缘软件常规开销约 150MB，具体视边缘管理压力适当增加。

为保证业务的正常运行，建议边缘节点的内存大于 256MB。

CPU

不小于 1 核

硬盘

不小于 1GB

容器运行时

Docker 版本必须高于 19.0，推荐使用 19.03.6 版本。

须知：如果 KubeEdge 版本高于或等于 1.14 且云端 Kubernetes 版本高于 1.24，除 Docker 之外，还需要安装 CRI-Dockerd 。

glibc

版本必须高于 2.17。

查看方法：

ubuntu：ldd --version

端口

边缘节点需要使用如下端口，请确保这些端口能够正常使用。

- 1883：边缘节点内置 MQTT broker 监听端口，并需要开放该端口。

时间同步

边缘节点时间需要与云端服务器时间保持一致，建议 UTC 标准时间。

什么是云边协同

近两年，全球预计有超过百亿数量的终端和传感器，其中我国约占全球数量的 7% 左右，其节点规模远远大于互联网， 物联网接入节点数量呈爆发性增长趋势。

物联网的发展，为网络传输的实时性、网络带宽带来新挑战，也对数据的存储、分析、处理提出新要求。 集中式的云计算模型已然无法满足万物互联下的海量数据的高效传输以及处理需求。伴随着 5G 的发展，边缘计算应运而生。

云边协同将云原生能力延伸至边缘，采用边缘节点模式，将数据处理、业务应用、AI 模型等下沉到边缘端执行， 解决物联网落地时响应实时性、数据隐私性、维护便利性等问题，从而满足行业数字化在敏捷联接、实时业务、数据优化、 应用智能、安全与隐私保护等方面的关键需求。

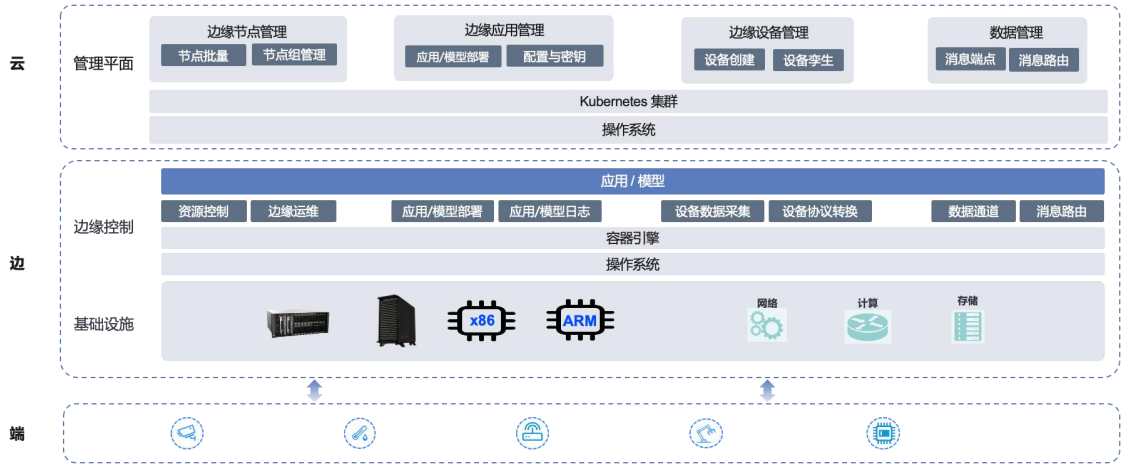


边缘地位

功能特性

云边协同将云原生能力延伸至边缘，采用“端–边–云”一体化协同架构，帮助企业快速拓展安全、灵活、高效、可靠的边缘云原生能力， 广泛适用于工业互联网、物联网、智慧工厂、智慧园区、智慧零售等领域。云端实现边缘节点的注册、管理及应用和配置的下发， 边缘侧负责运行边缘应用，实现边缘自治，设备侧支持多协议终端接入，并提供标准接口对接设备能力。

边缘计算平台的功能架构如下图：



功能架构

功能特性

云边协同支持以下功能特性：

边缘节点管理

支持接入海量的边缘节点，在平台中可以自动生成边缘节点的配置信息，支持在线和离线两种高效便捷的纳管边缘节点方式， 边缘节点可以在云端统一管理、监控和运维。

边缘应用/模型管理

- 边缘计算平台支持将边缘应用或模型以容器的形式快速部署到边缘节点运行。 用户可以将自己的边缘应用程序或模型打包成容器镜像，上传至镜像仓库，然后通过云边协同平台将容器镜像部署到边缘节点运行。
- 云边协同平台支持对已部署的实例的运行数据、事件和日志信息进行查看。
- 容器具有更繁荣的生态，能帮助用户的容器应用无缝切换到其他的运行环境中，具有更好的可移植性， 而且容器具有更好的资源隔离性，并支持 CPU 调度。

边缘设备管理

边缘计算平台支持终端设备连接到边缘节点，终端设备支持通过 Modbus 协议接入。终端设备接入后，可以在云端对终端设备进行统一管理。

数据管理

边缘计算平台提供消息路由功能，用户可以配置消息路由规则，平台根据配置的消息路由将边缘消息转发至对应的消息端点， 让消息按照配置的路径转发，灵活控制数据路由，并提高了数据安全性。

权限管理

平台提供平台/文件夹/工作空间三级权限管理，保障多层级不同角色间资源共享且相互隔离，充分保障资源安全性。

边缘控制

- 提供对边缘节点的资源调度和管理运维功能。
- 提供对边缘节点上运行的应用和模型进行全生命周期管理。
- 提供提供数据采集功能，并且支持设备协议转换，可采集与边缘节点连接的关联设备的数据。
- 提供消息和数据传输通道，支持将边缘消息以 EventBus 协议传输至云端 REST 协议， 云端 REST 协议传输至边缘 EventBus / ServiceBus 协议三种转发路径，同时支持非结构化数据，如图片、视频、文件等的云边双向传输。

应用场景

云边协同适用于以下场景。

智慧工厂

帮助用户快速搭建靠近工厂物联网设备数据源头的云边协同平台，提供实时数据采集分析，建立工厂分析模型， 感知并且降低环境和生产过程中的风险，提升生产的效率，降低生产的成本。

- 适配多种工业协议，实现生产设备统一接入管理，快速采集工业数据，并进行本地数据预处理和分析
- 边缘侧实时感知环境风险并监控生产过程，快速响应决策
- 实现统一应用、模型下发，节点状态统一监控



use case

智慧门店

用边缘计算、人工智能、云原生技术重构人、货、场（消费场景）的管理，寻找新机会、带来新效率、提升用户体验。

- 计算和业务能力下沉，响应速度快，耦合度低
- 复杂网络环境下，门店营销视频自主播控，稳定性高
- 边缘节点和应用监控、日志云端统一管理、分析，故障快速定位，提升运维效率



use case

智慧农业

云边协同平台能够对农业设备进行精确控制和实时数据传输，同时，边缘计算作为云计算的补充， 与人工智能一起在网络边缘侧和更接近数据源的设备侧提供实时、高效的本地决策。

- 云端对设备进行管理、监控和控制
- 边缘测实时响应温湿度变化，智能调节设备运行



use case

创建消息路由

本文介绍创建消息端点、消息路由以及删除消息路由的步骤。

创建消息端点

操作步骤如下：

1. 进入边缘单元详情页，选择左侧菜单边云消息 -> 消息端点。
2. 点击消息端点列表右上角创建消息端点按钮。



创建消息端点

3. 填写相关参数。
   - 消息端点类型：选择类型，当前支持 Rest 云上端点、Event Bus 边缘端点、Service Bus 边缘端点。
   - 命名空间：消息端点所在命名空间。
   - 消息端点名称：输入消息端点名称。
   - 服务端口：只有类型为 Service Bus 的边缘端点需要填写，范围为 1–65535。
4. 单击确定，即创建消息端点成功，返回到消息端点列表页面。



创建消息端点

创建消息路由

操作步骤如下：

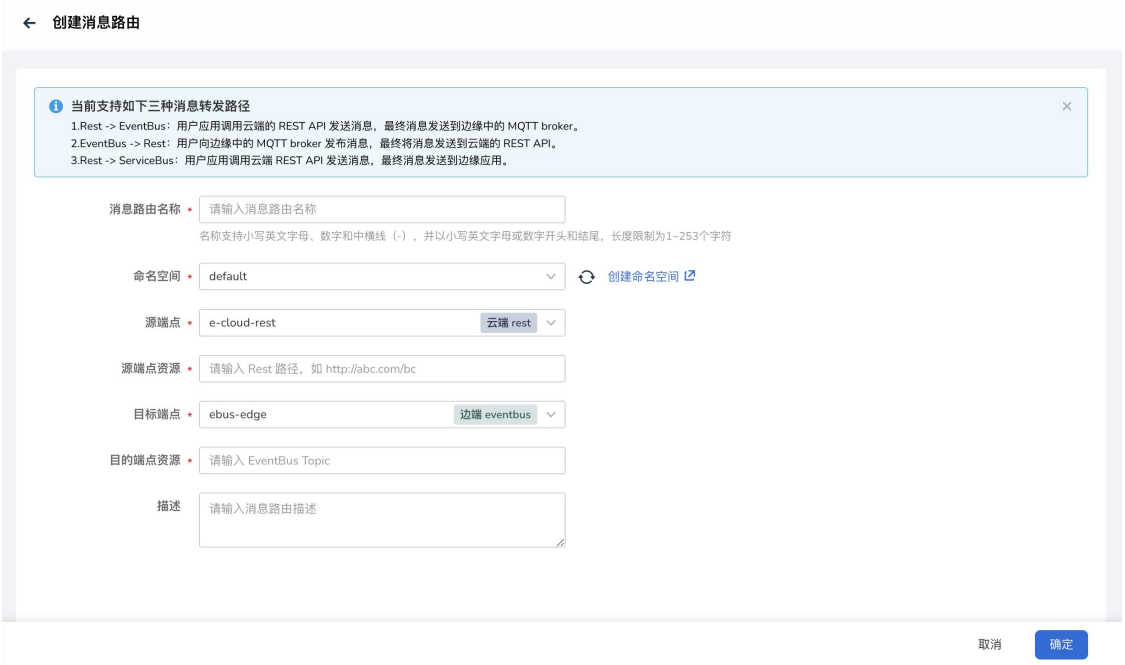
1. 进入边缘单元详情页，选择左侧菜单边云消息 –> 消息路由。
2. 点击消息路由列表右上角创建消息路由按钮。



创建消息路由

3. 填写相关参数。

- 消息路由名称：输入消息路由名称。
- 命名空间：消息路由所在的命名空间。
- 源端点：选择源端点，选项来源于创建的消息端点。
- 源端点资源：
  - 当源端点为云端 Rest 类型，输入 Rest 路径，如 /abc/bc。
  - 当源端点为边端 Event Bus 类型，输入 Topic，由字母、数字、下划线（_）、中划线（-）、斜杠（/）组成。
- 目标端点：选择目的端点，选项来源于创建的消息端点。
- 目标端点资源：
  - 当目的端点为云端 Rest 类型，输入 URL，如 http://127.0.0.1:8080/hello。
  - 当目的端点为边端 Service Bus 类型，输入 Service Bus 路径，如 /abc/bc。



创建消息路由

4. 单击确定，即创建路由规则成功，返回到消息路由列表页面。

路由规则创建完成后，系统将按照相应规则将发送到源端点指定资源的消息转发到目的端点的指定资源上。

删除消息路由

操作步骤如下：

1. 进入边缘单元详情页，选择左侧菜单边云消息 –> 消息路由。
2. 点击指定消息路由右侧删除按钮。
3. 在删除确认弹框中输入消息路由名称。
4. 点击删除，即删除成功，放回到消息路由列表页。



删除消息路由

边云消息概述

云边协同提供了边云消息路由功能，用户可以配置消息路由，平台根据消息路由将消息转发至对应端点，让消息按照规定的路径转发，灵活控制数据路由，并提高数据安全性。

- 消息端点：发送或接收消息的一方，可以是边缘应用、云服务等。
- 消息路由：消息转发的路径。

消息端点

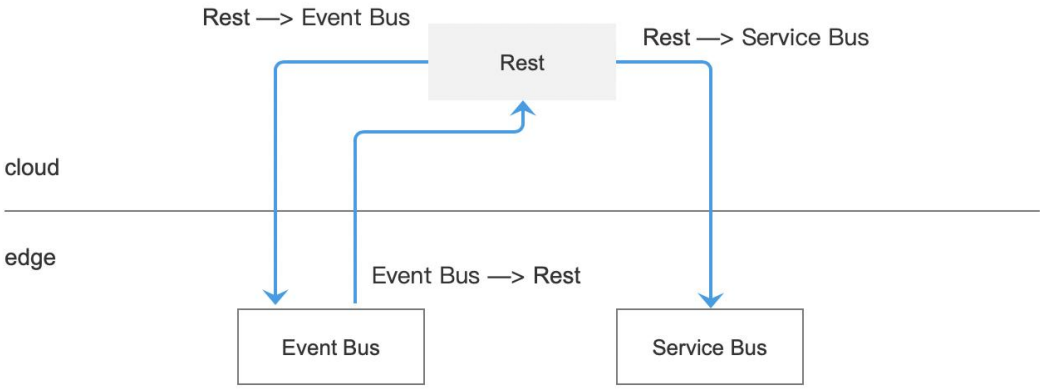
平台提供以下三种类型消息端点：

- Rest：云上端点，向边缘发送消息请求的源端点。或者目标端点，从边缘接收消息。
- Event Bus：边缘端点，向云端发送数据的源端点。或者目标端点，从云端接收消息。
- Service Bus：边缘端点，向边缘发送消息请求的源端点。或者目标端点，从边缘接收消息。

消息路由

对应不同类型的消息端点，平台提供以下消息转发路径：

- Rest –> EventBus：用户应用调用云端的 REST API 发送消息，最终消息发送到边缘中的 MQTT broker。
- EventBus –> Rest：用户向边缘中的 MQTT broker 发布消息，最终将消息发送到云端的 REST API。
- Rest –> ServiceBus：用户应用调用云端 REST API 发送消息，最终消息发送到边缘应用。



路径

后续操作：创建消息路由

可观测性的优势

- 多集群管理

  通过 Global 集群统一存储指标、日志、链路数据，实现对多集群的统一监控，用户可以查询多集群数据的聚合查询。

- 开箱即用

  - 开源精选仪表盘：Insight 提供了多种开箱即用的预制监控仪表盘，您可以通过内置仪表盘实现对集群、节点、工作负载等组件的全面监控。
  - 内置告警规则：Insight 提供了内置的告警规则，您可以在免配置的情况下实现对集群资源、系统组件等基础指标实现开箱即用的监控。

- 高可用性

  - 轻量级 Agent，支持通过 Helm 一键式安装。
  - 数据存储组件支持多副本，保证数据的高可用。

- 开源兼容

  - 兼容标准开源 Prometheus，支持原生的 PromQL 查询指标的数据。
  - 兼容标准开源 Prometheus.yaml 采集规则配置文件、适合自定义 Kubernetes 内监控采集规则 ServiceMonitor。
  - 遵循开源 OpenTelemetry 规范，支持 Jaeger 的链路数据接入。

Insight Release Notes

本页列出 Insight 可观测性的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023.07.30

v0.19.0

新增

- **新增** Kubenetes 事件分析及查询功能
- **新增** 内置告警策略提供告警修复建议
- **新增** 支持自定义日志导出字段及格式

优化

- **优化** 系统组件中获取 Elasticsearch 组件状态的逻辑
- **优化** 支持在安装参数中配置是否导入默认消息模板

修复

- **修复** 部分角色查看告警规则详情的指标时无权限
- **修复** 部分角色在创建、编辑、预览告警静默时无权限
- **修复** 部分角色查看告警消息的详情无权限
- **修复** 链路查询气泡图中显示的时间错误
- **修复** 查询容器日志时未传递命名空间的问题
- **修复** 消息模板中默认变量的自定义标签参数有误
- **修复** 在切换过滤条件时等未发送新请求
- **修复** 告警静默的创建、编辑、预览无权限
- **修复** 概览中日志统计值为 0 的问题
- **修复** 部分内置告警策略不生效的问题

2023.07.01

v0.18.0

新增

- **新增** 日志告警
- **新增** 采集管理增加集群状态
- **新增** 静默规则列表增加静默条件
- **新增** 邮箱配置检测
- **新增** 支持 PostgreSQL 和 Kingbase 作为系统数据库
- **新增** Nvidia GPU 监控仪表盘

优化

- **优化** 更新服务拓扑的图例
- **优化** 服务详情操作指标增加指标平均值并支持排序
- **优化** 链路查询支持 Span、延时、发生时间等排序
- **优化** 告警策略配置通知时下拉框增加搜索
- **优化** 告警模板支持格式化时区
- **优化** 升级 opentelemetry collector Chart 版本从 0.50.1 到 0.59.3
- **优化** 升级 opentelemetry Operator Chart 版本从 0.26.1 到 0.30.1

修复

- **修复** 告警静默预览匹配告警时预览和实际生效静默的告警不相同的问题
- **修复** 系统组件中组件版本错误的问题
- **修复** 仪表盘中列表标题错误
- **修复** 通过告警列表创建静默，预览匹配告警数据无返回
- **修复** 部分环境中 Fluentbit 第一次无法启动的问题

2023.06.01

v0.17.0

!!! warning

在 v0.17.x 版本中将 kube-prometheus-stack chart 版本从 41.9.1 升级至 45.28.1，其中使用的 CRD 也存在一些字段的升级，如 servicemonitor 的 `attachMetadata` 字段，

升级 insight agent 前，请参考：

[从 v0.16.x（或更低版本）升级到 v0.17.x](../quickstart/install/upgrade-note.md#v016x-v017x)。

新增

- **新增** 支持查看活动告警和历史告警详情
- **新增** 支持通过告警快速创建静默规则
- **新增** 告警支持短信通知
- **新增** 邮箱通知支持发送测试信息
- **新增** 邮箱通知的消息模板支持自定义邮箱主题
- **新增** 消息模板增加变量说明
- **新增** Insight Server 组件支持默认高可用

优化

- **优化** 链路查询展示完整 TraceID
- **优化** 链路查询增加服务为空时的提示
- **优化** JVM 监控增加无数据提示
- **优化** 告警策略详情不通知时不显示其他参数
- **优化** 增加 OpenTelemetry Operator 的资源限制
- **优化** 升级 Grafana 版本到 v9.3.14
- **优化** 升级 tailing sidecar 版本从 v0.5.6 到 v0.7.0
- **优化** 升级 kube-prometheus-stack 版本到 v45.28.1
- **优化** 升级 prometheus 版本到 v2.44.0

缺陷

- **修复** 服务拓扑命名空间为空时前端生效的问题
- **修复** 服务拓扑状态说明的文档链接无效的问题
- **修复** 创建告警策略时显示的命名空间和无状态负载重复的问题

- **修复** 告警列表和告警策略中的告警列表缺少触发值数据
- **修复** 告警静默添加静默条件时类型、关键字和值的必填校验错误的问题
- **修复** 告警静默时间范围时区不生效的问题
- **修复** 为工作负载类型创建 PromQL 规则时没有告警告警对象的问题
- **修复** 修改内置告警规则时无法保存的问题
- **修复** 组件中时区写死的问题

2023.04.28

v0.16.0

!!! warning

可观测性 Insight v0.16.0 使用了 vmalertmanagers CRD 的新特性参数 `disableRoute ContinueEnforce`,
升级 insight server 前，请参考[从 v0.15.x（或更低版本）升级到 v0.16.x](../quickstart/install/upgrade-note.md)

新增

- **新增** Java 应用的 JVM 监控。
- **新增** 设置告警静默通知。
- **新增** 告警策略功能，支持在一个告警策略中增加多条告警规则进行管理。
- **新增** Nginx Ingress、Contour 等组件监控仪表盘。
- **新增** 服务拓扑支持开启或关闭虚拟节点。
- **新增** 当前活动告警的数量统计。
- **新增** 内置 HwameiStor 组件的 ServiceMonitor。
- **新增** 域名支持访问 sub path。
- **新增** 【仪表盘】根据全局管理的配置增加备案信息
- **新增** 【仪表盘】域名支持访问 sub path。

优化

- **优化** 内置消息模板的内容。
- **优化** 调整内置告警规则到对应的策略中。

- **优化** 服务命名空间为空的筛选显示问题。
- **优化** 服务列表增加缓存。

修复

- **修复** 搜索中文无效的问题。
- **修复** 服务拓扑命名空间权限无法查看拓扑图的问题。
- **修复** 【仪表盘】麒麟图标的样式问题
- **修复** 【仪表盘】tooltip 过长的问题

2023.04.04

v0.15.4

优化

- **优化** 更新 ES 索引的默认主分片数，使其与中间件默认的 ES 节点数匹配。
- **优化** 修改 JVM 信息 API 响应体结构。

修复

- **修复** 修复多次触发警报记录
- **修复** Fluentbit CVE-2021-46848，从 2.0.5 升级到 2.0.8
- **修复** 检查许可证资源的任务
- **修复** 清除警报历史记录的 SQL 语句
- **修复** 英文仪表盘存在的中文问题

2023.03.30

v0.15.1

新增

- **新增** JVM 指标采集并集成监控面板
- **新增** 链路接入的引导
- **新增** 服务拓扑支持错误率、请求延时过滤
- **新增** 链路支持 Trace ID 搜索

- **新增** Prometheus 组件开启自动垂直扩容

优化

- **优化** 拓扑图虚拟节点的样式
- **优化** 服务拓扑虚拟节点增加开关

修复

- **修复** 容器组运行状态的显示样式
- **修复** 未开启链路功能时，隐藏相关配置参数
- **修复** 部分前端样式不生效问题
- **修复** 采集容器组指标无数据问题
- **修复** 在 OpenShift 集群无法安装 insight-agent 问题

2023.02.27

v0.14.6

新增

- **新增** 图表增加刷新按钮
- **新增** 服务拓扑支持选择多集群并支持通过服务名搜索
- **新增** 服务拓扑详情及流量出口、入口的指标
- **新增** 服务拓扑详情中点击服务名称可跳转到服务的详情
- **新增** 服务监控的列表及流量出口、入口的指标
- **新增** 系统组件监控列表
- **新增** CoreDNS 监控面板
- **新增** 安装时 Insight Agent 添加是否开启 kubeAudit 采集设置

优化

- **优化** 链路查询中的过滤条件并可查看存在 Error 的链路
- **优化** 链路查询的散点图更新为气泡图
- **优化** 将 Prometheus 的指标保留时间缩短至 2 小时
- **优化** VMStorage 的 retentionPeriod 默认参数调整为 1 个月

- **升级** fluentbit 的 helm chart 版本至 0.24.0
- **更新** tailing-sidecar/operator 的镜像
- **更新** 全局采集规则间隔为 60 秒

修复

- **修复** 内置 vmcluster 仪表盘
- **修复** 未开启链路时，导航栏无法加载
- **修复** 系统组件跳转查看详情的链接错误
- **修复** 采集管理列表快捷安装/卸载的链接不对
- **修复** 指标高级查询查询后，下拉框的指标联想与图表重合部分无法选中
- **修复** 修改历史告警存储时长时允许输入小数
- **修复** 当告警规则生成多个告警时发送多个通知
- **修复** vmalert and vmalertmanager 的 configmap-reload 镜像错误
- **修复** ARM 架构中 Insight Agent 的 fluentbit

2023.01.10

v0.13.2

修复

- **修复** insight-agent 中 kubernetes-event-exporter 镜像地址错误的问题
- **修复** 通过资源名称过滤告警 API

2023.12.30

v0.13.1

修复

- **修复** 构建离线包增加 .relok8s-images 文件
- **修复** 调整 insight-agent 中组件 otel-collector 端口对应的端口名

2022.12.29

v0.13.0

新功能

- **新增** 支持修改历史告警存储时间
- **新增** 采集管理组件状态详情
- **新增** 内置消息模板
- **新增** 图表指标计算说明

优化

- **优化** 日志列表字段显示
- **优化** insight-agent 的判断逻辑
- **升级** Jaeger 的 Chart 版本从 v0.62.1 升级到 0.65.1

修复

- **修复** 部分内置告警规则不生效
- **修复** 创建规则时修复名称可重名的错误
- **修复** 钉钉机器人以‘-’结尾的问题
- **修复** 告警规则中不区分大小写的模糊搜索
- **修复** 服务指标错误延迟计算不准确
- **修复** Jaeger 查询出现 too many open files 的问题
- **修复** es 索引翻转别名和清理策略未起作用的问题

2022.11.28

v0.12

新功能

- **新增** insight-agent Helm 模板安装时支持表单化

优化

- **优化** PromQL 查询支持原始的指标
- **优化** 拓扑图的样式
- **升级** 内置 MySQL 镜像版本，从 v5.7.34 升级到 v8.0.29.

- **升级** Fluentbit ARM 架构的 helm Chart 版本从
- **升级** kube-prometheus-stack 的 helm Chart 版本从 v39.6.0 升级至 v41.9.1
- **更新** 使用的 Bitnami 的镜像，包含 grafana-operator, grafana, kubernetes-event-exporter
- **更新** prometheus 相关的的 API 代理地址，将 /prometheus 修改为 /apis/insight.io/prometheus

修复

- **修复** 服务列表缓存逻辑
- **修复** 内置规则不生效的问题
- **修复** 请求延时单位问题
- **修复** Insight 内部链路的问题
- **禁用** vm-stack 中的 PSP 资源
- **修复** victoriaMetrics operator 在 Kubernetes 1.25 中不可用的问题。
- **修复** 前端镜像的浏览器兼容性问题

2022-11-21

v0.11

优化

- **增加** 链路排障和对组件 Jaeger 监控的仪表盘
- **优化** 告警列表、消息模板列表支持排序
- **优化** 过滤掉未安装 insight-agent 的集群
- **优化** 链路查询时默认按 span 开始时间排序

缺陷修复

- **修复** 无数据的 仪表盘，包含 OpenTelemetry 相关的仪表盘
- **修复** 部分日志路径下无内容的问题
- **修复** 删除错误的告警规则：KubeletPodStartUpLatencyHigh

其他

- victoria-metrics-k8s-stack helm chart 升级至 v0.12.6
- opentelemetry-collector helm chart 从 v0.23.0 升级至 v0.37.2
- jaeger helm chart 从 v0.57.0 升级至 v0.62.1
- fluentbit helm chart 从 v0.20.9 升级至 v1.9.9
- kubernetes-event-exporter helm chart 从 v1.4.21 升级至 v2.0.0

2022-10-20

v0.10

功能特性

- 支持与 OTel 服务名称关联的容器管理 Service 名称，以辨别是否启用了服务链路
- 在全局 OTel 一栏更新了默认的跟踪样本策略
- 将相扑（适用于审计日志）exporter port 8080 更改为 80
- 使用 go-migrate 管理数据库迁移版本
- 修复图形 API 中多集群和多命名空间过滤器不正常的问题
- 支持构建 ARM 镜像

安装

- Fluentbit 支持 Dockder 和 containerd 日志的解析器
- 修复 var/log/ UTC 问题
- Fluentbit 支持 elasticsearch 输出跳过 TLS 验证
- K8s 审计日志过滤器支持从 Helm 值获取规则
- 修复 centos7/ubuntu20 主机日志时间的解析问题
- 提升 OTel Operator 版本，移除 Operator 中随自签名证书一起部署的 cert-manager 依赖项
- 设计了 jaeger collector 指标
- 提升 tailing-sidecar 版本
- Jaeger 支持 elasticsearch 输出跳过 TLS 验证

- 在 A 模式中禁用 jaeger 组件

其他

- 新增 OTel collector grafana 仪表盘
- 新增 Insight 概览中文页面

2022-9-25

v0.9

功能特性

- Support kpanda service name associated with the otel service name, identify whether the service tracing enabled.
- Update default tracing sample policies in global otel col.
- Change sumologic(work for audit log) exporter port 8080 to 80.
- Use go-migrate to manage db migration version.
- Fix multi cluster and multi namespaces filter not work well in graph API.
- Support build arm image.

安装

- Fluentbit support parser both docker and containerd log.
- Fix /var/log/ UTC issue.
- Fluentbit support elasticsearch output skip verfify TLS.
- kube audit log filter support getting rule from helm values.
- Fix parse centos7/ubuntu20 host log time.
- Bump up otel operator version, remove cert-manager dependencies in operator deploy within self-signed cert.
- Scrape jaeger collector metrics.
- Bump up tailing-sidecar version.
- Jaeger support elasticsearch output skip verfify TLS.
- Disable jaeger components in Mode A.

其他

- Add otel collector grafana dashboard.
- Add Insight Overview Chinese version.

2022-8-21

v0.8

功能特性

- Migrate graph server into insight server.
- Add cluster_name param to graph query request.
- Add userinfo api.
- Add GetREDMetrics API in GraphQueryService.
- Add GetHelmInstallConfig api to get global cluster service addresses for agent to use.
- Complete auth module.
- Add init cmd/initcontainer for elasticsearch alias and ilm policy

架构调整

- Bump up otel operator in agent chart.
- Add kibana as builtin tools.
- Reduce traces/logs chart's default values.
- Add Helm values parameters documentation.
- Polished Helm parameters.

安装

- Add audit log enable/disable feature.
- Move Fluentbit config to a ConfigMap.

2022-7-20

v0.7

破坏变更

- Modify QueryOperations and GetServiceApdex's API definition in Tracing service.
- Remove resolve alert api.
- fix NFD master crash when CRDs missed.

功能特性

- Remove jaeger relate code in span-metric.
- Add index policy for skoala gateway logs.
- Add lua filter for Ghippo audit logs.
- Add global config api.
- Disable cache in vmselect component.
- Dock with ghippo roles.
- Expose metric insight_cluster_info in server.
- Add log.SearchLog API for SKoala, accept ES query DSL and return raw ES response.
- Bump up OTelcol helm chart version to 0.21.1 and update otelcol architecture.
- support mspider tracing.
- Bump up OTelcol helm chart version to 0.23.0.
- Add default tracing sample policies in global otel col.

架构调整

- Use GrafanaOperator Stack to replace original Grafana Stack.
- Replace insight-overview dashboard.
- Add GrafanaDashboard, GrafanaDatasource CRDs.

2022-6-23

v0.6

破坏变更

- Modify insight deployment and service name to insight-server.
- Modify trace relate metric query API response type.
- Using the unified paging mechanism

功能特性

- Add graph api through prometheus metrics of mesh layer.
- Add service graph api through prometheus metrics of general layer.
- Modify proto param, follow google style doc[https://developers.google.com/protocol-buffers/docs/style].
- Modify list api pagination and add sort.
- Add GProductVersion cr.
- Add insight metric config api.
- Manager insight license resource cr.
- Add traces api through access jaeger grpc endpoint with otlp/v2 protocol.
- Add service-detail api to get all metrics and scalars for a given service name.
- Add operation-detail api to get all metrics and scalars group by operation for a given service name.
- Add traces api through access jaeger grpc endpoint with jaeger v1 protocol.
- Add span metric protobuf style check

架构调整

- Add node-feature-discovery subchart for License module.
- Add opentelemetry-collector subcharts to insight chart.

- Delete audit log OUTPUT config in fluent-bit.
- Add groupbytrace processor to generate trace/span number metrics.
- Add built-in Elasticsearch chart and enabled by default.

安装

- Upgrade victoria-metrics-k8s-stack chart version from 0.6.5 to 0.9.3.
- Add servicemonitor for components in victoria-metrics-k8s-stack.
- Modify insight components resource.

2022-5-18

v0.5

功能特性

- 添加通知模板 API
- 完成规则和告警 API
- 添加服务 API
- 为 vmrules 实现增删改查
- 移除了 get 查询日志 API
- 支持用 fluentbit 收集 kube 审计日志
- 提供 Service Graph 的功能相关 API
- 增强 span_metric API：latencies, calls, errors 三个 API 支持实例维度的查询
- 增强 span_metric API：查询 Latency(with GroupByOperation) 能够返回 P99 P95 P90
- 增强 span_metric API：latencies, calls, errors 三个 API 支持 extension_filters 维度的查询
- 为查询延迟、调用和错误添加了聚合式 API
- 添加了 apdex API
- 重命名 span_metric API URL

安装

- 添加了内置 mysql
- 将 GO 版本升级到 1.17
- 将 insight 服务器服务端口从 8000 更改为 80
- 将 insight 服务器/指标端口从 2022 更改为 81

文档

- 新增文档站术语表
- 新增文档站基本概念任务和实例、数据模型、查询语言等 4 个页面
- 新增用户指南 – 场景监控、数据查询、告警中心等文档
- 文档站新增：产品优势、指标查询、链路查询、仪表盘、概述

2022-4-22

v0.4

功能特性

- 增加告警通知模块主要 API
- 升级并适配 kpanda 0.4.x API
- 为系统日志增加日志所属文件路径信息
- 增加查询单条日志上下文 API
- 增加查询 Kubernetes Event API
- 增强 Insight 自身可观测性能力，提供自身的指标接口和查询链路信息
- 通过反向代理 Jaeger Query 的 API 供前端使用
- 增加 Query Tracing Operations 相关 API
- 增加 Span Metric 相关 API

测试

- 增加 E2E 用例覆盖率徽章
- 补充告警通知相关的测试用例文档
- 增加日志相关接口的 E2E 测试

文档

- 添加整体双语文档站结构及主要内容
- 增加文档所需插件，优化渲染
- 完成 ROADMAP 内容
- 将文档 ROADMAP 内容合并如总 ROADMAP 文件
- 更新文档结构

2022-3-18

v0.3

功能特性

- gRPC 和 http 使用相同的端口
- 将 api 路径从 /api/insight/ 修改为 /api/insight.io/
- 从 kpanda 添加群集资源 api 代理
- ginkgo 从 1.x 升级到 2.x
- 整理 /api 下的 proto 文件
- 在 insight.proto 中拆分 insight 服务
- 将 kpanda api 更新为 0.3.41
- 完成群集/命名空间列表和群集摘要
- 添加批量查询即时和范围度量 api
- 添加节点和所有工作负载 api
- 添加 Otel tracing 以跟踪 insight
- 支持使用 extraLabels 进行度量查询
- 添加度量文档。
- 在 monitor 中实现基本场景案例

Helm Charts

- 添加 Jaeger helm chart
- 添加 OpenTelemetry collector helm chart
- 添加 tailing-sidecar-operator 作为日志收集的配件/解决方案/插件

- 在 fluentbit 中添加/变量/日志/消息收集
- 将 kube exporter 添加到 collecot kube 群集事件日志

可观测性权限说明

可观测性模块使用以下角色：

- Admin / Kpanda Owner
- Cluster Admin
- NS Admin / NS Edit
- NS View

各角色所具备的权限如下：

菜单	操作	Admin / Kpanda Owner	Cluster Admin	NS Admin / NS Edit	NS View
概览	查看概览	√	✗	✗	✗
仪表盘	查看仪表盘	√	✗	✗	✗
场景监控	查看集群监控	√	√	✗	✗
	查看节点监控	√	√	✗	✗
	查看容器监控	√	√	√	√
	查看服务监控	√	√	√	√
	查看拓扑	√	√	√	√
指标查询	查询节点指标 – 普通	√	√	✗	✗
	查询工作负载指标 – 普通	√	√	√	√

菜单	操作	Admin / Kpanda Owner	Cluster Admin	NS Admin / NS Edit	NS View
	查询指标 – 高级	√	√	√	√
日志查询	查询集群事件日志	√	√	✗	✗
	查询节点日志	√	√	✗	✗
	查询容器日志	√	√	√	√
链路查询	查询链路	√	√	√	√
告警列表	查看告警事件	√	√	√	√
告警规则	创建指标模板规则 – 工作负载	√	√	√	✗
	创建指标模板规则 – 节点	√	√	✗	✗
	修改指标模板规则 – 工作负载	√	√	√	✗
	修改指标模板规则 – 节点	√	√	✗	✗
	查看指标模板规则 – 工作负载	√	√	√	√
	查看指标模板规则 – 节点	√	√	✗	✗
	创建 promQL 规则	√	√	√	✗
	修改 promQL 规则	√	√	√	✗
	删除自定义告警规则	√	√	√	✗

菜单	操作	Admin / Kpanda Owner	Cluster Admin	NS Admin / NS Edit	NS View
	查看内置告警规则	√	✗	✗	✗
	修改内置告警规则	√	✗	✗	✗
通知对象	查看通知对象	√	√	√	√
	添加通知对象	√	✗	✗	✗
	修改通知对象	√	✗	✗	✗
	删除通知对象	√	✗	✗	✗
	查看消息模板	√	√	√	√
	添加消息模板	√	✗	✗	✗
	修改消息模板	√	✗	✗	✗
	删除消息模板	√	✗	✗	✗
采集管理	查看 Agent 列表	√	√	√	√
	安装/卸载 Agent	√	√	√	√
	查看 Agent 详情	√	√	√	√
系统配置	查看系统配置	√	✗	✗	✗

有关权限的更多信息，请参阅容器管理权限说明。

有关角色的创建、管理和删除，请参阅角色和权限管理。

什么是可观测模块

可观测模块（Insight）是以应用为中心、开箱即用的新一代云原生可观测性平台。能够实时监控应用及资源，采集各项指标、日志及事件等数据用来分析应用健康状态，不仅提供告警能力以及全面、清晰、多维度数据可视化能力，兼容主流开源组件，而且提供快捷故障定位及一键监控诊断的能力。

可观测模块实现了指标、日志、链路的统一采集，支持对指标、日志进行多维度的告警并提供简洁明了的可视化管理界面。

主要功能如下：

- 提供容器、服务、节点和集群等多维度的监控
- 支持查询 CPU、内存、存储、网络等监控指标
- 集成 Grafana，提供精选的开源仪表盘
- 支持集群工作负载日志，系统日志和 Kubernetes 事件的采集和查询
- 支持单条日志的上下文查询
- 以集群为维度生成服务拓扑，查看服务间调用关系
- 侵入式链路采集，支持查询服务的实时 RPS、错误率、时延等关键指标
- 提供开源的聚合链路查询
- 提供开箱即用的告警规则
- 支持自定义指标、日志等告警
- 支持灵活的配置告警级别、阈值、通知对象等
- 提供邮箱、企业微信、钉钉、Webhook 等多种通知方式
- 持久化存储指标、日志、链路数据

基本概念

可观测性（Insight）有关的基本概念如下。

#	术语	英文	定义
1	监控目标	Target	被监控的对象；系统会定时向监控点发起抓取任务，从中获取指标
2	指标	Metric	使用 open-metric 格式描述，衡量软件或硬件系统中某种属性的程度的标准
3	自定义指标	Recording Rule	一个被命名的 PromQL 表达式，这是将多个指标通过计算而得到的新指标，用来描述更加完整和复杂的系统状态
4	仪表盘	Dashboard	仪表盘是可视化管理的一种表现形式，即对数据、情报等状况一目了然的表现，它通过形象直观而又色彩适宜的各种视觉感知来展示信息。通过可视化图形展示平台的实时情况和 DCE 中所有的性能指标。
6	服务发现	Service Discovery	一个用于 Kubernetes 环境的服务发现配置，用于批量且自动地接入 Kubernetes 上的监控点
7	Exporter	Exporter	一个能够提供指标的服务，往往被理解为监控对象
8	告警规则	Rule	一个返回值是布尔值的 PromQL 表达式，它描述了指标或自定义指标是否处于阈值范围中，如果不满足将产生一条告警事件
9	告警消息	Event	告警规则被触发时的记录信息，记录了告警规则、触发时间、当前系统状态；同时将触发相应的动作，例如发送邮件
10	通知	Notification	由系统通过邮件等渠道发送给用户的告警事件信息
11	PromQL	PromQL	Prometheus 系统所支持的查询语句

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

应用场景

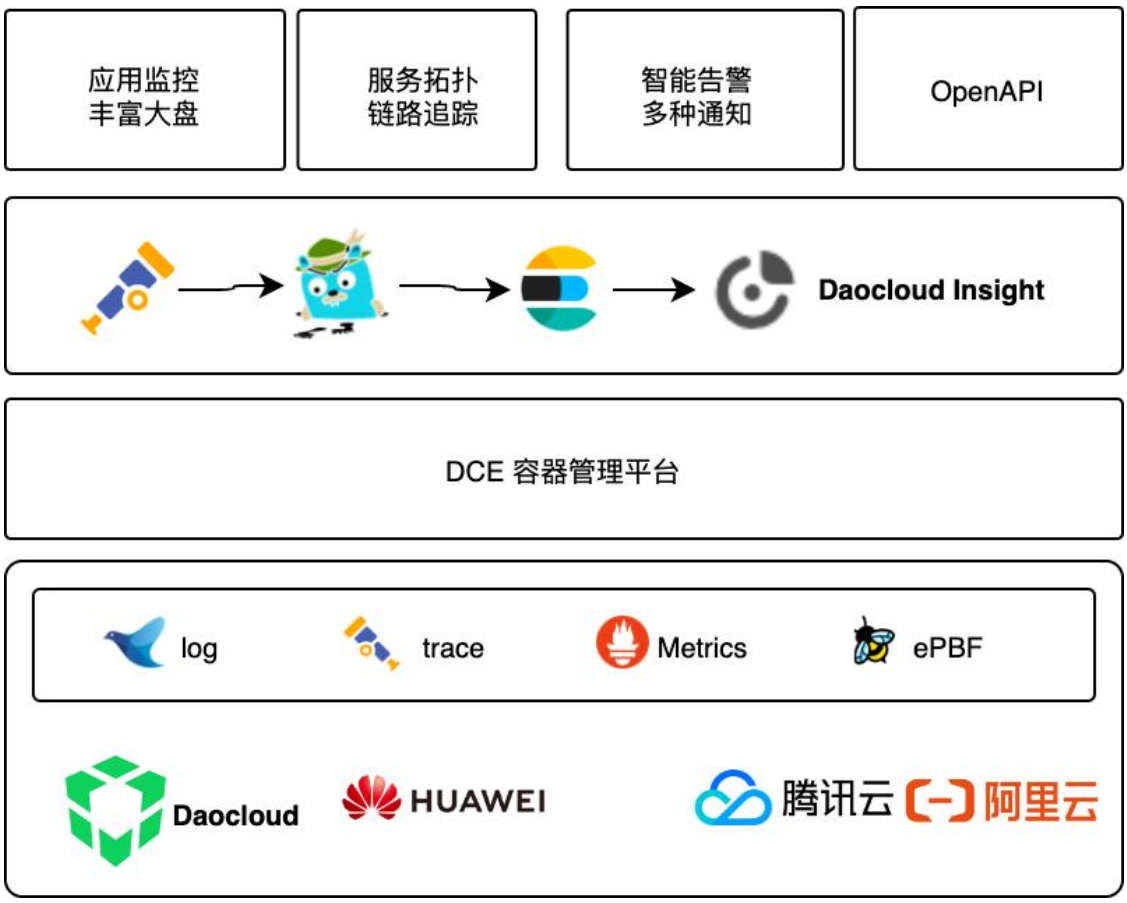
多集群数据统一采集及观测

痛点

- 云上云下及多云环境系统无法统一管理，出现故障要逐个排查，导致运维效率低且成本高。
- 不同数据分开采集，导致数据无法关联分析，排障难度高。

解决方案

- 支持 Helm 一键安装，支持图形化的配置管理，不限 Kubernetes 发行版，实现了 Insight on Any Kubernetes。
- 可观测性具备日志、指标、分布式链路追踪、事件日志等一站式采集和存储。全栈数据观测实现了数据采集、存储、分析、可视化、告警一体化。
- 实时监控各种维度资源，包括集群监控、节点监控、工作负载监控、服务监控等。



安装采集器

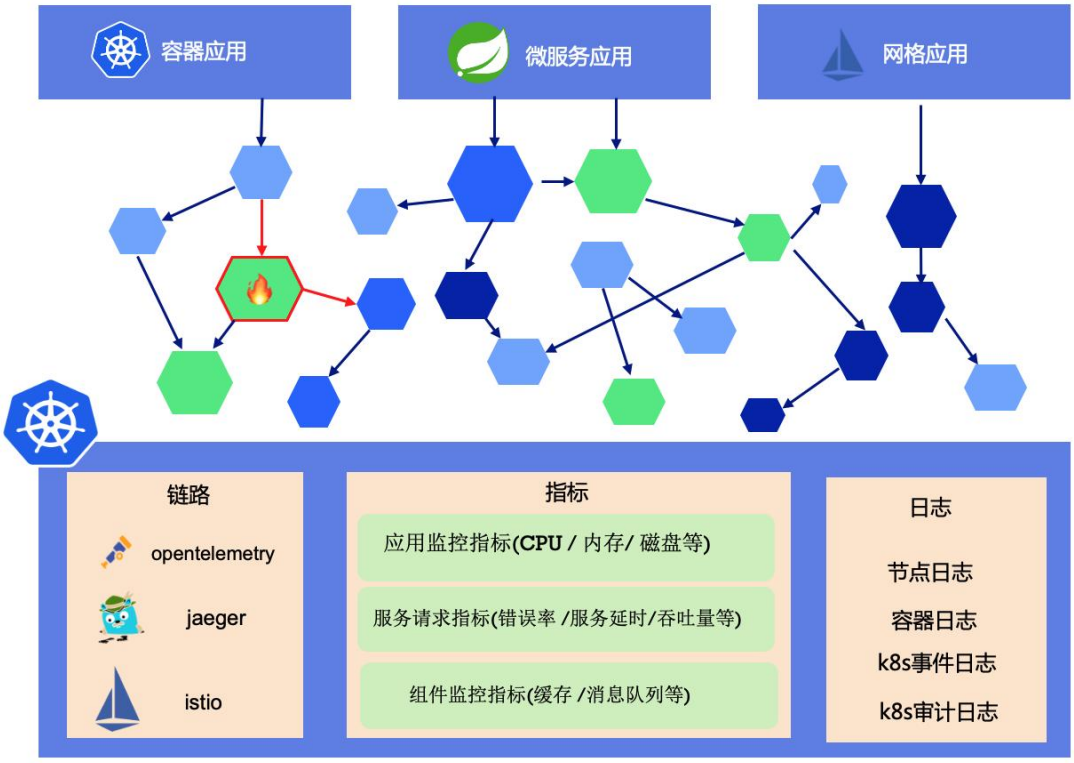
快速故障定位及排查

痛点

- 在复杂微服务架构中，监控配置的维护成本增加，服务故障的定位成本增加，监控的生效速度面临巨大挑战。
- 容器环境叠加微服务架构使得排障复杂，需要避免以单次异常为依据判断根因进行排障工作。

解决方案

- 根据链路数据绘制服务拓扑图，快速定位异常服务。
- 拓扑图中发现异常应用后，通过调用链一键下钻，故障根因清晰可见。
- 通过关联工作负载查询错误日志，快速解决故障。


容器应用
微服务应用
网格应用
链路
opentelemetry
jaeger
istio
指标
应用监控指标(CPU / 内存/ 磁盘等)
服务请求指标(错误率 /服务延时/吞吐量等)
组件监控指标(缓存 /消息队列等)
日志
节点日志
容器日志
k8s事件日志
k8s审计日志

安装采集器

功能列表

本页列出了可观测性 Insight 的功能特性，欢迎使用。

社区版 - 可观测性

DCE 5.0 社区版提供了以下可观测功能。

类别	子类	描述
资源监控	多集群监控	提供多集群业务集中可观测管理员统一管理多集群告警，且满足集群、租户管理员数据隔离支持持久化集群的指标、日志数据。
	场景监控	提供对单个集群的监控概览，可查看该集群的运行状态、了解集群的资源使用情况，以及当前集群正在发生的告警
	节点监控	支持查看节点运行状态等，并了解该节点的 CPU、内存、网络等资源变化情况
	容器监控	支持对无状态负载、守护进程、容器组等资源进行监控，可以监控该工作负载的运行状态，可查看正在告警的数量以及 CPU、内存等资源消耗的变化趋势图
仪表盘	平台组件监控	通过原生 Grafana 提供开源精选仪表盘，提供内置仪表盘支持对 etcd、APIServer 等组件进行监控

类别	子类	描述
	集群资源监控	对集群、节点、命名空间等多维度提供监控。Grafana 使用的数据源支持查看多集群的数据。
数据查询	指标查询	普通查询预订了基础指标，选择集群、类型、节点、指标名称等查询条件后可查询资源的变化趋势支持通过原生 PromQL 语句，查询指标图表及数据详情
	日志查询	可查询 Node、Pod、Depoyment 、Statefulset 等日志，可查询单条日志的上下文内容支持按照关键字进行搜索默认按照时间排序，通过直方图可查询日志数量的变化趋势支持查询单条日志的详细信息及上下文
	日志下载	支持根据搜索条件下载一段时间内的日志支持导出单条日志上下文的内容
告警中心	活动告警	提供直方图查看告警时间的变化趋势支持查看所有正在告警的规则及详情
	历史告警	可查询自动恢复或手动被解决后的所有告警
	告警规则	内置 100+ 告警规则，对集群组件、容器资源等提供预定义的告警规则管理员可创建全局告警规则，对已

类别	子类	描述
		安装 insight-agent 的集群进行统一告警支持通过预定义指标创建告警规则支持通过编写 PromQL 语句创建告警规则支持自定义阈值、持续时间及通知方式可自定义告警的级别，支持紧急、警告、提示三个等级
	通知配置	在通知配置页面，可以配置通过邮件组、企业微信、钉钉、Webhook 等方式向用户发送消息支持同时通知到多个告警对象
	消息模板	消息模板功能支持自定义消息模板的内容，并可邮件、企业微信、钉钉、Webhook 的形式通知指定的对象
日志采集和查询	统一日志采集	统一采集节点、容器、容器内、k8s 事件的日志数据采集全局管理平台的审计操作，默认不开启采集 k8s 审计日志
	日志持久化存储	日志可标注输出到 Elasticsearch 等中间件进行持久化
指标采集	指标数据采集	支持通过使用 ServiceMonitor 自行定义 Pod 发现的 Namespace 范围以及通过 matchLabel 来选择监听的 Service

类别	子类	描述
系统配置	系统配置	系统配置展示指标、日志、链路默认的保存时长以及默认的 Apdex 阈值支持自定义修改指标、日志、链路数据的存储时间
	系统组件	提供对可观测组件的统一监控，实时检测系统组件的健康状态

商业版 - 可观测性

在社区版的基础上，DCE 5.0 商业版提供了更加丰富和可定制的可观测功能。

类别	子类	描述
资源监控	多集群监控	提供多集群业务集中可观测管理员统一管理多集群告警，且满足集群、租户管理员数据隔离支持持久化集群的指标、日志数据。
	集群监控	提供对单个集群的监控概览，可查看该集群的运行状态、了解集群的资源使用情况，以及当前集群正在发生的告警
	节点监控	支持查看节点运行状态等，并了解该节点的 CPU、内存、网络等资源变化情况
	容器监控	支持对无状态负载、守护进程、容器组等资源进行监控，可以监控该工作负载的运行状态，可查看正在告警的数量以及 CPU、内存等资源消耗的变化趋势图
场景监控	服务监控[8]	可查看服务的实时吞吐量、请求数、请求延时和错误率等关键指标，以及一段时间的变化趋势可查看该服务一段时间内的请求，以及单个请求的实时吞吐量、请求

[8] 这是商业版才提供的功能。

类别	子类	描述
		数、请求延时和错误率的变化趋势
	拓扑图[9]	管理员可查看接入观测平台和链路采集的服务间的调用关系、健康状态，快速的故障定位可查看服务间请求的流量方向和关键指标可快速查看单个服务的实时吞吐量、请求数、请求延时和错误率
仪表盘	平台组件监控	通过原生 Grafana 提供开源精选仪表盘，提供内置仪表盘支持对 etcd、APIServer 等组件进行监控
	集群资源监控	对集群、节点、命名空间等多维度提供监控。Grafana 使用的数据源支持查看多集群的数据。
数据查询	指标查询	普通查询预订了基础指标，选择集群、类型、节点、指标名称等查询条件后可查询资源的变化趋势支持通过原生 PromQL 语句，查询指标图表及数据详情
	日志查询	可查询 Node、Pod、Depoyment 、Statefulset 等日志，可查询单条日志的上下文内容支持按照关键字进行搜索默认按照时间排序，通过直方图可查询日志数量的变化趋势支持查询单条日志的详细信息及上下文
	日志下载	支持根据搜索条件下载一段时间内的日志支持导出单条日志上下文的内容
	链路查询[10]	通过链路查询可查看服务在一段时间内的所有请求，支持配置集群、命名空间、服务、操作、标签后点击搜索进行精准搜索支持查看单个请求的聚合链路图，实现快速的故障定位
告警中心	活动告警	提供直方图查看告警时间的变化趋势支持查看所有正在

[9] 这是商业版才提供的功能。

[10] 这是商业版才提供的功能。

类别	子类	描述
		告警的规则及详情
	历史告警	可查询自动恢复或手动被解决后的所有告警
	告警规则	内置 100+ 告警规则，对集群组件、容器资源等提供预定义的告警规则管理员可创建全局告警规则，对已安装 insight-agent 的集群进行统一告警支持通过预定义指标创建告警规则支持通过编写 PromQL 语句创建告警规则支持自定义阈值、持续时间及通知方式可自定义告警的级别，支持紧急、警告、提示三个等级
	通知配置	在通知配置页面，可以配置通过邮件组、企业微信、钉钉、Webhook 等方式向用户发送消息支持同时通知到多个告警对象
	消息模板	消息模板功能支持自定义消息模板的内容，并可邮件、企业微信、钉钉、Webhook 的形式通知指定的对象
日志采集和查询	统一日志采集	统一采集节点、容器、容器内、k8s 事件的日志数据采集全局管理平台的审计操作，默认不开启采集 k8s 审计日志
	日志持久化存储	日志可标注输出到 Elasticsearch 等中间件进行持久化
指标采集	指标数据采集	支持通过使用 ServiceMonitor 自行定义 Pod 发现的 Namespace 范围以及通过 matchLabel 来选择监听的 Service
	组件状态[11]	支持查看采集组件的容器组的状态，并跳转到对应的容器组详情

[11] 这是商业版才提供的功能。

类别	子类	描述
链路采集[12]	链路数据采集	支持通过使用 OTEL SDK 非侵入或者少侵入的方式实现链路数据的采集支持通过在网格应用中注入 Sidecar 的方式采集链路数据
系统配置	系统配置	系统配置展示指标、日志、链路默认的保存时长以及默认的 Apdex 阈值支持自定义修改指标、日志、链路数据的存储时间
	系统组件	提供对可观测组件的统一监控，实时检测系统组件的健康状态

[12] 这是商业版才提供的功能。

服务拓扑

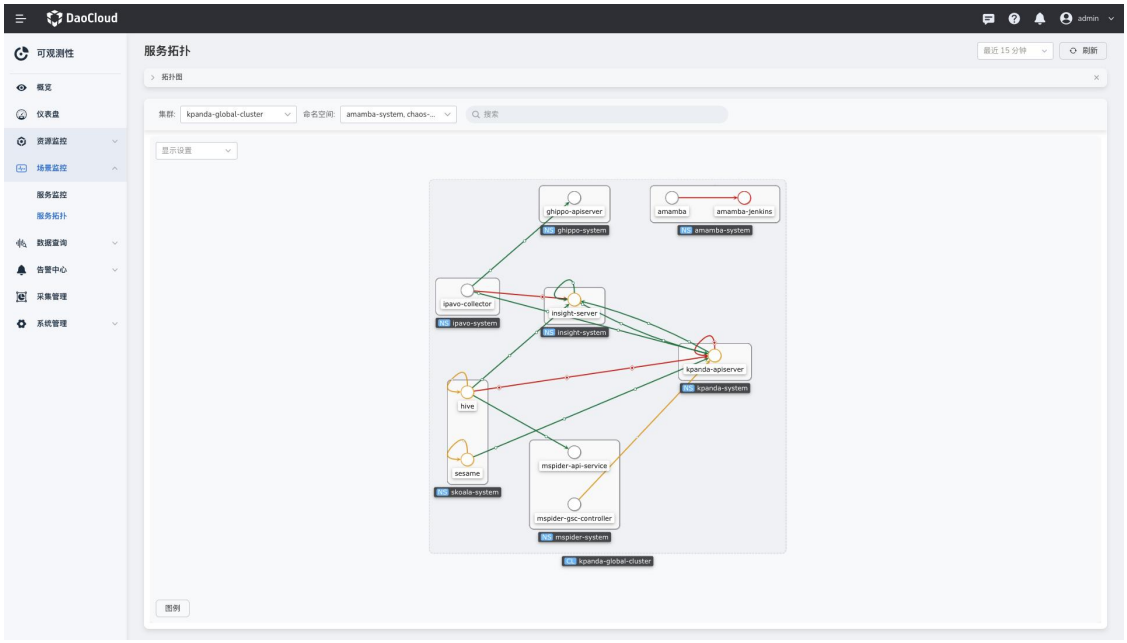
在服务拓扑页面可以查看接入链路数据的服务之前的调用、依赖关系。通过可视化拓扑了解服务间的调用关系，查看服务在指定时间内的调用及其性能状况。

服务拓扑

拓扑图的节点之间的联系代表两个服务在查询时间范围内服务之间的存在调用关系。 请按照以下步骤查看服务监控指标：

1. 进入可观测性 产品模块，
2. 在左边导航栏选择 场景监控 -> 服务拓扑。
3. 在拓扑图中，您可按需执行以下操作：

- 点击节点，查看服务的流量指标。
- 鼠标悬浮在连线上时，可查看服务之间请求的流量指标。
- 在 显示设置 模块，可配置拓扑图中的显示元素。


DaoCloud
admin
可观测性
概览
仪表盘
资源监控
场景监控
服务监控
服务拓扑
数据查询
告警中心
采集管理
系统管理
服务拓扑
最近 15 分钟
刷新
拓扑图
集群:
kpanda-global-cluster
命名空间:
amamba-system, chaos-...
搜索
显示设置
ghippo-apiserver
ghippo-system
amamba
amamba-jenkins
amamba-system
ipavo-collector
ipavo-system
insight-server
insight-system
kpanda-apiserver
kpanda-system
hive
sesame
skoala-system
mspider-api-service
mspider-gsc-controller
mspider-system
kpanda-global-cluster
图例

服务拓扑

容器监控

容器监控是对集群管理中工作负载的监控，在列表中可查看工作负载的基本信息和状态。在工作负载详情页，可查看正在告警的数量以及 CPU、内存等资源消耗的变化趋势。

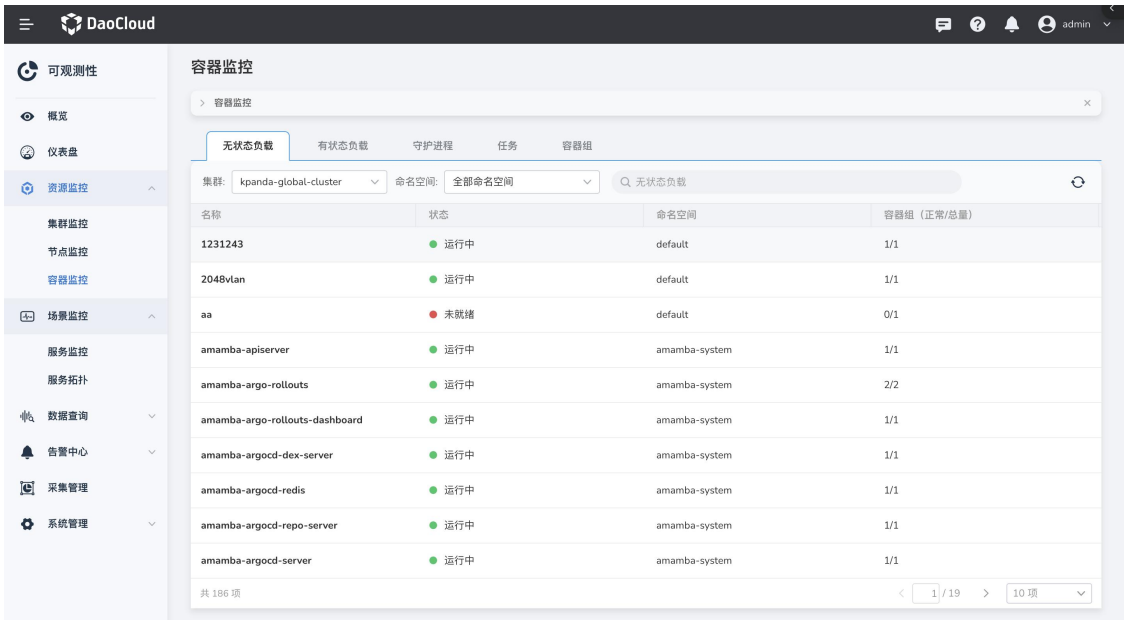
前提条件

集群已安装 Insight Agent，且所有的容器组处于运行中状态。安装 Insight Agent 请参考：在线安装 insight-agent 或离线升级 insight-agent 。

操作步骤

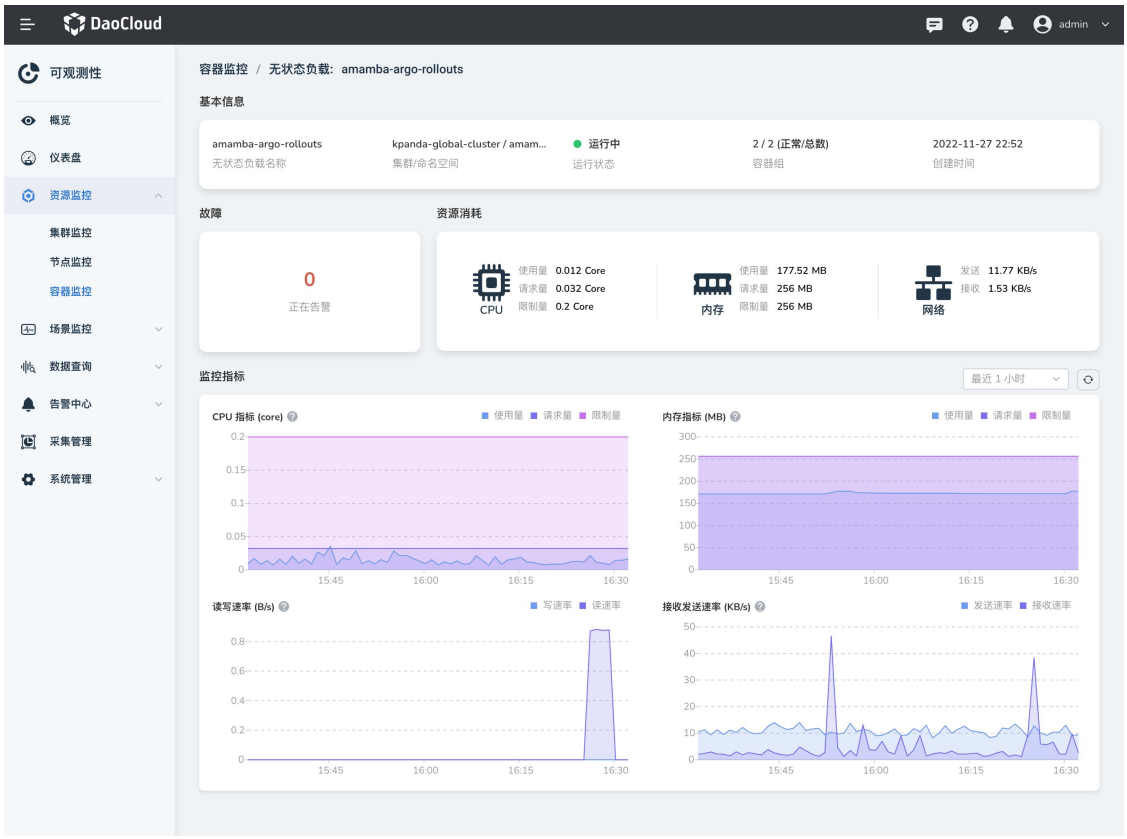
请按照以下步骤查看服务监控指标：

1. 进入可观测性 产品模块。
2. 在左边导航栏选择 资源监控 -> 容器监控。
3. 切换顶部 Tab，查看不同类型工作负载的数据。



容器监控

4. 点击目标工作负载名称查看详情。



容器监控

内容说明

1. 故障：在故障卡片中统计该工作负载当前正在告警的总数。
2. 资源消耗：在该卡片可查看工作负载的 CPU、内存、网络的使用情况。
3. 监控指标：可查看工作负载默认 1 小时的 CPU、内存、网络和磁盘的变化趋势。

指标名称	说明
CPU 使用量	工作负载下容器组的 CPU 使用量之和。
CPU 请求量	工作负载下容器组的 CPU 请求量之和。
CPU 限制量	工作负载下容器组的 CPU 限制量之和。

指标名称	说明
内存使用量	工作负载下容器组的内存使用量之和。
内存请求量	工作负载下容器组的内存使用量之和。
内存限制量	工作负载下容器组的内存限制量之和。
磁盘读写速率	指定时间范围内磁盘每秒连续读取和写入的总和，表示磁盘每秒读取和写入操作数的性能度量。
网络发送接收速率	指定时间范围内，按工作负载统计的网络流量的流入、流出速率。

集群监控

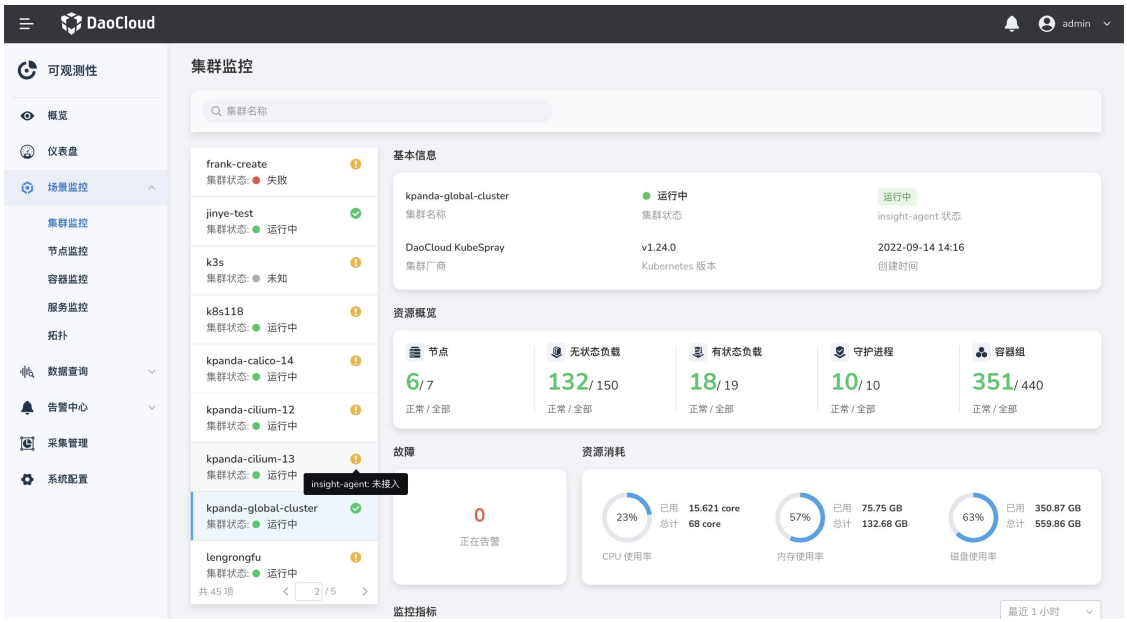
集群监控可查看集群的基本信息，该集群中的资源消耗以及一段时间的资源消耗变化趋势。

前提条件

集群中已安装 insight-agent 且应用处于运行中状态。

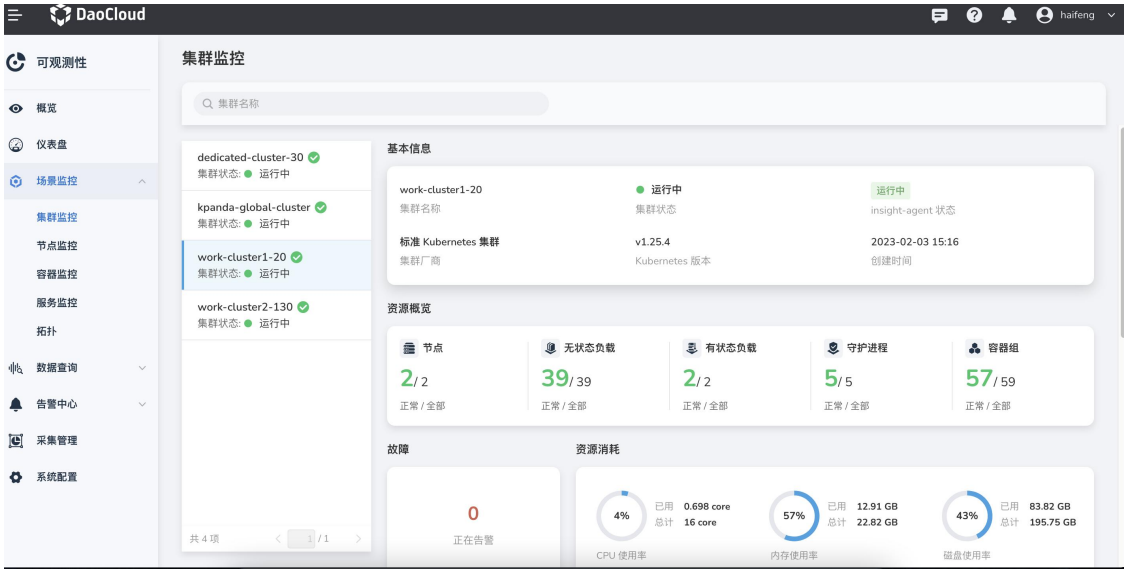
查看集群详情

1. 在左侧导航栏选择 场景监控 -> 集群监控，默认展示选中的第一个集群的信息。



容器监控

2. 在集群列表中选择目标集群，可查看该集群的详细信息。



容器监控

- CPU 使用率：该指标是指集群中所有 Pod 资源的实际 CPU 用量与所有节点的 CPU 总量的比率。
- CPU 分配率：该指标是指集群中所有 Pod 的 CPU 请求量的总和与所有节点的 CPU 总量的比率。
- 内存使用率：该指标是指集群中所有 Pod 资源的实际内存用量与所有节点的内存总量的比率。
- 内存分配率：该指标是指集群中所有 Pod 的内存请求量的总和与所有节点的内存总量的比率。

节点监控

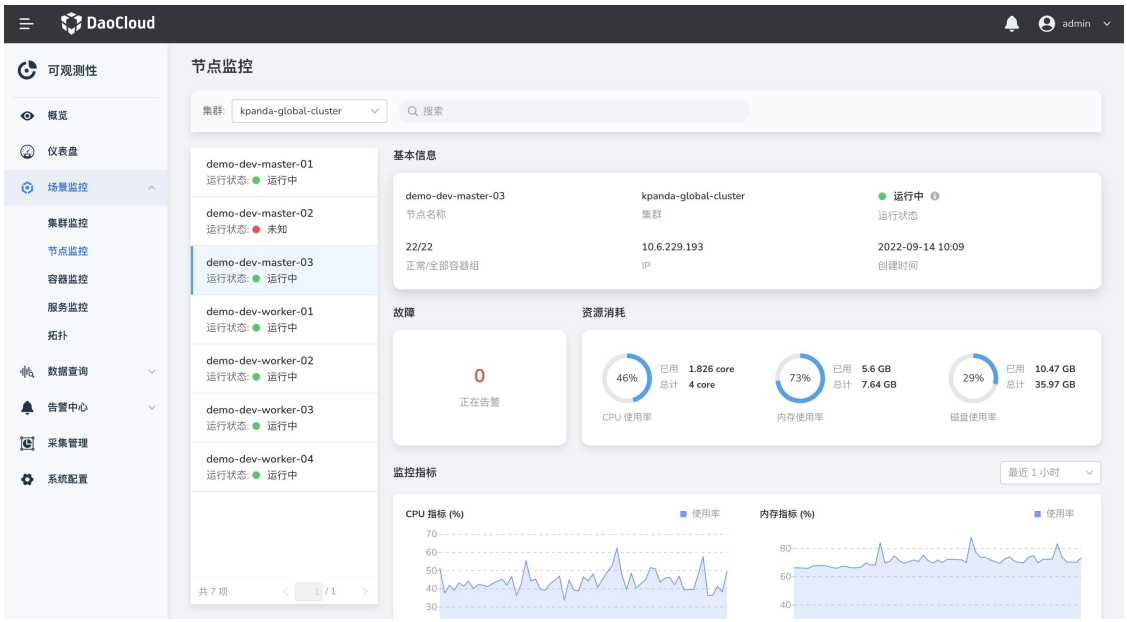
通过节点监控，可概览选中集群下节点的当前健康状态、对应容器组的异常数；在当前节点详情页，可查看正在告警的数量以及 CPU、内存、磁盘等资源消耗的变化趋势图。

前提条件

— 集群中已安装 insight-agent 且应用处于运行中状态。

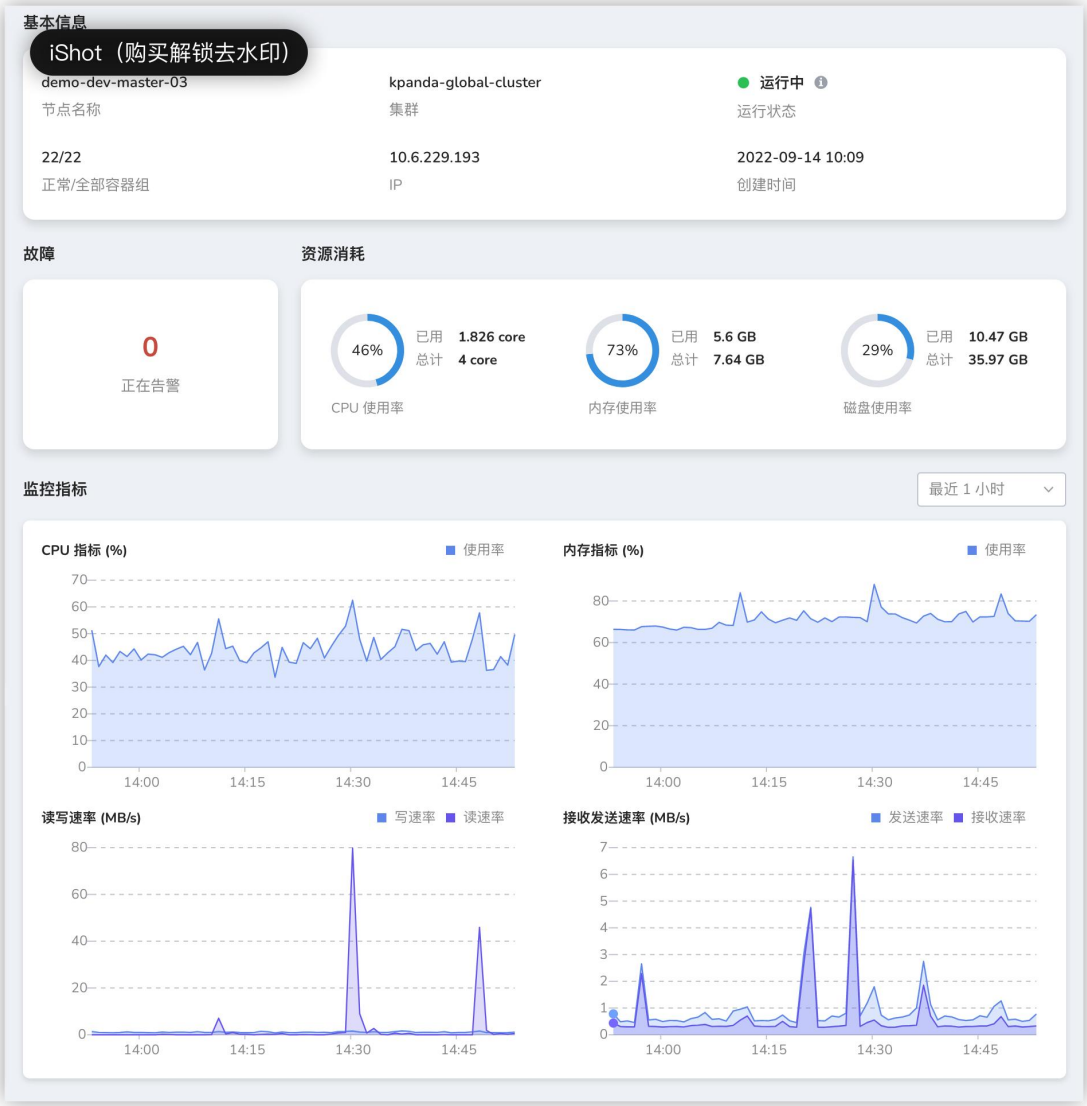
查看节点详情

1. 在左侧导航栏中，选择节点监控，默认展示第一个集群下的节点。



节点监控

2. 在筛选列表中，可按照集群、健康状态筛选节点，也可以通过搜索框进行搜索。


基本信息
iShot（购买解锁去水印）
demo-dev-master-03
节点名称
kpanda-global-cluster
集群
运行中
运行状态
22/22
正常/全部容器组
10.6.229.193
IP
2022-09-14 10:09
创建时间
故障
0
正在告警
资源消耗
46%
已用 1.826 core
总计 4 core
CPU 使用率
73%
已用 5.6 GB
总计 7.64 GB
内存使用率
29%
已用 10.47 GB
总计 35.97 GB
磁盘使用率
监控指标
最近 1 小时
CPU 指标 (%)
使用率
内存指标 (%)
使用率
读写速率 (MB/s)
写速率
读速率
接收发送速率 (MB/s)
发送速率
接收速率
14:00
14:15
14:30
14:45

节点监控

服务监控

在 可观测性 Insight 中服务是指使用 Opentelemtry SDK 接入链路数据，服务监控能够辅助运维过程中观察应用程序的性能和状态。

如何使用 OpenTelemetry 请参考：使用 OTel 赋予应用可观测性

名词解释

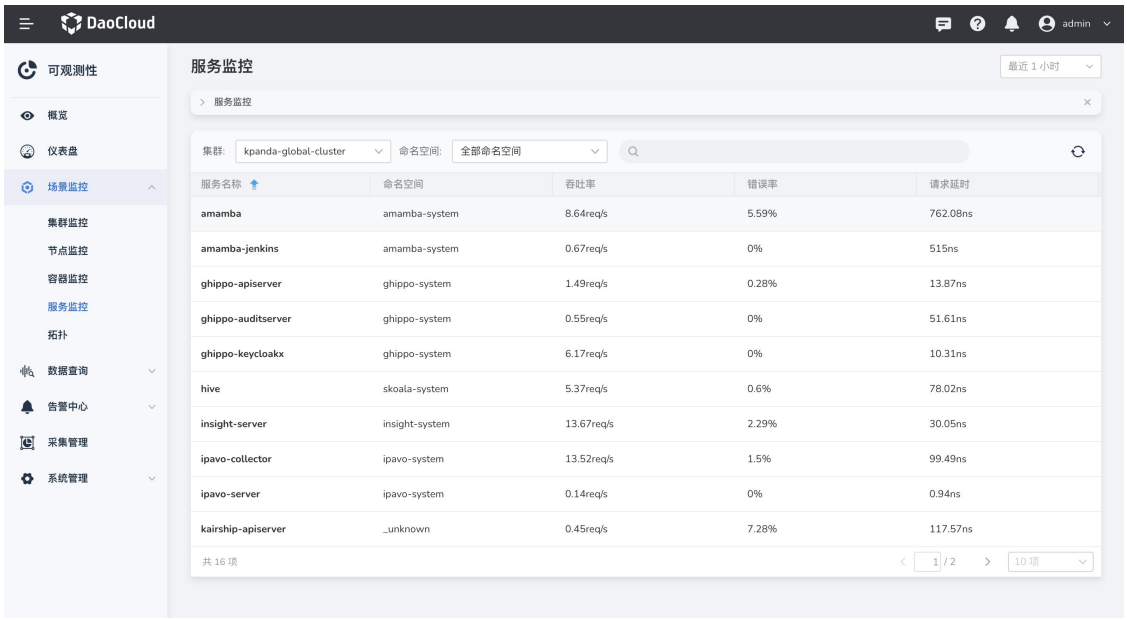
- 服务：服务表示为传入请求提供相同行为的一组工作负载。您可以在使用 OpenTelemetry SDK 时定义服务名称或使用 Istio 中定义的名称。
- 操作：操作是指一个服务处理的特定请求或操作，每个 Span 都有一个操作名称。
- 出口流量：出口流量是指当前服务发起请求的所有流量。
- 入口流量：入口流量是指上游服务对当前服务发起请求的所有流量。

服务列表

服务列表页面展示了集群中所有已接入链路数据的服务的吞吐率、错误率、请求延时等关键指标。 您可以根据集群、命名空间对服务进行过滤，也可以按照吞吐率、错误率、请求延时对该列表进行排序。列表中的指标数据默认时间为 1 小时，您可以自定义时间范围。

请按照以下步骤查看服务监控指标：

1. 进入可观测性 产品模块，
2. 在左边导航栏选择 场景监控 -> 服务监控。



服务监控

!!! attention

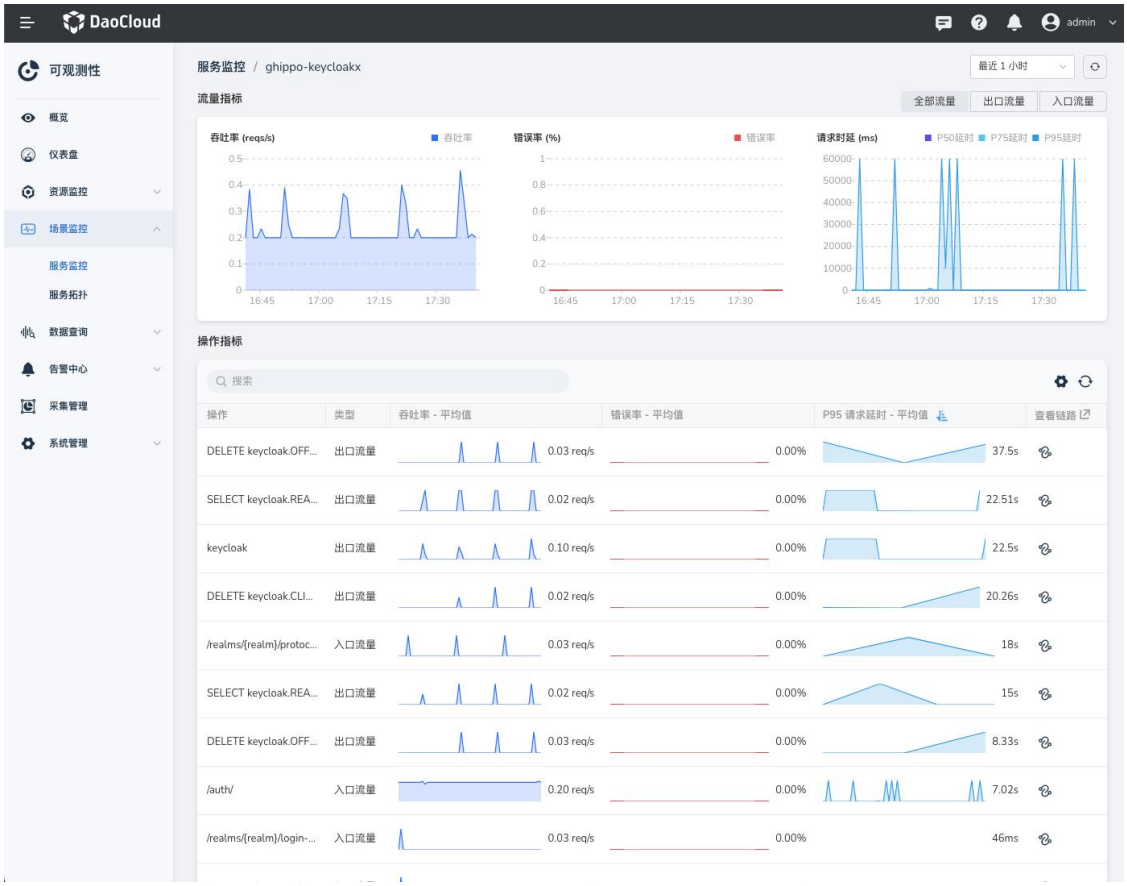
若列表中服务所在的命名空间为`_unknown` 时，则表示该服务未规范接入，建议重新接入。
如您接入的服务存在同名且均未正确填写环境变量中的`命名空间`时，列表及服务详情页中展示的监控数据为多个服务的汇总数据。

服务详情

点击服务名(以 ghippo-keycloakx 为例)，点击进入服务详情页，查看服务的详细指标和该服务的操作指标。

流量指标

在流量指标模块，您可查看到该服务默认一小时内全部请求（包含入口流量和出口流量）的监控指标。您可切换右上角的时间选择器快速选择时间范围，或自定义时间范围。同时，可选择单独查看该服务的入口请求的监控指标或出口请求的流量指标。



服务监控

操作指标

在操作指标模块，您可查看该服务各个操作的操作指标。

服务指标说明

参数	说明
吞吐率	单位时间内处理请求的数量。
错误率	查询时间范围内错误请求与请求总数的比值。
请求延时	单位时间内服务请求的平均响应时间。

服务拓扑元素说明

可观测性提供的服务拓扑能快速定位服务之间的请求关系，并通过不同的颜色来判断服务的健康状态，其中健康状态是根据服务单位时间内请求延时和错误率判断的。本文对服务拓扑中的元素进行说明。

节点状态说明

节点健康状态是根据当前服务全部流量的错误率及请求延时所决定的，判断逻辑如下：

颜色　状态　规则

灰色　健康　错误率等于 0% 且 请求延时小于 100ms

橙色　警告　错误率（0, 5%］或 请求延时（100ms, 200ms］

红色　异常　错误率（5%, 100%］或 请求延时（200ms，+Infinity）

连线状态说明

颜色　状态　规则

绿色　健康　错误率等于 0% 且 请求延时小于 100ms

橙色　警告　错误率（0, 5%］或 请求延时（100ms, 200ms］

红色　异常　错误率（5%, 100%］或 请求延时（200ms，+Infinity）

指标抓取方式

Prometheus 主要通过 Pull 的方式来抓取目标服务暴露出来的监控接口，因此需要配置对应的抓取任务来请求监控数据并写入到 Prometheus 提供的存储中，目前 Prometheus 服务提供了如下几个任务的配置：

- 原生 Job 配置：提供 Prometheus 原生抓取 Job 的配置。
- Pod Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Pod 上对应的监控数据。
- Service Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Service 对应 Endpoints 上的监控数据。

!!! note

[] 中的配置项为可选。

原生 Job 配置

相应配置项说明如下：

```
# 抓取任务名称，同时会在对应抓取的指标中加了一个 label(job=job_name)
job_name: <job_name>

# 抓取任务时间间隔
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# 抓取请求超时时间
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# 抓取任务请求 URI 路径
[ metrics_path: <path> | default = /metrics ]
```

```
# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true: 保留抓取到的 label，忽略与后端 Prometheus 冲突的 label；
# false: 对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的 label；
[ honor_labels: <boolean> | default = false ]

# 是否使用抓取到 target 上产生的时间。
# true: 如果 target 中有时间，使用 target 上的时间；
# false: 直接忽略 target 上的时间；
[ honor_timestamps: <boolean> | default = true ]

# 抓取协议: http 或者 https
[ scheme: <scheme> | default = http ]

# 抓取请求对应 URL 参数
params:
  [ <string>: [<string>, ...] ]

# 通过 basic auth 设置抓取请求头中 `Authorization` 的值，password/password_file 互斥，优先取 password_file 里面的值。
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取 bearer_token 里面的值。
[ bearer_token: <secret> ]

# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取 bearer_token 里面的值。
```

```
[ bearer_token_file: <filename> ]

# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
tls_config:
  [ <tls_config> ]

# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址。
[ proxy_url: <string> ]

# 通过静态配置来指定 target，详见下面的说明。
static_configs:
  [ - <static_config> ... ]

# CVM 服务发现配置，详见下面的说明。
cvm_sd_configs:
  [ - <cvm_sd_config> ... ]

# 在抓取数据之后，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下文说明。
relabel_configs:
  [ - <relabel_config> ... ]

# 数据抓取完成写入之前，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下文说明。
metric_relabel_configs:
  [ - <relabel_config> ... ]

# 一次抓取数据点限制，0：不作限制，默认为 0
[ sample_limit: <int> | default = 0 ]
```

```
# 一次抓取 Target 限制，0：不作限制，默认为 0
[ target_limit: <int> | default = 0 ]
```

Pod Monitor

相应配置项说明如下：

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Pod Monitor
kind: PodMonitor
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为 <namespace>/<name>
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus   # namespace 固定，不需要修改
# 描述抓取目标 Pod 的选取及抓取任务的配置
  label:
    operator.insight.io/managed-by: insight # Insight 管理的标签标识
spec:
  # 填写对应 Pod 的 label，pod monitor 会取对应的值作为 job label 的值。
  # 如果查看的是 Pod Yaml，取 pod.metadata.labels 中的值。
  # 如果查看的是 Deployment/Daemonset/Statefulset，取 spec.template.metadata.labels。
  [ jobLabel: string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
```

```
  podMetricsEndpoints:
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod [K8S metav1.LabelSelector](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.24/#labelselector-v1-meta)
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

举例 1

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus # namespace 固定，不要修改
  label:
    operator.insight.io/managed-by: insight  # Insight 管理的标签标识，必填。
spec:
  podMetricsEndpoints:
    - interval: 30s
      port: metric-port # 填写 pod yaml 中 Prometheus Exporter 对应的 Port 的 Name
      path: /metrics # 填写 Prometheus Exporter 对应的 Path 的值，不填默认 /metrics
      relabelings:
```

```
    - action: replace
      sourceLabels:
        - instance
      regex: (.*)
      targetLabel: instance
      replacement: "crs-xxxxxx" # 调整成对应的 Redis 实例 ID
    - action: replace
      sourceLabels:
        - instance
      regex: (.*)
      targetLabel: ip
      replacement: "1.x.x.x" # 调整成对应的 Redis 实例 IP
  namespaceSelector: # 选择要监控 Pod 所在的 namespace
    matchNames:
      - redis-test
  selector: # 填写要监控 Pod 的 Label 值，以定位目标 pod
    matchLabels:
      k8s-app: redis-exporter
```

举例 2

```
job_name: prometheus
scrape_interval: 30s
static_configs:
- targets:
  - 127.0.0.1:9090
```

Service Monitor

相应配置项说明如下：

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Service Monitor
kind: ServiceMonitor
```

```yaml
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为 Service 的名称。
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus   # namespace 固定，不需要修改
# 描述抓取目标 Pod 的选取及抓取任务的配置
  label:
    operator.insight.io/managed-by: insight   # Insight 管理的标签标识，必填。
spec:
  # 填写对应 Pod 的 label(metadata/labels)，service monitor 会取对应的值作为 job label 的值
  [ jobLabel: string ]
  # 把对应 service 上的 Label 添加到 Target 的 Label 中
  [ targetLabels: []string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
  endpoints:
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod [K8S metav1.LabelSelector](https://v1-17.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#labelselector-v1-meta)
```

```
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

举例

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo # 填写一个唯一名称
  namespace: cm-prometheus # namespace 固定，不要修改
  label:
    operator.insight.io/managed-by: insight  # Insight 管理的标签标识，必填。
spec:
  endpoints:
    - interval: 30s
      # 填写 service yaml 中 Prometheus Exporter 对应的 Port 的 Name
      port: 8080-8080-tcp
      # 填写 Prometheus Exporter 对应的 Path 的值，不填默认 /metrics
      path: /metrics
      relabelings:
        # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 ap
p，
        # 我们通过 relabel 的 replace 动作把它替换成了 application
        - action: replace
          sourceLabels: [__meta_kubernetes_pod_label_app]
          targetLabel: application
  # 选择要监控 service 所在的 namespace
  namespaceSelector:
    matchNames:
      - golang-demo
  # 填写要监控 service 的 Label 值，以定位目标 service
```

```
  selector:
    matchLabels:
      app: golang-app-demo
```

endpoint_config

相应配置项说明如下：

```
# 对应 port 的名称，这里需要注意不是对应的端口，默认：80，对应的取值如下：
# ServiceMonitor: 对应 Service>spec/ports/name;
# PodMonitor: 说明如下：
#   如果查看的是 Pod Yaml，取 pod.spec.containers.ports.name 中的值。
#   如果查看的是 Deployment/Daemonset/Statefulset，取值 spec.template.spec.co
ntainers.ports.name
[ port: string | default = 80]
# 抓取任务请求 URI 路径
[ path: string | default = /metrics ]
# 抓取协议: http 或者 https
[ scheme: string | default = http]
# 抓取请求对应 URL 参数
[ params: map[string][]string]
# 抓取任务间隔的时间
[ interval: string | default = 30s ]
# 抓取任务超时
[ scrapeTimeout: string | default = 30s]
# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
[ tlsConfig: TLSConfig ]
# 通过对应的文件读取 bearer token 对应的值，放到抓取任务的 header 中
[ bearerTokenFile: string ]
# 通过对应的 K8S secret key 读取对应的 bearer token，注意 secret namespace
需要和 PodMonitor/ServiceMonitor 相同
[ bearerTokenSecret: string ]
# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true: 保留抓取到的 label，忽略与后端 Prometheus 冲突的 label；
```

```
# false: 对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的 label；
[ honorLabels: bool | default = false ]
# 是否使用抓取到 target 上产生的时间。
# true: 如果 target 中有时间，使用 target 上的时间；
# false: 直接忽略 target 上的时间；
[ honorTimestamps: bool | default = true ]
# basic auth 的认证信息，username/password 填写对应 K8S secret key 的值，注意 secret namespace 需要和 PodMonitor/ServiceMonitor 相同。
[ basicAuth: BasicAuth ]
# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址
[ proxyUrl: string ]
# 在抓取数据之后，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下文说明
relabelings:
[ - <relabel_config> ...]
# 数据抓取完成写入之前，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下文说明
metricRelabelings:
[ - <relabel_config> ...]
```

relabel_config

相应配置项说明如下：

```
# 从原始 labels 中取哪些 label 的值进行 relabel，取出来的值通过 separator 中的定义进行字符拼接。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 sourceLabels
[ source_labels: '[' <labelname> [, ...] ']' ]
# 定义需要 relabel 的 label 值拼接的字符，默认为 ';'
[ separator: <string> | default = ; ]
```

```
# action 为 replace/hashmod 时，通过 target_label 来指定对应 label name。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 targetLabel
[ target_label: <labelname> ]

# 需要对 source labels 对应值进行正则匹配的表达式
[ regex: <regex> | default = (.*) ]

# action 为 hashmod 时用到，根据 source label 对应值 md5 取模值
[ modulus: <int> ]

# action 为 replace 的时候，通过 replacement 来定义当 regex 匹配之后需要替换的
表达式，可以结合 regex 正规则表达式替换
[ replacement: <string> | default = $1 ]

# 基于 regex 匹配到的值进行相关的操作，对应的 action 如下，默认为 replace：
# replace: 如果 regex 匹配到，通过 replacement 中定义的值替换相应的值，并通过
target_label 设值并添加相应的 label
# keep: 如果 regex 没有匹配到，丢弃
# drop: 如果 regex 匹配到，丢弃
# hashmod: 通过 moduels 指定的值把 source label 对应的 md5 值取模
# 并添加一个新的 label，label name 通过 target_label 指定
# labelmap: 如果 regex 匹配到，使用 replacement 替换对就的 label name
# labeldrop: 如果 regex 匹配到，删除对应的 label
# labelkeep: 如果 regex 没有匹配到，删除对应的 label
[ action: <relabel_action> | default = replace ]
```

配置服务发现规则

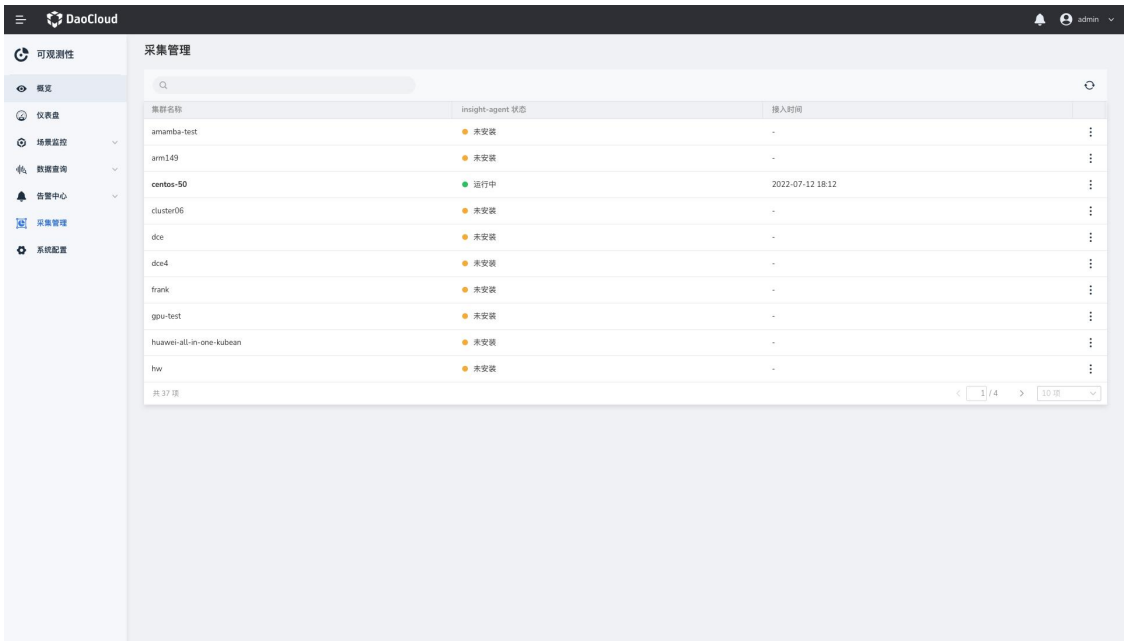
可观测 Insight 支持通过容器管理创建 CRD ServiceMonitor 的方式来满足您自定义服务发现的采集需求。 用户可以通过使用 ServiceMonitor 自行定义 Pod 发现的 Namespace 范围以及通过 matchLabel 来选择监听的 Service。

前提条件

集群已安装 Helm 应用 insight-agent 且处于运行中状态。

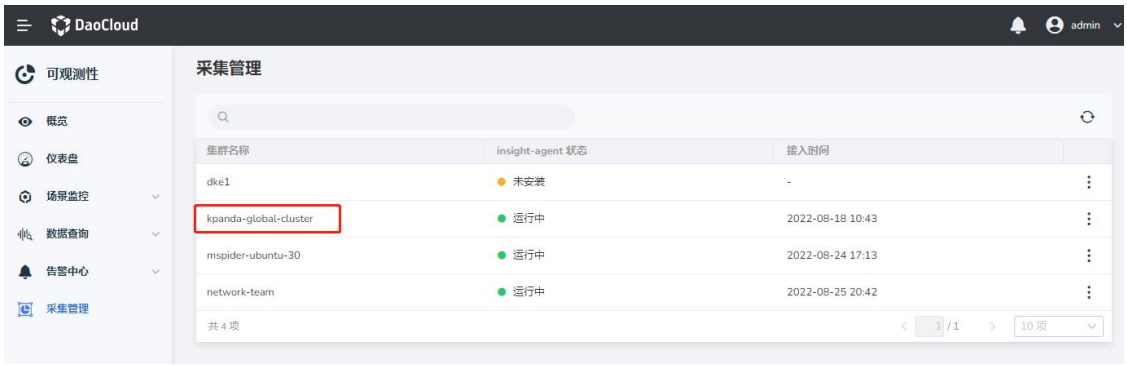
操作步骤

1. 选择左侧导航栏的采集管理，查看全部集群采集插件的状态。



集群列表

2. 点击列表中的某个集群名称进入采集配置详情。



集群列表

3. 点击链接跳转到容器管理 中创建 Service Monitor。

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: micrometer-demo  # (1)
  namespace: insight-system  # (2)
    labels:
      operator.insight.io/managed-by: insight
spec:
  endpoints:  # (3)
    - honorLabels: true
        interval: 15s
        path: /actuator/prometheus
        port: http
  namespaceSelector:  # (4)
    matchNames:
      - insight-system   # 此处匹配的命名空间为需要暴露指标的应用所在的。
  selector:  # (5)
    matchLabels:
        micrometer-prometheus-discovery: "true"
```

1. 指定 ServiceMonitor 的名称

2. 指定 ServiceMonitor 的命名空间
3. 这是服务端点，代表 Prometheus 所需的采集 Metrics 的地址。endpoints 为一个数组，同时可以创建多个 endpoints。每个 endpoints 包含三个字段，每个字段的含义如下：
    - interval：指定 Prometheus 对当前 endpoints 采集的周期。单位为秒，在本次示例中设定为 15s。
    - path：指定 Prometheus 的采集路径。在本次示例中，指定为 /actuator/prometheus。
    - port：指定采集数据需要通过的端口，设置的端口为采集的 Service 端口所设置的 name。
4. 这是需要发现的 Service 的范围。namespaceSelector 包含两个互斥字段，字段的含义如下：
    - any：有且仅有一个值 true，当该字段被设置时，将监听所有符合 Selector 过滤条件的 Service 的变动。
    - matchNames：数组值，指定需要监听的 namespace 的范围。例如，只想监听 default 和 insight-system 两个命名空间中的 Service，那么 matchNames 设置如下：

      ```yaml
      namespaceSelector:
        matchNames:
          - default
          - insight-system
      ```
5. 用于选择 Service。

insight-agent 组件状态说明

在 DCE 5.0 中可观测性 Insight 作为多集群观测产品，为了实现多集群观测数据的统一采集，需要用户安装 Helm 应用 insight-agent （默认安装在 insight-system 命名空间）。参阅如何安装 insight-agent。

状态说明

在 可观测性 -> 采集管理部分可查看各集群安装 insight-agent 的情况。

- 未安装：该集群中未在 insight-system 命名空间下安装 insight-agent
- 运行中：该集群中成功安装 insight-agent，且部署的所有组件均处于运行中状态
- 异常：若 insight-agent 处于此状态，说明 helm 部署失败或存在部署的组件处于非运行中状态

可通过以下方式排查：

1. 执行以下命令，若状态为 deployed，则执行下一步。若为 failed，由于会影响应用的升级，建议在 容器管理 -> helm 应用 卸载后重新安装 ：

   ```
   helm list -n insight-system
   ```

2. 执行以下命令或在 可观测性 -> 采集管理 中查看该集群部署的组件的状态，若存在非 运行中状态的容器组，请重启异常的容器组。

   ```
   kubectl get pods -n insight-system
   ```

补充说明

1. insight-agent 中指标采集组件 Prometheus 的资源消耗与集群中运行的容器组数量存在正比关系， 请根据集群规模调整 Prometheus 的资源，请参考：Prometheus 资源规划

2. 由于全局服务集群中指标存储组件 vmstorage 的存储容量与各个集群容器组数量总和存在正比关系。

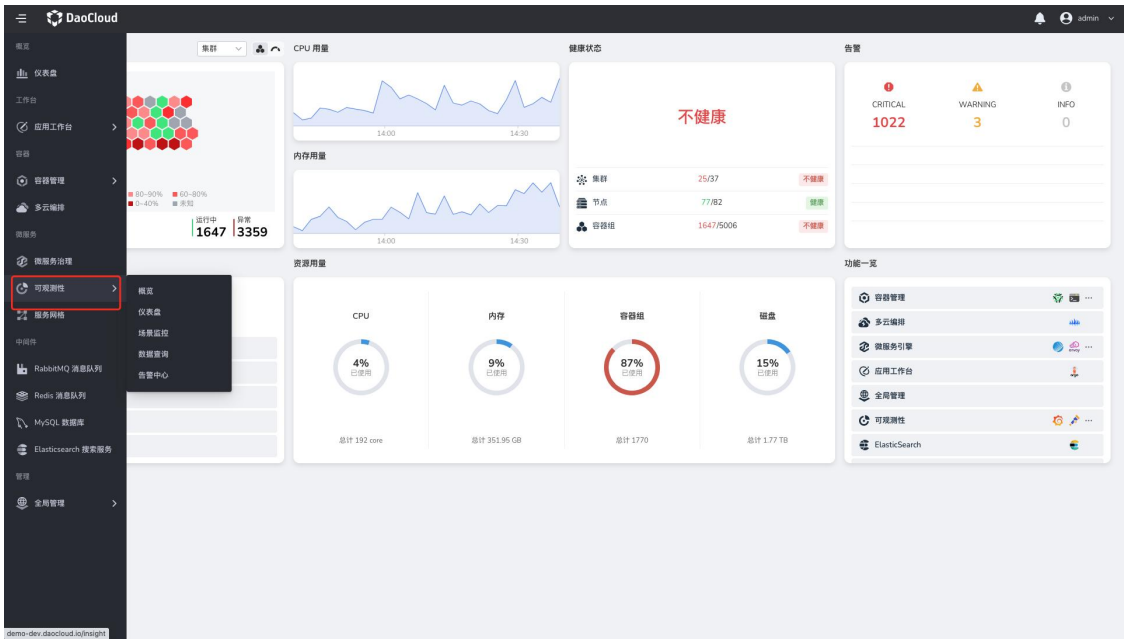
   - 请联系平台管理员根据集群规模调整 vmstorage 的磁盘容量，参阅 vmstorage 磁盘容量规划
   - 根据多集群规模调整 vmstorage 磁盘，参阅 vmstorge 磁盘扩容

采集管理

采集管理主要是集中管理、展示集群安装采集插件 insight-agent 的入口，帮助用户快速的查看集群采集插件的健康状态 ，并提供了快捷入口配置采集规则。

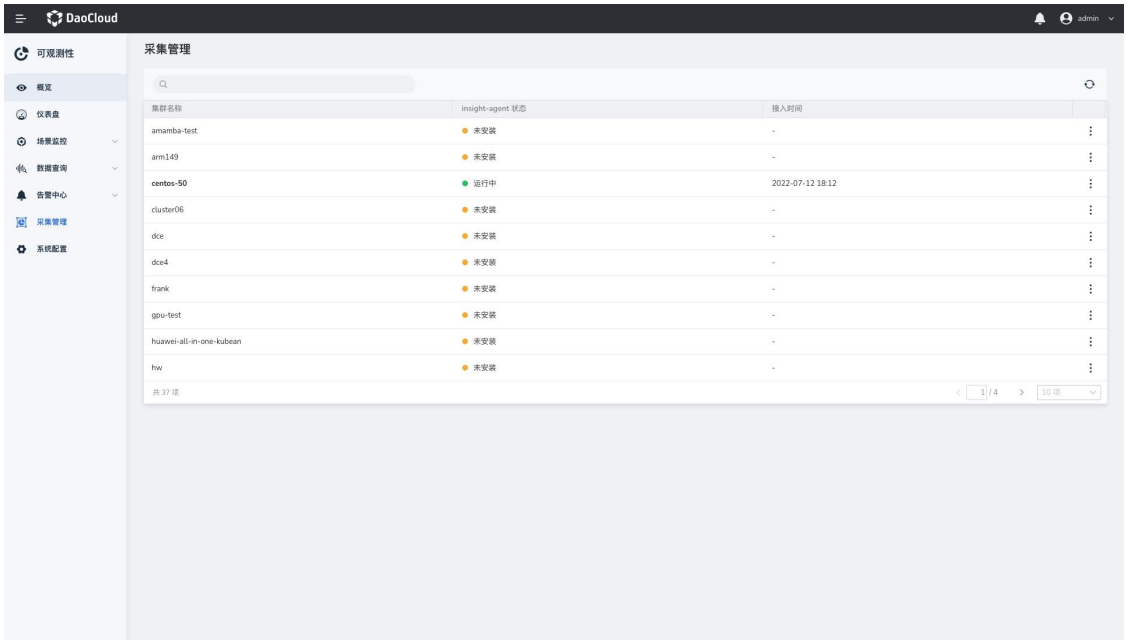
具体操作步骤如下：

1. 点击左上角的，选择可观测性。



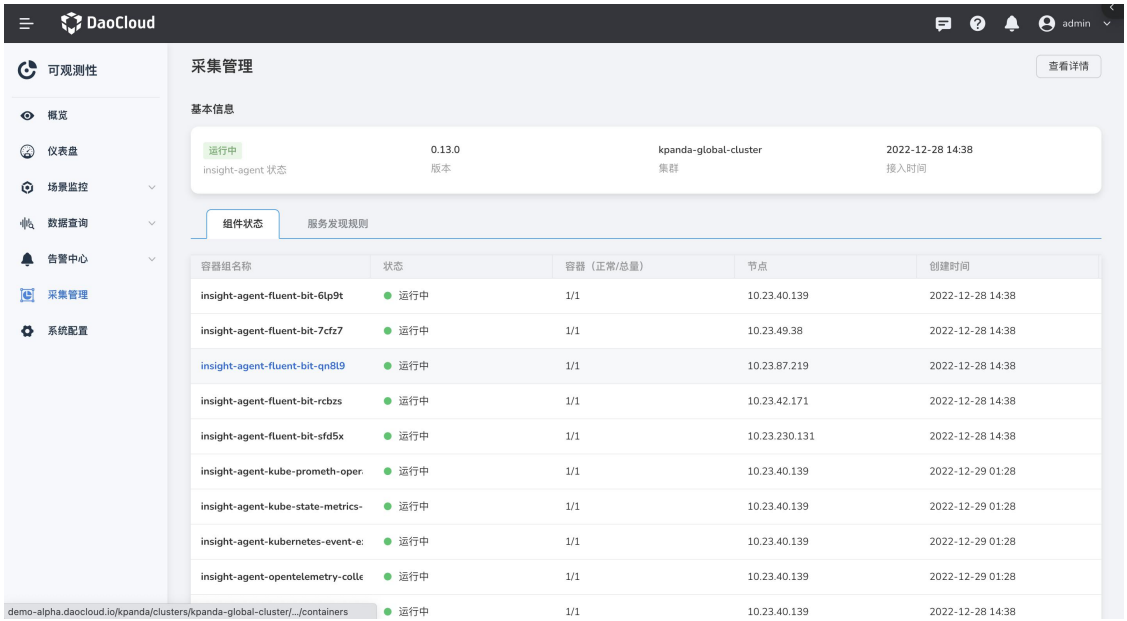
一级导航

2. 选择左侧导航栏的采集管理，查看全部集群采集插件的状态。



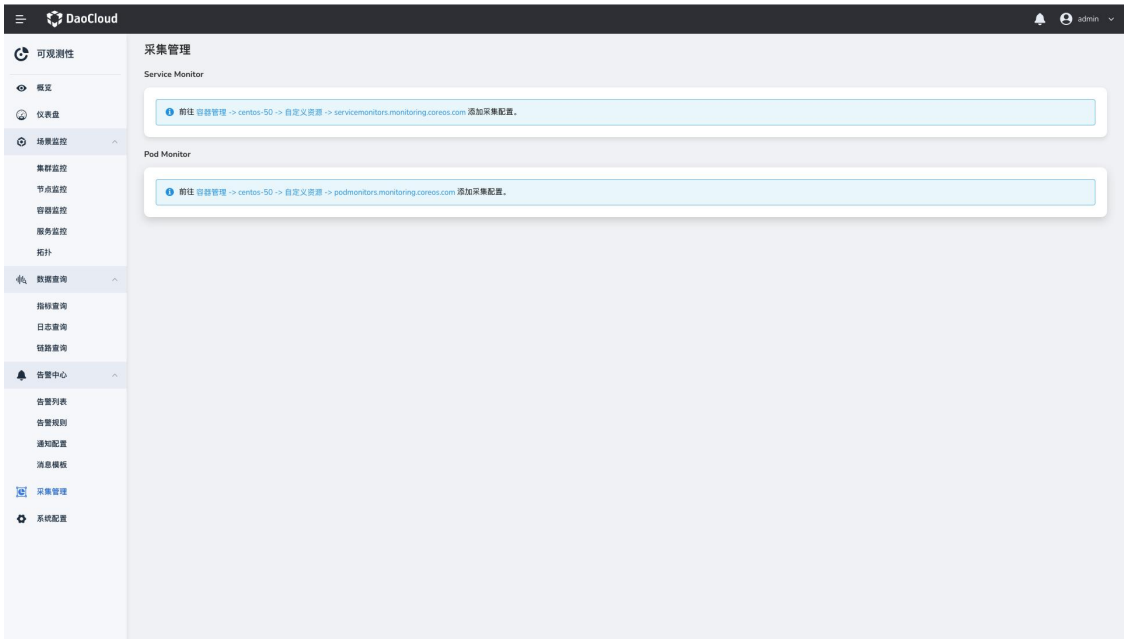
集群列表

3. 集群接入 insight-agent 且处于运行中状态时，点击某个集群名称进入详情。



集群详情

4. 在服务监控页签中，点击快捷链接跳转到容器管理 –> 自定义资源添加服务发现规则。

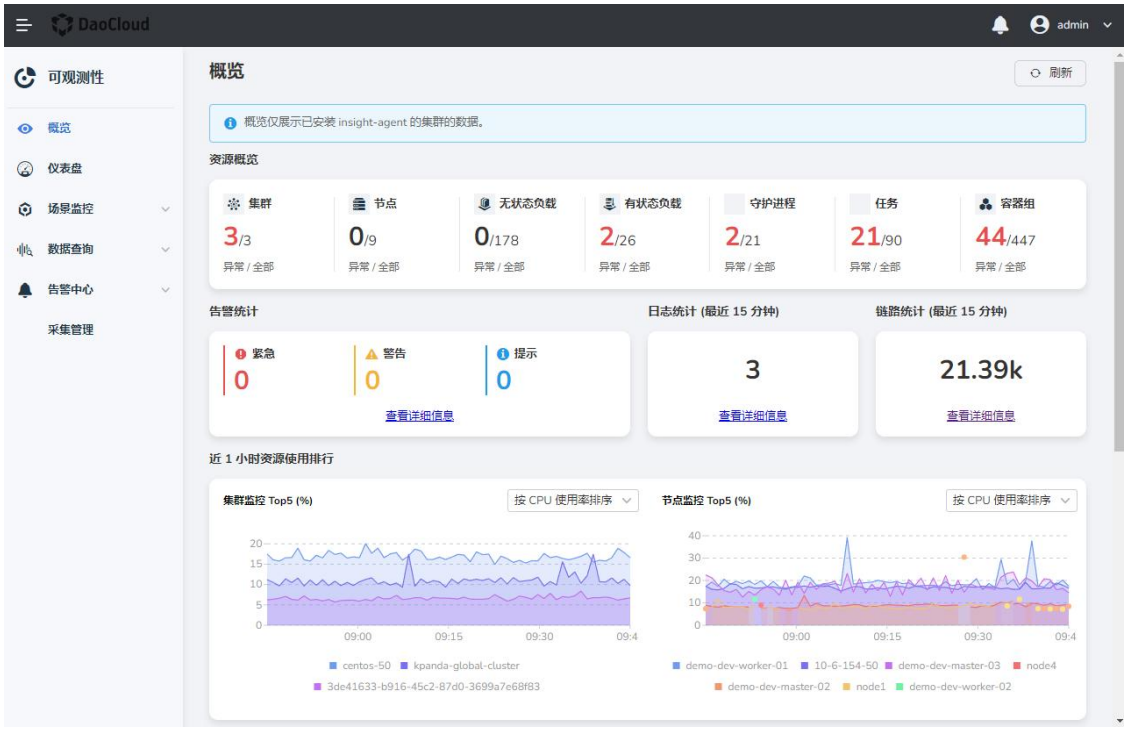


集群详情

概览

本文介绍了 容器管理 中所管理集群的总览信息，包括所有集群资源的异常数和总数、告警、链路数据、日志数据的统计，Top5 集群和节点的资源使用率，节点、容器组的变化趋势以及服务延时的 Top5。

1. 在左边导航栏选择概览，进入详情页。



概览

2. （可选）点击右上角刷新按钮，自动更新数据。

   - 告警统计：可查看所有集群的正在告警的统计数据。
   - 资源消耗：可按 CPU 使用率、内存使用率和磁盘使用率分别查看近一小时 TOP5 集群、节点的资源变化趋势。 默认按照根据 CPU 使用率排序。您可切换指标切换集群、节点的排序方式。

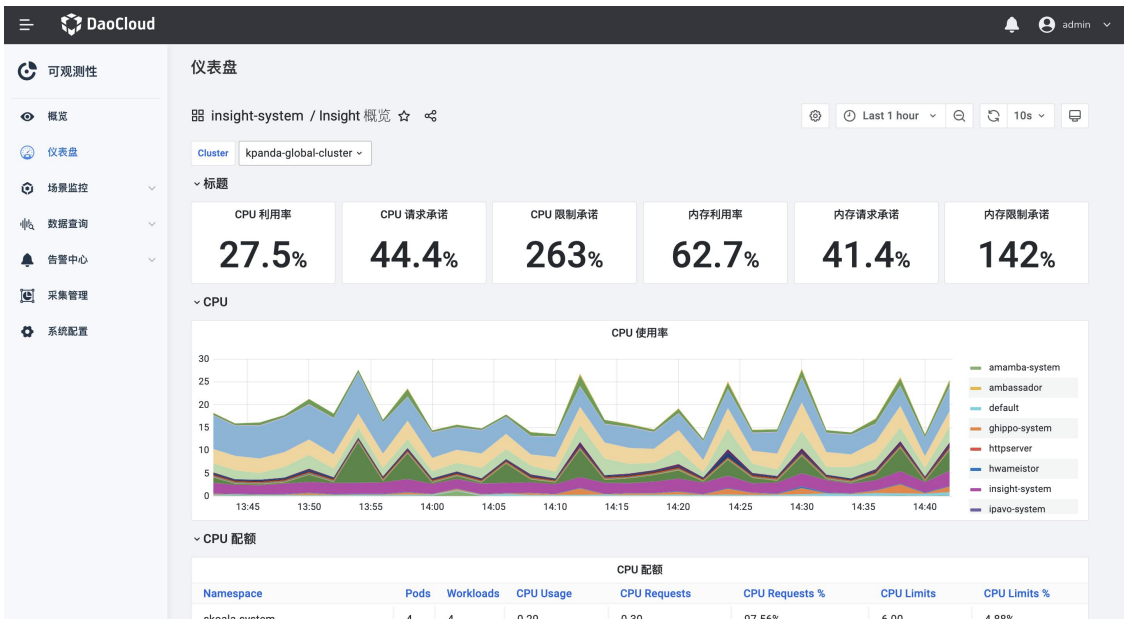
- 资源变化趋势：可查看近 15 天的节点个数趋势以及一小时 Pod 的运行趋势。
- 服务请求排行：可查看多集群中请求延时、错误率排行 TOP5 的服务及所在集群和命名空间。

仪表盘

Grafana 是一跨平台的开源的可视化分析工具。可观测性 Insight 使用开源 Grafana 提供监控服务，支持从集群、节点、命名空间等多维度查看资源消耗情况，

关于开源 Grafana 的详细信息，请参见 Grafana 官方文档。

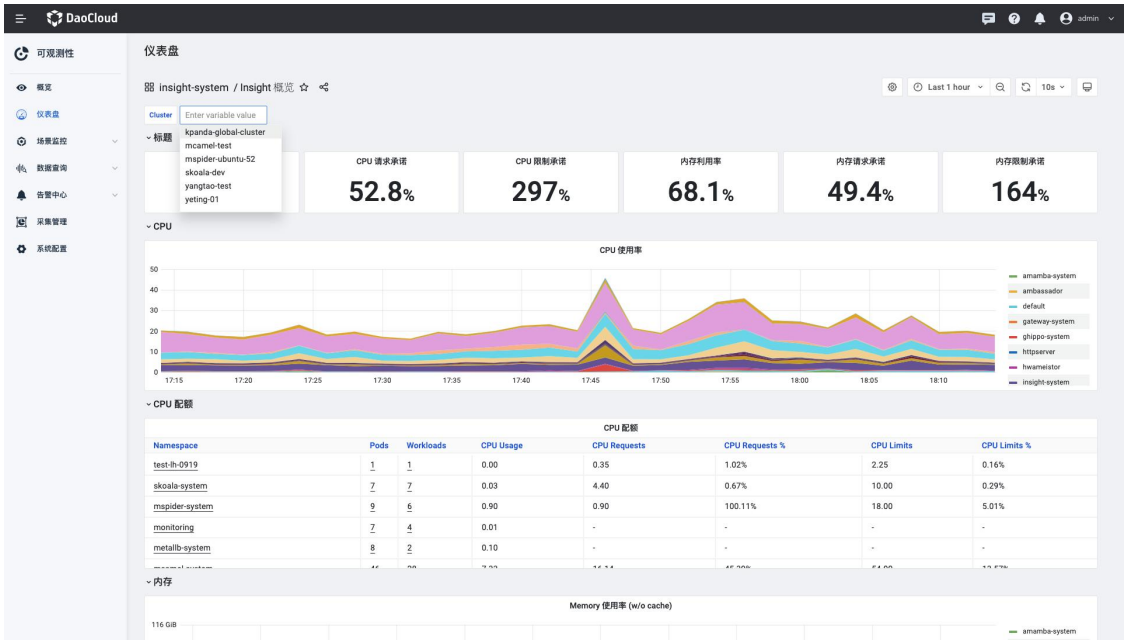
1. 在左侧导航栏选择仪表盘。



dashboard

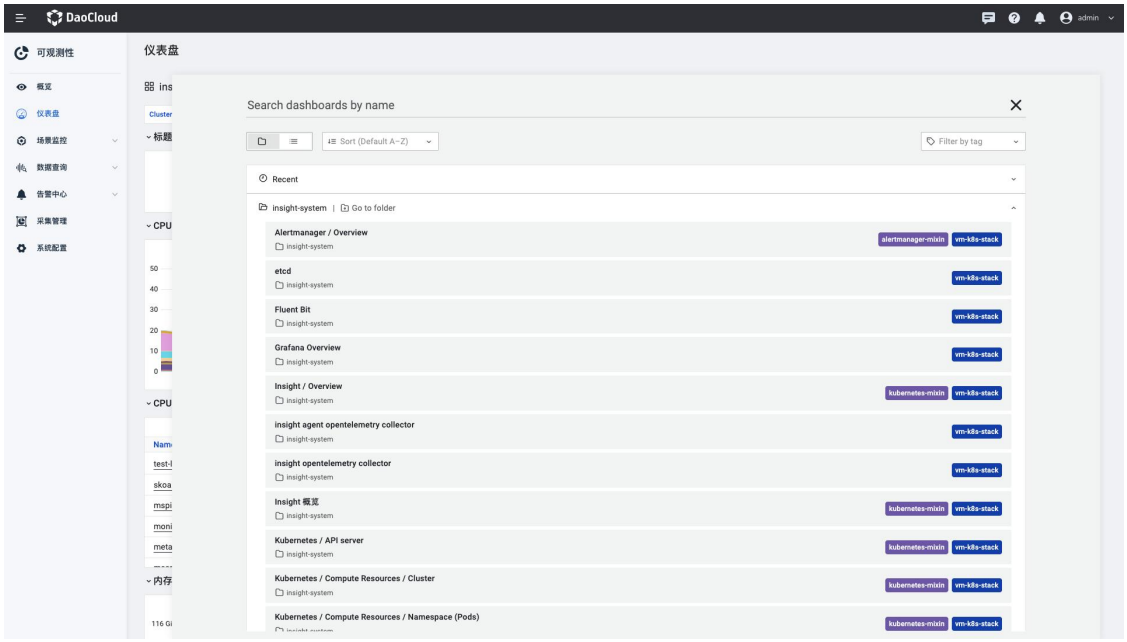
2. 以 Insight 概览 为例，执行以下步骤，设置相关参数，查看集群指标。

   在页面左上角处，选择集群，查看选中集群中的资源使用情况。



dashboard

3. 点击标题 Insight 概览 区域，可切换仪表盘。



dashboard

!!! note

访问原生 Grafana 请参考：[以管理员身份登录 Grafana](../../user-guide/dashboard/login-grafana.md)

导入自定义仪表盘请参考：[导入自定义仪表盘](./import-dashboard.md)

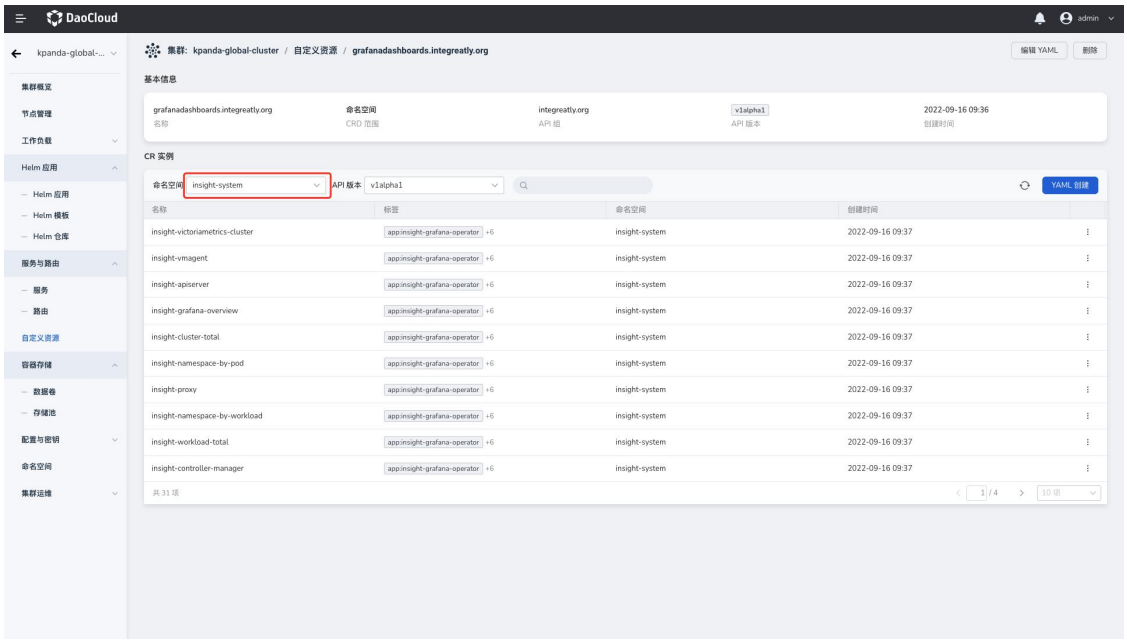
导入自定义仪表盘

本页介绍如何通过 CRD 和 UI 界面导入自定义的仪表盘。

通过 CRD 导入仪表盘

1. 登录 DCE 5.0 平台，从左侧导航栏进入容器管理，在集群列表中选择目标集群。

2. 选择左侧导航栏的自定义资源，在列表中查找 grafanadashboards.integreatly.org 文件，进入详情。



导入仪表盘

3. 点击 Yaml 创建，使用以下模板，在 Json 字段中替换仪表盘 JSON。

   - namespace：填写目标命名空间；
   - name：填写仪表盘的名称。

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
```

```yaml
metadata:
  labels:
    app: insight-grafana-operator
    operator.insight.io/managed-by: insight
  name: sample-dashboard
  namespace: insight-system
spec:
  json: >
    {
      "id": null,
      "title": "Simple Dashboard",
      "tags": [],
      "style": "dark",
      "timezone": "browser",
      "editable": true,
      "hideControls": false,
      "graphTooltip": 1,
      "panels": [],
      "time": {
        "from": "now-6h",
        "to": "now"
      },
      "timepicker": {
        "time_options": [],
        "refresh_intervals": []
      },
      "templating": {
        "list": []
      },
      "annotations": {
        "list": []
      },
```

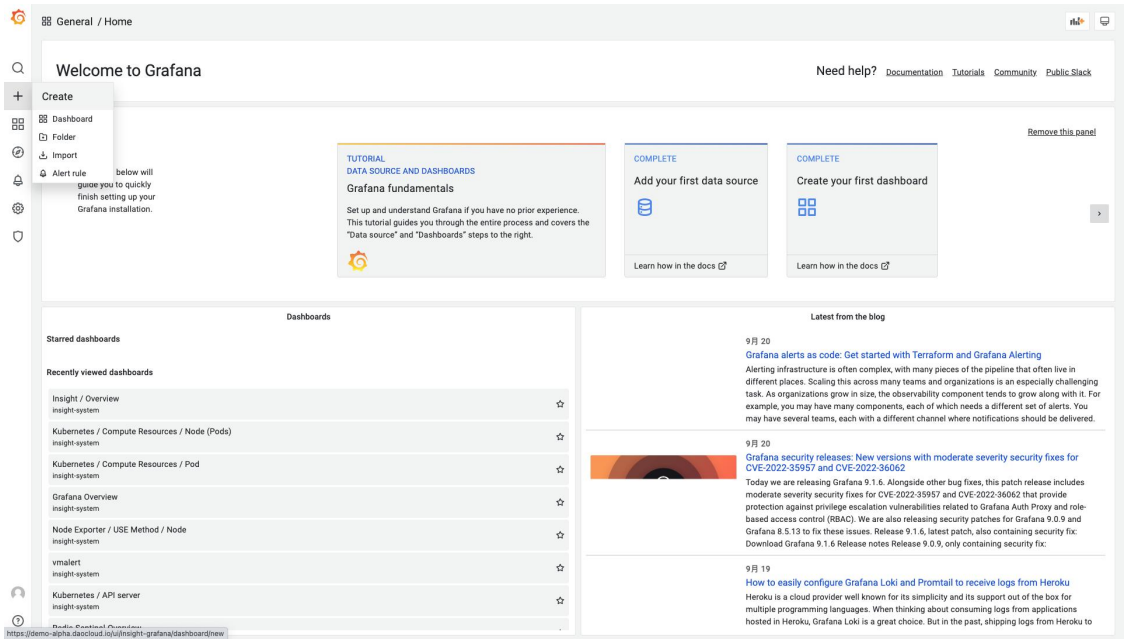
```
    "refresh": "5s",
    "schemaVersion": 17,
    "version": 0,
    "links": []
}
```


4. 点击确认后，稍等片刻即可在仪表盘中查看刚刚导入的仪表盘。

通过 UI 导入仪表盘

本节说明如何访问原生 Grafana 构建自定义仪表盘。

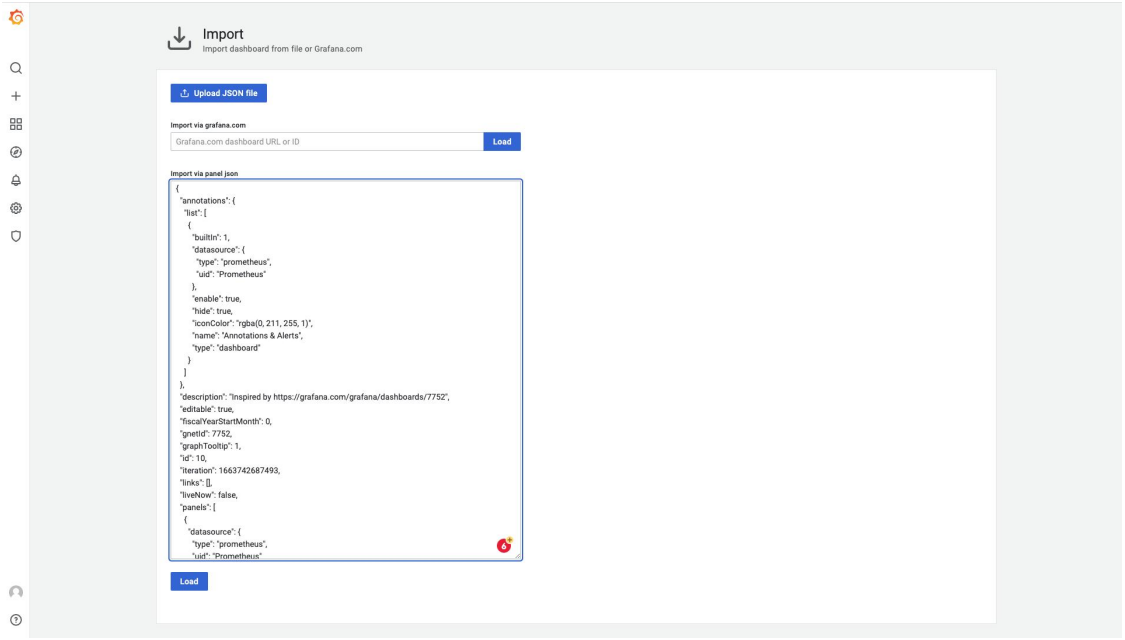
1. 点击左侧导航栏 Dashboards > Import。



导入仪表盘

2. 可选择以下方式导入：

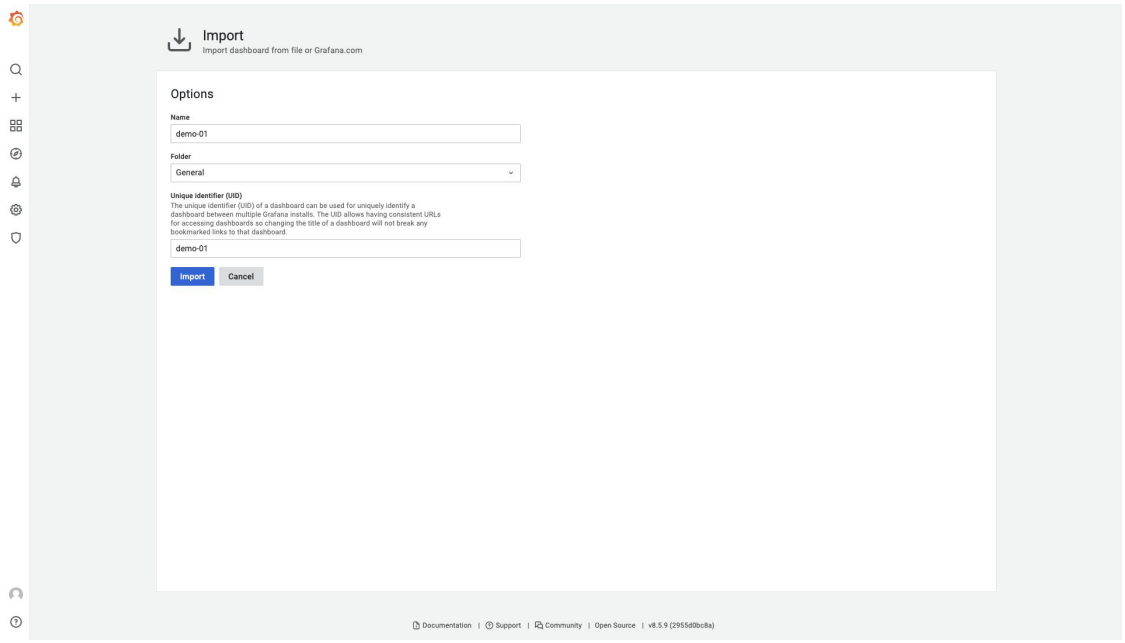
   - 上传仪表盘 JSON 文件
   - 粘贴 Grafana.com 仪表盘 URL
   - 粘贴仪表盘 JSON 到文本区域



导入仪表盘

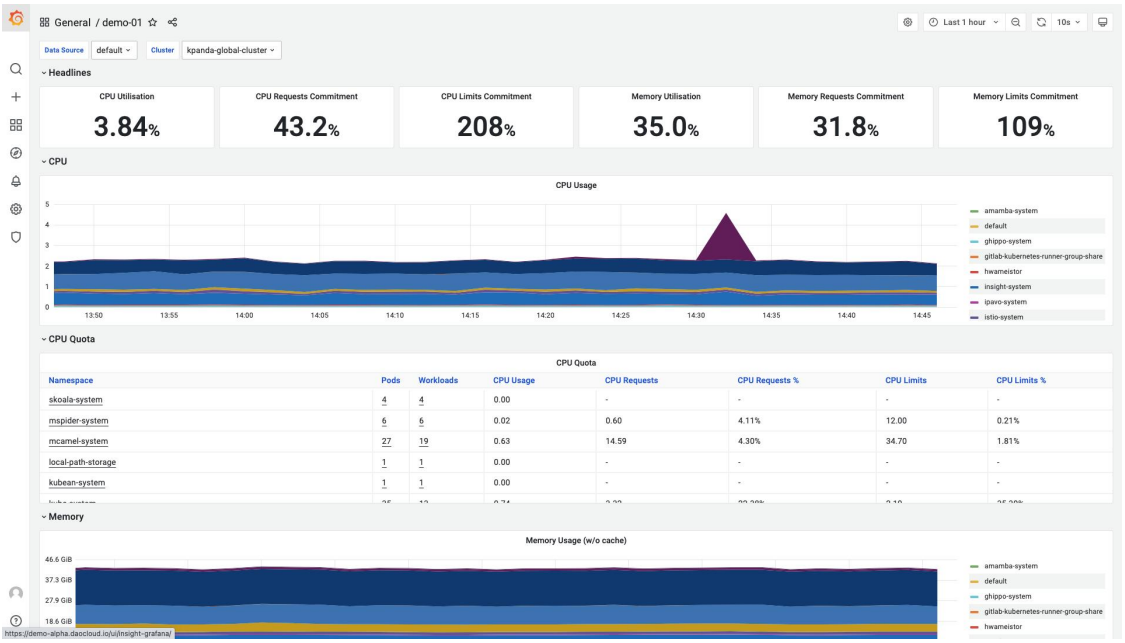
3. 点击 Load 后，填写以下参数：

   - name：设置仪表盘的名称
   - Floder：选择仪表盘存储的目标路径
   - Prometheus：选择数据源



导入仪表盘

4. 点击 Import 即可成功导入仪表盘。



导入仪表盘

!!! info

如果需要自定义仪表盘，请参考[添加仪表盘面板](https://grafana.com/docs/grafana/latest/dashboards/add-organize-panels/)。

!!! warning

目前无法为自定义的仪表盘提供持久化服务，请在升级或重启 insight 前备份您自定义的仪表盘。

访问原生 Grafana

请确认您的全局管理集群中 Helm 应用 Insight 处于运行中状态。

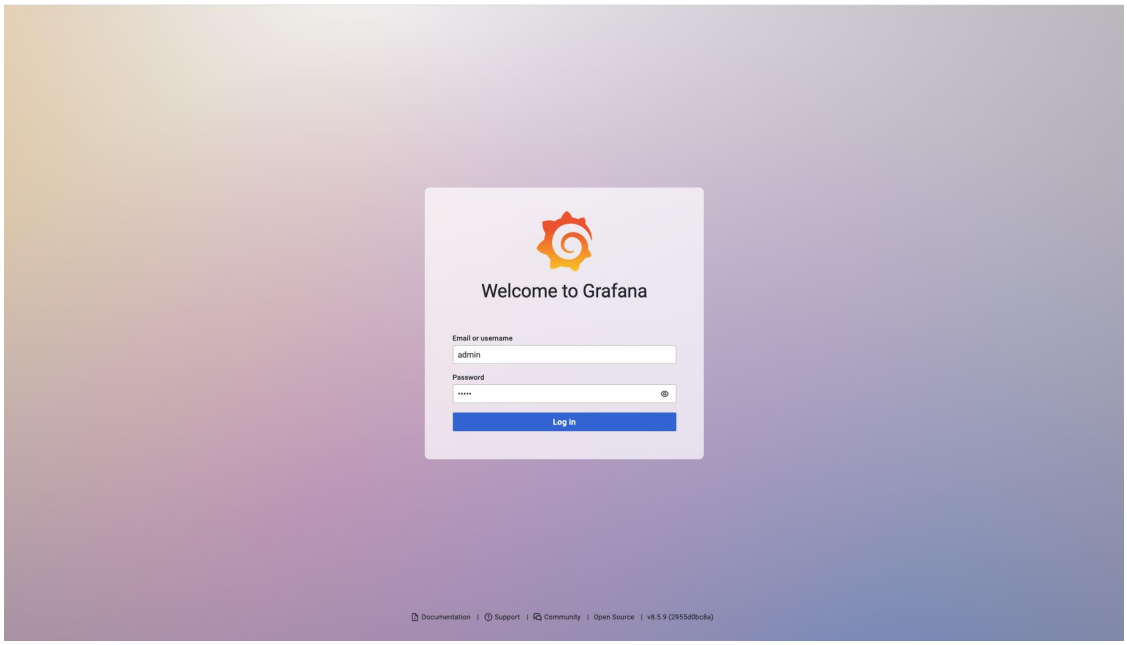
具体操作步骤如下：

1. 登录控制台，访问原生 Grafana。

   访问地址： http://ip:访问端口/ui/insight-grafana

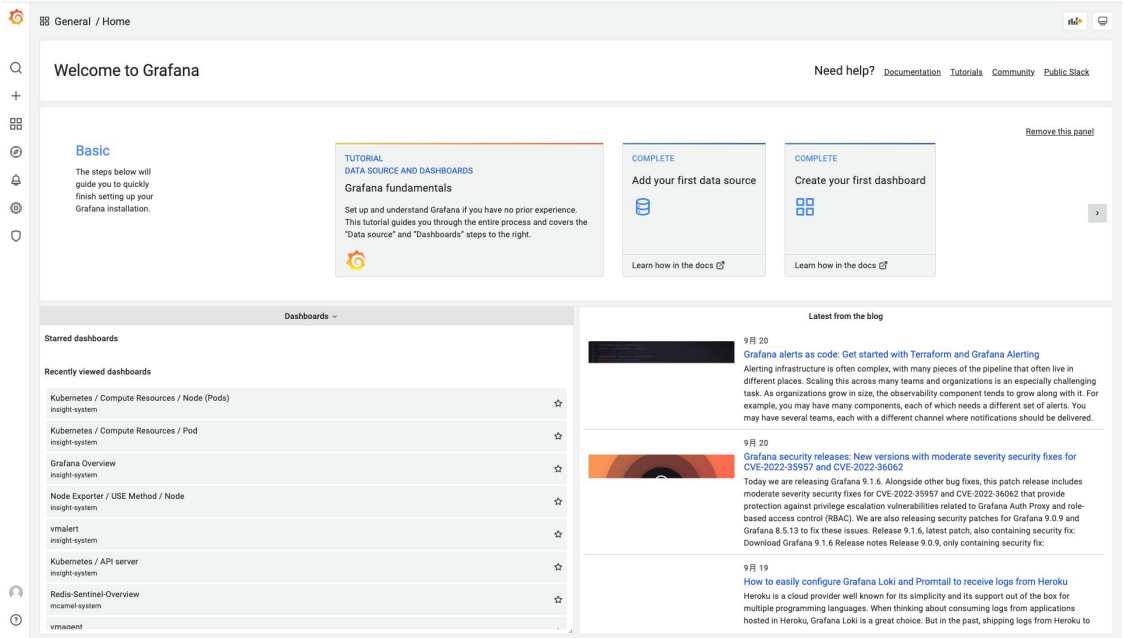
   例如： http://10.6.10.233:30209/ui/insight-grafana

2. 点击右下角的登录，使用默认用户名、密码（admin/admin）进行登录。



登录 grafana

3. 点击 Log in 完成登录。



成功登录 grafana

指标查询

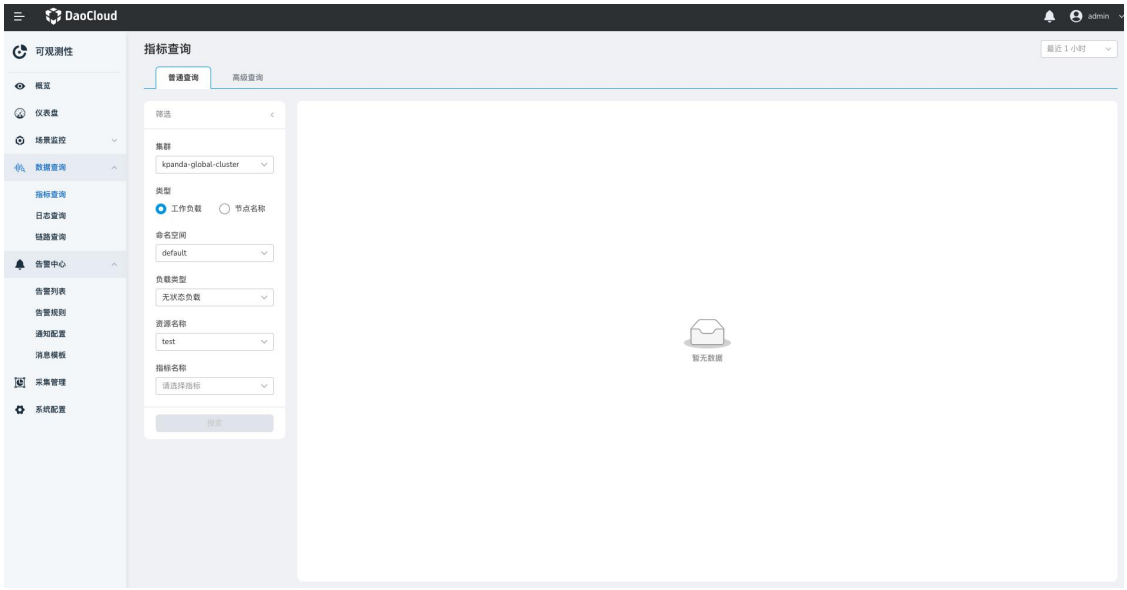
指标查询支持查询容器各资源的指标数据，可查看监控指标的趋势变化。同时，高级查询支持原生 PromQL 语句进行指标查询。

前提条件

一 集群中已安装 insight-agent 且应用处于运行中状态。

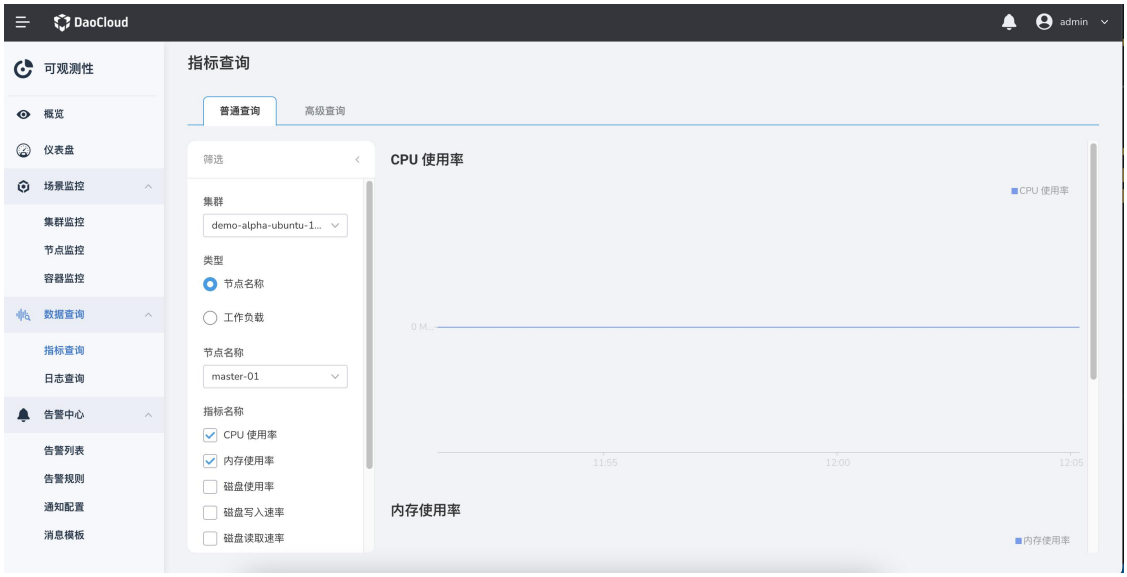
普通查询

1. 在左侧导航栏中，点击数据查询 -> 指标查询。



查询结果

2. 选择集群、类型、节点、指标名称查询条件后，点击搜索，屏幕右侧将显示对应指标图表及数据详情。



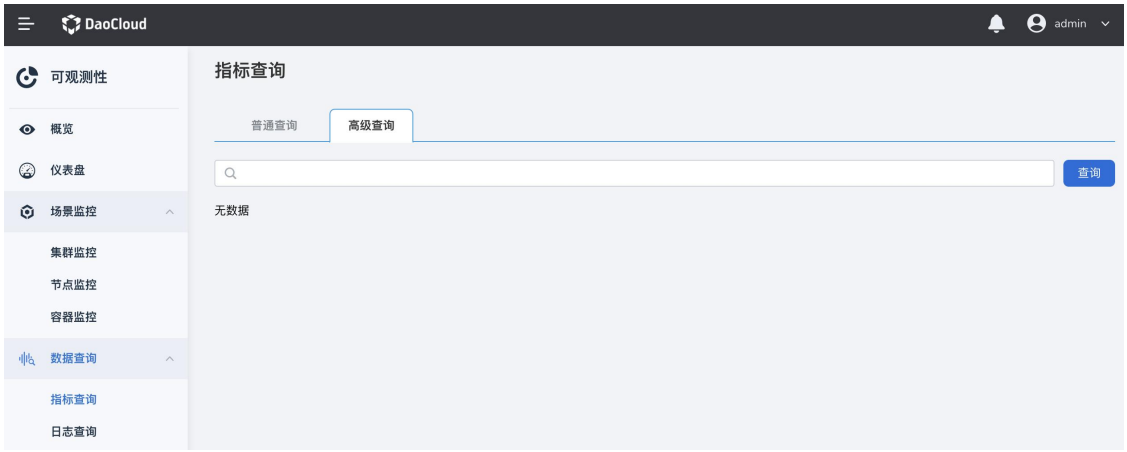
查询结果

!!! tip

支持自定义时间范围。可手动点击`刷新`图标或选择默认时间间隔进行刷新。

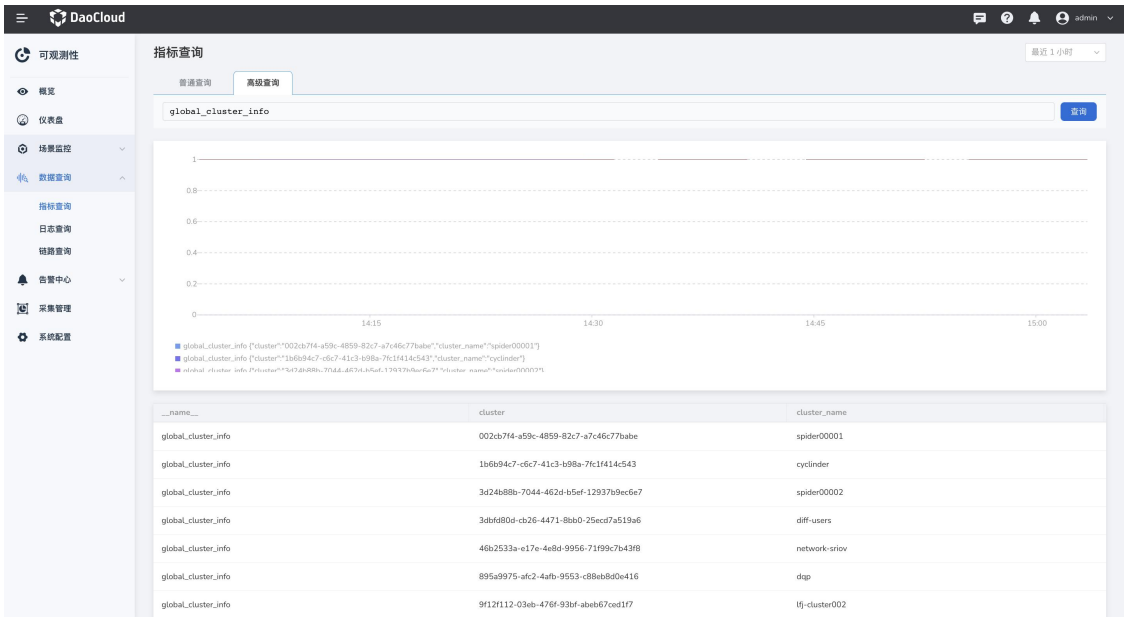
高级查询

1. 在左侧导航栏中，点击数据查询 -> 指标查询，点击高级查询页签切换到高级查询页面。



高级查询

2. 输入一条 PromQL 语句（参阅 PromQL 语法，点击查询，将显示查询的指标图表及数据详情。


DaoCloud
admin
可观测性
概览
仪表盘
场景监控
数据查询
指标查询
日志查询
链路查询
告警中心
采集管理
系统配置
指标查询
最近 1 小时
普通查询
高级查询
global_cluster_info
查询
1
0.8
0.6
0.4
0.2
0
14:15
14:30
14:45
15:00
global_cluster_info {"cluster":"002cb7f4-a59c-4859-82c7-a7c46c77babe","cluster_name":"spider00001"}
global_cluster_info {"cluster":"1b6b94c7-c6c7-41c3-b98a-7fc1f414c543","cluster_name":"cyclinder"}
__name__
cluster
cluster_name
global_cluster_info 002cb7f4-a59c-4859-82c7-a7c46c77babe spider00001
global_cluster_info 1b6b94c7-c6c7-41c3-b98a-7fc1f414c543 cyclinder
global_cluster_info 3d24b88b-7044-462d-b5ef-12937b9ec6e7 spider00002
global_cluster_info 3dbfd80d-cb26-4471-8bb0-25ecd7a519a6 diff-users
global_cluster_info 46b2533a-e17e-4e8d-9956-71f99c7b43f8 network-sriov
global_cluster_info 895a9975-afc2-4afb-9553-c88ebb8d0e416 dqp
global_cluster_info 9f12f112-03eb-476f-93bf-abeb67ced1f7 lfj-cluster002

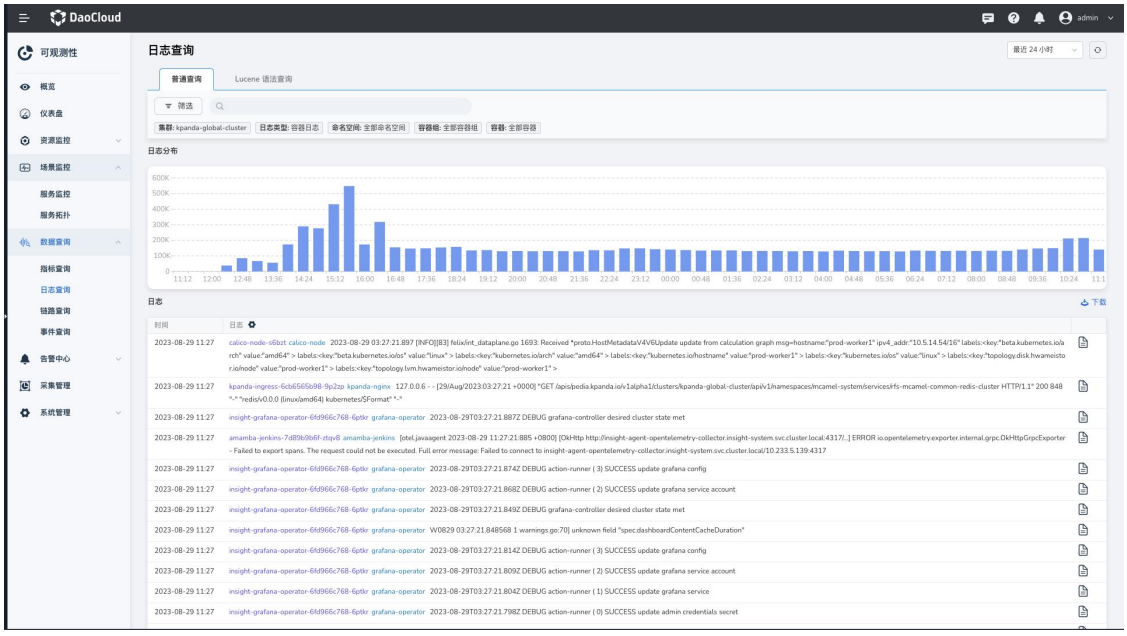
查询结果

日志查询

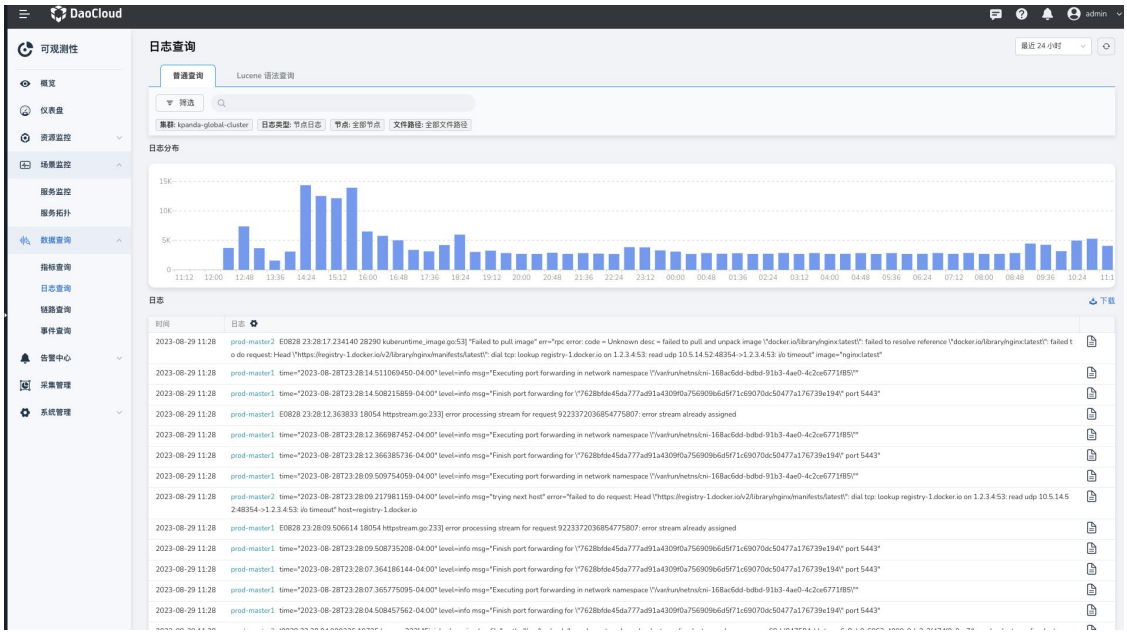
Insight 默认采集节点日志、容器日志以及 kubernetes 审计日志。在日志查询页面中，可查询登录账号权限内的标准输出（stdout）日志， 包括节点日志、产品日志、Kubenetes 审计日志等，快速在大量日志中查询到所需的日志，同时结合日志的来源信息和上下文原始数据辅助定位问题。

操作步骤

1. 点击一级导航栏进入可观测性。
2. 左侧导航栏中，选择数据查询 –> 日志查询。
   - 默认查询最近 24 小时；
   - 第一次进入时，默认根据登录账号权限查询有权限的集群或命名空间的容器日志；



log



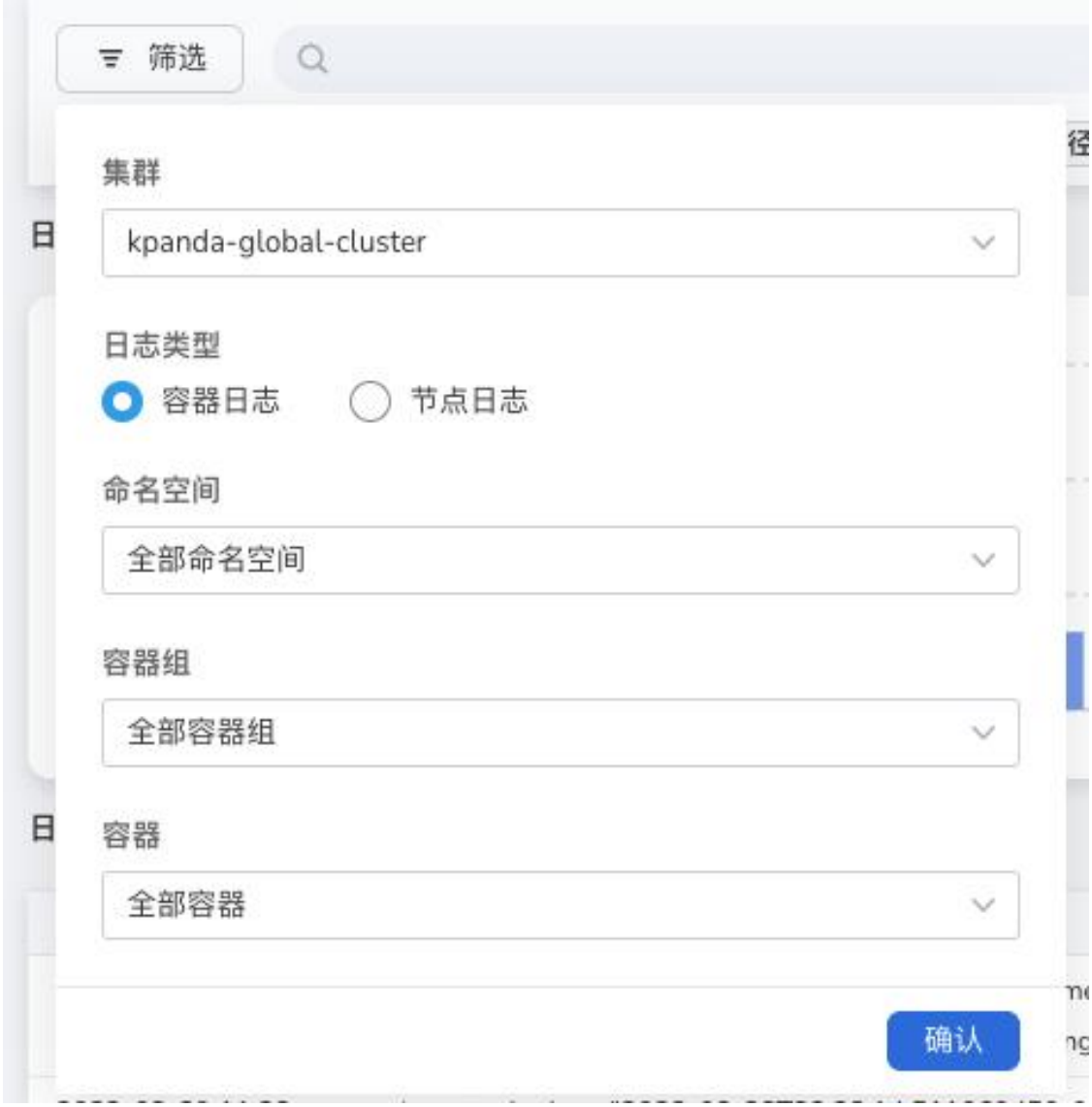
log

3. 顶部 Tab 默认进入 普通查询 。

    - 点击筛选 展开过滤面板，可切换日志搜索条件和类型。
    - 日志类型：

        - 容器日志：记录集群中容器内部的活动和事件，包括应用程序的输出、错误消息、警告和调试信息等。支持通过集群、命名空间、容器组、容器过滤日志。
        - 节点日志：记录集群中每个节点的系统级别日志。这些日志包含节点的操作系统、内核、服务和组件的相关信息。支持通过集群、节点、文件路径过滤日志。

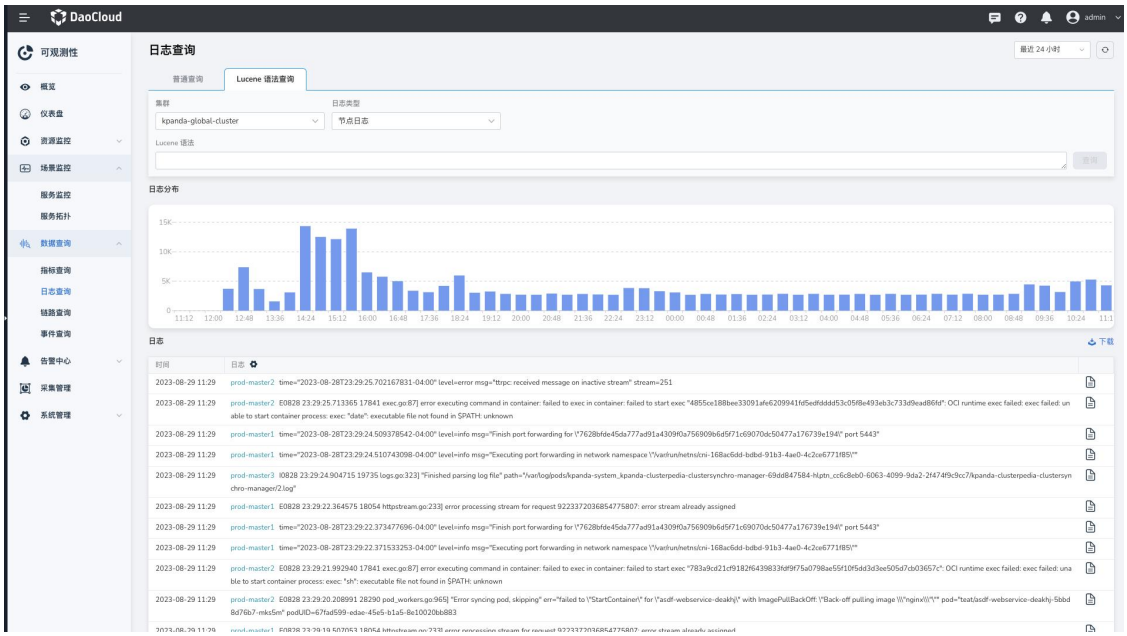
    - 支持对单个关键字进行模糊搜索。



log

4. 顶部切换 Tab 选择 Lucene 语法查询。

   第一次进入时，默认选择登录账号权限查询有权限的集群或命名空间的容器日志。



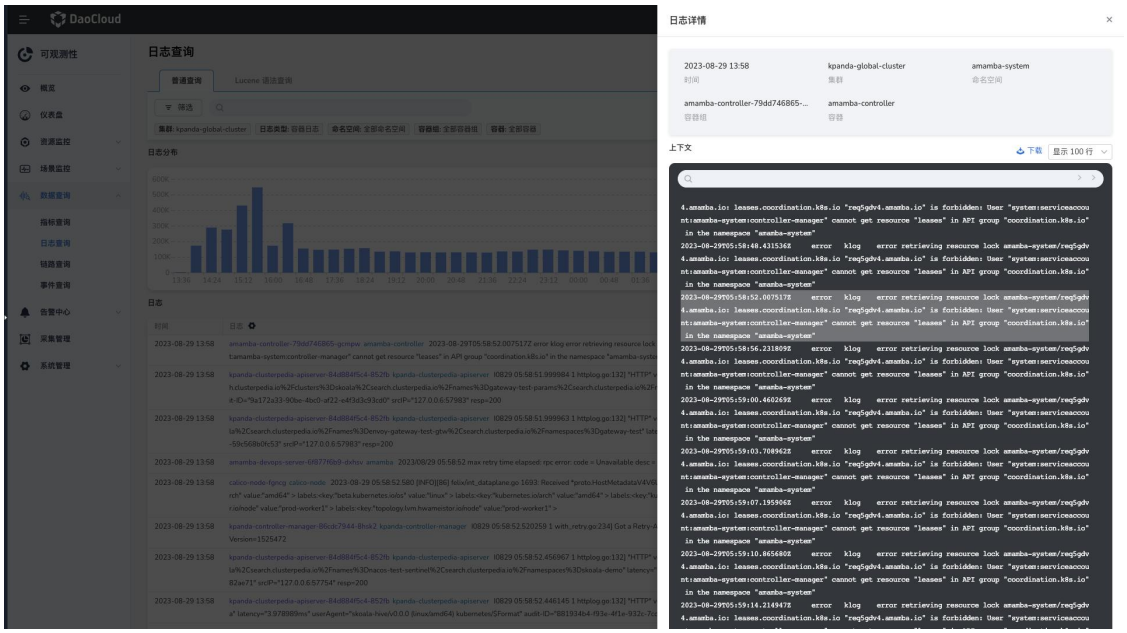
log

Lucene 语法说明：

1. 使用 逻辑操作符（AND、OR、NOT、“”）符查询多个关键字，例如：keyword1 AND (keyword2 OR keyword3) NOT keyword4。
2. 使用波浪号（~）实现模糊查询，在“~”后可指定可选的参数，用于控制模糊查询的相似度，不指定则默认使用 0.5。例如：error~ 。
3. 使用通配符（*、?）用作单字符通配符，表示匹配任意一个字符。
4. 使用方括号 [ ] 或花括号 { } 来查询范围，方括号 [ ] 表示闭区间，包含边界值。花括号 { } 表示开区间，排除边界值。范围查询只适用于能够进行排序的字段类型，如数字、日期等。例如：timestamp:[2022-01-01 TO 2022-01-31]。
5. 更多用法请查看：Lucene 语法说明。

查看日志上下文

点击日志后的按钮，在右侧划出面板中可查看该条日志的默认 100 条上下文。可切换显示行数查看更多上下文内容。



log

导出日志数据

点击列表右上侧的下载按钮。

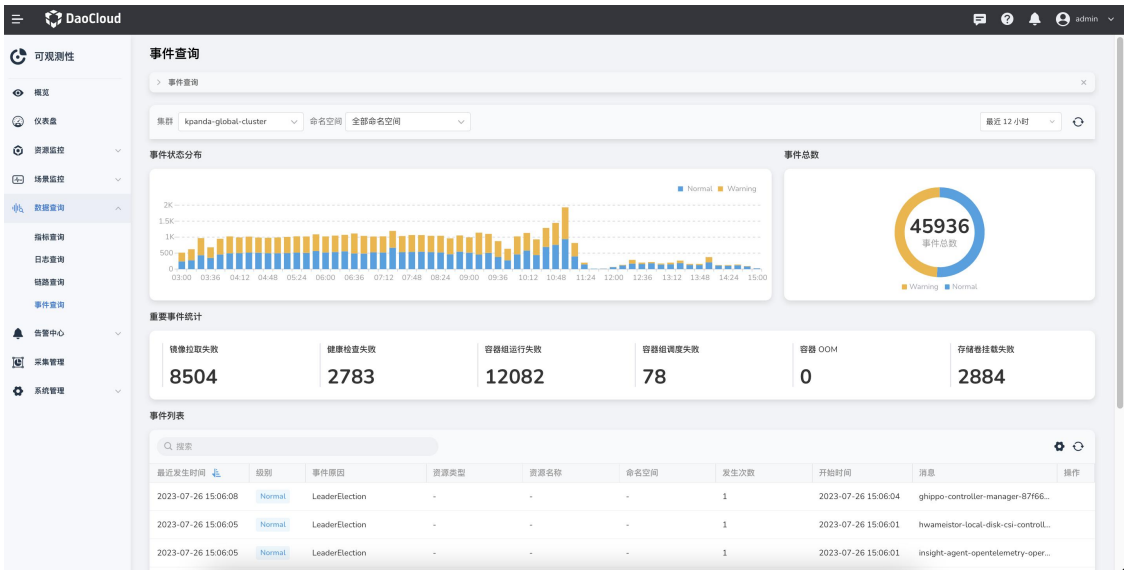
- 支持配置导出的日志字段，根据日志类型可配置的字段不同，其中日志内容字段为必选。
- 支持将日志查询结果导出为 **.txt** 或 **.csv** 格式。



log

事件查询

DCE 5.0 Insight 支持按集群、命名空间查询事件，



event

事件状态分布

默认显示最近 12 小时内发生的事件，您可以在右上角选择不同的时间范围来查看较长或较短的时间段。 您还可以自定义采样间隔为 1 分钟至 5 小时。

通过事件状态分布图，您可以直观地了解事件的密集程度和分散情况。 这有助于对后续的集群运维进行评估，并做好准备和安排工作。 如果事件密集发生在特定时段，您可能需要调配更多的资源或采取相应措施来确保集群稳定性和高可用性。而如果事件较为分散，在此期间您可以合理安排其他运维工作，例如系统优化、升级或处理其他任务。

通过综合考虑事件状态分布图和时间范围，您能更好地规划和管理集群的运维工作，确保系统稳定性和可靠性。

事件总数和统计

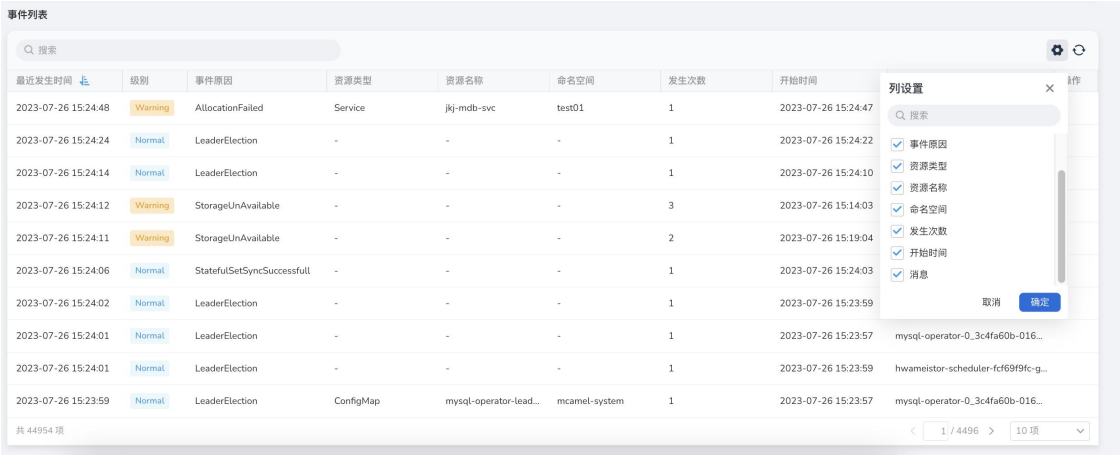
通过重要事件统计，您可以方便地了解镜像拉取失败次数、健康检查失败次数、容器组（Pod）运行失败次数、 Pod 调度失败次数、容器 OOM 内存耗尽次数、存储卷挂载失败次数以及所有事件的总数。这些事件通常分为「Warning」和「Normal」两类。

事件列表

事件列表以时间为轴，以流水的形式展示发生的事件。您可以根据「最近发生时间」和「级别」进行排序。

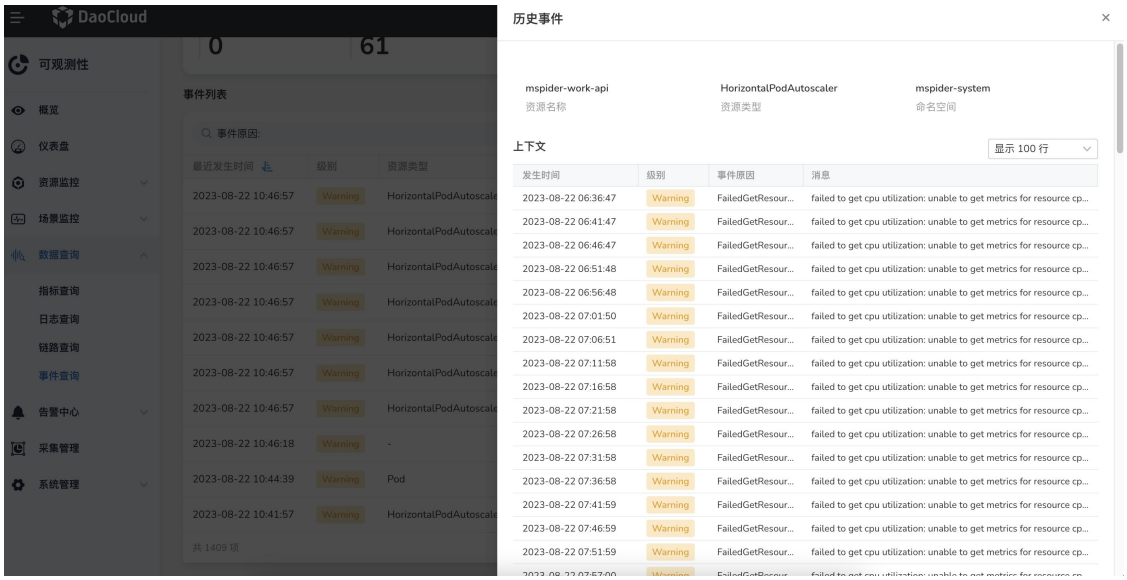
点击右侧的　　图标，您可以根据自己的喜好和需求来自定义显示的列。

在需要的时候，您还可以点击刷新图标来更新当前的事件列表。



list

在右侧的操作一列中，点击图标可以查看某一个 Event 的历史记录。



history

参考

有关系统自带的 Event 事件的详细含义，请参阅 Kubenetest API 事件列表。

链路查询

在链路查询页面，您可以过 TraceID 或精确查询调用链路详细情况或结合多种条件筛选查询调用链路。

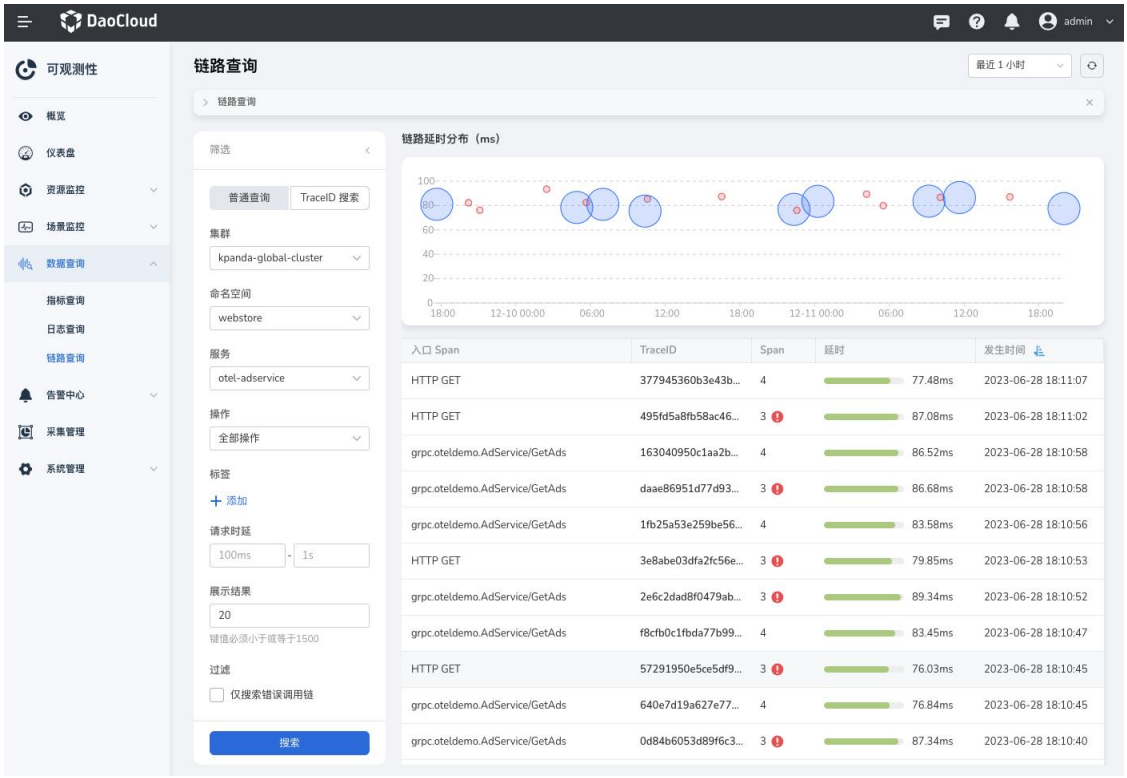
名词说明

- TraceID：用于标识一个完整的请求调用链路。
- 操作：描述 Span 所代表的具体操作或事件。
- 入口 Span：入口 Span 代表了整个请求的第一个请求。
- 延时：整个调用链从开始接收请求到完成响应的持续时间。
- Span：整个链路中包含的 Span 个数。
- 发生时间：当前链路开始的时间。
- Tag：一组键值对构成的 Span 标签集合，Tag 是用来对 Span 进行简单的注解和补充，每个 Span 可以有多个简直对形式的 Tag。

操作步骤

请按照以下步骤查询链路：

1. 进入可观测性 产品模块，
2. 在左边导航栏选择 数据查询 -> 链路查询。
3. 通过多种条件查询链路或通过 TraceID 精确查询链路。

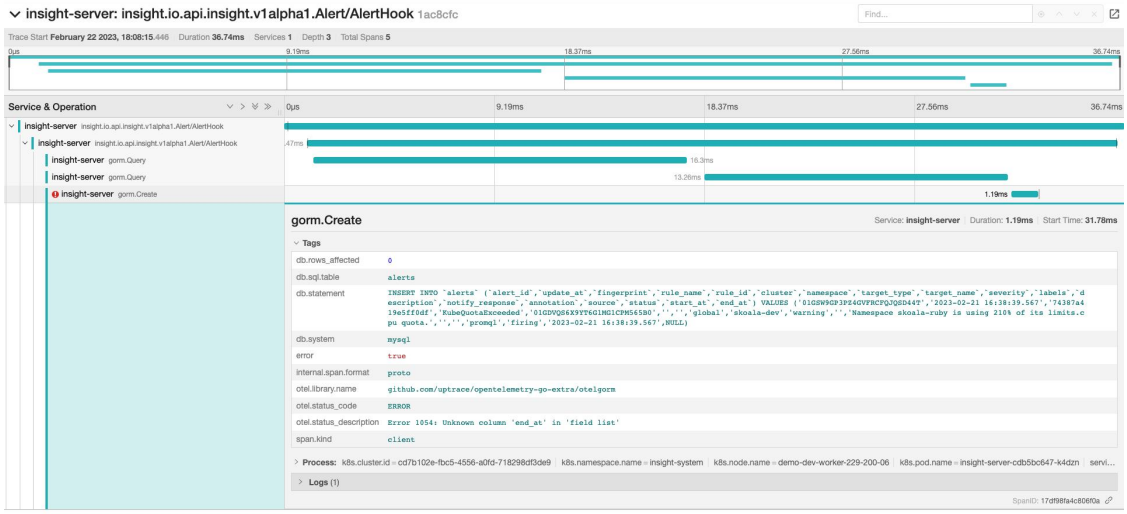


jaeger

!!! note

列表中支持对 Span 数、延时、发生时间进行排序。

4. 点击需要查询的 TraceID 名称，查看该链路的详情调用情况



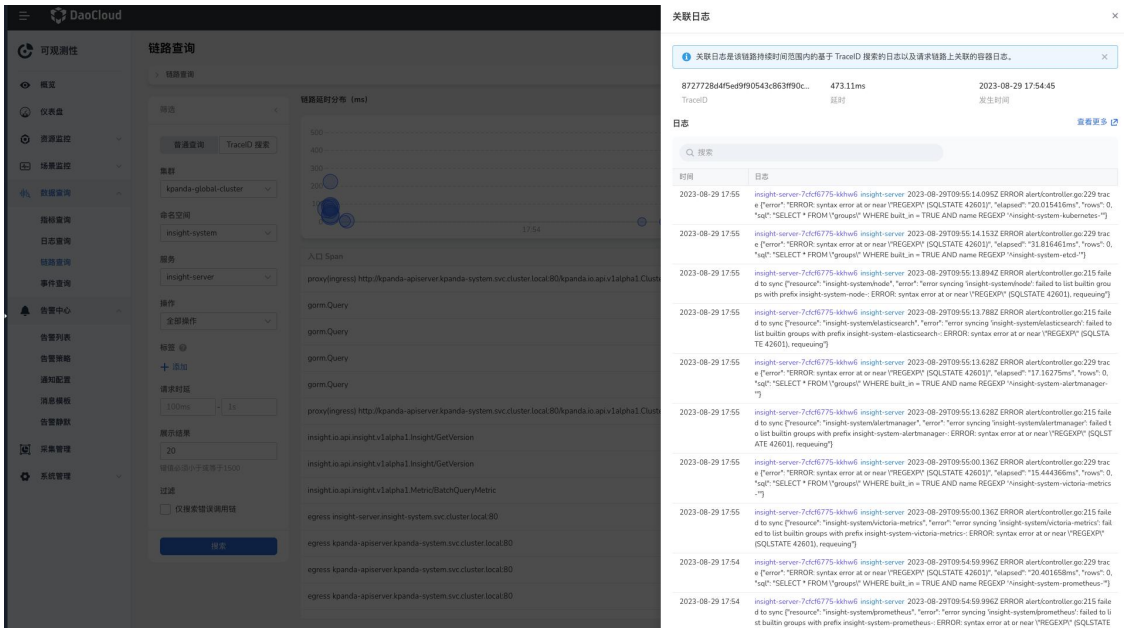
jaeger

!!! note

使用 TraceID 搜索请输入完整的 TraceID。

关联日志

1. 点击链路数据右侧的图标，可查询该链路的关联日志。
    - 默认查询该链路的持续时间及其结束之后一分钟内的日志数据。
    - 查询的日志内容为日志文本中包含该链路的 TraceID 的日志和链路调用过程中相关的容器日志。
2. 点击 查看更多 后可带条件跳转到日志查询 的页面。
3. 支持基于 容器组 过滤和关键字模糊搜索。



tracelog

!!! note

1. 由于链路会跨集群或跨命名空间，若用户权限不足，则无法查询该链路的关联日志。
2. 跳转到 `日志查询` 后会根据用户权限默认搜索一次。

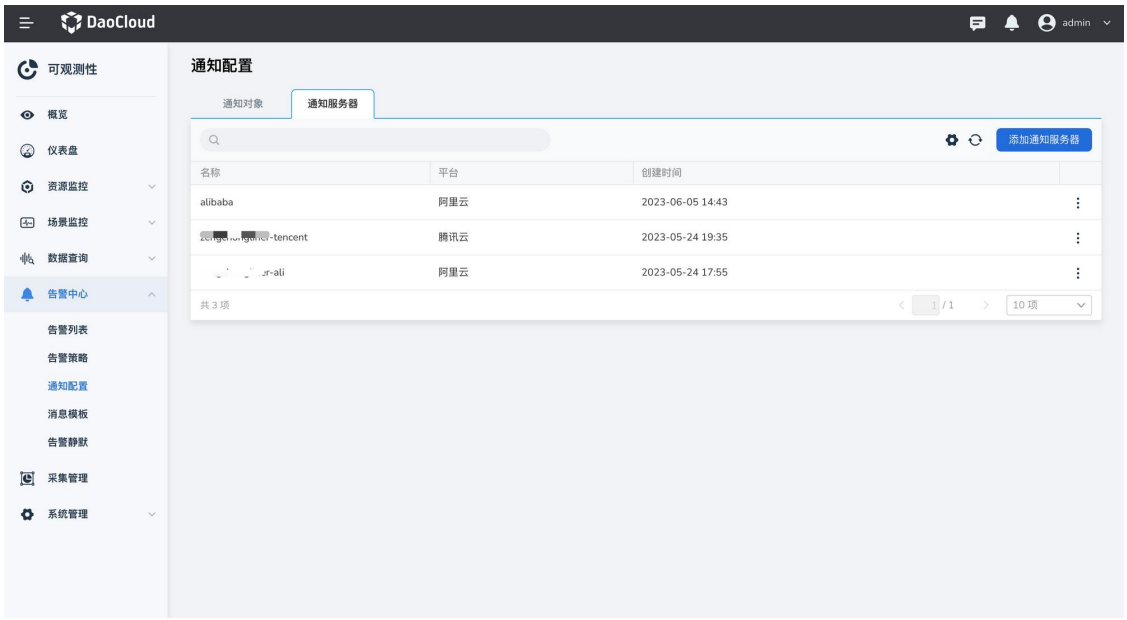
配置通知服务器

可观测性 Insight 支持短信通知，目前通过集成阿里云、腾讯云的短信服务发送告警消息。本文介绍了如何在 insight 中配置短信通知的服务器。短信签名中支持的变量为消息模板中的默认变量，同时由于短信字数有限，建议选择较为明确的变量。

如何配置短信接收人可参考文档：配置短信通知组。

操作步骤

1. 进入 告警中心 -> 通知配置 -> 通知服务器。



通知服务器

2. 点击添加通知服务器。

   - 配置阿里云服务器。

     申请阿里云短信服务，请参考阿里云短信服务。

     字段说明：

- AccessKey ID： 阿里云用于标识用户的参数。
- AccessKey Secret：阿里云用于验证用户的密钥。AccessKey Secret 必须保密。
- 短信签名：短信服务支持根据用户需求创建符合要求的签名。发送短信时，短信平台会将已审核通过的短信签名添加到短信内容中，再发送给短信接收方。
- 模板 CODE：短信模板是发送短信的具体内容。
- 参数模板：短信正文模板可以包含变量，用户可通过变量实现自定义短信内容。

请参考阿里云变量规范。



通知服务器

!!! note

举例：在阿里云定义的模板内容为：${severity}：${alertname} 在 ${startat} 被触发。参数模板中的配置参考上图。

一 配置腾讯云服务器。

申请腾讯云短信服务，请参考腾讯云短信。

字段说明：

- Secret ID：腾讯云用于标识 API 调用者身份参数。
- SecretKey：腾讯云用于验证 API 调用者的身份的参数。
- 短信模板 ID：短信模板 ID，由腾讯云系统自动生成。
- 签名内容：短信签名内容，即在腾讯云短信签名中定义的实际网站名的全称或简称。
- SdkAppId：短信 SdkAppId，在腾讯云短信控制台添加应用后生成的实际 SdkAppId。
- 参数模板：短信正文模板可以包含变量，用户可通过变量实现自定义短信内容。请参考：腾讯云变量规范。



通知服务器

!!! note

举例：在腾讯云定义的模板内容为：{1}：{2} 在 {3} 被触发。参数模板中的配置参考上图。

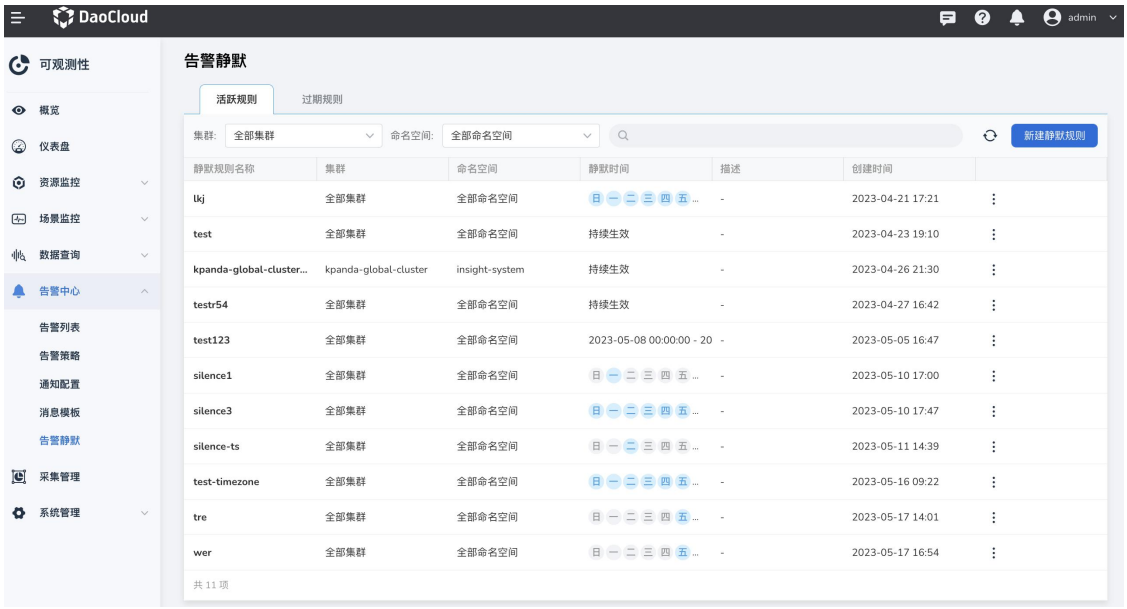
告警静默

告警静默是指在特定的时间范围内，根据定义好的规则对符合条件的告警不再发送告警通知。 这种功能可以帮助运维人员避免在某些操作或事件期间接收到过多的噪声告警，同时便于更加精确地处理真正需要解决的问题。

在告警静默页面上，用户可以看到两个页签：活跃规则和过期规则。 其中，活跃规则表示目前正在生效的规则，而过期规则则是以前定义过但已经过期（或者用户主动删除）的规则。

创建静默规则

1. 在左侧导航栏中，选择告警中心 -> 告警静默，点击新建静默规则按钮。



点击按钮

2. 填写静默规则的各项参数，如集群、命名空间、标签、时间等，以定义这条规则的作用范围和生效时间后点击确定。



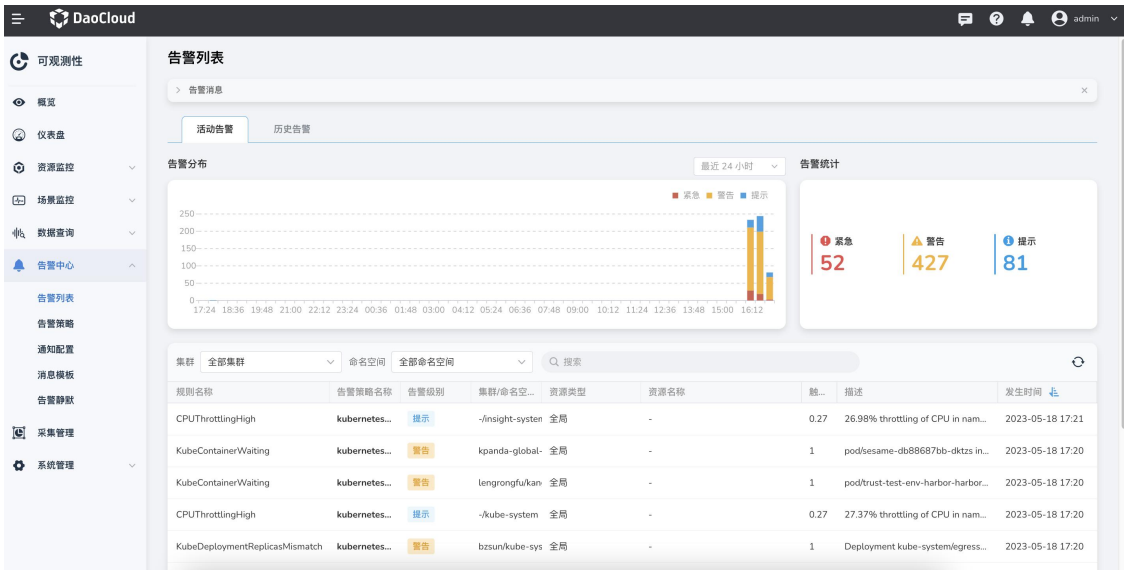
静默规则

3. 返回规则列表，在列表右侧点击 ：，可以编辑或删除静默规则。

通过告警静默功能，您可以灵活地控制哪些告警需要被忽略，在什么时间段内生效，从而提高运维效率，减少误报的可能性。

告警中心

告警中心是 DCE 5.0 提供的一个重要功能，它让用户可以通过图形界面方便地按照集群和命名空间查看所有活动和历史告警， 并根据告警级别（紧急、警告、提示）来搜索告警。



alert list

所有告警都是基于预设的告警规则设定的阈值条件触发的。在 DCE 5.0 中，内置了一些全局告警策略， 同时您也可以随时创建、删除告警策略，对以下指标进行设置：

- CPU 使用量
- 内存使用量
- 磁盘使用量
- 磁盘每秒读次数
- 磁盘每秒写次数
- 集群磁盘读取吞吐量
- 集群磁盘写入吞吐量

- 网络发送速率
- 网络接收速率

还可以为告警规则添加标签和注解。告警规则分为活跃和过期规则，支持启用/禁用某些规则来实现告警静默。

当达到阈值条件后，可以配置告警通知方式，包括邮件、钉钉、企业微信、Webhook 和短信通知。 所有通知的消息模板都可以自定义，同时还支持按设定的间隔时间发送通知。

此外，告警中心还支持通过阿里云、腾讯云等提供的短信服务将告警消息发送给指定用户，实现多种方式的告警通知。

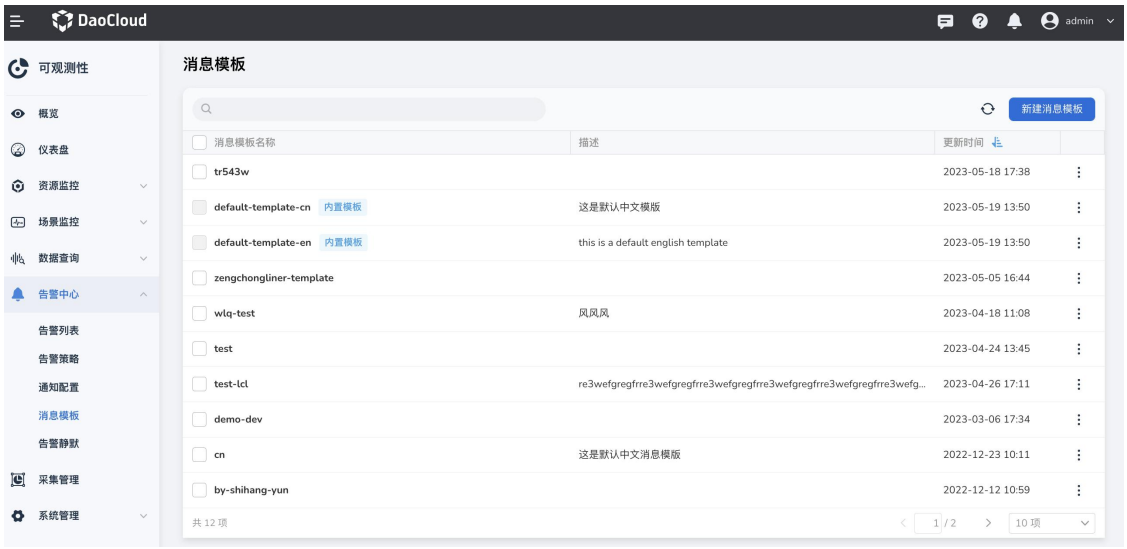
DCE 5.0 告警中心是一个功能强大的告警管理平台，可帮助用户及时发现和解决集群中出现的问题， 提高业务稳定性和可用性，便于集群巡检和故障排查。

消息模板

可观测性提供自定义消息模板内容的能力，支持邮件、企业微信、钉钉、Webhook 等不同的通知对象定义不同的消息通知内容。

创建消息模板

1. 在左侧导航栏中，选择告警中心 –> 消息模板，点击新建消息模板按钮。



点击按钮

2. 填写模板内容。



消息模板

!!! info

可观测性预置了消息模板。若需要定义模板的内容，请参考：[配置通知模板](../../best-practice/notify-helper.md)

编辑或删除消息模板

在列表右侧点击 ：，在弹出菜单中选择编辑或删除，可以修改或删除消息模板。

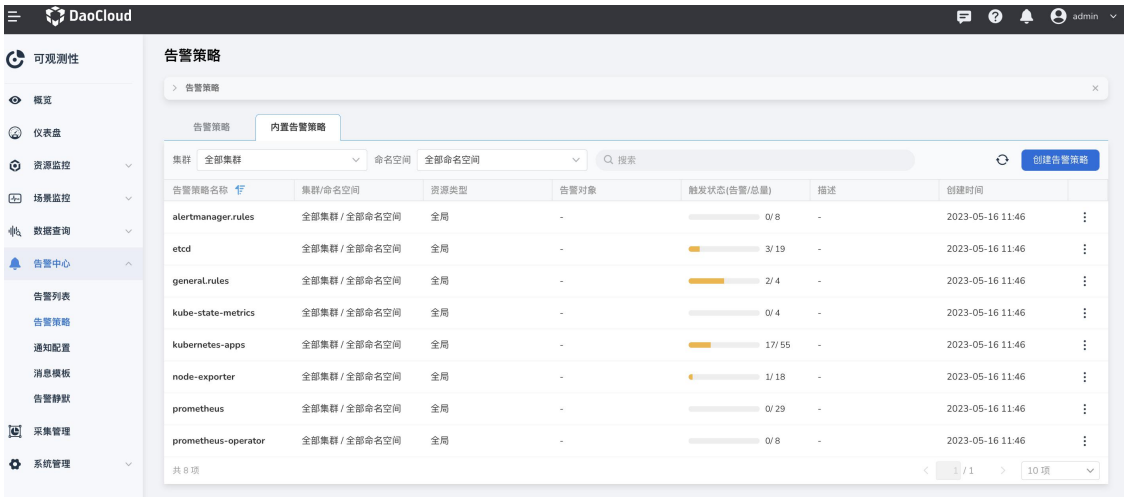
!!! warning

请注意，删除模板后无法恢复，请谨慎操作。

告警策略

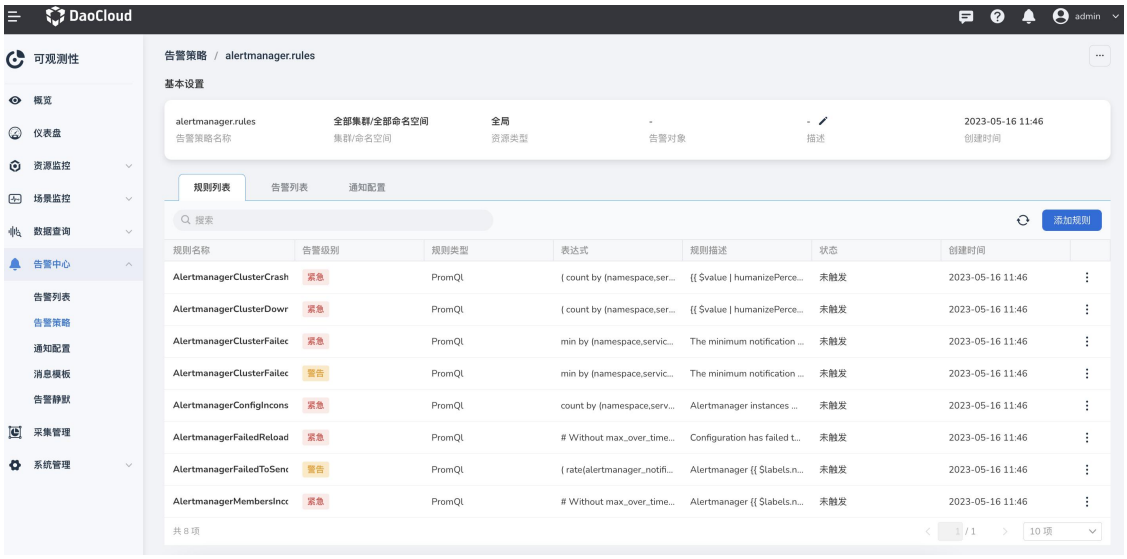
DCE 5.0 除了内置的告警策略外，还可以自定义创建新的告警策略。 每条告警策略是一组告警规则的集合，支持对集群、节点、工作负载等资源设置告警规则。当告警对象达到策略下任一规则设定的阈值，则会自动触发告警并发送通知。

以内置告警为例，点击第一条告警策略 alertmanager.rules。



告警策略

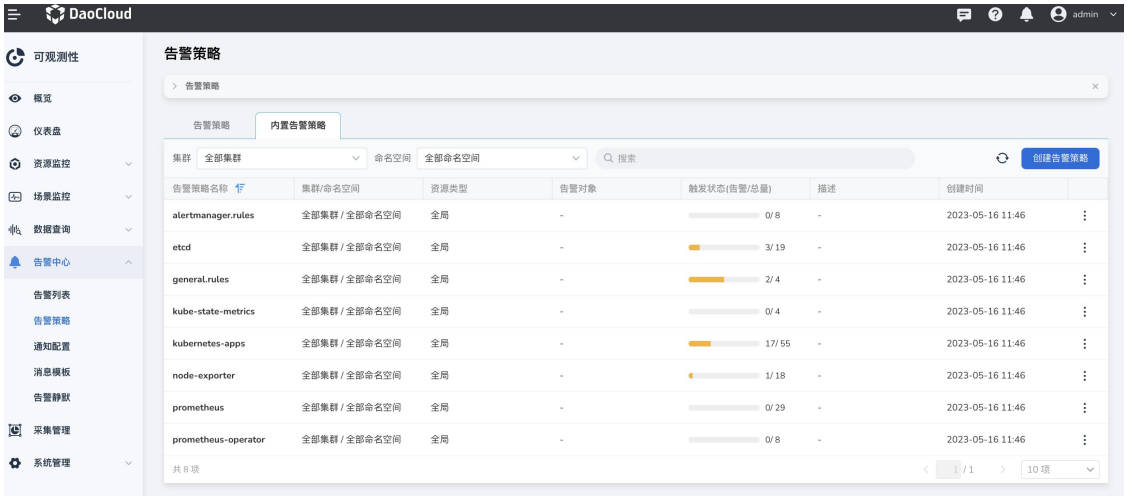
可以看到其下已设置了一些告警规则，可以在该策略下添加更多规则，也可以随时编辑或删除规则。还可以查看有关这条告警策略的历史告警和活动告警，编辑通知配置等。



alertmanager.rules

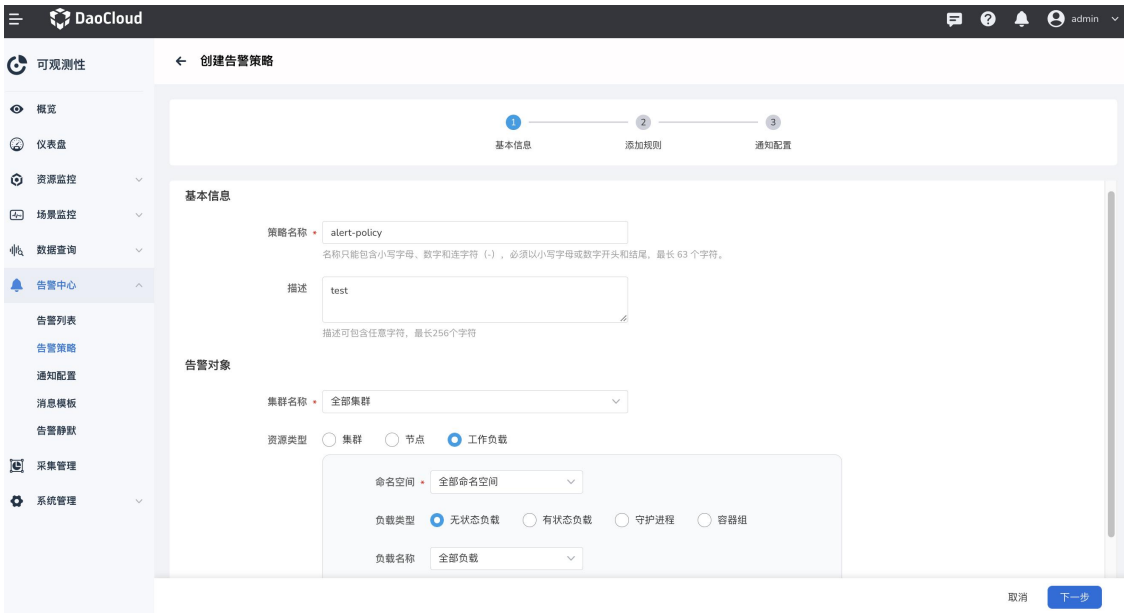
创建告警策略

1. 选择告警中心 –> 告警策略，点击创建告警策略按钮。



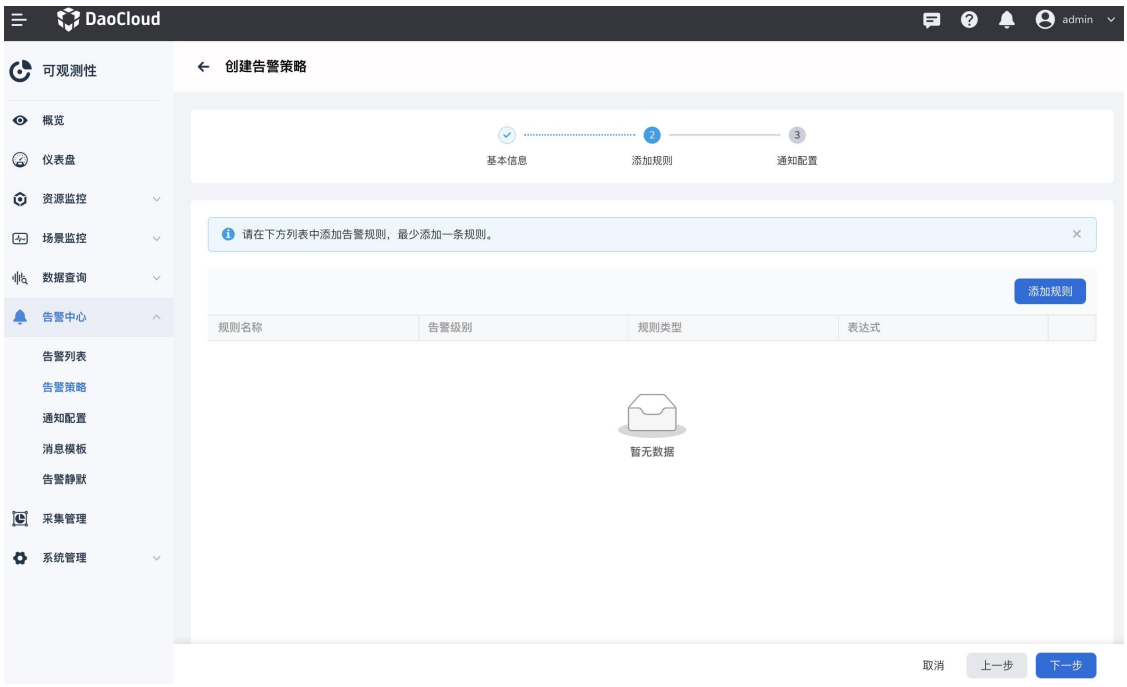
告警策略

2. 填写基本信息，选择一个或多个集群、节点或工作负载为告警对象后点击下一步。



基本信息

3. 列表需至少有一条规则。如果列表为空，请添加规则。



添加规则

在弹窗中创建告警规则，填写各项参数后点击确定。



创建规则

- 模板规则：预定义了基础指标，可以按 CPU、内存、磁盘、网络设定要监控的指标。
- PromQL 规则：输入一个 PromQL 表达式，具体请查询 Prometheus 表达式。
- 持续时长：告警被触发且持续时间达到该设定值后，告警策略将变为触发中状态。
- 告警级别：包含紧急、警告、信息三种级别。
- 高级设置：可以自定义标签和注解。

4. 点击下一步后配置通知。



通知配置

5. 配置完成后，点击确定按钮，返回告警策略列表。

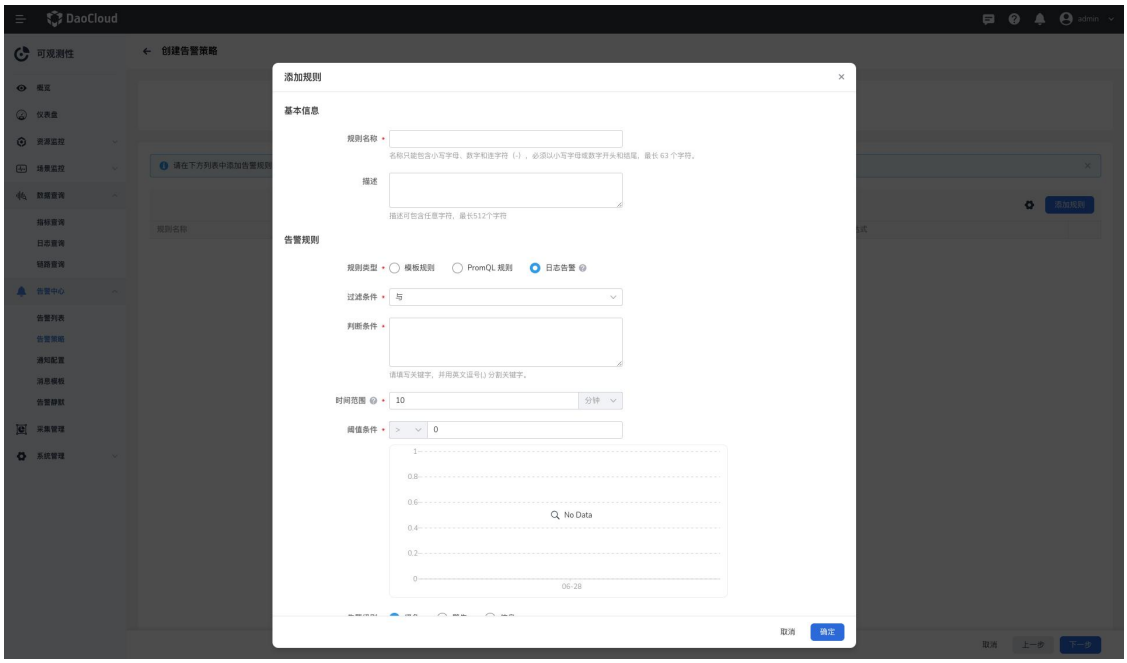
!!! tip

新建的告警策略为`未触发`状态。一旦满足规则中的阈值条件和持续时间后，将变为`触发中`状态。

创建日志规则

完成基本信息的填写后，点击添加规则，规则类型选择日志规则。

仅当资源对象选择节点或工作负载时，支持创建日志规则。



通知配置

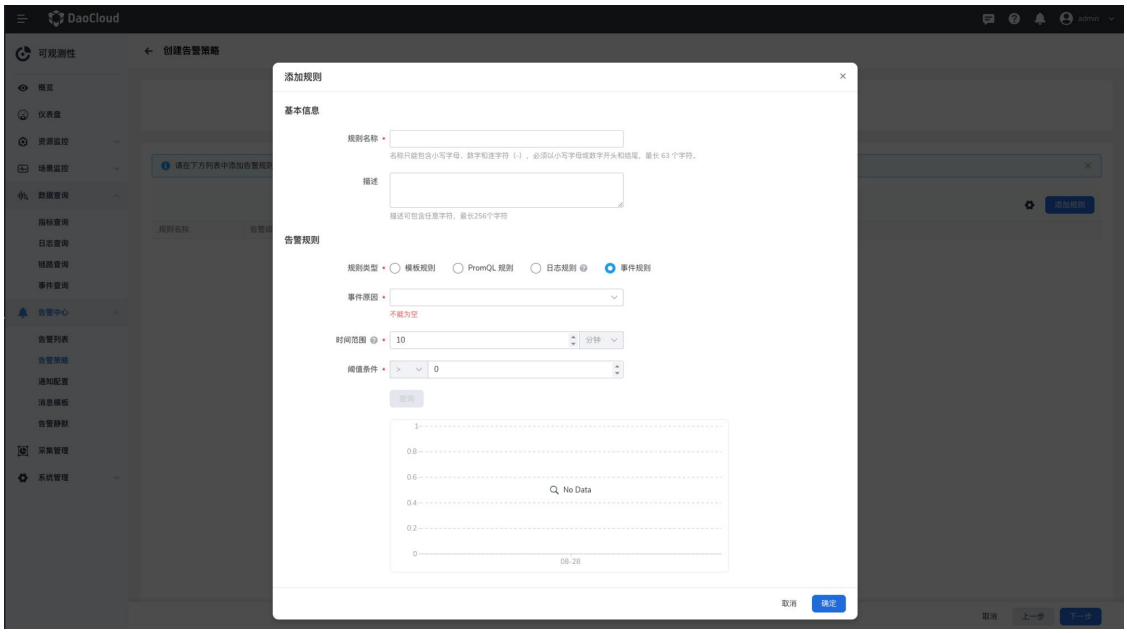
字段说明：

- 过滤条件：查询日志内容的字段，支持与、或、正则匹配、模糊匹配四种过滤条件。
- 判断条件：根据过滤条件，输入关键字或匹配条件。
- 时间范围：日志查询的时间范围。
- 阈值条件：在输入框中输入告警阈值。当达到设置的阈值时，则触发告警。支持的比较运算符有： >、≥、=、≤、< 。
- 告警级别：选择告警级别，用于表示告警的严重程度。

创建事件规则

完成基本信息的填写后，点击添加规则，规则类型选择事件规则。

仅当资源对象选择工作负载时，支持创建事件规则。



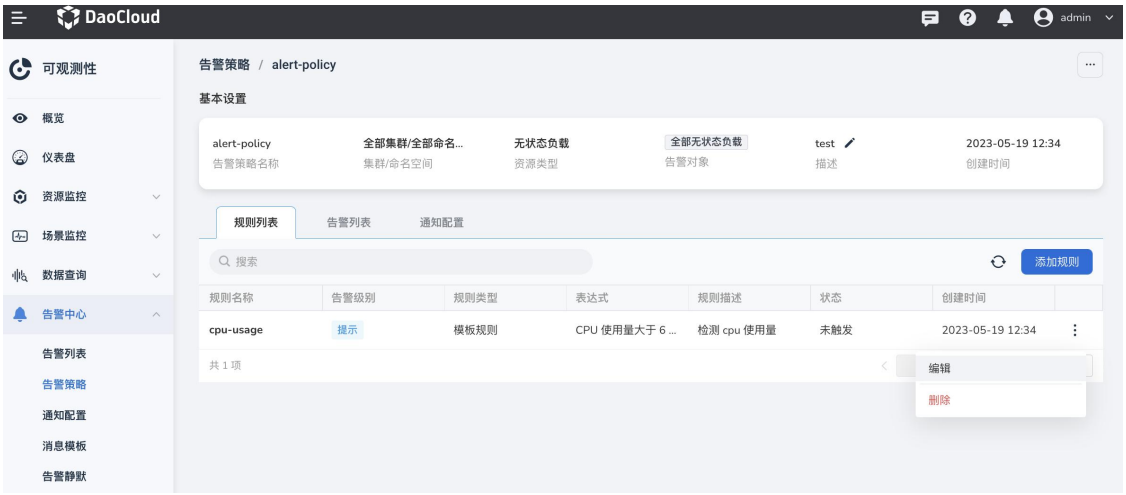
eventrule

字段说明：

- 事件规则：仅支持资源对象选择工作负载
- 事件原因：不同的工作负载类型的事件原因不同，事件原因之间是“和”的关系。
- 时间范围：检测该时间范围内产生数据，若达到设置的阈值条件，则触发告警事件。
- 阈值条件：当产生的事件达到设置的阈值时，则触发告警事件。
- 趋势图：默认查询 10 分钟内的事件变化趋势，每个点的数值统计的是当前时间点到之前的某段时间（时间范围）内发生的总次数。

其他操作

在列表右侧点击 ：，在弹出菜单中选择删除，可以删除告警策略。点击策略名称，进入策略详情，可以添加、编辑、删除其下的告警规则。



告警规则

!!! warning

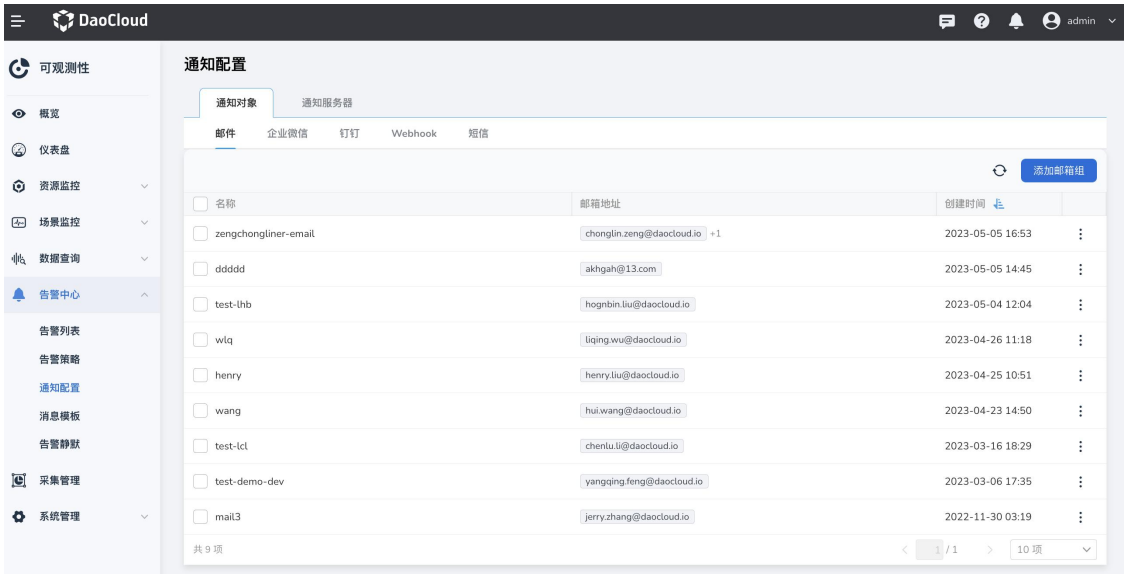
删除后的告警策略将完全消失，请谨慎操作。

通知配置

在通知配置页面，可以配置通过邮件、企业微信、钉钉、Webhook 和短信等方式向用户发送消息。

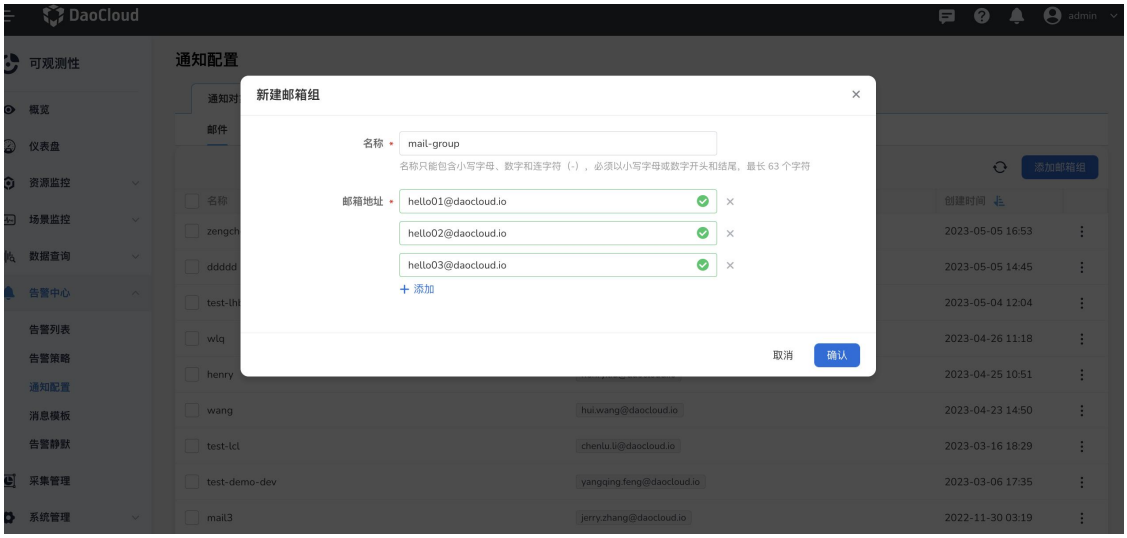
邮件组

1. 进入可观测性后，在左侧导航栏中点击告警中心 -> 通知配置，默认位于邮件通知对象，点击添加邮箱组，添加一个或多个邮件地址。



邮件

2. 支持添加多个邮箱地址。

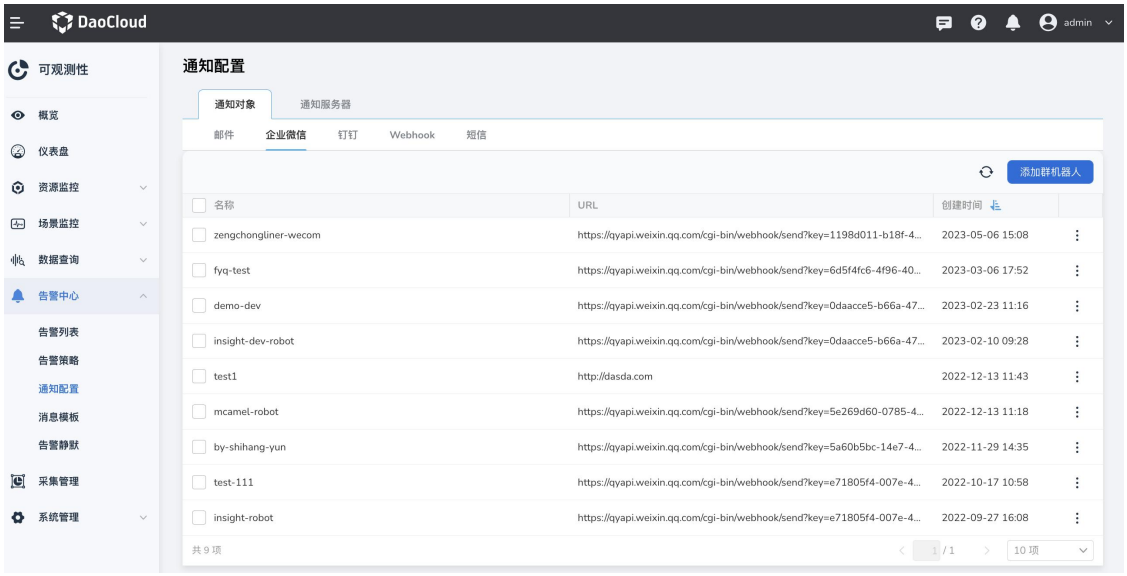


添加邮箱组

3. 配置完成后自动返回通知列表，点击列表右侧的 ：，可以编辑或删除邮箱组。

企业微信

1. 在左侧导航栏中点击告警中心 -> 通知配置 -> 企业微信，点击添加群机器人，添加一个或多个群机器人。



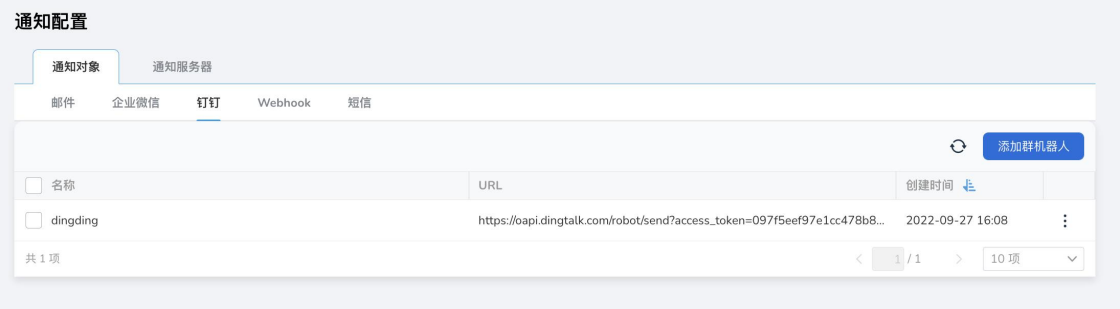
企业微信

有关企业微信群机器人的 URL，请参阅企业微信官方文档：如何使用群机器人。

2. 配置完成后自动返回通知列表，点击列表右侧的 ：，选择发送测试信息，还可以编辑或删除群机器人。

钉钉

1. 在左侧导航栏中点击告警中心 -> 通知配置 -> 钉钉，点击添加群机器人，添加一个或多个群机器人。



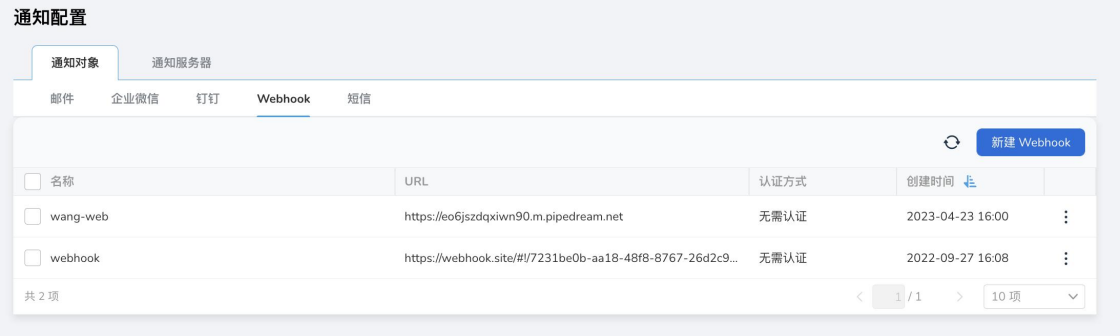
钉钉

有关钉钉群机器人的 URL，请参阅钉钉官方文档：自定义机器人接入。

2. 配置完成后自动返回通知列表，点击列表右侧的 ：，选择发送测试信息，还可以编辑或删除群机器人。

Webhook

1. 在左侧导航栏中点击告警中心 -> 通知配置 -> Webhook，点击新建 Webhook，添加一个或多个 Webhook。



webhook

有关 Webhook URL 及更多配置方式，请参阅 webhook 文档。

2. 配置完成后自动返回通知列表，点击列表右侧的 ：，选择发送测试信息，还可以编辑或删除 Webhook。

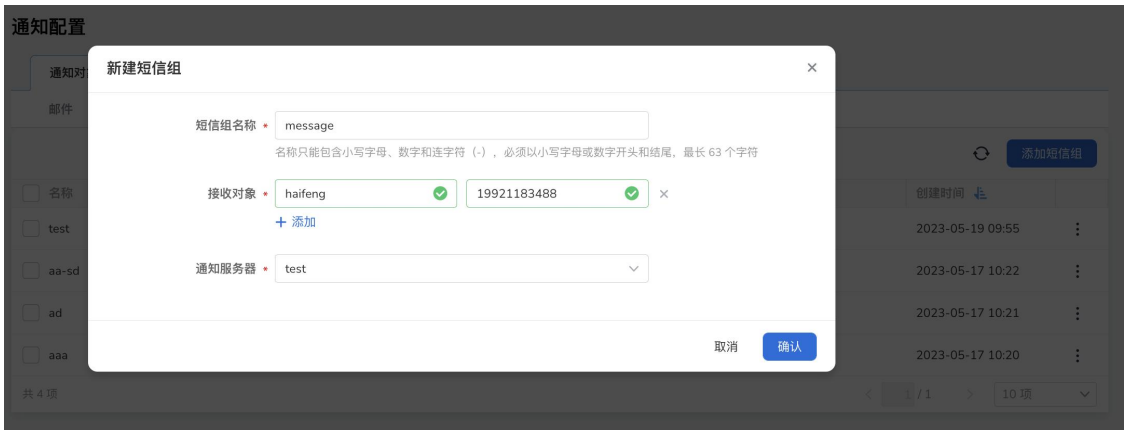
短信组

1. 在左侧导航栏中点击告警中心 -> 通知配置 -> 短信，点击添加短信组，添加一个或多个短信组。



messsage

2. 在弹窗中输入名称、接收短信的对象、手机号以及通知服务器。



mobile

通知服务器需要预先在通知配置 –> 通知服务器中添加创建。目前支持阿里云、腾讯云两种云服务器，具体配置的参数请参阅自己的云服务器信息。



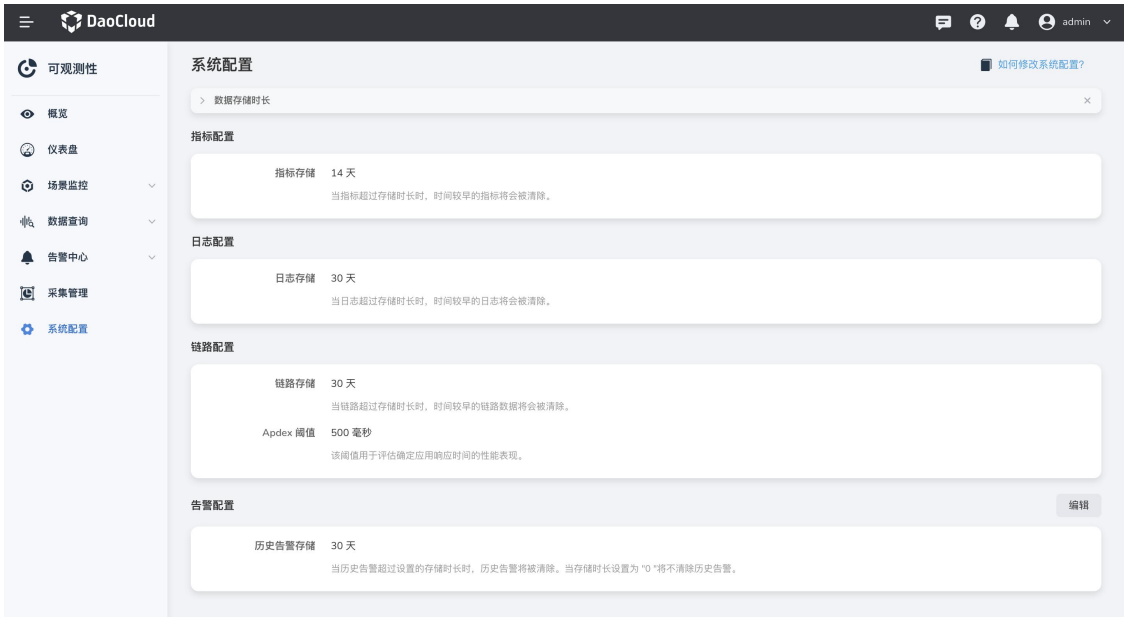
cloud–notify

3. 短信组添加成功后，自动返回通知列表，点击列表右侧的 ：，可以编辑或删除短信组。

系统配置

系统配置展示指标、日志、链路默认的保存时长以及默认的 Apdex 阈值。

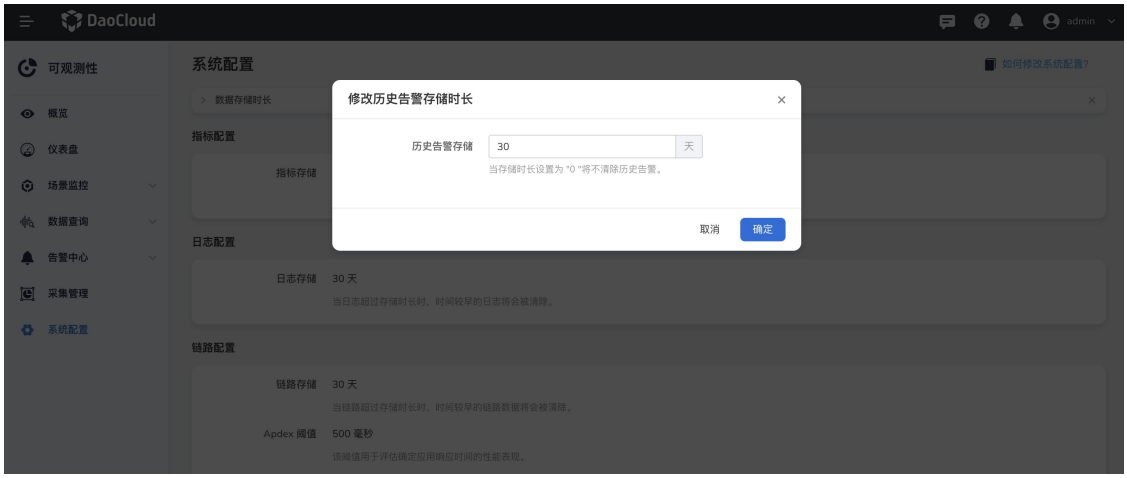
1. 点击右侧导航栏，选择系统配置。



系统配置

2. 目前仅支持修改历史告警存储时长，点击 编辑 输入目标时长。

   当存储时长设置为“0”将不清除历史告警。



系统配置

!!! note

修改其他配置，请点击查看[如何修改系统配置？](modify-config.md)

修改系统配置

可观测性会默认持久化保存指标、日志、链路的数据，您可参阅本文修改系统配置。

如何修改指标数据保留期限

先 ssh 登录到对应的节点，参考以下步骤修改指标数据保留期限。

1. 执行以下命令：

   ```
   kubectl edit vmcluster insight-victoria-metrics-k8s-stack -n insight-system
   ```

2. 在 Yaml 文件中，retentionPeriod 的默认值为 14，单位为天。您可根据需求修改参数。

   ```yaml
   apiVersion: operator.victoriametrics.com/v1beta1
   kind: VMCluster
   metadata:
     annotations:
       meta.helm.sh/release-name: insight
       meta.helm.sh/release-namespace: insight-system
     creationTimestamp: "2022-08-25T04:31:02Z"
     finalizers:
     - apps.victoriametrics.com/finalizer
     generation: 2
     labels:
       app.kubernetes.io/instance: insight
       app.kubernetes.io/managed-by: Helm
       app.kubernetes.io/name: victoria-metrics-k8s-stack
       app.kubernetes.io/version: 1.77.2
       helm.sh/chart: victoria-metrics-k8s-stack-0.9.3
   ```

```
    name: insight-victoria-metrics-k8s-stack
    namespace: insight-system
    resourceVersion: "123007381"
    uid: 55cee8d6-c651-404b-b2c9-50603b405b54
  spec:
    replicationFactor: 1
    retentionPeriod: "14"
    vminsert:
      extraArgs:
        maxLabelsPerTimeseries: "45"
      image:
        repository: docker.m.daocloud.io/victoriametrics/vminsert
        tag: v1.80.0-cluster
        replicaCount: 1
```

3. 保存修改后，负责存储指标的组件的容器组会自动重启，稍等片刻即可。

如何修改日志数据存储时长

先 ssh 登录到对应的节点，参考以下步骤修改日志数据保留期限：

方法一：修改 Json 文件

1. 修改以下文件中 rollover 字段中的 max_age 参数，并设置保留期限，默认存储时长为 7d。注意需要修改第一行中的 Elastic 用户名和密码、IP 地址和索引。

```
curl --insecure --location -u"elastic:amyVt4o826e322TUVi13Ezw6" -X PUT "https://172.30.47.112:30468/_ilm/policy/insight-es-k8s-logs-policy?pretty" -H 'Content-Type: application/json' -d'
{
    "policy": {
        "phases": {
            "hot": {
                "min_age": "0ms",
```

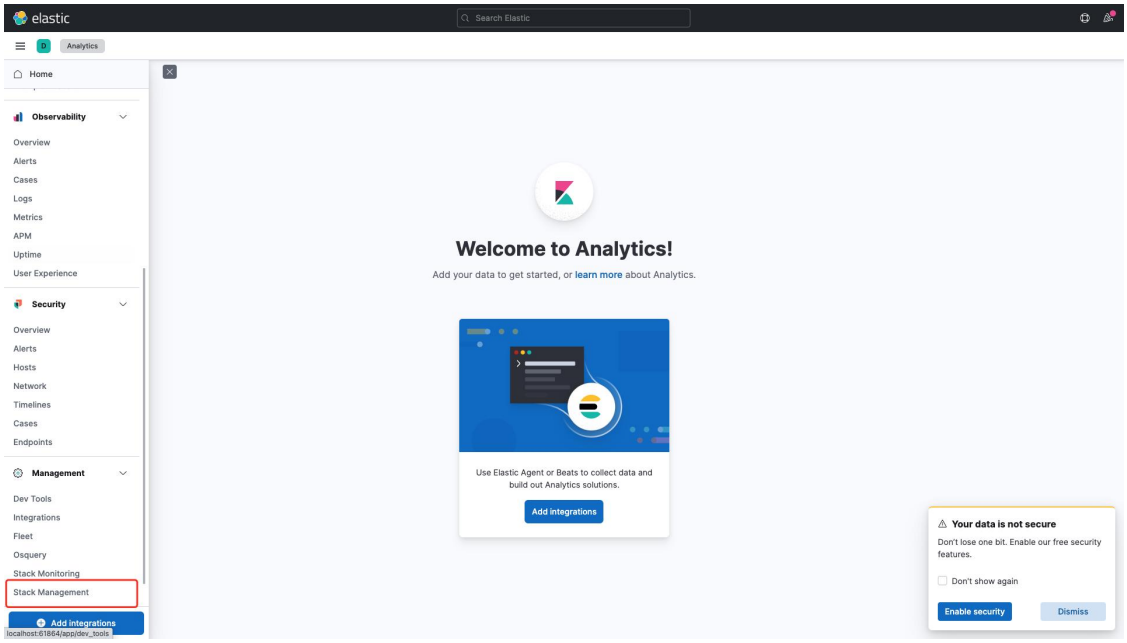
```
                "actions": {
                    "set_priority": {
                        "priority": 100
                    },
                    "rollover": {
                        "max_age": "8d",
                        "max_size": "10gb"
                    }
                }
            },
            "warm": {
                "min_age": "10d",
                "actions": {
                    "forcemerge": {
                        "max_num_segments": 1
                    }
                }
            },
            "delete": {
                "min_age": "30d",
                "actions": {
                    "delete": {}
                }
            }
        }
    }
}'
```

2. 修改完后 ，执行以上命令。它会打印出如下所示内容，则修改成功。

```
{
"acknowledged" : true
}
```

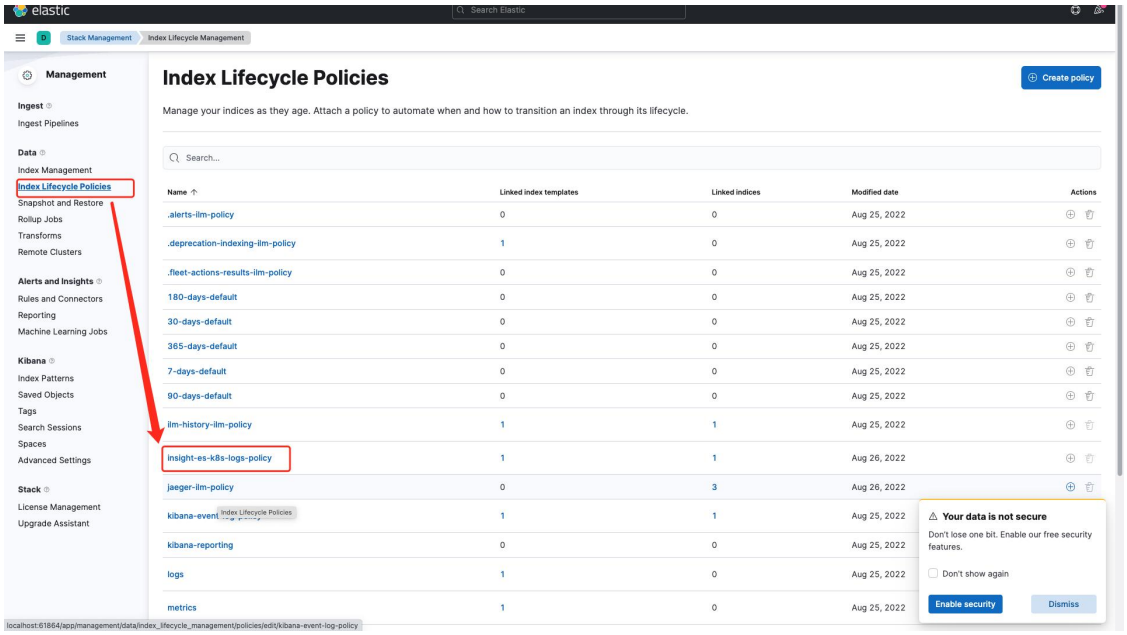
方法二：从 UI 修改

1. 登录 kibana，选择左侧导航栏 Stack Management。



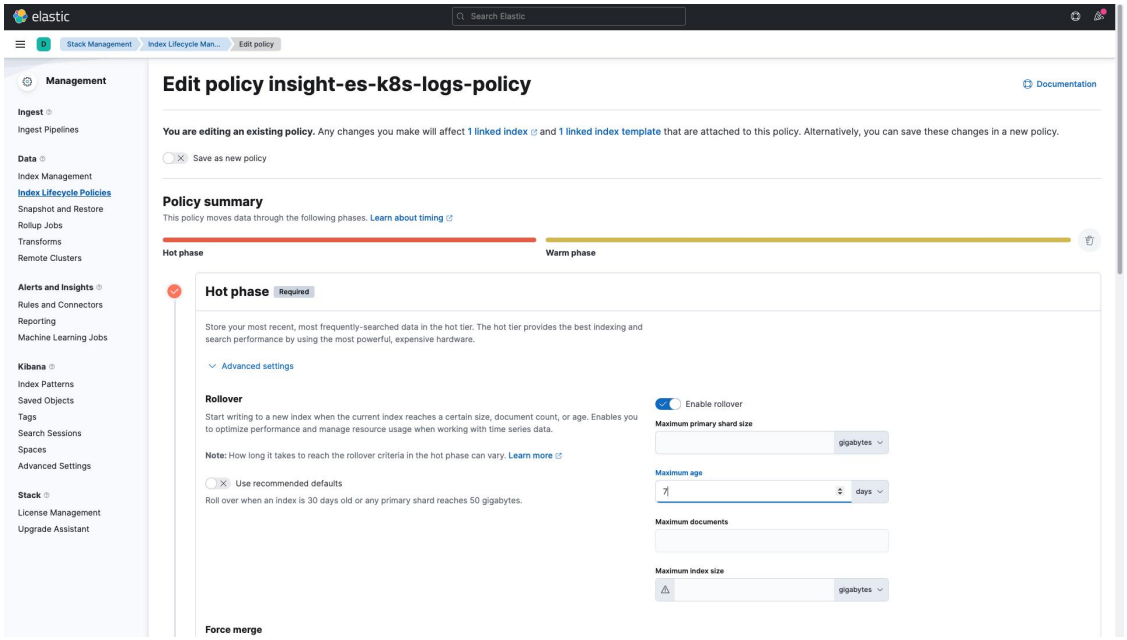
Stack Management

2. 选择左侧导航 Index Lifecycle Polices，并找到索引 insight-es-k8s-logs-policy，点击进入详情。



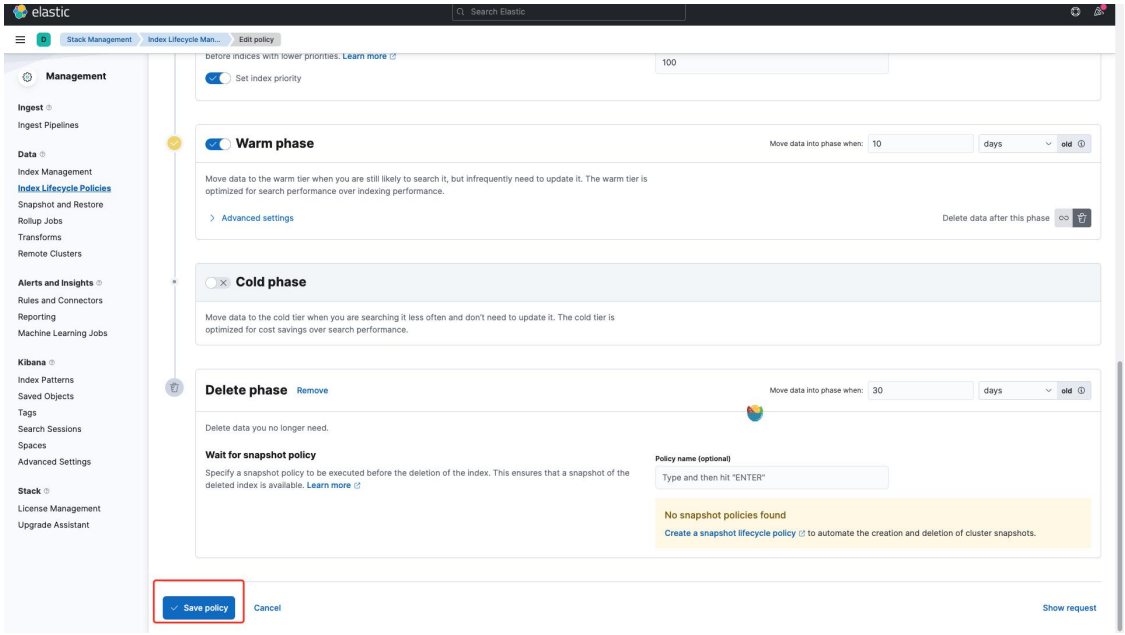
索引

3. 展开 Hot phase 配置面板，修改 Maximum age 参数，并设置保留期限，默认存储时长为 7d。



保留期限

4. 修改完后，点击页面底部的 Save policy 即修改成功。



保存

如何修改链路数据存储时长

先 ssh 登录到对应的节点，参考以下步骤修改链路数据保留期限：

方法一：修改 Json 文件

1. 修改以下文件中 rollover 字段中的 max_age 参数，并设置保留期限，默认存储时长为 7d。注意需要修改第一行中的 Elastic 用户名和密码、IP 地址和索引。

```
curl --insecure --location -u"elastic:amyVt4o826e322TUVi13Ezw6" -X PUT "https://172.30.47.112:30468/_ilm/policy/jaeger-ilm-policy?pretty" -H 'Content-Type: application/json' -d'
{
    "policy": {
        "phases": {
            "hot": {
                "min_age": "0ms",
                "actions": {
                    "set_priority": {
```

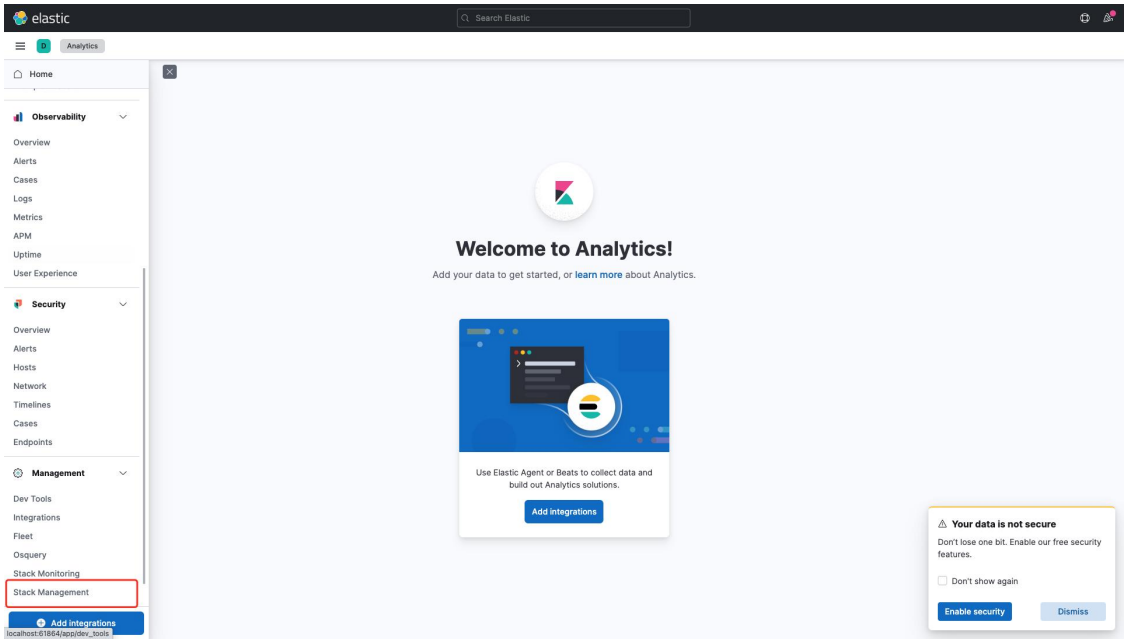
```
                "priority": 100
            },
            "rollover": {
                "max_age": "6d",
                "max_size": "10gb"
            }
        }
    },
    "warm": {
        "min_age": "10d",
        "actions": {
            "forcemerge": {
                "max_num_segments": 1
            }
        }
    },
    "delete": {
        "min_age": "30d",
        "actions": {
            "delete": {}
        }
    }
  }
 }
}'
```

2. 修改完后，在控制台执行以上命令。它会打印出如下所示内容，则修改成功。

```
{
"acknowledged" : true
}
```

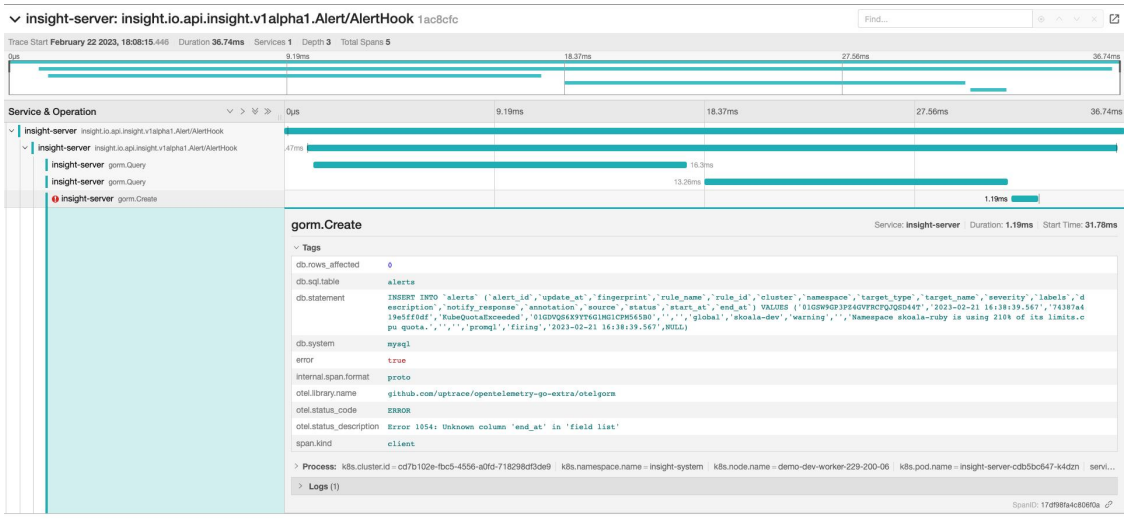
方法二：从 UI 修改

1. 登录 kibana，选择左侧导航栏 Stack Management。



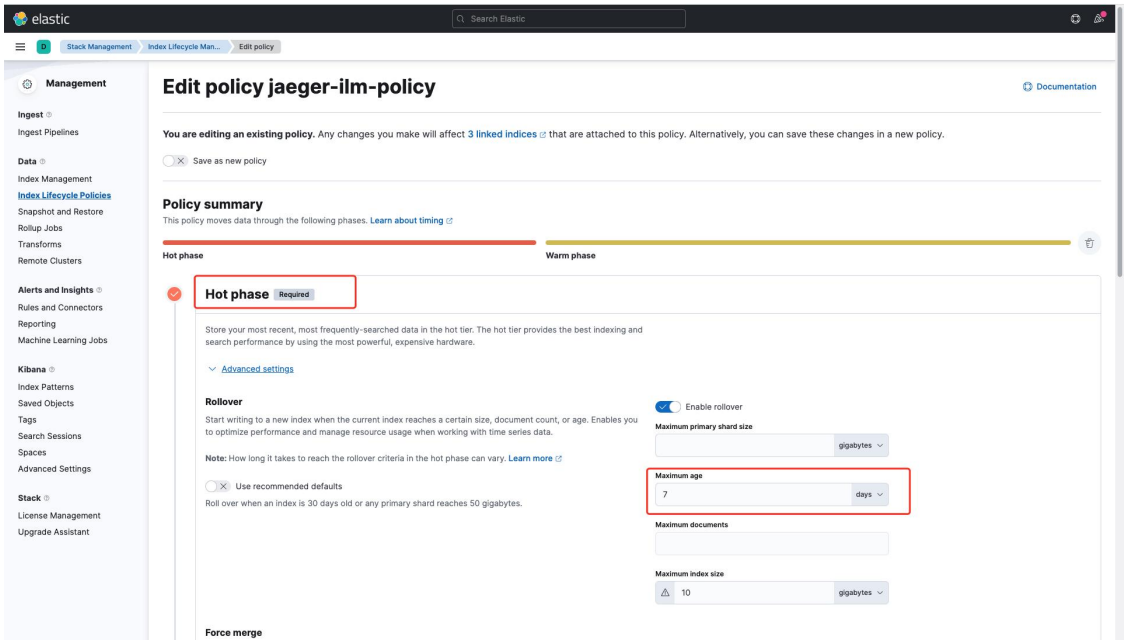
Stack Management

2. 选择左侧导航 Index Lifecycle Polices，并找到索引 jaeger-ilm-policy，点击进入详情。



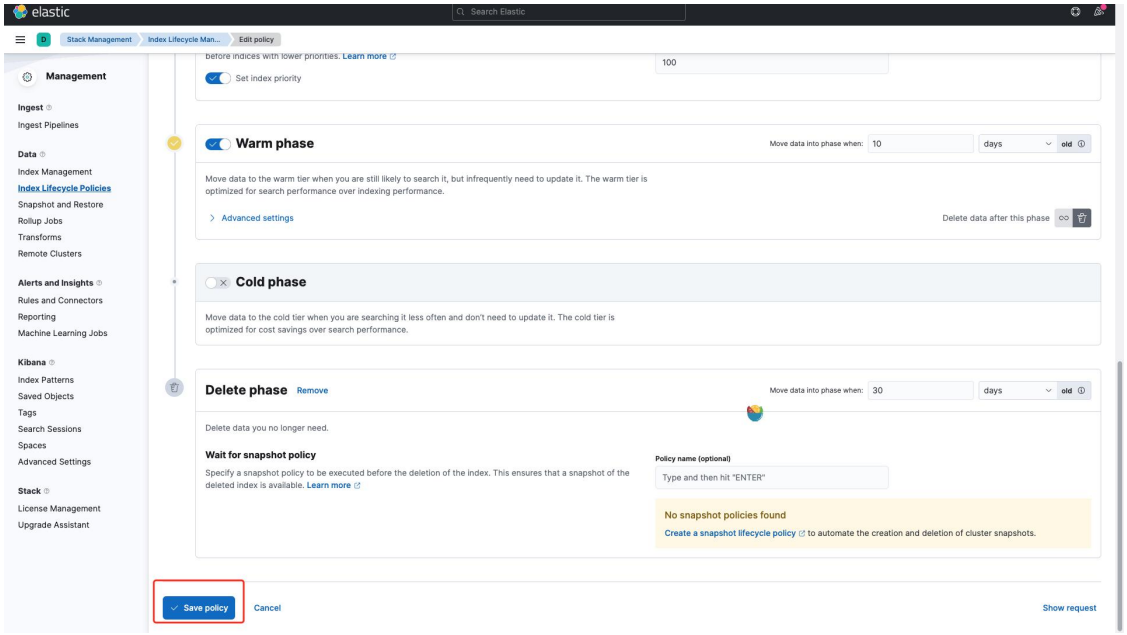
索引

3. 展开 Hot phase 配置面板，修改 Maximum age 参数，并设置保留期限，默认存储时长为 7d。



保留期限

4. 修改完后，点击页面底部的 Save policy 即修改成功。

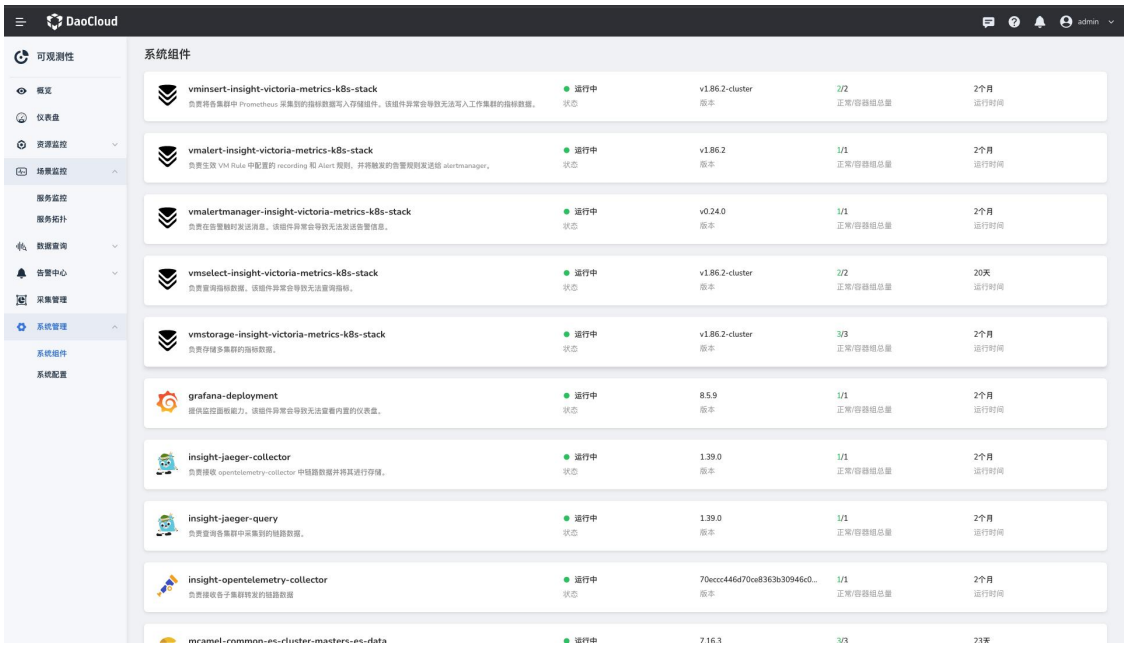


保存

系统组件

在系统组件页面可快速的查看可观测性模块中系统组件的运行状态，当系用组件发生故障时，会导致可观测模块中的部分功能不可用。

1. 进入可观测性 产品模块，
2. 在左边导航栏选择 系统管理 -> 系统组件。



系统组件

组件说明

模块	组件名称	说明
指标	vminsert-insight-victoria-metrics-k8s-	负责将各集群中 Prometheus 采集到的指标数据写入存储组件。该组件异常会导致无法写入工作集群的指标数据。

模块	组件名称	说明
	stack	
指标	vmalert-insight-victoria-metrics-k8s-stack	负责生效 VM Rule 中配置的 recording 和 Alert 规则，并将触发的告警规则发送给 alertmanager。
指标	vmalertmanager-insight-victoria-metrics-k8s-stack	负责在告警触时发送消息。该组件异常会导致无法发送告警信息。
指标	vmselect-insight-victoria-metrics-k8s-stack	负责查询指标数据。该组件异常会导致无法查询指标。
指标	vmstorage-insight-victoria-metrics-k8s-stack	负责存储多集群的指标数据。
仪表盘	grafana-deployment	提供监控面板能力。该组件异常会导致无法查看内置的仪表盘。
链路	insight-jaeger-collector	负责接收 opentelemetry-collector 中链路数据并将其进行存储。

模块	组件名称	说明
链路	insight-jaeger-query	负责查询各集群中采集到的链路数据。
链路	insight-opentelemetry-collector	负责接收各子集群转发的链路数据
日志	elasticsearch	负责存储各集群的日志数据。

链路数据中的时钟偏移

在一个分布式系统中，由于 Clock Skew（时钟偏斜调整）影响， 不同主机间存在时间漂移现象。通俗来说，不同主机在同一时刻的系统时间是有微小的偏差的。

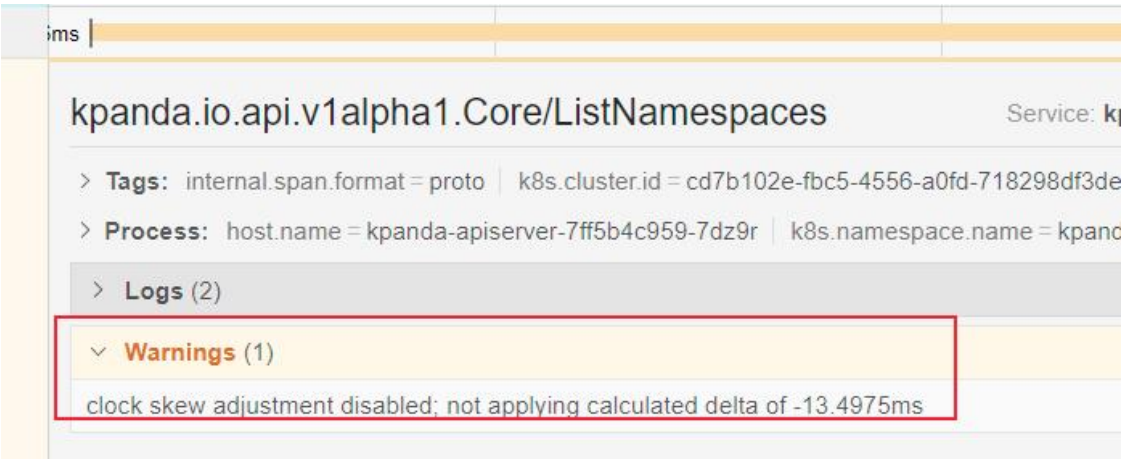
链路追踪系统是一个典型的分布式系统，它在涉及时间数据采集上也受这种现象影响，比如在一条链路中服务端 span 的开始时间早于客户端 span， 这种现象逻辑上是不存在的，但是由于时钟偏移影响，链路数据在各个服务中被采集到的那一刻主机间的系统时存在偏差，最终造成如下图所示的现象：



时钟偏移

上图中出现的现象理论上无法消除。但该现象较少，即使出现也不会影响服务间的调用关系。

目前 Insight 使用 Jaeger UI 来展示链路数据，UI 在遇到这种链路时会提醒：



时钟偏移

目前 Jaeger 的社区正在尝试通过 UI 层面来优化这个问题。

更多的相关资料，请参考：

- Clock Skew Adjuster considered harmful
- Add ability to display unadjusted trace in the UI
- Clock Skew Adjustment

Lucene 语法使用方法

Lucene 简介

Lucene 是 Apache 软件基金会 4 jakarta 项目组的一个子项目，是一个开放源代码的全文检索引擎工具包。 Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能。

Lucene 语法

Lucene 的语法搜索格式允许您以灵活的方式构建搜索查询，以满足不同的搜索需求。以下是 Lucene 的语法搜索格式的详细说明：

关键字查询

要通过 Lucene 语法实现多个关键字的查询，您可以使用布尔逻辑操作符来组合多个关键字。Lucene 支持以下几种操作符：

1. AND 操作符
    - 使用 AND 或 && 来表示逻辑与关系。
    - 例如：term1 AND term2 或 term1 && term2
2. OR 操作符
    - 使用 OR 或 || 来表示逻辑或关系。
    - 例如：term1 OR term2 或 term1 || term2
3. NOT 操作符
    - 使用 NOT 或 `` 来表示逻辑非关系。
    - 例如：term1 NOT term2 或 term1 –term2
4. 引号
    - 您可以将一个短语括在引号中以进行精确匹配。

- 例如："exact phrase"

举例

1. 指定字段

   field1:keyword1 AND (field2:keyword2 OR field3:keyword3) NOT field4:keyword4

   解释如下：

   - 查询字段 field1 必须包含关键字 keyword1。
   - 同时，字段 field2 必须包含关键字 keyword2 或字段 field3 必须包含关键字 keyword3。
   - 最后，字段 field4 不得包含关键字 keyword4。
2. 不指定字段

   keyword1 AND (keyword2 OR keyword3) NOT keyword4

   解释如下：

   - 查询关键字 keyword1 必须存在于任意可搜索的字段中。
   - 同时，关键字 keyword2 必须存在或关键字 keyword3 必须存在于任意可搜索的字段中。
   - 最后，关键字 keyword4 不得存在于任意可搜索的字段中。

模糊查询

在 Lucene 中，模糊查询可以通过波浪号 ~ 来实现近似匹配。您可以指定一个编辑距离来限制匹配的相似度程度。

term~

在上述示例中，term 是要进行模糊匹配的关键字。

请注意以下几点：

- 波浪号 ~ 后面可以指定一个可选的参数，用于控制模糊查询的相似度。

- 参数值范围为 0 到 2 之间，其中 0 表示完全匹配，1 表示一次编辑操作（如增加、删除或替换字符）内可以匹配，2 表示两次编辑操作内可以匹配。
- 如果不指定参数值，默认使用 0.5 作为相似度阈值。
- 模糊查询将返回与给定关键字相似的文档，但会有一定的性能开销，特别是对于较大的索引。

通配符

Lucene 支持以下两种通配符查询：

1. * 通配符：* 用于匹配零个或多个字符。

   例如，te*t 可以匹配“test”、“text”、“tempest”等。

2. ? 通配符：? 用于匹配单个字符。

   例如，te?t 可以匹配“test”、“text”等。

举例说明

te?t

在上述示例中，te?t 表示匹配以“te”开头，接着是一个任意字符，然后以“t”结尾的词。这种查询可以匹配例如“test”、“text”、“tent”等。

需要注意的是，问号只能代表一个字符，如果想要匹配多个字符或者是可变长度的字符，可以使用星号 * 进行多字符通配符匹配。 另外，问号不会匹配空字符串。

总结一下，Lucene 语法中的问号 ? 用作单字符通配符，表示匹配任意一个字符。通过在搜索关键字中使用问号，您可以进行更灵活和具体的模式匹配。

范围查询

Lucene 语法支持范围查询，您可以使用方括号 [ ] 或花括号 { } 来表示范围。以下是范围查询的示例：

1. 包含边界的范围查询：

- 方括号 [ ] 表示闭区间，包含边界值。
- 例如：field:[value1 TO value2] 表示 field 的取值范围从 value1 到 value2（包含两者）。

2. 排除边界的范围查询：

   - 花括号 { } 表示开区间，排除边界值。
   - 例如：field:{value1 TO value2} 表示 field 的取值范围在 value1 和 value2 之间（不包含两者）。

3. 省略边界的范围查询：

   - 可以省略一个或两个边界值来指定无限范围。
   - 例如：field:[value TO] 表示 field 的取值范围从 value 到正无穷，field:[TO value] 表示 field 的取值范围从负无穷到 value。

!!! note

请注意，范围查询只适用于能够进行排序的字段类型，如数字、日期等。同时，确保在查询时将边界值正确地指定为字段的实际值类型。

如果您希望在整个索引中进行范围查询而不指定特定字段，可以使用通配符查询 `*` 来代替字段名。

举例说明

1. 指定字段

   timestamp:[2022-01-01 TO 2022-01-31]

   这将检索 timestamp 字段在 2022 年 1 月 1 日到 2022 年 1 月 31 日之间的数据。

2. 不指定字段

   *:[value1 TO value2]

   这将在整个索引中搜索取值范围从 value1 到 value2 的文档。

告警通知流程说明

在创建告警策略时，可观测性 Insight 支持为同策略下不同级别触发的告警配置不同的通知发送间隔，但由于在 Alertmanager 原生配置中存在 group_interval 和 repeat_interval 两个参数，会导致告警通知的实际发送间隔存在偏差。

参数配置

在 Alertmanager 配置如下：

```
route:
  group_by: ["rulename"]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
```

参数说明：

- group_wait ：用于设置告警通知的等待时间。当 Alertmanager 接收到一组告警时，如果在 group_wait 指定的时间内没有更多的告警，则 Alertmanager 将等待一段时间以便收集到更多相同标签和内容的告警，并将所有符合条件的告警添加到同一通知中。
- group_interval：用于设置一组告警在被合并成单一通知前等待的时间。如果在这个时间内没有收到更多的来自同一组的告警，则 Alertmanager 将发送一个包含所有已接收告警的通知。
- repeat_interval：用于设置告警通知的重复发送间隔。当 Alertmanager 发送告警通知到接收器后，如果在 repeat_interval 参数指定的时间内持续收到相同标签和内容的告警，则 Alertmanager 将重复发送告警通知。

当同时设置了 group_wait、group_interval 和 repeat_interval 参数时，Alertmanager 将按照以下方式处理同一 group 下的告警通知：

1. 当 Alertmanager 接收到符合条件的告警时，它将等待至少 group_wait 参数中指定的时间以便收集更多相同标签和内容的告警，并将所有符合条件的告警添加到同一通知中。

2. 如果在 group_wait 时间内没有接收到更多的告警，则在该时间之后，Alertmanager 会将所收到的所有此类警报发送到接收器。如果有其他符合条件的告警在此期间内到达，则 Alertmanager 将继续等待，直到收集到所有告警或超时。

3. 如果在 group_interval 参数中指定的时间内接收到了更多的相同标签和内容的告警，则这些新告警也将被添加到先前的通知中并一起发送。如果在 group_interval 时间结束后仍然有未发送的告警，Alertmanager 将会重新开始一个新的计时周期，并等待更多的告警，直到再次达到 group_interval 时间或收到新的告警。

4. 如果在 repeat_interval 参数中指定的时间内持续收到相同标签和内容的告警，则 Alertmanager 将重复发送之前已经发送过的警报通知。在重复发送警报通知时，Alertmanager 不再等待 group_wait 或 group_interval，而是根据 repeat_interval 指定的时间间隔进行重复通知。

5. 如果在 repeat_interval 时间结束后仍有未发送的告警，则 Alertmanager 将重新开始一个新的计时周期，并继续等待相同标签和内容的新告警。这个过程将一直持续下去，直到没有新告警为止或 Alertmanager 被停止。

举例说明

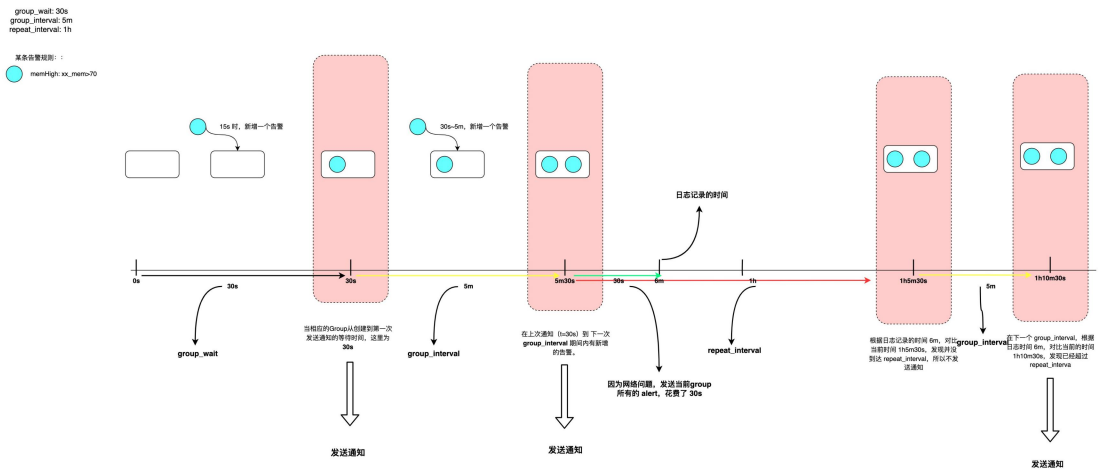
在下述示例中，Alertmanager 将所有 CPU 使用率高于阈值的告警分配到一个名为“critical_alerts”的策略中。

```
groups:
- name: critical_alerts
  rules:
  - alert: HighCPUUsage
    expr: node_cpu_seconds_total{mode="idle"} < 50
```

```
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High CPU usage detected on instance {{ $labels.instance }}"
  group_by: [rulename]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
```

在这种情况下：

- 当 Alertmanager 收到告警时，它将等待至少 30 秒以便收集更多相同标签和内容的告警，并将它们添加到同一通知中。
- 如果在 5 分钟内接收到了更多相同标签和内容的告警，则这些新告警也将被添加到先前的通知中并一起发送。如果在 15 分钟后仍有未发送的告警，则 Alertmanager 将重新开始一个新的计时周期，并等待更多的告警，直到再次达到 5 分钟或收到新告警。
- 如果在 1 小时内持续收到相同标签和内容的告警，则 Alertmanager 将重复发送之前已经发送过的警报通知。


group_wait: 30s
group_interval: 5m
repeat_interval: 1h
某条告警规则：：
memHigh: xx_mem>70
15s 时，新增一个告警
30s~5m，新增一个告警
日志记录的时间
0s
30s
30s
5m30s
6m
1h
1h5m30s
1h10m30s
group_wait
group_interval
repeat_interval
group_interval
5m
当相应的Group从创建到第一次发送通知的等待时间，这里为 30s
在上次通知（t=30s）到 下一次 group_interval 期间内有新增的告警。
因为网络问题，发送当前group 所有的 alert，花费了 30s
根据日志记录的时间 6m，对比当前时间 1h5m30s，发现并没到达 repeat_interval，所以不发送通知
在下一个 group_interval，根据日志时间 6m，对比当前的时间 1h10m30s，发现已经超过 repeat_interva
发送通知
发送通知
发送通知

安装

Kubernetes 集群兼容性测试

：测试通过；　　：测试未通过。

!!! note

表格中的测试功能非全量。

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
采集并查询 web 应用的指标	E2E										
添加自定义指标采集	E2E										
查询	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
实时指标											
瞬时指标查询	E2E										
瞬时指标 api 字段功能验证	E2E										
查询一段时间内指标	E2E										
查询一段时间内指	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
标											
api											
字段											
功能											
验证											
批量查询集群CPU、内存使用率、集群CPU总量、集群内存使用量，	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
集群节点总数											
批量查询节点CPU、内存使用率、节点CPU总量、节点内存使用量	E2E										
批量查询	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
一段时间内的集群CPU、内存使用率、集群CPU总量、集群内存使用量，集群节点总数											
批量	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
查询查询一段时间内指标											
api 字段功能验证											
查询 Pod 日志	E2E										
查询 SVC 日志	E2E										
查询 stateful set	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
日志查询 Deployment 日志	E2E										
查询 NPD 日志	E2E										
日志筛选	E2E										
日志模糊查询–workloadSearch	E2E										
日志	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
模糊查询–podSearch											
日志模糊查询–containerSearch	E2E										
日志精确查询–cluster	E2E										

case	测试方式	K8s 1.26	k8s 1.23.0 ~ 1.23.13	k8s 1.24.0 ~ 1.24.7	k8s 1.25.0 ~ 1.25.3	k8s 1.22	k8s 1.21	k8s 1.20	k8s 1.19	k8s 1.18	备注
日志精确查询–namespace	E2E										
日志查询api字段功能验证	E2E										

Openshift 4.x 集群兼容性测试

：测试通过；　　：测试未通过。

!!! note

表格中的测试功能非全量。

case	测试方式	ocp4.10(k8s 1.23.0)	备注
采集并查询 web 应用的指标	手工		
添加自定义指标采集	手工		
查询实时指标	手工		
瞬时指标查询	手工		
瞬时指标 api 字段功能验证	手工		
查询一段时间内指标	手工		
查询一段时间内指标 api 字段功能验证	手工		
批量查询集群 CPU、内存使用率、集群 CPU 总量、集群内存使用量，集群节点总数	手工		

case	测试方式	ocp4.10(k8s 1.23.0)	备注
批量查询节点 CPU、内存使用率、节点 CPU 总量、节点内存使用量	手工		
批量查询一段时间内的集群 CPU、内存使用率、集群 CPU 总量、集群内存使用量，集群节点总数	手工		
批量查询查询一段时间内指标 api 字段功能验证	手工		
查询 Pod 日志	手工		
查询 SVC 日志	手工		
查询 statefulset 日志	手工		
查询 Deployment 日志	手工		
查询 NPD 日志	手工		
日志筛选	手工		
日志模糊查询–workloadSearch	手工		
日志模糊查询–	手工		

case	测试方式	ocp4.10(k8s 1.23.0)	备注
podSearch			
日志模糊查询–containerSearch	手工		
日志精确查询–cluster	手工		
日志精确查询–namespace	手工		
日志查询 api 字段功能验证	手工		
告警规则–增删改查	手工		
告警模板–增删改查	手工		
通知方式–增删改查	手工		
链路查询	手工		
拓扑查询	手工		

Rancher 集群兼容性测试

：测试通过；　　：测试未通过。

!!! note

表格中的测试功能非全量。

case	测试方式	Rancher rke2c1(k8s 1.24.11)	备注
采集并查询 web 应用的指标	手工		
添加自定义指标采集	手工		
查询实时指标	手工		
瞬时指标查询	手工		
瞬时指标 api 字段功能验证	手工		
查询一段时间内指标	手工		
查询一段时间内指标	手工		

case	测试方式	Rancher rke2c1(k8s 1.24.11)	备注
api 字段功能验证			
批量查询集群 CPU、内存使用率、集群 CPU 总量、集群内存使用量，集群节点总数	手工		
批量查询节点 CPU、内存使用率、节点 CPU 总量、节点内存使用量	手工		
批量查询一段时间内的集群 CPU、内存使用率、集群 CPU 总量、集群内存使用量，集群节点总	手工		

case	测试方式	Rancher rke2c1(k8s 1.24.11)	备注
数			
批量查询查询一段时间内指标 api 字段功能验证	手工		
查询 Pod 日志	手工		
查询 SVC 日志	手工		
查询 statefulset 日志	手工		
查询 Deployment 日志	手工		
查询 NPD 日志	手工		
日志筛选	手工		
日志模糊查询–workloadSearch	手工		

case	测试方式	Rancher rke2c1(k8s 1.24.11)	备注
日志模糊查询–podSearch	手工		
日志模糊查询–containerSearch	手工		
日志精确查询–cluster	手工		
日志精确查询–namespace	手工		
日志查询 api 字段功能验证	手工		
告警规则–增删改查	手工		
告警模板–增删改查	手工		
通知方式–增删改查	手工		
链路查询	手工		
拓扑查询	手工		

离线升级可观测性模块

本页说明从下载中心下载可观测性模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `insight` 字样是可观测性模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

    !!! note

    该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

    ===“已安装 chart repo”

    若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

    ```yaml title="load-image.yaml"
    source:
      intermediateBundlesPath: insight-offline # (1)
    target:
      containerRegistry: 10.16.10.111 # (2)
      containerRepository: release.daocloud.io/insight # (3)
      repo:
        kind: HARBOR # (4)
    ```

```
    url: http://10.16.10.111/chartrepo/release.daocloud.io # (5)
    auth:
    username: "admin" # (6)
    password: "Harbor12345" # (7)
  containers:
    auth:
      username: "admin" # (8)
      password: "Harbor12345" # (9)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

“未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: insight-offline # (1)
target:
  containerRegistry: 10.16.10.111 # (2)
  containerRepository: release.daocloud.io/insight # (3)
  repo:
```

```
    kind: LOCAL
    path: ./local-repo  # (4)
  containers:
    auth:
      username: "admin"  # (5)
      password: "Harbor12345"  # (6)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar xvf insight.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```shell
docker load -i images.tar
```

=== “containerd”

```shell
ctr image import images.tar
```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

=== “通过 helm repo 升级”

1. 检查全局管理 helm 仓库是否存在。

    ```shell
    helm repo list | grep insight
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加全局管理的 helm 仓库。

```shell
heml repo add insight http://{harbor url}/chartrepo/{project}
```

1. 更新全局管理的 helm 仓库。

```shell
helm repo update insight # (1)
```

1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

1. 选择您想安装的全局管理版本（建议安装最新版本）。

```shell
helm search repo insight/insight --versions
helm search repo insight/insight-agent --versions
```

```none
[root@master ~]# helm search repo insight/insight --versions
NAME                    CHART VERSION   APP VERSION   DESCRIPTION
insight/insight         0.13.1          v0.13.1       A Helm chart for Insight
insight/insight-agent   0.13.1          v0.13.1       A Helm chart for Insight Agent
...
```

1. 备份 `--set` 参数。

在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values insight -n insight-system -o yaml > insight.yaml
helm get values insight-agent -n insight-system -o yaml > insight-agent.yaml
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 insight.yaml 和 insight-agent.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade insight insight/insight \
  -n insight-system \
  -f ./insight.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.13.1
```

以及

```shell
helm upgrade insight-agent insight/insight-agent \
  -n insight-system \
  -f ./insight-agent.yaml \
```

```
  --set global.imageRegistry=$imageRegistry \
  --version 0.13.1
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

   在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values insight -n insight-system -o yaml > insight.yaml
   ```

1. 执行 `helm upgrade`。

   升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

   ```shell
   export imageRegistry={你的镜像仓库}
   ```

   ```shell
   helm upgrade insight . \
     -n insight-system \
     -f ./insight.yaml \
     --set global.imageRegistry=$imageRegistry
   ```

   以及

```shell
helm upgrade insight-agent . \
  -n insight-system \
  -f ./insight-agent.yaml \
  --set global.imageRegistry=$imageRegistry
```

开始观测

DCE 5.0 平台实现了对多云多集群的纳管，并支持创建集群。在此基础上，可观测性 Insight 作为多集群统一观测方案，通过部署 insight-agent 插件实现对多集群观测数据的采集，并支持通过 DCE 5.0 可观测性产品实现对指标、日志、链路数据的查询。

insight-agent 是可观测性实现对多集群数据采集的工具，安装后无需任何修改，即可实现对指标、日志以及链路数据的自动化采集。

通过 容器管理 创建的集群默认会安装 insight-agent，故在此仅针对接入的集群如何开启观测能力提供指导。

- 在线安装 insight-agent

可观测性 Insight 作为多集群的统一观测平台，其部分组件的资源消耗与创建集群的数据、接入集群的数量息息相关，在安装 insight-agent 时，需要根据集群规模对相应组件的资源进行调整。

1. 根据创建集群的规模或接入集群的规模，调整 insight-agent 中采集组件 Prometheus 的 CPU 和内存，请参考: Prometheus 资源规划
2. 由于多集群的指标数据会统一存储，则需要 DCE 5.0 平台管理员根据创建集群的规模、接入集群的规模对应调整 vmstorage 的磁盘，请参考：vmstorage 磁盘容量规划。

- 如何调整 vmstorage 的磁盘，请参考：vmstorge 磁盘扩容。

由于 DCE 5.0 支持对多云多集群的纳管，insight-agent 目前也完成了部分验证，由于监控组件冲突问题导致在 DCE 4.0 集群和 Openshift 4.x 集群中安装 insight-agent 会出现问题，若您遇到同样问题，请参考以下文档：

- 在 DCE 4.0.x 安装 insight-agent

- 在 Openshift 4.x 安装 insight-agent

目前，采集组件 insight-agent 已对当前主流的 Kubernetes 版本完成了部分的功能测试，请参考：

- kubernetes 集群兼容性测试
- Openshift 4.x 集群兼容性测试
- Rancher 集群兼容性测试

2022-11-17

在线安装 insight-agent

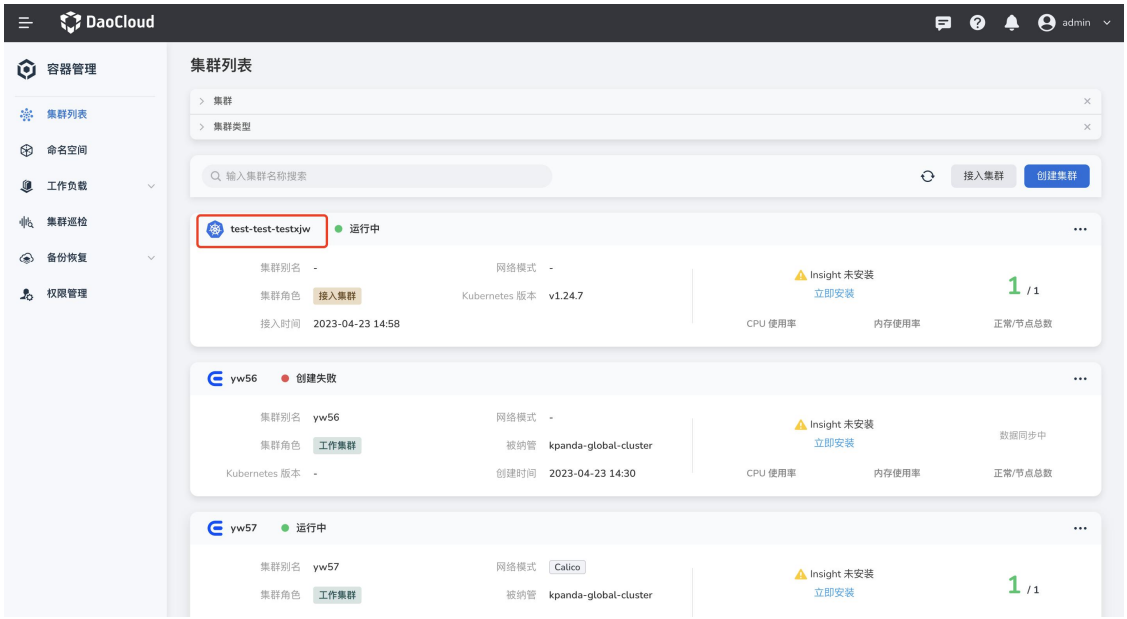
insight-agent 是集群观测数据采集的插件，支持对指标、链路、日志数据的统一观测。本文描述了如何在在线环境中为接入集群安装 insight-agent。

前提条件

- 集群已成功接入 `容器管理 模块。如何接入集群，请参考：

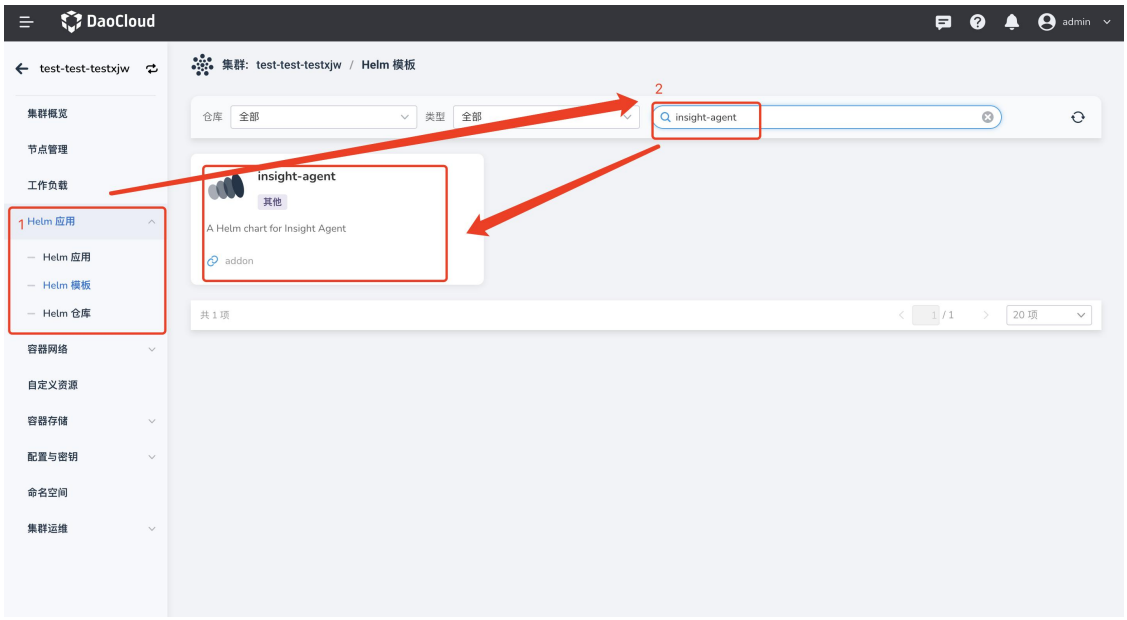
操作步骤

1. 进入容器管理模块，在容器列表中点击要安装 insight-agent 的集群名称。



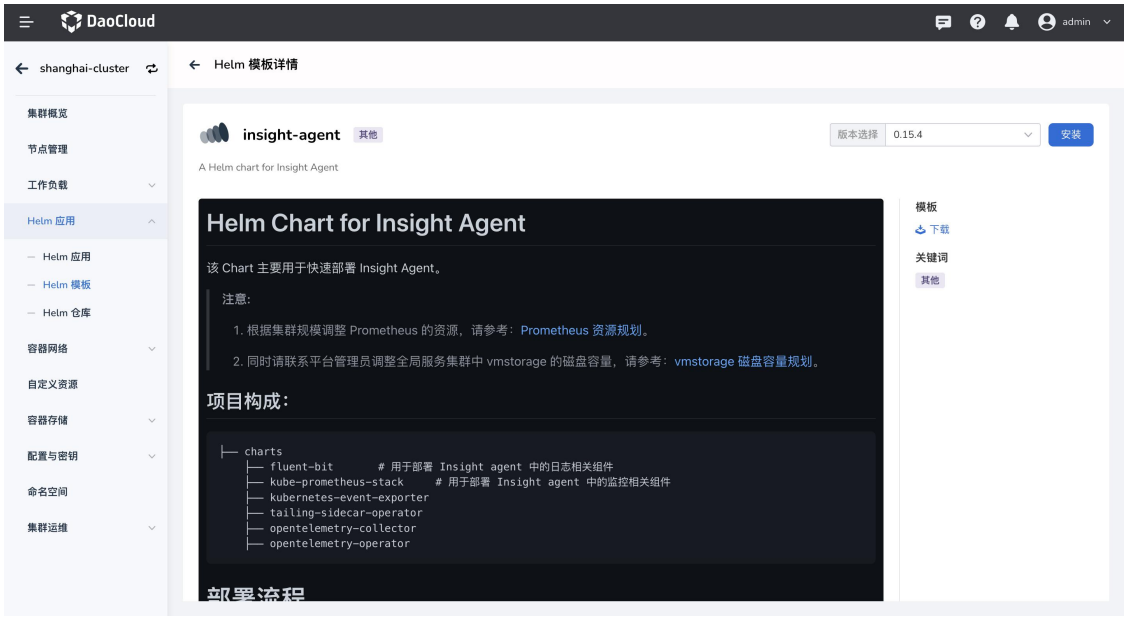
安装

2. 选择左侧导航栏 Helm 应用 -> Helm 模板，在搜索框查询 insight-agent ，点击该卡片进入详情。



安装

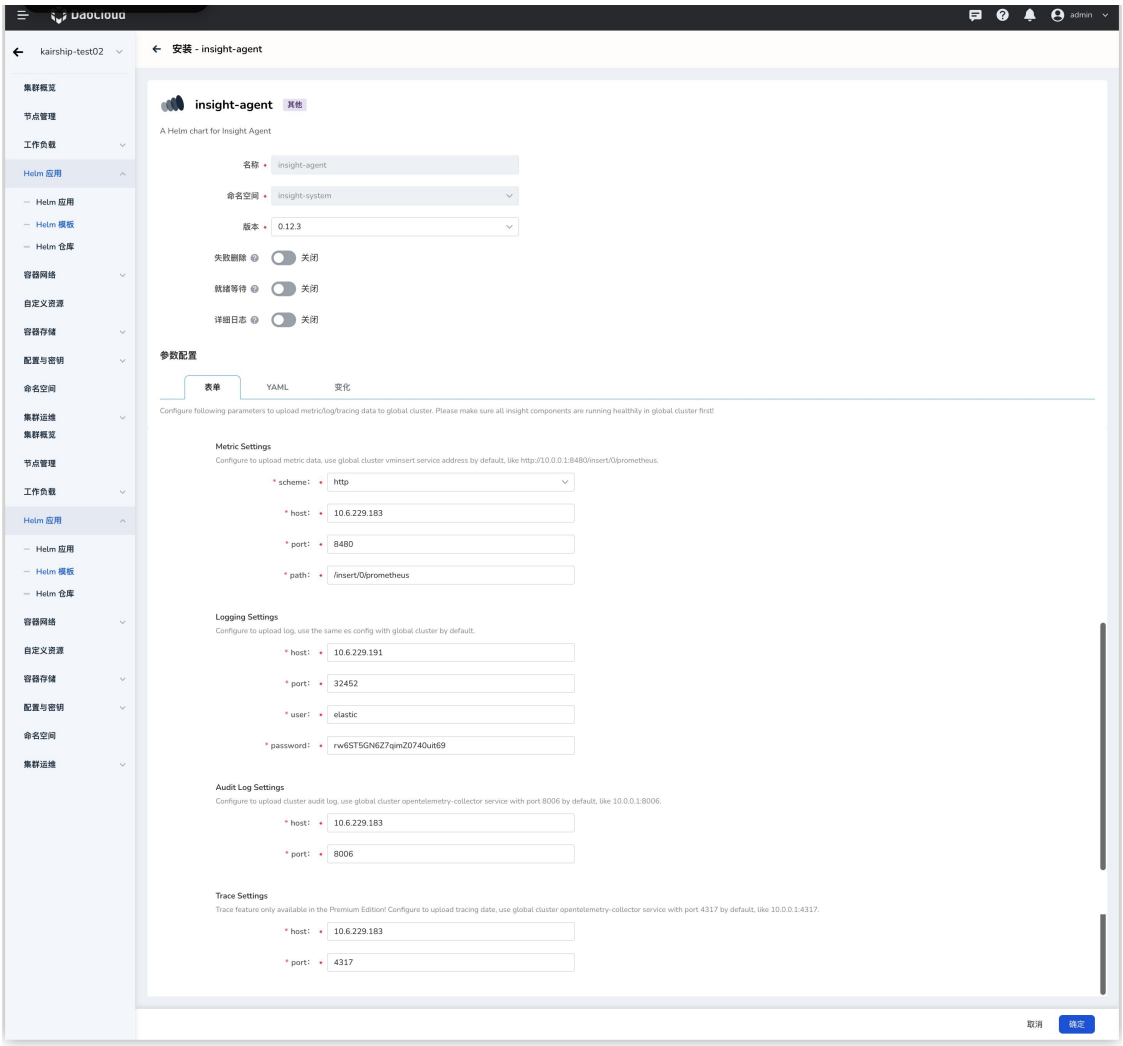
3. 查看 insight-agent 的 Readme，点击安装 进入下一步。



安装

4. 选择安装的版本并在下方表单分别填写全局管理集群中对应数据存储组件的地址，确认填写的信息无误后，点击确定 。

- insight-agent 默认部署在集群的 insight-system 命名空间下。
- 建议安装最新版本的 insight-agent。
- 系统默认已填写数据上报的组件的地址，仍请您检查无误后再点击确定 进行安装。 如需修改数据上报地址，请参考：获取数据上报地址。



安装

5. 系统将自动返回 Helm 应用 列表，当应用 insight-agent 的状态从 未就绪 变为 已部署 ，且所有的组件状态为 运行中 时，则安装成功。等待一段时间后，可在可观测性 模块查看该集群的数据。


DaoCloud
admin
zxw-dev-cluster
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
集群: zxw-dev-cluster / 命名空间: 全部命名空间 / Helm 应用
应用名称
状态
命名空间
Chart
仓库
更新时间
insight-agent
已部署
insight-system
insight-agent:0.10.0
system
2022-10-14 10:04
istiod
已部署
istio-system
istiod:1.15.0
-
2022-09-23 17:09
istio-base
已部署
istio-system
base:1.15.0
-
2022-09-23 17:08
共 3 项
1 / 1
10 项

安装

获取全局服务集群的数据存储地址

可观测性是多集群统一观测的产品，为实现对多集群观测数据的统一存储、查询，子集群需要将采集的观测数据上报给全局管理集群进行统一存储。本文提供了在安装采集组件 insight-agent 时必填的存储组件的地址。

在全局服务集群安装 insight-agent

如果在全局服务集群集群安装 insight-agent，推荐通过域名来访问集群：

```
export vminsert_host="vminsert-insight-victoria-metrics-k8s-stack.insight-system.svc.cluster.local" # 指标
export es_host="insight-es-master.insight-system.svc.cluster.local" # 日志
export otel_col_host="insight-opentelemetry-collector.insight-system.svc.cluster.local" # 链路
```

在其他集群安装 insight-agent

通过 Insight Server 提供的接口获取地址

1. 管理集群使用默认的 LoadBalancer 方式暴露

登录全局管理集群的控制台，执行以下命令：

!!! note

请替换命令中的 `${INSIGHT_SERVER_IP}` 参数。

```
export INSIGHT_SERVER_IP=$(kubectl get service insight-server -n insight-system --output=jsonpath={.spec.clusterIP})
curl --location --request POST 'http://'"${INSIGHT_SERVER_IP}"'/apis/insight.io/v1alpha1/agentinstallparam'
```

获得如下返回值：

{"values":"{\"global\":{\"exporters\":{\"logging\":{\"host\":\"10.6.182.32\"},\"metric\":{\"host\":\"10.6.182.32\"},\"auditLog\":{\"host\":\"10.6.182.32\"},\"trace\":{\"host\":\"10.6.182.32\"}}},\"opentelemetry-operator\":{\"enabled\":true},\"opentelemetry-collector\":{\"enabled\":true}}"}

参数说明：

- global.exporters.logging.host 是日志服务地址【不需要再设置对应服务的端口，都会使用相应默认值】；
- global.exporters.metric.host 是指标服务地址；
- global.exporters.trace.host 是链路服务地址；
- global.exporters.auditLog.host 是审计日志服务地址(和链路使用的同一个服务不同端口)。

2. 管理集群禁用 LoadBalancer

调用接口时需要额外传递集群中任意外部可访问的节点 IP，会使用该 IP 拼接出对应服务的完整访问地址。

```
export INSIGHT_SERVER_IP=$(kubectl get service insight-server -n insight-system --output=jsonpath={.spec.clusterIP})
curl --location --request POST 'http://'"${INSIGHT_SERVER_IP}"'/apis/insight.io/v1alpha1/agentinstallparam' --data '{"extra": {"EXPORTER_EXTERNAL_IP": "10.5.14.51"}}'
```

将获得如下的返回值：

{"values":"{\"global\":{\"exporters\":{\"logging\":{\"scheme\":\"https\",\"host\":\"10.5.14.51\",\"port\":32007,\"user\":\"elastic\",\"password\":\"j8V1oVoM1184HvQ1F3C8Pom2\"},\"metric\":{\"host\":\"10.5.14.51\",\"port\":30683},\"auditLog\":{\"host\":\"10.5.14.51\",\"port\":30884},\"trace\":{\"host\":\"10.5.14.51\",\"port\":30274}}},\"opentelemetry-operator\":{\"enabled\":true},\"opentelemetry-collector\":{\"enabled\":true}}"}

其中：

- global.exporters.logging.host 是日志服务地址；
- global.exporters.logging.port 是日志服务暴露的 nodePort；
- global.exporters.metric.host 是指标服务地址；
- global.exporters.metric.port 是指标服务暴露的 nodePort；
- global.exporters.trace.host 是链路服务地址；
- global.exporters.trace.port 是链路服务暴露的 nodePort；
- global.exporters.auditLog.host 是审计日志服务地址(和链路使用的同一个服务不同端口)；
- global.exporters.auditLog.host 是审计日志服务暴露的 nodePort；

通过 LoadBalancer 连接

上述的通过 Insight Server 提供的接口获取地址是通过查询集群的 LoadBalancer 以获取连接地址。除此之外，你也可以手动执行命令 kubectl get service -n insight-system | grep lb 获得相应服务的地址信息。其中：

```
[root@localhost ~]# kubectl get service -n insight-system | grep lb
lb-insight-es-master                                    LoadBalancer
   10.233.35.17     <pending>      9200:31529/TCP
                                   24d
lb-insight-opentelemetry-collector                      LoadBalancer
 10.233.23.12     <pending>      4317:31286/TCP,8006:31351/TCP
                                  24d
lb-vminsert-insight-victoria-metrics-k8s-stack          LoadBalancer
  10.233.63.67     <pending>      8480:31629/TCP
                                  24d
```

- lb-insight-es-master 是日志服务的地址；

- lb-vminsert-insight-victoria-metrics-k8s-stack 是指标服务的地址；
- lb-insight-opentelemetry-collector 是链路服务的地址；

通过 NodePort 连接

1. 全局服务集群启用 LB 特性【默认启用】

通过手动执行命令 kubectl get service -n insight-system | grep lb 获得相应服务的 NodePort 端口信息，参考通过 LoadBalancer 连接;

2.全局服务集群禁用 LB 特性

在该情况下，默认不会创建上述的 LoadBalancer 资源，对应服务名为

- vminsert-insight-victoria-metrics-k8s-stack（指标服务）；
- insight-es-master（日志服务）；
- insight-opentelemetry-collector（链路服务）。

上面两种情况获取到对应服务的对应端口信息后，进行如下设置：

```
--set global.exporters.logging.host=              #外部可访问的管理集群 NodeIP
--set global.exporters.logging.port=              #是日志服务 9200 端口对应的 Node
Port
--set global.exporters.metric.host=               #外部可访问的管理集群 NodeIP
--set global.exporters.metric.port=               #是指标服务 8480 端口对应的 Node
Port
--set global.exporters.trace.host=                #外部可访问的管理集群 NodeIP
--set global.exporters.trace.port=                #是链路服务 4317 端口对应的 NodeP
ort
--set global.exporters.auditLog.host=             #外部可访问的管理集群 NodeIP
--set global.exporters.auditLog.port=             #是链路服务 8006 端口对应的 Node
Port
```

升级注意事项

insight server

从 v0.17.x（或更低版本）升级到 v0.18.x

由于 0.18.x 中更新了 Jaeger 相关部署文件，因此需要在升级 insight server 前手动执行如下命令：

```
kubectl -n insight-system delete deployment insight-jaeger-collector
kubectl -n insight-system delete deployment insight-jaeger-query
```

由于 0.18.x 中指标名产生了变动，因此，需要在升级 Insight Server 之后，Insight Agent 也应该做升级。

此外，调整了开启链路模块的参数，以及 ElasticSearch 连接调整。具体参考以下参数：

```
+   --set global.tracing.enable=true \
-   --set jaeger.collector.enabled=true \
-   --set jaeger.query.enabled=true \
+   --set global.elasticsearch.scheme=${your-external-elasticsearch-scheme} \
+   --set global.elasticsearch.host=${your-external-elasticsearch-host} \
+   --set global.elasticsearch.port=${your-external-elasticsearch-port} \
+   --set global.elasticsearch.user=${your-external-elasticsearch-username} \
+   --set global.elasticsearch.password=${your-external-elasticsearch-password}
\
-   --set jaeger.storage.elasticsearch.scheme=${your-external-elasticsearch-sche
me} \
-   --set jaeger.storage.elasticsearch.host=${your-external-elasticsearch-host} \
-   --set jaeger.storage.elasticsearch.port=${your-external-elasticsearch-port} \
-   --set jaeger.storage.elasticsearch.user=${your-external-elasticsearch-usernam
e} \
```

- --set jaeger.storage.elasticsearch.password=${your-external-elasticsearch-password} \

从 v0.15.x（或更低版本）升级到 v0.16.x

由于 0.16.x 中使用了 vmalertmanagers CRD 的新特性参数 disableRouteContinueEnforce，因此需要在升级 insight server 前手动执行如下命令。

kubectl apply --server-side -f https://raw.githubusercontent.com/VictoriaMetrics/operator/v0.33.0/config/crd/bases/operator.victoriametrics.com_vmalertmanagers.yaml --force-conflicts

!!! note

如您是离线安装，可以在解压 insight 离线包后，请执行以下命令更新 CRD。

```shell
kubectl apply --server-side -f insight/dependency-crds --force-conflicts
```

insight-agent

从 v0.17.x（或更低版本）升级到 v0.18.x

由于 0.18.x 中更新了 Jaeger 相关部署文件，因此需要在升级 insight agent 前需要注意参数的改动。

```
+ --set global.exporters.trace.enable=true \
- --set opentelemetry-collector.enabled=true \
- --set opentelemetry-operator.enabled=true \
```

从 v0.16.x（或更低版本）升级到 v0.17.x

在 v0.17.x 版本中将 kube-prometheus-stack chart 版本从 41.9.1 升级至 45.28.1，其中使用的 CRD 也存在一些字段的升级，如 servicemonitor 的 attachMetadata 字段，因此需要在升级 insight agent 前执行如下命令：

kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.65.1/example/prometheus-operator-crd/monitoring.coreos.com_servicemonitors.yaml --force-conflicts

如您是离线安装，可以在解压 insight-agent 离线包后，在 insight-agent/dependency-crds 中找到上述 CRD 的 yaml。

从 v0.11.x（或更低版本）升级到 v0.12.x

在 v0.12.x 将 kube-prometheus-stack chart 从 39.6.0 升级到 41.9.1，其中包括 prometheus-operator 升级到 v0.60.1, prometheus-node-exporter chart 升级到 4.3.0 等。 prometheus-node-exporter 升级后使用了 Kubernetes 推荐 label，因此需要在升级前删除 node-exporter 的 daemonset。prometheus-operator 更新了 CRD，因此需要在升级 insight agent 前执行如下命令：

shell linenums="1" kubectl delete daemonset insight-agent-prometheus-node-exporter -n insight-system kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_alertmanagerconfigs.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_alertmanagers.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_podmonitors.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_probes.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_prometheuses.yaml --force-conflicts kubectl apply --

server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_prometheusrules.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_servicemonitors.yaml --force-conflicts kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/v0.60.1/example/prometheus-operator-crd/monitoring.coreos.com_thanosrulers.yaml --force-conflicts

!!! note

如您是离线安装，可以在解压 insight-agent 离线包后，执行以下命令更新 CRD。

```shell
kubectl apply --server-side -f insight-agent/dependency-crds --force-conflicts
```

Prometheus 资源规划

Prometheus 在实际使用过程中，受到集群容器数量以及开启 Istio 的影响，会导致 Prometheus 的 CPU、内存等资源使用量超出设定的资源。

为了保证不同规模集群下 Prometheus 的正常运行，需要根据集群的实际规模对 Prometheus 进行资源调整。

参考资源规划

在未开启网格情况下，测试情况统计出系统 Job 指标量与 Pod 的关系为：**Series 数量 = 800*Pod 数量**

在开启服务网格时，开启功能后 Pod 产生的 Istio 相关指标数量级为：**Series 数量 = 768*Pod 数量**

当未开启服务网格时

以下资源规划为 **未开启服务网格** 场景下，Prometheus 的资源规划推荐：

集群规模(Pod 数)	指标量(未开启服务网格)	CPU(core)	内存(GB)
100	8w	Request: 0.5Limit：1	Request：2GBLimit：4GB
200	16w	Request：1Limit：1.5	Request：3GBLimit：6GB
300	24w	Request：1Limit：2	Request：3GBLimit：6GB
400	32w	Request：1Limit：2	Request：4GBLimit：8GB
500	40w	Request：	Request：5GBLimit：

集群规模(Pod 数)	指标量(未开启服务网格)	CPU(core)	内存(GB)
		1.5Limit：3	10GB
800	64w	Request：2Limit：4	Request：8GBLimit：16GB
1000	80w	Request：2.5Limit：5	Request：9GBLimit：18GB
2000	160w	Request：3.5Limit：7	Request：20GBLimit：40GB
3000	240w	Request：4Limit：8	Request：33GBLimit：66GB

当开启服务网格功能时

以下资源规划为 **开启服务网格** 场景下，Prometheus 的资源规划推荐：

集群规模(Pod 数)	指标量(已开启服务网格)	CPU(core)	内存(GB)
100	15w	Request: 1Limit：2	Request：3GBLimit：6GB
200	31w	Request：2Limit：3	Request：5GBLimit：10GB
300	46w	Request：2Limit：4	Request：6GBLimit：12GB
400	62w	Request：2Limit：4	Request：8GBLimit：16GB
500	78w	Request：3Limit：6	Request：10GBLimit：20GB
800	125w	Request：	Request：15GBLimit：

集群规模(Pod 数)	指标量(已开启服务网格)	CPU(core)	内存(GB)
		4Limit：8	30GB
1000	156w	Request：5Limit：10	Request：18GBLimit：36GB
2000	312w	Request：7Limit：14	Request：40GBLimit：80GB
3000	468w	Request：8Limit：16	Request：65GBLimit：130GB

!!! note

1. 表格中的 `Pod 数量`指集群中基本稳定运行的 Pod 数量，如出现大量的 Pod 重启，则会造成短时间内指标量的陡增，此时资源需要进行相应上调。
2. Prometheus 内存中默认保存两小时数据，且集群中开启了 [Remote Write 功能](https://prometheus.io/docs/practices/remote_write/#memory-usage)时，会占用一定内存，资源超配比建议配置为 2。
3. 表格中数据为推荐值，适用于通用情况。如环境有精确的资源要求，建议在集群运行一段时间后，查看对应 Prometheus 的资源占用量进行精确配置。

vmstorage 磁盘容量规划

vmstorage 是负责存储可观测性多集群指标。 为保证 vmstorage 的稳定性，需要根据集群数量及集群规模调整 vmstorage 的磁盘容量。 更多资料请参考：vmstorage 保留期与磁盘空间。

测试结果

经过 14 天对不同规模的集群的 vmstorage 的磁盘观测， 我们发现 vmstorage 的磁盘用量与其存储的指标量和单个数据点占用磁盘正相关。

1. 瞬时存储的指标量 increase(vm_rows{ type != "indexdb"}[30s]) 以获取 30s 内增加的指标量
2. 单个数据点（datapoint）的占用磁盘：
   sum(vm_data_size_bytes{type!="indexdb"})
   / sum(vm_rows{type != "indexdb"})

计算方法

磁盘用量 = 瞬时指标量 x 2 x 单个数据点的占用磁盘 x 60 x 24 x 存储时间（天）

参数说明：

1. 磁盘用量单位为 Byte。
2. 存储时长(天) x 60 x 24 将时间(天)换算成分钟以便计算磁盘用量。
3. Insight Agent 中 Prometheus 默认采集时间为 30s ，故在 1 分钟内产生两倍的指标量。
4. vmstorage 中默认存储时长为 1 个月，修改配置请参考修改系统配置。

!!! warning

该公式为通用方案，建议在计算结果上预留冗余磁盘容量以保证 vmstorage 的正常运行。

参考容量

表格中数据是根据默认存储时间为一个月（30 天），单个数据点（datapoint）的占用磁盘取 0.9 计算所得结果。 多集群场景下，Pod 数量表示多集群 Pod 数量的总和。

当未开启服务网格时

集群规模（Pod 数）	指标量	磁盘容量
100	8w	6 GiB
200	16w	12 GiB
300	24w	18 GiB
400	32w	24 GiB
500	40w	30 GiB
800	64w	48 GiB
1000	80w	60 GiB
2000	160w	120 GiB
3000	240w	180 GiB

当开启服务网格时

集群规模（Pod 数）	指标量	磁盘容量
100	15w	12 GiB
200	31w	24 GiB
300	46w	36 GiB
400	62w	48 GiB
500	78w	60 GiB
800	125w	94 GiB
1000	156w	120 GiB
2000	312w	235 GiB

集群规模（Pod 数）	指标量	磁盘容量
3000	468w	350 GiB

举例说明

DCE 5.0 平台中有两个集群，其中全局管理集群(开启服务网格)中运行 500 个 Pod，工作集群(未开启服务网格)运行了 1000 个 Pod，预期指标存 30 天。

- 全局管理集群中指标量为 800x500 + 768x500 = 784000
- 工作集群指标量为 800x1000 = 800000

则当前 vmstorage 磁盘用量应设置为（784000+80000)x2x0.9x60x24x31 = 124384896000 byte = 116 GiB

!!! note

集群中指标量与 Pod 数量的关系可参考[Prometheus 资源规划](./prometheus-res.md)。

DCE 4.0 中安装 Insight Agent

在 DCE 5.0 中，可以将以前的 DCE 4.0 可作为子集群接入。 但由于大多数 DCE 4.0 集群已安装 dx-insight 作为监控系统，若此时安装 insight-agent 会与集群中已有的 prometheus operator 冲突，导致无法顺利安装。

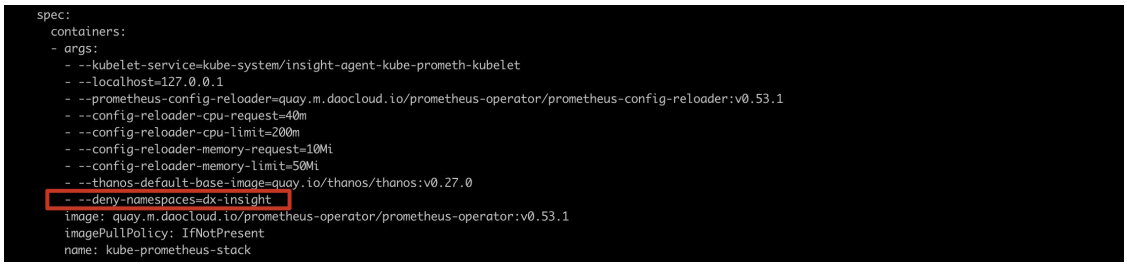
方案

解决方案是：开启 prometheus operator 的参数，保留 dx-insight 中的 prometheus operator，并兼容 5.0 中 insight-agent 的 prometheus operator。

具体操作步骤如下：

1. 登录到控制台。
2. 分别在两个 prometheus operator 中启用 --deny-namespaces 参数。
3. 执行以下命令（以下命令仅供参考，实际需替换命令中的 prometheus operator 名称和命名空间）。

   kubectl edit deploy insight-agent-kube-prometh-operator -n insight-system



   operatoryaml

!!! note

– 如上图 dx-insight 组件部署在 dx-insight 租户下，insight-agent 部署在 insight-system 租户下。

在 dx-insight 中的 prometheus operator 中添加 `--deny-namespaces=insight-system`，

在 insight-agent 中的 prometheus operator 中添加 `--deny-namespaces=dx-insight`。

– 只添加 deny namespace，两个 prometheus operator 都可以继续扫描除此之外的命名空间，而 kube-system 或客户业务命名空间下的相关采集资源不受影响。

– 请注意 node exporter 端口冲突的问题。

补充说明

开源 node-exporter 默认开启 hostnetwork 且默认端口为 9100。 若集群已有的监控系统已安装 node-exporter，则在安装 insight-agent 时会因为 node-exporter 端口冲突而无法正常运行。

!!! note

Insight 的 `node exporter` 会开启一些特性来采集特殊指标，所以推荐安装。

目前不支持在安装命令中修改端口，需要在 helm install insight-agent 之后手动修改 insight node-exporter daemonset 和 svc 的相关端口。

OpenShift 安装 Insight Agent

虽然 OpenShift 系统自带了一套监控系统，因为数据采集约定的一些规则，我们还是会安装 Insight Agent。

其中，安除了基础的安装配置之外，helm install 的时候还需要增加如下的参数：

```
## 针对 fluentbit 相关的参数；
--set fluent-bit.ocp.enabled=true \
--set fluent-bit.serviceAccount.create=false \
--set fluent-bit.securityContext.runAsUser=0 \
--set fluent-bit.securityContext.seLinuxOptions.type=spc_t \
--set fluent-bit.securityContext.readOnlyRootFilesystem=false \
--set fluent-bit.securityContext.allowPrivilegeEscalation=false \

## 启用适配 OpenShift4.x 的 Prometheus(CR)
--set compatibility.openshift.prometheus.enabled=true \

## 关闭高版本的 Prometheus 实例
--set kube-prometheus-stack.prometheus.enabled=false \
--set kube-prometheus-stack.kubeApiServer.enabled=false \
--set kube-prometheus-stack.kubelet.enabled=false \
--set kube-prometheus-stack.kubeControllerManager.enabled=false \
--set kube-prometheus-stack.coreDns.enabled=false \
--set kube-prometheus-stack.kubeDns.enabled=false \
--set kube-prometheus-stack.kubeEtcd.enabled=false \
--set kube-prometheus-stack.kubeEtcd.enabled=false \
--set kube-prometheus-stack.kubeScheduler.enabled=false \
--set kube-prometheus-stack.kubeStateMetrics.enabled=false \
--set kube-prometheus-stack.nodeExporter.enabled=false \
```

```
## 限制 PrometheusOperator 处理的 namespace，避免与 OpenShift 自带的 PrometheusOperator 相互竞争
--set kube-prometheus-stack.prometheusOperator.kubeletService.namespace="insight-system" \
--set kube-prometheus-stack.prometheusOperator.prometheusInstanceNamespaces="insight-system" \
--set kube-prometheus-stack.prometheusOperator.denyNamespaces[0]="openshift-monitoring" \
--set kube-prometheus-stack.prometheusOperator.denyNamespaces[1]="openshift-user-workload-monitoring" \
--set kube-prometheus-stack.prometheusOperator.denyNamespaces[2]="openshift-customer-monitoring" \
--set kube-prometheus-stack.prometheusOperator.denyNamespaces[3]="openshift-route-monitor-operator" \
```

通过 OpenShift 自身机制，将系统监控数据写入 Prometheus 中

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - queueConfig:
            batchSendDeadline: 60s
            maxBackoff: 5s
            minBackoff: 30ms
            minShards: 1
            capacity: 5000
            maxSamplesPerSend: 1000
            maxShards: 100
```

```
    remoteTimeout: 30s
    url: http://insight-agent-prometheus.insight-system.svc.cluster.local:90
90/api/v1/write
    writeRelabelConfigs:
    - action: keep
      regex: etcd|kubelet|node-exporter|apiserver|kube-state-metrics
      sourceLabels:
      - job
```

使用 OTel SDK 增强 Go 应用程序

本文包含如何在 Go 应用程序中设置 OpenTelemetry 增强的说明。

OpenTelemetry 也简称为 OTel，是一个开源的可观测性框架，可以帮助在 Go 应用程序中生成和收集遥测数据：链路、指标和日志。

使用 OTel SDK 增强 Go 应用

安装相关依赖

必须先安装与 OpenTelemetry exporter 和 SDK 相关的依赖项。如果您正在使用其他请求路由器，请参考请求路由。 切换/进入到应用程序源文件夹后运行以下命令：

```
go get go.opentelemetry.io/otel@v1.8.0 \
  go.opentelemetry.io/otel/trace@v1.8.0 \
  go.opentelemetry.io/otel/sdk@v1.8.0 \
  go.opentelemetry.io/contrib/instrumentation/github.com/gin-gonic/gin/otelgin@v0.33.0 \
  go.opentelemetry.io/otel/exporters/otlp/otlptrace@v1.7.0 \
  go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc@v1.4.1
```

使用 OTel SDK 创建初始化函数

为了让应用程序能够发送数据，需要一个函数来初始化 OpenTelemetry。在 main.go 文件中添加以下代码片段:

```
import（
    "context"
    "os"
    "time"

    "go.opentelemetry.io/otel"
```

```
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/propagation"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
    semconv "go.opentelemetry.io/otel/semconv/v1.7.0"
    "go.uber.org/zap"
    "google.golang.org/grpc"
)

var tracerExp *otlptrace.Exporter

func retryInitTracer() func() {
    var shutdown func()
    go func() {
        for {
            // otel will reconnected and re-send spans when otel col recover.
so, we don't need to re-init tracer exporter.
            if tracerExp == nil {
                shutdown = initTracer()
            } else {
                break
            }
            time.Sleep(time.Minute * 5)
        }
    }()
    return shutdown
}

func initTracer() func() {
    // temporarily set timeout to 10s
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
```

```
    defer cancel()

    serviceName, ok := os.LookupEnv("OTEL_SERVICE_NAME")
    if !ok {
        serviceName = "server_name"
        os.Setenv("OTEL_SERVICE_NAME", serviceName)
    }
    otelAgentAddr, ok := os.LookupEnv("OTEL_EXPORTER_OTLP_ENDPOINT")
    if !ok {
        otelAgentAddr = "http://localhost:4317"
        os.Setenv("OTEL_EXPORTER_OTLP_ENDPOINT", otelAgentAddr)
    }
    zap.S().Infof("OTLP Trace connect to: %s with service name: %s", otelAgent
Addr, serviceName)

    traceExporter, err := otlptracegrpc.New(ctx, otlptracegrpc.WithInsecure(), otlp
tracegrpc.WithDialOption(grpc.WithBlock()))
    if err != nil {
        handleErr(err, "OTLP Trace gRPC Creation")
        return nil
    }

    tracerProvider := sdktrace.NewTracerProvider(
        sdktrace.WithBatcher(traceExporter),
        sdktrace.WithSampler(sdktrace.AlwaysSample()),
    sdktrace.WithResource(resource.NewWithAttributes(semconv.SchemaURL)))

    otel.SetTracerProvider(tracerProvider)
    otel.SetTextMapPropagator(propagation.NewCompositeTextMapPropagator(pro
pagation.TraceContext{}, propagation.Baggage{}))
```

```
    tracerExp = traceExporter
    return func() {
        // Shutdown will flush any remaining spans and shut down the exporter.
        handleErr(tracerProvider.Shutdown(ctx), "failed to shutdown TracerProvid
er")
    }
}

func handleErr(err error, message string) {
    if err != nil {
        zap.S().Errorf("%s: %v", message, err)
    }
}
```

在 main.go 中初始化跟踪器

修改 main 函数以在 main.go 中初始化跟踪器。另外当您的服务关闭时，应该调用 TracerProvider.Shutdown() 确保导出所有 Span。该服务将该调用作为主函数中的延迟函数：

```
func main() {
    // start otel tracing
    if shutdown := retryInitTracer(); shutdown != nil {
            defer shutdown()
        }
    ......
}
```

为应用添加 OTel Gin 中间件

通过在 main.go 中添加以下行来配置 Gin 以使用中间件：

```
import (
    ....
  "go.opentelemetry.io/contrib/instrumentation/github.com/gin-gonic/gin/otelgin"
```

```
)

func main() {
    ......
    r := gin.Default()
    r.Use(otelgin.Middleware("my-app"))
    ......
}
```

运行应用程序

- 本地调试运行

  注意：此步骤仅用于本地开发调试，生产环境中 Operator 会自动完成以下环境变量的注入。

  以上步骤已经完成了初始化 SDK 的工作，现在如果需要在本地开发进行调试，需要提前获取到 insight-system 命名空间下 insight-agent-opentelemerty-collector 的地址，假设为：insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317。

  因此，可以在你本地启动应用程序的时候添加如下环境变量：

  ```
  OTEL_SERVICE_NAME=my-golang-app OTEL_EXPORTER_OTLP_ENDPOINT=http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317 go run main.go...
  ```

- 生产环境运行

请参考通过 Operator 实现应用程序无侵入增强 中只注入环境变量注解相关介绍，为 deployment yaml 添加注解：

```
instrumentation.opentelemetry.io/inject-sdk: "insight-system/insight-opentelemetry-autoinstrumentation"
```

如果无法使用注解的方式，您可以手动在 deployment yaml 添加如下环境变量：

```
......
env:
  - name: OTEL_EXPORTER_OTLP_ENDPOINT
    value: 'http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.
local:4317'
  - name: OTEL_SERVICE_NAME
    value: "your depolyment name" # (1)
  - name: OTEL_K8S_NAMESPACE
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: metadata.namespace
  - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: spec.nodeName
  - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: metadata.name
  - name: OTEL_RESOURCE_ATTRIBUTES
    value: 'k8s.namespace.name=$(OTEL_K8S_NAMESPACE),k8s.node.name=$(OT
EL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=$(OTEL_RESOURCE_A
TTRIBUTES_POD_NAME)'
......
```

1. 修改此值

请求路由

OpenTelemetry gin/gonic 增强

Add one line to your import() stanza depending upon your request router:
middleware "go.opentelemetry.io/contrib/instrumentation/github.com/gin-gonic/gin/otelgin"

然后注入 OpenTelemetry 中间件：

router.Use(middleware.Middleware("my-app"))

OpenTelemetry gorillamux 增强

Add one line to your import() stanza depending upon your request router:
middleware "go.opentelemetry.io/contrib/instrumentation/github.com/gorilla/mux/otelmux"

然后注入 OpenTelemetry 中间件：

router.Use(middleware.Middleware("my-app"))

gRPC 增强

同样，OpenTelemetry 也可以帮助您自动检测 gRPC 请求。要检测您拥有的任何 gRPC 服务器，请将拦截器添加到服务器的实例化中。

```
import (
  grpcotel "go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc"
)
func main() {
  [...]

    s := grpc.NewServer(
        grpc.UnaryInterceptor(grpcotel.UnaryServerInterceptor()),
        grpc.StreamInterceptor(grpcotel.StreamServerInterceptor()),
    )
}
```

需要注意的是，如果你的程序里面使用到了 Grpc Client 调用第三方服务，你还需要对 Grpc Client 添加拦截器：

```
    [...]

    conn, err := grpc.Dial(addr, grpc.WithTransportCredentials(insecure.NewCredentials()),
        grpc.WithUnaryInterceptor(otelgrpc.UnaryClientInterceptor()),
        grpc.WithStreamInterceptor(otelgrpc.StreamClientInterceptor()),
    )
```

如果不使用请求路由

```
import (
  "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)
```

在将 http.Handler 传递给 ServeMux 的每个地方，您都将包装处理程序函数。例如，将进行以下替换：

```
- mux.Handle("/path", h)
+ mux.Handle("/path", otelhttp.NewHandler(h, "description of path"))
---
- mux.Handle("/path", http.HandlerFunc(f))
+ mux.Handle("/path", otelhttp.NewHandler(http.HandlerFunc(f), "description of path"))
```

通过这种方式，您可以确保使用 othttp 包装的每个函数都会自动收集其元数据并启动相应的跟踪。

自定义 Span

很多时候，OpenTelemetry 提供的中间件不能帮助我们记录更多内部调用的函数，需要我们自定义 Span 来记录

```
 ......
    _, span := otel.Tracer("GetServiceDetail").Start(ctx,
```

```
        "spanMetricDao.GetServiceDetail",
        trace.WithSpanKind(trace.SpanKindInternal))
    defer span.End()
  ......
```

向 span 添加自定义属性和事件

也可以将自定义属性或标签设置为 Span。要添加自定义属性和事件，请按照以下步骤操作：

导入跟踪和属性库

```
import (
    ...
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/trace"
)
```

从上下文中获取当前 Span

```
span := trace.SpanFromContext(c.Request.Context())
```

在当前 Span 中设置属性

```
span.SetAttributes(attribute.String("controller", "books"))
```

为当前 Span 添加 Event

添加 span 事件是使用 span 对象上的 AddEvent 完成的。

```
span.AddEvent(msg)
```

记录错误和异常

```
import "go.opentelemetry.io/otel/codes"

// 获取当前 span
span := trace.SpanFromContext(ctx)

// RecordError 会自动将一个错误转换成 span even
span.RecordError(err)
```

```
// 标记这个 span 错误
span.SetStatus(codes.Error, "internal error")
```

参考

有关 Demo 演示请参考：

- otel-grpc-examples
- opentelemetry-demo/productcatalogservice
- opentelemetry-collector-contrib/demo

使用 OTel 自动探针增强 Go 应用程序（实验性功能）

如果不想手动更改应用代码，您可以尝试使用本文基于 eBPF 的自动增强方式。该功能目前还处于捐献到 OpenTelemetry 社区的评审阶段，还不支持 Operator 通过注解方式注入（未来会支持），因此需要手动更改 Deployment YAML 或采用 patch 的方式。

前提条件

请确保 Insight Agent 已经就绪。如若没有，请参阅安装 insight-agent 采集数据，并确保以下三项就绪：

- 为 Insight-agent 开启 trace 功能
- trace 数据的地址以及端口是否填写正确
- deployment/opentelemetry-operator-controller-manager 和 deployment/insight-agent-opentelemetry-collector 对应的 Pod 已经准备就绪

安装 Instrumentation CR

在 Insight-system namespace 下安装，如已安装可跳过此步骤。

注意：该 CR 目前只支持注入对接 Insight 所需要的环境变量（包括服务名、链路上报地址等等），未来会支持注入 Golang 探针。

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: insight-opentelemetry-autoinstrumentation
  namespace: insight-system
spec:
  # https://github.com/open-telemetry/opentelemetry-operator/blob/main/docs/
```

```
api.md#instrumentationspecresource
  resource:
    addK8sUIDAttributes: true
  env:
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317
  sampler:
    # Enum: always_on, always_off, traceidratio, parentbased_always_on, parentbased_always_off, parentbased_traceidratio, jaeger_remote, xray
    type: always_on
  java:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java:1.17.0
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "false"
      - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
        value: "true"
      - name: SPLUNK_PROFILER_ENABLED
        value: "false"
  nodejs:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs:0.31.0
  python:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python:0.34b0
  dotnet:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-dotnet:0.3.1-beta.1
EOF
```

更改应用程序部署文件

- 添加环境变量注解

  这类注解只有一个，用于添加 OpenTelemetry 相关的环境变量，比如链路上报地址、容器所在的集群 id、命名空间等：

  instrumentation.opentelemetry.io/inject-sdk: "insight-system/insight-opentelemetry-autoinstrumentation"

  其中 value 被 / 分成两部分，第一个值 insight-system 是第二步安装的 CR 的命名空间，第二个值 insight-opentelemetry-autoinstrumentation 是这个 CR 的名字。

- 添加 golang ebpf 探针容器

  以下是示例代码：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting
  namespace: emojivoto
  labels:
    app.kubernetes.io/name: voting
    app.kubernetes.io/part-of: emojivoto
    app.kubernetes.io/version: v11
spec:
  replicas: 1
  selector:
    matchLabels:
      app: voting-svc
      version: v11
  template:
    metadata:
```

```yaml
      labels:
        app: voting-svc
        version: v11
      annotations:
        instrumentation.opentelemetry.io/inject-sdk: "insight-system/insight-opentelemetry-autoinstrumentation" # (1)
    spec:
      containers:
        - env:
            - name: GRPC_PORT
              value: "8080"
            - name: PROM_PORT
              value: "8801"
          image: docker.l5d.io/buoyantio/emojivoto-voting-svc:v11 # (2)
          name: voting-svc
          command:
            - /usr/local/bin/emojivoto-voting-svc
          ports:
            - containerPort: 8080
              name: grpc
            - containerPort: 8801
              name: prom
          resources:
            requests:
              cpu: 100m
        - name: emojivoto-voting-instrumentation
          image: docker.m.daocloud.io/keyval/otel-go-agent:v0.6.0
          env:
            - name: OTEL_TARGET_EXE
              value: /usr/local/bin/emojivoto-voting-svc # (3)
          securityContext:
            runAsUser: 0
```

```
            capabilities:
              add:
                - SYS_PTRACE
            privileged: true
          volumeMounts:
            - mountPath: /sys/kernel/debug
              name: kernel-debug
      volumes:
        - name: kernel-debug
          hostPath:
            path: /sys/kernel/debug
```

1. 用于添加 OpenTelemetry 相关的环境变量
2. 假设这是您的 Golang 应用程序
3. 注意与上面 command 内容/usr/local/bin/emojivoto-voting-svc 保持一致

最终生成的 Yaml 内容如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: voting-84b696c897-p9xbp
  generateName: voting-84b696c897-
  namespace: default
  uid: 742639b0-db6e-4f06-ac90-68a80e2b8a11
  resourceVersion: '65560793'
  creationTimestamp: '2022-10-19T07:08:56Z'
  labels:
    app: voting-svc
    pod-template-hash: 84b696c897
    version: v11
  annotations:
```

```
    cni.projectcalico.org/containerID: 0a987cf0055ce0dfbe75c3f30d580719eb4fbbd7e1af367064b588d4d4e4c7c7
    cni.projectcalico.org/podIP: 192.168.141.218/32
    cni.projectcalico.org/podIPs: 192.168.141.218/32
    instrumentation.opentelemetry.io/inject-sdk: insight-system/insight-opentelemetry-autoinstrumentation
spec:
  volumes:
    - name: launcherdir
      emptyDir: {}
    - name: kernel-debug
      hostPath:
        path: /sys/kernel/debug
        type: ''
    - name: kube-api-access-gwj5v
      projected:
        sources:
          - serviceAccountToken:
              expirationSeconds: 3607
              path: token
          - configMap:
              name: kube-root-ca.crt
              items:
                - key: ca.crt
                  path: ca.crt
          - downwardAPI:
              items:
                - path: namespace
                  fieldRef:
                    apiVersion: v1
                    fieldPath: metadata.namespace
        defaultMode: 420
```

```
    containers:
      - name: voting-svc
        image: docker.l5d.io/buoyantio/emojivoto-voting-svc:v11
        command:
          - /odigos-launcher/launch
          - /usr/local/bin/emojivoto-voting-svc
        ports:
          - name: grpc
            containerPort: 8080
            protocol: TCP
          - name: prom
            containerPort: 8801
            protocol: TCP
        env:
          - name: GRPC_PORT
            value: '8080'
          - name: PROM_PORT
            value: '8801'
          - name: OTEL_TRACES_EXPORTER
            value: otlp
          - name: OTEL_EXPORTER_OTLP_ENDPOINT
            value: >-
              http://insight-agent-opentelemetry-collector.insight-system.svc.clust
er.local:4317
          - name: OTEL_EXPORTER_OTLP_TIMEOUT
            value: '200'
          - name: SPLUNK_TRACE_RESPONSE_HEADER_ENABLED
            value: 'true'
          - name: OTEL_SERVICE_NAME
            value: voting
          - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
            valueFrom:
```

```
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
        - name: OTEL_RESOURCE_ATTRIBUTES_POD_UID
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.uid
        - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
        - name: OTEL_PROPAGATORS
          value: jaeger,b3
        - name: OTEL_TRACES_SAMPLER
          value: always_on
        - name: OTEL_RESOURCE_ATTRIBUTES
          value: >-
            k8s.container.name=voting-svc,k8s.deployment.name=voting,k8s.deplo
yment.uid=79e015e2-4643-44c0-993c-e486aebaba10,k8s.namespace.name=defa
ult,k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name
=$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME),k8s.pod.uid=$(OTEL_RESOURCE_
ATTRIBUTES_POD_UID),k8s.replicaset.name=voting-84b696c897,k8s.replicaset.uid
=63f56167-6632-415d-8b01-43a3db9891ff
      resources:
        requests:
          cpu: 100m
      volumeMounts:
        - name: launcherdir
          mountPath: /odigos-launcher
        - name: kube-api-access-gwj5v
```

```
        readOnly: true
        mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    imagePullPolicy: IfNotPresent
  - name: emojivoto-voting-instrumentation
    image: keyval/otel-go-agent:v0.6.0
    env:
      - name: OTEL_TARGET_EXE
        value: /usr/local/bin/emojivoto-voting-svc
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
        value: jaeger:4317
      - name: OTEL_SERVICE_NAME
        value: emojivoto-voting
    resources: {}
    volumeMounts:
      - name: kernel-debug
        mountPath: /sys/kernel/debug
      - name: kube-api-access-gwj5v
        readOnly: true
        mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    imagePullPolicy: IfNotPresent
    securityContext:
      capabilities:
        add:
          - SYS_PTRACE
      privileged: true
      runAsUser: 0
......
```

更多参考

- Go OpenTelemetry Automatic Instrumentation 入门
- Donating ebpf based instrumentation

使用 OTel 赋予应用可观测性

> 增强是使应用程序代码能够生成遥测数据的过程。即一些可以帮助您监视或测量应用程序的性能和状态的东西。

OpenTelemetry 是领先的开源项目，为主要编程语言和流行框架提供检测库。它是云原生计算基金会下的一个项目，得到了社区庞大资源的支持。 它为采集的数据提供标准化的数据格式，无需集成特定的供应商。

Insight 支持用于检测应用程序的 OpenTelemetry 来增强您的应用程序。

本指南介绍了使用 OpenTelemetry 进行遥测增强的基本概念。 OpenTelemetry 还有一个由库、插件、集成和其他有用工具组成的生态系统来扩展它。 您可以在 Otel Registry 中找到这些资源。

您可以使用任何开放标准库进行遥测增强，并使用 Insight 作为可观察性后端来摄取、分析和可视化数据。

为了增强您的代码，您可以使用 OpenTelemetry 为特定语言提供的增强操作：

Insight 目前提供了使用 OpenTelemetry 增强 .Net NodeJS、Java、Python 和 Golang 应用程序的简单方法。请遵循以下指南。

链路增强

- 链路接入的最佳实践：通过 Operator 实现应用程序无侵入增强
- 以 Go 语言为例的手动埋点接入：使用 OpenTelemetry SDK 增强 Go 应用程序
- 利用 ebpf 实现 Go 语言无侵入探针（实验性功能）

指标增强

指标接入的最佳实践：使用 OpenTelemetry SDK 为应用程序暴露指标。

日志增强

TBD

使用 OTel SDK 为应用程序暴露指标

本文仅供希望评估或探索正在开发的 OTLP 指标的用户参考。

OpenTelemetry 项目要求以必须在 OpenTelemetry 协议（OTLP）中发出数据的语言提供 API 和 SDK。

针对 Golang 应用程序

Golang 可以通过 sdk 暴露 runtime 指标，具体来说，在应用中添加以下方法开启 metrics 暴露器：

安装相关依赖

切换/进入到应用程序源文件夹后运行以下命令：

```
go get go.opentelemetry.io/otel \
  go.opentelemetry.io/otel/attribute \
  go.opentelemetry.io/otel/exporters/prometheus \
  go.opentelemetry.io/otel/metric/global \
  go.opentelemetry.io/otel/metric/instrument \
  go.opentelemetry.io/otel/sdk/metric
```

使用 OTel SDK 创建初始化函数

```
import（
    .....

    "go.opentelemetry.io/otel/attribute"
    otelPrometheus "go.opentelemetry.io/otel/exporters/prometheus"
    "go.opentelemetry.io/otel/metric/global"
    "go.opentelemetry.io/otel/metric/instrument"
    "go.opentelemetry.io/otel/sdk/metric/aggregator/histogram"
    controller "go.opentelemetry.io/otel/sdk/metric/controller/basic"
    "go.opentelemetry.io/otel/sdk/metric/export/aggregation"
```

```
    processor "go.opentelemetry.io/otel/sdk/metric/processor/basic"
    selector "go.opentelemetry.io/otel/sdk/metric/selector/simple"
)
func (s *insightServer) initMeter() *otelPrometheus.Exporter {
    s.meter = global.Meter("xxx")

    config := otelPrometheus.Config{
        DefaultHistogramBoundaries: []float64{1, 2, 5, 10, 20, 50},
        Gatherer:                   prometheus.DefaultGatherer,
        Registry:                   prometheus.NewRegistry(),
        Registerer:                 prometheus.DefaultRegisterer,
    }

    c := controller.New(
        processor.NewFactory(
            selector.NewWithHistogramDistribution(
                histogram.WithExplicitBoundaries(config.DefaultHistogramBoundar
ies),
            ),
            aggregation.CumulativeTemporalitySelector(),
            processor.WithMemory(true),
        ),
    )

    exporter, err := otelPrometheus.New(config, c)
    if err != nil {
        zap.S().Panicf("failed to initialize prometheus exporter %v", err)
    }

    global.SetMeterProvider(exporter.MeterProvider())
```

```
    http.HandleFunc("/metrics", exporter.ServeHTTP)

    go func() {
        _ = http.ListenAndServe(fmt.Sprintf(":%d", 8888), nil)
    }()

    zap.S().Info("Prometheus server running on ", fmt.Sprintf(":%d", port))
    return exporter
}
```

以上方法会为您的应用暴露一个指标接口: http://localhost:8888/metrics

随后，在 main.go 中对其进行初始化：

```
func main() {
......
    tp := initMeter()
......
}
```

此外，如果想添加自定义指标，可以参考：

```
// exposeClusterMetric expose metric like "insight_logging_count{} 1"
func (s *insightServer) exposeLoggingMetric(lserver *log.LogService) {
    s.meter = global.Meter("insight.io/basic")

    var lock sync.Mutex
    logCounter, err := s.meter.AsyncFloat64().Counter("insight_log_total")
    if err != nil {
        zap.S().Panicf("failed to initialize instrument: %v", err)
    }

    _ = s.meter.RegisterCallback([]instrument.Asynchronous{logCounter}, func(ctx
context.Context) {
```

```
            lock.Lock()
            defer lock.Unlock()
            count, err := lserver.Count(ctx)
            if err == nil || count != -1 {
                logCounter.Observe(ctx, float64(count))
            }
        })
}
```

随后，在 main.go 调用该方法：

```
.....
s.exposeLoggingMetric(lservice)
.....
```

您可以通过访问 http://localhost:8888/metrics 来检查您的指标是否正常工作。

针对 Java 应用程序

Java 在使用 otel agent 在完成链路的自动接入的基础上，通过添加环境变量：

```
OTEL_METRICS_EXPORTER=prometheus
```

就可以直接暴露 JVM 相关指标，您可以通过访问 http://localhost:8888/metrics 来检查您的指标是否正常工作。

随后，再配合 prometheus serviceMonitor 即可完成指标的接入。 如果想暴露自定义指标请参阅 opentelemetry-java-docs/prometheus。

主要分以下两步：

- 创建 meter provider，并指定 prometheus 作为 exporter。

```
/*
* Copyright The OpenTelemetry Authors
* SPDX-License-Identifier: Apache-2.0
*/
```

```
package io.opentelemetry.example.prometheus;

import io.opentelemetry.api.metrics.MeterProvider;
import io.opentelemetry.exporter.prometheus.PrometheusHttpServer;
import io.opentelemetry.sdk.metrics.SdkMeterProvider;
import io.opentelemetry.sdk.metrics.export.MetricReader;

public final class ExampleConfiguration {

  /**
   * Initializes the Meter SDK and configures the prometheus collector wit
h all default settings.
   *
   * @param prometheusPort the port to open up for scraping.
   * @return A MeterProvider for use in instrumentation.
   */
  static MeterProvider initializeOpenTelemetry(int prometheusPort) {
    MetricReader prometheusReader = PrometheusHttpServer.builder().set
Port(prometheusPort).build();

    return SdkMeterProvider.builder().registerMetricReader(prometheusRead
er).build();
  }
}
```

- 自定义 meter 并开启 http server

```
package io.opentelemetry.example.prometheus;

import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.api.metrics.Meter;
```

```
import io.opentelemetry.api.metrics.MeterProvider;
import java.util.concurrent.ThreadLocalRandom;

/**
* Example of using the PrometheusHttpServer to convert OTel metrics to
 Prometheus format and expose
* these to a Prometheus instance via a HttpServer exporter.
*
* <p>A Gauge is used to periodically measure how many incoming mess
ages are awaiting processing.
* The Gauge callback gets executed every collection interval.
*/
public final class PrometheusExample {
  private long incomingMessageCount;

  public PrometheusExample(MeterProvider meterProvider) {
    Meter meter = meterProvider.get("PrometheusExample");
    meter
        .gaugeBuilder("incoming.messages")
        .setDescription("No of incoming messages awaiting processing")
        .setUnit("message")
        .buildWithCallback(result -> result.record(incomingMessageCount,
Attributes.empty()));
  }

  void simulate() {
    for (int i = 500; i > 0; i--) {
      try {
        System.out.println(
            i + " Iterations to go, current incomingMessageCount is: "
+ incomingMessageCount);
        incomingMessageCount = ThreadLocalRandom.current().nextLong(1
```

```
00);
            Thread.sleep(1000);
          } catch (InterruptedException e) {
            // ignored here
          }
        }
      }

      public static void main(String[] args) {
        int prometheusPort = 8888;

        // it is important to initialize the OpenTelemetry SDK as early as po
ssible in your process.
        MeterProvider meterProvider = ExampleConfiguration.initializeOpenTele
metry(prometheusPort);

        PrometheusExample prometheusExample = new PrometheusExample
(meterProvider);

        prometheusExample.simulate();

        System.out.println("Exiting");
      }
    }
```

随后，待 java 应用程序运行之后，您可以通过访问 http://localhost:8888/metrics 来检查您的指标是否正常工作。

Insight 采集指标

最后重要的是，您已经在应用程序中暴露出了指标，现在需要 Insight 来采集指标。

推荐的指标暴露方式是通过 servicemonitor 或者 podmonitor。

创建 servicemonitor/podmonitor

添加的 servicemonitor/podmonitor 需要打上 label："operator.insight.io/managed-by": "insight" 才会被 Operator 识别：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  labels:
    operator.insight.io/managed-by: insight
spec:
  selector:
    matchLabels:
      app: example-app
  endpoints:
  - port: web
  namespaceSelector:
    any: true
```

通过 Operator 实现应用程序无侵入增强

目前只有 Java、NodeJs、Python、.Net 支持 Operator 的方式无侵入接入，Golang 后续会完善。

前提条件

请确保 Insight Agent 已经就绪。如若没有，请参考安装 insight-agent 采集数据并确保以下三项就绪：

- 为 Insight-agent 开启 trace 功能
- trace 数据的地址以及端口是否填写正确
- deployment/insight-agent-opentelemetry-operator 和 deployment/insight-agent-opentelemetry-collector 对应的 Pod 已经准备就绪

安装 Instrumentation CR

在 insight-system 命名空间下安装，不同版本之间有一些细小的差别。

Insight v0.18.x

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: insight-opentelemetry-autoinstrumentation
  namespace: insight-system
spec:
  # https://github.com/open-telemetry/opentelemetry-operator/blob/main/docs/api.md#instrumentationspecresource
  resource:
    addK8sUIDAttributes: true
  env:
```

```
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317
  sampler:
    # Enum: always_on, always_off, traceidratio, parentbased_always_on, parentbased_always_off, parentbased_traceidratio, jaeger_remote, xray
    type: always_on
  java:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java:1.25.0
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "false"
      - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
        value: "true"
      - name: SPLUNK_PROFILER_ENABLED
        value: "false"
      - name: OTEL_METRICS_EXPORTER
        value: "prometheus"
      - name: OTEL_METRICS_EXPORTER_PORT
        value: "9464"
  nodejs:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs:0.37.0
  python:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python:0.38b0
  go:
    # Must set the default value manually for now.
    # See https://github.com/open-telemetry/opentelemetry-operator/issues/1756 for details.
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-go-instrumentation
```

```
/autoinstrumentation-go:v0.2.1-alpha
EOF
```

Insight v0.17.x

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: insight-opentelemetry-autoinstrumentation
  namespace: insight-system
spec:
  # https://github.com/open-telemetry/opentelemetry-operator/blob/main/docs/api.md#instrumentationspecresource
  resource:
    addK8sUIDAttributes: true
  env:
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317
  sampler:
    # Enum: always_on, always_off, traceidratio, parentbased_always_on, parentbased_always_off, parentbased_traceidratio, jaeger_remote, xray
    type: always_on
  java:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java:1.23.0
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "false"
      - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
        value: "true"
      - name: SPLUNK_PROFILER_ENABLED
        value: "false"
```

```
      - name: OTEL_METRICS_EXPORTER
        value: "prometheus"
      - name: OTEL_METRICS_EXPORTER_PORT
        value: "9464"
  nodejs:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs:0.34.0
  python:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python:0.33b0
EOF
```

Insight v0.16.x

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: insight-opentelemetry-autoinstrumentation
  namespace: insight-system
spec:
  # https://github.com/open-telemetry/opentelemetry-operator/blob/main/docs/api.md#instrumentationspecresource
  resource:
    addK8sUIDAttributes: true
  env:
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317
  sampler:
    # Enum: always_on, always_off, traceidratio, parentbased_always_on, parentbased_always_off, parentbased_traceidratio, jaeger_remote, xray
    type: always_on
  java:
```

```
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java:1.23.0
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "false"
      - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
        value: "true"
      - name: SPLUNK_PROFILER_ENABLED
        value: "false"
      - name: OTEL_METRICS_EXPORTER
        value: "prometheus"
      - name: OTEL_METRICS_EXPORTER_PORT
        value: "9464"
  nodejs:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs:0.34.0
  python:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python:0.33b0
EOF
```

与服务网格产品 Mspider 链路串联场景

如果您开启了服务网格的链路追踪能力，你需要在安装 Instrumentation CR 时，额外增加一个环境变量注入的配置：

```
    - name: OTEL_SERVICE_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.labels['app']
```

完整示例如下：

Insight v0.18.x

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: insight-opentelemetry-autoinstrumentation
  namespace: insight-system
spec:
  # https://github.com/open-telemetry/opentelemetry-operator/blob/main/docs/api.md#instrumentationspecresource
  resource:
    addK8sUIDAttributes: true
  env:
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://insight-agent-opentelemetry-collector.insight-system.svc.cluster.local:4317
    - name: OTEL_SERVICE_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.labels['app']
  sampler:
    # Enum: always_on, always_off, traceidratio, parentbased_always_on, parentbased_always_off, parentbased_traceidratio, jaeger_remote, xray
    type: always_on
  java:
    image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java:1.25.0
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "false"
      - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
        value: "true"
```

```
        - name: SPLUNK_PROFILER_ENABLED
          value: "false"
        - name: OTEL_METRICS_EXPORTER
          value: "prometheus"
        - name: OTEL_METRICS_EXPORTER_PORT
          value: "9464"
    nodejs:
      image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs:0.37.0
    python:
      image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python:0.38b0
    dotnet:
      image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrumentation-dotnet:0.6.0
    go:
      # Must set the default value manually for now.
      # See https://github.com/open-telemetry/opentelemetry-operator/issues/1756 for details.
      image: ghcr.m.daocloud.io/open-telemetry/opentelemetry-go-instrumentation/autoinstrumentation-go:v0.2.1-alpha
EOF
```

添加注解，自动接入链路

以上就绪之后，您就可以通过注解（Annotation）方式为应用程序接入链路追踪了，Otel 目前支持通过注解的方式接入链路。根据服务语言，需要添加上不同的 pod annotations。 每个服务可添加两类注解之一：

- 只注入环境变量注解

这类注解只有一个，用于添加 otel 相关的环境变量，比如链路上报地址、容器所在的集群 id、命名空间等（这个注解在应用不支持自动探针语言时十分有用）

instrumentation.opentelemetry.io/inject-sdk: "insight-system/insight-opentelemetry-autoinstrumentation"

其中 value 被 / 分成两部分，第一个值(insight-system）是上一步安装的 CR 的命名空间，第二个值(insight-opentelemetry-autoinstrumentation）是这个 CR 的名字。

- 自动探针注入以及环境变量注入注解

  这类注解目前有 4 个，分别对应 4 种不同的编程语言：java、nodejs、python、dotnet，使用它后就会对 spec.pod 下的第一个容器注入自动探针以及 otel 默认环境变量：

  1. Java 应用

     instrumentation.opentelemetry.io/inject-java: "insight-system/insight-opentelemetry-autoinstrumentation"

  2. NodeJs 应用

     instrumentation.opentelemetry.io/inject-nodejs: "insight-system/insight-opentelemetry-autoinstrumentation"

  3. Python 应用

     instrumentation.opentelemetry.io/inject-python: "insight-system/insight-opentelemetry-autoinstrumentation"

  4. Dotnet 应用

     instrumentation.opentelemetry.io/inject-dotnet: "insight-system/insight-opentelemetry-autoinstrumentation"

自动注入示例 Demo

注意这个 annotations 是加在 spec.annotations 下的。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  replicas: 1
  template:
    metadata:
      labels:
        app: my-app
      annotations:
        instrumentation.opentelemetry.io/inject-java: "insight-system/insight-opentelemetry-autoinstrumentation"
    spec:
      containers:
      - name: myapp
        image: jaegertracing/vertx-create-span:operator-e2e-tests
        ports:
          - containerPort: 8080
            protocol: TCP
```

最终生成的 Yaml 内容如下：

```yaml
apiVersion: v1
kind: Pod
```

```
metadata:
  name: my-deployment-with-sidecar-565bd877dd-nqkk6
  generateName: my-deployment-with-sidecar-565bd877dd-
  namespace: default
  uid: aa89ca0d-620c-4d20-8bc1-37d67bad4ea4
  resourceVersion: '2668986'
  creationTimestamp: '2022-04-08T05:58:48Z'
  labels:
    app: my-pod-with-sidecar
    pod-template-hash: 565bd877dd
  annotations:
    cni.projectcalico.org/containerID: 234eae5e55ea53db2a4bc2c0384b9a1021ed3
908f82a675e4a92a49a7e80dd61
    cni.projectcalico.org/podIP: 192.168.134.133/32
    cni.projectcalico.org/podIPs: 192.168.134.133/32
    instrumentation.opentelemetry.io/inject-java: 'true'
    sidecar.opentelemetry.io/inject: 'false'
spec:
  volumes:
    - name: kube-api-access-sp2mz
      projected:
        sources:
          - serviceAccountToken:
              expirationSeconds: 3607
              path: token
          - configMap:
              name: kube-root-ca.crt
              items:
                - key: ca.crt
                  path: ca.crt
          - downwardAPI:
              items:
```

```
            - path: namespace
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.namespace
        defaultMode: 420
    - name: opentelemetry-auto-instrumentation
      emptyDir: {}
  initContainers:
    - name: opentelemetry-auto-instrumentation
      image: >-
        ghcr.m.daocloud.io/open-telemetry/opentelemetry-operator/autoinstrume
ntation-java
      command:
        - cp
        - /javaagent.jar
        - /otel-auto-instrumentation/javaagent.jar
      resources: {}
      volumeMounts:
        - name: opentelemetry-auto-instrumentation
          mountPath: /otel-auto-instrumentation
        - name: kube-api-access-sp2mz
          readOnly: true
          mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: Always
  containers:
    - name: myapp
      image: ghcr.io/pavolloffay/spring-petclinic:latest
      env:
        - name: OTEL_JAVAAGENT_DEBUG
          value: 'true'
```

```
        - name: OTEL_INSTRUMENTATION_JDBC_ENABLED
          value: 'true'
        - name: SPLUNK_PROFILER_ENABLED
          value: 'false'
        - name: JAVA_TOOL_OPTIONS
          value: ' -javaagent:/otel-auto-instrumentation/javaagent.jar'
        - name: OTEL_TRACES_EXPORTER
          value: otlp
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: http://insight-agent-opentelemetry-collector.svc.cluster.local:431
7
        - name: OTEL_EXPORTER_OTLP_TIMEOUT
          value: '20'
        - name: OTEL_TRACES_SAMPLER
          value: parentbased_traceidratio
        - name: OTEL_TRACES_SAMPLER_ARG
          value: '0.85'
        - name: SPLUNK_TRACE_RESPONSE_HEADER_ENABLED
          value: 'true'
        - name: OTEL_SERVICE_NAME
          value: my-deployment-with-sidecar
        - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
        - name: OTEL_RESOURCE_ATTRIBUTES_POD_UID
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.uid
        - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
```

```
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: spec.nodeName
          - name: OTEL_RESOURCE_ATTRIBUTES
            value: >-
              k8s.container.name=myapp,k8s.deployment.name=my-deployment-with-sidecar,k8s.deployment.uid=8de6929d-dda0-436c-bca1-604e9ca7ea4e,k8s.namespace.name=default,k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME),k8s.pod.uid=$(OTEL_RESOURCE_ATTRIBUTES_POD_UID),k8s.replicaset.name=my-deployment-with-sidecar-565bd877dd,k8s.replicaset.uid=190d5f6e-ba7f-4794-b2e6-390b5879a6c4
          - name: OTEL_PROPAGATORS
            value: jaeger,b3
        resources: {}
        volumeMounts:
          - name: kube-api-access-sp2mz
            readOnly: true
            mountPath: /var/run/secrets/kubernetes.io/serviceaccount
          - name: opentelemetry-auto-instrumentation
            mountPath: /otel-auto-instrumentation
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        imagePullPolicy: Always
    restartPolicy: Always
    terminationGracePeriodSeconds: 30
    dnsPolicy: ClusterFirst
    serviceAccountName: default
    serviceAccount: default
    nodeName: k8s-master3
    securityContext:
```

```
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  schedulerName: default-scheduler
  tolerations:
    - key: node.kubernetes.io/not-ready
      operator: Exists
      effect: NoExecute
      tolerationSeconds: 300
    - key: node.kubernetes.io/unreachable
      operator: Exists
      effect: NoExecute
      tolerationSeconds: 300
  priority: 0
  enableServiceLinks: true
  preemptionPolicy: PreemptLowerPriority
```

链路查询

如何查询已经接入的服务，参考链路查询。

已有 JVM 指标的 Java 应用对接可观测性

如果您的 Java 应用通过其他方式（比如 Spring Boot Actuator）暴露了 JVM 的监控指标， 我们需要让监控数据被采集到。您可以通过在工作负载种添加注解（Kubernetes Annotations）的方式让 Insight 来采集已有的 JVM 指标：

```yaml
annatation:
  insight.opentelemetry.io/metric-scrape: "true"  # 是否采集
  insight.opentelemetry.io/metric-path: "/"       # 采集指标的路径
  insight.opentelemetry.io/metric-port: "9464"    # 采集指标的端口
```

例如为 my-deployment-app 添加注解：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment-app
spec:
  selector:
    matchLabels:
      app: my-deployment-app
      app.kubernetes.io/name: my-deployment-app
  replicas: 1
  template:
    metadata:
      labels:
        app: my-deployment-app
        app.kubernetes.io/name: my-deployment-app
      annotations:
        insight.opentelemetry.io/metric-scrape: "true"  # 是否采集
```

```
insight.opentelemetry.io/metric-path: "/"          # 采集指标的路径
insight.opentelemetry.io/metric-port: "9464"     # 采集指标的端口
```

使用 JMX Exporter 暴露 JVM 监控指标

JMX-Exporter 提供了两种用法:

1. 启动独立进程。JVM 启动时指定参数，暴露 JMX 的 RMI 接口，JMX Exporter 调用 RMI 获取 JVM 运行时状态数据， 转换为 Prometheus metrics 格式，并暴露端口让 Prometheus 采集。
2. JVM 进程内启动(in-process)。JVM 启动时指定参数，通过 javaagent 的形式运行 JMX-Exporter 的 jar 包， 进程内读取 JVM 运行时状态数据，转换为 Prometheus metrics 格式，并暴露端口让 Prometheus 采集。

!!! note

官方不推荐使用第一种方式，一方面配置复杂，另一方面因为它需要一个单独的进程，而这个进程本身的监控又成了新的问题，
所以本文重点围绕第二种用法讲如何在 Kubernetes 环境下使用 JMX Exporter 暴露 JVM 监控指标。

这里使用第二种用法，启动 JVM 时需要指定 JMX Exporter 的 jar 包文件和配置文件。 jar 包是二进制文件，不好通过 configmap 挂载，配置文件我们几乎不需要修改， 所以建议是直接将 JMX Exporter 的 jar 包和配置文件都打包到业务容器镜像中。

其中，第二种方式我们可以选择将 JMX Exporter 的 jar 文件放在业务应用镜像中， 也可以选择在部署的时候挂载进去。这里分别对两种方式做一个介绍:

方式一：将 JMX Exporter JAR 文件构建至业务镜像中

prometheus-jmx-config.yaml 内容如下:

yaml title="prometheus-jmx-config.yaml" ... ssl: false lowercaseOutputName: false lowercaseOutputLabelNames: false rules: - pattern: ".*"

!!! note

更多配置项请参考底部介绍或[Prometheus 官方文档](https://github.com/prometheus/jmx_exporter#configuration)。

然后准备 jar 包文件，可以在 jmx_exporter 的 Github 页面找到最新的 jar 包下载地址并参考如下 Dockerfile:

```
FROM openjdk:11.0.15-jre
WORKDIR /app/
COPY target/my-app.jar ./
COPY prometheus-jmx-config.yaml ./
RUN set -ex; \
    curl -L -O https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.17.2/jmx_prometheus_javaagent-0.17.2.jar;
ENV JAVA_TOOL_OPTIONS=-javaagent:/app/jmx_prometheus_javaagent-0.17.2.jar=8088:/app/prometheus-jmx-config.yaml
EXPOSE 8081 8999 8080 8888
ENTRYPOINT java $JAVA_OPTS -jar my-app.jar
```

注意：

- 启动参数格式：-javaagent:=:
- 这里使用了 8088 端口暴露 JVM 的监控指标，如果和 Java 应用冲突，可自行更改

方式二： 通过 init container 容器挂载

我们需要先将 JMX exporter 做成 Docker 镜像，以下 Dockerfile 仅供参考：

```
FROM alpine/curl:3.14
WORKDIR /app/
# 将前面创建的 config 文件拷贝至镜像
COPY prometheus-jmx-config.yaml ./
# 在线下载 jmx prometheus javaagent jar
```

```
RUN set -ex; \
    curl -L -O https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.17.2/jmx_prometheus_javaagent-0.17.2.jar;
```

根据上面 Dockerfile 构建镜像：docker build -t my-jmx-exporter .

在 Java 应用部署 Yaml 中加入如下 init container：

??? note “点击展开 YAML 文件”

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-demo-app
  labels:
    app: my-demo-app
spec:
  selector:
    matchLabels:
      app: my-demo-app
  template:
    metadata:
      labels:
        app: my-demo-app
    spec:
      imagePullSecrets:
      - name: registry-pull
      initContainers:
      - name: jmx-sidecar
        image: my-jmx-exporter
        command: ["cp", "-r", "/app/jmx_prometheus_javaagent-0.17.2.jar", "/target/jmx_prometheus_javaagent-0.17.2.jar"] ❶
        volumeMounts:
```

```
      - name: sidecar
        mountPath: /target
    containers:
    - image: my-demo-app-image
      name: my-demo-app
      resources:
        requests:
          memory: "1000Mi"
          cpu: "500m"
        limits:
          memory: "1000Mi"
          cpu: "500m"
      ports:
      - containerPort: 18083
      env:
      - name: JAVA_TOOL_OPTIONS
        value: "-javaagent:/app/jmx_prometheus_javaagent-0.17.2.jar=8088:/ap
p/prometheus-jmx-config.yaml" ❷
      volumeMounts:
      - name: host-time
        mountPath: /etc/localtime
        readOnly: true
      - name: sidecar
        mountPath: /sidecar
    volumes:
    - name: host-time
      hostPath:
        path: /etc/localtime
    - name: sidecar   #共享 agent 文件夹
      emptyDir: {}
    restartPolicy: Always
```

经过如上的改造之后，示例应用 my-demo-app 具备了暴露 JVM 指标的能力。运行服务之后，我们可以通过 http://lcoalhost:8088 访问服务暴露出来的 prometheus 格式的指标。

接着，您可以参考 已有 JVM 指标的 Java 应用对接可观测性。

使用 OpenTelemetry Java Agent 暴露 JVM 监控指标

在 Opentelemetry Agent v1.20.0 及以上版本中，Opentelemetry Agent 新增了 JMX Metric Insight 模块，如果你的应用已经集成了 Opentelemetry Agent 去采集应用链路，那么你不再需要另外引入其他 Agent 去为我们的应用暴露 JMX 指标。Opentelemetry Agent 也是通过检测应用程序中本地可用的 MBean 公开的指标，对其进行收集并暴露指标。

Opentelemetry Agent 也针对常见的 Java Server 或框架内置了一些监控的样例，请参考预定义的指标。

使用 OpenTelemetry Java Agent 同样需要考虑如何将 JAR 挂载进容器，除了可以参考上面 JMX Exporter 挂载 JAR 文件的方式外，我们还可以借助 Opentelemetry 提供的 Operator 的能力来实现自动为我们的应用开启 JVM 指标暴露：

如果你的应用已经集成了 Opentelemetry Agent 去采集应用链路，那么你不再需要另外引入其他 Agent 去为我们的应用暴露 JMX 指标。Opentelemetry Agent 通过检测应用程序中本地可用的 MBean 公开的指标，现在可以本地收集并暴露指标接口。

但是，截至目前版本，你仍然需要手动为应用加上相应注解之后，JVM 数据才会被 Insight 采集到，具体注解内容请参考 已有 JVM 指标的 Java 应用对接可观测性。

为 Java 中间件暴露指标

Opentelemetry Agent 也内置了一些中间件监控的样例，请参考 预定义指标。

默认没有指定任何类型，需要通过 -Dotel.jmx.target.system JVM Options 指定，比如 -Dotel.jmx.target.system=jetty,kafka-broker。

参考

- Gaining JMX Metric Insights with the OpenTelemetry Java Agent
- Otel jmx metrics

开始监控 Java 应用

此文档主要描述如何对客户 Java 应用的 JVM 进行监控。 其中会介绍已经暴露 JVM 指标和仍未暴露 JVM 指标的 Java 应用如何与可观测性 Insight 对接。

如果您的 Java 应用未开始暴露 JVM 指标，您可以参考如下文档：

- 使用 JMX Exporter 暴露 JVM 监控指标
- 使用 OpenTelemetry Java Agent 暴露 JVM 监控指标

如果您的 Java 应用已经暴露 JVM 指标，您可以参考如下文档：

- 已有 JVM 指标的 Java 应用对接可观测性

使用 Insight 定位应用异常

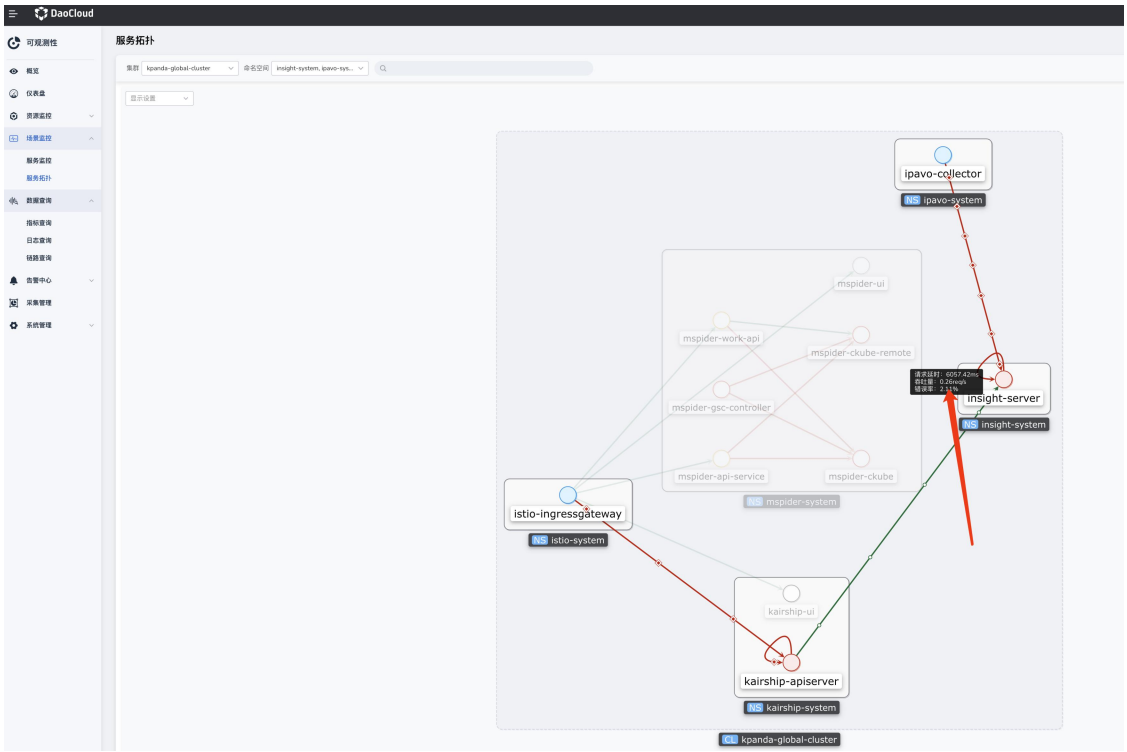
本文将以 DCE 5.0 中举例，讲解如何通过 Insight 发现 DCE 5.0 中异常的组件并分析出组件异常的根因。

本文假设你已经了解 Insight 的产品功能或愿景。

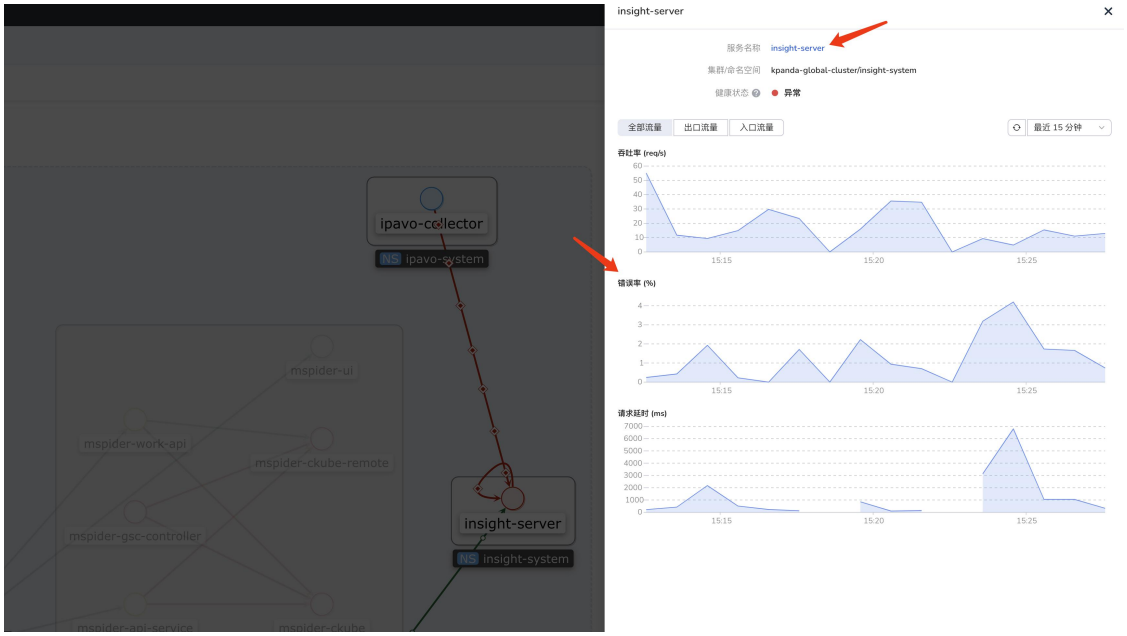
拓扑图 —— 从宏观察觉异常

随着企业对微服务架构的实践，企业中的服务数量可能会面临着数量多、调用复杂的情况，开发或运维人员很难理清服务之间的关系，因此，我们提供了拓扑图监控的功能，我们可以通过拓扑图对当前系统中运行的微服务状况进行初步诊断。

如下图所示，我们通过拓扑图发现其中 Insight-Server 这个节点的颜色为 红色，并将鼠标移到该节点上，发现该节点的错误率为 2.11%。因此，我们希望查看更多细节去找到造成该服务错误率不为 0 的原因:



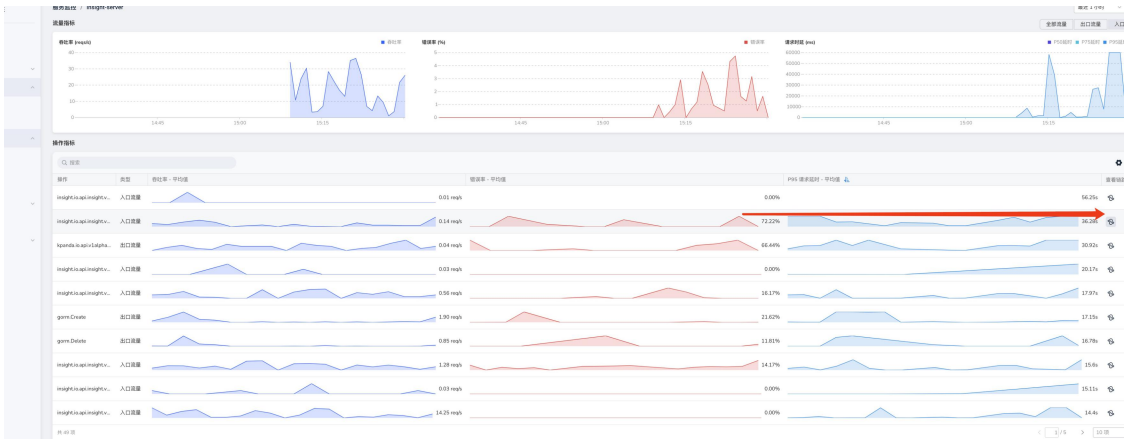
当然，我们也可以点击最顶部的服务名，进入到该服务的总览界面：



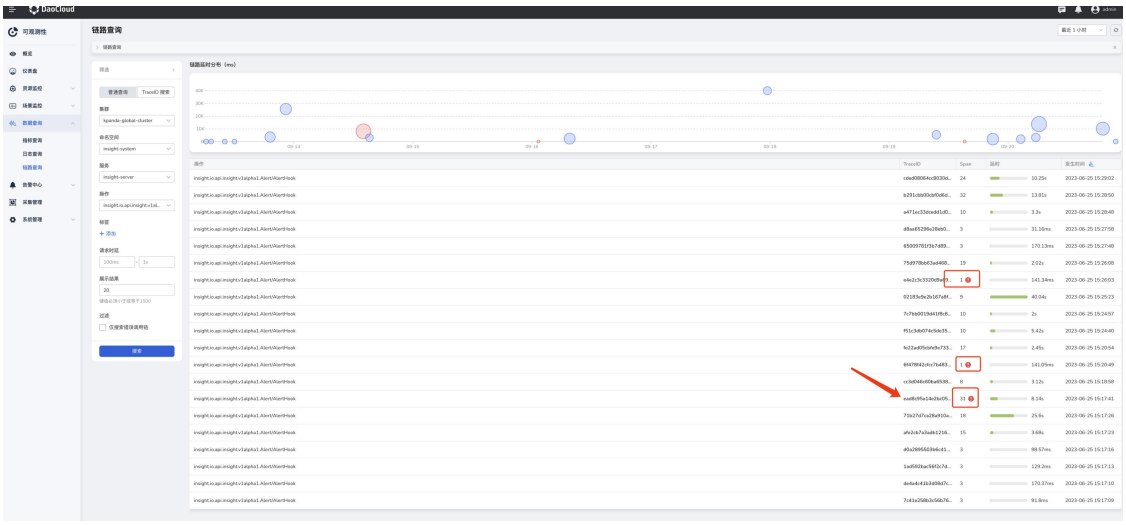
02

服务总览 —— 具体分析的开始

当你需要根据服务的入口和出口流量分别分析的时候，你可以在右上角进行筛选切换，筛选数据之后，我们发现该服务有很多 操作 对应的错误率都不为 0. 此时，我们可以通过点击 查看链路 对该操作 在这段时间产生的并记录下来的链路进行分析：



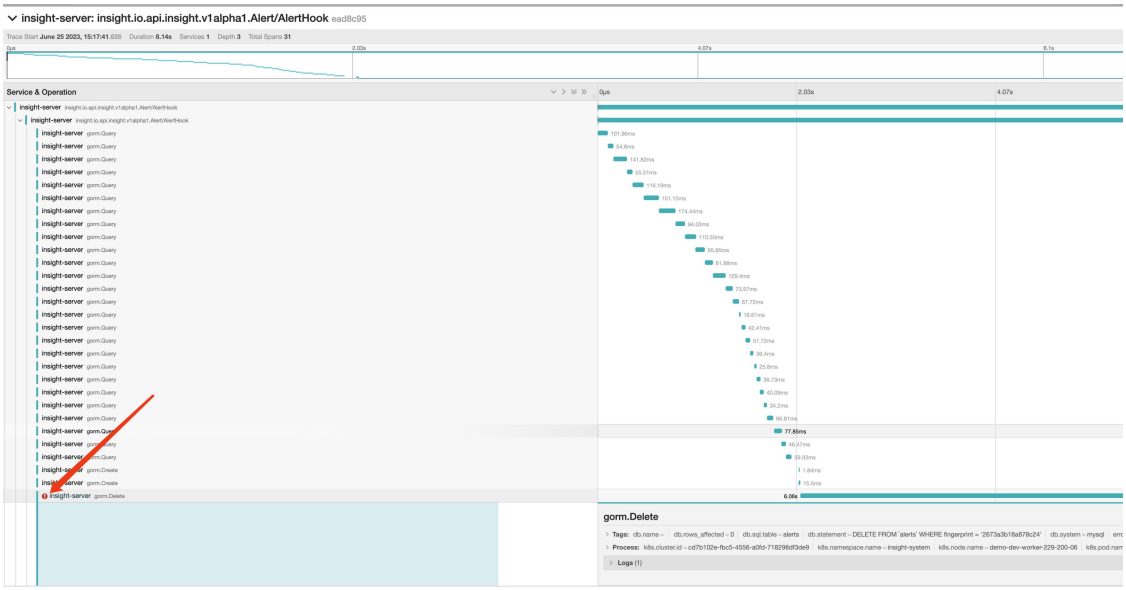
03



04

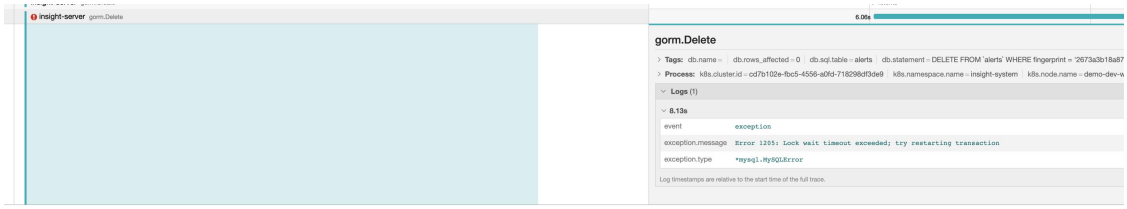
链路详情 —— 找到错误根因，消灭它们

在链路列表中，我们可以通过界面直观地发现链路列表中存在着错误的链路（上图中红框圈起来的），我们可以点击错误的链路查看链路详情，如下图所示：



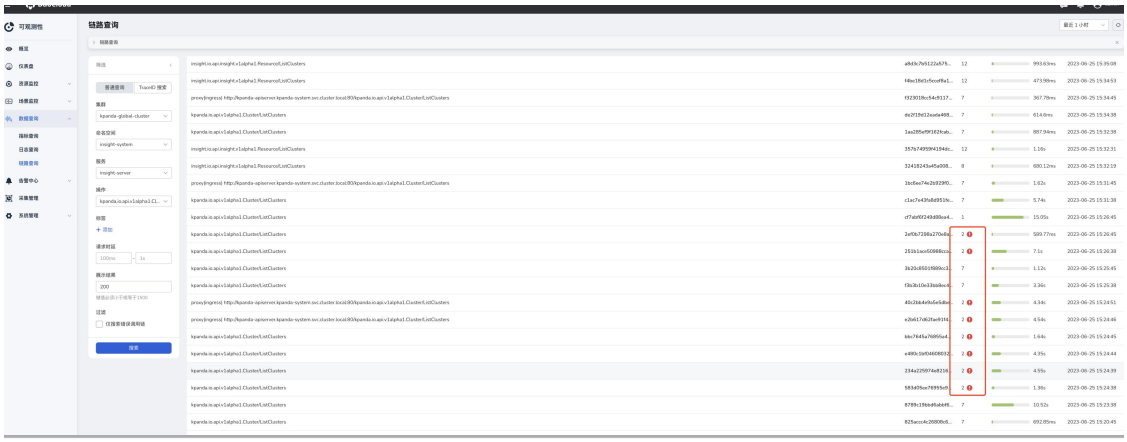
05

在链路图中我们也可以一眼就发现链路的最后一条数据是处于 错误 状态，将其右边 Logs 展开，我们定位到了造成这次请求错误的原因：

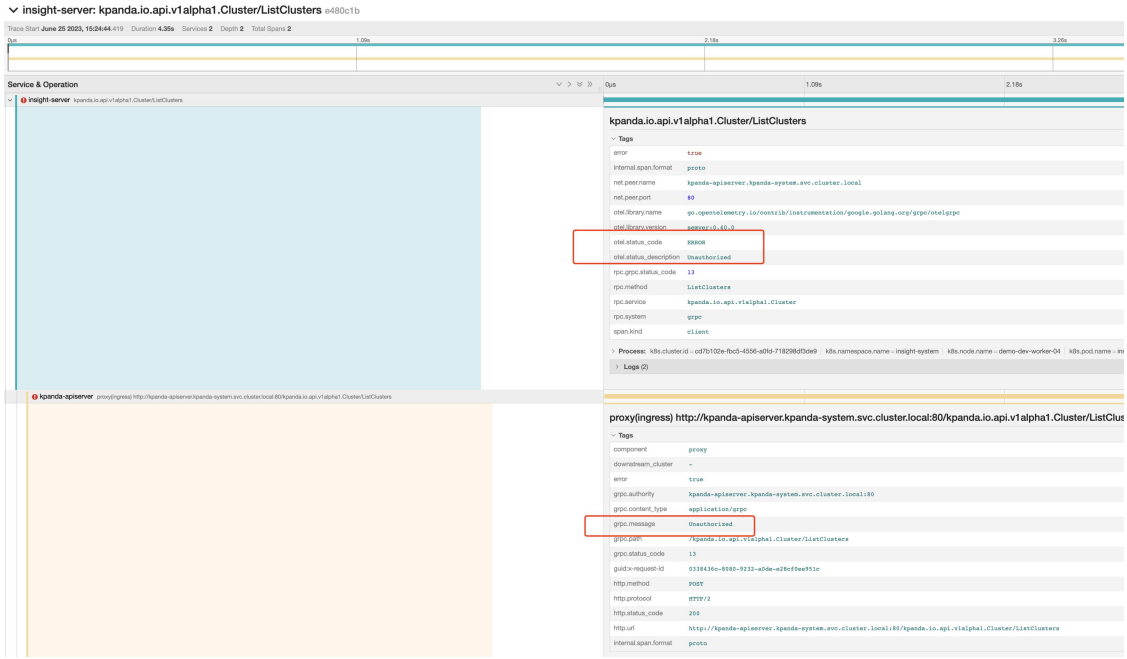


06

根据上面的分析方法，我们也可以定位到其他 操作 错误的链路：




insight-server: kpanda.io.api.v1alpha1.Cluster/ListClusters e480c1b
Trace Start June 25 2023, 15:24:44.419 Duration 4.35s Services 2 Depth 2 Total Spans 2
0µs
1.09s
2.18s
Service & Operation
0µs
1.09s
insight-server kpanda.io.api.v1alpha1.Cluster/ListClusters
kpanda-apiserver proxy(ingress) http://kpanda-apiserver.kpanda-system.svc.cluster.local:80/kpanda.io.api.v1alpha1.Cluster/ListClusters


insight-server: kpanda.io.api.v1alpha1.Cluster/ListClusters e480c1b
Service & Operation
kpanda.io.api.v1alpha1.Cluster/ListClusters
Tags
error true
internal.span.format proto
net.peer.name kpanda-apiserver.kpanda-system.svc.cluster.local
net.peer.port 80
otel.library.name go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc
otel.library.version semver:0.40.0
otel.status_code ERROR
otel.status_description Unauthorized
rpc.grpc.status_code 13
rpc.method ListClusters
rpc.service kpanda.io.api.v1alpha1.Cluster
rpc.system grpc
span.kind client
Process:
Logs (2)
kpanda-apiserver
proxy(ingress) http://kpanda-apiserver.kpanda-system.svc.cluster.local:80/kpanda.io.api.v1alpha1.Cluster/ListClusters
Tags
component proxy
downstream_cluster -
error true
grpc.authority kpanda-apiserver.kpanda-system.svc.cluster.local:80
grpc.content_type application/grpc
grpc.message Unauthorized
grpc.path /kpanda.io.api.v1alpha1.Cluster/ListClusters
grpc.status_code 13
http.method POST
http.protocol HTTP/2
http.status_code 200
http.url http://kpanda-apiserver.kpanda-system.svc.cluster.local:80/kpanda.io.api.v1alpha1.Cluster/ListClusters
internal.span.format proto

接下来 —— 你来分析！

配置通知模板

模板语法（Go Template）说明

告警通知模板采用了 Go Template 语法来渲染模板。

模板会基于下面的数据进行渲染。

```
{
    "status": "firing",
    "labels": {
        "alertgroup": "test-group",                    // 告警策略名称
        "alertname": "test-rule",                  // 告警规则名称
        "cluster": "35b54a48-b66c-467b-a8dc-503c40826330",
        "customlabel1": "v1",
        "customlabel2": "v2",
        "endpoint": "https",
        "group_id": "01gypg06fcdf7rmqc4ksv97646",
        "instance": "10.6.152.85:6443",
        "job": "apiserver",
        "namespace": "default",
        "prometheus": "insight-system/insight-agent-kube-prometh-prometheus",
        "prometheus_replica": "prometheus-insight-agent-kube-prometh-prometh
eus-0",
        "rule_id": "01gypg06fcyn2g9zyehbrvcdfn",
        "service": "kubernetes",
        "severity": "critical",
        "target": "35b54a48-b66c-467b-a8dc-503c40826330",
        "target_type": "cluster"
    },
    "annotations": {
        "customanno1": "v1",
```

```
            "customanno2": "v2",
            "description": "这是一条测试规则，10.6.152.85:6443 down",
            "value": "1"
        },
        "startsAt": "2023-04-20T07:53:54.637363473Z",
        "endsAt": "0001-01-01T00:00:00Z",
        "generatorURL": "http://vmalert-insight-victoria-metrics-k8s-stack-df987997b-npsl9:8080/vmalert/alert?group_id=16797738747470868115&alert_id=10071735367745833597",
        "fingerprint": "25c8d93d5bf58ac4"
}
```


使用说明

1. . 字符

   在当前作用域下渲染指定对象。

   示例 1：取顶级作用域下的所有内容，即示例代码中上下文数据的全部内容。

   ```
   {{ . }}
   ```

2. 判断语句 if / else

   使用 if 检查数据，如果不满足可以执行 else。

   ```
   {{if .Labels.namespace }}命名空间: {{ .Labels.namespace }} \n{{ end }}
   ```

3. 循环函数 for

   for 函数用于重复执行代码内容。

   示例 1：遍历 labels 列表，获取告警的所有 label 内容。

   ```
   {{ for .Labels}} \n {{end}}
   ```

阈值模板说明

Insight 内置 Webhook 告警模板如下，其他如邮件、企业微信等内容相同，只是对换行进行相应调整。

```
规则名称: {{ .Labels.alertname }} \n
策略名称: {{ .Labels.alertgroup }} \n
告警级别: {{ .Labels.severity }} \n
集群: {{ .Labels.cluster }} \n
{{if .Labels.namespace }}命名空间: {{ .Labels.namespace }} \n{{ end }}
{{if .Labels.node }}节点: {{ .Labels.node }} \n{{ end }}
资源类型: {{ .Labels.target_type }} \n
{{if .Labels.target }}资源名称: {{ .Labels.target }} \n{{ end }}
触发值: {{ .Annotations.value }} \n
发生时间: {{ .StartsAt }} \n
{{if ne "0001-01-01T00:00:00Z" .EndsAt }}结束时间: {{ .EndsAt }} \n{{ end }}
描述: {{ .Annotations.description }} \n
```

邮箱主题参数

由于 Insight 在发送告警消息时，会对同一时间同一条规则产生的消息进行合并发送， 所以 email 主题不同于上面四种模板，只会使用告警消息中的 commonLabels 内容对模板进行渲染。默认模板如下：

```
[{{ .status }}] [{{ .severity }}] 告警: {{ .alertname }}
```

其他可作为邮箱主题的字段如下：

```
{{ .status }} 告警消息的触发状态
{{ .alertgroup }} 告警所属的策略名称
{{ .alertname }} 告警所属的规则名称
{{ .severity }} 告警级别
{{ .target_type }} 告警资源类型
{{ .target }} 告警资源对象
{{ .规则其他自定义 label key }}
```

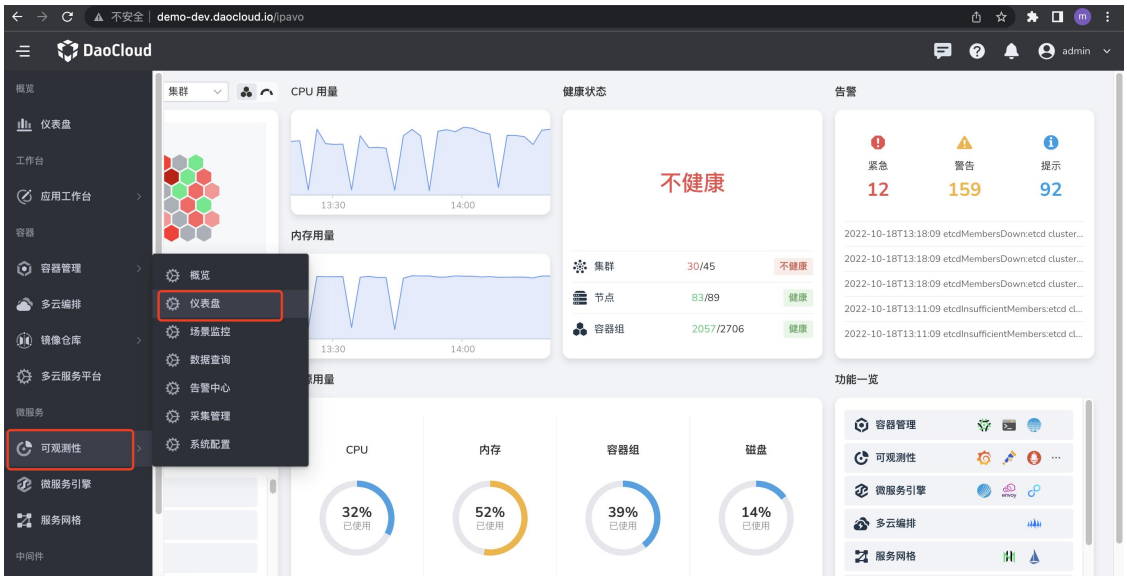
链路采集排障指南

在尝试排查链路数据采集的问题前，需先理解链路数据的传输路径，下面是链路数据传输示意图：

```
graph TB

sdk[Language proble / SDK] --> workload[Workload cluster otel collector]
--> otel[Global cluster otel collector]
--> jaeger[Global cluster jaeger collector]
--> es[Elasticsearch cluster]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class sdk,workload,otel,jaeger,es cluster
```

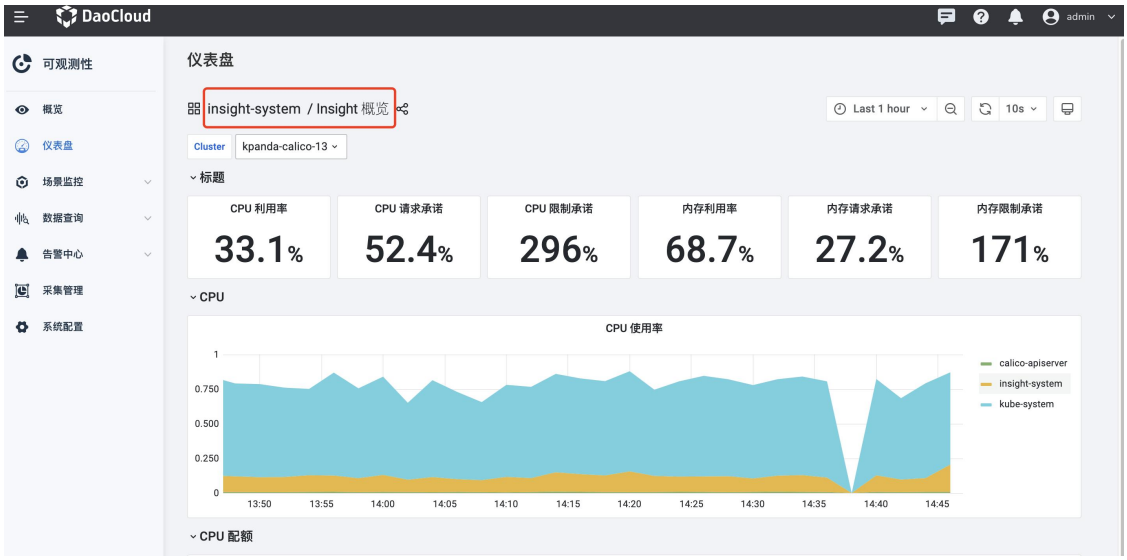
如上图所示，在任一步骤传输失败都会导致无法查询出链路数据。如果您在完成应用链路增强后发现没有链路数据，请执行以下步骤：

1. 使用 DCE 5.0 平台，进入可观测性，选择左侧导航栏的 仪表盘。



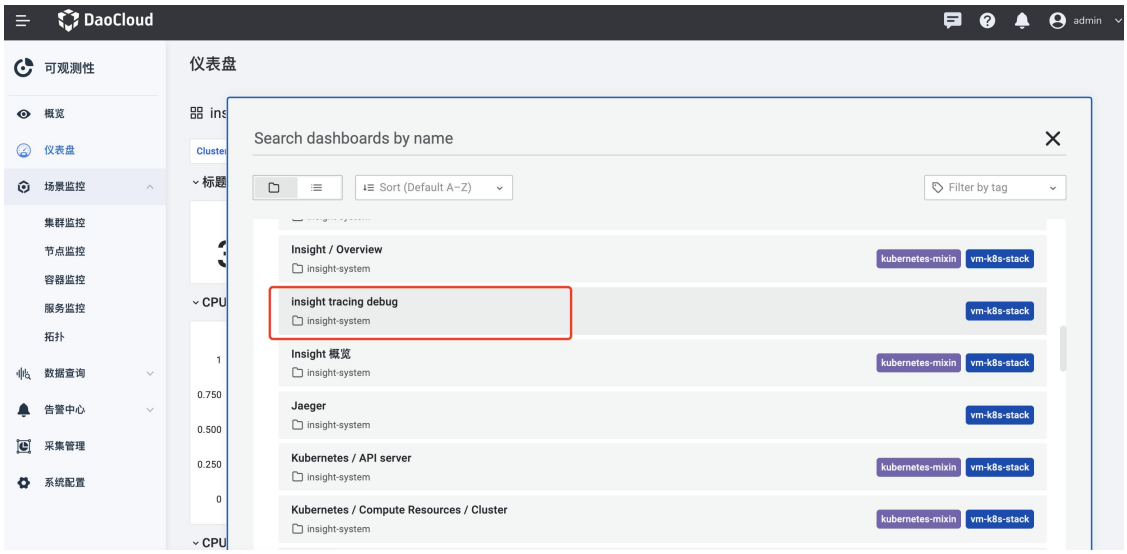
insight 入口

2. 点击仪表盘标题概览。



概览

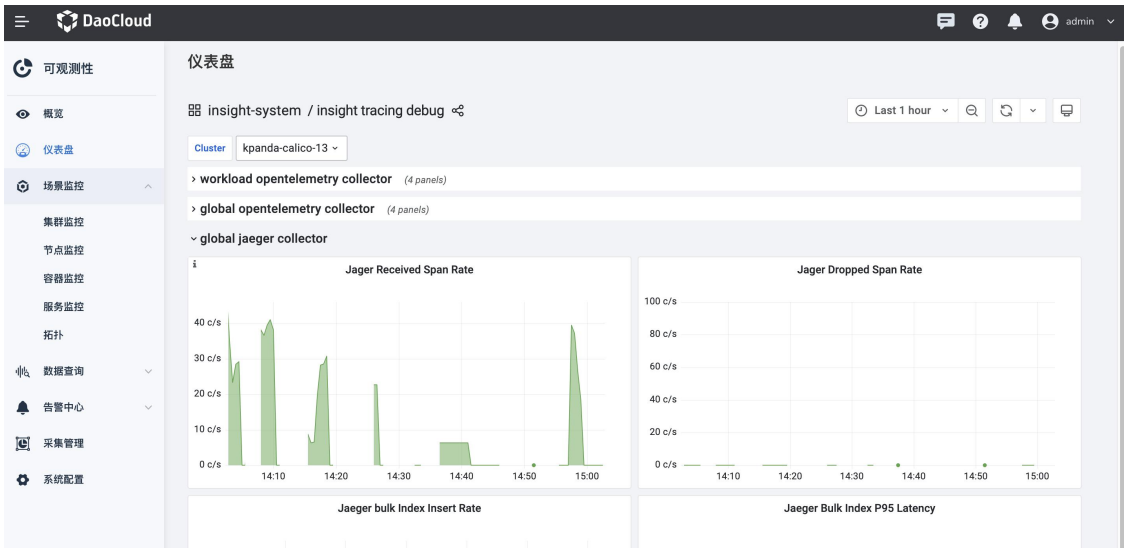
3. 切换到 insight-system -> insight tracing debug 仪表盘。



tracing debug

4. 可以看到该仪表盘由三个区块组成，分别负责监控不同集群、不同组件传输链路的数据情况。通过生成的时序图表，检查链路数据传输是否存在问题。

   - workload opentelemetry collector
   - global opentelemetry collector
   - global jaeger collector

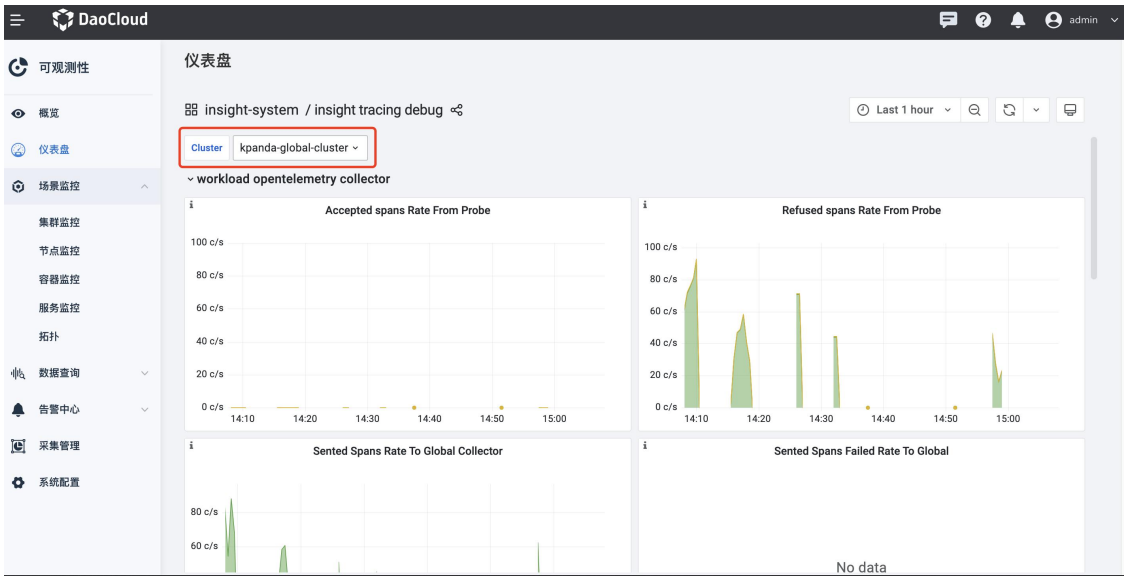


tracing debug

区块介绍

1. **workload opentelemetry collector**

   展示不同工作集群的 opentelemetry collector 在接受 language probe/SDK 链路数据，发送聚合链路数据情况。可以通过左上角的 Cluster 选择框选择所在的集群。



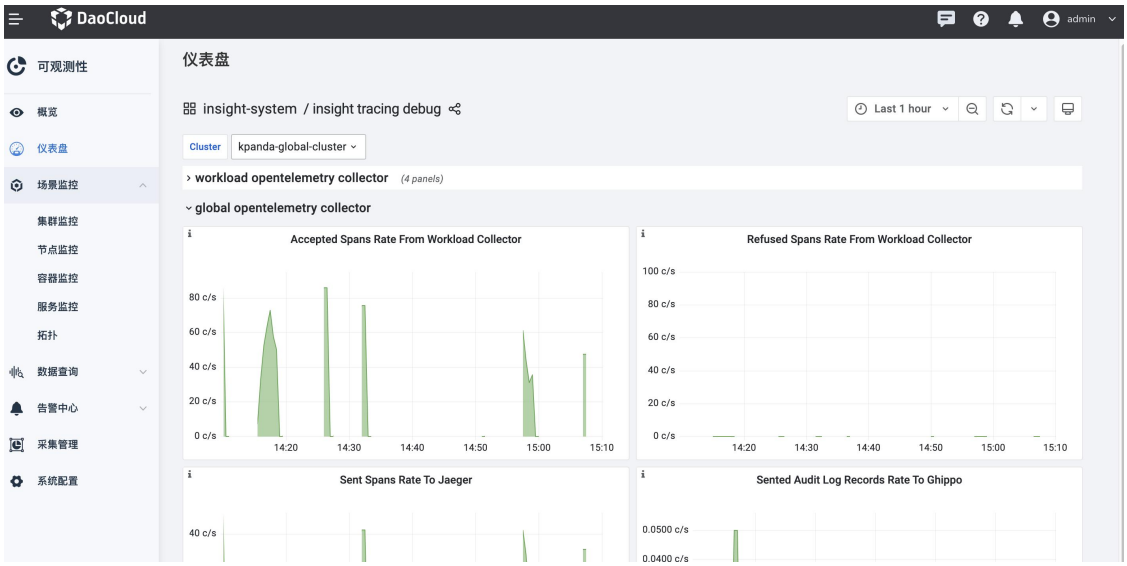
   tracing debug

   !!! note

   根据这四张时序图，可以判断出该集群的 `opentelemetry collector` 是否正常运行。

2. **global opentelemetry collector**

   展示全局服务集群的 opentelemetry collector 在接收工作集群中 otel collector 链路数据以及发送聚合链路数据的情况。



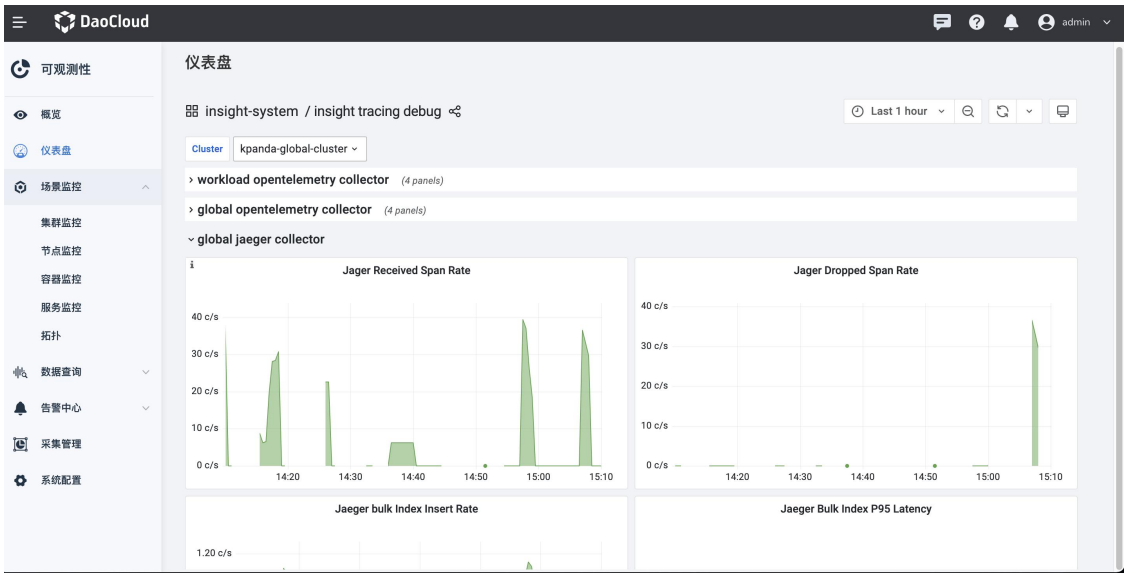
tracing debug

!!! note

`全局管理集群`的 `opentelemetry collector` 还负责发送所有工作集群的[全局管理模块](../../ghippo/intro/index.md)的[审计日志](../../ghippo/user-guide/audit/audit-log.md)以及 Kubernetes 审计日志（默认不采集）到全局管理模块的 `audit server` 组件。

3. **global jaeger collector**

展示全局管理集群的 jaeger collector 在接收全局管理集群中 otel collector 的数据，并发送链路数据到 ElasticSearch 集群的情况。



tracing debug

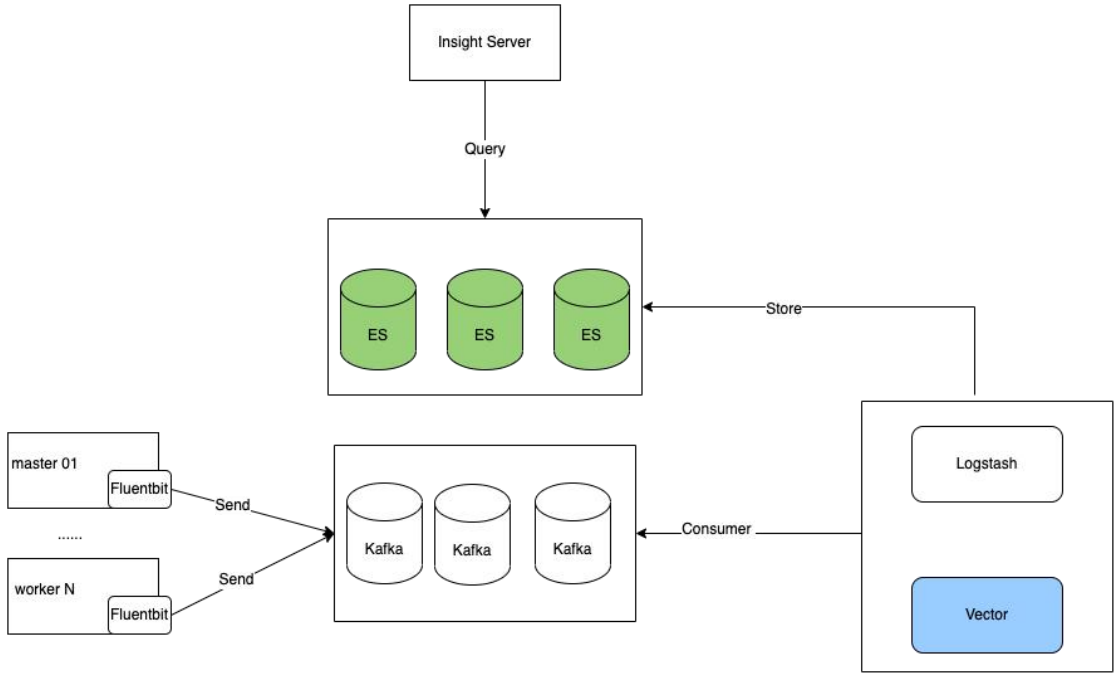
Kafka + Elasticsearch 流式架构应对超大规模日志方案

随着业务发展，越来越多的应用产生的日志数据会越来越多，为了保证系统能够正常采集并分析庞杂的日志数据时， 一般做法是引入 Kafka 的流式架构来解决大量数据异步采集的方案。采集到的日志数据会经过 Kafka 流转， 由相应的数据消费组件将数据从 Kafka 消费存入到 Elasticsearch 中，并通过 Insight 进行可视化展示与分析。

本文将介绍以下两种方案：

- Fluentbit + Kafka + Logstash + Elasticsearch
- Fluentbit + Kafka + Vector + Elasticsearch

当我们在日志系统中引入 Kafka 之后，数据流图如下图所示：



logging-kafka

上面两种方案中有共通的地方，不同之处在于消费 Kafka 数据的组件，同时，为了不影响 Insight 数据分析， 我们需要在消费 Kafka 数据并写入到 ES 的数据和原来 Fluentbit 直接写入 ES 的数据的格式一致。

首先我们来看看 Fluentbit 怎么将日志写入 Kafka：

修改 Fluentbit Output 配置

当 Kafka 集群准备就绪之后，我们需要修改 insihgt-system 命名空间下 ConfigMap 的内容， 新增以下三个 Kafka Output 并注释原来三个 Elasticsearch Output：

假设 Kafka Brokers 地址为： insight-kafka.insight-system.svc.cluster.local:9092

```
[OUTPUT]
    Name            kafka
    Match_Regex  (?:kube|syslog)\.(.*)
    Brokers         insight-kafka.insight-system.svc.cluster.local:9092
    Topics          insight-logs
    format          json
    timestamp_key  @timestamp
    rdkafka.batch.size  65536
    rdkafka.compression.level  6
    rdkafka.compression.type  lz4
    rdkafka.linger.ms  0
    rdkafka.log.connection.close  false
    rdkafka.message.max.bytes  2.097152e+06
    rdkafka.request.required.acks  1
[OUTPUT]
    Name            kafka
    Match_Regex  (?:skoala-gw)\.(.*)
    Brokers         insight-kafka.insight-system.svc.cluster.local:9092
    Topics          insight-gw-skoala
    format          json
```

```
    timestamp_key @timestamp
    rdkafka.batch.size 65536
    rdkafka.compression.level 6
    rdkafka.compression.type lz4
    rdkafka.linger.ms 0
    rdkafka.log.connection.close false
    rdkafka.message.max.bytes 2.097152e+06
    rdkafka.request.required.acks 1
[OUTPUT]
    Name kafka
    Match_Regex (?:kubeevent)\.(.*)
    Brokers insight-kafka.insight-system.svc.cluster.local:9092
    Topics insight-event
    format json
    timestamp_key @timestamp
    rdkafka.batch.size 65536
    rdkafka.compression.level 6
    rdkafka.compression.type lz4
    rdkafka.linger.ms 0
    rdkafka.log.connection.close false
    rdkafka.message.max.bytes 2.097152e+06
    rdkafka.request.required.acks 1
```

接下来就是消费 Kafka 数据之后写到 ES 的细微差别。 正如本文开始的描述，本文将介绍 Logstash 与 Vector 作为消费 Kafka 的两种方式。

消费 Kafka 并写入 Elasticsearch

假设 Elasticsearch 的地址为：https://mcamel-common-es-cluster-es-http.mcamel-system:9200

通过 Logstash 消费

如果你对 Logstash 技术栈比较熟悉，你可以继续使用该方式。

当你通过 Helm 部署 Logstash 的时候，增加如下 Pipeline 即可：

```
# Allows you to add any config files in /usr/share/logstash/config/
# such as logstash.yml and log4j2.properties
#
# Note that when overriding logstash.yml, `http.host: 0.0.0.0` should always be i
ncluded
# to make default probes work.
logstashConfig:
  logstash.yml: |
    http.host: 0.0.0.0
    xpack.monitoring.enabled: false
logstashPipeline:
  insight-logs.conf: |
    input {
      kafka {
        topics_pattern => "insight-logs"              # 也可以模糊匹配 如:all-log.*
        bootstrap_servers => "insight-kafka.insight-system.svc.cluster.local:9092"
   # kafka 的 ip 端口
        enable_auto_commit => true
        #codec => json                                  # 数据格式
        consumer_threads => 1                          # 对应 partition 的数量
        decorate_events => true
        #auto_offset_rest => "latest"                  # 默认值就是这个
        #group_id => "all-logs-group"                  # kafka 的消费组
        codec => "plain"
      }
    }

    filter {
      mutate { gsub => [ "message", "@timestamp", "_@timestamp"] }
      json {source => "message"}
```

```
        date {
            match => [ "_@timestamp", "UNIX" ]
            remove_field => "_@timestamp"
            remove_tag => "_timestampparsefailure"
        }
        mutate {
            remove_field => ["event", "message"]
        }
    }
    output {
        elasticsearch {
          hosts => ["https://mcamel-common-es-cluster-es-http.mcamel-system:
9200"]
          user => 'elastic'
          ssl => 'true'
          password => 'XAlJ948ZY0leE320SQ6hfv17'
          ssl_certificate_verification => 'false'
          index => "insight-es-k8s-logs-alias"
        }
    }


  insight-event.conf: |
    input {
        kafka {
          topics_pattern => "insight-event"             # 也可以模糊匹配 如:all-log.*
          bootstrap_servers => "insight-kafka.insight-system.svc.cluster.local:9092"
   # kafka 的 ip 端口
          enable_auto_commit => true
          #codec => json                                  # 数据格式
          consumer_threads => 1                          # 对应 partition 的数量
          decorate_events => true
          #auto_offset_rest => "latest"                  # 默认值就是这个
```

```
        #group_id => "all-logs-group"                    # kafka 的消费组
        codec => "plain"
      }
    }

    filter {
      mutate { gsub => [ "message", "@timestamp", "_@timestamp"] }
      json {source => "message"}
      date {
          match => [ "_@timestamp", "UNIX" ]
          remove_field => "_@timestamp"
          remove_tag => "_timestampparsefailure"
      }
      mutate {
          remove_field => ["event", "message"]
      }
    }
    output {
      elasticsearch {
        hosts => ["https://mcamel-common-es-cluster-es-http.mcamel-system:
9200"]
        user => 'elastic'
        ssl => 'true'
        password => 'XAlJ948ZY0leE320SQ6hfv17'
        ssl_certificate_verification => 'false'
        index => "insight-es-k8s-event-logs-alias"
      }
    }

  insight-gw-skoala.conf: |
    input {
      kafka {
```

```
        topics_pattern => "insight-gw-skoala"              # 也可以模糊匹配 如:all-lo
g.*
        bootstrap_servers => "insight-kafka.insight-system.svc.cluster.local:9092"
    # kafka 的 ip 端口
        enable_auto_commit => true
        #codec => json                                      # 数据格式
        consumer_threads => 1                              # 对应 partition 的数量
        decorate_events => true
        #auto_offset_rest => "latest"                      # 默认值就是这个
        #group_id => "all-logs-group"                      # kafka 的消费组
        codec => "plain"
      }
    }

    filter {
      mutate { gsub => [ "message", "@timestamp", "_@timestamp"] }
      json {source => "message"}
      date {
          match => [ "_@timestamp", "UNIX" ]
          remove_field => "_@timestamp"
          remove_tag => "_timestampparsefailure"
      }
      mutate {
          remove_field => ["event", "message"]
      }
    }
    output {
      elasticsearch {
        hosts => ["https://mcamel-common-es-cluster-es-http.mcamel-system:
9200"]
        user => 'elastic'
        ssl => 'true'
```

```
        password => 'XAlJ948ZY0leE320SQ6hfv17'
        ssl_certificate_verification => 'false'
        index => "skoala-gw-alias"
      }
    }
```

通过 Vector 消费

如果你对 Vector 技术栈比较熟悉，你可以继续使用该方式。

当你通过 Helm 部署 Vector 的时候，引用如下规则的 Configmap 配置文件即可：

```
metadata:
  name: vector
apiVersion: v1
data:
  aggregator.yaml: |
    api:
      enabled: true
      address: '0.0.0.0:8686'
    sources:
      insight_logs_kafka:
        type: kafka
        bootstrap_servers: 'insight-kafka.insight-system.svc.cluster.local:9092'
        group_id: consumer-group-insight
        topics:
          - insight-logs
      insight_event_kafka:
        type: kafka
        bootstrap_servers: 'insight-kafka.insight-system.svc.cluster.local:9092'
        group_id: consumer-group-insight
        topics:
          - insight-event
```

```
  insight_gw_skoala_kafka:
    type: kafka
    bootstrap_servers: 'insight-kafka.insight-system.svc.cluster.local:9092'
    group_id: consumer-group-insight
    topics:
      - insight-gw-skoala
transforms:
  insight_logs_remap:
    type: remap
    inputs:
      - insight_logs_kafka
    source: |2
          . = parse_json!(string!(.message))
          .@timestamp = now()
  insight_event_kafka_remap:
    type: remap
    inputs:
      - insight_event_kafka
      - insight_gw_skoala_kafka
    source: |2
          . = parse_json!(string!(.message))
          .@timestamp = now()
  insight_gw_skoala_kafka_remap:
    type: remap
    inputs:
      - insight_gw_skoala_kafka
    source: |2
          . = parse_json!(string!(.message))
          .@timestamp = now()
sinks:
  insight_es_logs:
    type: elasticsearch
```

```
    inputs:
      - insight_logs_remap
    api_version: auto
    auth:
      strategy: basic
      user: elastic
      password: 8QZJ656ax3TXZqQh205l3Ee0
    bulk:
      index: insight-es-k8s-logs-alias-1418
    endpoints:
      - 'https://mcamel-common-es-cluster-es-http.mcamel-system:9200'
    tls:
      verify_certificate: false
      verify_hostname: false
  insight_es_event:
    type: elasticsearch
    inputs:
      - insight_event_kafka_remap
    api_version: auto
    auth:
      strategy: basic
      user: elastic
      password: 8QZJ656ax3TXZqQh205l3Ee0
    bulk:
      index: insight-es-k8s-event-logs-alias-1418
    endpoints:
      - 'https://mcamel-common-es-cluster-es-http.mcamel-system:9200'
    tls:
      verify_certificate: false
      verify_hostname: false
  insight_es_gw_skoala:
    type: elasticsearch
```

```
    inputs:
      - insight_gw_skoala_kafka_remap
    api_version: auto
    auth:
      strategy: basic
      user: elastic
      password: 8QZJ656ax3TXZqQh205l3Ee0
    bulk:
      index: skoala-gw-alias-1418
    endpoints:
      - 'https://mcamel-common-es-cluster-es-http.mcamel-system:9200'
    tls:
      verify_certificate: false
      verify_hostname: false
```

检查是否正常工作

你可以通过查看 Insight 日志查询界面是否有最新的数据，或者查看原本 Elasticsearch 的索引的数量有没有增长，增长即代表配置成功。

参考

- Bitnami Logstash Helm Chart
- Vector Helm Chart
- Vector 实践
- Vector Perfomance

使用 OpenTelemetry 零代码接收 SkyWalking 链路数据

可观测性 Insight 通过 OpenTelemetry 将应用数据进行上报。若您的应用已使用 Skywalking 来采集链路，可参考本文进行零代码改造将链路数据接入 Insight 。

代码解读

为了能兼容不同的分布式追踪实现，OpenTelemetry 提供了组件植入的方式，让不同的厂商能够经由 OpenTelemetry 标准化数据处理后输出到不同的后端。Jaeger 与 Zipkin 在社区中实现了 JaegerReceiver、ZipkinReceiver。我们也为社区贡献了 SkyWalkingReceiver，并进行了持续的打磨，现在已经具备了在生产环境中使用的条件，而且无需修改任何一行业务代码。

OpenTelemetry 与 SkyWalking 有一些共同点：都是使用 Trace 来定义一次追踪，并使用 Span 来标记追踪里的最小粒度。但是在一些细节和实现上还是会有差别：

–	Skywalking	OpenTelemetry
数据结构	span –> Segment –> Trace	Span –> Trace
属性信息	Tags	Attributes
应用时间	Logs	Events
引用关系	References	Links

明确了这些差异后，就可以开始实现将 SkyWalking Trace 转换为 OpenTelemetry Trace。主要工作包括：

1. 如何构造 OpenTelemetry 的 TraceId 和 SpanId
2. 如何构造 OpenTelemetry 的 ParentSpanId

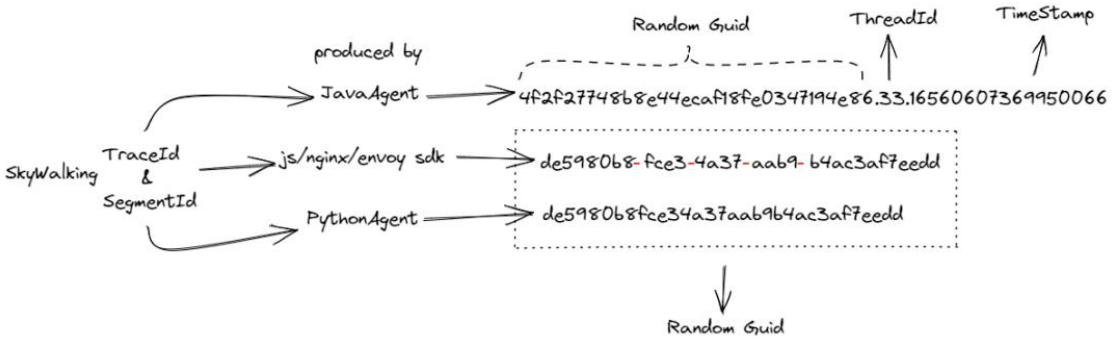
3. 如何在 OpenTelemetry Span 中保留 SkyWalking 的原始 TraceId、SegmentId、SpanId

首先，我们来看如何构造 OpenTelemetry 的 TraceId 和 SpanId。SkyWalking 和 OpenTelemetry 都是通过 TraceId 串联起各个分布式服务调用，并通过 SpanId 来标记每一个 Span，但是实现规格有较大差异：

代码实现见 GitHub：

1. https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/receiver/skywalkingreceiver/skywalkingproto_to_traces.go#L54
2. https://github.com/open-telemetry/opentelemetry-collector-contrib/pull/8107
3. https://github.com/open-telemetry/opentelemetry-collector-contrib/pull/8549

具体来讲，SkyWalking TraceId 和 SegmentId 所有可能的格式如下：



sw2otel-01

其中，在 OpenTelemetry 协议里，Span 在所有 Trace 中都是唯一的，而在 SkyWalking 中，Span 仅在每个 Segment 里是唯一的，这说明要通过 SegmentId 与 SpanId 结合才能在 SkyWalking 中对 Span 做唯一标识，并转换为 OpenTelemetry 的 SpanId。

代码实现见 GitHub：

1. https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/receiver/skywalkingreceiver/skywalkingproto_to_traces.go#L272
2. https://github.com/open-telemetry/opentelemetry-collector-contrib/pull/11562

接下来，我们来看如何构造 OpenTelemetry 的 ParentSpanId。在一个 Segment 内部，SkyWalking 的 ParentSpanId 字段可直接用于构造 OpenTelemetry 的 ParentSpanId 字段。但当一个 Trace 跨多个 Segment 时，SkyWalking 是通过 Reference 中的 ParentTraceSegmentId 和 ParentSpanId 表示的关联信息，于是此时需要通过 Reference 中的信息构建 OpenTelemetry 的 ParentSpanId 。

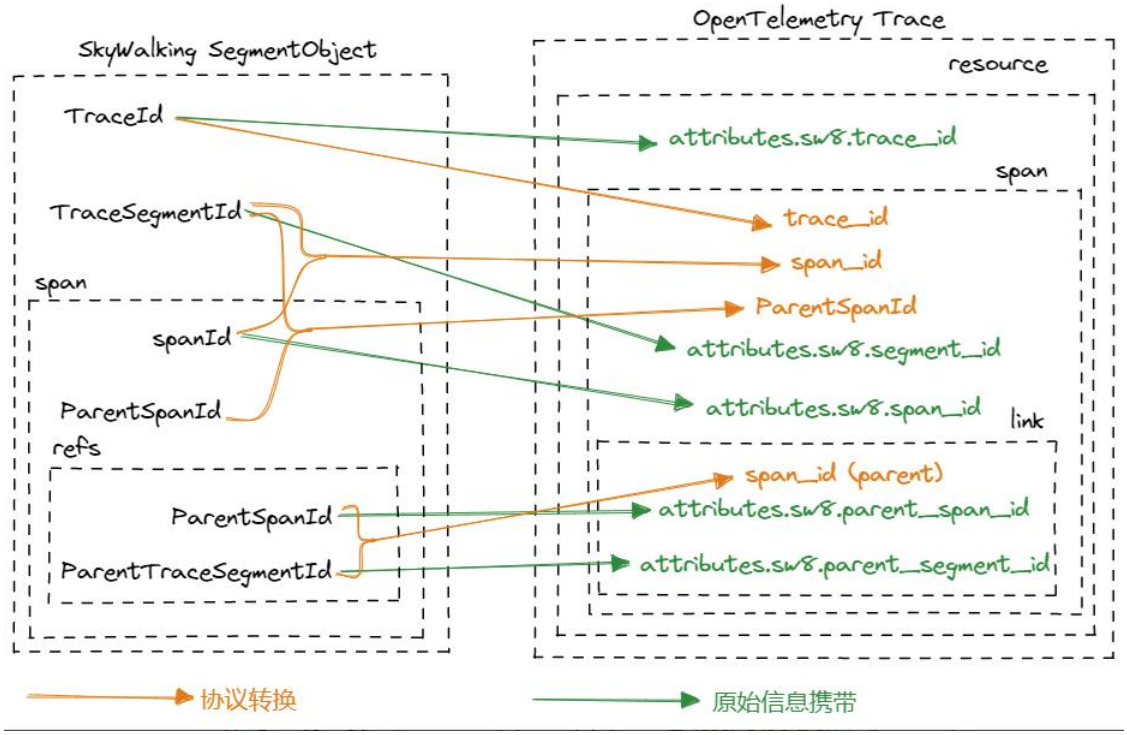
> 代码实现见 GitHub：https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/receiver/skywalkingreceiver/skywalkingproto_to_traces.go#L173

最后，我们来看如何在 OpenTelemetry Span 中保留 SkyWalking 的原始 TraceId、SegmentId、SpanId。我们携带这些原始信息是为了能将分布式追踪后端展现的 OpenTelemetry TraceId、SpanId 与应用程序日志中的 SkyWalking TraceId、SegmentId、SpanId 进行关联，打通追踪和日志。我们选择将 SkyWalking 中原有的 TraceId、SegmentId、ParentSegmentId 携带到 OpenTelemetry Attributes 中。

> 代码实现见 GitHub：

1. https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/receiver/skywalkingreceiver/skywalkingproto_to_traces.go#L201
2. https://github.com/open-telemetry/opentelemetry-collector-contrib/pull/12651

经过上述一系列转换后，我们将 SkyWalking Segment Object 完整的转换为了 OpenTelmetry Trace，如下图：



sw2otel-02

部署 Demo

下面我们以一个 Demo 来展示使用 OpenTelemetry 收集、展示 SkyWalking 追踪数据的完整过程。

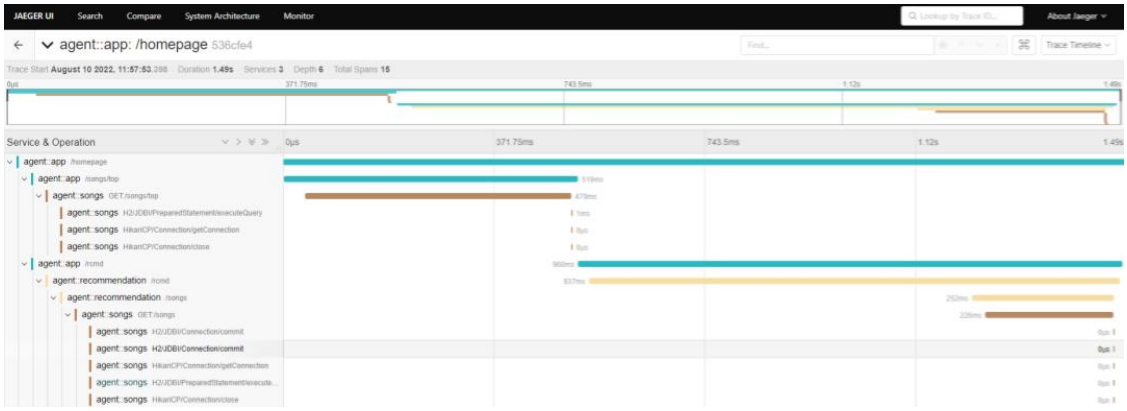
首先，在部署 OpenTelemetry Agent 之后，开启如下配置，即可在 OpenTelemetry 中拥有兼容 SkyWalking 协议的能力：

```
# otel-agent config
receivers:
  # add the following config
  skywalking:
    protocols:
      grpc:
        endpoint: 0.0.0.0:11800 # 接收 SkyWalking Agent 上报的 Trace 数据
```

```
        http:
          endpoint: 0.0.0.0:12800 # 接收从前端/ nginx 等 HTTP 协议上报的 Trace
数据
service:
  pipelines:
    traces:
      # add receiver `skywalking`
      receivers: [skywalking]

# otel-agent service yaml
spec:
  ports:
    - name: sw-http
      port: 12800
      protocol: TCP
      targetPort: 12800
    - name: sw-grpc
      port: 11800
      protocol: TCP
      targetPort: 11800
```

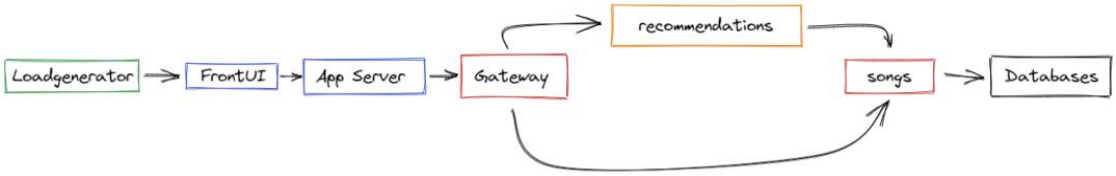
接下来需要将业务应用对接的 SkyWalking OAP Service（如 oap:11800）修改为 OpenTelemetry Agent Service（如 otel-agent:11800），就可以开始使用 OpenTelemetry 接收 SkyWalking 探针的追踪数据了。

我们以 SkyWalking-showcase Demo 为例展示整个效果。它使用 SkyWalking Agent 做追踪，通过 OpenTelemetry 标准化处理后使用 Jaeger 来呈现最终效果：



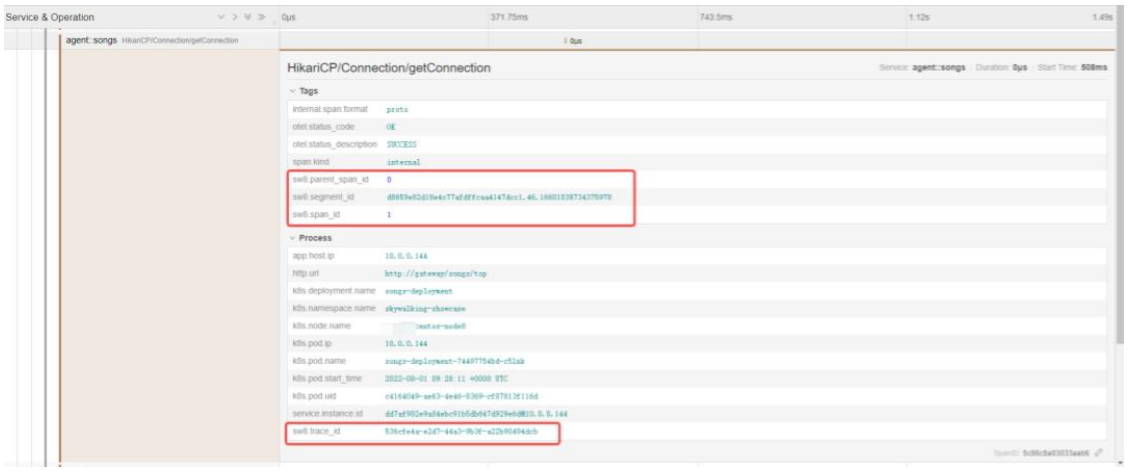
sw2otel-03

通过 SkyWalking Showcase 的架构图，可知 SkyWalking 的数据经过 OpenTelemetry 标准化后，依然完整。在这个 Trace 里，请求从 app/homepage 发起，之后在 app 同时发起两个请求 /rcmd/与/songs/top，分发到 recommandation/songs 两个服务中，并最终到达数据库进行查询，从而完成整个请求链路。



sw2otel-04

另外，我们也可从 Jaeger 页面中查看到原始 SkyWalking Id 信息，便于与应用日志关联：

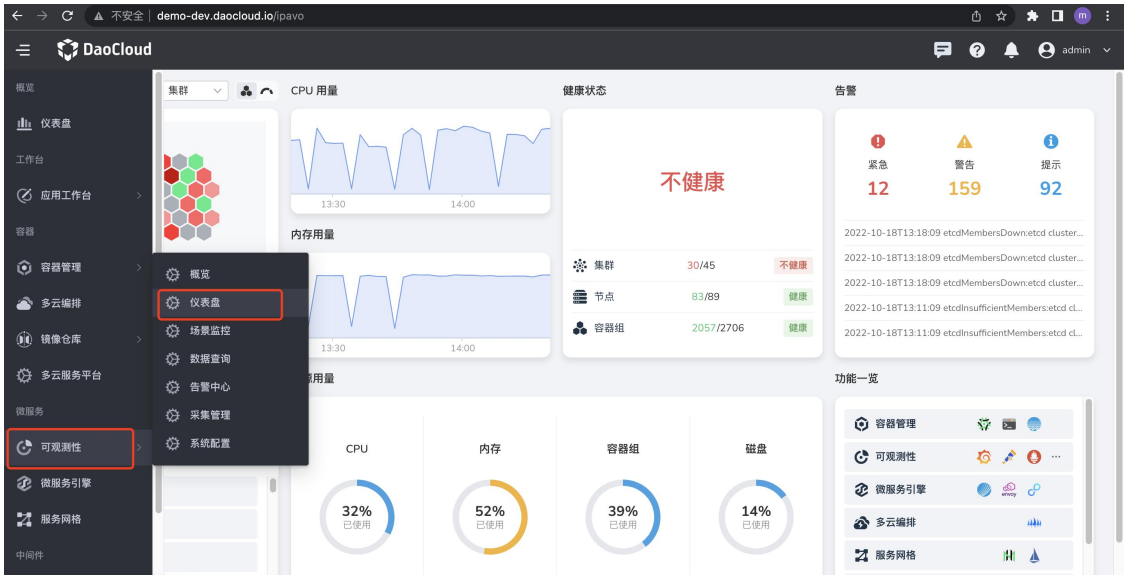

Service & Operation
0µs
371.75ms
743.5ms
1.12s
1.49s
agent::songs HikariCP/Connection/getConnection
HikariCP/Connection/getConnection
Service: agent::songs Duration: 0µs Start Time: 508ms
Tags
internal.span.format proto
otel.status_code OK
otel.status_description SUCCESS
span.kind internal
sw8.parent_span_id 0
sw8.segment_id
sw8.span_id 1
Process
app.host.ip 10.0.0.144
http.url http://gateway/songs/top
k8s.deployment.name songs-deployment
k8s.namespace.name skywalking-showcase
k8s.node.name
k8s.pod.ip 10.0.0.144
k8s.pod.name songs-deployment-74497754bd-c51xk
k8s.pod.start_time 2022-08-01 09:28:11 +0000 UTC
k8s.pod.uid
service.instance.id
sw8.trace_id

sw2otel-05

日志采集排障指南

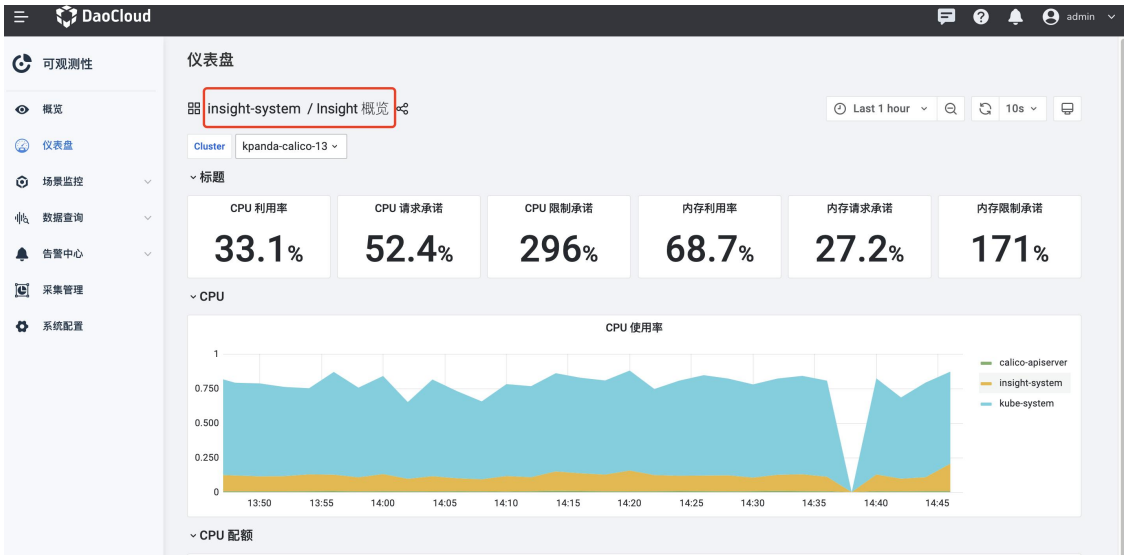
在集群中安装 insight-agent 后，insight-agent 中的 Fluent Bit 会默认采集集群中的日志，包括 Kubernetes 事件日志、节点日志、容器日志等。 Fluent Bit 已配置好各种日志采集插件、相关的过滤器插件及日志输出插件。 这些插件的工作状态决定了日志采集是否正常。 下面是一个针对 Fluent Bit 的仪表盘，它用来监控各个集群中 Fluent Bit 的工作情况和插件的采集、处理、导出日志的情况。

1. 使用 DCE 5.0 平台，进入可观测性，选择左侧导航栏的 仪表盘。



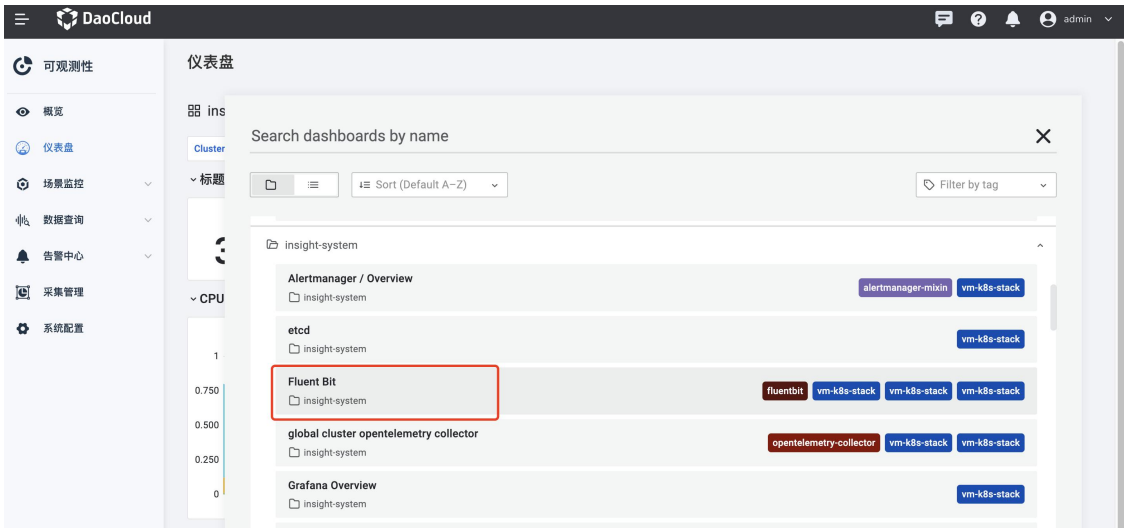
insight 入口

2. 点击仪表盘标题概览。



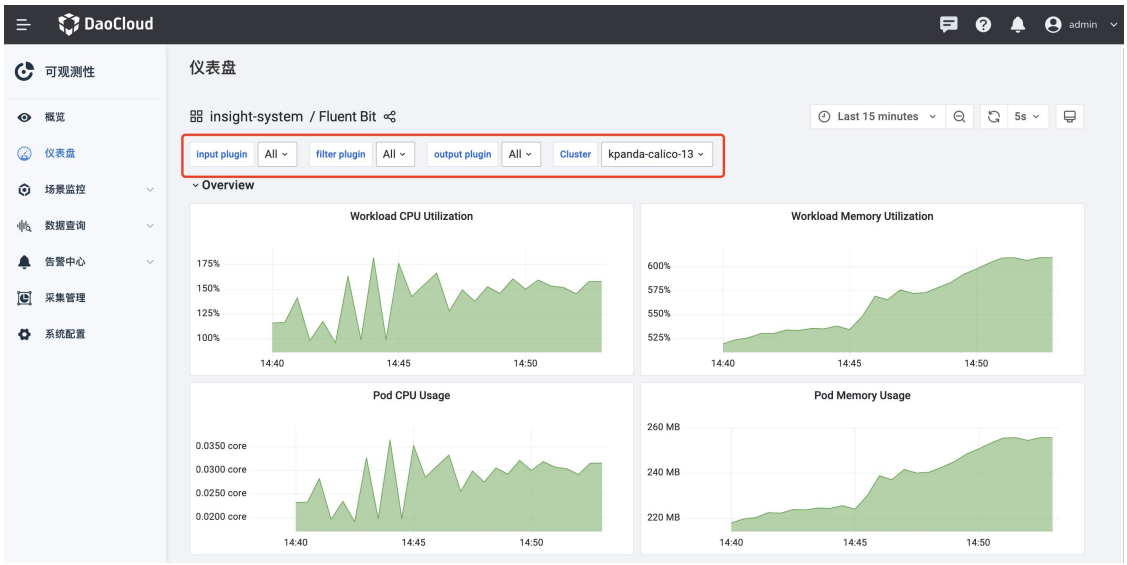
概览

3. 切换到 insight-system -> Fluent Bit 仪表盘。



fluentbit

4. Fluent Bit 仪表盘上方有几个选项框，可以选择日志采集插件、日志过滤插件、日志输出插件及所在集群名。



fluentbit

插件说明

此处说明 Fluent Bit 的几个插件。

日志采集插件

input plugin	插件介绍	采集目录
tail.kube	采集容器日志	/var/log/containers/*.log
tail.kubeevent	采集 Kubernetes 事件日志	/var/log/containers/*-kubernetes-event-exporter*.log
tail.syslog.dmesg	采集主机 dmesg 日志	/var/log/dmesg
tail.syslog.messages	采集主机常用日志	/var/log/secure, /var/log/messages, /var/log/syslog,/var/log/auth.log
syslog.syslog.	采集	

input plugin	插件介绍	采集目录
RSyslog	RSyslog 日志	
systemd.syslog.systemd	采集 Journald daemon 日志	
tail.audit_log.k8s	采集 Kubernetes 审计日志	/var/log//*audit*/.log
tail.audit_log.ghippo	采集全局管理审计日志	/var/log/containers/*_ghippo-system_audit-log*.log
tail.skoala-gw	采集微服务网关日志	/var/log/containers/*_skoala-gw*.log

日志过滤插件

filter plugin	插件介绍
Lua.audit_log.k8s	使用 lua 过滤符合条件的 Kubernetes 审计日志

!!! note

过滤器插件不止 Lua.audit_log.k8s，这里只介绍会丢弃日志的过滤器。

日志输出插件

output plugin	插件介绍
es.kube.kubeevent.syslog	把 Kubernetes 审计日志、事件日志，syslog 日志写入 ElasticSearch 集群
forward.audit_log	把 Kubernetes 审计日志和全局管理的审计日志发送到全局管理

output plugin	插件介绍
es.skoala	把微服务网关的请求日志和实例日志写入到 ElasticSearch 集群

通过 Sidecar 采集容器日志

Tailing Sidecar 是一个流式 Sidecar 容器， 是 Kubernetes 集群级的日志代理。Tailing Sidercar 可以在容器无法写入标准输出或标准错误流时，无需更改，即可自动收取和汇总容器内日志文件。

Insight 支持通过 Sidercar 模式采集日志，即在每个 Pod 中运行一个 Sidecar 容器将日志数据输出到标准输出流，以便 FluentBit 收集容器日志。

Insight Agent 中默认安装了 tailing-sidecar operator。 若您想开启采集容器内文件日志，请通过给 Pod 添加注解进行标记，tailing-sidecar operator 将自动注入 Tailing Sidecar 容器， 被注入的 Sidecar 容器读取业务容器内的文件， 并输出到标准输出流。

具体操作步骤如下：

1. 修改 Pod 的 YAML 文件，在 annotation 字段增加如下参数：

    ```
    metadata:
      annotations:
        tailing-sidecar:  <sidecar-name-0>:<volume-name-0>:<path-to-tail-0>;<sidecar-name-1>:<volume-name-1>:<path-to-tail-1>
    ```

    字段说明：

    - sidecar-name-0：tailing sidecar 容器名称（可选，如果未指定容器名称将自动创建，它将以“tailing-sidecar”前缀开头）
    - volume-name-0：存储卷名称；
    - path-to-tail-0：日志的文件路径

    !!! note

每个 Pod 可运行多个 Sidecar 容器，可以通过 `;` 隔离，实现不同 Sidecar 容器采集多个文件到多个存储卷。

2. 重启 Pod，待 Pod 状态变成运行中后，则可通过日志查询界面，查找该 Pod 的容器内日志。

功能优势

Hwameistor 云原生本地存储

IO 本地化

100% 本地吞吐，无网络开销，节点故障时 Pod 在副本节点启动，使用副本数据卷进行本地 IO 读写。

高性能、高可用性

- 100% IO 本地化，实现高性能本地吞吐
- 2 副本数据卷冗余备份，保障数据高可用

线性扩展

- 独立节点单元，最小 1 节点，扩展数量不限
- 控制平面、数据平面分离，节点扩展，不影响业务应用数据 I/O

CPU、内存开销小

IO 本地化，相同的 IO 读写，CPU 平稳，无较大波动，内存资源开销小

生产可运维

- 支持节点、磁盘、数据卷组（VG）维度迁移
- 支持换盘等运维行为

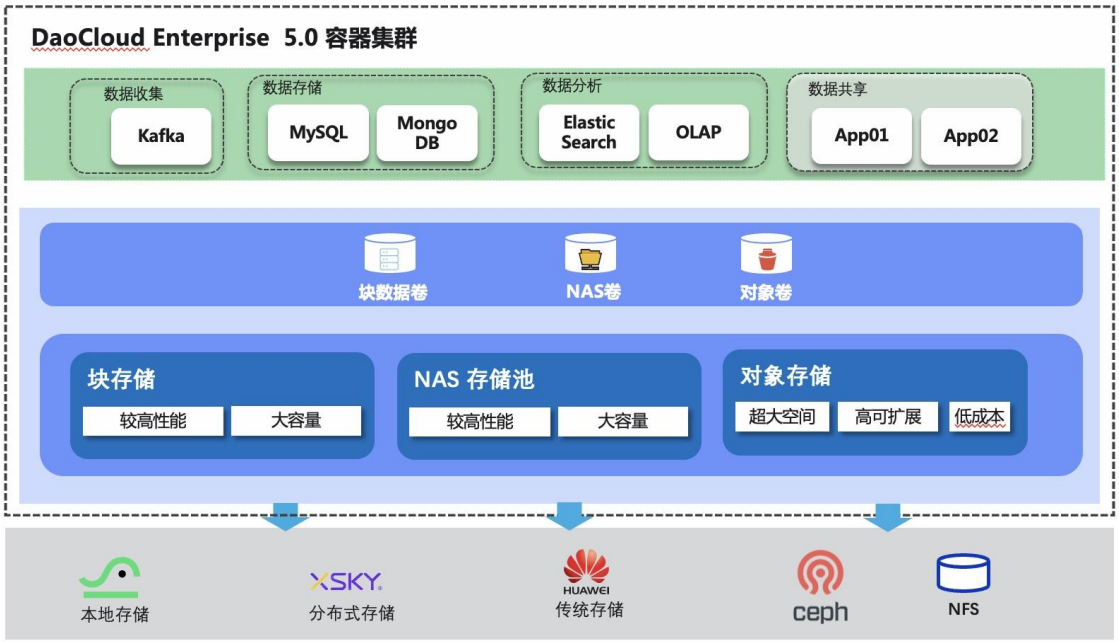
云原生存储

云原生化存储的几种类型

1. **传统存储云原生化**，通过 CSI 标准 同 Kubernetes 平台对接，此类型相对来说比较普遍， 用户可利用现有存储，并且基于传统存储的提供云原生存储能力稳定性好，SLA 强保障。
2. **软件定义存储云原生化**，软件定义存储，兼容传统应用和云原生应用。同样基于 CSI 标准同 Kubernetes 对接。软件定义存储通过网络使用企业中的每台机器上的磁盘空间， 并将这些分散的存储资源构成一个虚拟的存储设备，数据分散在不同的存储设备中。
3. **纯云原生存储**，此种类型的存储类型天然为云原生而生，构筑于云原生平台之上，能比较好的契合云原生特性， 并可随着应用 Pod 的迁移而迁移，具备如下特性：高可扩展性，高可用性，但相对于 通过 CSI 标准接入的传统存储可靠性低一些。

DCE 云原生存储

DCE 5.0 云原生存储基于 Kubernetes CSI 标准，可根据不同 SLA 要求及用户场景对接符合 CSI 标准的存储。 DaoCloud 推出的云原生本地存储天然具备云原生特性，满足容器场景中高扩展性、高可用性等特点。


DaoCloud Enterprise 5.0 容器集群
数据收集
Kafka
数据存储
MySQL
Mongo DB
数据分析
Elastic Search
OLAP
数据共享
App01
App02
块数据卷
NAS卷
对象卷
块存储
较高性能
大容量
NAS 存储池
较高性能
大容量
对象存储
超大空间
高可扩展
低成本
本地存储
XSKY
分布式存储
HUAWEI
传统存储
ceph
NFS

云原生存储

功能列表

云原生统一管理

- 混合存储接入

  统一 CSI 标准化接入，实现多数据访问类型（NFS、块存储、本地存储）接入，满足不同场景需求。

- 动态存储管理

  支持动态方式分配存储池资源，无需管理员手动管理、运维数据卷。

- 多数据卷创建方式

  可通过存储池动态创建数据卷，以及支持快照创建数据卷。

云原生本地存储（Hwameistor）

- 高性能本地卷

  无需外接存储设备，实现 IO 本地化，无需网络开销，实现高性能本地吞吐，支持 数据库、中间件等对性能有高要求的应用上云。

- 多类型数据卷

  支持 LVM 类型、裸磁盘类型数据卷，满足不同磁盘需求场景。

- CSI 标准

  通过标准的 CSI 标准接入 Hwameistor 本地存储，统一使用姿势。

- 主备高可用

  实现数据卷多副本冗余机制，保障数据高可用，提高数据读写可靠性。

- 数据卷扩容

业务无感扩容，支持应用在运行过程中，为挂载的数据卷进行弹性扩容。

生产可运维

- 无中断升级（不影响业务数据）

  数据平面、控制平面分离，控制平面升级/扩展节点时，业务应用数据读写 0 感知。

- 换盘

  支持磁盘告警后进行换盘，保障业务应用不受影响，生产可运维。

- 一键驱逐节点数据卷

  手动一键式驱逐某一节点数据卷，实现生产可运维。

- 自动驱逐节点数据卷

  结合 Kubernetes 驱逐行为自动感知并驱逐节点上的数据卷。

- 单磁盘维度（LD）数据迁移

  支持磁盘出现预警时并需要换盘，迁移所有数据，保障业务应用数据不丢失。

- 应用负载维度的数据迁移

  支持有状态应用重调度时，以应用为 维度实现数据迁移，保障业务应用 Pod 成功调度以及调度后的数据一致性。

- 统一仪表盘

  统一仪表盘展示资源使用率情况及分布 ，存储资源状态，及监控告警。

- 丰富 Metrics 指标

  全方位的数据服务监控告警，实现数据盘、存储池及存储驱动的全面监控，全面保障数据安全性。

背景与挑战

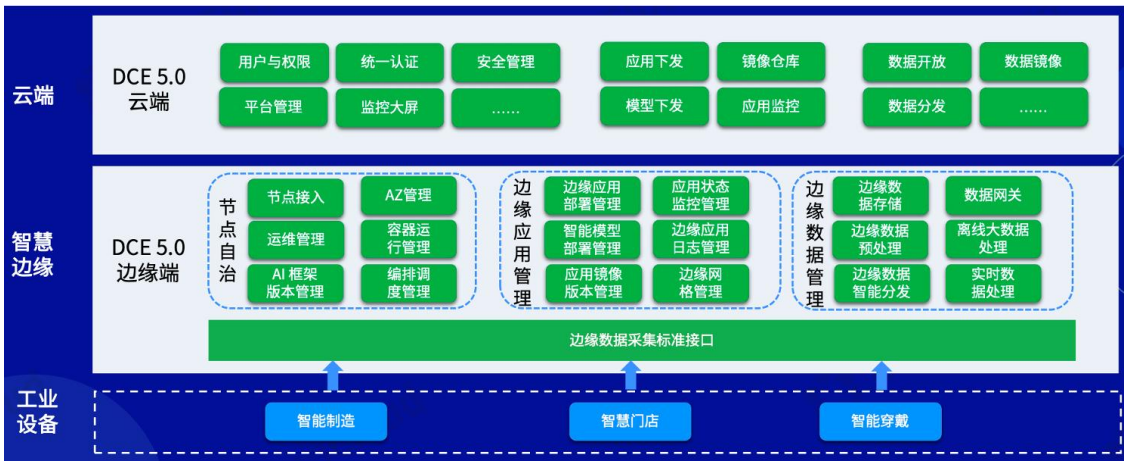
背景

中间件服务上云的存储需求

1. 云即将成为新数字体验的核心，数据库、消息队列等中间件上云数量正在激增。 而在中间件等有状态应用上云过程中，如何保障业务应用的高性能及高可靠性是企业需要面对的问题。
2. 对于物理设施存算融合场景，如何利用现有的设备存储空间满足有状态应用上云需求。
3. 随着越来越多关键性应用上云，对于云上存储如何实现高效运维，如何保障数据存储的可靠性，如何实行全面的包括应用/控制面/后端设备监控也将成为新的挑战。

云边协同场景中的存储方案

在 Garner 2021 的十大云趋势预测中，边缘计算成为新云。 在边缘场景中，边缘端也存在数据存储需求以及边缘数据预处理需求。 在边缘端，一般采用存算融合，但边缘的资源有限，如何基于边缘端进行一定的数据存储和数据计算，同样也是云边协同场景中需要解决的问题。



云边协同场景

挑战

企业就绪挑战

数据库、消息队列、大数据等有状态关键应用逐步上云，对于云上存储的性能（吞吐、时延）有新的更高的要求，此时如何满足高性能需求。

与此同时高效运维，保障数据存储的可靠性监控，实现全面的监控及问题预警，包括展示应用级/控制面挂载点以及后端设备的监控数据，并形成全链路监控，并能快速锁定存储问题。

敏捷化挑战

云原生应用场景对服务的敏捷度、灵活性要求非常高，很多场景期望容器的快速启动、灵活的调度，这样既需要存储卷，也需要敏捷地根据 Pod 的变化而调整。

1. 云盘挂载、卸载效率提高
2. 可以灵活的将块设备在不同节点进行快速的挂载切换
3. 提供存储服务的问题自动修复能力，减少人为干预
4. 在线扩容能力，为适用容器的快速扩容，如何实现应用业务不受影响的情况下在线扩容?
5. 随着越来越多的有状态应用容器化以及有状态应用部署在同一个集群，在集群规划业务分区域使用场景中，应用如何根据存储类型自动进行调度?

降本增效挑战

目前，数据中心计算密度提高的同时，如何解决存储性能瓶颈?

1. 定位排障性能和可用性问题（计划外的设备故障、故障定位及恢复）
2. 构建、配置、维护和扩展应用程序的复杂性（不同的部署环境及软件技术栈）
3. 存储瓶颈：解决计算瓶颈之后，则存储能力将成为制约计算密度的瓶颈
4. 业务应用按照业务峰值规划及设计，计算资源、存储资源的闲置率 ⩾ 50%。成本高，如何感知业务进行自动扩容等资源均衡。

CAS 存储

容器附加存储（Container Attached Storage, CAS）是一种由 Kubernetes 编排的基于微服务存储控制器的软件存储体系。 其存储控制器作为 Kubernetes 集群的一部分在容器中进行管理和运行。 这就让 CAS 具有可移植性，可以在任何 Kubernetes 平台上运行，包括任何云平台、裸金属服务器或传统共享存储系统。关键的是数据本身也可以通过容器访问，而不是存储在非平台共享的横向扩展存储系统中。 由于 CAS 利用微服务架构，将存储解决方案与绑定到物理存储设备的应用程序保持密切关联，从而减少了 I/O 访问时间。

CAS 这种模式非常符合分布式数据的趋势，也适合采用微型松散耦合工作负载的小型自治团队。 例如，一个团队可能需要 Postgres 来提供微服务，而另一个团队的研发数据可能依赖于 Redis 和 MongoDB。 一些用例可能对性能要求较高，一些可能在 20 分钟内消失，一些是写入/读取密集型等等。 在一个大型企业中，随着规模的增长，企业越来越信任各团队自行选择所用的工具，各团队所依赖技术的差异性将越来越大。

CAS 意味着开发人员可以在工作时不必担心企业存储体系结构的底层需求。 对于 CAS 来说，云盘与 SAN、裸机或虚拟机并没有什么不同。开发人员和平台 SRE 工程师没必要开会决定下一个存储供应商，也没必要反复讨论如何设置才能支持用例。 开发人员保持自治，可以使用 Kubernetes 集群可用的任何存储来启动本身的 CAS 容器。

CAS 反映了一个更广泛的解决方案趋势，其中许多方案均是 CNCF 的项目，以 Kubernetes 和微服务为基础进行构建，形成繁荣的云原生生态体系。 如今 CNCF 的云原生生态体系包含了安全、DNS、网络、网络策略管理、消息传输、跟踪、日志等众多项目。

CAS 的优势

敏捷性

CAS 中的每个存储卷都有一个容器化存储控制器和相应的容器化副本。因此，围绕这些组件的资源维护和调整是真正敏捷化的。 Kubernetes 的滚动升级功能实现了存储控制器和存储副本的无缝升级。CPU 和内存等资源可以使用容器 cgroup 进行优化。

存储策略的粒度

对存储软件进行容器化，并将存储控制器专用于每个卷，可以实现存储策略的最大粒度。使用 CAS 体系结构，您可以按卷配置所有存储策略。 此外，您可以监视每个卷的存储参数，并动态更新存储策略，以实现每个工作负载的预期结果。 随着卷存储策略中粒度的增加，对存储吞吐量、IOPS 和延迟的控制也会增加。

避免绑定

避免被云服务商绑定是许多 Kubernetes 用户的共同目标。 然而，有状态应用的数据通常仍然依赖于云服务商及其技术，或者依赖于底层的传统共享存储系统 NAS 或 SAN。 而使用 CAS 方法后，存储控制器可以在每个工作负载的后台进行数据迁移，使得实时迁移变得更简单。 换句话说，CAS 的控制粒度以无中断的方式简化了有状态工作负载从一个 Kubernetes 集群到另一个集群的迁移。

原生云

CAS 将存储软件容器化，并使用 Kubernetes CRD 来表示底层存储资源，如磁盘和存储池。这种模式使存储能够无缝地集成到其他云主机的工具中。 可以使用 Prometheus、Grafana、Fluentd、Weavescope、Jaeger 等云主机工具来调配、监控和管理存储资源。

另外，CAS 中卷的存储和性能是可扩缩的。由于每个卷都有自己的存储控制器，因此可以在节点存储容量的允许范围内弹性扩缩。 随着给定 Kubernetes 集群中容器应用数量的增加，会添加更多节点，从而提高存储容量和性能的总体可用性，使存储可用于新的应用容器。

更小的影响范围

由于 CAS 体系结构是按工作负载划分的，并且组件是松散耦合的，所以 CAS 的影响范围比典型的分布式存储体系结构小得多。

CAS 可以通过从存储控制器到存储副本的同步复制来提供高可用性。维护副本所需的元数据简化为副本节点的信息以及有关副本状态的信息。 如果某个节点出现故障，存储控制器将通过第二个或第三个副本在正运行且数据可用的节点上轮转。因此，使用 CAS 时，影响范围要小得多，影响仅局限在该节点上有副本的卷。

CRD 和 CR

CRD

CRD 是 Custom Resource Definition 的缩写，是 Kubernetes 内置原生的一个资源类型。 它是自定义资源（CR）的定义，用来描述什么是自定义资源。

CRD 可以向 Kubernetes 集群注册一种新资源，用于拓展 Kubernetes 集群的能力。 有了 CRD，就可以自定义底层基础设施的抽象，根据业务需求来定制资源类型，利用 Kubernetes 已有的资源和能力，通过乐高积木的模式定义出更高层次的抽象。

CR

CR 是 Custom Resource 的缩写，它实际是 CRD 的一个实例，是符合 CRD 中字段格式定义的一个资源描述。

CRDs + Controllers

我们都知道 Kubernetes 的扩展能力很强大，但仅有 CRD 并没有什么用，还需要有控制器（Custom Controller）的支撑，才能体现出 CRD 的价值，Custom Controller 可以监听 CR 的 CRUD 事件来实现自定义业务逻辑。

在 Kubernetes 中，可以说是 CRDs + Controllers = Everything。

另请参考 Kubernetes 官方文档：

- CustomResource
- CustomResourceDefinition

Volume

容器中的文件在磁盘上是临时存放的，这给容器中运行的较重要的应用程序带来一些问题。

- 问题一：当容器崩溃时文件会丢失。kubelet 会重新启动容器，但容器会以干净的状态重启。
- 问题二：在同一个 Pod 中运行多个容器并共享文件时出现此问题。

Kubernetes 卷（Volume）这一抽象概念能够解决这两个问题。

Kubernetes 支持很多类型的卷。Pod 可以同时使用任意数目的卷类型。 临时卷类型的生命周期与 Pod 相同，但持久卷可以比 Pod 的存活期长。 当 Pod 不再存在时，Kubernetes 也会销毁临时卷；不过 Kubernetes 不会销毁持久卷。 对于给定 Pod 中任何类型的卷，在容器重启期间数据都不会丢失。

卷的核心是一个目录，其中可能保存了数据，Pod 中的容器可以访问该目录中的数据。 所采用的特定卷类型将决定该目录如何形成、使用何种介质保存数据以及目录中存放的内容。

使用卷时，在 .spec.volumes 字段中设置为 Pod 提供的卷，并在 .spec.containers[*].volumeMounts 字段中声明卷在容器中的挂载位置。

参考 Kubernetes 官方文档：

- 卷
- 持久卷
- 临时卷

LVM

LVM 全称为 Logical Volume Manager，即逻辑卷管理器，在磁盘分区和文件系统之间添加一个逻辑层，为文件系统屏蔽下层磁盘分区布局提供一个抽象的盘卷，在盘卷上建立文件系统。

利用 LVM 可以在磁盘不用重新分区的情况下动态调整文件系统的大小，并且利用 LVM 管理的文件系统可以跨越磁盘。 当服务器添加了新的磁盘后，管理员不必将原有的文件移动到新的磁盘上，而是通过 LVM 直接扩展文件系统跨越磁盘。 也就是通过将底层的物理硬盘封装起来，然后以逻辑卷的方式呈现给上层应用。

LVM 通过对底层的硬盘进行封装，当我们对底层的物理硬盘进行操作时，不再是针对分区进行操作，而是通过一个叫做逻辑卷的东西对其进行底层的磁盘管理操作。

LVM 主要构成

- 物理存储介质（PM, physical media)：LVM 存储介质可以是分区、磁盘、RAID 阵列或 SAN 磁盘。
- 物理卷（PV, physical volume)：物理卷是 LVM 的基本存储逻辑块，但与基本的物理存储介质（如分区、 磁盘等）比较，却包含有与 LVM 相关的管理参数，创建物理卷可以用磁盘分区，也可以用磁盘本身。磁盘设备必须初始化为 LVM 物理卷，才能与 LVM 结合使用。
- 卷组（VG, Volume Group)：LVM 卷组由一个或多个物理卷组成。
- 逻辑卷（LV, logical volume)：LV 建立在 VG 之上，可以在 LV 之上建立文件系统。
- 物理范围（PE, physical extents)：PV 物理卷中可以分配的最小存储单元，PE 的大小是可以指定的，默认为 4MB。

- 逻辑范围（LE, logical extents）：LV 逻辑卷中可以分配的最小存储单元，在同一个卷组中，LE 的大小与 PE 是相同的， 并且一一对应。

LVM 优点

- 使用卷组，使多个硬盘空间看起来像是一个大的硬盘
- 使用逻辑卷，可以跨多个硬盘空间的分区 sdb1 sdb2 sdc1 sdd2 sdf
- 使用逻辑卷，可以在空间不足时动态调整它的大小
- 在调整逻辑卷大小时，不需要考虑逻辑卷在硬盘上的位置，不用担心没有可用的连续空间
- 可以在线对 LV、VG 进行创建、删除、调整大小等操作，LVM 上的文件系统也需要重新调整大小
- 允许创建快照，可以用来保存文件系统的备份
- RAID + LVM 结合使用：LVM 是软件的卷管理方式，而 RAID 是磁盘管理的方法。对于重要的数据，使用 RAID 来保护物理磁盘不会因为故障而中断业务，再用 LVM 来实现对卷的良性管理，更好地利用磁盘资源。

使用 LVM 的基本步骤

1. 物理磁盘被格式化为 PV，即空间被划分为一个个的 PE。PV 包含多个 PE。
2. 不同的 PV 加入到同一个 VG 中，即不同 PV 的 PE 全部进入到 VG 的 PE 池内。VG 包含多个 PV。
3. 在 VG 中创建 LV 逻辑卷，这个创建过程基于 PE，所以组成 LV 的 PE 可能来自不同的物理磁盘。LV 基于 PE 创建。
4. LV 直接可以格式化后挂载使用。
5. LV 的扩缩实际上就是增加或减少组成该 LV 的 PE 数量，其过程不会丢失原始数据。
6. 格式化 LV，并挂载使用。

LV 扩容

首先，确定是否有可用的扩容空间，因为空间是从 VG 里面创建的，并且 LV 不能跨 VG 扩容。若 VG 没有了容量，需要先扩 VG。步骤如下：

```
$ vgs
VG #PV #LV #SN Attr VSize VFree
vg-sdb1 1 8 1 wz--n- <16.00g <5.39g
$ lvextend -L +100M -r /dev/vg-sdb1/lv-sdb1 #将 /dev/vg-sdb1/lv-sdb 扩容 100M
```

VG 扩容

如果 VG 卷组中的空间不够了，需要添加新的磁盘，依次运行以下命令：

```
pvcreate /dev/sdc
vgextend vg-sdb1 /dev/sdb3
```

LV 快照

LVM 机制提供了对 LV 做快照的功能，以此来获得文件系统的状态一致性备份。LVM 采用写时复制技术（Copy-On-Write, COW），不用停止服务或将逻辑卷设为只读就可以进行备份，使用 LVM 快照功能既可以获得一致备份，又不会影响服务器的可用性。

LVM 采用的写时复制，是指创建 LVM 快照时，仅复制原始卷中数据的元数据。换句话说，也就是在创建 LVM 逻辑卷时，并不会发生数据的物理复制。 再换句话说，仅复制元数据，不复制物理数据，因此快照的创建几乎是实时的。 当原始卷上执行写入操作时，快照会跟踪原始卷中块的变更，这时原始卷上将要变更的数据会在变更之前拷贝到快照预留的空间。

PV 和 PVC

PV（PersistentVolume，持久卷）是对存储资源的抽象，将存储定义为一种容器应用可以使用的资源。 PV 由管理员创建和配置，与存储提供商的具体实现直接相关，例如文件存储、块存储、对象存储或 DRBD 等， 通过插件式的机制进行管理，供应用访问和使用。除 EmptyDir 类型的存储卷，PV 的生命周期独立于使用它的 Pod。

PVC（PersistentVolumeClaim，持久卷声明）是用户对存储资源的一个申请。就像 Pod 消耗 Node 的资源一样，PVC 会消耗 PV 的资源。PVC 可以申请存储空间的大小（Size）和访问模式（例如 ReadWriteOnce、ReadOnlyMany 或 ReadWriteMany）。

使用 PVC 申请的存储空间仍然不满足应用对存储设备的各种需求。 在很多情况下，应用程序对存储设备的特性和性能都有不同的要求，包括读写速度、并发性能、数据冗余等要求。 这就需要使用资源对象 StorageClass，用于标记存储资源和性能，根据 PVC 的需求动态供给合适的 PV 资源。 StorageClass 和存储资源动态供应的机制经完善后，实现了存储卷的按需创建，在共享存储的自动化管理进程中实现了重要的一步。

另请参考 Kubernetes 官方文档：

- 持久卷
- 存储类
- 动态卷供应

Kubernetes 存储

Kubernetes 为容器平台（或集群管理员）和应用开发人员提供了一些增强特性以支持运行有状态的工作负载。 这些特性确保了只要调度容器（包括卷的供应/创建、附加、挂载、解除挂载、拆分和删除）、存储容量管理（容器临时存储使用率、卷扩缩容等）， 都可以使用不同类型的文件和块存储（临时存储或持久化存储、本地存储或远程存储），这会影响基于存储的容器调度（数据重力、可用性等）和快照等常规存储操作。

在使用 HwameiStor 运行有状态工作负载时，需理解 Kubernetes 存储的几个抽象概念。

容器存储接口

容器存储接口（CSI）是在 Kubernetes 这种容器编排体系上将任意文件存储和块存储系统暴露给容器化工作负载的一个标准。 使用 CSI 后，像 HwameiStor 这样的第三方存储提供商可以写入和部署新的存储卷插件，例如 HwameiStor LocalDisk 和 LocalVolumeReplica，而无需修改 Kubernetes 核心代码。

当集群管理员安装 HwameiStor 时，所需的 HwameiStor CSI 驱动程序组件也将安装到 Kubernetes 集群中。

//在 CSI 之前，Kubernetes 支持使用 out-of-tree 资源调配器（也称为外部资源调配器）添加新的存储提供商。Kubernetes 树状卷（in-tree）早于外部资源调配器。而 Kubernetes 社区也在努力使用基于 CSI 的卷来替代树状卷。）

存储类和动态资源调配

StorageClass 为管理员提供了一种描述存储“类别”的方法。 不同的类别可能会映射到集群管理员确定的服务质量水平、备份策略或任意策略。 在某些存储系统中，StorageClass 这个概念称为“配置文件”。

有了动态资源调配功能后，集群管理员无需预先调配存储资源。 它会在用户请求时自动调配存储资源。动态卷资源调配的实现基于 StorageClass 这个抽象概念。 集群管理员可以根据需要，定义尽可能多的 StorageClass 对象，每个对象都会指定一个调配卷的插件（也称为调配器），并在资源调配时配置传递给该调配器的一组参数。

集群管理员可以在集群内定义和公开（来自相同或不同存储系统的）多种存储类型，每种类型都可以自定义参数。 这种设计还确保最终用户不必担心存储配置的复杂性和细微差别，但仍然能够从多个存储选项中进行选择。

安装 HwameiStor 后将附带了两个默认存储类，允许用户创建本地卷(HwameiStor LocalVolume）或副本（HwameiStor LocalVolumeReplica)。 集群管理员可以启用所需的存储引擎，然后为数据引擎创建存储类。

持久卷声明

PersistentVolumeClaim (PVC）是由集群管理员提供的 StorageClass 服务的用户存储请求。 在容器上运行的应用可以请求某种类型的存储。例如，容器可以指定所需的存储大小或存取数据的方式（只读、单次读/写、多次读写等）。

除了存储大小和存取模式之外，管理员还可以创建存储类，为 PV 提供自定义属性，例如磁盘类型（HDD 和 SSD）、性能水平或存储层级（常规存储或冷存储）。

持久卷

PersistentVolume (PV）由存储提供商在用户请求 PVC 时动态调配。PV 包含容器如何消耗存储的详细信息。 Kubernetes 和卷驱动程序使用 PV 中的细节将存储附加/解除附加到容器运行的节点，并将存储挂载/解除挂载到容器。

HwameiStor 控制面动态设置 HwameiStor 本地卷和副本，并帮助在集群中创建 PV 对象。

有状态和无状态

Kubernetes 提供了几个内置的有状态和无状态负载资源，让应用开发人员定义在 Kubernetes 上运行的负载。 通过创建 Kubernetes 无状态/有状态负载，并使用 PVC 将其连接到 PV，可以运行有状态负载。

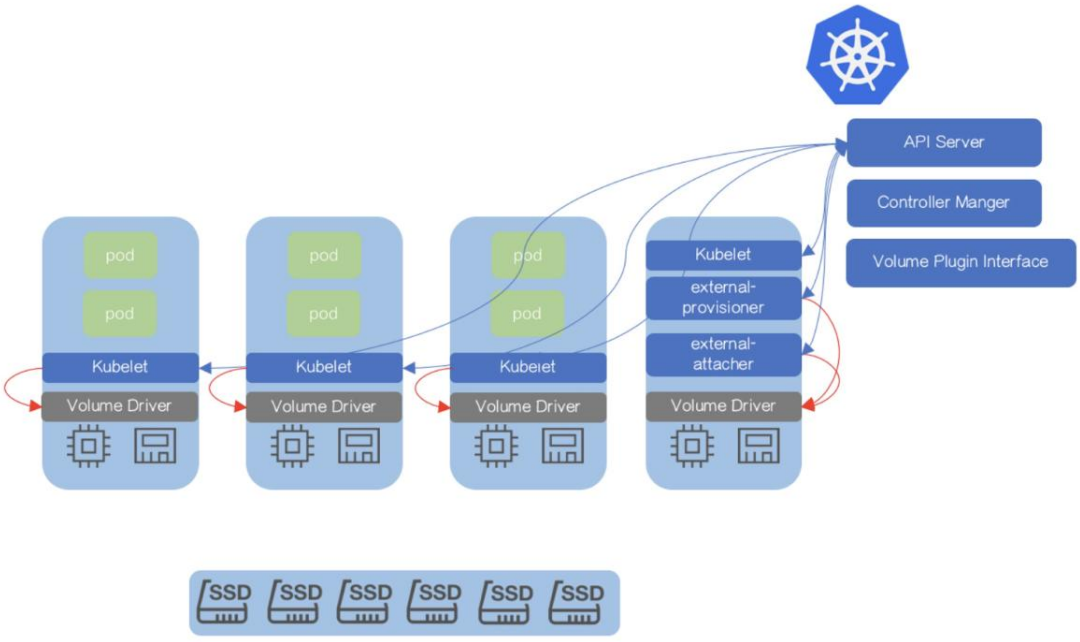
例如，可以用 YAML 创建引用 PVC 的 MySQL 无状态负载。 无状态负载引用的 MySQL PVC 应该使用请求的大小和 StorageClass 创建。 一旦 HwameiStor 控制平面为所需的 StorageClass 和容量提供了 PV，则声明设置为已满足。 然后，Kubernetes 将挂载 PV 并启动 MySQL 无状态负载。

CSI 接口

CSI 即 Container Storage Interfaces，容器存储接口。 目前，Kubernetes 中的存储子系统仍存在一些问题。 存储驱动程序代码在 Kubernetes 核心存储库中进行维护，这很难测试。Kubernetes 还需要授予存储供应商许可，便于将代码嵌入 Kubernetes 核心存储库。

CSI 旨在定义行业标准，该标准将使支持 CSI 的存储提供商能够在支持 CSI 的容器编排系统中使用。

下图描述了一种与 CSI 集成的高级 Kubernetes 原型。



CSI 接口

- 引入了三个新的外部组件以解耦 Kubernetes 和存储提供程序逻辑
- 蓝色箭头表示针对 API 服务器进行调用的常规方法
- 红色箭头显示 gRPC 以针对 Volume Driver 进行调用

扩展 CSI 和 Kubernetes

为了实现在 Kubernetes 上扩展卷的功能，应该扩展几个组件，包括 CSI 规范、“in-tree”卷插件、external-provisioner 和 external-attacher。

扩展 CSI 规范

最新的 CSI 0.2.0 仍未定义扩展卷的功能。应引入新的 3 个 RPC：RequiresFSResize、 ControllerResizeVolume 和 NodeResizeVolume。

```
service Controller {
  rpc CreateVolume (CreateVolumeRequest)
    returns (CreateVolumeResponse) {}
……
  rpc RequiresFSResize (RequiresFSResizeRequest)
    returns (RequiresFSResizeResponse) {}
  rpc ControllerResizeVolume (ControllerResizeVolumeRequest)
    returns (ControllerResizeVolumeResponse) {}
}
service Node {
  rpc NodeStageVolume (NodeStageVolumeRequest)
    returns (NodeStageVolumeResponse) {}
……
  rpc NodeResizeVolume (NodeResizeVolumeRequest)
    returns (NodeResizeVolumeResponse) {}
}
```

扩展 “In-Tree” 卷插件

除了扩展的 CSI 规范之外，Kubernetes 中的 csiPlugin 接口还应实现 expandablePlugin。 csiPlugin 接口将扩展代表 ExpanderController 的 PersistentVolumeClaim。

```
type ExpandableVolumePlugin interface {
VolumePlugin
```

ExpandVolumeDevice(spec Spec, newSize resource.Quantity, oldSize resource.Quantity) (resource.Quantity, error)
RequiresFSResize() bool
}

实现卷驱动程序

最后，为了抽象化实现的复杂性，应将单独的存储提供程序管理逻辑硬编码为以下功能，这些功能在 CSI 规范中已明确定义：

- CreateVolume
- DeleteVolume
- ControllerPublishVolume
- ControllerUnpublishVolume
- ValidateVolumeCapabilities
- ListVolumes
- GetCapacity
- ControllerGetCapabilities
- RequiresFSResize
- ControllerResizeVolume

展示

以具体的用户案例来演示此功能。

- 为 CSI 存储供应商创建存储类

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-qcfs
parameters:
  csiProvisionerSecretName: orain-test
```

```
  csiProvisionerSecretNamespace: default
provisioner: csi-qcfsplugin
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

- 在 Kubernetes 集群上部署包括存储供应商 csi-qcfsplugin 在内的 CSI 卷驱动
- 创建 PVC qcfs-pvc，它将由存储类 csi-qcfs 动态配置

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: qcfs-pvc
  namespace: default
....
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 300Gi
  storageClassName: csi-qcfs
```

- 创建 MySQL 5.7 实例以使用 PVC qcfs-pvc
- 为了反映完全相同的生产级别方案，实际上有两种不同类型的工作负载，包括：
  - 批量插入使 MySQL 消耗更多的文件系统容量
  - 浪涌查询请求
- 通过编辑 pvc qcfs-pvc 配置动态扩展卷容量

Rook-ceph 存储方案

DCE 5.0 支持众多第三方存储方案，我们针对 Rook-ceph 进行了详细的测试，并最终将其作为 Addon 集成了应用商店中。 以下是对 Rook-ceph 的详细调研和测评。

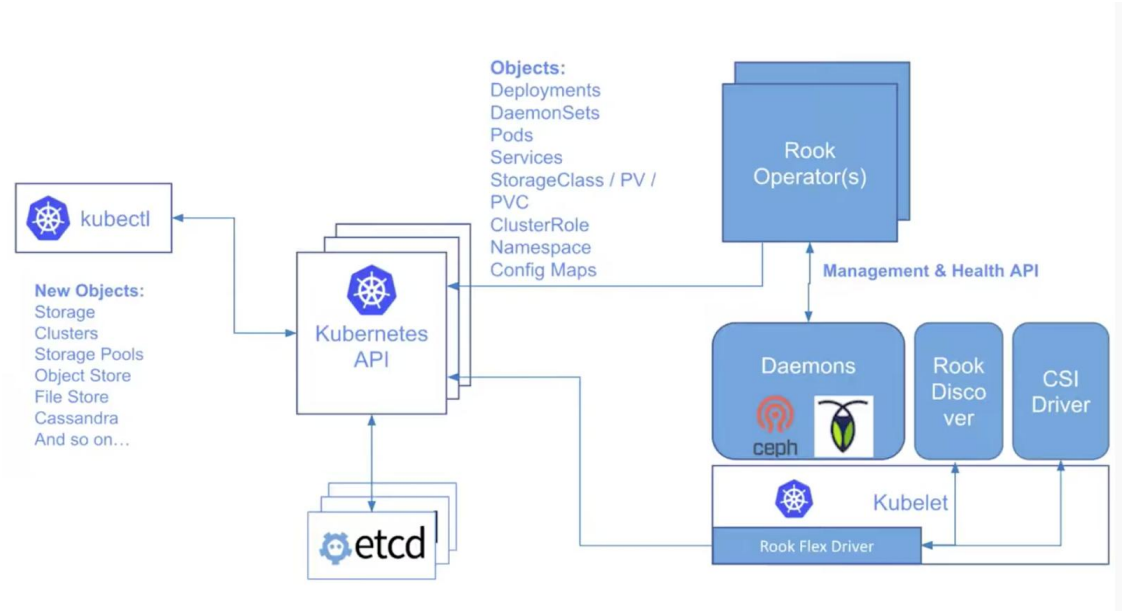
有关应用商店 Addon 的图形化界面安装、部署、卸载等操作说明，将于稍后提供。

简介

Rook 是一个开源云原生存储编排器，为各种存储解决方案（存储提供商）提供平台、框架和支持并与云原生环境原生集成。

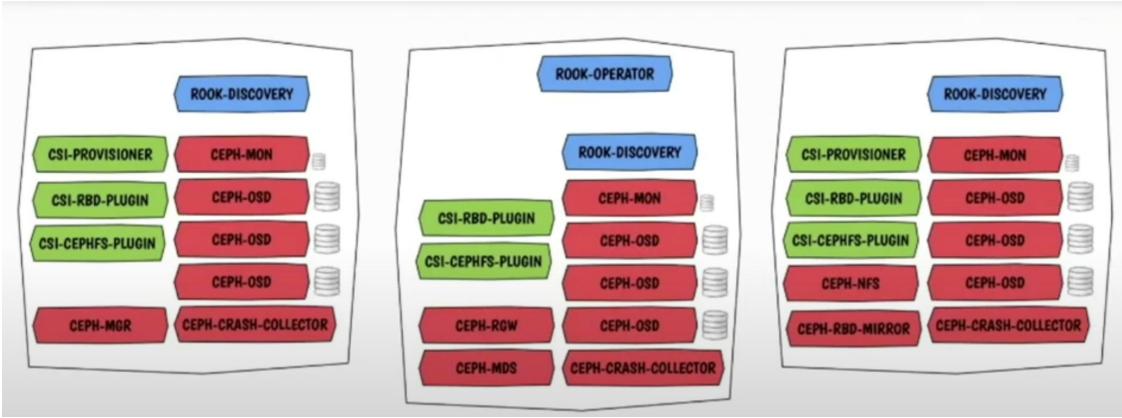
Rook 支持多个存储提供商。它将分布式存储软件转变为自我管理、自我扩展和自我修复的存储服务。 它通过自动执行存储提供商的部署、引导、配置、预配、扩展、升级、迁移、灾难恢复、监视和资源管理来实现此目的。 当与 Kubernetes 一起使用时，Rook 使用 Kubernetes 调度和编排平台提供的功能，为调度、生命周期管理、资源管理、安全性、监控和用户体验提供无缝体验。

- Rook-ceph 架构



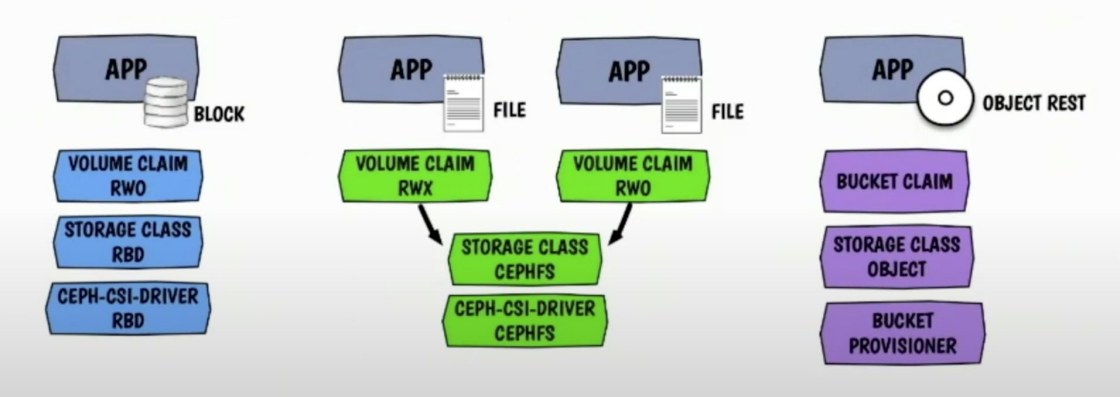
rook-ceph 架构

- Rook 管理



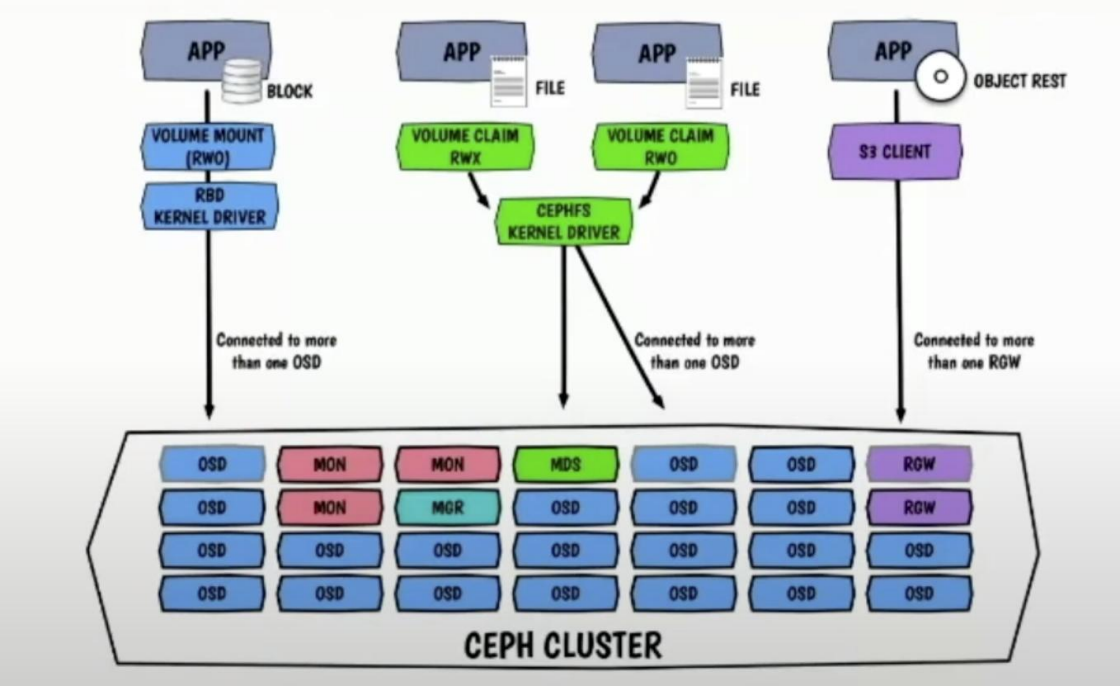
rook 管理

- CSI 存储资源调配



rook 资源调配

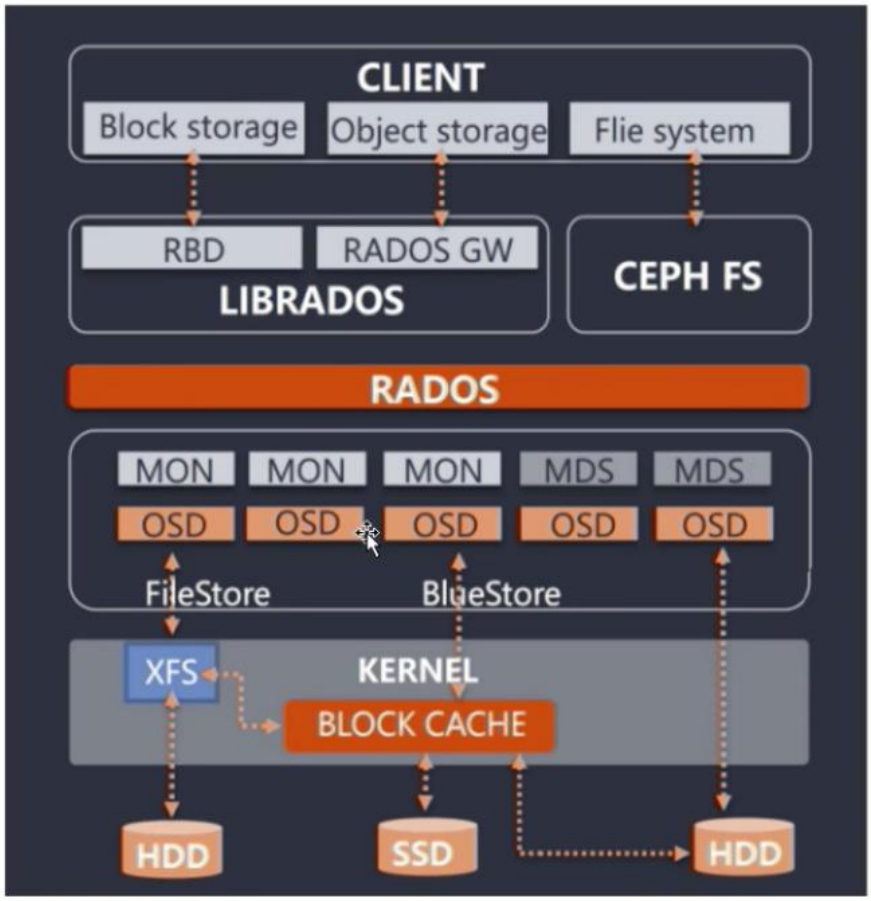
- Ceph 数据路径



rook 数据路径

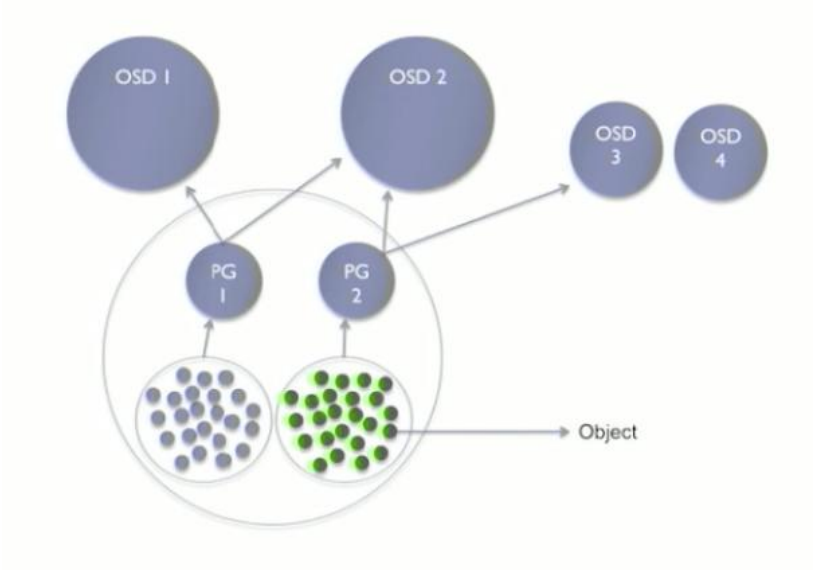
- 分布式架构

  Ceph 是一种分布式、可扩展的开源存储解决方案，适用于块、对象和共享文件系统存储。 Ceph 在过去几年中已经发展成为开源分布式存储解决方案的标准，在大中型企业中进行了多年的生产部署。



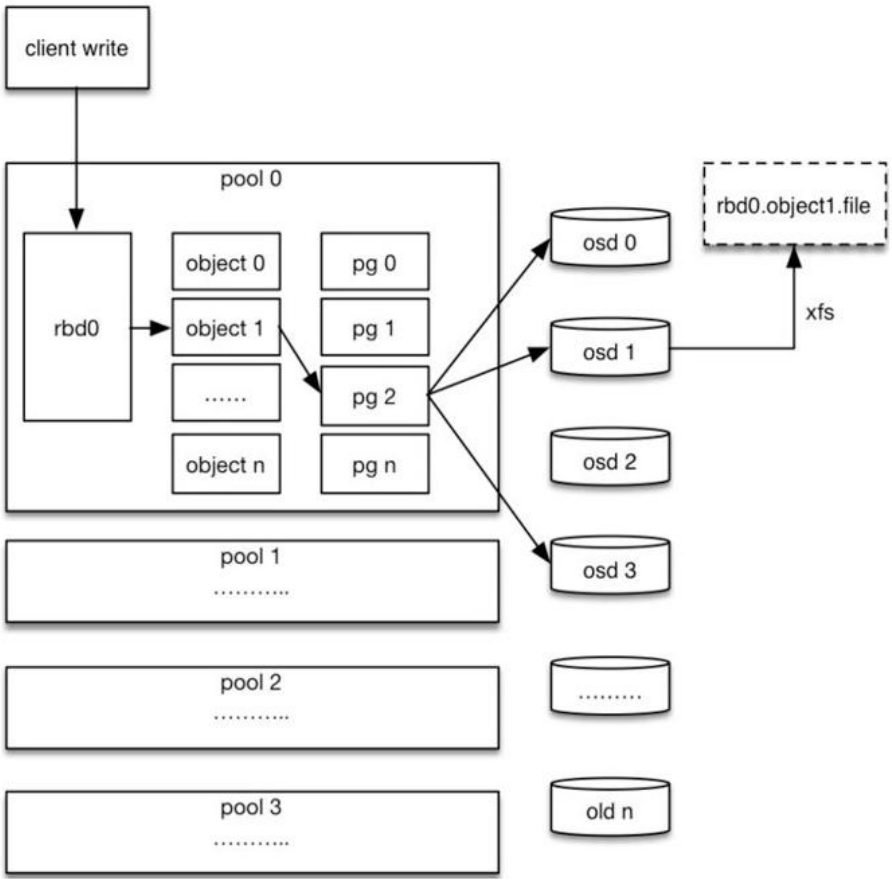
rook 分布式架构

- 组件：Ceph 存储池、归置组（PG）、对象、OSD



rook 组件

- Ceph RBD 工作原理



rook 原理

测试验证

准备环境

本次测试使用 4 个虚拟机节点来部署一个 Kubernetes 集群：

- 1 个 master 节点
- 3 个 worker 节点

其中 kubelet 版本为 1.23.6

```
[root@k8s-10-6-162-21 ~]# kubectl get no
NAME                          STATUS              ROLES
AGE           VERSION
```

```
k8s-10-6-162-21                    Ready            control-plane,master
    19d          v1.23.6
k8s-10-6-162-22                    Ready            <non
e>                          19d          v1.23.6
k8s-10-6-162-23                    Ready            <non
e>                          19d          v1.23.6
k8s-10-6-162-24                    Ready            <non
e>                          13d          v1.23.6
```

部署 Rook-Ceph

克隆 GitHub 源码

```
[root@k8s-10-6-162-21 ~]# git clone https://github.com/rook/rook.git

[root@k8s-10-6-162-21 ~]# ls
anaconda-ks.cfg calico.yaml rook rook-ceph-image rook-ceph-image.zip

[root@k8s-10-6-162-21 rook]# git branch
master
* remotes/origin/release-1.7

[root@k8s-10-6-162-21 ceph]# cd /root/rook/cluster/examples/kubernetes/ceph

[root@k8s-10-6-162-21 ceph]# ls
ceph-client.yaml create-external-cluster-resources.py flex object-openshift.yaml
rgw-external.yaml
cluster-external-management.yaml create-external-cluster-resources.sh flex-migr
ator.yaml object-test.yaml scc.yaml
cluster-external.yaml csi images.txt object-user.yaml storageclass-bucket-delete.
yaml
cluster-on-local-pvc.yaml dashboard-external-https.yaml import-external-cluster.
sh object.yaml storageclass-bucket-retain.yaml
cluster-on-pvc.yaml dashboard-external-http.yaml monitoring operator-openshift.
```

yaml test-data

cluster-stretched-aws.yaml dashboard-ingress-https.yaml nfs-test.yaml operator.yaml toolbox-job.yaml

cluster-stretched.yaml dashboard-loadbalancer.yaml nfs.yaml osd-purge.yaml toolbox.yaml

cluster-test.yaml direct-mount.yaml object-bucket-claim-delete.yaml pool-ec.yaml volume-replication-class.yaml

cluster.yaml filesystembak.yaml object-bucket-claim-retain.yaml pool-mirrored.yaml volume-replication.yaml

common-external.yaml filesystem-ec.yaml object-ec.yaml pool-test.yaml

common-second-cluster.yaml filesystem-mirror.yaml object-external.yaml pool.yaml

common.yaml filesystem-test.yaml object-multisite-pull-realm.yaml pre-k8s-1.16 crds.yaml filesystem.yaml object-multisite.yaml rbdmirror.yaml

部署 Operator、CRD 和参数

[root@k8s-10-6-162-21 ceph]# kubectl create -f crds.yaml -f common.yaml -f operator.yaml

[root@k8s-10-6-162-21 ceph]# kubectl get crd | grep ceph

cephblockpools.ceph.rook.io 2022-10-14T02:22:34Z

cephclients.ceph.rook.io 2022-10-14T02:22:34Z

cephclusters.ceph.rook.io 2022-10-14T02:22:35Z

cephfilesystemmirrors.ceph.rook.io 2022-10-14T02:22:35Z

cephfilesystems.ceph.rook.io 2022-10-14T02:22:35Z

cephnfses.ceph.rook.io 2022-10-14T02:22:35Z

cephobjectrealms.ceph.rook.io 2022-10-14T02:22:35Z

cephobjectstores.ceph.rook.io 2022-10-14T02:22:35Z

cephobjectstoreusers.ceph.rook.io 2022-10-14T02:22:35Z

cephobjectzonegroups.ceph.rook.io 2022-10-14T02:22:35Z

cephobjectzones.ceph.rook.io 2022-10-14T02:22:35Z

cephrbdmirrors.ceph.rook.io 2022-10-14T02:22:35Z

objectbucketclaims.objectbucket.io 2022-10-14T02:22:35Z
objectbuckets.objectbucket.io 2022-10-14T02:22:35Z
volumes.rook.io 2022-10-14T02:22:35Z

部署 cluster 和 toolbox

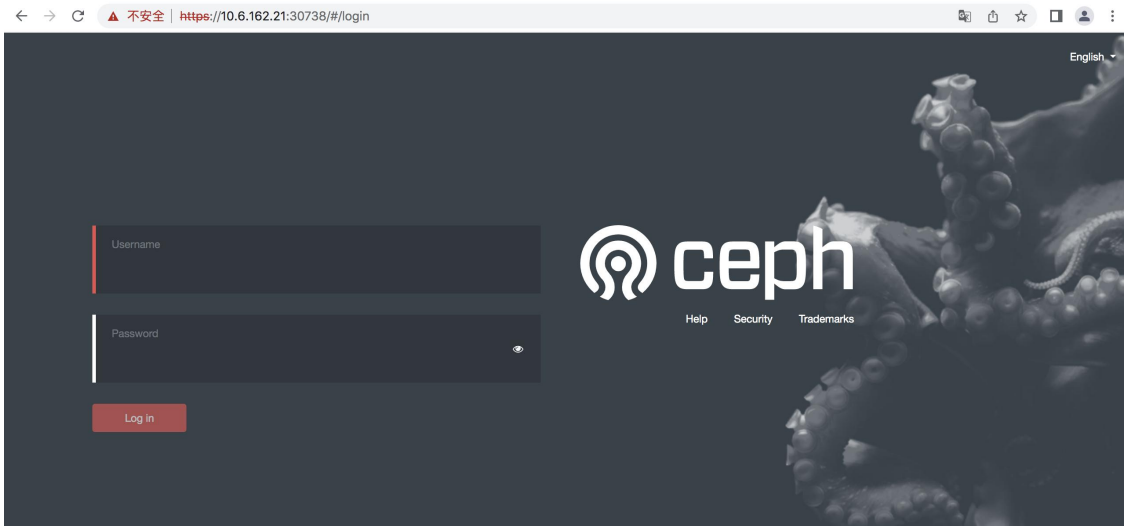
```
[root@k8s-10-6-162-21 ceph]# kubectl create -f cluster.yaml

[root@k8s-10-6-162-21 ceph]# kubectl create -f toolbox.yaml

[root@k8s-10-6-162-21 ceph]# kubectl get po -n rook-ceph -owide
NAME READY STATUS RESTARTS AGE IP
NODE NOMINATED NODE READINESS GATES
csi-cephfsplugin-bmbtc 3/3 Running 6 (4d1h ago) 7d7h
10.6.162.24 k8s-10-6-162-24 <none> <none>
csi-cephfsplugin-provisioner-5c8b6d6f4-hvfg6 6/6 Running 678 (38m ago)
7d7h 10.244.56.106 k8s-10-6-162-23 <none> <none>
csi-cephfsplugin-provisioner-5c8b6d6f4-w6snr 6/6 Running 545 (38m ago)
4d1h 10.244.169.249 k8s-10-6-162-22 <none> <none>
csi-cephfsplugin-r87qt 3/3 Running 6 (4d1h ago) 7d7h
10.6.162.23 k8s-10-6-162-23 <none> <none>
csi-cephfsplugin-xqrmr 3/3 Running 6 (4d1h ago) 7d7h
10.6.162.22 k8s-10-6-162-22 <none> <none>
csi-rbdplugin-mcpm6 3/3 Running 6 (4d1h ago) 7d7h
10.6.162.22 k8s-10-6-162-22 <none> <none>
csi-rbdplugin-pj4cc 3/3 Running 6 (4d2h ago) 7d7h
10.6.162.23 k8s-10-6-162-23 <none> <none>
csi-rbdplugin-provisioner-8564cfd44-mnjr8 6/6 Running 665 (38m ago) 4d8h
10.244.56.102 8s-10-6-162-23 <none> <none>
csi-rbdplugin-provisioner-8564cfd44-tgzc8 6/6 Running 545 (37m ago) 4d1h
10.244.169.248 k8s-10-6-162-22 <none> <none>
csi-rbdplugin-tnpbc 3/3 Running 6 (4d1h ago) 7d7h
10.6.162.24 k8s-10-6-162-24 <none> <none>
```

rook-ceph-crashcollector-k8s-10-6-162-22-5c4444c7bb-pd6tf 1/1 Running 0 27h 10.244.169.208 k8s-10-6-162-22 <none> <none>
rook-ceph-crashcollector-k8s-10-6-162-23-775d89c75-kgnzq 1/1 Running 2 (4d2h ago) 7d7h 10.244.56.105k8s-10-6-162-23 <none> <none>
rook-ceph-crashcollector-k8s-10-6-162-24-6cc87876c6-tk6q2 1/1 Running 0 27h 10.244.49.18 k8s-10-6-162-24 <none> <none>
rook-ceph-mds-myfs-a-94f74bd56-k2gt7 1/1 Running 0 27h 10.244.49.22 k8s-10-6-162-24 <none> <none>
rook-ceph-mds-myfs-b-5679797bbd-jr8gt 1/1 Running 69 (38m ago) 27h 10.244.169.209 k8s-10-6-162-22 <none> <none>
rook-ceph-mgr-a-5b69d56b4d-6z87x 1/1 Running 0 2d7h 10.244.49.9 k8s-10-6-162-24 <none> <none>
rook-ceph-mon-a-7589d48f7d-lgps2 1/1 Running 25 (3h44m ago) 21h 10.244.56.122 k8s-10-6-162-23 <none> <none>
rook-ceph-mon-b-6577cdc487-fk564 1/1 Running 161 (3h14m ago) 7d7h 10.244.169.244 k8s-10-6-162-22 <none> <none>
rook-ceph-mon-d-66bdbd9dc-pn6wc 1/1 Running 83 (38m ago) 4d 10.244.49.61 k8s-10-6-162-24 <none> <none>
rook-ceph-operator-866c96498c-mm2xd 1/1 Running 4 (4d1h ago) 7d7h 10.244.169.242 k8s-10-6-162-22 <none> <none>
rook-ceph-osd-0-6d774dcf99-6qggj 1/1 Running 167 (38m ago) 7d3h 10.244.169.245 k8s-10-6-162-22 <none> <none>
rook-ceph-osd-1-7f7c599677-qx4sh 1/1 Running 26 (4d2h ago) 7d3h 10.244.56.101 k8s-10-6-162-23 <none> <none>
rook-ceph-osd-2-7564c58d96-bknjf 1/1 Running 125 (35m ago) 7d3h 10.244.169.243 k8s-10-6-162-22 <none> <none>
rook-ceph-osd-3-5599b7bdcb-wmdkx 1/1 Running 0 4d1h 10.244.49.51 k8s-10-6-162-24 <none> <none>
rook-ceph-osd-4-7c6d79cdff-4fzdc 1/1 Running 0 4d1h 10.244.49.55 k8s-10-6-162-24 <none> <none>
rook-ceph-osd-5-ffb7b5ff7-phssl 1/1 Running 26 (4d2h ago) 7d3h 10.244.56.103 k8s-10-6-162-23 <none> <none>

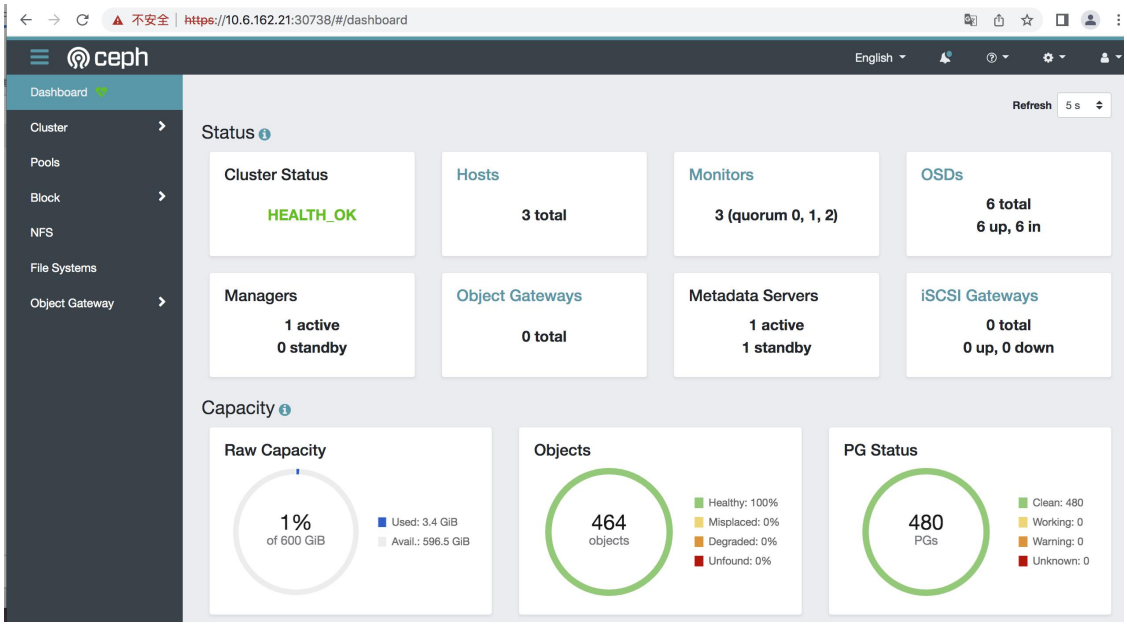
```
rook-ceph-osd-prepare-k8s-10-6-162-22-kjwzt 0/1 Completed 0 5h34m
    10.244.169.222 k8s-10-6-162-22 <none> <none>
rook-ceph-osd-prepare-k8s-10-6-162-23-fngbr 0/1 Completed 0 5h34m
    10.244.56.67 k8s-10-6-162-23 <none> <none>
rook-ceph-osd-prepare-k8s-10-6-162-24-p7c28 0/1 Completed 0 5h34m
    10.244.49.27 k8s-10-6-162-24 <none> <none>
rook-ceph-tools-55f548895f-8pdhm 1/1 Running 0 4d1h
  10.244.56.107 k8s-10-6-162-23 <none> <none>
```

部署 dashboard

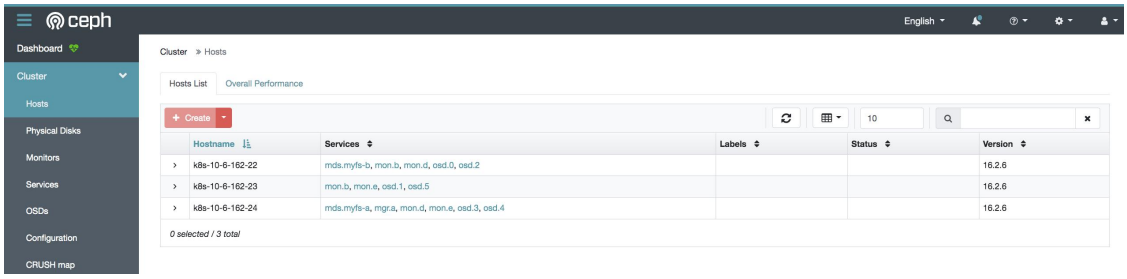
```
[root@k8s-10-6-162-21 ~]# kubectl apply -f dashboard-external-https.yaml

[root@k8s-10-6-162-21 ~]# kubectl get svc -n rook-ceph
NAME TYPE CLUSTER-IP EXTERNAL
-IP PORT(S) AGE
csi-cephfsplugin-metrics ClusterIP 10.111.92.191 <none> 8
080/TCP,8081/TCP 8d
csi-rbdplugin-metrics ClusterIP 10.106.136.163 <none>
8080/TCP,8081/TCP 8d
rook-ceph-mgr ClusterIP 10.97.170.29 <none>
9283/TCP 8d
rook-ceph-mgr-dashboard ClusterIP 10.103.18.81 <none>
 8443/TCP 8d
rook-ceph-mgr-dashboard-external-https NodePort 10.111.121.36 <none>
 8443:30738/TCP 8d
rook-ceph-mon-a ClusterIP 10.107.216.105 <non
e> 6789/TCP,3300/TCP 8d
rook-ceph-mon-b ClusterIP 10.100.168.72 <non
e> 6789/TCP,3300/TCP 8d
rook-ceph-mon-d ClusterIP 10.101.56.41 <none>
6789/TCP,3300/TCP 5d16h
```



dashboard-1



dashboard-2



dashboard-3

操作 Ceph 存储管理所用的 Rook 工具

[root@k8s-10-6-162-21 ceph]# kubectl exec -it rook-ceph-tools-55f548895f-fzbq2 -n rook-ceph -- bash

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph -s
cluster:
id: d65486e5-a8c3-4ca5-8ff6-9daddfc0f6c4
health: HEALTH_OK

services:
mon: 3 daemons, quorum a,b,d (age 71m)
mgr: a(active, since 30h)
mds: 1/1 daemons up, 1 hot standby
osd: 6 osds: 6 up (since 68m), 6 in (since 7d)

data:
volumes: 1/1 healthy
pools: 4 pools, 480 pgs
objects: 464 objects, 1009 MiB
usage: 3.8 GiB used, 596 GiB / 600 GiB avail
pgs: 480 active+clean

io:
client: 1.5 KiB/s rd, 2 op/s rd, 0 op/s wr

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph mgr services
{
"dashboard": "https://10.244.49.9:8443/",
"prometheus": "http://10.244.49.9:9283/"
}

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph df

--- RAW STORAGE ---
CLASS SIZE AVAIL USED RAW USED %RAW USED
hdd 600 GiB 596 GiB 3.8 GiB 3.8 GiB 0.63
TOTAL 600 GiB 596 GiB 3.8 GiB 3.8 GiB 0.63

--- POOLS ---
POOL ID PGS STORED OBJECTS USED %USED MAX AVAIL
device_health_metrics 1 64 0 B 0 0 B 0 189 GiB
replicapool 4 128 880 MiB 290 2.6 GiB 0.45 189 GiB
myfs-metadata 7 256 1.5 MiB 25 3.0 MiB 0 283 GiB
myfs-data0 8 32 115 MiB 149 231 MiB 0.04 283 GiB

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph osd status
ID HOST USED AVAIL WR OPS WR DATA RD OPS RD DATA STATE
0 k8s-10-6-162-22 524M 99.4G 0 0 0 89 exists,up
1 k8s-10-6-162-23 677M 99.3G 0 0 0 0 exists,up
2 k8s-10-6-162-22 501M 99.5G 0 0 0 0 exists,up
3 k8s-10-6-162-24 694M 99.3G 0 0 1 15 exists,up
4 k8s-10-6-162-24 744M 99.2G 0 0 0 0 exists,up
5 k8s-10-6-162-23 747M 99.2G 0 0 0 0 exists,up

Ceph 存储配置

创建 RBD 池和归置组 (PG):

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph osd pool create replicapool 128

[root@k8s-10-6-162-21 kubernetes]# ceph osd pool application enable replicapool rbd

Creating storage class:

```
[root@k8s-10-6-162-21 rbd]# pwd
/root/rook/deploy/examples/csi/rbd
[root@k8s-10-6-162-21 rbd]# kubectl apply -f storageclass.yaml

[root@k8s-10-6-162-21 ~]# kubectl get sc
NAME                  PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
rook-ceph-block       rook-ceph.rbd.csi.ceph.com Delete Immediate true 5d20h
rook-cephfs           rook-ceph.cephfs.csi.ceph.com Delete Immediate true 2d17h
```

应用部署验证

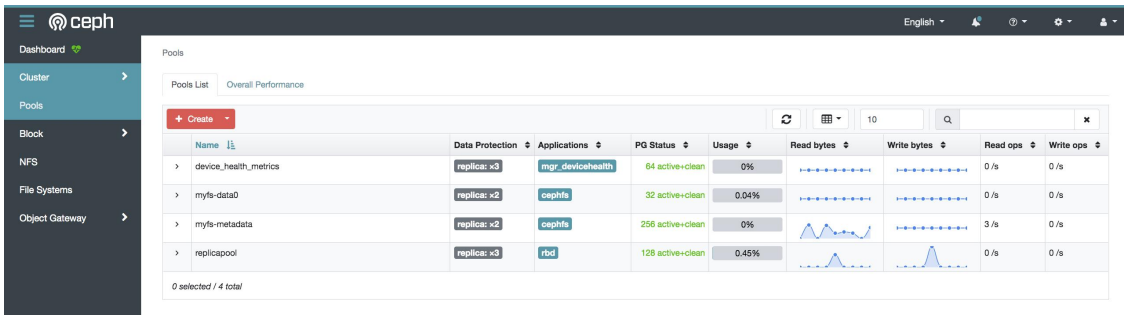
在 RBD 上部署 Mysql 和 WordPress

```
[root@k8s-10-6-162-21 examples]# cd kubernetes/
ceph  mysql2-cephfs.yaml  mysql.yaml  README.md  wordpress.yaml
[root@k8s-10-6-162-21 kubernetes]# pwd
/root/rook/cluster/examples/kubernetes
[root@k8s-10-6-162-21 kubernetes]# kubectl apply -f mysql.yaml
[root@k8s-10-6-162-21 kubernetes]# kubectl apply -f wordpress.yaml
```

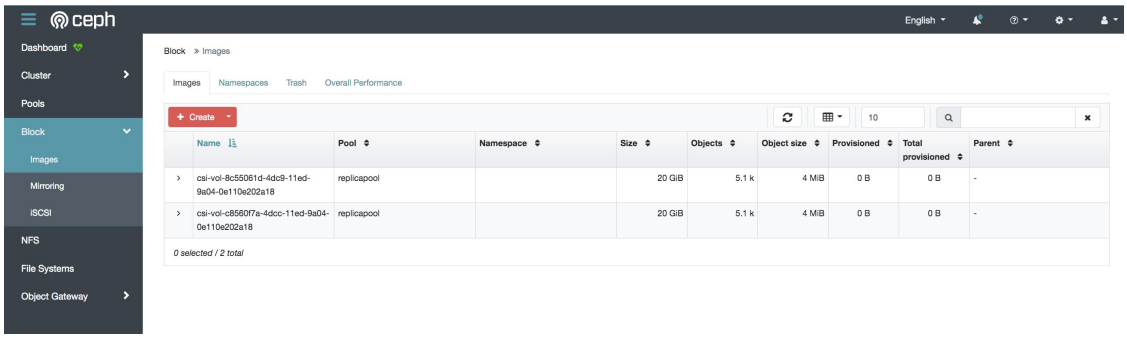
在 Cephfs 上部署 Mysql

```
[root@k8s-10-6-162-21 kubernetes]# kubectl apply -f mysql2-cephfs.yaml

[root@k8s-10-6-162-21 kubernetes]# kubectl get po
NAME                                READY   STATUS    RESTARTS      AGE
nginx-85b98978db-gbjtb              1/1     Running   0             6d18h
nginx-85b98978db-rbprt              1/1     Running   0             6d18h
nginx-85b98978db-w5m24              1/1     Running   0             6d18h
wordpress-b98c66fff-2bh8n           1/1     Running   5 (6d17h ago)   6d17h
wordpress-mysql-79966d6c5b-wqv9b    1/1     Running   0             6d17h
```

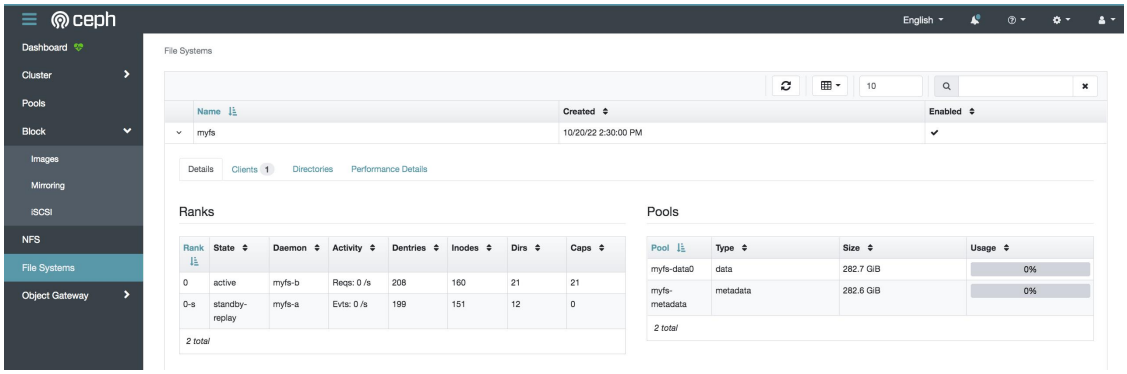
```
wordpress-mysql2-6857bbcf7d-2rl9w                1/1 Running 0 3d19h

[root@k8s-10-6-162-21 kubernetes]# kubectl get svc
NAME                 TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes           ClusterIP 10.96.0.1 <none> 443/TCP 22d
nginx                NodePort 10.107.222.243 <none> 80:31090/TCP 21d
wordpress            LoadBalancer 10.109.11.61 <pending> 80:32249/TCP 6d
23h
wordpress-mysql      ClusterIP None <none> 3306/TCP 6d23h
wordpress-mysql2     ClusterIP None <none> 3306/TCP 3d19h

[root@rook-ceph-tools-55f548895f-8pdhm /]# ceph osd pool ls
device_health_metrics
replicapool
myfs-metadata
myfs-data0
```



dashboard-4



dashboard-5



dashboard-6

[root@k8s-10-6-162-21 kubernetes]# kubectl get pv,pvc

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

persistentvolume/pvc-19515505-cd8f-41d1-ae91-bb42c3eb64f3 20Gi RWO Delete Bound default/mysql-pv-claim2 rook-cephfs 3d20h

persistentvolume/pvc-7647bc80-febc-4299-a62a-8446d2c364c6 20Gi RWO Delete Bound default/wp-pv-claim rook-ceph-block 6d23h

persistentvolume/pvc-b07feec8-adc8-4e22-abd6-177b5db4fdb1 20Gi RWO Delete Bound default/mysql-pv-claim rook-ceph-block 7d

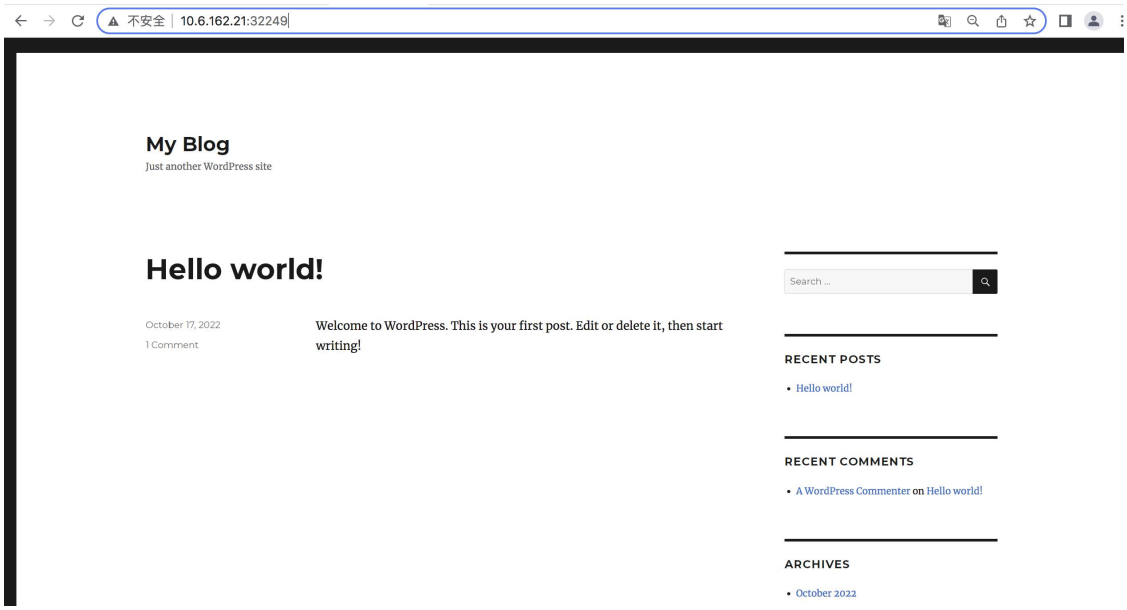
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

persistentvolumeclaim/mysql-pv-claim Bound pvc-b07feec8-adc8-4e22-abd6-177b5db4fdb1 20Gi RWO rook-ceph-block 7d

persistentvolumeclaim/mysql-pv-claim2 Bound pvc-19515505-cd8f-41d1-ae91-bb42c3eb64f3 20Gi RWO rook-cephfs 3d20h

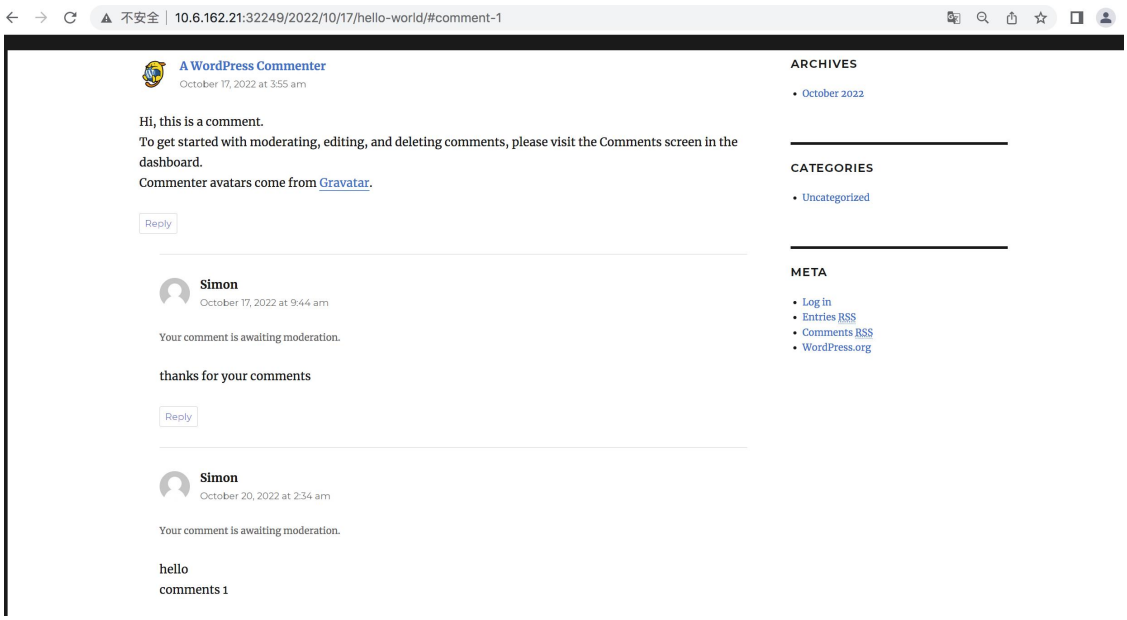
persistentvolumeclaim/wp-pv-claim Bound pvc-7647bc80-febc-4299-a62a-8446d2c364c6 20Gi RWO rook-ceph-block 6d23h

成功部署后，查看 wordpress/mysql 上的博客：



blog-1

还在博客上做了评论：

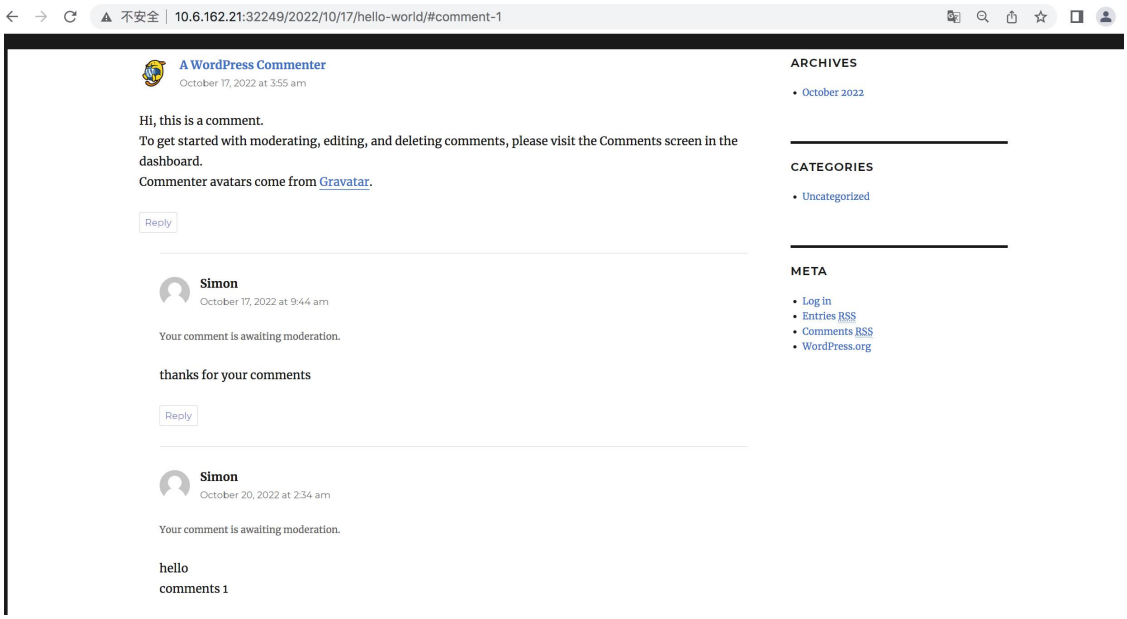


blog-2

数据持久性验证

删除并重启 mysql / wordpress Pod：

[root@k8s-10-6-162-21 kubernetes]# kubectl delete po wordpress-mysql-79966d6c5b-wc6fs



blog-3

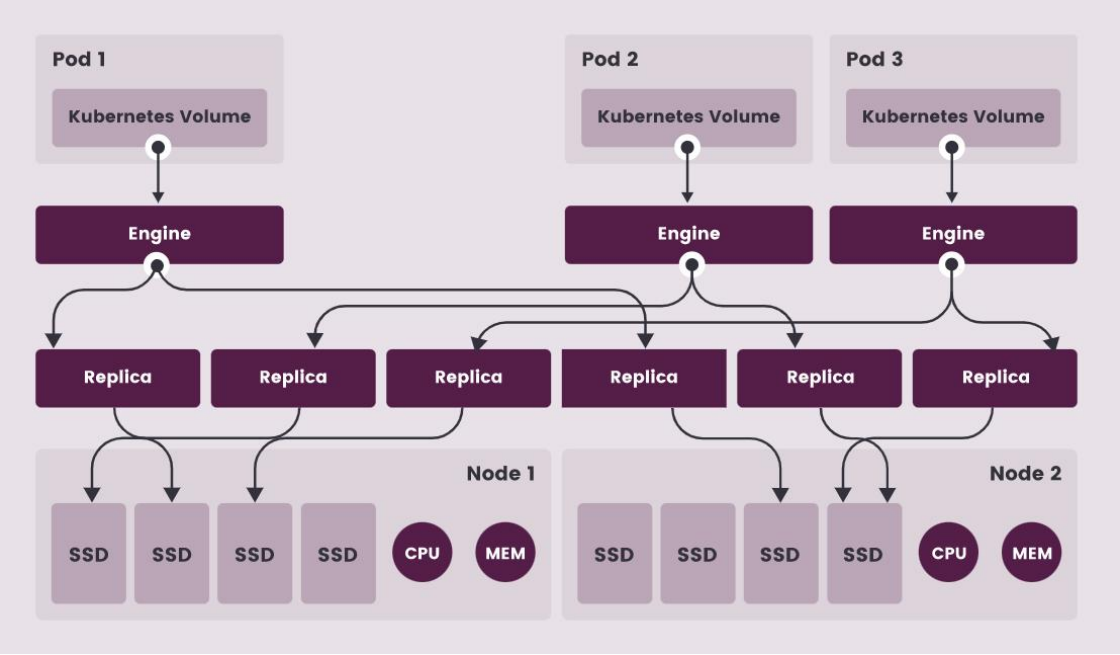
Longhorn 存储方案

DCE 5.0 支持众多第三方存储方案，我们针对 Longhorn 进行了详细的测试，并最终将其作为 Addon 集成了应用商店中。 以下是对 Longhorn 的简单调研和测评报告。

Longhorn 是一个轻量级的云原生 Kubernetes 分布式存储平台，可以在任意基础设施上运行。 Longhorn 与 DCE 可以结合使用部署高可用性持久化块存储。

1. 设计与架构

   - 控制平面：Longhorn Manager 以 DaemonSet 部署
   - 数据平面：Longhorn Engine 是 storage controller 可以有多个 replicas

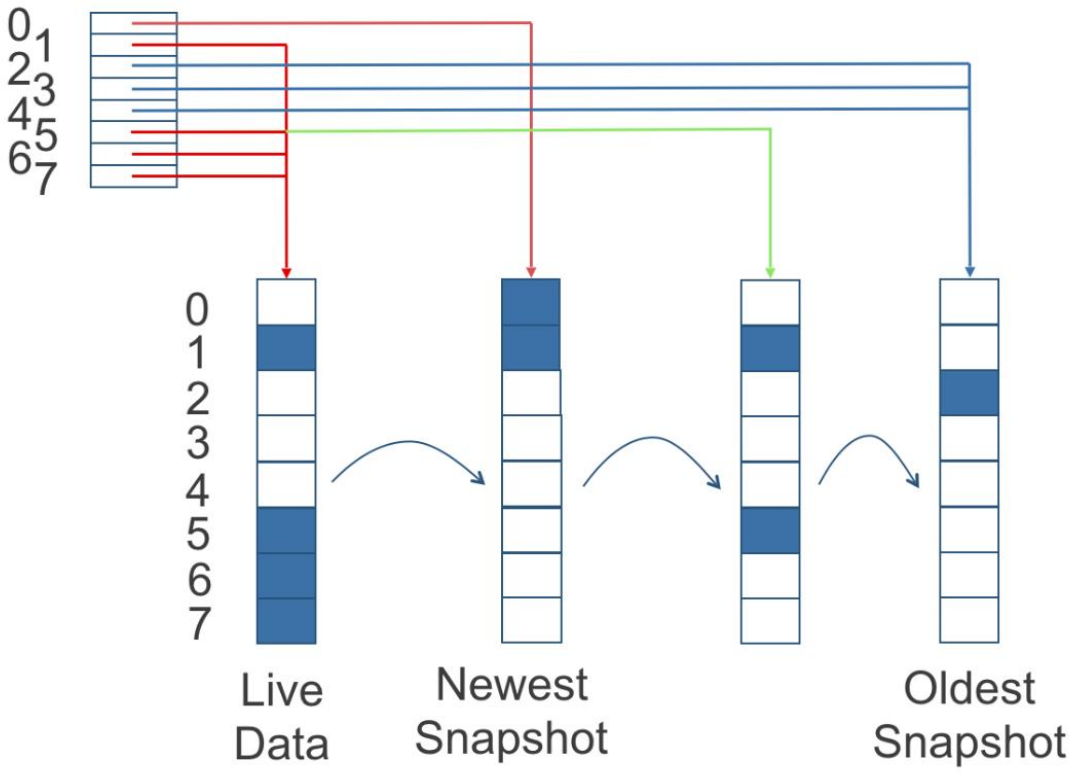


架构图

2. Longhorn 存储卷

   - 支持存储卷的 Thin provisioning

- 存储卷维护模式 maintenance mode 用于 snapshot reverting operation 操作
- 每一个存储卷的 replica 包含多个快照 snapshots.
- 默认的存储卷的 replica 数量可以在 settings 设置. 当存储卷被挂载以后 replica 数量可以通过 UI 改变。
- Longhorn 是一个 crash-consistent 块存储方案，在创建快照 snapshot 之前会自动先同步 sync 命令

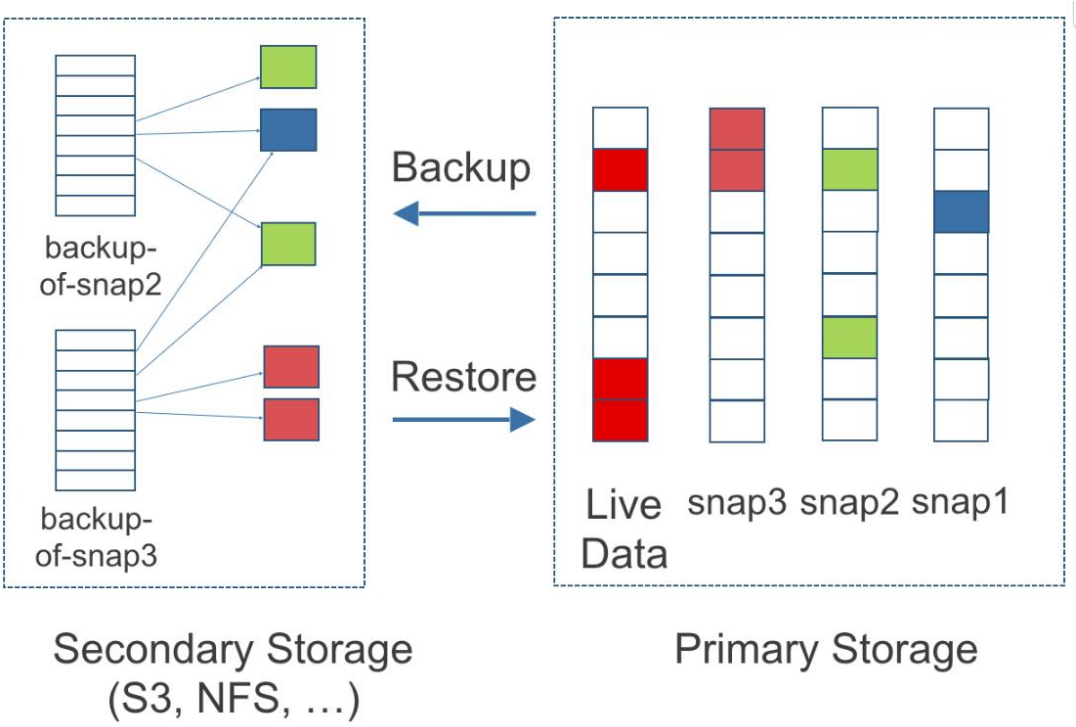


存储卷

3. 数据备份及外部二级存储

- 用于备份的 NFS/S3 兼容的外部二级存储是独立于 Kubernetes 集群以外的。即使在 Kubernetes 集群不可用的情况下数据依然可用
- Longhorn 也会将存储卷同步到灾备集群（DR）的二级存储用于灾难数据恢复

- 备份是多个快照数据的扁平化的集合。
- 支持连续重复快照及备份。
- 支持 CSI 存储卷的 Clone



数据备份

4. 高可用 High Availability

   - 支持 Replica 自动平衡设置
   - 支持 data locality setting：使用存储卷的 pod 运行的节点上至少有一个 replica 副本
   - 支持显示节点存储空间使用。
   - 支持 Kubernetes Cluster Autoscaler (Experimental)
   - 支持存储卷意外卸载后的的自动恢复
   - 支持集群节点失效后的存储卷的自动恢复

5. 监控 Monitoring

   - 支持 Prometheus and Grafana 监控 Longhorn

- Longhorn metrics 可以被整合进 DCE 监控系统
- 支持 Kubelete Metrics 监控
- 支持 Longhorn 告警策略

6. 高级功能

- 支持 Backing Image
- 支持 Orphaned Replica Directories
- 支持 DCE 集群恢复：集群所有存储卷的恢复
- 支持多写操作 ReadWriteMany (RWX) workloads （NFSv4）
- 支持 Longhorn Volume 作为 iSCSI Target

通过 Helm 部署 Rook-Ceph

本文将提供用 Helm 部署 Rook-Ceph 云原生存储系统的操作步骤及说明。

Helm 安装

[root@k8s-10-6-162-31 ~]# wget https://get.helm.sh/helm-v3.10.1-linux-amd64.tar.gz

[root@k8s-10-6-162-31 ~]# tar xvfz helm-v3.10.1-linux-amd64.tar.gz
linux-amd64/
linux-amd64/helm
linux-amd64/LICENSE
linux-amd64/README.md
[root@k8s-10-6-162-31 ~]#
[root@k8s-10-6-162-31 ~]#
[root@k8s-10-6-162-31 ~]# ls
anaconda-ks.cfg calico.yaml helm-v3.10.1-linux-amd64.tar.gz linux-amd64 rook rook-ceph-image rook-ceph-image.zip
[root@k8s-10-6-162-31 ~]# cd linux-amd64
[root@k8s-10-6-162-31 linux-amd64]# ls
helm LICENSE README.md
[root@k8s-10-6-162-31 linux-amd64]# mv helm /usr/bin
[root@k8s-10-6-162-31 linux-amd64]#
[root@k8s-10-6-162-31 linux-amd64]# cd /root
[root@k8s-10-6-162-31 ~]# helm
The Kubernetes package manager

Common actions for Helm:

- helm search: search for charts

– helm pull: download a chart to your local directory to view
– helm install: upload the chart to Kubernetes
– helm list: list releases of charts

添加 rook repo

helm repo add rook-release https://charts.rook.io/release

[root@k8s-10-6-162-31 ~]# helm repo list
NAME URL
rook-release https://charts.rook.io/release
stable http://mirror.azure.cn/kubernetes/charts/
aliyun https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

[root@k8s-10-6-162-31 ~]# helm search repo rook-ceph
NAME CHART VERSION APP VERSION DESCRIPTION
rook-release/rook-ceph v1.10.5 v1.10.5 File, Block, and Object Storage Services for yo...
rook-release/rook-ceph-cluster v1.10.5 v1.10.5 Manages a single Ceph cluster namespace for Rook

Helm 安装 rook operator

pwd[root@k8s-10-6-162-31 ~]# helm install --namespace rook-ceph rook-ceph rook-release/rook-ceph --create-namespace --set image.repository=rook/ceph --set csi.cephcsi.image=quay.io/cephcsi/cephcsi:v3.7.2 --set csi.registrar.image=registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.5.1 --set csi.provisioner.image=registry.k8s.io/sig-storage/csi-provisioner:v3.3.0 --set csi.snapshotter.image=registry.k8s.io/sig-storage/csi-snapshotter:v6.1.0 --set csi.attacher.image=registry.k8s.io/sig-storage/csi-attacher:v4.0.0 --set csi.resizer.image=registry.k8s.io/sig-storage/csi-resizer:v1.6.0

```
[root@k8s-10-6-162-31 ~]# helm ls -A
NAME                    NAMESPACE       REVISION        UPDATED
                                        STATUS          CHART                           APP VER
SION
rook-ceph               rook-ceph       1               2022-11-07 13:58:04.376723834
 +0800 CST              deployed        rook-ceph-v1.10.5               v1.10.5

[root@k8s-10-6-162-31 ~]# kubectl get po -n rook-ceps
NAMESPACE               NAME                                            READY
                        STATUS                  RESTARTS                AGE
rook-ceph               rook-ceph-operator-964d7fbbd-j2krp              1/1
                        Running                 0                       39m
```

Helm 安装 rook-ceph cluster 及 ceph tool

```
[root@k8s-10-6-162-31 ~]# helm install --namespace rook-ceph rook-ceph-clu
ster rook-release/rook-ceph-cluster --set operatorNamespace=rook-ceph --set
 cephClusterSpec.storage.deviceFilter="^sd." --set cephClusterSpec.cephVersion.i
mage=quay.io/ceph/ceph:v17.2.3
NAME: rook-ceph-cluster
LAST DEPLOYED: Mon Nov 7 14:49:28 2022
NAMESPACE: rook-ceph
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Ceph Cluster has been installed. Check its status by running:
kubectl --namespace rook-ceph get cephcluster

Visit https://rook.io/docs/rook/latest/CRDs/ceph-cluster-crd/ for more informati
on about the Ceph CRD.

Important Notes:
```

– You can only deploy a single cluster per namespace
– If you wish to delete this cluster and start fresh, you will also have to wipe t
he OSD disks using `sfdisk`

```
[root@k8s-10-6-162-31 examples]# helm ls -A
NAME                    NAMESPACE       REVISION        UPDATED
                                        STATUS          CHART
        APP VERSION
rook-ceph               rook-ceph       1               2022-11-07 13:5
8:04.376723834 +0800 CST        deployed        rook-ceph-v1.10.5       v1.
10.5
rook-ceph-cluster       rook-ceph       1               2022-11-07 14:4
9:28.709538725 +0800 CST        deployed        rook-ceph-cluster-v1.10.5   v1.
10.5

[root@k8s-10-6-162-31 ceph]# kubectl create -f toolbox.yaml
deployment.apps/rook-ceph-tools created

[root@k8s-10-6-162-31 examples]# kubectl get po -n rook-ceph
NAME                                                    READY
        STATUS          RESTARTS        AGE
csi-cephfsplugin-6jxvj                                  2/2
        Running         0               90m
csi-cephfsplugin-h68jt                                  2/2
        Running         0               90m
csi-cephfsplugin-provisioner-785d976fcf-p96s9           5/5
        Running         0               30m
csi-cephfsplugin-provisioner-785d976fcf-w7j7p           5/5
        Running         0               90m
csi-cephfsplugin-rzq9d                                  2/2
        Running         0               90m
csi-rbdplugin-7jcgf                                     2/2
```

```
        Running                0                90m
csi-rbdplugin-7z4v8                                                2/2
        Running                0                90m
csi-rbdplugin-8l6c9                                                2/2
        Running                0                90m
csi-rbdplugin-provisioner-dc499c7dd-62d2p                          5/5
          Running                2                90m
csi-rbdplugin-provisioner-dc499c7dd-kv6c5                          5/5
          Running                0                30m
rook-ceph-crashcollector-k8s-10-6-162-31-67678cc9f9-8ctkb          1/1
            Running                0                46m
rook-ceph-crashcollector-k8s-10-6-162-32-8655c84654-9mj22          1/1
              Running                0                18m
rook-ceph-crashcollector-k8s-10-6-162-33-5dbcc6965c-8dk89          1/1
              Running                0                46m
rook-ceph-mds-ceph-filesystem-a-b8fdd6876-s8s94                    2/2
              Running                0                30m
rook-ceph-mds-ceph-filesystem-b-fd667b865-49vx2                    2/2
              Running                0                32m
rook-ceph-mgr-a-5dbd5b49d5-2thts                                   3/3
            Running                0                46m
rook-ceph-mgr-b-6bbc5966c8-gt2fg                                   3/3
              Running                0                30m
rook-ceph-mon-a-56d4476dfd-hjr6b                                   2/2
            Running                1                (47m ago) 89m
rook-ceph-mon-b-5978cc6657-zd2tb                                   2/2
              Running                0                86m
rook-ceph-mon-c-5d94c6c9cc-9s6p9                                   2/2
              Running                0                63m
rook-ceph-operator-964d7fbbd-d6s6w                                 1/1
            Running                0                30m
rook-ceph-osd-0-79467bcb49-qtf6g                                   2/2
```

Running 0 30m
rook-ceph-osd-1-64c5945d78-72z4q 2/2
Running 0 30m
rook-ceph-osd-2-f68b5bcb7-59nwb 2/2
Running 0 30m
rook-ceph-osd-3-99c576b7-j7prn 2/2
Running 0 30m
rook-ceph-osd-4-579d5f966f-cqw5f 2/2
Running 0 30m
rook-ceph-osd-5-655d7bfbd7-kztj2 2/2
Running 0 30m
rook-ceph-osd-prepare-k8s-10-6-162-31-tzz2f 0/1
Completed 0 23m
rook-ceph-osd-prepare-k8s-10-6-162-32-7blmc 0/1
Completed 0 22m
rook-ceph-osd-prepare-k8s-10-6-162-33-pjjmd 0/1
Completed 0 22m
rook-ceph-rgw-ceph-objectstore-a-6d8b954f7d-jx9zx 2/2
Running 0 18m
rook-ceph-tools-7c8ddb978b-2mf52 1/1
Running 0 40m

Ceph 工具验证

[root@k8s-10-6-162-31 ~]# kubectl exec -it rook-ceph-tools-7c8ddb978b-2mf5
2 -n rook-ceph -- bash
bash-4.4$

bash-4.4$ ceph -s
cluster:
id: fcba483d-2c38-48cd-9137-fb93be678383
health: HEALTH_WARN
1 MDSs report slow metadata IOs

Reduced data availability: 6 pgs inactive

services:
mon: 3 daemons, quorum a,b,c (age 16m)
mgr: a(active, since 115s), standbys: b
mds: 1/1 daemons up, 1 standby
osd: 6 osds: 0 up, 6 in (since 96s)

data:
volumes: 1/1 healthy
pools: 6 pools, 6 pgs
objects: 0 objects, 0 B
usage: 0 B used, 0 B / 0 B avail
pgs: 100.000% pgs unknown
6 unknown

至此，用 Helm 部署安装 rook-ceph 完成。

通过 Helm 部署并验证 OpenEBS

本文将提供用 Helm 部署并验证 OpenEBS 云原生存储系统的操作步骤及说明。DCE 5.0 支持众多第三方存储方案，我们针对 OpenEBS 进行了相关测试，并最终将其作为 Addon 集成了应用商店中。 以下是对 OpenEBS 的调研和测评。

有关应用商店 Addon 的图形化界面安装、部署、卸载等操作说明，将于稍后提供。

测试环境

本次测试使用三个虚拟机节点部署一个 Kubernetes 集群：1 个 Master + 2 个 Worker 节点，kubelet 版本为 1.23.6。

```
[root@k8s-10-6-162-31 ~]# kubectl get no
NAME                STATUS   ROLES                  AGE    VERSION
k8s-10-6-162-31     Ready    control-plane,master   114d   v1.23.6
k8s-10-6-162-32     Ready    <none>                 114d   v1.23.6
k8s-10-6-162-33     Ready    <none>                 114d   v1.23.6
```

添加并更新 OpenEBS repo

```
[root@k8s-10-6-162-31 ~]# helm repo add openebs https://openebs.github.io/charts
"openebs" has been added to your repositories
[root@k8s-10-6-162-31 ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "longhorn" chart repository
...Successfully got an update from the "openebs" chart repository
Update Complete. ⎈Happy Helming!⎈
[root@k8s-10-6-162-31 ~]# helm repo list
NAME  URL
```

longhorn https://charts.longhorn.io
openebs https://openebs.github.io/charts

Helm 安装 OpenEBS

[root@k8s-10-6-162-31 ~]# helm install openebs --namespace openebs openebs/openebs --create-namespace
NAME: openebs
LAST DEPLOYED: Tue Jan 31 14:44:11 2023
NAMESPACE: openebs
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Successfully installed OpenEBS.

Check the status by running: kubectl get pods -n openebs

The default values will install NDM and enable OpenEBS hostpath and device storage engines along with their default StorageClasses. Use `kubectl get sc` to see the list of installed OpenEBS StorageClasses.

Note: If you are upgrading from the older helm chart that was using cStor and Jiva (non-csi) volumes, you will have to run the following command to include
the older provisioners:

helm upgrade openebs openebs/openebs \
--namespace openebs \
--set legacy.enabled=true \
--reuse-values

For other engines, you will need to perform a few more additional steps to

enable the engine, configure the engines (e.g. creating pools) and create StorageClasses.

For example, cStor can be enabled using commands like:

```
helm upgrade openebs openebs/openebs \
--namespace openebs \
--set cstor.enabled=true \
--reuse-values
```

For more information,
- view the online documentation at https://openebs.io/docs or
- connect with an active community on Kubernetes slack #openebs channel.

查看已安装 OpenEBS 集群资源

```
[root@k8s-10-6-162-31 ~]# kubectl get po -nopenebs
NAME                                           READY   STATUS    RESTARTS      AGE
openebs-localpv-provisioner-5646cc6748-sdh2g   1/1     Running   6 (4m44s ago)   47m
openebs-ndm-6dffs                              1/1     Running   0             47m
openebs-ndm-fs2jk                              1/1     Running   0             47m
openebs-ndm-m5x6d                              1/1     Running   0             47m
openebs-ndm-operator-65fdff8c8d-zvl8v          1/1     Running   0             47m

[root@k8s-10-6-162-31 ~]# mkdir -p /data/openebs/local
[root@k8s-10-6-162-31 ~]# cd /data/openebs/local
[root@k8s-10-6-162-31 local]# pwd
```

/data/openebs/local

[root@k8s-10-6-162-31 ~]# kubectl get sc -A

NAME PROVISIONER RECLAIMPOLICY VOLU

MEBINDINGMODE ALLOWVOLUMEEXPANSION AGE

longhorn (default) driver.longhorn.io Delete Immediate

true 48d

openebs-device openebs.io/local Delete WaitForFirst

Consumer false 19h

openebs-hostpath openebs.io/local Delete WaitForFirst

Consumer false 19h

创建 local-hostpath-pvc 资源

[root@k8s-10-6-162-31 ~]# cat local-hostpath-pvc.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: local-hostpath-pvc

spec:

storageClassName: openebs-hostpath

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5G

[root@k8s-10-6-162-31 ~]#

[root@k8s-10-6-162-31 ~]# cat local-hostpath-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hello-local-hostpath-pod

spec:

```
volumes:
- name: local-storage
persistentVolumeClaim:
claimName: local-hostpath-pvc
containers:
- name: hello-container
image: busybox
command:
- sh
- -c
- 'while true; do echo "`date` [`hostname`] Hello from OpenEBS Local PV." >>
/data/greeting.txt; sleep $(($RANDOM % 5 + 300)); done'
volumeMounts:
- mountPath: /data
name: local-storage

[root@k8s-10-6-162-31 ~]# kubectl apply -f local-hostpath-pvc.yaml
persistentvolumeclaim/local-hostpath-pvc created
[root@k8s-10-6-162-31 ~]# kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
local-hostpath-pvc Pending openebs-hostpath 8s

[root@k8s-10-6-162-31 ~]# kubectl apply -f local-hostpath-pod.yaml
pod/hello-local-hostpath-pod created
[root@k8s-10-6-162-31 ~]#
[root@k8s-10-6-162-31 ~]# kubectl get po
NAME READY STATUS RESTARTS AGE
hello-local-hostpath-pod 0/1 Pending 0 11s
longhorn-iscsi-installation-2thd5 1/1 Running 1 48d
longhorn-iscsi-installation-ctqtg 1/1 Running 1 (47d ago) 48d
longhorn-iscsi-installation-mrm4h 1/1 Running 1 (47d ago) 48d
```

```
[root@k8s-10-6-162-31 ~]# kubectl get po
NAME READY STATUS RESTARTS AGE
hello-local-hostpath-pod 1/1 Running 0 4m56s
longhorn-iscsi-installation-2thd5 1/1 Running 1 48d
longhorn-iscsi-installation-ctqtg 1/1 Running 1 (47d ago) 48d
longhorn-iscsi-installation-mrm4h 1/1 Running 1 (47d ago) 48d
```

创建 workload 并验证

```
[root@k8s-10-6-162-31 ~]# kubectl exec hello-local-hostpath-pod -- cat /data
/greeting.txt
Wed Feb 1 03:50:57 UTC 2023 [hello-local-hostpath-pod] Hello from OpenEBS
 Local PV.

[root@k8s-10-6-162-31 ~]# kubectl exec -it hello-local-hostpath-pod -sh
error: you must specify at least one command for the container
[root@k8s-10-6-162-31 ~]# kubectl exec -it hello-local-hostpath-pod -- sh
/ # ls
bin data dev etc home proc root sys tmp usr var
/ # cd data
/data # ls
greeting.txt
/data # cat greeting.txt
Wed Feb 1 03:50:57 UTC 2023 [hello-local-hostpath-pod] Hello from OpenEBS
 Local PV.
Wed Feb 1 03:55:59 UTC 2023 [hello-local-hostpath-pod] Hello from OpenEBS
 Local PV.
Wed Feb 1 04:01:01 UTC 2023 [hello-local-hostpath-pod] Hello from OpenEBS
Local PV.
Wed Feb 1 04:06:04 UTC 2023 [hello-local-hostpath-pod] Hello from OpenEBS
 Local PV.

[root@k8s-10-6-162-31 ~]# kubectl get -o yaml pv |grep 'path:'
```

```
path: /var/openebs/local/pvc-44db3536-2dc3-4b4d-bcd6-6d388a534fcd
[root@k8s-10-6-162-31 ~]#
[root@k8s-10-6-162-31 ~]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY      STATUS      CLAIM                          STORAGECLA
SS          REASON      AGE
pvc-44db3536-2dc3-4b4d-bcd6-6d388a534fcd   5G         RWO
 Delete              Bound       default/local-hostpath-pvc     openebs-hostpath
            71m
[root@k8s-10-6-162-31 ~]# kubectl get -o yaml pv pvc-44db3536-2dc3-4b4d-
bcd6-6d388a534fcd |grep 'path:'
path: /var/openebs/local/pvc-44db3536-2dc3-4b4d-bcd6-6d388a534fcd
```

安装 dbench 性能测试工具

```
[root@k8s-10-6-162-31 ~]# cat fio-deploy.yaml
** NOTE: For details of params to construct an fio job, refer to this link:
** https://fio.readthedocs.io/en/latest/fio_doc.html


---
apiVersion: batch/v1
kind: Job
metadata:
generateName: dbench-
spec:
template:
spec:
containers:
- name: dbench
image: openebs/perf-test:latest
imagePullPolicy: IfNotPresent
env:
```

** storage mount point on which testfiles are created

- name: DBENCH_MOUNTPOINT
value: /data

测试 io 性能

kubectl delete pod hello-local-hostpath-pod

[root@k8s-10-6-162-31 ~]# kubectl create -f fio-deploy.yaml
job.batch/dbench-rmdqr created

[root@k8s-10-6-162-31 ~]# kubectl get pod
NAME READY STATUS RESTARTS AGE
dbench-rmdqr-qbl74 1/1 Running 0 4m59s
longhorn-iscsi-installation-2thd5 1/1 Running 1 54d
longhorn-iscsi-installation-ctqtg 1/1 Running 1 (52d ago) 54d
longhorn-iscsi-installation-mrm4h 1/1 Running 1 (52d ago) 54d

[root@k8s-10-6-162-31 ~]# kubectl logs -f dbench-729cw-nqfpt
Error from server (NotFound): pods "dbench-729cw-nqfpt" not found
[root@k8s-10-6-162-31 ~]# kubectl logs -f dbench-rmdqr-qbl74
Working dir: /data

Testing Read IOPS...
read_iops: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 40
96B-4096B, ioengine=libaio, iodepth=64
fio-3.13
Starting 1 process
read_iops: Laying out IO file (1 file / 2048MiB)

read_iops: (groupid=0, jobs=1): err= 0: pid=17: Mon Feb 6 06:27:30 2023
read: IOPS=137, BW=566KiB/s (580kB/s)(8600KiB/15193msec)

bw (KiB/s): min= 8, max= 1752, per=100.00%, avg=570.00, stdev=518.97, sampl
es=29
iops : min= 2, max= 438, avg=142.41, stdev=129.78, samples=29
cpu : usr=0.22%, sys=1.13%, ctx=1984, majf=0, minf=16
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=100.
0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.
0%
issued rwts: total=2087,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
READ: bw=566KiB/s (580kB/s), 566KiB/s-566KiB/s (580kB/s-580kB/s), io=860
0KiB (8806kB), run=15193-15193msec

Disk stats (read/write):
dm-0: ios=2475/39, merge=0/0, ticks=1076796/25702, in_queue=1114531, util=10
0.00%, aggrios=2494/51, aggrmerge=0/3, aggrticks=1094965/36042, aggrin_que
ue=1130703, aggrutil=99.98%
sda: ios=2494/51, merge=0/3, ticks=1094965/36042, in_queue=1130703, util=99.
98%

Testing Write IOPS...
write_iops: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 40
96B-4096B, ioengine=libaio, iodepth=64
fio-3.13
Starting 1 process

write_iops: (groupid=0, jobs=1): err= 0: pid=33: Mon Feb 6 06:27:49 2023

write: IOPS=199, BW=815KiB/s (835kB/s)(13.0MiB/16358msec); 0 zone resets
bw (KiB/s): min= 40, max= 3408, per=100.00%, avg=871.87, stdev=932.70, samples=30
iops : min= 10, max= 852, avg=217.87, stdev=233.18, samples=30
cpu : usr=0.17%, sys=1.05%, ctx=958, majf=0, minf=16
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=100.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
issued rwts: total=0,3271,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
WRITE: bw=815KiB/s (835kB/s), 815KiB/s-815KiB/s (835kB/s-835kB/s), io=13.0MiB (13.7MB), run=16358-16358msec

Disk stats (read/write):
dm-0: ios=0/4802, merge=0/0, ticks=0/1128001, in_queue=1129209, util=99.50%, aggrios=0/4826, aggrmerge=0/4, aggrticks=0/1147779, aggrin_queue=1147776, aggrutil=99.41%
sda: ios=0/4826, merge=0/4, ticks=0/1147779, in_queue=1147776, util=99.41%

Testing Read Bandwidth...
read_bw: (g=0): rw=randread, bs=(R) 128KiB-128KiB, (W) 128KiB-128KiB, (T) 128KiB-128KiB, ioengine=libaio, iodepth=64
fio-3.13
Starting 1 process

read_bw: (groupid=0, jobs=1): err= 0: pid=49: Mon Feb 6 06:28:09 2023

read: IOPS=128, BW=16.5MiB/s (17.3MB/s)(280MiB/16965msec)
bw (KiB/s): min= 256, max=50176, per=100.00%, avg=17994.26, stdev=14173.96, samples=31
iops : min= 2, max= 392, avg=140.48, stdev=110.71, samples=31
cpu : usr=0.18%, sys=1.66%, ctx=2092, majf=0, minf=16
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=100.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
issued rwts: total=2180,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
READ: bw=16.5MiB/s (17.3MB/s), 16.5MiB/s-16.5MiB/s (17.3MB/s-17.3MB/s), io=280MiB (294MB), run=16965-16965msec

Disk stats (read/write):
dm-0: ios=2366/3, merge=0/0, ticks=1093765/4077, in_queue=1100333, util=99.51%, aggrios=2366/6, aggrmerge=0/0, aggrticks=1096465/8183, aggrin_queue=1104639, aggrutil=99.42%
sda: ios=2366/6, merge=0/0, ticks=1096465/8183, in_queue=1104639, util=99.42%

Testing Write Bandwidth...
write_bw: (g=0): rw=randwrite, bs=(R) 128KiB-128KiB, (W) 128KiB-128KiB, (T) 128KiB-128KiB, ioengine=libaio, iodepth=64
fio-3.13
Starting 1 process

write_bw: (groupid=0, jobs=1): err= 0: pid=65: Mon Feb 6 06:28:27 2023
write: IOPS=69, BW=9453KiB/s (9680kB/s)(145MiB/15707msec); 0 zone resets
bw (KiB/s): min= 256, max=29952, per=100.00%, avg=10800.19, stdev=9114.78, samples=26
iops : min= 2, max= 234, avg=84.35, stdev=71.20, samples=26
cpu : usr=0.15%, sys=0.67%, ctx=555, majf=0, minf=16
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=100.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.9%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
issued rwts: total=0,1097,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
WRITE: bw=9453KiB/s (9680kB/s), 9453KiB/s-9453KiB/s (9680kB/s-9680kB/s), io=145MiB (152MB), run=15707-15707msec

Disk stats (read/write):
dm-0: ios=0/1499, merge=0/0, ticks=0/1047546, in_queue=1137554, util=99.39%, aggrios=0/1502, aggrmerge=0/3, aggrticks=0/1152415, aggrin_queue=1152412, aggrutil=99.25%
sda: ios=0/1502, merge=0/3, ticks=0/1152415, in_queue=1152412, util=99.25%

Testing Read Latency...
read_latency: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=4
fio-3.13
Starting 1 process

read_latency: (groupid=0, jobs=1): err= 0: pid=81: Mon Feb 6 06:28:46 2023
read: IOPS=9, BW=39.7KiB/s (40.7kB/s)(652KiB/16407msec)
slat (usec): min=25, max=224, avg=106.02, stdev=36.24
clat (usec): min=165, max=3176.6k, avg=395710.97, stdev=707730.31
lat (usec): min=227, max=3176.7k, avg=395817.82, stdev=707732.74
clat percentiles (usec):
| 1.00th=[229], 5.00th=[2147], 10.00th=[10683],
| 20.00th=[23462], 30.00th=[35390], 40.00th=[57934],
| 50.00th=[91751], 60.00th=[145753], 70.00th=[263193],
| 80.00th=[505414], 90.00th=[1652556], 95.00th=[2021655],
| 99.00th=[3170894], 99.50th=[3170894], 99.90th=[3170894],
| 99.95th=[3170894], 99.99th=[3170894]
bw (KiB/s): min= 7, max= 216, per=100.00%, avg=63.80, stdev=57.40, samples=20
iops : min= 1, max= 54, avg=15.80, stdev=14.41, samples=20
lat (usec) : 250=1.25%, 500=0.62%, 1000=1.25%
lat (msec) : 2=1.88%, 4=2.50%, 10=2.50%, 20=7.50%, 50=19.38%
lat (msec) : 100=16.25%, 250=18.12%, 500=10.00%, 750=6.25%, 1000=1.25%
cpu : usr=0.01%, sys=0.13%, ctx=162, majf=0, minf=17
IO depths : 1=0.0%, 2=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
issued rwts: total=160,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=4

Run status group 0 (all jobs):
READ: bw=39.7KiB/s (40.7kB/s), 39.7KiB/s-39.7KiB/s (40.7kB/s-40.7kB/s), io=652KiB (668kB), run=16407-16407msec

Disk stats (read/write):

dm-0: ios=195/15, merge=0/0, ticks=62870/9094, in_queue=113278, util=99.51%, aggrios=195/22, aggrmerge=0/4, aggrticks=72364/49995, aggrin_queue=122359, aggrutil=99.49%
sda: ios=195/22, merge=0/4, ticks=72364/49995, in_queue=122359, util=99.49%

Testing Write Latency...
write_latency: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=4
fio-3.13
Starting 1 process

write_latency: (groupid=0, jobs=1): err= 0: pid=97: Mon Feb 6 06:29:04 2023
write: IOPS=30, BW=123KiB/s (126kB/s)(1868KiB/15174msec); 0 zone resets
slat (usec): min=11, max=235, avg=58.23, stdev=38.98
clat (usec): min=205, max=2619.4k, avg=136822.73, stdev=357077.08
lat (usec): min=281, max=2619.5k, avg=136881.82, stdev=357079.12
clat percentiles (usec):
| 1.00th=[277], 5.00th=[383], 10.00th=[537],
| 20.00th=[10421], 30.00th=[17171], 40.00th=[23200],
| 50.00th=[32375], 60.00th=[44303], 70.00th=[68682],
| 80.00th=[92799], 90.00th=[274727], 95.00th=[784335],
| 99.00th=[1937769], 99.50th=[2365588], 99.90th=[2634023],
| 99.95th=[2634023], 99.99th=[2634023]
bw (KiB/s): min= 7, max= 688, per=100.00%, avg=184.90, stdev=186.47, samples=20
iops : min= 1, max= 172, avg=46.00, stdev=46.69, samples=20
lat (usec) : 250=0.43%, 500=9.05%, 750=2.59%
lat (msec) : 2=0.86%, 4=1.29%, 10=5.82%, 20=15.73%, 50=26.72%
lat (msec) : 100=19.40%, 250=7.54%, 500=5.39%, 750=0.43%, 1000=1.29%
cpu : usr=0.06%, sys=0.20%, ctx=222, majf=0, minf=17
IO depths : 1=0.0%, 2=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.

0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
issued rwts: total=0,464,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=4

Run status group 0 (all jobs):
WRITE: bw=123KiB/s (126kB/s), 123KiB/s-123KiB/s (126kB/s-126kB/s), io=1868KiB (1913kB), run=15174-15174msec

Disk stats (read/write):
dm-0: ios=0/475, merge=0/0, ticks=0/68658, in_queue=69443, util=99.48%, aggrios=0/475, aggrmerge=0/0, aggrticks=0/69983, aggrin_queue=69983, aggrutil=99.40%
sda: ios=0/475, merge=0/0, ticks=0/69983, in_queue=69983, util=99.40%

Testing Read Sequential Speed...
read_seq: (g=0): rw=read, bs=(R) 1024KiB-1024KiB, (W) 1024KiB-1024KiB, (T) 1024KiB-1024KiB, ioengine=libaio, iodepth=16
fio-3.13
Starting 1 thread

read_seq: (groupid=0, jobs=1): err= 0: pid=113: Mon Feb 6 06:29:28 2023
read: IOPS=17, BW=17.8MiB/s (18.7MB/s)(358MiB/20078msec)
bw (KiB/s): min= 2043, max=81920, per=100.00%, avg=24218.28, stdev=22785.58, samples=29
iops : min= 1, max= 80, avg=23.52, stdev=22.30, samples=29
cpu : usr=0.03%, sys=0.60%, ctx=351, majf=0, minf=0
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, >=64=0.

0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.7%, 8=0.0%, 16=0.3%, 32=0.0%, 64=0.0%, >=64=0.0%
issued rwts: total=343,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
READ: bw=17.8MiB/s (18.7MB/s), 17.8MiB/s-17.8MiB/s (18.7MB/s-18.7MB/s), io=358MiB (375MB), run=20078-20078msec

Disk stats (read/write):
dm-0: ios=764/10, merge=0/0, ticks=589803/16574, in_queue=609402, util=99.61%, aggrios=764/13, aggrmerge=0/1, aggrticks=592905/15938, aggrin_queue=608802, aggrutil=99.52%
sda: ios=764/13, merge=0/1, ticks=592905/15938, in_queue=608802, util=99.52%

Testing Write Sequential Speed...
write_seq: (g=0): rw=write, bs=(R) 1024KiB-1024KiB, (W) 1024KiB-1024KiB, (T) 1024KiB-1024KiB, ioengine=libaio, iodepth=16
...
fio-3.13
Starting 4 threads
write_seq: Laying out IO file (1 file / 2798MiB)

write_seq: (groupid=0, jobs=1): err= 0: pid=129: Mon Feb 6 06:29:59 2023
write: IOPS=5, BW=6679KiB/s (6839kB/s)(187MiB/28670msec); 0 zone resets
bw (KiB/s): min= 2043, max=67449, per=59.42%, avg=16001.91, stdev=15698.61, samples=22

iops : min= 1, max= 65, avg=15.50, stdev=15.23, samples=22
cpu : usr=0.04%, sys=0.17%, ctx=149, majf=0, minf=0
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, >=64=0.
0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.4%, 8=0.0%, 16=0.6%, 32=0.0%, 64=0.0%, >=64=0.
0%
issued rwts: total=0,172,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16
write_seq: (groupid=0, jobs=1): err= 0: pid=130: Mon Feb 6 06:29:59 2023
write: IOPS=6, BW=6827KiB/s (6991kB/s)(189MiB/28350msec); 0 zone resets
bw (KiB/s): min= 2048, max=57344, per=60.09%, avg=16183.05, stdev=15552.2
7, samples=21
iops : min= 2, max= 56, avg=15.67, stdev=15.15, samples=21
cpu : usr=0.04%, sys=0.18%, ctx=160, majf=0, minf=0
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, >=64=0.
0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.4%, 8=0.0%, 16=0.6%, 32=0.0%, 64=0.0%, >=64=0.
0%
issued rwts: total=0,174,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16
write_seq: (groupid=0, jobs=1): err= 0: pid=131: Mon Feb 6 06:29:59 2023
write: IOPS=6, BW=6932KiB/s (7098kB/s)(189MiB/27921msec); 0 zone resets
bw (KiB/s): min= 2048, max=55185, per=75.56%, avg=20347.18, stdev=14701.69,
 samples=17
iops : min= 2, max= 53, avg=19.71, stdev=14.20, samples=17
cpu : usr=0.03%, sys=0.19%, ctx=163, majf=0, minf=0
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, >=64=0.
0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.4%, 8=0.0%, 16=0.6%, 32=0.0%, 64=0.0%, >=64=0.

0%
issued rwts: total=0,174,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16
write_seq: (groupid=0, jobs=1): err= 0: pid=132: Mon Feb 6 06:29:59 2023
write: IOPS=6, BW=6767KiB/s (6929kB/s)(189MiB/28600msec); 0 zone resets
bw (KiB/s): min= 2048, max=67584, per=60.14%, avg=16196.64, stdev=15468.10, samples=22
iops : min= 2, max= 66, avg=15.73, stdev=15.17, samples=22
cpu : usr=0.04%, sys=0.19%, ctx=162, majf=0, minf=0
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, >=64=0.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=99.4%, 8=0.0%, 16=0.6%, 32=0.0%, 64=0.0%, >=64=0.0%
issued rwts: total=0,174,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
WRITE: bw=26.3MiB/s (27.6MB/s), 6679KiB/s-6932KiB/s (6839kB/s-7098kB/s), io=754MiB (791MB), run=27921-28670msec

Disk stats (read/write):
dm-0: ios=0/1536, merge=0/0, ticks=0/3438611, in_queue=3460488, util=99.70%, aggrios=0/1553, aggrmerge=0/0, aggrticks=0/3505870, aggrin_queue=3506177, aggrutil=99.72%
sda: ios=0/1553, merge=0/0, ticks=0/3505870, in_queue=3506177, util=99.72%

Testing Read/Write Mixed...
rw_mix: (g=0): rw=randrw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=64
fio-3.13

Starting 1 process

rw_mix: (groupid=0, jobs=1): err= 0: pid=148: Mon Feb 6 06:30:22 2023
read: IOPS=294, BW=1185KiB/s (1214kB/s)(23.5MiB/20301msec)
bw (KiB/s): min= 24, max=10088, per=100.00%, avg=2173.91, stdev=2974.57, samples=22
iops : min= 6, max= 2522, avg=543.41, stdev=743.66, samples=22
write: IOPS=93, BW=377KiB/s (386kB/s)(7656KiB/20301msec); 0 zone resets
bw (KiB/s): min= 8, max= 3000, per=100.00%, avg=754.55, stdev=998.04, samples=20
iops : min= 2, max= 750, avg=188.60, stdev=249.52, samples=20
cpu : usr=0.25%, sys=1.12%, ctx=2198, majf=0, minf=16
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=100.0%
submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%, >=64=0.0%
issued rwts: total=5971,1896,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
READ: bw=1185KiB/s (1214kB/s), 1185KiB/s-1185KiB/s (1214kB/s-1214kB/s), io=23.5MiB (24.6MB), run=20301-20301msec
WRITE: bw=377KiB/s (386kB/s), 377KiB/s-377KiB/s (386kB/s-386kB/s), io=7656KiB (7840kB), run=20301-20301msec

Disk stats (read/write):
dm-0: ios=2926/2535, merge=0/0, ticks=676178/621959, in_queue=1332944, util=99.60%, aggrios=2926/2630, aggrmerge=0/21, aggrticks=676172/1200603, aggrin_queue=1875954, aggrutil=100.00%
sda: ios=2926/2630, merge=0/21, ticks=676172/1200603, in_queue=1875954, util=100.00%

All tests complete.

==========

= Dbench Summary =

==========

Random Read/Write IOPS: 137/199. BW: 16.5MiB/s / 9453KiB/s

Average Latency (usec) Read/Write: 395817.82/136881.82

Sequential Read/Write: 17.8MiB/s / 26.3MiB/s

Mixed Random Read/Write IOPS: 294/93

如何实现 Ceph Dashboard 仪表盘

本页演示如何在 DCE 5.0 中导入并成功使用 Ceph 监控面板。

在 DCE 5.0 中部署 Rook-ceph

先部署 Rook-ceph，再部署 rook-ceph-cluster。

- 由于目前 Ceph 还没离线化支持，所以需要在工作集群的所在节点上增加代理（测试中是在 demo-dev 环境以及内网搭建的工作集群）

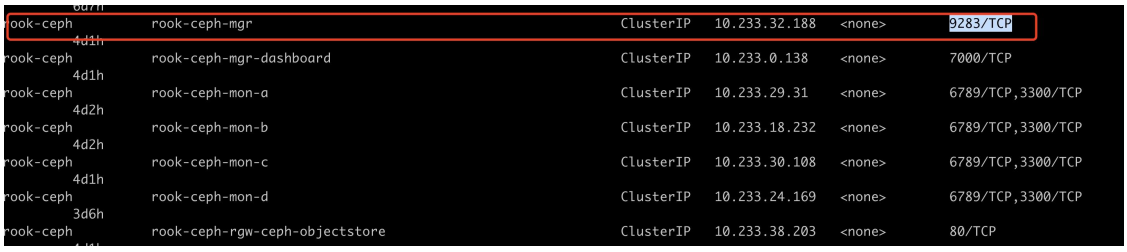
  ip r add default via 10.6.102.1 dev ens192

- 部署 rook-ceph-cluster 时由于工作节点数量有限，还需要设置：
  allowMultiplePerNode: true

在工作集群中部署 Insight Agent

采集 rook-ceph-cluster 的监控指标需要先安装 Insight Agent， 然后创建 CR ServiceMonitor 来采集 rook-ceph-cluster 的监控信息。

1. rook-ceph-cluster 的监控指标通过 9283 端口暴露。



   port 9283

2. 在工作集群中为 rook-ceph-mgr 创建 ServiceMonitor。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
```

```
    labels:
      operator.insight.io/managed-by: insight
    name: rook-ceph-sm
    namespace: rook-ceph
    spec:
    endpoints:
      - honorLabels: true
      port: http-metrics
    namespaceSelector:
      any: true
    selector:
      matchLabels:
      app: rook-ceph-mgr
      rook_cluster: rook-ceph
```

在 Global 集群中部署 GrafanaDashboard

参考 Dashboard 模板部署 GrafanaDashboard。

导入模板时请参阅 Insight 导入仪表盘。

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
metadata:
  labels:
    app: insight-grafana-operator
    operator.insight.io/managed-by: insight
  name: ceph-dashboard
  namespace: insight-system
spec:
  json: >
    {
        "__inputs": [],
        "__elements": {},
```

```
    "__requires": [
    {
        "type": "panel",
        "id": "gauge",
        "name": "Gauge",
        "version": ""
    },
    {
        "type": "grafana",
        "id": "grafana",
        "name": "Grafana",
        "version": "9.0.5"
    },
....
```


查看 Ceph 监控面板

目前的面板还未区分集群，后续将陆续优化增加 cluster 标识选项。

iShot（购买解锁去水印）

admin

可观测性

概览
仪表盘
资源监控
场景监控
数据查询
告警中心
采集管理
系统管理

仪表盘

insight-system / Ceph - Cluster

Last 12 hours | 1m

Data source: Prometheus | Interval: 10s

CLUSTER STATE

ERROR	Write Throughput	Read Throughput	Cluster Capacity	Available Capacity	Number of Objects	Bytes Written	Bytes Read
				99.6%	408		

Alerts	Write IOPS	Read IOPS	Used Capacity	Mon Session Num	Monitors In Quorum
0				30	3

OSD STATE

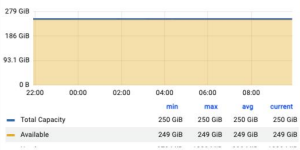
OSDs OUT	OSDs DOWN	OSDs UP	OSDs IN	Avg PGs	Avg Apply Latency	Avg Commit Latency	Avg Op Write Latency	Avg Op Read Latency
0	0	3	3	288	0 ms	0 ms	0.0499 ms	0.000337 ms

› Alerts (3 panels)

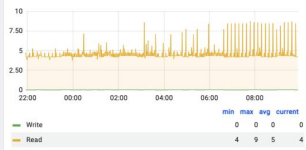
› Ceph Versions (4 panels)

CLUSTER

Capacity



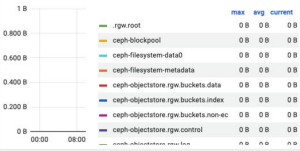
IOPS



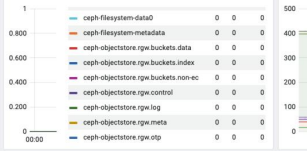
Throughput



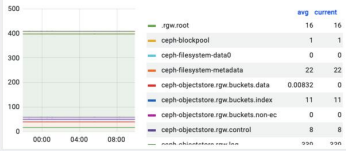
Pool Bytes Recovered



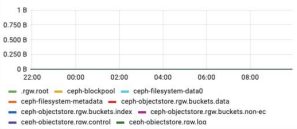
Pool Objects Recovered



Objects Per Pool



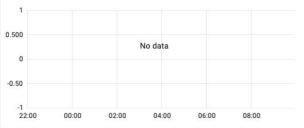
Pool Quota Bytes



Pool Objects Quota

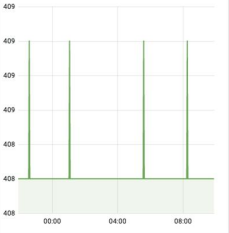


OSD Type Count

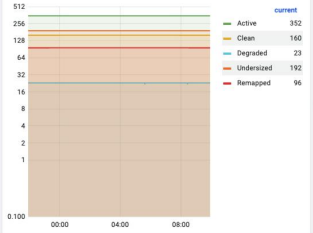


OBJECTS

Objects in the Cluster



PGs State



Stuck PGs

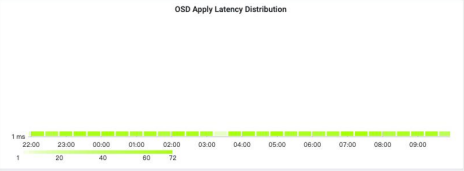


Recovery Operations

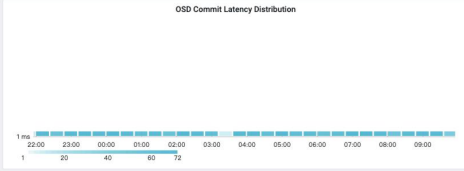


LATENCY

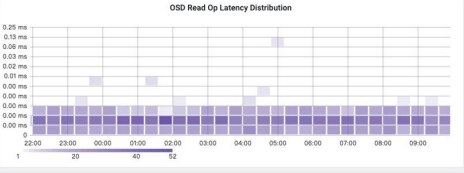
OSD Apply Latency Distribution



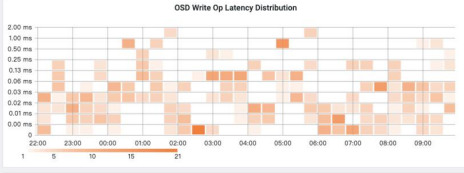
OSD Commit Latency Distribution



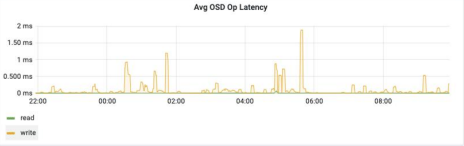
OSD Read Op Latency Distribution



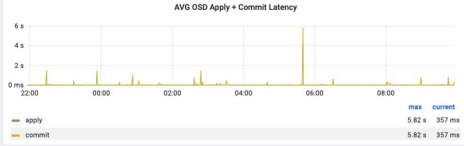
OSD Write Op Latency Distribution



Avg OSD Op Latency



AVG OSD Apply + Commit Latency



监控面板

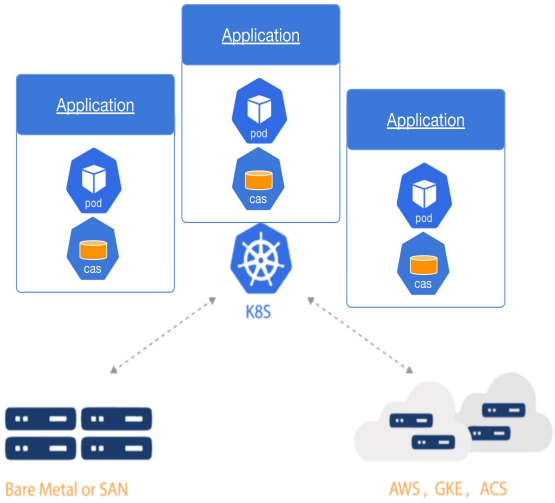
OpenEBS 存储方案

DCE 5.0 支持众多第三方存储方案，我们针对 OpenEBS 进行了相关测试，并最终将其作为 Addon 集成了应用商店中。 以下是对 OpenEBS 的调研和测评。

有关应用商店 Addon 的图形化界面安装、部署、卸载等操作说明，将于稍后提供。

什么是 CAS

容器附加存储 (CAS)是一种包含由 Kubernetes 编排的基于微服务的存储控制器的软件。 这些存储控制器可在 Kubernetes 能运行的任何地方运行，包括任何云、裸机服务器或传统共享存储系统之上。 至关重要的是，数据本身也可以通过容器访问，而不是存储在平台外共享的横向扩展存储系统中。



openebs-1

CAS 是一种非常符合非聚合数据的趋势以及运行小型、松散耦合工作负载的小型自治团队。 开发人员能保持自主并可以使用 Kubernetes 集群可用的任何存储来启动他们自己的 CAS 容器。

CAS 也反映了更广泛的解决方案趋势：通过在 Kubernetes 和微服务上构建并为基于 Kubernetes 的微服务环境提供功能来重塑特定类别的资源或创建新类别资源。 例如，安全、DNS、网络、网络策略管理、消息传递、跟踪、日志记录等新项目已经出现在云原生生态系统中。

CAS 的优势

敏捷

CAS 中的每个存储卷都有一个容器化存储控制器和对应的容器化副本。 因此，围绕这些组件的资源维护和调整是真正敏捷的。 Kubernetes 的滚动升级能力可以实现存储控制器和存储副本的无缝升级。 CPU 和内存等资源可以使用容器 cGroups 进行调整。

存储策略的粒度

将存储软件容器化并将存储控制器专用于每个卷，可以最大程度地细化存储策略。使用 CAS 架构，您可以在每个卷的基础上配置所有存储策略。 此外，您可以监控每个卷的存储参数并动态更新存储策略以获得每个工作负载的预期结果。 存储吞吐量、IOPS 和延迟的控制随着卷存储策略中粒度的增加而增加。

避免锁定

避免云供应商锁定是许多 Kubernetes 用户的共同目标。 然而，有状态应用程序的数据通常仍然依赖于云提供商和技术，或者依赖于底层的传统共享存储系统、NAS 或 SAN。 使用 CAS 方法，存储控制器可以根据工作负载在后台迁移数据，实时迁移变得更加简单。 换句话说，CAS 的控制粒度以一种无中断的方式简化了有状态工作负载从一个 Kubernetes 集群到另一个集群的移动。

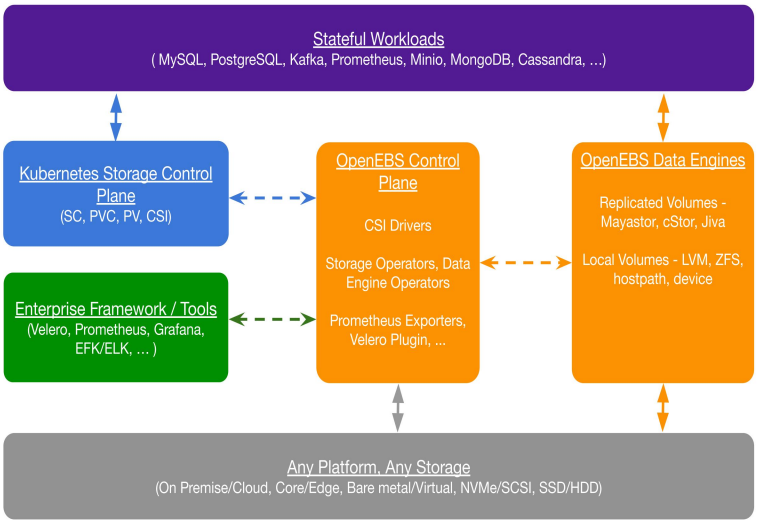
云原生

CAS 将存储软件容器化，并使用 Kubernetes 自定义资源定义（CRD）来表示底层存储资源，例如磁盘和存储池。 该模型使存储能够无缝集成到其他云原生工具中。 可以使用 Prometheus、Grafana、Fluentd、Weavescope、Jaeger 等云原生工具来配置、监控和管理存储资源。

与超融合系统类似，CAS 中卷的存储和性能是可扩展的。 由于每个卷都有自己的存储控制器，因此存储可以在节点存储容量的允许范围内扩展。 随着给定 Kubernetes 集群中容器应用程序数量的增加，将添加更多节点，从而提高存储容量和性能的整体可用性，从而使存储可用于新的应用程序容器。 这种可扩展性过程类似于 Nutanix 等成功的超融合系统。

OpenEBS 原理及架构

OpenEBS 是容器附加存储（CAS）模式的领先开源实现。 作为这种方法的一部分，OpenEBS 使用容器来动态配置卷并提供高可用性等数据服务。 OpenEBS 依赖并扩展 Kubernetes 本身来编排其卷服务。



openebs-2

OpenEBS 有很多组件，可以分为以下两大类：

- OpenEBS 数据引擎
- OpenEBS 控制平面

数据引擎

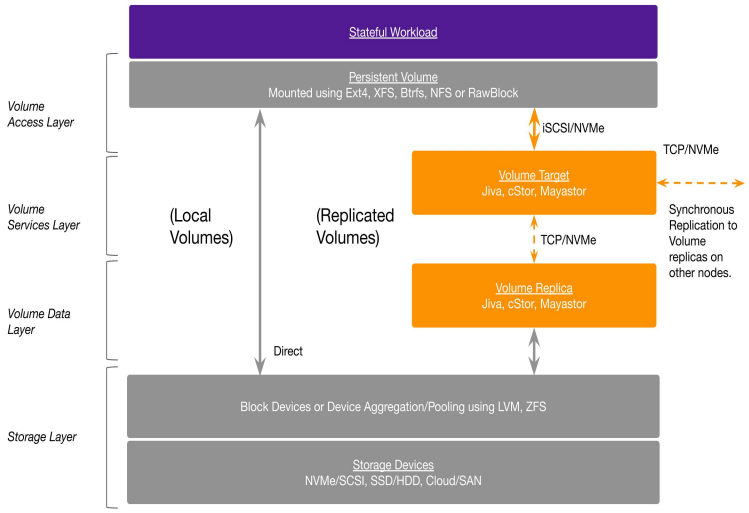
数据引擎是 OpenEBS 的核心，负责代表它们所服务的有状态工作负载对底层持久存储执行读写操作。

数据引擎负责：

- 聚合分配给它们的块设备中的可用容量，然后为应用程序划分卷。
- 提供标准系统或网络传输接口（NVMe/iSCSI）以连接到本地或远程卷
- 提供卷服务，如同步复制、压缩、加密、维护快照、访问数据的增量或完整快照等
- 在将数据持久化到底层存储设备的同时提供强一致性

OpenEBS 遵循微服务模型来实现数据引擎，其中功能进一步分解为不同的层，允许灵活地交换层并使数据引擎为未来的应用程序和数据中心技术的变化做好准备。

OpenEBS 数据引擎由以下几层组成：



openebs-3

卷访问层

有状态工作负载使用标准的 POSIX 兼容机制来执行读写操作。 根据工作负载的类型，应用程序可能更喜欢直接对原始块设备或使用标准文件系统（如 XFS、Ext4）执行读取和写入。

CSI 节点驱动程序或 Kubelet 将负责将卷附加到 Pod 运行所在的所需节点，必要时进行格式化并挂载文件系统以供 Pod 访问。 用户可以选择在此层设置挂载选项和文件系统权限，这将由 CSI 节点驱动程序或 kubelet 执行。

附加卷（使用本地、iSCSI 或 NVMe）和安装（Ext4、XFS 等）所需的详细信息可通过持久卷规范获得。

卷服务层

该层通常称为 Volume Target Layer 甚至 Volume Controller 层，因为它负责提供逻辑卷。 应用程序读取和写入是通过 Volume Targets 执行的 – 它控制对卷的访问、将数据同步复制到集群中的其他节点， 并帮助决定哪个副本充当主副本并促进将数据重建到旧的或重新启动的副本。

数据引擎用于提供高可用性的实现模式是 OpenEBS 与其他传统存储控制器的区别。 与使用单个存储控制器在多个卷上执行 IO 不同，OpenEBS 为每个卷创建一个存储控制器（称为 Target/Nexus），并具有将保存卷数据的特定节点列表。每个节点将具有使用同步复制分发的卷的完整数据。

使用单个控制器将数据同步复制到固定的节点集（而不是通过多个元数据控制器分发），减少了管理元数据的开销，还减少了与节点故障和参与重建的其他节点相关的爆炸半径失败的节点。

OpenEBS 卷服务层将卷公开为：

- 本地 PV 情况下的设备或目录路径，
- cStor 和 Jiva 情况下的 iSCSI 目标
- NVMe Target 在 Mayastor 的情况下

卷数据层

OpenEBS 数据引擎在存储层之上创建卷副本。卷副本被固定到一个节点，并在存储层之上创建。 副本可以是以下任何一项：

- 子目录 – 如果使用的存储层是文件系统目录
- 完整设备或分区设备 – 如果使用的存储层是块设备
- 逻辑卷 – 如果使用的存储层是来自 LVM 或 ZFS 的设备池。
- 如果应用程序只需要本地存储，那么将使用上述目录、设备（或分区）或逻辑卷之一创建持久卷。 OpenEBS 控制平面将用于提供上述副本之一。

OpenEBS 可以使用其复制引擎之一——Jiva、cStor 和 Mayastor，在本地存储之上添加高可用性层。 在这种情况下，OpenEBS 使用轻量级存储定义的存储控制器软件，该软件可以通过网络端点接收读/写操作， 然后传递到底层存储层。OpenEBS 然后使用此副本网络端点来跨节点维护卷的同步副本。

OpenEBS 卷副本通常经历以下状态：

- 正在初始化，在初始配置期间正在注册到其卷
- 健康，当副本可以参与读/写操作时
- 离线，当副本所在的节点或存储发生故障时
- 重建，当节点或存储故障已得到纠正并且副本正在从其他健康副本接收数据时
- 终止，当卷被删除并且副本被删除并且空间被回收时

存储层

存储层构成了持久化数据的基本构建块。存储层由连接到节点的块设备组成（通过 PCIe、SAS、NVMe 在本地或通过远程 SAN/云）。 存储层也可以是挂载文件系统之上的子目录。

存储层不在 OpenEBS 数据引擎的范围内，可用于使用标准操作系统或 Linux 软件结构的 Kubernetes 存储结构。

数据引擎将存储用作设备或设备池或文件系统目录。

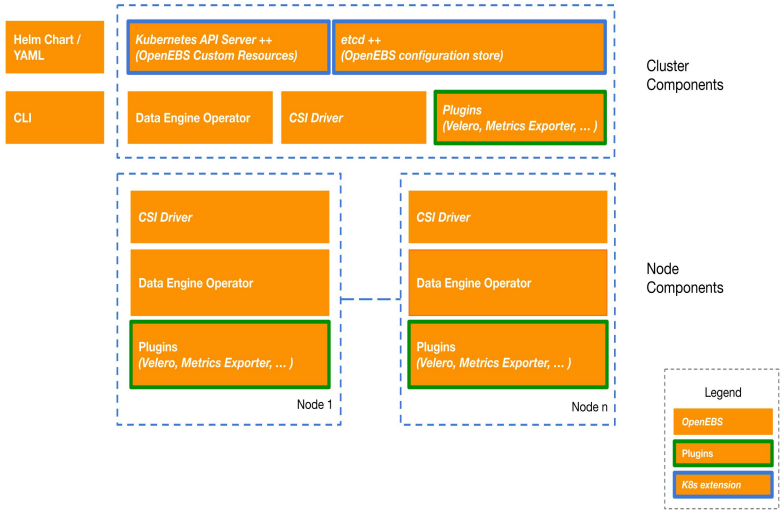
控制平面

OpenEBS 上下文中的控制平面是指部署在集群中的一组工具或组件，它们负责：

- 管理 kubernetes 工作节点上可用的存储
- 配置和管理数据引擎
- 与 CSI 接口以管理卷的生命周期
- 与 CSI 和其他工具进行接口，执行快照、克隆、调整大小、备份、恢复等操作。

- 集成到其他工具中，如 Prometheus/Grafana 以进行遥测和监控
- 集成到其他工具中进行调试、故障排除或日志管理

OpenEBS 控制平面由一组微服务组成，这些微服务本身由 Kubernetes 管理，使 OpenEBS 真正成为 Kubernetes 原生的。 由 OpenEBS 控制平面管理的配置被保存为 Kubernetes 自定义资源。控制平面的功能可以分解为以下各个阶段：



openebs-4

YAML 或 Helm Chart

管理员可以使用高度可配置的 Helm Chart 或 kubectl/YAML 安装 OpenEBS 组件。 OpenEBS 安装也通过管理 Kubernetes 产品支持，例如 OpenShift、EKS、DO、Rancher 作为市场应用程序或作为附加组件或插件紧密集成到 Kubernetes 发行版中，例如 MicroK8s、Kinvolk、Kubesphere。

作为 OpenEBS 安装的一部分，所选数据引擎的控制平面组件将使用标准 Kubernetes 原语（如 Deployments、DaemonSets、Statefulsets 等）安装为集群和/或节点组件。 OpenEBS 安装还负责将 OpenEBS 自定义资源定义加载到 Kubernetes 中。

OpenEBS 控制平面组件都是无状态的，依赖于 Kubernetes etcd 服务器（自定义资源）来管理其内部配置状态并报告各种组件的状态。

声明式 API

OpenEBS 支持声明式 API 来管理其所有操作，并且 API 公开为 Kubernetes 自定义资源。 Kubernetes CRD 验证器和 admission webhooks 用于验证用户提供的输入并验证操作是否被允许。

声明式 API 是 Kubernetes 管理员和用户习惯的自然扩展，他们可以在其中通过 YAML 定义意图，然后 Kubernetes 和相关的 OpenEBS 操作员将状态与用户的意图相协调。

声明式 API 可用于配置数据引擎和设置卷配置文件/策略。 甚至数据引擎的升级也是使用此 API 执行的。这些 API 可用于：

- 管理每个数据引擎的配置
- 管理存储需要管理的方式或存储池
- 管理卷及其服务 – 创建、快照、克隆、备份、恢复、删除
- 管理池和卷的升级

数据引擎 Operator

从发现底层存储到创建池和卷的所有数据引擎操作都打包为 Kubernetes Operators。 每个数据引擎要么在安装期间提供的配置之上运行，要么通过相应的 Kubernetes 自定义资源进行控制。

数据引擎 Operator 可以在集群范围内或在特定节点上操作。 集群范围操作员通常参与涉及与 Kubernetes 组件交互的操作 – 协调各种节点上池和卷的调度或迁移。 节点级运算符对本地操作进行操作，例如在节点上可用的存储或池上创建卷、副本、快照等。

数据引擎 Operator 通常也被称为数据引擎的控制平面，因为它们有助于管理相应数据引擎提供的卷和数据服务。 根据提供或需要的功能，某些数据引擎（如

cstor、jiva 和 mayastor）可以有多个 Operator，其中本地卷操作可以直接嵌入到相应的 CSI 控制器/配置器中。

CSI 驱动程序（动态卷配置器）

CSI 驱动程序充当管理 Kubernetes 中卷生命周期的促进者。 CSI 驱动程序操作由 StorageClass 中指定的参数控制或自定义。CSI 驱动程序由三层组成：

- Kubernetes 或 Orchestrator 功能 – Kubernetes 原生并将应用程序绑定到卷
- Kubernetes CSI 层 – 将 Kubernetes 本机调用转换为 CSI 调用 – 以标准方式将用户提供的信息传递给 CSI 驱动程序
- 存储驱动程序——它们是 CSI 投诉的，与 Kubernetes CSI 层密切合作以接收请求并处理它们。
- 存储驱动程序负责：

公开数据引擎的功能：

- 直接与数据引擎或数据引擎操作员交互以执行卷创建和删除操作
- 与数据引擎接口以将卷附加/分离到使用卷的容器正在运行的节点
- 与标准 linux 实用程序接口以格式化、安装/卸载卷到容器

Plug-in 插件

OpenEBS 专注于存储操作，并为其他流行工具提供插件，以执行核心存储功能之外的操作，但对于在生产中运行 OpenEBS 非常重要。 此类操作的示例包括：

- 应用程序一致的备份和恢复（通过集成到 Velero 中提供）
- 监控和警报（通过集成到 Prometheus、Grafana、警报管理器中提供）
- 执行安全策略（通过与 PodSecurityPolicies 或 Kyerno 的集成提供）
- 日志记录（通过与 ELK、Loki、Logstash 等管理员设置的任何标准日志记录堆栈集成提供）
- 可视化（通过标准 Kubernetes 仪表板或自定义 Grafana 仪表板提供）

CLI 命令行

OpenEBS 上的所有管理功能都可以通过 kubectl 执行，因为 OpenEBS 使用自定义资源来管理其所有配置并报告组件的状态。

此外，OpenEBS 还发布了 alpha 版 kubectl 插件，以帮助使用单个命令提供有关池和卷的信息，该命令聚合了通过多个 kubectl 命令获得的信息。

通过应用商店部署 Longhorn

本文将提供在 DCE 5.0 应用商店以 Addon 的图形化界面安部署 Longhorn 云原生存储系统的操作步骤及说明。

Longhorn helm chart 格式转换

添加 repo

```
[root@k8s-10-6-162-31 ~]# helm search repo longhorn
NAME CHART VERSION APP VERSION DESCRIPTION
longhorn/longhorn 1.3.2 v1.3.2 Longhorn is a distributed block storage system ...
```

拉取 Longhorn helm chart 并解压

```
[root@k8s-10-6-162-31 ~]# helm pull longhorn/longhorn
[root@k8s-10-6-162-31 ~]# ls
anaconda-ks.cfg calico.yaml helm-v3.10.1-linux-amd64.tar.gz linux-amd64 longh
orn-1.3.2.tgz rook rook-ceph-image rook-ceph-image.zip

[root@k8s-10-6-162-31 ~]# tar xvfz longhorn-1.3.2.tgz
longhorn/Chart.yaml
longhorn/values.yaml
longhorn/templates/NOTES.txt
longhorn/templates/_helpers.tpl
longhorn/templates/clusterrole.yaml
longhorn/templates/clusterrolebinding.yaml
longhorn/templates/crds.yaml
longhorn/templates/daemonset-sa.yaml
longhorn/templates/default-setting.yaml
longhorn/templates/deployment-driver.yaml
longhorn/templates/deployment-ui.yaml
longhorn/templates/deployment-webhook.yaml
longhorn/templates/ingress.yaml
```

longhorn/templates/postupgrade-job.yaml
longhorn/templates/psp.yaml
longhorn/templates/registry-secret.yaml
longhorn/templates/serviceaccount.yaml
longhorn/templates/services.yaml
longhorn/templates/storageclass.yaml
longhorn/templates/tls-secrets.yaml
longhorn/templates/uninstall-job.yaml
longhorn/.helmignore
longhorn/README.md
longhorn/app-readme.md
longhorn/questions.yaml

[root@k8s-10-6-162-31 ~]# cd longhorn
[root@k8s-10-6-162-31 longhorn]# ls
app-readme.md Chart.yaml questions.yaml README.md templates values.yaml

将 Longhorn 的 values.yaml 转成 json 格式

[root@k8s-10-6-162-31 ~]# helm plugin install https://github.com/karuppiah7890/helm-schema-gen.git
karuppiah7890/helm-schema-gen info checking GitHub for tag '0.0.4'
karuppiah7890/helm-schema-gen info found version: 0.0.4 for 0.0.4/Linux/x86_64
karuppiah7890/helm-schema-gen info installed ./bin/helm-schema-gen
Installed plugin: schema-gen
[root@k8s-10-6-162-31 longhorn]# helm schema-gen values.yaml > values.schema.json
[root@k8s-10-6-162-31 longhorn]# ls
app-readme.md Chart.yaml questions.yaml README.md templates values.schema.json values.yaml

将含有 json 文件的 chart 打包压缩

```
[root@k8s-10-6-162-31 longhorn]# cd ..
[root@k8s-10-6-162-31 ~]# ls
anaconda-ks.cfg helm-v3.10.1-linux-amd64.tar.gz longhorn rook rook-ceph-imag
e.zip
calico.yaml linux-amd64 longhorn-1.3.2.tgz rook-ceph-image values.schema.json
[root@k8s-10-6-162-31 ~]# tar zcvf longhorn-v1.3.2.tgz longhorn
longhorn/
longhorn/Chart.yaml
longhorn/values.yaml
longhorn/templates/
longhorn/templates/NOTES.txt
longhorn/templates/_helpers.tpl
longhorn/templates/clusterrole.yaml
longhorn/templates/clusterrolebinding.yaml
longhorn/templates/crds.yaml
longhorn/templates/daemonset-sa.yaml
longhorn/templates/default-setting.yaml
longhorn/templates/deployment-driver.yaml
longhorn/templates/deployment-ui.yaml
longhorn/templates/deployment-webhook.yaml
longhorn/templates/ingress.yaml
longhorn/templates/postupgrade-job.yaml
longhorn/templates/psp.yaml
longhorn/templates/registry-secret.yaml
longhorn/templates/serviceaccount.yaml
longhorn/templates/services.yaml
longhorn/templates/storageclass.yaml
longhorn/templates/tls-secrets.yaml
longhorn/templates/uninstall-job.yaml
longhorn/.helmignore
longhorn/README.md
```

longhorn/app-readme.md

longhorn/questions.yaml

longhorn/values.schema.json

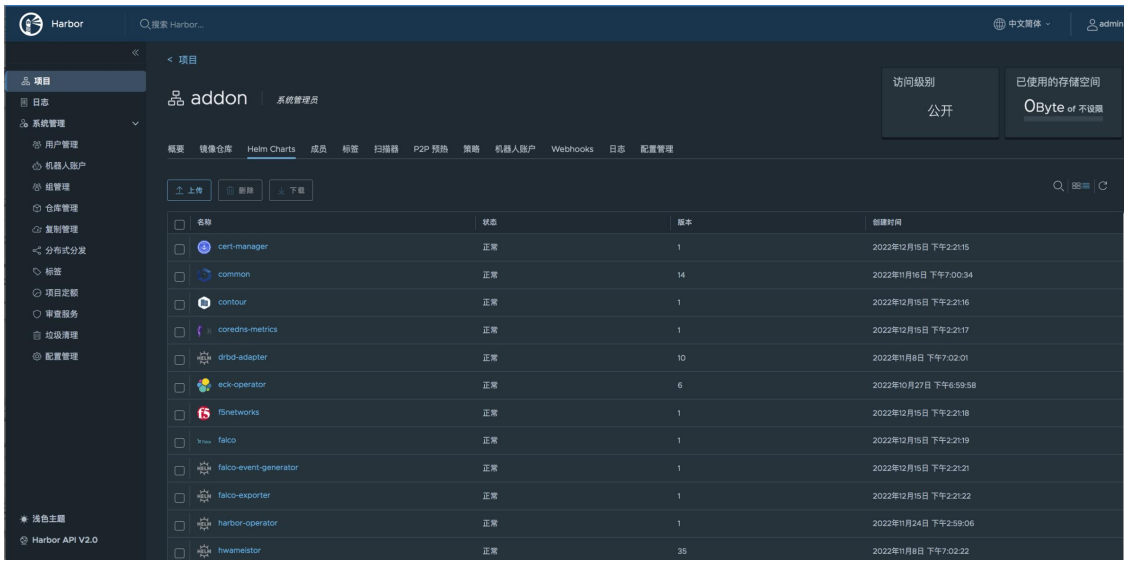
[root@k8s-10-6-162-31 ~]# ls

anaconda-ks.cfg helm-v3.10.1-linux-amd64.tar.gz longhorn longhorn-v1.3.2.tgz ro

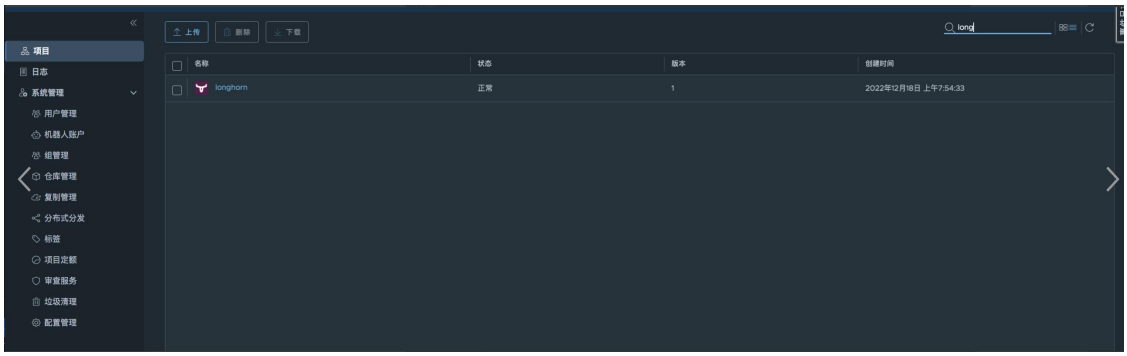
ok-ceph-image values.schema.json

calico.yaml linux-amd64 longhorn-1.3.2.tgz rook rook-ceph-image.zip

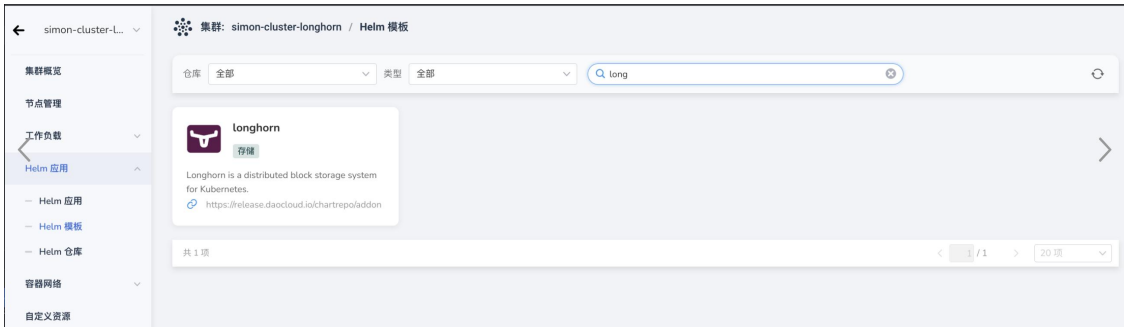
上传 chart 包至 DCE 5.0 镜像仓库



dce 镜像仓库-1

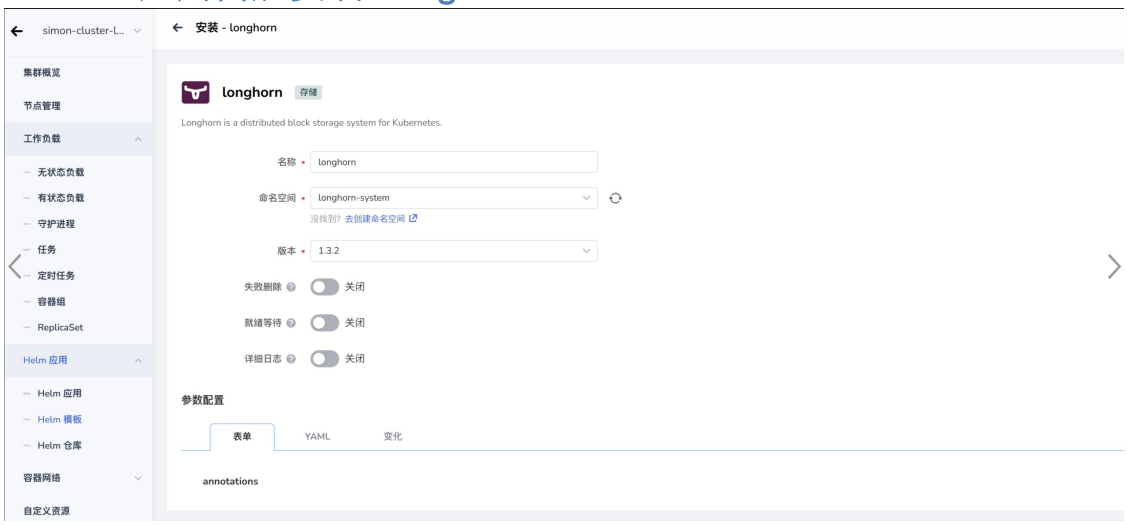


dce 镜像仓库-2



dce 应用商店-1

DCE 5.0 应用商店安装 Longhorn



dce 应用商店-2

```
[root@k8s-10-6-162-31 ~]# kubectl get po -n longhorn-system
NAME READY STATUS RESTARTS AGE
csi-attacher-7bf4b7f996-7g2h6 1/1 Running 0 3h13m
csi-attacher-7bf4b7f996-d996x 1/1 Running 1 (108m ago) 3h13m
csi-attacher-7bf4b7f996-nxwcw 1/1 Running 0 3h13m
csi-provisioner-869bdc4b79-4xgbs 1/1 Running 1 (108m ago) 3h13m
csi-provisioner-869bdc4b79-8bjb5 1/1 Running 0 3h13m
csi-provisioner-869bdc4b79-zmqkp 1/1 Running 0 3h13m
csi-resizer-6d8cf5f99f-lfvhq 1/1 Running 1 (108m ago) 3h13m
csi-resizer-6d8cf5f99f-lzkd4 1/1 Running 0 3h13m
csi-resizer-6d8cf5f99f-zcd6g 1/1 Running 0 3h13m
```

csi-snapshotter-588457fcdf-crddw 1/1 Running 1 (108m ago) 3h13m
csi-snapshotter-588457fcdf-qnghk 1/1 Running 0 3h13m
csi-snapshotter-588457fcdf-xbl4q 1/1 Running 0 3h13m
engine-image-ei-a5371358-qtbzf 1/1 Running 0 3h14m
engine-image-ei-a5371358-sgrkv 1/1 Running 0 3h14m
engine-image-ei-a5371358-t8gd6 1/1 Running 0 3h14m
helm-operation-install-longhorn-deploy-jf5rm-hdx82 0/1 Completed 0 3h20m
instance-manager-e-2894727b 1/1 Running 0 3h14m
instance-manager-e-c285811f 1/1 Running 0 3h14m
instance-manager-e-c33ef405 1/1 Running 0 3h14m
instance-manager-r-5eed6e09 1/1 Running 0 3h14m
instance-manager-r-7aea1541 1/1 Running 0 3h14m
instance-manager-r-832995c5 1/1 Running 0 3h14m
longhorn-admission-webhook-84dbdf4b-7k9pf 1/1 Running 0 3h20m
longhorn-admission-webhook-84dbdf4b-w9xjz 1/1 Running 0 3h20m
longhorn-conversion-webhook-77f447c97b-fj52b 1/1 Running 0 3h20m
longhorn-conversion-webhook-77f447c97b-fxqck 1/1 Running 0 3h20m
longhorn-csi-plugin-mvwwd 2/2 Running 0 3h13m
longhorn-csi-plugin-nj6pg 2/2 Running 0 3h13m
longhorn-csi-plugin-t8smg 2/2 Running 0 3h13m
longhorn-driver-deployer-7cd5cfcd64-lsjnm 1/1 Running 0 3h20m
longhorn-manager-lwrxj 1/1 Running 0 3h20m
longhorn-manager-x7vgm 1/1 Running 0 3h20m
longhorn-manager-xsn8k 1/1 Running 1 (3h14m ago) 3h20m
longhorn-ui-68bc57db67-46brf 1/1 Running 0 3h20m

修改 Longhorn 前端 service 端口：

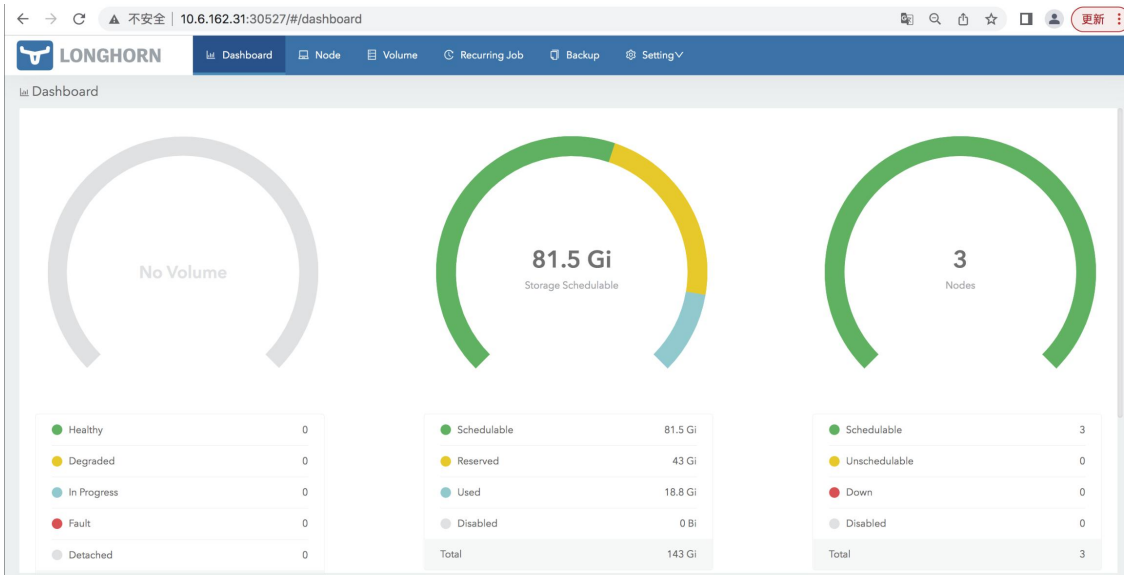
```
[root@k8s-10-6-162-31 ~]# kubectl edit svc longhorn-frontend -n longhorn-system -oyaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    meta.helm.sh/release-name: longhorn-deploy
    meta.helm.sh/release-namespace: longhorn-system
  creationTimestamp: "2022-12-19T02:08:30Z"
  labels:
    app: longhorn-ui
    app.kubernetes.io/instance: longhorn-deploy
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: longhorn
    app.kubernetes.io/version: v1.3.2
    helm.sh/chart: longhorn-1.3.2
  name: longhorn-frontend
  namespace: longhorn-system
  resourceVersion: "484650"
  uid: 8b7b5891-af86-41d4-b9ab-c2dc67812379
spec:
  clusterIP: 10.96.255.100
  clusterIPs:
  - 10.96.255.100
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    nodePort: 30527
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app: longhorn-ui
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

longhorn-svc-1

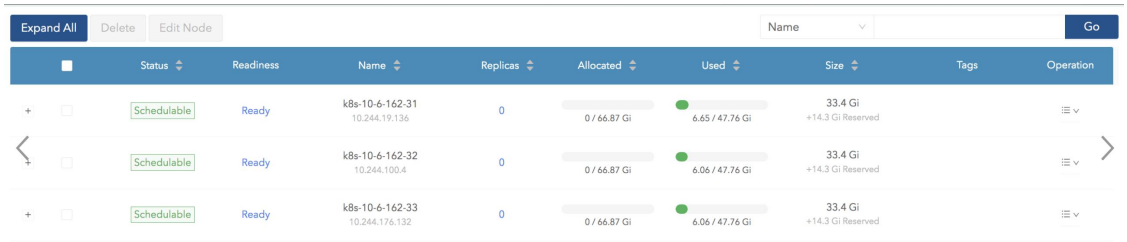
```
[root@k8s-10-6-162-31 ~]# kubectl get svc -A
NAMESPACE         NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                  AGE
default           dao-2048-1                    ClusterIP   10.102.6.66      <none>        80/TCP                   3d1h
default           kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP                  55d
kube-system       kube-dns                      ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP,9153/TCP   55d
longhorn-system   csi-attacher                  ClusterIP   10.105.81.243    <none>        12345/TCP                176m
longhorn-system   csi-provisioner               ClusterIP   10.104.50.241    <none>        12345/TCP                176m
longhorn-system   csi-resizer                   ClusterIP   10.104.71.174    <none>        12345/TCP                176m
longhorn-system   csi-snapshotter               ClusterIP   10.98.3.30       <none>        12345/TCP                176m
longhorn-system   longhorn-admission-webhook    ClusterIP   10.106.113.117   <none>        9443/TCP                 3h3m
longhorn-system   longhorn-backend              ClusterIP   10.101.182.137   <none>        9500/TCP                 3h3m
longhorn-system   longhorn-conversion-webhook   ClusterIP   10.108.25.76     <none>        9443/TCP                 3h3m
longhorn-system   longhorn-engine-manager       ClusterIP   None             <none>        <none>                   3h3m
longhorn-system   longhorn-frontend             NodePort    10.96.255.100    <none>        80:30527/TCP             3h3m
longhorn-system   longhorn-replica-manager      ClusterIP   None             <none>        <none>                   3h3m
```

longhorn-svc-2

进入 Longhorn UI：



longhorn 界面



longhorn 界面

至此，在 DCE 5.0 应用商店成功部署了 Longhorn 存储系统！

常见问题

问题 1：HwameiStor 本地存储调度器 scheduler 在 Kubernetes 平台中是如何工作的?

HwameiStor 的调度器是以 Pod 的形式部署在 HwameiStor 的命名空间。

```
[root@dce-10-6-163-21 ~]# kubectl get po -nhwameistor
NAME                                        READY   STATUS    RESTARTS   AGE
hwameistor-csi-controller-775f588656-nnnjl  3/3     Running   9          58d
hwameistor-local-disk-manager-2dktz         1/1     Running   0          58d
hwameistor-local-disk-manager-gptwt         1/1     Running   0          58d
hwameistor-local-disk-manager-jfbm8         1/1     Running   0          58d
hwameistor-local-disk-manager-nn8qm         1/1     Running   0          58d
hwameistor-local-disk-manager-xkxx5         1/1     Running   0          58d
hwameistor-local-storage-kjjsb              2/2     Running   0          56d
hwameistor-local-storage-qfrwv              2/2     Running   0          56d
hwameistor-scheduler-7bf84f8b6b-lsfb9       1/1     Running   0          39h
[root@dce-10-6-163-21 ~]#
```

img

当应用（Deployment 或 StatefulSet ）被创建后，应用的 Pod 会被自动部署到已配置好具备 HwameiStor 本地存储能力的 Worker 节点上。

问题 2: HwameiStor 如何应对应用多副本工作负载的调度？与传统通用型共享存储有什么不同?

HwameiStor 建议使用有状态的 StatefulSet 用于多副本的工作负载。

有状态应用 StatefulSet 会将复制的副本部署到同一 Worker 节点，但会为每一个 Pod 副本创建一个对应的 PV 数据卷。如果需要部署到不同节点分散 workload，需要通过 pod affinity 手动配置。

```
[root@dce-10-6-163-21 ~]# kubectl get po -owide
NAME                                                READY   STATUS    RESTARTS   AGE    IP              NODE              NOMINATED NODE   READINESS GATES
dao-2048-cal-defaultnm-1-dao-2048-66f9956479-4wfwg  1/1     Running   0          82d    172.29.116.68   dce-10-6-163-25   <none>           <none>
dao-2048-cal-defaultnm-1-dao-2048-66f9956479-7lcr8  1/1     Running   0          82d    172.29.116.67   dce-10-6-163-25   <none>           <none>
dao-2048-cal-defaultnm-1-dao-2048-66f9956479-bbvsh  1/1     Running   0          82d    172.29.171.4    dce-10-6-163-24   <none>           <none>
dao-2048-cal-defaultnm-1-dao-2048-66f9956479-pqlt7  1/1     Running   0          120d   172.29.171.1    dce-10-6-163-24   <none>           <none>
dao-2048-cal-defaultnm-1-dao-2048-66f9956479-qvjjp  1/1     Running   0          120d   172.29.116.66   dce-10-6-163-25   <none>           <none>
dao-before-dao-2048-68cbbd865f-f2cqb                1/1     Running   0          82d    172.29.116.69   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm-5fd47ff85f-2qmxj            1/1     Running   0          23h    172.29.116.90   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm-5fd47ff85f-n7t2q            1/1     Running   0          24h    172.29.116.87   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm-sts-0                       1/1     Running   0          14h    172.29.116.98   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm-sts-1                       1/1     Running   0          14h    172.29.116.99   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm2-5849f7b887-h4ktj           1/1     Running   0          21h    172.29.116.91   dce-10-6-163-25   <none>           <none>
nginx-local-storage-lvm2-5849f7b887-tgx7q           1/1     Running   0          21h    172.29.116.92   dce-10-6-163-25   <none>           <none>
[root@dce-10-6-163-21 ~]#
[root@dce-10-6-163-21 ~]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                                          STORAGECLASS          REASON   AGE
dce-registry                               100Gi      RWX            Retain           Bound    kube-system/dce-registry                                                     125d
pvc-32a3b31e-c2bd-4657-9348-f69f11d19cf4   1Gi        RWO            Delete           Bound    default/www-nginx-local-storage-lvm-sts-1      local-storage-hdd-lvm   14h
pvc-6b1cf31a-9de0-4da0-a572-9fe54819a2f4   1Gi        RWO            Delete           Bound    default/www-nginx-local-storage-lvm-sts-0      local-storage-hdd-lvm   14h
pvc-a7552471-361a-43d9-841f-88fa2253e393   1Gi        RWO            Delete           Bound    default/local-storage-pvc-lvm                  local-storage-hdd-lvm   39h
pvc-d930c316-01da-4108-8a3b-bf7ee5455215   1Gi        RWO            Delete           Bound    default/local-storage-pvc-lvm2                 local-storage-hdd-lvm   21h
[root@dce-10-6-163-21 ~]#
```

img

由于无状态应用 deployment 不能共享 block 数据卷，所以建议使用单副本。

对于传统通用型共享存储：

有状态应用 statefulSet 会将复制的副本优先部署到其他节点以分散 workload，但会为每一个 Pod 副本创建一个对应的 PV 数据卷。 只有当副本数超过 Worker 节点数的时候会出现多个副本在同一个节点。

无状态应用 deployment 会将复制的副本优先部署到其他节点以分散 workload，并且所有的 Pod 共享一个 PV 数据卷（目前仅支持 NFS）。 只有当副本数超过 Worker 节点数的时候会出现多个副本在同一个节点。对于 block 存储，由于数据卷不能共享，所以建议使用单副本。

Hwameistor Release Notes

本页列出 Hwameistor 相关的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-7-05

v0.11.1

新功能

新增 支持自动检测 cgroup 版本

2023-6-25

v0.11.0

新功能

新增 实现 IO 限制或 QoS

新增 使用 /virtual/ 检测来识别虚拟块设备

优化

修复 存储池创建时间字段不一致问题

2023-5-26

v0.10.3

优化

优化 Helm 模板中权限功相关内容

2023-5-25

v0.10.2

优化

优化 Node Name . 自动转成 – 问题，影响 LDM 心跳检测

新增 Admin 权限功能

2023-5-17

v0.10.0

优化

- **新增** 识别并设置本地磁盘 Manager Owner 信息
- **优化** 当接收到 移除时间，将磁盘状态标记为‘inactive’
- **优化** 磁盘 Smart 指标仪表盘
- **新增** 本地存储池
- **优化** UI 组件状态展示
- **优化** 分离磁盘分配和磁盘状态更新过程
- **优化** 将 Exportor 端口重命名为 http-metrics
- **优化** 在 exporter service 中添加端口名称

缺陷

- **修复** LD bound，但是 LSN 中没有容量的问题
- **修复** Metrics 端口监听问题
- **修复** May occur not found 的问题
- **修复** Helm 中 UI tag 的问题

v0.9.3

优化

- **优化** 在使用阶段填充磁盘所有者信息
- **优化** udev 事件触发时合并磁盘自身属性
- **优化** 给 svc 添加标签
- **优化** 分离磁盘分配和磁盘状态更新过程
- **优化** 将 Exportor 端口重命名为 http-metrics
- **优化** 在 exporter service 中添加端口名称

缺陷

- **修复** LD bound，但是 LSN 中没有容量的问题
- **修复** Metrics 端口监听问题
- **修复** May occur not found 的问题
- **修复** Helm 中 UI tag 的问题

2023-3-30

v0.9.2

优化

- **新增** UI relok8s

v0.9.1

优化

- **新增** Volume Status 监控 Issue #741
- **修复** Local Storage 部署参数 Issue #742

v0.9.0

新功能

- **新增** 磁盘 Owner Issue #681
- **新增** Grafana DashBoard Issue #733
- **新增** Operator 安装方式，安装时自动拉取 UI Issue #679
- **新增** UI 应用 Label Issue #710

优化

- **新增** 磁盘的已使用容量 Issue #681
- **优化** 当未发现可用磁盘时，跳过打分机制 Issue #724
- **设置** DRDB 端口默认为 43001 Issue #723

2023-1-30

v0.8.0

优化

- **优化** 中文文档
- **优化** value.yaml 文件
- **更新** Roadmap
- **优化** 当安装失败时，设置默认的失败策略

2022-12-30

v0.7.1

新功能

- **新增** Hwameistor DashBoard UI，可展现存储资源、存储节点等使用状态
- **新增** 界面管理 Hwameistor 存储节点、本地磁盘、迁移记录
- **新增** 存储池管理功能，支持界面展现存储池基本信息，及存储池对应节点信息
- **新增** 本地卷管理功能 ，支持界面执行数据卷迁移、高可用转换

优化

- **优化** 数据迁移前不必要的日志，并规避其他 Namespace 下的 Job 执行影响

卸载 Hwameistor

本章节介绍了两种卸载 HwameiStor 系统的方式。

!!! danger

请务必先备份好所有数据，再卸载 HwameiStor。

方式一：卸载并保留已有数据

如果想要卸载 HwameiStor 的系统组件，但是保留已经创建的数据卷并服务于数据应用，采用下列方式：

```
$ kubectl get cluster.hwameistor.io
NAME                AGE
cluster-sample      21m

$ kubectl delete clusters.hwameistor.io  hwameistor-cluster
```

最终，所有的 HwameiStor 系统组件（Pods）将被删除。用下列命令检查，结果为空。

```
kubectl -n hwameistor get pod
```

方式二：完全卸载

如果想要卸载 HwameiStor 所有组件，并删除所有数据卷及数据，采用下列方式，请谨慎操作。

1. 清理有状态数据应用
    1. 删除应用
    2. 删除数据卷 PVC

        相关的 PV、LV、LVR、LVG 都将被删除。

2. 清理 HwameiStor 系统组件

    1. 删除 HwameiStor 组件

        ```
        kubectl delete clusters.hwameistor.io  hwameistor-cluster
        ```

    2. 删除 HwameiStor 系统空间

        ```
        kubectl delete ns hwameistor
        ```

    3. 删除 CRD、Hook 以及 RBAC

        ```
        kubectl get crd,mutatingwebhookconfiguration,clusterrolebinding,clusterrole -o name \
          | grep hwameistor \
          | xargs -t kubectl delete
        ```

    4. 删除 StorageClass

        ```
        kubectl get sc -o name \
          | grep hwameistor-storage-lvm- \
          | xargs -t kubectl delete
        ```

    5. 删除 hwameistor Operator

        ```
        helm uninstall hwameistor-operator -n hwameistor
        ```

最后，你仍然需要清理每个节点上的 LVM 配置，并采用额外的系统工具（例如 wipefs）清除磁盘上的所有数据。

存储网卡配置

HwameiStor 支持使用单独的网卡进行数据卷同步，可以避免使用通信网卡带来流量阻塞问题。

!!! note

– 【��提前配置】：存储网卡的配置属于存储系统的前置性配置，建议在 HwameiStor 系统安装前 **提前配置**。
– 【运行中配置】：如果 HwameiStor 已经部署，后面进行上述配置修改，那么**对之前已经创建出来的数据卷不会生效**，也就是还会使用之前的网卡进行数据卷同步。
– 如需要修改多个节点存储网卡，请逐个配置，目前无法批量配置

前提条件

已经提前完成存储网卡规划，请参考网卡规划。

配置步骤

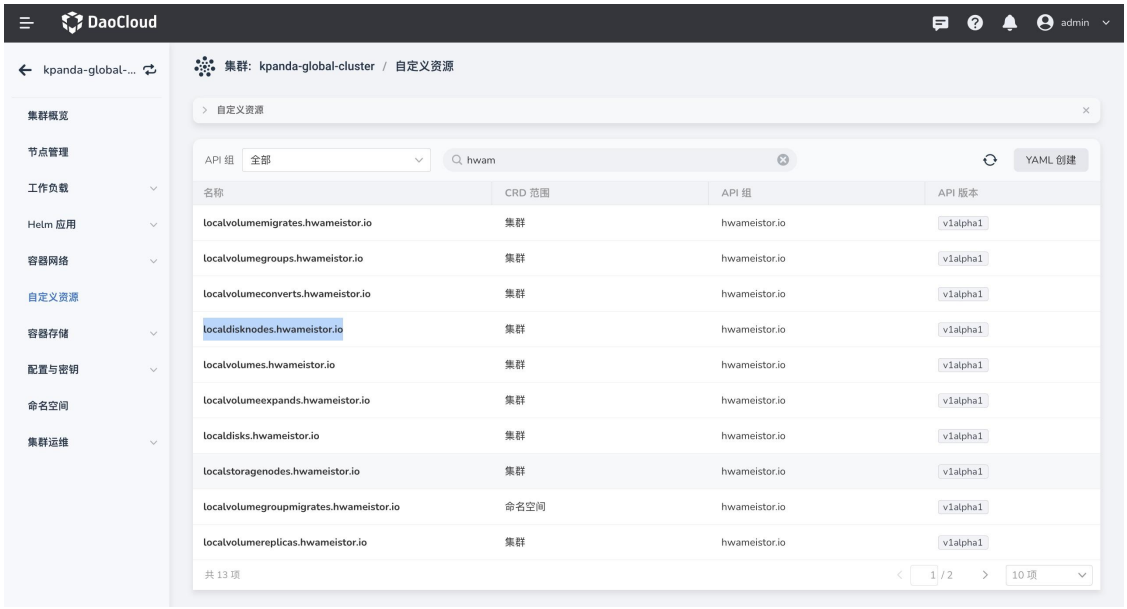
配置方式分为 2 种：

1. 通过 LocalStorage CR 配置
2. 通过节点注释标记

修改 LocalStorage CR 配置

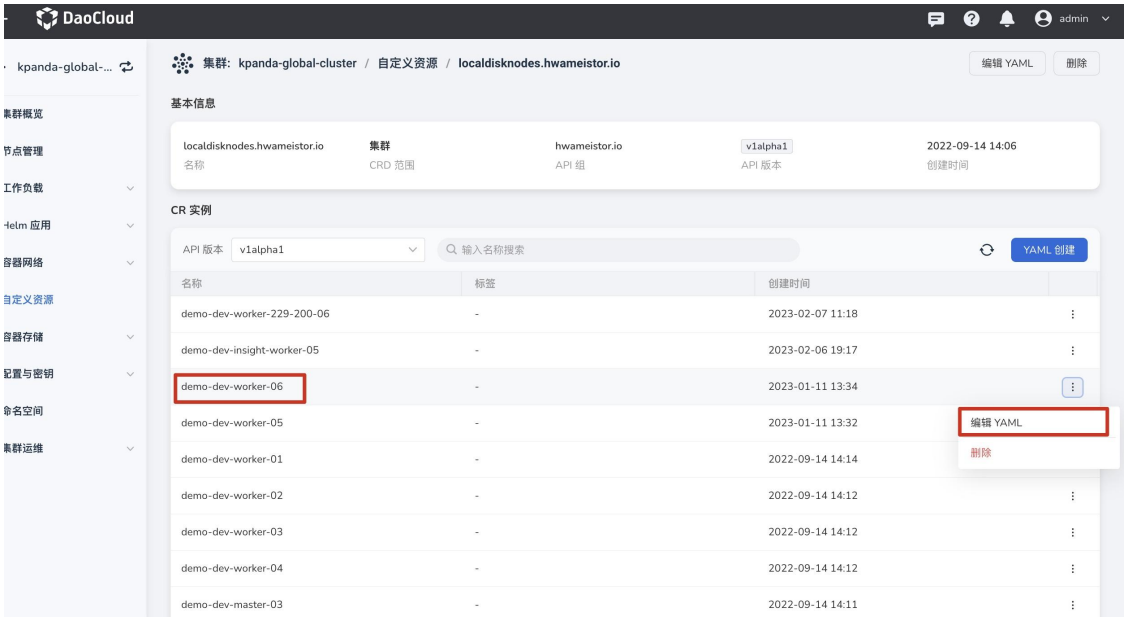
1. 在左侧导航栏点击 容器管理 —> 集群列表，找到待修改网卡配置的集群，进入集群详情 。

2. 在左侧导航栏中选择 自定义资源，找到 localdisknodes.hwameistor.io，点击进入详情。

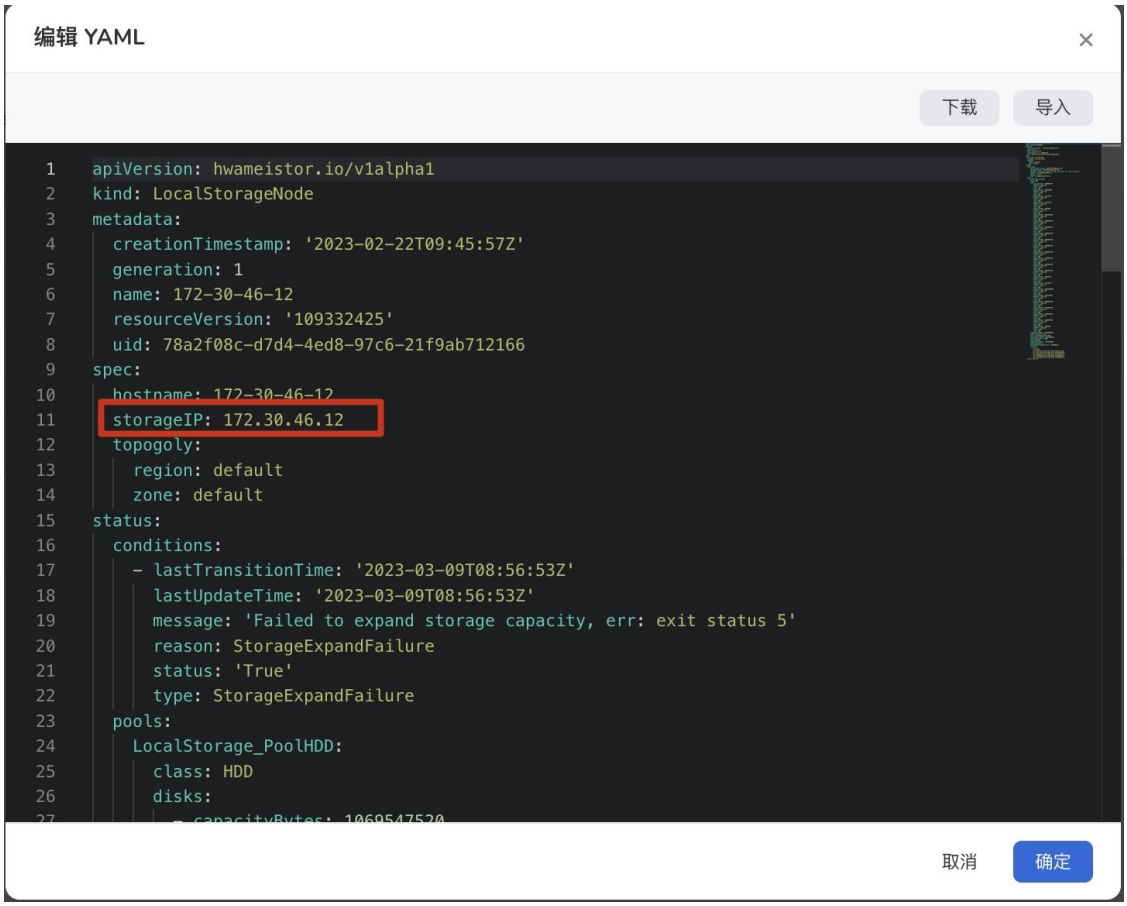


Ethedit01

3. 找到待修改节点并点击编辑 YAML，修改 spec 中的 storage-ipv4=172.30.40.12 参数， 指定 IP 地址为已规划网卡 IP 网卡规划。



ethedit02



ethedit03

4. 完成后点击保存，并选择下一个节点进行修改。

通过节点注释标记

1. 查看 local-storage 的 ENV:

   NODE_ANNOTATION_KEY_STORAGE_IPV4 的值，默认是 **localstorage.hwameistor.io/storage-ipv4**

2. 将节点上的 **存储网卡地址通过注释的方式标记**

   kubectl annotate node <your_storage_node> localstorage.hwameistor.io``/storage-ipv4``=172.30.46.12

3. **重启** 节点上的 local-storage 服务

4. **验证** 配置是否生效

```
kubectl get lsn <your_storage_node> -o yaml |``grep` `-i storageIP
```

5. 修改成功后，进行下一个节点修改。

生产环境资源要求

如果在生产环境中部署 Hwameistor，请指定资源配置，并且避免部署到 Master 节点上，因此 values.extra.prod.yaml 文件中提供了一些推荐值，资源配置如下：

??? note “点击查看 values.extra.prod.yaml”

```yaml title="values.extra.prod.yaml"
scheduler:
  replicas: 3
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 1m
      memory: 20Mi

admission:
  replicas: 3
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 1m
      memory: 20Mi

evictor:
  replicas: 2
  resources:
```

```
      limits:
        cpu: 300m
        memory: 300Mi
      requests:
        cpu: 1m
        memory: 20Mi

metrics:
  replicas: 2
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 1m
      memory: 20Mi

apiserver:
  replicas: 2
  resources:
    limits:
      cpu: 300m
      memory: 300Mi
    requests:
      cpu: 1m
      memory: 20Mi

localDiskManager:
  tolerationsOnMaster: false
  registrar:
    resources:
      limits:
```

```
            cpu: 500m
            memory: 500Mi
          requests:
            cpu: 1m
            memory: 20Mi
    manager:
      resources:
        limits:
          cpu: 300m
          memory: 300Mi
        requests:
          cpu: 1m
          memory: 20Mi

localDiskManagerCSIController:
  replicas: 3
  priorityClassName: system-node-critical
  provisioner:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
        memory: 20Mi
  attacher:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
```

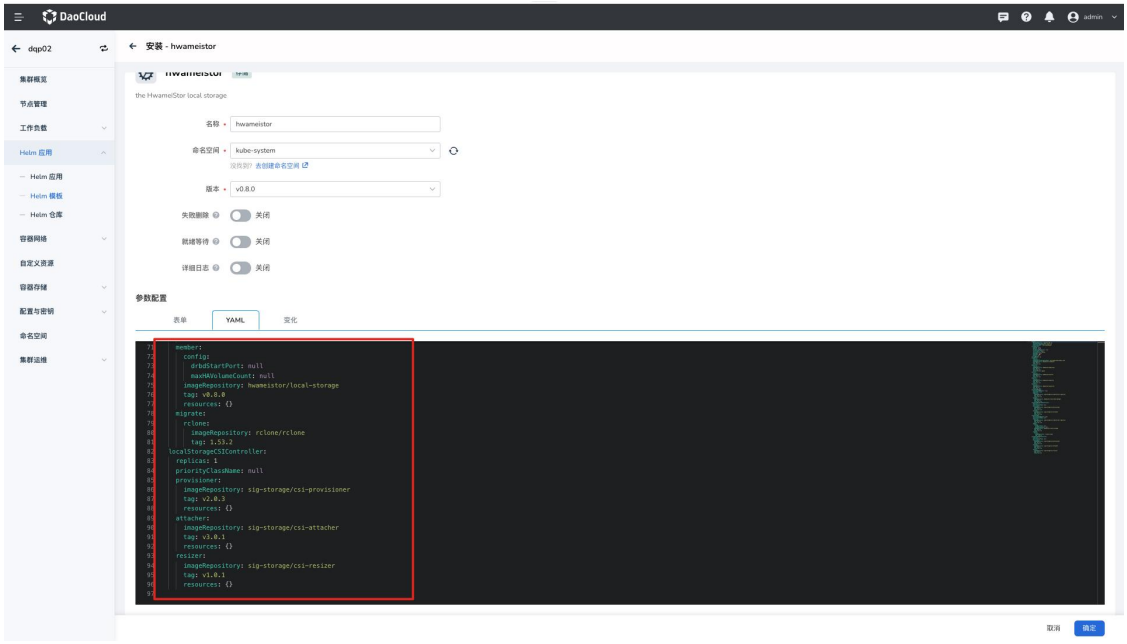
```
        memory: 20Mi

localStorage:
  tolerationsOnMaster: false
  priorityClassName: system-node-critical
  registrar:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
        memory: 20Mi
  member:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
        memory: 20Mi

localStorageCSIController:
  replicas: 3
  priorityClassName: system-node-critical
  provisioner:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
```

```
        memory: 20Mi
  attacher:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
        memory: 20Mi
  resizer:
    resources:
      limits:
        cpu: 500m
        memory: 500Mi
      requests:
        cpu: 1m
        memory: 20Mi
```

1. 如通过 Helm 创建时可通过如下方式修改资源：

   ```
   helm install hwameistor ./hwameistor \
       -n hwameistor --create-namespace \
       -f ./hwameistor/values.yaml \
       -f ./hwameistor/values.extra.prod.yaml
   ```

2. 如通过 UI 界面安装，请手动将如上资源通过 YAML 中的 Resource 值进行配置，否则默认不配置：



pro-Resource

Hwameistor 安装

本文介绍在平台界面通过 Hwameistor Operator 安装 Hwameistor，Operator 安装后会自动将 Hwameistor 相关组件拉起。HwameiStor Operator 负责如下事项：

- 所有组件的全生命周期管理（LCM）：
  - LocalDiskManager
  - LocalStorage
  - Scheduler
  - AdmissionController
  - VolumeEvictor
  - Exporter
  - Apiserver
  - Graph UI
- 根据不同目的和用途配置节点磁盘
- 自动发现节点磁盘的类型，并以此自动创建 HwameiStor 存储池
- 根据 HwameiStor 系统的配置和功能自动创建相应的 StorageClass

前提条件

- 待使用节点上已准备空闲 HDD、SSD 磁盘。
- 已完成准备工作中事项。
- 如部署环境为生产环境，请提前阅读生产环境资源要求。
- 如果您的 Kubernetes 发行版使用不同的 kubelet 目录，请提前确认 kubeletRootDir。 详细信息请参考自定义 Kubelet 根目录。

!!! info

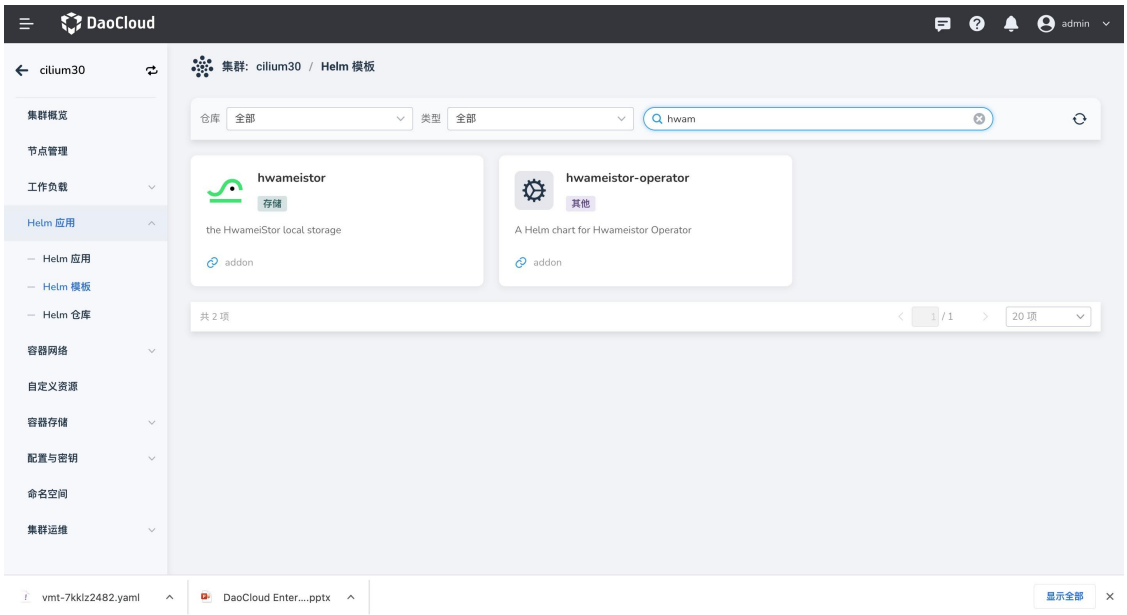
如果没有可用的干净磁盘，Operator 不会自动创建 StorageClass。
Operator 会在安装过程中自动纳管磁盘，可用的磁盘会被添加到 LocalStorage 的 pool 里。
如果可用磁盘是在安装后提供的，则需要手动下发 LocalDiskClaim 将磁盘纳管到 LocalStorageNode 里。
一旦 LocalStorageNode 的 pool 里有磁盘，Operator 就会自动创建 StorageClass。
也就是说，如果没有容量，就不会自动创建 StorageClass。

安装步骤

请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Hwameistor。

1. 在左侧导航栏点击 容器管理 —> 集群列表，然后找到准备安装 Hwameistor 的集群名称。

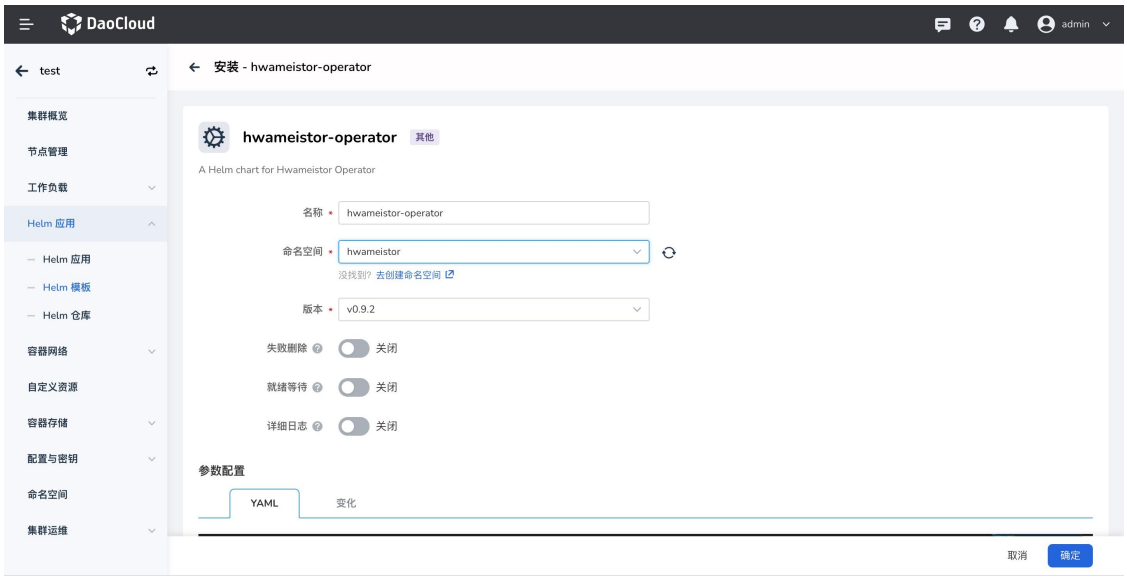
2. 在左侧导航栏中选择 Helm 应用 –> Helm 模板，找到并点击 Hwameistor Operator。



点击

3. 在版本选择中选择希望安装的版本，点击安装。

4. 在安装界面，填写所需的安装参数。



Operator02

Value.yaml 参数如下，默认可不进行修改：

```yaml
global:
  targetNamespace: hwameistor
  hwameistorImageRegistry: ghcr.io
  k8sImageRegistry: registry.k8s.io
  hwameistorVersion: v0.9.2
operator:
  replicas: 1
  imageRepository: hwameistor/operator
  tag: ''
localDiskManager:
  tolerationOnMaster: true
  kubeletRootDir: /var/lib/kubelet
  manager:
    imageRepository: hwameistor/local-disk-manager
    tag: ''
  csi:
    registrar:
```

```
      imageRepository: sig-storage/csi-node-driver-registrar
      tag: v2.5.0
    provisioner:
      imageRepository: sig-storage/csi-provisioner
      tag: v2.0.3
    attacher:
      imageRepository: sig-storage/csi-attacher
      tag: v3.0.1
localStorage:
  tolerationOnMaster: true
  kubeletRootDir: /var/lib/kubelet
  member:
    imageRepository: hwameistor/local-storage
    tag: ''
  csi:
    registrar:
      imageRepository: sig-storage/csi-node-driver-registrar
      tag: v2.5.0
    provisioner:
      imageRepository: sig-storage/csi-provisioner
      tag: v2.0.3
    attacher:
      imageRepository: sig-storage/csi-attacher
      tag: v3.0.1
    resizer:
      imageRepository: sig-storage/csi-resizer
      tag: v1.0.1
    monitor:
      imageRepository: sig-storage/csi-external-health-monitor-controller
      tag: v0.8.0
scheduler:
  imageRepository: hwameistor/scheduler
```

```
  tag: ''
admission:
  imageRepository: hwameistor/admission
  tag: ''
evictor:
  imageRepository: hwameistor/evictor
  tag: ''
apiserver:
  imageRepository: hwameistor/apiserver
  tag: ''
exporter:
  imageRepository: hwameistor/exporter
  tag: ''
ui:
  imageRepository: hwameistor/hwameistor-ui
  tag: ''
ha:
  module: drbd
  deployOnMaster: 'yes'
```

- hwameistorImageRegistry：

  设置 Hwameistor 镜像的仓库地址，默认已经填写了可用的在线仓库。 如果是私有化环境，可修改为私有仓库地址。

- K8s image Registry：

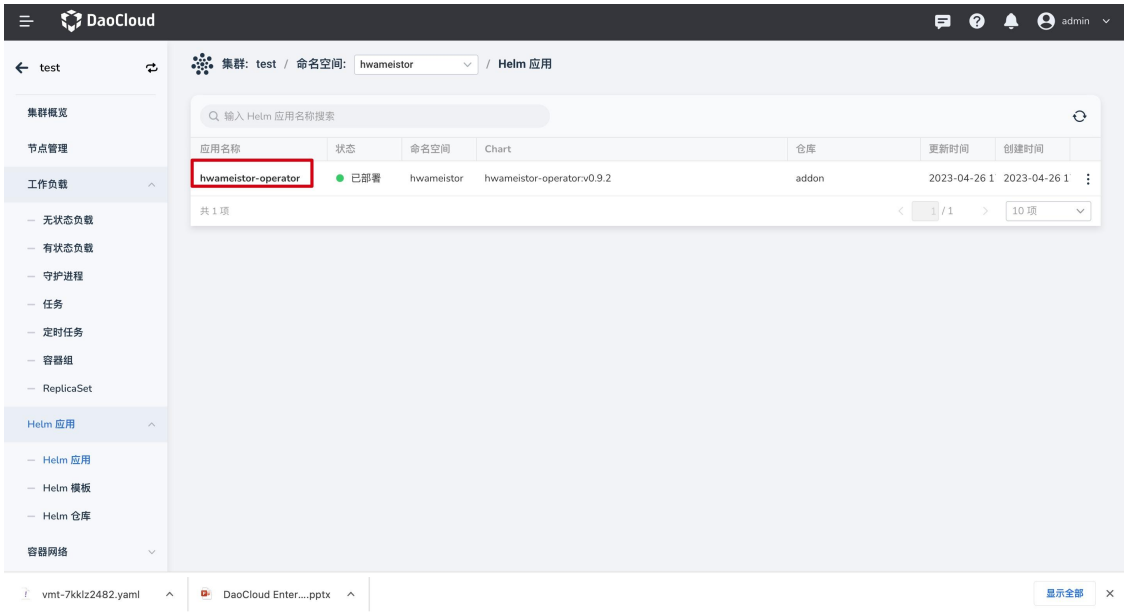
  设置 K8S 镜像仓库地址，默认已经填写可用在线仓库。 如果私有化环境，可修改为私有仓库地址。

- DRDB：

  如果需要使用 高可用数据卷，请开启 DRDB 模块，如安装时未开启，请参考 开启 DRDB

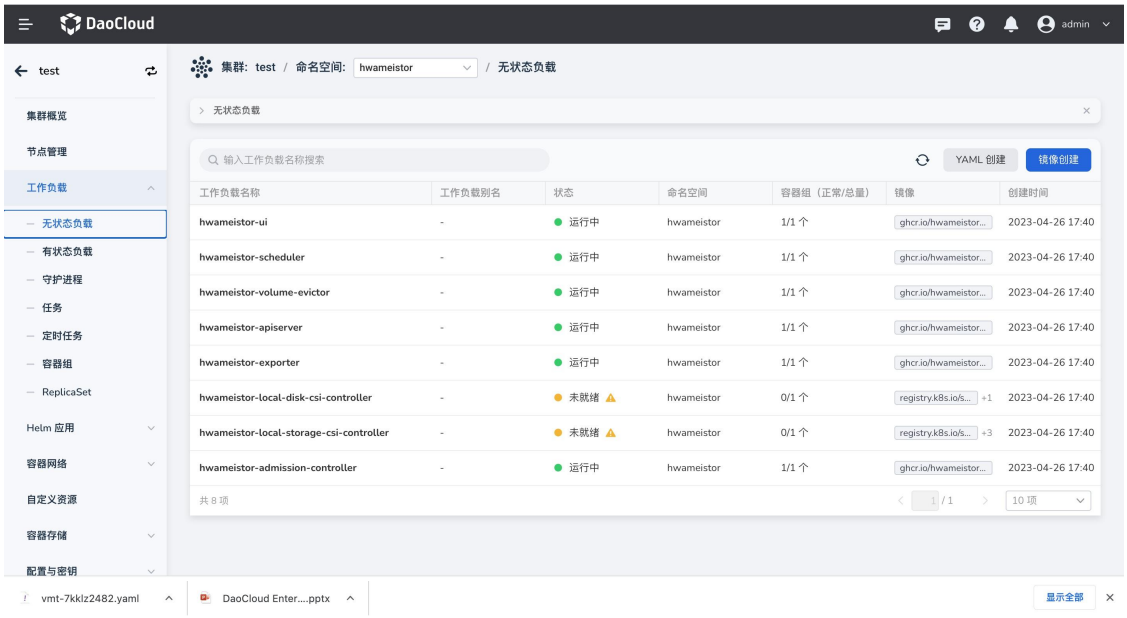
- replicas：各组件副本数量，建议使用 2 副本。

5. 确认参数无误后，点击确定完成安装，完成安装后可点击 Helm 应用查看 Hwameistor Operator 安装状态。



Operator03

6. Operator 安装完成后，Hwameistor 组件（Local Storage、Local Disk Manager 等）默认进行安装！ 可点击工作负载->无状态工作负载，选择对应命名空间，查看 Hwameistor 组件状态。



hwameistor 状态

通过命令行验证安装效果，请参安装后检查。

安装后检查

下文的示例来自一个 4 节点的 Kubernetes 集群。

```
$ kubectl get no
NAME                STATUS    ROLES    AGE    VERSION
k8s-master-1    Ready     master   82d    v1.24.3-2+63243a96d1c393
k8s-worker-1    Ready     worker   36d    v1.24.3-2+63243a96d1c393
k8s-worker-2    Ready     worker   59d    v1.24.3-2+63243a96d1c393
k8s-worker-3    Ready     worker   36d    v1.24.3-2+63243a96d1c393
```

检查 Hwameistor 系统组件

运行以下命令检查 hwameistor 相关 Pod。

```
$ kubectl -n hwameistor get pod
NAME                                                          READY    STATUS
                    RESTARTS    AGE
drbd-adapter-10-6-234-40-rhel7-shfgt                          0/2      Complete
d    0            39s
drbd-adapter-10-6-234-41-rhel7-sw75z                          0/2      Complete
d    0            39s
drbd-adapter-10-6-234-42-rhel7-4vnl9                          0/2      Complete
d    0            39s
hwameistor-admission-controller-6995559b48-rxs4s              1/1      Running
    0            40s
hwameistor-apiserver-677ddbdb8-9969c                          1/1      Running
    0            40s
hwameistor-exporter-66784d5745-xwsxx                          1/1      Running
    0            39s
hwameistor-local-disk-csi-controller-5d74d6c8cf-8vx5v         2/2      Running
   0            40s
```

```
hwameistor-local-disk-manager-9vc75                    2/2     Running
   0          40s
hwameistor-local-disk-manager-r42sg                    2/2     Running
   0          40s
hwameistor-local-disk-manager-v75qb                    2/2     Running
   0          40s
hwameistor-local-storage-csi-controller-758c94489-7g5tl   4/4     Running
  0          40s
hwameistor-local-storage-pr265                         2/2     Running
  0          40s
hwameistor-local-storage-qvrgb                         2/2     Running
  0          40s
hwameistor-local-storage-zvggz                         2/2     Running
  0          40s
hwameistor-scheduler-7585d88d9-8tbms                     1/1     Running
    0          40s
hwameistor-ui-9885d9dc5-l4tlx                          1/1     Running
  0          40s
hwameistor-volume-evictor-56df755847-m4h8b               1/1     Running
    0          40s
```

!!! info

`local-disk-manager` 和 `local-storage` 是 `DaemonSet`。在每个 Kubernetes 节点上都应该有一个 DaemonSet Pod。

查看 HwameiStor CRD（即 API）

以下 HwameiStor CRD 必须安装在系统上。

```
$ kubectl api-resources --api-group hwameistor.io
NAME                 SHORTNAMES   APIVERSION                NAMESPACED   KIND
localdiskclaims      ldc          hwameistor.io/v1alpha1    false        Lo
```

calDiskClaim
localdisknodes ldn hwameistor.io/v1alpha1 false L
ocalDiskNode
localdisks ld hwameistor.io/v1alpha1 false Lo
calDisk
localdiskvolumes ldv hwameistor.io/v1alpha1 false Lo
calDiskVolume
localstoragenodes lsn hwameistor.io/v1alpha1 false L
ocalStorageNode
localvolumeconverts lvconvert hwameistor.io/v1alpha1 true L
ocalVolumeConvert
localvolumeexpands lvexpand hwameistor.io/v1alpha1 false
LocalVolumeExpand
localvolumegroupconverts lvgconvert hwameistor.io/v1alpha1 true L
ocalVolumeGroupConvert
localvolumegroupmigrates lvgmigrate hwameistor.io/v1alpha1 true L
ocalVolumeGroupMigrate
localvolumegroups lvg hwameistor.io/v1alpha1 true L
ocalVolumeGroup
localvolumemigrates lvmigrate hwameistor.io/v1alpha1 true L
ocalVolumeMigrate
localvolumereplicas lvr hwameistor.io/v1alpha1 false Lo
calVolumeReplica
localvolumes lv hwameistor.io/v1alpha1 false L
ocalVolume

查看 LocalDiskNode 和 localDisks

```
$ kubectl get localdisknodes
NAME            TOTALDISK   FREEDISK
10-6-234-40     1
10-6-234-41     8
10-6-234-42     8
```

```
$ kubectl get localdisks
NAME                 NODEMATCH       PHASE
10-6-234-40-sda      10-6-234-40     Bound
10-6-234-41-sda      10-6-234-41     Bound
10-6-234-41-sdb      10-6-234-41     Bound
10-6-234-41-sdc      10-6-234-41     Bound
10-6-234-41-sdd      10-6-234-41     Bound
10-6-234-41-sde      10-6-234-41     Bound
10-6-234-41-sdf      10-6-234-41     Bound
10-6-234-41-sdg      10-6-234-41     Bound
10-6-234-41-sdh      10-6-234-41     Bound
10-6-234-42-sda      10-6-234-42     Bound
10-6-234-42-sdb      10-6-234-42     Bound
10-6-234-42-sdc      10-6-234-42     Bound
10-6-234-42-sdd      10-6-234-42     Bound
10-6-234-42-sde      10-6-234-42     Bound
10-6-234-42-sdf      10-6-234-42     Bound
10-6-234-42-sdg      10-6-234-42     Bound
10-6-234-42-sdh      10-6-234-42     Bound
```

查看 LocalStorageNodes 及存储池

HwameiStor 为每个存储节点创建一个 CRD 资源 LocalStorageNode (LSN)。 每个 LSN 将会记录该存储节点的状态，及节点上的所有存储资源，包括存储池、数据卷、及相关配置信息。

```
$ kubectl get lsn
NAME            IP              STATUS     AGE
10-6-234-40     10.6.234.40     Ready      3m52s
10-6-234-41     10.6.234.41     Ready      3m54s
10-6-234-42     10.6.234.42     Ready      3m55s
```

```
$ kubectl get lsn 10-6-234-41 -o yaml
apiVersion: hwameistor.io/v1alpha1
kind: LocalStorageNode
metadata:
  creationTimestamp: "2023-04-11T06:46:52Z"
  generation: 1
  name: 10-6-234-41
  resourceVersion: "13575433"
  uid: 4986f7b8-6fe1-43f1-bdca-e68b6fa53f92
spec:
  hostname: 10-6-234-41
  storageIP: 10.6.234.41
  topogoly:
    region: default
    zone: default
status:
  pools:
    LocalStorage_PoolHDD:
      class: HDD
      disks:
      - capacityBytes: 10733223936
        devPath: /dev/sdb
        state: InUse
        type: HDD
      - capacityBytes: 1069547520
        devPath: /dev/sdc
        state: InUse
        type: HDD
      - capacityBytes: 1069547520
        devPath: /dev/sdd
        state: InUse
        type: HDD
```

```
    - capacityBytes: 1069547520
      devPath: /dev/sde
      state: InUse
      type: HDD
    - capacityBytes: 1069547520
      devPath: /dev/sdf
      state: InUse
      type: HDD
    - capacityBytes: 1069547520
      devPath: /dev/sdg
      state: InUse
      type: HDD
    freeCapacityBytes: 16080961536
    freeVolumeCount: 1000
    name: LocalStorage_PoolHDD
    totalCapacityBytes: 16080961536
    totalVolumeCount: 1000
    type: REGULAR
    usedCapacityBytes: 0
    usedVolumeCount: 0
    volumeCapacityBytesLimit: 16080961536
  state: Ready
```

查看 StorageClass

HwameiStor Operator 在完成 HwameiStor 系统组件安装和系统初始化之后，会根据系统配置（例如：是否开启 HA 模块、磁盘类型等）自动创建相应的 StorageClass 用于创建数据卷。

```
$ kubectl get sc
NAME                                PROVISIONER            RECLAIMPOLI
CY     VOLUMEBINDINGMODE         ALLOWVOLUMEEXPANSION     AGE
hwameistor-storage-lvm-hdd          lvm.hwameistor.io      Delete
```

```
WaitForFirstConsumer      false                    23h
hwameistor-storage-lvm-hdd-convertible      lvm.hwameistor.io      Delete
WaitForFirstConsumer      false                    23h
hwameistor-storage-lvm-hdd-ha               lvm.hwameistor.io      Delete
 WaitForFirstConsumer      false                    23h
```

（可选）检查 DRBD 的安装

在每个 Worker 节点上，DRBD 的内核模块必须被加载，例如在节点 k8s-worker-1 上：

```
[root@k8s-worker-1 ~]$ lsmod | grep ^drbd
drbd_transport_tcp      22227  0
drbd                   606840  1 drbd_transport_tcp


$ cat /proc/drbd
version: 9.0.32-1 (api:2/proto:86-121)
GIT-hash: 7d2933d5a3764fcc5e0bf54b71fd9cfb0363be1a build by @4904565a90
1d, 2022-09-07 08:53:17
Transports (api:17): tcp (9.0.32-1)
```

安装 DRBD（可选）

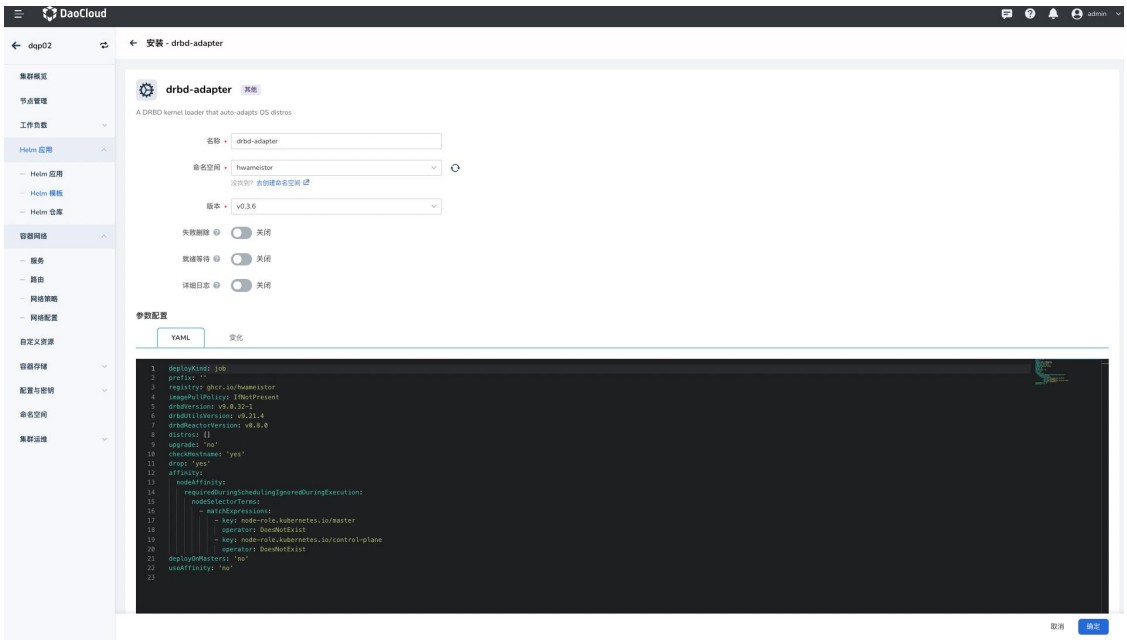
如需要使用 高可用数据卷，请在部署 Hwameistor 时开启 DRDB，如下提供安装方式：

Hwameistor 安装时同步安装

Hwameistor 安装时，可直接启用 DRDB 组件，详情可查看 Hwameistor 安装

通过 UI 界面

1. 请 进入 容器管理一>Helm 应用，选择 drbd-adapter。
2. 点击 drdb-adapter，点击安装，进入配置页面。



drbd02

- 命名空间：建议同 Hwameistor 部署在相同的命名空间中，此示例创建的命名空间为 Hwameistor。
- 版本：默认选择最新版本。

- 失败删除：默认关闭。开启后，将默认同步开启安装等待，将在安装失败时删除安装。
- 就绪等待：默认关闭。 启用后，将等待应用下所有关联资源处于就绪状态才标记应用安装成功。
- 详细日志：默认关闭。开启安装过程日志的详细输出。

3. 点击确定完成创建。

通过 Helm 安装

如通过 Helm 安装，请使用如下方式进行安装:

```
helm repo add hwameistor https://hwameistor.io/hwameistor

helm repo update hwameistor

helm pull hwameistor/drbd-adapter --untar

helm install drbd-adapter ./drbd-adapter -n hwameistor --create-namespace
```

国内用户可以使用镜像仓库 daocloud.io/daocloud 加速：

```
helm install drbd-adapter ./drbd-adapter \
    -n hwameistor --create-namespace \
    --set imagePullPolicy=Always \
    --set registry=daocloud.io/daocloud
```

通过界面安装

本文介绍如何通过平台界面安装 Hwameistor。

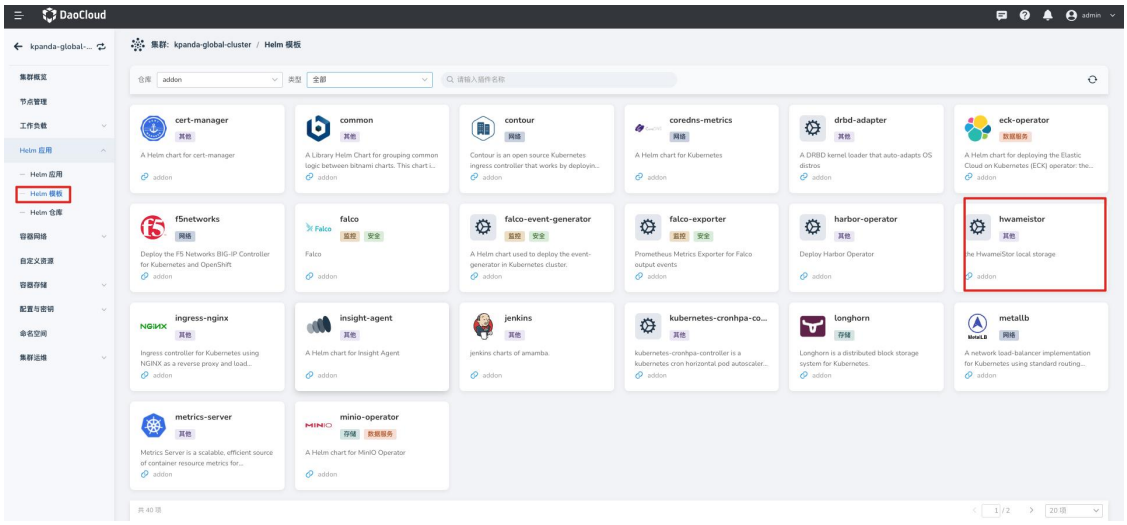
前提条件

- 待使用节点上已准备空闲 HDD、SSD 磁盘
- 已完成准备工作中事项
- 如需要使用高可用数据卷，请提前完成 DRDB 安装
- 如部署环境为生产环境，请提前阅读生产环境资源要求
- 如果您的 Kubernetes 发行版使用不同的 kubelet 目录，请提前确认 kubeletRootDir。 详细信息请参考自定义 Kubelet 根目录。

安装步骤

请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装 Hwameistor。

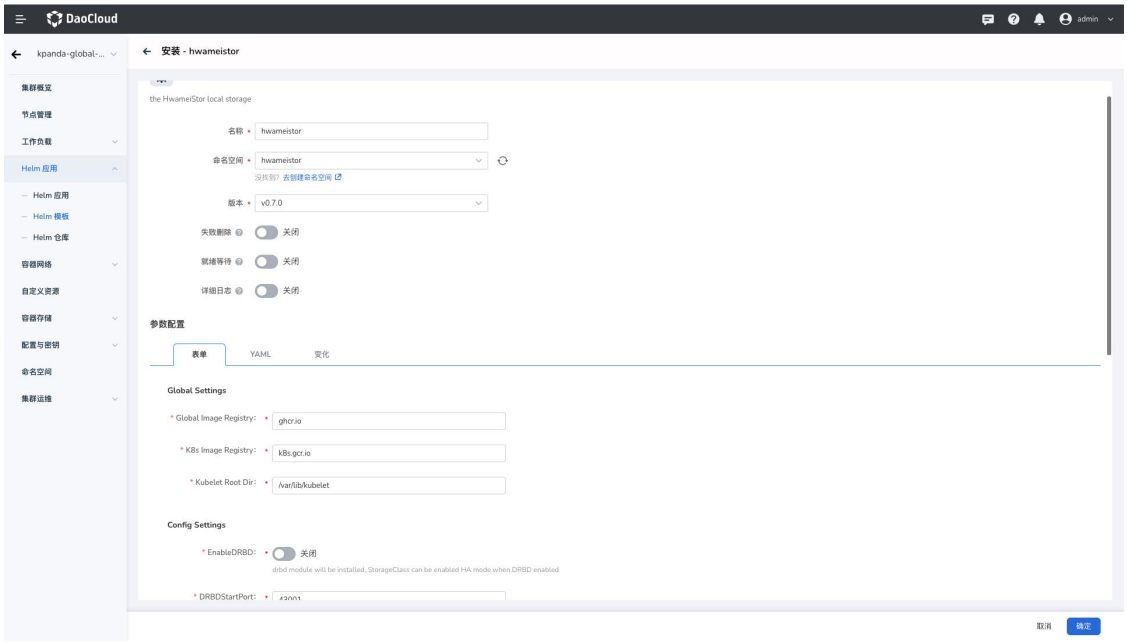
1. 在左侧导航栏点击 容器管理 —> 集群列表，然后找到准备安装 Hwameistor 的集群名称。
2. 在左侧导航栏中选择 Helm 应用 –> Helm 模板，找到并点击 Hwameistor。



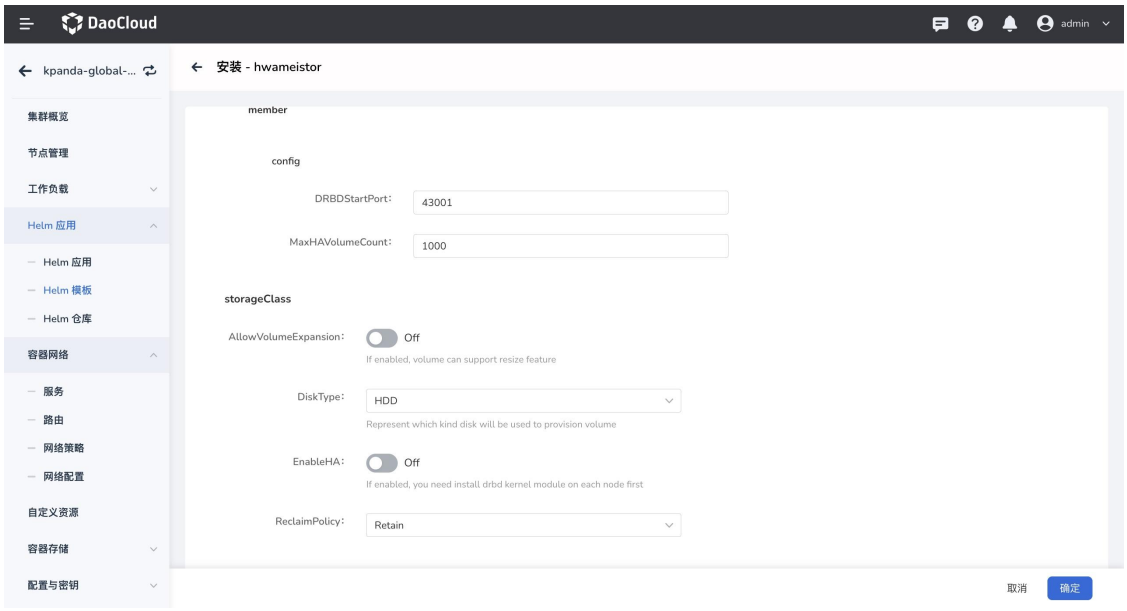
UI Install01

3. 在版本选择中选择希望安装的版本，点击安装。

4. 在安装界面，填写所需的安装参数。如需要部署生产环境，建议调整资源配置：生产环境资源要求。



HwameistorUI02



HwameistorUI03

- Global Setting —> global image Registry：

  设置所有镜像的仓库地址，默认已经填写了可用的在线仓库。 如果是私有化环境，可修改为私有仓库地址。

- Global Setting —> K8s image Registry：

  设置 K8S 镜像仓库地址，默认已经填写可用在线仓库。 如果私有化环境，可修改为私有仓库地址。

- Global Setting —> Kubelet Root Dir：

  默认的 kubelet 目录为 /var/lib/kubelet。 如果您的 Kubernetes 发行版使用不同的 kubelet 目录，必须设置参数 kubeletRootDir。详细信息请参考自定义 Kubelet 根目录。

- Config Settings —> DRDBStartPort：

  默认以 43001 开始，当开启 DRDB 时，每创建一个高可用数据卷，需要占用主副本数据卷所在节点的一组端口。 请在安装之前完成 DRDB 安装。

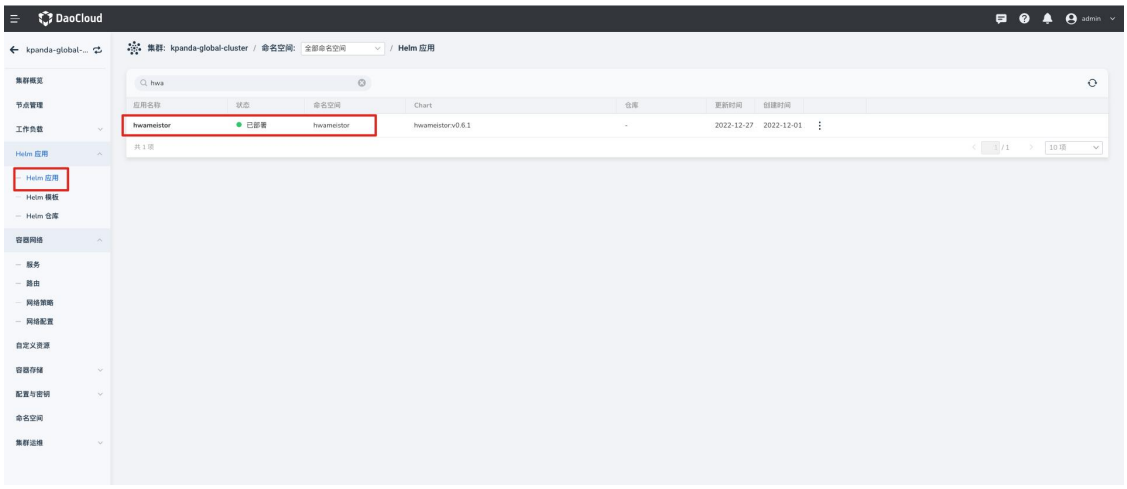
- **Storage Class 配置**

  Hwameistor 部署后，会自动创建 Storage Class。

  - AllowVolumeExpansion：默认为关闭状态，开启后，基于 Storage Class 创建的数据卷可以扩容。
  - DiskType：创建的存储池（Storage Class）的磁盘类型，支持类型有：HDD、SSD。默认为 HDD。
  - Enable HA：默认关闭 HA，即创建的数据卷为非高可用，当开启后，使用该 Storage Class 创建的数据卷可以设置为高可用数据卷。请在开启前完成 DRDB 安装。
  - Replicas：非 HA 模式下，Replicas 数量为 1；当开启 HA 模式后，Replicas 数量可以为 1 或者 2，并且数量为 1 时，可以转换为 2。
  - ReclaimPolicy: 数据卷删除时，数据的保留策略，默认为 delete。

    1. Delete：当删除数据卷时，数据一并删除。
    2. Retain：当删除数据卷时，保留数据。

5. 参数输入完成后，点击确定完成创建，完成创建后可点击 Helm 应用查看 Hwameistor 安装状态。



HwameistorUI04

6. 安装完成！要验证安装效果，请参见下一章安装后检查。

通过 Helm Chart 安装

HwameiStor 的任何组件都可以通过 Helm Chart 进行安装。

前提条件

- 待使用节点上已准备空闲 HDD、SSD 磁盘
- 已完成准备工作中事项
- 如需要使用高可用数据卷，请提前完成 DRDB 安装
- 如部署环境为生产环境，请提前阅读生产环境资源要求
- 如果您的 Kubernetes 发行版使用不同的 kubelet 目录，请提前确认 kubeletRootDir。 详细信息请参考自定义 Kubelet 根目录。

操作步骤

1. 准备 Helm 工具，安装 Helm 命令行工具，请参阅 Helm 文档。

2. 下载 hwameistor Repo，下载并解压 Repo 文件到本地：

   ```
   helm repo add hwameistor http://hwameistor.io/hwameistor
   helm repo update hwameistor
   helm pull hwameistor/hwameistor --untar
   ```

3. 安装 HwameiStor，命名如下：

   ```
   helm install hwameistor ./hwameistor \
       -n hwameistor --create-namespace
   ```

!!! tip

默认的镜像仓库是 `registry.k8s.io` 和 `ghcr.io`。

如果无法访问，可尝试使用 DaoCloud 提供的镜像源：`m.daocloud.io` 和 `ghcr.m.daocloud.io`。

要切换镜像仓库的镜像，请使用 --set 更改这两个参数值：

global.k8sImageRegistry 和 global.hwameistorImageRegistry。

```
helm install hwameistor ./hwameistor \
    -n hwameistor --create-namespace \
    --set global.k8sImageRegistry=m.daocloud.io/registry.k8s.io \
    --set global.hwameistorImageRegistry=ghcr.m.daocloud.io
```

!!! success

安装完成！要验证安装效果，请参见下一章[安装后检查](./post-check.md)。

如需要自定义 Kubelet 根目录 请参考自定义 Kubelet 根目录。

安装完成！要验证安装效果，请参见下一章安装后检查。

准备工作

平台准备

- Kubernetes 容器平台版本为 1.18+
- 已部署 CoreDNS
- 高可用功能需要安装和当前运行的 Kernel 版本一致的 kernel-devel，可通过命令检查：

  uname -r
  3.10.0-1160.el7.x86_64
  yum list installed |grep kernel
  kernel.x86_64 3.10.0-1160.el7 @ana
  conda
  kernel-tools.x86_64 3.10.0-1160.el7 @ana
  conda
  kernel-tools-libs.x86_64 3.10.0-1160.el7 @ana
  conda

  如不一致，可通过如下方式安装：

  yum install -y kernel-devel-$(uname -r) # 安装 kernel-devel

- 已安装 LVM2，如未安装请参考如下安装方式:

=== “CentOS/RHEL、Rocky 和 Kylin”

```console
yum install -y lvm2
```

=== “Ubuntu”

```console
apt-get install -y lvm2
apt-get install -y linux-headers-$(uname -r)
```

支持的操作系统

架构	支持操作系统	推荐
AMD 64	centos 7.4+	操作系统推荐 CentOS 7.9
	Redhat 8.4+	操作系统推荐 Redhat 8.4
	Redhat 7.4+	操作系统推荐 Redhat 7.9
	Ubuntu 19+	操作系统推荐 Ubuntu 20.04
ARM 64	银河麒麟 OS V10 SP2	银河麒麟 OS V10 SP2

Secure Boot

高可用功能暂时不支持 Secure Boot，确认 Secure Boot 是 disabled 状态：

```
$ mokutil --sb-state
SecureBoot disabled


$ dmesg | grep secureboot
[    0.000000] secureboot: Secure boot disabled
```

磁盘类型

HwameiStor 支持物理硬盘（HDD)、固态硬盘（SSD）和 NVMe 闪存盘.

测试环境里，每个主机必须要有至少一块空闲的 10GiB 数据盘。

生产环境里，建议每个主机至少要有一块空闲的 200GiB 数据盘，而且建议使用固态硬盘（SSD)。

网络规划

生产环境里，开启高可用模式后，建议使用有冗余保护的万兆 TCP/IP 网络。 可通过修改网卡的方式指定网卡 IP 提前进行规划。

升级

Helm 让 HwameiStor 的升级变得非常简单。只需运行以下命令：

```
helm upgrade -n hwameistor hwameistor -f new.values.yaml
```

升级过程中将以滚动的方式重启每个 HwameiStor Pod。

!!! warning

在升级 HwameiStor 期间，这些卷将继续不间断地为 Pod 服务。

自定义 Kubelet 根目录

!!! warning

默认的 `kubelet` 目录为 `/var/lib/kubelet`。
如果您的 Kubernetes 发行版使用不同的 `kubelet` 目录，必须设置参数 `kubeletRoot Dir`。

例如，在将 /var/snap/microk8s/common/var/lib/kubelet/ 用作 kubelet 目录的 Canonical 的 MicroK8s 上。

1. 通过界面操作，HwameiStor 需要需要此修改参数界面中 Global Setting —> Kubelet Root Dir 参数，并将参数设置为 /var/snap/microk8s/common/var/lib/kubelet/。详情请参考通过界面创建

2. 如果通过 Helm 创建，请执行如下安装命令。

```
helm install hwameistor ./hwameistor \
    -n hwameistor --create-namespace \
    --set kubeletRootDir=/var/snap/microk8s/common/var/lib/kubelet/
```

本地磁盘管理器

本地磁盘管理器（Local Disk Manager, LDM）是 HwameiStor 系统的一个重要功能模块。LDM 旨在简化管理节点上的磁盘。它将磁盘抽象成一种可以被管理和监控的资源。它本身是一种 DaemonSet 对象，集群中每一个节点都会运行该服务，通过该服务检测存在的磁盘并将其转换成相应的 LocalDisk 资源。



LDM 架构图

目前 LDM 还处于 alpha 阶段。

基本概念

LocalDisk(LD): 这是 LDM 抽象的磁盘资源，一个 LD 代表了节点上的一块物理磁盘。

LocalDiskClaim (LDC): 这是系统使用磁盘的方式，通过创建 LDC 对象来向系统申请磁盘。用户可以添加一些对磁盘的描述来选择磁盘。

目前，LDC 支持以下对磁盘的描述选项：

- NodeName
- Capacity
- DiskType(e.g. HDD/SSD)

用法

如果想完整地部署 HwameiStor，请参考使用 Helm Chart 安装部署。

如果只想单独部署 LDM，可以参考下面的步骤进行安装。

安装本地磁盘管理器

1. 克隆 repo 到本机

   ```
   git clone https://github.com/hwameistor/local-disk-manager.git
   ```

2. 进入 repo 对应的目录

   ```
   cd deploy
   ```

3. 安装 CRDs 和 运行 LocalDiskManager

   安装 LocalDisk 和 LocalDiskClaim 的 CRD

   ```
   kubectl apply -f deploy/crds/
   ```

   安装权限认证的 CR 以及 LDM 的 Operators

   ```
   kubectl apply -f deploy/
   ```

4. 查看 LocalDisk 信息

   ```
   $ kubectl get localdisk
   10-6-118-11-sda     10-6-118-11     Unclaimed
   10-6-118-11-sdb     10-6-118-11     Unclaimed
   ```

   该命令用于获取集群中磁盘资源信息，获取的信息总共有四列，含义分别如下：

   - **NAME：** 代表磁盘在集群中的名称。
   - **NODEMATCH：** 表明磁盘所在的节点名称。
   - **CLAIM：** 表明这个磁盘是被哪个 Claim 所引用。
   - **PHASE：** 表明这个磁盘当前的状态。

通过 kuebctl get localdisk <name> -o yaml 查看更多关于某块磁盘的信息。

5. 申请可用磁盘

创建 LocalDiskClaim

kubectl apply -f deploy/samples/hwameistor.io_v1alpha1_localdiskclaim_cr.yaml

该命令用于创建一个磁盘使用的申请请求。 在这个 yaml 文件里面，您可以在 description 字段添加对申请磁盘的描述，比如磁盘类型、磁盘的容量等等。

查看 LocalDiskClaim 信息

kubectl get localdiskclaim <name>

查看 Claim 的 Status 字段信息。 如果存在可用的磁盘，您将会看到该字段的值为 Bound。

资源

Hwameistor 在 Kubernetes 已有的 PV 和 PVC 对象类基础上，Hwameistor 定义了更丰富的对象类，把 PV/PVC 和本地数据盘关联起来。

Kind	缩写	功能
LocalDiskNode	ldn	注册节点
LocalDisk	ld	注册节点上数据盘，自动识别空闲可用的数据盘
LocalDiskClaim	ldc	筛选并注册本地数据盘
LocalStorageNode	lsn	自动创建存储池，也就是 LVM 逻辑卷组
LocalVolume	lv	创建 LVM 逻辑卷，分配给 PersistentVolume
LocalDiskExpand	lvexpand	存储池扩容

准入控制器

准入控制器是一种 webhook，可以自动验证 HwameiStor 数据卷，协助将 schedulerName 修改为 hwameistor-scheduler。 具体信息，请参见 Kubernetes 动态准入控制。

- 识别 HwameiStor 数据卷

  准入控制器可以获取 Pod 使用的所有 PVC，并检查每个 PVC 存储制备器。 如果制备器的名称后缀是 *.hwameistor.io，表示 Pod 正在使用 HwameiStor 提供的数据卷。

- 验证资源

  准入控制器只验证 Pod 资源，并在创建资源时就进行验证。

  !!! info

  为确保 HwameiStor 的 Pod 可以顺利启动，不会校验 HwameiStor 所在的命名空间下的 Pod。

DRBD 安装器

DRBD (Distributed Replicated Block Device) 通过 Linux 内核模块和相关脚本构成，用以构建高可用性的集群。其实现方式是通过网络来镜像整个设备，可以把它看作是一种网络 RAID。

这个安装器可以直接安装 DRBD 到容器集群，目前该模块仅用于测试用途。

关于 DRBD 的商业化产品，请联系 Linbit。

调度器

调度器（scheduler）是 HwameiStor 的重要组件之一。 它自动将 Pod 调度到配有 HwameiStor 存储卷的正确节点。 使用调度器后，Pod 不必再使用 NodeAffinity 或 NodeSelector 字段来选择节点。 调度器能处理 LVM 和 Disk 存储卷。

- 安装

  调度器应在集群中以 HA 模式部署，这是生产环境中的最佳实践。

- 通过 Helm Chart 部署

  调度器必须与本地磁盘和本地磁盘管理器配合使用。建议通过 Helm Chart 进行安装。

- 通过 YAML 部署（针对开发）

  ```
  kubectl apply -f deploy/scheduler.yaml
  ```

什么是 HwameiStor

HwameiStor 是一款 Kubernetes 原生的容器附加存储（CAS）解决方案，将 HDD、SSD 和 NVMe 磁盘形成本地存储资源池进行统一管理， 使用 CSI 架构提供分布式的本地数据卷服务，为有状态的云原生应用或组件提供数据持久化能力。



System architecture

具体的功能特性如下：

1. 自动化运维管理

   可以自动发现、识别、管理、分配磁盘。根据亲和性，智能调度应用和数据。自动监测磁盘状态，并及时预警。

2. 高可用的数据

   使用跨节点副本同步数据， 实现高可用。发生问题时，会自动将应用调度到高可用数据节点上，保证应用的连续性。

3. 丰富的数据卷类型

聚合 HDD、SSD、NVMe 类型的磁盘，提供非低延时，高吞吐的数据服务。

4. 灵活动态的线性扩展

   可以根据集群规模大小进行动态的扩容，灵活满足应用的数据持久化需求。

HwameiStor 发行版本 下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

本地存储

本地存储（local Storage, LS）是 HwameiStor 的一个模块，它旨在为应用提供高性能的本地持久化 LVM 存储卷。



本地存储架构图.png

目前支持的本地持久化数据卷类型：LVM。

目前支持的本地磁盘类型：HDD、SSD、NVMe。

适用场景

HwameiStor 提供两种本地数据卷：LVM、Disk。 本地存储作为 HwameiStor 的一部分，负责提供 LVM 本地数据卷，包括高可用 LVM 数据卷、非高可用 LVM 数据卷。

非高可用的 LVM 本地数据卷，适用下列场景和应用：

- 具备高可用特性的 **数据库** ，例如 MongoDB 等
- 具备高可用特性的 **消息中间件** ，例如 Kafka、RabbitMQ 等
- 具备高可用特性的 **键值存储系统** ，例如 Redis 等
- 其他具备高可用功能的应用

高可用的 LVM 本地数据卷，适用下列场景和应用：

- **数据库** ，例如 MySQL、PostgreSQL 等
- 其他需要数据高可用特性的应用

使用 Helm Chart 安装部署

本地存储是 HwameiStor 的一部分，必须与本地磁盘管理器一起工作。建议用户通过 helm-charts 部署。

独立安装部署方式

开发者可以独立安装 local-storage，从源代码进行安装，主要用于开发、测试场景。 对于这种安装方式，您需要事先安装好本地磁盘管理器。

数据卷 IO 限速

在 HwameiStor 中，它允许用户指定 Kuberentes 集群上卷的最大 IOPS 和吞吐量。

请按照以下步骤创建具有最大 IOPS 和吞吐量的卷并创建工作负载来使用它。

使用最大 IOPS 和吞吐量参数创建新的 StorageClass

默认情况下，HwameiStor 在安装过程中不会自动创建这样的 StorageClass，因此您需要手动创建 StorageClass。

示例 StorageClass 如下：

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hwameistor-storage-lvm-hdd-sample
parameters:
  convertible: "false"
  csi.storage.k8s.io/fstype: xfs
  poolClass: HDD
  poolType: REGULAR
  provision-iops-on-creation: "100"
  provision-throughput-on-creation: 1Mi
  replicaNumber: "1"
  striped: "true"
  volumeKind: LVM
provisioner: lvm.hwameistor.io
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

与 HwameiStor 安装程序创建的常规 StorageClass 相比，添加了以下参数：

- Provision-iops-on-creation：指定创建时卷的最大 IOPS。
- Provision-throughput-on-creation：它指定创建时卷的最大吞吐量。

创建 StorageClass 后，您可以使用它来创建 PVC。

使用 StorageClass 创建 PVC

示例 PVC 如下：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-sample
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: hwameistor-storage-lvm-hdd-sample
```

创建 PVC 后，您可以创建 Deployment 来使用 PVC。

创建带有 PVC 的 Deployment

示例 Deployment 如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: pod-sample
  name: pod-sample
```

```yaml
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pod-sample
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: pod-sample
    spec:
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: pvc-sample
      containers:
      - command:
        - sleep
        - "100000"
        image: busybox
        name: busybox
        resources: {}
        volumeMounts:
        - name: data
          mountPath: /data
status: {}
```

创建 Deployment 后，您可以使用以下命令测试卷的 IOPS 和吞吐量：

```
kubectl exec -it pod-sample-5f5f8f6f6f-5q4q5 -- /bin/sh
dd if=/dev/zero of=/data/test bs=4k count=1000000 oflag=direct
```

注意：由于 cgroupv1 限制，最大 IOPS 和吞吐量的设置可能对非直接 IO 不生效。

如何更改数据卷的最大 IOPS 和吞吐量

最大 IOPS 和吞吐量在 StorageClass 的参数上指定，您不能直接更改它，因为它现在是不可变的。

与其他存储厂商不同的是，HwameiStor 是一个基于 Kubernetes 的存储解决方案，它定义了一组操作原语 基于 Kubernetes CRD。 这意味着您可以修改相关的 CRD 来更改卷的实际最大 IOPS 和吞吐量。

以下步骤显示如何更改数据卷的最大 IOPS 和吞吐量。

查找给定 PVC 对应的 LocalVolume CR

```
$ kubectl get pvc pvc-sample

NAME                STATUS   VOLUME                                     C
APACITY   ACCESS MODES   STORAGECLASS                  AGE
demo                Bound    pvc-c354a56a-5cf4-4ff6-9472-4e24c7371e10   1
0Gi       RWO            hwameistor-storage-lvm-hdd     5d23h
pvc-sample          Bound    pvc-cac82087-6f6c-493a-afcd-09480de712ed   1
0Gi       RWO            hwameistor-storage-lvm-hdd-sample   5d23h

$ kubectl get localvolume

NAME                                       POOL                  REPLICA
S   CAPACITY      USED        STATE   RESOURCE   PUBLISHED   FSTYPE
  AGE
pvc-c354a56a-5cf4-4ff6-9472-4e24c7371e10   LocalStorage_PoolHDD   1
    10737418240   33783808    Ready   -1         master      xfs         5d23
h
pvc-cac82087-6f6c-493a-afcd-09480de712ed   LocalStorage_PoolHDD   1
```

10737418240 33783808 Ready -1 master xfs 5d2 3h

根据打印输出，PVC 的 LocalVolume CR 为 pvc-cac82087-6f6c-493a-afcd-09480de712ed。

修改 LocalVolume CR

kubectl edit localvolume pvc-cac82087-6f6c-493a-afcd-09480de712ed

在编辑器中，找到 spec.volumeQoS 部分并修改 iops 和 throughput 字段。 顺便说一下，空值意味着没有限制。

最后，保存更改并退出编辑器。设置将在几秒钟后生效。

注意：将来，一旦 Kubernetes 支持它，我们将允许用户直接修改卷的最大 IOPS 和吞吐量。

如何检查数据卷的实际 IOPS 和吞吐量

HwameiStor 使用 cgroupv1 来限制数据卷的 IOPS 和吞吐量，因此您可以使用以下命令来检查数据卷的实际 IOPS 和吞吐量。

```
$ lsblk
NAME                  MAJ:MIN  RM     SIZE  RO  TYPE  MOUNTPOINT
sda                     8:0      0     160G   0  disk
 ├─sda1                 8:1      0       1G   0  part  /boot
 └─sda2                 8:2      0     159G   0  part
   ├─centos-root  253:0      0     300G   0  lvm   /
   ├─centos-swap  253:1      0     7.9G   0  lvm
   └─centos-home  253:2      0  101.1G   0  lvm   /home
sdb                     8:16     0     100G   0  disk
 ├─LocalStorage_PoolHDD-pvc--cac82087--6f6c--493a--afcd--09480de712ed
                       253:3      0      10G   0  lvm   /var/lib/kubelet/pods/3d6bc980-68
ae-4a65-a1c8-8b410b7d240f/v
 └─LocalStorage_PoolHDD-pvc--c354a56a--5cf4--4ff6--9472--4e24c7371e10
                       253:4      0      10G   0  lvm   /var/lib/kubelet/pods/521fd7b4-3be
```

```
f-415b-8720-09225f93f231/v
sdc                     8:32    0   300G  0 disk
└─sdc1                  8:33    0   300G  0 part
  └─centos-root 253:0   0   300G  0 lvm   /
sr0                     11:0    1   973M  0 rom

$ cat /sys/fs/cgroup/blkio/blkio.throttle.read_iops_device
253:3 100

$ cat /sys/fs/cgroup/blkio/blkio.throttle.write_iops_device
253:3 100

$ cat /sys/fs/cgroup/blkio/blkio.throttle.read_bps_device
253:3 1048576

$ cat /sys/fs/cgroup/blkio/blkio.throttle.write_bps_device
253:3 1048576
```

卷的扩容

HwameiStor 支持 CSI 卷扩容 。这个功能实现了通过修改 PVC 的大小在线扩容卷。

下面的例子里，我们把 PVC data-sts-mysql-local-0 从 1GiB 扩容到 2GiB。

当前 PVC/PV 大小：

kubectl get pvc data-sts-mysql-local-0

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
data-sts-mysql-local-0 Bound pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8 1Gi RWO hwameistor-storage-lvm-hdd 85m

kubectl get pv pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8 1Gi RWO Delete Bound default/data-sts-mysql-local-0 hwameistor-storage-lvm-hdd 85m

查看 StorageClass 是否使用了参数 allowVolumeExpansion: true

kubectl get pvc data-sts-mysql-local-0 -o jsonpath='{.spec.storageClassName}'

hwameistor-storage-lvm-hdd

kubectl get sc hwameistor-storage-lvm-hdd -o jsonpath='{.allowVolumeExpansion}'

true

修改 PVC 的大小

kubectl get edit pvc data-sts-mysql-local-0

```
...
spec:
  resources:
    requests:
      storage: 2Gi
...
```

观察扩容过程

增加的容量越多，扩容所需时间越长。可以在 PVC 的事件日志中观察整个扩容的过程.

kubectl describe pvc data-sts-mysql-local-0

```
Events:
  Type     Reason                    Age                From
                    Message
  ----     ------                    ----               ----
                    -------
  Warning  ExternalExpanding         34s                volume_expand
                    Ignoring the PVC: didn't find a plugin capable of expanding t
he volume; waiting for an external controller to process this PVC.
  Warning  VolumeResizeFailed        33s                external-resizer lvm.
hwameistor.io  resize volume "pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8" b
y resizer "lvm.hwameistor.io" failed: rpc error: code = Unknown desc = volume
expansion not completed yet
  Normal   Resizing                  32s (x2 over 33s)  external-resizer lvm.
hwameistor.io  External resizer is resizing volume pvc-b9fc8651-97b8-414c-8bc
f-c8d2708c4ee8
  Normal   FileSystemResizeRequired  32s                external-resizer lvm.
hwameistor.io  Require file system resize of volume on node
```

Normal FileSystemResizeSuccessful 11s kubelet MountVolume.NodeExpandVolume succeeded for volume "pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8" k8s-worker-3

观察扩容完成后的 PVC/PV

kubectl get pvc data-sts-mysql-local-0

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
data-sts-mysql-local-0 Bound pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8 2Gi RWO hwameistor-storage-lvm-hdd 96m

kubectl get pv pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE
pvc-b9fc8651-97b8-414c-8bcf-c8d2708c4ee8 2Gi RWO Delete Bound default/data-sts-mysql-local-0 hwameistor-storage-lvm-hdd 96m

卷的迁移

Migrate 迁移功能是 HwameiStor 中重要的运维管理功能，当应用绑定的数据卷所在节点副本损坏时， 卷副本可以通过迁移到其他节点，并在成功迁移到新节点后，将应用重新调度到新节点，并进行数据卷的绑定挂载。

基本概念

LocalVolumeGroup(LVG)（数据卷组）管理是 HwameiStor 中重要的一个功能。当应用 Pod 申请多个数据卷 PVC 时， 为了保证 Pod 能正确运行，这些数据卷必须具有某些相同属性，例如：数据卷的副本数量，副本所在的节点。 通过数据卷组管理功能正确地管理这些相关联的数据卷，是 HwameiStor 中非常重要的能力。

前提条件

LocalVolumeMigrate 需要部署在 Kubernetes 系统中，需要部署应用满足下列条件：

- LVM2 已安装，支持 lvm 类型的卷
- convertible 类型卷（需要在 sc 中增加配置项 convertible: true）
  - 应用 Pod 申请多个数据卷 PVC 时，对应数据卷需要使用相同配置 sc
  - 基于 LocalVolume 粒度迁移时，默认所属相同 LocalVolumeGroup 的数据卷不会一并迁移（若一并迁移，需要配置开关 MigrateAllVols：true）

界面操作步骤

1. 创建 convertible StorageClass

   通过界面安装，请参考如何创建 StorageClass

2. 创建多个 PVC

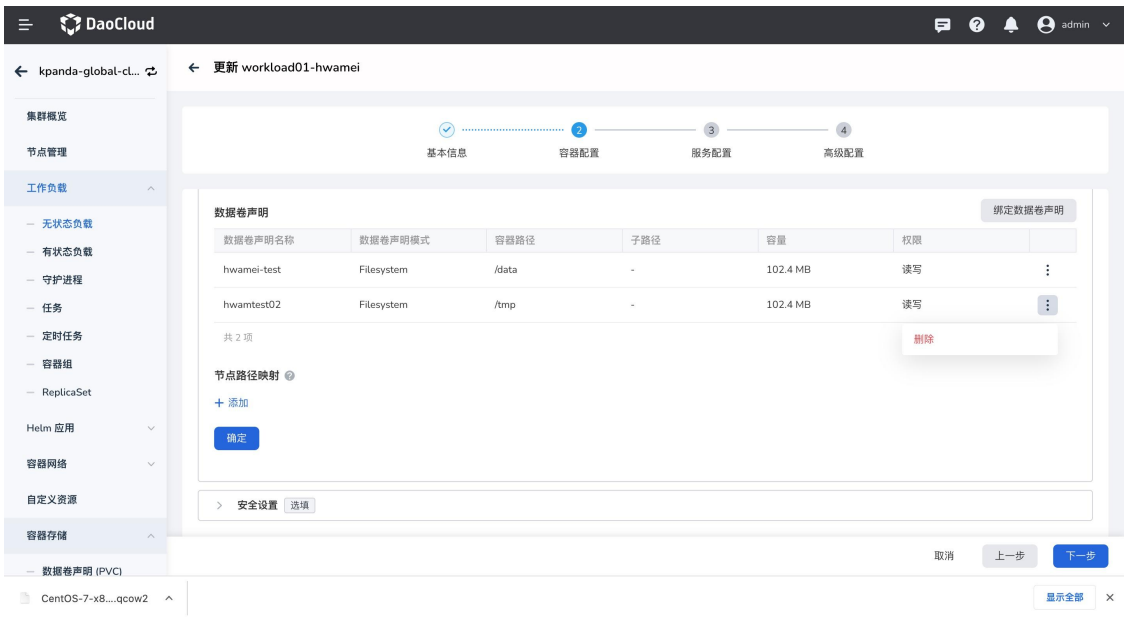
   通过界面创建多个 PVC，请参考如何创建 PVC

3. 部署多数据卷 Pod

   通过界面创建应用，请参考如何创建工作负载，并挂载已创建好 2 个 PVC

4. 解挂载多数据卷 Pod

   迁移之前请先解除 PVC 挂载，可通过编辑工作负载进行解挂载。



unbound01

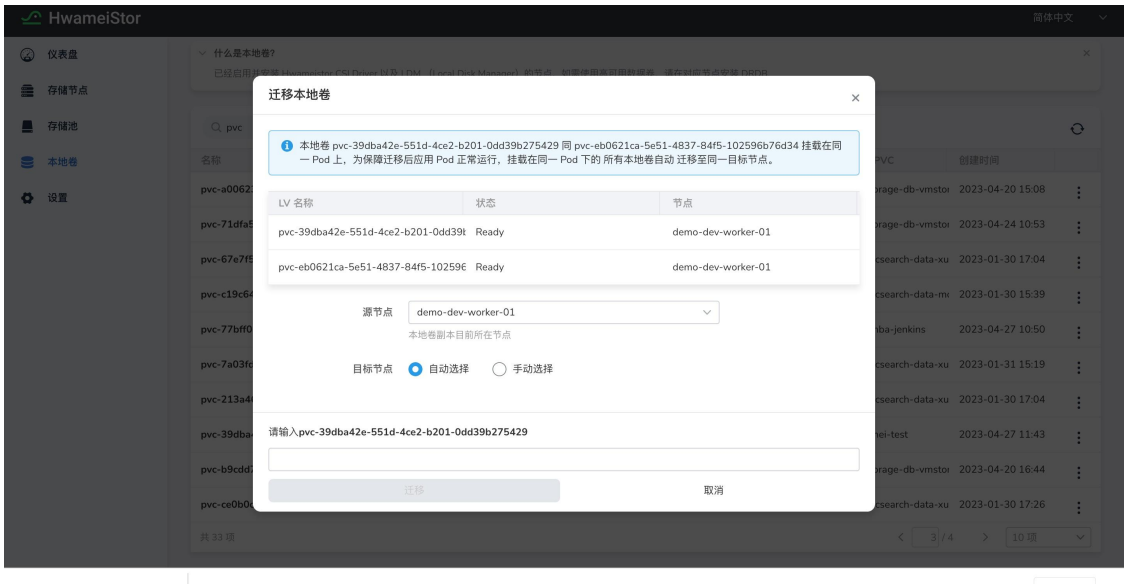
5. 创建迁移任务

   进入对应集群—> 点击左侧容器存储—>Hwameistor 进入 Hwameistor 界面，选择 已经解绑的本地卷， 对应的 PVC 为 pvc-test01、pvc-test02，点击 ... 选择迁移,选择员源节点，目标节点。

   源节点： 本地卷副本所在的节点。

目标节点： 指定后原副本将迁移至目标节点，如选择自动选择，则本地卷副本将自动调度至其他节点。

如两个/多个本地卷挂载在同一个应用上，则两个卷会自动组成 本地卷组统一进行迁移。



migration01

6. 点击对应的本地卷，进入详情查看迁移状态。

在线试用步骤

1. 创建 convertible StorageClass

   可执行如下命令，PVC 进行创建：

   cd ../../deploy/
   kubectl apply -f storageclass-convertible-lvm.yaml

2. 创建多个 PVC

   执行如下命令，PVC 进行创建：

   kubectl apply -f pvc-multiple-lvm.yaml

3. 部署多数据卷 Pod

   执行如下命令，PVC 进行创建：

   ```
   kubectl apply -f nginx-multiple-lvm.yaml
   ```

4. 解挂载多数据卷 Pod

   执行如下命令，PVC 进行创建：

   ```
   kubectl patch deployment nginx-local-storage-lvm --patch '{"spec": {"replicas": 0}}' -n hwameistor
   ```

5. 创建迁移任务

   ```
   cat > ./migrate_lv.yaml <<- EOF
   apiVersion: hwameistor.io/v1alpha1
   kind: LocalVolumeMigrate
   metadata:
     namespace: hwameistor
     name: <localVolumeMigrateName>
   spec:
     targetNodesNames:
     - <targetNodesName1>
     - <targetNodesName2>
     sourceNodesNames:
     - <sourceNodesName1>
     - <sourceNodesName2>
     volumeName: <volName>
     migrateAllVols: <true/false>
   EOF
   ```

   ```
   kubectl apply -f ./migrate_lv.yaml
   ```

6. 查看迁移状态

```
kubectl get LocalVolumeMigrate  -o yaml

apiVersion: v1
items:
- apiVersion: hwameistor.io/v1alpha1
  kind: LocalVolumeMigrate
  metadata:
  annotations:
  kubectl.kubernetes.io/last-applied-configuration: |
  {"apiVersion":"hwameistor.io/v1alpha1","kind":"LocalVolumeMigrate","metadata":{"annotations":{},"name":"localvolumemigrate-1","namespace":"hwameistor"},"spec":{"migrateAllVols":true,"sourceNodesNames":["dce-172-30-40-61"],"targetNodesNames":["172-30-45-223"],"volumeName":"pvc-1a0913ac-32b9-46fe-8258-39b4e3b696a4"}}
  creationTimestamp: "2022-07-07T12:34:31Z"
  generation: 1
  name: localvolumemigrate-1
  namespace: hwameistor
  resourceVersion: "12828637"
  uid: 78af7f1b-d701-4b03-84de-27fafca58764
  spec:
  abort: false
  migrateAllVols: true
  sourceNodesNames:
  - dce-172-30-40-61
    targetNodesNames:
  - 172-30-45-223
    volumeName: pvc-1a0913ac-32b9-46fe-8258-39b4e3b696a4
    status:
    replicaNumber: 1
    state: InProgress
    kind: List
```

```
metadata:
resourceVersion: ""
selfLink: ""
```

7. 查看迁移成功状态

```
[root@172-30-45-222 deploy]# kubectl get lvr

NAME                                                    CAPACITY    NOD
E              STATE    SYNCED    DEVICE
                                                  AGE
pvc-1a0913ac-32b9-46fe-8258-39b4e3b696a4-9cdkkn    1073741824    1
72-30-45-223    Ready    true    /dev/LocalStorage_PoolHDD-HA/pvc-1
a0913ac-32b9-46fe-8258-39b4e3b696a4    77s
pvc-d9d3ae9f-64af-44de-baad-4c69b9e0744a-7ppmrx    1073741824    1
72-30-45-223    Ready    true    /dev/LocalStorage_PoolHDD-HA/pvc-
d9d3ae9f-64af-44de-baad-4c69b9e0744a    77s
```

8. 迁移成功后，重新挂载数据卷 Pod

```
kubectl patch deployment nginx-local-storage-lvm --patch '{"spec": {"rep
licas": 1}}' -n hwameistor
```

LVM 节点扩展

存储系统可以通过增加存储节点实现扩容。在 HwameiStor 里，通过下列步骤可以添加新的存储节点。

步骤

1. 准备新的存储节点

在 Kubernetes 集群中新增一个节点，或者，选择一个已有的集群节点（非 HwameiStor 节点）。该节点必须满足 Prerequisites 要求的所有条件。 本例中，所用的新增存储节点和磁盘信息如下所示：

- name: k8s-worker-4
- devPath: /dev/sdb
- diskType: SSD disk

新增节点已经成功加入 Kubernetes 集群之后，检查并确保下列 Pod 正常运行在该节点上，以及相关资源存在于集群中：

```
kubectl get node

NAME           STATUS   ROLES    AGE   VERSION
k8s-master-1   Ready    master   96d   v1.24.3-2+63243a96d1c393
k8s-worker-1   Ready    worker   96h   v1.24.3-2+63243a96d1c393
k8s-worker-2   Ready    worker   96h   v1.24.3-2+63243a96d1c393
k8s-worker-3   Ready    worker   96d   v1.24.3-2+63243a96d1c393
k8s-worker-4   Ready    worker   1h    v1.24.3-2+63243a96d1c393

kubectl -n hwameistor get pod -o wide | grep k8s-worker-4

hwameistor-local-disk-manager-c86g5   2/2   Running   0   19h   10.6.
182.105   k8s-worker-4   <none>   <none>
```

```
hwameistor-local-storage-s4zbw                2/2     Running   0          19h   192.16
8.140.82       k8s-worker-4   <none>   <none>
```

检查 LocalStorageNode 资源

```
kubectl get localstoragenode k8s-worker-4
```

```
NAME                IP              ZONE      REGION    STATUS   AGE
k8s-worker-4   10.6.182.103   default   default   Ready    8d
```

2. 添加新增存储节点到 HwameiStor 系统

为增加存储节点创建资源 LocalStorageClaim，以此为新增存储节点构建存储池。这样，节点就已经成功加入 HwameiStor 系统。具体如下：

```
$ kubectl apply -f - <<EOF
apiVersion: hwameistor.io/v1alpha1
kind: LocalDiskClaim
metadata:
  name: k8s-worker-4
spec:
  nodeName: k8s-worker-4
  description:
    diskType: SSD
EOF
```

3. 后续检查

完成上述步骤后，检查新增存储节点及其存储池的状态，确保节点和 HwameiStor 系统的正常运行。具体如下：

```
kubectl get localstoragenode k8s-worker-4
```

```
apiVersion: hwameistor.io/v1alpha1
kind: LocalStorageNode
metadata:
  name: k8s-worker-4
```

```
spec:
  hostname: k8s-worker-4
  storageIP: 10.6.182.103
  topogoly:
    region: default
    zone: default
status:
  pools:
    LocalStorage_PoolSSD:
      class: SSD
      disks:
        - capacityBytes: 214744170496
          devPath: /dev/sdb
          state: InUse
          type: SSD
      freeCapacityBytes: 214744170496
      freeVolumeCount: 1000
      name: LocalStorage_PoolSSD
      totalCapacityBytes: 214744170496
      totalVolumeCount: 1000
      type: REGULAR
      usedCapacityBytes: 0
      usedVolumeCount: 0
      volumeCapacityBytesLimit: 214744170496
      volumes:
  state: Ready
```

裸磁盘节点扩容

裸磁盘存储节点为应用提供裸磁盘类型的数据卷，并且维护了该存储节点上面的裸磁盘和裸磁盘数据卷的对应关系。

步骤

1. 准备新的存储节点

在 Kubernetes 集群中新增一个节点，或者，选择一个已有的集群节点（非 HwameiStor 节点）。该节点必须满足 Prerequisites 要求的所有条件。 本例中，所用的新增存储节点和磁盘信息如下所示：

- name: k8s-worker-2
- devPath: /dev/sdb
- diskType: SSD disk

新增节点已经成功加入 Kubernetes 集群之后，检查并确保下列 Pod 正常运行在该节点上，以及相关资源存在于集群中：

```
$ kubectl get node
NAME           STATUS   ROLES                  AGE     VERSION
k8s-master-1   Ready    master                 96d     v1.24.3-2+63243a96d1c393
k8s-worker-1   Ready    worker                 96h     v1.24.3-2+63243a96d1c393
k8s-worker-2   Ready    worker                 96h     v1.24.3-2+63243a96d1c393

$ kubectl -n hwameistor get pod -o wide | grep k8s-worker-2
hwameistor-local-disk-manager-sfsf1       2/2     Running   0          19h   10.6.12
8.150          k8s-worker-2   <none>   <none>

# 检查 LocalDiskNode 资源
$ kubectl get localdisknode k8s-worker-2
```

```
NAME          FREECAPACITY   TOTALCAPACITY   TOTALDISK   STATUS
 AGE
k8s-worker-2                                             Ready
 21d
```

2. 添加新增存储节点到 HwameiStor 系统

首先，需要将磁盘 sdb 的 owner 信息修改成 local-disk-manager，具体如下：

```
$ kubectl edit ld k8s-worker-2-sdb
apiVersion: hwameistor.io/v1alpha1
kind: LocalDisk
metadata:
  name: k8s-worker-2-sdb
spec:
  devicePath: /dev/sdb
  nodeName: k8s-worker-2
+ owner: local-disk-manager
...
```

为增加存储节点创建资源 LocalStorageClaim，以此为新增存储节点构建存储池。这样，节点就已经成功加入 HwameiStor 系统。具体如下：

```
$ kubectl apply -f - <<EOF
apiVersion: hwameistor.io/v1alpha1
kind: LocalDiskClaim
metadata:
  name: k8s-worker-2
spec:
  nodeName: k8s-worker-2
  owner: local-disk-manager
  description:
    diskType: SSD
EOF
```

3. 后续检查

完成上述步骤后，检查新增存储节点及其存储池的状态，确保节点和 HwameiStor 系统的正常运行。具体如下：

```
$ kubectl get localdisknode k8s-worker-2 -o yaml
apiVersion: hwameistor.io/v1alpha1
kind: LocalDiskNode
metadata:
  name: k8s-worker-2
spec:
  nodeName: k8s-worker-2
status:
  pools:
    LocalDisk_PoolSSD:
      class: SSD
      disks:
      - capacityBytes: 214744170496
        devPath: /dev/sdb
        state: Available
        type: SSD
      freeCapacityBytes: 214744170496
      freeVolumeCount: 1
      totalCapacityBytes: 214744170496
      totalVolumeCount: 1
      type: REGULAR
      usedCapacityBytes: 0
      usedVolumeCount: 0
      volumeCapacityBytesLimit: 214744170496
      volumes:
  state: Ready
```

预留磁盘

本章节介绍如何在界面上预留磁盘，此功能的使用场景如下：

运维管理员有磁盘的规划需求，需要预留部分磁盘不被 Hwameistor 使用，需要自行进行 Reserved 操作

使用说明

1. Local Disk 的使用状态和预留状态为 2 个状态，任何状态的 LD 都可以 Reserved/Unreserved。
2. Hwameistor 在启动初期会自动检测系统盘并标记为 Reserveds
3. LD 只有三种状态：**Pending，Available，Bound**。LD 使用状态和 Reserved 情况如下：

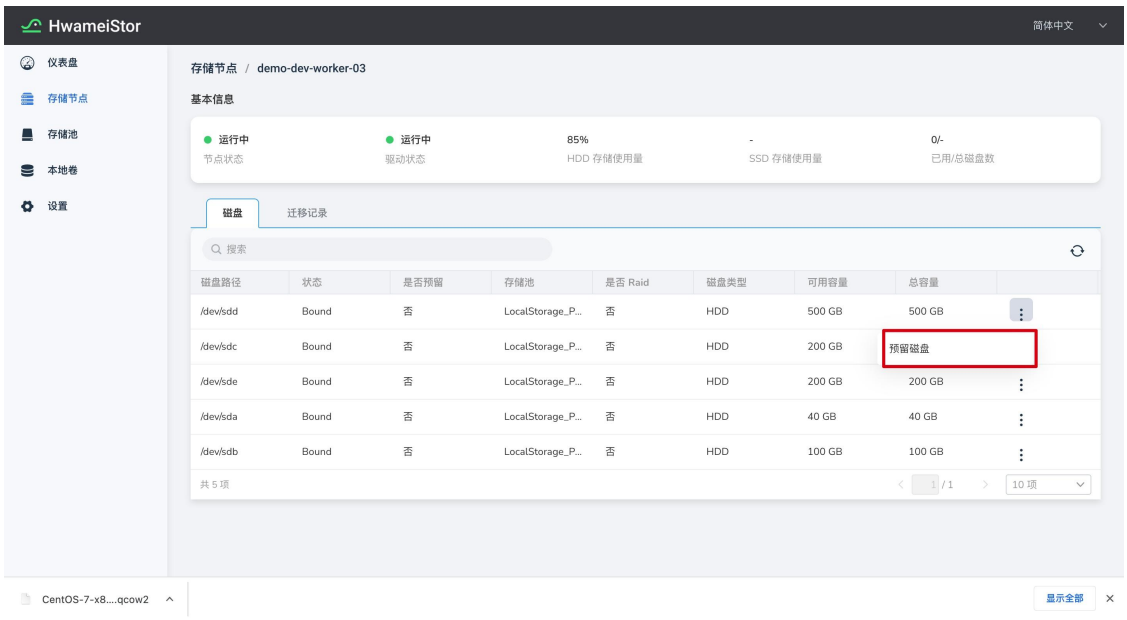
使用状态	是否 Reserved	场景说明
Pending	—	初始化状态
Pending	Reserved	初始化瞬间被用户预留
Available	—	空闲磁盘，可被 Hwameistor 分配
Available	Reserved	空闲磁盘但是规划作为他用，无法被 Hwameistor 分配无文件系统，但实际此磁盘已被使用，同时被用户标记成 Reserved
Bound	—	被 Hwameistor 使用的磁盘被系统或者外部程序使用，并手动去除了 Reserved 状态
Bound	Reserved	被系统或者外部程序使用，并手动去除 Reserved 状态被 Hwamristor 使用，同时用户手动标记为 Reserved 状态，

使用状态	是否 Reserved	场景说明
		当磁盘上的数据释放后，不在被 Hwameistor 系统使用

操作步骤

进入对应集群，选择存储一>Hwameistor；点击节点 进入节点详情页面，找到对应的磁盘；点击 ...，选择预留磁盘；点击确认进行预留。

预留后磁盘将不会被 Hwameistor 使用。



数据盘扩展

当存储系统中的某个节点存储容量不足时，可以通过为该节点增加磁盘扩充容量。在 HwameiStor 中，可以通过下列步骤完成为节点增加磁盘（数据盘）。

具体操作步骤如下：

步骤

准备新的存储磁盘

在 Kubernetes 集群中新增一个节点，或者，选择一个已有的集群节点（非 HwameiStor 节点）。 该节点必须满足 Prerequisites 要求的所有条件。 本例中，所用的新增存储节点和磁盘信息如下所示：

- name: k8s-worker-4
- devPath: /dev/sdb
- diskType: SSD disk

在新增磁盘被插入到 HwameiStor 存储节点 k8s-worker-4 后，检查该节点上的新磁盘状态，如下：

1. 检查新增磁盘是否成功插入节点，并被正确识别

```
# 1. 检查新增磁盘是否成功插入节点，并被正确识别
$ ssh root@k8s-worker-4
$ lsblk | grep sdc
sdc            8:32       0       20G   1 disk


# 2. 检查 HwameiStor 是否为新增磁盘正确创建资源 LocalDisk，并且状态为 `Unclaimed`
$ kubectl get localdisk | grep k8s-worker-4 | grep sdc
k8s-worker-4-sdc     k8s-worker-4          Unclaimed
```

2. 检查 HwameiStor 是否为新增磁盘正确创建资源 LocalDisk，并且状态为 Unclaimed

```
kubectl get localdisk | grep k8s-worker-4 | grep sdc

k8s-worker-4-sdc    k8s-worker-4          Unclaimed
```

将新增磁盘加入到节点的存储池

通过创建资源 LocalDiskClaim，将新增磁盘加入节点的存储池。 完成下列操作后，新磁盘应该被自动加入节点的 SSD 存储池中。如果该节点上没有 SSD 存储池，HwameiStor 会为其自动创建，并将新磁盘加入其中。

```
$ kubectl apply -f - <<EOF
apiVersion: hwameistor.io/v1alpha1
kind: LocalDiskClaim
metadata:
  name: k8s-worker-4-expand
spec:
  nodeName: k8s-worker-4
  description:
    diskType: SSD
EOF
```

后续检查

完成上述步骤后，检查新增磁盘及其存储池的状态，确保节点和 HwameiStor 系统的正常运行。具体如下：

```
apiVersion: hwameistor.io/v1alpha1
kind: LocalStorageNode
metadata:
  name: k8s-worker-4
spec:
  hostname: k8s-worker-4
  storageIP: 10.6.182.103
```

```
  topogoly:
    region: default
    zone: default
status:
  pools:
    LocalStorage_PoolSSD:
      class: SSD
      disks:
      - capacityBytes: 214744170496
        devPath: /dev/sdb
        state: InUse
        type: SSD
      - capacityBytes: 214744170496
        devPath: /dev/sdc
        state: InUse
        type: SSD
      freeCapacityBytes: 429488340992
      freeVolumeCount: 1000
      name: LocalStorage_PoolSSD
      totalCapacityBytes: 429488340992
      totalVolumeCount: 1000
      type: REGULAR
      usedCapacityBytes: 0
      usedVolumeCount: 0
      volumeCapacityBytesLimit: 429488340992
      volumes:
  state: Ready
```

创建存储池

下面的例子来自于一个 4 节点的 Kubernetes 集群：

```
$ kubectl get no
NAME              STATUS     ROLES     AGE     VERSION
k8s-master-1      Ready      master    82d     v1.24.3-2+63243a96d1c393
k8s-worker-1      Ready      worker    36d     v1.24.3-2+63243a96d1c393
k8s-worker-2      Ready      worker    59d     v1.24.3-2+63243a96d1c393
k8s-worker-3      Ready      worker    36d     v1.24.3-2+63243a96d1c393
```

创建 LocalDiskClaim 对象

HwameiStor 根据存储介质类型创建 LocalDiskClaim 对象来创建存储池。 要在所有 Kubernetes Worker 节点上创建一个 HDD 存储池，用户需要通过 storageNodes 参数输入各个 Worker 节点名：

```
helm template ./hwameistor \
    -s templates/post-install-claim-disks.yaml \
    --set storageNodes='{k8s-worker-1,k8s-worker-2,k8s-worker-3}' \
    | kubectl apply -f -
```

或者通过以下方法指定所有 Worker 节点：

```
sn="$( kubectl get no -l node-role.kubernetes.io/worker -o jsonpath="{.items[*].metadata.name}" | tr ' ' ',' )"

helm template ./hwameistor \
     -s templates/post-install-claim-disks.yaml \
     --set storageNodes="{$sn}" \
  | kubectl apply -f -
```

验证 LocalDiskClaim 对象

运行以下命令：

```
$ kubectl get ldc
NAME                NODEMATCH         PHASE
k8s-worker-1    k8s-worker-1    Bound
k8s-worker-2    k8s-worker-2    Bound
k8s-worker-3    k8s-worker-3    Bound
```

验证 StorageClass

运行以下命令：

```
$ kubectl get sc hwameistor-storage-lvm-hdd
NAME                                PROVISIONER              RECLAIMPOLICY    VOLU
MEBINDINGMODE          ALLOWVOLUMEEXPANSION    AGE
hwameistor-storage-lvm-hdd    lvm.hwameistor.io    Delete                WaitForFirst
Consumer    true                        114s
```

验证 LocalDisk 对象

运行以下命令：

```
kubectl get ld
```

输出类似于：

```
NAME                NODEMATCH         CLAIM              PHASE
k8s-worker-1-sda    k8s-worker-1                         Inuse
k8s-worker-1-sdb    k8s-worker-1    k8s-worker-1    Claimed
k8s-worker-1-sdc    k8s-worker-1    k8s-worker-1    Claimed
k8s-worker-2-sda    k8s-worker-2                         Inuse
k8s-worker-2-sdb    k8s-worker-2    k8s-worker-2    Claimed
k8s-worker-2-sdc    k8s-worker-2    k8s-worker-2    Claimed
k8s-worker-3-sda    k8s-worker-3                         Inuse
```

```
k8s-worker-3-sdb    k8s-worker-3    k8s-worker-3    Claimed
k8s-worker-3-sdc    k8s-worker-3    k8s-worker-3    Claimed
```

观察 VG (可选)

在一个 Kubernetes Worker 节点上，观察为 LocalDiskClaim 对象创建 VG。

运行以下命令：

```
$ vgdisplay LocalStorage_PoolHDD
  --- Volume group ---
  VG Name               LocalStorage_PoolHDD
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               199.99 GiB
  PE Size               4.00 MiB
  Total PE              51198
  Alloc PE / Size       0 / 0
  Free  PE / Size       51198 / 199.99 GiB
  VG UUID               jJ3s7g-iyoJ-c4zr-3Avc-3K4K-BrJb-A5A5Oe
```

!!! note

在安装 HwameiStor 期间也可以通过设置 `storageNode` 参数配置存储池：

```console
helm install hwameistor ./hwameistor \
    -n hwameistor --create-namespace \
    --set storageNodes='{k8s-worker-1,k8s-worker-2,k8s-worker-3}'
```

概览

HwameiStor 提供两种本地数据卷：LVM、Disk。

本地磁盘作为 HwameiStor 的一部分，负责提供 LVM 本地数据卷，其中包括高可用 LVM 数据卷、非高可用 LVM 数据卷。

非高可用的 LVM 本地数据卷，适用下列场景和应用：

- 具备高可用特性的 **数据库**，例如 MongoDB
- 具备高可用特性的 **消息中间件**，例如 Kafka、RabbitMQ
- 具备高可用特性的 **键值存储系统**，例如 Redis
- 其他具备高可用功能的应用

高可用的 LVM 本地数据卷，适用下列场景和应用：

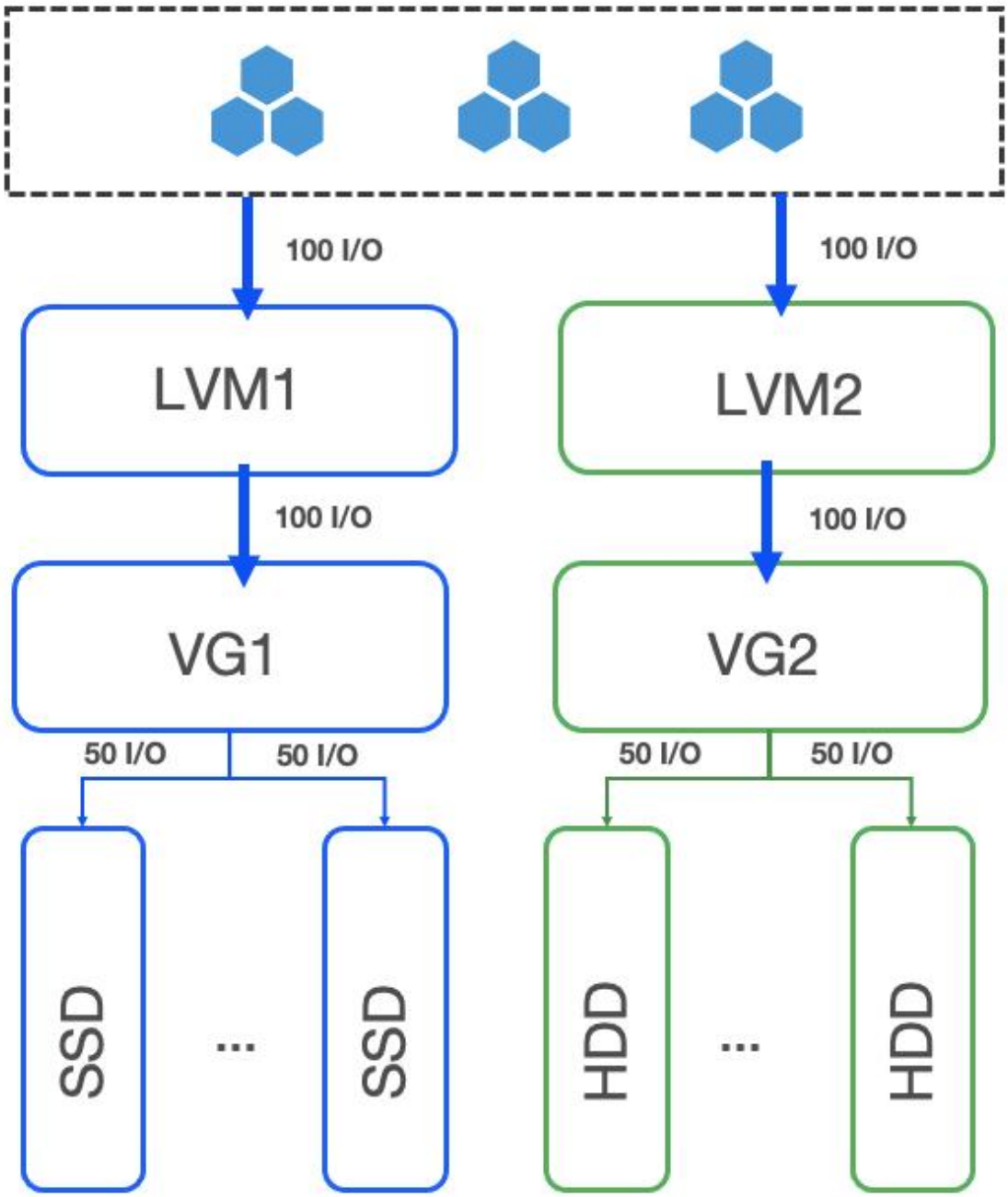
- **数据库**，例如 MySQL、PostgreSQL
- 其他需要数据高可用特性的应用

热备份机制

单节点热备份

Raid 5 保障，可容忍 1 组磁盘故障。

控制流和数据流彼此独立，保证数据访问的稳定性。

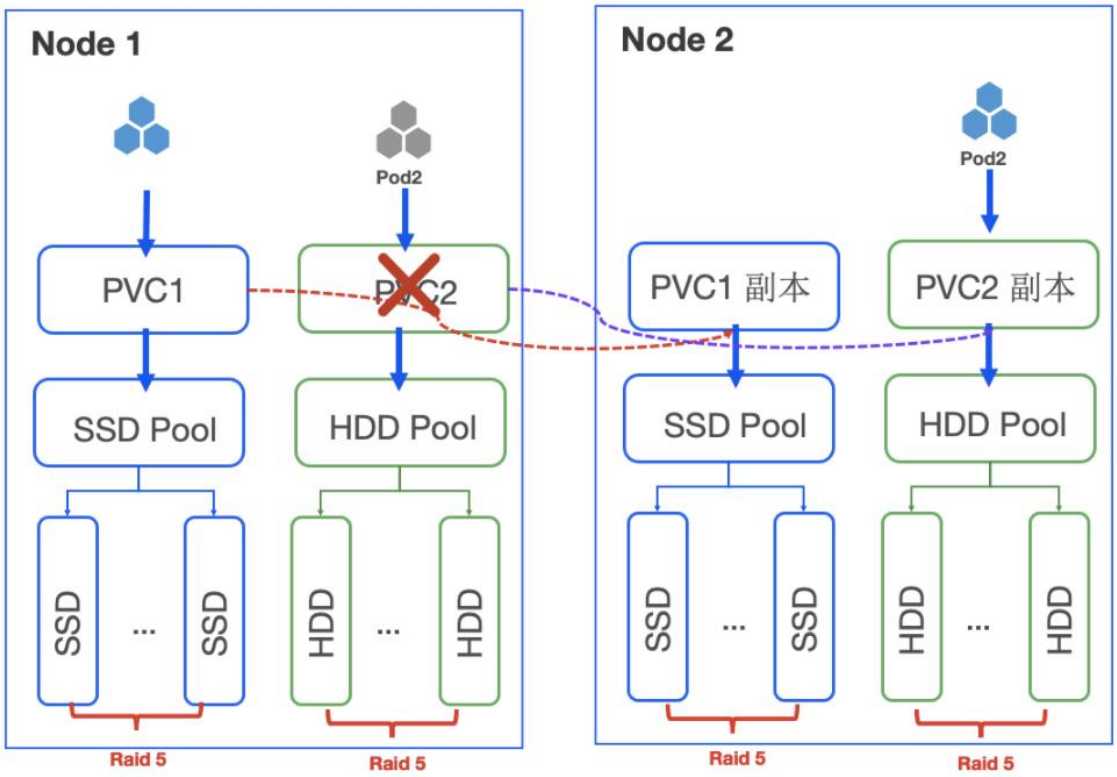


单节点热备份

跨节点热备份

Raid 5 + 主备副本保障。

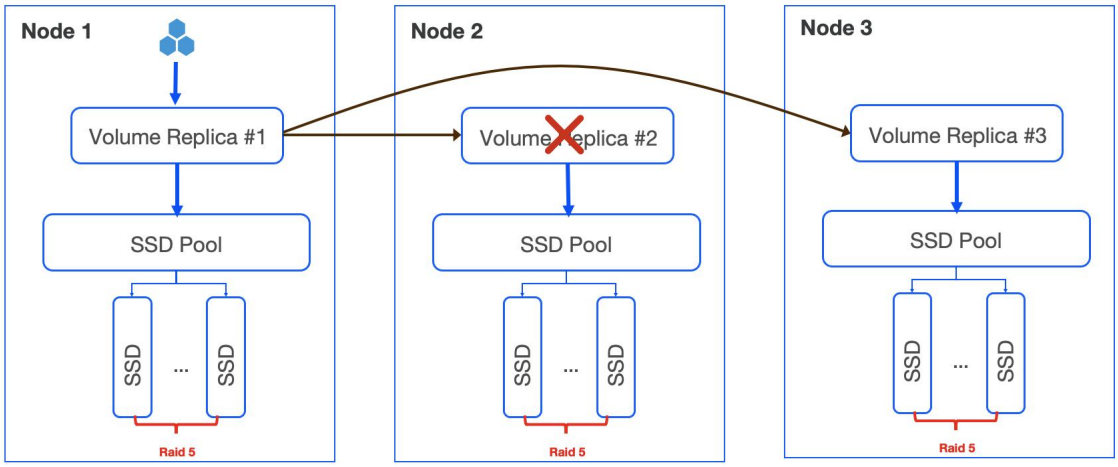
规划了 HA 专用网络逻辑接口 dce-storage 在节点间同步存储流量。跨节点同步复制数据，保证数据的热备份。



跨节点热备份

数据再平衡

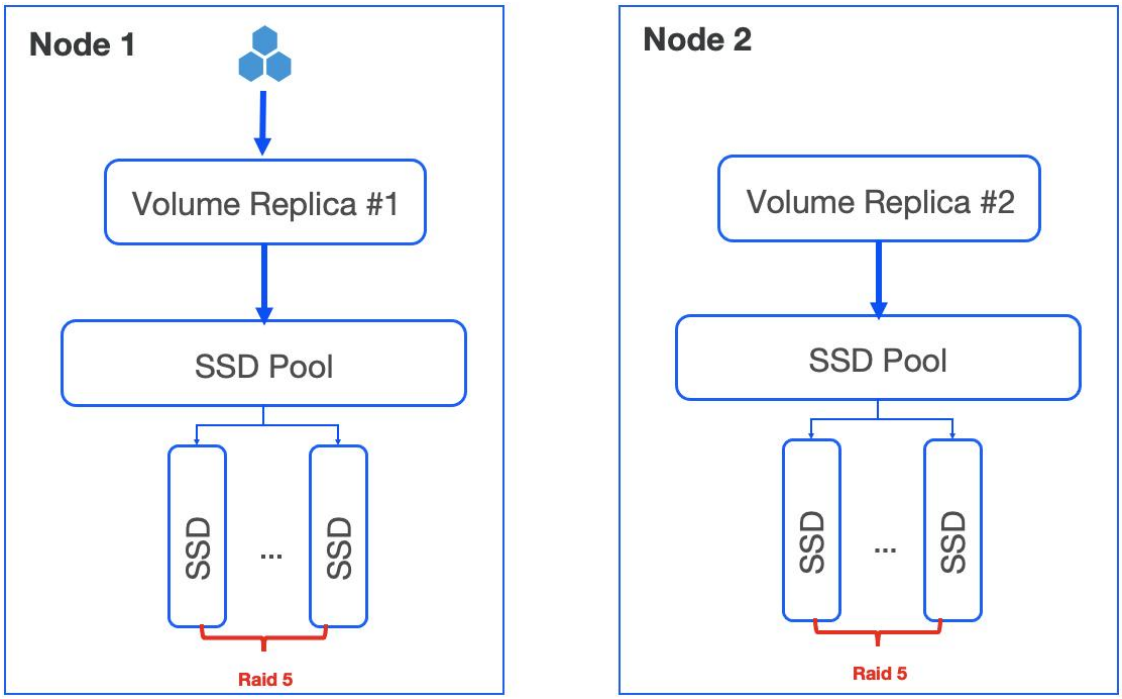
通过数据卷迁移技术实现数据在集群中的均衡放置。在线移动数据到有更多富裕空间的节点上。



数据再平衡

数据卷类型变更

为了支持某些特殊场景，允许单副本数据卷变更为多副本，支持跨节点热备份。



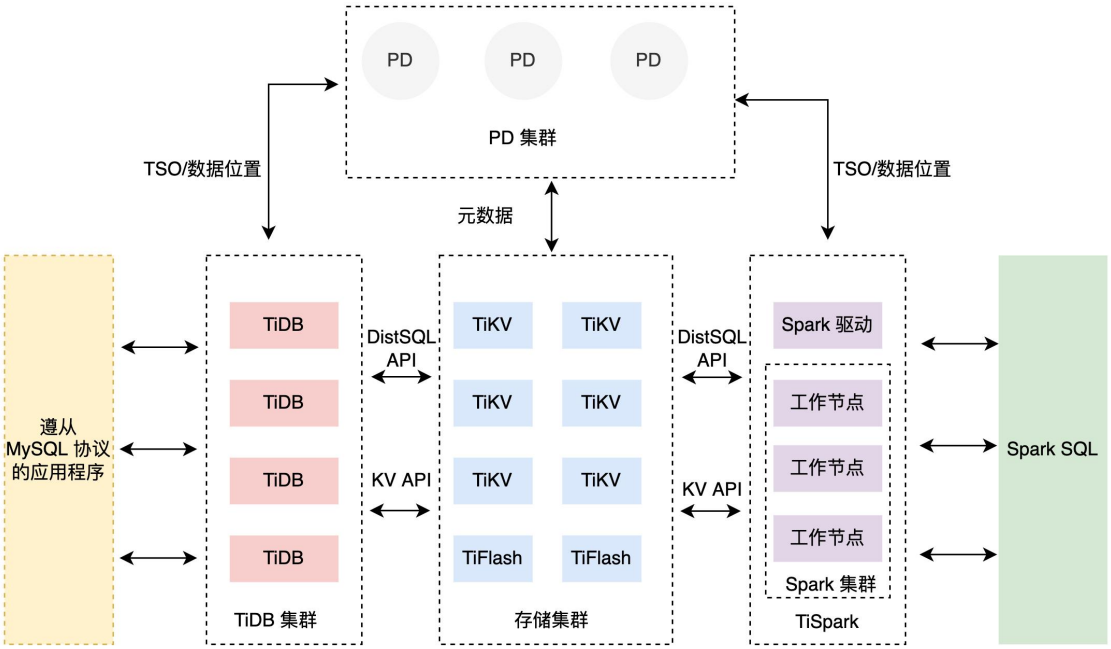
数据卷类型变更

TiDB

TiDB 是一款同时支持在线事务处理（OLTP）与在线分析处理（OATP）的融合型分布式数据库产品，具备水平扩缩容、金融级高可用、实时 HTAP（即同时支持 OLTP 和 OATP）、云原生的分布式数据库，兼容 MySQL 5.7 协议和 MySQL 生态等重要特性。TiDB 的目标是为用户提供一站式的 OLTP、OLAP、HTAP 解决方案，适合高可用、强一致要求较高、数据规模较大等各种应用场景。

TiDB 整体架构

TiDB 分布式数据库将整体架构拆分成了多个模块，各模块之间互相通信，组成完整的 TiDB 系统。对应的架构图如下：



TiDB 架构图

- **TiDB Server**

  SQL 层对外暴露 MySQL 协议的连接端点，负责接受客户端的连接，执行 SQL 解析和优化，最终生成分布式执行计划。TiDB 层本身是无状态的，

实践中可以启动多个 TiDB 实例，通过负载均衡组件（如 LVS、HAProxy 或 F5）对外提供统一的接入地址，客户端的连接可以均匀地分摊在多个 TiDB 实例上以达到负载均衡的效果。TiDB Server 本身并不存储数据，只是解析 SQL，将实际的数据读取请求转发给底层的存储节点 TiKV（或 TiFlash）。

- **PD (Placement Driver) Server**

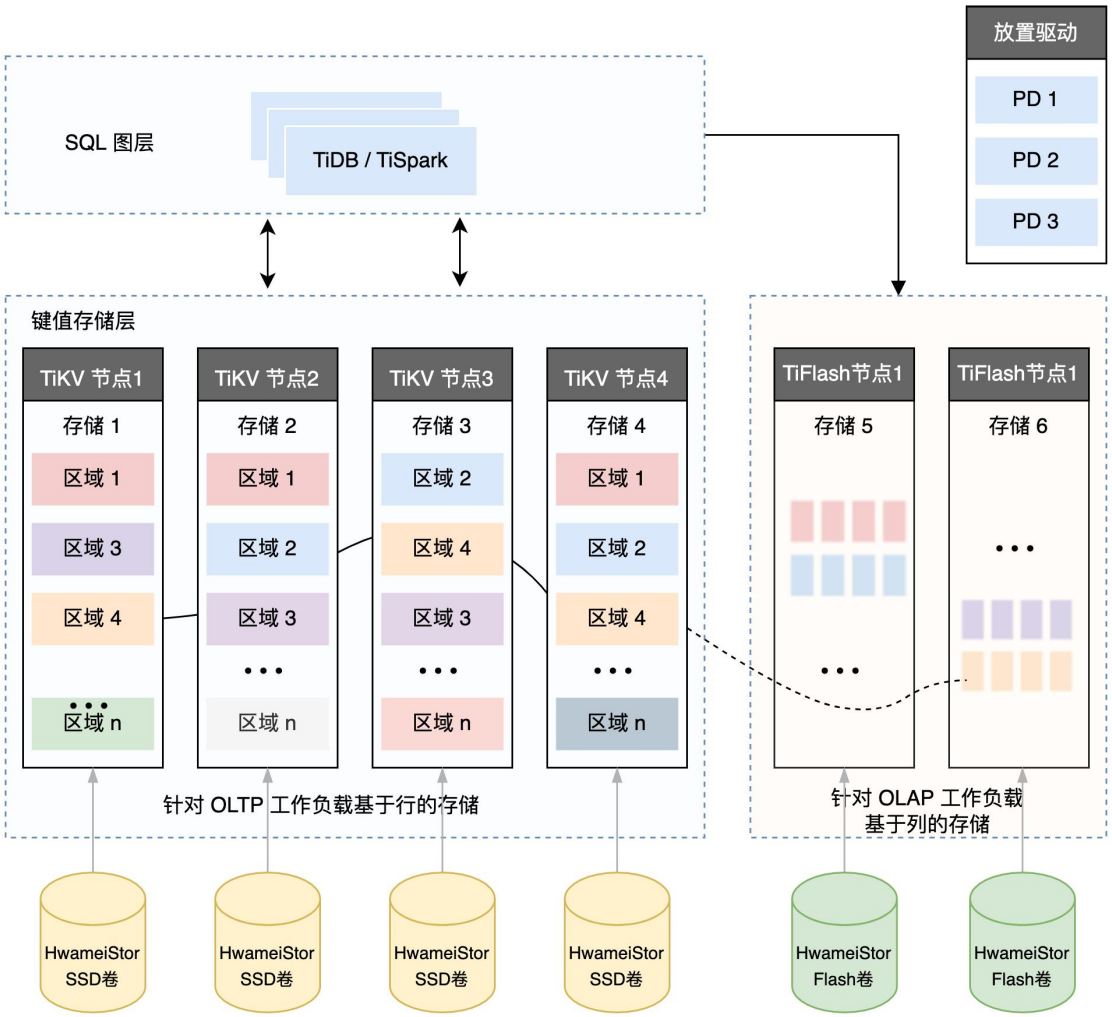
  整个 TiDB 集群的元信息管理模块，负责存储每个 TiKV 节点实时的数据分布情况和集群的整体拓扑结构，提供 TiDB Dashboard 管控界面，并为分布式事务分配事务 ID。PD 不仅存储元信息，同时还会根据 TiKV 节点实时上报的数据分布状态，下发数据调度命令给具体的 TiKV 节点，可以说是整个集群的“大脑”。此外，PD 本身也是由至少 3 个节点构成，拥有高可用的能力。建议部署奇数个 PD 节点。

- **存储节点**

  – TiKV Server：负责存储数据，从外部看 TiKV 是一个分布式的提供事务的 Key-Value 存储引擎。存储数据的基本单位是 Region，每个 Region 负责存储一个 Key Range（从 StartKey 到 EndKey 的左闭右开区间）的数据，每个 TiKV 节点会负责多个 Region。TiKV 的 API 在 KV 键值对层面提供对分布式事务的原生支持，默认提供了 SI (Snapshot Isolation) 的隔离级别，这也是 TiDB 在 SQL 层面支持分布式事务的核心。TiDB 的 SQL 层做完 SQL 解析后，会将 SQL 的执行计划转换为对 TiKV API 的实际调用。所以，数据都存储在 TiKV 中。另外，TiKV 中的数据都会自动维护多副本（默认为三副本），天然支持高可用和自动故障转移。

  – TiFlash：TiFlash 是一类特殊的存储节点。和普通 TiKV 节点不一样的是，在 TiFlash 内部，数据是以列式的形式进行存储，主要的功能是为分析型的场景加速。

TiDB 数据库的存储



TiDB 数据库的存储

- **键值对（Key-Value Pair）**

  TiKV 的选择是 Key-Value 模型，并且提供有序遍历方法。TiKV 数据存储的两个关键点：

  - 这是一个巨大的 Map（可以类比一下 C++ 的 std::map），也就是存储的是 Key-Value Pairs。
  - 这个 Map 中的 Key-Value pair 按照 Key 的二进制顺序有序，也就是可以 Seek 到某一个 Key 的位置，然后不断地调用 Next 方法以递增的顺序获取比这个 Key 大的 Key-Value。

- **本地存储（Rocks DB）**

  任何持久化的存储引擎，数据终归要保存在磁盘上，TiKV 也不例外。但是 TiKV 没有选择直接向磁盘上写数据，而是把数据保存在 RocksDB 中，具体的数据落地由 RocksDB 负责。这样做的原因是开发一个单机存储引擎工作量很大，特别是要做一个高性能的单机引擎，需要做各种细致的优化，而 RocksDB 是由 Facebook 开源的一个非常优秀的单机 KV 存储引擎，可以满足 TiKV 对单机引擎的各种要求。这里可以简单地认为 RocksDB 是一个单机的持久化 Key-Value Map。

- **Raft 协议**

  TiKV 选择了 Raft 算法来保证单机失效的情况下数据不丢失不出错。简单来说，就是把数据复制到多台机器上，这样某一台机器无法提供服务时，其他机器上的副本还能提供服务。这个数据复制方案可靠并且高效，能处理副本失效的情况。

- **Region**

  TiKV 选择了按照 Key 划分 Range。某一段连续的 Key 都保存在一个存储节点上。将整个 Key-Value 空间分成很多段，每一段是一系列连续的 Key，称为一个 Region。尽量让每个 Region 中保存的数据不超过一定的大小，目前在 TiKV 中默认是不超过 96MB。每一个 Region 都可以用 [StartKey，EndKey] 这样的左闭右开区间来描述。

- **MVCC**

  TiKV 实现了多版本并发控制（MVCC）。

- **分布式 ACID 事务**

  TiKV 的事务采用的是 Google 在 BigTable 中使用的事务模型：Percolator。

搭建测试环境

Kubernetes 集群

本次测试使用三台虚拟机节点部署 Kubernetes 集群，包括 1 个 master 节点和 2 个 worker 节点。Kubelete 版本为 1.22.0。

```
[root@10-6-118-10 ~]# kubectl get no
NAME          STATUS   ROLES                  AGE    VERSION
10-6-118-10   Ready    control-plane,master   115d   v1.22.0
10-6-118-11   Ready    <none>                 115d   v1.22.0
10-6-118-12   Ready    <none>                 115d   v1.22.0
```

Kubernetes cluster

HwameiStor 本地存储

1. 在 Kubernetes 集群上部署 HwameiStor 本地存储



HwameiStor 本地存储

2. 在两台 worker 节点上分别为 HwameiStor 配置一块 100G 的本地磁盘 sdb

```
[root@10-6-118-11 ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0  100G  0 disk
|-sda1        8:1    0  600M  0 part /boot/efi
|-sda2        8:2    0    1G  0 part /boot
`-sda3        8:3    0 48.4G  0 part
  |-cl-root 253:0    0 43.4G  0 lvm  /
  `-cl-swap 253:1    0    5G  0 lvm
sdb           8:16   0  100G  0 disk
|-LocalStorage_PoolHDD-pvc--4da25356--8391--4e41--8968--5ca568f58628
|           253:2    0    1G  0 lvm  /var/lib/kubelet/pods/cf5c5a69-9328-
`-LocalStorage_PoolHDD-pvc--15a41340--d5da--40d9--832a--f7d890f0effc
            253:3    0    1G  0 lvm  /var/lib/kubelet/pods/12d18384-f553-
sr0          11:0    1 1024M  0 rom
```

sdb1

```
[root@10-6-118-12 ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0  100G  0 disk
|-sda1        8:1    0  600M  0 part /boot/efi
|-sda2        8:2    0    1G  0 part /boot
`-sda3        8:3    0 48.4G  0 part
  |-cl-root 253:0    0 43.4G  0 lvm  /
  `-cl-swap 253:1    0    5G  0 lvm
sdb           8:16   0  100G  0 disk
|-LocalStorage_PoolHDD-pvc--cab39d37--252a--45c6--ad16--dfb499933abb
|           253:2    0    1G  0 lvm
|-LocalStorage_PoolHDD-pvc--2c4fcf89--28a9--40b5--9ca2--cb713ebbcae0
|           253:3    0    1G  0 lvm
`-LocalStorage_PoolHDD-pvc--68286885--303a--4234--9953--3034ca1f4809
            253:4    0    1G  0 lvm
sdc           8:32   0   20G  0 disk
|-LocalStorage_PoolHDD-pvc--cab39d37--252a--45c6--ad16--dfb499933abb
|           253:2    0    1G  0 lvm
|-LocalStorage_PoolHDD-pvc--2c4fcf89--28a9--40b5--9ca2--cb713ebbcae0
|           253:3    0    1G  0 lvm
|-LocalStorage_PoolHDD-pvc--68286885--303a--4234--9953--3034ca1f4809
|           253:4    0    1G  0 lvm
`-LocalStorage_PoolHDD-pvc--e6020796--366b--4654--8192--c146b2ee2d79
            253:5    0    1G  0 lvm  /var/lib/kubelet/pods/e84287a9-979e-4954-9ff2-73e95ef
sr0          11:0    1 1024M  0 rom
```

sdb2

3. 创建 storagClass

```
[root@10-6-118-10 ~]# kubectl get sc
NAME                          PROVISIONER         RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
local-storage-hdd-lvm (default)   lvm.hwameistor.io   Delete          WaitForFirstConsumer   true
```

创建 StorageClass

在 Kubernetes 上部署 TiDB

可以使用 TiDB Operator 在 Kubernetes 上部署 TiDB。TiDB Operator 是 Kubernetes 上的 TiDB 集群自动运维系统，提供包括部署、升级、扩缩容、备份恢复、配置变更的 TiDB 全生命周期管理。借助 TiDB Operator，TiDB 可以无缝运行在公有云或私有部署的 Kubernetes 集群上。

TiDB 与 TiDB Operator 版本的对应关系如下：

TiDB 版本	适用的 TiDB Operator 版本
dev	dev
TiDB >= 5.4	1.3
5.1 <= TiDB < 5.4	1.3（推荐），1.2
3.0 <= TiDB < 5.1	1.3（推荐），1.2，1.1
2.1 <= TiDB < 3.0	1.0（停止维护）

部署 TiDB Operator

1. 安装 TiDB CRDs

   ```
   kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-operator/master/manifests/crd.yaml
   ```

2. 安装 TiDB Operator

   ```
   helm repo add pingcap https://charts.pingcap.org/
   kubectl create namespace tidb-admin
   helm install --namespace tidb-admin tidb-operator pingcap/tidb-operator --version v1.3.2 \
   --set operatorImage=registry.cn-beijing.aliyuncs.com/tidb/tidb-operator:v1.3.2 \
   ```

--set tidbBackupManagerImage=registry.cn-beijing.aliyuncs.com/tidb/tidb-backup-manager:v1.3.2 \

--set scheduler.kubeSchedulerImageName=registry.cn-hangzhou.aliyuncs.com/google_containers/kube-scheduler

3. 检查 TiDB Operator 组件

```
[root@10-6-118-10 ~]# kubectl get po -n tidb-admin -owide
NAME                                       READY   STATUS    RESTARTS   AGE   IP              NODE
tidb-controller-manager-85bf9df576-m7ft9   1/1     Running   0          24d   100.71.91.101   10-6-118-11
tidb-scheduler-8665f9c787-zjj45            2/2     Running   0          24d   100.71.91.100   10-6-118-11
```

检查 TiDB Operator 组件

部署 TiDB 集群

```
kubectl create namespace tidb-cluster && \
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/pingcap/tidb-operator/master/examples/basic/tidb-cluster.yaml
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com /pingcap/tidb-operator/master/examples/basic/tidb-monitor.yaml
```

```
[root@10-6-118-10 ~]# kubectl get po -n tidb-cluster -owide
NAME                               READY   STATUS    RESTARTS   AGE   IP              NODE
basic-discovery-857c766dbf-c8qvw   1/1     Running   0          24d   100.125.92.38   10-6-118-12
basic-monitor-0                    4/4     Running   0          24d   100.125.92.39   10-6-118-12
basic-pd-0                         1/1     Running   0          23d   100.71.91.103   10-6-118-11
basic-tidb-0                       2/2     Running   0          23d   100.71.91.105   10-6-118-11
basic-tikv-0                       1/1     Running   0          23d   100.71.91.104   10-6-118-11
```

部署 TiDB 集群

连接 TiDB 集群

```
yum -y install mysql-client
```

```
[root@10-6-118-10 ~]# kubectl get svc -n tidb-cluster
NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)              AGE
basic-discovery           ClusterIP   10.110.2.5       <none>        10261/TCP,10262/TCP  24d
basic-grafana             ClusterIP   10.98.41.45      <none>        3000/TCP             24d
basic-monitor-reloader    ClusterIP   10.104.251.157   <none>        9089/TCP             24d
basic-pd                  ClusterIP   10.111.197.222   <none>        2379/TCP             24d
basic-pd-peer             ClusterIP   None             <none>        2380/TCP,2379/TCP    24d
basic-prometheus          ClusterIP   10.101.187.152   <none>        9090/TCP             24d
basic-tidb                ClusterIP   10.105.49.180    <none>        4000/TCP,10080/TCP   23d
basic-tidb-peer           ClusterIP   None             <none>        10080/TCP            23d
basic-tikv-peer           ClusterIP   None             <none>        20160/TCP            23d
[root@10-6-118-10 ~]#
```

connecttidb

kubectl port-forward -n tidb-cluster svc/basic-tidb 4000 > pf4000.out &

```
[root@10-6-118-10 ~]# kubectl port-forward -n tidb-cluster svc/basic-tidb 4000 > pf4000.out &
[1] 1890112
```

连接 TiDB 集群

```
[root@10-6-118-10 ~]# cat pf4000.out
Forwarding from 127.0.0.1:4000 -> 4000
Forwarding from [::1]:4000 -> 4000
Handling connection for 4000
```

连接 TiDB 集群

```
[root@10-6-118-10 ~]# mysql -h 127.0.0.1 -P 4000 -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 413987
Server version: 5.7.25-TiDB-v5.4.0 TiDB Server (Apache License 2.0) Community Edition, MySQL 5.7 compatible

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

连接 TiDB 集群

检查并验证 TiDB 集群状态

1. 创建 Hello_world 表

```
create table hello_world (id int unsigned not null auto_increment primary key, v varchar(32));
```

```
mysql> create table hello_world (id int unsigned not null auto_increment primary key, v varchar(32));
Query OK, 0 rows affected (0.11 sec)

mysql> show tables;
+------------------------+
| Tables_in_testdb       |
+------------------------+
| employees              |
| employees1             |
| employees2             |
| hello_world            |
| members                |
| quarterly_report_status |
| seq2                   |
| t                      |
| t1                     |
| t_test                 |
| t_test1                |
| t_test_1               |
| t_test_boolean_type    |
| t_test_char_type       |
| t_test_int_type        |
| t_test_time_type       |
| t_test_varchar_type    |
| test_json              |
| testlob                |
+------------------------+
19 rows in set (0.00 sec)
```

创建 Hello_world 表

2. 查询 TiDB 版本号

select tidb_version()\G;

```
mysql> Select tidb_version()\G
*************************** 1. row ***************************
tidb_version(): Release Version: v5.4.0
Edition: Community
Git Commit Hash: 55f3b24c1c9f506bd652ef1d162283541e428872
Git Branch: heads/refs/tags/v5.4.0
UTC Build Time: 2022-01-25 08:39:26
GoVersion: go1.16.4
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
1 row in set (0.01 sec)
```

连接 TiDB 集群

3. 查询 Tikv 存储状态

```
select * from information_schema.tikv_store_status\G;
```

```
mysql> select * from information_schema.tikv_store_status\G;
*************************** 1. row ***************************
         STORE_ID: 1001
          ADDRESS: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
      STORE_STATE: 0
 STORE_STATE_NAME: Up
            LABEL: null
          VERSION: 5.4.0
         CAPACITY: 1014MiB
        AVAILABLE: 550.7MiB
     LEADER_COUNT: 102
    LEADER_WEIGHT: 1
     LEADER_SCORE: 102
      LEADER_SIZE: 127
     REGION_COUNT: 102
    REGION_WEIGHT: 1
     REGION_SCORE: 127
      REGION_SIZE: 127
         START_TS: 2022-04-28 02:01:11
LAST_HEARTBEAT_TS: 2022-05-22 01:07:31
           UPTIME: 575h6m20.278854064s
1 row in set (0.01 sec)

ERROR:
No query specified
```

查询 Tikv 存储状态

HwameiStor 存储配置

从 storageClass local-storage-hdd-lvm 分别为 tidb-tikv 及 tidb-pd 创建一个 PVC:

```
[root@10-6-118-10 ~]# kubectl get pvc -A
NAMESPACE      NAME                  STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS           AGE
hwameistor     local-storage-pvc-lvm   Bound    pvc-e6020796-366b-4654-8192-c146b2ee2d79   1Gi        RWO            local-storage-hdd-lvm   28d
tidb-cluster   pd-basic-pd-0           Bound    pvc-4da25356-8391-4e41-8968-5ca568f58628   1Gi        RWO            local-storage-hdd-lvm   24d
tidb-cluster   tikv-basic-tikv-0       Bound    pvc-15a41340-d5da-40d9-832a-f7d890f0effc   1Gi        RWO            local-storage-hdd-lvm   24d
[root@10-6-118-10 ~]#
```

HwameiStor 存储配置

```
[root@10-6-118-10 ~]# kubectl get pvc pd-basic-pd-0 -oyaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    tidb.pingcap.com/pod-name: basic-pd-0
    volume.beta.kubernetes.io/storage-provisioner: lvm.hwameistor.io
    volume.kubernetes.io/selected-node: 10-6-118-11
  creationTimestamp: "2022-04-28T01:56:53Z"
  finalizers:
  - kubernetes.io/pvc-protection
  labels:
    app.kubernetes.io/component: pd
    app.kubernetes.io/instance: basic
    app.kubernetes.io/managed-by: tidb-operator
    app.kubernetes.io/name: tidb-cluster
    tidb.pingcap.com/cluster-id: "7091468044581664630"
    tidb.pingcap.com/member-id: "7441053368211532809"
    tidb.pingcap.com/pod-name: basic-pd-0
  name: pd-basic-pd-0
  namespace: tidb-cluster
  resourceVersion: "25932900"
  uid: 4da25356-8391-4e41-8968-5ca568f58628
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: local-storage-hdd-lvm
  volumeMode: Filesystem
  volumeName: pvc-4da25356-8391-4e41-8968-5ca568f58628
status:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Gi
  phase: Bound
```

HwameiStor 存储配置

```
[root@10-6-118-10 ~]# kubectl get pvc tikv-basic-tikv-0 -oyaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    tidb.pingcap.com/pod-name: basic-tikv-0
    volume.beta.kubernetes.io/storage-provisioner: lvm.hwameistor.io
    volume.kubernetes.io/selected-node: 10-6-118-11
  creationTimestamp: "2022-04-28T01:58:53Z"
  finalizers:
  - kubernetes.io/pvc-protection
  labels:
    app.kubernetes.io/component: tikv
    app.kubernetes.io/instance: basic
    app.kubernetes.io/managed-by: tidb-operator
    app.kubernetes.io/name: tidb-cluster
    tidb.pingcap.com/cluster-id: "7091468044581664630"
    tidb.pingcap.com/pod-name: basic-tikv-0
    tidb.pingcap.com/store-id: "1001"
  name: tikv-basic-tikv-0
  namespace: tidb-cluster
  resourceVersion: "25932910"
  uid: 15a41340-d5da-40d9-832a-f7d890f0effc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: local-storage-hdd-lvm
  volumeMode: Filesystem
  volumeName: pvc-15a41340-d5da-40d9-832a-f7d890f0effc
status:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 1Gi
  phase: Bound
```

HwameiStor 存储配置

kubectl get po basic-tikv-0 -oyaml

```
    volumeMounts:
    - mountPath: /etc/podinfo
      name: annotations
      readOnly: true
    - mountPath: /var/lib/tikv
      name: tikv
    - mountPath: /etc/tikv
      name: config
      readOnly: true
    - mountPath: /usr/local/bin
      name: startup-script
      readOnly: true
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-6wfl9
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  hostname: basic-tikv-0
  nodeName: 10-6-118-11
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  subdomain: basic-tikv-peer
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: tikv
    persistentVolumeClaim:
      claimName: tikv-basic-tikv-0
  - downwardAPI:
      defaultMode: 420
```

HwameiStor 存储配置

```
kubectl get po basic-pd-0 -oyaml
```

```
    volumeMounts:
    - mountPath: /etc/podinfo
      name: annotations
      readOnly: true
    - mountPath: /etc/pd
      name: config
      readOnly: true
    - mountPath: /usr/local/bin
      name: startup-script
      readOnly: true
    - mountPath: /var/lib/pd
      name: pd
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-sbqvd
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  hostname: basic-pd-0
  nodeName: 10-6-118-11
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  subdomain: basic-pd-peer
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: pd
    persistentVolumeClaim:
      claimName: pd-basic-pd-0
  - downwardAPI:
      defaultMode: 420
```

HwameiStor 存储配置

测试内容

数据库 SQL 基本能力测试

完成部署数据库集群后，执行了以下基本能力测试，全部通过。

分布式事务

测试目的：支持在多种隔离级别下，实现分布式数据操作的完整性约束即 ACID 属性

测试步骤：

1. 创建测试数据库 CREATE DATABASE testdb
2. 创建测试用表 CREATE TABLE t_test (id int AUTO_INCREMENT, name varchar(32), PRIMARY KEY (id))
3. 运行测试脚本

测试结果：支持在多种隔离级别下，实现分布式数据操作的完整性约束即 ACID 属性

对象隔离

测试目的：测试不同 schema 实现对象隔离

测试脚本：

```
create database if not exists testdb;
use testdb
create table if not exists t_test
( id                    bigint,
  name                    varchar(200),
  sale_time               datetime default current_timestamp,
  constraint pk_t_test primary key (id)
);
```

```
insert into t_test(id,name) values (1,'a'),(2,'b'),(3,'c');
create user 'readonly'@'%' identified by "readonly";
grant select on testdb.* to readonly@'%';
select * from testdb.t_test;
update testdb.t_test set name='aaa';
create user 'otheruser'@'%' identified by "otheruser";
```

测试结果：支持创建不同 schema 实现对象隔离

表操作支持

测试目的：测试是否支持创建、删除和修改表数据、DML、列、分区表

测试步骤：连接数据库后按步骤执行测试脚本

测试脚本：

```
# 创建和删除表
drop table if exists t_test;
create table if not exists t_test
( id                    bigint default '0',
  name                    varchar(200) default '' ,
  sale_time               datetime default current_timestamp,
  constraint pk_t_test primary key (id)
);
# 删除和修改
insert into t_test(id,name) values (1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e');
update t_test set name='aaa' where id=1;
update t_test set name='bbb' where id=2;
delete from t_dml where id=5;
# 修改、增加、删除列
alter table t_test modify column name varchar(250);
alter table t_test add column col varchar(255);
insert into t_test(id,name,col) values(10,'test','new_col');
alter table t_test add column colwithdefault varchar(255) default 'aaaa';
```

```
insert into t_test(id,name) values(20,'testdefault');
insert into t_test(id,name,colwithdefault ) values(10,'test','non-default ');
alter table t_test drop column colwithdefault;
# 分区表类型（仅摘录部分脚本）
CREATE TABLE employees (
    id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
)
```

测试结果：支持创建、删除和修改表数据、DML、列、分区表

索引支持

测试目的：验证多种类型的索引（唯一、聚簇、分区、Bidirectional indexes、Expression-based indexes、哈希索引等等）以及索引重建操作。

测试脚本：

```
alter table t_test add unique index udx_t_test (name);
# 默认就是主键聚簇索引
ADMIN CHECK TABLE t_test;
create index time_idx on t_test(sale_time);
alter table t_test drop index time_idx;
admin show ddl jobs;
admin show ddl job queries 156;
create index time_idx on t_test(sale_time);
```

测试结果：支持创建、删除、组合、单列、唯一索引

表达式

测试目的：验证分布式数据库的表达式支持 if、casewhen、forloop、whileloop、loop exit when 等语句（上限 5 类）

前提条件：数据库集群已经部署完成。

测试脚本：

```
SELECT CASE id WHEN 1 THEN 'first' WHEN 2 THEN 'second' ELSE 'OTHERS'
 END AS id_new  FROM t_test;
SELECT IF(id>2,'int2+','int2-') from t_test;
```

测试结果：支持 if、case when、for loop、while loop、loop exit when 等语句（上限 5 类）

执行计划解析

测试目的：验证分布式数据库的执行计划解析支持

前提条件：数据库集群已经部署完成。

测试脚本：

```
explain analyze select * from t_test where id NOT IN (1,2,4);
explain analyze select * from t_test a where EXISTS (select * from t_test b wh
ere a.id=b.id and b.id<3);
explain analyze SELECT IF(id>2,'int2+','int2-') from t_test;
```

测试结果：支持执行计划的解析

执行计划绑定

测试目的：验证分布式数据库的执行计划绑定功能

测试步骤：

1. 查看 sql 语句的当前执行计划
2. 使用绑定特性

3. 查看该 sql 语句绑定后的执行计划

4. 删除绑定

测试脚本：

```
explain select * from employees3 a join employees4 b on a.id = b.id where a.lname='Johnson';
explain select /*+ hash_join(a,b) */ * from employees3 a join employees4 b on a.id = b.id where a.lname='Johnson';
```

测试结果：没有使用 hint 时可能不是 hash_join，使用 hint 后一定是 hash_join。

常用函数

测试目的：验证分布式数据库的标准的数据库函数(支持的函数类型)

测试结果：支持标准的数据库函数

显式/隐式事务

测试目的：验证分布式数据库的事务支持

测试结果：支持显示与隐式事务

字符集

测试目的：验证分布式数据库的数据类型支持

测试结果：目前只支持 UTF-8 mb4 字符集

锁支持

测试目的：验证分布式数据库的锁实现

测试结果：描述了锁的实现方式，R-R/R-W/W-W 情况下阻塞情况，死锁处理方式

隔离级别

测试目的：验证分布式数据库的事务隔离级别

测试结果：支持 si 隔离级别，支持 rc 隔离级别（4.0 GA 版本）

分布式复杂查询

测试目的：验证分布式数据库的分布式复杂查询能力

测试结果：支持跨节点 join 等分布式复杂查询、操作等，支持窗口函数、层次查询

系统安全测试

这部分测试系统安全，完成数据库集群部署后，以下安全测试全部通过。

账号管理与权限测试

测试目的：验证分布式数据库的账号权限管理

测试脚本：

```
select host,user,authentication_string from mysql.user;
create user tidb IDENTIFIED by 'tidb';
select host,user,authentication_string from mysql.user;
set password for tidb =password('tidbnew');
select host,user,authentication_string,Select_priv from mysql.user;
grant select on *.* to tidb;
flush privileges ;
select host,user,authentication_string,Select_priv from mysql.user;
grant all privileges on *.* to tidb;
flush privileges ;
select * from  mysql.user where user='tidb';
revoke select on *.* from tidb;
flush privileges ;
revoke all privileges on *.* from tidb;
flush privileges ;
```

```
grant select(id) on test.TEST_HOTSPOT to tidb;
drop user tidb;
```

测试结果：

- 支持创建、修改删除账号，并配置和密码，支持安全、审计和数据管理三权分立
- 根据不同账号，对数库各个级别权限控制包括：实例/库/表/列级别

访问控制

测试目的：验证分布式数据库的权限访问控制，数据库数据通过赋予基本增删改查访问权限控制

测试脚本：

```
mysql -u root -h 172.17.49.222 -P 4000
drop user tidb;
drop user tidb1;
create user tidb IDENTIFIED by 'tidb';
grant select on tidb.* to tidb;
grant insert on tidb.* to tidb;
grant update on tidb.* to tidb;
grant delete on tidb.* to tidb;
flush privileges;
show grants for tidb;
exit;
mysql -u tidb -h 172.17.49.222 -ptidb -P 4000 -D tidb -e 'select * from aa;'
mysql -u tidb -h 172.17.49.222 -ptidb -P 4000 -D tidb -e 'insert into aa values(2);'
mysql -u tidb -h 172.17.49.222 -ptidb -P 4000 -D tidb -e 'update aa set id=3;'
mysql -u tidb -h 172.17.49.222 -ptidb -P 4000 -D tidb -e 'delete from aa where id=3;'
```

测试结果：数据库数据通过赋予基本增删改查访问权限控制。

白名单

测试目的：验证分布式数据库的白名单功能

测试脚本：

```
mysql -u root -h 172.17.49.102 -P 4000
drop user tidb;
create user tidb@'127.0.0.1' IDENTIFIED by 'tidb';
flush privileges;
select * from mysql.user where user='tidb';
mysql -u tidb -h 127.0.0.1 -P 4000 -ptidb
mysql -u tidb -h 172.17.49.102 -P 4000 -ptidb
```

测试结果：支持 IP 白名单功能，支持 IP 段通配操作

操作日志记录

测试目的：验证分布式数据库的操作监控能力

测试脚本：kubectl -ntidb-cluster logs tidb-test-pd-2 --tail 22

测试结果：记录用户通过运维管理控制台或者 API 执行的关键操作或者错误操作

运维管理测试

这部分测试系统运维，完成数据库集群部署后，以下运维管理测试全部通过。

数据导入导出

测试目的：验证分布式数据库的数据导入导出的工具支持

测试脚本：

```
select * from sbtest1 into outfile '/sbtest1.csv';
load data local infile '/sbtest1.csv' into table test100;
```

测试结果：支持按表、schema、database 级别的逻辑导出导入

慢日志查询

测试目的：获取慢查询的 SQL 信息

前提条件：SQL 执行时间需大于配置的慢查询记录阈值,且 SQL 执行完毕

测试步骤：

1. 调整慢查询阈值到 100ms
2. 执行 sql
3. 查看 log/系统表/dashboard 中的慢查询信息

测试脚本：

```
show variables like 'tidb_slow_log_threshold';
set tidb_slow_log_threshold=100;
select query_time, query from information_schema.slow_query where is_internal =
 false order by query_time desc limit 3;
```

测试结果：可以获取慢查询信息

有关测试详情，请查阅 TiDB on hwameiStor 部署及测试记录。

使用高可用卷

HwameiStor 使用开源的 DRBD 数据同步技术创建**高可用卷**，本章节展示 高可用卷的使用这里我们使用一个 MySQL 应用作为例子。

!!! note

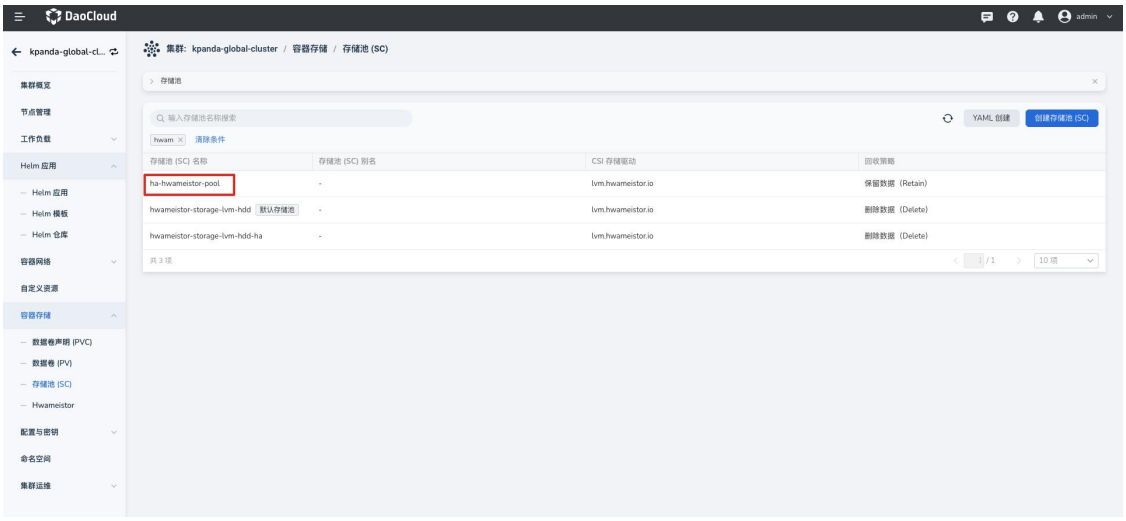
下面的 MySQL Yaml 文件来自于 [Kubernetes 的官方 Repo](https://github.com/kubernetes/website/blob/main/content/en/examples/application/mysql/mysql-statefulset.yaml)

前提条件

- Hwameistor 已经安装成功
- 已经完成高可用存储池创建，如未创建请进行存储池 StorageClass 创建。
- 已经完成 DRDB 内核组件部署

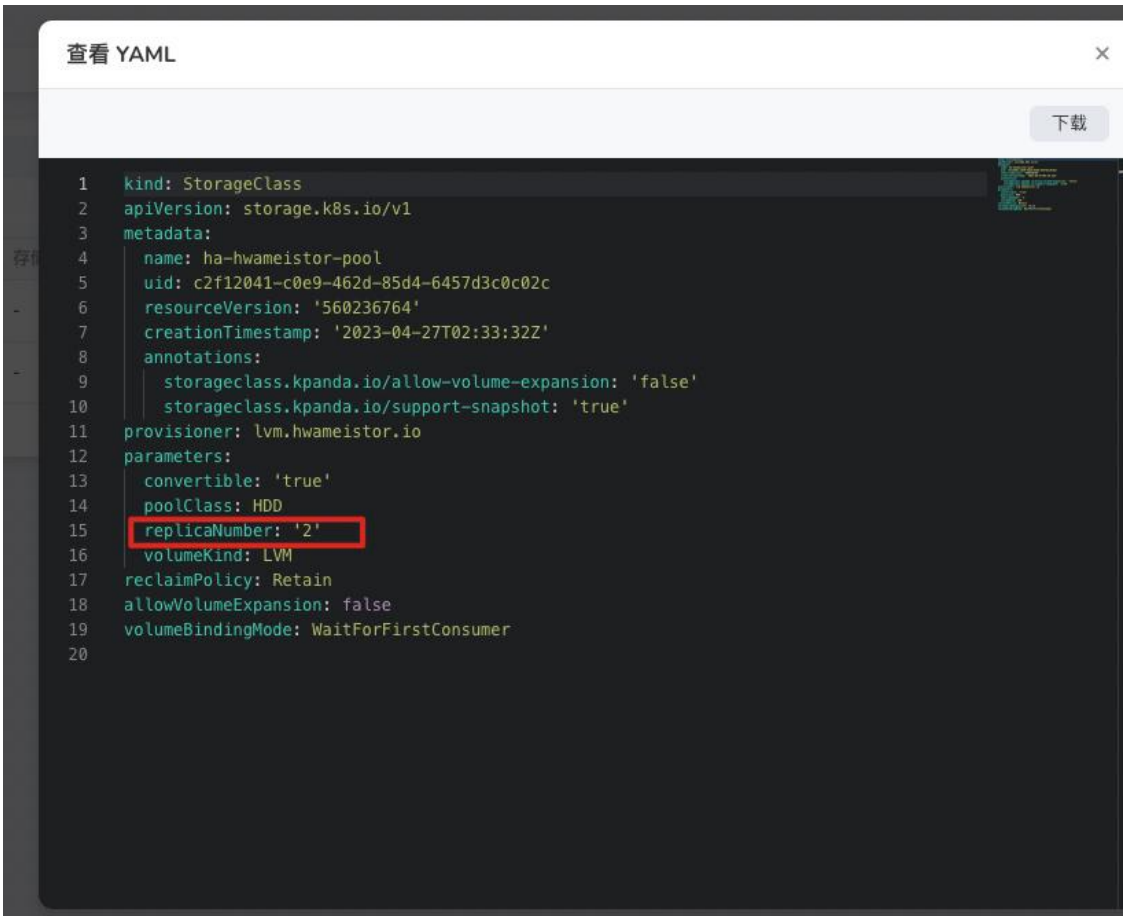
  目前 HwameiStor 安装成功后，Helm chart 会默认安装一个名为 hwameistor-storage-lvm-hdd 的 StorageClass，可使用此存储池创建本地数据卷。

  1. 点击容器管理 -> 选择对应集群，进入集群详情，点击容器存储，确认是否已创建高可用存储池



sc01

2. 点击查看 YAML，查看详情。StorageClass“hwameistor-storage-lvm-hdd-ha”使用参数 replicaNumber: "2" 开启高可用功能：



sc-yaml

$ kubectl apply -f examples/sc_ha.yaml

$ kubectl get sc hwameistor-storage-lvm-hdd-ha -o yaml

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: hwameistor-storage-lvm-hdd-ha

parameters:

  replicaNumber: "2"

  convertible: "false"

  csi.storage.k8s.io/fstype: xfs

  poolClass: HDD

```
    poolType: REGULAR
    striped: "true"
    volumeKind: LVM
  provisioner: lvm.hwameistor.io
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
  allowVolumeExpansion: true
```

创建 StatefulSet

在 HwameiStor 和 StorageClass 就绪后，一条命令就能创建 MySQL 容器和它的数据卷:

```
kubectl apply -f exapmles/sts-mysql_ha.yaml
```

请注意 volumeClaimTemplates 使用 storageClassName: hwameistor-storage-lvm-hdd-ha:

```
spec:
  volumeClaimTemplates:
  - metadata:
      name: data
      labels:
        app: sts-mysql-ha
        app.kubernetes.io/name: sts-mysql-ha
    spec:
      storageClassName: hwameistor-storage-lvm-hdd-ha
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 1Gi
```

查看 MySQL Pod and PVC/PV

在这个例子里，MySQL 容器被调度到了节点 k8s-worker-3。

```
$ kubectl get po -l app=sts-mysql-ha -o wide
NAME                 READY   STATUS    RESTARTS   AGE     IP             NODE
sts-mysql-ha-0       2/2     Running   0          3m08s   10.1.15.151    k8s-worker-1

$ kubectl get pvc -l app=sts-mysql-ha
NAME                          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS                 AGE   VOLUMEMODE
data-sts-mysql-ha-0           Bound    pvc-5236ee6f-8212-4628-9876-1b620a4c4c36   1Gi        RWO            hwameistor-storage-lvm-hdd   3m    Filesystem
```

查看 LocalVolume and LocalVolumeReplica 对象

通过查看和 PV 同名的 LocalVolume(LV)，可以看到本地卷创建在了节点 k8s-worker-1 和节点 k8s-worker-2.

```
$ kubectl get lv pvc-5236ee6f-8212-4628-9876-1b620a4c4c36

NAME                                       POOL                  REPLICAS   CAPACITY     ACCESSIBILITY   STATE   RESOURCE   PUBLISHED   AGE
pvc-5236ee6f-8212-4628-9876-1b620a4c4c36   LocalStorage_PoolHDD   1          1073741824                 Ready   -1         k8s-worker-1,k8s-worker-2   3m
```

LocalVolumeReplica (LVR) 进一步显示每个节点上的后端逻辑卷设备：

```
kubectl get lvr
NAME                                       CAPACITY   NODE          STATE   SYNCED   DEVICE   AGE
```

5236ee6f-8212-4628-9876-1b620a4c4c36-d2kn55 1073741824 k8s-worker-1 Ready true /dev/LocalStorage_PoolHDD-HA/5236ee6f-8212-4628-9876-1b620a4c4c36 4m
5236ee6f-8212-4628-9876-1b620a4c4c36-glm7rf 1073741824 k8s-worker-3 Ready true /dev/LocalStorage_PoolHDD-HA/5236ee6f-8212-4628-9876-1b620a4c4c36 4m

使用非高可用本地卷

使用 HwameiStor 能非常轻松的运行有状态的应用。本文使用一个 MySQL 应用作为例子。

前提条件

- Hwameistor 已经安装成功
- 已经完成存储池创建，如未创建请进行存储池 StorageClass 创建。

  目前 HwameiStor 安装成功后，Helm chart 会默认安装一个名为 hwameistor-storage-lvm-hdd 的 StorageClass，可使用此存储池创建本地数据卷。

  1. 点击`容器管理` -> 选择对应集群，进入集群详情，点击`容器存储`，选择`存储池`

     ![sc01](https://docs.daocloud.io/daocloud-docs-images/docs/storage/images/sc01.jpg)

  2. 点击`查看 YAML`，查看详情。

     ```yaml
     apiVersion: storage.k8s.io/v1
     kind: StorageClass
     metadata:
       name: hwameistor-storage-lvm-hdd
     parameters:
       convertible: "false"
       csi.storage.k8s.io/fstype: xfs
       poolClass: HDD
     ```

```
    poolType: REGULAR
    replicaNumber: "1"
    striped: "true"
    volumeKind: LVM
  provisioner: lvm.hwameistor.io
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
  allowVolumeExpansion: true
```

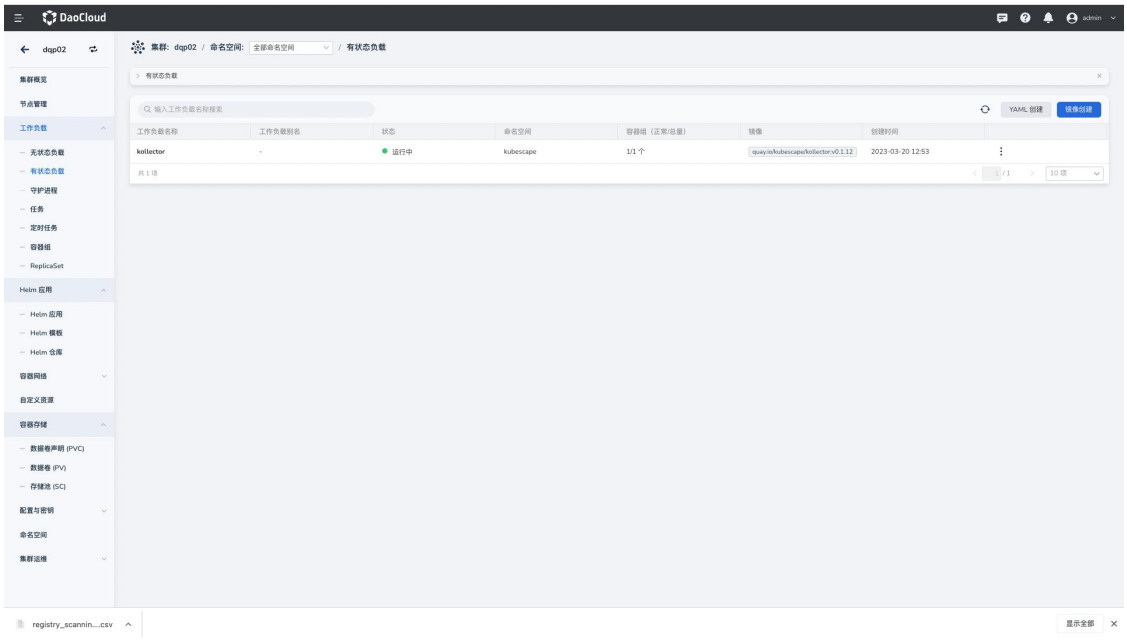
如果这个 `storageClass` 在安装时创建失败，可以[通过界面创建存储池 StorageClass](../../../kpanda/user-guide/storage/sc.md)或 YAML 创建方式完成安装，YAML 方式如下：

```sh
kubectl apply -f examples/sc-local.yaml
```

操作步骤

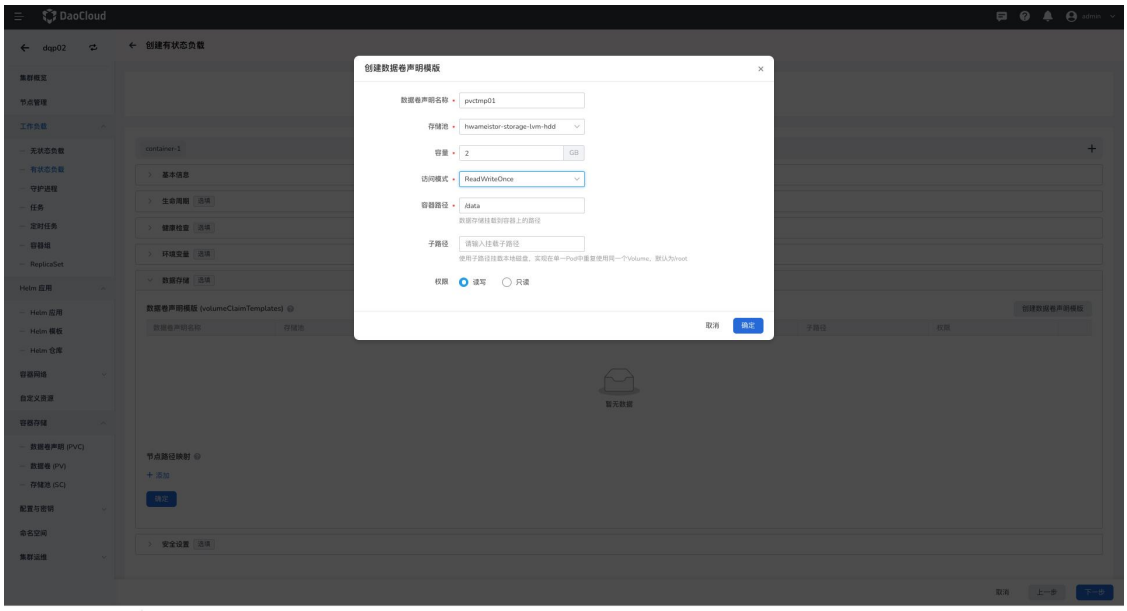
通过界面创建 StatefulSet

1. 点击 容器管理 -> 选择对应集群，进入集群详情，点击容器存储选择工作负载下的有状态工作负载，点击镜像创建。



imagecreate

2. 完成基本信息进入到下一步，点击创建数据卷声明模板输入如下参数信息：

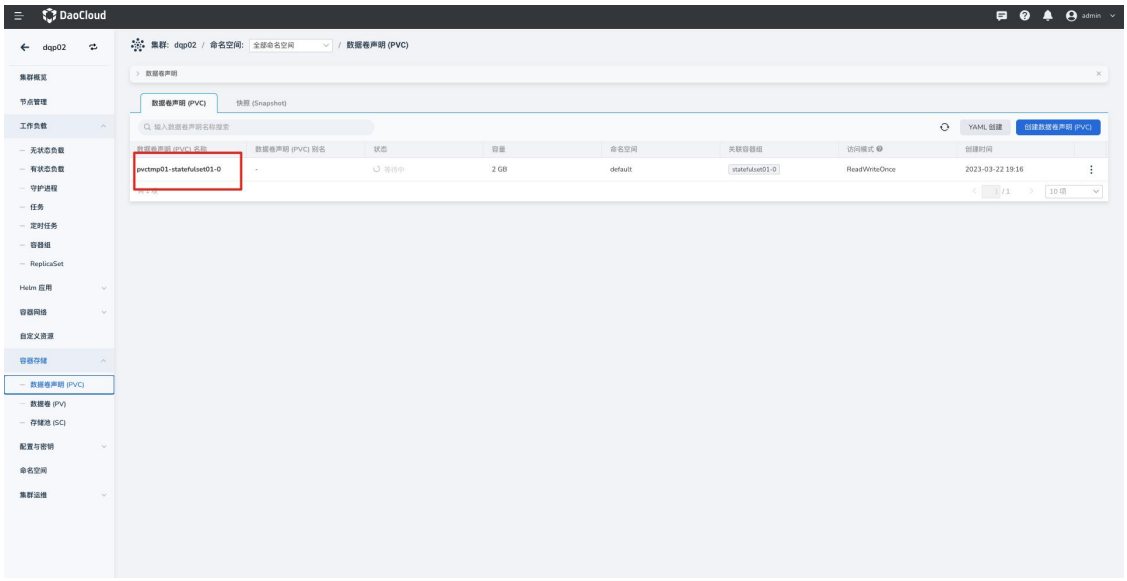


pvctmp

- 存储池：已经创建的本地存储池。
- 容量：本地数据卷容量大小。

- 访问模式：Pod 读写模式，建议使用 ReadWriteOnce。
- 容器路径：数据存储挂载到容器上的路径。

3. 完成后点击确定，连续点击下一步完成创建。创建完成后点击数据卷列表查看对应的数据卷状态是否正常。



pvctmp02

通过 YAML 创建 StatefulSet

!!! note

下面的 MySQL Yaml 文件来自于 [Kubernetes 的官方 Repo](https://github.com/kubernetes/website/blob/main/content/en/examples/application/mysql/mysql-statefulset.yaml)

在 HwameiStor 和 StorageClass 就绪后，一条命令就能创建 MySQL 容器和它的数据卷:

```
kubectl apply -f sts-mysql_local.yaml
```

请注意 volumeClaimTemplates 使用 storageClassName: hwameistor-storage-lvm-hdd:

```
spec:
  volumeClaimTemplates:
  - metadata:
      name: data
      labels:
        app: sts-mysql-local
        app.kubernetes.io/name: sts-mysql-local
    spec:
      storageClassName: hwameistor-storage-lvm-hdd
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 1Gi
```

和 schedulerName: hwameistor-scheduler:

```
spec:
  template:
    spec:
      schedulerName: hwameistor-scheduler
```

查看 MySQL 容器和 PVC/PV

在这个例子里，MySQL 容器被调度到了节点 k8s-worker-3。

```
$ kubectl get po -l app=sts-mysql-local -o wide
NAME                   READY   STATUS    RESTARTS   AGE     IP
 NODE
sts-mysql-local-0   2/2     Running   0          3m08s   10.1.15.154   k8s-w
orker-3

$ kubectl get pvc -l app=sts-mysql-local
NAME                         STATUS   VOLUME
      CAPACITY   ACCESS MODES   STORAGECLASS                  AGE   V
```

OLUMEMODE
data-sts-mysql-local-0 Bound pvc-accf1ddd-6f47-4275-b520-dc317c90f8
0b 1Gi RWO hwameistor-storage-lvm-hdd 3m Filesyst
em

查看 LocalVolume 对象

通过查看和 PV 同名的 LocalVolume(LV)，可以看到本地卷创建在了节点 k8s-worker-3 上：

```
$ kubectl get lv pvc-accf1ddd-6f47-4275-b520-dc317c90f80b

NAME POOL REPLICA
S CAPACITY ACCESSIBILITY STATE RESOURCE PUBLISHED A
GE
pvc-accf1ddd-6f47-4275-b520-dc317c90f80b LocalStorage_PoolHDD 1
1073741824 Ready -1 k8s-worker-3 3m
```

[可选] 扩展 MySQL 应用成一个三节点的集群

HwameiStor 支持 StatefulSet 的横向扩展. StatefulSet 容器都会挂载一个独立的本地卷：

```
$ kubectl scale sts/sts-mysql-local --replicas=3

$ kubectl get po -l app=sts-mysql-local -o wide
NAME READY STATUS RESTARTS AGE IP
NODE
sts-mysql-local-0 2/2 Running 0 4h38m 10.1.15.154 k8s-
worker-3
sts-mysql-local-1 2/2 Running 0 19m 10.1.57.44 k8s-
worker-2
sts-mysql-local-2 0/2 Init:0/2 0 14s 10.1.42.237 k8s-wo
rker-1
```

```
$ kubectl get pvc -l app=sts-mysql-local -o wide
NAME                          STATUS   VOLUME
      CAPACITY   ACCESS MODES   STORAGECLASS                 AGE
 VOLUMEMODE
data-sts-mysql-local-0   Bound    pvc-accf1ddd-6f47-4275-b520-dc317c90f8
0b   1Gi        RWO            hwameistor-storage-lvm-hdd   3m07s   Filesy
stem
data-sts-mysql-local-1   Bound    pvc-a4f8b067-9c1d-450f-aff4-5807d61f5d8
8    1Gi        RWO            hwameistor-storage-lvm-hdd   2m18s   Filesys
tem
data-sts-mysql-local-2   Bound    pvc-47ee308d-77da-40ec-b06e-4f5149952
0c1  1Gi        RWO            hwameistor-storage-lvm-hdd   2m18s   Files
ystem

$ kubectl get lv
NAME                                       POOL                 REPLICA
S   CAPACITY     ACCESSIBILITY   STATE   RESOURCE   PUBLISHED   A
GE
pvc-47ee308d-77da-40ec-b06e-4f51499520c1   LocalStorage_PoolHDD   1
      1073741824                 Ready   -1         k8s-worker-1   2m50s
pvc-a4f8b067-9c1d-450f-aff4-5807d61f5d88   LocalStorage_PoolHDD   1
    1073741824                 Ready   -1         k8s-worker-2   2m50s
pvc-accf1ddd-6f47-4275-b520-dc317c90f80b   LocalStorage_PoolHDD   1
      1073741824                 Ready   -1         k8s-worker-3   3m40s
```

应用工作台视频教程

我们为应用工作台精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

命名空间

参阅命名空间管理文档页面。

流水线

参阅流水线管理文档页面。

GitOps

基于 Git 仓库创建微服务应用

参阅 基于 Git 仓创建微服务应用 文档页面。

基于流水线和 GitOps 实现 CICD

参阅基于流水线和 GitOps 实现 CICD 文档页面。

灰度发布

基于 Argo CD 实现持续部署

参阅创建 Argo CD 应用文档页面。

基于 Argo Rollout 实现渐进式灰度发布

参阅创建金丝雀发布任务文档页面。

镜像仓库视频教程

我们为镜像仓库精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

仓库集成（工作空间）

参阅仓库集成文档页面。

仓库集成（管理员）

参阅仓库集成文档页面。

托管 Harbor

参阅托管 Harbor 文档页面。

申请社区免费体验

云原生网络视频教程

我们为云原生网络精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

- :one: 创建 Pod 多网卡场景
- :two: F5network 的安装与使用
- :three: SR-IOV 的安装与使用
- :four: Ingress Nginx 的安装与使用
- :five: MetalLB 的安装与使用
- :six: kdoctor 的安装与使用

创建 Pod 多网卡场景

参阅 multus-underlay 文档页面。

F5network 的安装与使用

参阅 F5 安装 文档页面。

SR-IOV 的安装与使用

参阅 SR-IOV 文档页面。

Ingress Nginx 的安装与使用

参阅 Ingress Nginx 文档页面。

MetalLB 的安装与使用

参阅 MetalLB 文档页面。

kdoctor 的安装与使用

申请社区免费体验

可观测性视频教程

我们为可观测性 Insight 精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

> 若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

- :one: 安装 insight-agent
- :two: 仪表盘
- :three: 容器监控
- :four: 场景监控和链路追踪
- :five: 日志查询
- :six: 事件查询
- :seven: 告警中心
- :eight: 系统管理

安装 insight-agent

参阅安装 insight-agent 文档页面。

仪表盘

参阅仪表盘文档页面。

容器监控

参阅容器监控文档页面。

场景监控

参阅安装 Insight Agent、通过 Operator 实现应用无侵入增强、链路追踪等文档。

日志查询

参阅日志查询文档页面。

事件查询

参阅事件查询文档页面。

告警中心

参阅告警中心文档页面。

系统管理

参阅修改系统配置文档页面。

申请社区免费体验

数据服务 - 中间件视频教程

我们为中间件精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

Elasticsearch 搜索服务实例管理

参阅 Elasticsearch 搜索服务实例管理文档页面。

Kafka 消息实例管理

参阅 Kafka 消息实例管理文档页面。

MinIO 存储实例管理

参阅 MinIO 消息实例管理文档页面。

MySQL 数据库实例管理

参阅 MySQL 数据库实例管理文档页面。

RabbitMQ 消息队列实例管理

参阅 RabbitMQ 消息服务实例管理文档页面。

Redis 缓存实例管理

参阅 Redis 缓存服务文档页面。

中间件工作负载反亲和性

申请社区免费体验

容器管理视频教程

我们为容器管理精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

集群

接入集群

参阅接入集群文档页面。

创建集群

参阅创建集群文档页面。

集群升级

参阅集群升级文档页面。

命名空间独享节点

参阅命名空间独享节点文档页面。

工作负载

创建无状态负载

参阅创建无状态负载文档页面。

管理 Helm 应用

参阅管理 Helm 应用文档页面。

创建基于指标的 HPA 策略

参阅创建基于指标的 HPA 策略文档页面。

备份恢复

安装 velero 插件实现应用跨集群的备份恢复。

HwameiStor 存储

参阅 HwameiStor 简介文档页面，本视频也上传到了 YouTube。

申请社区免费体验

安装教程

DCE 5.0 分为社区版和商业版。社区版可免费使用，包括全局管理、容器管理、可观测性三大模块。 商业版可按需订购微服务、网格、多云、中间件等所有模块。更多详情可参考什么是 DCE 5.0 及其适用场景。

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

使用 kind 搭建集群

参阅使用 kind 集群在线安装 DCE 5.0 社区版。

在线安装社区版

参阅在线安装 DCE 5.0 社区版。

全局管理视频教程

我们为全局管理精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

用户与访问控制

创建用户并授权

参阅创建用户并授权文档页面。

对接 LDAP 用户和用户组

参阅 LDAP 文档页面。

对接 OIDC 用户

参阅 OIDC 文档页面。

工作空间与层级

管理工作空间 (Workspace)

参阅管理工作空间文档页面。

管理文件夹 (Folder)

参阅管理文件夹文档页面。

资源限额

参阅资源限额文档页面。

审计日志

开启审计日志

参阅开启审计日志文档页面。

关闭审计日志

参阅关闭审计日志文档页面。

平台外观定制

参阅外观定制文档页面。

申请社区免费体验

视频教程

我们制作了精良的视频教程，供大家快速上手。

- :fontawesome-solid-jet-fighter-up: **安装** 安装社区版和商业版
- :octicons-heart-fill-24: **最佳实践** 基于具体使用场景的视频教程
- :octicons-container-16: **容器管理** 集群、节点、负载等基础设施
- :fontawesome-solid-user-group: **全局管理** 设置登录、访问权限、外观
- :material-monitor-dashboard: **可观测性** 一站式图形化仪表盘
- :material-microsoft-azure-devops: **应用工作台** CI/CD 流水线
- :material-cloud-check: **多云编排** 多云实例/负载/策略
- :material-engine: **微服务引擎** 微服务治理中心和网关
- :material-table-refresh: **服务网格** 非侵入式服务治理
- :material-middleware: **中间件** ES, Kafka, MinIO, MySQL 等
- :material-warehouse: **镜像仓库** 镜像集成和托管服务
- :material-dot-net: **网络** 多 CNI 融合方案
- :fontawesome-brands-edge-legacy: **云边协同** 容器能力延伸至边缘

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

多云编排视频教程

我们为多云编排精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

- :one: 多云模块介绍
- :two: 创建多云实例
- :three: 创建多云工作负载
- :four: 创建多云任务
- :five: 管理多云资源
- :six: DCE4 应用升级为 DCE5 应用

多云模块介绍

参阅多云介绍文档页面。

创建多云实例

参阅创建多云实例文档页面。

创建多云工作负载

参阅创建多云工作负载文档页面。

创建多云任务和多云定时任务

参阅创建多云任务、创建多云定时任务文档页面。

管理多云资源

参阅多云服务、多云命名空间、多云配置项、多云密钥文档页面。

DCE 4.0 应用升级为 DCE 5.0 应用

申请社区免费体验

云边协同视频教程

我们为云边协同精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

创建边缘单元

参阅云边协同文档页面。

服务网格视频教程

我们为服务网格精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

- :one: 创建网格
- :two: 创建网格网关
- :three: 通过外部应用访问网格服务
- :four: 安全治理
- :five: Istio 版本升级
- :six: 边车升级
- :seven: 流量透传
- :eight: 多云网络互联
- :nine: 自定义工作负载类型

创建网格

参阅创建网格文档页面。

创建网格网关

参阅网格网关管理文档页面。

通过外部应用访问网格服务

参阅外部应用对网格内服务访问文档页面。

安全治理

参阅安全治理文档页面。

Istio 版本升级

参阅 Istio 版本升级文档页面。

边车升级

参阅边车升级文档页面。

流量透传

参阅流量透传文档页面。

多云网络互联

参阅多云网络互联文档页面。

自定义工作负载类型

参阅自定义工作负载类型文档页面。

申请社区免费体验

微服务引擎视频教程

我们为微服务引擎精心制作了以下视频教程，欢迎任何形式的反馈，谢谢 ��

若视频显示有问题，请刷新页面，或尝试清除浏览器缓存。

云原生网关

管理网关和 API

参阅管理网关和 API 文档页面。

API 策略的高级配置（1）

参阅 API 策略配置文档页面。

API 策略的高级配置（2）

参阅 API 策略配置文档页面。

云原生微服务

参阅通过网格治理微服务流量文档页面。

传统微服务

注册配置中心

参阅微服务配置列表文档页面。

接入注册中心

参阅接入注册中心文档页面。

Sentinel 流控规则

参阅 Sentinel 流控规则文档页面。

最佳实践

最佳实践视频是指，基于具体的业务场景演示如何使用 DCE 5.0 的各项特色，从而满足一些常见的、实际的业务需求。

!!! tip

在右侧的页内导航点击视频名称，可快速定位到对应的视频画面。

如何将单云应用一键转换为多云应用？

如何在 DCE 5.0 中部署混合云应用？

如何将微服务接入 DCE 5.0 并治理南北向流量？

如何借助服务网格治理传统微服务东西向流量？

如何将 DCE 5.0 集成到客户平台？

如何将客户平台集成到 DCE 5.0？

如何让传统应用上云后仍通过固定 IP 对外通信？

如何实现应用加速与跨集群通信？

如何利用可观测性快速定位异常并排障？

如何利用 CICD 快速交付应用？

如何利用 GitOps 持续部署云原生应用？

如何在 DCE 5.0 中部署 ChatGPT 应用？

如何实现跨集群数据同步？

离线升级全局管理模块

本页说明从下载中心下载全局管理模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `ghippo` 字样是全局管理模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

    !!! note

    该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

    ===“已安装 chart repo”

    若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

    ```yaml title="load-image.yaml"
    source:
      intermediateBundlesPath: ghippo-offline # (1)
    target:
      containerRegistry: 10.16.10.111 # (2)
      containerRepository: release.daocloud.io/ghippo # (3)
      repo:
        kind: HARBOR # (4)
    ```

```
    url: http://10.16.10.111/chartrepo/release.daocloud.io # (5)
    auth:
    username: "admin" # (6)
    password: "Harbor12345" # (7)
  containers:
    auth:
      username: "admin" # (8)
      password: "Harbor12345" # (9)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

===“未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: ghippo-offline # (1)
target:
  containerRegistry: 10.16.10.111 # (2)
  containerRepository: release.daocloud.io/ghippo # (3)
  repo:
```

```
      kind: LOCAL
      path: ./local-repo  # (4)
  containers:
    auth:
      username: "admin"  # (5)
      password: "Harbor12345"  # (6)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar xvf ghippo.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```shell
docker load -i images.tar
```

=== “containerd”

```shell
ctr image import images.tar
```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

!!! note

当从 v0.11.x（或更低版本）升级到 v0.12.0（或更高版本）时，需要将 `bak.yaml` 中所有 keycloak key 修改为 `keycloakx`。

这个 key 的修改示例：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
keycloak:
    ...
```

修改为：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
keycloakx:
    ...
```

!!! note

当从 v0.15.x（或更低版本）升级到 v0.16.0（或更高版本）时，需要修改数据库连接参数。

数据库连接参数的修改示例：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
global:
  database:
    host: 127.0.0.1
    port: 3306
    apiserver:
      dbname: ghippo
      password: passowrd
      user: ghippo
    keycloakx:
      dbname: keycloak
      password: passowrd
      user: keycloak
  auditDatabase:
    auditserver:
      dbname: audit
      password: passowrd
      user: audit
    host: 127.0.0.1
```

```
    port: 3306
```

修改为：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
global:
  storage:
    ghippo:
    - driver: mysql
      accessType: readwrite
      dsn: {global.database.apiserver.user}:{global.database.apiserver.password}@tcp({global.database.host}:{global.database.port})/{global.database.apiserver.dbname}?charset=utf8mb4&multiStatements=true&parseTime=true
    audit:
    - driver: mysql
      accessType: readwrite
      dsn: {global.auditDatabase.auditserver.user}:{global.auditDatabase.auditserver.password}@tcp({global.auditDatabase.host}:{global.auditDatabase.port})/{global.auditDatabase.auditserver.dbname}?charset=utf8mb4&multiStatements=true&parseTime=true
    keycloak:
    - driver: mysql
      accessType: readwrite
      dsn: {global.database.keycloakx.user}:{global.database.keycloakx.password}@tcp({global.database.host}:{global.database.port})/{global.database.keycloakx.dbname}?charset=utf8mb4
```

===“通过 helm repo 升级”

1. 检查全局管理 helm 仓库是否存在。

    ```shell
    helm repo list | grep ghippo
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加全局管理的 helm 仓库。

    ```shell
    heml repo add ghippo http://{harbor url}/chartrepo/{project}
    ```

1. 更新全局管理的 helm 仓库。

    ```shell
    helm repo update ghippo # (1)
    ```

    1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

1. 选择您想安装的全局管理版本（建议安装最新版本）。

    ```shell
    helm search repo ghippo/ghippo --versions
    ```

```none
[root@master ~]# helm search repo ghippo/ghippo --versions
NAME                    CHART VERSION   APP VERSION   DESCRIPTION
ghippo/ghippo   0.9.0           v0.9.0        A Helm chart for GHippo
...
```

1. 备份 `--set` 参数。

   在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values ghippo -n ghippo-system -o yaml > bak.yaml
   ```

1. 更新 ghippo crds

   ```shell
   helm pull ghippo/ghippo --version 0.9.0 && tar -zxf ghippo-0.9.0.tgz
   kubectl apply -f ghippo/crds
   ```

1. 执行 `helm upgrade`。

   升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

   ```shell
   export imageRegistry={你的镜像仓库}
   ```

```shell
helm upgrade ghippo ghippo/ghippo \
  -n ghippo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.9.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

   在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values ghippo -n ghippo-system -o yaml > bak.yaml
   ```

1. 更新 ghippo crds

   ```shell
   kubectl apply -f ./crds
   ```

1. 执行 `helm upgrade`。

   升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

   ```shell
   export imageRegistry={你的镜像仓库}
```

```

```shell
helm upgrade ghippo . \
  -n ghippo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```
```

登录

用户在使用一个新系统前，在这个系统中是没有任何数据的，系统也无法识别这个新用户。为了标识用户身份、绑定用户数据，用户需要一个能唯一标识用户身份的帐号。

DCE 5.0 在用户与访问控制中通过管理员创建新用户的方式为用户分配一个附有一定权限的账号。该用户产生的所有行为都将关联到自己的帐号。

用户通过账号/密码进行登录，系统验证身份是否合法，如果验证合法，则用户成功登录。

!!! note

如果用户登录后 2 小时内无任何操作，将自动退出登录状态。如果登录的用户始终活跃，将持续处于登录状态。

用户登录的简单流程如下图。

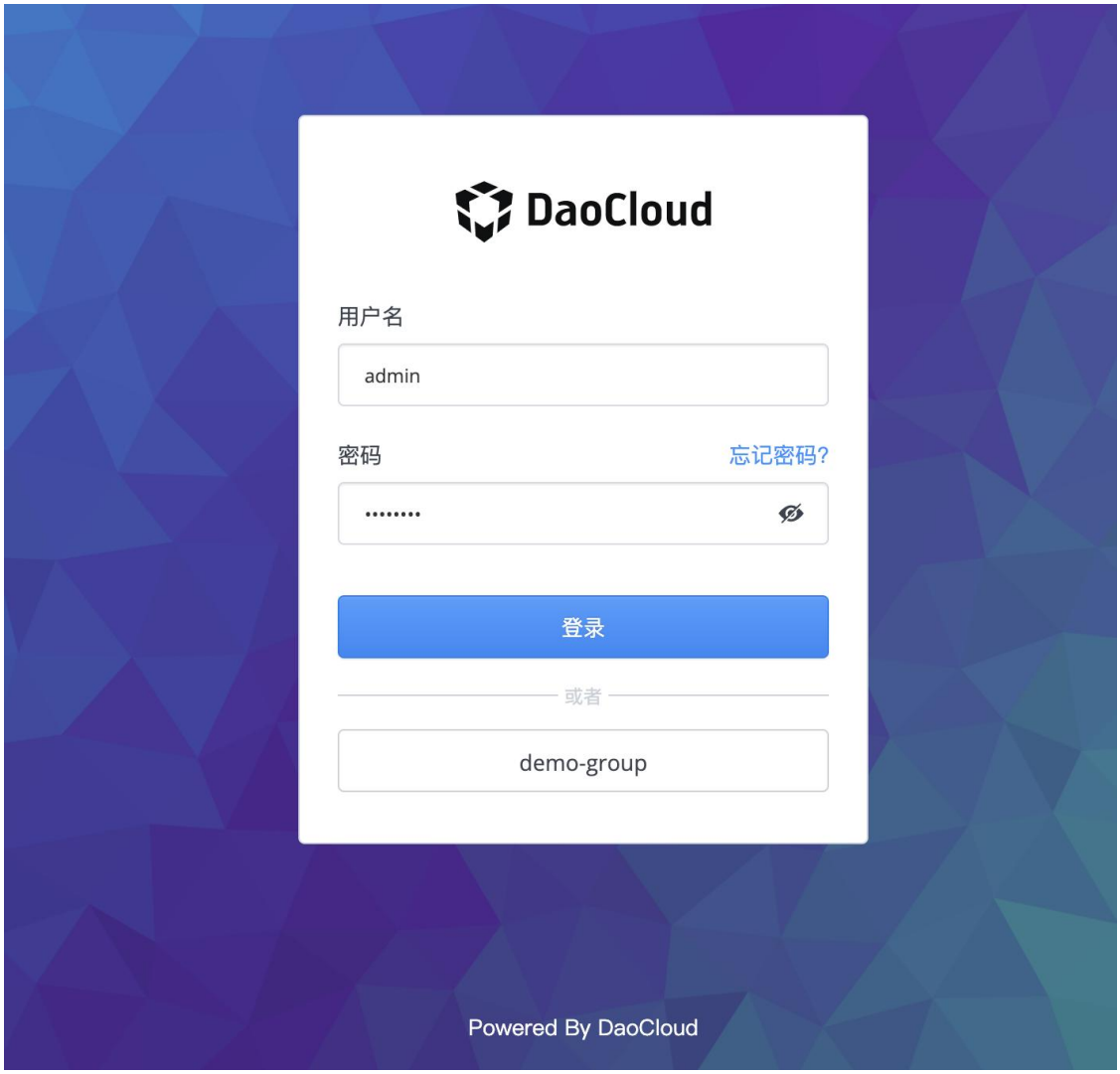
```
graph TB

user[输入用户名] --> pass[输入密码] --> judge([点击登录并校验用户名和密码])
judge -.正确.->success[登录成功]
judge -.错误.->fail[提示错误]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class user,pass cluster;
class judge plain
class success,fail k8s
```

用户登录界面如下图。具体登录画面，请与实际产品为准。



登录界面

使用国密网关代理 DCE 5.0

参照以下步骤为 DCE 5.0 配置国密网关。

软件介绍

Tengine: Tengine 是由淘宝网发起的 Web 服务器项目。它在 Nginx 的基础上，针对大访问量网站的需求，添加了很多高级功能和特性。

Tongsuo: 铜锁/Tongsuo（原 BabaSSL）是一个提供现代密码学算法和安全通信协议的开源基础密码库，为存储、网络、密钥管理、隐私计算等诸多业务场景提供底层的密码学基础能力，实现数据在传输、使用、存储等过程中的私密性、完整性和可认证性，为数据生命周期中的隐私和安全提供保护能力。

准备工作

一台安装了 Docker 的 Linux 主机，并且确保它能访问互联网

编译和安装 Tengine & Tongsuo

注：此配置仅供参考

```
FROM docker.m.daocloud.io/debian:11.3

# Version
ENV TENGINE_VERSION="2.3.4" \
    TONGSUO_VERSION="8.3.2"

# Install required system packages and dependencies
RUN apt update && \
    apt -y install \
    wget \
    gcc \
    make \
```

```
    libpcre3 \
    libpcre3-dev \
    zlib1g-dev \
    perl \
    && apt clean

# Build tengine
RUN mkdir -p /tmp/pkg/cache/ && cd /tmp/pkg/cache/ \
    && wget https://github.com/alibaba/tengine/archive/refs/tags/${TENGINE_VERSION}.tar.gz -O tengine-${TENGINE_VERSION}.tar.gz \
    && tar zxvf tengine-${TENGINE_VERSION}.tar.gz \
    && wget https://github.com/Tongsuo-Project/Tongsuo/archive/refs/tags/${TONGSUO_VERSION}.tar.gz -O Tongsuo-${TONGSUO_VERSION}.tar.gz \
    && tar zxvf Tongsuo-${TONGSUO_VERSION}.tar.gz \
    && cd tengine-${TENGINE_VERSION} \
    && ./configure \
        --add-module=modules/ngx_openssl_ntls \
        --with-openssl=/tmp/pkg/cache/Tongsuo-${TONGSUO_VERSION} \
        --with-openssl-opt="--strict-warnings enable-ntls" \
        --with-http_ssl_module --with-stream \
        --with-stream_ssl_module --with-stream_sni \
    && make \
    && make install \
    && ln -s /usr/local/nginx/sbin/nginx /usr/sbin/ \
    && rm -rf /tmp/pkg/cache

EXPOSE 80 443
STOPSIGNAL SIGTERM
CMD ["nginx", "-g", "daemon off;"]

docker build -t tengine:0.0.1 .
```

生成 SSL 证书（SM2 和 RSA 证书）

注：此证书仅适用于测试环境

您可以参考官方文档使用 Tongsuo 生成 SSL 证书，或者访问 https://www.gmssl.cn/gmssl/index.jsp?go=CA 申请。

SM2 证书

```
-rw-r--r-- 1 root root  749 Dec  8 02:59 sm2.*.io.enc.crt.pem
-rw-r--r-- 1 root root  258 Dec  8 02:59 sm2.*.io.enc.key.pem
-rw-r--r-- 1 root root  749 Dec  8 02:59 sm2.*.io.sig.crt.pem
-rw-r--r-- 1 root root  258 Dec  8 02:59 sm2.*.io.sig.key.pem
```

RSA 证书（可选）

```
-rw-r--r-- 1 root root  216 Dec  8 03:21 rsa.*.io
-rw-r--r-- 1 root root 4096 Dec  8 02:59 sm2.*.io
```

给 nginx 配置 SSL 证书

支持 SM2 和 RSA 两种证书。双证书的优点是：当浏览器不支持国密证书时，自动切换到 RSA 证书。

更多详细配置，请参考官方文档。

```
# 进入 nginx 配置文件存放目录
cd /usr/local/nginx/conf

# 创建 cert 文件夹，用于存放 SSL 证书
mkdir cert

# 把 SM2、RSA（可选）证书拷贝到 `/usr/local/nginx/conf/cert` 目录下
cp sm2.*.enc.crt.pem sm2.*.io.enc.key.pem  sm2.*.sig.crt.pem  sm2.*.sig.key.pem
 /usr/local/nginx/conf/cert
cp rsa.*.crt.pem  rsa.*.key.pem /usr/local/nginx/conf/cert

# 编辑 nginx.conf 配置
```

```
vim nginx.conf
...
server {
  listen 443              ssl;
  proxy_http_version  1.1;
  # 开启国密功能
  enable_ntls             on;

  # 国际 RSA 证书（可选）
  ssl_certificate                 /usr/local/nginx/conf/cert/rsa.demo-dev.daoclo
ud.io.crt.pem;
  ssl_certificate_key             /usr/local/nginx/conf/cert/rsa.demo-dev.daoclo
ud.io.key.pem;

  # 国密签名证书
  ssl_sign_certificate            /usr/local/nginx/conf/cert/sm2.demo-dev.daoclo
ud.io.sig.crt.pem;
  ssl_sign_certificate_key        /usr/local/nginx/conf/cert/sm2.demo-dev.daocl
oud.io.sig.key.pem;
  # 国密加密证书
  ssl_enc_certificate             /usr/local/nginx/conf/cert/sm2.demo-dev.daocl
oud.io.enc.crt.pem;
  ssl_enc_certificate_key         /usr/local/nginx/conf/cert/sm2.demo-dev.daocl
oud.io.enc.key.pem;
  ssl_protocols                   TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;

  location / {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header REMOTE-HOST $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass https://istio-ingressgateway.istio-system.svc.cluster.local;
```

```
    }
}
```

重新加载 nginx，使其配置生效

```
nginx -s reload
```

下一步

国密网关部署成功之后，自定义 DCE 5.0 反向代理服务器地址。

自定义 DCE 5.0 反向代理服务器地址

设置步骤

1. 设置环境变量，方便在下文中使用。

   # 您的反向代理地址，例如：`export DCE_PROXY="https://demo-alpha.daocloud.io"`
   export DCE_PROXY="https://domain:port"

   # helm --set 参数备份文件
   export GHIPPO_VALUES_BAK="ghippo-values-bak.yaml"

   # 获取当前 ghippo 的版本号
   export GHIPPO_HELM_VERSION=$(helm get notes ghippo -n ghippo-system | grep "Chart Version" | awk -F ': ' '{ print $2 }')

2. 更新全局管理 Helm 仓库。

   helm repo update ghippo

3. 备份 一set 参数。

   helm get values ghippo -n ghippo-system -o yaml > ${GHIPPO_VALUES_BAK}

4. 添加您的反向代理地址。

   !!! note

   - 如果可以，您可以使用 `yq` 命令：

     ```shell
     yq -i ".global.reverseProxy = \"${DCE_PROXY}\"" ${GHIPPO_VALUES_
     ```

BAK}
    ```

- 或者您可以使用 `vim` 命令编辑并保存：

    ```shell
    vim ${GHIPPO_VALUES_BAK}

    USER-SUPPLIED VALUES:
    ...
    global:
      ...
      reverseProxy: ${DCE_PROXY} # 只需要修改这一行
    ```

5. 执行 helm upgrade 使配置生效。

    ```
    helm upgrade ghippo ghippo/ghippo \
      -n ghippo-system \
      -f ${GHIPPO_VALUES_BAK} \
      --version ${GHIPPO_HELM_VERSION}
    ```

6. 使用 kubectl 重启全局管理 Pod，使配置生效。

    ```
    kubectl rollout restart deploy/ghippo-apiserver -n ghippo-system
    kubectl rollout restart statefulset/ghippo-keycloakx -n ghippo-system
    ```
    ```

全局管理 Release Notes

本页列出全局管理各版本的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-08-30

v0.20.0

功能

- 支持对接企业微信
- 密码支持 8-32 位
- 支持云边协同 Kant 对接 license
- [运营管理] 支持 postgres/kingbase 数据库

优化

- 给部分没有主键的数据库表加上主键

2023-07-28

v0.19.0

功能

- 支持对接项目组 IDP 登录免点击
- 资源组绑定 mesh / mesh-namespace
- 自定义角色权限点增加 tips
- 平台设置-安全策略-对单个用户的多重并发会话进行限制
- 平台设置-安全策略-能够对系统的最大并发会话连接数进行限制

优化

- Webhook URL 支持自定义参数

修复

- 修复填写邮件服务器时报错问题

- 修复 LDAP 用户加入用户组失败问题

2023-07-06

v0.18.1

修复

- 修复工作空间集群名称可能为空的问题
- 修复 Folder Admin 角色工作空间授权列表权限问题
- 修复 GHippo 0.17 升级到 0.18 失败的问题
- 修复新增共享资源，选择集群类型 options 都相同的问题
- 修复资源组列表，网格类型资源不能解绑

2023-06-29

v0.18.0

功能

- 移动工作空间
- 接入管理 – Webhook（创建/编辑/删除/列表/查看）
- 接入管理 – Webhook 记录后台自动清理
- License 过期前提醒
- License 过期后功能不可用
- 用户在修改密码时需要输入旧密码
- 资源组支持绑定服务网格资源
- 支持外接 Postgres 数据库
- 运营管理 – 报表支持更多类型的货币
- istio-ingressgateway 和 istiod 支持高可用

优化

- 后台配置的密码存入 secret

修复

- 修复审计日志时区问题

- 修复工作空间授权时角色列表不能显示问题

功能

- 移动工作空间
- License 过期前提醒
- License 过期后功能不可用
- Webhook 记录后台清理功能
- 用户在修改密码时需要输入旧密码
- 支持外接 kingbase 数据库
- 运营管理 – 报表支持更多类型的货币
- istio-ingressgateway 和 istiod 支持高可用

优化

- 后台配置的密码存入 secret

修复

- 修复审计日志时区问题

2023-06-02

v0.17.1

修复

- 修复迁移文件错误导致迁移失败的问题
- 修复创建工作空间时，角色只能选择 Workspace Admin 的问题

2023-05-30

v0.17.0

功能

- 接入管理：接入客户端（创建/查看/列表/更新/删除）
- 接入管理：Webhook API（创建/编辑/删除/列表/查看）
- 用户 CRUD/Login/Logout 事件触发 Webhook
- 接入管理提供权限点

- 用户名/用户组名支持 . 和 @
- 平台默认语言改成 自动检测浏览器首选项
- 审计日志支持下载为 Excel 和 CSV 格式
- 审计日志：两种日志（系统和用户）分开显示
- 禁止移除 admin 用户的 admin 角色或编辑 admin 的权限
- 工作空间授权展示角色的权限详情
- 单元测试的覆盖率，只能够上升，不能够下降
- OpenAPI 支持版本，OpenAPI 文档支持弃用（其他 GProduct 模块必须同步更新到安装器 v0.8.0 里对应的版本）
- 运营管理：GMagpie 使用数据库配置规范（Helm 参数、DSN）对接安装器
- 运营管理：报表支持下载为 Excel 和 CSV 格式
- 运营管理：支持 Workspace 报表
- 运营管理：支持 Workspace 计费报表
- 运营管理：支持容器组报表
- 运营管理：支持容器组计费报表
- 运营管理：支持 Namespace 计费报表

优化

- 优化角色列表默认排序
- 修改 Workspace Editor 权限点

修复

- 修复 LDAP 同步用户组的同步时间问题

2023-04-28

v0.16.1

修复

- 修复死循环导致的 CPU 占用过高问题
- 修复审计日志导出不全的问题

2023-04-27

v0.16.0

功能

- LDAP 支持 AD
- LDAP 支持更多参数
- 支持修改登录页背景图
- 支持首页和登录页下方备案信息修改（更新/展示）
- 数据库连接 helm 参数格式修改
- 接入管理–接入客户端 API（创建/查看/列表/更新/删除）
- 反代 URL 支持前面加 Path
- 审计日志 SDK 直接调用 AuditServer API
- 审计日志用户和系统两种审计分表存，分 API 存取
- 去除 API Server 访问外网的依赖

优化

- 创建自定义角色时，过滤不存在的权限
- 简化审计日志数据库分区功能

修复

- 修复自定义角色类型显示错误问题

2023-03-29

v0.15.0

功能

- 自定义角色功能(创建/编辑/删除/查看/列表)
- 支持 GProduct 权限点及自定义角色功能对接
- 失联集群资源处理
- GHippo OpenAPI 文档实现
- 给 GProduct 提供插入审计日志 SDK

修复

- 修复密钥过期后 token 依旧可以使用的问题
- 修复镜像仓库不受 license 控制问题

2023-02-27

v0.14.0

功能

- 平台设置->外观定制->高级定制->登录页定制及登录后页面定制
- 支持 keycloak quarkus 架构在双栈环境运行
- CI 过程中加入 job 检测 helm chart 包中是否有不合规的镜像和检测安装参数是否正确
- 在 helm values 里设置开关，可以一键开关 audit 相关功能
- 审计日志支持记录 kpanda 页面操作

优化

- OpenAPI 调用方式优化

修复

- 修复部分错误的审计日志名称
- 修改 keycloak 启动探针 failureThreshold 值，提高启动成功率
- 修复绑定/解绑资源错误 i18n

2022-12-30

v0.13.2

修复

- 界面补充了缺少权限说明的英文文案
- 修复数据库表更新时，可能因为数据库编码导致报错

2022-12-28

v0.13.0

新功能

- 支持在 DCE 5.0 前面部署一个国密网关，并用国密浏览器来访问 DCE 5.0
- 在 helm values 里设置开关，可以一键开关 Istio Sidecar 功能
- 工作空间与层级增加 Workspace and Folder Owner 角色
- 拥有 Workspace/Folder Admin 和 Kpanda Owner 权限的用户才可进行资源绑定
- 对所用的库，进行开源 License 扫描
- 用户列表新增状态列
- 对内提供离线安装文档
- SDK 单元测试达 65%
- 界面支持发送测试邮件和无账号密码邮件服务器
- 界面支持对不符合系统要求的用户名进行提示

优化

- Ghippo 鉴权代码优化（减少内存使用量）
- 前端界面低网络情况下预加载机制优化

修复

- 修复 OpenAPI cycle 为必填参数问题，修复后为可选参数

2022-11-30

v0.12.1

优化

- 通过 CI 自动构建纯离线包
- 优化 GHippo 升级文档

2022-11-28

v0.12.0

新功能

- 资源组里的模块改为来源
- SDK 提供 Workspace 和 Resource 的绑定变化通知
- 完整对接 Insight metrics 和 otel tracing（加入 keycloak 和 db 链路）
- Keycloak 改成 Quarkus 架构
- Keycloak 镜像升级成 20.0.1 版本

优化

- 重构导出审计日志 http 接口为 gprc stream 接口
- SDK 内存使用量优化，峰值减少 50%
- 审计日志部分代码优化
- e2e 的 kind 镜像切到 1.25
- 提高资源使用效率到 40% 以上

修复

- 修复强绑定集群问题
- 修复配置身份提供商界面选择 Client secret sent as basic auth 没有保存到 keycloak 里的问题

2022-11-01

v0.11.2

修复

- 关闭资源组绑定集群功能
- 修复无工作空间时无法创建工作空间问题

2022-10-28

v0.11.0

新功能

- 给第三方应用提供接口在 Keycloak 创建 SSO 对接 Client
- 支持 Mysql8
- 对接 Insight（metrics, log, otel tracing）
- License 模块名支持 i18n
- 支持一个 License 中可以包含多个 GProduct
- 资源组新增绑定集群类型资源
- 资源组列表增加“模块”字段
- 资源组列表增加已绑定标识
- 资源绑定接口支持 Registry 资源种类。

优化

- 资源种类改枚举
- GProduct license 是否需要灌入变量改为可配
- 优化 CICD 流程

修复

- 修复已经删除的集群依然存在问题
- 修复 keycloak jwks 变化后没有重置 istio 缓存问题
- 修复用户组创建时间零值问题
- 修复访问密钥最后使用时间字段在未使用时返回空字符

2022-09-28

v0.10.0

新功能

- 支持登录
- 支持忘记密码
- 支持站内信增删改查功能

- 支持 SMTP 设置邮件服务器
- 支持顶部导航栏获取查询
- 支持顶部导航栏更新
- 支持用户角色权限管理 CRUD
- 工作空间 -> 生命周期管理（创建/编辑/删除/查看/列表）
- 工作空间 -> 层级关系管理（绑定/列表）
- 工作空间 -> 工作空间与资源关系管理（绑定/解绑/列表）
- 工作空间 -> 工作空间和角色和用户（组）关系管理（绑定/解绑/列表）（API/SDK）
- 工作空间 -> 鉴权（API/SDK）
- 工作空间 -> GProduct 资源名字注册
- 关于 -> 产品版本（创建/编辑/删除/查看/列表）
- 关于 -> 开源软件（列表/初始化）
- 关于 -> 技术团队（列表/初始化）
- 许可证 -> 生命周期管理（创建/编辑/删除/查看/列表）
- 许可证 -> 获取 ESN 序列号
- 许可证 -> 未灌入或错误情况处理
- 工作空间 -> 资源配额管理（创建/编辑/删除/查看/列表/计算已分配）
- 工作空间 -> GProduct 资源配额注册
- 用户与访问控制 -> 鉴权（APIServer/SDK）
- 审计日志 -> 展示（查看/列表/清理设置/导出）
- 审计日志 -> 批量插入
- 身份提供商 -> 对接 LDAP -> 用户/用户组同步设置（创建/编辑/删除/查看/同步）
- 平台设置 -> 安全策略 -> 密码策略设置
- 个人中心 -> 访问密钥(创建/编辑/删除/查看/列表)
- 审计日志 -> 全局管理操作插入审计日志

- 审计日志 -> 对接 Insight 来收集审计日志
- 平台设置 -> 安全策略 -> 账号锁定策略
- 平台设置 -> 安全策略 -> 浏览器关闭策略
- 身份提供商 -> 对接 IDP (OIDC 协议)
- 工作空间 -> 共享集群权限管理
- 工作空间 -> 共享集群配额管理 -> 存储
- 平台设置 -> 顶部导航外观定制 -> 重置功能
- 平台设置 -> 安全策略 -> 会话超时策略
- 审计日志 -> 自动清理功能
- 平台设置 -> 安全策略-账号锁定策略
- 平台设置 -> 顶部导航外观定制-还原功能
- 平台设置 -> 登录页外观定制-还原功能
- 产品导航 -> 首页仅对 admin 用户展示
- 工作空间 -> 用户仅能查看有权限的 workspace & folder 树状结构
- Keycloak 高可用
- 邮件服务器配置 -> 支持 Insight 和应用工作台发送邮件
- 满足 Helm 规范，支持安装器和离线化
- 审计日志 -> 数据库自动创建和合并分区
- 支持 ARM64 架构
- 支持 https
- 登录 -> 背景 theme 支持动画
- 授权鉴权 -> 给前端提供当前登录用户的权限列表
- 关于 -> 软件版本 -> 模块支持中文名
- 添加整体双语文档站结构及主要内容

优化

- 授权鉴权 -> 提供一个 job 来确保 db 和 cr 的同步
- LDAP -> 配置错误检查

- 各功能操作反馈和提示语报错支持中英文
- 工作空间及层级 -> 删除前对是否存在子资源进行检查
- 优化 keycloak jvm 参数
- 通过 mockery 框架简化 mock

常见术语

本节列出全局管理的常见术语。

IAM

IAM（Identity and access management）是用户与访问控制模块的简称，该模块的管理员被称为 IAM Admin，拥有该模块的最高权限。 被赋予 IAM Admin 的用户（用户组），将拥有用户与访问控制的全部且最高权限。

更多详细信息，参见什么是 IAM。

RBAC

RBAC（Role-based access control，基于角色的访问控制），基本理念是将角色这个概念赋予用户，在用户与权限之间通过角色进行关联，实现灵活配置。RBAC 模型的三要素为：用户、角色、权限。使用 RBAC 机制授权 IAM 用户访问平台资源。

用户

用户是发起操作的主体，每个用户都有唯一的 ID，并被授予不同的角色。 默认创建的 IAM 用户没有任何权限，需要将其加入用户组，授予角色或策略，才能让用户获得对应的权限。

用户以用户名登录 DCE，按照被授予的权限操作平台资源和服务。 所以用户是资源归属的主体，对其拥有的资源具有相应权限。

用户可以在个人中心修改用户信息，设置密码、访问密钥和 UI 语言。

用户组

用户组是一个或多个用户的集合，IAM 可以通过用户组实现用户的授权。 通常先创建一个 IAM 用户，加入某个用户组，该用户将继承这个用户组的权限。当一个用户加入多个用户组时，该用户将同时拥有多个用户组的权限。

角色

角色是连接用户与权限的桥梁，一个角色对应一组权限，不同角色具有不同的权限。向用户授予某角色，即授予该角色所包含的所有权限。全局管理中有两种角色：

- 预定义角色：由系统创建，用户只能使用不能修改，每个子模块都有一个管理员 Admin 角色。
- 自定义角色：用户自主创建、更新和删除，自定义角色中的权限由用户自己维护。同时因为全局管理汇聚了多个子模块，各子模块也拥有相应的管理员角色，例如：
  - IAM Admin：管理用户与访问控制，即管理用户/用户组以及授权
  - Workspace Admin：管理层级及工作空间的权限，仅此权限可以创建层级
  - Audit Admin：管理审计日志

权限

权限指是否允许用户对某种资源执行某种操作。 为了降低使用门槛，DCE 采用 RBAC 模型将权限聚合成一个个角色，管理员只需要将角色授权给用户，该用户就一次性得到了该角色下聚合的一组权限。

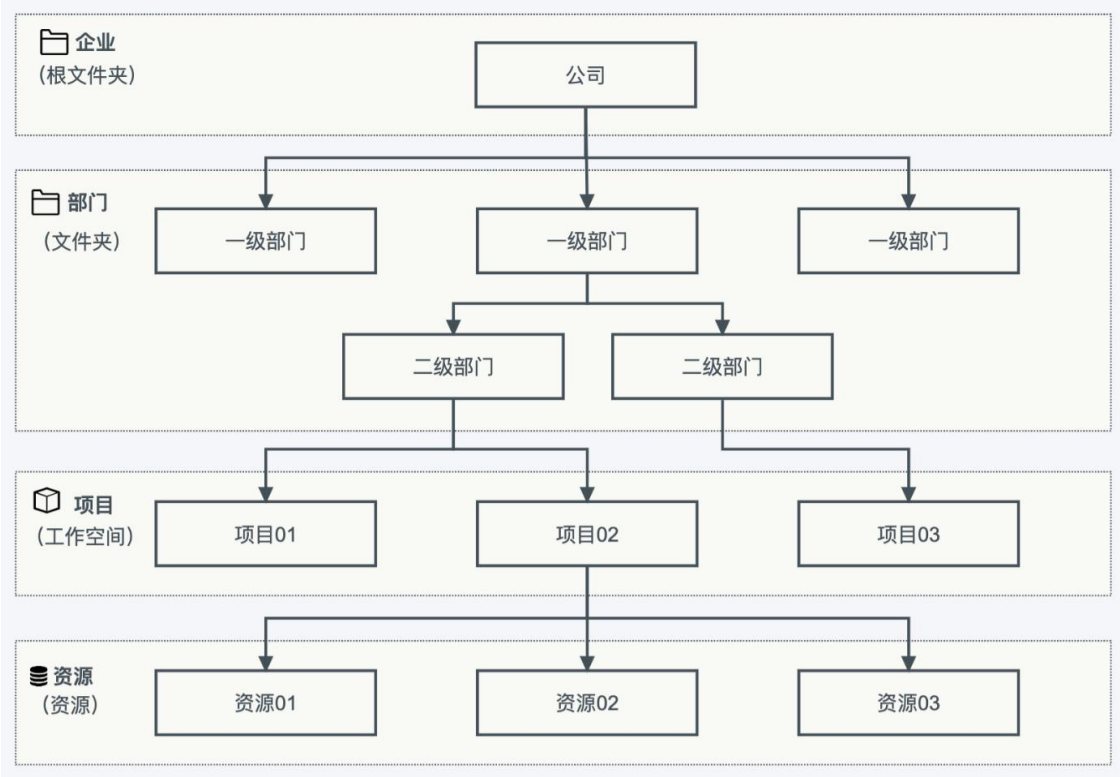
默认情况下，管理员创建的 IAM 用户没有任何角色权限，需要对其单独授予角色或将其加入用户组并给用户组授予角色，才能使得用户获得对应的角色权限，这一过程称为授权。 授权后，用户就可以基于被授予的角色权限对平台资源进行操作。

授权

授权指将用户完成具体工作所需的权限授予用户，授权通过系统角色或自定义角色的权限生效。 用户获得具体的权限后，可以对资源或服务进行操作。

工作空间

通过工作空间协调全局管理和子模块的权限关系，解决资源聚合和映射层级关系。通常一个工作空间对应一个项目，可以为每个工作空间分配不同的资源，指派不同的用户和用户组。



工作空间

层级

参见上图，为了满足企业内各个部门的分支划分，DCE 引入了层级的概念，通常层级对应着不同的部门，每个层级可以包含一个或多个工作空间。

资源

资源（Resource）泛指 DCE 平台上通过各个子模块创建的资源，是完成授权的具体数据。 通常资源描述一个或多个操作对象，每个子模块拥有其各自的资源和对应的资源定义详情，如集群、Namesapce、网关等。

资源的拥有者是主账号 Super Admin。Super Admin 具有在各子模块创建/管理/删除资源的权限，普通用户在没有授权的情况下，不会自动拥有资源的访问权限，需要 Super Admin 进行授权。 工作空间支持跨子模块授权用户（用户组）对于资源的访问权限。

身份凭证

身份凭证是识别用户身份的依据，登录和使用资源时，需要有身份凭证才能通过系统的鉴权认证。 身份凭证包括密码和访问密钥，可以通过 IAM 管理自己以及下属 IAM 用户的身份凭证。

有关更多术语，请参阅本站收录的云原生词汇大全。

什么是全局管理？

全局管理是以用户为中心的综合性服务板块，包含用户与访问控制、企业空间、审计日志、平台设置等基础服务模块。

- 用户与访问控制：帮助用户安全管理资源的访问权限。您可以通过用户与访问控制创建、管理、删除用户/用户组，并灵活配置用户/用户组权限，来完成用户职能权限的划分。
- 企业空间：具有层级结构和访问权限控制的资源隔离单元。您可以按照企业开发环境、部门结构等设置层级结构，并控制哪些人对哪些资源具有访问权限。
- 审计日志：提供资源的操作记录。通过操作记录您可以快速实现安全分析、资源变更、问题定位等。
- 平台设置：通过平台安全策略、邮件服务器、外观定制等，实现用户信息的安全性和平台的个性化。

全局管理优势

- 综合性汇总式服务

  将平台的基础服务模块汇聚在一起，减少菜单切换和功能分散带来的困扰，使平台管理员在一个菜单下就能够完成基础性平台设置和用户管理。

- 用户与租户相结合

  通过扁平化用户、用户组管理和层级式租户（工作空间）协作，多维度多场景实现基于平台的访问控制以及基于资源的权限划分。

- 个性化定制

支持个性化定制平台外观，包括登录页定制和顶部导航栏定制，通过可视化页面轻轻松松配置一个属于您的个性化平台。

- 简化操作，上手即用

  预先为用户完成了绝大多数的平台设置，包括密码规则配置、会话超时策略等，简化用户操作，实现上手即用。

操作步骤

1. 使用 DCE 平台管理员（Admin）或具有管理员权限的用户登录 DCE 平台
2. 进入用户与访问控制，创建用户并授权，创建用户组并授权，创建身份提供商
3. 进入工作空间，创建层级（企业层级关系），创建工作空间（租户）
4. 进入审计日志，查看并导出审计日志
5. 进入平台设置，设置安全策略、邮件服务器、外观定制、正版授权

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

功能总览

本页说明全局管理的各项功能特性。

1. 用户管理

   拥有用户帐号，是用户访问 DCE 平台的前提。 用户由平台管理员 Super Admin 或者用户与访问控制管理员 IAM Admin 在全局管理 -> 用户与访问控制 -> 用户页面创建，或者通过 LDAP 对接而来。 每位用户拥有独立的用户名和密码，通过给单个或者一组用户授予不同的权限，让不同的用户拥有不同资源的访问权限。

   ```
   graph LR

   admin([平台管理员或<br>用户与访问控制管理员]) --> |创建并管理用户|user[用户]
   user --> user1[用户 A]
   user --> user2[用户 B]
   user --> user3[用户 C]

   click user "https://docs.daocloud.io/ghippo/user-guide/access-control/user/"

   classDef plain fill:#ddd,stroke:#fff,stroke-width:0px,color:#000;
   classDef k8s fill:#326ce5,stroke:#fff,stroke-width:0px,color:#fff;
   classDef cluster fill:#fff,stroke:#bbb,stroke-width:2px,color:#326ce5;
   class admin plain;
   class user1,user2,user3 k8s;
   class user cluster
   ```

2. 用户组管理

用户组是多个用户的集合。 用户可以通过加入用户组，实现继承用户组的角色权限。通过用户组批量地给用户进行授权，可以更好地管理用户及其权限。

```
graph LR

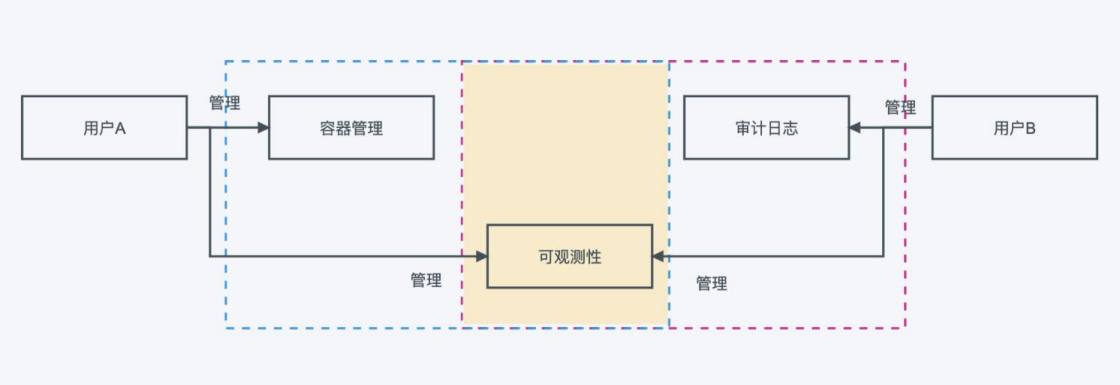
    admin([平台管理员或<br>用户与访问控制管理员])
    admin --> user[创建用户]
    admin --> group[创建用户组]
    admin --> add[将用户加入用户组]

click user "https://docs.daocloud.io/ghippo/user-guide/access-control/user/"
click group "https://docs.daocloud.io/ghippo/user-guide/access-control/group/"
click add "https://docs.daocloud.io/ghippo/user-guide/access-control/group/#_5"

classDef plain fill:#ddd,stroke:#fff,stroke-width:0px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:0px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:2px,color:#326ce5;
class admin plain;
class user,group,add cluster
```

3. 角色管理

一个角色对应一组权限。 权限决定了可以对资源执行的操作。向用户授予某个角色，即授予该角色所包含的所有权限。 您可以将不同模块的管理权限划分给不同的用户， 比如用户 A 管理容器管理模块，用户 B 管理应用工作台模块，共同管理可观测性模块。



角色

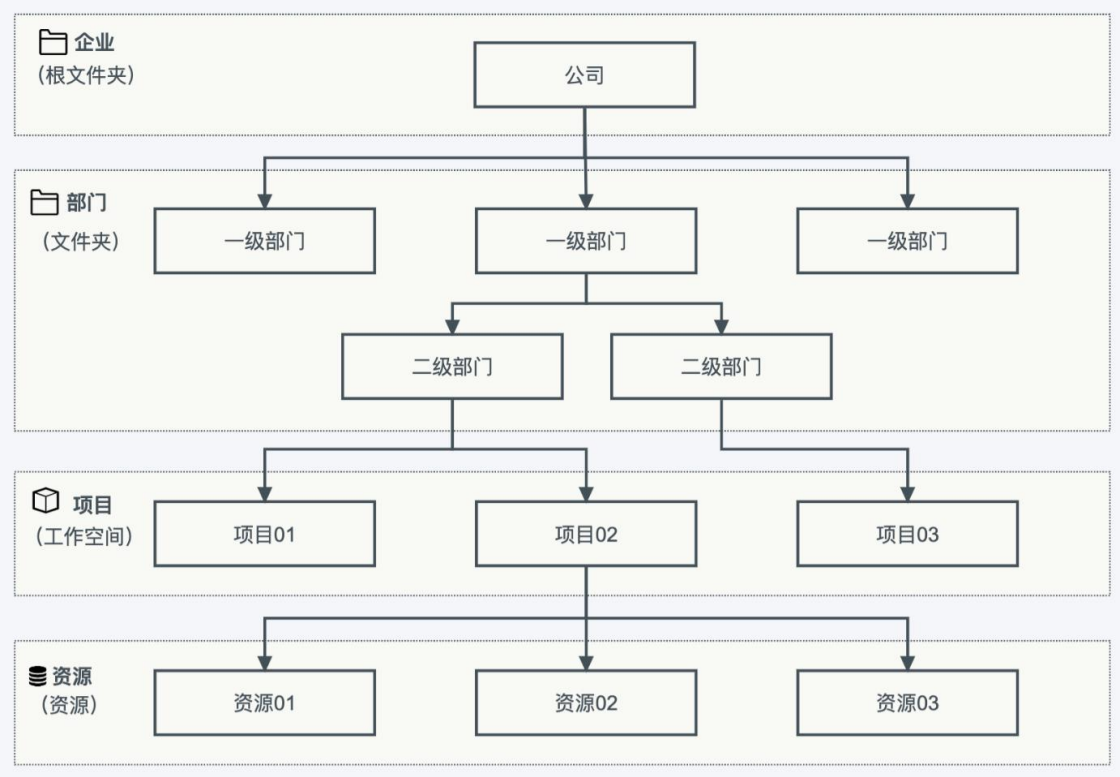
4. 工作空间

   工作空间用于管理资源，包含层级和工作空间两部分。

   层级是资源层次结构中的节点，一个层级可以包含工作空间、其他层级或两者的组合。 可以将层级理解为有层次结构的部门、环境、供应商等多种概念。

   工作空间可以理解为部门下的项目，管理员通过层级和工作空间映射企业中的层级关系。

   虽然一个层级可以包含多个层级或工作空间，但是给定的层级或者工作空间只能有一个父级。



工作空间

5. 审计日志

审计日志完整地记录用户的各项操作行为， 包括用户通过页面或 API 接口发起的操作以及各服务内部自触发的操作。 支持通过事件来源、资源类型、操作状态等多个维度进行组合查询，支持审计日志导出。

6. 平台设置

平台设置包括账号安全设置、 外观定制、 邮件服务器等。 当需要对账号的安全信息、平台 logo、许可证授权、邮件服务器等平台级设置进行管理时， 可以通过平台设置进行操作，平台设置仅平台管理员具有管理权限。

```
graph LR

    admin([平台管理员]) --> |管理|about[平台设置]
    about --> password[用户密码等安全策略]
```

```
    about --> appear[平台外观定制]
    about --> mail[邮件服务器]
    about --> license[正版授权]

    click about "https://docs.daocloud.io/ghippo/user-guide/platform-setting/about/"
    click password "https://docs.daocloud.io/ghippo/user-guide/password/"
    click appear "https://docs.daocloud.io/ghippo/user-guide/platform-setting/appearance/"
    click mail "https://docs.daocloud.io/ghippo/user-guide/platform-setting/mail-server/"
    click license "https://docs.daocloud.io/dce/license0/"

    classDef plain fill:#ddd,stroke:#fff,stroke-width:0px,color:#000;
    classDef k8s fill:#326ce5,stroke:#fff,stroke-width:0px,color:#fff;
    classDef cluster fill:#fff,stroke:#bbb,stroke-width:2px,color:#326ce5;
    class admin plain;
    class about,password,appear,mail,license cluster
```

功能清单

具体的功能清单如下所述。

1. 用户与访问控制
    1. 用户
        - 列表展示用户名、描述、创建时间、最近一次登录时间
        - 列表支持通过用户名搜索用户
        - 列表支持给用户快捷授权
        - 列表支持将用户快捷加入用户组
        - 列表支持批量删除用户
        - 列表支持创建用户

- 详情支持编辑用户基本信息，包括邮箱、描述、启用 / 禁用等
- 详情支持记录用户授权信息，支持添加 / 移除权限
- 详情支持记录用户加入用户组信息，支持用户加入新的用户组，支持将用户从老的用户组移除
- 支持管理员帮助用户修改密码
- 支持管理员帮助用户创建访问密钥

2. 用户组
    - 列表展示用户组名、组内用户数、描述、创建时间
    - 列表支持通过用户组名搜索
    - 列表支持给用户组快捷授权
    - 列表支持给用户组快捷添加用户
    - 列表支持批量删除用户组
    - 列表支持创建用户组
    - 详情支持编辑用户组基本信息，例如描述
    - 详情支持记录用户组授权信息，支持添加 / 移除权限
    - 详情支持记录用户组成员信息，支持给用户组添加新成员，支持将成员从用户组移除
3. 角色
    - 列表展示系统角色名称、描述
    - 详情记录角色授权信息，支持将角色授予用户 / 用户组，支持将用户 / 用户组从该角色移除
    - 支持 Folder Admin、Folder Editor、Folder Viewer 三种预定义文件夹角色
    - 支持 Workspace Admin、Workspace Editor、Workspace Viewer 三种预定义工作空间角色
4. 身份提供商

- 支持 LDAP、OIDC 两种协议对接外部用户
- LDAP 协议支持手动 / 自动同步外部用户
- LDAP 协议支持手动同步外部用户组
- OIDC 协议支持手动同步外部用户

2. 工作空间与层级
    1. 文件夹
        - 支持树状结构展示文件夹和工作空间
        - 支持通过文件夹名称和工作空间名称搜索
        - 支持 5 级文件夹映射企业层级
        - 支持为文件夹添加用户 / 用户组并授权
        - 支持权限继承，子文件夹、工作空间能够继承用户 / 用户组在上级文件夹权限
        - 支持移动文件夹。映射企业中的部门变动，移动后其下的子文件夹、工作空间及其资源 / 成员均会跟随该文件夹移动，权限的继承关系重新发生变化
    2. 工作空间
        - 支持为工作空间添加用户 / 用户组并授权
        - 支持添加资源到工作空间 – 资源组，支持 6 种资源类型
        - 支持权限继承，资源组中的资源能够继承用户 / 用户组在工作空间和上级文件夹的角色
        - 支持移动工作空间。映射企业中的项目变动，移动后其下的资源 / 成员均会跟随该工作空间移动，权限的继承关系重新发生变化（开发中）
        - 支持添加资源到工作空间 – 共享资源，将一个集群资源共享给多个工作空间使用
        - 支持针对每个共享的集群资源进行资源限额

- 支持通过 CPU Limit、CPU Request 、内存 Limit、内存 Request、存储请求总量 Request Storage、存储卷声明 PersistentVolumeClaim 六个维度进行资源限额

3. 审计日志
   - 列表展示事件名称、资源类型、资源名称、状态、操作人、操作时间
   - 列表支持查看审计日志详情
   - 列表支持展示最近一天或自定义时间内的审计日志
   - 列表支持通过状态、操作人、资源类型、资源名称搜索审计日志
   - 列表支持 .csv 格式导出审计日志
   - 默认采集全局管理的全部审计日志
   - 默认保存 365 天内的审计日志
   - 支持手动 / 自动两种方式清理全局管理的审计日志
   - 支持开启 / 关闭收集 K8s 的审计日志
   - 支持手动 / 自动两种方式清理 K8s 的审计日志
4. 平台设置
   1. 安全策略
      - 支持自定义密码规则
      - 支持自定义密码策略
      - 支持自定义会话超时策略，超出时长自动退出当前账号
      - 支持自定义账号锁定策略，限制时间内多次登录失败，账号将被锁定
      - 支持登录登出策略，开启后关闭浏览器的同时退出登录
   2. 邮件服务器设置：支持管理员配置邮件服务器，支持通过邮件方式找回用户密码，接收告警通知等
   3. 外观定制
      - 支持自定义登录页，包括更换平台 LOGO、登录页图标、标签页图标等

- 支持一键还原登录页外观配置
- 支持自定义顶部导航栏，包括导航栏图标、标签页图标等
- 支持一键还原顶部导航栏外观配置

4. 正版授权
    - 列表展示许可证名称、所属模块、许可证级别、许可证状态过期时间
    - 支持管理许可证，确保子模块在有效期内
5. 关于平台
    - 支持展示模块版本
    - 支持展示平台使用的开源软件
    - 支持展示技术团队风采

5. 个人中心
    1. 安全设置
        - 支持用户更新登录密码
        - 支持通过邮箱找回登录密码
    2. 访问密钥：支持每个用户创建独立的 API 密钥，支持进行 API 密钥过期设置，以确保系统安全
    3. 语言设置：支持多语言，支持简体中文、英文、自动检测您的浏览器首选语言三种方式确定语言

密码重置

如果您忘记密码，可以按本页面说明重置密码。

重置密码步骤

管理员最初创建一个用户时，会为其设置用户名和密码。 该用户登录后，在个人中心填写邮箱并修改密码。 若该用户未设置邮箱，则只能联系管理员进行密码重置。

1. 如果用户忘记了密码，可以在登录界面点击忘记密码。



登录界面

2. 输入用户名，点击提交。

   为防止滥发邮件，点击提交后，该按钮将变灰并显示一个为期 1 分钟的倒计时。 如果 1 分钟后还未收到邮件，请再次点击此按钮。



密码重置流程

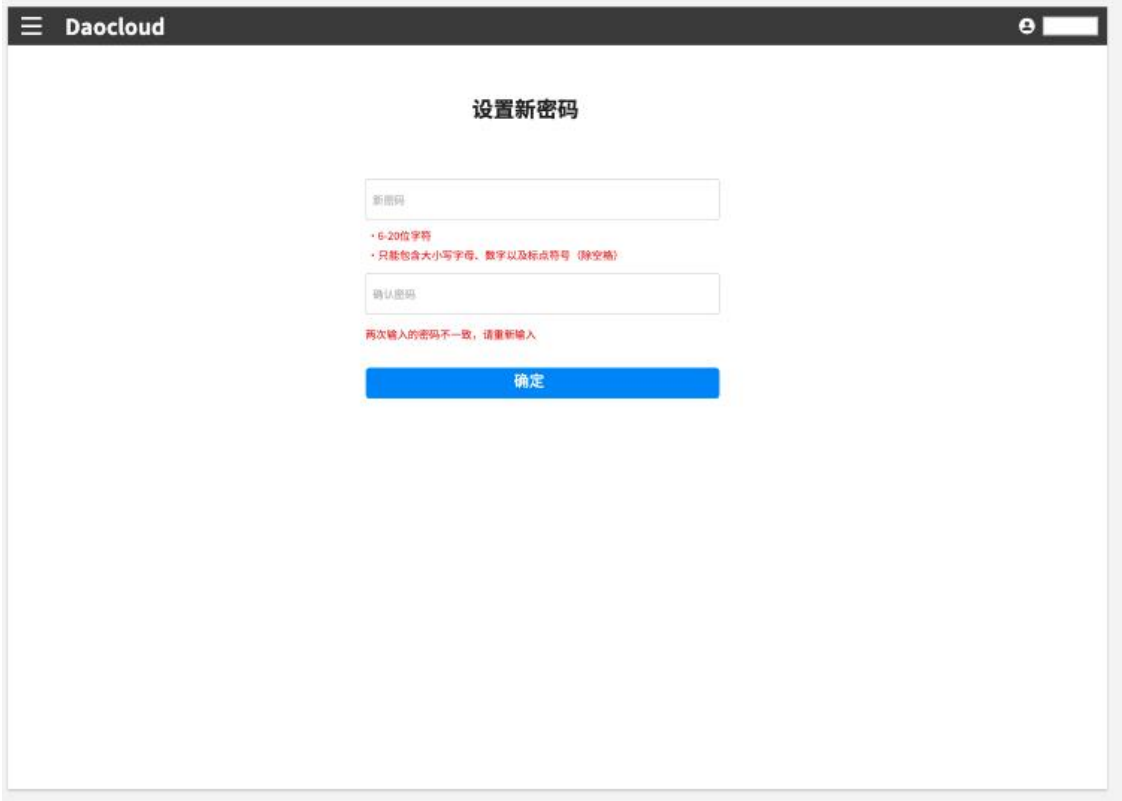
3. 在邮箱中找到密码重置邮件，点击密码重置按钮。

邮件名称：DaoCloud Enterprise-密码重置



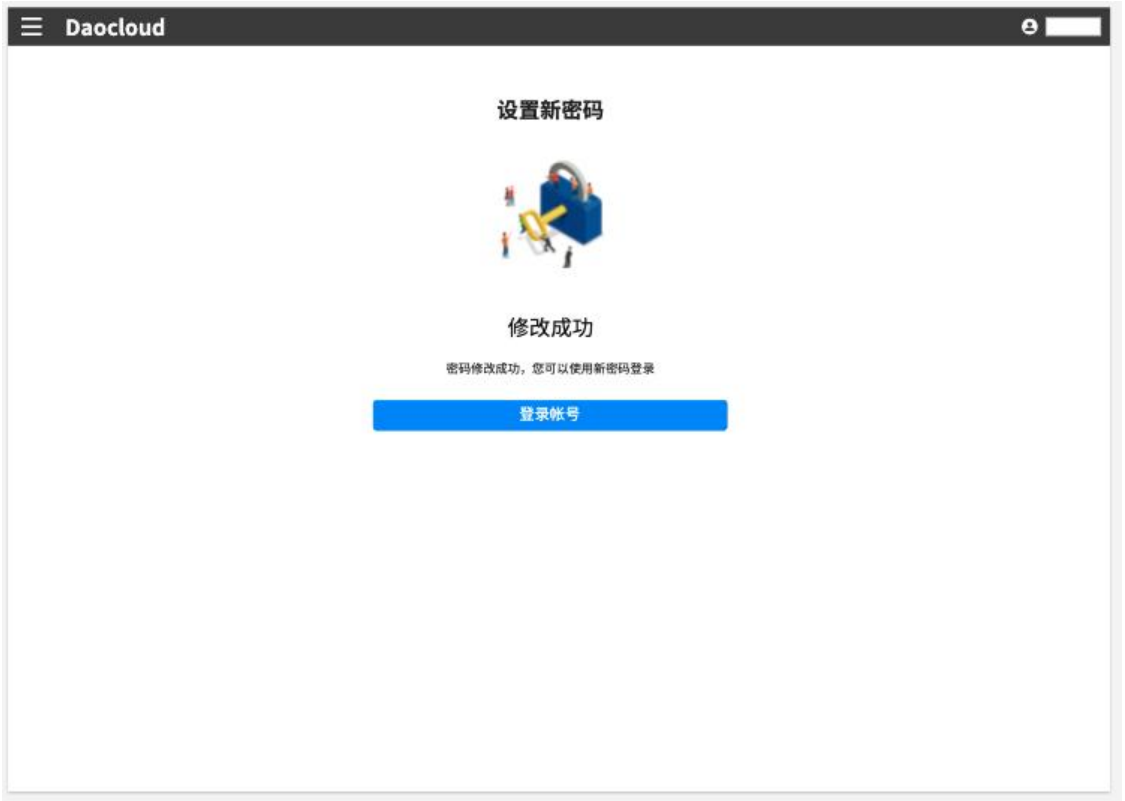
密码重置流程

4. 按照屏幕提示，设置新密码，点击确定。设置新密码的要求与创建用户时的密码规则一致。



密码重置流程

5. 屏幕提示密码修改成功。



密码重置流程

重置密码流程

整个密码重置的流程示意图如下。

```
graph TB

pass[忘记密码] --> usern[输入用户名]
--> button[点击发送验证邮件的按钮] --> judge1[判断用户名是否正确]

    judge1 -.正确.-> judge2[判断是否绑定邮箱]
    judge1 -.错误.-> tip1[提示用户名不正确]

        judge2 -.已绑定邮箱.-> send[发送重置邮件]
        judge2 -.未绑定邮箱.-> tip2[提示未绑定邮箱<br>联系管理员重置密码]
```

```
send --> click[点击邮件中的链接] --> reset[重置密码]
--> success[成功重置密码]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class pass,usern,button,tip1,send,tip2,send,click,reset,success plain;
class judge1,judge2 k8s
```

语言设置

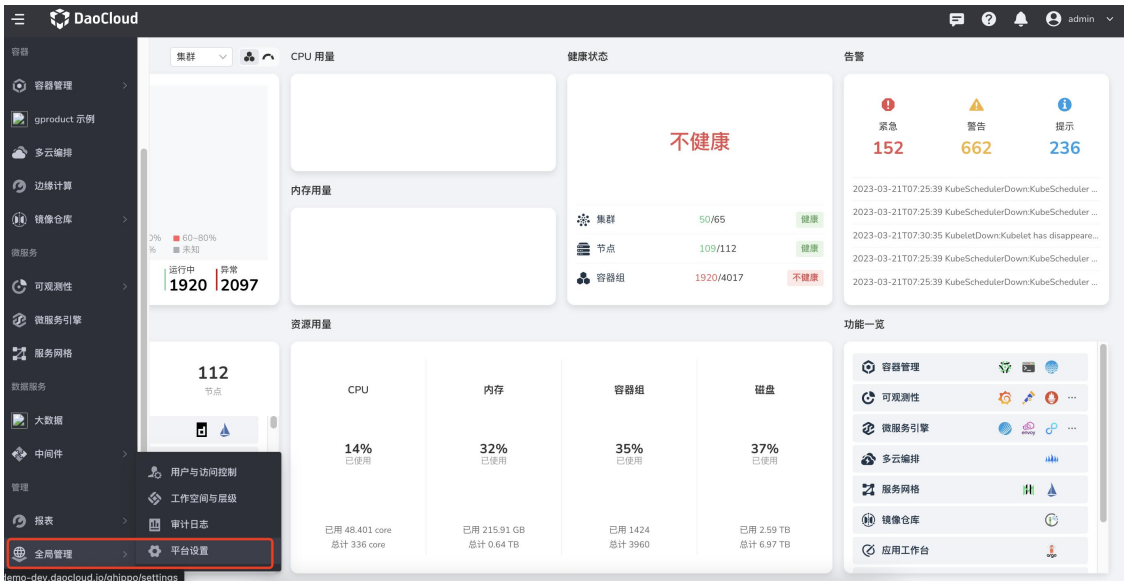
本节说明如何设置界面语言。目前支持中文、English 两个语言。

语言设置是平台提供多语言服务的入口，平台默认显示为中文，用户可根据需要选择英语或自动检测浏览器语言首选项的方式来切换平台语言。 每个用户的多语言服务是相互独立的，切换后不会影响其他用户。

平台提供三种切换语言方式：中文、英语–English、自动检测您的浏览器语言首选项。

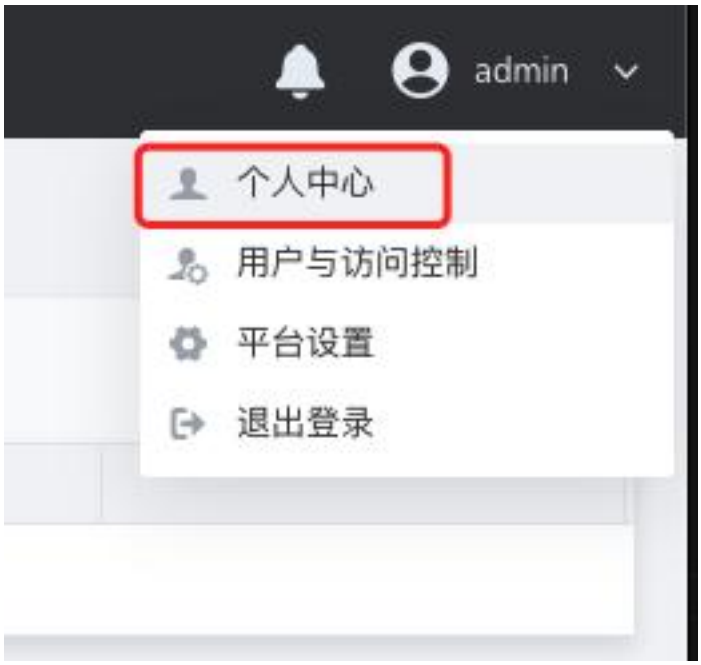
操作步骤如下。

1. 使用您的用户名/密码登录 DCE 5.0。点击左侧导航栏底部的全局管理。



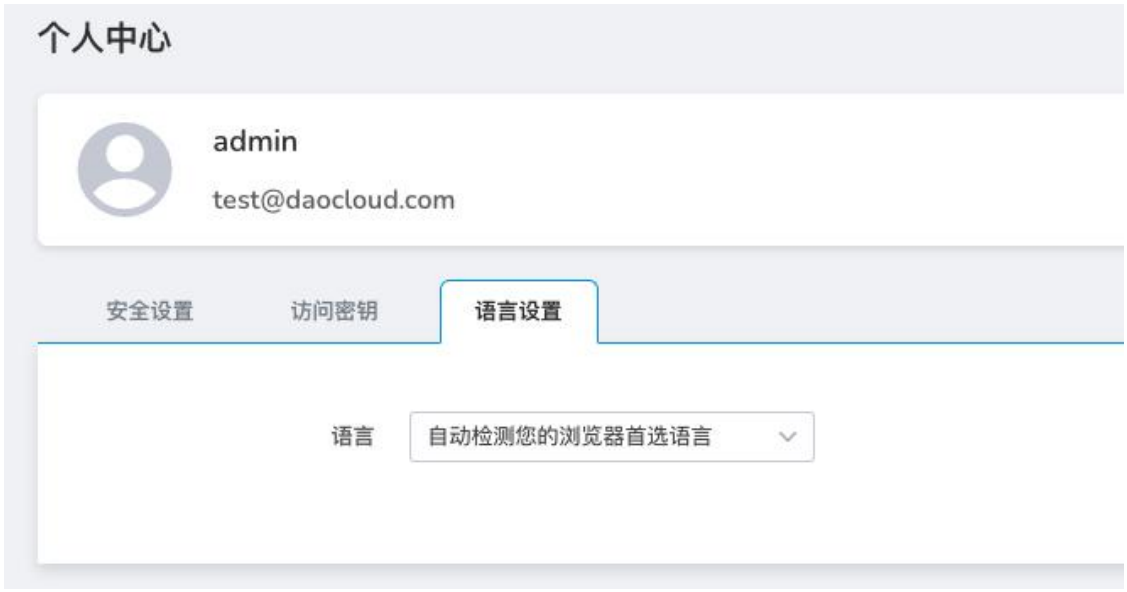
全局管理

2. 点击右上角的用户名位置，选择个人中心。



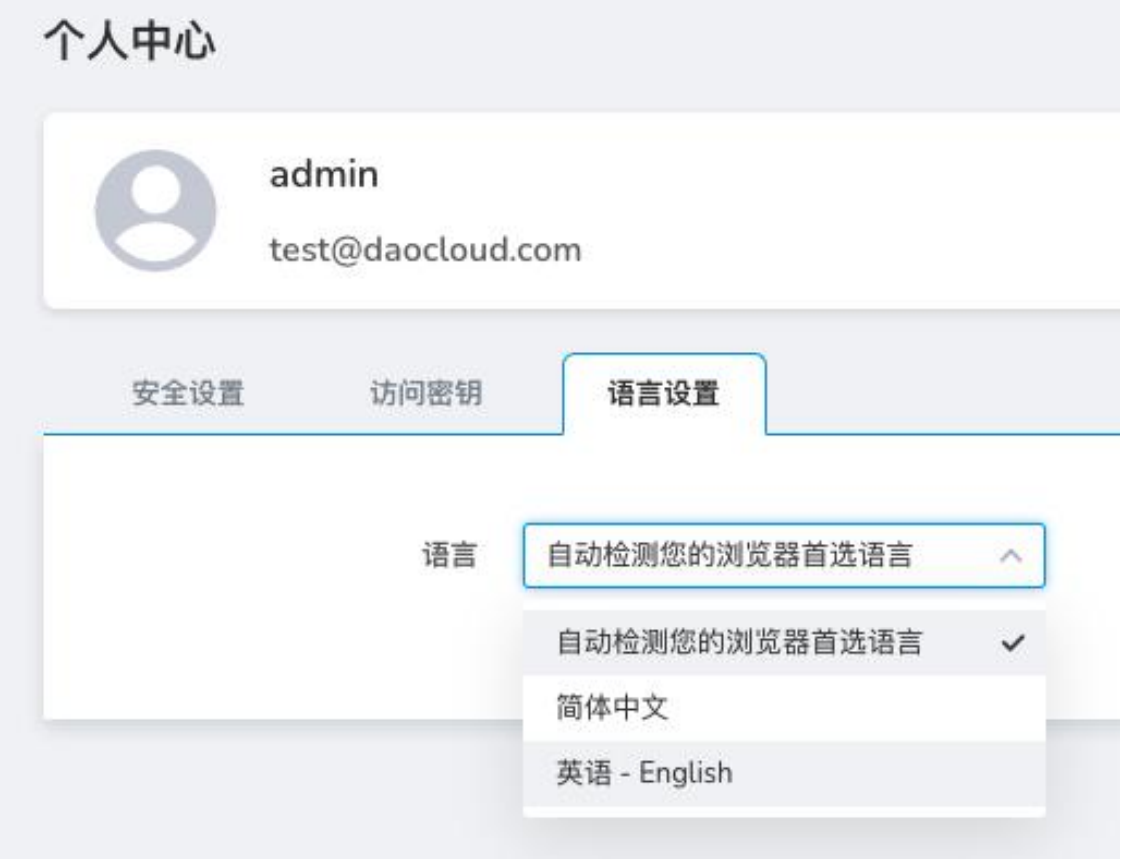
个人中心

3. 点击语言设置页签。



语言设置

4. 切换语言选项。


个人中心
admin
test@daocloud.com
安全设置
访问密钥
语言设置
语言
自动检测您的浏览器首选语言
自动检测您的浏览器首选语言
简体中文
英语 - English

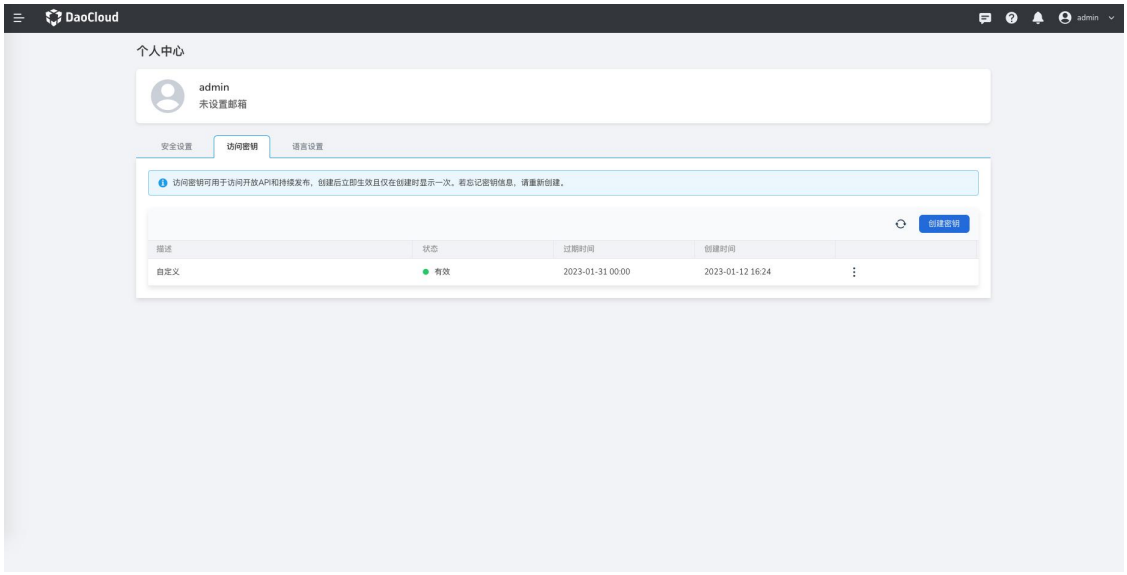
切换语言

访问密钥

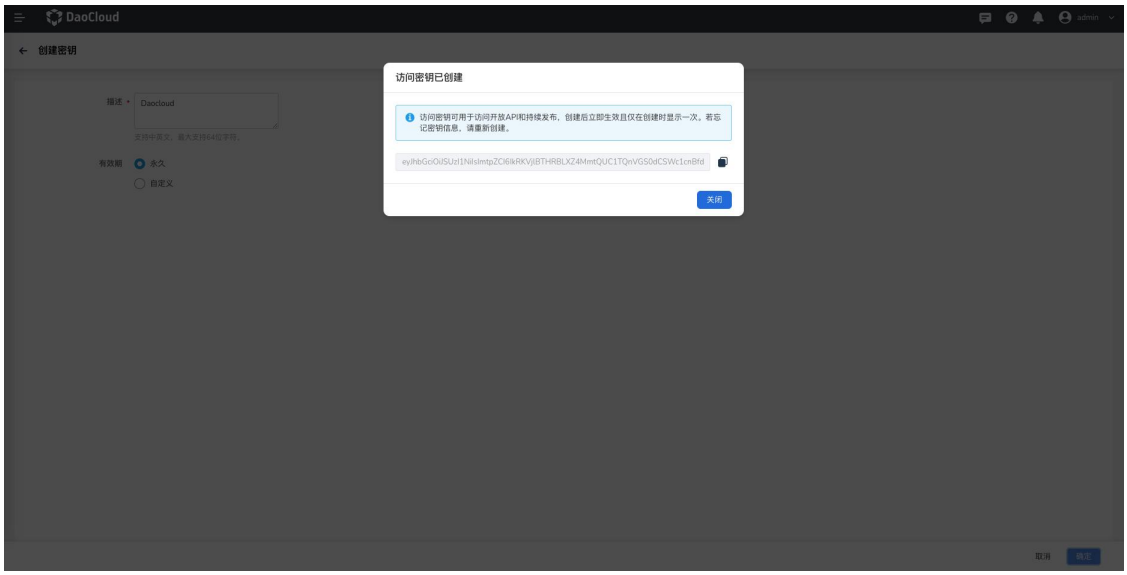
访问密钥可用于访问开放 API 和持续发布，用户可在个人中心参照以下步骤获取密钥并访问 API。

获取密钥

登录 DCE 5.0，在右上角的下拉菜单中找到个人中心，可以在访问密钥页面管理账号的访问密钥。



img



img

访问密钥信息仅显示一次。如果您忘记了访问密钥信息，您需要重新创建新的访问密钥。

使用密钥访问 API

在访问 DCE 5.0 openAPI 时，在请求中加上请求头 Authorization:Bearer ${token} 以标识访问者的身份， 其中 ${token} 是上一步中获取到的密钥，具体接口信息参见 OpenAPI 接口文档。

请求示例

curl -X GET -H 'Authorization:Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IkRKVjlBTHRBLXZ4MmtQUC1TQnVGS0dCSWc1cnBfdkxiQVVqM2U3RVByWnMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE0MTU5NjksImlhdCI6MTY2MDgxMTE2OSwiaXNzIjoiZ2hpcHBvLmlvIiwic3ViIjoiZjdjOGIxZjUtMTc2MS00NjYwLTg2MWQtOWI3MmI0MzJmNGViIiwicHJlZmVycmVkX3VzZXJuYW1lIjoiYWRtaW4iLCJncm91cHMiOltdfQ.RsUcrAYkQQ7C6BxMOrdD3qbBRUt0VVxynIGeq4wyIgye6R8Ma4cjxG5CbU1WyiHKpvIKJDJbeFQHro2euQyVde3ygA672ozkwLTnx3Tu-_mB1BubvWCBsDdUjIhCQfT39rk6EQozMjb-1X1sbLwzkfzKMls-oxkjagI_RFrYITVPwT3Oaw-qOyuIRSw7Dxd7jb0vINPq84vmIQIsI3UuTZSNO5BCgHpubcWwBss-Aon_DmYA-Et_-QtmPBA3k8E2hzDSzc7eqK0I68P25r9rwQ3DeK

wD1dbRyndqWORRnz8TLEXSiCFXdZT2oiMrcJtO188Ph4eLGut1-4PzKhwgrQ' https://demo-dev.daocloud.io/apis/ghippo.io/v1alpha1/users?page=1&pageSize=10 -k

请求结果


```
{
    "items": [
        {
            "id": "a7cfd010-ebbe-4601-987f-d098d9ef766e",
            "name": "a",
            "email": "",
            "description": "",
            "firstname": "",
            "lastname": "",
            "source": "locale",
            "enabled": true,
            "createdAt": "1660632794800",
            "updatedAt": "0",
            "lastLoginAt": ""
        }
    ],
    "pagination": {
        "page": 1,
        "pageSize": 10,
        "total": 1
    }
}
```

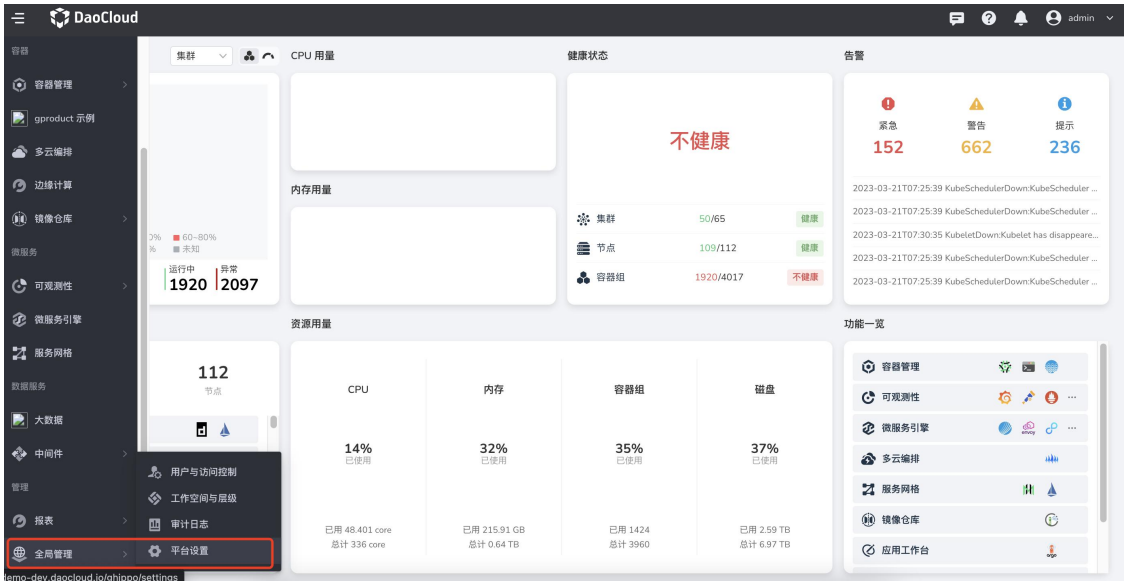
安全设置

功能说明：用于填写邮箱地址和修改登录密码。

- 邮箱：当管理员配置邮箱服务器地址之后，用户能够通过登录页的忘记密码按钮，填写该处的邮箱地址以找回密码。
- 密码：用于登录平台的密码，建议定期修改密码。

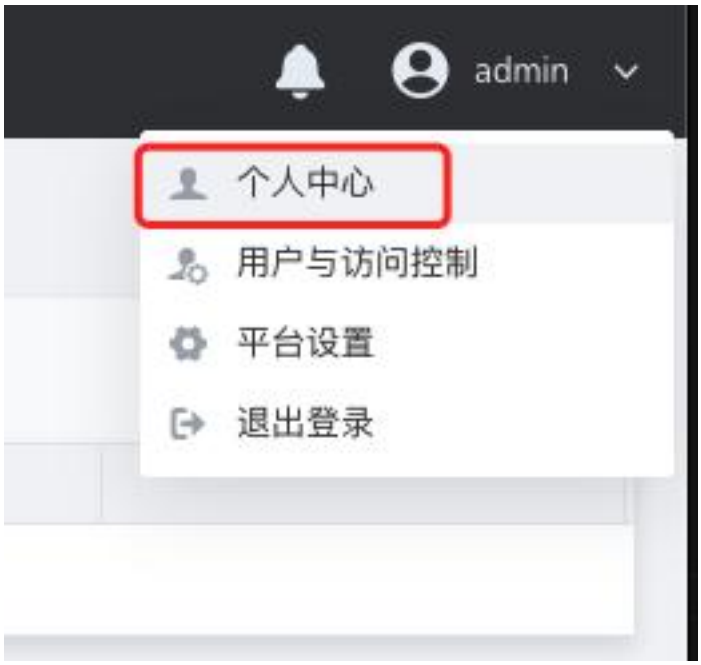
具体操作步骤如下：

1. 使用您的用户名/密码登录 DCE 5.0。点击左侧导航栏底部的全局管理。



全局管理

2. 点击右上角的用户名位置，选择个人中心。



个人中心

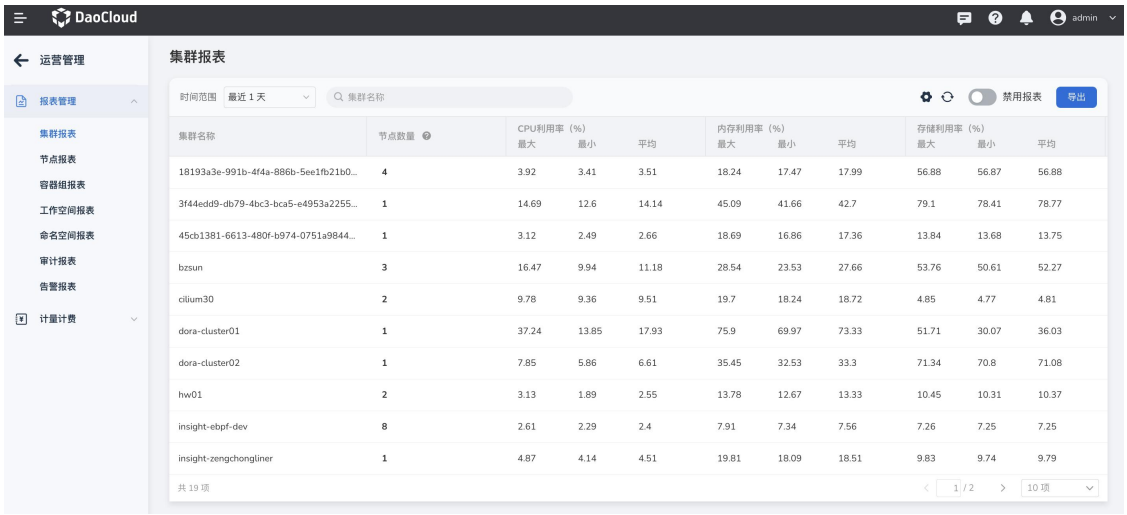
3. 点击安全设置页签。填写您的邮箱地址或修改登录密码。



安全设置

报表管理

报表管理模块通过整合管理员的作业执行情况，结合集群的硬件资源配置，分别从集群、节点、Pod、工作空间、命名空间维度进行统计， 将计算资源的利用率直观地呈现给用户，帮助用户更好地进行计算资源的划分和调度。



报表

启用某种报表后将在 20 分钟内开始采集并展示集群报表数据。 禁用后，将在 20 分钟内停止采集报表的数据。往期已采集到的数据还将正常显示。

目前提供以下几种报表：

- 集群报表：显示所有集群名称、包含的节点数量（点击数字可跳转到节点报表）和资源利用率。 可以按集群名称进行搜索。
- 节点报表：显示节点 IP、类型、所属集群以及资源利用率。可以自定义报表的时间范围， 并按节点名称、所属集群、IP 地址进行搜索。
- Pod 报表：显示 Pod 的名称、命名空间、集群、工作空间和资源利用率。可以自定义报表的时间范围， 并按 Pod 名称、命名空间、集群和工作空间进行搜索。

- 工作空间报表：显示工作空间名称、包含的命名空间数量（点击数字可跳转到命名空间报表）、容器组数量 （点击数字可跳转到 Pod 报表）和资源利用率。可以自定义报表的时间范围，并按工作空间进行搜索。
- 命名空间报表：显示命名空间名称、Pod 数量（点击数字可跳转到 Pod 报表）、所属集群、所属工作空间和资源利用率。 可以自定义报表的时间范围，并按命名空间、集群和工作空间进行搜索。
- 审计报表：统计用户操作以及针对资源执行过的操作，方便管理员控制资源使用情况，跟踪用户操作痕迹。 可以按用户名搜索用户的操作，可以按资源类型/事件名称搜索针对资源的操作。
- 告警报表：适用于生成集群巡检报告。可以自定义报表的时间范围，并按节点名称、所属集群进行搜索。

这几种报表的统计信息如下：

报表类型	显示内容	可搜索项	可导出/下载	启用/禁用	自定义时间范围	齿轮图标设置
集群报表	集群名称、节点数量、资源利用率	集群名称	是	是	是	是
节点报表	节点 IP、类型、所属集群、资源利用率	节点名称、所属集群、IP 地址	是	是	是	是
Pod	Pod 名称、命名空间、集群、工作空间、资源利用	Pod 名称、命名空间、	是	是	是	是

报表类型	显示内容	可搜索项	可导出/下载	启用/禁用	自定义时间范围	齿轮图标设置
报表	率	集群、工作空间				
工作空间报表	工作空间名称、命名空间数量、容器组数量、资源利用率	工作空间	是	是	是	是
命名空间报表	命名空间名称、Pod 数量、所属集群、所属工作空间、资源利用率	命名空间、集群、工作空间	是	是	是	是
审计报表	用户操作和针对资源的操作统计	用户名、资源类型/事件名称、工作空间、命名空间、集群	是	是	是	否
告警报表	集群巡检报告	节点名称、所属集群	是	是	是	否

从上表可以看出，每种报表都有其特定的显示内容和可搜索项。此外，每种报表都可以导出或下载以供管理员查看。 所有报表的启用和禁用都是可控的，并且可以自定义时间范围。除了审计报表和告警报表之外， 其他报表都提供了齿轮图标设置功能，以便管理员根据实际需求配置要显示的列。

离线升级运营管理模块

本页说明从下载中心下载运营管理模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `gmagpie` 字样是运营管理模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

    !!! note

    该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

    === “已安装 chart repo”

    若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

    ```yaml title="load-image.yaml"
    source:
      intermediateBundlesPath: gmagpie-offline # (1)
    target:
      containerRegistry: 10.16.10.111 # (2)
      containerRepository: release.daocloud.io/gmagpie # (3)
      repo:
        kind: HARBOR # (4)
    ```

```
    url: http://10.16.10.111/chartrepo/release.daocloud.io # (5)
    auth:
    username: "admin" # (6)
    password: "Harbor12345" # (7)
  containers:
    auth:
      username: "admin" # (8)
      password: "Harbor12345" # (9)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

“未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: gmagpie-offline # (1)
target:
  containerRegistry: 10.16.10.111 # (2)
  containerRepository: release.daocloud.io/gmagpie # (3)
  repo:
```

```
    kind: LOCAL
    path: ./local-repo  # (4)
  containers:
    auth:
      username: "admin"  # (5)
      password: "Harbor12345"  # (6)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar xvf gmagpie.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```shell
docker load -i images.tar
```

=== “containerd”

```shell
ctr image import images.tar
```

!!! note

每个节点都需要执行 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

!!! note

当从 v0.1.x（或更低版本）升级到 v0.2.0（或更高版本）时，需要修改数据库连接参数。

数据库连接参数的修改示例：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
global:
  database:
    host: 127.0.0.1
    port: 3306
    dbname: gmagpie
    password: passowrd
```

```
    user: gmagpie
```

修改为：

```yaml title="bak.yaml"
USER-SUPPLIED VALUES:
global:
  storage:
    gmagpie:
    - driver: mysql
      accessType: readwrite
      dsn: {global.database.apiserver.user}:{global.database.apiserver.password}@tcp({global.database.host}:{global.database.port})/{global.database.apiserver.dbname}?charset=utf8mb4&multiStatements=true&parseTime=true
```

===“通过 helm repo 升级”

1. 检查运营管理 Helm 仓库是否存在。

    ```shell
    helm repo list | grep gmagpie
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加运营管理的 Helm 仓库。

   ```shell
   heml repo add gmagpie http://{harbor url}/chartrepo/{project}
   ```

1. 更新运营管理的 Helm 仓库。

   ```shell
   helm repo update gmagpie # (1)
   ```

   1. Helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

1. 选择您想安装的运营管理版本（建议安装最新版本）。

   ```shell
   helm search repo gmagpie/gmagpie --versions
   ```

   ```none
   [root@master ~]# helm search repo gmagpie/gmagpie --versions
   NAME                    CHART VERSION   APP VERSION   DESCRIPTION
   gmagpie/gmagpie   0.3.0           v0.3.0        A Helm chart for GHippo
   ...
   ```

1. 备份 `--set` 参数。

   在升级运营管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values gmagpie -n gmagpie-system -o yaml > bak.yaml
```

1. 更新 Gmagpie CRD。

```shell
helm pull gmagpie/gmagpie --version 0.3.0 && tar -zxf gmagpie-0.3.0.tgz
kubectl apply -f gmagpie/crds
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade gmagpie gmagpie/gmagpie \
  -n gmagpie-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.3.0
```

===“通过 Chart 包升级”

1. 备份 `--set` 参数。

在升级运营管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values gmagpie -n gmagpie-system -o yaml > bak.yaml
```

1. 更新 Gmagpie CRD。

```shell
kubectl apply -f ./crds
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade gmagpie . \
  -n gmagpie-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

计费计量

首先通过计费配置来实现 CPU、内存、存储、进出站流量的单价以及货币种类的设置。目前支持人民币和美元两种货币。



计费配置

目前支持针对集群、节点、Pod、命名空间和工作空间实现计费统计。 管理员启用某种计费后，计费都会在 20 分钟内开始采集数据并展示计费信息。 如果禁用某种计费，将在 20 分钟内停止数据采集，之前采集到的数据还可正常显示。

下表展示了每种计费的显示内容和可搜索项，同时提供了导出/下载数据的功能，并支持自定义时间范围和齿轮图标设置功能，以方便管理员根据需求进行列的定制。

计量计费	显示内容	可搜索项	可导出/下载	启用/禁用	自定义时间范围	齿轮图标设置
集群计费	集群名称、节点数量、资源计费情况	集群名称	是	是	是	是
节点计费	节点 IP、类型、所属集群、资源计费情况	节点名称、所属集群、IP 地址	是	是	是	是
Pod 计费	Pod 名称、命名空间、集群、工作空间、资源计费情况	Pod 名称、命名空间、集群、工作空间	是	是	是	是
工作空间计费	工作空间名称、命名空间数量、容器组数量、资源计费情况	工作空间	是	是	是	是
命名空间计费	命名空间名称、Pod 数量、所属集群、所属工作空间、资源计费情况	命名空间、集群、工作空间	是	是	是	是

运营管理

IT 运营管理是整个企业信息化建设的重要组成部分。 DCE 5.0 提供了报表管理和计费计量两个运营管理功能， 旨在帮助企业更好地了解其 IT 基础架构使用情况以及 IT 服务提供的价值。

报表管理

报表管理是 IT 运营管理的核心工作之一，通过对企业 IT 资源的采集、整合、分析并可视化展示， 为企业管理人员提供了决策的支持，从而实现对企业 IT 资产的有效控制。

首先，通过集中展示各种类型的报表，企业可以更准确地了解 IT 资源的利用率和使用情况。 例如，在集群报表中，管理员可以查看不同集群的节点数量和资源利用率等信息； 在节点报表中，管理员可以查看每个节点的 IP 地址、类型、所属集群以及资源利用率等信息。 通过这些报表，管理员可以更好地进行计算资源的划分和调度。

其次，通过审计报表，企业可以了解用户的操作以及针对资源的操作，方便企业掌握资源利用情况， 并跟踪用户操作痕迹。通过告警报表，企业可以生成集群巡检报告，从而快速诊断问题并进行处理。

最后，报表管理模块还支持自定义时间范围和搜索，帮助管理员更好地了解企业 IT 资源的使用情况。 此外，还提供了导出/下载数据的功能，让管理员可以对数据进行深入分析。

计费计量

另一方面，计费计量也是 IT 运营管理的重要组成部分之一。计费计量主要通过对企业 IT 资源的采集和分析，实现对企业 IT 服务价值的衡量和计费。

首先，计费计量模块可以帮助企业了解 IT 服务提供的真正价值，从而确定合理的收费标准。 例如，在云计算环境中，通过计算用户实际使用的资源量，企业可以根据实际使用量来收取费用。

其次，计费计量模块还可以帮助企业实现精细化计费，提高企业的盈利能力。 例如，在虚拟机场景下，管理员可以基于 CPU、内存、存储等指标来计算每个用户的使用费用， 并根据用户的实际使用情况进行差异化收费。

最后，计费计量模块还支持自定义计费规则和周期，并且可以生成详细的账单，方便企业对 IT 服务的费用进行跟踪与管理。

综上所述，IT 运营管理包括报表管理和计费计量两个方面，它们可以帮助企业更好地了解其 IT 基础架构使用情况以及 IT 服务提供的价值。通过报表管理和计费计量模块的实施， 企业可以更高效地运营 IT 环境，并在保持盈利的同时，提升 IT 服务质量和用户满意度。

文件夹权限说明

文件夹具有权限映射能力，能够将用户/用户组在本文件夹的权限映射到其下的子文件夹、工作空间以及资源上。

若用户/用户组在本文件夹是 Folder Admin 角色，映射到子文件夹仍为 Folder Admin 角色，映射到其下的工作空间则为 Workspace Admin； 若在工作空间与层级 -> 资源组中绑定了 Namespace，则映射后该用户/用户组同时还是 Namespace Admin。

!!! note

文件夹的权限映射能力不会作用到共享资源上，因为共享是将集群的使用权限共享给多个工作空间，而不是将管理权限受让给工作空间，因此不会实现权限继承和角色映射。

应用场景

文件夹具有层级能力，因此将文件夹对应于企业中的部门/供应商/项目等层级时，

- 若用户/用户组在一级部门具有管理权限（Admin），其下的二级、三级、四级部门或项目同样具有管理权限；
- 若用户/用户组在一级部门具有使用权限（Editor），其下的二级、三级、四级部门或项目同样具有使用权限；
- 若用户/用户组在一级部门具有只读权限（Viewer），其下的二级、三级、四级部门或项目同样具有只读权限。

对象	操作	Folder Admin	Folder Editor	Folder Viewer
对文件夹本身	查看	√	√	√
	授权	√	✗	✗
	修改别名	√	√	✗

对象	操作	Folder Admin	Folder Editor	Folder Viewer
对子文件夹	创建	√	✗	✗
	查看	√	√	√
	授权	√	✗	✗
	修改别名	√	√	✗
对其下的工作空间	创建	√	✗	✗
	查看	√	√	√
	授权	√	✗	✗
	修改别名	√	√	✗
对其下的工作空间 – 资源组	查看	√	√	√
	资源绑定	√	✗	✗
	解除绑定	√	✗	✗
对其下的工作空间 – 共享资源	查看	√	√	√
	新增共享	√	✗	✗
	解除共享	√	✗	✗
	资源限额	√	✗	✗

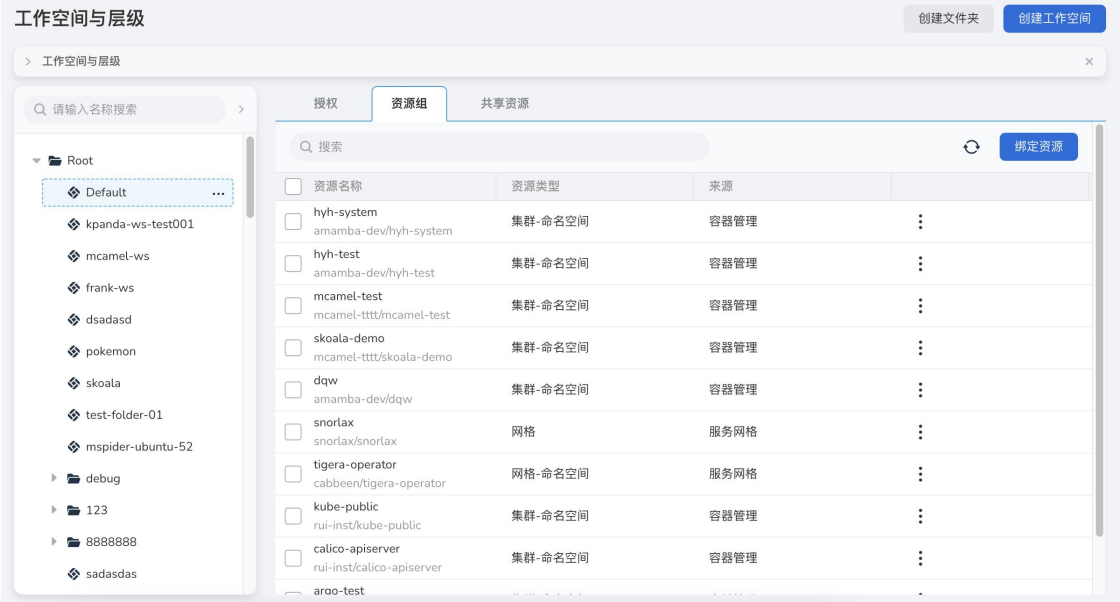
工作空间最佳实践

工作空间是一种资源分组单元，大多数资源都可以绑定到某一个工作空间中。 而工作空间通过授权和资源绑定，能够实现用户与角色的绑定关系，并一次性应用到工作空间的所有资源上。

通过工作空间，可以轻松管理团队与资源，解决跨模块、跨集群的资源授权问题。

工作空间的功能

工作空间包含三个功能：授权、资源组和共享资源。主要解决资源统一授权、资源分组及资源配额问题。



工作空间

1. 授权：为用户/用户组授予该工作空间的不同角色，并将角色应用到工作空间的资源上。

   最佳实践：普通用户想要使用应用工作台、微服务引擎、服务网格、中间件模块功能，或者需要拥有容器管理、服务网格中部分资源的使用权限时，需要管理员授予该工作空间的使用权限（Workspace Admin、Workspace

Edit、Workspace View）。 这里的管理员可以是 Admin 角色、该工作空间的 Workspace Admin 角色或该工作空间上层的 Folder Admin 角色。查看 Folder 与 Workspace 的关系。

2. 资源组：资源组和共享资源是工作空间的两种资源管理模式。

   资源组支持 Cluster、Cluster-Namespace (跨集群)、Mesh、Mesh-Namespace 四种资源类型。 一个资源只能绑定一个资源组，资源被绑定到资源组后，工作空间的所有者将拥有该资源的所有管理权限，相当于该资源的所有者，因此不受资源配额的限制。

   最佳实践：工作空间通过“授权”功能可以给部门成员授予不同角色权限，而工作空间能够把人与角色的授权关系一次性应用到工作空间的所有资源上。因此运维人员只需将资源绑定到资源组，将部门中的不同角色加入不同的资源组，就能确保资源的权限被正确分配。

部门	角色	集群 Cluster	跨集群 Cluster-Namespace	Mesh	Mesh-Namespace
部门管理员	Workspace Admin	√	√	√	√
部门核心成员	Workspace Edit	√	✗	√	✗
其他成员	Workspace View	√	✗	✗	✗

3. 共享资源：共享资源功能主要针对集群资源。

   一个集群可以共享给多个工作空间（指工作空间中的共享资源功能）；一个工作空间也可以同时使用多个集群的资源。 但是资源共享不意味着被共享者（工作空间）可以无限制地使用共享的资源（集群），因此通常会限制共享者（工作空间）能够使用的资源限额。

同时，与资源组不同，工作空间成员只是共享资源的使用者，能够在资源限额下使用集群中的资源。比如前往应用工作台创建命名空间、部署应用等，但并不具备集群的管理权限，限制后在该工作空间下创建/绑定的命名空间的资源配额总和不能超过集群在该工作空间设置的资源使用上限。

最佳实践：运维部门手中有一个高可用集群 01，想要分配给部门 A（工作空间 A）和部门 B（工作空间 B）使用，其中部门 A 分配 CPU 50 核，部门 B 分配 CPU 100 核。 那么可以借用共享资源的概念，将集群 01 分别共享给部门 A 和部门 B，并限制部门 A 的 CPU 使用额度为 50，部门 B 的 CPU 使用额度为 100。 那么部门 A 的管理员（工作空间 A Admin）能够在应用工作台创建并使用命名空间，其中命名空间额度总和不能超过 50 核，部门 B 的管理员（工作空间 B Admin）能够在应用工作台创建并使用命名空间，其中命名空间额度总和不能超过 100 核。 部门 A 的管理员和部门 B 管理员创建的命名空间会被自动绑定在该部门，部门中的其他成员将对应的拥有命名空间的 Namesapce Admin、Namesapce Edit、Namesapce View 角色（这里部门指的是工作空间，工作空间还可以映射为组织、供应商等其他概念）。整个过程如下表：

部门	角色	集群 Cluster	资源配额
部门管理员 A	Workspace Admin	CPU 50 核	CPU 50 核
部门管理员 B	Workspace Admin	CPU 100 核	CPU 100 核
部门其他成员	Namesapce AdminNamesapce EditNamesapce View	按需分配	按需分配

工作空间对 DCE 模块的作用

1. 模块名称：应用工作台、微服务引擎

进入上述几个模块的前提是拥有某个工作空间的权限，因此在使用模块功能前必须具有 Admin 角色或者拥有某个工作空间的一定角色权限。

- 工作空间的角色会自动应用到工作空间包含的资源上。比如当您拥有工作空间 A 的 Workspace Admin 角色，那么对于该工作空间中的所有资源您都是 Admin 角色；
- 如果您是 Workspace Edit，那么对于工作空间中的所有资源您都是 Edit 角色；
- 如果您是 Workspace View，那么对于工作空间中的所有资源您都是 View 角色。

另外，您在这些模块创建的资源也将自动被绑定到对应的工作空间中，而不需要其他额外的操作。

2. 模块名称：容器管理、服务网格

   由于功能模块的特殊性，在容器管理模块创建的资源不会自动被绑定到某个工作空间。

   如果您需要通过工作空间对人和资源进行统一授权管理，可以手动将需要的资源绑定到某个工作空间中，从而将用户在该工作空间的角色应用到资源上（这里的资源是可以跨集群的）。

   另外，在资源的绑定入口上容器管理与服务网格稍有差异，工作空间提供了容器管理中的 Cluster 和 Cluster-Namesapce 资源的绑定入口，但尚未开放对服务网格的 Mesh 和 Mesh-Namespace 资源的绑定。

   对于 Mesh 和 Mesh-Namespace 资源，可以在服务网格的资源列表进行手动绑定。

工作空间的使用场景

- 映射为不同的部门、项目、组织等概念，同时可以将工作空间中 Workspace Admin、Workspace Edit 和 Workspace View 角色对应到部门、项目、组织中的不同角色

- 将不同用途的资源加入不同的工作空间中分开管理和使用
- 为不同工作空间设置完全独立的管理员，实现工作空间范围内的用户与权限管理
- 将资源共享给不同的工作空间使用，并限制工作空间能够使用的资源额度上限

创建/删除工作空间

工作空间是一种资源范畴，代表一种资源层级关系。 工作空间可以包含集群、命名空间、注册中心等资源。 通常一个工作空间对应一个项目，可以为每个工作空间分配不同的资源，指派不同的用户和用户组。

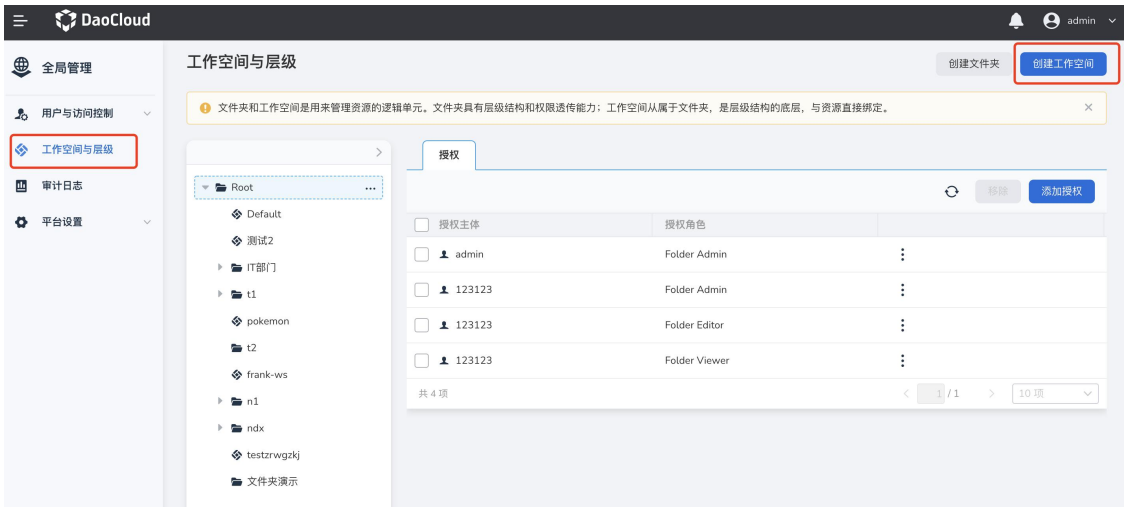
参照以下步骤创建一个工作空间。

1. 使用 admin/folder admin 角色的用户登录 DCE 5.0，点击左侧导航栏底部的全局管理 -> 工作空间与层级。



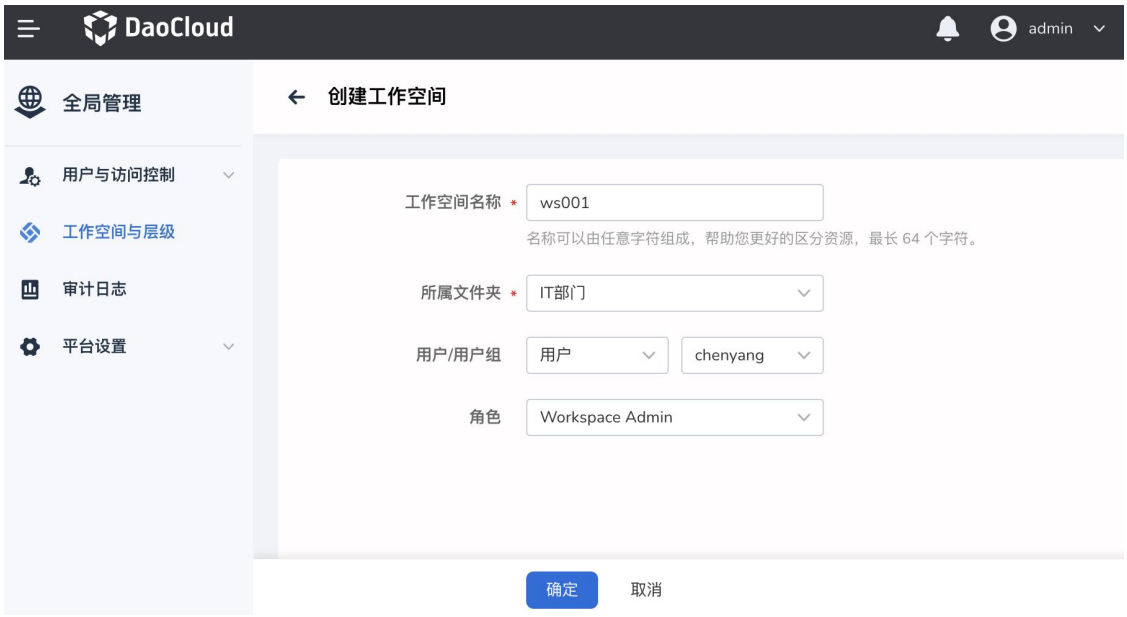
全局管理

2. 点击右上角的创建工作空间按钮。



创建工作空间

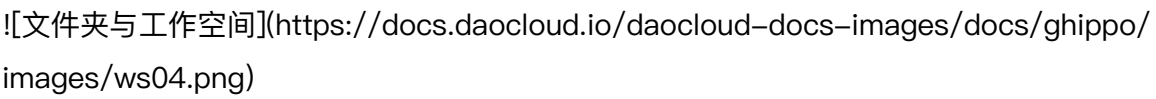
3. 填写工作空间名称、所属文件夹等信息后，点击确定，完成创建工作空间。



确定

!!! tip

创建成功后工作空间名称将显示在左侧的树状结构中，以不同的图标表示文件夹和工作空间。



!!! note

选中某一个工作空间或文件夹，点击右侧的 `...` 可以进行编辑或删除。

- 当该工作空间下资源组、共享资源中存在资源时，该工作空间无法被删除，需要将所有资源解绑后再删除。

- 当微服务引擎模块在该工作空间下存在接入注册中心资源时，该工作空间无法被删除，需要将所有接入注册中心移除后再删除工作空间。

工作空间权限说明

工作空间具有权限映射和资源隔离能力，能够将用户/用户组在工作空间的权限映射到其下的资源上。 若用户/用户组在工作空间是 Workspace Admin 角色，同时工作空间-资源组中绑定了资源 Namespace，则映射后该用户/用户组将成为 Namespace Admin。

!!! note

工作空间的权限映射能力不会作用到共享资源上，因为共享是将集群的使用权限共享给多个工作空间，而不是将管理权限受让给工作空间，因此不会实现权限继承和角色映射。

应用场景

通过将资源绑定到不同的工作空间能够实现资源隔离。 因此借助权限映射、资源隔离和共享资源能力能够将资源灵活分配给各个工作空间（租户）。

通常适用于以下两个场景：

- 集群一对一

普通集群	部门/租户（工作空间）	用途
集群 01	A	管理和使用
集群 02	B	管理和使用

- 集群一对多

集群	部门/租户（工作空间）	资源限额
集群 01	A	100 核 CPU
	B	50 核 CPU

权限说明

操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
本身	查看	√	√	√
–	授权	√	✗	✗
–	修改别名	√	√	✗
资源组	查看	√	√	√
–	资源绑定	√	✗	✗
–	解除绑定	√	✗	✗
共享资源	查看	√	√	√
–	新增共享	√	✗	✗
–	解除共享	√	✗	✗
–	资源限额	√	✗	✗
–	使用共享资源 [13]	√	✗	✗

[13] 授权用户可前往应用工作台、微服务引擎、中间件、多云编排、服务网格等模块使用工作空间中的资源。 有关 Workspace Admin、Workspace Editor、Workspace Viewer 角色在各产品模块的操作范围，请查阅各模块的权限说明：

- 应用工作台权限说明
- 服务网格权限说明
- 中间件权限说明
- 微服务引擎权限说明
- 容器管理权限说明

将集群分配给多个工作空间（租户）

集群资源通常由运维人员进行管理。在分配资源分配时，他们需要创建命名空间来隔离资源，并设置资源配额。 这种方式有个弊端，如果企业的业务量很大，手动分配资源需要较大的工作量，而想要灵活调配资源额度也有不小难度。

DCE 为此引入了工作空间的概念。工作空间通过共享资源可以提供更高维度的资源限额能力，实现工作空间（租户）在资源限额下自助式创建 Kubernetes 命名空间的能力。

举例而言，如果想要让几个部门共享不同的集群。

	Cluster01（普通）	Cluster02（高可用）
部门（工作空间）A	50 quota	10 quota
部门（工作空间）B	100 quota	20 quota

可以参照以下流程将集群分享给多个部门/工作空间/租户：

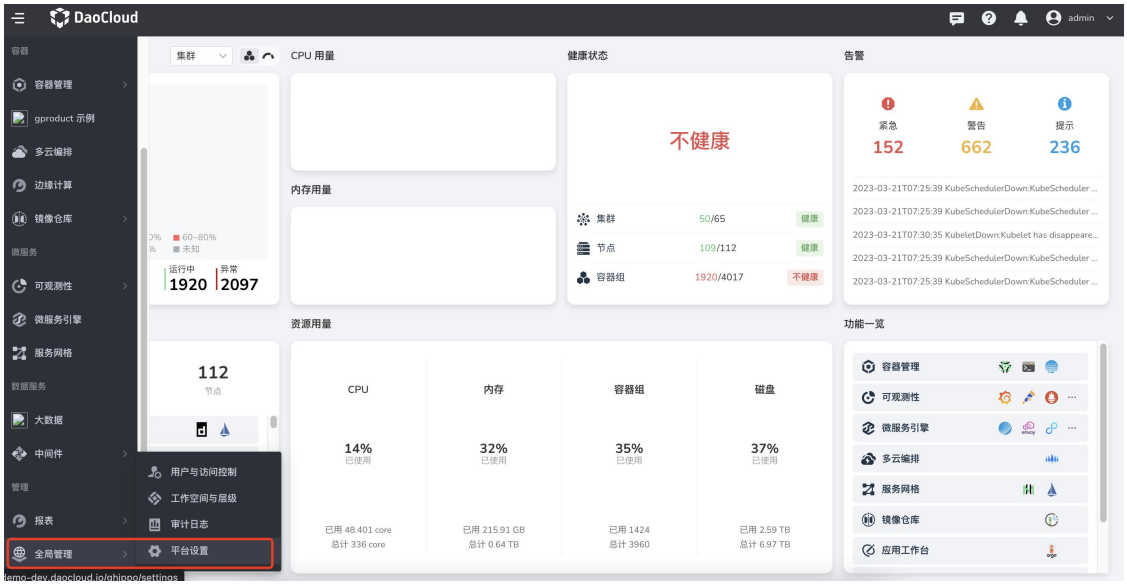
```
graph TB

preparews[准备工作空间] --> preparecs[准备集群]
--> share[将集群共享到工作空间]
--> judge([判断工作空间剩余额度])
judge -.大于剩余额度.->modifyns[修改命名空间额度]
judge -.小于剩余额度.->createns[创建命名空间]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class preparews,preparecs,share, cluster;
```

```
class judge plain
class modifyns,createns k8s

click preparews "https://docs.daocloud.io/ghippo/user-guide/workspace/cluster-for-multiws/#_2"
click preparecs "https://docs.daocloud.io/ghippo/user-guide/workspace/cluster-for-multiws/#_3"
click share "https://docs.daocloud.io/ghippo/user-guide/workspace/cluster-for-multiws/#_4"
click createns "https://docs.daocloud.io/amamba/user-guide/namespace/namespace/#_3"
click modifyns "https://docs.daocloud.io/amamba/user-guide/namespace/namespace/#_4"
```

准备一个工作空间

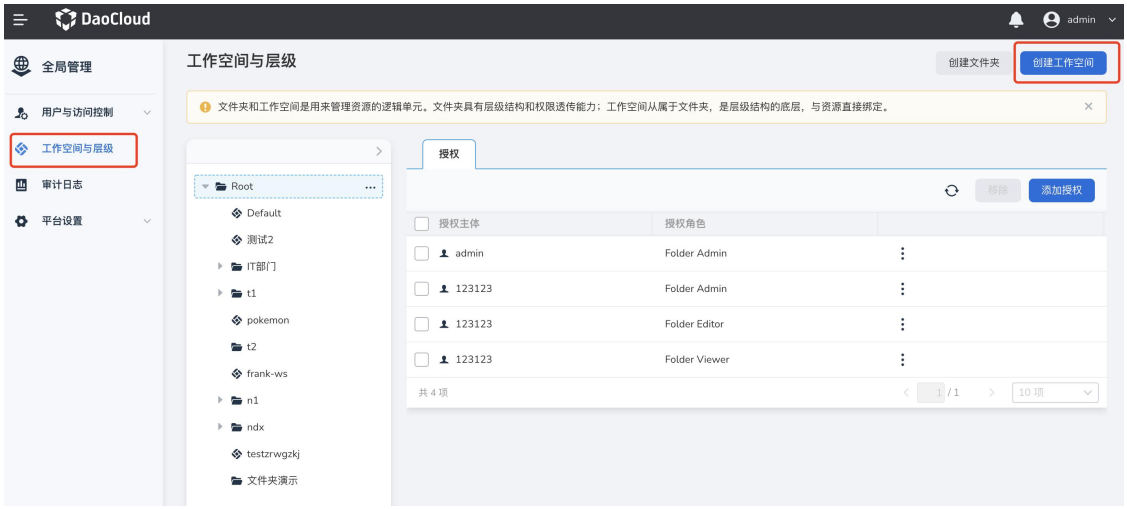
工作空间是为了满足多租户的使用场景，基于集群、集群命名空间、网格、网格命名空间、多云、多云命名空间等多种资源形成相互隔离的资源环境， 工作空间可以映射为项目、租户、企业、供应商等多种概念。

1. 使用 admin/folder admin 角色的用户登录 DCE 5.0，点击左侧导航栏底部的全局管理。



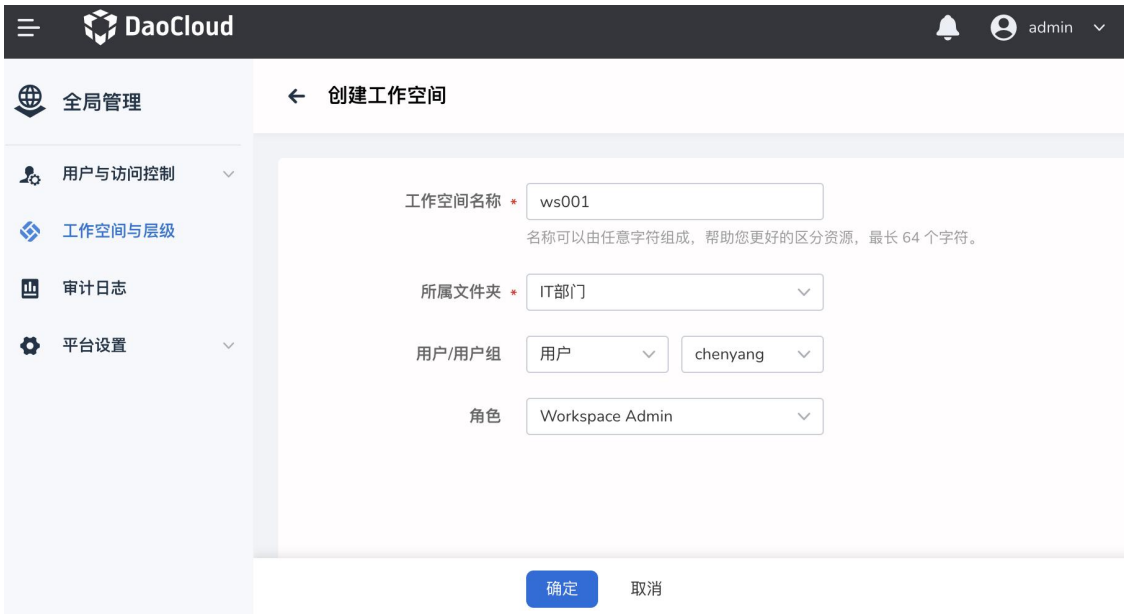
全局管理

2. 点击左侧导航栏的工作空间与层级，点击右上角的创建工作空间按钮。



创建工作空间

3. 填写工作空间名称、所属文件夹等信息后，点击确定，完成创建工作空间。



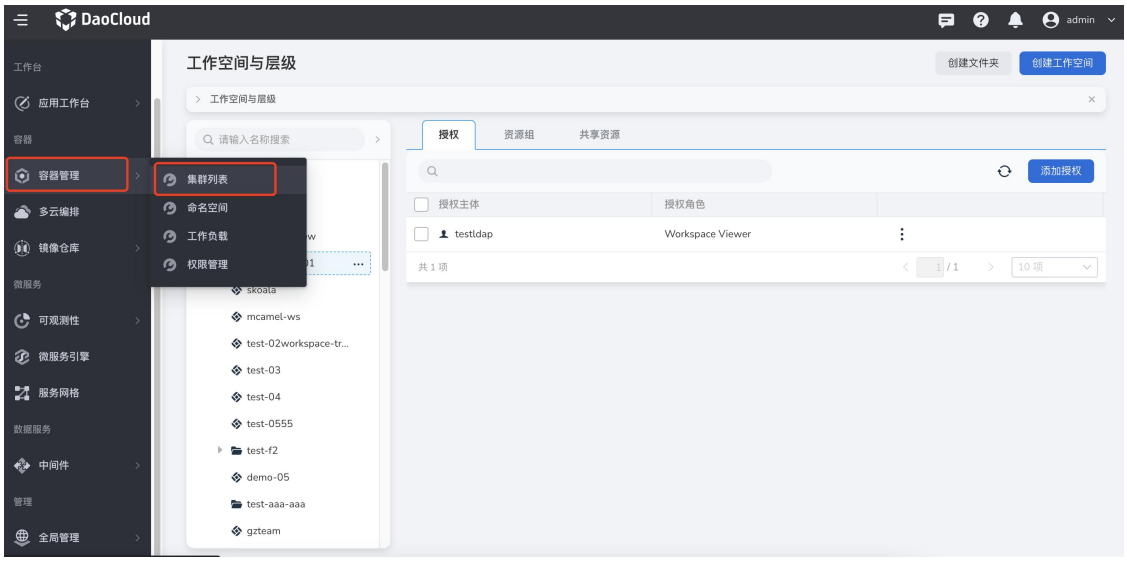
确定

准备一个集群

工作空间是为了满足多租户的使用场景，基于集群、集群命名空间、网格、网格命名空间、多云、多云命名空间等多种资源形成相互隔离的资源环境，工作空间可以映射为项目、租户、企业、供应商等多种概念。

参照以下步骤准备一个集群。

1. 点击左侧导航栏底部的容器管理，选择集群列表。



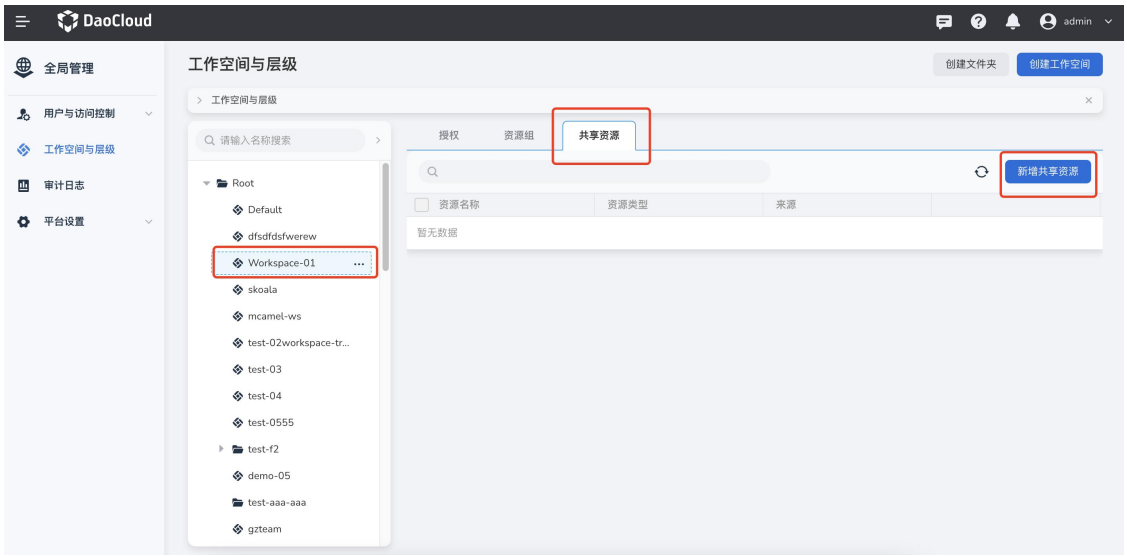
容器管理

2. 点击创建集群创建一个集群，或点击接入集群接入一个集群。

在工作空间添加集群

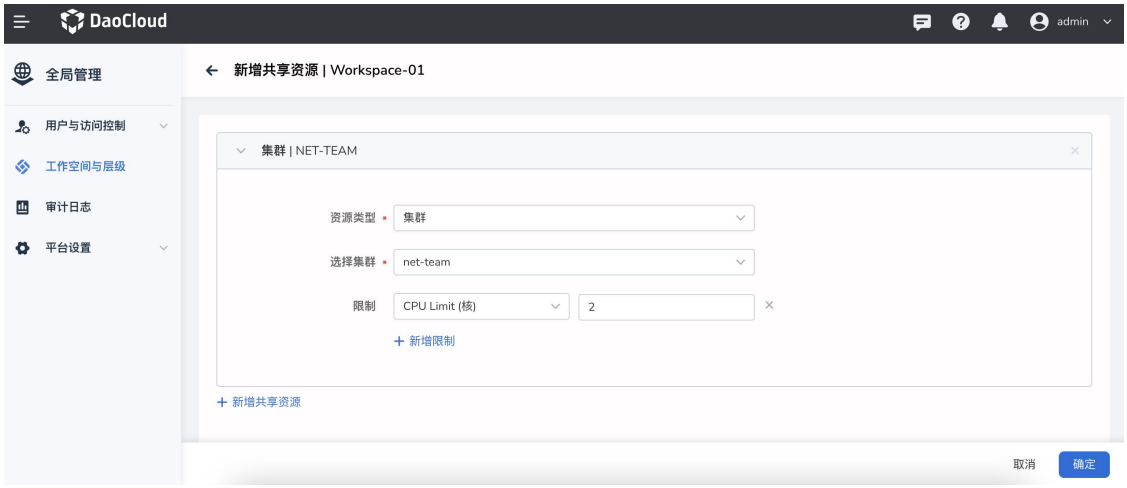
返回全局管理，为工作空间添加集群。

1. 依次点击全局管理 -> 工作空间与层级 -> 共享资源，点击某个工作空间名称后，点击新增共享资源按钮。



新增资源

2. 选择集群，填写资源限额后，点击确定。



新增资源

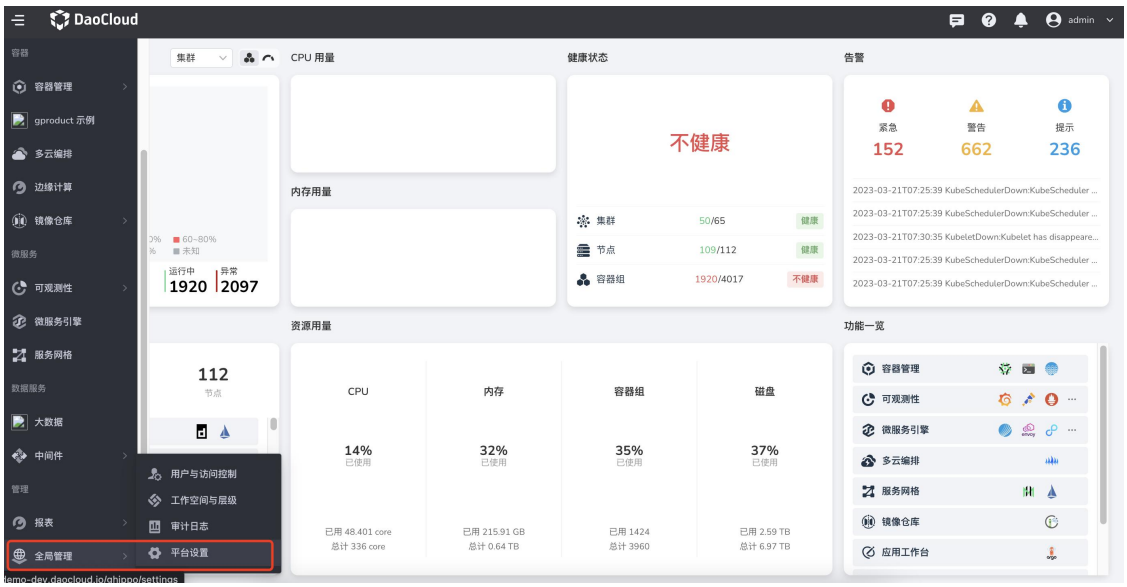
下一步：将集群资源分配给多个工作空间后，用户可以前往应用工作台在这些工作空间下创建命名空间并部署应用。

创建/删除文件夹

文件夹具有权限映射能力，能够将用户/用户组在本文件夹的权限映射到其下的子文件夹、工作空间以及资源上。

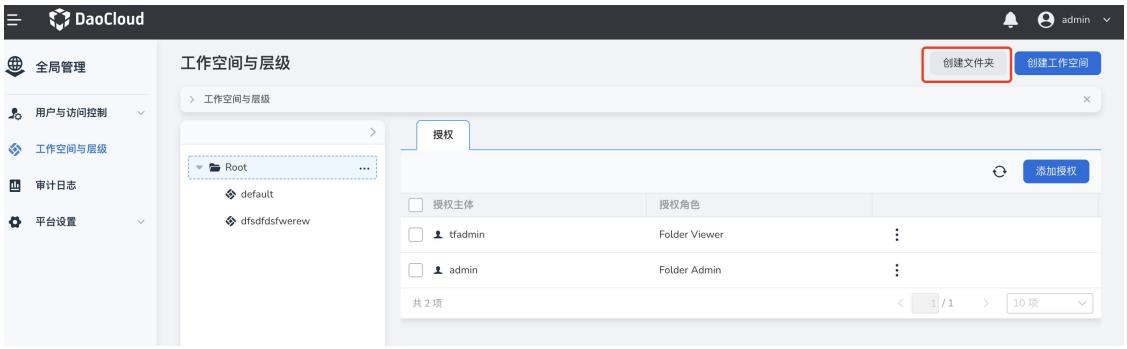
参照以下步骤创建一个文件夹。

1. 使用 admin/folder admin 角色的用户登录 DCE 5.0，点击左侧导航栏底部的全局管理 -> 工作空间与层级。



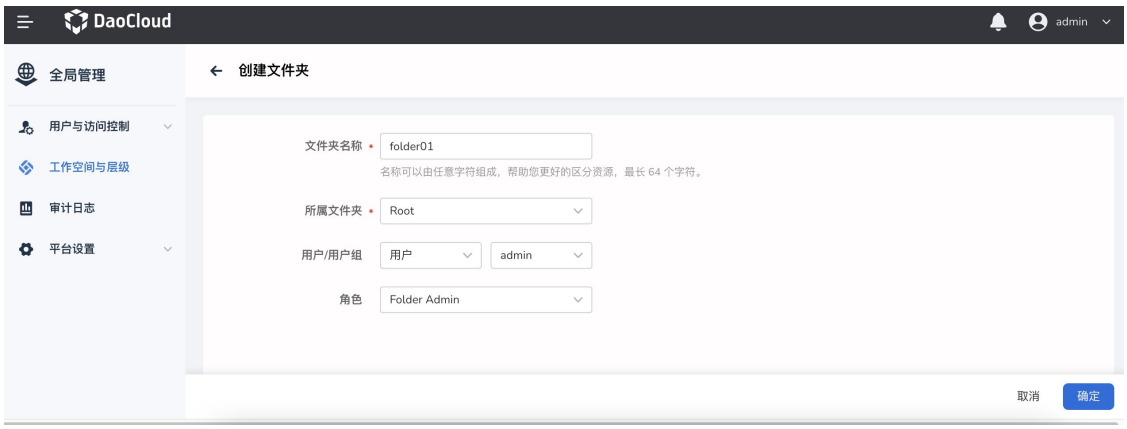
全局管理

2. 点击右上角的创建文件夹按钮。



创建文件夹

3. 填写文件夹名称、上一级文件夹等信息后，点击确定，完成创建文件夹。



确定

!!! tip

创建成功后文件夹名称将显示在左侧的树状结构中，以不同的图标表示工作空间和文件夹。

![工作空间和文件夹](https://docs.daocloud.io/daocloud-docs-images/docs/ghippo/images/ws04.png)

!!! note

选中某一个文件夹或文件夹，点击右侧的 `...` 可以进行编辑或删除。

- 当该文件夹下资源组、共享资源中存在资源时，该文件夹无法被删除，需要将所有资源

解绑后再删除。

– 当微服务引擎模块在该文件夹下存在接入注册中心资源时，该文件夹无法被删除，需要将所有接入注册中心移除后再删除文件夹。

资源组与共享资源的区别

资源组与共享资源均支持绑定集群，但使用上存在很大区别。

使用场景区别

- 资源组绑定集群：资源组绑定集群通常被用来批量授权。资源组绑定集群后， 工作空间管理员将被映射为集群管理员，能够管理并使用集群资源。
- 共享资源绑定集群：资源共享绑定集群通常被用来做资源限额。 典型的场景是平台管理员将集群分配给一级供应商后，再由一级供应商分配给二级供应商并对二级供应商进行资源限额。



diff

说明：在该场景中，需要平台管理员对二级供应商进行资源限制，暂时还不支持一级供应商限制二级供应商的集群额度。

集群额度的使用区别

- 资源组绑定集群：工作空间的管理员将被映射为该集群的管理员，相当于在容器管理–权限管理中被授予 Cluster Admin 角色， 能够无限制支配该集群资源，管理节点等重要内容，且资源组不能够被资源限额。

- 共享资源绑定资源：工作空间管理员仅能够使用集群中的额度在应用工作台创建命名空间，不具备集群的管理权限。 若对该工作空间限制额度，则工作空间管理仅能够在额度范围内创建并使用命名空间。

资源类型的区别

- 资源组：能够绑定集群、集群-命名空间、多云、多云-命名空间、网格、网格-命名空间
- 共享资源：仅能够绑定集群

资源组与共享资源的相同点

在资源组/共享资源绑定集群后都可以前往应用工作台创建命名空间，创建后命名空间将自动绑定到工作空间。

普通用户授权规划

普通用户是指能够使用 DCE 大部分产品模块及功能（管理功能除外），对权限范围内的资源有一定的操作权限，能够独立使用资源部署应用。

对这类用户的授权及资源规划流程如下图所示。

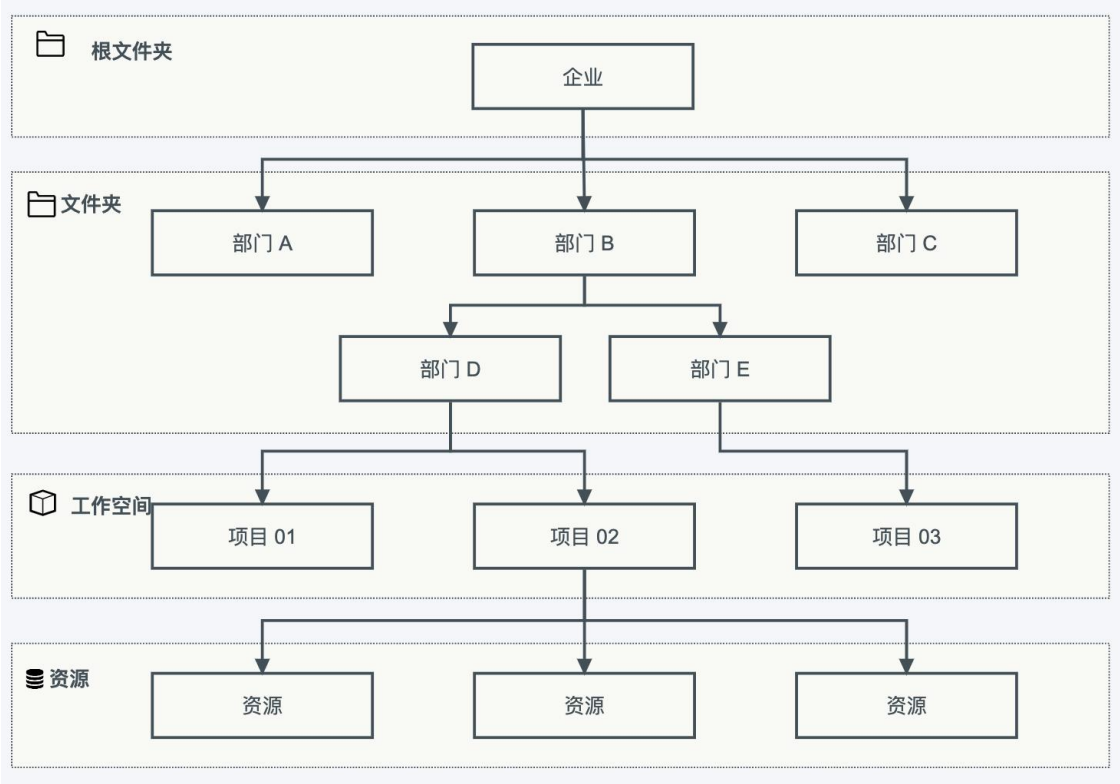
```
graph TB

    start([开始]) --> user[1. 创建用户]
    user --> ns[2. 准备 Kubernetes 命名空间]
    ns --> ws[3. 准备工作空间]
    ws --> ws-to-ns[4. 工作空间绑定命名空间]
    ws-to-ns --> authu[5. 给用户授权 Workspace Editor]
    authu --> complete([结束])

click user "https://docs.daocloud.io/ghippo/user-guide/access-control/user/"
click ns "https://docs.daocloud.io/kpanda/user-guide/namespaces/createns/"
click ws "https://docs.daocloud.io/ghippo/user-guide/workspace/workspace/"
click ws-to-ns "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/"
click authu "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-permission/"

 classDef plain fill:#ddd,stroke:#fff,stroke-width:4px,color:#000;
 classDef k8s fill:#326ce5,stroke:#fff,stroke-width:4px,color:#fff;
 classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
 class user,ns,ws,ws-to-ns,authu cluster;
 class start,complete plain;
```

授权后普通用户在各模块的权限为：

- 应用工作台
- 微服务引擎
- 服务网格（需要重复 2 – 4 准备网格/网格命名空间并绑定到工作空间）
- 可观测性
- 数据服务

工作空间与层级

工作空间与层级是一个具有层级的资源隔离和资源分组特性，主要解决资源统一授权、资源分组以及资源限额问题。

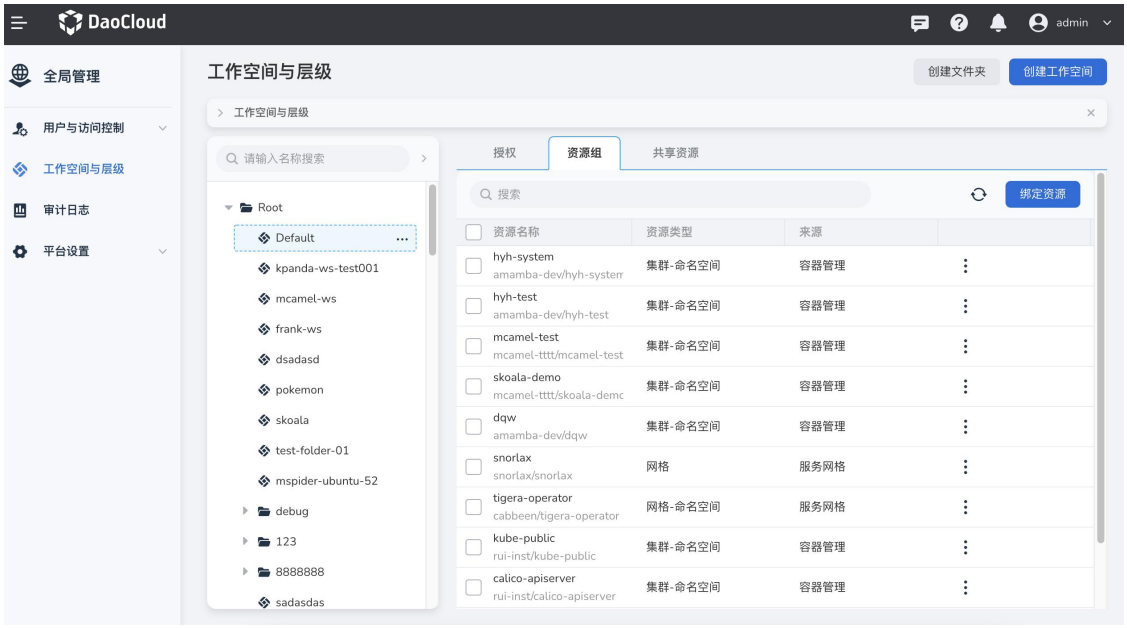


层级结构

工作空间与层级有两个概念：工作空间和文件夹。

工作空间

工作空间可通过授权、资源组和共享资源来管理资源，使用户（用户组）之间能够共享工作空间中的资源。



工作空间

- 资源

  资源处于资源管理模块层级结构的最低层级，资源包括 Cluster、Namespace、Pipeline、网关等。 所有这些资源的父级只能是工作空间，而工作空间作为资源容器是一种资源分组单位。

- 工作空间

  工作空间通常代指一个项目或环境，每个工作空间的资源相对于其他工作空间中的资源时逻辑隔离的。 您可以通过工作空间中的授权，授予用户（用户组）同一组资源的不同访问权限。

  从层次结构的底层算起，工作空间位于第一层，且包含资源。 除共享资源外，所有资源有且仅有一个父项。所有工作空间也有且仅有一个父级文件夹。

  资源通过工作空间进行分组，而工作空间中存在两种分组模式，分别是资源组和共享资源。

- 资源组

一个资源只能加入一个资源组，资源组与工作空间一一对应。 资源被加入到资源组后，Workspace Admin 将获得资源的管理权限，相当于该资源的所有者。

- 共享资源

  而对于共享资源来说，多个工作空间可以共享同一个或者多个资源。 资源的所有者，可以选择将自己拥有的资源共享给工作空间使用，一般共享时资源所有者会限制被共享工作空间能够使用的资源额度。 资源被共享后，Workspace Admin 仅具有资源限额下的资源使用权限，无法管理资源或者调整工作空间能够使用的资源量。

  同时共享资源对于资源本身也具有一定的要求，只有 Cluster（集群）资源可以被共享。 Cluster Admin 能够将 Cluster 资源分享给不同的工作空间使用，并且限制工作空间在此 Cluster 上的使用额度。

  Workspace Admin 在资源限额内能够创建多个 Namespace，但是 Namespace 的资源额度总和不能超过 Cluster 在该工作空间的资源限额。 对于 Kubernetes 资源，当前能够分享的资源类型仅有 Cluster。

文件夹

文件夹可用于构建企业业务层级关系。

- 文件夹是在工作空间基础之上的进一步分组机制，具有层级结构。 一个文件夹可以包含工作空间、其他文件夹或两者的组合，能够形成树状的组织关系。
- 借助文件夹您可以映射企业业务层级关系，按照部门对工作空间进行分组。文件夹不直接与资源挂钩，而是通过工作空间间接实现资源分组。
- 文件夹有且仅有一个父级文件夹，而根文件夹是层次结构的最高层级。 根文件夹没有父级，文件夹和工作空间均挂靠到根文件夹下。

另外，用户（用户组）在文件夹中能够通过层级结构继承来自于父项的权限。 用户在层次结构中的权限来自当前层级的权限以及继承其父项权限的组合结果，权限之间是加合关系不存在互斥。

资源配额（Quota）

共享资源并非意味着被共享者可以无限制地使用被共享的资源。 Admin、Kpanda Owner 和 Workspace Admin 可以通过共享资源中的资源配额功能限制某个用户的最大使用额度。 若不限制，则表示可以无限制使用。

- CPU 请求（Core）
- CPU 限制（Core）
- 内存请求（MB）
- 内存限制（MB）
- 存储请求总量（GB）
- 存储卷声明（个）

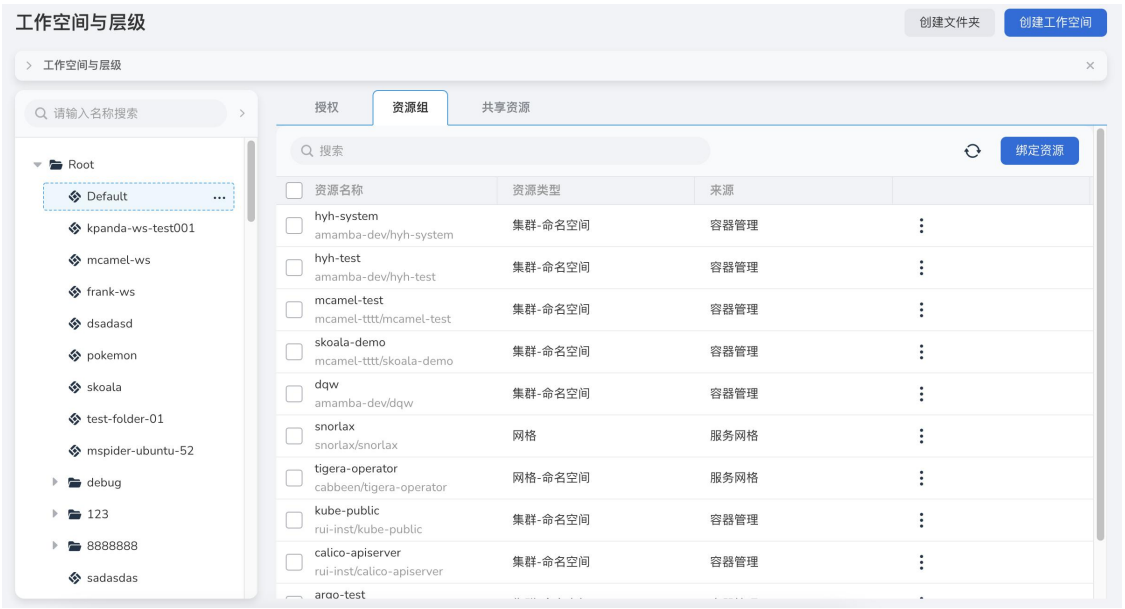
一个资源（集群）可以被多个工作空间共享，一个工作空间也可以同时使用多个共享集群中的资源。

资源组和共享资源

共享资源和资源组中的集群资源均来自容器管理，但是集群绑定和共享给同一个工作空间将会产生两种截然不同的效果。

1. 绑定资源

   使工作空间中的用户/用户组具有该集群的全部管理和使用权限，Workspace Admin 将被映射为 Cluster Admin。 Workspace Admin 能够进入容器管理模块管理该集群。



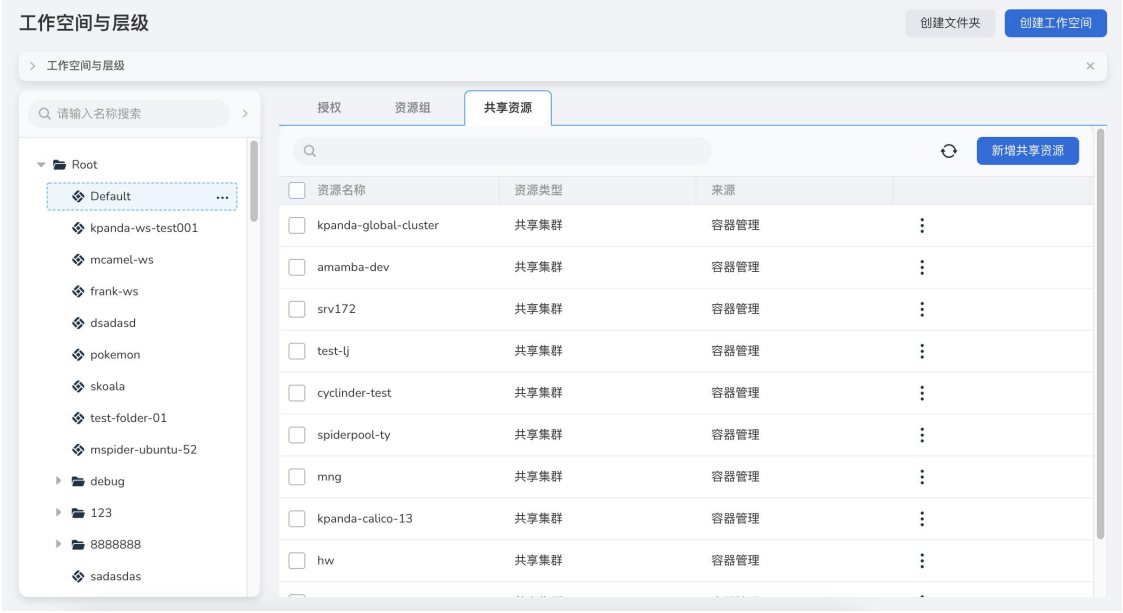
资源组

!!! note

当前容器管理模块暂无 Cluster Editor 和 Cluster Viewer 角色，因此 Workspace Editor、Workspace Viewer 还无法映射。

2. 新增共享资源

使工作空间中的用户/用户组具有该集群资源的使用权限，这些资源可以在创建命名空间（Namespace）时使用。



共享资源

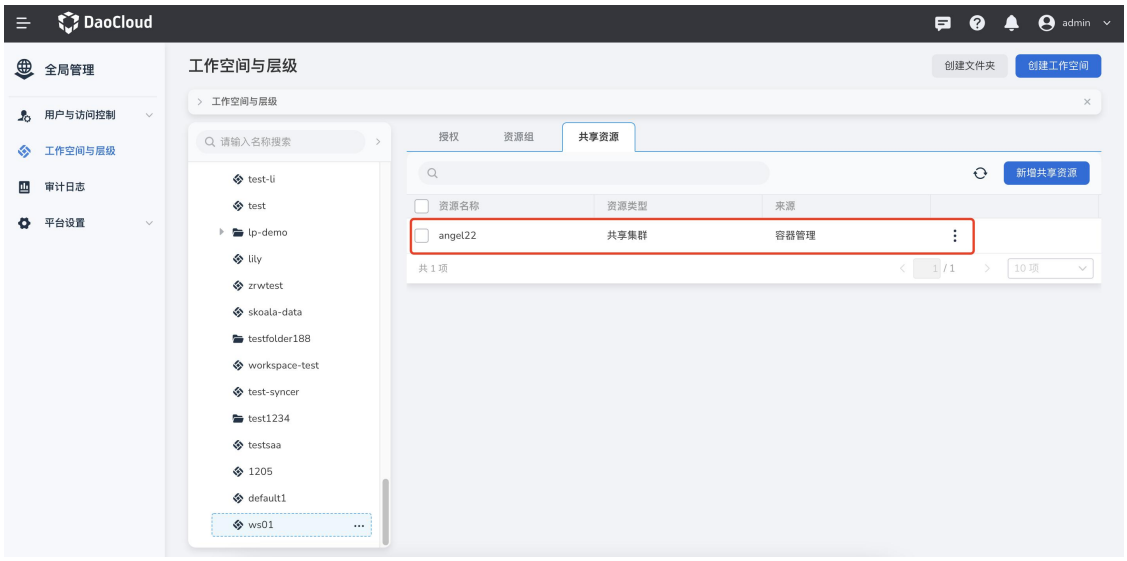
与资源组不同，将集群共享到工作空间时，用户在工作空间的角色不会映射到资源上，因此 Workspace Admin 不会被映射为 Cluster admin。

本节展示 3 个与资源配额有关的场景。

创建命名空间

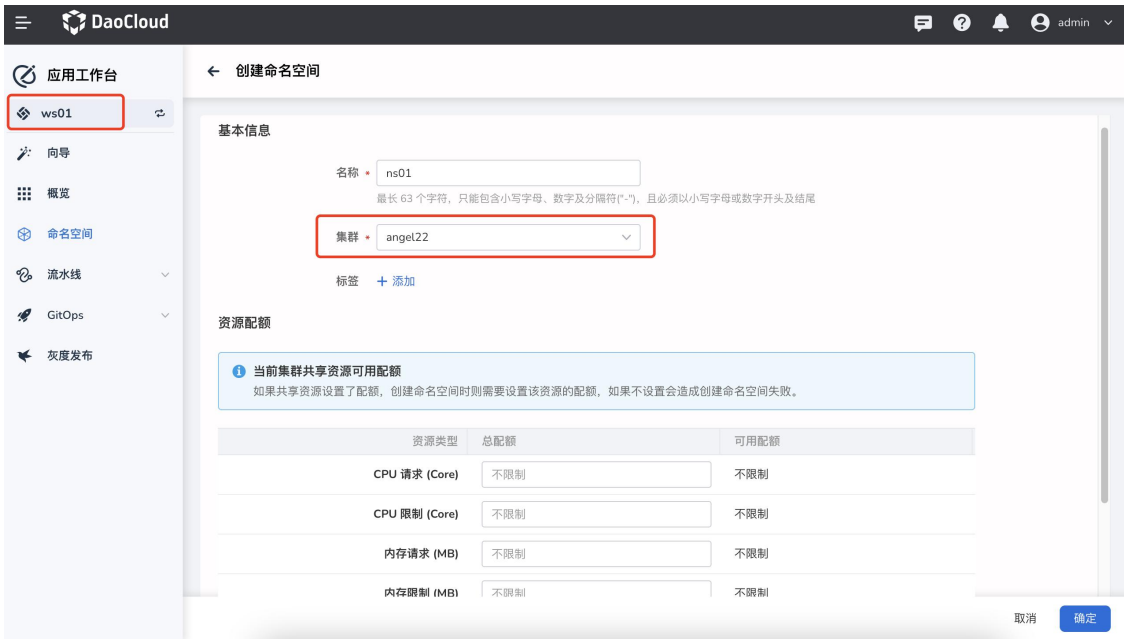
创建命名空间时会涉及到资源配额。

1. 在工作空间 ws01 新增一个共享集群。



新增共享集群

2. 在应用工作台选择工作空间 ws01 和共享集群，创建命名空间 ns01。



创建命名空间

- 若在共享集群中未设置资源配额，则创建命名空间时可不设置资源配额。

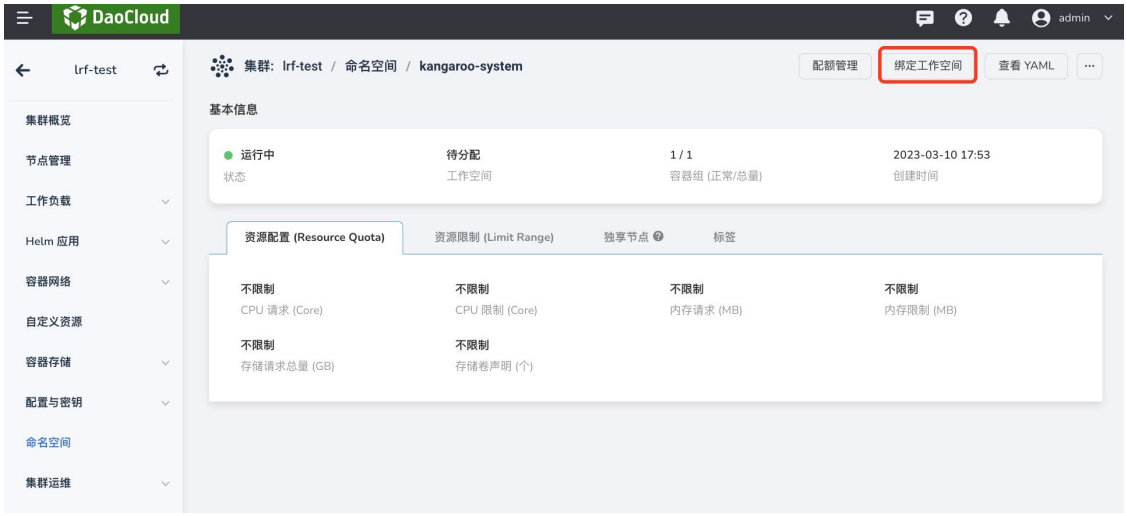
- 若在共享集群中已设置资源配额（例如 CPU 请求 = 100 core），则创建命名空间时 CPU 请求 ⩽ 100 core。

命名空间绑定到工作空间

前提：工作空间 ws01 已新增共享集群，操作者为 Workspace Admin + Kpanda Owner 或 Admin 角色。

以下两种绑定方式的效果相同。

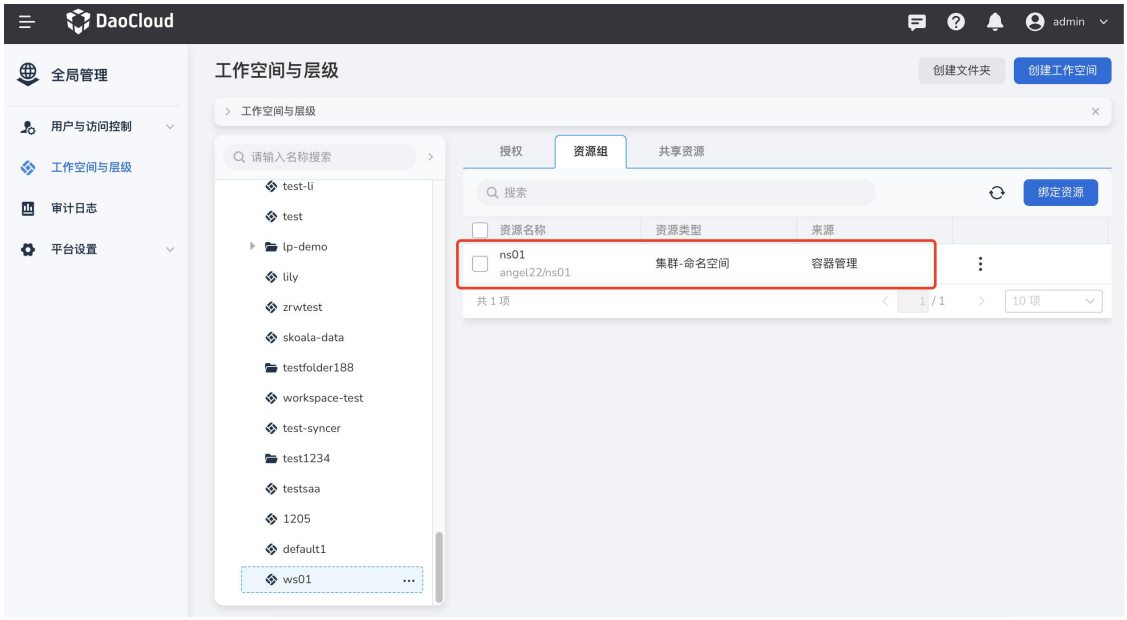
- 在容器管理中将创建的命名空间 ns01 绑定到 ws01



绑定到工作空间

- 若在共享集群未设置资源配额，则命名空间 ns01 无论是否已设置资源配额，均可成功绑定。
- 若在共享集群已设置资源配额 CPU 请求 = 100 core，则命名空间 ns01 必须满足 CPU 请求 ⩽ 100 core 才能绑定成功。

- 在全局管理中，将命名空间 ns01 绑定到 ws01



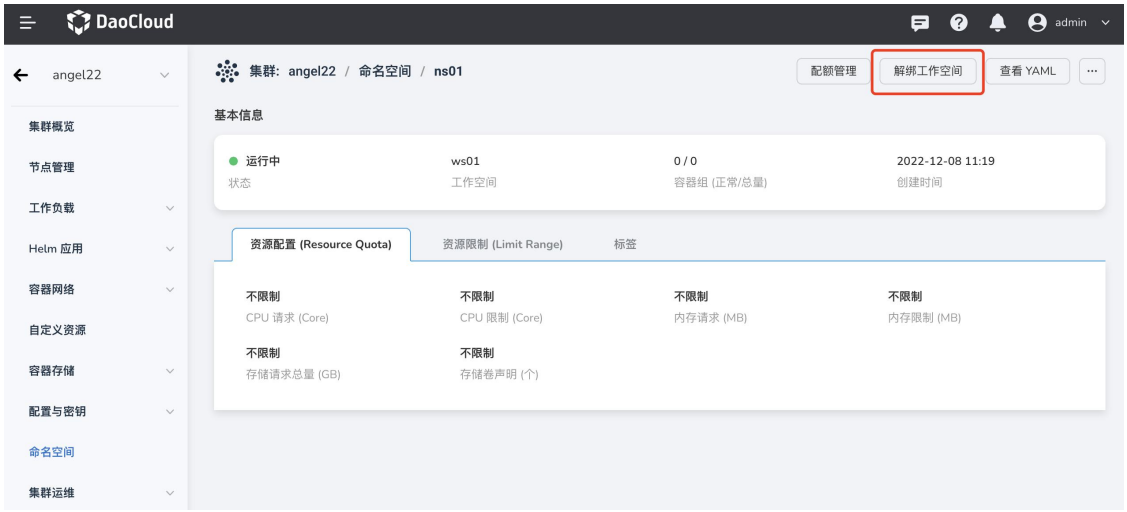
绑定到工作空间

- 若在共享集群未设置资源配额，则命名空间 ns01 无论是否已设置资源配额，均可成功绑定。
- 若在共享集群已设置资源配额 CPU 请求 = 100 core，则命名空间 ns01 必须满足 CPU 请求 ⩽ 100 core 才能绑定成功。

从工作空间解绑命名空间

以下两种解绑方式的效果相同。

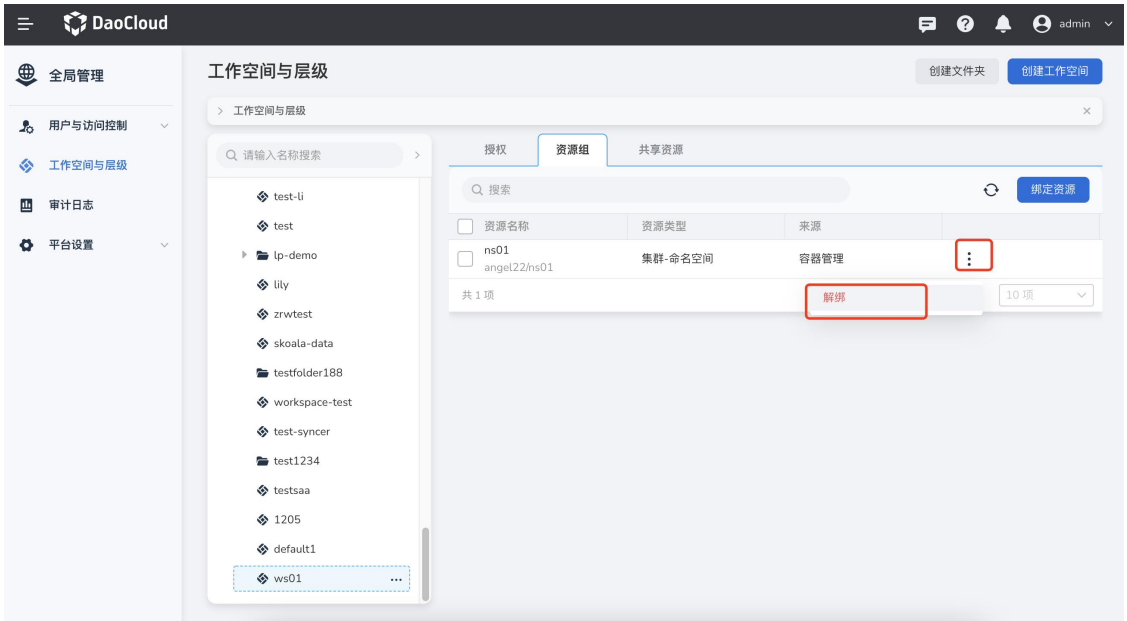
- 在容器管理中将命名空间 ns01 从工作空间 ws01 解绑



绑定到工作空间

- 若在共享集群中未设置资源配额，则命名空间 ns01 无论是否已设置资源配额，解绑后均不会对资源配额产生影响。
- 若在共享集群已设置资源配额 CPU 请求 = 100 core，命名空间 ns01 也设置了资源配额，则解绑后将释放相应的资源额度。

- 在全局管理中将命名空间 ns01 从工作空间 ws01 解绑



绑定到工作空间

- 若在共享集群未设置资源配额，则命名空间 ns01 无论是否已设置资源配额，解绑后均不会对资源配额产生影响。
- 若在共享集群已设置资源配额 CPU 请求 = 100 core，命名空间 ns01 也设置了资源配额，则解绑后将释放相应的资源额度。

文件夹最佳实践

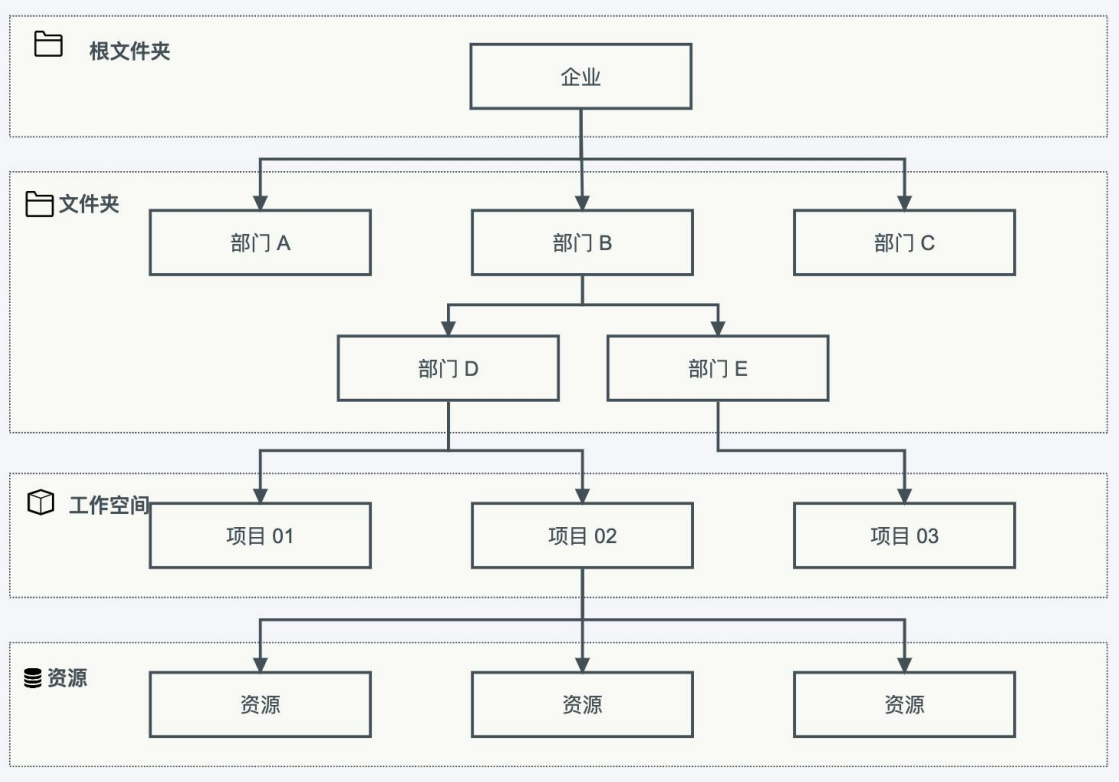
文件夹代表一个组织机构（例如一个部门），是资源层次结构中的一个节点。

一个文件夹可以包含工作空间、子文件夹或两者的组合。 它提供了身份管理、多层级和权限映射能力，能够将用户/用户组在文件夹中的角色映射到其下的子文件夹、工作空间和资源上。 因此借助于文件夹，企业管理者能够集中管控所有资源。

1. 构建企业层级关系

   首先要按照现有的企业层级结构，构建与企业相同的文件夹层级。 DCE 支持 5 级文件夹，可以根据企业实际情况自由组合，将文件夹和工作空间映射为企业中的部门、项目、供应商等实体。

   文件夹不直接与资源挂钩，而是通过工作空间间接实现资源分组。



层级结构

2. 用户身份管理

   文件夹提供了 Folder Admin、Folder Editor、Folder Viewer 三种角色。查看角色权限，可通过授权给同一文件夹中的用户/用户组授予不同的角色。

3. 角色权限映射

   企业管理者：在根文件夹授予 Folder Admin 角色。他将拥有所有部门、项目及其资源的管理权限。

   部门管理者：在各个子文件夹、工作空间单独授予管理权限。

   项目成员：在工作空间、资源层级单独授予管理权限。

工作空间（租户）绑定跨集群的命名空间

工作空间（租户）下绑定来自不同集群的命名空间，能够使工作空间（租户）灵活纳管平台上任意集群下的 Kubernetes Namespace。 同时平台提供了权限映射能力，能够将用户在工作空间的权限映射到绑定的命名空间身上。

当工作空间（租户）下绑定一个或多个跨集群的命名空间时，管理员无需再次给工作空间中的成员授权， 成员们在工作空间上的角色将根据以下映射关系自动映射完成授权，避免了多次授权的重复性操作：

- Workspace Admin 对应 Namespace Admin
- Workspace Editor 对应 Namespace Editor
- Workspace Viewer 对应 Namespace Viewer

以下是一个例子：

用户	工作空间	角色
用户 A	Workspace01	Workspace Admin

将一个命名空间绑定到工作空间后：

用户	所属范畴	角色
用户 A	Workspace01	Workspace Admin
	Namespace01	Namespace Admin

实现方案

将来自不同集群的不同命名空间绑定到同一工作空间（租户），给工作空间（租户）下的成员使用流程如图。

graph TB

```
preparews[准备工作空间] --> preparens[准备命名空间]
--> judge([命名空间是否与绑定到其他工作空间])
judge -.未绑定.->nstows[将命名空间绑定到工作空间] --> wsperm[管理工作空间访问权限]
judge -.已绑定.->createns[创建新的命名空间]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class preparews,preparens,createns,nstows,wsperm cluster;
class judge plain

click preparews "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/#_3"
click preparens "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/#_4"
click nstows "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/#_5"
click wsperm "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/#_6"
click createns "https://docs.daocloud.io/ghippo/user-guide/workspace/ws-to-ns/#_4"
```

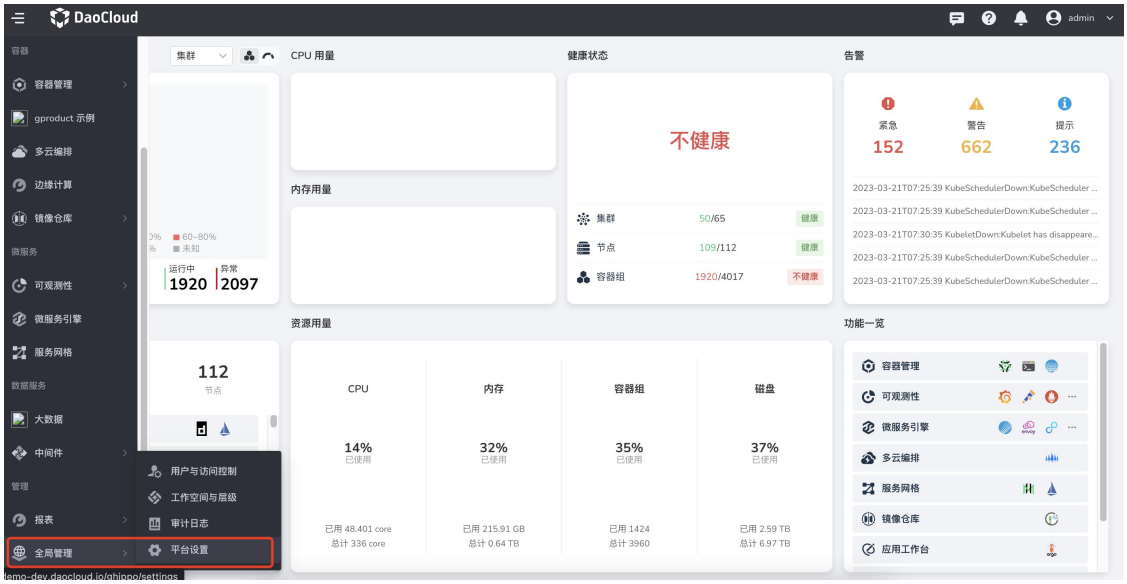
!!! tip

一个命名空间只能被一个工作空间绑定。

准备工作空间

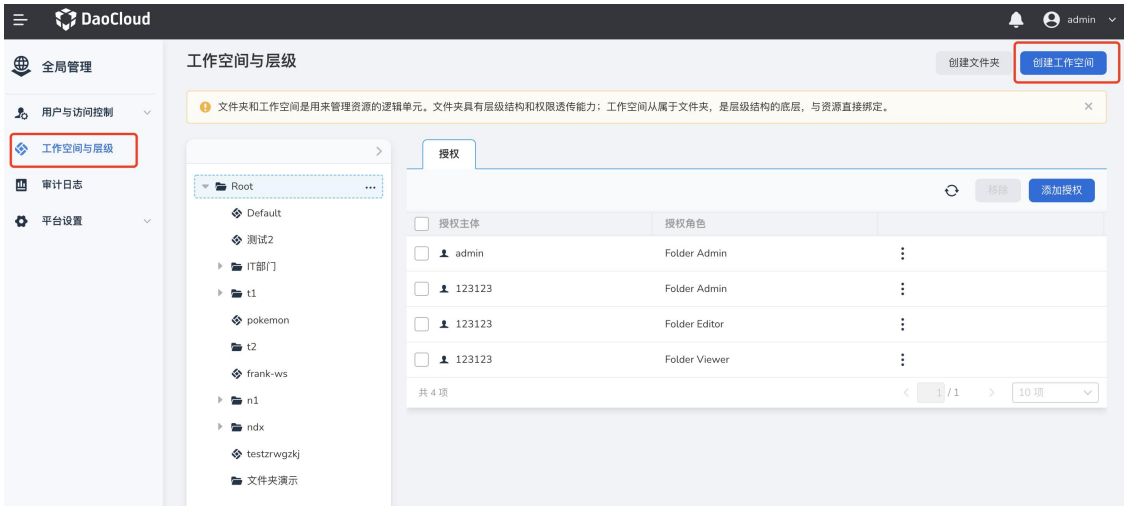
工作空间是为了满足多租户的使用场景，基于集群、集群命名空间、网格、网格命名空间、多云、多云命名空间等多种资源形成相互隔离的资源环境。 工作空间可以映射为项目、租户、企业、供应商等多种概念。

1. 使用 admin/folder admin 角色的用户登录 DCE 5.0，点击左侧导航栏底部的全局管理。



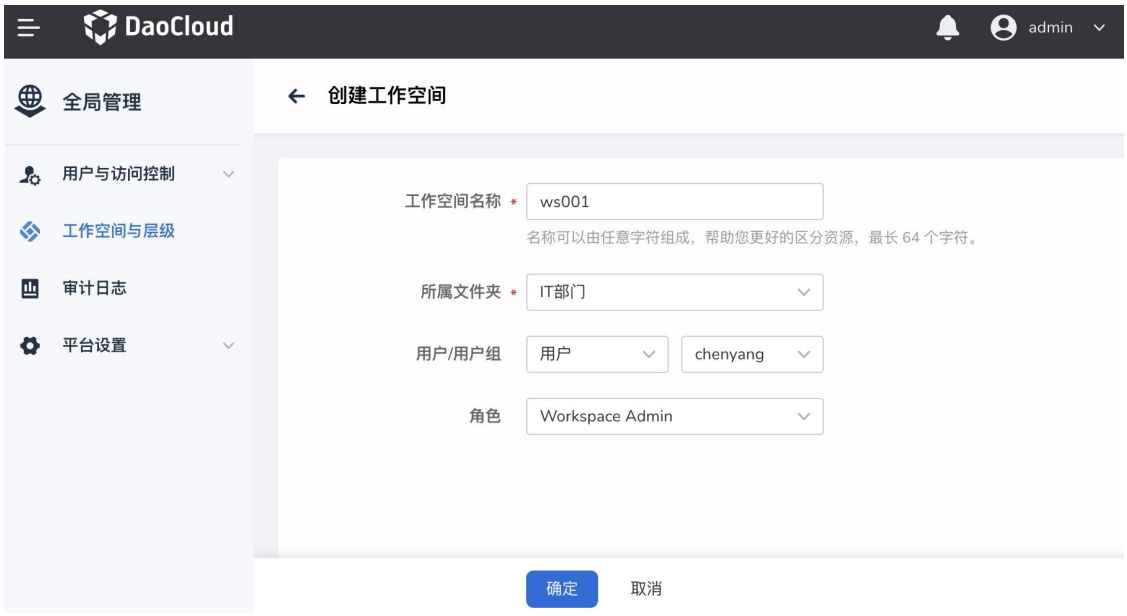
全局管理

2. 点击左侧导航栏的工作空间与层级，点击右上角的创建工作空间按钮。



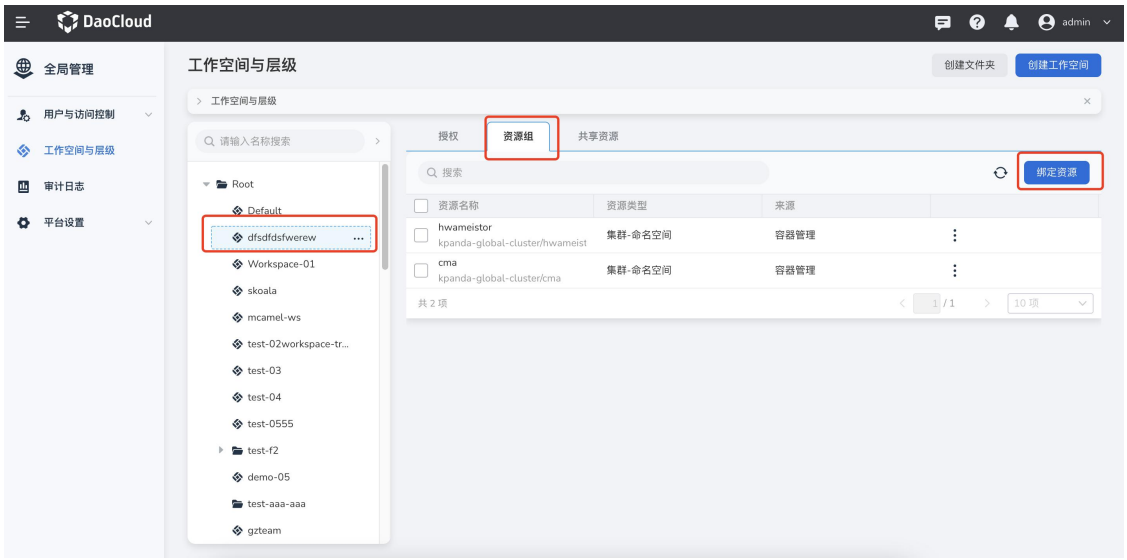
创建工作空间

3. 填写工作空间名称、所属文件夹等信息后，点击确定，完成创建工作空间。



确定

提示：若平台中已存在创建好的命名空间，点击某个工作空间，在资源组页签下，点击绑定资源，可以直接绑定命名空间。



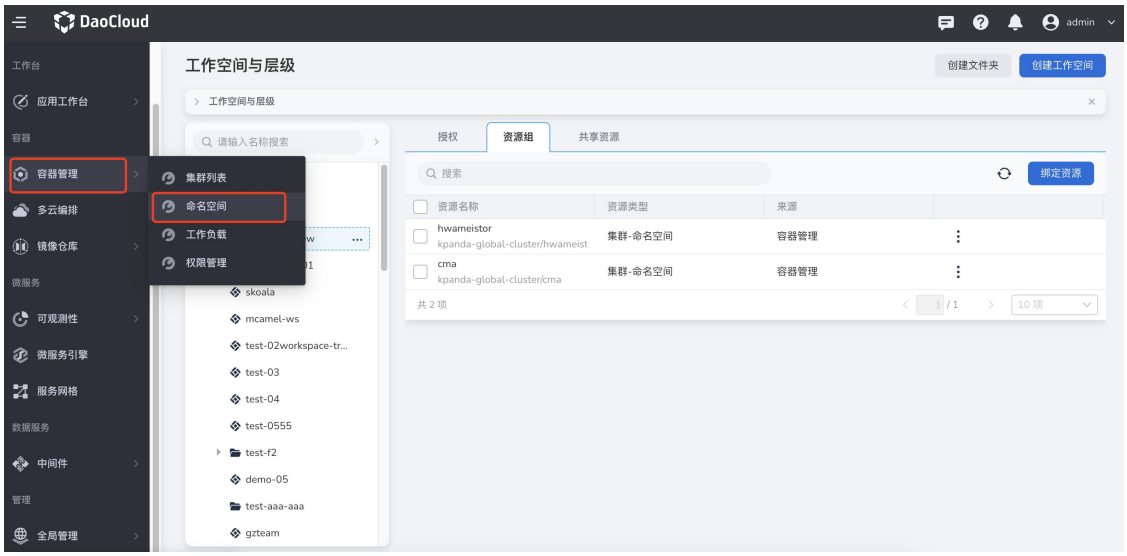
弹出菜单绑定

准备命名空间

命名空间是更小的资源隔离单元，将其绑定到工作空间后，工作空间的成员就可以进行管理和使用。

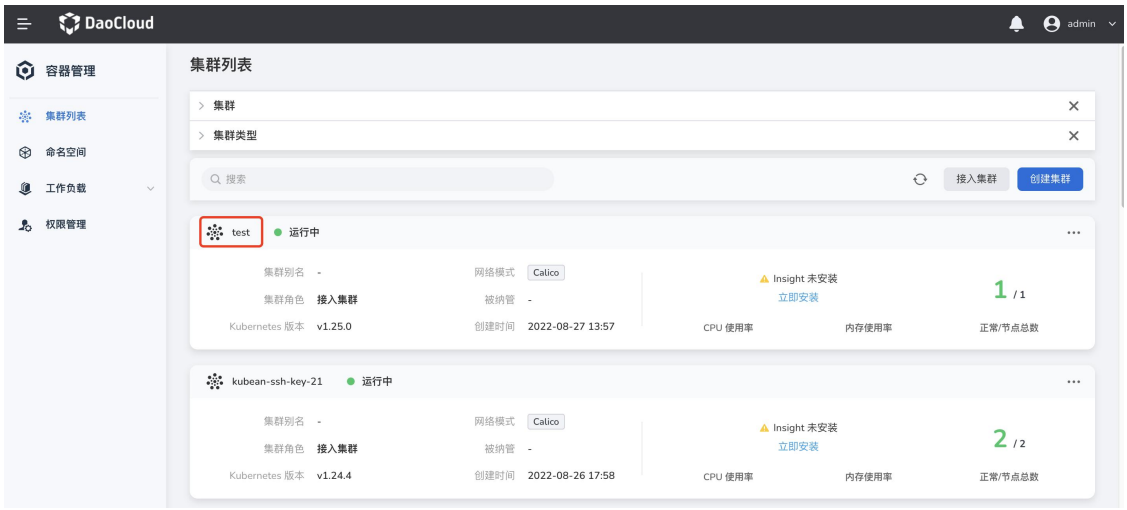
参照以下步骤准备一个还未绑定到任何工作空间的命名空间。

1. 点击左侧导航栏底部的容器管理。



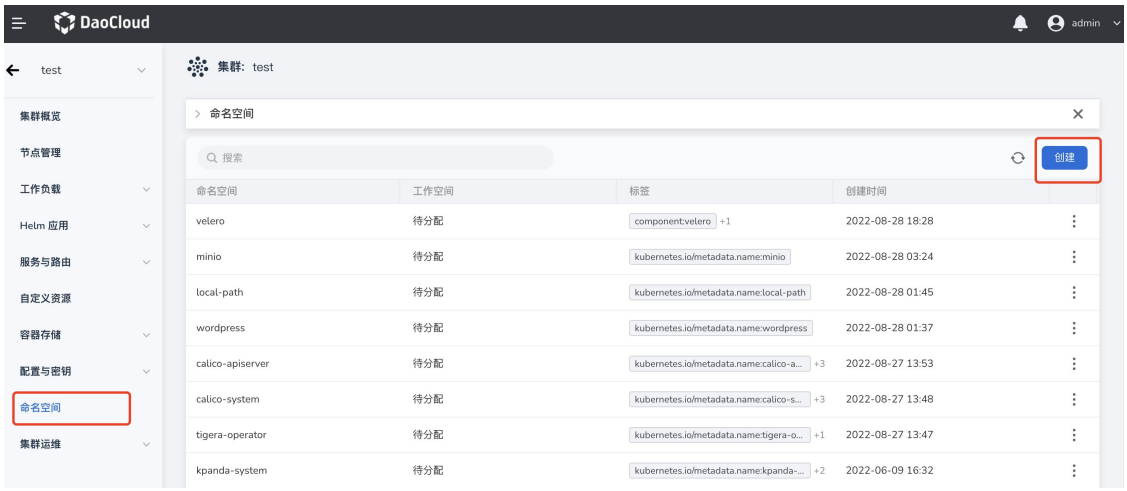
容器管理

2. 点击目标集群的名称，进入集群详情。



集群详情

3. 在左侧导航栏点击命名空间，进入命名空间管理页面，点击页面右侧的创建按钮。

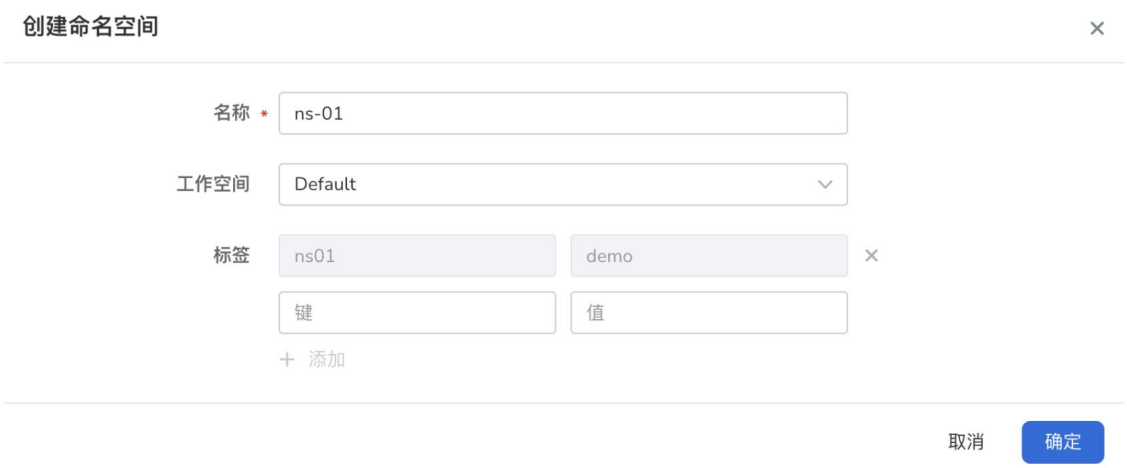


创建命名空间

4. 填写命名空间的名称，配置工作空间和标签（可选设置），然后点击确定。

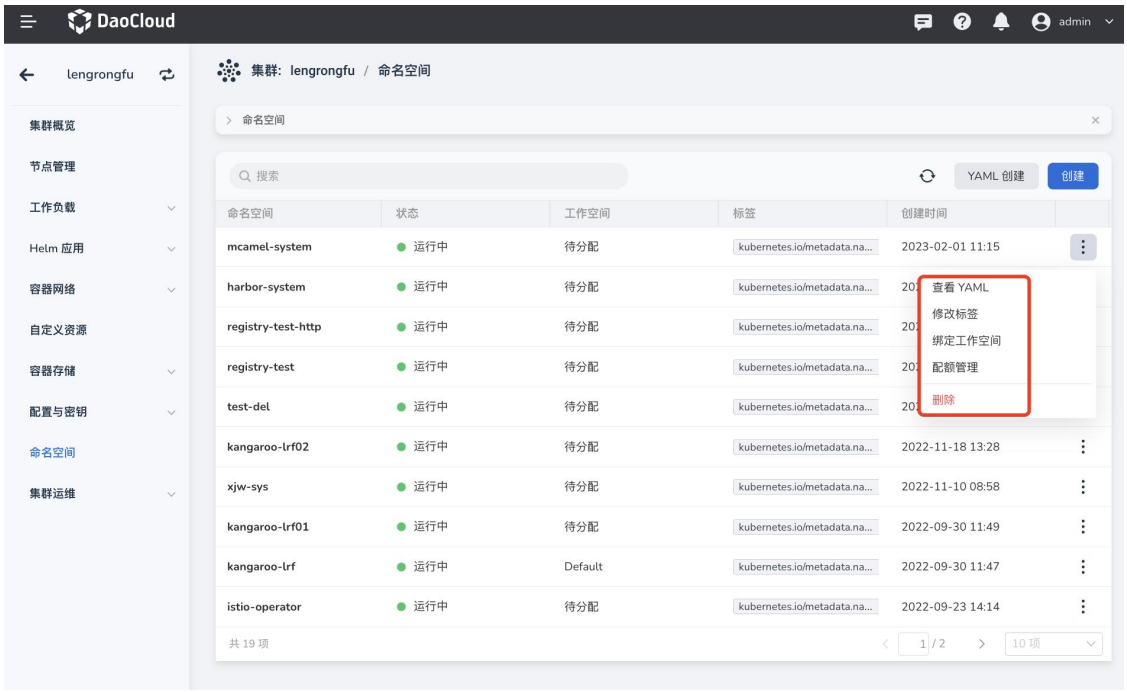
   !!! info

   工作空间主要用于划分资源组并为用户（用户组）授予对该资源的不同访问权限。有关工作空间的详细说明，可参考[工作空间与层级](workspace.md)。



填写

5. 点击确定，完成命名空间的创建。在命名空间列表右侧，点击 ⋮，可以从弹出菜单中选择绑定工作空间。

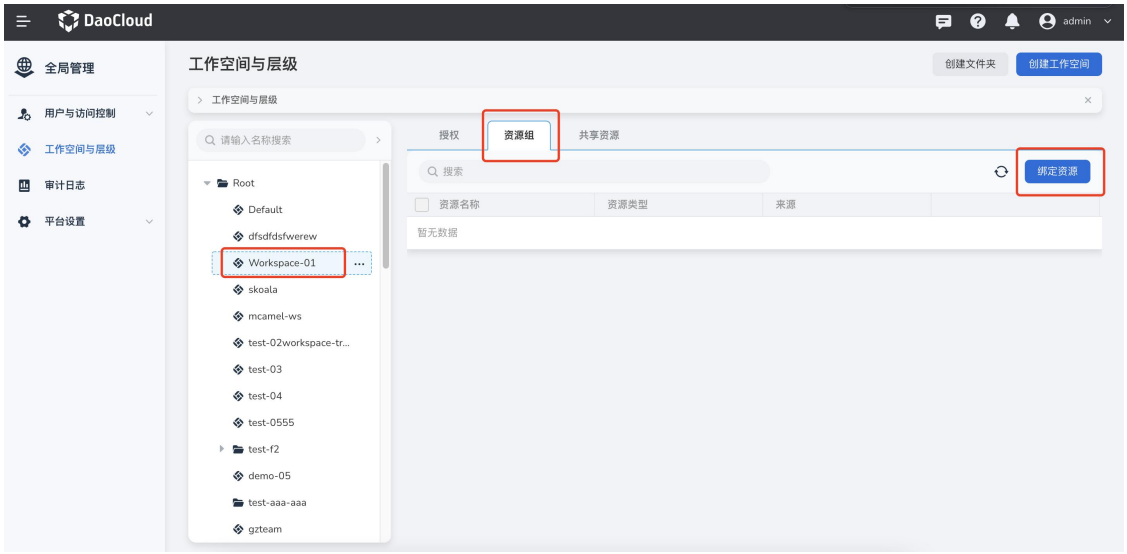


确定和绑定

将命名空间绑定到工作空间

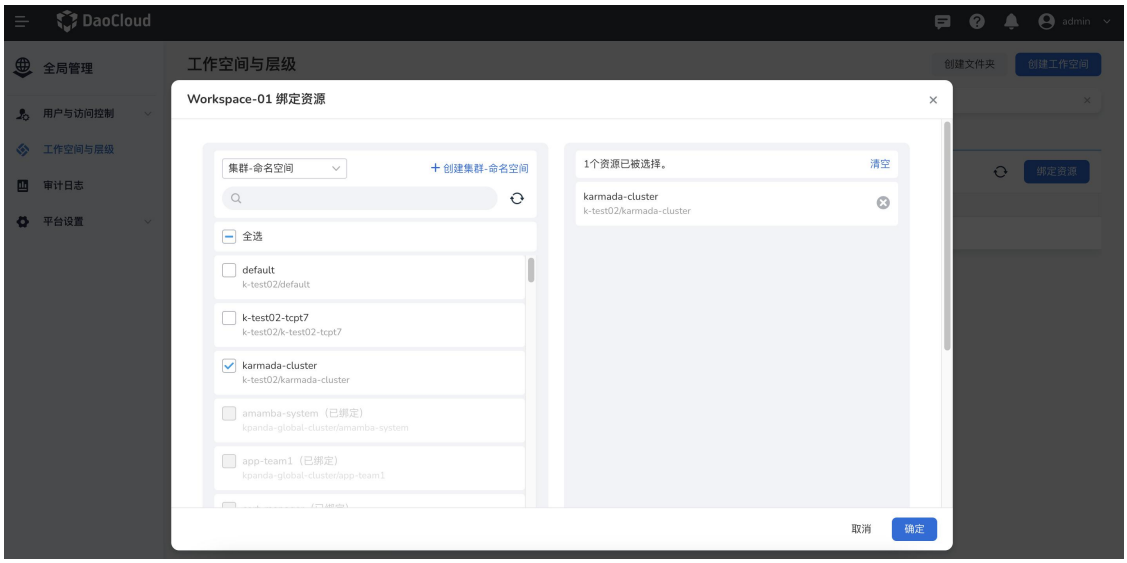
除了在命名空间列表中绑定外，也可以返回全局管理，按照以下步骤绑定工作空间。

1. 依次点击全局管理 -> 工作空间与层级 -> 资源组，点击某个工作空间名称后，点击绑定资源按钮。



绑定资源

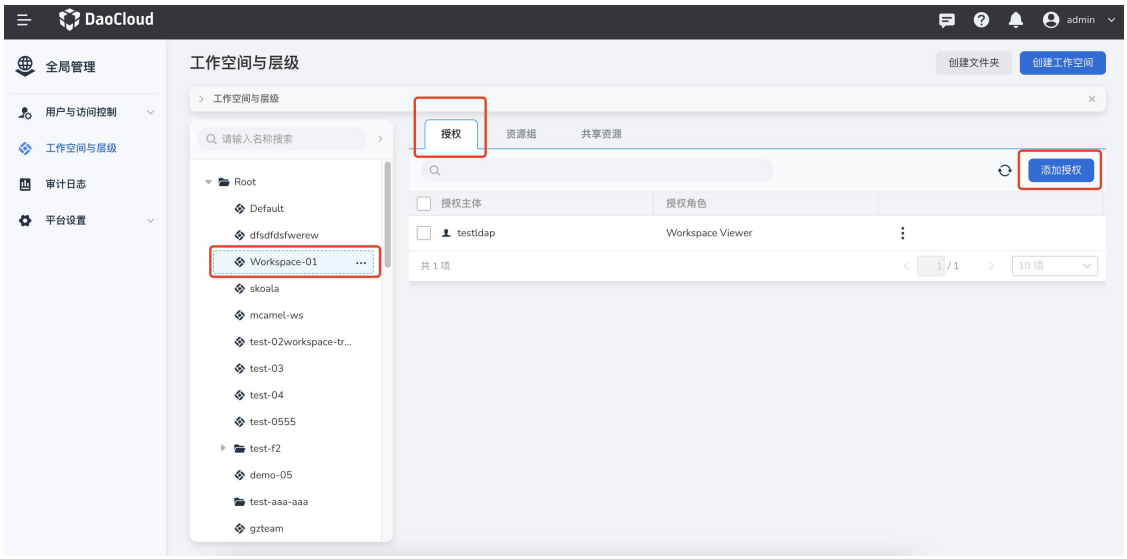
2. 选中要绑定的工作空间（可多选），点击确定完成绑定。



确定

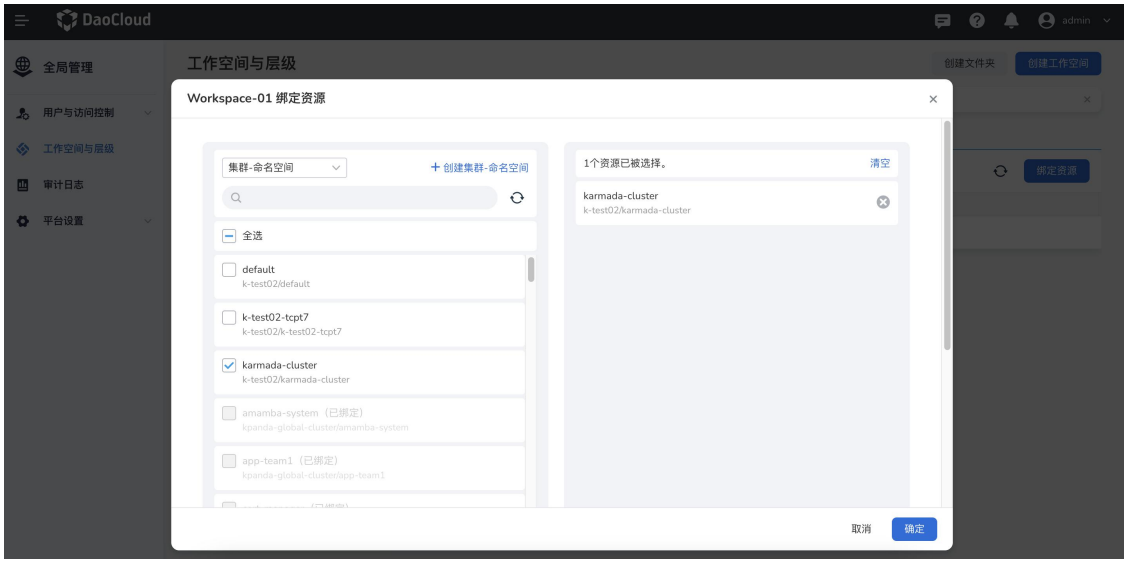
为工作空间添加成员并授权

1. 在 工作空间与层级 -> 授权中，点击某个工作空间名称后，点击添加授权按钮。



添加授权

2. 选择要授权的用户/用户组、角色后，点击确定完成授权。



确定

开启/关闭 K8s 审计日志的输出

默认 Kubernetes 集群不会输出审计日志信息。通过以下配置，可以开启 Kubernetes 的审计日志功能。

!!! note

公有云环境中可能无法控制 kubernetes 审计日志输出及输出路径。

1. 准备审计日志的 Policy 文件
2. 配置 API 服务器，开启审计日志
3. 重启并验证

准备审计日志 Policy 文件

??? note“点击查看审计日志 Policy YAML 文件”

```yaml title="policy.yaml"
apiVersion: audit.k8s.io/v1
kind: Policy
# Don't generate audit events for all requests in RequestReceived stage.
omitStages:
  - "ResponseStarted"
  - "RequestReceived"
  - "Panic"
rules:
  # The following requests were manually identified as high-volume and low-risk,
  # so drop them.
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
```

```
    - group: "" # core
      resources: ["endpoints", "services", "services/status"]
  - level: None
    # Ingress controller reads `configmaps/ingress-uid` through the unsecured p
ort.
    # TODO(#46983): Change this to the ingress controller service account.
    users: ["system:unsecured"]
    namespaces: ["kube-system"]
    verbs: ["get"]
    resources:
    - group: "" # core
      resources: ["configmaps"]
  - level: None
    users: ["kubelet"] # legacy kubelet identity
    verbs: ["get"]
    resources:
    - group: "" # core
      resources: ["nodes", "nodes/status"]
  - level: None
    userGroups: ["system:nodes"]
    verbs: ["get"]
    resources:
    - group: "" # core
      resources: ["nodes", "nodes/status"]
  - level: None
    users:
    - system:kube-controller-manager
    - system:kube-scheduler
    - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
    resources:
```

```
    - group: "" # core
      resources: ["endpoints"]
- level: None
  users: ["system:apiserver"]
  verbs: ["get"]
  resources:
    - group: "" # core
      resources: ["namespaces", "namespaces/status", "namespaces/finalize"]
# Don't log HPA fetching metrics.
- level: None
  users:
    - system:kube-controller-manager
  verbs: ["get", "list"]
  resources:
    - group: "metrics.k8s.io"
# Don't log these read-only URLs.
- level: None
  nonResourceURLs:
    - /healthz*
    - /version
    - /swagger*
# Don't log events requests.
- level: None
  resources:
    - group: "" # core
      resources: ["events"]


# new start
# 忽略所有访问非认证端口的 API，通常是系统组件如 Kube-Controller 等。
- level: None
  users: ["system:unsecured"]
```

```
  # 忽略 kube-admin 的审计日志
  - level: None
    users: ["kube-admin"]
  # 忽略所有资源状态更新的 API need add
  - level: None
    resources:
    - group: "" # core
      resources: ["events", "nodes/status", "pods/status", "services/status"]
    - group: "authorization.k8s.io"
      resources: ["selfsubjectrulesreviews"]
  # 忽略 leases need add
  - level: None
    resources:
    - group: "coordination.k8s.io"
      resources: ["leases"]
  - level: Request
    verbs: ["create", "update", "patch", "delete"]
    users: ["kube-admin"]
  #new end


  # Secrets, ConfigMaps, and TokenReviews can contain sensitive & binary dat
a,
  # so only log at the Metadata level.
  - level: Metadata
    resources:
      - group: "" # core
        resources: ["secrets", "configmaps"]
      - group: authentication.k8s.io
        resources: ["tokenreviews"]
    omitStages:
      - "RequestReceived"
  # Get responses can be large; skip them.
```

```
- level: Request
  verbs: ["get", "list", "watch"]
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apiextensions.k8s.io"
    - group: "apiregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
    - group: "autoscaling"
    - group: "batch"
    - group: "certificates.k8s.io"
    - group: "extensions"
    - group: "metrics.k8s.io"
    - group: "networking.k8s.io"
    - group: "policy"
    - group: "rbac.authorization.k8s.io"
    - group: "settings.k8s.io"
    - group: "storage.k8s.io"
  omitStages:
    - "RequestReceived"
# Default level for known APIs
- level: RequestResponse
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apiextensions.k8s.io"
    - group: "apiregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
```

```
      - group: "autoscaling"
      - group: "batch"
      - group: "certificates.k8s.io"
      - group: "extensions"
      - group: "metrics.k8s.io"
      - group: "networking.k8s.io"
      - group: "policy"
      - group: "rbac.authorization.k8s.io"
      - group: "settings.k8s.io"
      - group: "storage.k8s.io"
    omitStages:
      - "RequestReceived"
  # Default level for all other requests.
  - level: Metadata
    omitStages:
      - "RequestReceived"
```

将以上审计日志文件放到 /etc/kubernetes/audit-policy/ 文件夹下，并取名为 apiserver-audit-policy.yaml。

配置 API 服务器

打开 API 服务器的配置文件 kube-apiserver.yaml，一般会在 /etc/kubernetes/manifests/ 文件夹下，并添加以下配置信息：

这一步操作前请备份 kube-apiserver.yaml，并且备份的文件不能放在 /etc/kubernetes/manifests/ 下，建议放在 /etc/kubernetes/tmp。

1. 在 spec.containers.command 下添加命令：

```
--audit-log-maxage=30
--audit-log-maxbackup=1
--audit-log-maxsize=100
```

```
--audit-log-path=/var/log/audit/kube-apiserver-audit.log
--audit-policy-file=/etc/kubernetes/audit-policy/apiserver-audit-policy.yaml
```

2. 在 spec.containers.volumeMounts 下添加：

```
- mountPath: /var/log/audit
  name: audit-logs
- mountPath: /etc/kubernetes/audit-policy
  name: audit-policy
```

3. 在 spec.volumes 下添加：

```
- hostPath:
  path: /var/log/kubernetes/audit
  type: ""
  name: audit-logs
- hostPath:
  path: /etc/kubernetes/audit-policy
  type: ""
  name: audit-policy
```

测试并验证

稍等一会，API 服务器会自动重启，执行以下命令查看/var/log/kubernetes/audit 目录下是否有审计日志生成，若有，则表示 k8s 审计日志成功开启。

```
ls /var/log/kubernetes/audit
```

如果想关闭，去掉 spec.containers.command 中的相关命令即可。

审计日志

审计日志帮助您监控并记录每个用户的活动，提供了与安全相关的、按时间顺序排列的记录的收集、存储和查询功能。 借助审计日志服务，您可以持续监控并保留用户在全局管理模块的使用行为，包括但不限于创建用户、用户登录/登出、用户授权以及与 Kubernetes 相关的用户操作行为。

功能特性

审计日志功能具有以下特点：

- 开箱即用：在安装使用该平台时，审计日志功能将会被默认启用，自动记录与用户相关的各种行为， 如创建用户、授权、登录/登出等。默认可以在平台内查看 365 天的用户行为。
- 安全分析：审计日志会对用户操作进行详细的记录并提供导出功能，通过这些事件您可以判断账号是否存在风险。
- 实时记录：迅速收集操作事件，用户操作后可在审计日志列表进行追溯，随时发现可疑行为。
- 方便可靠：审计日志支持手动清理和自动清理两种方式，可根据您的存储大小配置清理策略。

查看审计日志

1. 使用具有 admin 或 Audit Owner 角色的用户登录 DCE 5.0。



登录 DCE 5.0

2. 在左侧导航栏底部，点击全局管理 -> 审计日志。

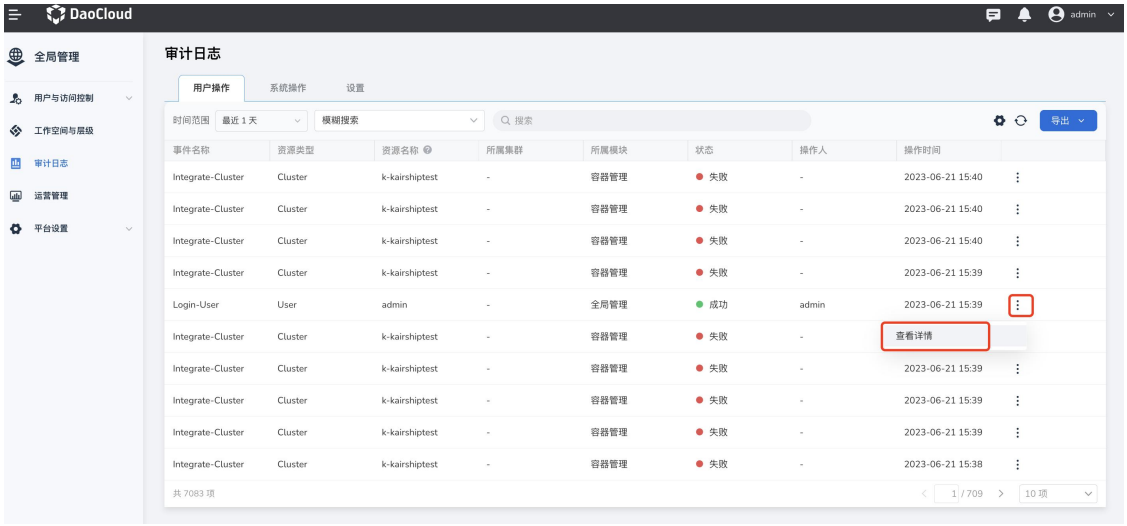


审计日志

用户操作

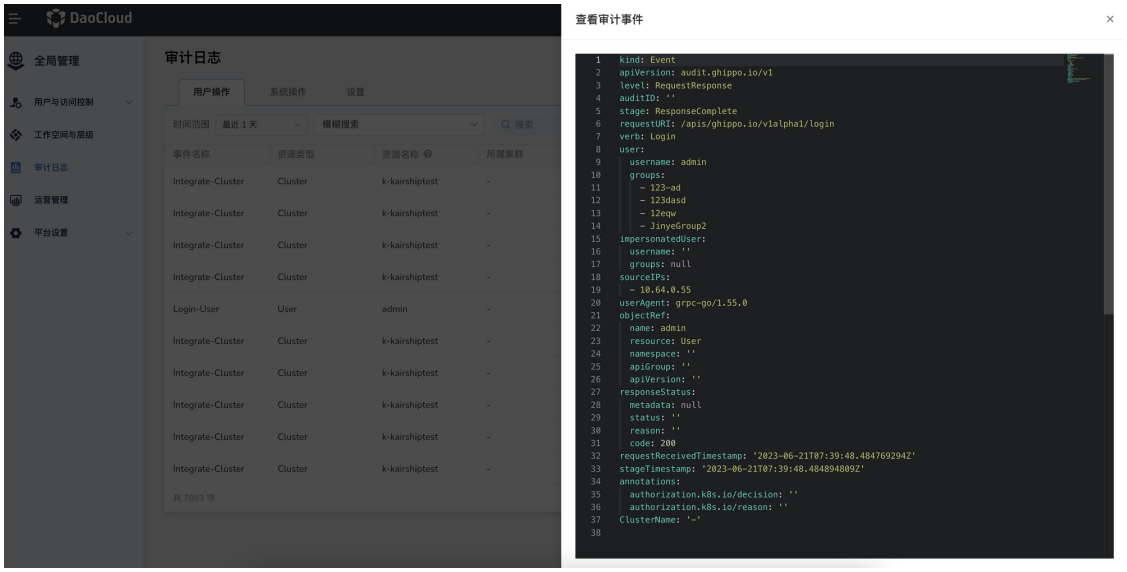
在用户操作页签中，可以按时间范围，也可以通过模糊搜索、精确搜索来查找用户操作事件。

点击某个事件最右侧的 ⋮，可以查看事件详情。



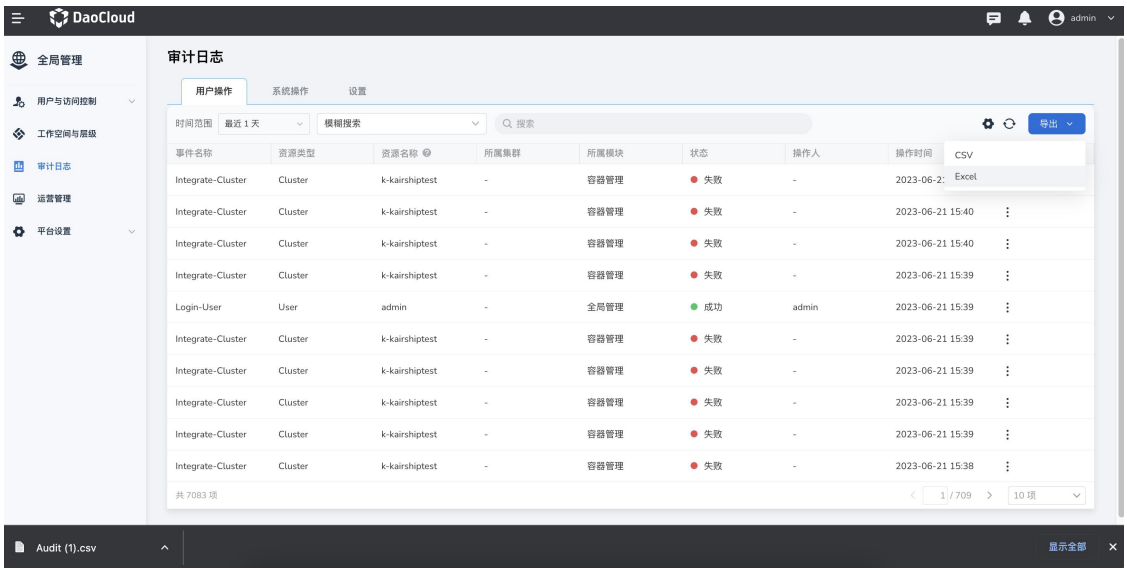
用户审计日志

事件详情如下图所示。



用户事件详情

点击右上角的导出按钮，可以按 CSV 和 Excel 格式导出当前所选时间范围内的用户操作日志。

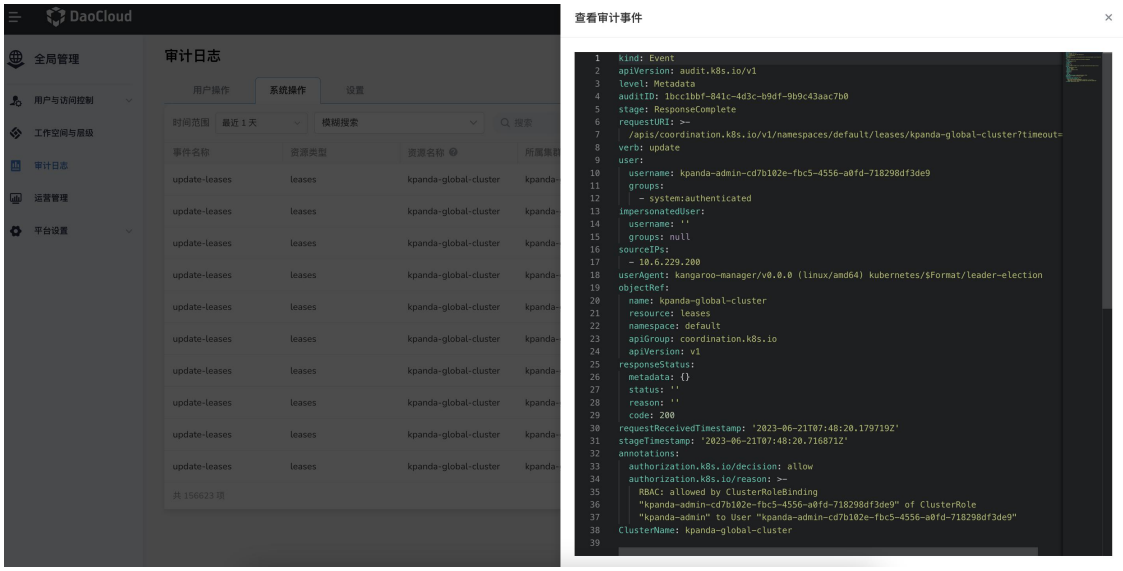


导出

系统操作

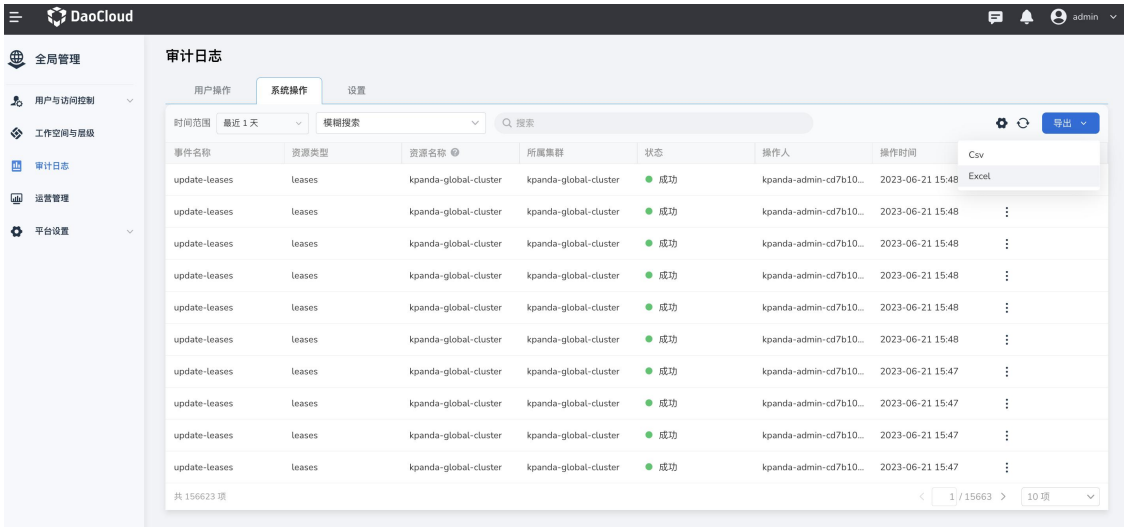
在系统操作页签中，可以按时间范围，也可以通过模糊搜索、精确搜索来查找系统操作事件。

同样点击某个事件最右侧的 ⋮，可以查看事件详情。



系统事件详情

点击右上角的导出按钮，可以按 CSV 和 Excel 格式导出当前所选时间范围内的系统操作日志。



导出

设置

在设置页签中，您可以清理用户操作和系统操作的审计日志。



清理

可以手动清理，建议清理前先导出并保存。也可以设置日志的最长保存时间实现自动清理。

!!! note

审计日志中与 Kubernetes 相关的日志记录由可观测性模块提供，为减轻审计日志的存储压力，全局管理默认不采集 Kubernetes 相关日志。
如需记录请参阅开启 [K8s 审计日志](./open-k8s-audit.md)。开启后的清理功能与全局管理的清理功能一致，但互不影响。

开启/关闭 K8s 审计日志的采集

- K8s 审计日志：K8s 本身生成审计日志，开启该功能后，会在指定目录下生成 K8s 审计日志的日志文件
- 采集 K8s 审计日志：通过 insight-agent 采集上述 'K8s 审计日志'的日志文件，'采集 K8s 审计日志' 的前提条件是集群开启了 'K8s 审计日志'

DCE 5.0 安装完成时状态

- 社区版安装管理集群过程中未操作 K8s 审计日志开关
- 商业版管理集群的 K8s 审计日志开关默认开启
  - 如需设置成默认关闭，可修改安装器 clusterConfigt.yaml 来配置（logPath 设置为空 ""）
- 管理集群的采集 K8s 审计日志开关默认关闭
  - 默认设置不支持配置

管理集群采集 K8s 审计日志开关

商业版安装环境

确认开启 K8s 审计日志

执行以下命令查看/var/log/kubernetes/audit 目录下是否有审计日志生成。 若有，则表示 K8s 审计日志成功开启。

```
ls /var/log/kubernetes/audit
```

若未开启，请参考文档的开启关闭 K8s 审计日志。

开启采集 K8s 审计日志流程

1. 添加 chartmuseum 到 helm repo 中

   ```
   helm repo add chartmuseum http://10.5.14.30:8081
   ```

   这条命令中的 IP 需要修改为火种节点的 IP 地址。

!!! note

使用自建 Harbor 仓库的情况下，请修改第一步中的 chart repo 地址为自建仓库的 insight-agent chart 地址。

2. 保存当前 insight-agent helm value

   helm get values insight-agent -n insight-system -o yaml > insight-agent-values-bak.yaml

3. 获取当前版本号 ${insight_version_code}

   insight_version_code=`helm list -n insight-system |grep insight-agent | awk {'print $10'}`

4. 更新 helm value 配置

   helm upgrade --install --create-namespace --version ${insight_version_code} --cleanup-on-fail insight-agent chartmuseum/insight-agent -n insight-system -f insight-agent-values-bak.yaml --set global.exporters.auditLog.kubeAudit.enabled=true

5. 重启 insight-system 下的所有 fluentBit pod

   fluent_pod=`kubectl get pod -n insight-system | grep insight-agent-fluent-bit | awk {'print $1'} | xargs`
   kubectl delete pod ${fluent_pod} -n insight-system

关闭采集 K8s 审计日志

其余步骤和开启采集 K8s 审计日志一致，仅需修改上一节中第 4 步：更新 helm value 配置。

helm upgrade --install --create-namespace --version ${insight_version_code} --cleanup-on-fail insight-agent chartmuseum/insight-agent -n insight-system -f insight-agent-values-bak.yaml --set global.exporters.auditLog.kubeAudit.enabled=false

社区版在线安装环境

!!! note

若在 Kind 集群中安装社区版 DCE5.0 社区版，需要在 Kind 容器内进行以下操作。

确认开启 K8s 审计日志

执行以下命令查看/var/log/kubernetes/audit 目录下是否有审计日志生成，若有，则表示 K8s 审计日志成功开启。

ls /var/log/kubernetes/audit

若未开启，请参考文档的开启关闭 K8s 审计日志。

开启采集 K8s 审计日志流程

1. 保存当前 value

   helm get values insight-agent -n insight-system -o yaml > insight-agent -values-bak.yaml

2. 获取当前版本号 ${insight_version_code}，然后更新配置

   insight_version_code=`helm list -n insight-system |grep insight-agent | aw k {'print $10'}`

3. 更新 helm value 配置

   helm upgrade --install --create-namespace --version ${insight_version_c ode} --cleanup-on-fail insight-agent insight-release/insight-agent -n ins ight-system -f insight-agent-values-bak.yaml --set global.exporters.audit Log.kubeAudit.enabled=true

   如果因为版本未找到而升级失败，请检查命令中使用的 helm repo 是否有这个版本。 若没有，请尝试更新 helm repo 后重试。

   helm repo update insight-release

4. 重启 insight-system 下的所有 fluentBit pod

```
fluent_pod=`kubectl get pod -n insight-system | grep insight-agent-fluent-bit | awk {'print $1'} | xargs`
kubectl delete pod ${fluent_pod} -n insight-system
```

关闭采集 K8s 审计日志

其余步骤和开启采集 K8s 审计日志一致，仅需修改上一节中第 3 步： 更新 helm value 配置

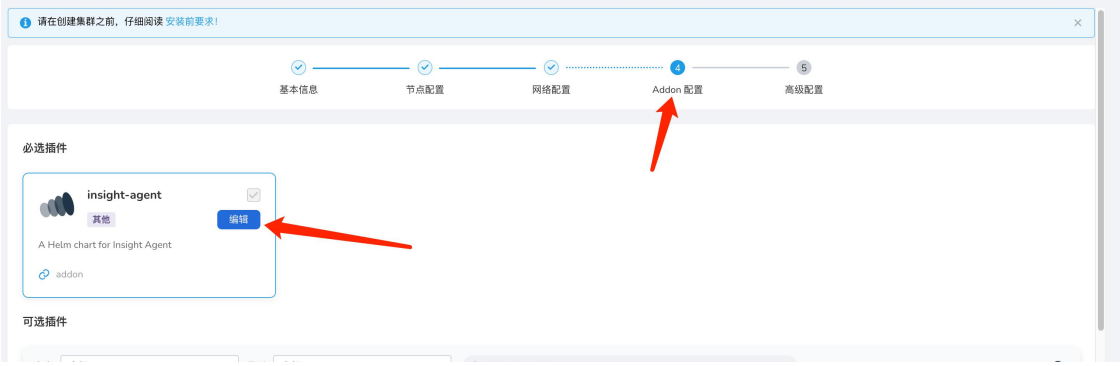
```
helm upgrade --install --create-namespace --version ${insight_version_code} --cleanup-on-fail insight-agent insight-release/insight-agent -n insight-system -f insight-agent-values-bak.yaml --set global.exporters.auditLog.kubeAudit.enabled=false
```

工作集群开关

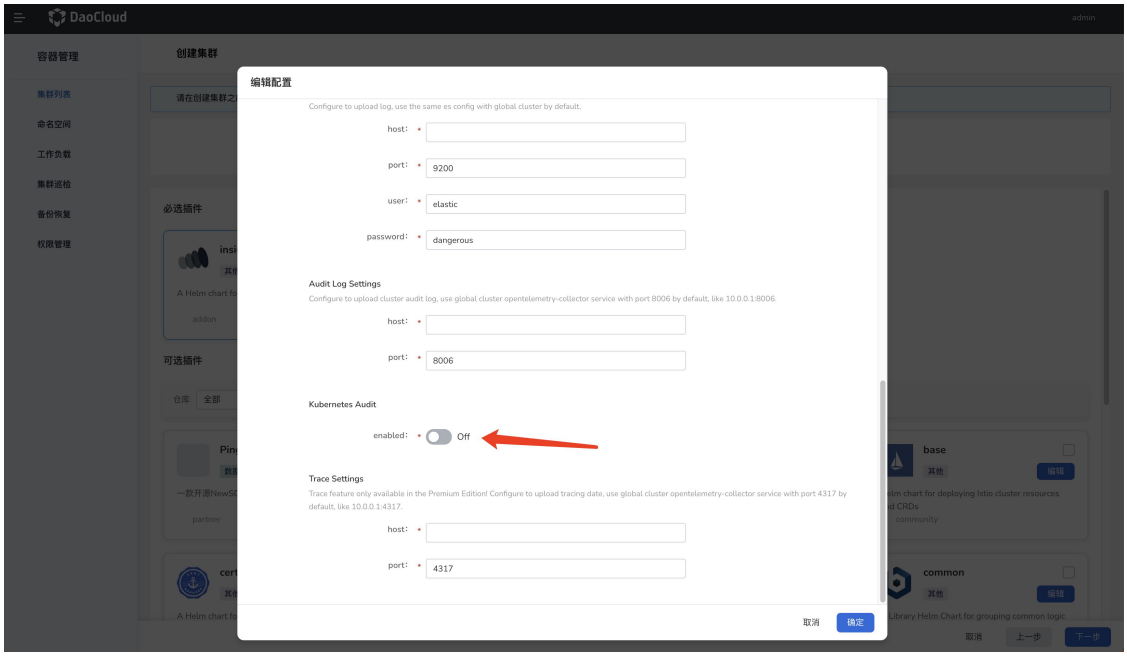
各工作集群开关独立，按需开启。

创建集群时打开采集审计日志步骤

采集 K8s 审计日志功能默认为关闭状态。若需要开启，可以按照如下步骤：



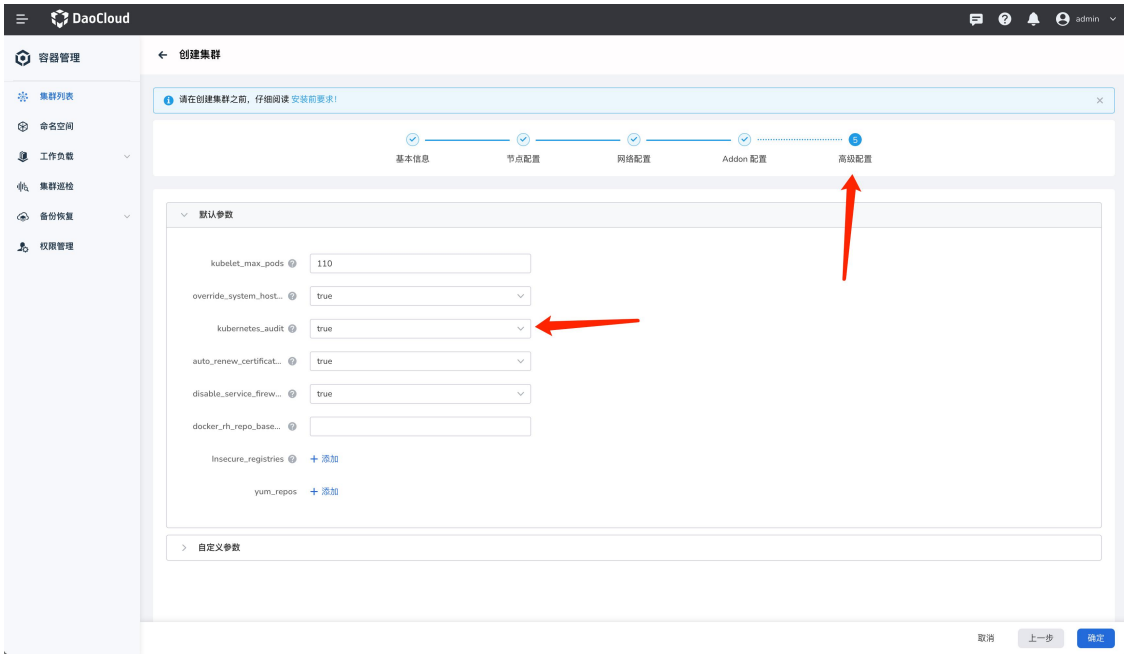
默认关闭



开启审计日志

将该按钮设置为启用状态，开启采集 K8s 审计日志功能。

通过 DCE 5.0 创建工作集群时，确认该集群的 K8s 审计日志选择‘true’，这样创建出来的工作集群 K8s 审计日志是开启的。



审计日志开启

等待集群创建成功后，该工作集群的 K8s 审计日志将被采集。

接入的集群和创建完成后开关步骤

确认开启 K8s 审计日志

执行以下命令查看/var/log/kubernetes/audit 目录下是否有审计日志生成，若有，则表示 K8s 审计日志成功开启。

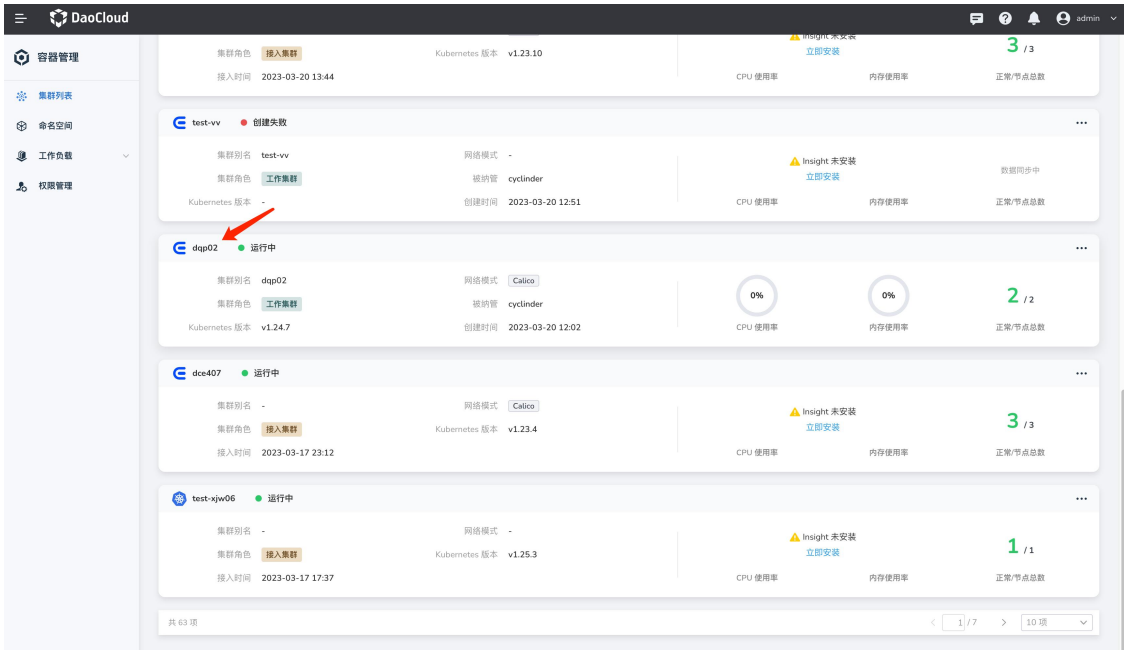
ls /var/log/kubernetes/audit

若未开启，请参考文档的开启关闭 K8s 审计日志

开启采集 K8s 审计日志

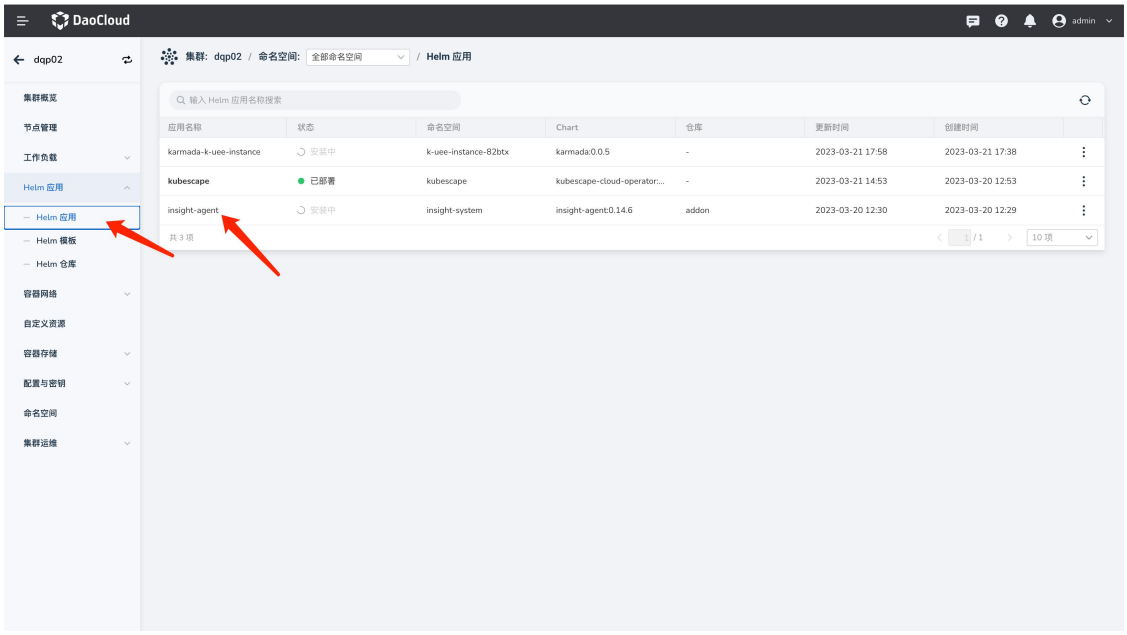
采集 K8s 审计日志功能默认为关闭状态，若需要开启，可以按照如下步骤：

1. 选中已接入并且需要开启采集 K8s 审计日志功能的集群



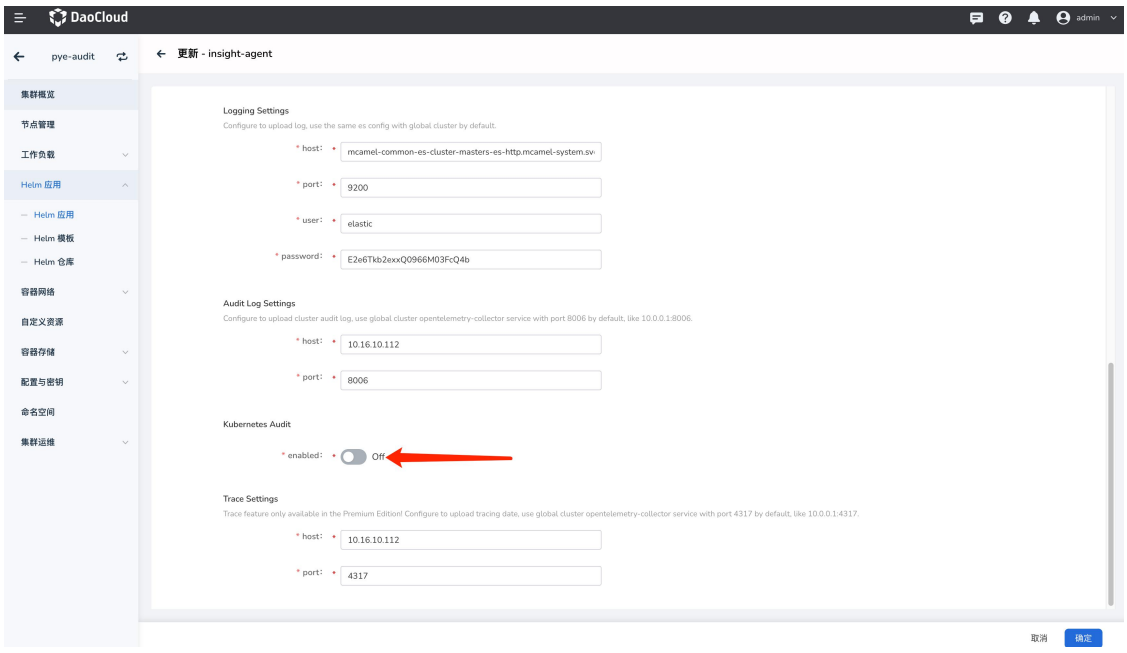
选中集群

2. 进入 helm 应用管理页面，更新 insight-agent 配置 （若未安装 insight-agent，可以安装 insight-agent）



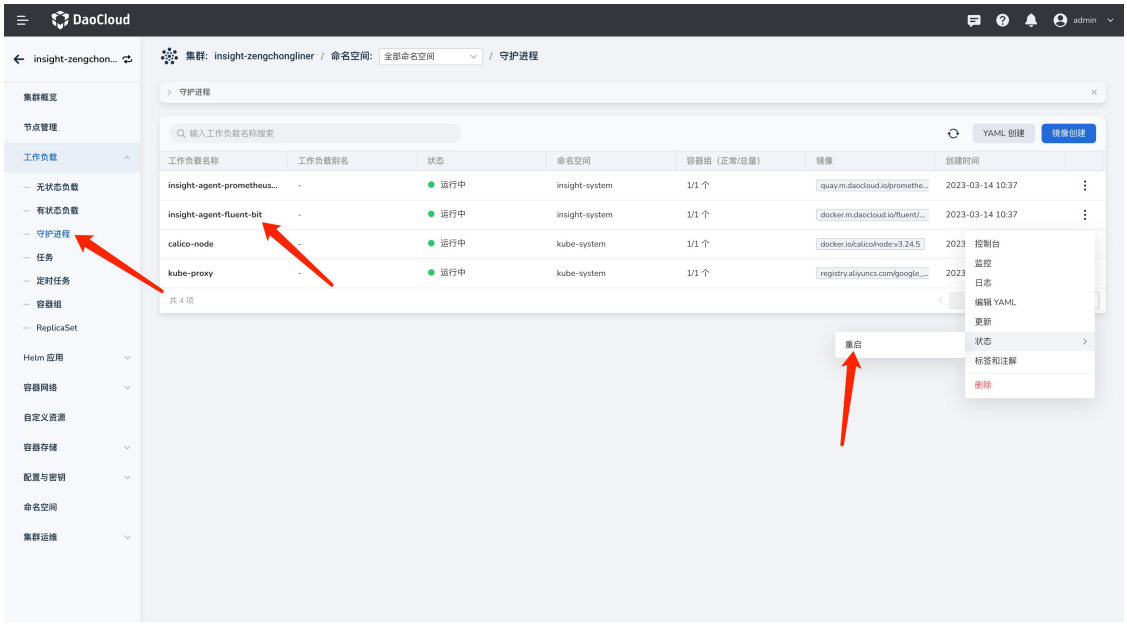
进入 Helm 应用页面

3. 开启/关闭采集 K8s 审计日志按钮



开启/关闭按钮

4. 接入集群的情况下开关后仍需要重启 fluent-bit pod 才能生效


DaoCloud
admin
insight-zengchon...
集群概览
节点管理
工作负载
无状态负载
有状态负载
守护进程
任务
定时任务
容器组
ReplicaSet
Helm 应用
容器网络
自定义资源
容器存储
配置与密钥
命名空间
集群运维
集群: insight-zengchongliner / 命名空间: 全部命名空间 / 守护进程
守护进程
输入工作负载名称搜索
YAML 创建
镜像创建
工作负载名称
工作负载别名
状态
命名空间
容器组（正常/总量）
镜像
创建时间
insight-agent-prometheus...
运行中
insight-system
1/1 个
2023-03-14 10:37
insight-agent-fluent-bit
运行中
insight-system
1/1 个
2023-03-14 10:37
calico-node
运行中
kube-system
1/1 个
docker.io/calico/node:v3.24.5
kube-proxy
运行中
kube-system
1/1 个
registry.aliyuncs.com/google_...
共 4 项
控制台
监控
日志
编辑 YAML
更新
状态
标签和注解
删除
重启

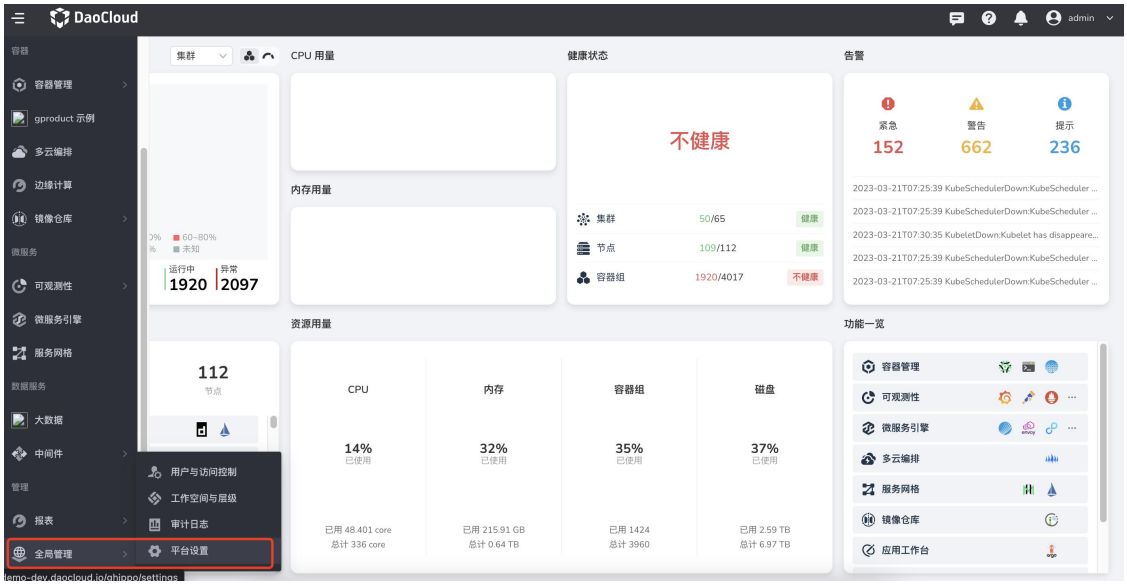
重启

邮件服务器

DCE 5.0 会在用户忘记密码时，向用户发送电子邮件以验证电子邮件地址，确保用户是本人操作。 要使 DCE 5.0 能够发送电子邮件，需要先提供您的邮件服务器地址。

具体操作步骤如下：

1. 使用具有 admin 角色的用户登录 DCE 5.0。点击左侧导航栏底部的全局管理。



全局管理

2. 点击平台设置，选择邮件服务器设置。



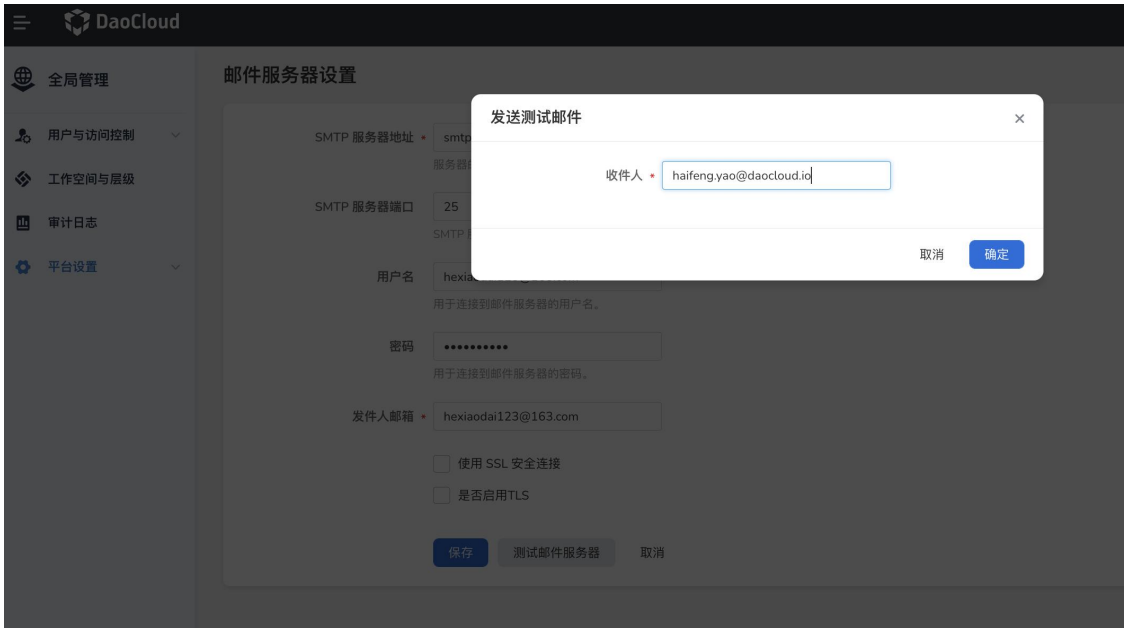
邮件服务器

填写以下字段配置邮件服务器：

字段	描述	举例值
SMTP 服务器地址	能够提供邮件服务的 SMTP 服务器地址	smtp.163.com
SMTP 服务器端口	发送邮件的端口	25
用户名	SMTP 用户的名称	test@163.com
密码	SMTP 账号的密码	123456
发件人邮箱	发件人的邮箱地址	test@163.com

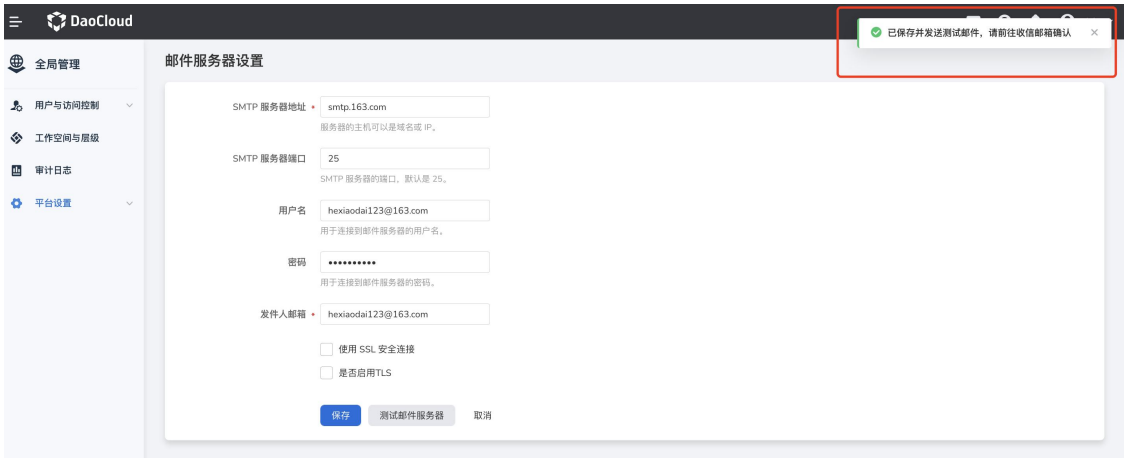
字段	描述	举例值
使用 SSL 安全连接	SSL 可以用于加密邮件，从而提高通过邮件传输的信息的安全性，通常需为邮件服务器配置证书	不开启

3. 配置完成后点击保存，点击测试邮件服务器。



测试

4. 屏幕右上角出现成功发送邮件的提示，则表示邮件服务器被成功设置。



成功

常见问题

问：邮件服务器设置后用户仍无法找回密码是什么原因?

答：用户可能未设置邮箱或者设置了错误的邮箱地址；此时可以让 admin 角色的用户在全局管理 -> 用户与访问控制中通过用户名找到该用户，并在用户详情中为该用户设置新的登录密码。

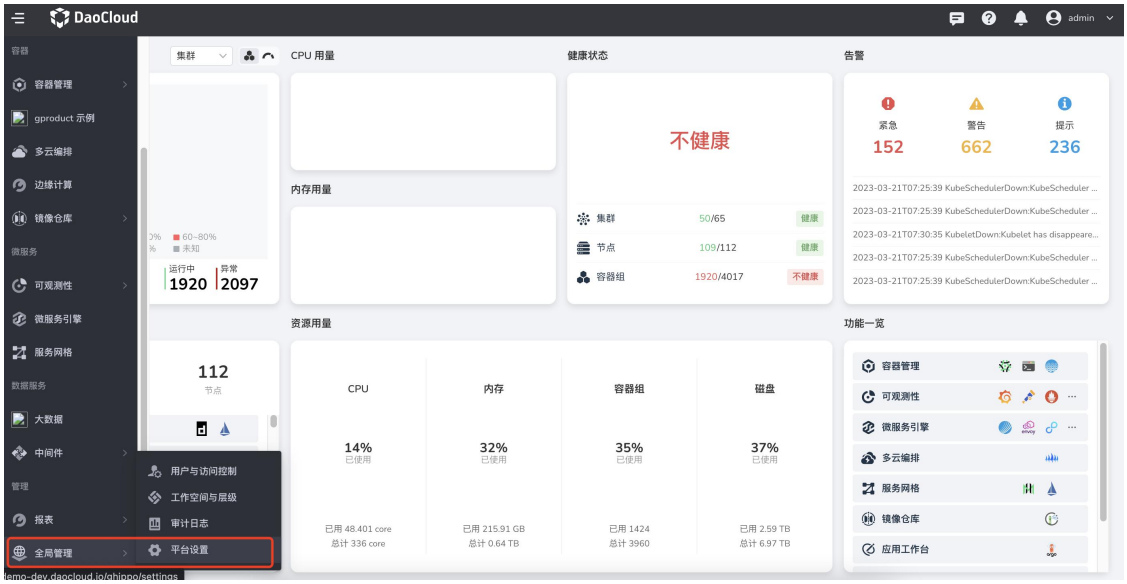
如果邮件服务器没有连通，请检查邮件服务器地址、用户名及密码是否正确。

外观定制

在 DCE 5.0 中，可通过外观定制更换登录界面、顶部导航栏以及底部版权和备案信息，帮助用户更好地辨识产品。

定制说明

1. 使用具有 admin 角色的用户登录 DCE 5.0。点击左侧导航栏底部的全局管理 –> 平台设置。



全局管理

2. 选择外观定制。



外观定制

3. 在登录页定制页签中，修改登录页的图标和文字后，点击保存。



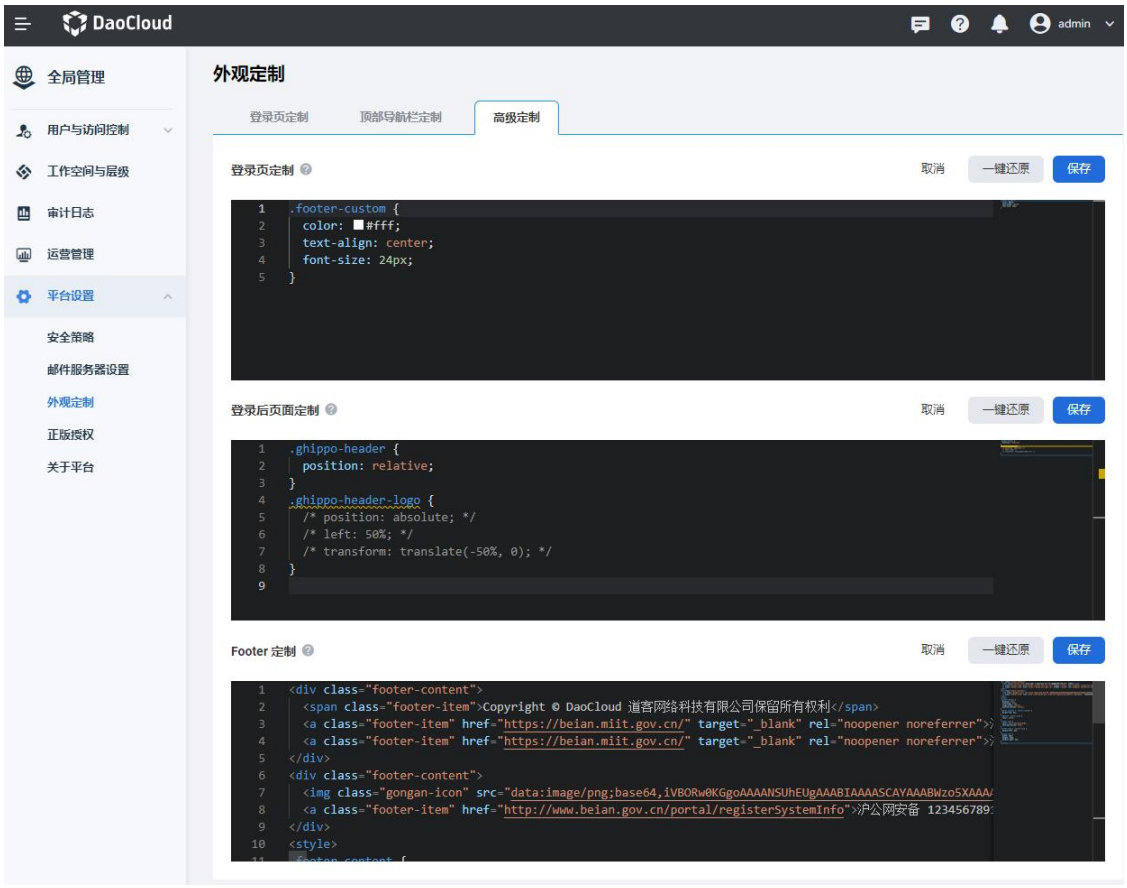
登录页

4. 点击顶部导航栏定制页签，修改导航栏的图标和文字后，点击保存。



顶部导航栏

5. 点击高级定制，可以用 CSS 样式设置登录页、导航栏、底部版权及备案信息。



高级定制

高级定制

高级定制能够通过 CSS 样式来修改整个容器平台的颜色、字体间隔、字号等。您需要熟悉 CSS 语法。删除黑色输入框的内容，可恢复到默认状态，当然也可以点击一键还原按钮。

登录页定制的 CSS 样例：

```
.test {
  width: 12px;
}

#kc-login {
```

```
  /* color: red!important; */
}
```

登录后页面定制的 CSS 样例：

```
.dao-icon.dao-iconfont.icon-service-global.dao-nav__head-icon {
    color: red!important;
}
.ghippo-header-logo {
  background-color: green!important;
}
.ghippo-header {
  background-color: rgb(128, 115, 0)!important;
}
.ghippo-header-nav-main {
  background-color: rgb(0, 19, 128)!important;
}
.ghippo-header-sub-nav-main .dao-popper-inner {
  background-color: rgb(231, 82, 13) !important;
}
```

Footer（页面底部的版权、备案等信息）定制示例

```
<div class="footer-content">
  <span class="footer-item">Copyright © DaoCloud 道客网络科技有限公司保留所有权利</span>
  <a class="footer-item" href="https://beian.miit.gov.cn/" target="_blank" rel="noopener noreferrer">沪 ICP 备 14048409 号 - 1</a>
  <a class="footer-item" href="https://beian.miit.gov.cn/" target="_blank" rel="noopener noreferrer">沪 ICP 备 14048409 号 - 2</a>
</div>
<div class="footer-content">
  <img class="gongan-icon" src="
```

WlDQ1BQaG90b3Nob3AgSUNDlHByb2ZpbGUAAHjanVN3WJP3Fj7f92UPVkLY8LGXb
lEAliOsCMgQWalQkgBhhBASQMWFiApWFBURnEhVxlLVCkidiOKgKLhnQYqlWotVXD
juH9yntX167+3t+9f7vOec5/zOec8PgBESJpHmomoAOVKFPDrYH49PSMTJvYACFUj
gBCAQ5svCZwXFAADwA3l4fnSwP/wBr28AAgBw1S4kEsfh/4O6UCZXACCRAOAiEuc
LAZBSAMguVMgUAMgYALBTs2QKAJQAAGx5fEliAKoNAOz0ST4FANipk9wXANiiHK
klAl0BAJkoRyQCQLsAYFWBUiwCwMlAoKxAli4EwK4BgFm2MkcCgL0FAHaOWJAPQ
GAAgJlCLMwAlDgCAEMeE80DlEwDoDDSv+CpX3CFuEgBAMDLlc2XS9lzFLiV0Bp38
vDg4iHiwmyxQmEXKRBmCeQinJeblxNl5wNMzgwAABr50cH+OD+Q5+bk4eZm52zv9
MWi/mvwbyl+lfHf/ryMAgQAEE7P79pf5eXWA3DHAbB1v2upWwDaVgBo3/ldM9sJoF
oK0Hr5i3k4/EAenqFQyDwdHAoLC+0lYqG9MOOLPv8z4W/gi372/EAe/tt68ABxmkC
ZrcCjg/1xYW52rlKO58sEQjFu9+cj/seFf/2OKdHiNLFcLBWK8ViJuFAiTcd5uVKRRCH
JleIS6X8y8R+W/QmTdw0ArlZPwE62B7XLbMB+7gECiw5Y0nYAQH7zLYwaC5EAEGc
0Mnn3AACTv/mPQCsBAM2XpOMAALzoGFyolBdMxggAAESggSqwQQcMwRSswA6
cwR28wBcCYQZEQAwkwDwQQgbkgBwKoRiWQRlUwDrYBLWwAxqgEZrhELTBMTg
N5+ASXlHrcBcGYBiewhi8hgkEQcglE2EhOogRYo7Yls4lF5mOBCJhSDSSgKQg6YgU
USLFyHKkAqlCapFdSCPyLXlUOY1cQPqQ28ggMor8irxHMZSBslED1AJ1QLmoHxqKxq
Bz0XQ0D12AlqJr0Rq0Hj2AtqKn0UvodXQAfYqOY4DRMQ5mjNlhXlyHRWCJWBomxx
Zj5Vg1Vo81Yx1YN3YVG8CeYe8lJAKLgBPsCF6EEMJsgpCQR1hMWEOoJewjtBK6CFc
Jg4QxwicIk6hPtCV6EvnEeGl6sZBYRqwm7iEelZ4lXicOE1+TSCQOyZLkTgohJZAySQt
Ja0jbSC2kU6Q+0hBpnEwm65Btyd7kCLKArCCXkbeQD5BPkvvJw+S3FDrFiOJMCalk
UqSUEko1ZT/lBKWfMkKZoKpRzame1AiqiDqfWkltoHZQL1OHqRM0dZolzZsWQ8ukLa
PV0JppZ2n3aC/pdLoJ3YMeRZfQl9Jr6Afp5+mD9HcMDYYNg8dlYigZaxl7GacYtxkvm
UymBdOXmchUMNcyG5lnmA+Yb1VYKvYqfBWRyhKVOpVWlX6V56pUVXNVP9V5qgt
Uq1UPq15WfaZGVbNQ46kJ1Bar1akdVbupNq7OUndSj1DPUV+jvl/9gvpjDbKGhUaghkij
VGO3xhmNlRbGMmXxWELWclYD6yxrmE1iW7L57Ex2Bfsbdi97TFNDc6pmrGaRZp3m
cc0BDsax4PA52ZxKziHODc57LQMtPy2x1mqtZq1+rTfaetq+2mLtcu0W7eva73VwnUC
dLJ31Om0693UJuja6UbqFutt1z+o+02PreekJ9cr1Dund0Uf1bfSj9Rfq79bv0R83MDQlN
pAZbDE4Y/DMkGPoa5hpuNHwhOGoEctoupHEaKPRSaMnuCbuh2fjNXgXPmasbxxir
DTeZdxrPGFiaTLbpMSkxeS+Kc2Ua5pmutG003TMzMgs3KzYrMnsjjnVnGueYb7ZvNv
8jYWlRZzFSos2i8eW2pZ8ywWWTZb3rJhWPlZ5VvVW16xJ1lzrLOtt1ldsUBtXmwybOpv
Ltqitm63Edptt3xTiFl8p0in1U27aMez87ArsmuwG7Tn2YfYl9m32zx3MHBld1jt0O3xyd
HXMdmxwvOuk4TTDqcSpw+lXZxtnoXOd8zUXpkuQyxKXdpcXU22niqdun3rLleUa7rrS

tdP1o5u7m9yt2W3U3cw9xX2r+00umxvJXcM970H08PdY4nHM452nm6fC85DnL152
Xlle+70eT7OcJp7WMG3I28Rb4L3Le2A6Pj1l+s7pAz7GPgKfep+Hvqa+It89viN+1n6Zfg
f8nvs7+sv9j/i/4XnyFvFOBWABwQHlAb2BGoGzA2sDHwSZBKUHNQWNBbsGLww+F
UlMCQ1ZH3KTb8AX8hv5YzPcZyya0RXKCJ0VWhv6MMwmTB7WEY6GzwjfEH5vpvlM
6cy2CljgR2yluB9pGZkX+X0UKSoyqi7qUbRTdHF09yzWrORZ+2e9jvGPqYy5O9tqtnJ2
Z6xqbFJsY+ybulC4qriBelf4RfGXEnQTJAntieTE2MQ9ieNzAudsmjOc5JpUlnRjruXcork
X5unOy553PFk1WZB8OlWYEpeyP+WDlEJQLxhP5aduTR0T8oSbhU9FvqKNolGxt7hK
PJLmnVaV9jjdO31D+miGT0Z1xjMJT1lreZEZkrkj801WRNberM/ZcdktOZSclJyjUg1plrQ
r1zC3KLdPZisrkw3keeZtyhuTh8r35CP5c/PbFWyFTNGjtFKuUA4WTC+oK3hbGFt4uEi
9SFrUM99m/ur5lwuCFny9kLBQuLCz2Lh4WfHglr9FuxYji1MXdy4xXVK6ZHhp8NJ9y2j
Lspb9UOJYUlXyannc8o5Sg9KlpUMrglc0lamUycturvRauWMVYZVkVe9ql9VbVn8qF5
VfrHCsqK74sEa45uJXTl/VfPV5bdra3kq3yu3rSOuk626s91m/r0q9akHV0lbwDa0b8Y
3lG19tSt50oXpq9Y7NtM3KzQM1YTXtW8y2rNvyoTaj9nqdf13LVv2tq7e+2Sba1r/dd3vz
DoMdFTve75TsvLUreFdrvUV99W7S7oLdjxpiG7q/5n7duEd3T8Wej3ulewf2Re/ranRvb
Nyvv7+yCW1SNo0eSDpw5ZuAb9qb7Zp3tXBaKg7CQeXBJ9+mfHvjUOihzsPcw83fmX
+39QjrSHkr0jq/dawto22gPaG97+iMo50dXh1Hvrf/fu8x42N1xzWPV56gnSg98fnkgpPj
p2Snnp1OPz3Umdx590z8mWtdUV29Z0PPnj8XdO5Mt1/3yfPe549d8Lxw9CL3Ytslt0u
tPa49R35w/eFlr1tv62X3y+1XPK509E3rO9Hv03/6asDVc9f41y5dn3m978bsG7duJt0c
uCW69fh29u0XdwruTNxdeo94r/y+2v3qB/oP6n+0/rFlwG3g+GDAYM/DWQ/vDgmH
nv6U/9OH4dJHzEfVl0YjjY+dHx8bDRq98mTOk+GnsqcTz8p+Vv9563Or59/94vtLz1j8
2PAL+YvPv655qfNy76uprzrHl8cfvM55PfGm/K3O233vuO+638e9H5ko/ED+UPPR+m
PHp9BP9z7nfP78L/eE8/sl0p8zAAAAlGNlUk0AAHolAACAgwAA+f8AAlDpAAB1MAAA
6mAAADqYAAAXb5JfxUYAAAQjSURBVHjaVNNZbFRlGlDh95w525zpdGa6TVtbykBby
iICxQY0AhYTJUCiiYqGqEEiJhKQmBg0ESPeeCGRENEYb4jhBr0gNQrRlCBiSgyLaSlS
aKEs3Wemy+xnzuqFYdD/6rt6ku/N9wue55EcPwWArCgIgkx5ZRuYVxsnJ801Z05f3jY
1MRnb/HxHV+uSph9RKq4mhkdwbZVgdQ2SHkPTwgj/h1QUWWi8/tfg/hM/XN/Y2zfa
ZnkSnuRDtLMsXhBOvrJtya/LlrcdMs1Qb1lVRQmSAEDAsU1kxpgamXp3y+azu1esreK9
dyRqs9PljkW6OsLx7lTV1ld/237s8HRV57MbnvO8CA+e9GCQFTk6Mza+4/0P+t9a9V
SEI3uyTH/eR27aB2Ed31Q/Hx1sI6BHOPT13c5Frd0HW9p3HPUQEwAigJW9RDp+bstr
Oy981nVGLN/7RpHUV70YfXnEAtjxFPasxPDBQXatjzNTdOQXtg983H/51AFFy1KClg2
bNldC+8270NwmUmelsXqSqHkDK5PDl8iCW0QcnEW+lqCjvcjQuMZ4YnQRTkotQUZ
u4GkjcfZNv19G011kXw4vayNYNvqCCvSVTciOgABgeuhBGwhgz5zbkl2ff7HUqJiNR2Q

ktbbSYnBYYqbMT/ilKI4SlbT/GcRylbnvLmJ2X8N7tJ7rR8OE/BbliqEYea81WIotmOs02
WFpc55Lf0f5/mSl3dsamOgxSX7ZjaALuBmB6M6FnB+S+POCwmOLk1QFFAqZyQWl1
YrpiRZJLvDkygyC5NJ1XCax7xYNiTQVEYVluUulaylcGeLkpw6WK7GuPY/fb2CkhleXlF
Fe8XPGaKBj9QxLW1lk0bg8EuT2zRCJYZvZlYepe0EGbvi4bQUJVZhs2phADFYj+df0lB
qJUnaekS4SUHXe3jrOnoE2PhSewHfRpfZGgcrylvfHdQruQlLo7Ns6QizqkJ31ClUlqwQ
JXuWUpDXj6qOsW32HT3YNlmll9FwJsb4jyaLmWQ4fa6a+2sQw0ry8YZSiHcPxxXBtM
fCv4XkUCrfliWs/fTE31rtTVfv9vsIorvQlniMhqXM4popVcJFVMHMpfMEaLPdxR1Tnna1
b1vl6tGntpAjgCTNWONZylFBR8Ydtr6EgrCl3VySfzZPLBDHyIq5gkpmzcOUmTGMF+b
h7M9LYulfWzMmHBzk7Fpq9deWEYxjrtaCMXjWfstp6BCGNXZzBdYqYhogWqkMum4
+oBVD0YnP63u/fFqbv1D+M7VSlBbmmK5uYaLYLYwslfwFVAyXQiOfcx3XyyGlM8DDn
0lgWyGokHogu/0UJxpL/+f2e569s/CZQZ53OpzJr0+NXludUfb5jVdf7VUGXJUPlZast1
S9PeII6jFDT5xMjFwO1S4c8zwTgnwEAxufYSzA67PMAAAAASUVORK5CYII=" >

```
  <a class="footer-item" href="http://www.beian.gov.cn/portal/registerSystemInfo
">沪公网安备 12345678912345 号</a>
</div>
<style>
.footer-content {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  justify-content: center;
}
.footer-content + .footer-content {
  margin-top: 8px;
}
.login-pf .footer-item {
  color: white;
}
.footer-item {
  color: var(--dao-gray-010);
  text-decoration: none;
}
```

```
.footer-item + .footer-item {
  margin-left: 8px;
}
.gongan-icon {
  width: 18px;
  height: 18px;
  margin-right: 4px;
}
</style>
```

!!! note

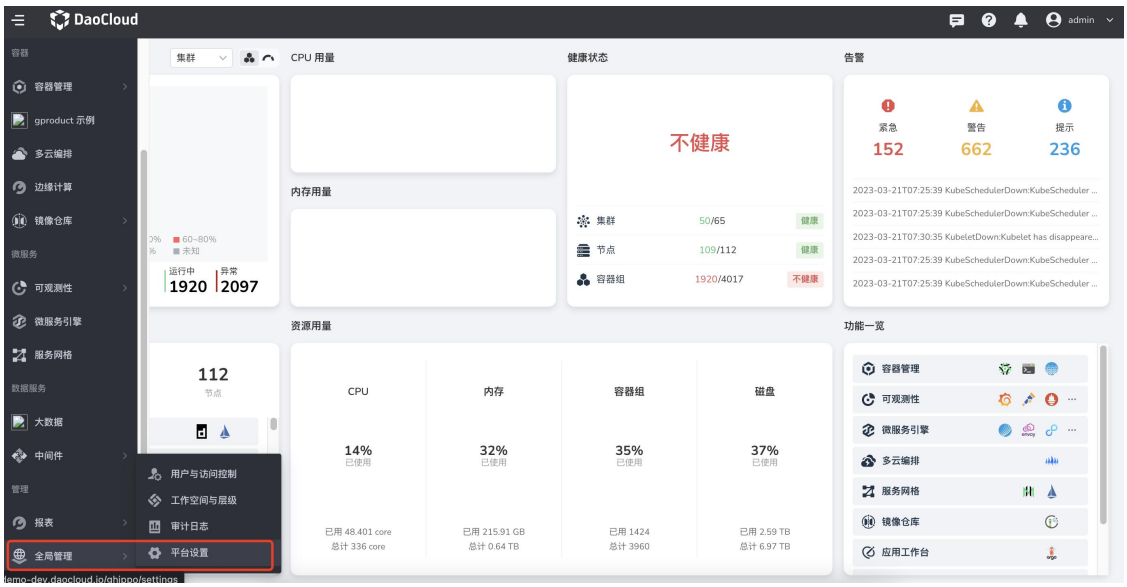
如果想要恢复默认设置，可以点击`一键还原`。请注意，一键还原后将丢弃所有自定义设置。

关于平台

关于平台主要呈现平台各个子模块当前更新的版本，声明了平台使用的各个开源软件，并以动画视频的方式致谢了平台的技术团队。

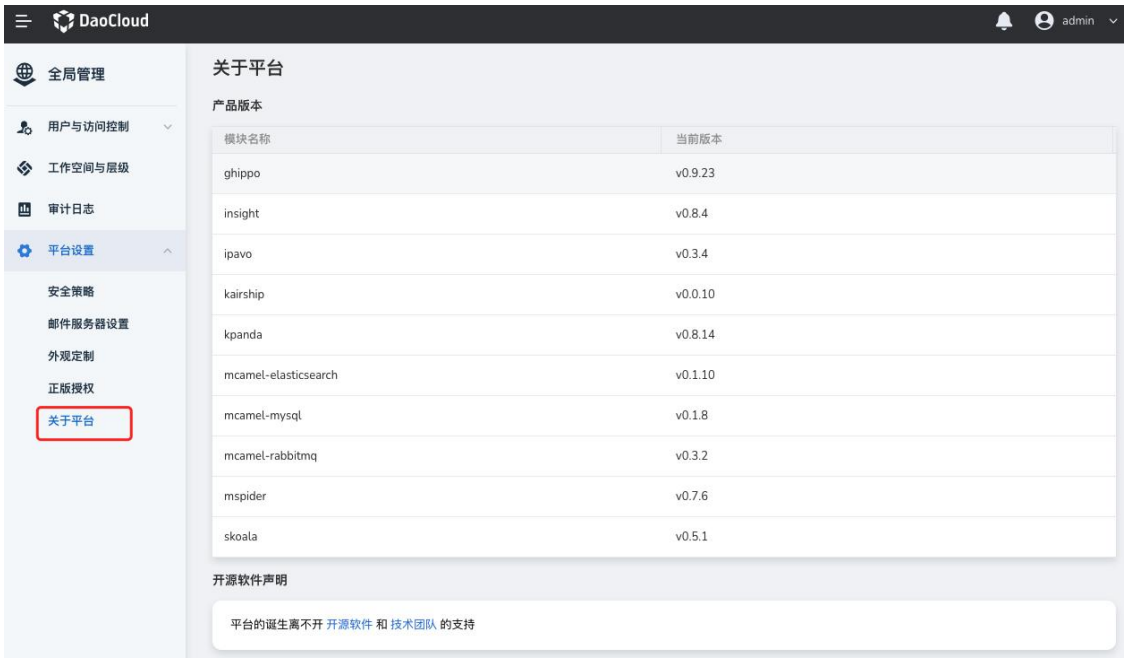
查看步骤：

1. 使用具有 Admin 角色的用户登录 DCE 5.0。点击左侧导航栏底部的全局管理。



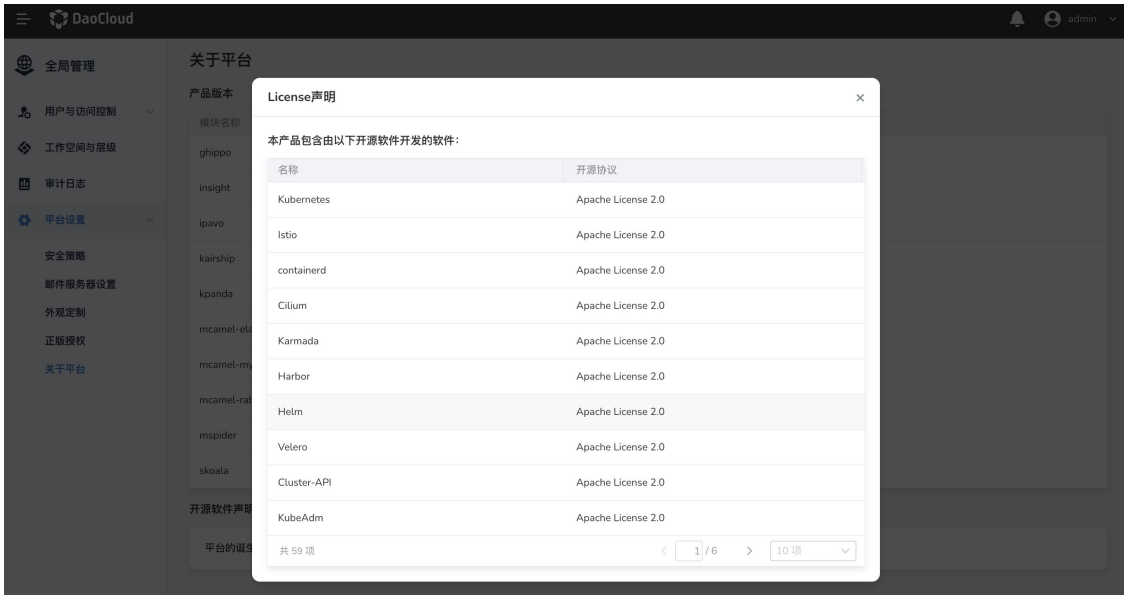
全局管理

2. 点击平台设置，选择关于平台，查看产品版本、开源软件声明和技术团队。



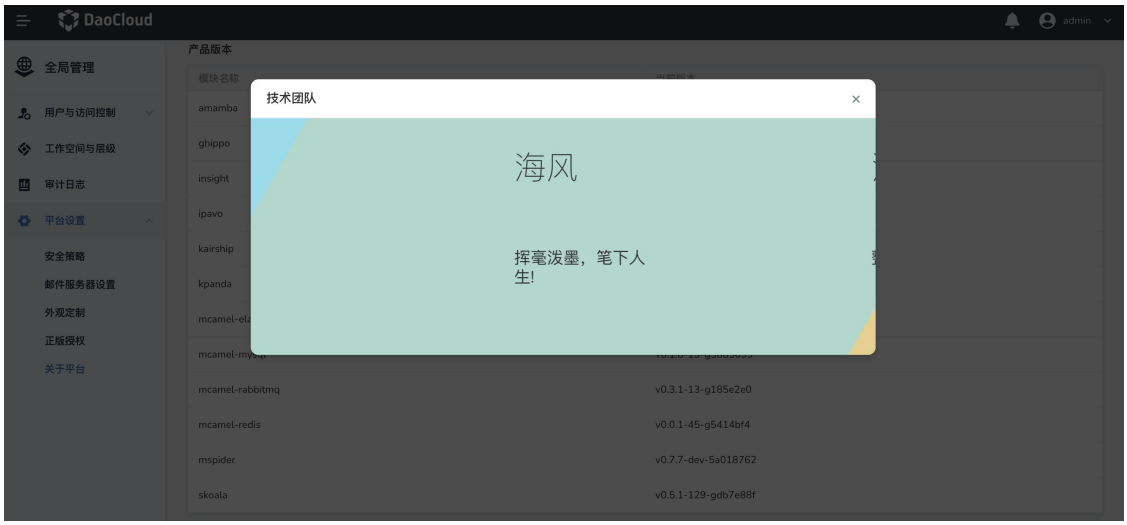
关于平台

License 声明



license 声明

技术团队


DaoCloud
admin
全局管理
产品版本
用户与访问控制
工作空间与层级
审计日志
平台设置
安全策略
邮件服务器设置
外观定制
正版授权
关于平台
模块名称
amamba
ghippo
insight
ipavo
kairship
kpanda
mcamel-rabbitmq
mcamel-redis
mspider
skoala
v0.3.1-13-g185e2e0
v0.0.1-45-g5414bf4
v0.7.7-dev-5a018762
v0.5.1-129-gdb7e88f
技术团队
海风
挥毫泼墨，笔下人生!

技术团队

安全策略

DCE 5.0 在图形界面上提供了基于密码和访问控制的安全策略。

密码策略

- 新密码不能与最近的历史密码相同。
- 密码过期后，系统强制要求修改密码。
- 密码不能与用户名相同。
- 密码不能和用户的邮箱地址相同。
- 自定义密码规则。
- 自定义密码最小长度。

访问控制策略

- 会话超时策略：用户在 x 小时内没有操作，退出当前账号。
- 账号锁定策略：限制时间内多次登录失败，账号将被锁定。
- 登录/退出策略：关闭浏览器的同时退出登录。

进入全局管理后，在左侧导航栏点击平台设置 -> 安全策略，即可设置密码策略和访问控制策略。


DaoCloud
haifeng
全局管理
用户与访问控制
工作空间与层级
审计日志
平台设置
安全策略
邮件服务器设置
外观定制
正版授权
关于平台
安全策略
密码策略设置
访问控制
新密码不能与最近的历史密码相同。
密码过期后，系统强制要求修改密码。
密码不能与用户名相同。
密码不能和用户的邮箱地址相同。
自定义密码规则。
密码最小长度 * 6
请输入1~32之间的整数。
保存
取消

安全策略

用户组

用户组是用户的集合，用户可以通过加入用户组，继承用户组的角色权限。通过用户组批量地给用户进行授权，可以更好地管理用户及其权限。

适用场景

当用户权限发生变化时，只需将其移到相应的用户组下，不会对其他用户产生影响。

当用户组的权限发生变化时，只需修改用户组的角色权限，即可应用到组内的所有用户。

创建用户组

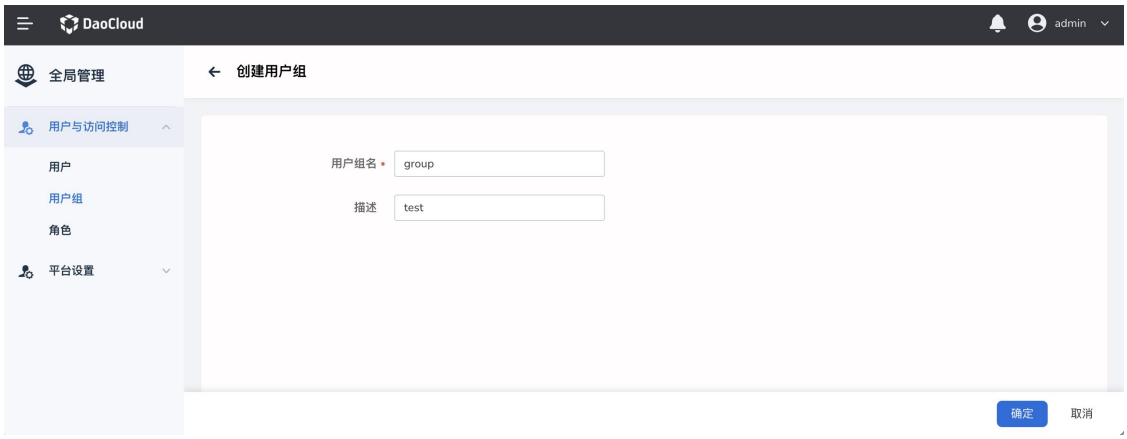
前提：拥有平台管理员 Super Admin 权限或者用户与访问控制管理员 IAM Admin 权限。

1. 管理员进入用户与访问控制，选择用户组，进入用户组列表，点击右上方的创建用户组。



创建用户组

2. 在创建用户组页面填写用户组信息。



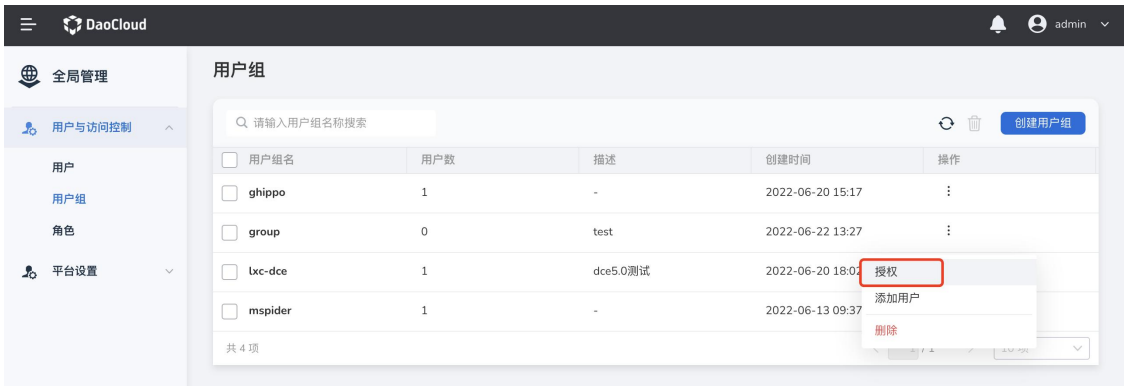
创建用户组

3. 点击确定，创建用户组成功，返回用户组列表页面。列表中的第一行是新创建的用户组。

为用户组授权

前提：该用户组已存在。

1. 管理员进入用户与访问控制，选择用户组，进入用户组列表，点击 ... –> 授权。



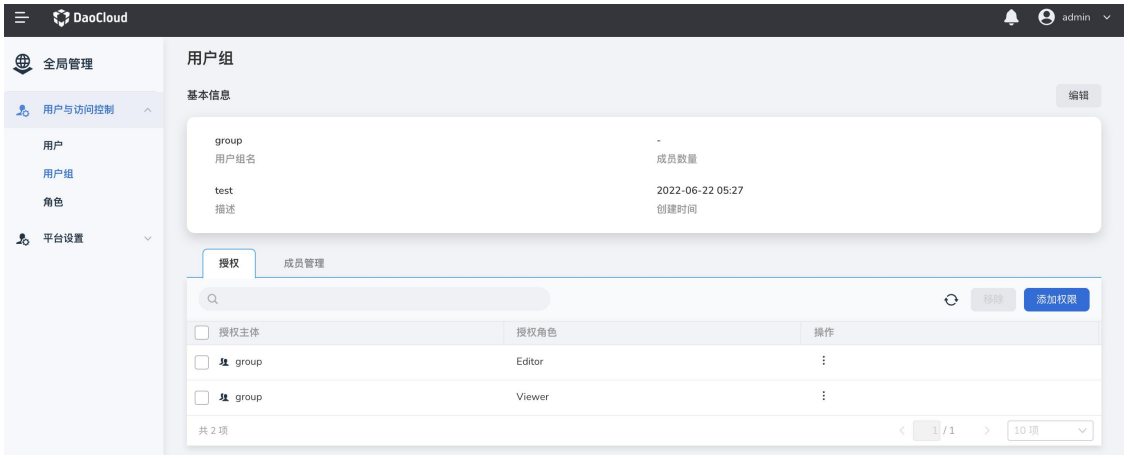
创建用户组按钮

2. 在授权页面勾选需要的角色权限（可多选）。



创建用户组按钮

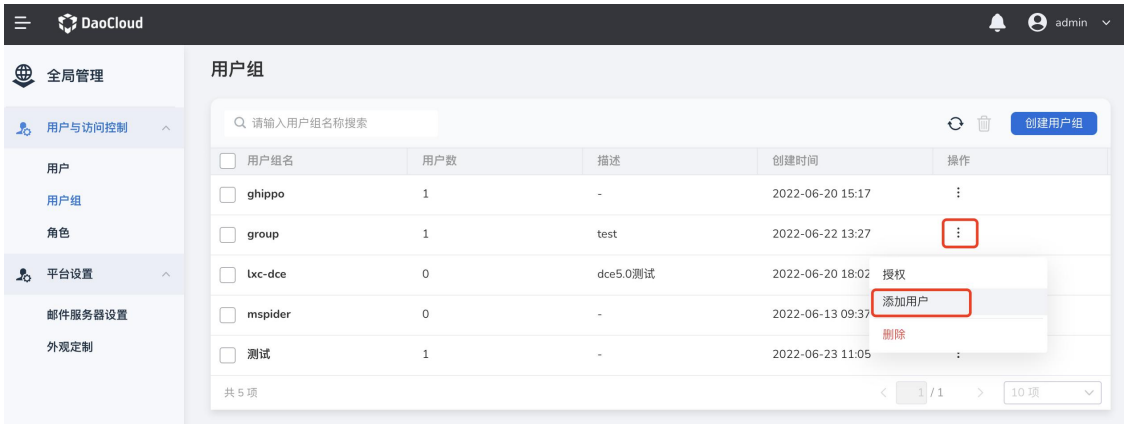
3. 点击确定完成为用户组的授权。自动返回用户组列表，点击某个用户组，可以查看用户组被授予的权限。



创建用户组按钮

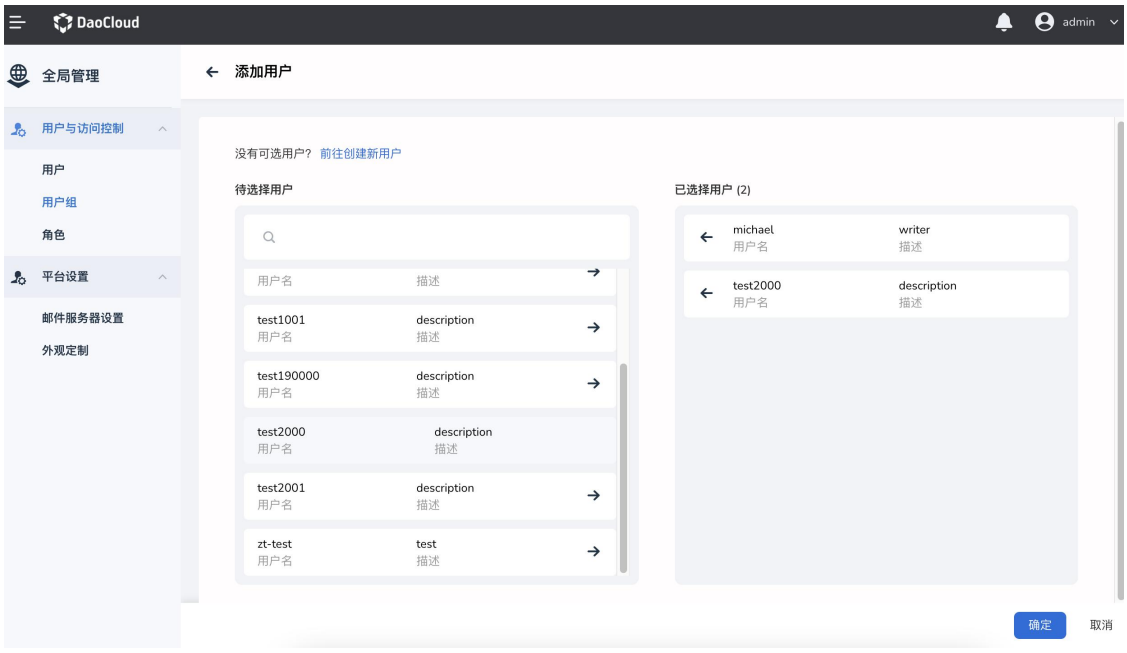
给用户组添加用户

1. 管理员进入用户与访问控制，选择用户组进入用户组列表，在某个用户组右侧，点击 … –> 添加用户。



添加用户

2. 在添加用户页面点选需要添加的用户（可多选）。若没有可选的用户，点击前往创建新用户，先前往创建用户，再返回该页面点击刷新按钮，显示刚创建的用户。



选择用户

3. 点击确定完成给用户组添加用户。

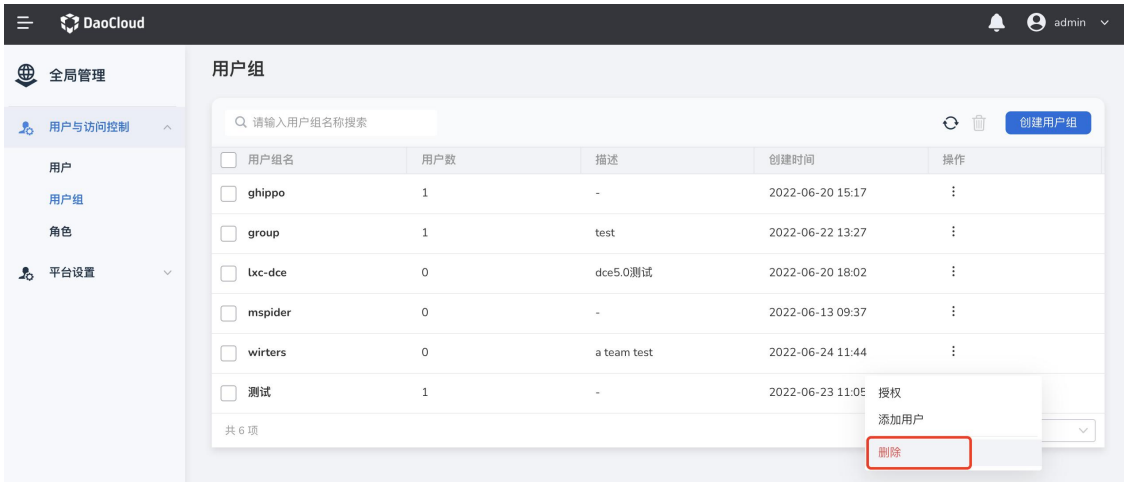
!!! note

用户组中的用户会继承用户组的权限；可以在用户组详情中查看加入该组的用户。

删除用户组

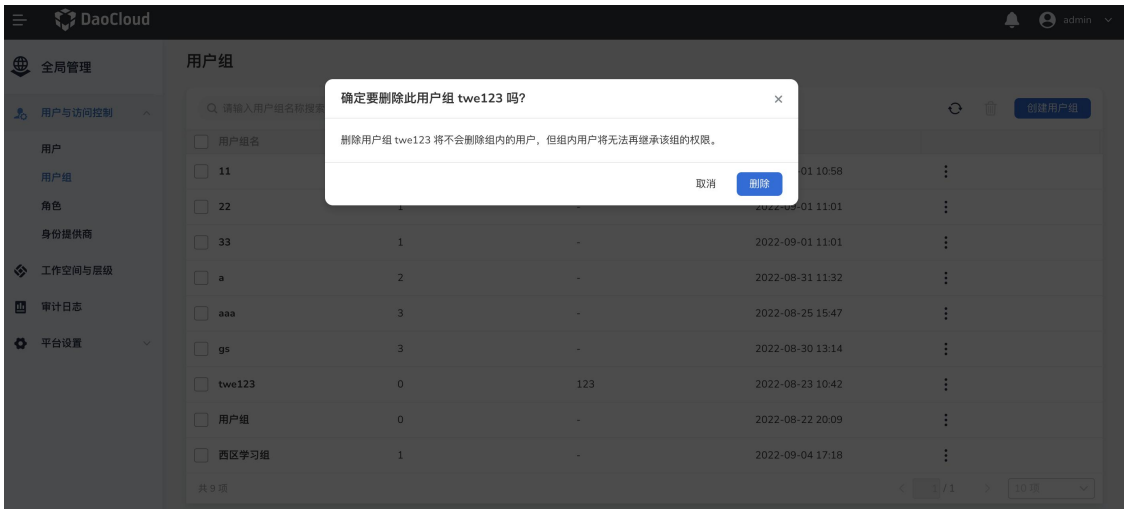
说明：删除用户组，不会删除组内的用户，但组内用户将无法再继承该组的权限

1. 管理员进入用户与访问控制，选择用户组进入用户组列表，在某个用户组右侧，点击 ... -> 删除。



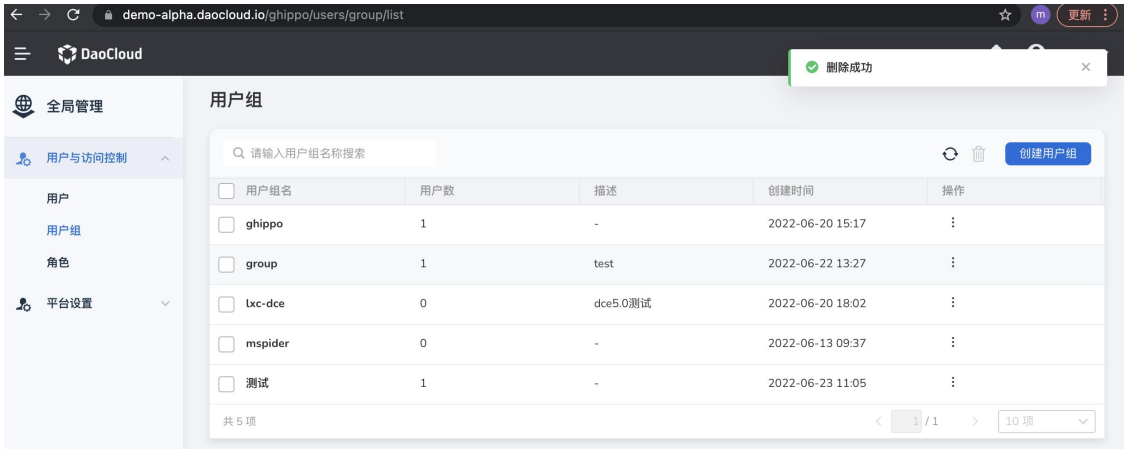
删除按钮

2. 点击移除删除用户组。



确认删除

3. 返回用户组列表，屏幕上方将提示删除成功。



删除提示

!!! note

说明：删除用户组，不会删除组内的用户，但组内用户将无法再继承该组的权限。

身份提供商

全局管理支持基于 LDAP 和 OIDC 协议的单点登录，如果您的企业或组织已有自己的账号体系，同时希望管理组织内的成员使用 DCE 5.0 资源，您可以使用全局管理提供的身份提供商功能，而不必在您的 DCE 5.0 中为每一位组织成员创建用户名/密码。您可以向这些外部用户身份授予使用 DCE 5.0 资源的权限。

基本概念

- 身份提供商（Identity Provider，简称 IdP）

  负责收集和存储用户身份信息、用户名、密码等，在用户登录时负责认证用户的服务。在企业与 DCE 5.0 进行身份认证的过程中，身份提供商指企业自身的身份提供商。

- 服务提供商（Service Provider，简称 SP）

  服务提供商通过与身份提供商 IdP 建立信任关系，使用 IDP 提供的用户信息，为用户提供具体的服务。在企业与 DCE 5.0 进行身份认证的过程中，服务提供商指 DCE 5.0。

- LDAP

  LDAP 指轻型目录访问协议（Lightweight Directory Access Protocol），常用于单点登录，即用户可以在多个服务中使用一个账号密码进行登录。全局管理支持 LDAP 进行身份认证，因此与 DCE 5.0 通过 LDAP 协议建立身份认证的企业 IdP 必须支持 LDAP 协议。关于 LDAP 的详细描述请参见：欢迎使用 LDAP。

- OIDC

  OIDC 是 OpenID Connect 的简称，是一个基于 OAuth 2.0 协议的身份认证标准协议。全局管理支持使用 OIDC 协议进行身份认证，因此与

DCE 5.0 通过 OIDC 协议建立身份认证的企业 IdP 必须支持 OIDC 协议。关于 OIDC 的详细描述请参见：欢迎使用 OpenID Connect。

- OAuth 2.0

  OAuth 2.0 是 Open Authorization 2.0 的简称，是一种开放授权协议，授权框架支持第三方应用程序以自己的名义获取访问权限。

功能特性

- 管理员无需重新创建 DCE 5.0 用户

  使用身份提供商进行身份认证前，管理员需要在企业管理系统和 DCE 5.0 中分别为用户创建账号；使用身份提供商进行身份认证后，企业管理员只需要在企业管理系统中为用户创建账号，用户即可同时访问两个系统，降低了人员管理成本。

- 用户无需记住两套平台账号

  使用身份提供商进行身份认证前，用户访问企业管理系统和 DCE 5.0 需要使用两个系统的账号登录；使用身份提供商进行身份认证后，用户在本企业管理系统中登录即可访问两个系统。

LDAP

LDAP 英文全称为 Lightweight Directory Access Protocol，即轻型目录访问协议，这是一个开放的、中立的工业标准应用协议， 通过 IP 协议提供访问控制和维护分布式信息的目录信息。

如果您的企业或组织已有自己的账号体系，同时您的企业用户管理系统支持 LDAP 协议，就可以使用全局管理提供的基于 LDAP 协议的身份提供商功能，而不必在 DCE 5.0 中为每一位组织成员创建用户名/密码。 您可以向这些外部用户身份授予使用 DCE 5.0 资源的权限。

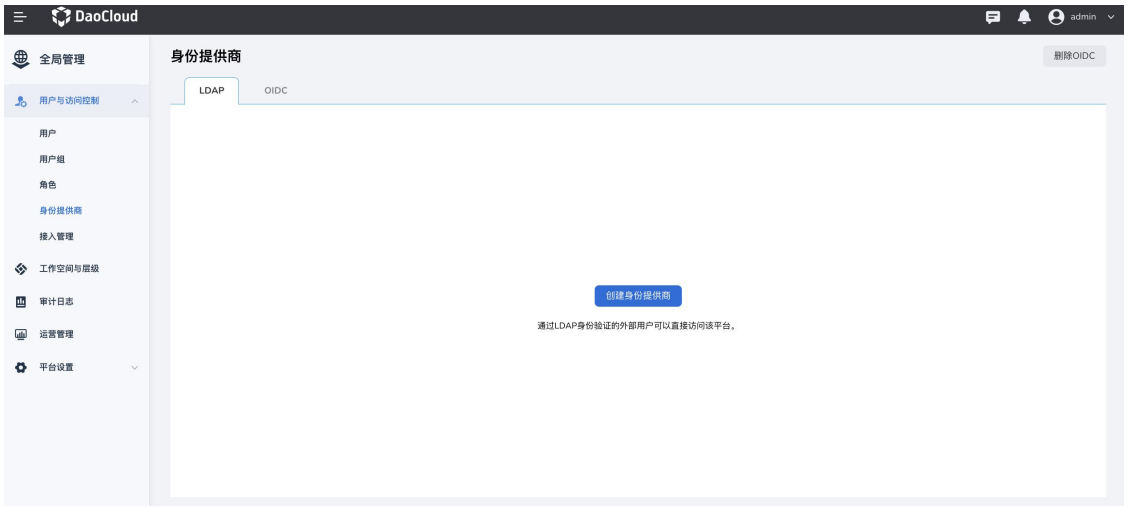
在全局管理中，其操作步骤如下：

1. 使用具有 admin 角色的用户登录 DCE 5.0。点击左侧导航栏左下角的全局管理 -> 用户与访问控制。



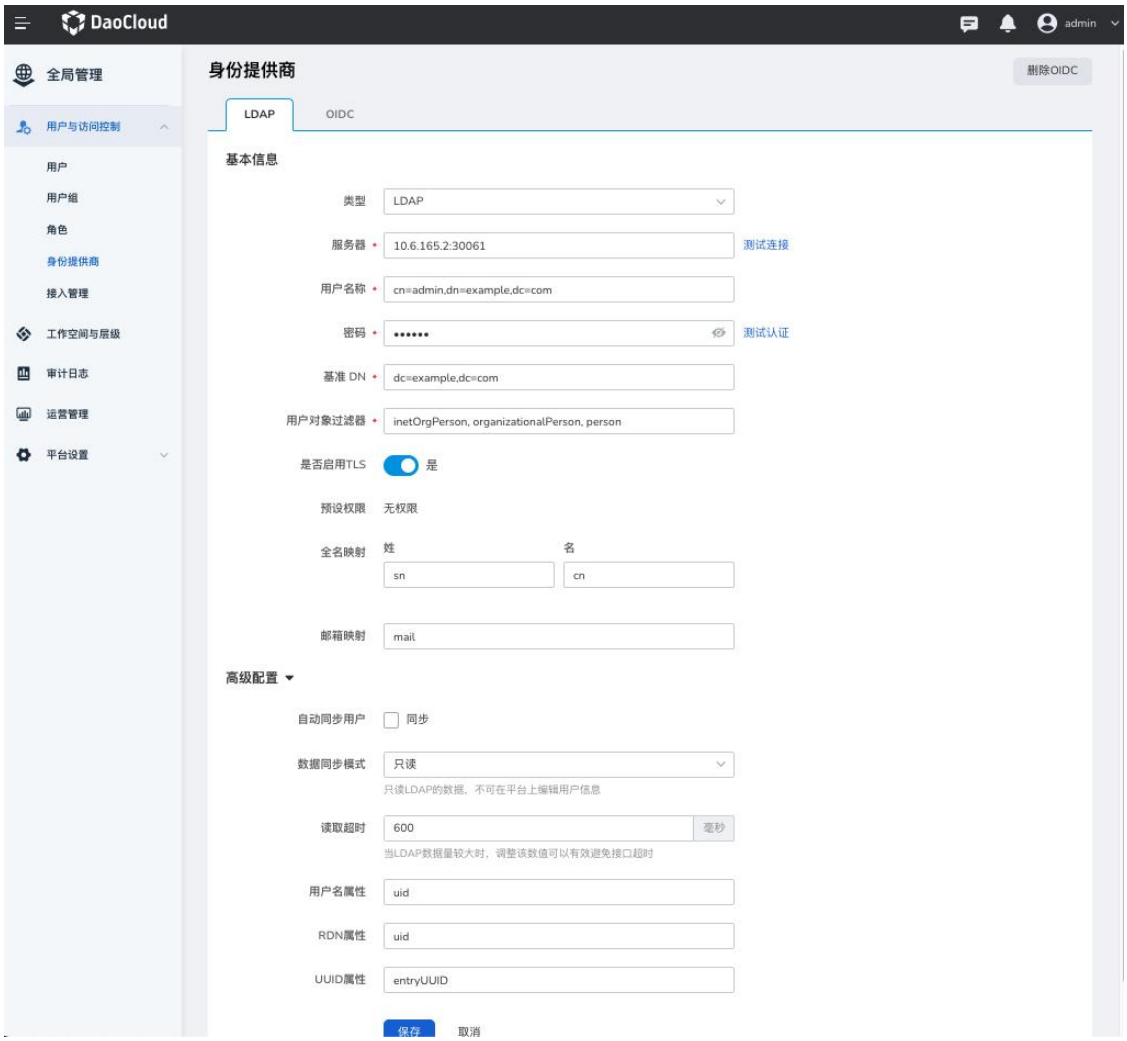
global

2. 在左侧导航栏点击身份提供商，点击创建身份提供商按钮。



身份提供商

3. 在 LDAP 页签中，填写以下字段后点击保存，建立与身份提供商的信任关系及用户的映射关系。



ldap

字段	描述	举例值
类型	支持 LDAP (Lightweight Directory Access Protocol) 和 AD (Active Directory)	LDAP
服务器	LDAP 服务的地址和端口号	10.6.165.2:30061
用户名称	登录的用户名	cn=admin,dn=daocloud,dc=com
密码	登录的密码	password
基准	admin 的 DN，用于访问 LDAP 服务器	dc=daocloud,dc=io

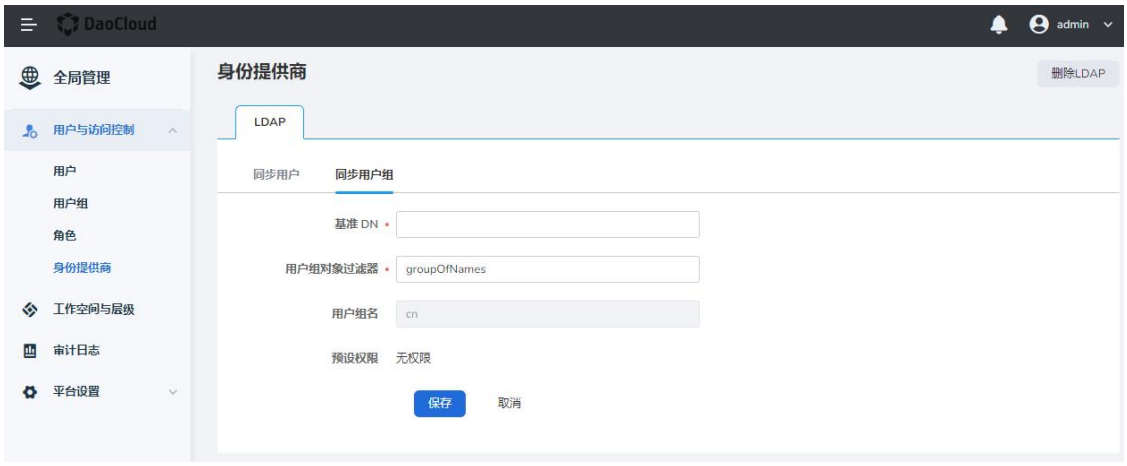
字段	描述	举例值
DN		
用户对象过滤器	LDAP 用户的 LDAP objectClass 新建的用户将与所有这些对象类一起写入 LDAP，并且只要它们包含所有这些对象类，就会找到现在的 LDAP 用户记录。DCE 5.0 已经帮用户自动填入，如需修改可直接编辑。	inetOrgPerson, organizationalPerson
是否启用 TLS	启用后将加密 DCE 5.0 与 LDAP 的连接	否
全名映射	姓–sn；名–cn	不可更改
邮箱映射	是指将用户的电子邮件地址与 LDAP 帐户相关联。用于允许只有特定电子邮件域名的用户能够访问某些资源，如内部网站或文件共享	邮箱地址，不可更改

高级配置

字段	描述	举例值
自动同步	默认 1 小时同步一次，可以自行配置	勾选
数据同步模式	对于只读 LDAP 的数据，不可以在 DCE 5.0 平台上编辑用户信息对于写入 LDAP 的数据，可以在 DCE 5.0 上编辑用户信息后再同步回 LDAP	只读
读取超时	当 LDAP 数据量较大时，调整该数值可以有效避免接口超时	600 毫秒
用户名属性	是指在认证和授权过程中用于标识用户的属性。用户名属性是唯一的且不可更改的，可	uid

字段	描述	举例值
	以是用户的电子邮件地址、登录名或其他由系统管理员定义的属性。用户名属性用作用户的唯一标识符，以便系统可以识别特定用户并授予其相应的权限。	
RDN 属性	是指用于创建 Relative Distinguished Name（RDN）的属性。在 X.500 和 LDAP 目录服务中，RDN 属性通常是唯一的，并且是用于标识目录树（Naming Context）中某个对象的一部分。例如，在“cn=John Doe,ou=People,dc=example,dc=com”中，“cn”就是 RDN 属性之一。RDN 属性定义了该对象在其父级对象下的相对名称，因此它必须唯一。当新对象添加到目录中时，它的 RDN 属性必须与同一层级中其他对象的 RDN 属性不同，否则将导致命名冲突。	uid
UUID 属性	是指唯一标识符（Universally Unique Identifier）属性。UUID 是由数字、字母和连字符组成的 36 个字符的字符串，用于在计算机系统中标识对象。UUID 可以确保在任何给定时间内，不同计算机上的对象都具有唯一标识符。	entryUUID

4. 在同步用户组页签中，填写以下字段配置用户组的映射关系后，再次点击保存。



身份提供商

字段	描述	举例值
基准 DN	用户组在 LDAP 树状结构中的位置	ou=groups,dc=example,dc=org
用户组对象过滤器	用户组的对象类，如果需要更多类，则用逗号分隔。在典型的 LDAP 部署中，通常是“groupOfNames”，系统已自动填入，如需更改请直接编辑。* 表示所有。	*
用户组名	cn	不可更改

!!! note

1. 当您通过 LDAP 协议将企业用户管理系统与 DCE 5.0 建立信任关系后，可通过手动同步或自动同步的方式，将企业用户管理系统中的用户或用户组一次性同步至 DCE 5.0。
1. 同步后管理员可对用户组/用户组进行批量授权，同时用户可通过在企业用户管理系统中的账号/密码登录 DCE 5.0。
1. 有关实际操作教程，请参阅 [LDAP 操作演示视频](../../../videos/ghippo.md#ldap)。

系统角色

适用场景

DCE 5.0 提供了预置的系统角色，帮助用户简化角色权限的使用步骤。

!!! note

DCE 5.0 提供了三种类型的系统角色，分别为平台角色、工作空间角色和文件夹角色。

- 平台角色：对平台上所有相关资源具有相应权限，请前往用户/用户组授权。
- 工作空间角色：对某个工作空间具有相应权限，请前往具体工作空间授权。
- 文件夹角色：对某个文件夹、子文件夹及其工作空间下的资源具有相应权限，请前往具体文件夹授权。

平台角色

在用户与访问控制中预定义了 5 个系统角色，分别是：Admin、IAM Owner、Audit Owner、 Kpanda Owner 和 Workspace and Folder Owner 。这 5 个角色由系统创建，用户只能使用不能修改。角色对应的权限如下：

角色名称	角色类型	所属模块	角色权限
Admin	系统角色	全部	平台管理员，管理所有平台资源，代表平台的最高权限
IAM Owner	系统角色	用户	用户与访问控制的管理员，拥有该服务下的所有权限，如管理用户/用户组及授权

角色名称	角色类型	所属模块与访问控制	角色权限
Audit Owner	系统角色	审计日志	审计日志的管理员，拥有该服务下的所有权限，如设置审计日志策略，导出审计日志
Kpanda Owner	系统角色	容器管理	容器管理的管理员，拥有该服务下的所有权限，如创建/接入集群，部署应用，给用户/用户组授予集群/命名空间相关的权限
Workspace and Folder Owner	系统角色	工作空间与层级	工作空间与层级管理员，拥有该服务下的所有权限，如创建文件夹/工作空间，给用户/用户组授权文件夹/工作空间的相关权限，在工作空间下使用应用工作台、微服务引擎等功能

工作空间角色

在用户与访问控制中预定义了 3 个系统角色，分别是：Workspace Admin、Workspace Editor、Workspace Viewer。这 3 个角色由系统创建，用户只能使用不能修改。角色对应的权限如下：

角色名称	角色类型	所属模块	角色权限
Workspace Admin	系统角色	工作空间	工作空间的管理权限
Workspace Editor	系统角色	工作空间	工作空间的编辑权限
Workspace Viewer	系统角色	工作空间	工作空间的只读权限

文件夹角色

在用户与访问控制中预定义了 3 个系统角色，分别是：Folder Admin、Folder Editor、Folder Viewer。这 3 个角色由系统创建，用户只能使用不能修改。角色对应的权限如下：

角色名称	角色类型	所属模块	角色权限
Folder Admin	系统角色	工作空间	文件夹及其下子文件夹、工作空间的管理权限
Folder Editor	系统角色	工作空间	文件夹及其下子文件夹、工作空间的编辑权限
Folder Viewer	系统角色	工作空间	文件夹及其下子文件夹、工作空间的只读权限

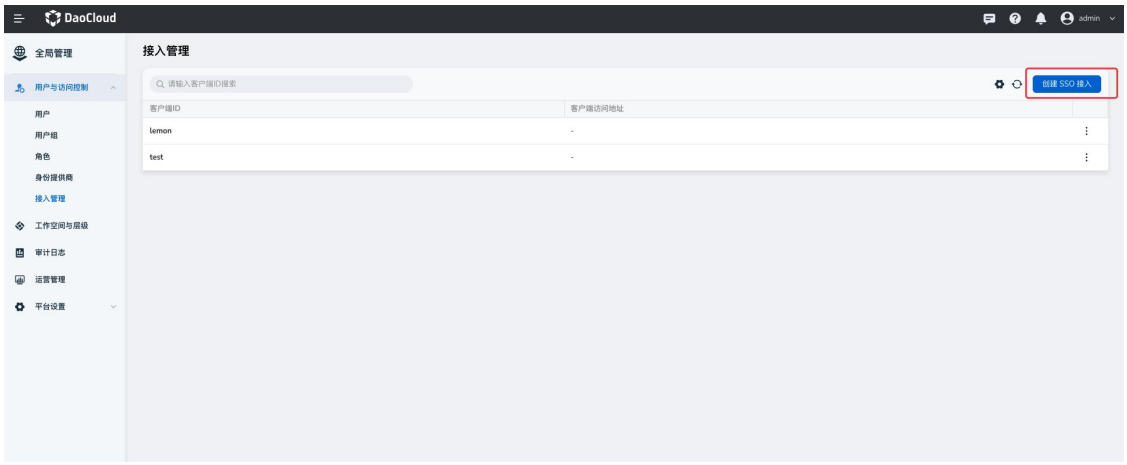
接入管理

当两个或两个以上平台相互对接或嵌入时，通常需要进行用户体系打通。 在用户打通过程中，接入管理主要提供 SSO 接入能力，当您需要将 DCE 5.0 作为用户源接入客户平台时， 您可以通过接入管理创建 SSO 接入来实现。

创建 SSO 接入

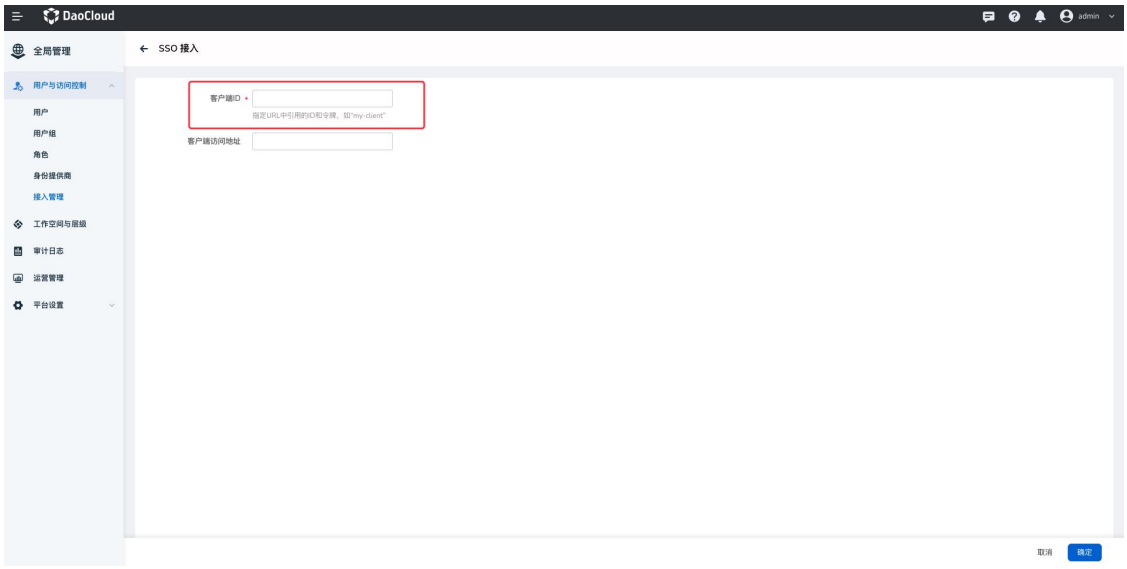
前提：拥有平台管理员 Admin 权限或者用户与访问控制管理员 IAM admin 权限。

1. 管理员进入用户与访问控制，选择接入管理，进入接入管理列表，点击右上方的创建 SSO 接入。



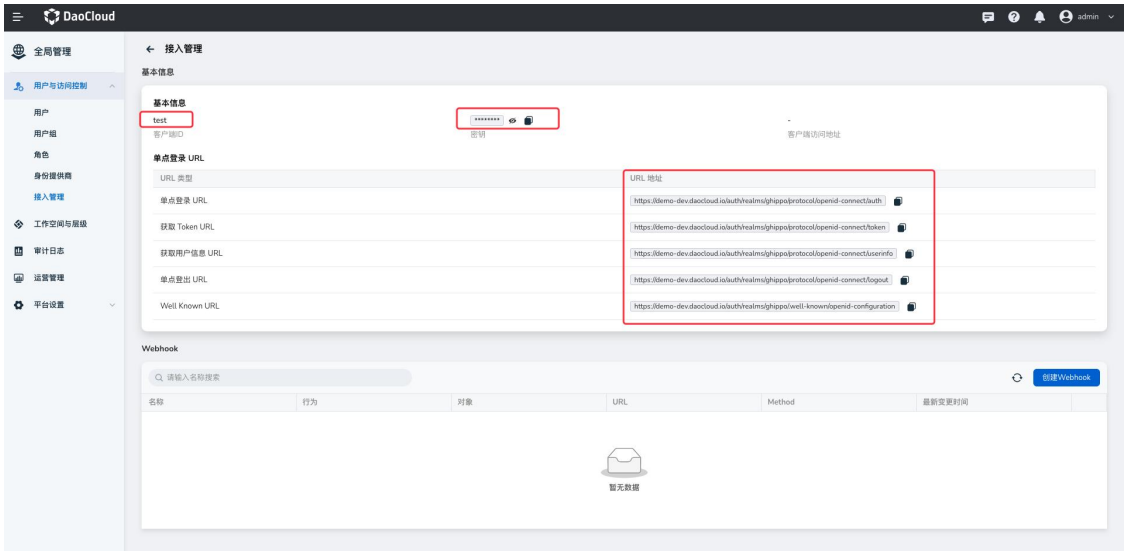
创建 SSO 接入按钮

2. 在创建 SSO 接入页面填写客户端 ID。



创建 SSO 接入

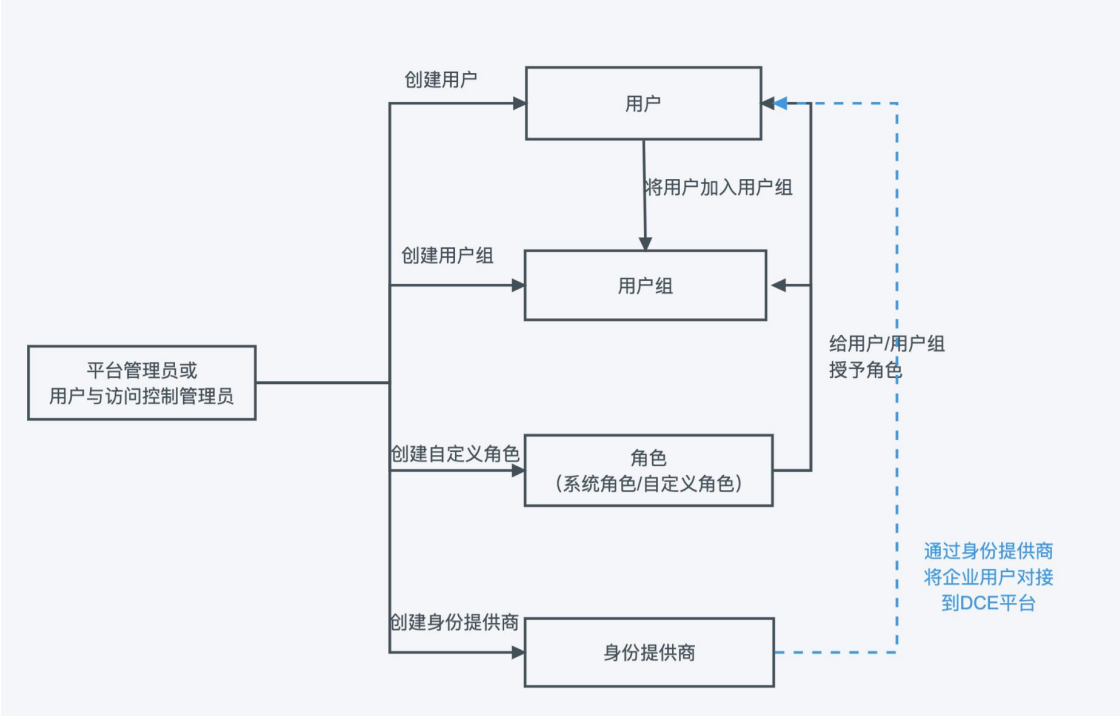
3. 创建 SSO 接入成功后，在接入管理管理列表，点击刚创建的客户端 ID 进入详情， 复制客户端 ID、密钥和单点登录 URL 信息，填写至客户平台完成用户体系打通。



接入管理详情

什么是用户与访问控制

IAM（Identity and Access Management，用户与访问控制）是全局管理的一个重要模块，您可以通过用户与访问控制模块创建、管理和销毁用户（用户组），并使用系统角色和自定义角色控制其他用户使用 DCE 平台的权限。



IAM 定义

优势

- 简洁流畅

  企业内部的结构和角色可能非常复杂，项目、工作小组及授权的管理都在不断地变化。用户与访问控制采用清晰整洁的页面，打通用户、用户组、角色之间的授权关系，以最短链路实现对用户（用户组）的授权。

- 适当的角色

用户与访问控制为每个子模块预定义了一个管理员角色，无需用户维护，您可以直接将平台预定义的系统角色授权给用户，实现平台的模块化管理（细粒度权限请参阅权限管理。

- 企业级访问控制

  当您希望本企业员工可以使用企业内部的认证系统登录 DCE 平台，而不需要在 DCE 平台创建对应的用户，您可以使用用户与访问控制的身份提供商功能，建立您所在企业与 DCE 的信任关系，通过联合认证使员工使用企业已有账号直接登录 DCE 平台，实现单点登录。

使用流程

1. 使用 DCE 平台主账号（Super Admin）或具有管理员权限的用户账号登录 DCE 平台。
2. 创建用户，参见用户。
3. 为用户授权，参见权限管理。
4. 创建用户组，参见用户组。
5. 创建自定义角色，参见自定义角色。
6. 创建身份提供商，参见身份提供商。

OIDC

OIDC（OpenID Connect）是建立在 OAuth 2.0 基础上的一个身份层，是基于 OAuth2 协议的身份认证标准协议。

如果您的企业或组织已有自己的账号体系，同时您的企业用户管理系统支持 OIDC 协议，可以使用全局管理提供的基于 OIDC 协议的身份提供商功能，而不必在 DCE 5.0 中为每一位组织成员创建用户名/密码。 您可以向这些外部用户身份授予使用 DCE 5.0 资源的权限。

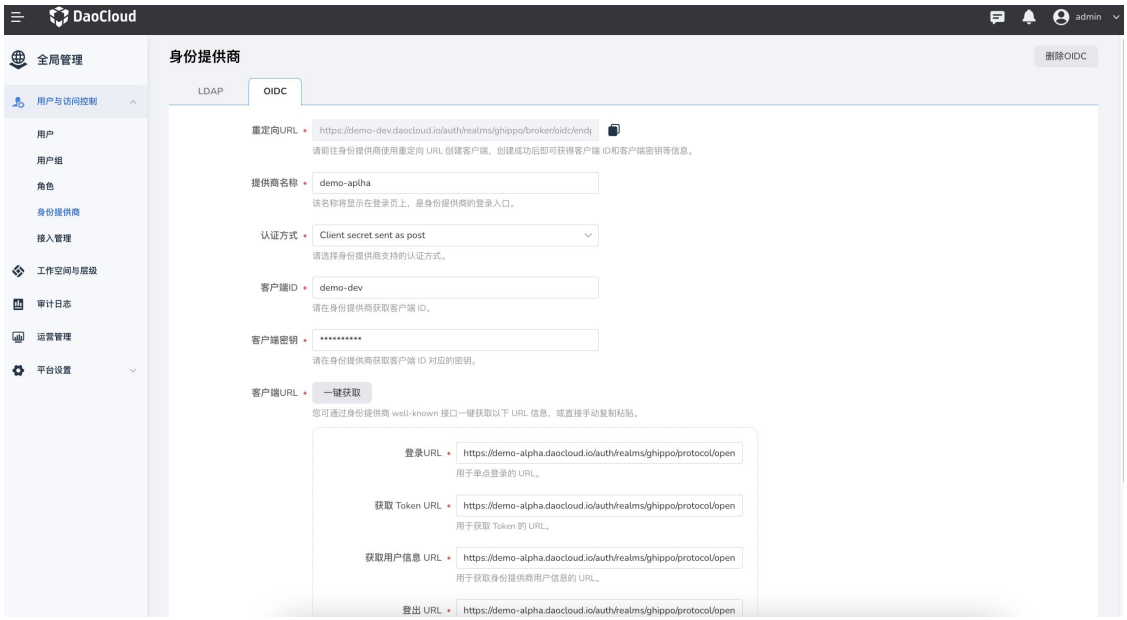
具体操作步骤如下。

1. 使用具有 admin 角色的用户登录 DCE 5.0。点击左侧导航栏底部的全局管理 -> 用户与访问控制。



全局管理

2. 在左侧导航栏选择身份提供商，点击 OIDC 页签。填写表单字段，建立与身份提供商的信任关系后，点击保存。



身份提供商

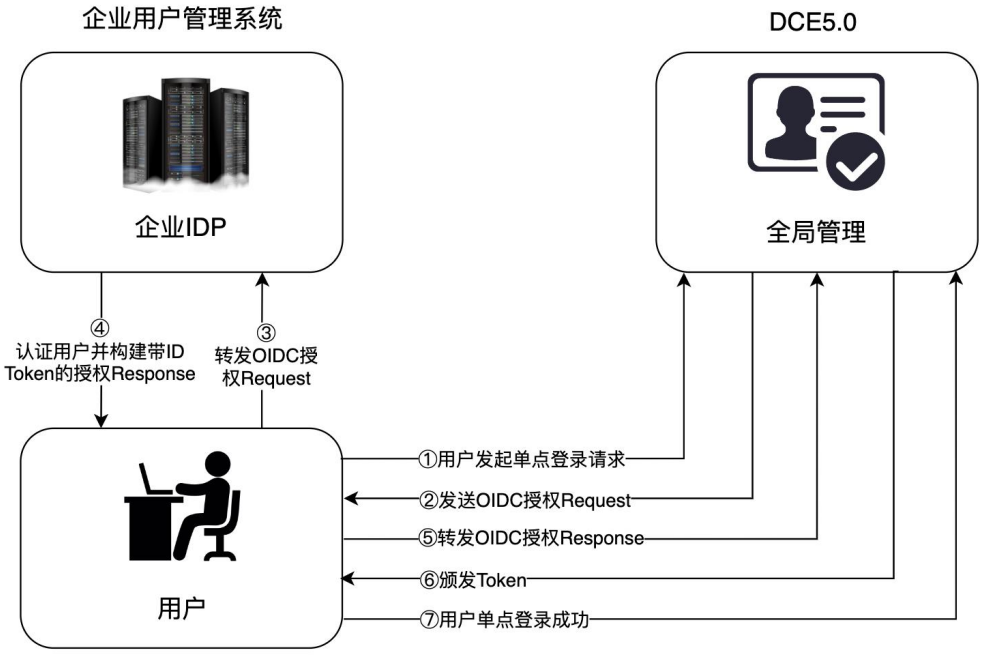
字段	描述
重定向 URL	重定向到 DCE 5.0 的 URL
提供商名称	显示在登录页上，是身份提供商的入口
认证方式	客户端身份验证方法。如果 JWT 使用私钥签名，请下拉选择 JWT signed with private key。具体参阅 Client Authentication。
客户端 ID	客户端的 ID
客户端密钥	客户端密码
客户端 URL	可通过身份提供商 well-known 接口一键获取登录 URL、Token URL、用户信息 URL 和登出 URL
自动关联	开启后当身份提供商用户名/邮箱与 DCE 5.0 用户名/邮箱重复时将自动使二者关联

!!! note

1. 当用户通过企业用户管理系统完成第一次登录 DCE 5.0 后，用户信息才会被同步至 DCE 5.0 的`用户与访问控制` -> `用户列表`。
1. 初次登录的用户不会被赋予任何默认权限，需要有管理员给其赋权（管理员可以是平台管理员、子模块管理员或资源管理员）。
1. 有关实际操作教程，请参阅 [OIDC 操作演示视频](../../../videos/ghippo.md#oidc)，也可参考 [Azure OpenID Connect（OIDC）接入流程](https://learn.microsoft.com/zh-cn/azure/active-directory/develop/v2-protocols-oidc)。

用户身份认证交互流程

用户身份认证的交互流程为：



oidc

1. 使用浏览器发起单点登录 DCE 5.0 的请求。
2. DCE 5.0 根据登录链接中携带的信息，查找全局管理 -> 用户与访问控制 -> 身份提供商中对应的配置信息，构建 OIDC 授权 Request，发送给浏览器。

3. 浏览器收到请求后，转发 OIDC 授权 Request 给企业 IdP。
4. 在企业 IdP 的登录页面中输入用户名和密码，企业 IdP 对提供的身份信息进行验证，并构建携带用户信息的 ID Token，向浏览器发送 OIDC 授权 Response。
5. 浏览器响应后转发 OIDC 授权 Response 给 DCE 5.0。
6. DCE 5.0 从 OIDC 授权 Response 中取出 ID Token，并根据已配置的身份转换规则映射到具体的用户列表，颁发 Token。
7. 完成单点登录，访问 DCE 5.0。

自定义角色

DCE 5.0 支持创建三种范围的自定义角色：

- **平台角色** 的权限对平台所有相关资源生效
- **工作空间角色** 的权限对该用户所在的工作空间下的资源生效
- **文件夹角色** 的权限对该用户所在的文件夹及其下的子文件夹和工作空间资源生效

创建平台角色

平台角色是粗粒度角色，能够对所选权限内的所有资源生效。如授权后用户可以拥有所有工作空间的查看权限、所有集群的编辑权限等，而不能针对某个工作空间或某个集群生效。平台角色创建完成后可以在用户/用户组列表中进行授权。

1. 从左侧导航栏点击全局管理 -> 用户与访问控制 -> 角色，点击创建自定义角色。



创建自定义角色

2. 输入名称、描述，选择平台角色，勾选角色权限后点击确定。



平台角色

3. 返回角色列表，搜索刚创建的自定义角色，点击右侧的 ⋮，可以执行复制、编辑和删除等操作。



更多操作

4. 平台角色创建成功后，可以去用户/用户组授权，为这个角色添加用户和用户组。

创建工作空间角色

工作空间角色是细粒度角色，针对某个工作空间生效。如在该角色中选择应用工作台的全部权限，给用户在某个工作空间下授予该角色后，该用户将仅能在该工作空间下使用应用工作台相关的功能，而无法使用如微服务引擎、中间件等其他模块的能力。工作空间角色创建完成后，可以在工作空间与层级中选择工作空间后进行授权。

1. 从左侧导航栏点击全局管理 -> 用户与访问控制 -> 角色，点击创建自定义角色。



创建自定义角色

2. 输入名称、描述，选择工作空间角色，勾选角色权限后点击确定。



工作空间角色

3. 返回角色列表，搜索刚创建的自定义角色，点击右侧的 ⋮，可以执行复制、编辑和删除等操作。



更多操作

4. 工作空间角色创建成功后，可以去工作空间授权，设定这个角色可以管理哪些工作空间。

创建文件夹角色

文件夹角色针对某个文件夹和该文件夹下的所有子文件夹及工作空间生效。如在该角色中选择全局管理–工作空间和应用工作台，给用户在某个文件夹下授予该角色后，该用户将能够在其下的所有工作空间中使用应用工作台的相关功能，而无法使用如微服务引擎、中间件等其他模块的能力。文件夹角色创建完成后，可以在工作空间与层级中选择文件夹后进行授权。 请注意：应用工作台、多云编排、镜像仓库、微服务引擎、服务网格和中间件均依赖于工作空间，因此在创建文件夹角色时大部分场景下需要用到工作空间，请注意在全局管理–工作空间下勾选。

1. 从左侧导航栏点击全局管理 –> 用户与访问控制 –> 角色，点击创建自定义角色。



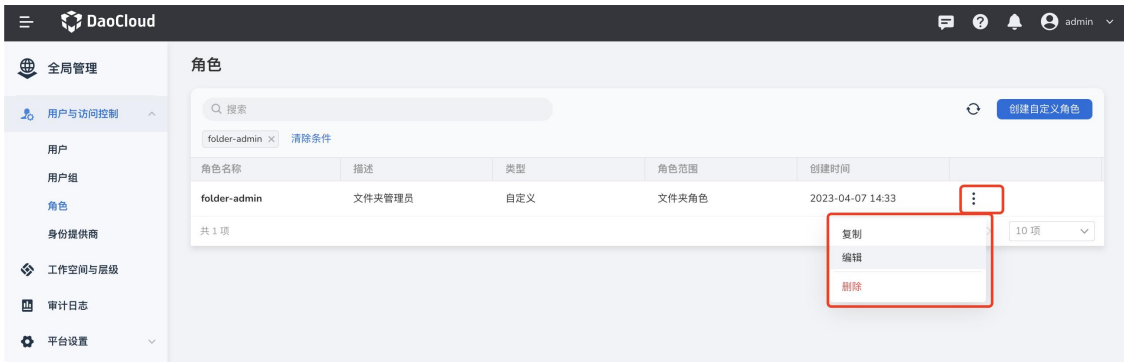
创建自定义角色

2. 输入名称、描述，选择文件夹角色，勾选角色权限后点击确定。



文件夹角色

3. 返回角色列表，搜索刚创建的自定义角色，点击右侧的 ⋮，可以执行复制、编辑和删除等操作。



更多操作

4. 文件夹角色创建成功后，可以去文件夹授权，设定这个角色可以管理哪些文件夹。

用户

用户指的是由平台管理员 admin 或者用户与访问控制管理员 IAM admin 在 全局管理 -> 用户与访问控制 -> 用户 页面创建的用户，或者通过 LDAP / OIDC 对接过来的用户。 用户名代表账号，用户通过用户名和密码登录 DaoCloud Enterprise 平台。

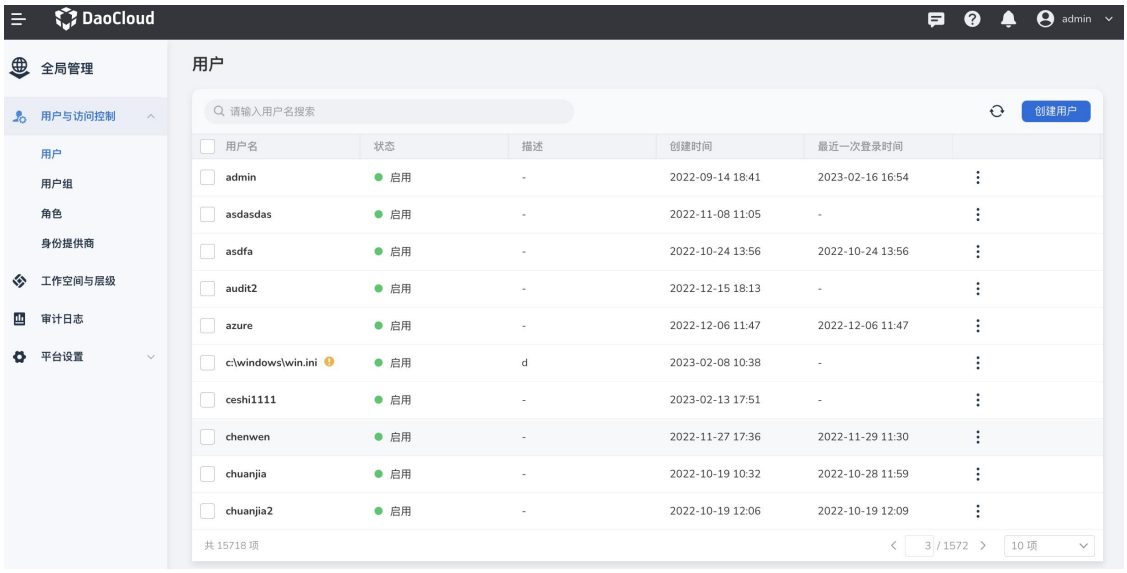
拥有一个用户账号是用户访问平台的前提。新建的用户默认没有任何权限，例如您需要给用户赋予相应的角色权限，比如在用户列表或用户详情授予子模块的管理员权限。 子模块管理员拥有该子模块的最高权限，能够创建、管理、删除该模块的所有资源。 如果用户需要被授予具体资源的权限，比如某个资源的使用权限，请查看资源授权说明。

本页介绍用户的创建、授权、禁用、启用、删除等操作。

创建用户

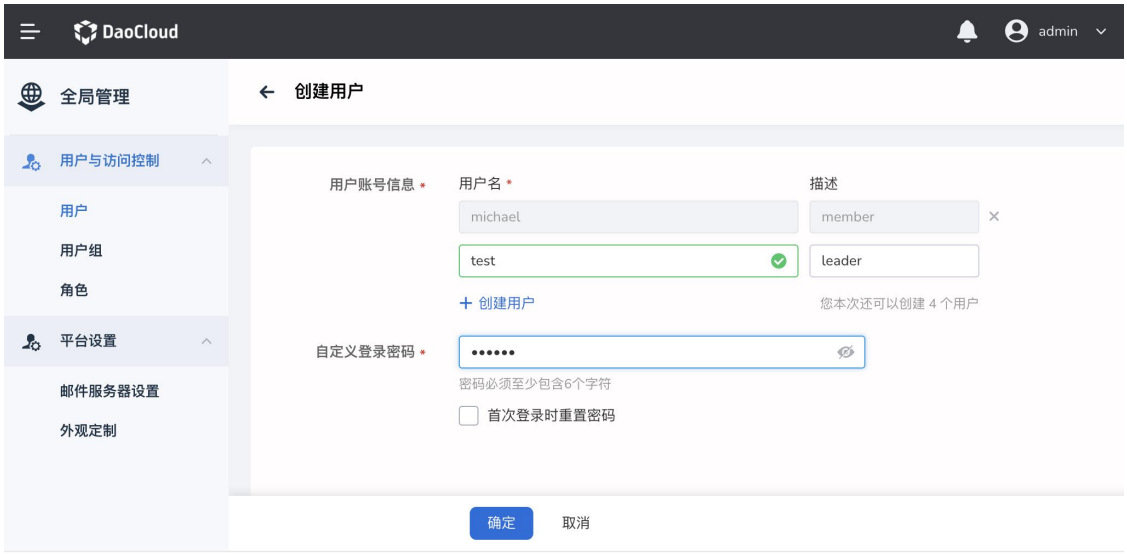
前提：拥有平台管理员 Admin 权限或者用户与访问控制管理员 IAM admin 权限。

1. 管理员进入用户与访问控制，选择用户，进入用户列表，点击右上方的创建用户。



创建用户按钮

2. 在创建用户页面填写用户名和登录密码。如需一次性创建多个用户，可以点击创建用户后进行批量创建，一次性最多创建 5 个用户。根据您的实际情况确定是否设置用户在首次登录时重置密码。



设置用户名和密码

3. 点击确定，创建用户成功，返回用户列表页。

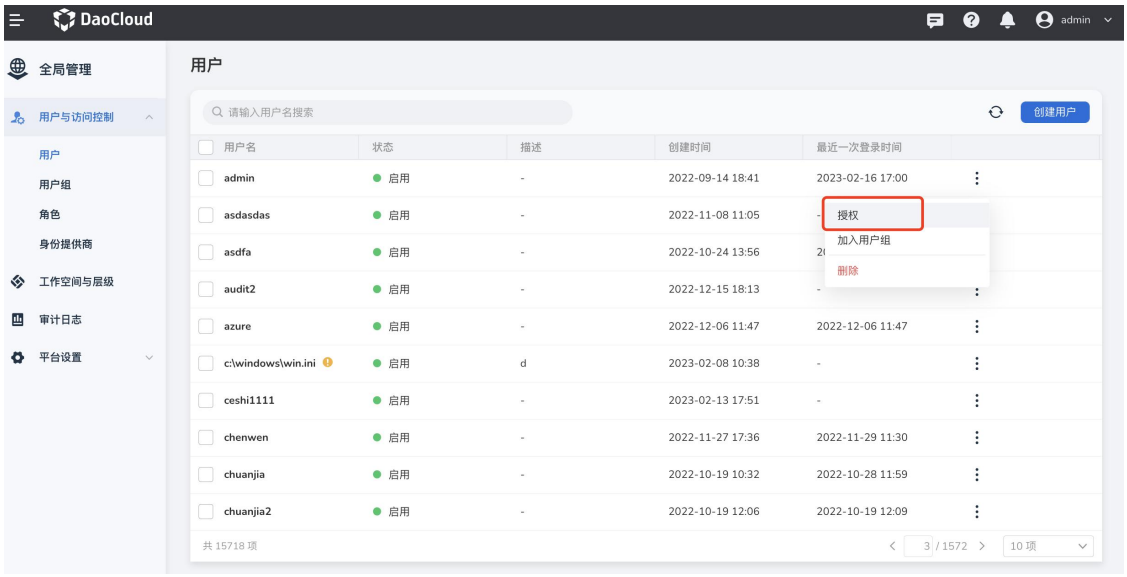
!!! note

此处设置的用户名和密码将用于登录平台。

为用户授予子模块管理员权限

前提：该用户已存在。

1. 管理员进入用户与访问控制，选择用户，进入用户列表，点击 ⋮ -> 授权。



授权菜单

2. 在授权页面勾选需要的角色权限（可多选）。



授权界面

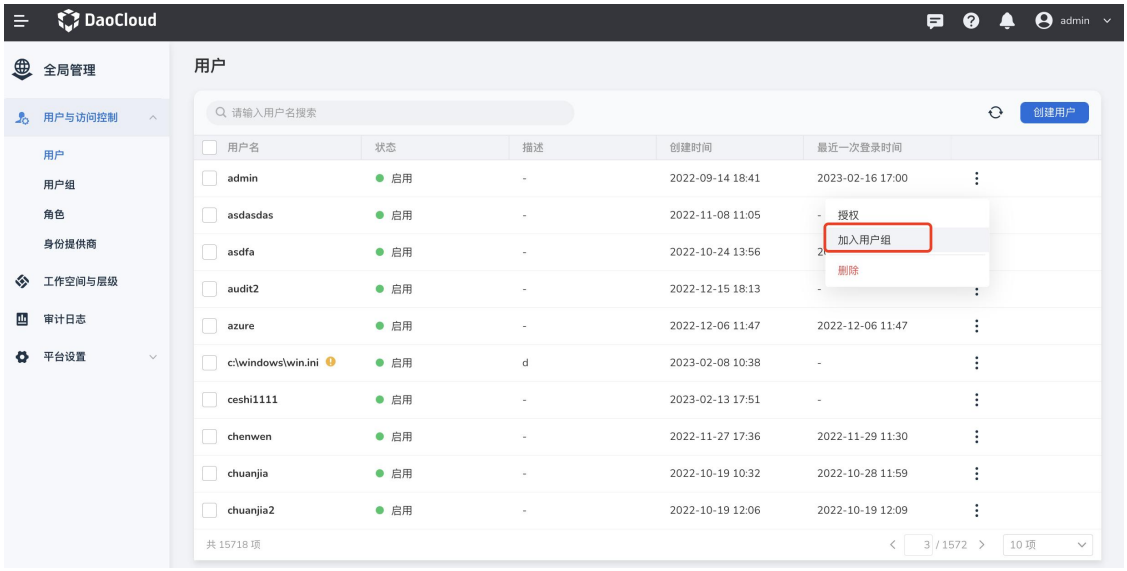
3. 点击确定完成为用户的授权。

!!! note

在用户列表中，点击某个用户，可以进入用户详情页面。

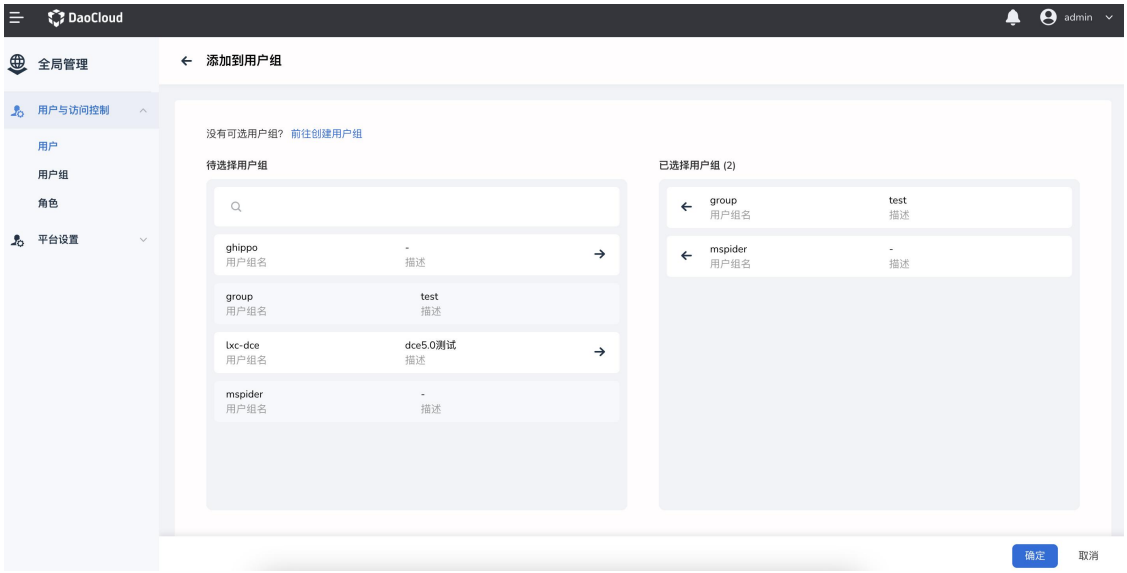
将用户加入用户组

1. 管理员进入用户与访问控制，选择用户，进入用户列表，点击 ⋮ -> 加入用户组。



加入用户组菜单

2. 在加入用户组页面勾选需要加入的用户组（可多选）。若没有可选的用户组，点击创建用户组创建用户组，再返回该页面点击刷新按钮，显示刚创建的用户组。



加入用户组界面

3. 点击确定将用户加入用户组。

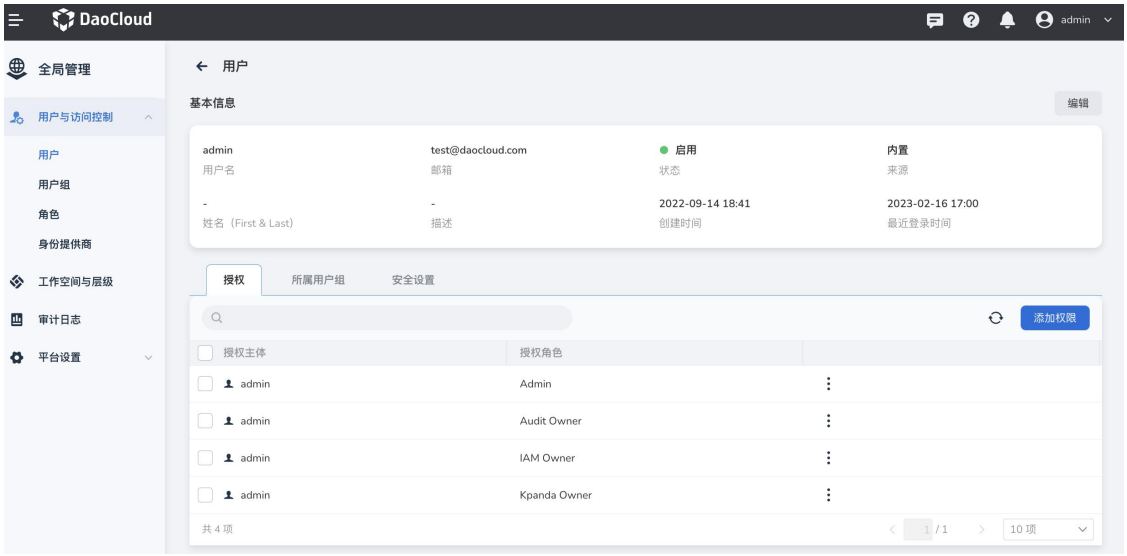
!!! note

用户会继承用户组的权限，可以在`用户详情`中查看该用户已加入的用户组。

启用/禁用用户

禁用用户后，该用户将无法再访问平台。与删除用户不同，禁用的用户可以根据需要再次启用，建议删除用户前先禁用，以确保没有关键服务在使用该用户创建的密钥。

1. 管理员进入用户与访问控制，选择用户，进入用户列表，点击一个用户名进入用户详情。



用户详情

2. 点击右上方的编辑，关闭状态按钮，使按钮置灰且处于未启用状态。



编辑

3. 点击确定完成禁用用户的操作。

忘记密码

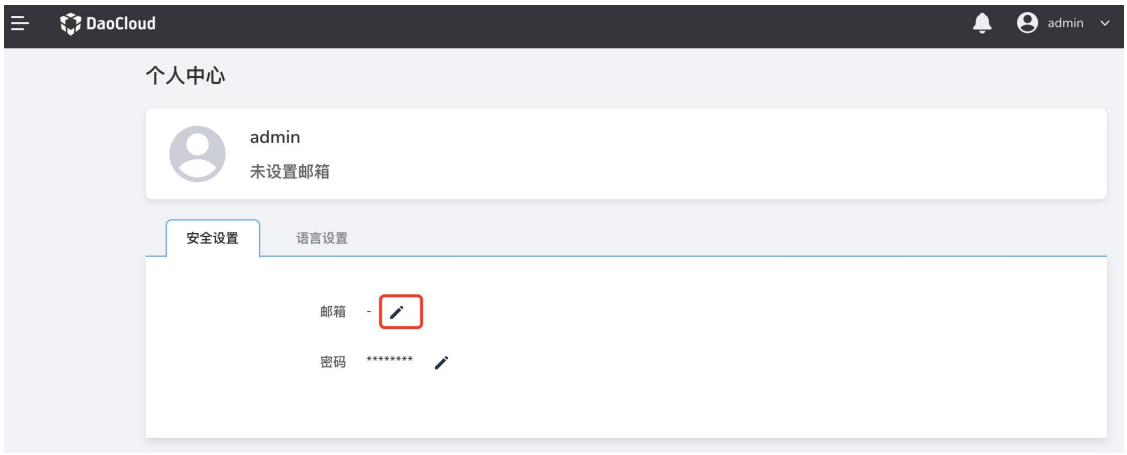
前提：需要设置用户邮箱，有两种方式可以设置用户邮箱。

- 管理员在该用户详情页面，点击编辑，在弹出框输入用户邮箱地址，点击确定完成邮箱设置。



编辑

- 用户还可以进入个人中心，在安全设置页面设置邮箱地址。



个人中心

如果用户登录时忘记密码，请参考重置密码。

删除用户

!!! warning

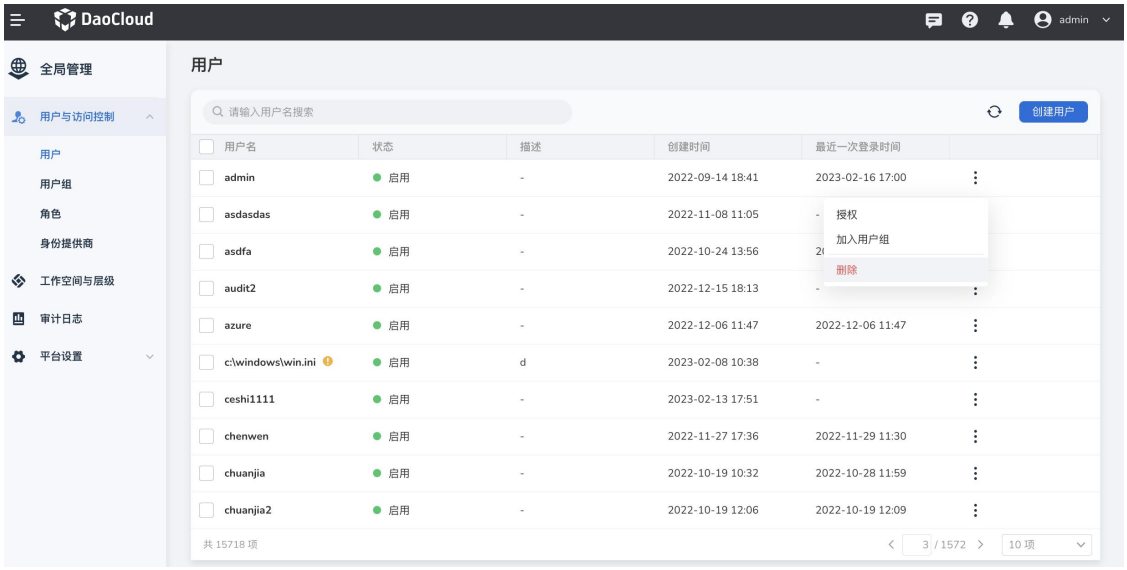
删除用户后，该用户将无法再通过任何方式访问平台资源，请谨慎删除。

在删除用户之前，请确保您的关键程序不再使用该用户创建的密钥。

如果您不确定，建议在删除前先禁用该用户。

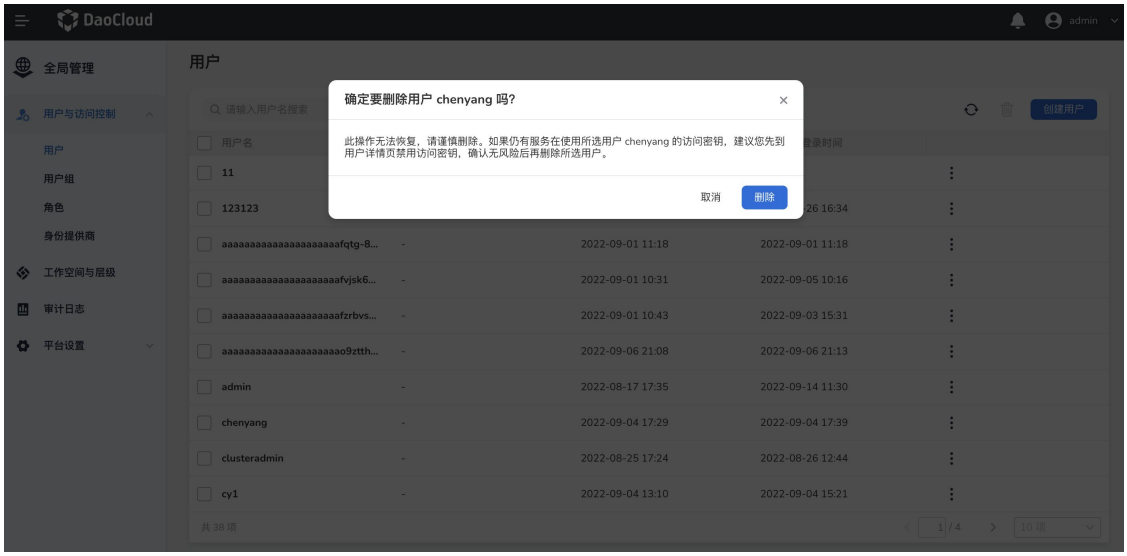
如果您删除了一个用户，然后再创建一个同名的新用户，则新用户将被视为一个新的独立身份，它不会继承已删除用户的角色。

1. 管理员进入用户与访问控制，选择用户，进入用户列表，点击 ⋮ -> 删除。



删除用户菜单

2. 点击移除完成删除用户的操作。


DaoCloud
admin
全局管理
用户与访问控制
用户
用户组
角色
身份提供商
工作空间与层级
审计日志
平台设置
用户
请输入用户名搜索
创建用户
用户名
登录时间
确定要删除用户 chenyang 吗?
此操作无法恢复，请谨慎删除。如果仍有服务在使用所选用户 chenyang 的访问密钥，建议您先到用户详情页禁用访问密钥，确认无风险后再删除所选用户。
取消
删除
11
123123
aaaaaaaaaaaaaaaaaaaaafqtg-8...
2022-09-01 11:18
2022-09-01 11:18
aaaaaaaaaaaaaaaaaaaaafvjsk6...
2022-09-01 10:31
2022-09-05 10:16
aaaaaaaaaaaaaaaaaaaaafzrbvs...
2022-09-01 10:43
2022-09-03 15:31
aaaaaaaaaaaaaaaaaaaaao9ztth...
2022-09-06 21:08
2022-09-06 21:13
admin
2022-08-17 17:35
2022-09-14 11:30
chenyang
2022-09-04 17:29
2022-09-04 17:39
clusteradmin
2022-08-25 17:24
2022-08-26 12:44
cy1
2022-09-04 13:10
2022-09-04 15:21
共 38 项
1 / 4
10 项

删除用户确认

角色和权限管理

一个角色对应一组权限。权限决定了可以对资源执行的操作。向用户授予某角色，即授予该角色所包含的所有权限。

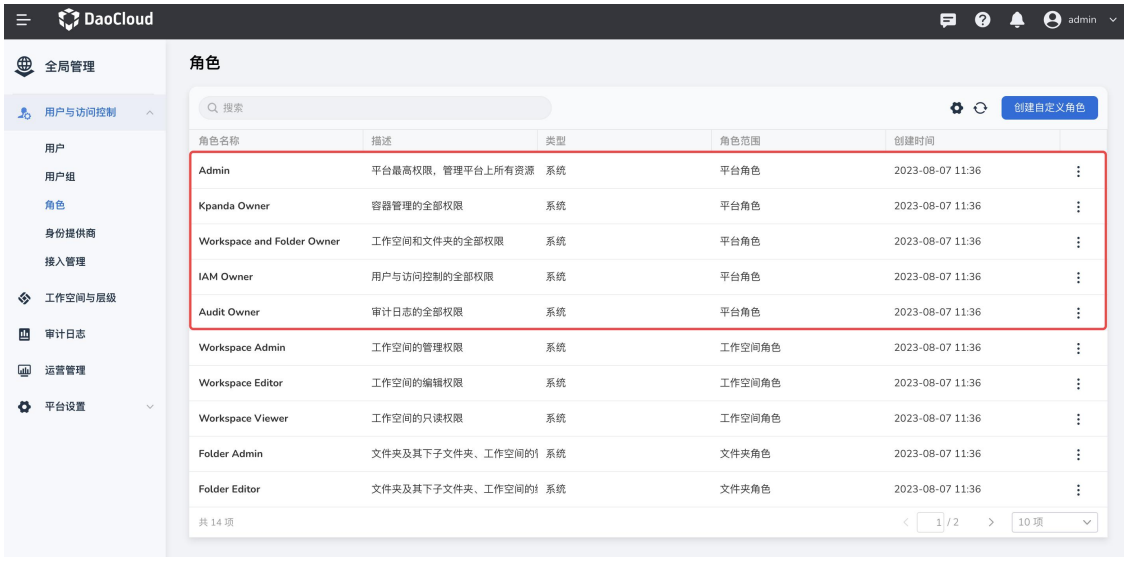
DCE 5.0 平台存在三种角色范围，能够灵活、有效地解决您在权限上的使用问题：

- 平台角色
- 工作空间角色
- 文件夹角色

平台角色

平台角色是粗粒度权限，对平台上所有相关资源具有相应权限。通过平台角色可以赋予用户对所有集群、所有工作空间等的增删改查权限， 而不能具体到某一个集群或某一个工作空间。DCE 5.0 提供了 5 个预置的、用户可直接使用的平台角色：

- Admin
- Kpanda Owner
- Workspace and Folder Owner
- IAM Owner
- Audit Owner



5 个预置平台角色

同时，DCE 5.0 还支持用户创建自定义平台角色，可根据需要自定义角色内容。如创建一个平台角色，包含应用工作台的所有功能权限，由于应用工作台依赖于工作空间， 因此平台会帮助用户默认勾选工作空间的查看权限，请不要手动取消勾选。 若用户 A 被授予该 Workbench（应用工作台）角色，将自动拥有所有工作空间下的应用工作台相关功能的增删改查等权限。

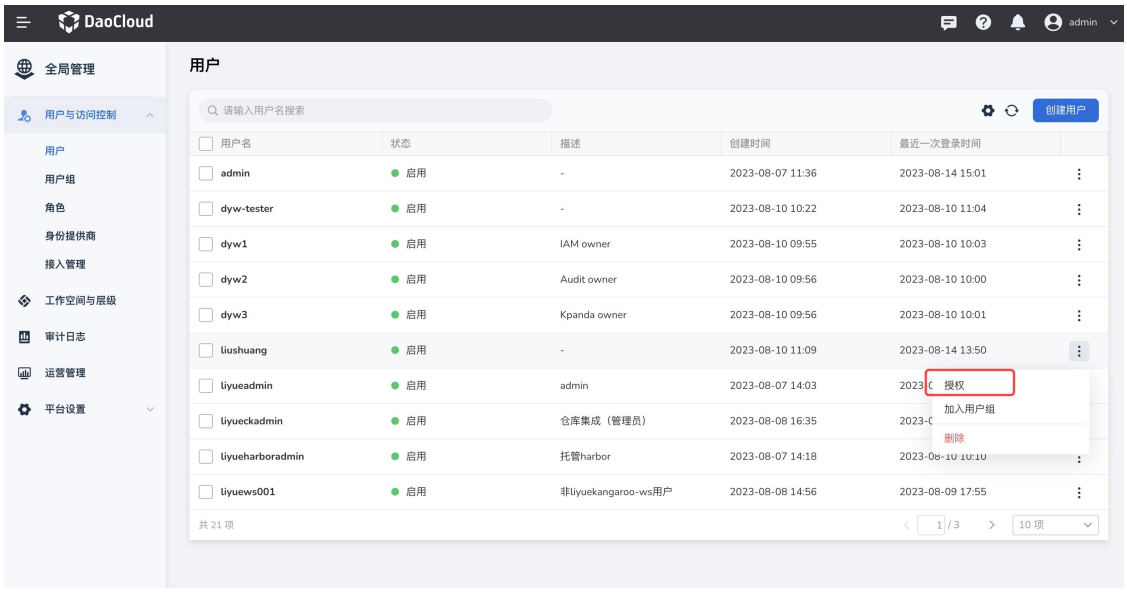


权限列表

平台角色授权方式

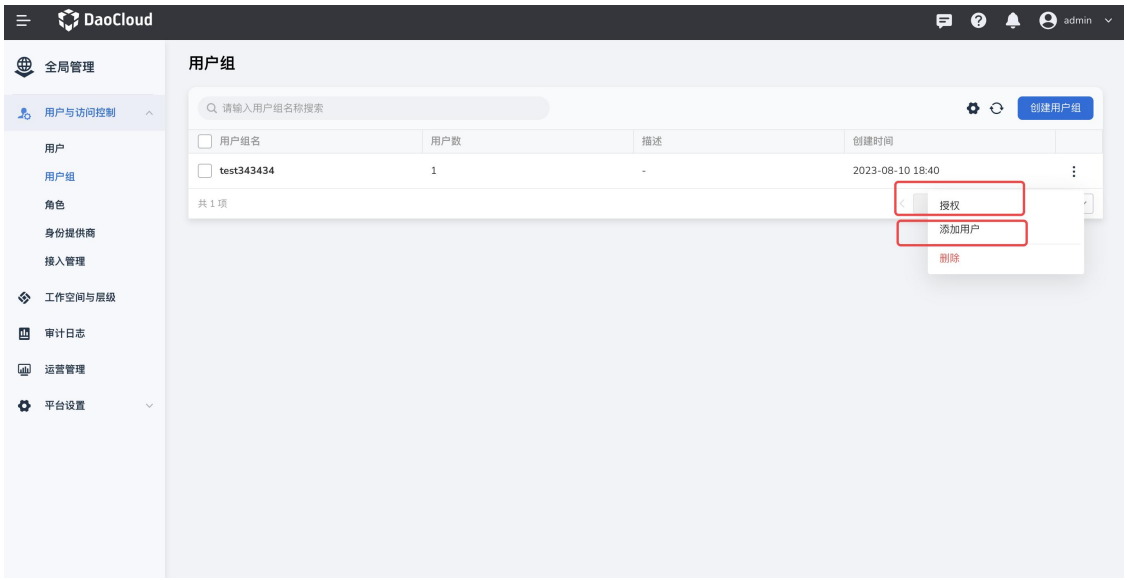
给平台角色授权共有三种方式：

- 在全局管理 -> 用户与访问控制 -> 用户的用户列表中，找到该用户，点击 ...，选择授权，为该用户赋予平台角色权限。



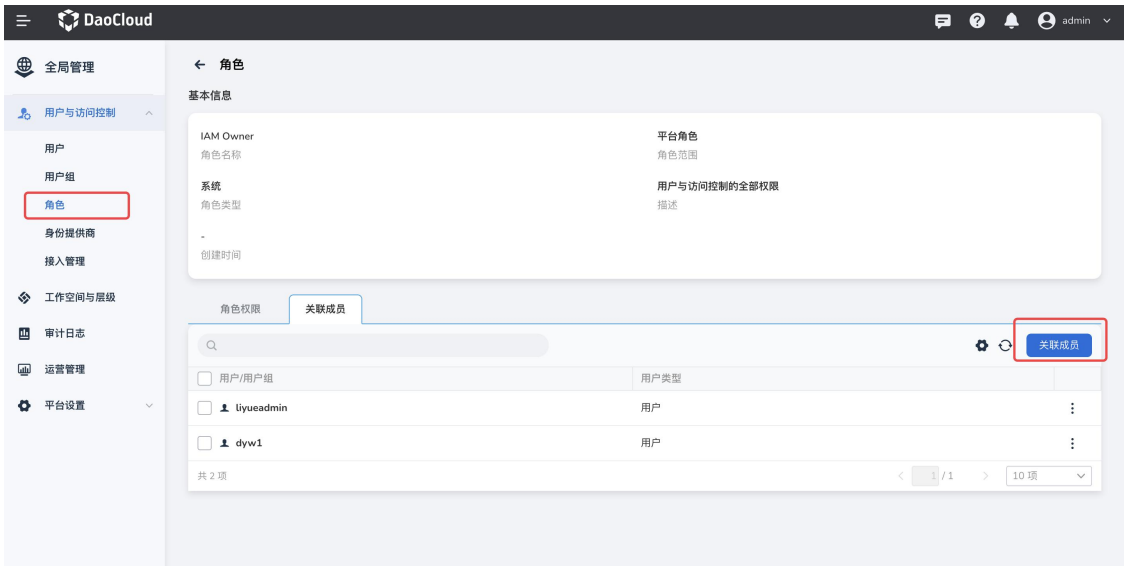
点击授权

- 在全局管理 -> 用户与访问控制 -> 用户组的用户组列表中创建用户组，将该用户加入用户组，并给用户组授权 （具体操作为：在用户组列表找到该用户组，点击 ...，选择授权，为该用户组赋予平台角色）。



点击授权

- 在全局管理 –> 用户与访问控制 –> 角色的角色列表中，找到相应的平台角色， 点击角色名称进入详情，点击关联成员 按钮，选中该用户或用户所在的用户组，点击确定。

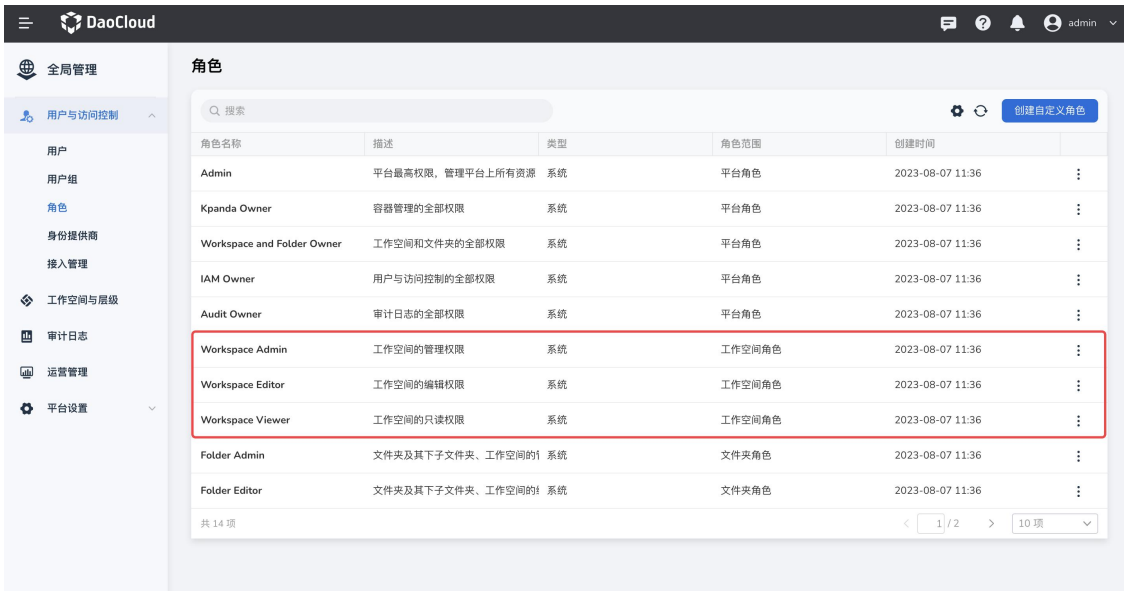


关联成员按钮

工作空间角色

工作空间角色是细粒度角色，通过工作空间角色可以赋予用户某个工作空间的管理权限、查看权限或该工作空间应用工作台相关的权限等。 获得该角色权限的用户只能管理该工作空间，而无法访问其他工作空间。DCE 5.0 提供了 3 个预置的、用户可直接使用的工作空间角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer



3 种工作空间预置角色

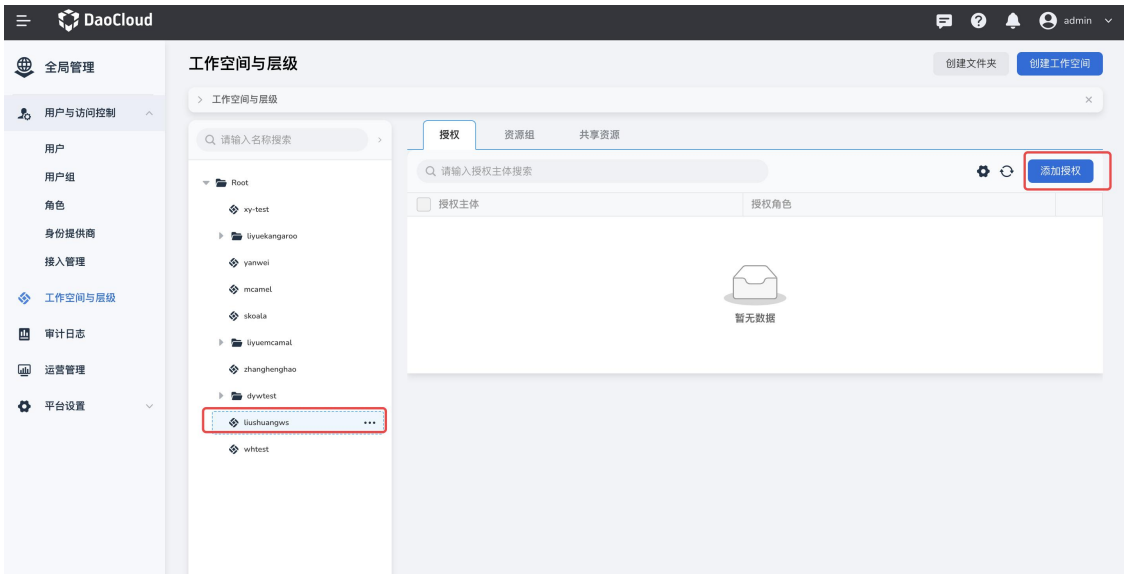
同时，DCE 5.0 还支持用户创建自定义工作空间角色，可根据需要自定义角色内容。如创建一个工作空间角色， 包含应用工作台的所有功能权限，由于应用工作台依赖于工作空间，因此平台会帮助用户默认勾选工作空间的查看权限， 请不要手动取消勾选。若用户 A 在工作空间 01 中被授予该角色，将拥有工作空间 01 下的应用工作台相关功能的增删改查权限。

!!! note

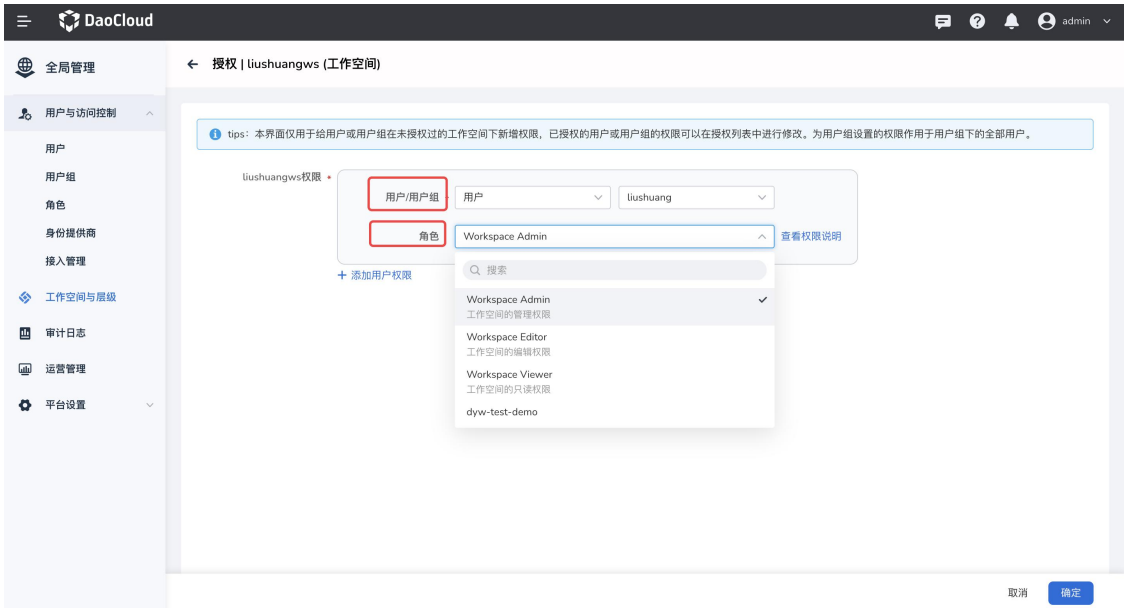
与平台角色不同，工作空间角色被创建后需要前往工作空间使用，被授权后用户仅在该工作空间下拥有该角色中的功能权限。

工作空间角色授权方式

在全局管理 -> 工作空间与层级 列表中，找到该工作空间，点击添加授权，为该用户赋予工作空间角色权限。



添加授权按钮

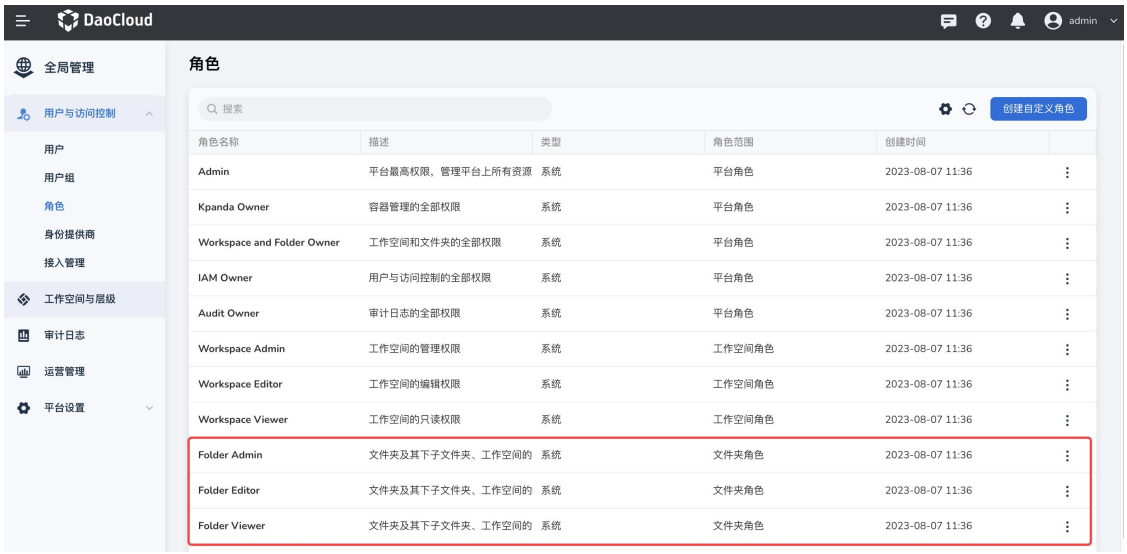


填写和选择

文件夹角色

文件夹角色的权限粒度介于平台角色与工作空间角色之间，通过文件夹角色可以赋予用户某个文件夹及其子文件夹和该文件夹下所有工作空间的管理权限、查看权限等， 常适用于企业中的部门场景。比如用户 B 是一级部门的 Leader，通常用户 B 能够管理该一级部门、其下的所有二级部门和部门中的项目等， 在此场景中给用户 B 授予一级文件夹的管理员权限，用户 B 也将拥有其下的二级文件夹和工作空间的相应权限。 DCE 5.0 提供了 3 个预置的、用户可直接使用文件夹角色：

- Folder Admin
- Folder Editor
- Folder Viewer



3 种预置文件夹角色

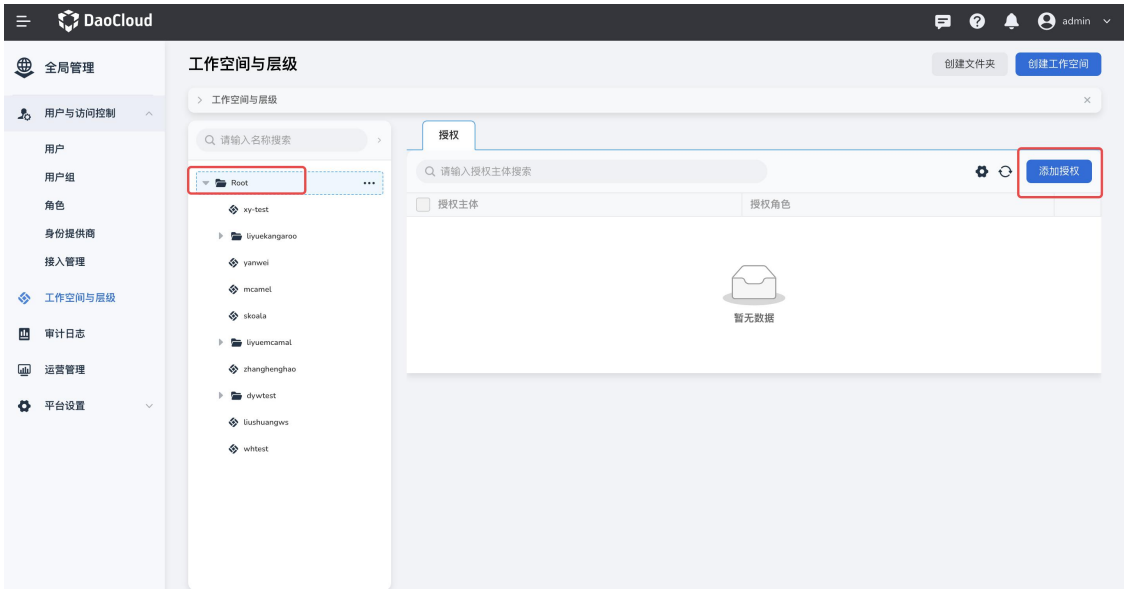
同时，DCE 5.0 还支持用户创建自定义文件夹角色，可根据需要自定义角色内容。 如创建一个文件夹角色，包含应用工作台的所有功能权限。若用户 A 在文件夹 01 中被授予该角色， 将拥有该文件夹下所有工作空间中应用工作台相关功能的增删改查权限。

!!! note

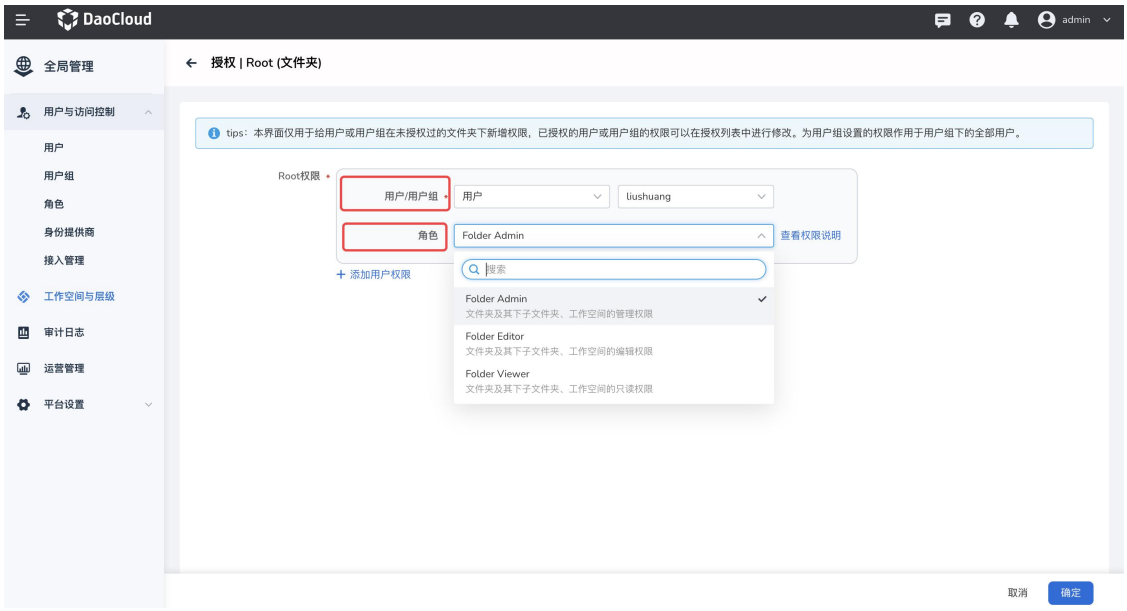
功能模块本身依赖的是工作空间，文件夹是工作空间上的进一步分组机制且具有权限继承能力，
因此文件夹权限不光包含文件夹本身，还包括其下的子文件夹和工作空间。

文件夹角色授权方式

在全局管理 –> 工作空间与层级列表中，找到该文件夹，点击添加授权，为该用户赋予文件夹角色权限。



添加授权按钮


DaoCloud
admin
全局管理
用户与访问控制
用户
用户组
角色
身份提供商
接入管理
工作空间与层级
审计日志
运营管理
平台设置
授权 | Root (文件夹)
tips：本界面仅用于给用户或用户组在未授权过的文件夹下新增权限，已授权的用户或用户组的权限可以在授权列表中进行修改。为用户组设置的权限作用于用户组下的全部用户。
Root权限
用户/用户组
用户
liushuang
角色
Folder Admin
查看权限说明
添加用户权限
搜索
Folder Admin
文件夹及其下子文件夹、工作空间的管理权限
Folder Editor
文件夹及其下子文件夹、工作空间的编辑权限
Folder Viewer
文件夹及其下子文件夹、工作空间的只读权限
取消
确定

填写和选择

如何将 DCE 5.0 集成到客户平台（OEM OUT）

OEM OUT 是指将 DCE 5.0 作为子模块接入其他产品，出现在其他产品的菜单中。 用户登录其他产品后可直接跳转至 DCE 5.0 无需二次登录。实现 OEM OUT 共分为 5 步，分别是：

- 统一域名
- 打通用户体系
- 对接导航栏
- 定制外观
- 打通权限体系(可选)

具体操作演示请参见 OEM OUT 最佳实践视频教程。

统一域名

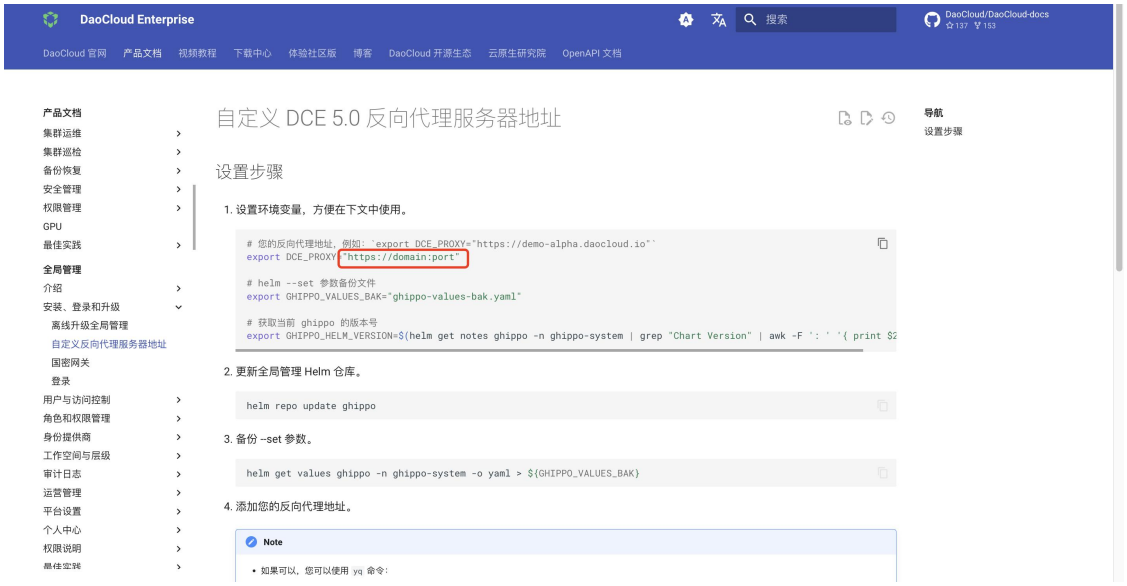
1. 部署 DCE 5.0（假设部署完的访问地址为 https://10.6.8.2:30343/）
2. 客户系统和 DCE 5.0 前可以放一个 Nginx 反代来实现同域访问， / 路由到客户系统，/dce5 (subpath) 路由到 DCE 5.0 系统，vi /etc/nginx/conf.d/default.conf 示例如下：

```
server {
    listen          80;
    server_name    localhost;

    location /dce5/ {
      proxy_pass https://10.6.8.2:30343/;
      proxy_http_version 1.1;
      proxy_read_timeout 300s; # 如需要使用 kpanda cloudtty 功能需要这行，否则可以去掉
      proxy_send_timeout 300s; # 如需要使用 kpanda cloudtty 功能需要这
```

行，否则可以去掉

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        proxy_set_header Upgrade $http_upgrade; # 如需要使用 kpanda cloudtty 功能需要这行，否则可以去掉
        proxy_set_header Connection $connection_upgrade; # 如需要使用 kpanda cloudtty 功能需要这行，否则可以去掉
    }

    location / {
        proxy_pass https://10.6.165.50:30443/; # 假设这是客户系统地址(如意云)
        proxy_http_version 1.1;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

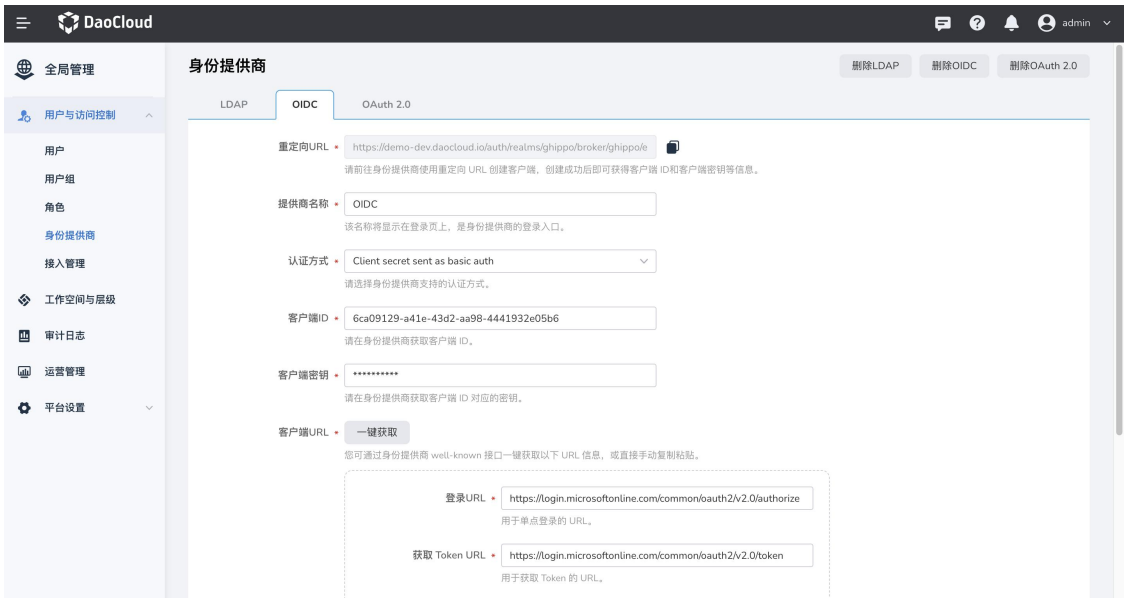
3. 假设 Nginx 入口地址为 10.6.165.50，按自定义 DCE 5.0 反向代理服务器地址把 DCE_PROXY 反代设为 http://10.6.165.50/dce5。确保能够通过 http://10.6.165.50/dce5 访问 DCE 5.0。 客户平台也需要进行反代设置，需要根据不同平台的情况进行处理。



反向代理

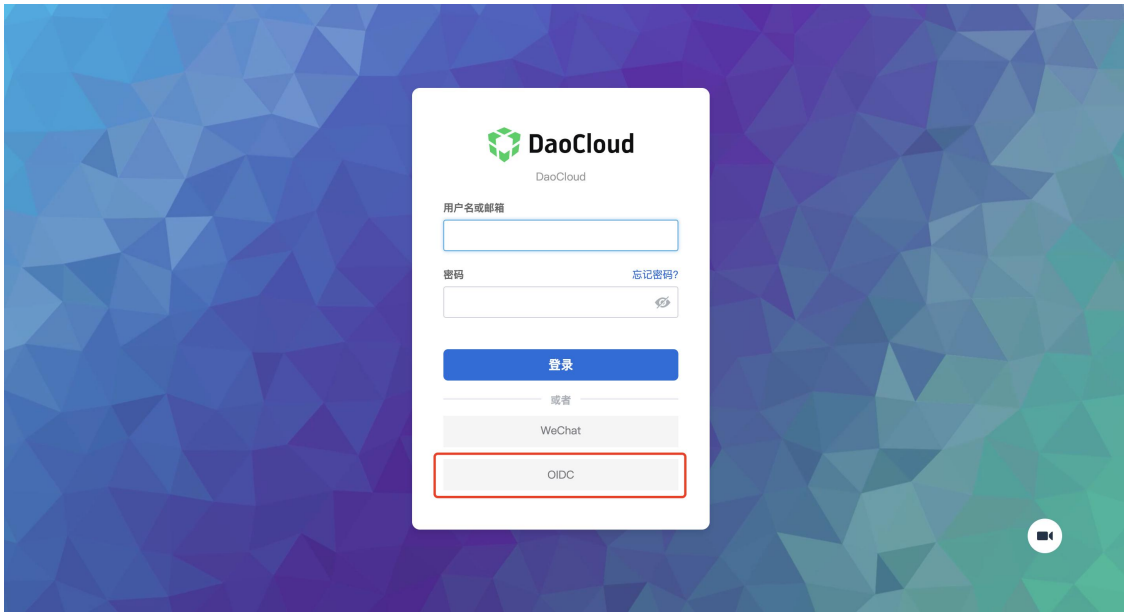
打通用户体系

将客户平台与 DCE 5.0 平台通过 OIDC/OAUTH 等协议对接，使用户登录客户平台后进入 DCE 5.0 时无需再次登录。 在拿到客户平台的 OIDC 信息后填入全局管理 –> 用户与访问控制 –> 身份提供商中。



身份提供商

对接完成后，DCE 5.0 登录页面将出现 OIDC（自定义）选项，首次从客户平台进入 DCE 5.0 时选择通过 OIDC 登录， 后续将直接进入 DCE 5.0 无需再次选择。



登录页

对接导航栏

对接导航栏是指 DCE 5.0 出现在客户平台的菜单中，用户点击相应的菜单名称能够直接进入 DCE 5.0。 因此对接导航栏依赖于客户平台，不同平台需要按照具体情况进行处理。

定制外观

通过全局管理 -> 平台设置 -> 外观定制可以自定义平台背景颜色、logo、名称等， 具体操作请参照外观定制。

打通权限体系（可选）

打通权限较为复杂，如有需求请联系全局管理团队。

OEM IN

OEM IN 是指合作伙伴作为 Anyproduct 嵌入 DCE 5.0，作为 DCE 5.0 的子模块出现在一级导航栏。 DCE 5.0 为用户源，用户通过 DCE 5.0 进行登录和统一管理。

AnyProduct：当客户系统需要使用 DCE 5.0 作为身份提供商（用户源）， 将 DCE 5.0 作为门户入口时，该客户系统被称为 DCE 5.0 的 AnyProduct

如何实现 OEM IN

打通用户体系

用 DCE 5.0 作为用户源，实现统一登录认证，接入到导航栏，这一步是必需的。

具体操作步骤为：

1. 导航栏接入
2. AnyProduct 通过 OIDC 协议 接入 DCE 5.0 的用户系统。
3. （可选）如果 AnyProduct 需进行用户同步，DCE 5.0 可提供 Webhook 注册功能， 将每一次的用户变化事件通过 Webhook 的方式通知到 AnyProduct。

嵌入 DCE 5.0 界面并将 DCE 5 作为门户

这一步操作可选。

前提条件：AnyProduct 系统本身需要支持 url 加前缀来访问（以 DX-ARCH 为例，需要给 url 加前缀）

具体操作步骤如下：

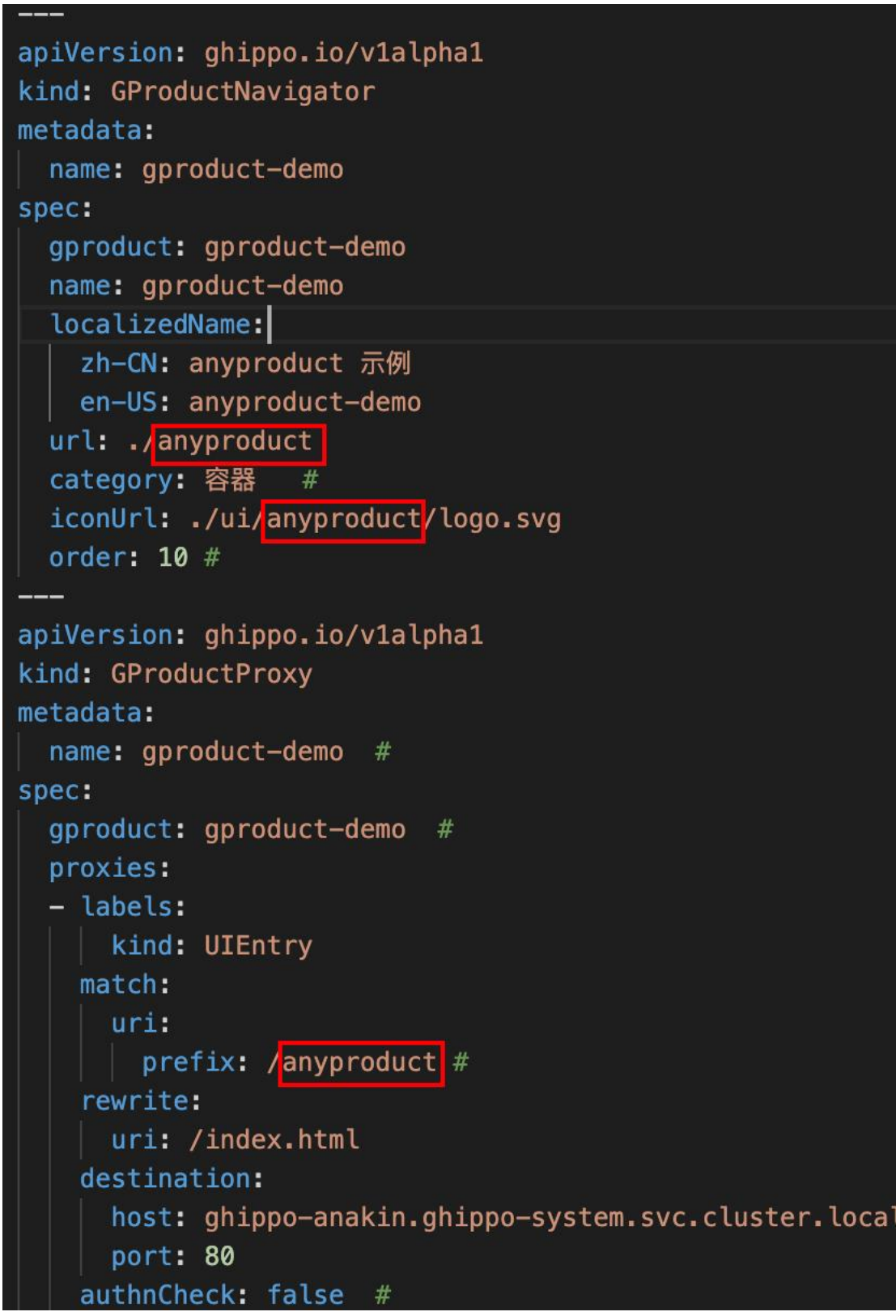
1. 部署 DCE 5.0（假设部署完的访问地址为 https://10.6.165.50:30443/ ）， 部署 AnyProduct 系统（假设 DX-ARCH 地址为 https://10.6.165.2:30034/ ）
2. DCE 5.0 和 AnyProduct 前可放一个 Nginx 反代来实现同域访问，/ 路由到 DCE 5.0， /dx-arch/ 路由到 AnyProduct 系统。
    1. 配置 /etc/nginx/conf.d/default.conf
    2. 配置 /etc/nginx/nginx.conf

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;
    map $http_upgrade $connection_upgrade {
      default upgrade;
      ''      close;
    }

    include /etc/nginx/conf.d/*.conf;
}
```

var

3. 假设 nginx 入口地址为 10.6.165.50，按设置 DCE 5.0 反向代理步骤把 DCE_PROXY 设置为 http://10.6.165.50/
4. 参考全局管理 GProduct 对接参考文档来实现一个空壳的 GProduct 前端子应用，把 DX-ARCH 以 iframe 形式放进该空壳应用里。

1. git clone 这个仓库 https://gitlab.daocloud.cn/henry.liu/gproduct-demo
2. 在 App-iframe.vue 中，把 src 属性值改成你需要的相对地址，如 src="./dx-arch"， 如果想跳转到 http://10.6.165.50/dx-arch/ram/config/user，可以设置为 src="./dx-arch/ram/config/user"
3. 删除 App.vue 和 main.ts，将 App-iframe.vue 重命名成 App.vue，将 main-iframe.ts 重命名成 main.ts
4. 如你希望用浏览器访问该 dx-arch 时使用 anyproduct 作为 subpath，如 http://{dce5_domain}/anyproduct，需改如下内容：
    1. 在 nginx.conf 中，将 location ~ /ui/demo/(._)$ 改成 location ~ /ui/anyproduct/(._)$
    2. 按 gproduct-demo 步骤 build 出 image
    3. 在 demo.yaml 中，将 image 改成 build 出来的名字，这三个地方的 demo 改成 anyproduct：



```yaml
---
apiVersion: ghippo.io/v1alpha1
kind: GProductNavigator
metadata:
  name: gproduct-demo
spec:
  gproduct: gproduct-demo
  name: gproduct-demo
  localizedName:
    zh-CN: anyproduct 示例
    en-US: anyproduct-demo
  url: ./anyproduct
  category: 容器    #
  iconUrl: ./ui/anyproduct/logo.svg
  order: 10 #
---
apiVersion: ghippo.io/v1alpha1
kind: GProductProxy
metadata:
  name: gproduct-demo  #
spec:
  gproduct: gproduct-demo  #
  proxies:
  - labels:
      kind: UIEntry
    match:
      uri:
        prefix: /anyproduct #
    rewrite:
      uri: /index.html
    destination:
      host: ghippo-anakin.ghippo-system.svc.cluster.local
      port: 80
    authnCheck: false  #
```

yaml

4. 把 demo.yaml 和 image 放到 DCE 5.0 集群上执行 apply 命令

```
docker save anyproduct-dx-arch:v0.1.0 -o ./anyproduct-dx-arch-v0.1.0.tar
scp ./anyproduct-dx-arch-v0.1.0.tar root@10.6.165.51:~
ssh 到 51，docker load < anyproduct-dx-arch-v0.1.0.tar
```

用户信息同步(可选)

方案思路为：

通过 Webhook 方式，将每一次的用户变化都通知到 AnyProduct。

租户打通(可选)

方案思路为：

通过 webhook 方式，将每一次的租户变化都通知到 AnyProduct (如有需求，后续可实现)

权限打通(可选)

方案思路一：

定制化团队可实现一定制模块，DCE 5 将每一次的用户登录事件通过 Webhook 的方式通知到定制模块， 定制模块可自行调用 AnyProduct 和 DCE 5.0 的 OpenAPI 作该用户的权限信息同步。

方案思路二：

通过 Webhook 方式，将每一次的授权变化都通知到 AnyProduct (如有需求，后续可实现)。

AnyProduct 使用 DCE 5.0 的其他能力(可选)

方法为：调用 DCE 5.0 OpenAPI

参考资料

- 参考 OEM OUT 文档
- 参阅 GProduct-demo 对接 tar 包

gproduct-demo

搭建环境

npm install

编译和热加载开发环境：

npm run serve

编译和构建：

npm run build

补全 Lint 检查文件：

npm run lint

自定义配置

参见配置参考。

构建镜像：

docker build -t release.daocloud.io/henry/gproduct-demo .

在 K8s 上运行：

kubectl apply -f demo.yaml

Webhook 消息通知

DCE 5.0 在接入客户的系统后，可以创建 Webhook，在用户创建/更新/删除/登录/登出之时发送消息通知。

Webhook 是一种用于实现实时事件通知的机制。它允许一个应用程序将数据或事件推送到另一个应用程序， 而无需轮询或持续查询。通过配置 Webhook，您可以指定在某个事件发生时，由目标应用程序接收并处理通知。

Webhook 的工作原理如下：

1. 源应用程序（DCE 5.0）执行某个特定操作或事件。
2. 源应用程序将相关数据和信息打包成 HTTP 请求，并将其发送到目标应用程序指定的 URL（例如企业微信群机器人）。
3. 目标应用程序接收到请求后，根据其中的数据和信息进行相应的处理。

通过使用 Webhook，您可以实现以下功能：

- 实时通知：当某个特定事件发生时，通过 Webhook 及时通知其他应用程序。
- 自动化处理：目标应用程序可以根据收到的 Webhook 请求自动触发事先定义好的操作，无需手动干预。
- 数据同步：通过 Webhook 将数据从一个应用程序传递到另一个应用程序，实现数据的同步更新。

常见的应用场景包括：

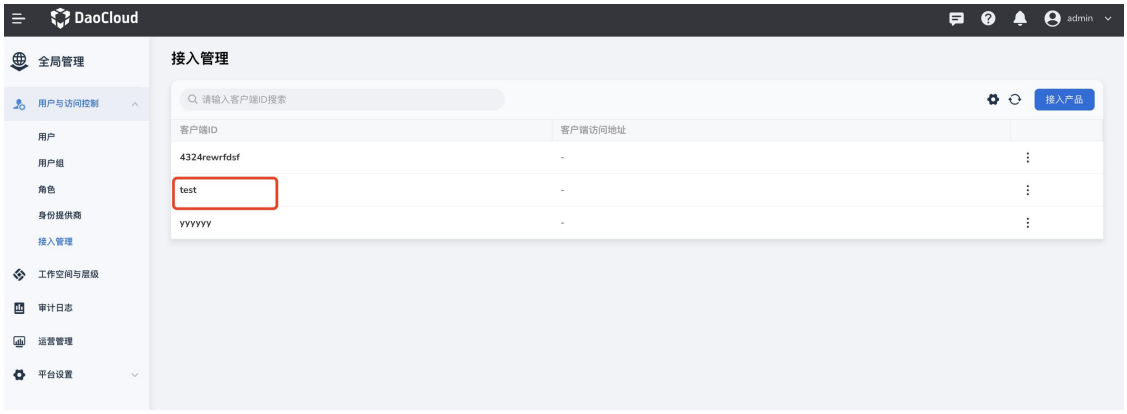
- 版本控制系统（例如 GitHub、GitLab）中，当代码仓库发生变动时，自动触发构建和部署操作。
- 电子商务平台中，当订单状态发生变化时，向物流系统发送更新通知。

- 聊天机器人平台中，当接收到用户消息时，通过 Webhook 将消息推送到目标服务器进行处理。

配置步骤

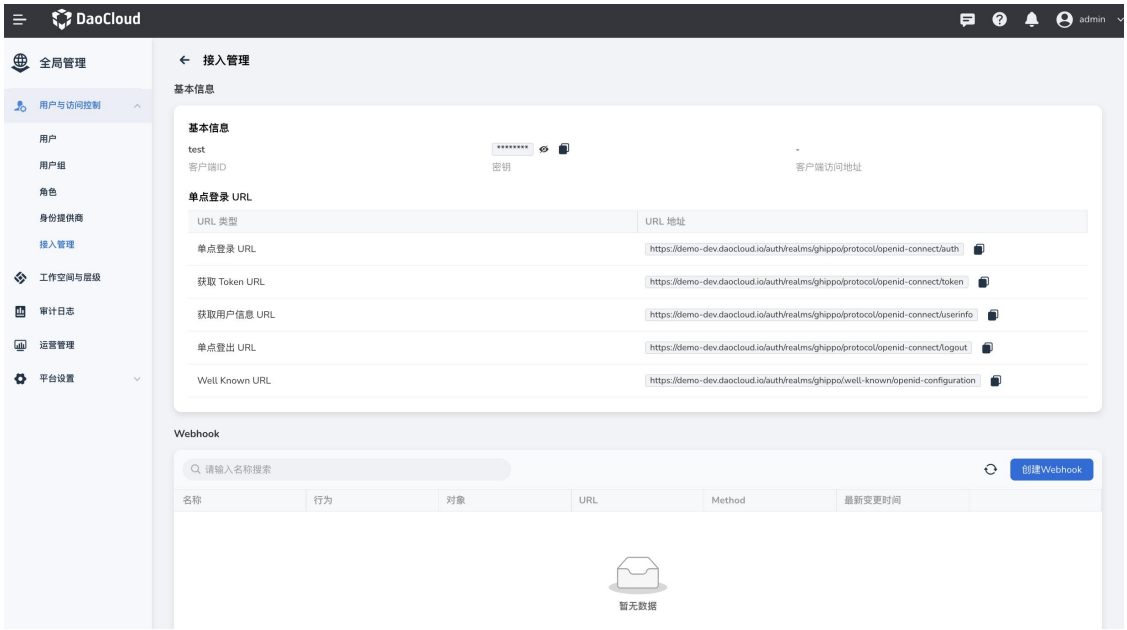
DCE 5.0 图形化配置 Webhook 的操作步骤如下：

1. 在全局管理 -> 用户与访问控制 -> 接入管理，创建一个客户端 ID。



oem in

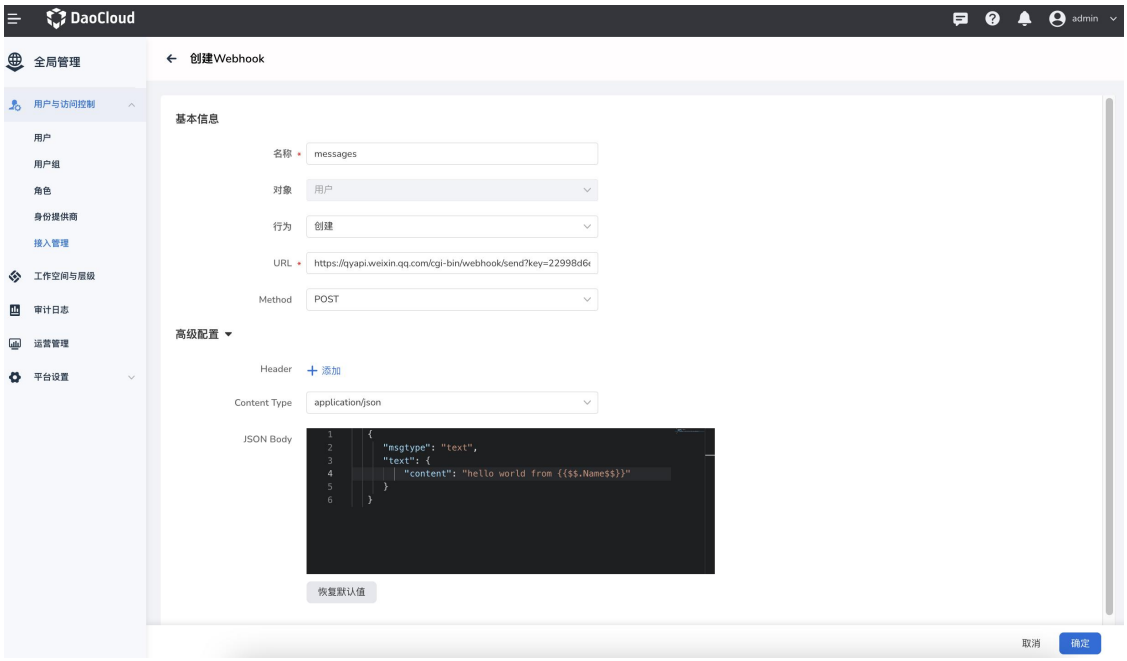
2. 点击某个客户端 ID，进入详情页，点击创建 Webhook 按钮。



button

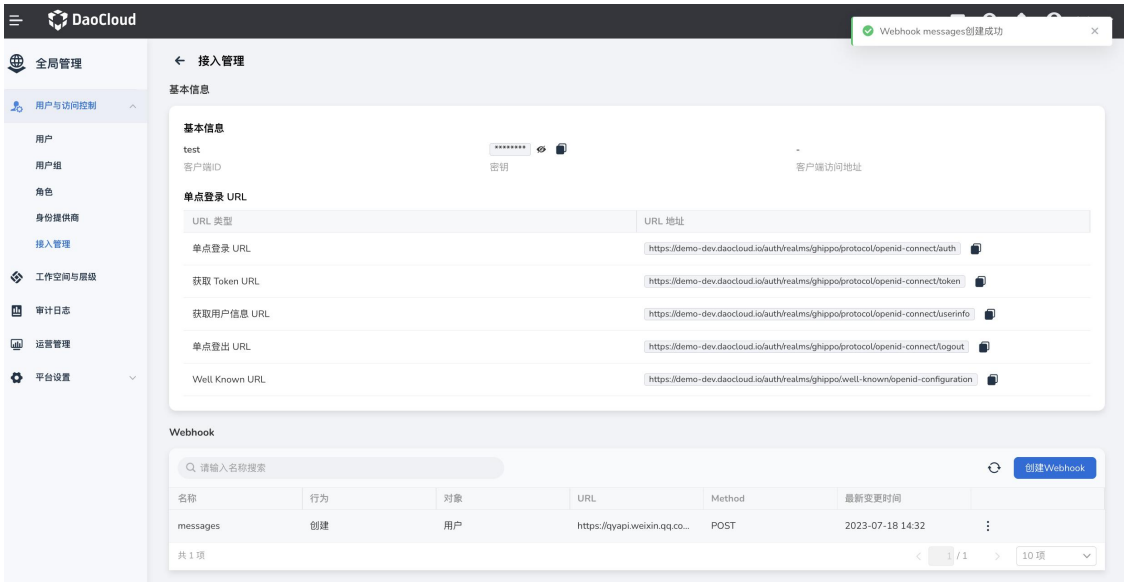
3. 在弹窗中填入字段信息后点击确定。

   - 对象：目前仅支持用户对象
   - 行为：用户创建/更新/删除/登录/登录时发送 Webhook 消息
   - URL：接收消息的地址
   - Method：视情况选择适用的方法，例如企业微信推荐使用 POST 方法
   - 高级配置：可以用 Json 编写消息体。如果是企业微信群，请参阅群机器人配置说明



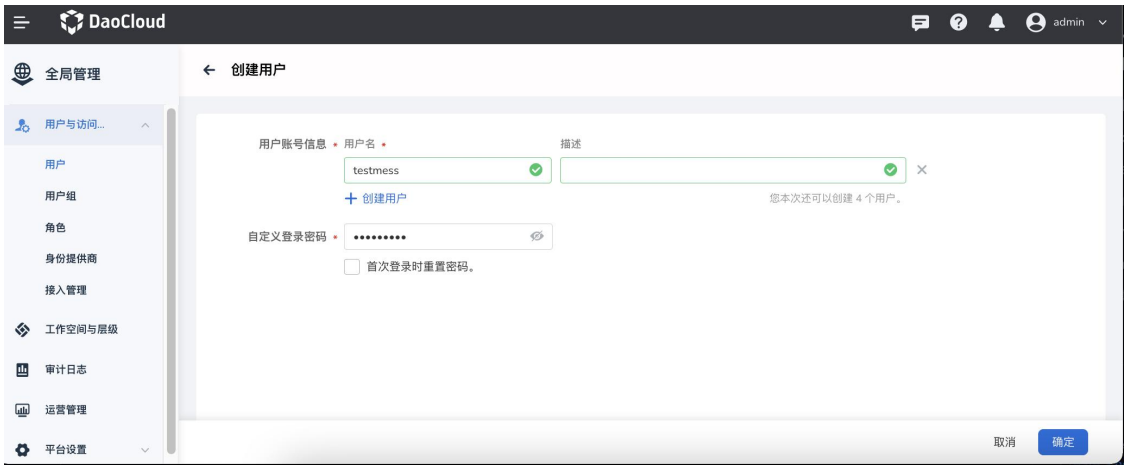
fill

4. 屏幕提示 Webhook 创建成功。



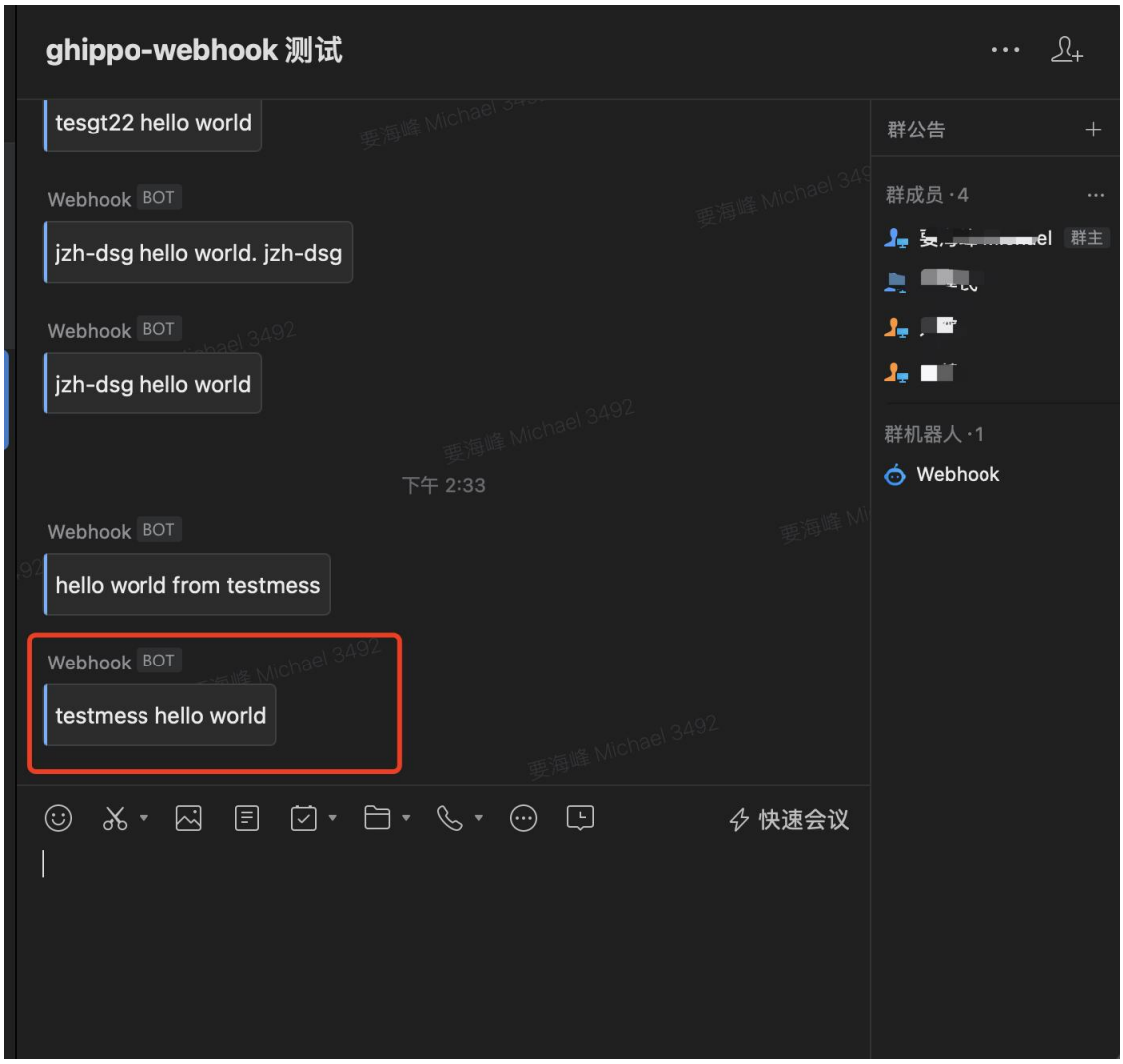
success

5. 现在去试着创建一个用户。



create

6. 用户创建成功，可以看到企业微信群收到了一条消息。



message

高级配置示例

系统默认的消息体

DCE 5.0 预先定义了一些变量，您可以根据自己情况在消息体中使用这些变量。

```
{
  "id": "{{$$.ID$$}}",
  "email": "{{$$.Email$$}}",
  "username": "{{$$.Name$$}}",
  "last_name": "{{$$.LastName$$}}",
```

```
  "first_name": "{{$$.FirstName$$}}",
  "created_at": "{{$$.CreatedAt$$}}",
  "enabled": "{{$$.Enabled$$}}"
}
```

企业微信群机器人的 Message Body

```
{
    "msgtype": "text",
    "text": {
        "content": "{{$$.Name$$}} hello world"
    }
}
```

Keycloak 自定义 IdP

要求：keycloak >= v20

已知问题 keycloak >= v21，删除了旧版 theme 的支持，可能会在 v22 修复。参见 Issue #15344。

此次 demo 使用 Keycloak v20.0.5。

基于 source 开发

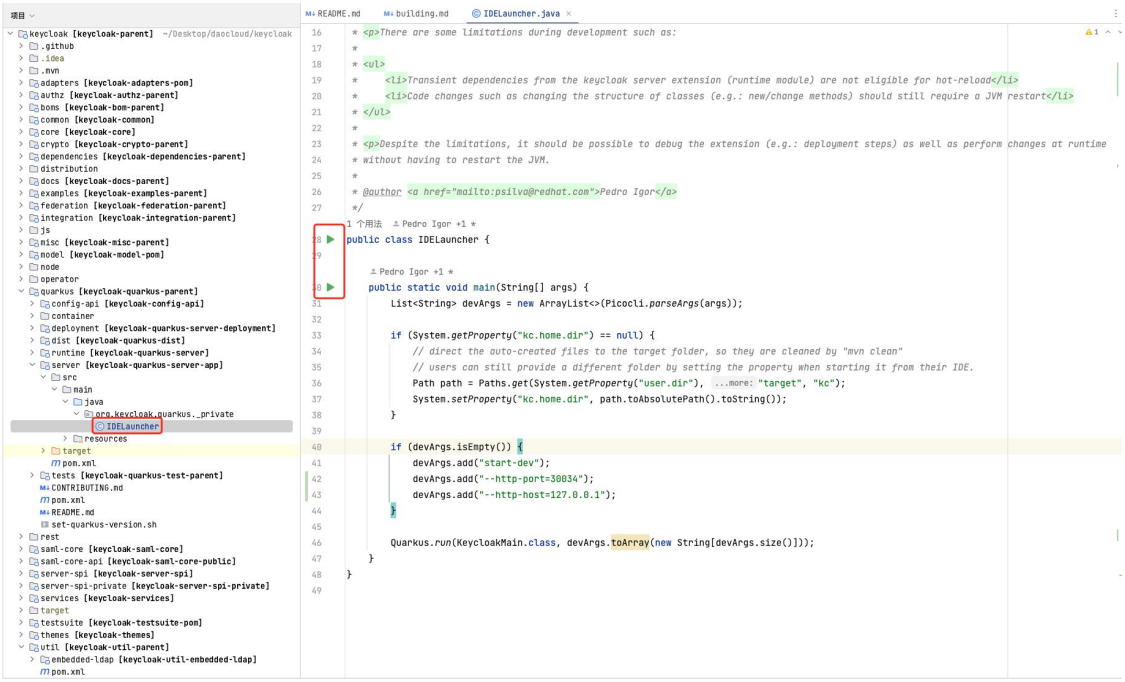
配置环境

参照 keycloak/building.md 配置环境。

参照 keycloak/README.md 运行以下命令：

```
cd quarkus
mvn -f ../pom.xml clean install -DskipTestsuite -DskipExamples -DskipTests
```

从 IDE 运行



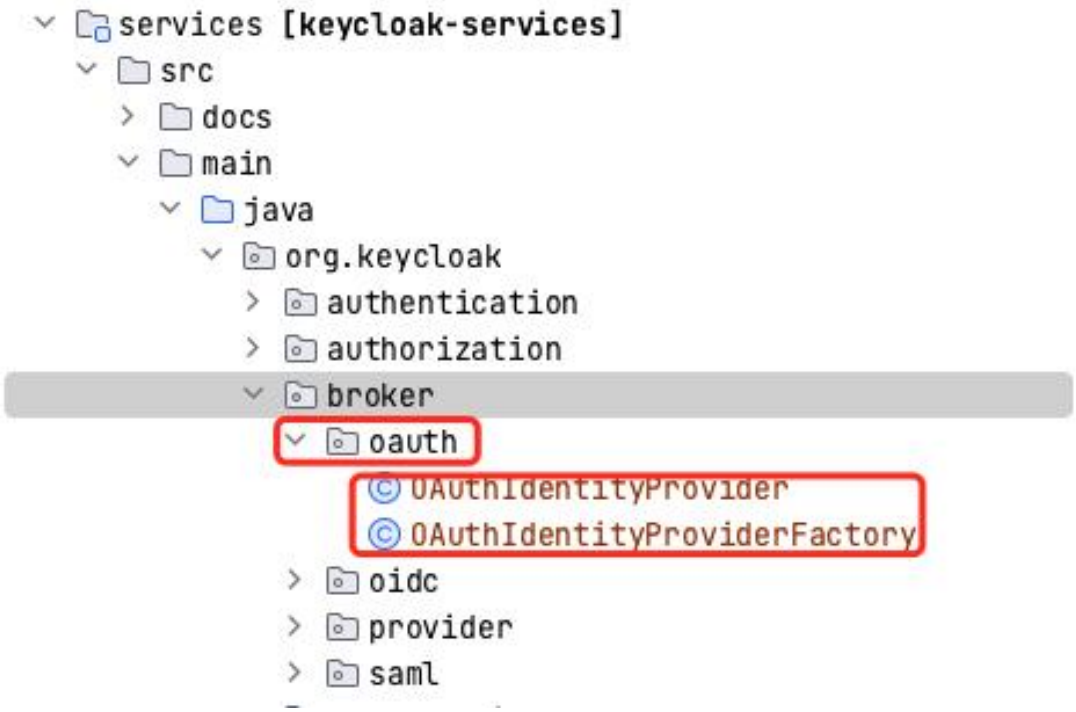
从 IDE 运行

添加 service 代码

如果可从 keycloak 继承部分功能

在目录 services/src/main/java/org/keycloak/broker 下添加文件：

文件名需要是 xxxProvider.java 和 xxxProviderFactory.java



java

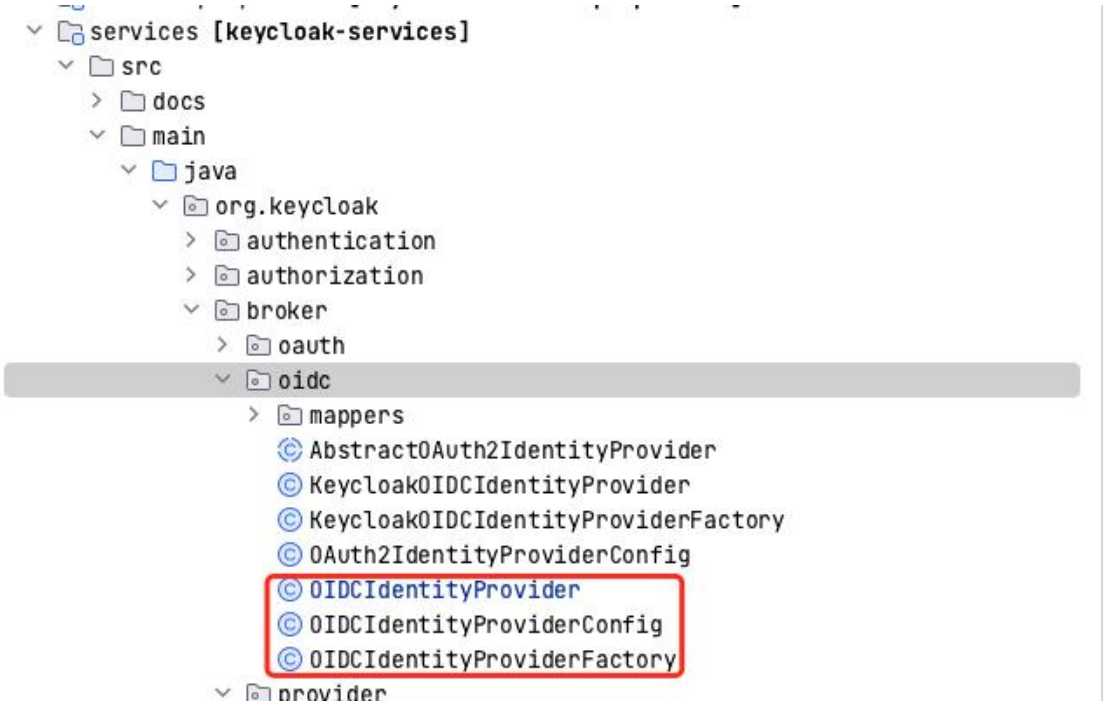
xxxProviderFactory.java 示例：

留意 PROVIDER_ID = "oauth"；这个变量，后面定义 html 会用到。

xxxProvider.java 示例

如果不能从 keycloak 继承功能

参考下图中的三个文件编写你的代码：



none heritance

添加 xxxProviderFactory 到 resource service

在 services/src/main/resources/META-INF/services/org.keycloak.broker.provider.IdentityProviderFactory 添加 xxxProviderFactory，这样刚刚编写的能工作了：

```
#
# Copyright 2016 Red Hat, Inc. and/or its affiliates
# and other contributors as indicated by the @author tags.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

org.keycloak.broker.oidc.OIDCIdentityProviderFactory
org.keycloak.broker.oidc.KeycloakOIDCIdentityProviderFactory
org.keycloak.broker.saml.SAMLIdentityProviderFactory
org.keycloak.broker.oauth.OAuthIdentityProviderFactory
```

running

添加 html 文件

复制 themes/src/main/resources/theme/base/admin/resources/partials/realm-identity-provider-oidc.html 文件到（改名为 realm-identity-provider-oauth.html，还记得上文中需要留意的变量吗）
themes/src/main/resources/theme/base/admin/resources/partials/realm-identity-provider-oauth.html

到此所有的文件都添加完成了，开始调试功能。

打包成 jar 作为插件运行

新建一个 java 项目，并将上面的代码复制到项目中，如下所示：

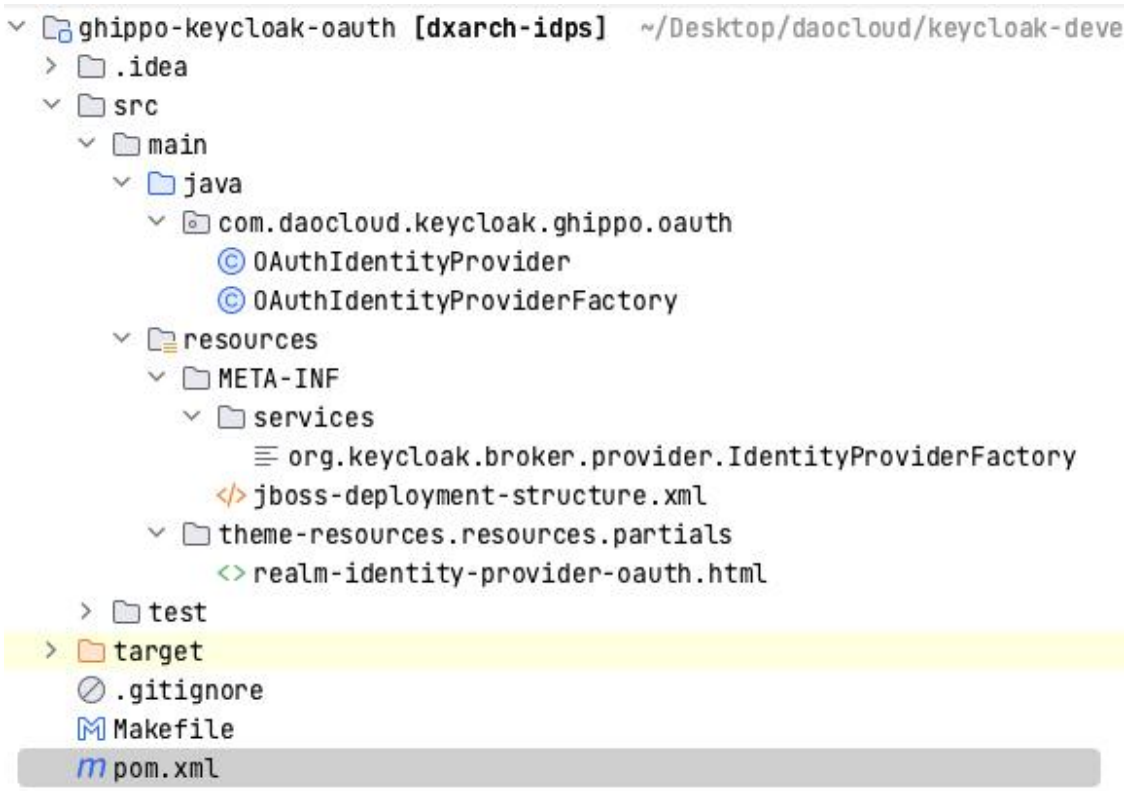
```
#
# Copyright 2016 Red Hat, Inc. and/or its affiliates
# and other contributors as indicated by the @author tags.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

org.keycloak.broker.oidc.OIDCIdentityProviderFactory
org.keycloak.broker.oidc.KeycloakOIDCIdentityProviderFactory
org.keycloak.broker.saml.SAMLIdentityProviderFactory
org.keycloak.broker.oauth.OAuthIdentityProviderFactory
```

pom

参见 pom.xml。

运行 mvn clean package，打包完成得到 xxx-jar-with-dependencies.jar 文件。

下载 keycloak Release 20.0.5 zip 包并解压。



release

将 xxx-jar-with-dependencies.jar 复制到 keycloak-20.0.5/providers 目录中。

运行以下命令查看功能是否完整：

bin/kc.sh start-dev

定制 DCE 5.0 对接外部身份提供商（IdP)

身份提供商（IdP, Identity Provider）：当 DCE 5.0 需要使用客户系统作为用户源， 使用客户系统登录界面来进行登录认证时，该客户系统被称为 DCE 5 的身份提供商

适用场景

如果客户对 Ghippo 登录 IdP 有高度定制需求，例如支持企业微信、微信等其他社会组织登录需求，请根据本文档实施。

支持版本

Ghippo 0.15.0 及以上版本。

具体方法

自定义 ghippo keycloak plugin

1. 定制 plugin

   参考 keycloak 官方文档和 keycloak 自定义 IdP 进行开发。

2. 构建镜像

   ```
   # FROM scratch
   FROM scratch


   # plugin
   COPY ./xxx-jar-with-dependencies.jar /plugins/
   ```

!!! note

如果需要两个定制化 IdP，需要复制两个 jar 包。

部署 ghippo keycloak plugin 步骤

1. 把 ghippo 升级到 0.15.0 或以上。 您也可以直接安装部署 ghippo 0.15.0 版本，但需要把以下信息手动记录下来。

```
helm -n ghippo-system get values ghippo -o yaml
```

```
apiserver:
  image:
    repository: release.daocloud.io/ghippo-ci/ghippo-apiserver
    tag: v0.4.2-test-3-gaba5ec2
controllermanager:
  image:
    repository: release.daocloud.io/ghippo-ci/ghippo-apiserver
    tag: v0.4.2-test-3-gaba5ec2
global:
  database:
    builtIn: true
  reverseProxy: http://192.168.31.10:32628
```

2. 升级成功后，手工跑一个安装命令，--set 里设的参数值从上述保存的内容里得到，并且外加几个参数值：

   - global.idpPlugin.enabled：是否启用定制 plugin，默认已关闭
   - global.idpPlugin.image.repository：初始化自定义 plugin 的 initContainer 用的 image 地址
   - global.idpPlugin.image.tag：初始化自定义 plugin 的 initContainer 用的 image tag
   - global.idpPlugin.path：自定义 plugin 的目录文件在上述 image 里所在的位置

   具体示例如下：

```
helm upgrade \
    ghippo \
```

```
    ghippo-release/ghippo \
    --version v0.4.2-test-3-gaba5ec2 \
    -n ghippo-system \
    --set apiserver.image.repository=release.daocloud.io/ghippo-ci/ghippo
-apiserver \
    --set apiserver.image.tag=v0.4.2-test-3-gaba5ec2 \
    --set controllermanager.image.repository=release.daocloud.io/ghippo-c
i/ghippo-apiserver \
    --set controllermanager.image.tag=v0.4.2-test-3-gaba5ec2 \
    --set global.reverseProxy=http://192.168.31.10:32628 \
    --set global.database.builtIn=true \
    --set global.idpPlugin.enabled=true \
    --set global.idpPlugin.image.repository=chenyang-idp \
    --set global.idpPlugin.image.tag=v0.0.1 \
    --set global.idpPlugin.path=/plugins/.
```

3. 在 keycloak 管理页面选择所要使用的插件。

应用工作台权限说明

应用工作台支持三种用户角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer

每种角色具有不同的权限，具体说明如下。

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
应用	查看应用列表	√	√	√
	查看详情（跳转到容器管理）	√	√	√
	查看应用日志（跳转到可观测）	√	√	√
	查看应用监控（跳转到可观测）	√	√	√
	查看 RabbitMQ 详情 – 基本信息	√	√	√
	查看服务网格（跳转到服务网格）	√	√	√
	查看微服务引擎（跳转到微服务引擎）	√	√	√
	创建应用	√	√	✗
	编辑 YAML	√	√	✗
	更新副本数量	√	√	✗

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	更新容器镜像	√	√	✗
	编辑流水线	√	√	✗
	应用分组	√	√	✗
	删除	√	√	✗
命名空间	查看	√	√	√
	创建	√	✗	✗
	编辑标签	√	✗	✗
	编辑资源配额	√	✗	✗
	删除	√	✗	✗
流水线	查看流水线	√	√	√
	查看运行记录	√	√	√
	创建	√	√	✗
	运行	√	√	✗
	删除	√	√	✗
	复制	√	√	✗
	编辑	√	√	✗
	取消运行	√	√	✗
凭证	查看	√	√	√
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
持续部署	查看	√	√	√
	创建	√	√	✗
	同步	√	√	✗
	编辑	√	√	✗
	删除	√	√	√
代码仓库	查看	√	√	✗
	导入	√	√	✗
	删除	√	√	✗
灰度发布	查看	√	√	√
	创建	√	√	✗
	发布	√	√	✗
	继续发布	√	√	✗
	终止发布	√	√	✗
	更新	√	√	✗
	回滚	√	√	✗
	删除	√	√	✗

!!! note

有关角色和权限管理的完整介绍，请参考[角色和权限管理](../user-guide/access-control/role.md)。

中间件数据服务权限说明

中间件数据服务包括六大中间件：MySQL、RabbitMQ、Elasticsearch、Kafka、Redis、MinIO。中间件数据服务支持三种用户角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer

每种角色具有不同的权限，具体说明如下。

中间件数据服务权限说明

中间件模块	菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
MySQL	MySQL 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗
	MySQL 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置参数	√	√	✗
		查看实例访问	√	√	✗

中间件模块	菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		密码			
		查看实例备份列表	√	√	√
		实例创建备份	√	√	✗
		实例修改自动备份任务	√	√	✗
		使用备份创建新实例	√	√	✗
	备份配置管理	备份配置列表	√	✗	✗
		创建备份配置	√	✗	✗
		修改备份配置	√	✗	✗
		删除备份配置	√	✗	✗
RabbitMQ	RabbitMQ 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗
	RabbitMQ 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置	√	√	✗

中间件模块	菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		参数			
		查看实例访问密码	√	√	✗
Elasticsearch	Elasticsearch 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗
	Elasticsearch 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置参数	√	√	✗
		查看实例访问密码	√	√	✗
Redis	Redis 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗

中间件模块	菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	Redis 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置参数	√	√	✗
		查看实例访问密码	√	√	✗
Kafka	Kafka 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗
	Kafka 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置参数	√	√	✗
		查看实例访问密码	√	√	✗

中间件模块	菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
MinIO	MinIO 实例列表	查看列表	√	√	√
		实例名称搜索	√	√	√
		创建实例	√	√	✗
		更新实例配置	√	√	✗
		删除实例	√	✗	✗
	MinIO 实例详情	实例概览	√	√	√
		实例监控	√	√	√
		查看实例配置参数	√	√	√
		修改实例配置参数	√	√	✗
		查看实例访问密码	√	√	✗

服务网格权限说明

服务网格支持以下几种用户角色：

- Admin
- Workspace Admin
- Workspace Editor
- Workspace Viewer

!!! info

自 DCE 5.0 的 [v0.6.0](../../download/index.md) 起，全局管理模块支持为服务网格配置自定义角色，即除了使用系统默认角色外，还可以在服务网格中自定义角色并授予不同权限。

这些角色的具体权限如下表所示。

菜单对象	操作	Admin	Workspace Admin	Workspace Editor	Workspace Viewer
服务网格列表	创建网格				
	更新网格				
	删除网格				
	查看网格列表				
网格概览	查看				
服务列表	查看				

菜单对象	操作	Admin	Workspace Admin	Workspace Editor	Workspace Viewer
服务条目	创建				
	编辑				
	删除				
	查看				
虚拟服务	创建				
	编辑				
	删除				
	查看				
目标规则	创建				
	编辑				
	删除				
	查看				
网关规则	创建				
	编辑				
	删除				
	查看				
对等身份认证	创建				
	编辑				
	删除				
	查看				

菜单对象	操作	Admin	Workspace Admin	Workspace Editor	Workspace Viewer
请求身份认证	创建				
	编辑				
	删除				
	查看				
授权策略	创建				
	编辑				
	删除				
	查看				
命名空间边车管理	启用注入				
	禁用注入				
	查看				
工作负载边车管理	启用注入				
	禁用注入				
	边车资源设置				
	查看				
全局边车注入	启用注入				
	禁用注入				

菜单对象	操作	Admin	Workspace Admin	Workspace Editor	Workspace Viewer
	边车资源设置				
	查看				
集群纳管(仅托管网格)	集群接入				
	集群移除				
	查看				
网格网关	创建				
	编辑				
	删除				
	查看				
Istio 资源管理	创建				
	编辑				
	删除				
	查看				
TLS 证书管理	创建				
	编辑				
	删除				
	查看				

菜单对象	操作	Admin	Workspace Admin	Workspace Editor	Workspace Viewer
多云网络互联	启用				
	查看				
	编辑				
	删除				
	关闭				
系统升级	Istio 升级				
	边车升级				
	查看				
工作空间管理	绑定				
	解绑				
	查看				

微服务引擎权限说明

微服务引擎包括微服务治理中心和微服务网关两部分。微服务引擎支持三种用户角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer

每种角色具有不同的权限，具体说明如下。

微服务治理中心权限说明

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
托管注册中心列表	查看列表	√	√	√
托管注册中心	查看基础信息	√	√	√
	创建	√	√	✗
	重启	√	√	✗
	编辑	√	√	✗
	删除	√	✗	✗
	上/下线	√	√	✗
微服务命名空间	查看	√	√	√
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗
微服务列表	查看	√	√	√

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	筛选命名空间	√	√	√
	创建	√	√	✗
	治理	√	√	✗
	删除	√	√	✗
服务治理规则–Sentinel	查看	√	√	√
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗
服务治理规则–Mesh	治理	√	√	✗
实例列表	查看	√	√	√
	上/下线	√	√	✗
	编辑	√	√	✗
服务治理策略–Sentinel	查看	√	√	√
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗
服务治理策略–Mesh	查看	√	√	√
	创建	√	√	✗
	YAML 创	√	√	✗

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	建			
	编辑	√	√	✗
	YAML 编辑	√	√	✗
	删除	√	√	✗
微服务配置列表	查看	√	√	√
	筛选命名空间	√	√	√
	批量删除	√	√	✗
	导出/导入	√	√	✗
	创建	√	√	✗
	克隆	√	√	✗
	编辑	√	√	✗
	历史查询	√	√	√
	回滚	√	√	✗
	监听查询	√	√	√
业务监控	查看	√	√	√
资源监控	查看	√	√	√
请求日志	查看	√	√	√
实例日志	查看	√	√	√
插件中心	查看	√	√	√
	开启	√	√	✗
	关闭	√	√	✗

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	编辑	√	√	✗
	查看详情	√	√	√
接入注册中心列表	查看	√	√	√
	接入	√	√	✗
	编辑	√	√	✗
	移除	√	✗	✗
微服务	查看列表	√	√	√
	查看详情	√	√	√
	治理	√	√	✗
服务治理策略–Mesh	查看	√	√	√
	创建	√	√	✗
	YAML 创建	√	√	✗
	编辑	√	√	✗
	YAML 编辑	√	√	✗
	删除	√	√	✗

微服务网关权限说明

对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
网关列表	查看	√	√	√
网关实例	查	√	√	√

对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	查看			
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	✗	✗
诊断模式	查看	√	√	√
	调试	√	√	✗
服务列表	查看	√	√	√
	添加	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗
服务详情	查看	√	√	√
服务来源管理	查看	√	√	√
	添加	√	√	✗

对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	编辑	√	√	✗
	删除	√	√	✗
API 列表	查看	√	√	√
	创建	√	√	✗
	编辑	√	√	✗
	删除	√	√	✗
请求日志	查看	√	√	√
实例日志	查看	√	√	√
插件中心	查看	√	√	√
	启用	√	√	✗
	禁用	√	√	✗
插件配置	查看	√	√	√
	启	√	√	✗

对象	操作用	Workspace Admin	Workspace Editor	Workspace Viewer
域名列表	查看	√	√	√
	添加	√	√	✗
	编辑	√	√	✗
	删除	√	√	√
监控告警	查看	√	√	√

!!! note

有关角色和权限管理的完整介绍，请参考[角色和权限管理](../user-guide/access-control/role.md)。

对接导航栏

以容器管理（开发代号 kpanda）为例，对接到导航栏。

对接后的预期效果如图：



对接效果

对接方法

参照以下步骤对接 GProduct：

1. 通过 GProductNavigator CR 将容器管理的各功能项注册到导航栏菜单。

```yaml
apiVersion: ghippo.io/v1alpha1
kind: GProductNavigator
metadata:
  name: kpanda
spec:
  gproduct: kpanda
  name: 容器管理
  localizedName:
```

```
    zh-CN: 容器管理
    en-US: Container Management
  url: /kpanda
  category: 容器   # (1)
  iconUrl: /kpanda/nav-icon.png
  order: 10 # (2)
  menus:
  - name: 备份管理
    localizedName:
      zh-CN: 备份管理
      en-US: Backup Management
    iconUrl: /kpanda/bkup-icon.png
    url: /kpanda/backup
```

1. 当前只支持概览、工作台、容器、微服务、数据服务、管理，六选一
2. 数字越大排在越上面

全局管理的导航栏 category 配置在 ConfigMap，暂时不能以注册方式增加，需要联系全局管理团队来添加。

2. Kpanda 前端作为微前端接入到 DCE 5.0 父应用 Anakin 中

前端使用 qiankun 来接入子应用 UI， 可以参考快速上手。

在注册 GProductNavigator CR 后，接口会生成对应的注册信息，供前端父应用注册使用。 例如 Kpanda 就会生成以下注册信息：

```
{
  "id": "kpanda",
  "title": "容器管理",
  "url": "/kpanda",
  "uiAssetsUrl": "/ui/kpanda/", // 结尾的/是必须的
  "needImportLicense": false
},
```

以上注册信息与 qiankun 子应用信息字段的对应关系是：

```
{
    name: id,
    entry: uiAssetsUrl,
    container: '#container',
    activeRule: url,
    loader,
    props: globalProps,
}
```

container 和 loader 由前端父应用提供，子应用无需关心。 props 会提供一个包含用户基本信息、子产品注册信息等的 pinia store。

qiankun 启动时会使用如下参数：

```
start({
  sandbox: {
    experimentalStyleIsolation: true,
  },
  // 去除子应用中的 favicon 防止在 Firefox 中覆盖父应用的 favicon
  getTemplate: (template) => template.replaceAll(/<link\s* rel="[\w\s]*icon[\w\s]*"\s*( href=".*?")?\s*\/?>/g, ''),
});
```

请参阅前端团队出具的 GProduct 对接 demo tar 包。

GProduct 如何对接全局管理

GProduct 是 DCE 5.0 中除全局管理外的所有其他模块的统称，这些模块需要与全局管理对接后才能加入到 DCE 5.0 中。

对接什么

- 对接导航栏

  入口统一放在左侧导航栏。

- 接入路由和 AuthN

  统一 IP 或域名，将路由入口统一走全局管理的 Istio Gateway。

- 统一登录 / 统一 AuthN 认证

  登录统一使用全局管理（Keycloak）登录页，API authn token 验证使用 Istio Gateway。 GProduct 对接全局管理后不需要关注如何实现登录和认证。

DCE 5.0 对接客户平台视频演示

接入路由和登录认证

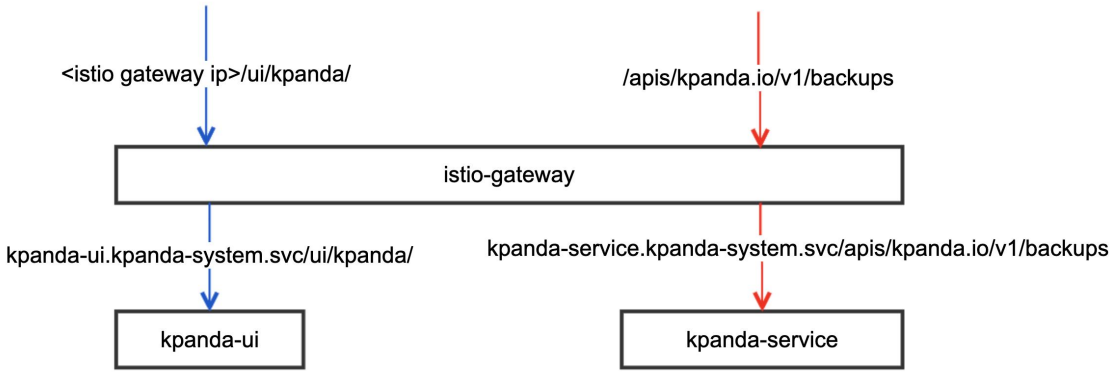
接入后统一登录和密码验证，效果如下图：



接入效果

各个 GProduct 模块的 API bear token 验证都走 Istio Gateway。

接入后的路由映射图如下：



接入效果

接入方法

以 kpanda 为例注册 GProductProxy CR。

```
# GProductProxy CR 示例，包含路由和登录认证

# spec.proxies: 后写的路由不能是先写的路由子集，反之可以
# spec.proxies.match.uri.prefix: 如果是后端 api，建议在 prefix 末尾添加 "/" 表述这段 path 结束（特殊需求可以不用加）
# spec.proxies.match.uri: 支持 prefix 和 exact 模式；Prefix 和 Exact 只能 2 选 1；Prefix 优先级大于 Exact

apiVersion: ghippo.io/v1alpha1
kind: GProductProxy
metadata:
  name: kpanda   # (1)
spec:
  gproduct: kpanda   # (2)
  proxies:
  - labels:
      kind: UIEntry
    match:
      uri:
```

```
      prefix: /kpanda  # (3)
  rewrite:
    uri: /index.html
  destination:
    host: ghippo-anakin.ghippo-system.svc.cluster.local
    port: 80
  authnCheck: false   # (4)
- labels:
    kind: UIAssets
  match:
    uri:
      prefix: /ui/kpanda/  # (5)
  destination:
    host: kpanda-ui.kpanda-system.svc.cluster.local
    port: 80
  authnCheck: false
- match:
    uri:
      prefix: /apis/kpanda.io/v1/a
  destination:
    host: kpanda-service.kpanda-system.svc.cluster.local
    port: 80
  authnCheck: false
- match:
    uri:
      prefix: /apis/kpanda.io/v1 # (6)
  destination:
    host: kpanda-service.kpanda-system.svc.cluster.local
    port: 80
  authnCheck: true
```

1. cluster 级别 CRD

2. 需要用小写指定 GProduct 名字
3. 还可支持 exact
4. 是否需要 istio-gateway 给该条路由 API 作 AuthN Token 认证, false 为跳过认证
5. UIAssets 建议末尾添加 / 表示结束（不然前端可能会出现问题）
6. 后写的路由不能是先写的路由的子集，反之可以

DaoCloud Enterprise 5.0

DaoCloud Enterprise 5.0 有两个版本：社区版和商业版。

目前下载中心仅提供离线安装包。有关在线安装方式，请参阅在线安装。

下载社区版

此版本面向社区，欢迎免费体验。

离线包	版本	下载	更新日期
offline-community-v0.9.0-arm64.tar	v0.9.0	:arrow_right: 下载页	2023-07-07
offline-community-v0.9.0-amd64.tar	v0.9.0	:arrow_right: 下载页	2023-07-07

更多历史版本

下载商业版

此版本适合搭建私有云、混合云、多云、微服务等场景，可以定制丰富多样的云原生模块。

离线包	版本	下载	更新日期
offline-v0.9.0-arm64.tar	v0.9.0	:arrow_right: 下载页	2023-07-07

离线包	版本	下载	更新日期
offline-v0.9.0-amd64.tar	v0.9.0	:arrow_right: 下载页	2023-07-07

更多历史版本

下载 Addon 离线包

DCE 5.0 针对不同的应用场景提供了完善的 Addon，您可以按需下载使用。

离线包	版本	下载	更新日期
addon-offline-full-package-v0.9.0-arm64.tar.gz	v0.9.0	:arrow_right: 下载页	2023-07-07
addon-offline-full-package-v0.9.0-amd64.tar.gz	v0.9.0	:arrow_right: 下载页	2023-07-07

更多历史版本

下载子模块

DCE 5.0 包含的众多子模块可以单独下载，按需对每个模块进行升级。

模块	文件大小	下载	更新日期
全局管理	440 MB	:arrow_right: 下载页	2023-08-28
容器管	2.41	:arrow_right: 下载页	2023-

模块	文件大小	下载	更新日期
理	GB		07–06
集群巡检	205.11 MB	:arrow_right: 下载页	2023–08–08
应用备份	59.29 MB	:arrow_right: 下载页	2023–08–08
安全管理	163 MB	:arrow_right: 下载页	2023–08–25
可观测性	2.64 GB	:arrow_right: 下载页	2023–08–04
服务网格	2.41 GB	:arrow_right: 下载页	2023–08–31
网络模块	55.47 MB	:arrow_right: 下载页	2023–04–26
镜像仓库	293.24 MB	:arrow_right: 下载页	2023–08–22
运营管理	56.90 MB	:arrow_right: 下载页	2023–08–02

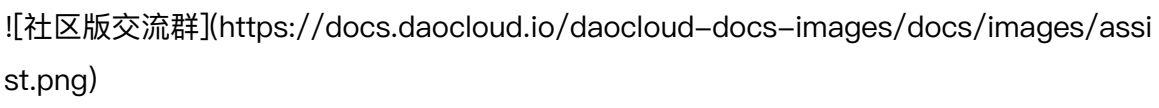
!!! note

– 升级子模块之后，可能需要重新灌入 License 并激活。

– DaoCloud Enterprise 5.0 还处于发布初期，将陆续增加各个子模块的离线包。

– 若有任何疑问，请[提出反馈](https://github.com/DaoCloud/DaoCloud-docs/issues)。

– 欢迎扫描二维码，与开发者畅快交流：



安装 DCE 5.0 申请社区免费体验

2023-05-09

DCE 5.0 社区版 v0.7.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.7.0-amd64.tar	v0.7.0	AMD64	5.96GB	:arrow_down: 下载	2023-05-09
offline-community-v0.7.0-arm64.tar	v0.7.0	ARM64	5.60GB	:arrow_down: 下载	2023-05-09

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "e80596e138a932d0d28101440ae22fbb319f048ad92043507e0202696ebc4a3
9717e65cbc12b6a6c02f5c9eb8c9fe7d027f381f037b6ae5dc1c21af00106e2b7  offlin
e-community-v0.7.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.7.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "3f68086d55ba413473eb3abae5756779ef22bc264950d0a3ef21714977463e3
ac7fa01a574a0488f3a547fc6a2c9d0b979e1e3a02c9632b222f879c5e0a32b78  offl
ine-community-v0.7.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.7.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.7.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.7.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.16.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.17.0
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.16.1

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-04-12

DCE 5.0 社区版 v0.6.1

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.6.1-amd64.tar	v0.6.1	AMD64	5.89GB	:arrow_down: 下载	2023-04-12
offline-community-v0.6.1-arm64.tar	v0.6.1	ARM64	5.52GB	:arrow_down: 下载	2023-04-12

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "f452f0b5757220b7630334916209275207c7f4672da80bb2de721d186f57bfc5744155a514a88f0271ec9a02f90f831baa4bcb32fd3b169d0255773916f10c32  offline-community-v0.6.1-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.6.1-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "2036e4b4a0d0fa9016b6bc0ceb28b2f6dd7ec9f856f2ef976dacdedd8d9e0500dbf47c7f070d70954fc9a3fd6adf060e550cf52a2343443531c85493e308de9f  offline-community-v0.6.1-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.6.1-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.6.1-amd64.tar
```

===“ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.6.1-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.15.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.16.1
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.15.4

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-04-07

DCE 5.0 社区版 v0.6.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.6.0-amd64.tar	v0.6.0	AMD64	5.89GB	:arrow_down: 下载	2023-04-07
offline-community-v0.6.0-arm64.tar	v0.6.0	ARM64	5.52GB	:arrow_down: 下载	2023-04-07

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "2419aaef4d003f51d35aed7f35fc4b311ac728330ad91e52aa4adcbeb5c60b310
6ffa8f94f7669d32e868e80d87ae3b1f2eef55c3d69211199f9cdfb677cc809  offline-
community-v0.6.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.6.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "494fb0c10f4ad693519f3153ef97a3072fecd348eb56ec28582eab59ef78f3a98
c14479abdb6e2064c204924f8bc60ee0b717644b96bee7f2f132b7f53ade86c  offlin
e-community-v0.6.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.6.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.6.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.6.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.15.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.16.0
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.15.4

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-01-05

DCE 5.0 社区版 v0.4.0-rc2

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.4.0-rc2-amd64.tar	v0.4.0-rc2	AMD64	5.73GB	:arrow_down: 下载	2023-01-05
offline-community-v0.4.0-rc2-arm64.tar	v0.4.0-rc2	ARM64	5.16GB	:arrow_down: 下载	2023-01-05

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "a9501539eb5e0241e4c3035e0870637d770b12e18971cb5dae1288682db4aa2
99fe42c964f556fa80d27e2ea6723b65a741f11f7a5f3696915c0a4f97015480e  offlin
e-community-v0.4.0-rc2-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.4.0-rc2-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "7f79739afd14804172d876d8421b6768ce2d040f225b9d251bdd21118fa58ec9d
82204a7a3f660d3673afc77b6be838d60f710a357b047bdbf8b65addd0e5977  offlin
e-community-v0.4.0-rc2-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.4.0-rc2-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.4.0-rc2-amd64.tar
```

===“ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.4.0-rc2-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.11
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-01-03

DCE 5.0 社区版 v0.3.30

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.3.30-amd64.tar	v0.3.30	AMD64	5.73GB	:arrow_down: 下载	2023-01-03
offline-community-v0.3.30-arm64.tar	v0.3.30	ARM64	5.16GB	:arrow_down: 下载	2023-01-03

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "469c98d6a60c7055f3e4159ffcdb6f65bb44cade4345400ad1b4067f8c3c89ef057983accaf413f76dc71b9a5592e0ef97600fa731bd715acacbdab1c653601b  offline-community-v0.3.30-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.3.30-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "9d965d49d3b09231fadae7fe713da7284b408e36f6d24d2863678dc4edf239abedc68a47e5d020bf02688ad197803a908db379e481340e13c86735fa29fd8d14  offline-community-v0.3.30-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.3.30-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.3.30-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.3.30-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.13
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.14
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.13

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2022-11-23

DaoCloud Enterprise 5.0 社区版历史版本

社区版

文件名称	版本	架构	文件大小	下载	更新日期
offline-community-v0.9.0-arm64.tar	v0.9.0	ARM 64	5.77GB	:arrow_right: 下载页	2023-07-07
offline-community-v0.9.0-amd64.tar	v0.9.0	AMD 64	6.14GB	:arrow_right: 下载页	2023-07-07
offline-community-v0.8.0-arm64.tar	v0.8.0	ARM 64	5.64GB	:arrow_right: 下载页	2023-06-06
offline-community-v0.8.0-amd64.tar	v0.8.0	AMD 64	6.01GB	:arrow_right: 下载页	2023-06-06
offline-community-v0.7.0-arm64.tar	v0.7.0	ARM 64	5.60GB	:arrow_right: 下载页	2023-05-09
offline-	v0.7.0	AMD 64	5.96GB	:arro	2023-05-09

文件名称	版本	架构	文件大小	下载	更新日期
community-v0.7.0-amd64.tar				w_right: 下载页	
offline-community-v0.6.1-arm64.tar	v0.6.1	ARM 64	5.52GB	:arrow_right: 下载页	2023-04-12
offline-community-v0.6.1-amd64.tar	v0.6.1	AMD 64	5.89GB	:arrow_right: 下载页	2023-04-12
offline-community-v0.6.0-arm64.tar	v0.6.0	ARM 64	5.52GB	:arrow_right: 下载页	2023-04-07
offline-community-v0.6.0-amd64.tar	v0.6.0	AMD 64	5.89GB	:arrow_right: 下载页	2023-04-07
offline-community-v0.5.0-arm64.tar	v0.5.0	ARM 64	5.27GB	:arrow_right: 下载页	2023-03-06
offline-community-v0.5.0-amd64.tar	v0.5.0	AMD 64	5.62GB	:arrow_right: 下载页	2023-03-06

2022-11-23

DCE 5.0 社区版 v0.3.27

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

版本名称	安装包
offline-community-v0.3.27.tar	:arrow_down: 下载

校验

进入离线安装包下载目录。执行以下命令校验安装包：

```
echo "7a2f07fd9d09c827347d2994c0b6d4852f55e1f6de149e7b95e29625c4081eba3d415d3dbb261d226d8179c3251ac8e67c2de898d3eb6a58ff218f79fd31d42e  offline-community-v0.3.27.tar" | sha512sum -c
```

校验成功会打印：

```
offline-community-v0.3.27.tar: OK
```

安装

成功校验离线包之后，解压缩 tar 包：

```
tar -zxvf offline-community-v0.3.28.tar
```

然后参阅社区版安装流程进行安装。

成功安装之后请申请免费社区体验。

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	发布说明
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11.1

更多

- 在线文档
- 报告 bug

DCE 5.0 社区版 v0.3.29

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-centos7-community-v0.3.29-amd64.tar	v0.3.29	AMD64	9.2 GB	:arrow_down: 下载	2022-12-16
offline-kylin-v10sp2-community-v0.3.29-arm64.tar	v0.3.29	ARM64	6.9 GB	:arrow_down: 下载	2022-12-16

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
echo "988bf4397f555804fb004a83e01169fd4cfb995d0659170197cab4d07c26aefb6c916ce42c0655d207a2ae7bddd5c28c6c66fc7645c67a174a8919e7e040cbd8  offli
```

ne-centos7-community-v0.3.29-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-centos7-community-v0.3.29-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "86dcb1f8b155d37a19a1b6c81a74a3758f443a79a8ffd95b9f5a634d992932714
d8bce9805ab52d9fffbfdcbc82873e7c7132a7d3e9a45d5fe00f46de16ab717  offline
-kylin-v10sp2-community-v0.3.29-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-kylin-v10sp2-community-v0.3.29-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
```
```

tar –zxvf offline–centos7–community–v0.3.29–amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar –zxvf offline–kylin–v10sp2–community–v0.3.29–arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.12
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.13
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.12
```

更多

- 在线文档
- 报告 bug

2023-07-07

DCE 5.0 社区版 v0.9.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.9.0-amd64.tar	v0.9.0	AMD64	6.14 GB	:arrow_down: 下载	2023-07-07
offline-community-v0.9.0-arm64.tar	v0.9.0	ARM64	5.77 GB	:arrow_down: 下载	2023-07-07

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "b77127bbde1003501d32d58ca408c6bc6637cbe85d20696ce7403b1ded73406
38cc7c7a447fe52b055ff7068e3d85399f6a68a7b9d47cd0e7bbfc4c77be4dab2  offl
ine-community-v0.9.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.9.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "001a369379dc3299b6d0e00e46c83b9567323c2d52620d85815917a552fbc1
3c2d7076a2ad71eaff7dfbfe7ed82f68e5d30c0e53f47fa5055ef07588b4355bc3  offl
ine-community-v0.9.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.9.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.9.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.9.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.18.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.19.0
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.18.2

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-03-06

DCE 5.0 社区版 v0.5.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.5.0-amd64.tar	v0.5.0	AMD64	5.62GB	:arrow_down: 下载	2023-03-06
offline-community-v0.5.0-arm64.tar	v0.5.0	ARM64	5.27GB	:arrow_down: 下载	2023-03-06

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "1c8b629dc6f842a6406a198ee5bffd2f751464745b9f4b4ef8899492691a58bdf5e0b204df6cb013285f0326a709ea10fff1c7d71ea88fe7f17b3f820c850361  offline-community-v0.5.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.5.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "4f060c189b29b3d08bce8287db48232809220162574a59fbd8050ffca414f0d0ba3adf3d68cb2f66fe868aab4059fa3c205bcb7058952d6df51acc23cac32c40  offline-community-v0.5.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.5.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.5.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.5.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.14.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.15.1
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.14.6

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-01-12

DCE 5.0 社区版 v0.4.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.4.0-amd64.tar	v0.4.0	AMD64	5.73GB	:arrow_down: 下载	2023-01-12
offline-community-v0.4.0-arm64.tar	v0.4.0	ARM64	5.16GB	:arrow_down: 下载	2023-01-12

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "41f7705d2be5487a721b936ba16b89ad2f35011b0d1a98d71d29ab51cf36ef2bf34283be384e76b0438c172ff9e236c44c33843e9855e9af253b1db4b84144fe offline-community-v0.4.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.4.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "1d0476965fd73002c379639353b0bd0e09cefe99156ef448c42a4c10aff60a9836981c86e914ba3f614617a455b67a8c3ce4d82d53b3e47a22222c34020d0a00 offline-community-v0.4.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.4.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.4.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.4.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.13.2
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.14.0
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.13.2

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2023-06-06

DCE 5.0 社区版 v0.8.0

本页可下载 DCE 5.0 社区版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-community-v0.8.0-amd64.tar	v0.8.0	AMD64	6.01GB	:arrow_down: 下载	2023-06-06
offline-community-v0.8.0-arm64.tar	v0.8.0	ARM64	5.64GB	:arrow_down: 下载	2023-06-06

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "886f1e1622115e3d2bfa6e41e26ba3c02419177ee7cae422000b28f6e9cd9b2b370a8a737be90328ee1b048c02811b4b31443638960b3cd24acf9ce0b9848320  offline-community-v0.8.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.8.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "73ccc2305196caa7c8152369016b5bc2fa38f874028f94961482ec5610158c7e7e8b4c3f7a335e473a28953e5ffeff27bb6ee7d132b3b1ae8e49ddd711993c21  offline-community-v0.8.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-community-v0.8.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.8.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-community-v0.8.0-arm64.tar
```

- 安装请参阅社区版安装流程
- 成功安装之后请申请免费社区体验

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.17.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.18.1
可观	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.17.2

模块测性

介绍

最新动态

更多

- 在线文档
- 报告 bug

2022-11-17

DCE 5.0 社区版 v0.3.28

本页可下载 DCE 5.0 社区版的离线安装包。

下载

版本名称	文件大小	安装包	更新日期
offline-community-v0.3.28.tar	5.8 GB	:arrow_down: 下载	2022-11-18

校验

进入离线安装包下载目录。执行以下命令校验安装包：

```
echo "4e1d839ee51f6ff837e2363576cd7ccb794bf81de5fac3c840d14781abc3e077f9014466a3f21d29b83af12643e59e4fa310ecd08831266d2b361ba9e9b81933  offline-community-v0.3.28.tar" | sha512sum -c
```

校验成功会打印：

```
offline-community-v0.3.28.tar: OK
```

安装

成功校验离线包之后，解压缩 tar 包：

```
tar -zxvf offline-community-v0.3.28.tar
```

然后参阅社区版安装流程进行安装。 如果是初次安装或遗忘了许可密钥，请申请免费社区体验。

模块

DCE 5.0 社区版默认包含以下模块：

模块	介绍	模块版本
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.11
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11

更多

- 在线文档
- 报告 bug

2023-05-09

DCE 5.0 商业版 v0.7.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.7.0-amd64.tar	v0.7.0	AMD64	17.29GB	:arrow_down: 下载	2023-05-09
offline-v0.7.0-arm64.tar	v0.7.0	ARM64	15.89GB	:arrow_down: 下载	2023-05-09

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "d6d88a2274f75c952e079c8f0702d3bf6c2fae7d75771bd02e8539f2afd3933b
df153e5cb41237ce5285b04fd6fb6075389ea80f16713bdfbe620f86509eee42  offlin
e-v0.7.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.7.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "b159b5c7b8c224eb76a532ea0a2f2c55143cb240a1b7aa2ea97ab6acf244ff8c
c0e74a46150c4ffa8d79409950573067ff5f9f8841fecb2af36ed20c4ffc048d  offline
-v0.7.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.7.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.7.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.7.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.16.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.17.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.16.1
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.16.1

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.8.1
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.21.2
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.15.1
中间件 Elasticsearch	目前首选的全文搜索引擎	0.7.2
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.5.2
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.5.2
中间件 MySQL	应用最广泛的开源关系数据库	0.8.2
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.10.2
中间件 Redis	一款内存数据库缓存服务	0.7.2
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.7.2
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.6.0

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.9.3

更多

- 在线文档
- 报告 bug

2023-04-12

DCE 5.0 商业版 v0.6.1

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.6.1-amd64.tar	v0.6.1	AMD64	16.37GB	:arrow_down: 下载	2023-04-12
offline-v0.6.1-arm64.tar	v0.6.1	ARM64	15.15GB	:arrow_down: 下载	2023-04-12

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "00a785ba2f3b2abe7f634410ae08b798771d2d9de27b7d39e9ed6a9d9eff0e04ea544478b04f75610190b7f46c33179711ac5be6ecd219ea4f407c38850d350c  offline-v0.6.1-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.6.1-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "1ebbd5fb7841d9e36e972ec6316cf43f62fe770401026fa07e9da58908f71b9ef9ae30c4b345efc60fb1a8ee3a6a2ba1f1e50b9858da223b56dca17a32548733  offline-v0.6.1-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.6.1-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.6.1-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.6.1-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.15.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.16.1
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.15.4
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.15.1

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.7.4
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.19.4
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.14.3
中间件 Elasticsearch	目前首选的全文搜索引擎	0.6.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.4.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.4.1
中间件 MySQL	应用最广泛的开源关系数据库	0.7.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.9.1
中间件 Redis	一款内存数据库缓存服务	0.6.2
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.6.2
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.5.0

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.9.2

更多

- 在线文档
- 报告 bug

2023-04-07

DCE 5.0 商业版 v0.6.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.6.0-amd64.tar	v0.6.0	AMD64	16.36GB	:arrow_down: 下载	2023-04-07
offline-v0.6.0-arm64.tar	v0.6.0	ARM64	15.12GB	:arrow_down: 下载	2023-04-07

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "4e810a67268f8e0125a486018c3da8e8165d16cd0206d6d6e0560773caf8983cfe2b0827692e3ac6efbd26345c0bbfed58139ff01affd96d525b59f797cec5d9  offline-v0.6.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.6.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "4c778c6aad168e19bdffec234c5096795b6b7cfa9c5d17c9d9e64cb7ddc526263d6f0d863c631e2e4cc5d205866f29a69d9a38c92b8c751af0ed1db25304ba7d  offline-v0.6.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.6.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.6.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.6.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.15.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.16.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.15.4
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.15.1

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.7.4
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.19.1
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.14.3
中间件 Elasticsearch	目前首选的全文搜索引擎	0.6.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.4.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.4.1
中间件 MySQL	应用最广泛的开源关系数据库	0.7.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.9.1
中间件 Redis	一款内存数据库缓存服务	0.6.2
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.6.2
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.5.0

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.9.2

更多

- 在线文档
- 报告 bug

2023-01-05

DCE 5.0 商业版 v0.4.0-rc2

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-centos7-v0.4.0-rc2-amd64.tar	v0.4.0-rc2	AMD64	16.33GB	:arrow_down: 下载	2023-01-05
offline-kylin-v10sp2-v0.4.0-rc2-arm64.tar	v0.4.0-rc2	ARM64	14.96GB	:arrow_down: 下载	2023-01-05

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "9fdc24ea0eda2994cb1ab253d3c6f079b76b138618fb5601176be5a4d660893c3cc64dfb151e32d57a1f843c6b598bed270a20654e6cd8d815c7901124b04431  offline-centos7-v0.4.0-rc2-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-centos7-v0.4.0-rc2-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "ec6eca321a80f26f70c11739eb15dc67914579af610e2fb9cb49fcf8d9691de2ceddac7cfd8bd40ce3f3927cfea3166d15366f06909fcbbf15a330bc718fd358  offline-kylin-v10sp2-v0.4.0-rc2-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-kylin-v10sp2-v0.4.0-rc2-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-centos7-v0.4.0-rc2-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-kylin-v10sp2-v0.4.0-rc2-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.11
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	v0.9

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.3
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	v0.11
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.10
中间件 Elasticsearch	目前首选的全文搜索引擎	v0.3
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	v0.1
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	v0.1
中间件 MySQL	应用最广泛的开源关系数据库	v0.4
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	v0.6
中间件 Redis	一款内存数据库缓存服务	v0.2
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	发布说明
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	发布说明

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	发布说明

更多

- 在线文档
- 报告 bug

2023-01-03

DCE 5.0 商业版 v0.3.30

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-centos7-v0.3.30-amd64.tar	v0.3.0	AMD64	16.33GB	:arrow_down: 下载	2023-01-03
offline-kylin-v10sp2-v0.3.30-arm64.tar	v0.3.0	ARM64	14.96GB	:arrow_down: 下载	2023-01-03

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "08ccfce8e3f551e82bdb89f26d0e9bb9b0f40e02cd5bcd0db8662c70d22932f
24b2958ab3ea71e0ec497d8ad75a1cc134cdd24eabbfe9003c3c120c83d4d0417  offl
ine-centos7-v0.3.30-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-centos7-v0.3.30-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "a9705268767754b763d8cc3df32ac522075ad2e08989a9bbad08b3a9be75366
be6208f6e8d2865ee05c07291545373ef045bfc6fb31179df1ad2140b2b8741998  offl
ine-kylin-v10sp2-v0.3.30-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-kylin-v10sp2-v0.3.30-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-centos7-v0.3.30-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-kylin-v10sp2-v0.3.30-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.13
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.14
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.13
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	v0.12

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.5
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	v0.16
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.12
中间件 Elasticsearch	目前首选的全文搜索引擎	v0.4
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	v0.2
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	v0.2
中间件 MySQL	应用最广泛的开源关系数据库	v0.5
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	v0.7
中间件 Redis	一款内存数据库缓存服务	v0.4
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	发布说明
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	发布说明

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	发布说明

更多

- 在线文档
- 报告 bug

2022-11-23

DaoCloud Enterprise 5.0 商业版历史版本

商业版

文件名称	版本	架构	文件大小	下载	更新日期
offline-v0.9.0-arm64.tar	v0.9.0	ARM 64	17.37GB	:arrow_right: 下载页	2023-07-07
offline-v0.9.0-amd64.tar	v0.9.0	AMD 64	19.01GB	:arrow_right: 下载页	2023-07-07
offline-v0.8.0-arm64.tar	v0.8.0	ARM 64	16.99GB	:arrow_right: 下载页	2023-06-06
offline-v0.8.0-amd64.tar	v0.8.0	AMD 64	18.43GB	:arrow_right: 下载页	2023-06-06
offline-v0.7.0-arm64.tar	v0.7.0	ARM 64	15.89GB	:arrow_right: 下载页	2023-05-09
offline-v0.7.0-	v0.7.0	AMD 64	17.29GB	:arro	2023-05-09

文件名称	版本	架构	文件大小	下载	更新日期
amd64.tar				w_rig ht: 下载页	
offline-v0.6.1-arm64.tar	v0.6.1	ARM 64	15.15GB	:arro w_rig ht: 下载页	2023-04-12
offline-v0.6.1-amd64.tar	v0.6.1	AMD 64	16.37GB	:arro w_rig ht: 下载页	2023-04-12
offline-v0.6.0-arm64.tar	v0.6.0	ARM 64	15.12GB	:arro w_rig ht: 下载页	2023-04-07
offline-v0.6.0-amd64.tar	v0.6.0	AMD 64	16.36GB	:arro w_rig ht: 下载页	2023-04-07
offline-v0.5.0-arm64.tar	v0.5.0	ARM 64	14.56GB	:arro w_rig ht: 下载页	2023-03-06
offline-v0.5.0-amd64.tar	v0.5.0	AMD 64	15.86GB	:arro w_rig ht: 下载页	2023-03-06

2022-11-23

DCE 5.0 商业版 v0.3.27

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

版本名称	安装包
offline-v0.3.27.tar	:arrow_down: 下载

校验

进入离线安装包下载目录。执行以下命令校验安装包：

```
echo "f637ec103af6e77d1af85bf0708bef71aee123ce4ac71c4a0adef539492cdbb1661a479d3e999cd51aa7cb47d49e001565908b237ef7999140e3435f6219bb08 offline-v0.3.27.tar" | sha512sum -c
```

校验成功会打印：

```
offline-v0.3.27.tar: OK
```

安装

成功校验离线包之后，请参阅商业版安装流程进行安装。

成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	发布说明
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11.1
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水线作业	发布说明
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.3.0
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	发布说明

模块	介绍	最新动态
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.10.0
中间件	包含 RabbitMQ、Kafka、Elasticsearch、MySQL、Redis、MinIO 等精选中间件	发布说明
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	发布说明
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	发布说明
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	发布说明

更多

- 在线文档
- 报告 bug

DCE 5.0 商业版 v0.3.29

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-centos7-v0.3.29-amd64.tar	v0.3.29	AMD64	22.9 GB	:arrow_down: 下载	2022-12-16
offline-kylin-v10sp2-v0.3.29-arm64.tar	v0.3.29	ARM64	19.9 GB	:arrow_down: 下载	2022-12-16

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
echo "ef7fd779d3b5bc50bccf80e29934002c60f53143319c80dd9aad85fc1404ac7d309997e0d9c829612c1b400cd4d4861fb1b6f91efee8c236ada930cbb44ca1c1 offline-centos7-v0.3.29-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-centos7-v0.3.29-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "1cc8b41bfaad23cb1b14170b68ee581d2eb384fe6d803ce34a1a09f1d41e76407
68cca8a7f8a3f6a881ecfddaaa73756247676b6e0bc72b7ca651cc855ce2ff4 offline
-kylin-v10sp2-v0.3.29-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-kylin-v10sp2-v0.3.29-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-centos7-v0.3.29-amd64.tar
```

===“ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-kylin-v10sp2-v0.3.29-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.12
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.13
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.12
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水线作业	v0.10
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.4
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	v0.12
服务网	基于 Istio 开源技术构建的面向云原	v0.11

模块	介绍	最新动态
格	生应用的下一代服务网格	
中间件 Elastics earch	目前首选的全文搜索引擎	v0.3
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	v0.1
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	v0.1
中间件 MySQL	应用最广泛的开源关系数据库	v0.4
中间件 RabbitM Q	实现了高级消息队列协议（AMQP）的开源消息代理软件	v0.6
中间件 Redis	一款内存数据库缓存服务	v0.3
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	发布说明
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	发布说明
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	发布说明

更多

- 在线文档
- 报告 bug

2023-07-07

DCE 5.0 商业版 v0.9.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.9.0-amd64.tar	v0.9.0	AMD64	19.01 GB	:arrow_down: 下载	2023-07-07
offline-v0.9.0-arm64.tar	v0.9.0	ARM64	17.37 GB	:arrow_down: 下载	2023-07-07

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "520ef719605042cd2b9289de795d609cde7f0aff9f7189d43e31bbe016b33deb
c715f8e0de24c8f3c3685d54f7d6b2595651bcfa9695c9b98210d161cfddc241  offlin
e-v0.9.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.9.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "ae541aa5c2150af45b6dc3c3e432205683211d1b5d8ec816b37344c2234f3c05
fbe2be7526b4b5832c5db0439c7d501ce2f1c1492aa5cfe045bbdd321d662e22  offli
ne-v0.9.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.9.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.9.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.9.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.18.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.19.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.18.2
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.18.1

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.10.3
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.24.2
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.17.0
中间件 Elasticsearch	目前首选的全文搜索引擎	0.9.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.7.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.7.0
中间件 MySQL	应用最广泛的开源关系数据库	0.10.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.12.0
中间件 Redis	一款内存数据库缓存服务	0.9.0
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.9.1
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.8.0

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.10.4

更多

- 在线文档
- 报告 bug

2023-03-06

DCE 5.0 商业版 v0.5.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.5.0-amd64.tar	v0.5.0	AMD64	15.86GB	:arrow_down: 下载	2023-03-06
offline-v0.5.0-arm64.tar	v0.5.0	ARM64	14.56GB	:arrow_down: 下载	2023-03-06

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "83f0bc6522eb525bc8d3c2478cf68235972d583171a1e471d45f338d1acde6f6
6a5ed68144bfd6a067b2462a1c27e17d95c13408b8cbd83fd93d1dbe152779a3  offlin
e-v0.5.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.5.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "ca857e62f67fba0c734a3e51658cf0900311db116dfb2dc82ed54cc8b8387ad0e
3d53e95a0df06e913cf62038858d585593990587bdf802790e2fa6050759ec2  offli
ne-v0.5.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.5.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.5.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.5.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.14.0
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.15.1
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.14.6
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.14.4

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.6.2
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.18.0
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.13.1
中间件 Elasticsearch	目前首选的全文搜索引擎	0.5.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.3.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.3.0
中间件 MySQL	应用最广泛的开源关系数据库	0.6.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.8.0
中间件 Redis	一款内存数据库缓存服务	0.5.0
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.5.2
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.4.4

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.8.0

更多

- 在线文档
- 报告 bug

2023-01-12

DCE 5.0 商业版 v0.4.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-centos7-v0.4.0-amd64.tar	v0.4.0	AMD64	16.33GB	:arrow_down: 下载	2023-01-12
offline-kylin-v10sp2-v0.4.0-arm64.tar	v0.4.0	ARM64	14.96GB	:arrow_down: 下载	2023-01-12

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "5b94cb38a07b2f629d14a66b9a986bce11b0e584803072dd54956b142ed3c4c6967de337e4f8a27a726e94c20ad697ebaa080433fa062e9029b2f1983fa8b80d  offline-centos7-v0.4.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-centos7-v0.4.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "eea9853a8dcdcb1ddf73aff771326757d21fe130eedfc1fafa8457b521d5fe014ca11adbde48ff3d49c7d5af530c1e9fbdd8e18a9a190b77f09a277b8acc8ee4  offline-kylin-v10sp2-v0.4.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-kylin-v10sp2-v0.4.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-centos7-v0.4.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-kylin-v10sp2-v0.4.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.13.2
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.14.0
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.13.2
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	v0.12.0

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.5.1
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	v0.16.1
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.12.1
中间件 Elasticsearch	目前首选的全文搜索引擎	v0.4.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	v0.2.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	v0.2.0
中间件 MySQL	应用最广泛的开源关系数据库	v0.5.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	v0.7.0
中间件 Redis	一款内存数据库缓存服务	v0.4.1
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	v0.4.0
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	v0.4.2

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	0.7.1

更多

- 在线文档
- 报告 bug

2023-06-06

DCE 5.0 商业版 v0.8.0

本页可下载 DCE 5.0 商业版的离线安装包和校验文件。

下载

文件名	版本	架构	文件大小	下载	更新日期
offline-v0.8.0-amd64.tar	v0.8.0	AMD64	18.43GB	:arrow_down: 下载	2023-06-06
offline-v0.8.0-arm64.tar	v0.8.0	ARM64	16.99GB	:arrow_down: 下载	2023-06-06

校验

进入离线安装包下载目录。

===“AMD64”

执行以下命令校验安装包：

```sh
```

```
echo "fbd3e646f5849406db1c221ab6956afd0e3dfd76e75394dfaeec06d2f2ec74934801cd7118c4bf2f51a3610dcb69fd7a010c613fcda3339abd20a1630029723e offline-v0.8.0-amd64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.8.0-amd64.tar: OK
```

===“ARM64”

执行以下命令校验安装包：

```sh
echo "f2854f0a90c2db16b9c206708b140069a953b68a575ce59b1dc656f84cb47c42647697067582e28e16175f4bfbcfcdb6c14d79c3d999c7646f1c58c40f1b35cc offline-v0.8.0-arm64.tar" | sha512sum -c
```

校验成功会打印：

```none
offline-v0.8.0-arm64.tar: OK
```

安装

成功校验离线包之后，

===“AMD64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.8.0-amd64.tar
```

=== “ARM64”

执行以下命令解压缩 tar 包：

```sh
tar -zxvf offline-v0.8.0-arm64.tar
```

- 安装请参阅商业版安装流程
- 成功安装之后请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	最新动态
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	0.17.1
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	0.18.1
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	0.17.2
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水	0.17.3

模块	介绍	最新动态
	线作业	
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	0.9.1
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	0.22.2
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.16.2
中间件 Elasticsearch	目前首选的全文搜索引擎	0.8.0
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	0.6.0
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	0.6.0
中间件 MySQL	应用最广泛的开源关系数据库	0.9.0
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	0.11.0
中间件 Redis	一款内存数据库缓存服务	0.8.0
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	0.8.1
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	0.7.0

模块	介绍	最新动态
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	v0.10.2

更多

- 在线文档
- 报告 bug

2022-11-23

DCE 5.0 商业版 v0.3.28

本页可下载 DCE 5.0 商业版的离线安装包。

下载

版本名称	文件大小	安装包	更新日期
offline-v0.3.28.tar	21 GB	:arrow_down: 下载	2022-11-18

校验

进入离线安装包下载目录。执行以下命令校验安装包：

```
echo "88d89872d04d95ee44073c70460c2eb3ae4785a150fbfce672a5882c6f7d77f0d8f58359c5c8695e80d7e5fce93431c0c5ec6b710c080f4840d8adbb25daeb55  offline-v0.3.28.tar" | sha512sum -c
```

校验成功会打印：

```
offline-v0.3.28.tar: OK
```

安装

成功校验离线包之后，解压缩 tar 包：

```
tar -zxvf offline-v0.3.28.tar
```

然后参阅安装 DCE 5.0 商业版进行安装。 如果是初次安装，请联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

模块

DCE 5.0 商业版包含以下模块，按需即插即用满足各类应用场景：

模块	介绍	模块版本
全局管理	负责用户访问控制、权限、企业空间、审计日志、个性化外观设置等	v0.11
容器管理	管理集群、节点、工作负载、Helm 应用、CRD、命名空间等 K8s 核心功能	v0.12
可观测性	提供丰富的仪表盘、场景监控、数据查询、告警等图文信息	v0.11
应用工作台	基于容器的 DevOps 应用平台，支持 Jenkins, Tekton, GitOps 等流水线作业	v0.9
多云编排	集中管理多云、混合云、跨云资源的应用编排，具备多云灾备、故障恢复等能力	v0.3
微服务引擎	提供注册发现、服务治理、配置管理、微服务网关等治理能力	v0.11
服务网格	基于 Istio 开源技术构建的面向云原生应用的下一代服务网格	v0.10
中间件 Elasticsearch	目前首选的全文搜索引擎	v0.3

模块	介绍	模块版本
中间件 Kafka	基于开源软件 Kafka 提供的分布式消息队列服务	v0.1
中间件 MinIO	一款非常热门的轻量、开源对象存储方案	v0.1
中间件 MySQL	应用最广泛的开源关系数据库	v0.4
中间件 RabbitMQ	实现了高级消息队列协议（AMQP）的开源消息代理软件	v0.6
中间件 Redis	一款内存数据库缓存服务	v0.2
镜像仓库	用于存储 K8s、DevOps 和容器应用开发的镜像	发布说明
网络	针对不同的 Linux 内核，支持多种 CNI 组合方案	发布说明
存储	提供统一数据存储服务，支持文件、对象、块、本地存储，轻松接入存储厂商方案	发布说明

更多

- 在线文档
- 报告 bug

下载 Addon 离线包

本页可下载针对 DCE 5.0 的 Addon 离线包。 有关离线 Addon 的使用，可以在 clusterConfig.yaml 中进行配置，请查阅 DCE 5.0 安装流程。

下载

文件名	版本	架构	文件大小	下载	更新日期
{{amd.fileName}}	{{version}}	AMD64	{{amd.size}}	:arrow_down: 下载	{{date}}
{{arm.fileName}}	{{version}}	ARM64	{{arm.size}}	:arrow_down: 下载	{{date}}

文件名	版本	架构	文件大小	下载	更新日期
{{amd.fileName}}	{{version}}	amd64	{{amd.size}}	{{amd.downloadUrl}}	{{date}}
{{arm.fileName}}	{{version}}	arm64	{{arm.size}}	{{arm.downloadUrl}}	{{date}}

校验

进入离线包的下载目录。

===“AMD64”

执行以下命令校验完整性：

```sh
echo "{{amd.checksum}}" | sha512sum -c
```

校验成功后打印结果类似于：

```none
{{amd.fileName}}: ok
```

===“ARM64”

执行以下命令校验完整性：

```sh
echo "{{arm.checksum}}" | sha512sum -c
```

校验成功后打印结果类似于：

```none
{{arm.fileName}}: ok
```

使用

DCE 5.0 支持 Addon 离线化，下载离线包后，打开 clusterConfig.yaml，修改 addonOfflinePackagePath 字段，指定 Addon 所在的路径。

Addon 离线包的组件

{{version}} 版本的 Addon 离线包内含以下组件：

===“AMD64”

仓库	组件	版本
addon	metrics-server	{{amd.version.metrics-server}}
	kubernetes-cronhpa-controller	{{amd.version.kubernetes-cronhpa-controller}}
	vpa	{{amd.version.vpa}}
	skoala-init	{{amd.version.skoala-init}}
	velero	{{amd.version.velero}}
	rabbitmq-cluster-operator	{{amd.version.rabbitmq-cluster-operator}}
	eck-operator	{{amd.version.eck-operator}}
	redis-cluster-operator	{{amd.version.redis-cluster-operator}}
	strimzi-kafka-operator	{{amd.version.strimzi-kafka-operato

r}} |
	minio-operator	{{amd.version.minio-operator}}
	jenkins	{{amd.version.jenkins}}
	insight-agent	{{amd.version.insight-agent}}
	coredns-metrics	{{amd.version.coredns-metrics}}
	harbor-operator	{{amd.version.harbor-operator}}
	spiderpool	{{amd.version.spiderpool}}
	contour	{{amd.version.contour}}
	multus-underlay	{{amd.version.multus-underlay}}
	f5networks	{{amd.version.f5networks}}
	metallb	{{amd.version.metallb}}
	nmstate	{{amd.version.nmstate}}
	neuvector	{{amd.version.neuvector}}
	falco	{{amd.version.falco}}
	falco-event-generator	{{amd.version.falco-event-generato
r}}
	falco-exporter	{{amd.version.falco-exporter}}
community	dao-2048	{{amd.version.dao-2048}}
	kube-node-tuning	{{amd.version.kube-node-tuning}}
	node-problem-detector	{{amd.version.node-problem-detec
tor}}
	kuberhealthy	{{amd.version.kuberhealthy}}
	kube-bench	{{amd.version.kube-bench}}
	vela-core	{{amd.version.vela-core}}
	wordpress	{{amd.version.wordpress}}
	docker-registry	{{amd.version.docker-registry}}
	argo-cd	{{amd.version.argo-cd}}
	kubeview	{{amd.version.kubeview}}
	prometheus-adapter	{{amd.version.prometheus-adapter}}
	prometheus-node-exporter	{{amd.version.prometheus-node-e

xporter}} |
| | kube-state-metrics | {{amd.version.kube-state-metrics}}
|

=== “ARM64”

仓库	组件	版本
addon	metrics-server	{{arm.version.metrics-server}}
	kubernetes-cronhpa-controller	{{arm.version.kubernetes-cronhpa-c
ontroller}}
	vpa	{{arm.version.vpa}}
	skoala-init	{{arm.version.skoala-init}}
	rabbitmq-cluster-operator	{{arm.version.rabbitmq-cluster-oper
ator}}
	eck-operator	{{arm.version.eck-operator}}
	redis-cluster-operator	{{arm.version.redis-cluster-operator}}
	strimzi-kafka-operator	{{arm.version.strimzi-kafka-operato
r}}
	minio-operator	{{arm.version.minio-operator}}
	jenkins	{{arm.version.jenkins}}
	insight-agent	{{arm.version.insight-agent}}
community	dao-2048	{{arm.version.dao-2048}}
	kube-node-tuning	{{arm.version.kube-node-tuning}}
	node-problem-detector	{{arm.version.node-problem-detect
or}}
	kuberhealthy	{{arm.version.kuberhealthy}}
	vela-core	{{arm.version.vela-core}}
	docker-registry	{{arm.version.docker-registry}}
	prometheus-adapter	{{arm.version.prometheus-adapter}}

| | prometheus-node-exporter | {{arm.version.prometheus-node-exporter}} |
| | kube-state-metrics | {{arm.version.kube-state-metrics}} |

2023-06-05

Addon 历史版本

文件名称	版本	架构	文件大小	下载	更新日期
addon-offline-full-package-v0.9.0-arm64.tar.gz	v0.9.0	ARM 64		:arrow_right: 下载页	2023-07-07
addon-offline-full-package-v0.9.0-amd64.tar.gz	v0.9.0	AMD 64		:arrow_right: 下载页	2023-07-07
addon-offline-full-package-v0.8.0-rc2-arm64.tar.gz	v0.8.0-rc2	ARM 64		:arrow_right: 下载页	2023-06-15
addon-offline-full-package-v0.8.0-rc2-amd64.tar.gz	v0.8.0-rc2	AMD 64		:arrow_right: 下载页	2023-06-15
addon-offline-full-package-v0.8.0-amd64.tar.gz	v0.8.0	AMD64	13.8 GB	:arrow_right: 下载页	2023-06-01
addon-offline-full-package-v0.8.0-	v0.8.0	ARM64	13.8 GB	:arrow_right: 下	2023-06-01

文件名称	版本	架构	文件大小	下载	更新日期
arm64.tar.gz				载页	
addon-offline-full-package-v0.6.0-amd64.tar.gz	v0.6.0	AMD64	9.9 GB	:arrow_right: 下载页	2023-04-14
addon-offline-full-package-v0.6.0-arm64.tar.gz	v0.6.0	ARM64	5.2 GB	:arrow_right: 下载页	2023-04-14

下载 Addon 离线包

本页可下载针对 DCE 5.0 的 Addon 离线包。 有关离线 Addon 的使用，可以在 clusterConfig.yaml 中进行配置，请查阅 DCE 5.0 安装流程。

下载

文件名	版本	架构	文件大小	下载	更新日期
addon-offline-full-package-v0.6.0-amd64.tar.gz	v0.6.0	AMD64	9.4 GB	:arrow_down: 下载	2023-10-04
addon-offline-full-package-v0.6.0-arm64.tar.gz	v0.6.0	ARM64	5.2 GB	:arrow_down: 下载	2023-10-04

文件名	版本	架构	文件大小	下载	更新日期

校验

进入离线包的下载目录。

===“AMD64”

执行以下命令校验完整性：

```sh
echo "54dc896be65785a8b7403524bd4bf4f88bf805aeec293cc17f229b5ab48ba925bd2895b8a51be2ce6a1cebb999918898c8d0ce74bc86f1d5567096485d8a708e addon-offline-full-package-v0.6.0-amd64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.6.0-amd64.tar: ok
```

===“ARM64”

执行以下命令校验完整性：

```sh
echo "a006eab626ee767544cfd8757cdff6335ea1089e90f7aa3333fa2a33c0c4f87dca0afb3c7275bce940fd76984257da0e577c58d951647527933b937dc7fffa46 addo
```

n-offline-full-package-v0.6.0-arm64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.6.0-arm64.tar: ok
```

使用

DCE 5.0 支持 Addon 离线化，下载离线包后，打开 clusterConfig.yaml，修改 addonOfflinePackagePath 字段，指定 Addon 所在的路径。

Addon 离线包的组件

v0.6.0 版本的 Addon 离线包内含以下组件：

=== “AMD64”

仓库	组件	版本
addon	metrics-server	v3.8.3
	kubernetes-cronhpa-controller	v1.3.0
	vpa	v1.5.0
	skoala-init	v0.18.0
	velero	3.0.0
	rabbitmq-cluster-operator	v0.8.0
	eck-operator	v0.5.0
	redis-cluster-operator	v0.5.0
	strimzi-kafka-operator	v0.3.0
	minio-operator	v0.3.0
	jenkins	v0.1.4
	insight-agent	v0.14.5
	coredns-metrics	v0.1.0
```

	harbor-operator	1.4.0
	spiderpool	0.3.6
	contour	10.2.2
	multus-underlay	0.2.1
	f5networks	0.2.2
	metallb	0.13.9
	nmstate	0.74.0
	neuvector	2.4.2
	falco	2.0.17
	falco-event-generator	0.1.1
	falco-exporter	0.8.2
community	dao-2048	v1.3.1
	kube-node-tuning	v0.3.0
	node-problem-detector	v1.3.1
	kuberhealthy	v90
	kube-bench	0.1.5
	vela-core	v1.6.7
	wordpress	15.2.39
	docker-registry	v2.2.2
	argo-cd	5.16.13
	kubeview	2.7.1
	prometheus-adapter	v4.1.1
	prometheus-node-exporter	v4.6.0
	kube-state-metrics	v2.3.0

=== “ARM64”

仓库	组件	版本
addon	metrics-server	v3.8.3
	kubernetes-cronhpa-controller	v1.3.0
	vpa	v1.5.0
	skoala-init	v0.18.0

	rabbitmq-cluster-operator	v0.8.0
	eck-operator	v0.5.0
	redis-cluster-operator	v0.5.0
	strimzi-kafka-operator	v0.3.0
	minio-operator	v0.3.0
	jenkins	v0.1.4
	insight-agent	v0.14.5
community	dao-2048	v1.3.1
	kube-node-tuning	v0.3.0
	node-problem-detector	v1.3.1
	kuberhealthy	v90
	vela-core	v1.6.7
	docker-registry	v2.2.2
	prometheus-adapter	v4.1.1
	prometheus-node-exporter	v4.6.0
	kube-state-metrics	v2.3.0

下载 Addon 离线包

本页可下载针对 DCE 5.0 的 Addon 离线包。 有关离线 Addon 的使用，可以在 clusterConfig.yaml 中进行配置，请查阅 DCE 5.0 安装流程。

下载

文件名	版本	架构	文件大小	下载	更新日期
addon-offline-full-package-v0.9.0-amd64.tar.gz	v0.9.0	AMD64	14.4 GB	:arrow_down: 下载	2023-07-07
addon-offline-full-package-v0.9.0-arm64.tar.gz	v0.9.0	ARM64	9.40 GB	:arrow_down: 下载	2023-07-

文件名	版本	架构	文件大小	下载	更新日期

校验

进入离线包的下载目录。

===“AMD64”

执行以下命令校验完整性：

```sh
echo "c4aa34b881380613535c1fce4d97310d69092d3501286c08e12e9374bcfe014d6fda5d94f2d3d3429b734b75471331f2bcd79e31e1c03170bf27adc474b99259  addon-offline-full-package-v0.9.0-amd64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.9.0-amd64.tar.gz: ok
```

===“ARM64”

执行以下命令校验完整性：

```sh
echo "9069ba5ed9fda8386b17187c353d0cc9d11ae64ee9a67c4a463d35e654c6693d6b42b3e59b1cb0f0a749e9b2a60aa181d2d90e33688b0f7a4df072aaf1b5ea98  add
```

on-offline-full-package-v0.9.0-arm64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.9.0-arm64.tar.gz: ok
```

使用

DCE 5.0 支持 Addon 离线化，下载离线包后，打开 clusterConfig.yaml，修改 addonOfflinePackagePath 字段，指定 Addon 所在的路径。

Addon 离线包的组件

v0.9.0 版本的 Addon 离线包内含以下组件：

===“AMD64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.24.1
	velero	3.0.0
	rabbitmq-cluster-operator	0.8.0
	eck-operator	0.5.0
	redis-cluster-operator	0.5.0
	strimzi-kafka-operator	0.3.1
	minio-operator	0.3.0
	jenkins	0.1.13
	insight-agent	0.18.2
	coredns-metrics	0.1.0
```

	harbor-operator	1.4.0
	spiderpool	0.5.0
	contour	11.1.2
	multus-underlay	0.2.3
	f5networks	0.2.0
	metallb	0.13.9
	nmstate	0.74.0
	neuvector	2.4.2
	falco	2.0.17
	falco-event-generator	0.1.1
	falco-exporter	0.8.2
community	dao-2048	1.3.1
	kube-node-tuning	0.3.1
	node-problem-detector	2.3.1
	kuberhealthy	90
	kube-bench	0.1.5
	vela-core	1.6.7
	wordpress	15.2.39
	docker-registry	2.2.2
	argo-cd	5.16.13
	kubeview	2.7.1
	prometheus-adapter	4.1.1
	prometheus-node-exporter	4.6.0
	kube-state-metrics	4.22.3

=== “ARM64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.24.1

	rabbitmq-cluster-operator	0.8.0
	eck-operator	0.5.0
	redis-cluster-operator	0.5.0
	strimzi-kafka-operator	0.3.1
	minio-operator	0.3.0
	jenkins	0.1.13
	insight-agent	0.18.2
community	dao-2048	1.3.1
	kube-node-tuning	0.3.1
	node-problem-detector	2.3.1
	kuberhealthy	90
	vela-core	1.6.7
	docker-registry	2.2.2
	prometheus-adapter	4.1.1
	prometheus-node-exporter	4.6.0
	kube-state-metrics	4.22.3

下载 Addon 离线包

本页可下载针对 DCE 5.0 的 Addon 离线包。 有关离线 Addon 的使用，可以在 clusterConfig.yaml 中进行配置，请查阅 DCE 5.0 安装流程。

下载

文件名	版本	架构	文件大小	下载	更新日期
addon-offline-full-package-v0.8.0-rc2-amd64.tar.gz	v0.8.0-rc2	AMD64		:arrow_down: 下载	2023-06-15
addon-offline-full-package-v0.8.0-rc2-arm64.tar.gz	v0.8.0-rc2	AFM64		:arrow_down: 下载	2023-06-

文件名	版本	架构	文件大小	下载	更新日期
		c			12

校验

进入离线包的下载目录。

===“AMD64”

执行以下命令校验完整性：

```sh
echo "73001a2363b0c7ff804616434b5f7a77e2ea1a56471e54004a0ac5b04350692ca7973f1990080d2905ed0133005343333fb52216fdac4d6551e3e12555fc87d1  addon-offline-full-package-v0.8.0-rc2-amd64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.8.0-rc2-amd64.tar.gz: ok
```

===“ARM64”

执行以下命令校验完整性：

```sh
echo "83205e13b21ef8e9cb4104a6ed9c0463b95331f40e3c5f8a3eb703c0c5a17e1c31d172cedd5b079337ded6c24731d772091f8811a1ffb3b6e328b7cb632732fa  addon
```

-offline-full-package-v0.8.0-rc2-arm64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.8.0-rc2-arm64.tar.gz: ok
```

使用

DCE 5.0 支持 Addon 离线化，下载离线包后，打开 clusterConfig.yaml，修改 addonOfflinePackagePath 字段，指定 Addon 所在的路径。

Addon 离线包的组件

v0.8.0-rc2 版本的 Addon 离线包内含以下组件：

=== “AMD64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.22.2
	velero	3.0.0
	rabbitmq-cluster-operator	0.8.0
	eck-operator	0.5.0
	redis-cluster-operator	0.5.0
	strimzi-kafka-operator	0.3.1
	minio-operator	0.3.0
	jenkins	0.1.10
	insight-agent	0.17.1
	coredns-metrics	0.1.0
```

	harbor-operator	1.4.0
	spiderpool	0.5.0
	contour	11.1.2
	multus-underlay	0.2.3
	f5networks	0.2.0
	metallb	0.13.9
	nmstate	0.74.0
	neuvector	2.4.2
	falco	2.0.17
	falco-event-generator	0.1.1
	falco-exporter	0.8.2
community	dao-2048	1.3.1
	kube-node-tuning	0.3.1
	node-problem-detector	2.3.1
	kuberhealthy	90
	kube-bench	0.1.5
	vela-core	1.6.7
	wordpress	15.2.39
	docker-registry	2.2.2
	argo-cd	5.16.13
	kubeview	2.7.1
	prometheus-adapter	4.1.1
	prometheus-node-exporter	4.6.0
	kube-state-metrics	4.22.3

=== “ARM64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.22.2

	rabbitmq-cluster-operator	0.8.0
	eck-operator	0.5.0
	redis-cluster-operator	0.5.0
	strimzi-kafka-operator	0.3.1
	minio-operator	0.3.0
	jenkins	0.1.10
	insight-agent	0.17.1
community	dao-2048	1.3.1
	kube-node-tuning	0.3.1
	node-problem-detector	2.3.1
	kuberhealthy	90
	vela-core	1.6.7
	docker-registry	2.2.2
	prometheus-adapter	4.1.1
	prometheus-node-exporter	4.6.0
	kube-state-metrics	4.22.3

下载 Addon 离线包

本页可下载针对 DCE 5.0 的 Addon 离线包。 有关离线 Addon 的使用，可以在 clusterConfig.yaml 中进行配置，请查阅 DCE 5.0 安装流程。

下载

文件名	版本	文件大小	下载	更新日期
addon-offline-full-package-v0.8.0-amd64.tar.gz	v0.8.0	18.4 GB	:arrow_down: 下载	2023-06-01
addon-offline-full-package-v0.8.0-arm64.tar.gz	v0.8.0	8.3 GB	:arrow_down: 下载	2023-06-01

文件名	版本	架构	文件大小	下载	更新日期

校验

进入离线包的下载目录。

===“AMD64”

执行以下命令校验完整性：

```sh
echo "9397d2af8cae997996b6f08cebc6318c1219c57c2f723cca76053cf34520ab83bc43125fff9498056ca732645b6941620a182792ee8da2b6a38d07b2cb55c9a2 addon-offline-full-package-v0.8.0-amd64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.8.0-amd64.tar.gz: ok
```

===“ARM64”

执行以下命令校验完整性：

```sh
echo "d522da328a286b0d063d018ee82fa74b1b8da8b7f601432c9653287774e362669e1569c87de705a14eec6d4d7613dc4b995eb104f3bc686d44d5f6dcee8a8f7d add
```

on-offline-full-package-v0.8.0-arm64.tar.gz" | sha512sum -c
```

校验成功后打印结果类似于：

```none
addon-offline-full-package-v0.8.0-arm64.tar.gz: ok
```

使用

DCE 5.0 支持 Addon 离线化，下载离线包后，打开 clusterConfig.yaml，修改 addonOfflinePackagePath 字段，指定 Addon 所在的路径。

Addon 离线包的组件

v0.8.0 版本的 Addon 离线包内含以下组件：

=== “AMD64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.22.2
	velero	3.0.0
	rabbitmq-cluster-operator	0.8.0
	eck-operator	0.5.0
	redis-cluster-operator	0.5.0
	strimzi-kafka-operator	0.3.1
	minio-operator	0.3.0
	jenkins	0.1.10
	insight-agent	0.17.0-rc5
	coredns-metrics	0.1.0
```

	harbor-operator	1.4.0
	spiderpool	0.5.0
	contour	11.1.2
	multus-underlay	0.2.3
	f5networks	0.2.0
	metallb	0.13.9
	nmstate	0.74.0
	neuvector	2.4.2
	falco	2.0.17
	falco-event-generator	0.1.1
	falco-exporter	0.8.2
community	dao-2048	1.3.1
	kube-node-tuning	0.3.1
	node-problem-detector	2.3.1
	kuberhealthy	90
	kube-bench	0.1.5
	vela-core	1.6.7
	wordpress	15.2.39
	docker-registry	2.2.2
	argo-cd	5.16.13
	kubeview	2.7.1
	prometheus-adapter	4.1.1
	prometheus-node-exporter	4.6.0
	kube-state-metrics	4.22.3

=== “ARM64”

仓库	组件	版本
addon	metrics-server	3.8.3
	kubernetes-cronhpa-controller	1.3.0
	vpa	1.5.0
	skoala-init	0.22.2

		rabbitmq-cluster-operator	0.8.0
		eck-operator	0.5.0
		redis-cluster-operator	0.5.0
		strimzi-kafka-operator	0.3.1
		minio-operator	0.3.0
		jenkins	0.1.10
		insight-agent	0.17.0-rc5
community		dao-2048	1.3.1
		kube-node-tuning	0.3.1
		node-problem-detector	2.3.1
		kuberhealthy	90
		vela-core	1.6.7
		docker-registry	2.2.2
		prometheus-adapter	4.1.1
		prometheus-node-exporter	4.6.0
		kube-state-metrics	4.22.3

应用工作台 Amamba

本页可下载应用工作台模块各版本的离线安装包。

下载

版本	文件大小	安装包	校验文件	更新日期
0.19.5	7 3 . 9 3 MB	:arrow_down: amamba_0.19.5_amd64.tar	:arrow_down: amamba_0.19.5_amd64.tar.sha256	2026-03-08

版本	安装包文件大小	更新日期	校验文件
			.5_amd64_checksum.sha512sum

版本	离线安装包文件大小	校验文件	更新日期
			m

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat amamba_v0.19.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
amamba_v0.19.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

镜像仓库 Kangaroo

本页可下载镜像仓库模块各版本的离线安装包。

下载

版本	文件大小	安装包	校验文件	更新日期
v0.10.0	29.42 MB	:arrow_down: kangaroo_v0.10.0_amd64.tar	:arrow_down: kangaroo_v0.10.0_amd64.tar.sha256	2023-08-22

版本	安装包文件大小	更新日期	校验文件
			.10.0._amd64_checksum.sha512

版本	安装包文件大小	校验文件	更新日期

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat kangaroo_0.10.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
kangaroo_0.10.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

安全管理 Dowl

本页可下载安全管理模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.4.0	AMD64	16 3 MB	:arrow_down: dowl_v0.4.0_amd64.tar	:arrow_down: dowl_v0.4.0_amd64.tar.sha256	2026-03-18

版本	架构	文件大小	安装包	更新日期	校验文件

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat dowl_v0.4.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
dowl_v0.4.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

可观测性 Insight

本页可下载可观测性模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.19.1	AMD64	2.6 4 GB	:arrow_down: insight_v0.19.1_amd64.tar	:arrow_down: insight_v0.19.1_amd64.tar.sha256sum	2025-03-04

版本	架构	文件大小	安装包	校验文件	更新日期
				md64_checksum.sha512sum	
v0.13.1	AMD64	2.41	:arrow_down: insight_v0.13.1_amd64.tar	:arrow	2022

版本	安装包文件大小	校验文件	更新日期
4	4GB	_download:insight_v0.13.1_amd64_check	–12–30

版本	架构	文件大小	安装包	校验文件	更新日期
				sum.sha512sum	

校验

在下载离线安装包和校验文件的目录，以 v0.13.1_amd64 为例，执行以下命令校验完整性：

```
echo "$(cat insight_v0.13.1_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
insight_v0.13.1_amd64.tar: ok
```

安装

参阅可观测性安装流程进行安装。

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

容器管理 Kpanda

本页可下载容器管理模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.19.0	AMD64	2.4 GB	:arrow_down: kpanda_v0.19.0_amd64.tar	:arrow_down: kpanda_v0.19.0_amd64.tar.sha256sum	2024-03-07

版本	安装包文件大小	更新日期	校验文件
9.0	amd64	check	sum.sha512s

版本	架构	文件大小	安装包	校验文件	更新日期

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat kpanda_v0.19.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
kpanda_v0.19.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

运营管理

本页可下载运营管理模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.4.1	AMD64	93.72MB	:arrow_down: gmagpie_v0.4.1_amd64.tar	:arrow_down: gmagpie_v0.4.1_	2023-08-29

版本	架构	文件大小	安装包	校验文件	更新日期
				amd64_checksum.sha512sum	
v0.3.2	AMD	91.	:arrow_down: gmagpie_v0.3.2_amd64.tar	:arro	202

版本	架构	文件大小	安装包校验文件	更新日期
			w_download:gmagpie_v0.3.2_amd64_ch	3–08–02

版本	架构	文件大小	安装包	校验文件	更新日期
v0.3.0	AMD64	56.90 MB	:arrow_down: gmagpie_v0.3.0_amd64.tar	:arrow_down: gmagpie_v0.3.0_amd64.tar.sha512sum	2023-06-28

版本	架构	安装包文件大小	更新日期	校验文件
				agpie_v0.3.0_amd64_checksum.sh

版本	架构	文件大小	安装包	校验文件	更新日期
v0.2.2	AMD64	31.64MB	:arrow_down: gmagpie_v0.2.2_amd64.tar	:arrow_down: gmagpie_v0.2.2_amd64.tar.sum	2023-05-30

版本	安装包文件大小	更新日期	校验文件
			_amd64_checksum.sha512sum

校验

在下载离线安装包和校验文件的目录，以 v0.2.2_amd64 为例，执行以下命令校验完整性：

```
echo "$(cat gmagpie_v0.2.2_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
gmagpie_v0.2.2_amd64.tar: ok
```

安装

参阅全局管理安装流程进行安装。

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

应用备份 Kcoral

本页可下载应用备份模块各版本的离线安装包。

下载

版本	文件大小	安装包	校验文件	更新日期
v0.4.0	59.62 MB	:arrow_down: kcoral_v0.4.0_amd64.tar	:arrow_down: kcoral_v0.4.0_amd64.tar.sha256sum	2023-08-08

版本	安装包文件大小	更新日期	校验文件
			.0_amd64_checksum.sha512sum

版本	离线安装包文件大小	校验文件	更新日期
			m

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat kcoral_v0.4.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
kcoral_v0.4.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

全局管理

本页可下载全局管理模块各版本的离线安装包。

下载

版本	文件大小	安装包	校验文件	更新日期
v0.20.0	57.02MB	:arrow_down: ghippo_v0.20.0_amd64.tar	:arrow_down: ghippo_v0.20.0_amd64.tar.sha256sum	2023-08-28

版本	架构	文件大小	安装包	校验文件	更新日期
v0.19.0	AMD64	45	:arrow_down: ghippo_v0.19.0_amd64.tar	:arrow_down: ghippo_v0.19.0_amd64_checksum.sha512sum	2020

版本	架构	文件大小	安装包	校验文件	更新日期
	D64	9.68MB		ow_down:ghippo_v0.19.0_amd64_ch	23-07-28

版本	架构	文件大小	安装包	校验文件	更新日期
v0.18.1	AMD64	42.59.62 7MB	:arrow_down: ghippo_v0.18.1_amd64.tar	:arrow_down: ghippo_v0.18.1_amd64.tar.sha512sum	2023-07-07

版本	架构	安装包文件大小	更新日期	校验文件
				ppo_v0.18.1_amd64_checksum.sha

版本	架构	文件大小	安装包	校验文件	更新日期
v0.18.0	AMD64	47MB	:arrow_down: ghippo_v0.18.0_amd64.tar	:arrow_down: ghippo_v0.18.0_amd64.tar.sum	2023-06-27

版本	架构	文件大小	安装包	校验文件	更新日期
v0.17.1	AMD64	10.0	:arrow_down: ghippo_v0.17.1_amd64.tar	:arrow_down:	2023

版本	架构	文件大小	安装包	校验文件	更新日期
		604 KB		_download:ghippoo_v0.17.1_amd64_check	–06–08

版本	架构	文件大小	安装包	校验文件	更新日期
v0.17.0	AMD64	10.06 KB	:arrow_down: ghippo_v0.17.0_amd64.tar	:arrow_down: ghippo_v0.17.0_amd64.tar.sha512sum	2023-05-31

版本	安装包文件大小	更新日期	校验文件
			o_v0.17.0_amd64_checksum.sha512

版本	架构	文件大小	安装包	校验文件	更新日期
v0.16.1	AMD64	10.60KB	:arrow_down: ghippo_v0.16.1_amd64.tar	:arrow_down: ghippo_v0.16.1_amd64.tar.sum	2023-04-27

版本	架构	文件大小	安装包	校验文件	更新日期
v0.16.0	AMD64	10.06 KB	:arrow_down: ghippo_v0.16.0_amd64.tar	:arrow_down: ghippo_v0.16.0_amd64_checksum.sha512sum	2023-10-20

版本	架构	文件大小	安装包	校验文件	更新日期
		B		own:ghippo_v0.16.0_amd64_checks	4-27

版本	架构	文件大小	安装包	校验文件	更新日期
v0.15.0	AMD64	42.60MB	:arrow_down: ghippo_v0.15.0_amd64.tar	:arrow_down: ghippo_v0.15.0_amd64.tar.sha512sum	2023-03-29

版本	架构	文件大小	安装包	校验文件	更新日期
				0.15.0_amd64_checksum.sha512su	

版本	架构	文件大小	安装包	校验文件	更新日期
v0.14.0	AMD64	10.0 KB	:arrow_down: ghippo_v0.14.0_amd64.tar	:arrow_down: ghippo_v0.14.0_amd64.md5	2023-02-27

版本	架构	文件大小	安装包	校验文件	更新日期
v0.13.2	AMD64	49.90MB	:arrow_down: ghippo_v0.13.2_amd64.tar	:arrow_down: ghippo_v0.13.2_amd64.tar.sha512sum	2022-12-02

版本	架构	安装包文件大小	更新日期	校验文件
			9	https://github.com/ppo_v0.13.2_amd64_checksum

版本	架构	文件大小	安装包	校验文件	更新日期
v0.13.0	AMD64	489MB	:arrow_down: ghippo_v0.13.0_amd64.tar	.sha512sum :arrow_down: ghippo_v0.13.0_amd64.tar.sha512sum	2022-12-29

版本	架构	安装包文件大小	校验文件	更新日期
v0.12.1	AMD64	:arrow_down: ghippo-0.12.1-amd64.0_checksum.sha512sum	:a	2

版本	架构	文件大小	安装包	校验文件	更新日期
	MD64	4 2 6 4 B	amd64.bundle.tar	rrow_download:ghippo_v0.12.1_amd64_c	02–11–29

版本	架构	安装包文件大小	校验文件	更新日期

校验

在下载离线安装包和校验文件的目录，以 v0.13.2_amd64 为例，执行以下命令校验完整性：

```
echo "$(cat ghippo_v0.13.2_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
ghippo_v0.13.2_amd64.tar: ok
```

安装

参阅全局管理安装流程进行安装。

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

集群巡检 Kcollie

本页可下载集群巡检模块各版本的离线安装包。

下载

版本	文件大小	安装包	校验文件	更新日期
v0.4.0	20.15 MB	:arrow_down: kcollie_v0.4.0_amd64.tar	:arrow_down: kcollie_v0.4.0_amd64_checksum.sha512sum	2023-10-08

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat kcollie_v0.4.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
kcollie_v0.4.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

网络 Spidernet

本页可下载网络 Spidernet 模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.6.0	AMD64	547MB	:arrow_down: spidernet_v0.6.0_amd64.tar	:arrow_down: spidernet_v0.6.0_amd64_	2023-04-26

版本	架构	文件大小	安装包	校验文件	更新日期
v0.5.0	AMD64	51.68MB	:arrow_down: spidernet_v0.5.0_amd64.tar	:arrow_down: spidernet_v0.5.0_amd64.tar.sha512sum	2023-04-25

版本	架构	文件大小	安装包	校验文件	更新日期

校验

在下载离线安装包和校验文件的目录，以 v0.5.0_amd64 为例，执行以下命令校验完整性：

```
echo "$(cat spidernet_v0.5.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
spidernet_v0.5.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

服务网格 Mspider

本页可下载服务网格模块各版本的离线安装包。

下载

版本	架构	文件大小	安装包	校验文件	更新日期
v0.19.0	AMD64	88.68MB	:arrow_down: mspider_v0.19.0_amd64.tar	:arrow_down: mspider_v0.19.0_amd64.tar.sha256sum.txt	2023-08-31

版本	安装包文件大小	更新日期	校验文件
9.0	amd64	check	sum.sha512s

版本	架构	文件大小	安装包	校验文件	更新日期
v0.18.0	AMD64	241GB	:arrow_down: mspider_v0.18.0_amd64.tar	:arrow_down: mspider_v0.18.0_amd64.tar.sum	2023-08-23

版本	安装包文件大小	更新日期	校验文件
			amd64_checksum.sha512sum

校验

在下载离线安装包和校验文件的目录，执行以下命令校验完整性：

```
echo "$(cat mspider_v0.18.0_amd64_checksum.sha512sum)" | sha512sum -c
```

校验成功后打印结果类似于：

```
mspider_v0.18.0_amd64.tar: ok
```

安装

如果是初次安装，请申请免费体验或联系我们授权：电邮 info@daocloud.io 或致电 400 002 6898。 如果有任何许可密钥相关的问题，请联系 DaoCloud 交付团队。

离线升级

应用工作台支持离线升级。您需要先从安装包中加载镜像，然后执行相应命令进行升级。

从安装包中加载镜像

支持通过两种方式加载镜像。

通过 charts-syncer 同步镜像

如果环境中存在镜像仓库，建议通过 charts-syncer 将镜像同步到镜像仓库，更加高效便捷。

1. 创建 load-image.yaml 文件。

   注意：该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

   yaml title="load-image.yaml" source: intermediateBundlesPath: ghippo-offline # 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径 target: containerRegistry: 10.16.10.111 # 镜像仓库 url containerRepository: amamba # 镜像仓库的具体项目 repo: kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别，比如 ChartMuseum url: http://10.16.10.111/chartrepo/amamba # 需更改为 chart repo url auth: # 用户名/密码 username: "admin" password: "Harbor12345" containers: auth: # 用户名/密码 username: "admin" password: "Harbor12345"

   !!! note “若当前环境未安装 chart repo，也可以通过 chart-syncer 将 chart 导出为 tgz 文件，并存放在指定路径。”

   ```yaml title="load-image.yaml"
   source:
   ```

```
    intermediateBundlesPath: amamba-offline #  到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
  target:
    containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
    containerRepository: release.daocloud.io/amamba # 需更改为你的镜像仓库
    repo:
      kind: LOCAL
      path: ./local-repo # chart 本地路径
    containers:
      auth:
        username: "admin" # 你的镜像仓库用户名
        password: "Harbor12345" # 你的镜像仓库密码
```

2. 执行如下命令同步镜像。

```
charts-syncer sync --config load-image.yaml
```

通过 Docker 或 containerd 直接加载镜像

1. 执行下列命令解压镜像。

```
tar xvf amamba.bundle.tar
```

解压成功后会得到 3 个文件: images.tar、hints.yaml、original-chart。

2. 执行如下命令从本地加载镜像到 Docker 或 containerd。

注意：需要在集群中的 **每个节点** 上执行如下操作。加载完成后需要为镜像打标签，确保 Registry、Repository 与安装时一致。

```
docker load -i images.tar # for Docker
ctr image import images.tar # for containerd
```

升级

1. 检查应用工作台的 Helm 仓库是否存在。

helm repo list | grep amamba

若返回结果为空或出现 Error: no repositories to show 提示，则执行如下命令添加应用工作台的 Helm 仓库。

heml repo add amamba http://{harbor url}/chartrepo/{project}

2. 更新应用工作台的 Helm 仓库。

Helm 版本过低会导致失败。若失败，请尝试执行 helm update repo
helm repo update amamba

3. 备份 --set 参数。在升级全局管理版本之前，建议执行如下命令备份旧版本的 --set 参数。

helm get values ghippo -n ghippo-system -o yaml > amamba.bak.yaml

4. 选择想安装的应用工作台版本（建议安装最新版本）。

$ helm search repo amamba-release-ci --versions |head
NAME CHART VERSION APP VERSION DESCRIPTION
amamba-release-ci/amamba 0.14.0 0.14.0 Amamba is the entrypoint to DCE 5.0, provides de...

5. 修改 amamba.bak.yaml 文件里的 registry 和 tag。

??? note“点击查看示例的 YAML 文件”

```yaml title="amamba.bak.yaml"
amambaSyncer:
  resources:
    limits:
      cpu: 200m
      memory: 256Mi
    requests:
```

```
        cpu: 20m
        memory: 128Mi
  apiServer:
    configMap:
      debug: true
    fromJar:
      image:
        registry: releas-ci.daocloud.io
        repository: docker/library/openjdk
        tag: 11.0-jre-slim
    image:
      registry: release-ci.daocloud.io
      repository: amamba/amamba-apiserver
    resources:
      limits:
        cpu: "2"
        memory: 2Gi
      requests:
        cpu: 20m
        memory: 150Mi
  argo-cd:
    applicationSet:
      enabled: false
      image:
        repository: release-ci.daocloud.io/quay/argoproj/argocd
        tag: v2.4.12
    controller:
      image:
        repository: release-ci.daocloud.io/quay/argoproj/argocd
        tag: v2.4.12
      resources:
        limits:
```

```
        cpu: "2"
        memory: 2Gi
      requests:
        cpu: 100m
        memory: 256Mi
dex:
  enabeld: true
  image:
    repository: release-ci.daocloud.io/ghcr/dexidp/dex
    tag: v2.32.0
  initImage:
    repository: release-ci.daocloud.io/quay/argoproj/argocd
    tag: v2.4.12
  resources:
    limits:
      cpu: 50m
      memory: 64Mi
    requests:
      cpu: 10m
      memory: 16Mi
enabled: true
notifications:
  enabled: false
redis:
  enabled: true
  image:
    repository: release-ci.daocloud.io/docker/library/redis
    tag: 7.0.4-alpine
  metrics:
    enabled: false
  resources:
    limits:
```

```
        cpu: 200m
        memory: 128Mi
      requests:
        cpu: 5m
        memory: 16Mi
  repoServer:
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 5m
        memory: 8Mi
  server:
    image:
      repository: release-ci.daocloud.io/quay/argoproj/argocd
      tag: v2.4.12
    resources:
      limits:
        cpu: 250m
        memory: 256Mi
      requests:
        cpu: 5m
        memory: 8Mi
    service:
      nodePortHttp: 31886
      nodePortHttps: 31887
      type: NodePort
argo-rollouts:
```

```
  controller:
    image:
      registry: release-ci.daocloud.io
      repository: quay/argoproj/argo-rollouts
      tag: v1.2.0
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 16m
        memory: 128Mi
  dashboard:
    enabled: true
    image:
      registry: release-ci.daocloud.io
      repository: quay/argoproj/kubectl-argo-rollouts
      tag: v1.2.0
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
      requests:
        cpu: 10m
        memory: 16Mi
    service:
      nodePort: 30070
      type: NodePort
  enabled: true
devopsServer:
  enabled: true
  image:
```

```
      registry: release-ci.daocloud.io
      repository: amamba/amamba-devops-server
    resources:
      limits:
        cpu: "2"
        memory: 2Gi
      requests:
        cpu: 50m
        memory: 160Mi
global:
  amamba:
    imageTag: v0.13-dev-a8ca782a
  imageRegistry: release-ci.daocloud.io
mysql:
  busybox:
    repository: docker/busybox
    tag: 1.35.0
  image:
    repository: docker/mysql
    tag: 5.7.30
  persistence:
    size: 20Gi
  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 20m
      memory: 256Mi
ui:
  image:
    tag: v0.15.0-dev-fd64789e
```

```
  resources:
    limits:
      cpu: 100m
      memory: 128Mi
    requests:
      cpu: 5m
      memory: 4Mi
```

6. 执行如下命令进行升级

```
helm upgrade amamba . \
  -n amamba-system \
  -f ./amamba.bak.yaml
```

优势

「DaoCloud 道客」凭借多年云原生领域的领先经验，全新开发的应用工作台具有以下产品优势：

- 企业级 CI/CD

  使用集中管理的容器化工作流，持续开发持续交付，支持横向扩展，可以在高可用性（HA）环境中支持成千上万个并发用户和流水线。

- 云原生为底座

  应用工作台实现了应用构建环境的容器化管理，自定义构建机器等资源，确保流水线级资源隔离，同时支持多种类型的应用部署，使得云原生应用程序的交付更加简单。

- 提升研发效能

  在可视化的流水线构建环境中，预先封装声明性步骤，无需脚本就能创建复杂的流水线，操作简单。可根据资源大小自动创建/销毁环境，避免资源浪费。

- 灰度发布

  帮助用户渐进式更新应用，从而保障新特性能平稳上线。支持金丝雀部署、蓝绿部署等应用发布策略。

  如果一个应用的发布策略配置为 ABTest，其 A 版本访问流量为 80%，B 版本访问流量为 20%，则灰度发布后将能够持续访问应用的两个版本，轻松统计出该应用不同版本的分流比例。

- GitOps

提供基于 Kubernetes 的声明性 GitOps 持续交付能力，通过版本控制应用，从而实现自动化的应用部署与生命周期管理。

- 流水线即代码

  基于声明性（语法）的流水线易于学习，不同步骤的语法实现了标准化，其配置具有版本可控、模块化、可重用和声明性等特点。

- 综合集成

  集成社区流行的 DevOps 工具，单个流水线中的步骤可以在多操作系统、多体系结构节点上运行，支持在公有云、私有云或主机上独立部署，能很好地集成企业自有的系统和平台。 支持集成业界主流软件，如 Kubernetes、GitLab、SonarQube、Harbor 等。

- 多年行业沉淀

  在云原生领域耕耘多年，在金融、电商、制造等行业上百家企业有大规模实践经验，能够提供稳定可靠的 IT 交付生产线，帮助企业打造适合研发运行一体化平台。

应用工作台 Release Notes

本页列出应用工作台的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-07-31

v0.19.2

新功能

- **新增** Issues 界面列表支持跳转到 Jira 查看详情
- **新增** 集群增加了状态标识
- **新增** OAM 应用支持编辑 YAML、添加组件、编辑组件、编辑运维特征等功能
- **新增** 支持流水线根据最近时间运行时间排序
- **新增** 流水线支持 Sonarqube 配置、代码质量结果步骤
- **新增** 工具链支持 Sonarqube 实例的集成
- **新增** 向导基于 git、jar、容器镜像创建应用时默认绑定原生应用

优化

- **优化** 优化集成 gitlab 时到 jenkins 凭证的步骤
- **优化** 管理员视角下列出绑定了工作空间的 sonarqube 的绑定时间以及增加了分野
- **优化** 修复了 apiserver 的缓存不过期的问题

2022-06-30

v0.18.1

新功能

- **新增** 流水线权限支持了运行流水线权限点
- **新增** 支持对原生应用的更新、删除、拓扑

- **新增** 流水线支持选择集成的 gitlab 仓库的项目
- **新增** 支持 OAM 应用的创建、详情等基础功能
- **新增** 安装 Kubevela 开源组件
- **新增** 支持工具链资源的自动同步和手动同步，同时暴露对应的 metric 指标
- **新增** 流水线模板添加了版本信息
- **新增** 支持创建原生应用，查看原生应用的资源信息
- **新增** 新增集群范围的 nginx ingress-class 的检查

修复

- **修复** 目标集群没有安装原生应用的 CRD 时，原生应用列表页会报错
- **修复** 更新流水线凭证时，当不返回密码信息的时候，没有修改任何值导致返回 null 的修复
- **修复** 同步导致的多分支流水线的“孤儿策略”为空的时候显示为 -1 的问题

优化

- **优化** API 优化了流水线异常场景的处理
- **优化** API 为集群添加一个状态字段显示集群运行时状态
- **优化** 硬编码优化，支持部署到不同的 namespace 中
- **优化** 当集成 gitlab 的时候，返回 gitlab 的 address
- **优化** 基于 Jenkinsfile 的流水线和多分支流水线集成 gitlab 的时候返回 toolchainID

2023-6-15

v0.17.4

修复

- **修复** 修复了金丝雀发布任务中操作异常问题

2022-05-31

v0.17.3

新功能

- **新增** API 支持通过代码选择器来选择集成的代码仓库
- **新增** 支持基于 nginx-ingress 的灰度发布策略
- **新增** API 支持应用的资源拓扑
- **新增** API 统一了凭证
- **新增** 流水线模板添加了版本信息
- **新增** 支持创建原生应用，查看原生应用的资源信息

修复

- **修复** 基于容器镜像中容器不支持下划线的问题
- **修复** 蓝绿发布的副本数报错的问题
- **修复** 同步导致的多分支流水线的“孤儿策略”为空的时候显示为-1 的问题

优化

- **优化** jira-provider 的 http-client 增加了重试功能。处理流水线事件的处理速度
- **优化** 处理流水线事件的处理速度
- **优化** 重构了 from-git 和 from-jar 的流水线模板
- **优化** 向 Jenkins 请求的 url， 普通流水线的和多分支统一 url

2022-04-30

v0.16.1

新功能

- **新增** 创建应用时独立出可观测性相关的配置，包括指标监测，链路追踪和 JVM 监控
- **新增** 集成工具链支持 Jenkins
- **新增** 支持多分支流水线的创建及管理
- **新增** 支持管理员视角下的工具链集成功能

- **新增** API 支持了蓝绿发布，包括创建，删除，回滚，升级，详情等

修复

- **修复** 管理员绑定项目到指定工作空间下下，获取关联工具链实例详情时，无法正常获取的 bug
- **修复** 基于 from-jar 和 from-git 创建应用过程中出现 IMAGE_TAG 为空的时候，在对应的模板文件里里设置为 latest

优化

- **优化** 统一整合工具链接口

2022-03-31

v0.15.1

新功能

- **新增** 容器配置支持更多选项(生命周期，环境变量和数据存储)
- **新增** jira 工具链集成接口，包括 jira 实例的 CRUD 和实例项目的列表
- **新增** 缺陷列表支持模糊查询、类型、状态、优先级的查询
- **新增** 针对 ghippo 的 smtp 配置的 list/watch 机制，支持 jenkins 的邮件通知功能
- **新增** 允许创建自定义角色
- **新增** API 支持工具链集成添加 jenkins 类型，添加获取 jenkins 列表和详情
- **新增** API 支持多分支流水线
- **新增** 流水线运行记录详情支持查看测试报告
- **新增** 流水线新增 SVN 、Junit 收集测试报告步骤
- **新增** 支持通过模板创建流水线
- **新增** 支持创建自定义流水线模板

优化

优化 调整 jenkins 的 cpu，内存参数

修复

修复 修复了如果 page 小于 0，则无法搜索到对应工具链数据的问题，添加了部分可供前端展示的字段

2022-02-22

v0.14.0

新功能

- **新增** API 支持使用内置模板创建流水线
- **新增** API 支持自定义模板的创建、更新、删除
- **新增** apiserver、devopserver 支持访问 https 协议的 Jenkins
- **新增** event-proxy 组件支持 Jenkins 的事件持久化
- **新增** 访问 Jenkins 时支持重试和限流

优化

优化 优化了重新运行某条流水线失败时的提示语

修复

- **修复** 修复了凭证在数据库和 Jenkins 里不一致时的问题
- **修复** 修复了创建灰度发布时流量比例大于 100 也能创建成功的问题

2022-01-30

v0.13.0

新功能

- **新增** 支持级联删除，可以根据实际情况选择需要删除的资源
- **新增** 灰度发布支持编辑 YAML 功能
- **新增** 部署应用工作台的 Chart 包中移除了 Jenkins，支持使用单独的 Helm Chart 安装 Jenkins

优化

优化 制品报告下载，不在 apiserver 中全部缓存，采用 io 分块的模式部分缓存转发

修复

修复 修复了持续部署中添加私有仓库报错问题

2022-12-30

v0.12.0

新功能

- **新增** 创建应用时支持同时开启传统微服务和服务网格
- **新增** 查询灰度发布支持模糊查询，查询持续部署支持模糊查询和根据状态检索，查询 gitops 仓库支持模糊查询和状态检索
- **新增** 添加根据 GVR 获取 jso 数据的接口
- **新增** 添加获取 helm 信息的接口，可以通过 argocd 部署 helm chart
- **新增** Jenkins 配置文件的热加载以及 syncer 配置文件的热加载
- **新增** ListRegistry 接口返回值新增 Nacos 是否需要认证字段
- **新增** 流水线制品报告列表接口和下载制品报告接口
- **新增** 流水线步骤日志支持获取全量日志
- **新增** 查询插件信息接口添加了缓存
- **新增** 直接复制已有的流水线来生成新的流水线
- **新增** 开启微服务时接入指标监控提供差异化配置，可自定义路径
- **新增** Jenkins 的 trace 的对接
- **新增** ListWorkload 接口返回值添加应用关联资源

修复

- **修复** 重新运行流水线，概率性的 panic 的修复
- **修复** 创建微服务时链路追踪中用到的 instrumentation 镜像离线化问题
- **修复** 灰度发布的监控分析失败
- **修复** 支持设置 argo-rollouts controller replicas

2022-12-18

v0.11.0

新功能

- **新增** Ghippo 和 Skoala 的 trace 对接，Ghippo 自身只支持部分 api 的 trace
- **新增** helm 包离线情况下的镜像检查的流水线
- **新增** 针对添加迁移 jenkins 后，credential 失效的问题,添加解决方案
- **新增** 凭证列表支持按名称模糊搜索

优化

- **优化** 通过调整 jvm 参数，降低 jenkins 在没有工作负载的情况下的内存占用
- **优化** gitops 模块创建仓库时增加对仓库可连接性的判断
- **优化** 实时同步流水线运行状态等信息导致 jenkins 压力过大，改成懒加载的方式
- **优化** 优化了 api 发版流程，支持单独发版

修复

- **修复** 启用微服务的情况下如果创建失败则无限制进行回滚的问题
- **修复** 当集群的 kubeconfig 变更导致的 argocd 应用部署/删除失败，以及 jenkins 中 credential 失效的问题

2022-11-30

v0.10.0

新功能

- **新增** 新增了 gitops 中的仓库功能，支持导入、删除
- **新增** 新增了 gitops 应用的同步功能

优化

优化 优化了应用接入服务网格流程

修复

- **修复** 修复了 admin 用户未对部署目标（cluster/namespace）鉴权的问题
- **修复** 修复了 gitops 应用创建时间、同步开始时间和同步结束时间为 Invalida date 的错误
- **修复** 修复了获取 nacos 注册中心列表数据的错误
- **修复** 修复了列出工作负载接口通过名称排序报错
- **修复** 修复了集群解绑并重新绑定后，在 ArgoCD 中 destionation 中的 cluster 和 namespace 丢失的问题
- **修复** 修复了在更新 namespace 的 label 导致 namespace 和 workspace 绑定关系丢失的问题
- **修复** 修复了在完成流水线后，同步 jenkins 的 config 到数据库时，trigger 转换的错误
- **修复** 修复了因为集群的 kubeconfig 变更导致的 ArgoCD cluster 和 jenkins 中 kubeconfig 类型的 credential 不同步的问题
- **修复** 修复了仓库列表出现的无序和分页问题
- **修复** 修复了 from-jar 上传超过 32M 文件失败的问题

2022-11-18

v0.9.0

新功能

- **新增** jenkins-agent 镜像持续发布
- **新增** 添加了使用中间件的数据库的选项
- **升级** jenkins 从 2.319.1 升级到 2.346.2,kubernetes plugin 升级到 3734.v562b_b_a_627ea_c，相关的 plugins 也作了升级

优化

- **优化** 获取 rollout 镜像列表，应用组列表，原生应用列表等性能优化
- **优化** from-jar 用到的 image 不再写死在源代码中，通过 env 的方式传递，并保证安装器能正确获取

修复

- **修复** rollout 在不同 workspace 下无法区分的问题
- **修复** gitops 模块未鉴权的问题
- **修复** 偶现的 pipeline step 状态不正确的问题
- **修复** 获取 helmchart 的 description 为空的问题
- **修复** 创建 namespace 没有校验 storage 的问题
- **修复** list argocd repository 出现的无序和分页问题的修复
- **修复** from-jar 上传超过 32M 文件失败的问题的修复
- **修复** 获取 pipeline 日志的时候如果日志量过大则无法获取全量日志的问题

流水线相关问题

本页列出使用应用工作台的流水线功能时时可能遇到的一些问题，并给出相应的解决方案。

执行流水线时报错

当 Jenkins 所在集群与应用部署集群跨数据中心时，网络通信延迟会变高，可能遇到如下的报错信息：

E0113 01:47:27.690555 50 request.go:1058] Unexpected error when reading response body: net/http: request canceled (Client.Timeout or context cancellation while reading body)
error: unexpected error when reading response body. Please retry. Original error: net/http: request canceled (Client.Timeout or context cancellation while reading body)

解决方案：

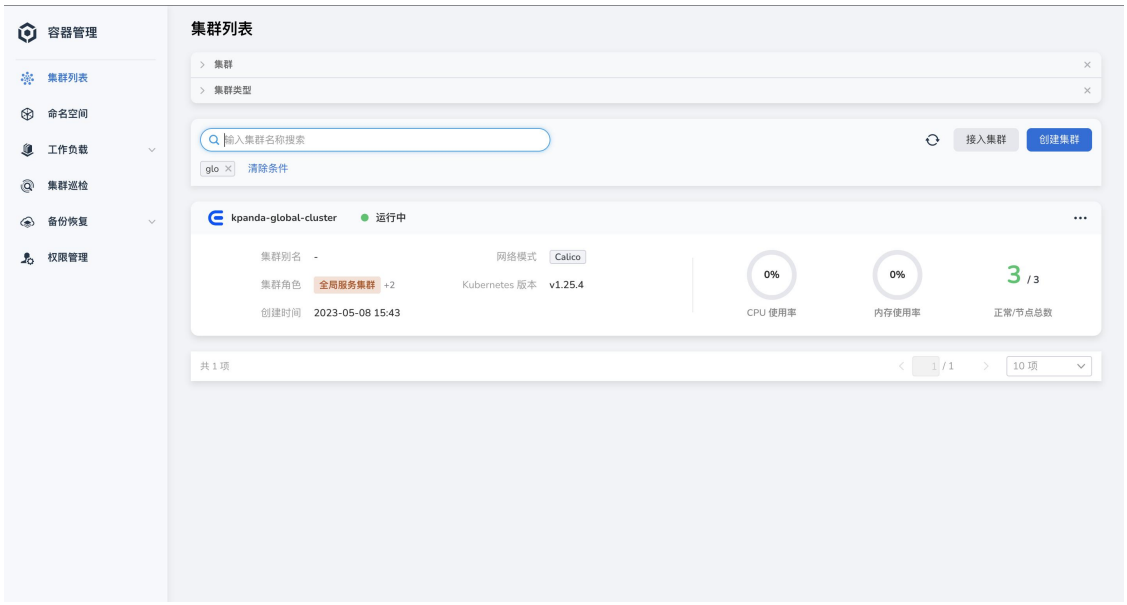
在该流水线的 Jenkinsfile 中将部署命令由 kubectl apply –f 修改为 kubectl apply –f . ––request–timeout=30m。

如何更新内置 Label 的 podTemplate 镜像？

应用工作台通过 podTemplate 能力声明了 7 个 label：base、maven、mavenjdk11、go、go16、node.js 和 python。您可以指定具体的 Agent 标签来使用对应的 podTemplate。

如果内置 podTemplate 中的镜像不满足您的需求，可以通过以下方式替换容器镜像或者添加容器镜像。

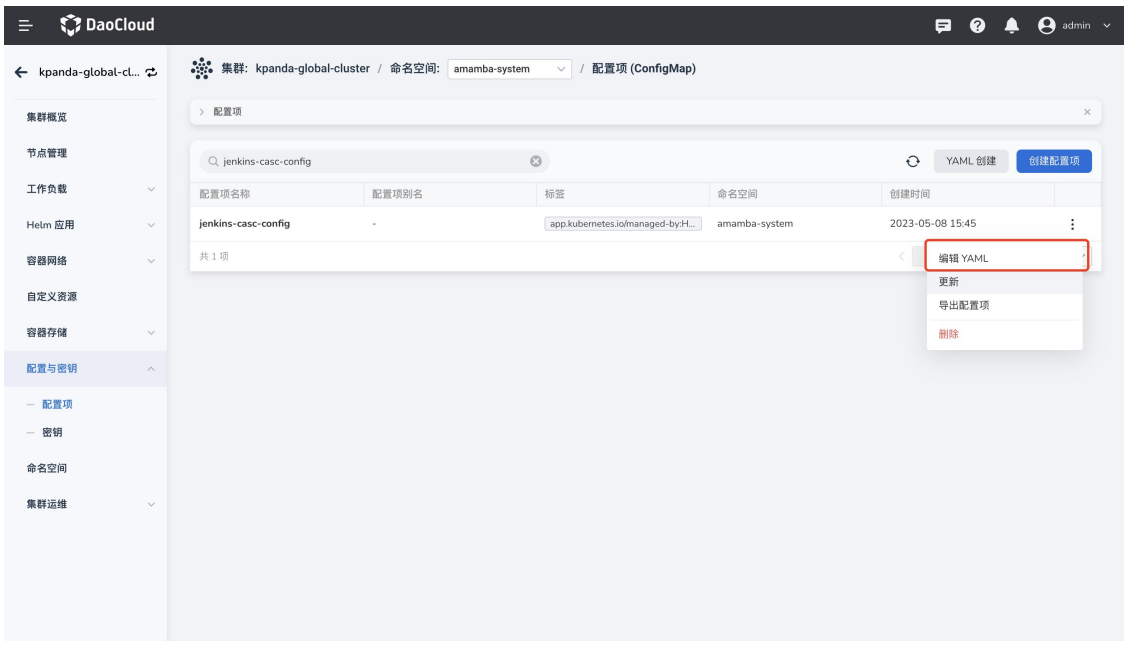
1. 前往容器管理模块，找到 Jenkins 组件所在的集群，点击集群名称。



faq-ci2

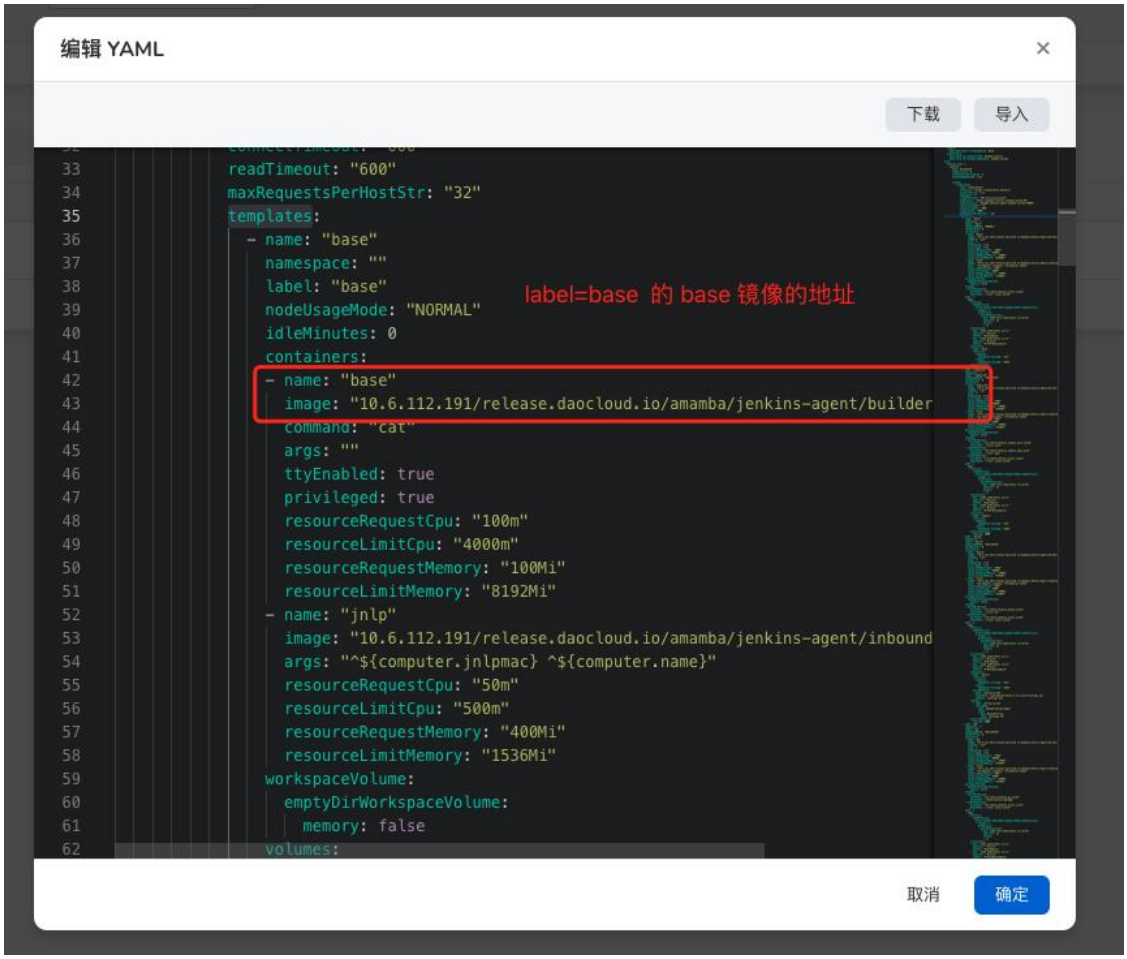
2. 在左侧导航栏依次点击配置与密钥->配置项。

3. 搜索 jenkins-casc-config ，在操作列点击编辑 YAML 。



faq-ci3

4. 在 data –> jenkins.yaml –> jenkins.clouds.kubernetes.templates 字段下选择需要更改的 podTemplate 的镜像。



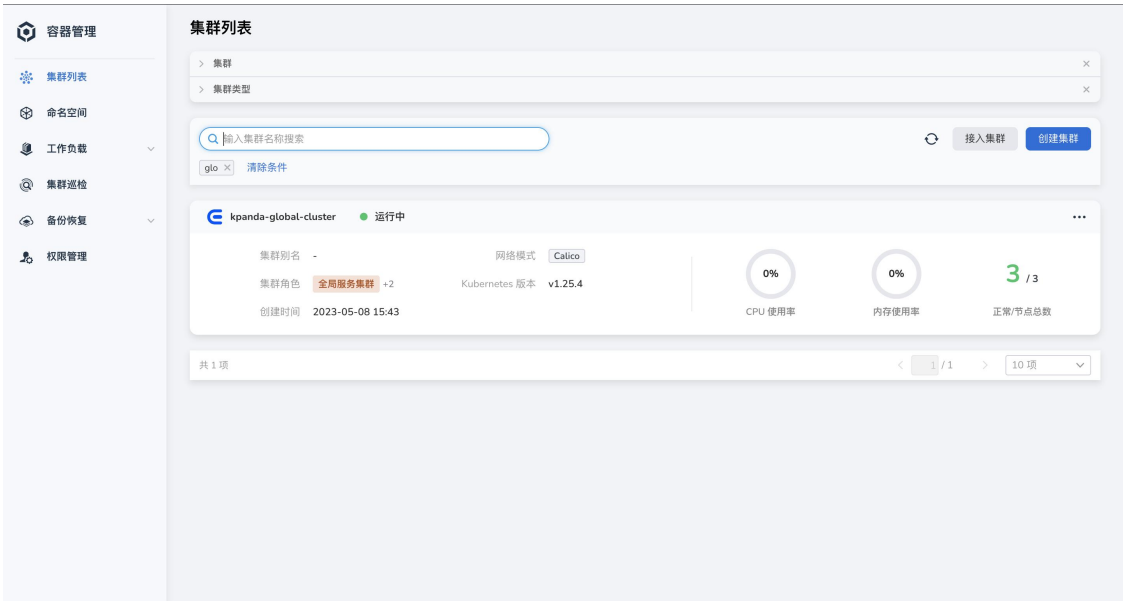
faq-ci4

5. 更新完成后，前往工作负载重启 jenkins。

流水线构建环境为 maven 时，如何在 settings.xml 中修改依赖包来源？

当流水线构建环境为 maven 时，大多数客户需要修改 settings.xml 以更换依赖源。可参考以下步骤：

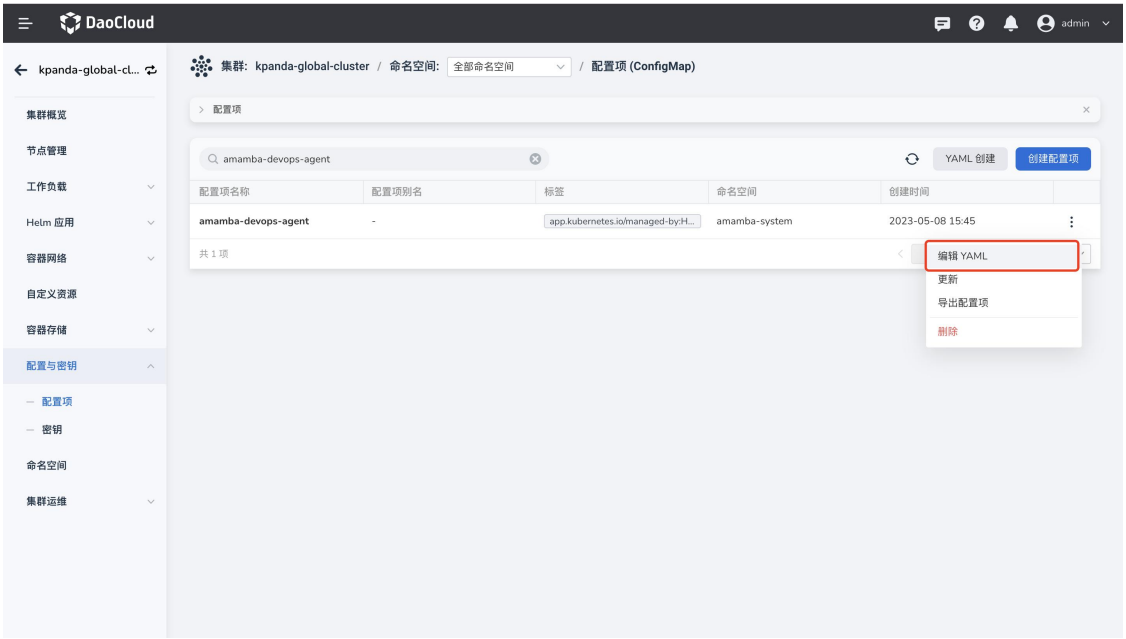
1. 前往容器管理模块，在集群列表 界面选择 Jenkins 组件所在的集群，点击集群名称。



faq-ci2

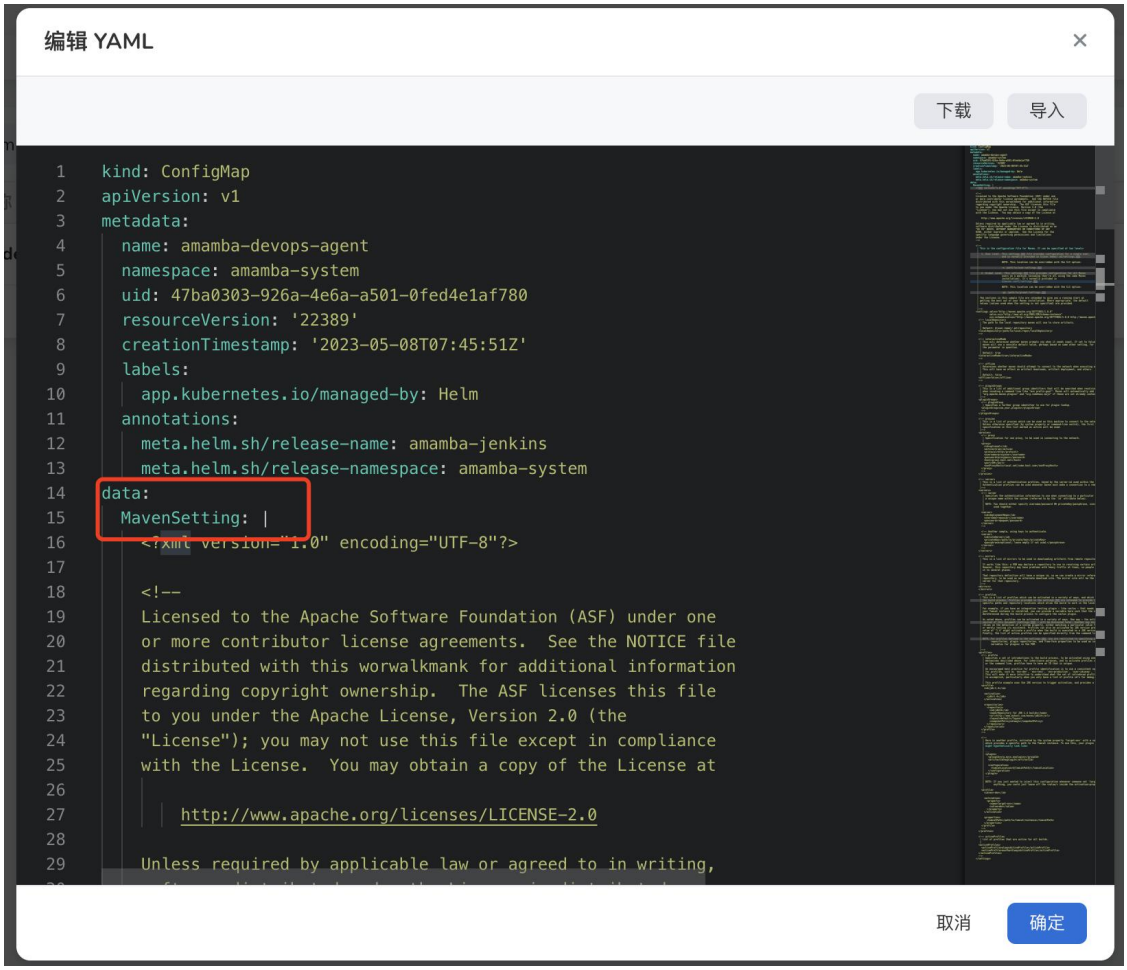
2. 在左侧导航栏依次点击配置与密钥->配置项。

3. 搜索 amamba-devops-agent ，在操作列点击 编辑 YAML 。



faq-ci5

4. 在 data 模块 下的 MavenSetting 按需修改。



faq-ci6

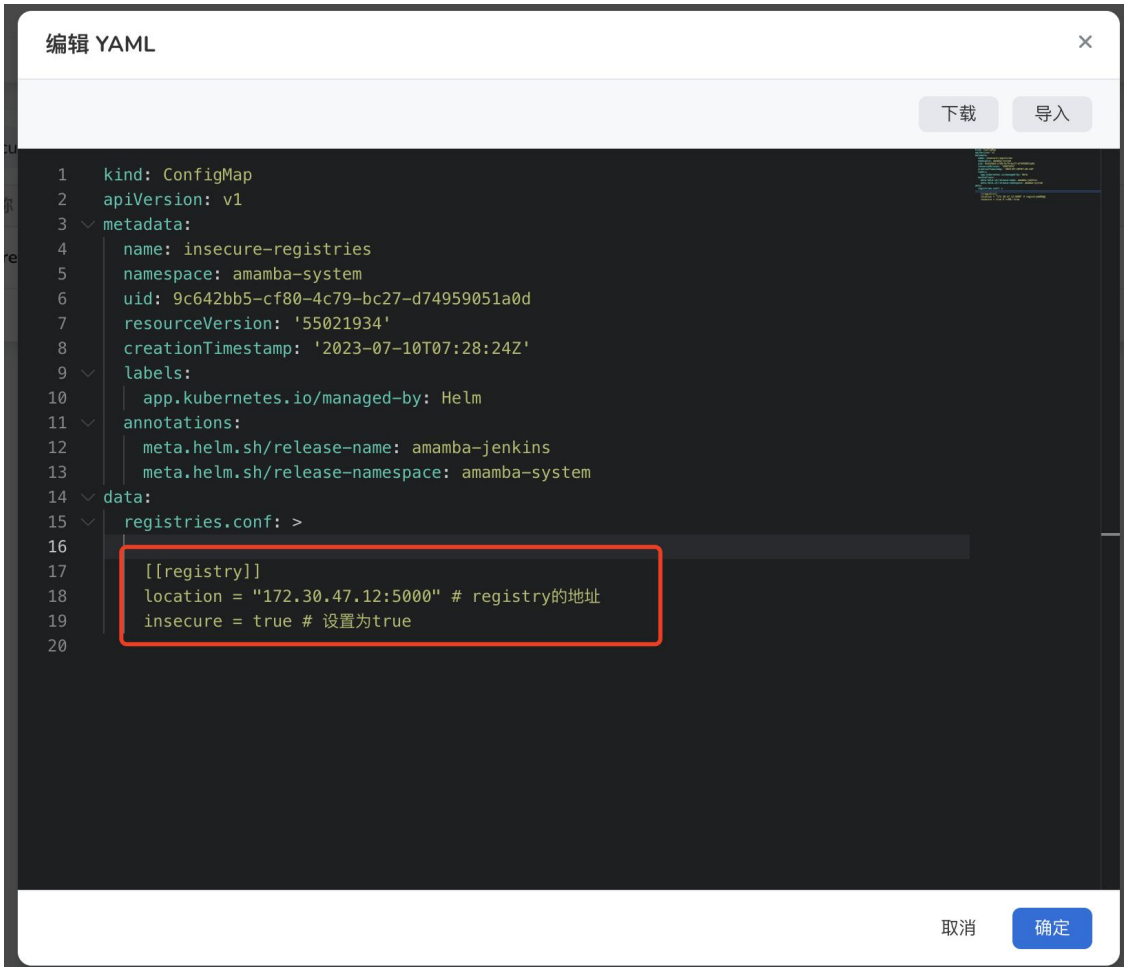
5. 更新完成后，需要前往工作负载重启 Jenkins。

通过 Jenkins 构建镜像时，容器无法访问私有镜像仓库

集群运行时为 podman

1. 在容器管理模块的 集群列表 界面找到 Jenkins 组件所在的集群，点击集群名称。
2. 在左侧导航栏依次点击配置与密钥->配置项。
3. 搜索 insecure-registries，在操作列点击编辑 YAML 。
4. 在 data 模块下的 registries.conf 下配置。

修改时注意格式缩进，并且每个 registry 需要一个单独的 [[registry]] 部分，如下图所示：



faq-ci1

基本概念

应用工作台提供了 DCE 5.0 应用部署的统一入口，支持云原生应用的整个生命周期管理。 降低了企业使用云原生应用门槛，提高了企业软件研发到应用交付的效率。

- 工作空间 Workspace

  通过工作空间协调全局管理和子模块的权限关系，解决资源聚合和映射层级关系。 并且一个工作空间对应一个项目，可以为每个工作空间分配不同的资源，指派不同的用户和用户组。

- 命名空间 Namespace

  命名空间在平台上是工作空间下相互隔离的更小的资源空间，也是用户实现作业生产的工作区间。 一个工作空间下可以创建多个命名空间，可占用的资源配额总和不能超过工作空间配额。 命名空间更细粒度的划分了资源配额的同时，还限制了命名空间下容器的大小（CPU、内存），有效的提升了资源利用率。

- 流水线 Pipeline

  流水线提供可视化、可定制的自动交付流水线，帮助企业缩短交付周期，提升交付效率。目前流水线是基于 Jenkins 实现。

- 凭证 Credential

  流水线与第三方应用程序进行交互时，需要用户配置 Jenkins 凭证，流水线就可以与第三方应用程序交互。

- GitOps

GitOps 的核心观点是使用包含当前期望的(生产环境基础设施的)声明式描述的 Glt 仓库，并通过自动化流程来确保生产环境与仓库中的期望状态保持一致。 如果您想要部署一个新的应用或更新一个现有的应用，只需要更新相应的仓库即可(自动化流程会处理后续的事情)。这就像在生产中使用巡航控制来管理应用程序一样。

- 灰度发布

  灰度发布是可以帮助用户渐进式更新应用的工具。它实现了多版本共存，发布暂停，流量百分比切换等功能，极大解放灰度发布过程中的手动操作，全自动化实现线上灰度流量切换。

- 工具链集成

  集成 DevOps 工具链，允许团队将已了解和已在使用的现有工具引入到应用工作台之中。从而避免登录到多个平台之中以及应对在不同工具间没有统一视角的难题。

GitOps 相关问题

本页列出使用应用工作台 GitOps 时可能遇到的一些问题并给出相应的解决方案。

在 GitOps 模块下添加 GitHub 仓库时报错

由于 GitHub 移除了对 username/password 的支持，所以通过 HTTP 方式导入 **GitHub** 仓库时会导入失败，可能出现如下报错信息：

```
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/DaoCloud/dce-installer.git/'
```

解决方案：

使用 SSH 方式导入 **GitHub** 仓库。

在某个工作空间下在 GitOps 模块中添加仓库时提示仓库已经存在

目前一个仓库绑定只能到一个工作空间，不能绑定到不同工作空间。如果一个仓库已经绑定到了 A 工作空间，此时尝试在 B 工作空间下将其添加到 GitOps 中就会提示仓库已经存在。

解决方案：

先将该仓库与其当前绑定的工作空间解绑，然后再重新将其添加到目标的工作空间。

应用工作台权限说明

应用工作台支持三种用户角色：

- Workspace Admin
- Workspace Editor
- Workspace Viewer

每种角色具有不同的权限，具体说明如下。

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
应用	查看应用列表			
	查看详情（跳转到容器管理）			
	查看应用日志（跳转到可观测）			
	查看应用监控（跳转到可观测）			
	查看 RabbitMQ 详情 – 基本信息			
	查看服务网格（跳转到服务网格）			
	查看微服务引擎（跳转到微服务引擎）			
	创建应用			
	编辑 YAML			
	更新副本数量			

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	更新容器镜像			
	编辑流水线			
	应用分组			
	删除			
命名空间	查看			
	创建			
	编辑标签			
	编辑资源配额			
	删除			
流水线	查看流水线			
	查看运行记录			
	创建			
	运行			
	删除			
	复制			
	编辑			
	取消运行			
凭证	查看			
	创建			
	编辑			
	删除			

菜单对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
持续部署	查看			
	创建			
	同步			
	编辑			
	删除			
代码仓库	查看			
	导入			
	删除			
灰度发布	查看			
	创建			
	发布			
	继续发布			
	终止发布			
	更新			
	回滚			
	删除			

!!! note

有关角色和权限管理的完整介绍，请参考[角色和权限管理](../../ghippo/user-guide/access-control/role.md)。

什么是应用工作台？

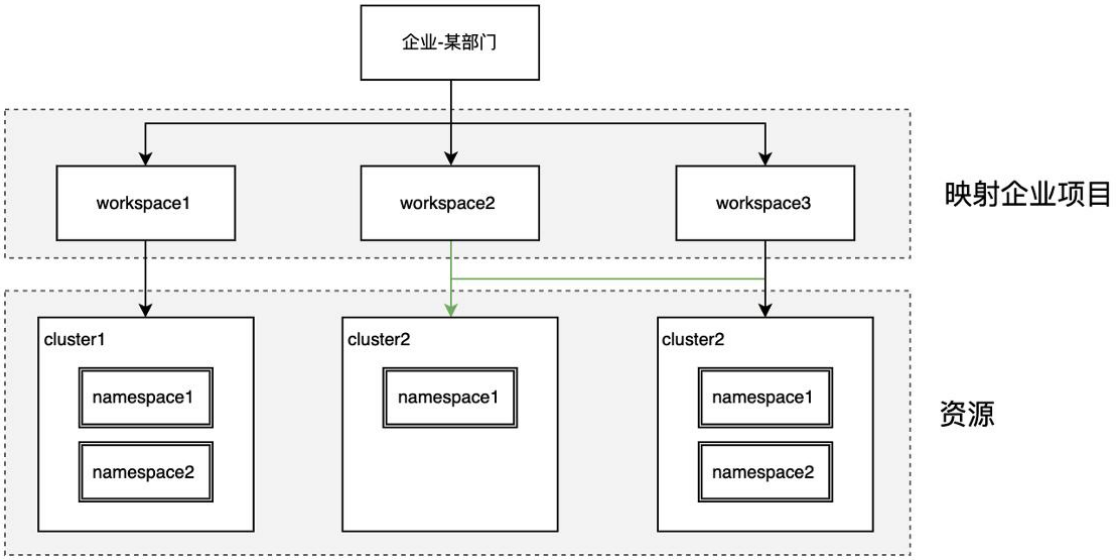
应用工作台是基于容器的 DevOps 云原生应用平台，提供了 DCE 5.0 应用创建的统一入口，通过界面化表单创建多种流水线、GitOps、金丝雀、蓝绿、AB 等渐进式发布策略、项目管理、工具链集成等多种功能。

应用工作台着重企业应用自动化交付和基础设施变更的过程，提供业务应用从“开发 -> 测试 -> 部署 -> 运维”的全生命周期管理，能有效帮助企业实现数字化转型，提升企业的 IT 交付能力和竞争力。

===“层级多租户资源管理”

以工作空间作为最小的租户单元，不仅支持单集群的独享资源的能力，还支持跨集群共享资源的能力：

- 支持弱绑定集群，从而获得跨集群、跨命名空间共享资源的能力
- 支持强绑定集群，从而获得独享资源的能力
- 工作空间管理成员可以在关联的集群中创建命名空间资源
- 自助式资源创建的模式，用户可以在工作空间内自助创建命名空间来进行资源隔离



多租户资源管理

===“以云原生应用为中心”

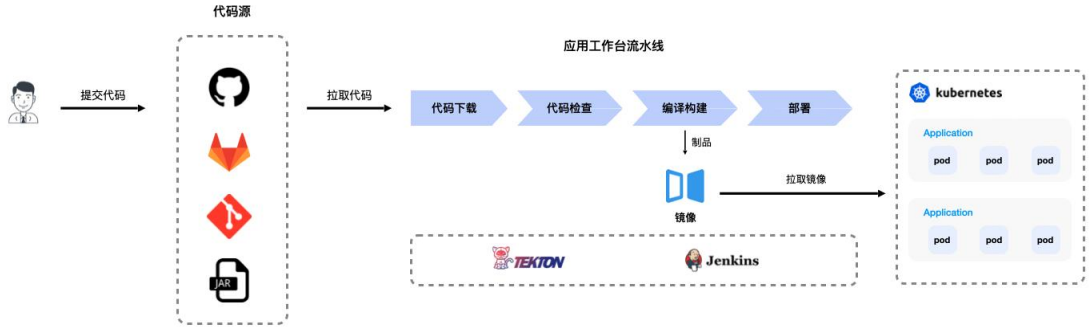
支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。 能够接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的微服务引擎、服务网格无缝集成。支持云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等等。



多形态云原生应用

===“高效的持续集成”

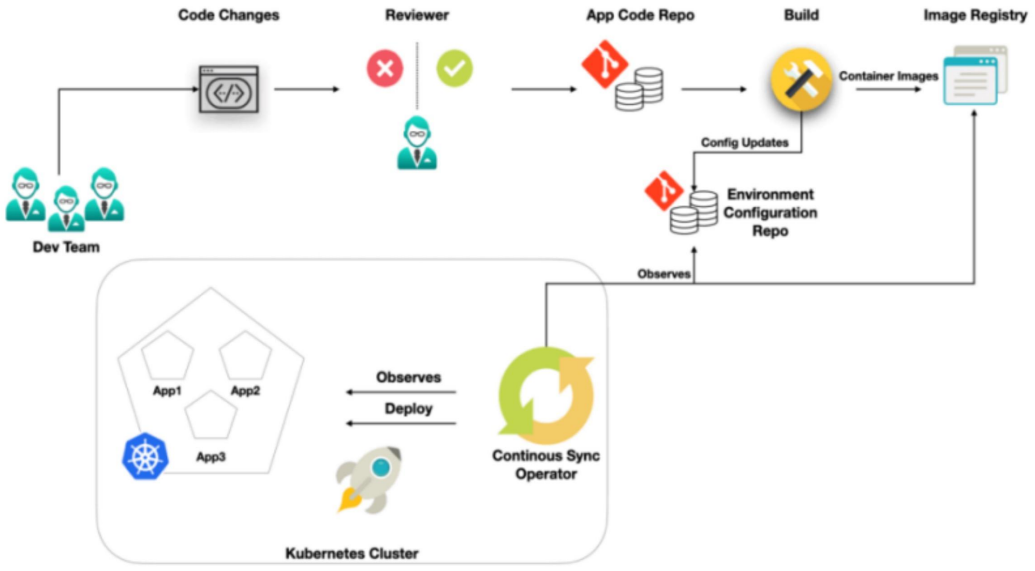
支持 Jenkins 和 Tekton 双流水线引擎系统。采用图形化方式编辑流水线，做到所见即所得的效果。可以用不同来源的代码构建应用。



持续集成

===“安全自动化渐进式交付”

应用工作台引入一种为云原生应用实现持续部署的理念 — GitOps，全面拥抱 GitOps，以集成渐进式交付组件 Argo Rollout，支持灰度发布，从而提高应用交付的稳定性和效率。



自动化渐进式交付

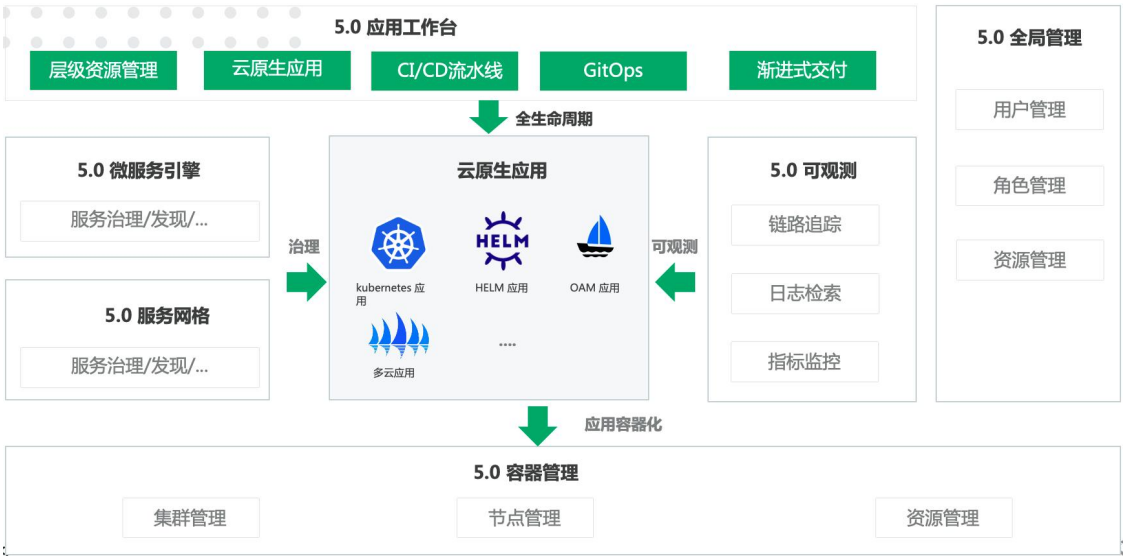
!!! info

渐进式交付是一种将应用程序的新版本逐步暴露给最初的一小部分用户，然后逐渐变得越来越大的子集的做法，以减轻负面影响（例如错误）的风险。

Argo-Rollout Kubernetes Progressive Delivery Controller，提供更强大的部署能力。包括灰度发布、蓝绿部署、更新测试（experimentation）、渐进式交付（progressive delivery）等特性。

在 DCE 5.0 中的地位

以容器管理为底座，借助全局管理实现层级资源管理，以 CI/CD 流水线和 GitOps 流程增删改查云原生应用，实现渐进式交付。



应用工作台在 DCE 5.0 中的地位

部署方法

建议通过【DCE 5.0 商业版】的安装包（../../install/commercial/start-install.md）安装 Workbench，因为这样可以一次性安装 DCE 5.0 的所有模块 ，无需担心组件不兼容的问题。

但是，如果你想单独安装或升级应用工作台模块，执行以下命令即可：

```
export VERSION=**** # 修改为实际部署的版本。
helm repo add amamba-release https://release.daocloud.io/chartrepo/amamba
```

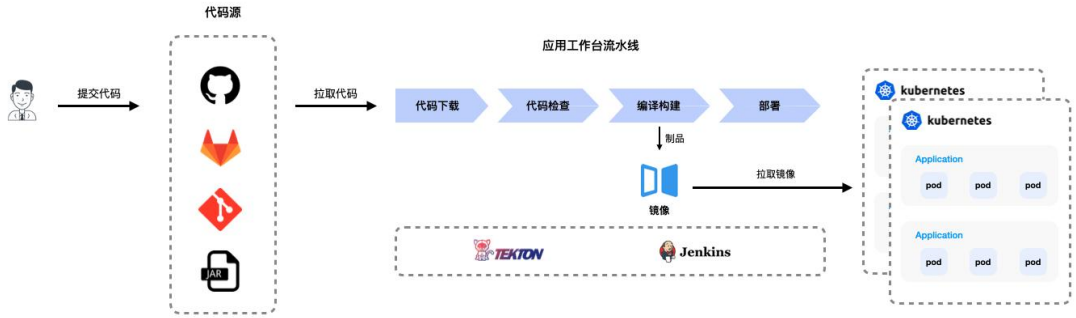
```
helm repo update amamba
helm upgrade --install --create-namespace --cleanup-on-fail amamba amamb
a-release/amamba -n amamba-system --version=${VERSION}
```

下载 DCE 5.0 安装 DCE 5.0

适用场景

- 持续交付流水线

  对于复杂的业务系统，从项目创建、编译、构建、自验、集成验证、类生产验证、上线的各个阶段都需要耗费大量的人力和时间，并且容易受到人为因素影响而出错。 持续集成和持续交付由于具有标准化和自动化特点，以代码变更为流动单元，基于发布流水线，拉通开发、测试、运维所有职能，持续、快速、高可靠地发布软件，可以很好的解决该问题。



image

- 基于流水线 + GitOps 实现云原生下的应用持续交付

  得益于 GitOps 理念，因此采用开源软件 Argo CD 来实践 Kubernetes 的应用发布。 用户仅需将 Kubernetes 的 YAML 文件提交至代码仓库中，GitOps 会自动感知到 YAML 文件的变化，配合代码仓库的合并请求功能， 将 YAML 文件的变更自动推送至集群中，整个过程无需学习 Kubernetes 的发布命令，也无需直接操作集群。

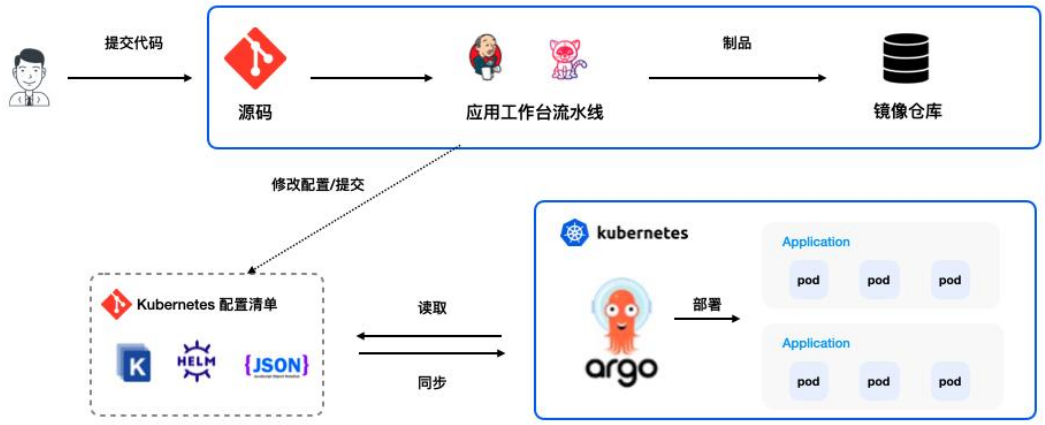

提交代码
源码
应用工作台流水线
制品
镜像仓库
修改配置/提交
Kubernetes 配置清单
读取
同步
kubernetes
argo
部署
Application
pod
pod
pod
Application
pod
pod
pod

image2

功能总览

应用工作台提供了以下功能。

- 应用管理
  - 支持云原生场景下的“多形态”的云原生应用，包括 Kubernetes 原生应用、Helm 应用、OAM 应用等。
  - 提供云原生应用的全生命周期管理，例如应用的扩缩容、日志、监控查看、更新应用等。
  - 支持接入 SpingCloud、Dubbo、ServiceMesh 框架的微服务应用，实现微服务的治理，与 DCE 5.0 的微服务引擎、服务网格无缝集成。
- 流水线编排

  流水线是一个自定义的 CI/CD 模式。
  - 支持四种创建流水线的模式：自定义创建、基于 jenkinsfile 创建、基于模板创建、创建多分支流水线。
  - 支持流水线图形化编辑。
  - 可通过 Git 源码、Jar 包、Helm chart、容器镜像的方式来构建应用。
- 凭证管理

  为流水线中的代码库、镜像仓库提供不同类型的凭证管理功能。
- 持续部署

  引入 GitOps 理念实现应用持续部署，用于把控代码构建之后的应用发布与部署交付流程。

- 基于 Argo CD，以自动化方式，频繁且持续性地将企业的应用部署到生产环境。
- 提供 Argo CD Application 的创建、同步、删除管理。

- 仓库管理

  支持导入 Git 的代码仓库，导入后您可以在持续部署中使用该仓库进行应用的持续部署。

- 灰度发布

  灰度发布可以保证整体系统的稳定，在初始灰度时就可以发现、调整问题，减少问题的影响范围。

  - 支持金丝雀发布、蓝绿部署、A/B Testing 高级发布策略。
  - 金丝雀发布支持自动化渐进式的发布流程。
  - 支持通过监控指标分析来快速回滚应用。

OAM 概念介绍

OAM 应用功能基于开源软件 KubeVela ，通过开放应用模型（OAM）作为应用交付的顶层抽象，主要是对 Kubernetes 的资源的抽象与整合。

一个 OAM 应用由一个或多个组件和各项运维动作组成，从而实现在混合环境中标准化和高效率的应用交付。

组件

组件（Components）可以定义一个制品或云服务的交付和管理形式，一个应用中可以包括多个组件。最佳的实践方案是一个应用包括一个主组件（核心业务）和附属组件（强依赖或独享的中间件，运维组件等）。有关组件类型说明，可参考 Component Definition。

目前应用工作台基于 kubevela 开源组件内置了 **Cron-Task、Daemon、K8s-Objects、Task、Webservice** 五种组件。有关各种组件的具体参数介绍，可参考：内置组件列表。

运维特征

运维特征（Traits）是可以随时绑定给待部署组件的、模块化、可拔插的运维能力，比如：副本数调整（手动、自动）、数据持久化、设置网关策略、自动设置 DNS 解析等。用户可以从社区获取成熟的能力，也可以自行定义。

目前应用工作台基于 kubevela 开源组件内置了 **Affinity、Annotations、Command、Container-Image、Cpuscaler** 等多种运维特征。有关运维特征的具体参数介绍，可参考：内置运维特征列表。

自定义功能

Kubevela 可以根据您的需求轻松实现原地定制和扩展。

自定义组件

组件定义（ComponentDefinition）的设计目标是，允许平台管理员将任何类型的可部署制品封装成待交付的“组件”。只要定义好之后，这种类型的组件就可以被用户在部署计划（Application）中引用、实例化并交付出去。

有关自定义组件的具体操作，可参考官方文档：自定义组件入门。

自定义运维特征

运维特征定义（TraitDefinition）为组件提供了一系列可被按需绑定的运维动作，这些运维动作通常都是由平台管理员提供的运维能力，为这个组件提供一系列的运维操作和策略，比如添加一个负载均衡策略、路由策略、或者执行弹性扩缩容、灰度发布策略等等。

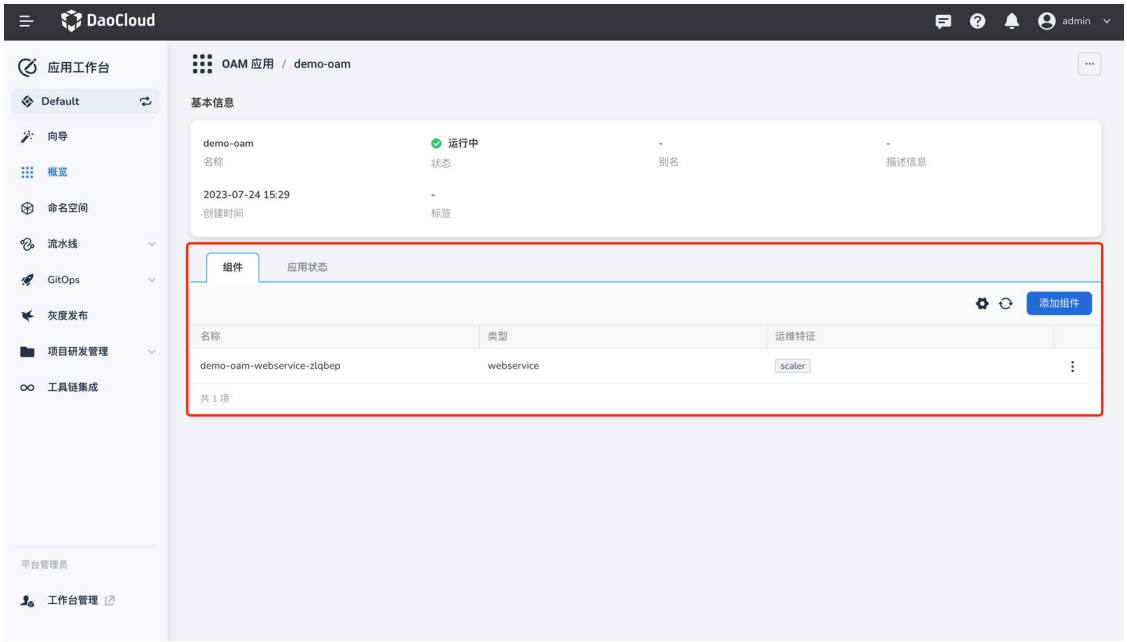
有关自定义运维特征的具体操作，可参考官方文档：自定义运维特征。

OAM 应用管理

创建好 OAM 应用之后，可以查看应用详情并更新应用的组件或运维特征。

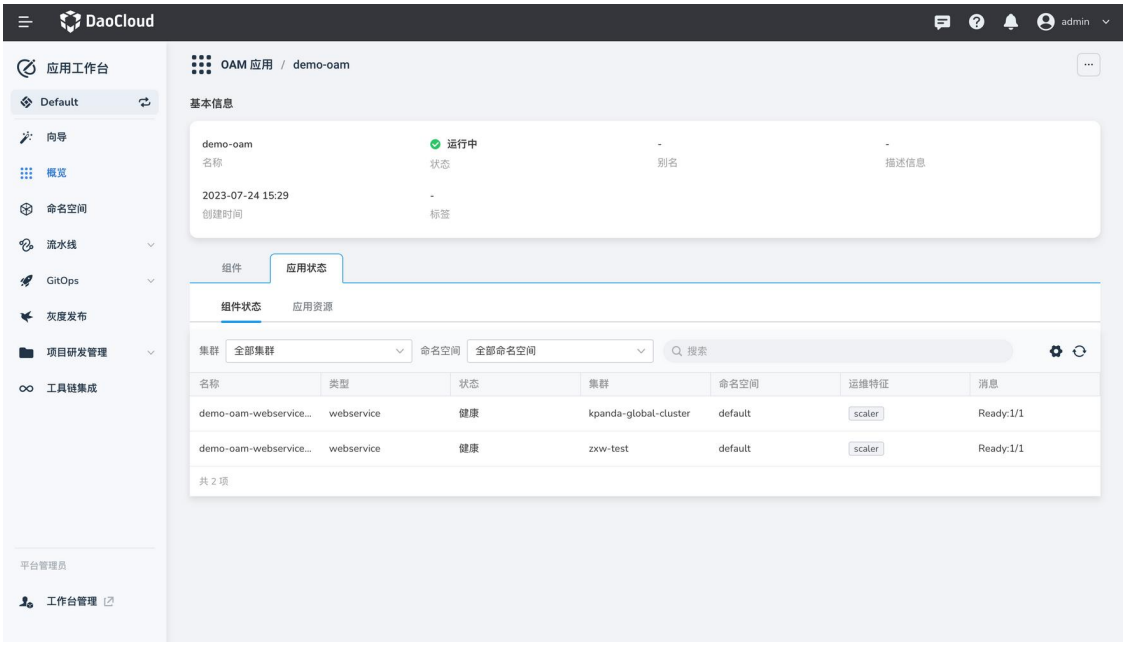
查看应用详情

1. 在应用工作台–>概览页面，点击 OAM 应用页签，查看 OAM 应用列表
2. 点击应用名称可以查看应用详情，包括应用名称、状态、别名、描述信息、创建时间等字段。
3. 在应用详情页面点击组件页签，可以查看当前应用定义的组件数量、类型、运维特征。



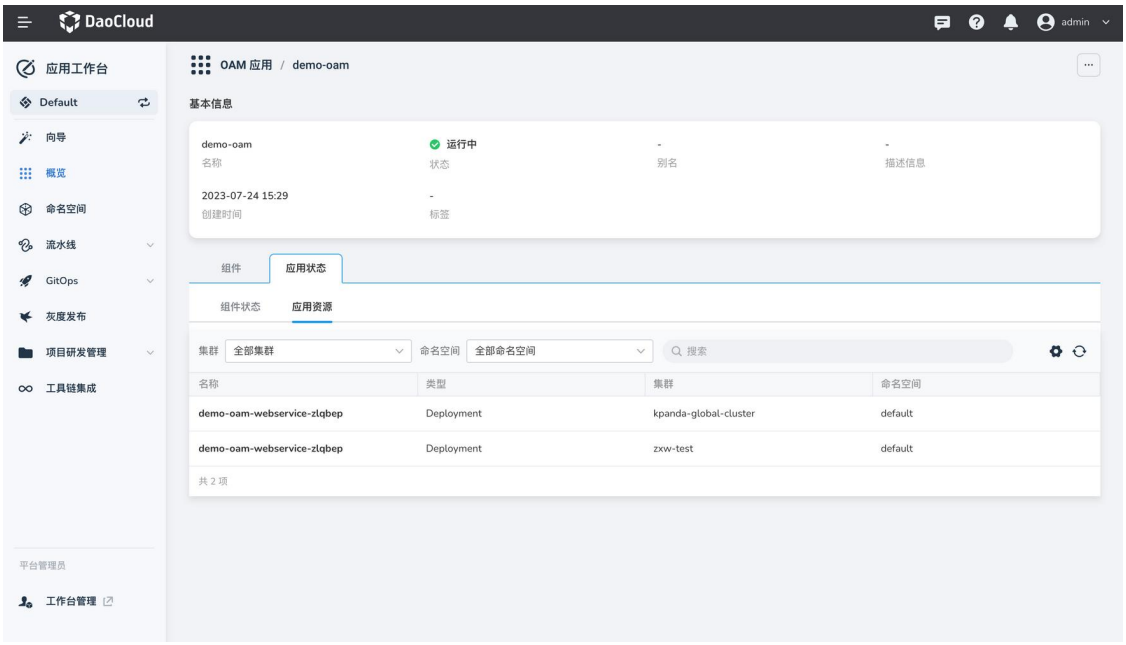
component

4. 在应用详情页面点击应用状态页签，可以查看组件状态和应用资源。
    - 组件状态：展示当前应用下的每个组件的健康状态。存在不健康的组件时需要及时排查运维。



status

- 应用资源：当前应用下部署的的 Kubernetes 资源的信息。

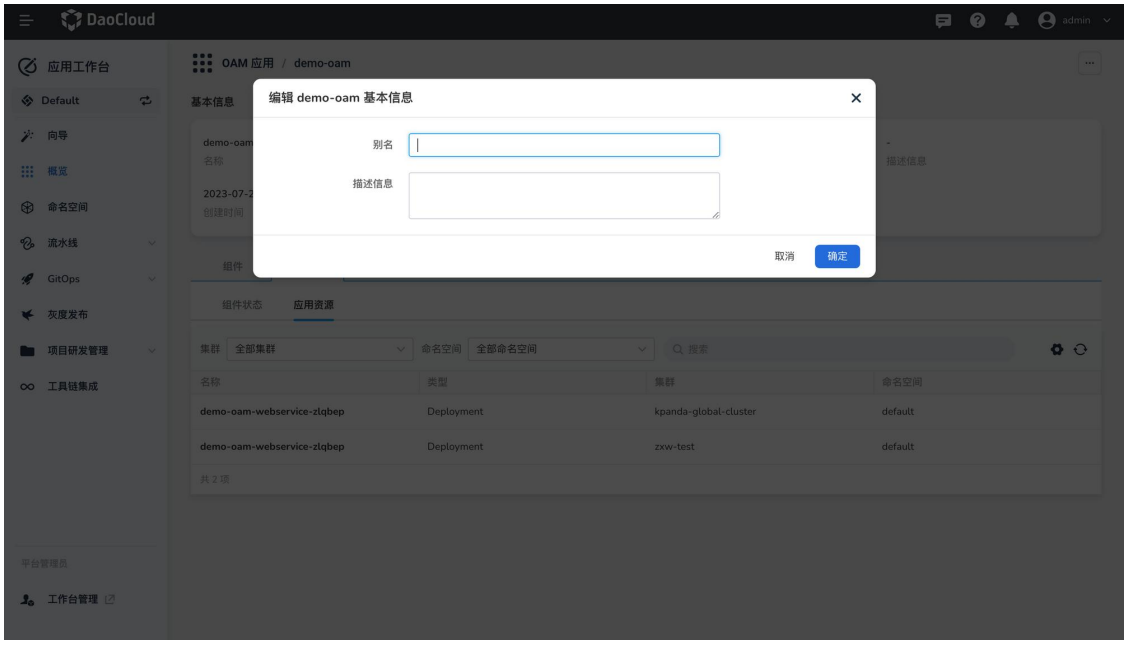


resources

编辑 OAM 应用基本信息

1. 点击 OAM 应用名称，然后在页面右上角点击 选择编辑基本信息。

2. 根据需要设置别名，或补充描述信息描述信息。

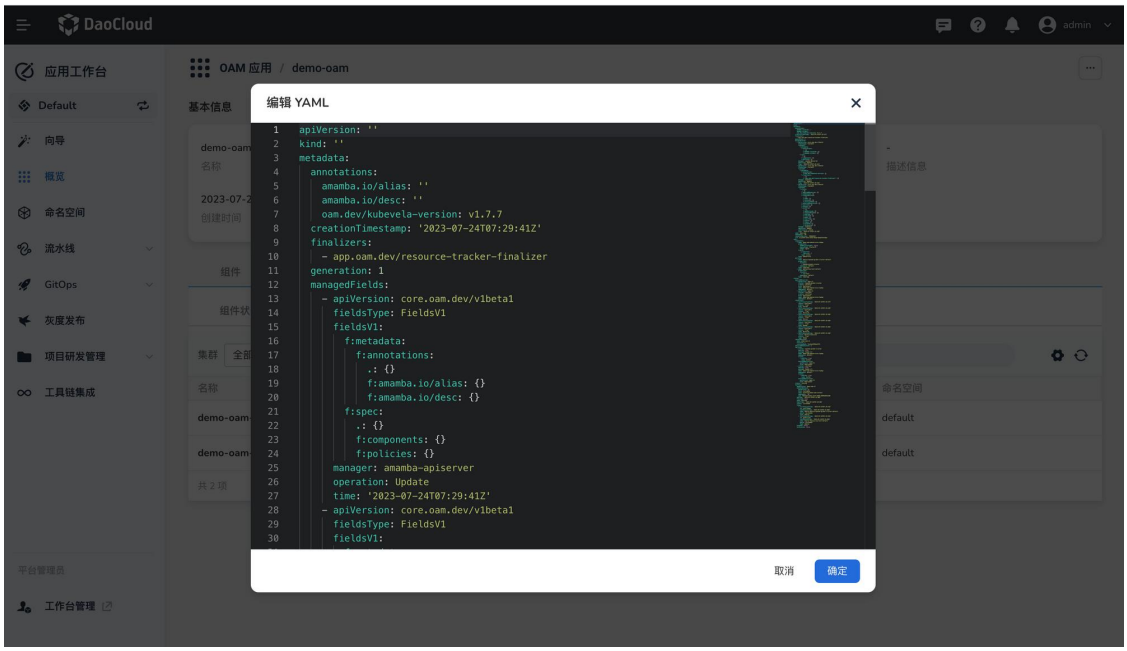


edit

编辑 OAM 应用 YAML 文件

1. 点击 OAM 应用名称，然后在页面右上角点击 选择编辑 YAML。

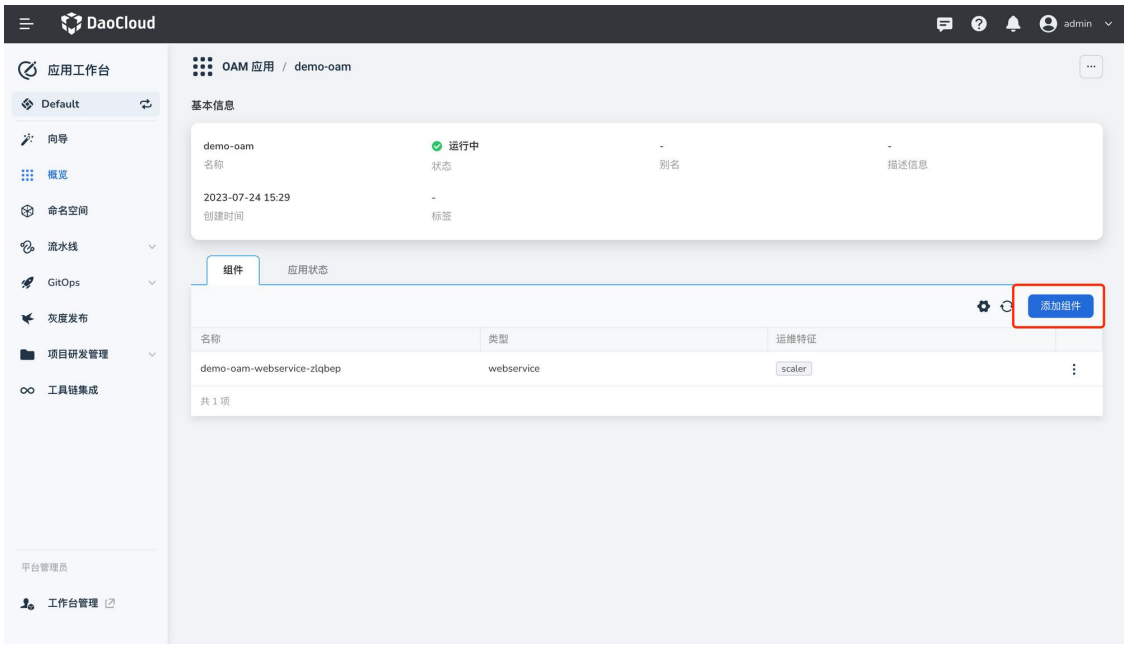
2. 根据需要编辑 OAM 应用的 YAML 文件。



edit

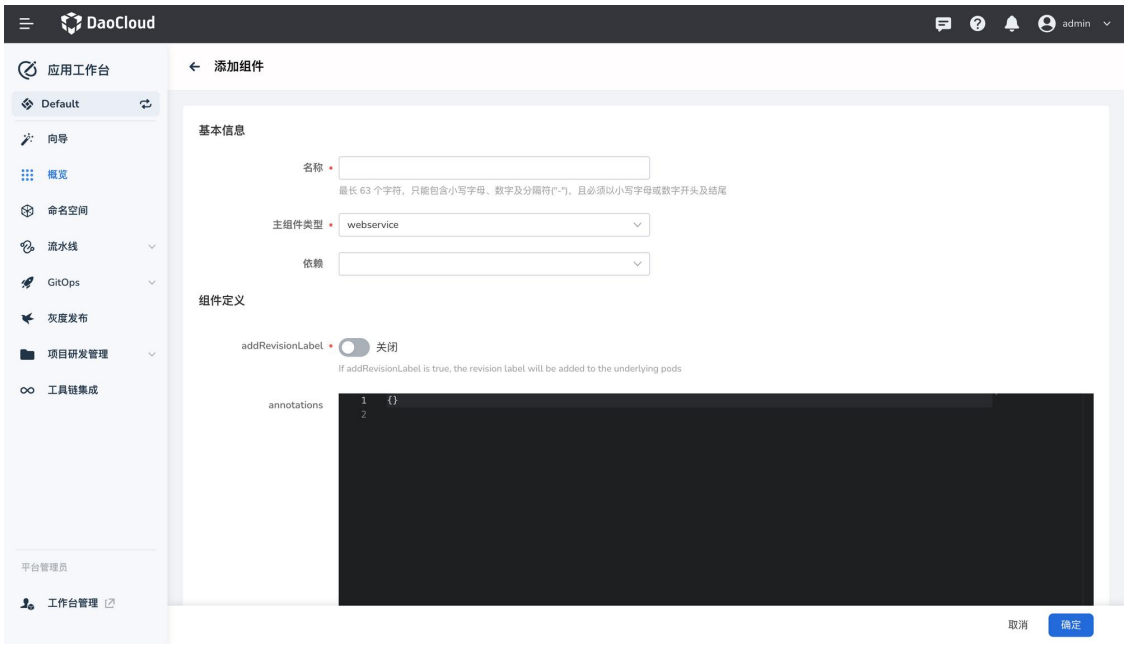
添加组件

1. 点击 OAM 应用名称，点击组件页签，然后在右侧点击添加组件。



add-component

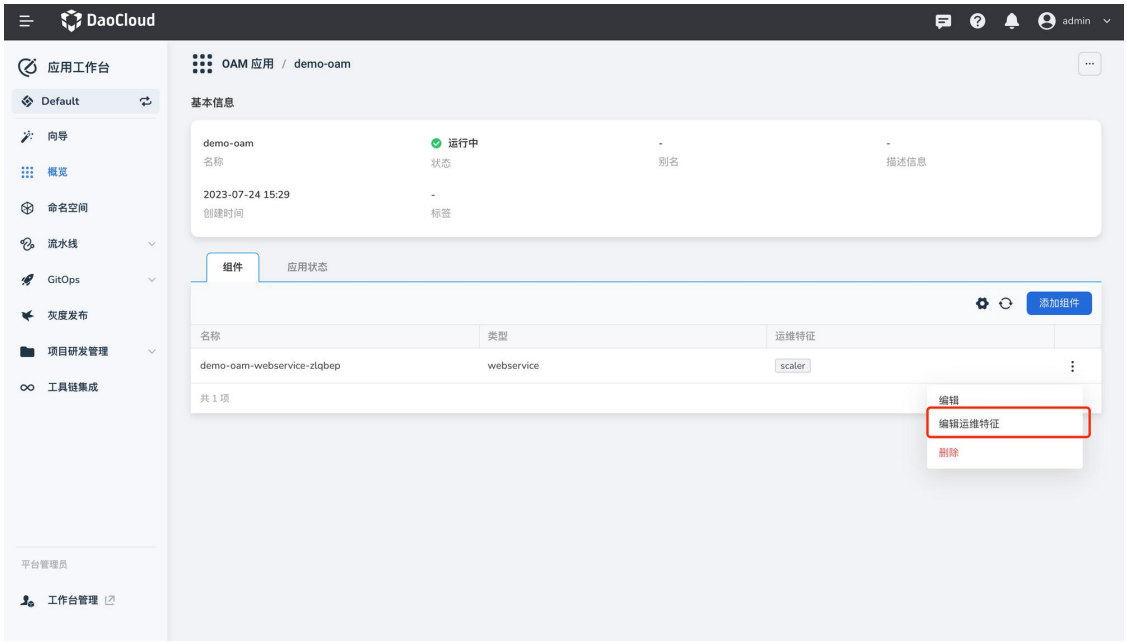
2. 选择需要添加的组件类型，参考内置组件列表填写对应的组件参数。



add-component

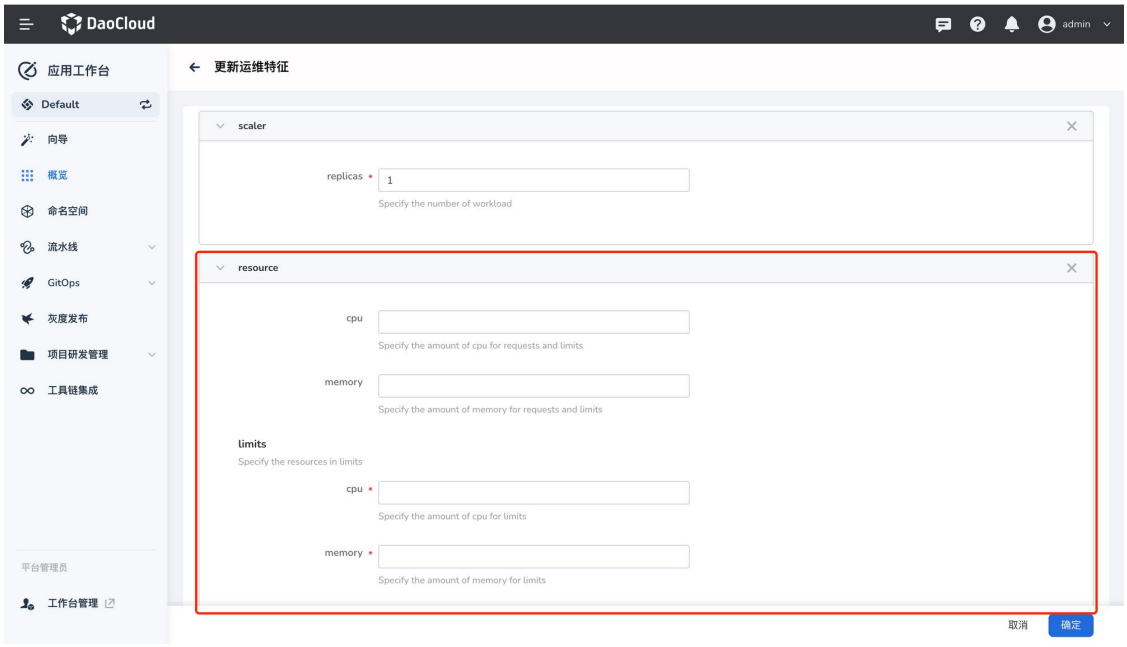
添加/更新运维特征

1. 点击 OAM 应用名称，点击组件页签，在组件右侧点击 选择编辑运维特征。



add-component

2. 参考内置运维特征列表更新运维特征属性。



add-component

创建 OAM 应用

OAM 应用功能基于开源软件 KubeVela ，通过开放应用模型（OAM）作为应用交付的顶层抽象，主要是对 Kubernetes 的资源的抽象与整合。一个 OAM 应用由一个或多个组件和各项运维动作组成，从而实现在混合环境中标准化和高效率的应用交付。

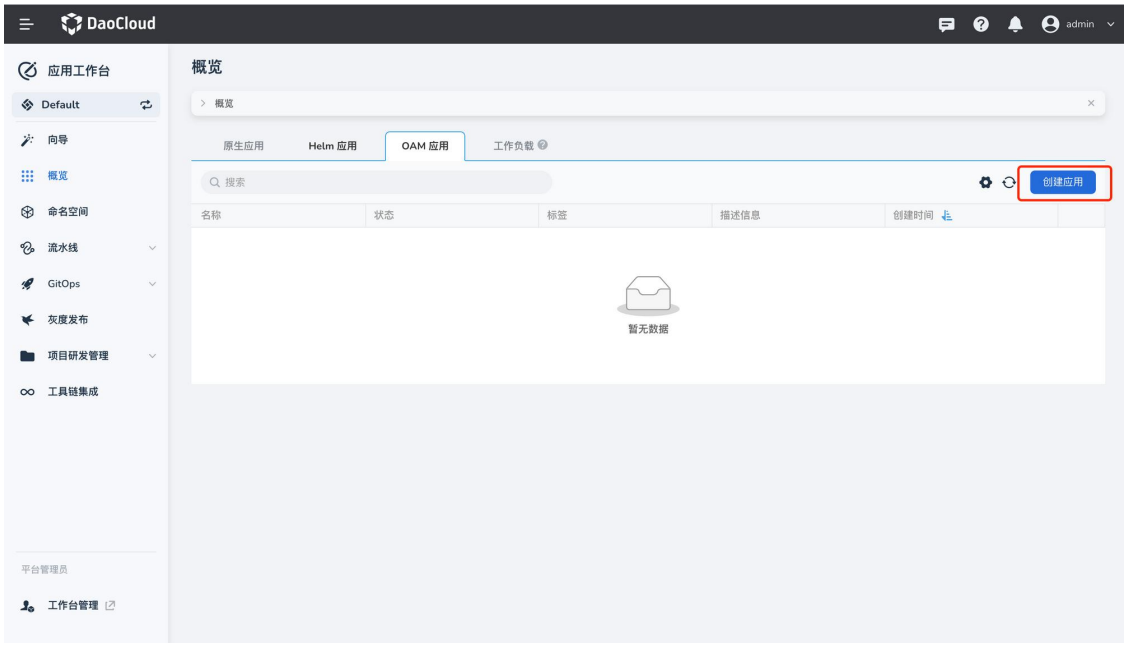
有关 OAM 的概念介绍可参考 OAM 概念介绍或参考 KubeVela 官方文档

前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 Workspace Editor 或更高权限。

操作步骤

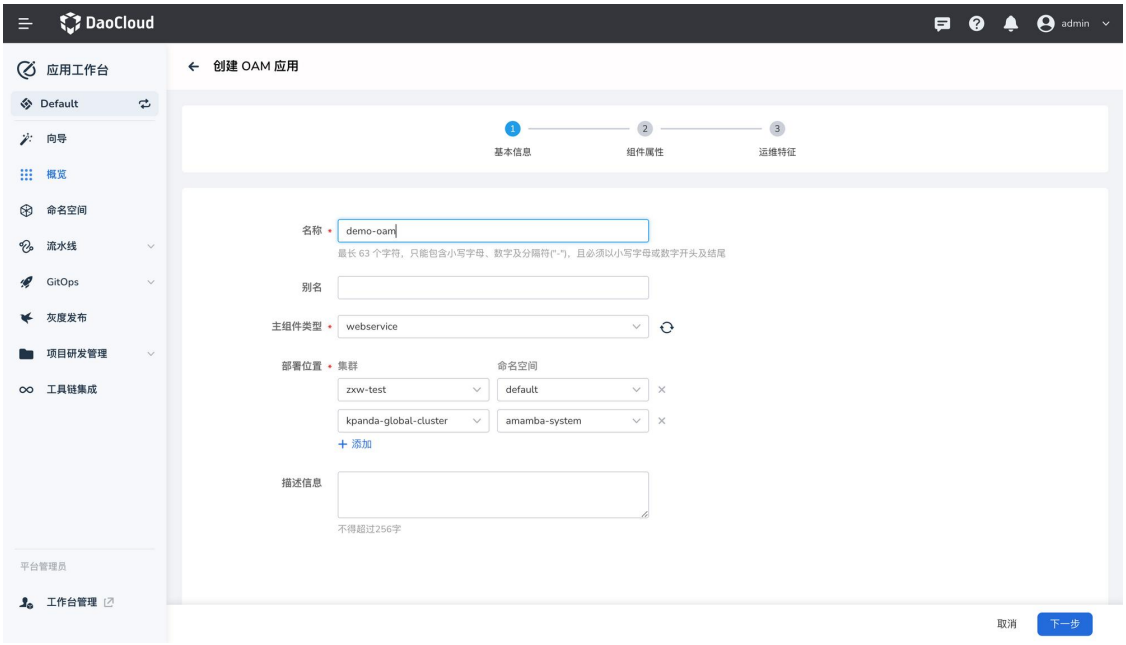
1. 在应用工作台–>概览页面中，OAM 应用页签下点击创建应用。



step1

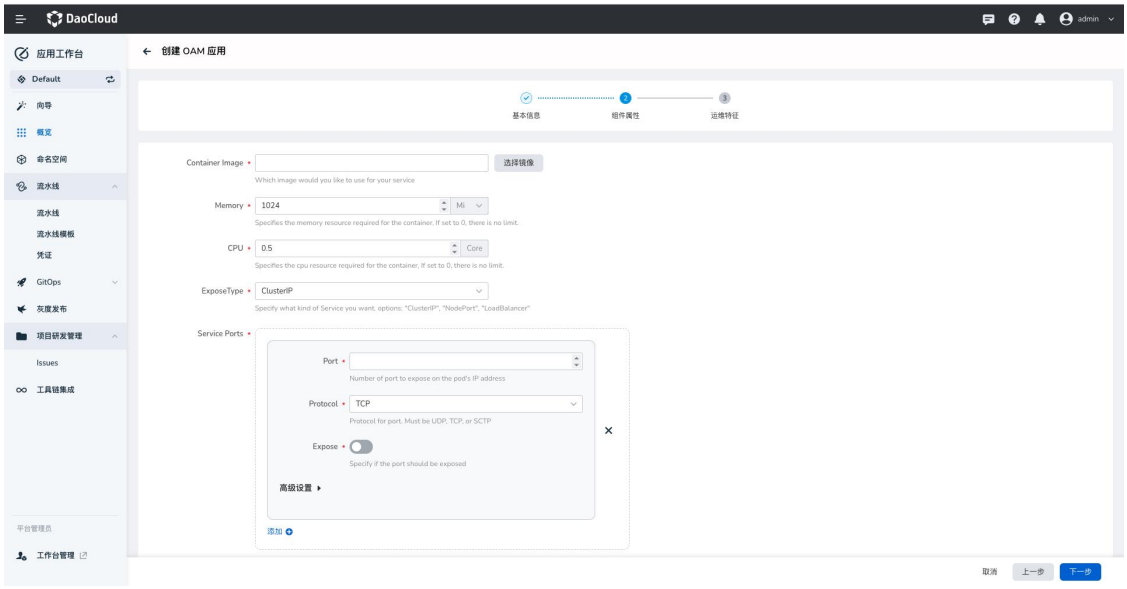
2. 参考以下说明填写基本信息，然后点击下一步：

- 名称/别名：填写 OAM 应用的名称/别名。
- 主组件类型：不同组件需要配置的参数有所不同。有关各种组件类型的详细介绍，可参考内置组件列表。
    - CronTask：定义一个周期性运行代码或者脚本的任务。
    - Task：定义一个只执行一次代码或者脚本的任务。
    - Daemon：定义一个在 Kubernetes 每个节点上都运行的服务。
    - K8s-Object：列表中的元素为完整的 Kubernetes 资源结构体。
    - Webservice：定义一个长期运行的、可伸缩的容器化服务，并且暴露一个服务端点用来接受来自客户的外部流量。
- 部署位置：选择将应用部署到哪个集群下的哪个命名空间，支持多个集群环境。



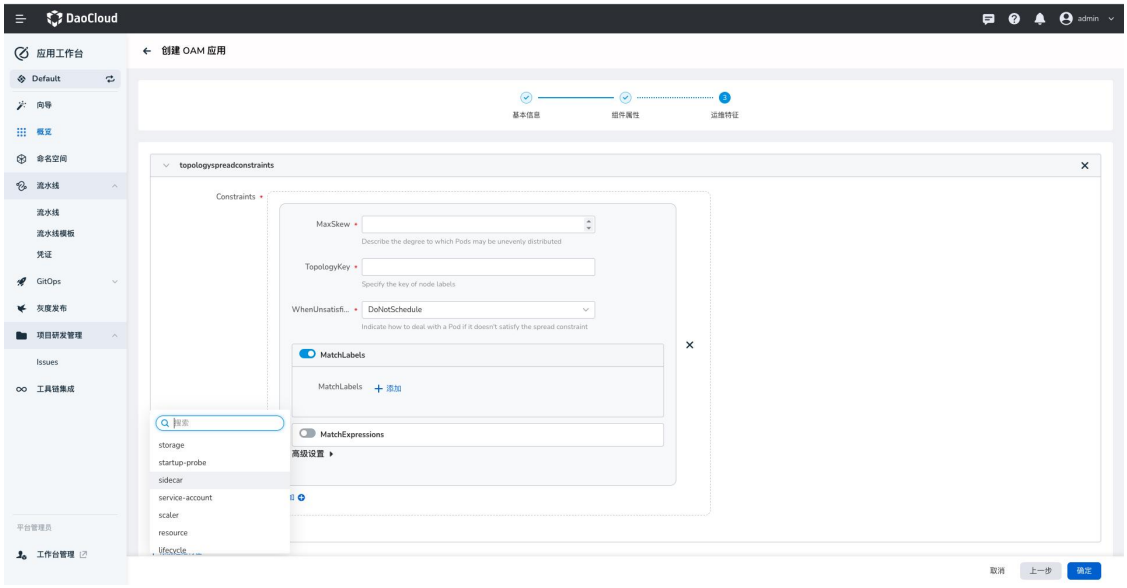
step2

3. 参考内置组件列表为所选组件配置参数。



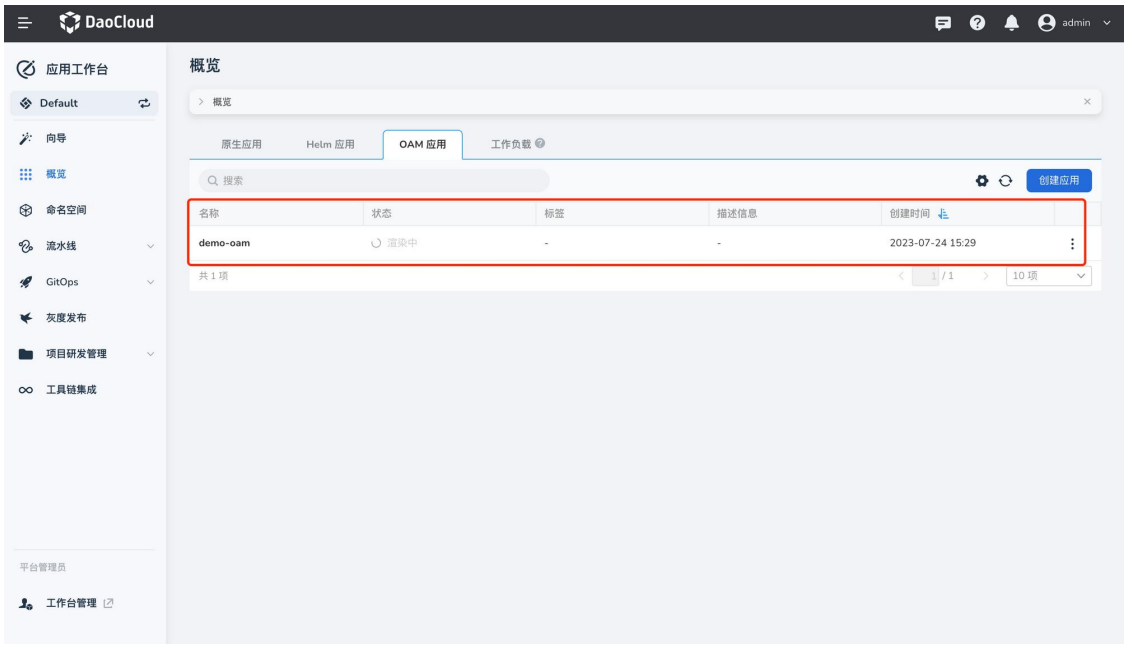
step3

4. 参考内置运维特征列表为所选组件配置运维特征属性。



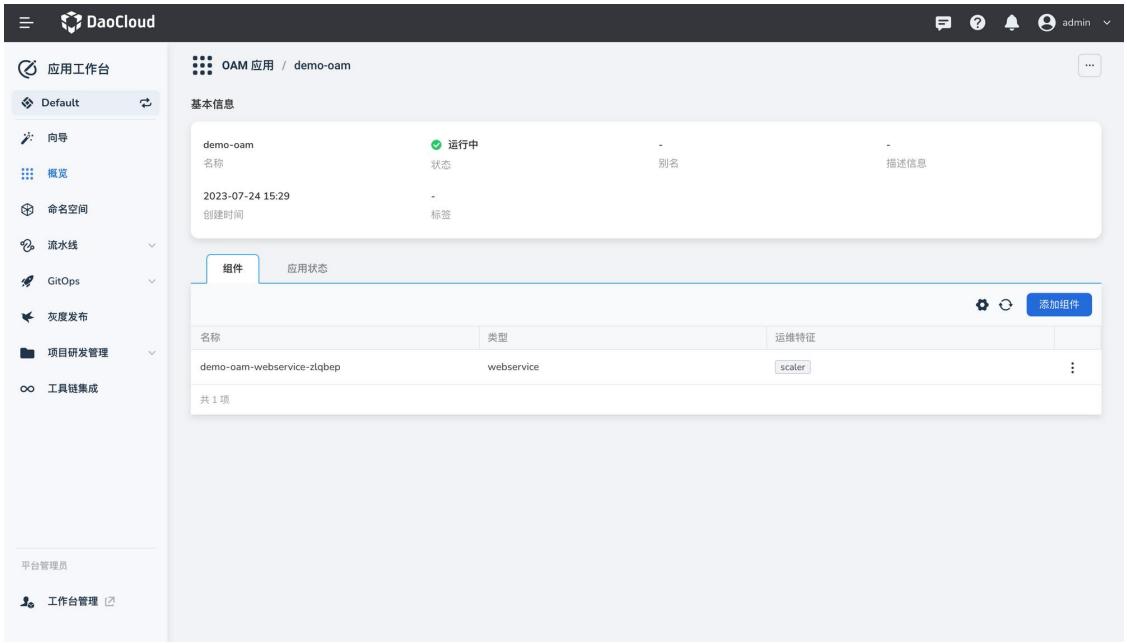
step4

5. 点击确定完成创建，创建成功后返回 OAM 应用列表页面。



step5

6. 点击应用名称，进去 OAM 应用详情页面。支持查看 OAM 应用基本信息、包含的组件列表、部署的应用资源状态，以及支持添加组件、添加运维特征等操作。



step6

查看缺陷

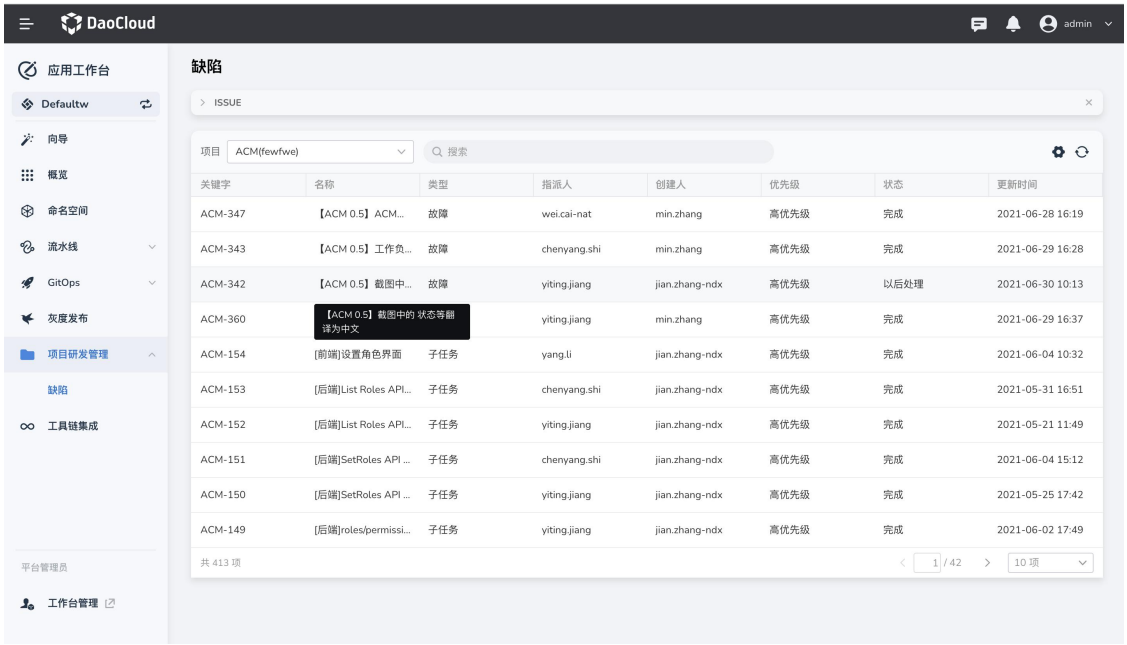
查看 Jira 实例中项目内的缺陷信息。

前提条件

- 在当前平台已经成功集成一个 Jira 实例，参考集成工具链。
- 将集成的 Jira 实例下的项目分配到当前工作空间内，参考管理工具链实例

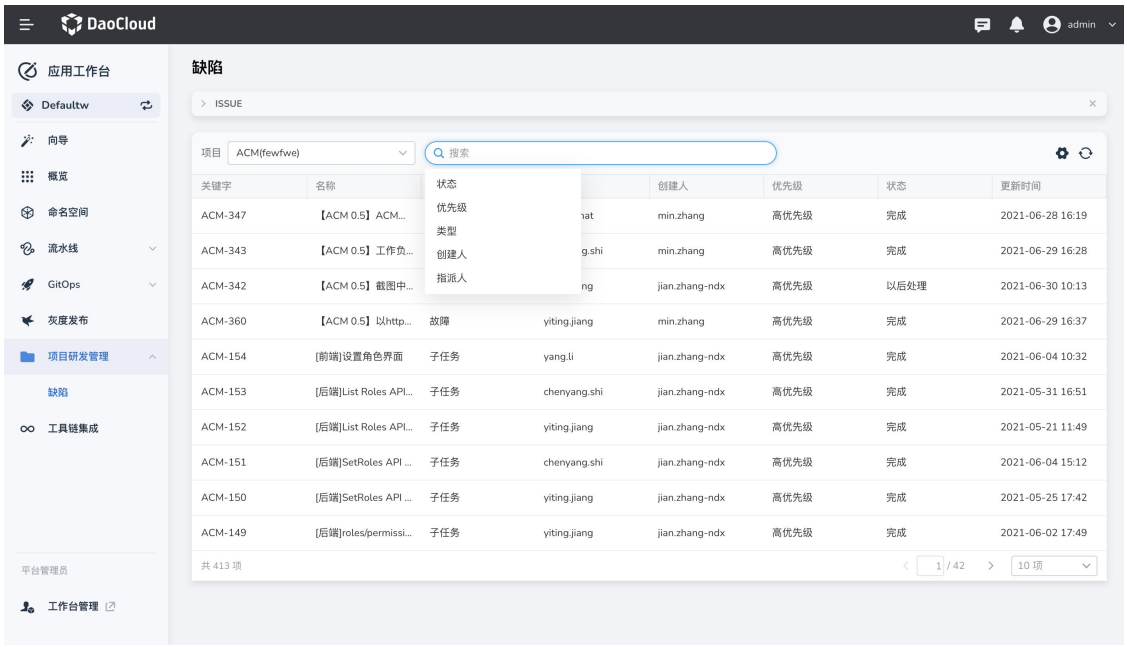
操作步骤

1. 在应用工作台选择一个工作空间，进入项目研发管理 -> 缺陷 页面。
2. 在缺陷页面查看 Jira 项目下的 Issues 信息，一个工作空间可以绑定多个已集成的 Jira 实例下的项目。



issue01

3. 支持通过状态、优先级、类型、创建人、指派人来进行筛选 Issues。


DaoCloud
admin
应用工作台
Defaultw
向导
概览
命名空间
流水线
GitOps
灰度发布
项目研发管理
缺陷
工具链集成
平台管理员
工作台管理
缺陷
ISSUE
项目
ACM(fewfwe)
搜索
状态
优先级
类型
创建人
指派人
关键字
名称
创建人
优先级
状态
更新时间
ACM-347 【ACM 0.5】ACM... min.zhang 高优先级 完成 2021-06-28 16:19
ACM-343 【ACM 0.5】工作负... min.zhang 高优先级 完成 2021-06-29 16:28
ACM-342 【ACM 0.5】截图中... jian.zhang-ndx 高优先级 以后处理 2021-06-30 10:13
ACM-360 【ACM 0.5】以http... 故障 yiting.jiang min.zhang 高优先级 完成 2021-06-29 16:37
ACM-154 [前端]设置角色界面 子任务 yang.li jian.zhang-ndx 高优先级 完成 2021-06-04 10:32
ACM-153 [后端]List Roles API... 子任务 chenyang.shi jian.zhang-ndx 高优先级 完成 2021-05-31 16:51
ACM-152 [后端]List Roles API... 子任务 yiting.jiang jian.zhang-ndx 高优先级 完成 2021-05-21 11:49
ACM-151 [后端]SetRoles API ... 子任务 chenyang.shi jian.zhang-ndx 高优先级 完成 2021-06-04 15:12
ACM-150 [后端]SetRoles API ... 子任务 yiting.jiang jian.zhang-ndx 高优先级 完成 2021-05-25 17:42
ACM-149 [后端]roles/permissi... 子任务 yiting.jiang jian.zhang-ndx 高优先级 完成 2021-06-02 17:49
共 413 项
1 / 42
10 项

issue01

凭证管理

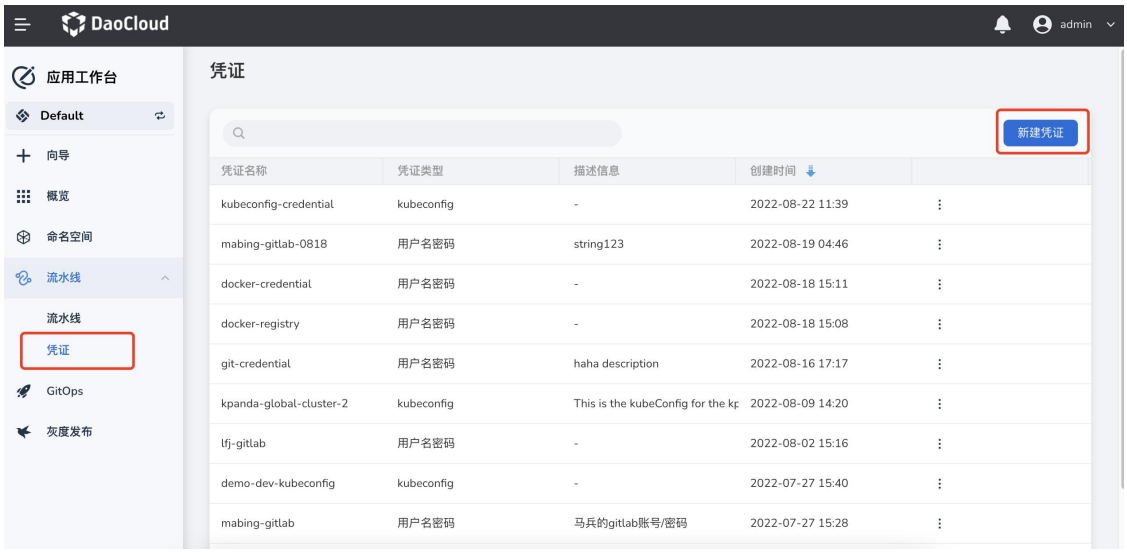
凭证可以保存敏感信息，例如用户名密码、访问令牌（Secret text）、Kubeconfig 等，使保存的数据更安全灵活并免于暴露到镜像中。 当流水线运行时，大多数情况下会与第三方网站或应用程序进行交互，如 Git 仓库、镜像仓库等。 此过程中需要提供相应的凭证，所以需要用户来为流水线配置凭证，用户配置凭证后，便可以利用这些凭证来与第三方网站或应用程序进行交互。

目前，您可以在应用工作台中创建以下 3 种类型的凭证：

- **用户名和密码**：用于存储用户名和密码的认证信息，若第三方网站或应用程序支持用户名/密码的方式访问，则可以选择这种类型，例如 GitHub、GitLab 和 Docker Hub 的帐户。
- **访问令牌（Secret text）**：API token 之类的 token (如 GitHub 个人访问 token)。
- **kubeconfig**：用于配置跨集群认证。

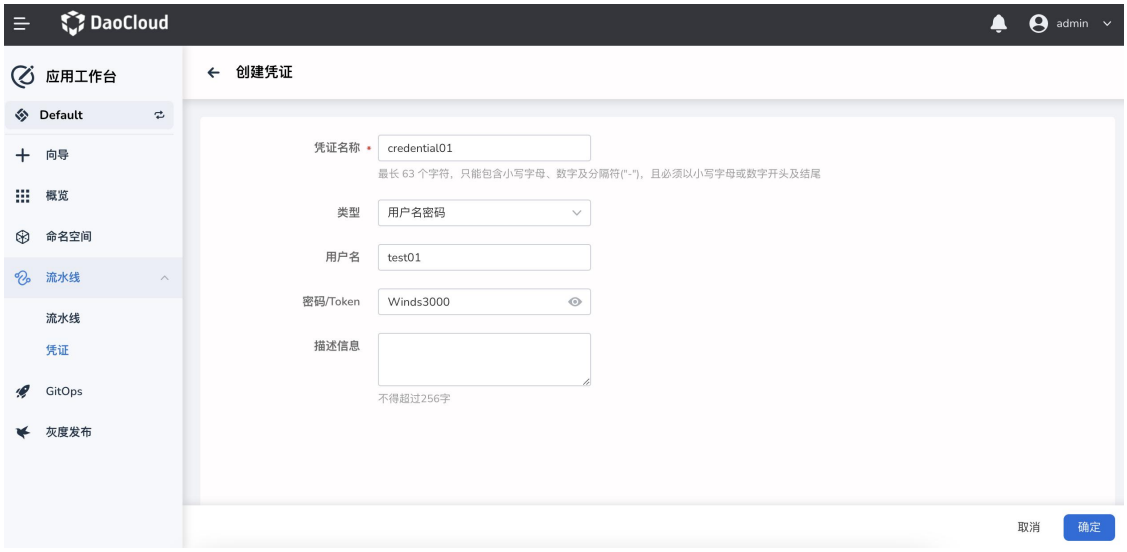
创建和管理凭证的具体步骤如下：

1. 在左侧导航栏中点击流水线–>凭证，进入凭证列表，点击右上角的新建凭证。



createcredential

2. 在创建凭证页面中，配置相关参数后点击确定。

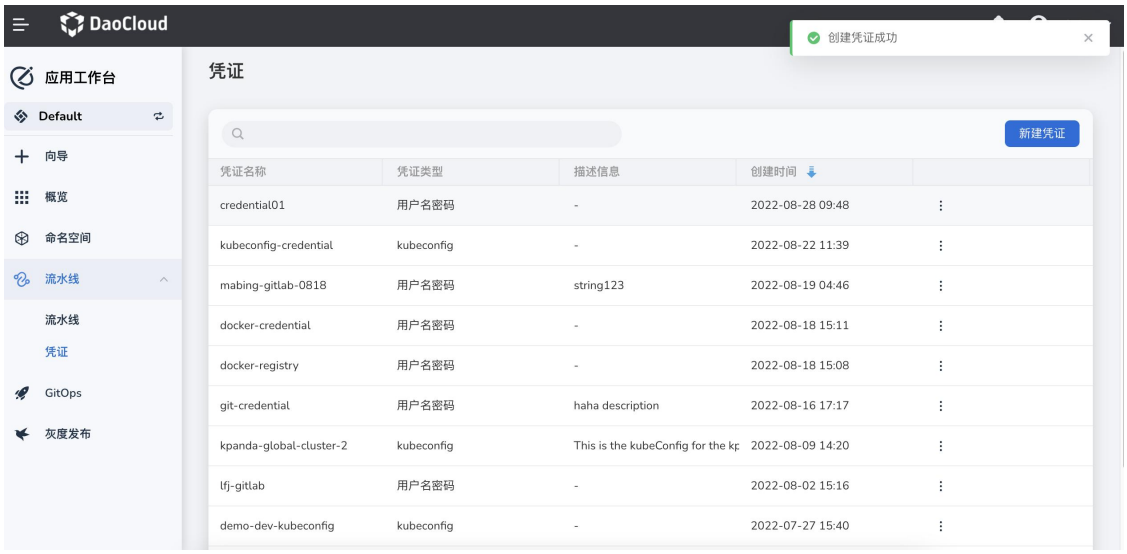


createcredential

- 填写凭证名称，设置可以在流水线中使用的 ID，例如 dockerhub-id。注意：一旦设置就不能再进行更改。
- 在类型字段中，选择要添加的凭证类型。
- 根据不同的凭证类型填写相应的字段：

- 用户名和密码：在对应字段指定凭证的用户名和密码。
- 访问令牌（Secret text）：复制加密文本到令牌字段中。
- Kubeconfig：复制集群证书到 Kubeconfig 字段中。

3. 屏幕提示创建成功，新创建的凭证默认位于第一个。



createcredential

4. 点击列表右侧的 ⋮，可以在弹出菜单中选择编辑或删除等操作。

!!! warning

如果删除了某个流水线正使用的凭证，可能会影响流水线的访问，请谨慎操作。

编辑流水线

创建自定义流水线之后，需要手动定义流水线阶段。目前支持通过编辑 Jenkinsfile 或通过图形化界面表单为流水线定义执行阶段。此外，如需更改流水线各阶段的配置，也可参考本文调整各项配置。

!!! note

- 有关编辑流水线时涉及到的概念，可参考[流水线中的概念](config/concept.md)
- 有关编辑流水线时涉及到的参数，可参考[图形化任务模板参数说明](config/step.md)

前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 workspace editor 或更高权限。
- 创建可以访问代码仓库、镜像仓库、集群的三个凭证，可参考凭证管理。
- 创建自定义流水线，并且需要在构建参数中添加两个字符串参数。这些参数将用于镜像构建的命令中。参数说明如下：

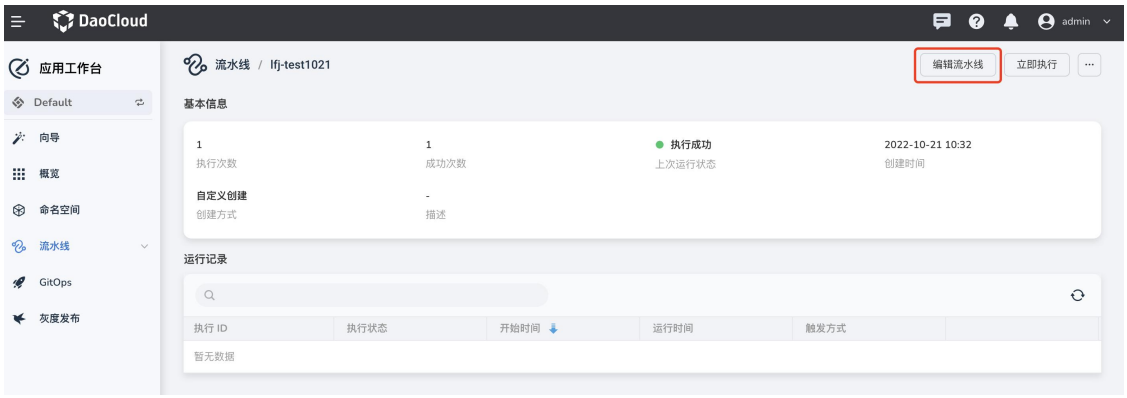
参数类型	参数名称	描述信息
字符串	registry	镜像仓库地址。本例中使用 release-ci.daocloud.io
字符串	project	镜像仓库中的项目名称。本例中使用 demo
字符串	name	镜像的名称。本例中使用 http-hello

通过界面表单编辑流水线

应用工作台设计了图形化流水线编辑视图，能够兼容编辑 Jenkinsfile 中的大部分自定义操作，便于查看或定义流水线的每个阶段（Stage)和每个步骤（Step)，实现所见即所得的流水线编辑体验。

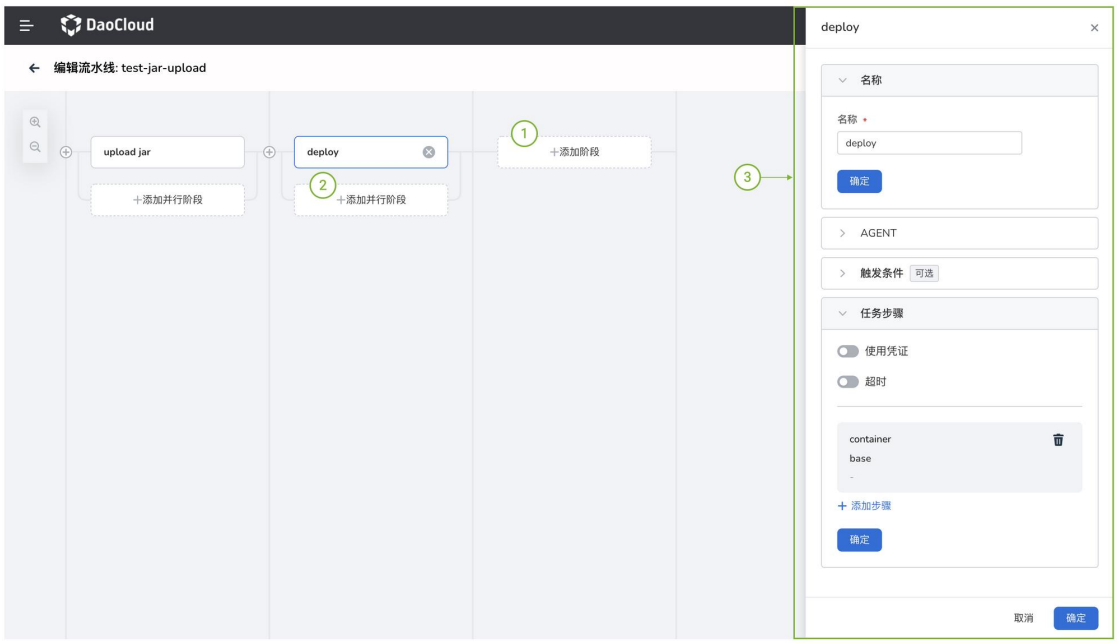
界面布局说明

1. 点击一个创建好的自定义流水线。在右上角点击编辑流水线，进入图形化编辑页面。



pipelisetting

2. 图形化编辑页面包括两个区域：**画布（左侧）** 和 **阶段设置（右侧）**。



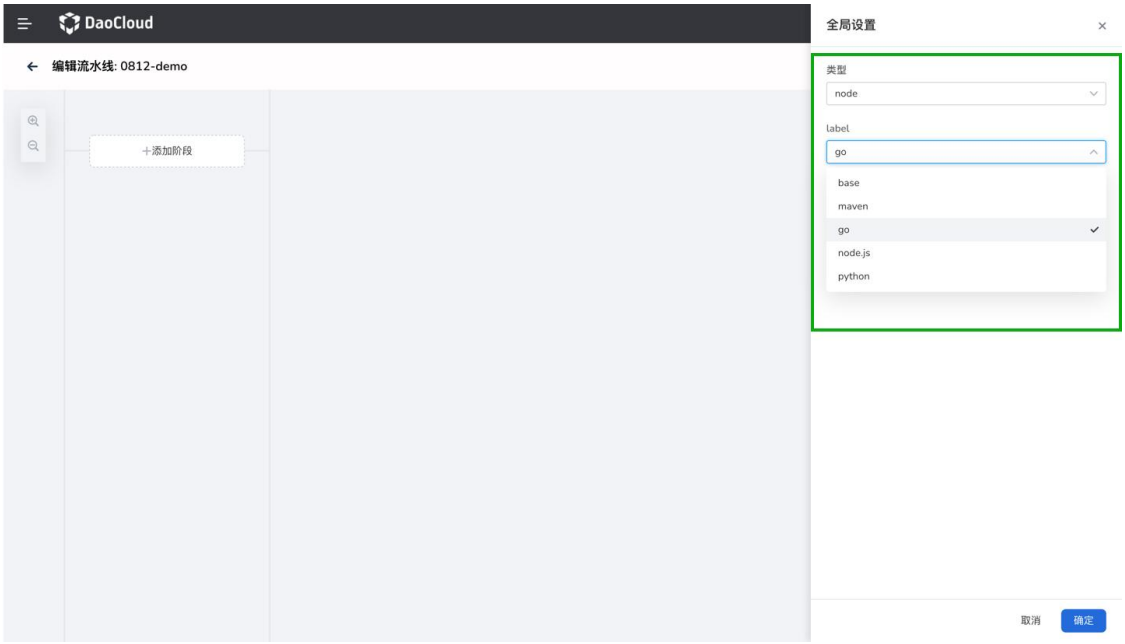
canvasdescribe

- 点击 +添加阶段后，会生成一个串行阶段，点击后，会生成一个新的阶段，可以在该阶段中添加流水线步骤，通过选择步骤类型，可以快速创建当前阶段中的流水线步骤。

- 点击+添加并行阶段后，会生成一个并行阶段，点击后，会生成一个新的阶段，可以在该阶段中添加流水线步骤，通过选择步骤类型，可以快速创建当前阶段中的流水线步骤。
- 阶段创建后，点击阶段，可以用过阶段进行阶段配置。支持设置阶段的名称、Agent、触发条件、步骤。

配置全局设置

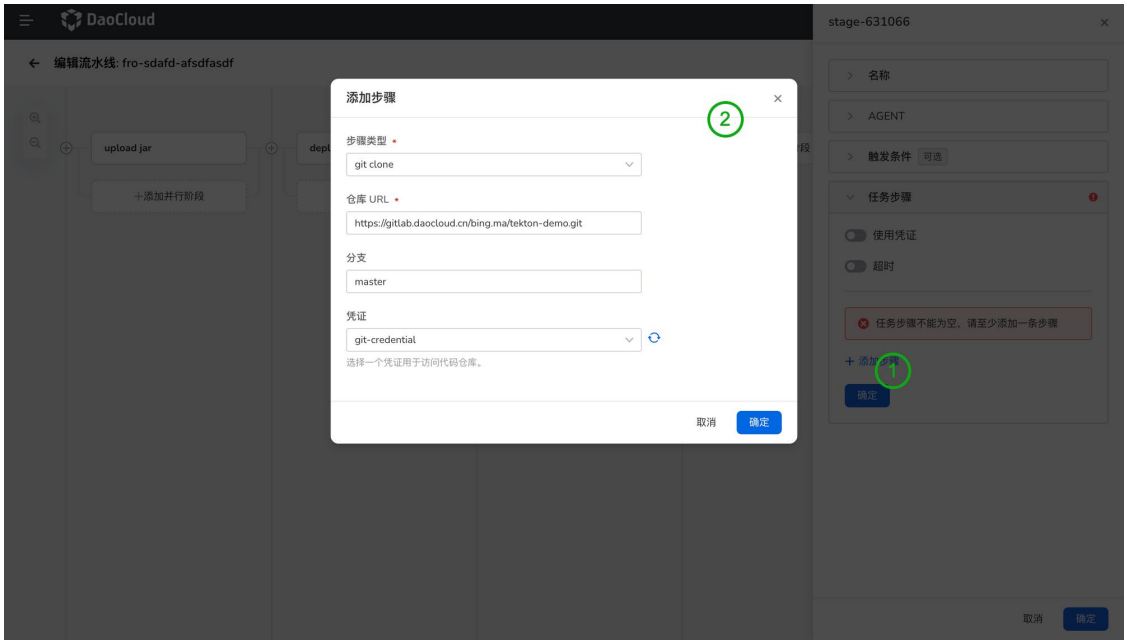
点击全局设置，从类型下拉列表中选择 node，从 label 下拉列表选择 go 16。



pipelisetting

添加阶段 - 拉取源代码

1. 点击画布中的添加阶段。在右侧的阶段设置中设置名称：git clone。
2. 点击添加步骤，在弹出的对话框中步骤类型下选择 git clone，参照下表，对相关参数进行配置。
   - 仓库 URL：输入仓库地址。
   - 分支：不填写，默认为 master 分支。
   - 凭证：如果您的仓库属于私有仓库则需要提供一个凭证。

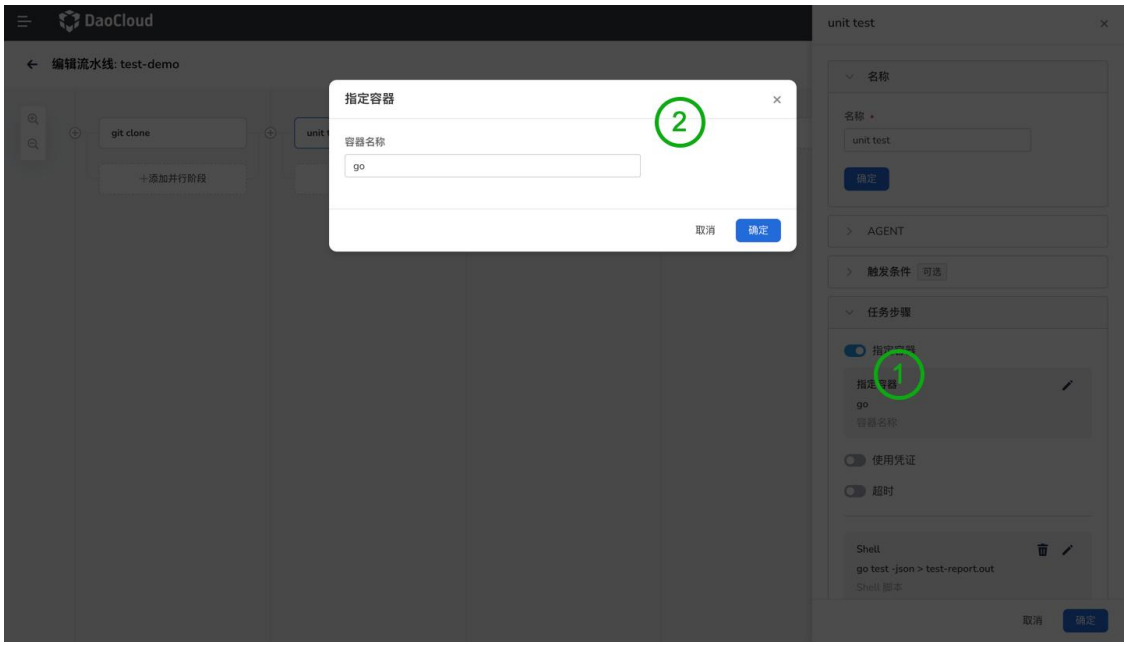


git_clone

添加阶段 - 单元测试

1. 点击画布中的添加阶段。在右侧的阶段设置中设置名称：unit test。

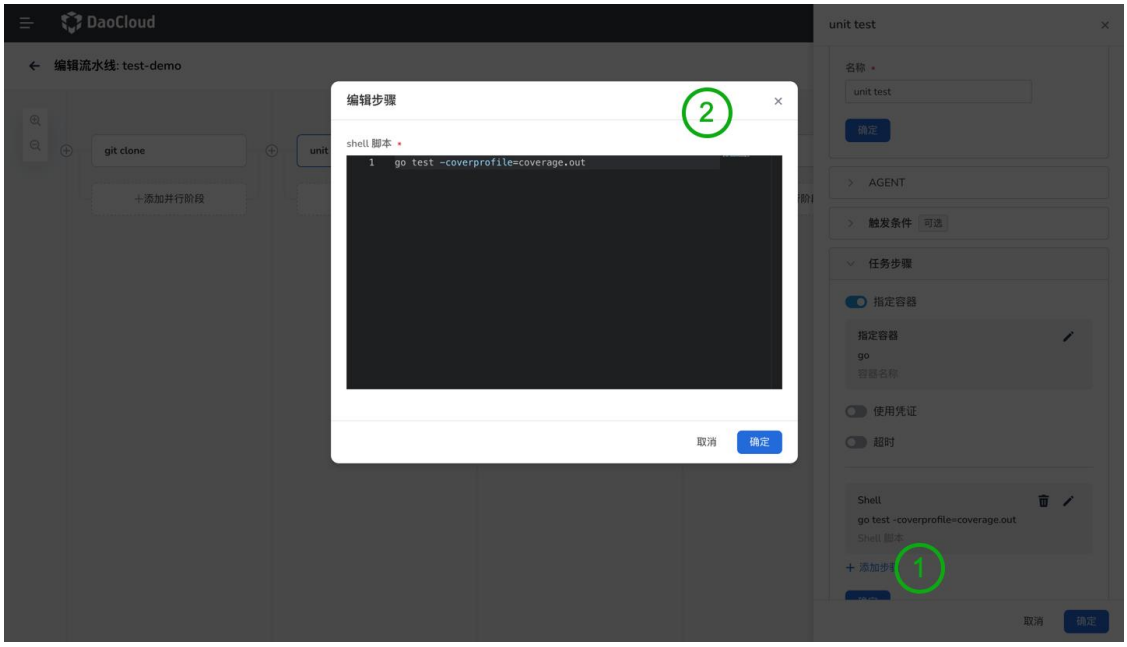
2. 在步骤模块中选择开启指定容器，在弹出的对话框中填写容器名称：go，然后点击确定。



container1

3. 点击添加步骤以对代码进行单元测试，并生成测试报告，在弹出的对话框中步骤类型下选择 shell，并在命令行中输入以下命令，然后点击确定。

```
go test -coverprofile=coverage.out
```

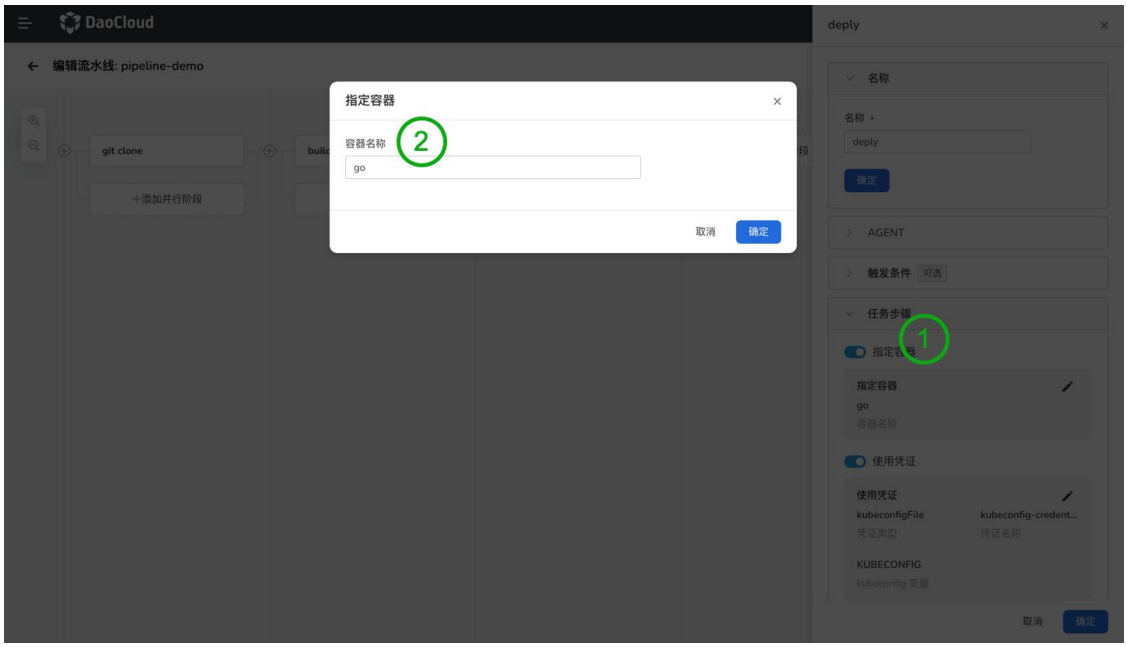


go-test

添加阶段 - 构建并推送镜像

1. 点击画布中的添加阶段。在右侧的阶段设置中设置名称：build & push。

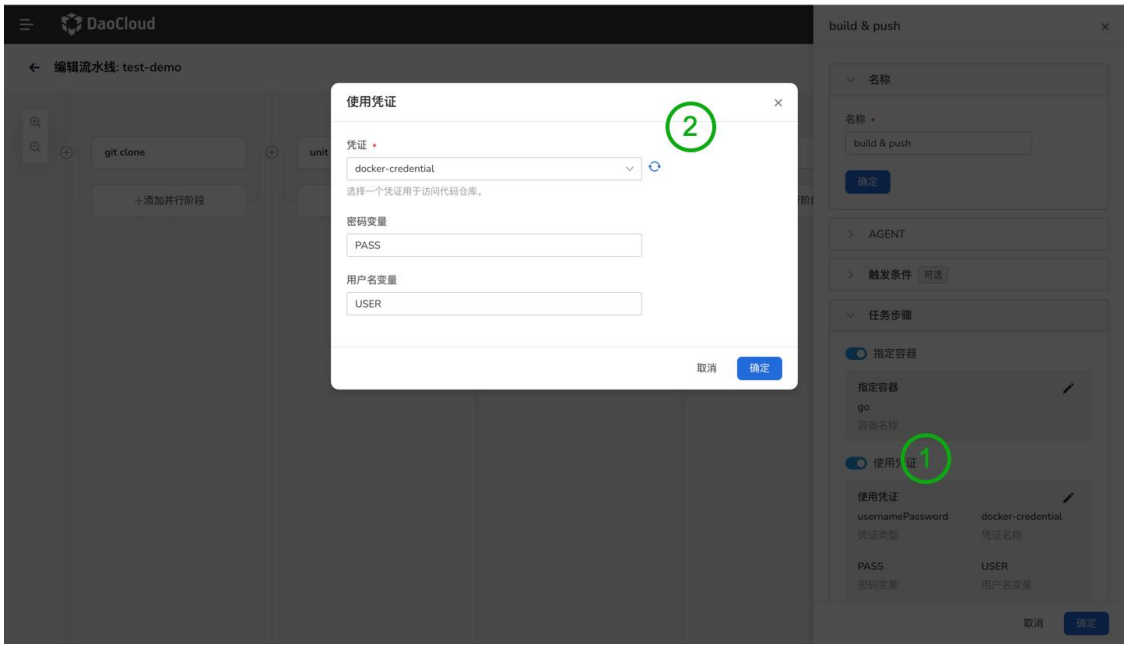
2. 在步骤模块中选择开启指定容器，在弹出的对话框中填写容器名称：go，然后点击确定。



container2

3. 在步骤模块中选择开启使用凭证，在弹出的对话框中填写相关参数，然后点击确定。

   - 凭证：选择创建的 docker hub 凭证，用户访问镜像仓库。选择已创建好的“docker-credential”凭证。
   - 密码变量：PASS
   - 用户名变量：USER



creditional1

4. 点击添加步骤进行代码构建，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

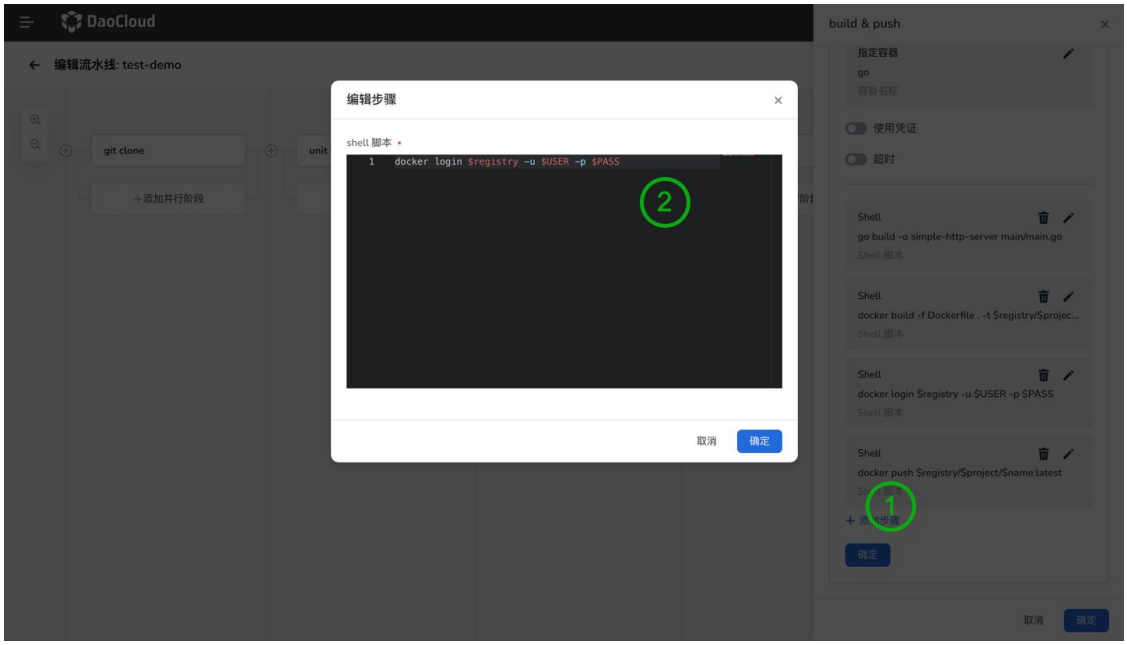
   ```
   go build -o simple-http-server main.go
   ```

5. 点击添加步骤以根据源码中的 Dockerfile 构建 Docker 镜像，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

   ```
   docker build -f Dockerfile . -t $REGISTRY/$PROJECT/$NAME:latest
   ```

6. 点击添加步骤以登录镜像仓库并将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

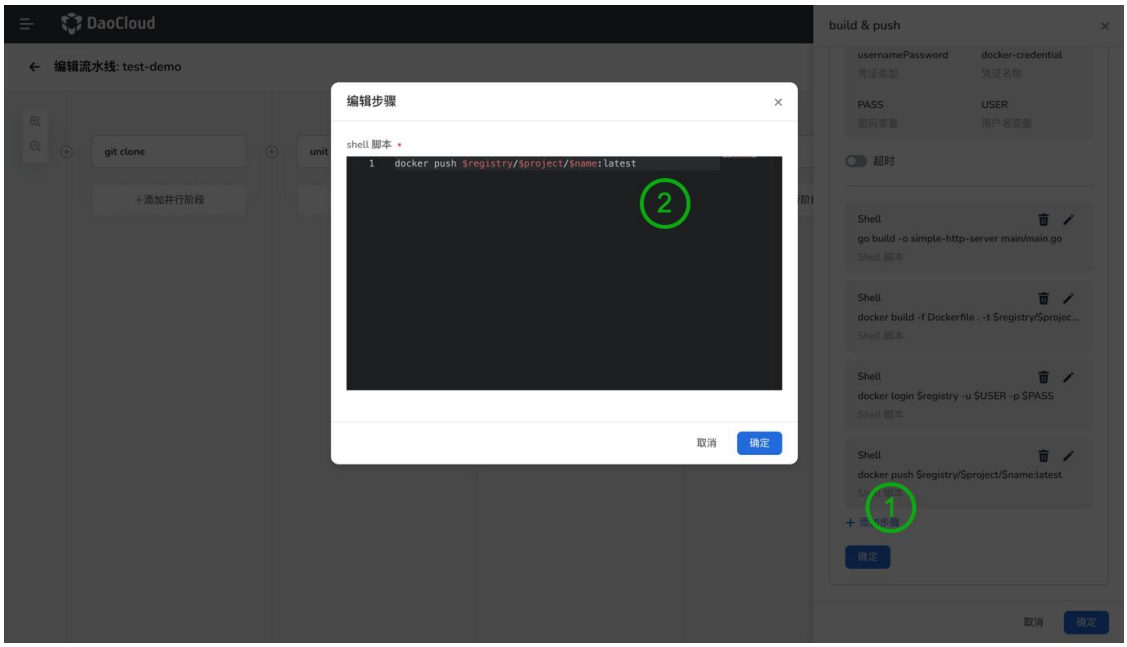
   ```
   docker login $REGISTRY -u $USER -p $PASS
   ```



build3

7. 点击添加步骤以将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

   docker push $REGISTRY/$PROJECT/$NAME:latest



build4

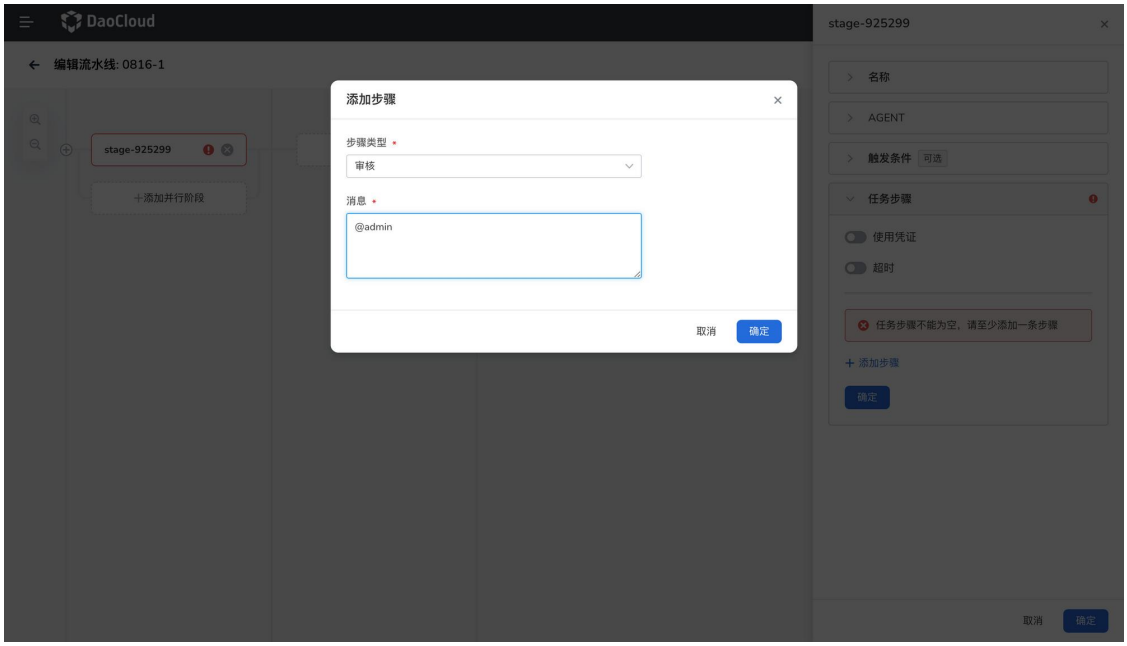
!!! note

镜像更新后还可以实现触发流水线，具体操作请参考[触发流水线](run/trigger.md)

添加阶段 - 审核

1. 点击画布中的添加阶段。在右侧的阶段设置中设置名称：review。

2. 点击添加步骤，在弹出的对话框中步骤类型下选择审核，在消息字段中填入 @admin，即 admin 帐户在流水线运行到该阶段时会进行审核，然后点击确定。

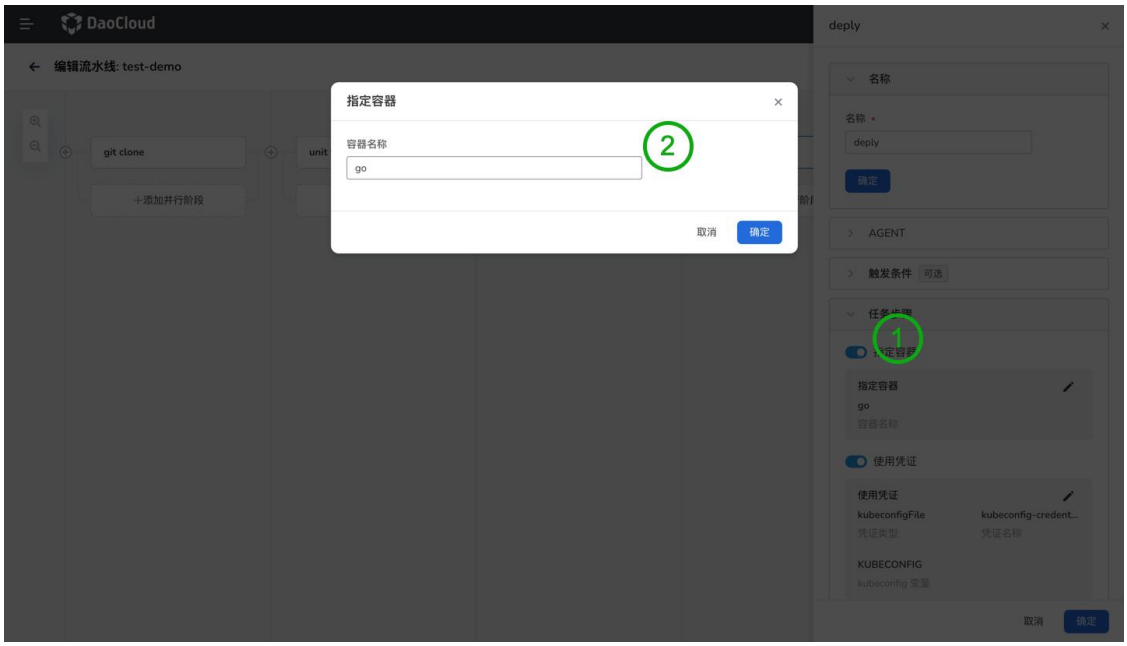


review

添加阶段 - 部署至集群

1. 点击画布中的添加阶段。在右侧的阶段设置中设置名称：deploy。

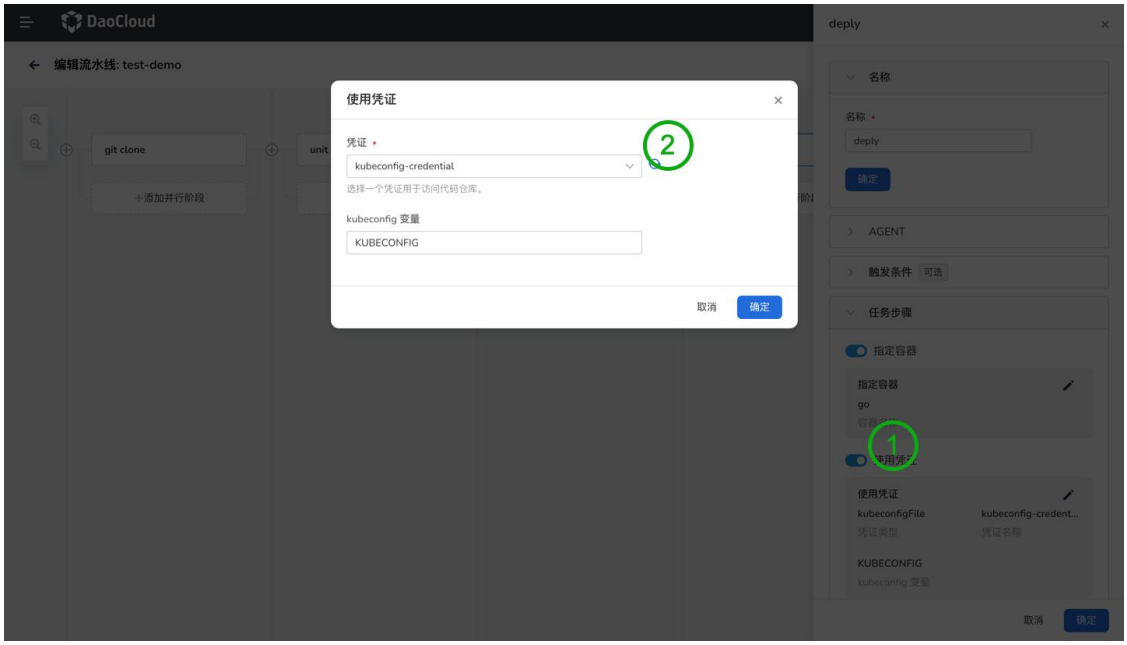
2. 在步骤模块中选择开启指定容器，在弹出的对话框中填写容器名称：go，然后点击确定。



container3

3. 在步骤模块中选择开启使用凭证，在弹出的对话框中填写相关参数，然后点击确定。

   - 凭证：选择 kubeconfig 类型的凭证。
   - kubeconfig 变量：如果使用的是 kubectl apply 的部署方式，变量值必须为 KUBECONFIG。



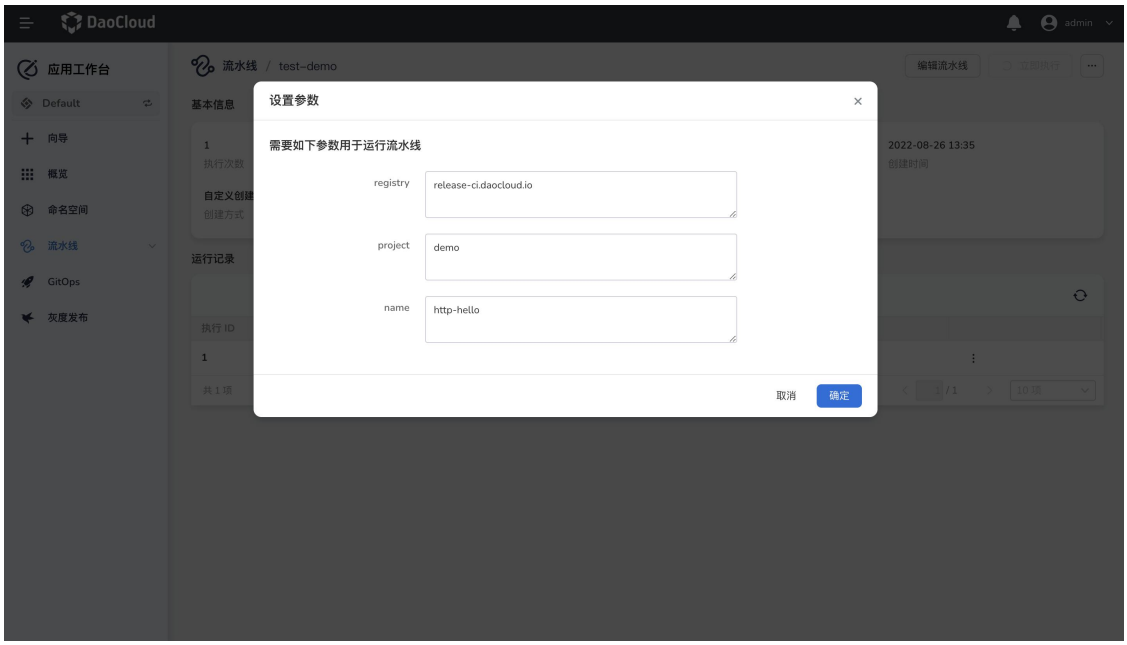
credentials2

4. 点击添加步骤以进行集群部署操作，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

   kubectl apply -f deploy.yaml

运行流水线

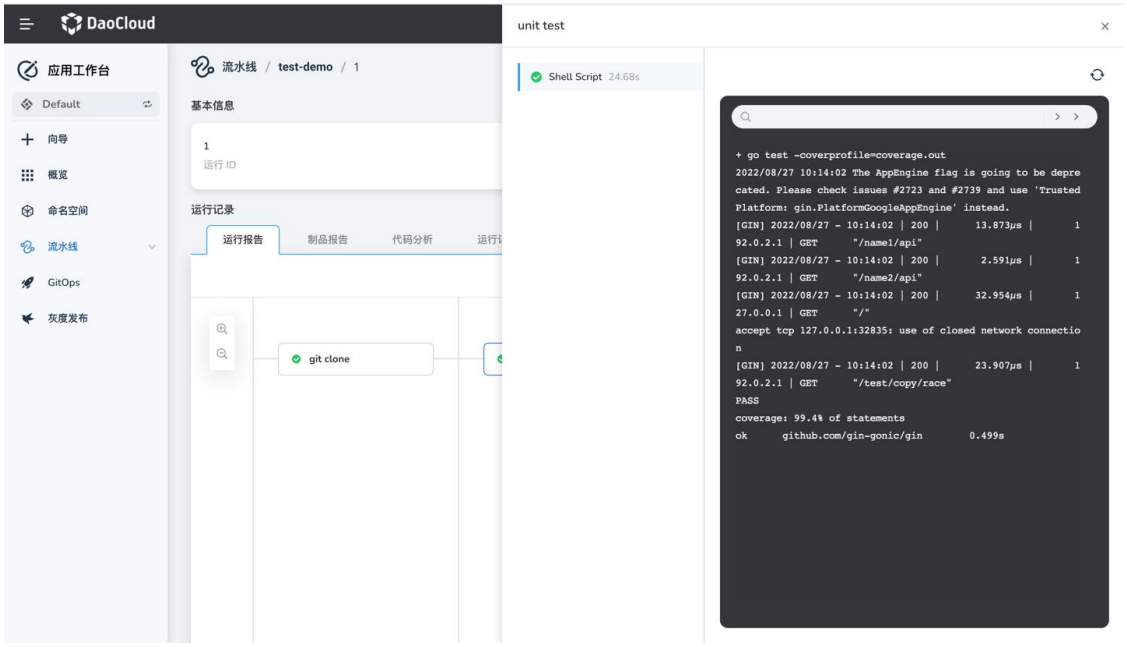
1. 在流水详情页面点击立即执行。在弹出的对话框中设置前提条件中定义的三个字符串参数，点击确定来运行流水线。



run

2. 成功开始运行后，页面会自动切换到流水线详情页面，点击当前运行的流水线记录。

3. 进入流水线记录详情页面后，您可以查看到当前流水线运行的过程，需要 admin 或者平台管理员来为流水线进行审核，审核成功后，资源会部署至集群。

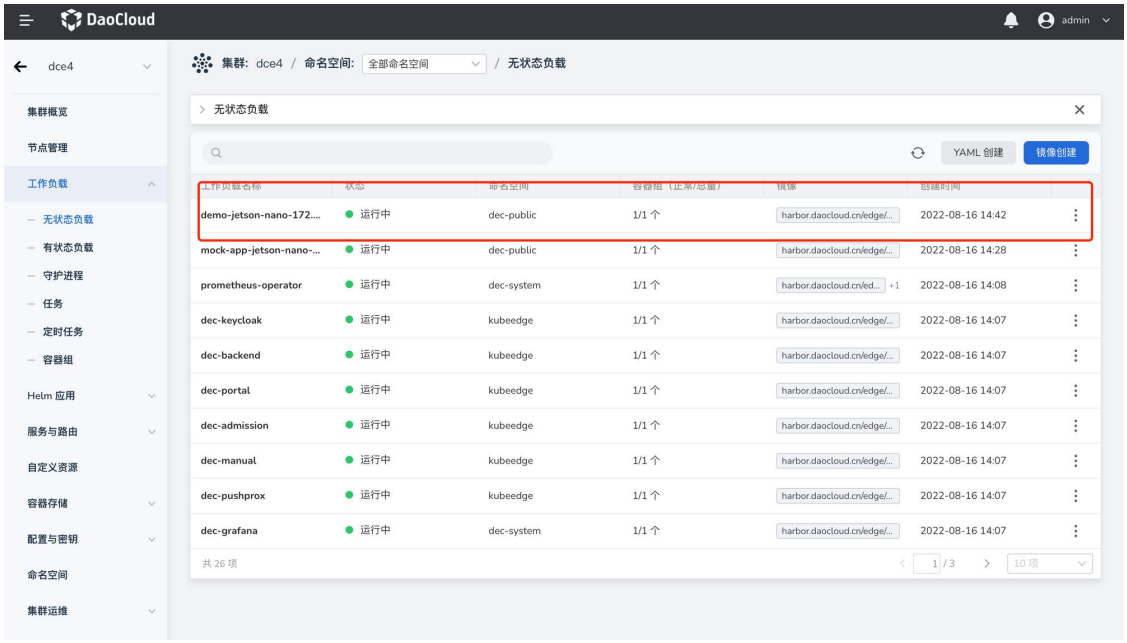


report

验证集群资源

1. 如果流水线的每个阶段都成功运行，则会自动构建一个 Docker 镜像并推送至您的 Docker Hub 仓库。最终，流水线将在您事先设置的项目中自动创建一个无状态负载。

2. 前往容器管理平台，点击集群下的工作负载，您可以看到列表中显示的无状态工作负载。

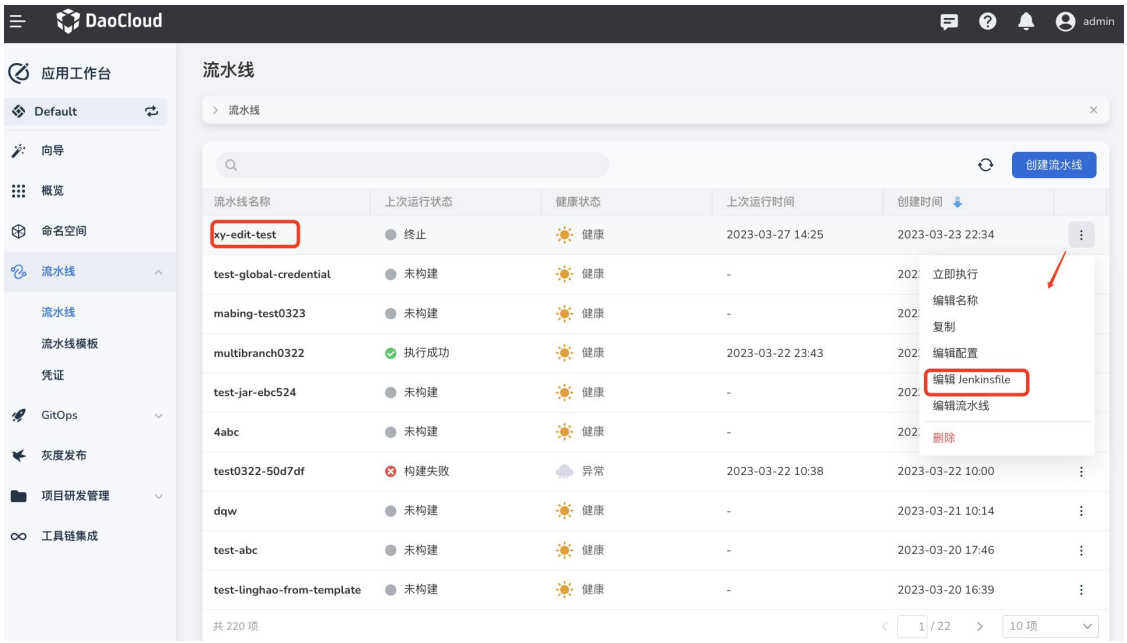


deployment

通过 Jenkinsfile 编辑流水线

通过 Jenkinsfile 可以快速定义流水线的各个阶段。

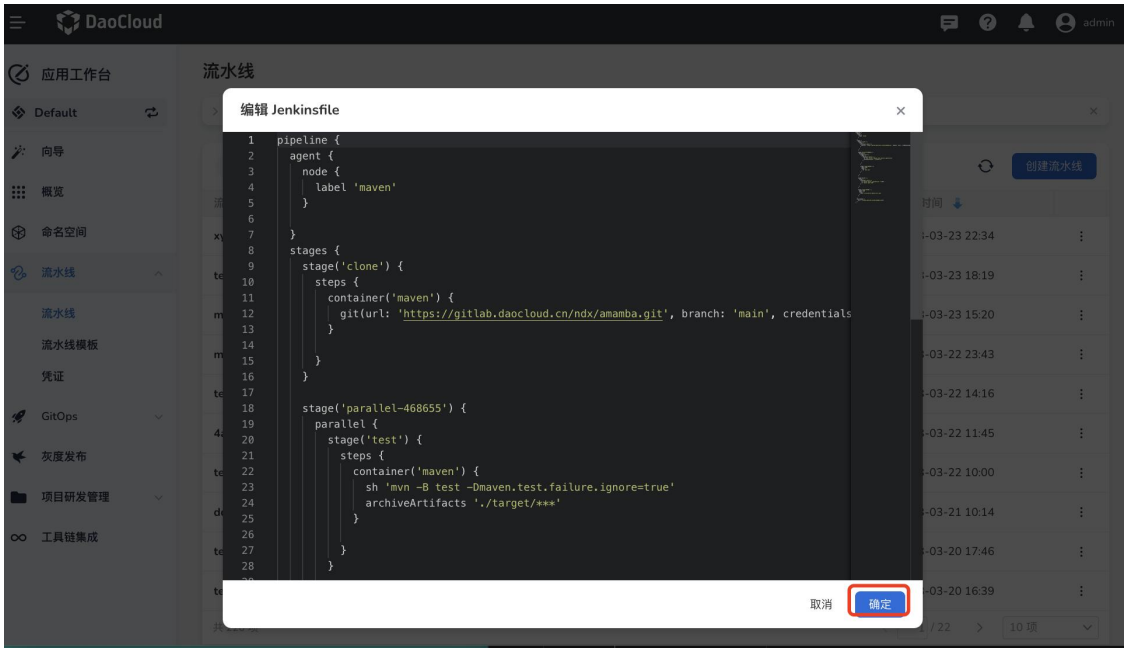
1. 在流水线列表页面找到需要新建的流水线，在右侧点击编辑 Jenkinsfile。



edit

也可以点击流水线名称后，在页面右上角点击编辑 Jenkinsfile。

2. 输入或粘贴事先准备好的 Jenkinsfile，然后点击确定即可。



edit

安装 Jenkins

前提条件

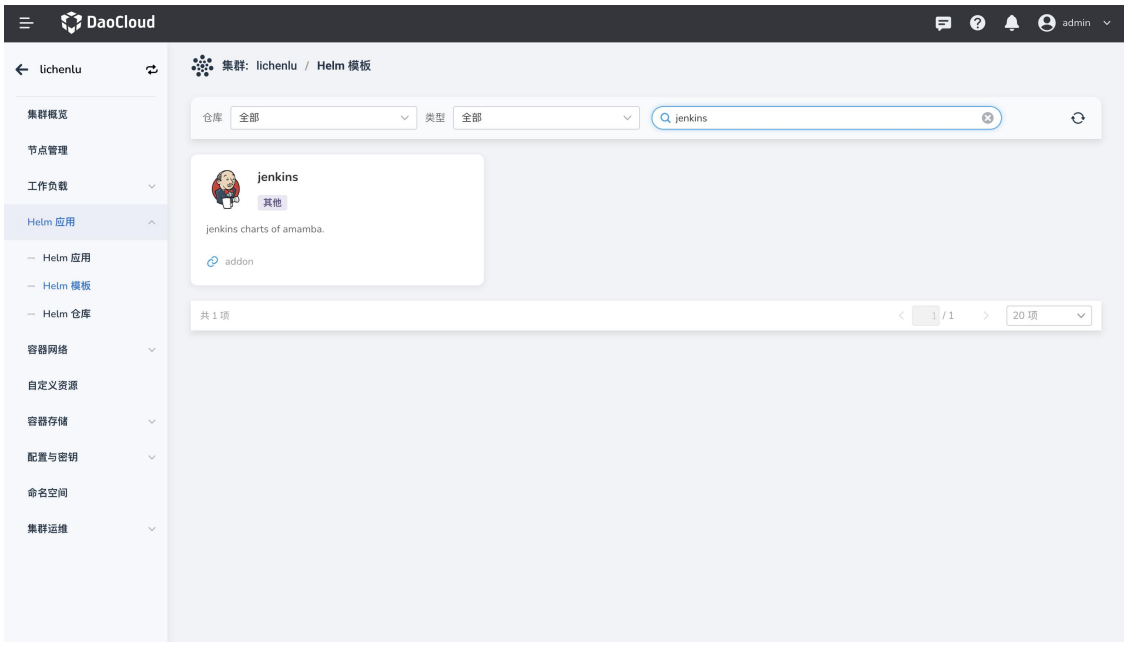
- 安装 Jenkins 之前需要确保将要 Jenkins 的集群中存在默认的存储类。
- 请确保安装在 amamba-system 命名空间下。
- 如果在全局服务集群安装，请确保在容器管理->helm 应用，amamba-system 命名空间下的 amamba-jenkins 实例。

开始安装

1. 进入容器管理模块，在集群列表中找到需要安装 Jenkins 的集群，点击该集群的名称。

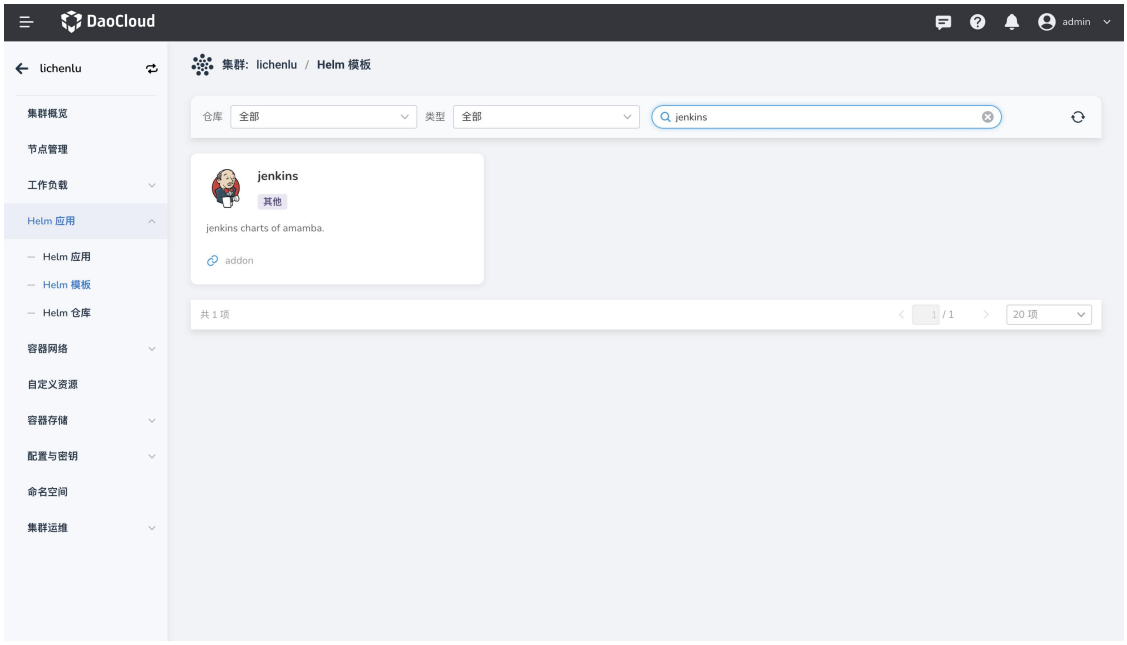
   !!! note

   需要根据实际情况选择 Jenkins 的部署集群。目前不建议将其部署在全局服务集群，因为 Jenkins 执行流水线高并发时会占用大量资源，可能会导致全局服务集群的瘫痪。



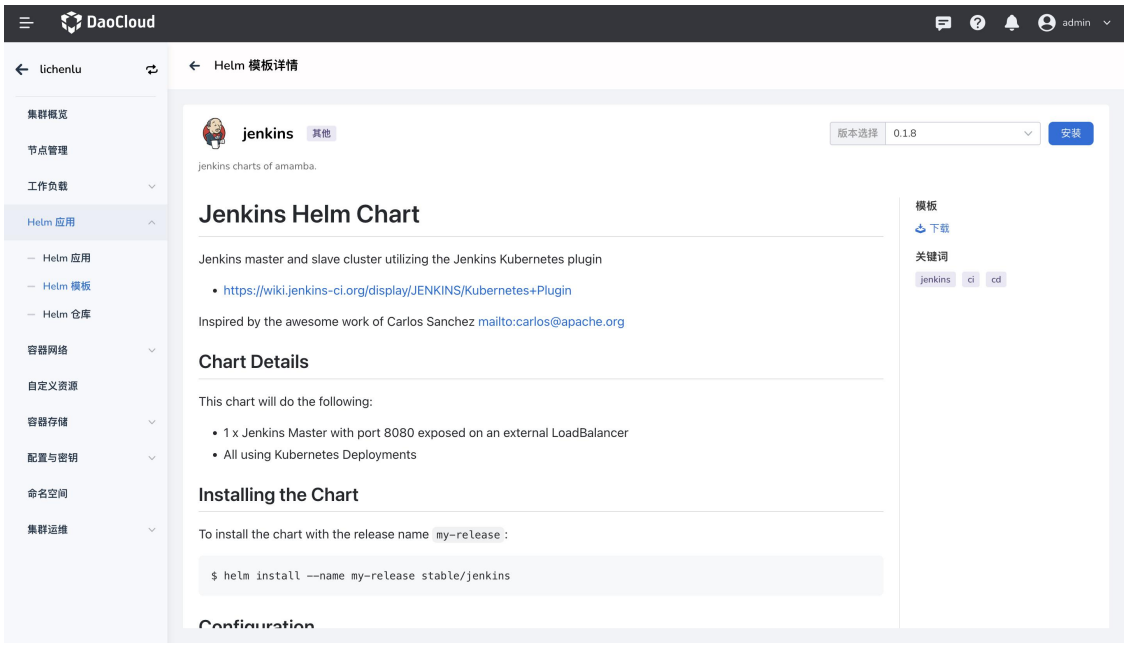
点击集群名称

2. 在左侧导航栏中选择 Helm 应用 –> Helm 模板，找到并点击 Jenkins。



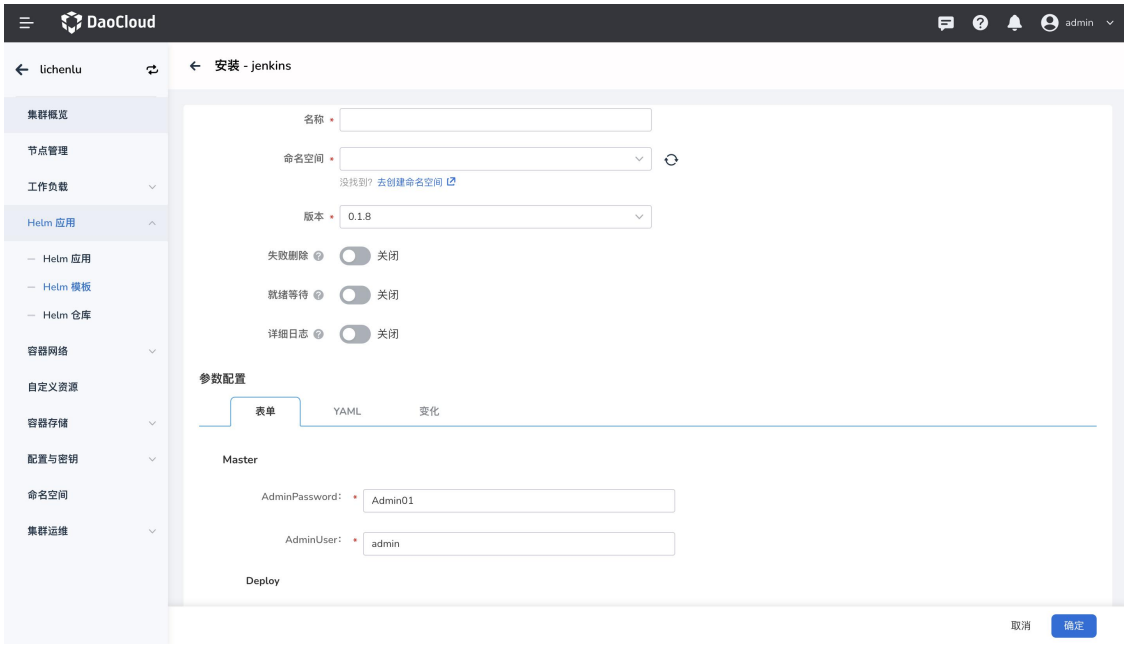
jenkins helm

3. 在版本选择中选择想要安装的版本，点击安装。



安装

4. 在安装界面，填写所需的安装参数，最后在右下角点击确定按钮。



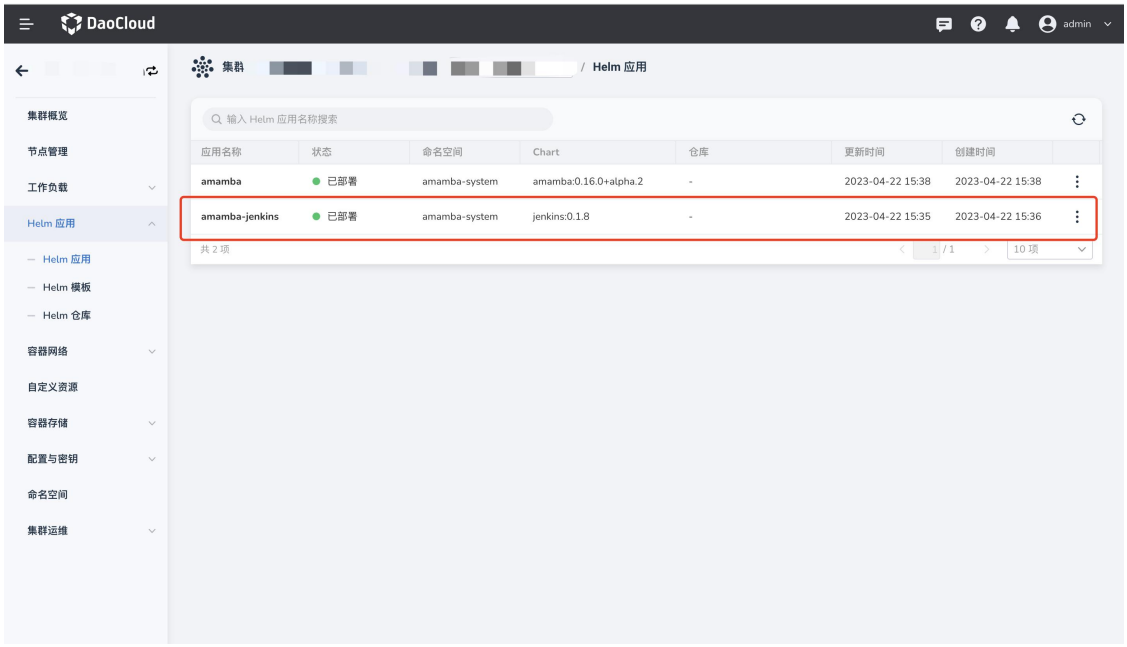
填写配置

以下是重要参数说明，根据实际业务需求进行参数更换编写。

参数	描述
AdminUser	Jenkins 的用户名
AdminPassword	Jenkins 的密码
Deploy.JenkinsHost	Jenkins 的访问链接。如果选择 Node Port 方式，访问地址规则为：http://{集群地址:端口}
JavaOpts	指定启动 Jenkins 的 JVM 启动参数
ServiceType	默认为 ClusterIP，支持 ClusterIP、NodePort、LoadBalancer
ServicePort	服务访问端口
NodePort	如果 ServiceType=NodePort 则需要必填，范围为：30000-32767
resources.requests	Jenkins 的资源请求值
resources.limits	Jenkins 的资源限制值
image.registry	jenkins 镜像
eventProxy.enabled	true 代表部署在非全局服务集群 false 代表部署在全局服务集群
eventProxy.image.registry	如果 enabled=true 必须填写
eventProxy.image.repository	如果 enabled=true 必须填写
eventProxy.image.tag	如果 enabled=true 必须填写
eventProxy.imagePullPolicy	如果 enabled=true 必须填写
eventProxy.configMap.eventroxy.host	如果 enabled=true 必须填写
eventProxy.configMap.eventroxy.proto	如果 enabled=true 必须填写
eventProxy.configMap.even	如果 enabled=true 必须填写 token 获取方式参

参数	描述
troxy.token	考全局管理访问密钥文档：https://docs.daocloud.io/ghippo/04UserGuide/06PersonalCenter/Password/

5. 前往 Helm 应用查看部署结果。

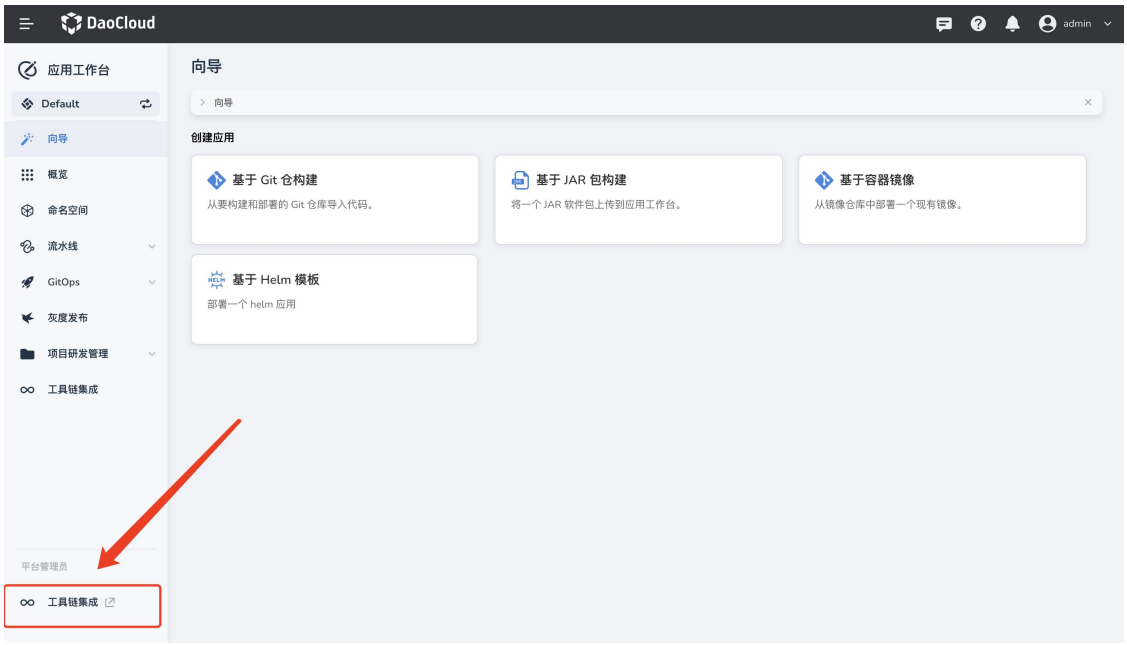


完成创建

集成 Jenkins

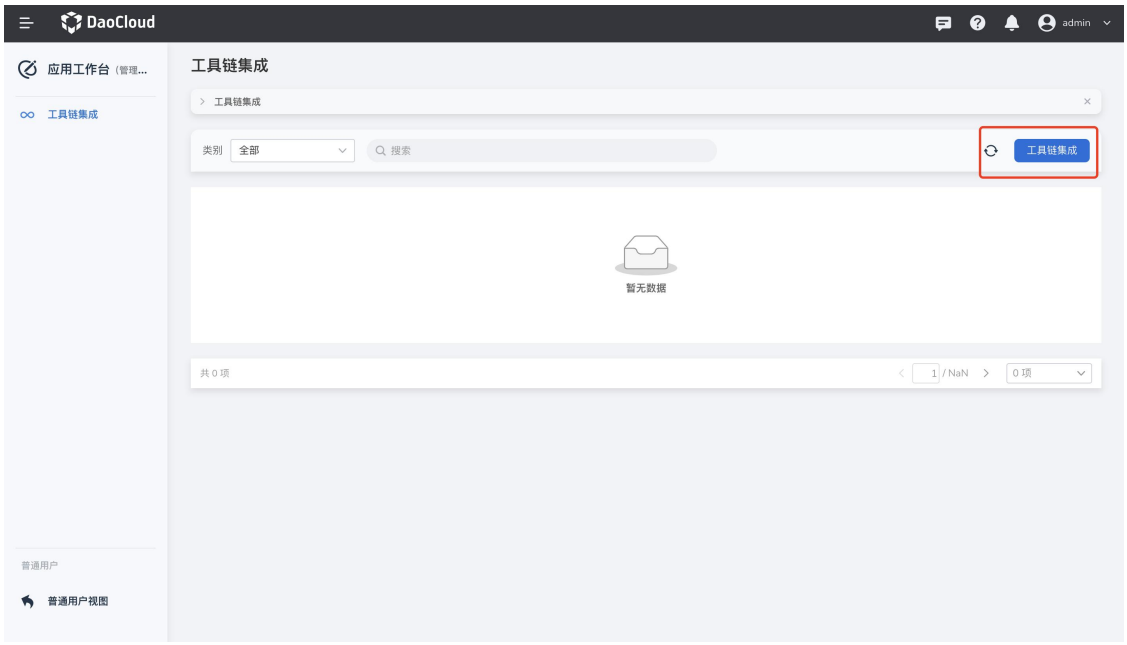
说明：目前仅支持集成通过 DCE 5.0 平台安装的 Jenkins。

1. 使用具有应用工作台管理员角色的用户登录 DCE 5.0 并进入应用工作台



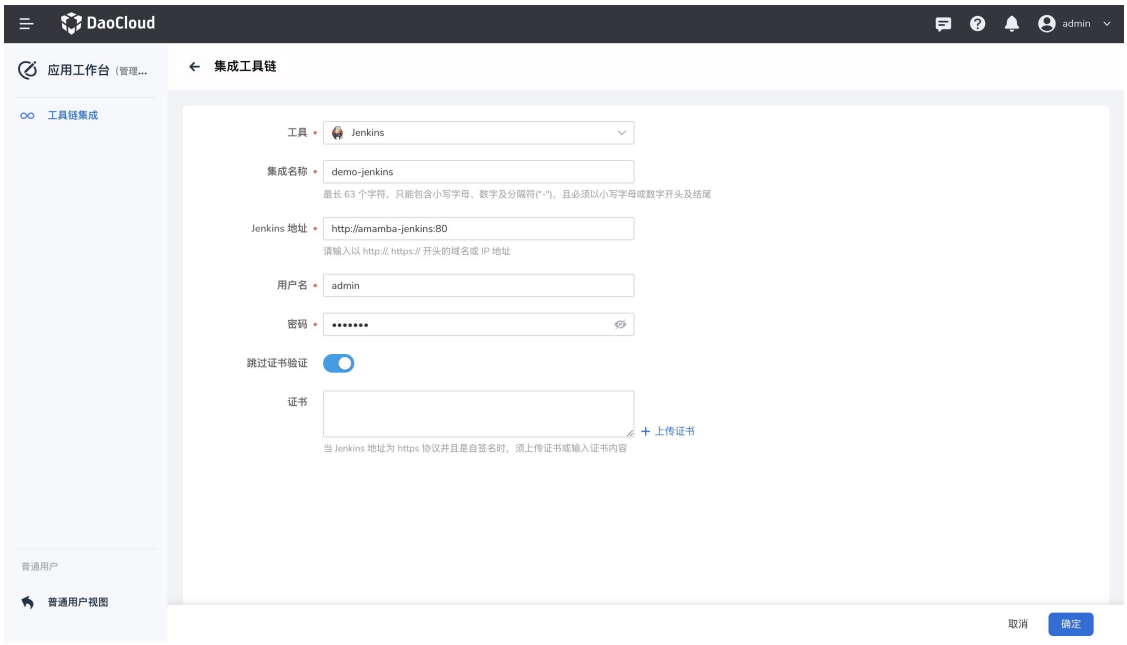
完成创建

2. 在左侧导航栏点击平台管理下的工具链集成，点击右上角的集成按钮。



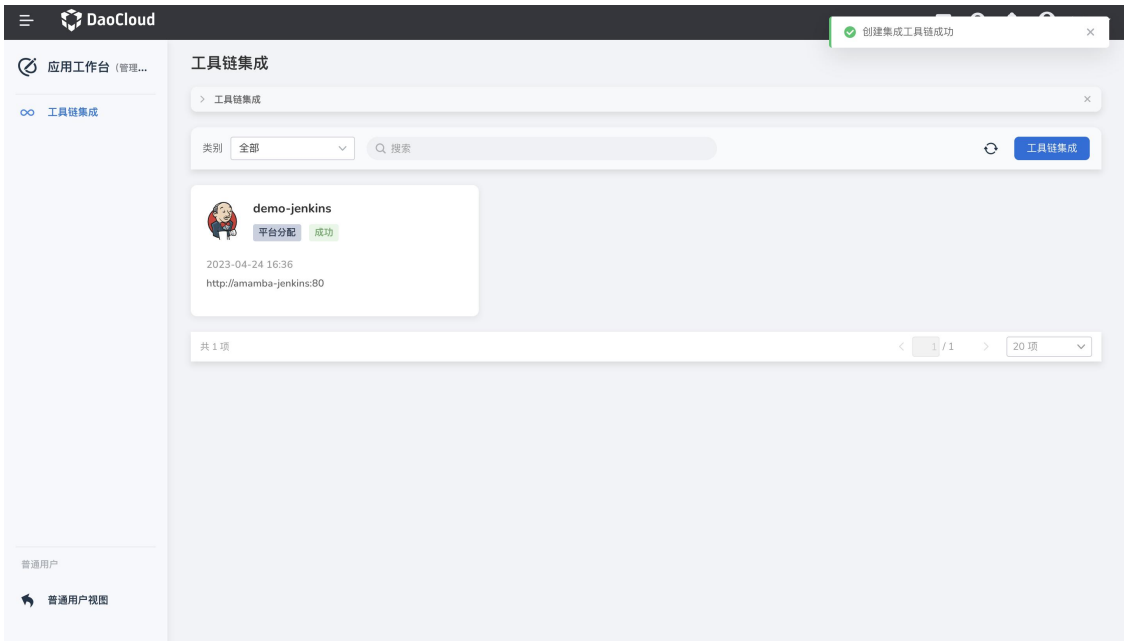
完成创建

3. 选择工具链类型 Jenkins，填写集成名称、Jenkins 地址、用户名和密码。如果 Jenkins 地址为 https 协议时，需要提供证书。通过 Helm 部署出来的 Jenkins 默认账户密码为 admin/Admin01。



完成创建

4. 集成完毕后会在工具链列表页面成功生成一条记录。



完成创建

5. 接下来就可以前往工作空间内创建流水线。

流水线节点（Agent）

Agent 描述了整个流水线执行过程或者某个阶段的执行环境，必须出现在描述文件顶部或每一个阶段。

本文基于 Kubernetes plugin for Jenkins 插件描述如何扩展 Kubernetes 中运行的 Jenkins Agent。

Kubernetes Pod 模板介绍

这个 Kubernetes 插件会在 Jenkins Agent Pod 中运行一个特殊的容器 jnlp，目的是为了在 Jenkins Controller 和 Jenkins Agent 之间进行通信，所以需要定义其他容器来运行流水线步骤，并且可以通过 container 命令来切换不同的容器。

使用内置 Label

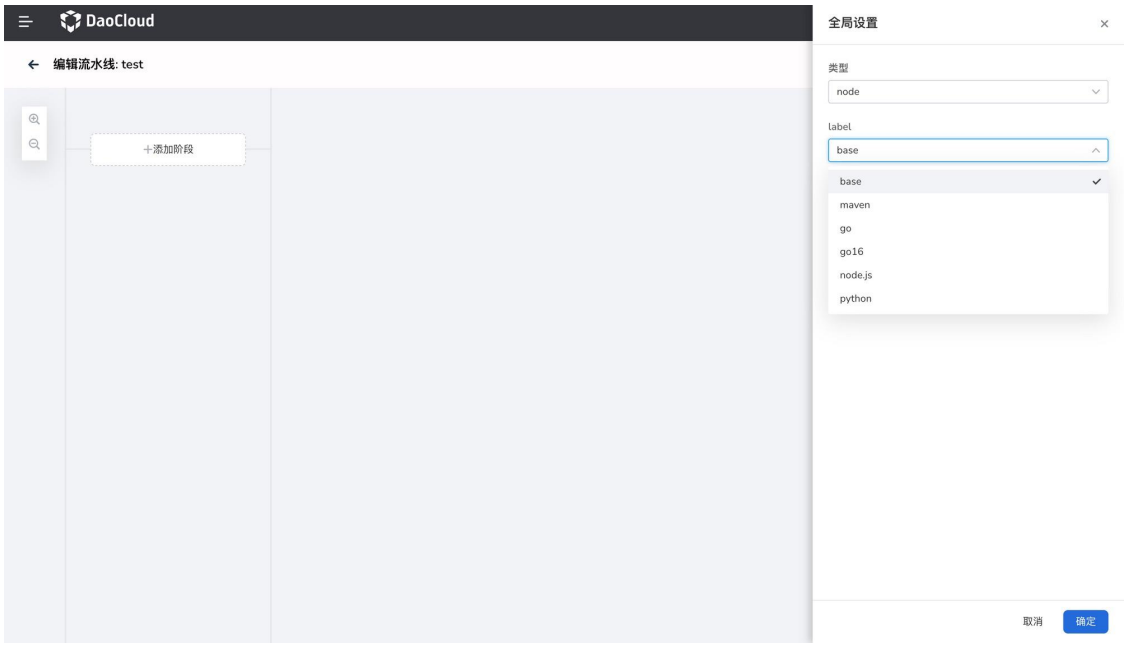
应用工作台通过 podTemplate 能力声明了 7 个 label：base、maven、mavenjdk11、go、go16、node.js 和 python。您可以指定具体的 Agent 标签来使用对应的 podTemplate。

- 可以在 Jenkinsfile 中通过 node('go') 使用 go 的 podTemplate。

```
pipeline {
  agent {
    node {
      label 'go'
    }
  }

  stages {
    stage('go') {
      steps {
        container('go') {
```

```
                sh 'go version'
            }
        }
    }
  }
}
```

- 也可以在编辑流水线页面上选择类型为 node 且 label 为 go 的 Agent。



agent-base

内置 Label 环境说明

Jenkins Agent Label: base

名称	类型/版本
容器名称	base
操作系统	centos-7 (7.9.2009)

名称	类型/版本
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: maven

名称	类型/版本
容器名称	maven
操作系统	centos-7 (7.9.2009)
Jdk	openjdk-1.8.0_322
Maven	3.5.3
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: mavenjdk11

名称	类型/版本
容器名称	maven

名称	类型/版本
操作系统	centos-7 (7.9.2009)
Jdk	openjdk-11.0.19
Maven	3.5.3
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: go

名称	类型/版本
容器名称	go
操作系统	centos-7 (7.9.2009)
Go	1.12.10
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
podman	podman version 3.0.1
Kubectl	v1.22.0

名称	类型/版本
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: go16

名称	类型/版本
容器名称	go
操作系统	centos-7 (7.9.2009)
Go	1.16.8
GOPATH	/home/jenkins/go
GOROOT	/usr/local/go
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

Jenkins Agent Label: node.js

名称	类型/版本
容器名称	nodejs
操作系统	centos-7 (7.9.2009)

名称	类型/版本
统	
Node	v10.16.3
Yarn	1.16.0
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

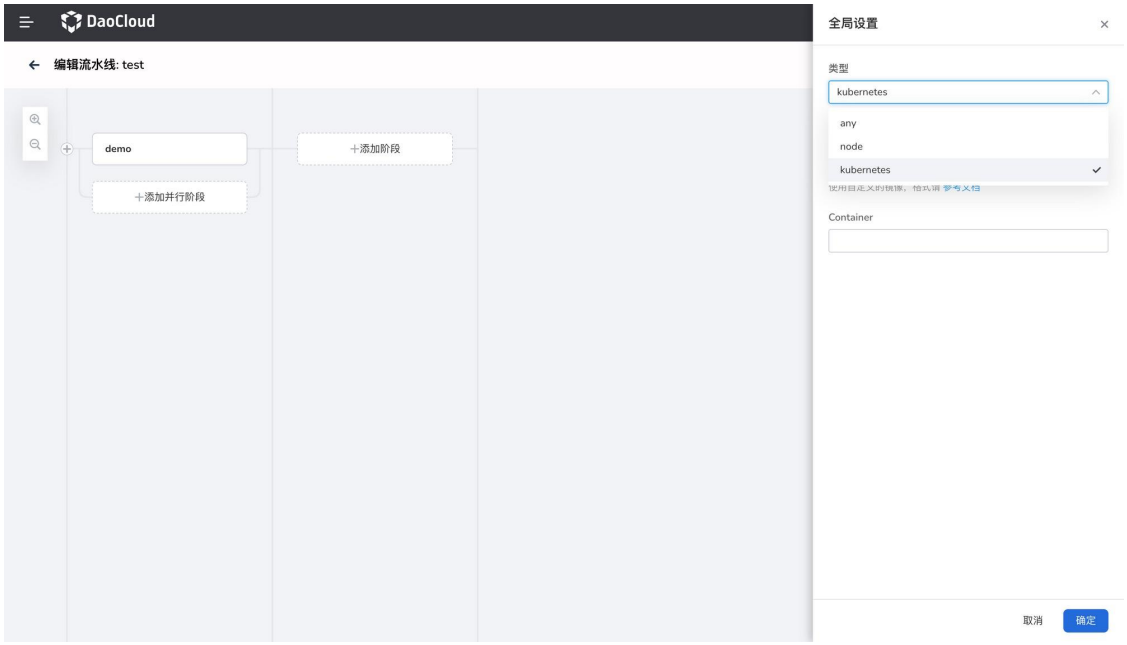
Jenkins Agent Label: python

名称	类型/版本
容器名称	python
操作系统	centos-7 (7.9.2009)
Python	3.7.11
podman	podman version 3.0.1
Kubectl	v1.22.0
内置工具	unzip、which、make(GNU Make 3.82)、wget、zip、bzip2、git (2.9.5)

使用 YAML 自定义 podTemplate

如果需要运行特定环境的 Jenkins Agent，可以在流水线上自定义 Jenkins Agent。

1. 在编辑流水线页面上选择 Agent 类型为 kubernetes。



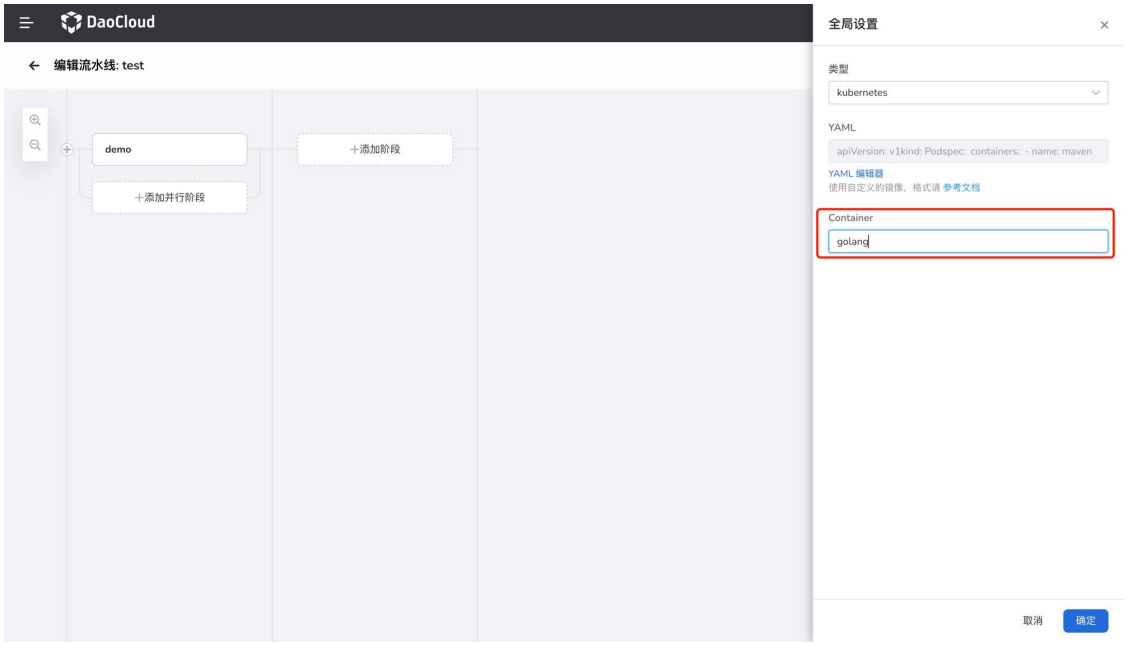
agent-kubernets

2. 点击 YAML 编辑器，在对话框中填写 YAML 语句，请参考以下示例：

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: maven
    image: maven:3.8.1-jdk-8
    command:
    - sleep
    args:
    - 99d
```

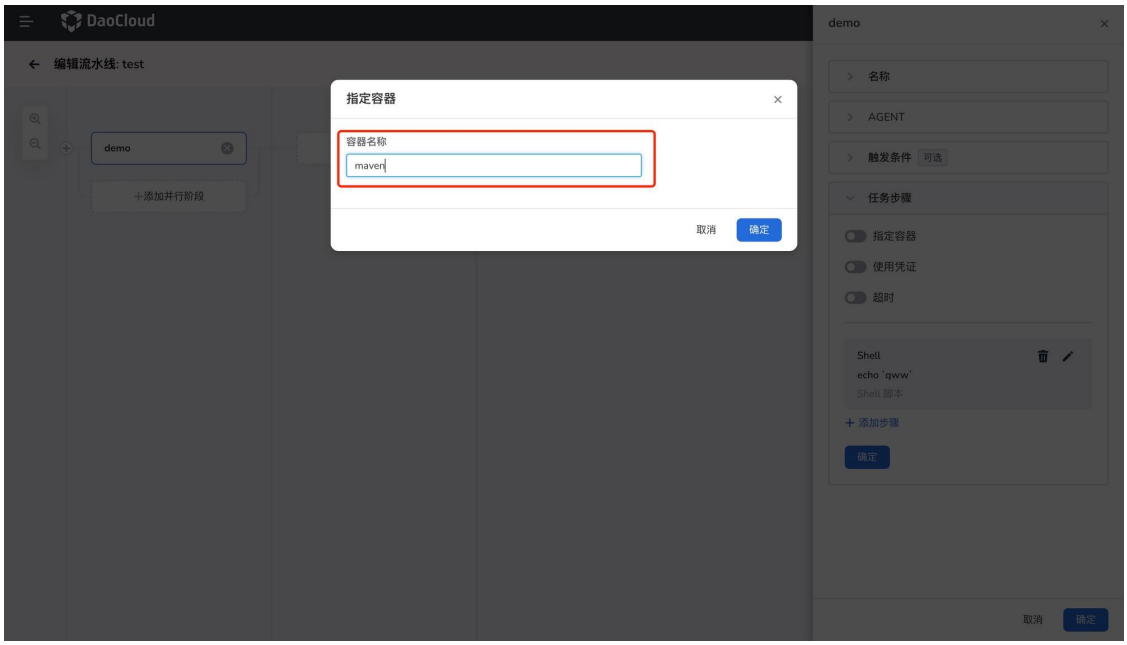
```
- name: golang
  image: golang:1.16.5
  command:
  - sleep
  args:
  - 99d
```

3. 在 Container 中输入 golang 作为流水线运行的默认容器。



agent-golang

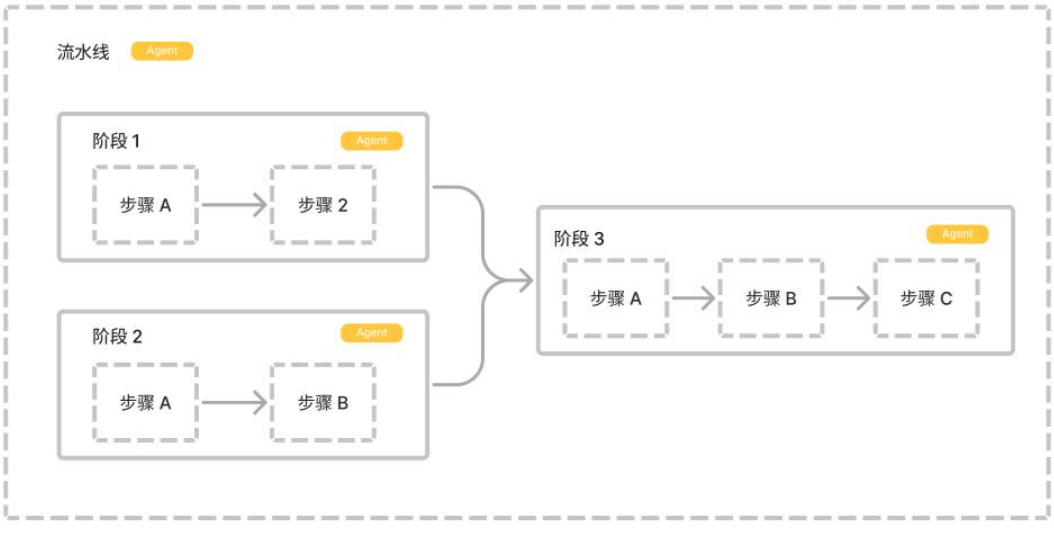
4. 要在流水线其他步骤中使用上述示例的其他容器，可以选择指定容器填写所需要的容器名称。


DaoCloud
编辑流水线: test
demo
+添加并行阶段
指定容器
容器名称
maven
取消
确定
demo
名称
AGENT
触发条件
可选
任务步骤
指定容器
使用凭证
超时
Shell
echo `qww`
Shell 脚本
添加步骤
确定
取消
确定

agent–maven

流水线中的概念

不论图形化或文本类型，编辑器本质上都是用来方便用户查看与编辑构建流程的核心：Jenkinsfile（过程描述文件）。因此在讨论编辑器之前需理解「过程描述文件」的几个重要概念。



pipeline

- 流水线

  流水线是用户定义的一个工作模型，流水线的代码定义了软件交付的完整流程，一般包含构建、测试和交付应用程序等阶段。 有关流水线语法，请参见 Jenkins 官方文档。

- Agent

  Agent 描述了整个流水线执行过程或者某个阶段的执行环境，必须出现在描述文件顶格或者每一个阶段。 有关更多信息，请参见选择 Jenkins Agent。

- 阶段

一个阶段定义了一系列紧密相关的步骤。每个阶段在整条流水线中各自承担了独立、明确的责任。 比如“Build”、“Test”或“Deploy”阶段。通常来讲，所有的实际构建过程都放置在阶段里面。 有关更多信息，请参见选择 Jenkins Stage。

- 并行阶段

  并行用来声明一些并行执行的阶段，通常适用于阶段与阶段之间不存在依赖关系的情况下，用来加快执行速度。 有关更多信息，请参见选择 Jenkins Agent。

- 步骤

  步骤列表描述了一个阶段内具体要做什么事，具体要执行什么命令。比如有一个步骤（step）需要系统打印一条构建中...的消息，即执行命令 echo '构建中...'。 有关更多信息，请参见选择 Jenkins Step。

图形化任务模板参数说明

以下为图形化流水线任务模板参数说明。

Git Clone

参数	说明
代码仓库	填写远程代码仓库的地址
分支	填写想要基于哪个分支的代码构建流水线，默认为 master
凭证	对于私有仓库，需要提前创建凭证并在使用时选择对应的凭证

Shell

如果要执行 shell 命令，请选择此模板，支持多行。

打印消息

如果要在终端输出一些消息，请选择此模板。

保存制品

参数	说明
用于归档的文件	使用正则表达式规定期望将文件存储在什么路径下，例如：module/dist/*/.zip，多个文件请用英文逗号隔开。

审核

如果添加此步骤，流水线运行至任务将会暂停，创建人和被 @ 的人可以选择继续或终止。

参数	说明
消	流水线运行状态中将会展示此消息，支持在输入 @ + 用户的方式来选择可

参数	说明
息	审核的人。

使用凭证

使用凭证组件在图形化编辑中是一个较为特殊的步骤，如果选择开启后，该阶段定义的所有步骤都将会被嵌套在使用凭证步骤中。

类型	说明
用户名和密码	需要填写以下参数：用户名变量：在流水线构建期间用户名的环境变量的名称。密码变量：在流水线构建期间密码的环境变量的名称。
访问令牌	需要填写以下参数：文本变量：在流水线构建期间文本的环境变量的名称。
kubeconfig	需要填写以下参数：kubeconfig 变量：在流水线构建期间 kubeconfig 环境变量的名称。

超时

当前任务的执行时间若超过超时时间，则会中止任务，流水线状态变为失败。

参数	说明
时间	设置超时的时间。
单位	设置时间单位，支持秒、分钟、小时、天

SVN

参数	说明
代码仓库	填写远程的 svn 的地址，例如 http://svn.apache.org/repos/asf/ant/
凭证	对于私有仓库，需要提前创建凭证并在使用时选择对应的凭证

收集测试报告

收集 JUnit 的测试报告，必须为 xml 格式，支持填写多个地址，用逗号隔开。

参数	说明
测试报告	指定生成的 xml 报告文件的位置，例如 myproject/target/test-reports/*.xml

SonarQube 配置

如果您需要使用 SonarQube 来对代码仓库中的代码进行扫描，请选择当前步骤，使用之前请确保您已经集成 SonaQube 实例并绑定到当前工作空间。

参数	说明
SonarQube 实例	下来选项中选择当前工作空间绑定的 SonaQube 实例
代码语言	由于不同代码语言对应的 SonaQube 的扫描命令不同，如果 java 语言，请选择 maven，其他语言则选择其他
项目	本次扫描在 SonaQube 中对应的项目名称
扫描文件	需要扫描代码仓库中文件目录地址

代码质量检查结果

此步骤需要配合 SonarQube 配置步骤一起使用，定义该步骤后会暂停流水线执行，等待 SonarQube 代码扫描分析完成后返回质量门禁状态，来判断是否终止运行当前流水线。

另外实际使用过程中，需要定义超时 步骤共同使用，以等待 SonarQube 代码扫描的返回结果，如下：

```
steps {
        timeout(time: 1, unit: 'HOURS') {
          waitForQualityGate abortPipeline: true
```

```
        }
    }
```

参数	说明
等待检查结果是否暂停流水线	支持 true/fasle 两个选项，如果设置为 true，当 SonarQube 代码扫描分析完成后返回质量门襟状态为不健康时，将会终止流水线

流水线模板文件

流水线模板文件按照 YAML 的格式组织，主要包含两部分：parameterDefinitions 和 jenkinsfileTemplate。

- parameterDefinitions 区域：定义流水线模板暴露哪些参数。支持布尔值、下拉列表、凭证、密码、文本等多种参数类型。
- jenkinsfileTemplate 区域：定义 Jenkins 流水线的 jenkinsfile，使用 go template 作为模板引擎渲染，可以引用 parameterDefinitions 中暴露的参数。

parameterDefinitions 区域

字段	类型	说明	默认值	是否必填
name	string	参数名称，命名需遵循 go template specification，不允许重复，正则“[14][a-zA-Z0-9_]*$”	–	必填
displayName	[]byte	表单上显示的名称，长度小于 24 个字符	“”	非必填
description	string	参数的描述	“”	非必填
default	json.Value	为对应的参数	nil	非必填

[14] a-zA-Z_

字段	类型	说明	默认值	是否必填
		设置默认值		
type	string	参数的类型，支持布尔值、下拉列表、凭证、密码、文本	string	非必填

支持的参数类型

- boolean：布尔值，默认值只能为 true 或 false，在 UI 上是一个 checkbox
- string: 字符串，输入框
- text：文本，文本框
- password：字符串，密码输入框
- choice：在 UI 上是一个下拉列表，必须按列表的格式提供默认值，例如

```
type: choice
default:
  - choice 1
  - choice 2
```

- credential：凭证，在 UI 上是一个下拉列表，会获取当前工作空间下的凭证列表

jenkinsfileTemplate 区域

整体上依然遵循 Jenkinsfile 的语法，但是可以使用 go template 作为模板引擎，基于 parameterDefinitions 中定义的参数渲染。

变量

通过 {{ .params.<name> }} 的方式引用上文定义的参数，例如 {{ .params.gitCloneURL }}。

条件语句

支持以下三种格式的条件语句：

{{**if** pipeline}} T1 {{end}}

如果 pipeline 的值是空，不会输出任何内容；否则，执行 T1。

空值在模板中主要是 false 的布尔值和长度为零的字符串两种情况。

{{**if** pipeline}} T1 {{**else**}} T0 {{end}}

如果 pipeline 的值是空，执行 T0；否则，执行 T1。

{{**if** pipeline}} T1 {{**else if** pipeline}} T0 {{end}}

为了简化 if-else 链的外观，if 的 else 动作可以直接包含另一个 if；效果与写法完全相同

{{**if** pipeline}} T1 {{**else**}}{{**if** pipeline}} T0 {{end}}{{end}}

循环语句

支持以下两种格式的循环语句：

{{**range** pipeline}} T1 {{end}}

pipeline 的值必须是数组、切片、map 或者 channel。

我们暂时不支持定义上述类型的 Parameter。

{{**range** pipeline}} T1 {{**else**}} T0 {{end}}

如果 pipeline 的长度是空，执行 T0；否则，执行 T1。

其他

上文描述的只是部分 go template 的语法，除此以外，你还能使用管道符|，函数（例如 printf,），注释等，更多语法请参考 go template。

模板文件示例

```yaml
parameterDefinitions:
  - name: gitCloneURL
    displayName: 代码仓库地址
    description: The git clone url of the source code
    type: string
  - name: gitRevision
    displayName: 代码仓库分支
    description: The git revision of the source code
    type: string
    default: master
  - name: gitCredential
    displayName: 凭证
    description: The credential to access the source code
    type: credential
    default: ""
  - name: testCommand
    displayName: 测试命令
    description: The command to run the test
    type: string
    default: go test -v -coverprofile=coverage.out
  - name: reportLocation
    displayName: 测试报告位置
    description: The location of the test report
    type: string
    default: ./target/**
  - name: dockerfilePath
    displayName: Dockerfile 路径
    description: The path of the Dockerfile
    type: string
    default: .
  - name: image
```

```
    displayName: 目标镜像地址
    description: The target image to build
    type: string
  - name: tag
    displayName: tag
    description: The tag of the target image
    type: string
    default: latest
  - name: registryCredential
    displayName: 镜像仓库凭证
    description: The credential to access the image registry
    type: credential
    default: ""
jenkinsfileTemplate: |
  pipeline {
    agent {
      node {
        label 'go'
      }
    }
    environment {
      IMG = '{{.params.image}}:{{.params.tag}}'
    }
    stages {
      stage('clone') {
        steps {
          container('go') {
            git(url: '{{ .params.gitCloneURL }}', branch: '{{ .params.gitRevision }}',
 credentialsId: '{{ .params.gitCredential }}')
          }
        }
      }
```

```
        stage('test') {
          steps {
            container('go') {
              sh '{{ .params.testCommand }}'
              archiveArtifacts '{{ .params.reportLocation }}'
            }
          }
        }
        stage('build') {
          steps {
            container('go') {
              sh 'docker build -f {{.params.dockerfilePath}} -t $IMG .'
            {{- if .params.registryCredential }}
              withCredentials([usernamePassword(credentialsId: '{{ .params.registry
Credential }}', passwordVariable: 'PASS', usernameVariable: 'USER',)]) {
                sh 'docker login {{ .params.image }} -u $USER -p $PASS'
                sh 'docker push $IMG'
              }
            {{- else }}
              sh 'docker push $IMG'
            {{- end }}
            }
          }
        }
      }
```

创建自定义流水线模板

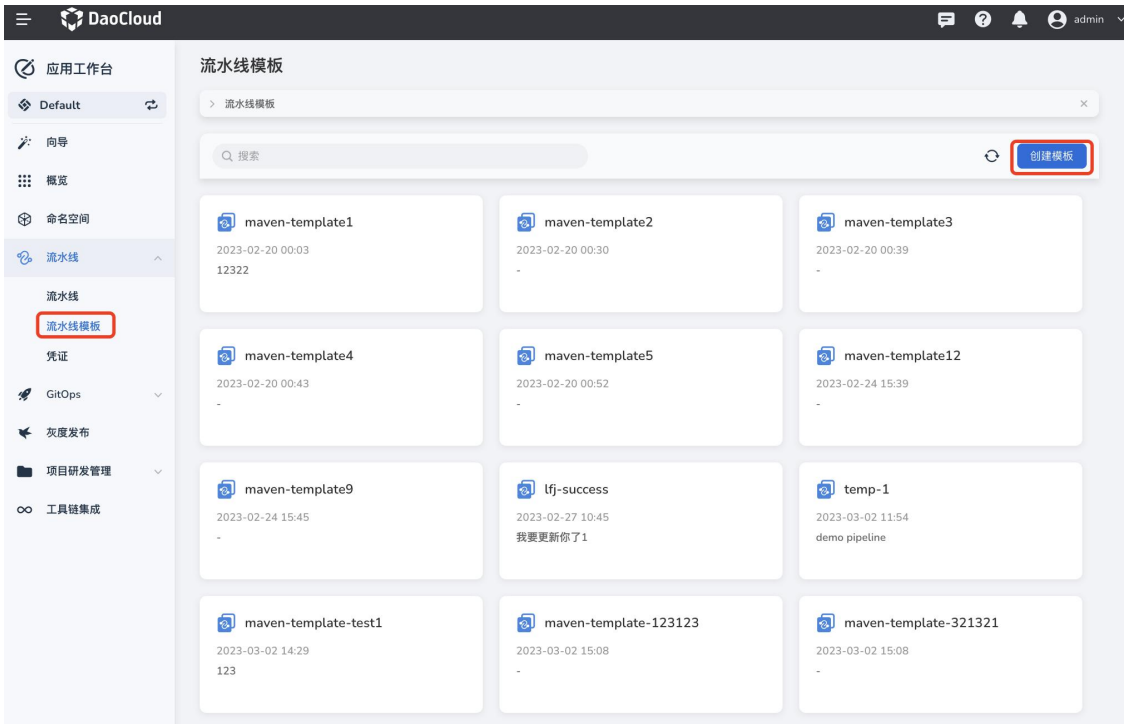
应用工作台模块支持自定义流水线模板，用户可以根据真实业务场景制作流水线模板，然后基于自定义模板快速创建符合要求的流水线。

前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 workspace editor 或更高权限。
- 配置好流水线模板文件。

操作步骤

1. 在自定义流水线模板页面点击创建模板

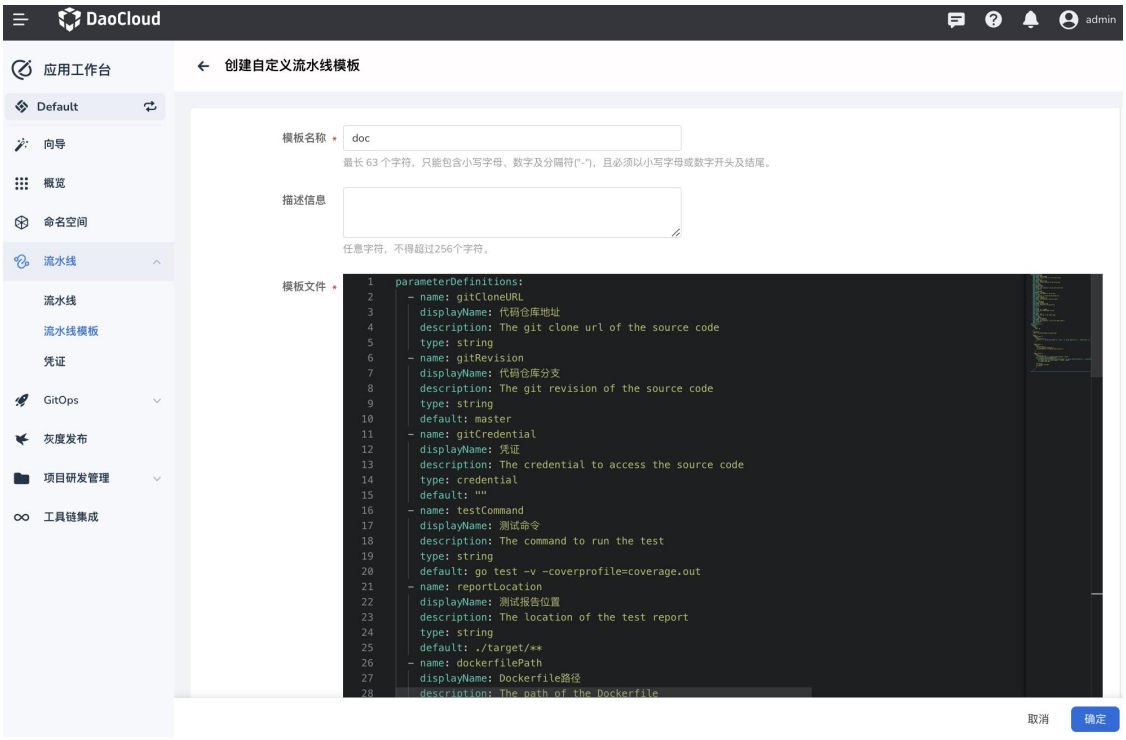


click-create

2. 参考以下说明填写基本信息，然后点击确定 。

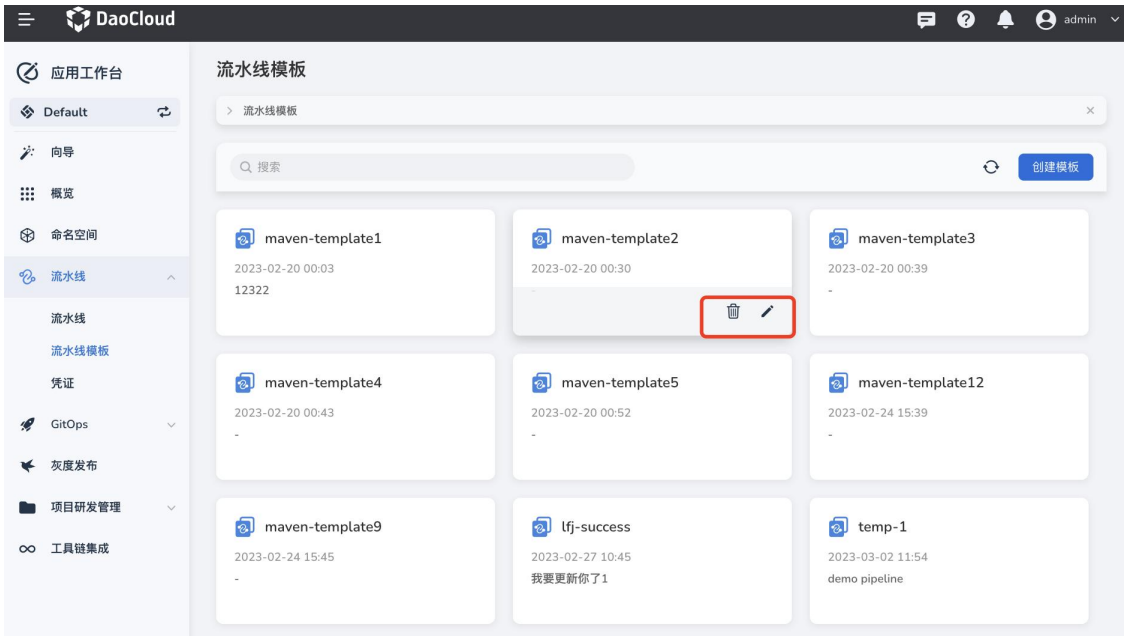
    - 模板名称：填写流水线模板的名称。

- 描述信息：对当前流水线的描述信息，支持中文。
- 模板文件：参考流水线模板文件填写或粘贴事先准备好的参数。



config

3. 返回流水线模板列表页面，点击模板卡片可以删除或修改模板。

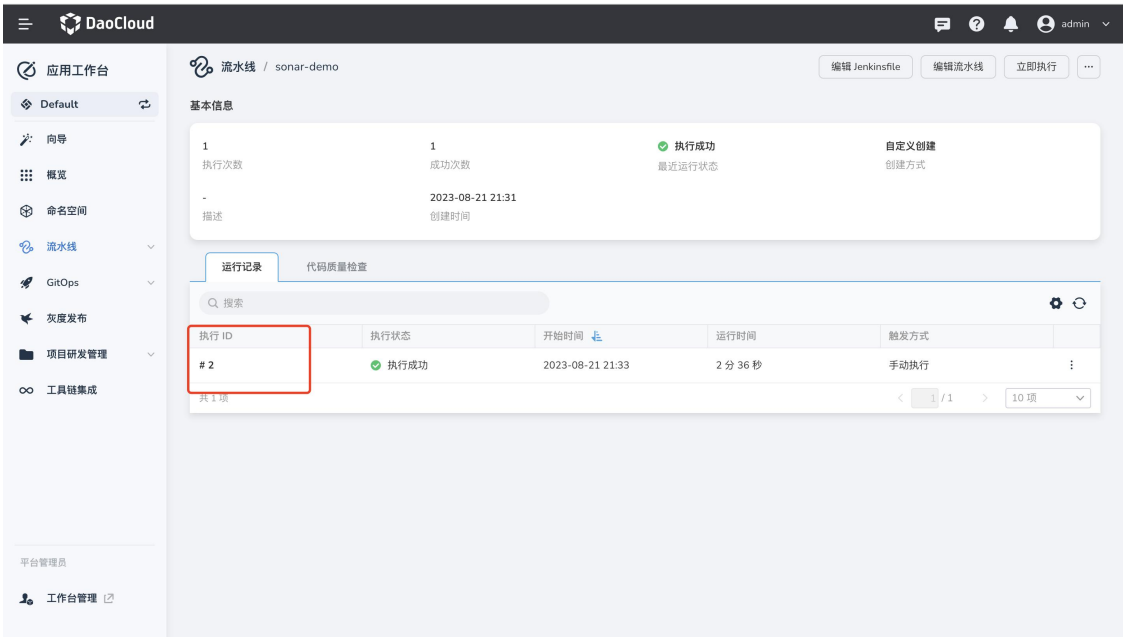

DaoCloud
admin
应用工作台
Default
向导
概览
命名空间
流水线
流水线
流水线模板
凭证
GitOps
灰度发布
项目研发管理
工具链集成
流水线模板
流水线模板
搜索
创建模板
maven-template1
2023-02-20 00:03
12322
maven-template2
2023-02-20 00:30
maven-template3
2023-02-20 00:39
-
maven-template4
2023-02-20 00:43
-
maven-template5
2023-02-20 00:52
-
maven-template12
2023-02-24 15:39
-
maven-template9
2023-02-24 15:45
-
lfj-success
2023-02-27 10:45
我要更新你了1
temp-1
2023-03-02 11:54
demo pipeline

config

查看流水线运行记录

创建好流水线之后并运行过流水线，可以查看其流水线运行记录详情。

在应用工作台–>流水线 → 进入流水线详情页面，选择某一条流水线运行记录，点击执行 ID。



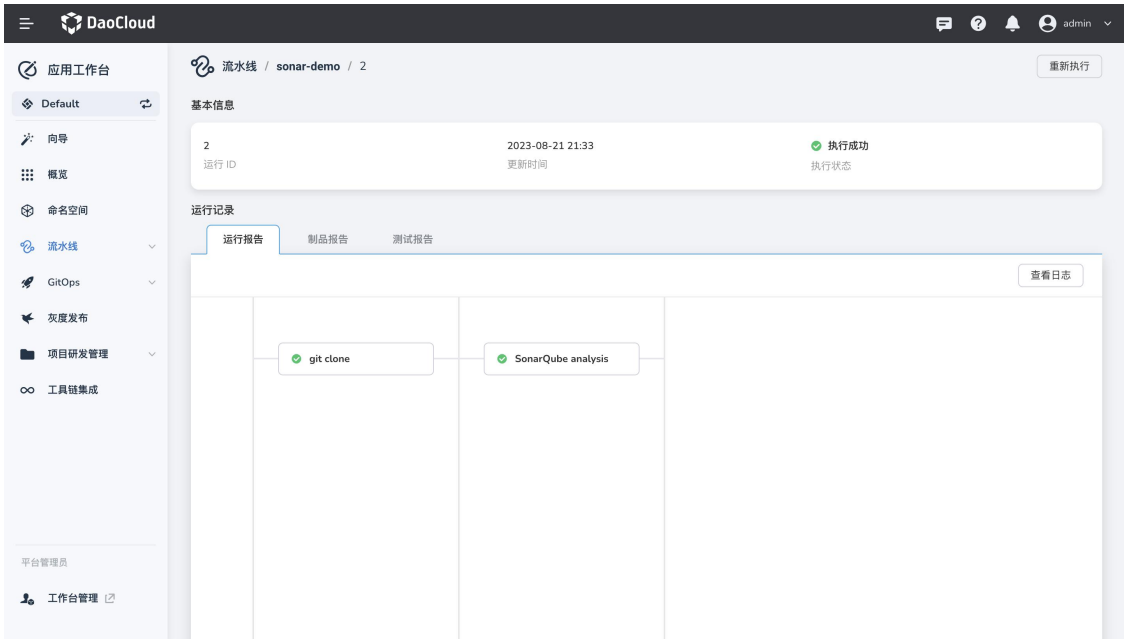
detail–run1

进入到流水线运行记录详情页面，可以查看到当前流水线运行的基本信息。

- 运行 ID：当前流水线运行记录的 ID，与流水线的第几次执行一致。
- 更新时间：当前流水线运行的时间
- 执行状态：当前流水线运行的状态，主要有以下状态：

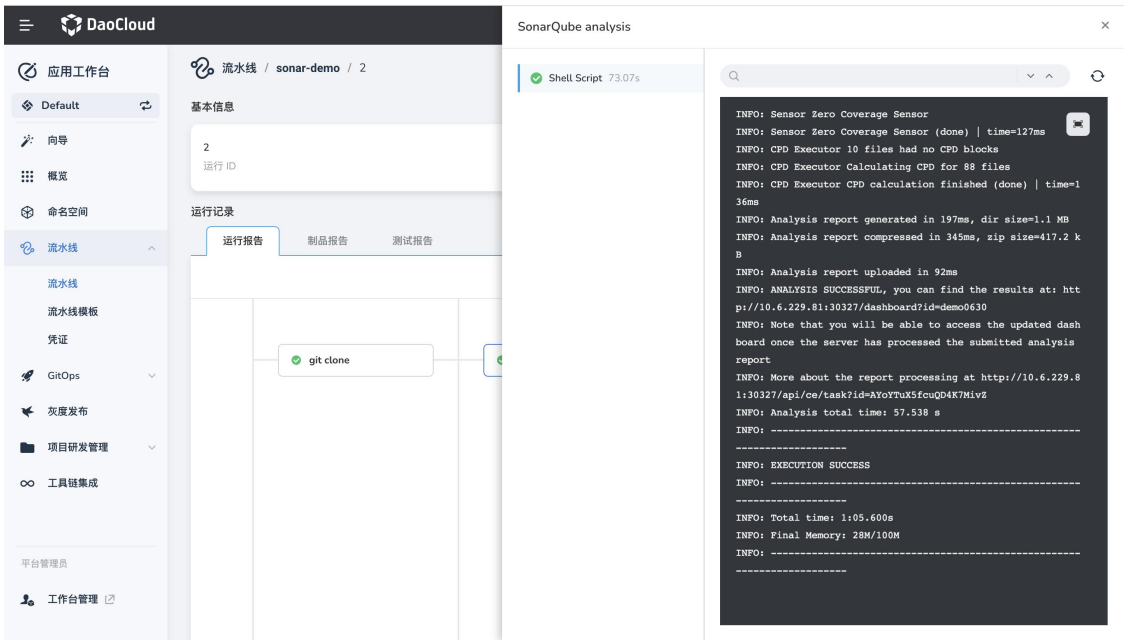
状态	描述
执行成功	成功执行当前流水线所有的阶段
执行失败	流水线中存在阶段运行失败

状态	描述
执行中	流水线正在执行中
队列中	流水线任务已经下发，等待分配容器执行流水线任务
终止	人为终止运行当前流水线



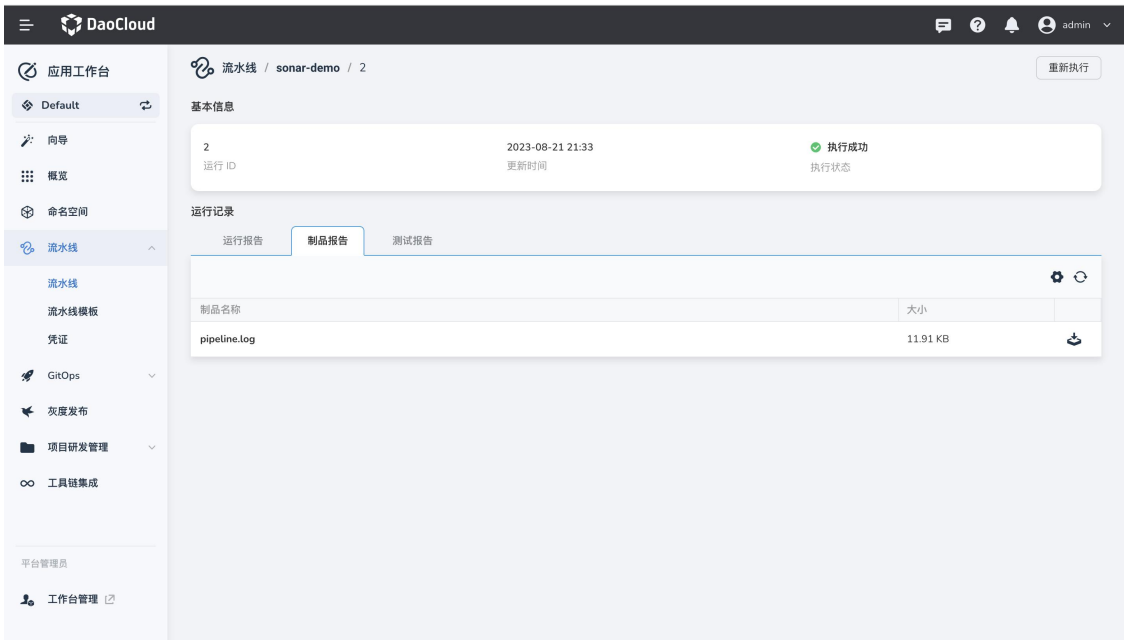
detail-run2

查看当前流水线记录的运行报告。点击流水线下的某个步骤，可查看相应步骤的执行日志及运行时间，也可查看全部日志。



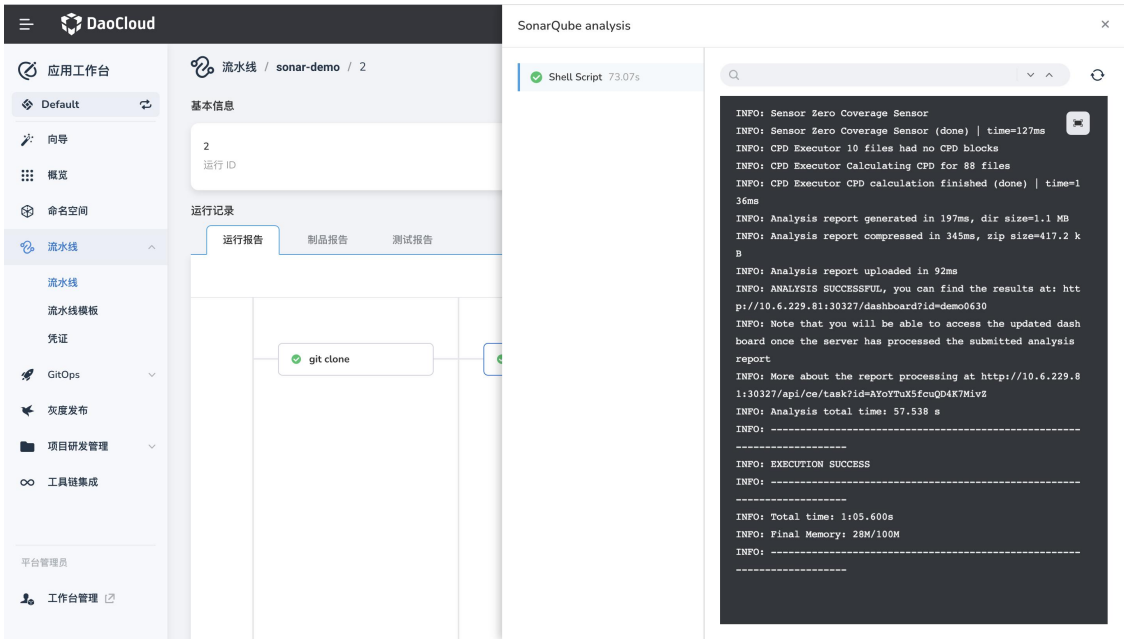
detail–run5

查看当前流水线记录的制品报告。当流水线中定义了保存制品 步骤，并且流水线执行该步骤后，即可缓存流水线定义中的制品，并支持用户进行下载，其中流水线的日志会默认保存为制品，如下图：



detail–run3

查看当前流水线记录的测试报告。当流水线中拉取了远程代码并且定义了收集测试报告 步骤，并且流水线执行该步骤后，即可在当前界面查看到当前查看到测试报告的信息，如下图：

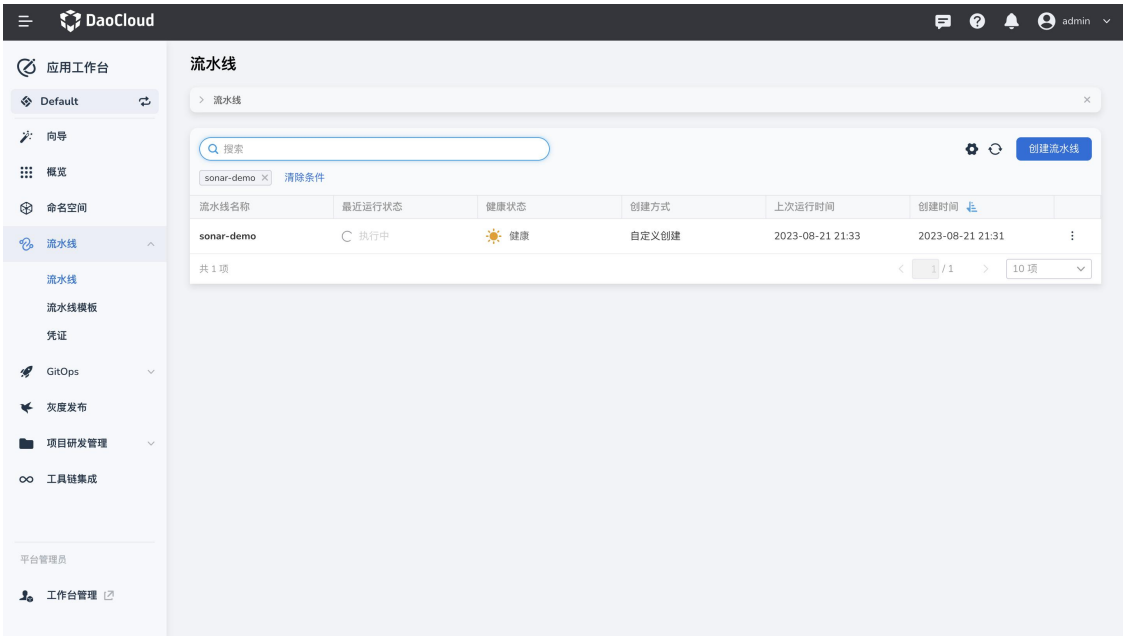


detail-run5

查看流水线详情

创建好流水线之后，可以根据需要查看流水线详情或更新配置，流水线详情页面包含流水线的运行记录、代码质量扫描。

在应用工作台–>流水线页面，选择某一条流水线，点击流水线的名称。

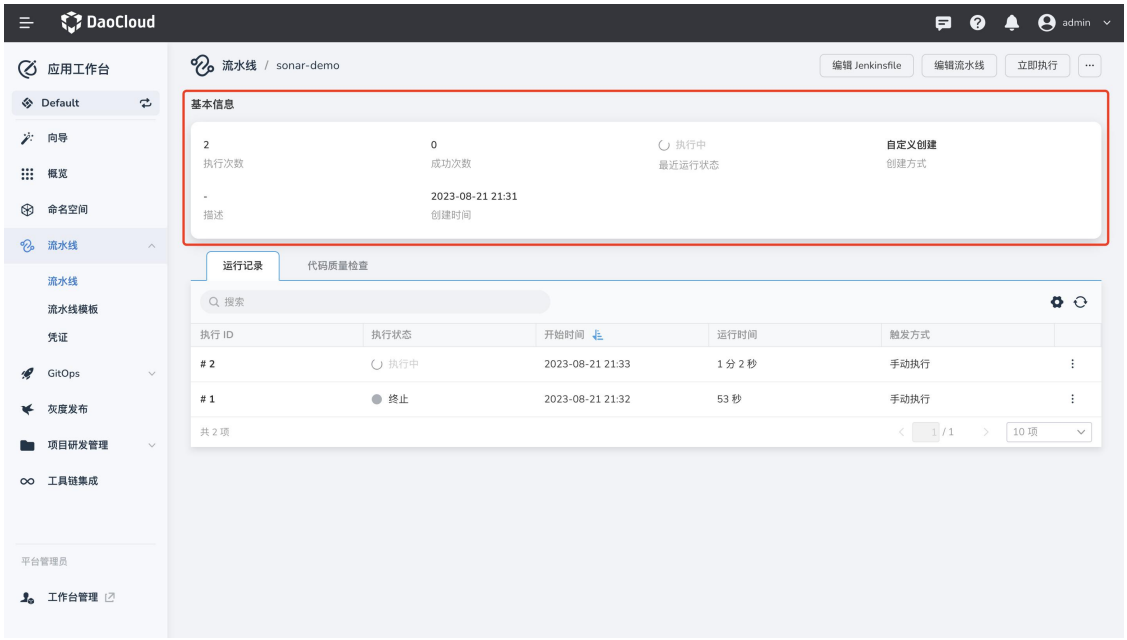


detail–pipeline1

进入到流水线详情页面，可以查看到当前流水线的基本信息。

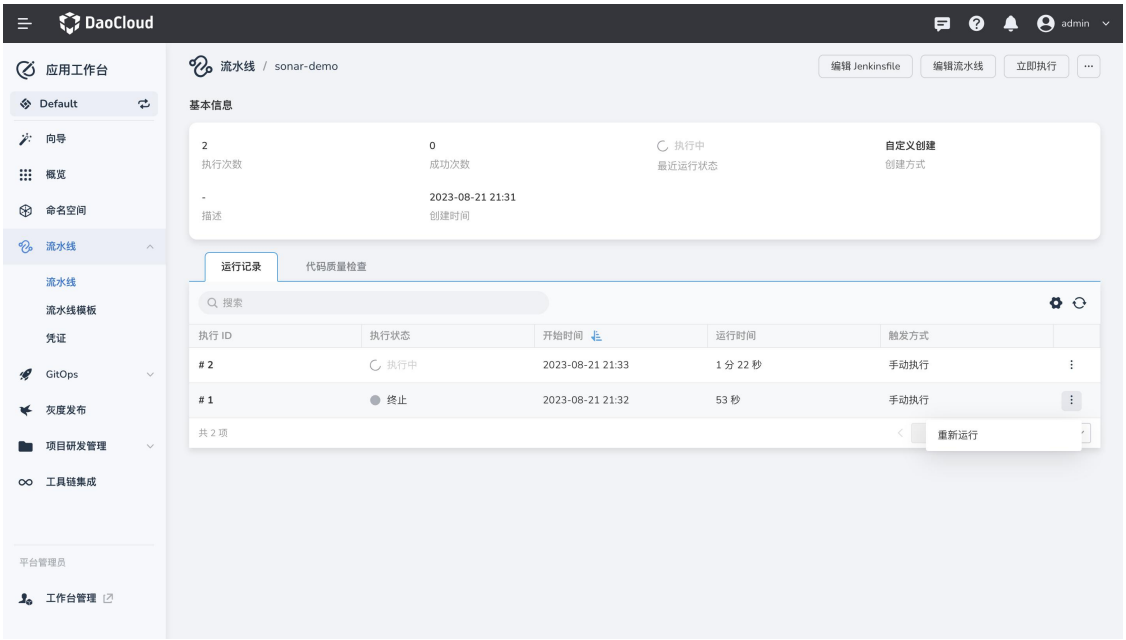
- 执行次数：流水线运行的总次数
- 成功次数：流水线运行成功的次数
- 最近运行状态：流水线最新一条运行记录的状态
- 创建方式：流水线的创建方式，有自定义创建、多分支创建、模版创建、基于代码仓库 Jenkinfile 创建四种创建方式
- 描述：流水线的具体描述

- 创建时间：流水线的创建时间



detail–pipeline2

查看流水运行记录列表，展示当前流水线运行的记录的列表，支持通过 ID 搜索，还支持取消、重新运行操作。

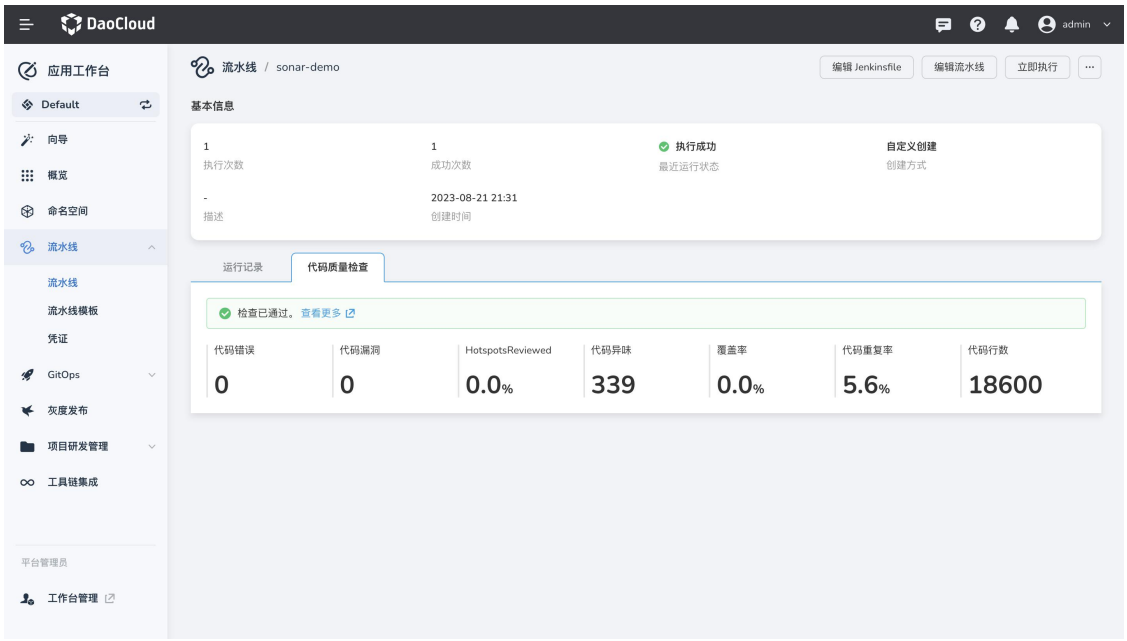


detail–pipeline3

查看流水线代码质量扫描结果。查看结果需要确保完成以下要求：

- 请确保当前工作空间已经集成或绑定了 SonarQube 实例
- 当前流水线定义了 SonarQube 配置步骤
- 成功执行流水线

完成上述要求后，您可以在详情界面查看到扫描结果，点击查看更多可以跳转到 SonarQube 后台查看更详细的结果，如下图：



detail-pipeline4

触发器触发流水线

为流水线配置触发器，周期性地自动触发流水线的执行。在编辑流水线时点击编辑配置，即可配置构建触发器。



trigger01

触发器类型

触发器类型	说明
代码源触发器	在指定的时间检查代码仓库是否有变更。例如填写 H/15 * * * * 表示每隔 15 分钟系统会自动检查代码仓库是否有新提交的变更，若有新提交的变更则会执行流水线。
定时触发器	在指定的时间触发流水线，开启定时触发器后，支持直接选择预设的规则，也可以选择自定义 CRON 表达式，输入自定义触发器规则。

触发规则

触发规则语法遵循 CRON 的语法（但有细微差别）。具体来说，每行由 5 个 Tab 或空格分隔的字段组成：MINUTE HOUR DOM MONTH DOW

字段	说明	取值范围
MINUTE	分钟	0 ~ 59
HOUR	小时	0 ~ 23
DOM	天	1 ~ 31
MONTH	月	1 ~ 12
DOW	星期	1 ~ 6 代表周一至周六，0、7 代表周日

要为一个字段指定多个值，可以按照优先顺序使用以下运算符：

运算符	说明
*	表示匹配取值范围内的所有值
M–N	表示指定范围内的所有值
M–N/X 或 */X	表示在指定范围或整个有效范围内每隔 X 触发一次
A,B,…,Z	表示匹配多个值

为了使定期调度的任务在系统上产生均匀的负载，应尽可能使用符号 H（代表“哈希”）。 例如，对十几个日常工作使用 0 0 * * * 将导致午夜时出现较大的峰值，可能会导致资源紧张。 相比之下，使用 H H * * * 仍然会每天执行一次每个作业，但不是同时执行一次，这样能够更好地使用有限的资源。

H 可以与范围一起使用。例如，H H(0–7) * * * 表示 00:00 到 7:59 之间的某个时间。

由于不同的月份天数不同，所以当 DOM 中出现 /3 或 H/3 等短周期时，在大多数月份结束时不会触发。 例如，*/3 将在每个月的第 1 天、第 4 天、……、第 31 天触发任务。如果下个月有 30 天，最后一次触发任务的时间为第 28 天。

另外，@yearly、@annually、@monthly、@weekly、@daily、@midnight 和 @hourly 作为方便的别名。 这些使用哈希进行自动匹配。例如，@hourly 与 H * * * * 相同，可以表示一小时内的任何时间。 @midnight 代表每天 0:00 ~ 2:59 之间的一段时间。

规则举例

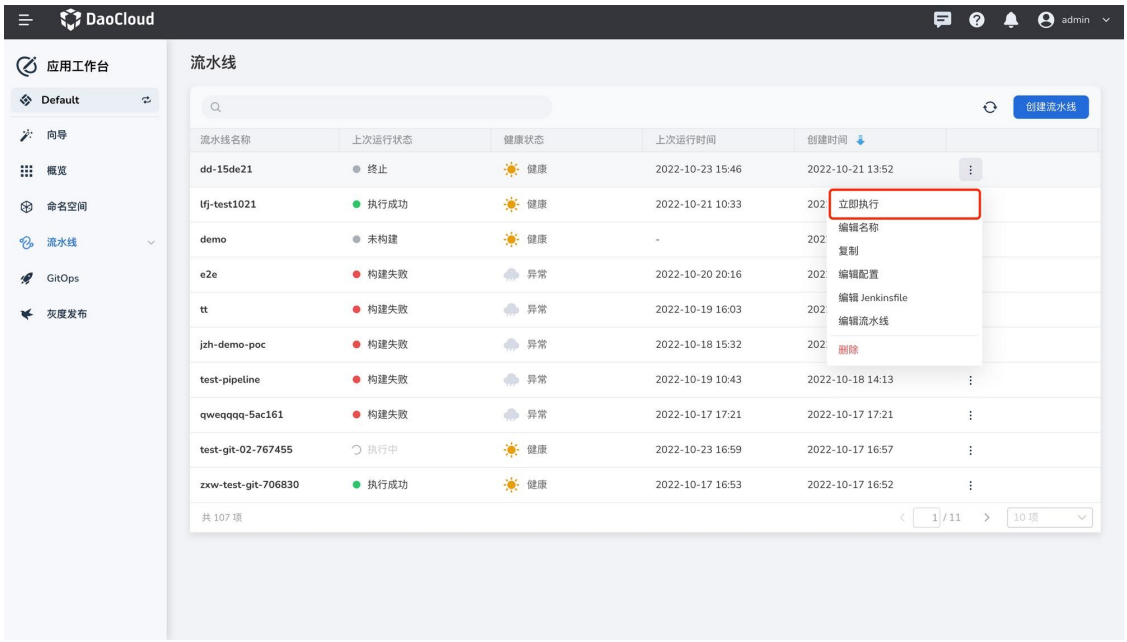
规则	说明
H/15 * * * *	表示每隔 15 分钟触发一次，例如每小时的 07 分、22 分、37 分和 52 分
H(0–29)/10 * * * *	表示在每个小时的前半小时中，每 10 分钟触发一次，例如每小时的 04 分、14 分和 24 分
45 9–16/2 * * 1–5	表示每个工作日从 9 点 45 分开始下午 15 点 45 分结束，每隔 2 小时，在第 45 分钟触发一次
H H(8–15)/2 * * 1–5	表示每个工作日从 8 点到 16 点这段时间内，每隔 2 小时之间的某刻触发一次，例如在 9:38、11:38、13:38、15:38
H H 1,15 1–11	表示每月 1 日和 15 日每天一次，除 12 月外

手动执行流水线

本页介绍在图形界面上手动执行流水线，包括立即执行、重新运行和取消运行。

立即执行流水线

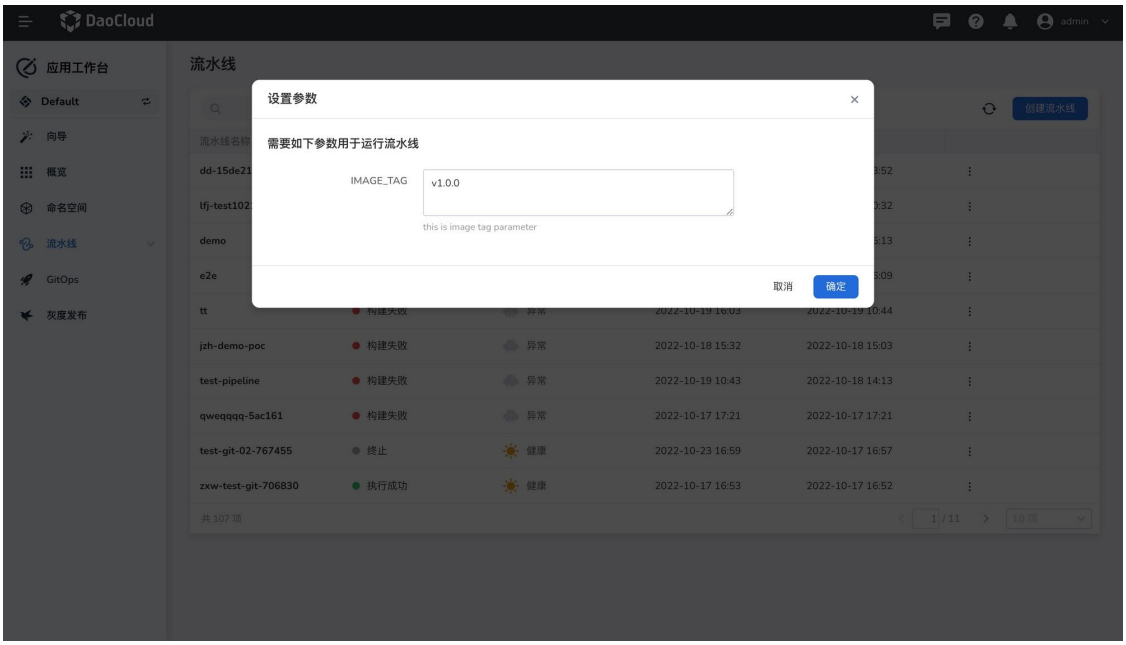
1. 在流水线列表页选择某一个流水线，点击 ： ，在弹出的菜单中点击立即执行 。



handrun01

2. 根据流水线是否配置了构建参数，执行后会出现一下两种情况：

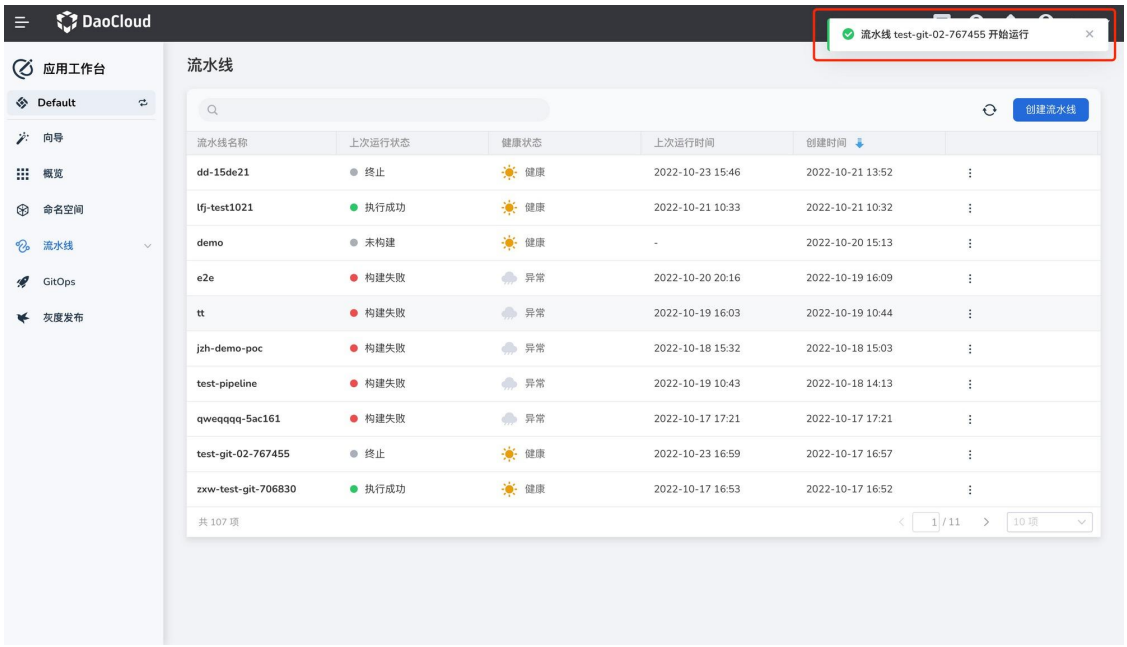
   - 如果配置了构建参数，则会出现对话框并展示相关内容进行参数配置。



handrun02

- 如果未配置构建参数，则流水线会立即执行。

3. 流水线开始执行。

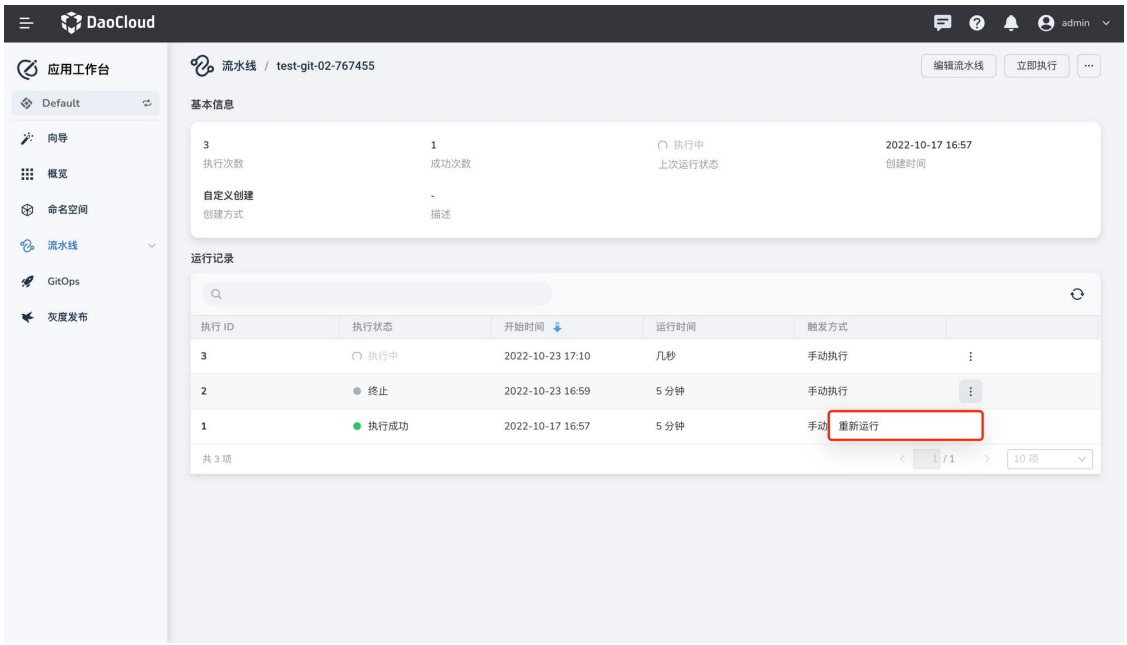


handrun03

重新运行流水线

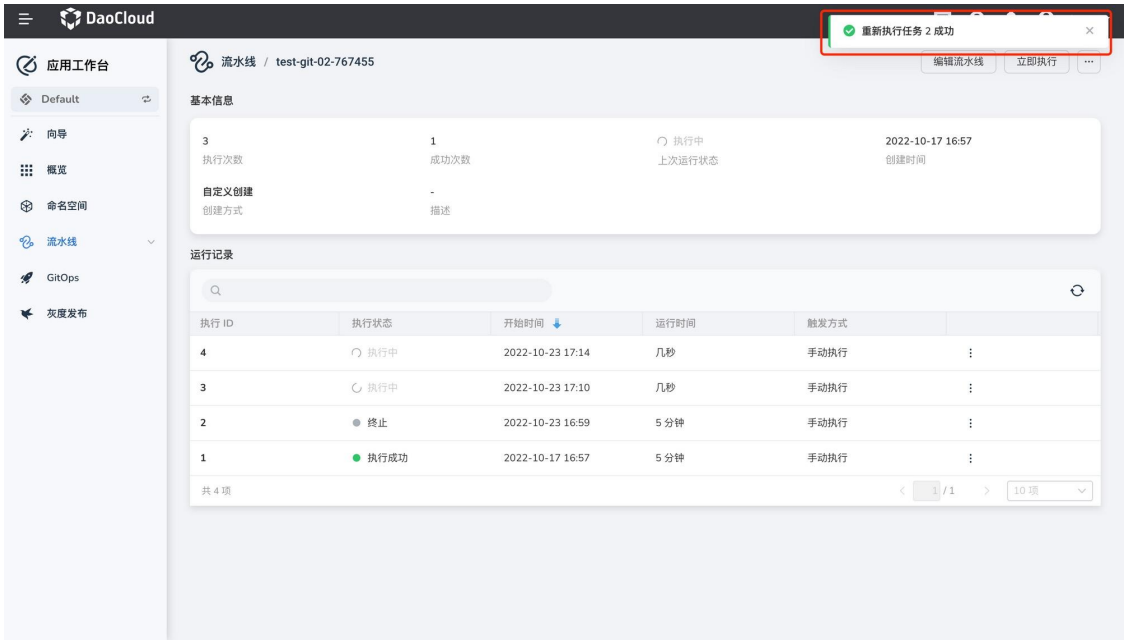
在流水线详情页面，根据流水线运行记录中的执行 ID 可以重新运行已运行过的流水线。

1. 在流水线列表中，点击某一个流水线的名称，进入流水线详情页面。
2. 在运行记录区域，找到需要重新运行的执行 ID。
3. 点击右侧的 ⋮，在弹出的菜单中点击重新运行。



handrun04

4. 流水线重新运行成功。

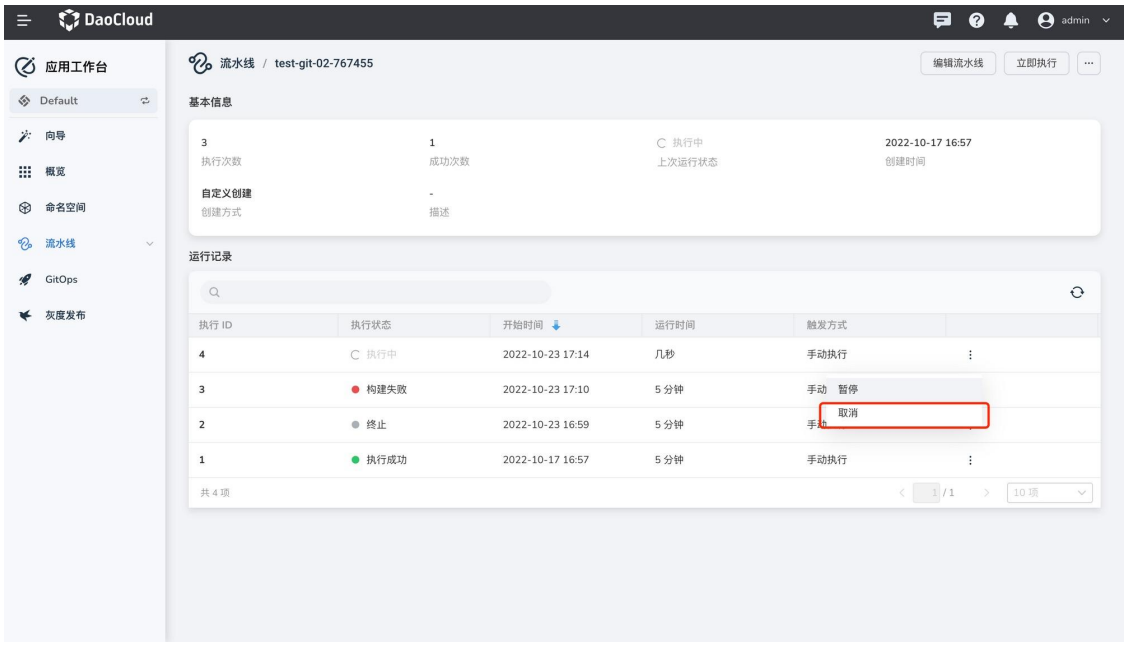


handrun05

取消运行流水线

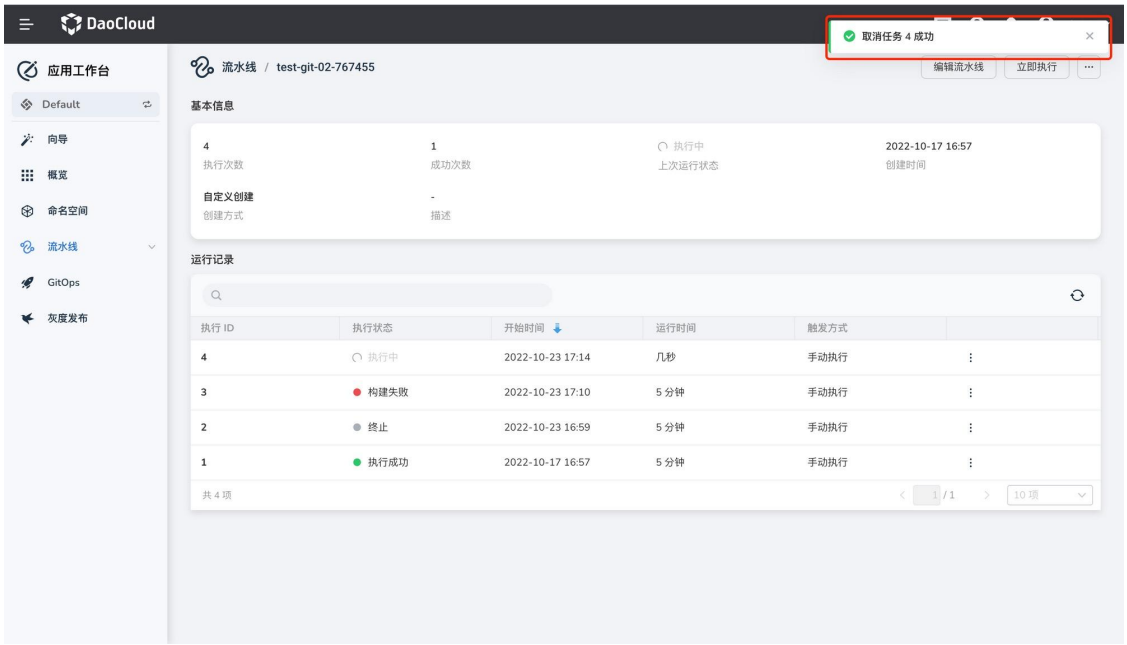
在流水线详情页面，根据流水线运行记录中的执行 ID 可以取消正在执行的流水线。

1. 在流水线列表中，点击某一个流水线的名称，进入流水线详情页面。
2. 在运行记录区域，找到需要取消的执行 ID。
3. 点击右侧的 ：，在弹出的菜单中点击取消（适用于状态为执行中的流水线）。



handrun06

4. 流水线取消运行成功。



handrun07

基于 Jenkinsfile 创建流水线

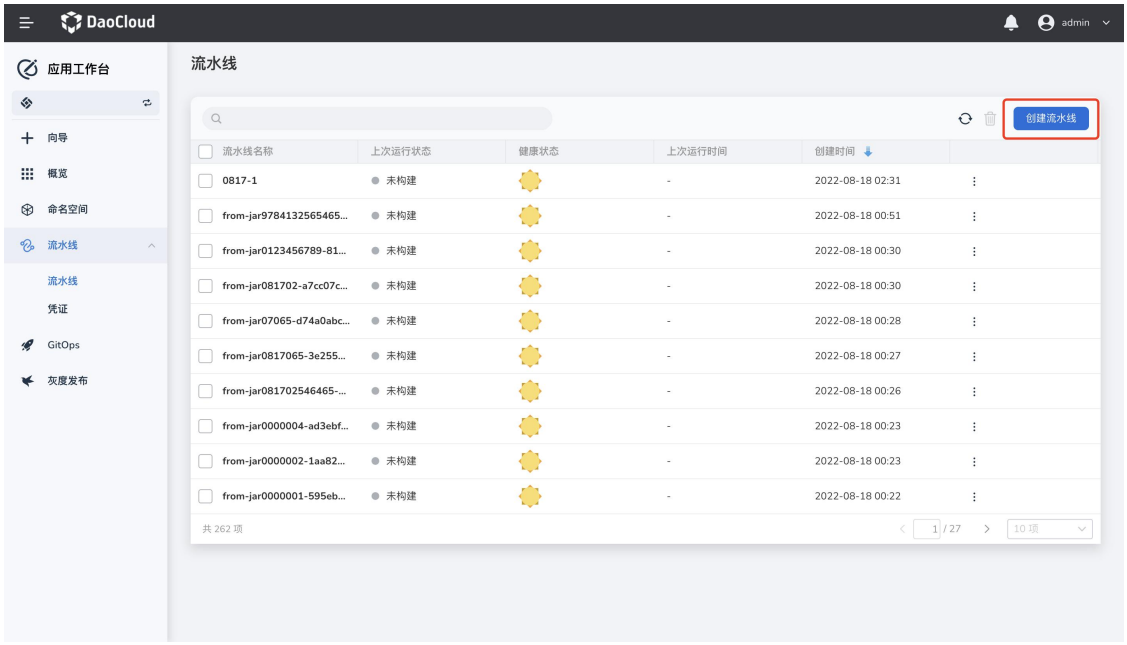
应用工作台流水线支持使用代码仓库中的 Jenkinsfile 创建流水线。

前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 workspace editor 或更高权限。
- 提供一个代码仓库，并且代码仓库的源码有一个 Jenkinsfile 文本文件。
- 如果是私有仓库，需要事先创建仓库访问凭证。

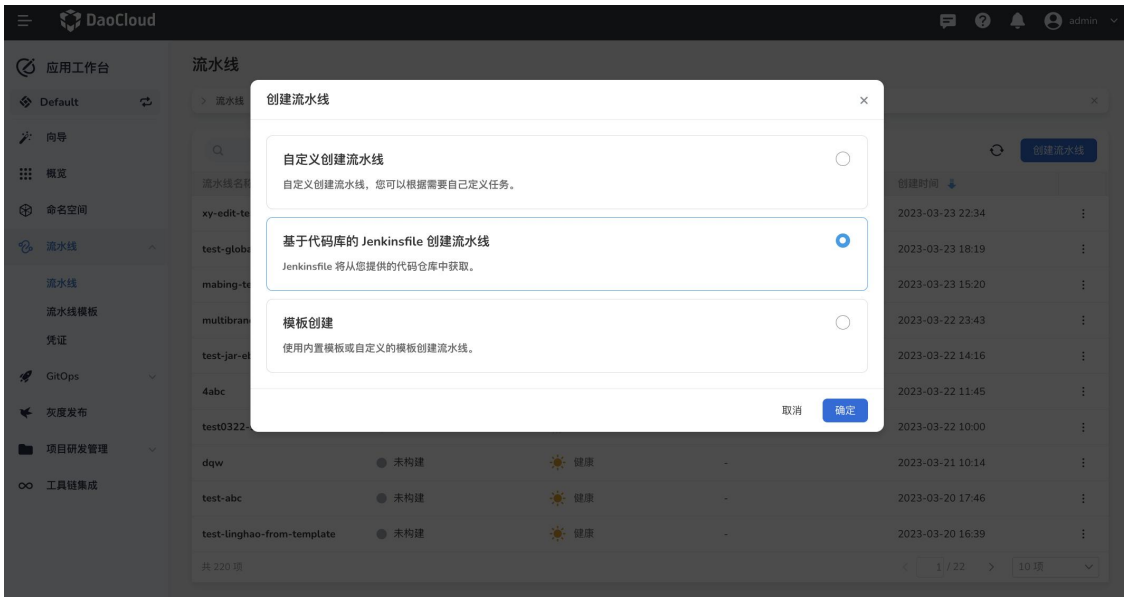
具体操作步骤如下：

1. 在流水线列表页点击创建流水线。



click-create

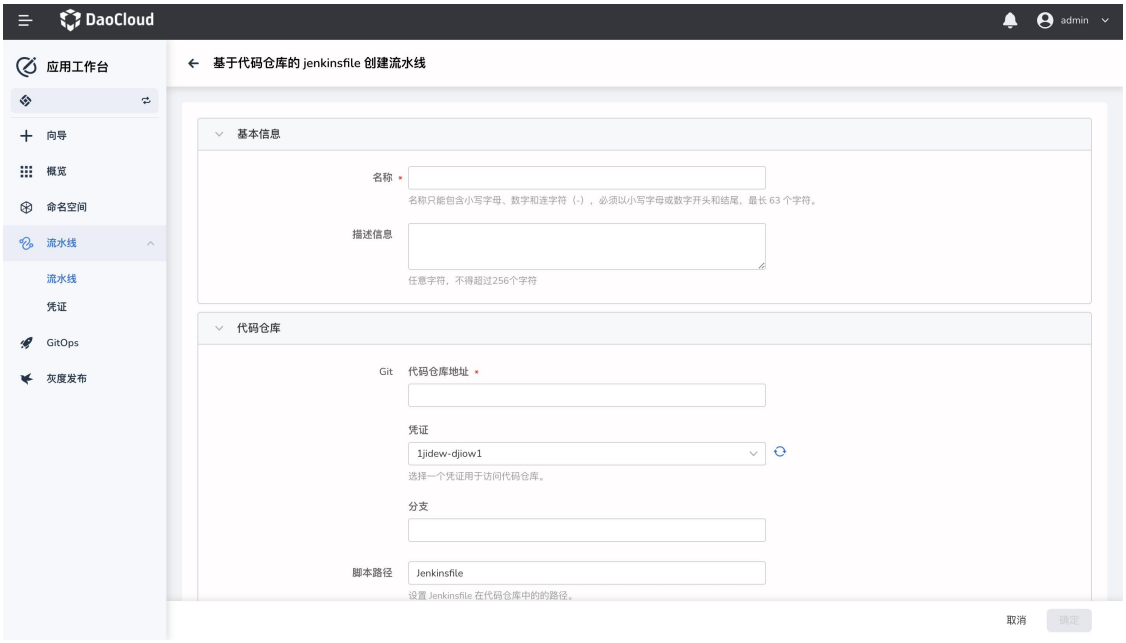
2. 选择基于代码库的 Jenkinsfile 创建流水线，点击确定。



select-type

3. 参考下方说明填写基本信息和代码仓库信息。

   - 名称：流水线的名称。同一个工作空间下流水线名称必须唯一。
   - 代码仓库地址：填写远程代码仓库的地址。
   - 凭证：对于私有仓库，需要提前创建仓库访问凭证并在此处选择该凭证。
   - 分支：基于哪个分支的代码构建流水线，默认为 master 分支。
   - 脚本路径：Jenkinsfile 文件在代码仓库中的绝对路径。



pipeline01

4. 参考下方说明填写构建设置和构建参数。

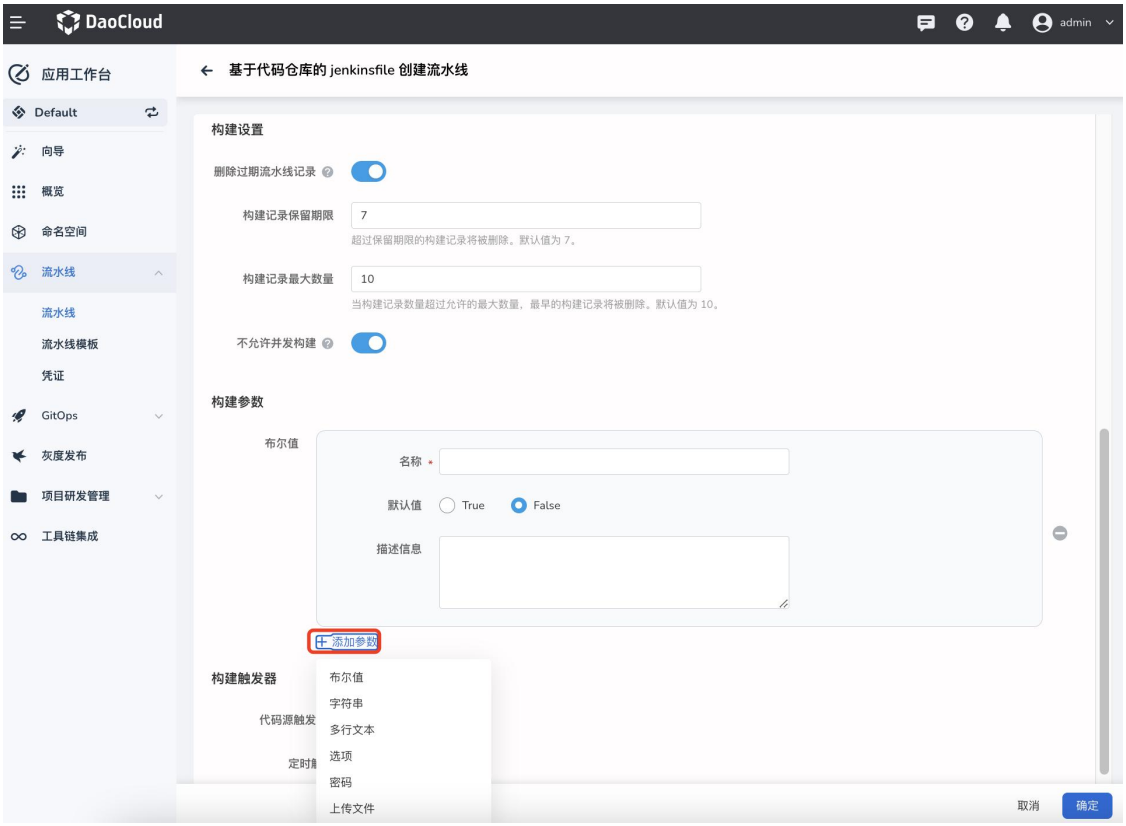
   - 删除过期流水线记录：删除之前的构建记录，以节省 Jenkins 使用的磁盘空间。

     - 构建记录保留期限：最多保留几天的构建记录，默认值为 7 天，即七天前的构建记录将被删除。
     - 构建记录最大数量：最多保留几条构建记录，默认值为 10，即最多保留 10 条记录。超过 10 条记录时，时间最早的记录最先被删除。
     - 保留期限和最大数量这两条规则同时生效，只要满足其中之一就会开始删除记录。

   - 不允许并发构建：开启后，一次只能执行一个流水线构建任务。

   - 构建参数：在开始运行流水线时传入一个或多个构建参数。默认提供五种参数类型：布尔值、字符串、多行文本、选项、密码、上传文件。

- 添加构建参数后，运行流水线时需要为每个构建参数输入对应的取值。



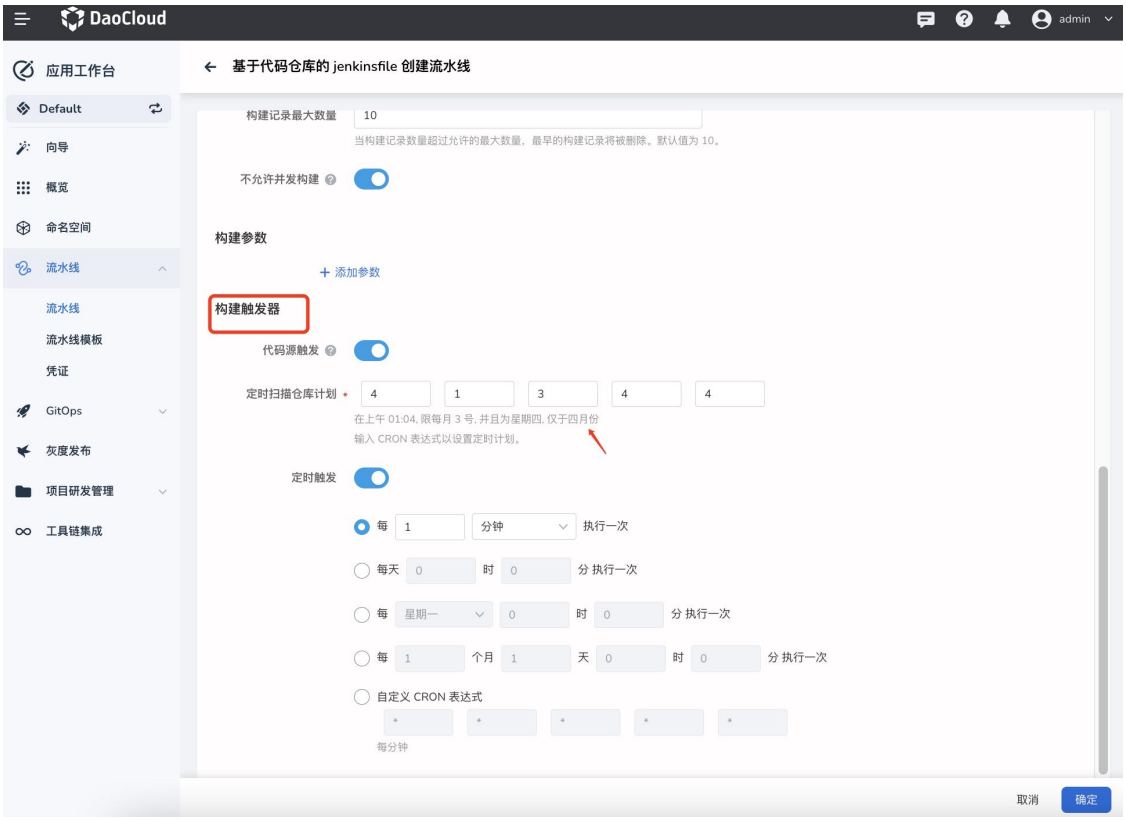
pipeline02

5. 参考下方说明填写构建触发器。

   - 代码源触发：开启后，系统会根据定时仓库扫描计划定期扫描仓库代码中用于构建流水线的特定分支，如果有更新内容，则重新运行流水线。

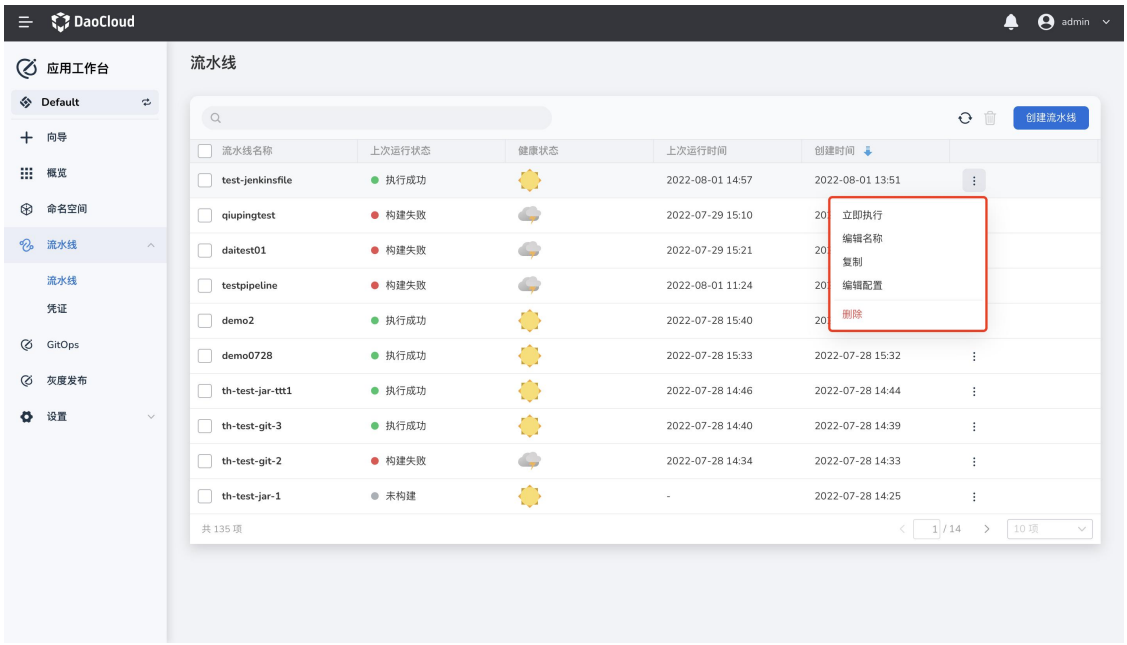
   - 定时仓库扫描计划：输入 CRON 表达式定义扫描仓库的时间周期。**输入表达式后下方会提示当前表达式的含义**。有关详细的表达式语法规则，可参考 Cron 时间表语法。

   - 定时触发：定时触发构建流水线，无论代码仓库是否有更新，都会在规定时间重新运行流水线。



pipeline04

6. 完成创建。确认所有参数输入完成后，点击确定按钮，完成自定义流水线创建，自动返回流水线列表。点击列表右侧的 ⋮ 可以执行各项操作。


DaoCloud
admin
应用工作台
Default
向导
概览
命名空间
流水线
流水线
凭证
GitOps
灰度发布
设置
流水线
创建流水线
流水线名称
上次运行状态
健康状态
上次运行时间
创建时间
test-jenkinsfile
执行成功
2022-08-01 14:57
2022-08-01 13:51
qiupingtest
构建失败
2022-07-29 15:10
daitest01
构建失败
2022-07-29 15:21
testpipeline
构建失败
2022-08-01 11:24
demo2
执行成功
2022-07-28 15:40
demo0728
执行成功
2022-07-28 15:33
2022-07-28 15:32
th-test-jar-ttt1
执行成功
2022-07-28 14:46
2022-07-28 14:44
th-test-git-3
执行成功
2022-07-28 14:40
2022-07-28 14:39
th-test-git-2
构建失败
2022-07-28 14:34
2022-07-28 14:33
th-test-jar-1
未构建
-
2022-07-28 14:25
立即执行
编辑名称
复制
编辑配置
删除
共 135 项
1 / 14
10 项

pipeline05

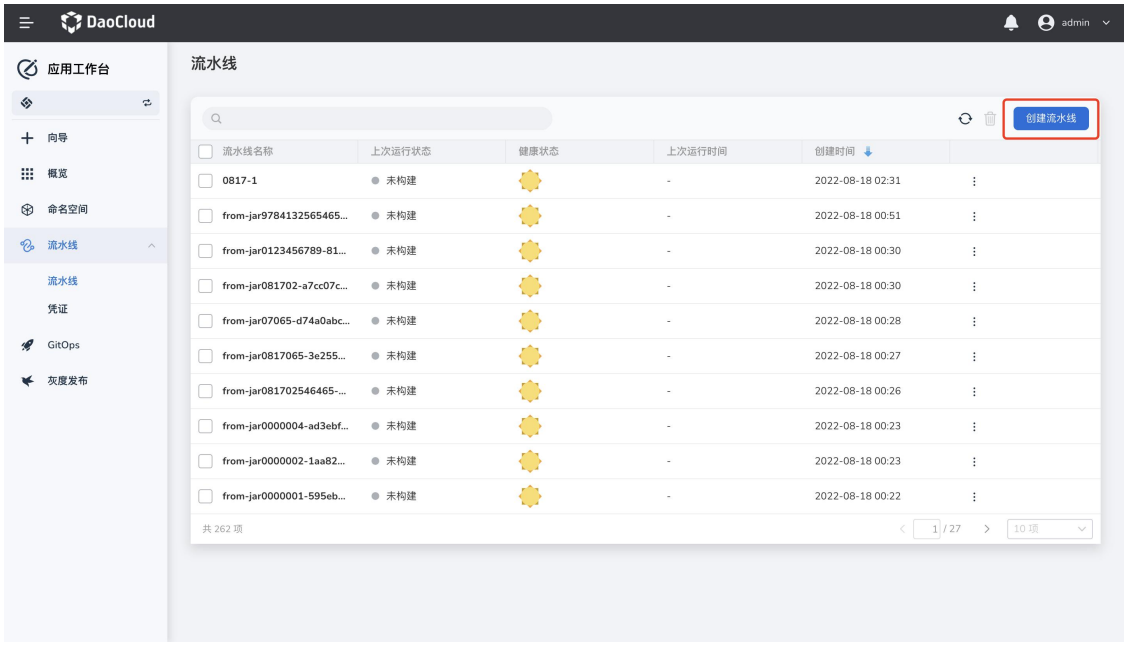
基于内置模板创建流水线

应用工作台模块支持基于内置模板或用户的自定义模板快速创建流水线。根据常见的使用场景，应用工作台模块内置了 Golang、Nodejs、Maven 模板。

操作步骤

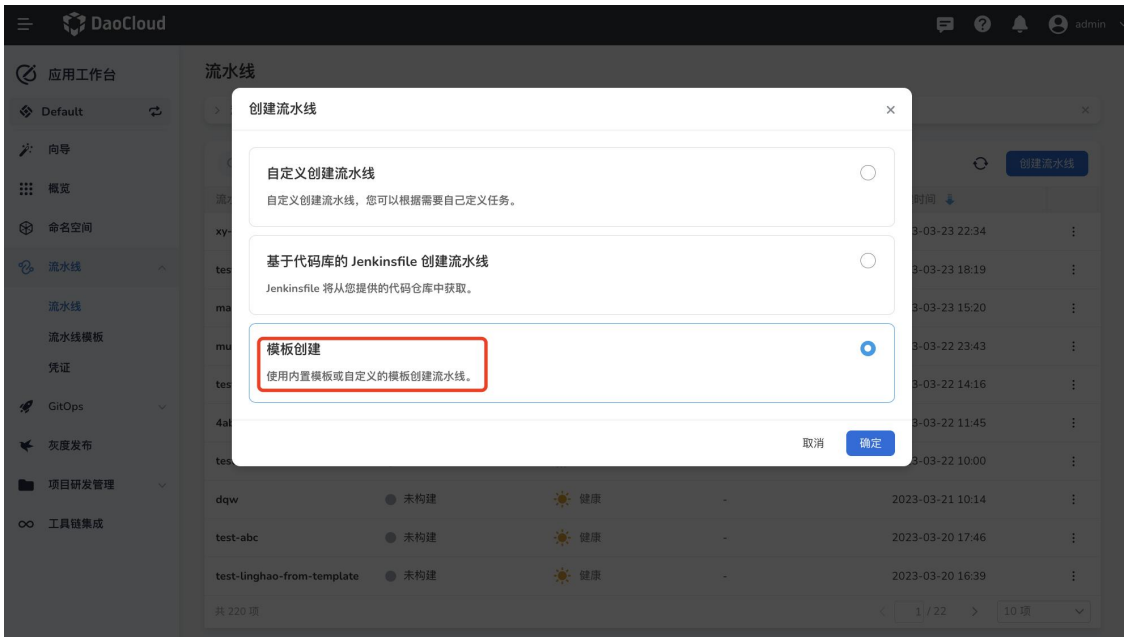
基于内置模板创建流水线的步骤如下：

1. 在流水线列表页点击创建流水线。



click-create

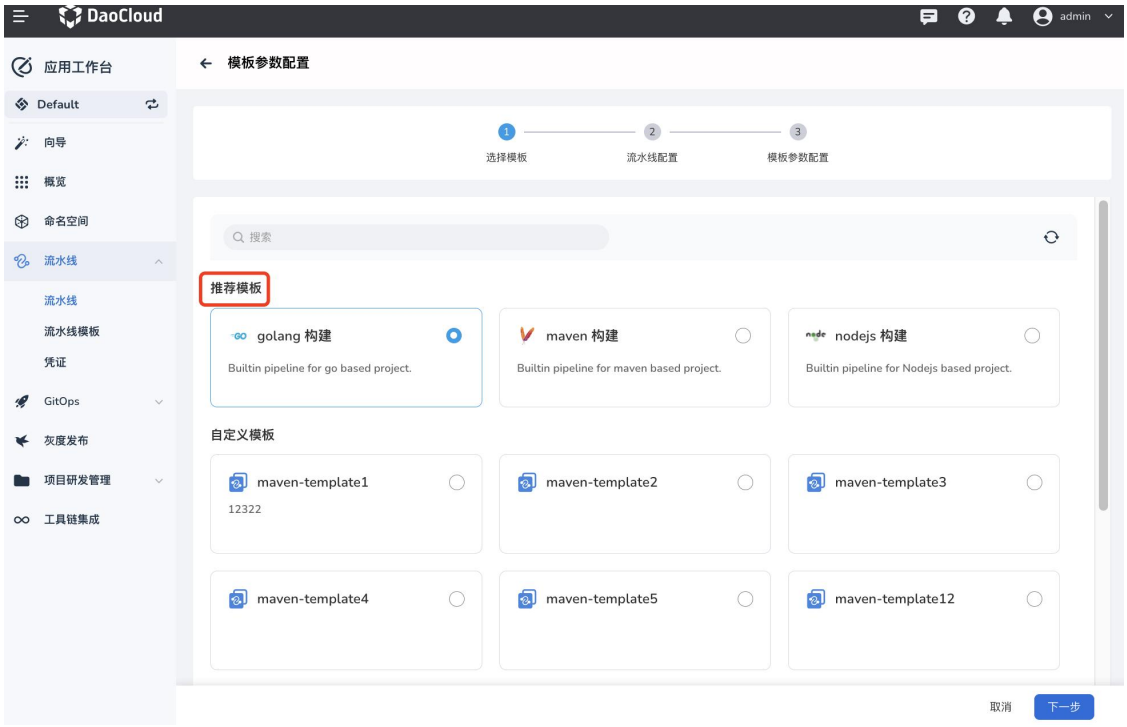
2. 在弹出的对话框中，选择模板创建，点击确定。



select-type

3. 选择合适的流水线模板，然后点击下一步。

   推荐模板标题下列出了所有的内置模板，自定义模板标题下列出了用户创建的所有自定义模板。



选择模板

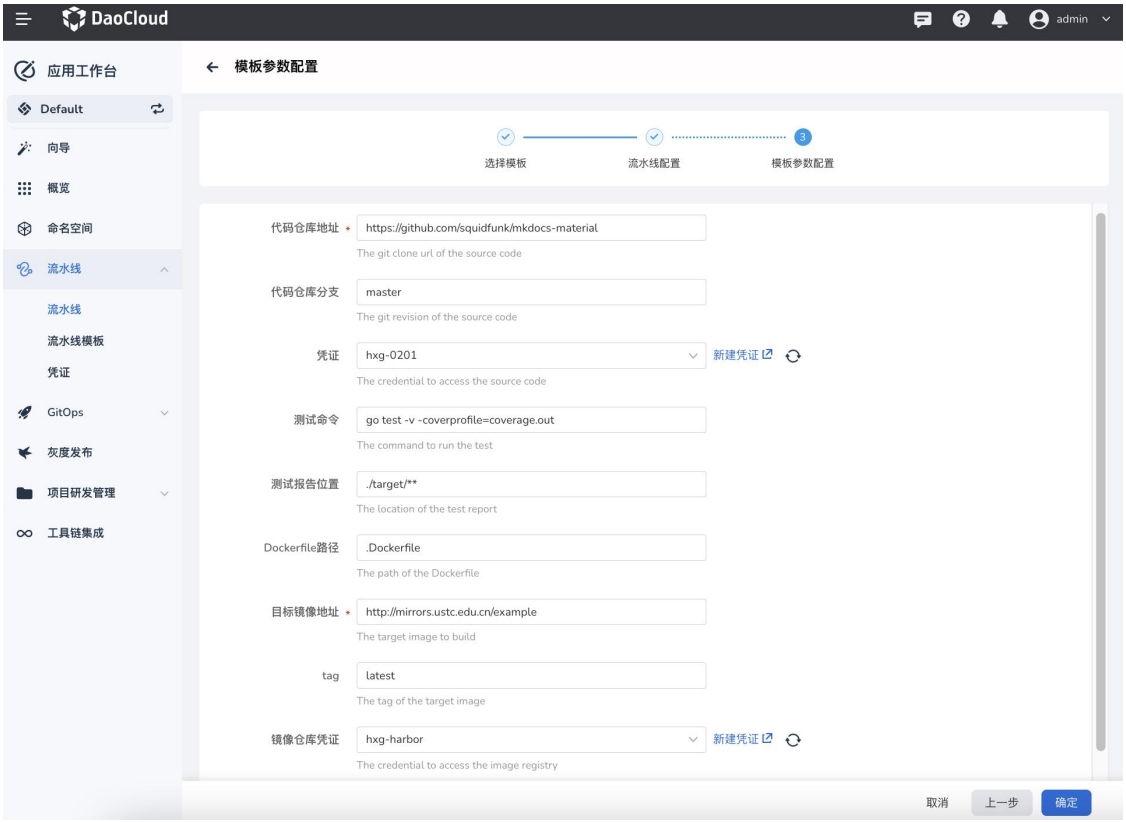
4. 参考自定义创建流水线填写流水线配置，然后点击下一步。

5. 参考下列说明填写模板参数，然后点击确定。

   - 代码仓库地址：远程代码仓库的地址，必填
   - 分支：基于哪个分支的代码构建流水线，必填
   - 凭证：如果是私有仓库，需要提前创建凭证并在此处选择该凭证
   - 测试命令：单元测试命令

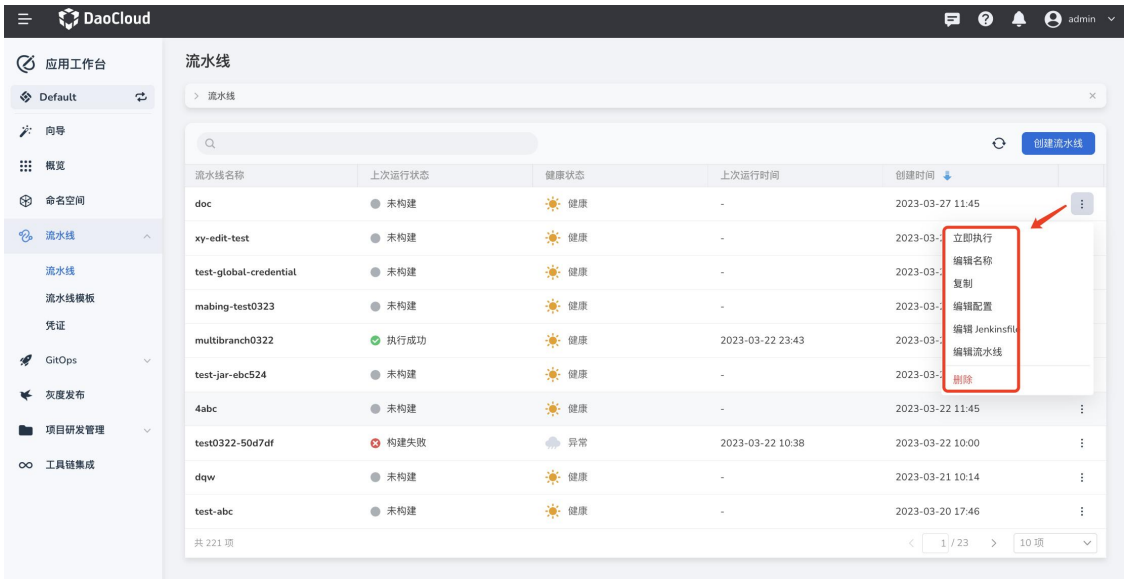
     如果使用 Golang 模板，则测试命令默认为 go test -v -coverprofile=coverage.out 如果使用 Nodejs 模板，则测试命令默认为 npm test 如果使用 Maven 模板，则测试命令默认为 mvn -B test -Dmaven.test.failure.ignore=true
   - 测试报告位置：测试报告所在的位置，并进行分析生成测试报告

- Dockerfile 路径：Dockerfile 文件在代码仓库中的绝对路径，必填
- 目标镜像地址：输入镜像仓库名称，必填
- tag：为运行此流水线后新生成的镜像添加 tag
- 像仓库凭证：访问镜像仓库的凭证。如果是私有仓库，需要提前创建凭证并在此处选择该凭证



golang

6. 完成创建后，可以在流水线列表查看新建的流水线。在流水线右侧点击更多操作按钮，可以进行执行、编辑、复制流水线等操作。


DaoCloud
admin
应用工作台
Default
向导
概览
命名空间
流水线
流水线
流水线模板
凭证
GitOps
灰度发布
项目研发管理
工具链集成
流水线
流水线
创建流水线
流水线名称
上次运行状态
健康状态
上次运行时间
创建时间
doc 未构建 健康 - 2023-03-27 11:45
xy-edit-test 未构建 健康 -
test-global-credential 未构建 健康 -
mabing-test0323 未构建 健康 -
multibranch0322 执行成功 健康 2023-03-22 23:43
test-jar-ebc524 未构建 健康 -
4abc 未构建 健康 - 2023-03-22 11:45
test0322-50d7df 构建失败 异常 2023-03-22 10:38 2023-03-22 10:00
dqw 未构建 健康 - 2023-03-21 10:14
test-abc 未构建 健康 - 2023-03-20 17:46
立即执行
编辑名称
复制
编辑配置
编辑 Jenkinsfile
编辑流水线
删除
共 221 项
1 / 23
10 项

actions

创建自定义流水线

应用工作台流水线支持自定义创建，通过该方式创建的流水线，您可对流水线进行可视化编排。

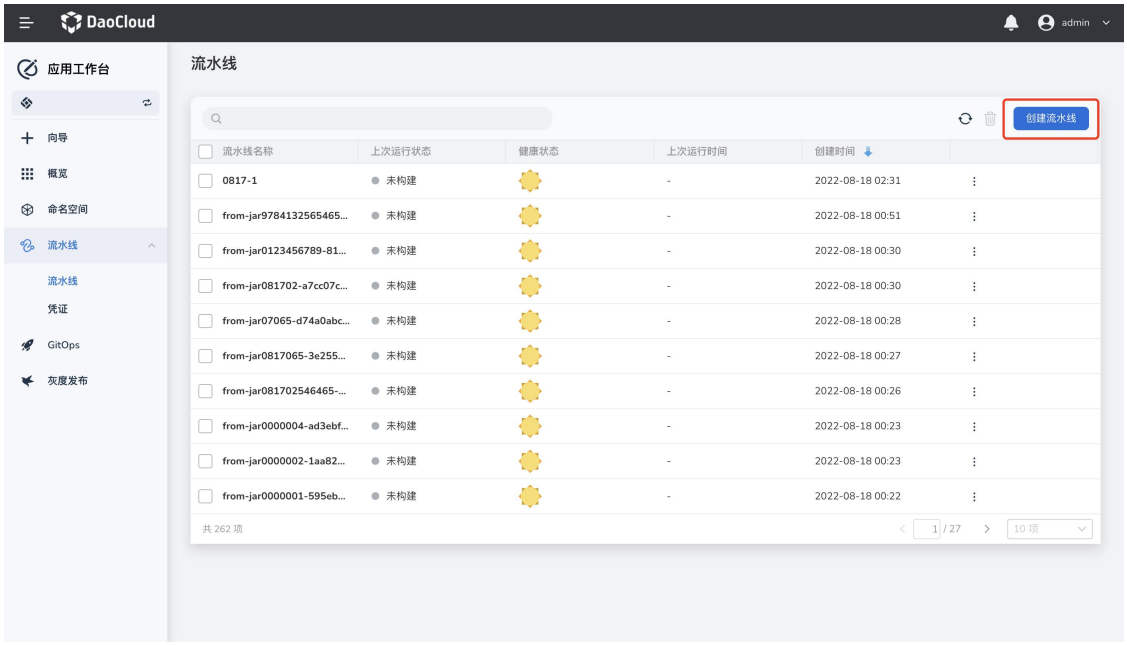
前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 workspace editor 或更高权限。

操作步骤

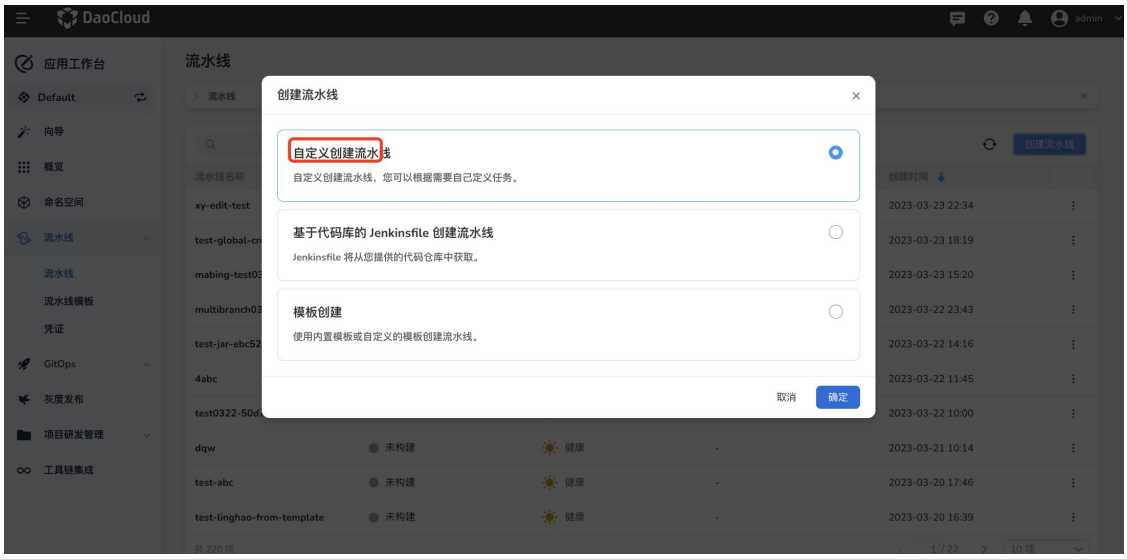
具体操作步骤如下：

1. 在流水线列表页点击创建流水线。



createpipelinbutton

2. 在弹出的对话框中，选择自定义创建流水线，点击确定。

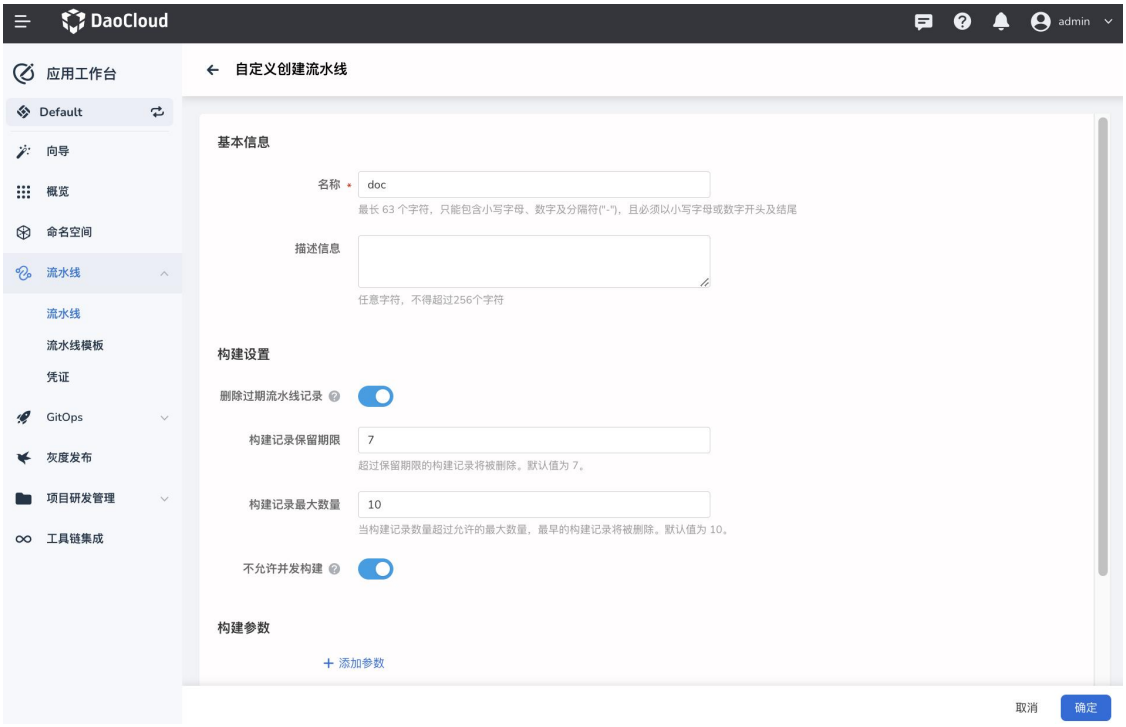


selecttype

3. 参考下列说明填写基本信息、构建设置、构建参数。

   - 名称：流水线的名称。同一个工作空间下流水线名称必须唯一。
   - 删除过期流水线记录：删除之前的构建记录，以节省 Jenkins 使用的磁盘空间。
     - 构建记录保留期限：最多保留几天的构建记录，默认值为 7 天，即七天前的构建记录将被删除。
     - 构建记录最大数量：最多保留几条构建记录，默认值为 10，即最多保留 10 条记录。超过 10 条记录时，时间最早的记录最先被删除。
     - 保留期限和最大数量这两条规则同时生效，只要满足其中之一就会开始删除记录。
   - 不允许并发构建：开启后，一次只能执行一个流水线构建任务。
   - 构建参数：在开始运行流水线时传入一个或多个构建参数。默认提供五种参数类型：布尔值、字符串、多行文本、选项、密码、上传文件。

- 添加构建参数后，运行流水线时需要为每个构建参数输入对应的取值。



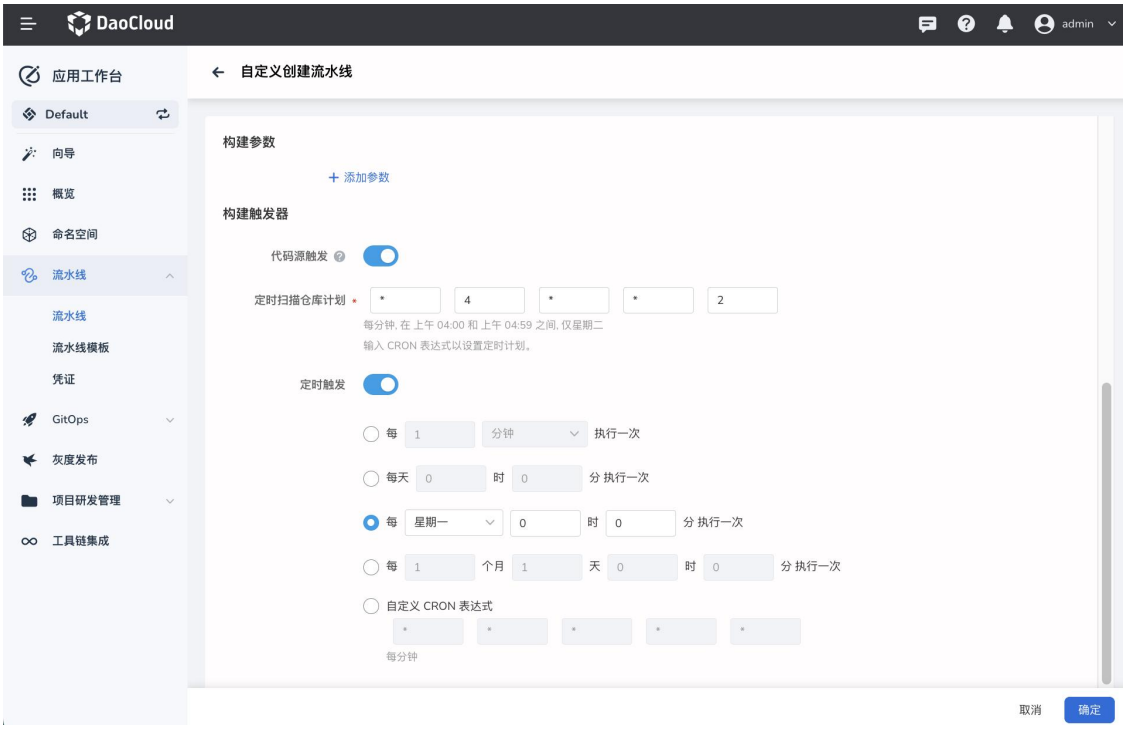
构建设置

4. 参考下列说明填写触发器参数。

   - 代码源触发：开启后，系统会根据定时仓库扫描计划定期扫描仓库代码中用于构建流水线的特定分支，如果有更新内容，则重新运行流水线。

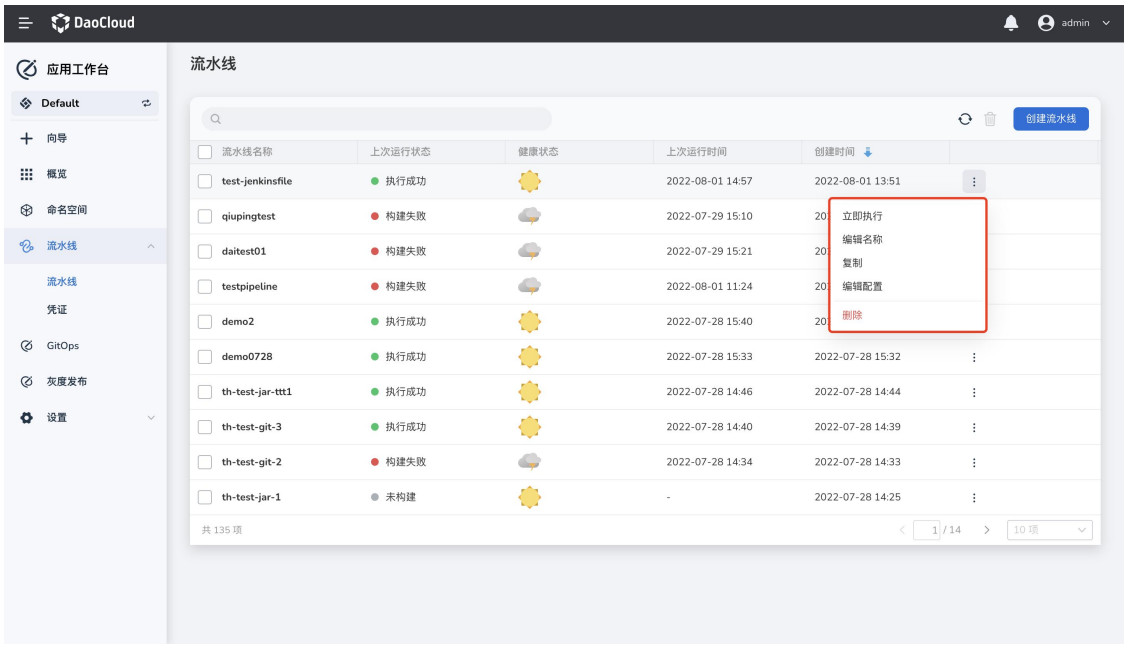
   - 定时仓库扫描计划：输入 CRON 表达式定义扫描仓库的时间周期。**输入表达式后下方会提示当前表达式的含义**。有关详细的表达式语法规则，可参考 Cron 时间表语法。

   - 定时触发：定时触发构建流水线，无论代码仓库是否有更新，都会在规定时间重新运行流水线。



pipeline01

5. 完成创建。确认所有参数输入完成后，点击确定按钮，完成自定义流水线创建，自动返回流水线列表。点击列表右侧的 ： 可以执行各项操作。



pipeline05

!!! warning

– 自定义流水线创建之后，需要手动定义流水线的各个阶段（即编辑流水线），然后才能运行流水线。如果定义流程就直接运行流水线，会出现`构建失败`的错误。
– 如果[基于模板创建流水线](template.md)或[基于 Jenkinsfile 创建流水线](jenkins.md)，则可以直接运行，无需手动编辑流水线。

下一步：编辑流水线

创建多分支流水线

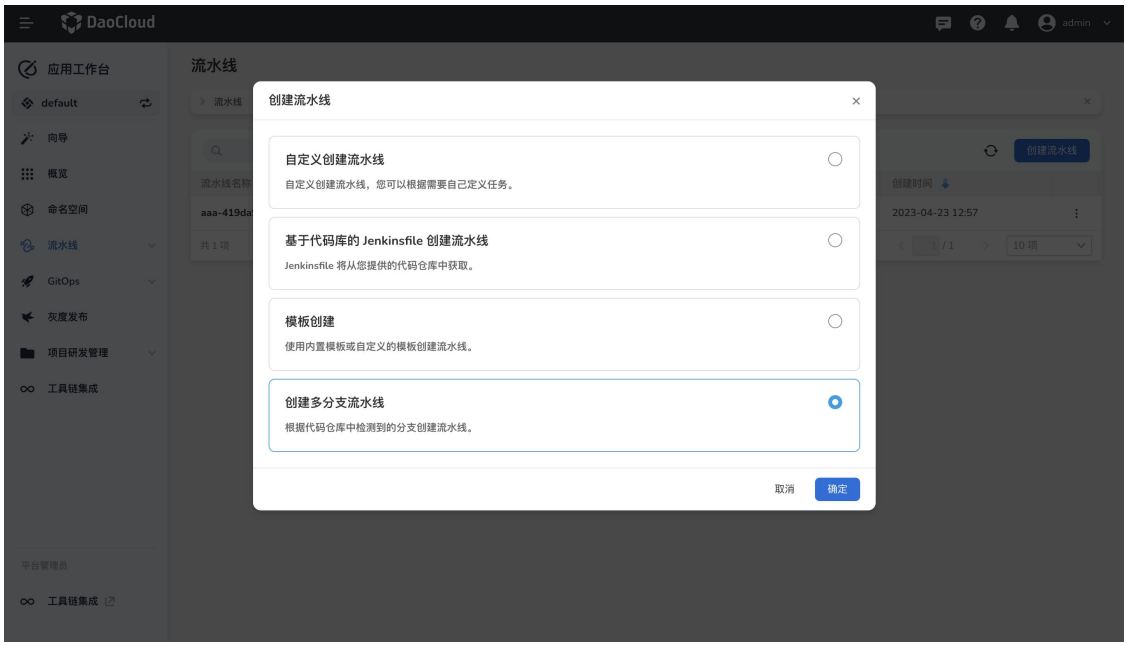
应用工作台流水线支持基于代码仓库创建多分支流水线。

前提条件

- 创建工作空间、创建用户。
- 将该用户添加至该工作空间，并赋予 workspace editor 或更高权限。
- 提供一个代码仓库，并且代码仓库的源码有多个分支，并且均有 Jenkinsfile 文本文件。
- 如果是私有仓库，需要事先创建仓库访问凭证。

操作步骤

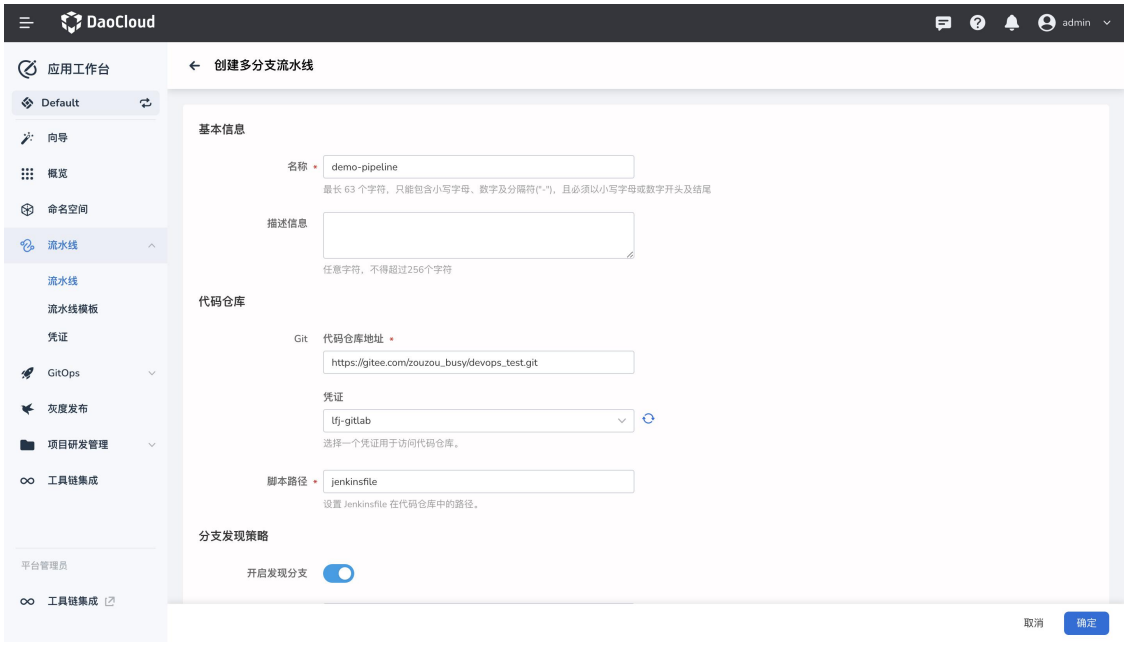
1. 在流水线列表页点击创建流水线。
2. 选择创建多分支流水线，点击确定。



mutilpipeline01

3. 参考下方说明填写基本信息和代码仓库信息。

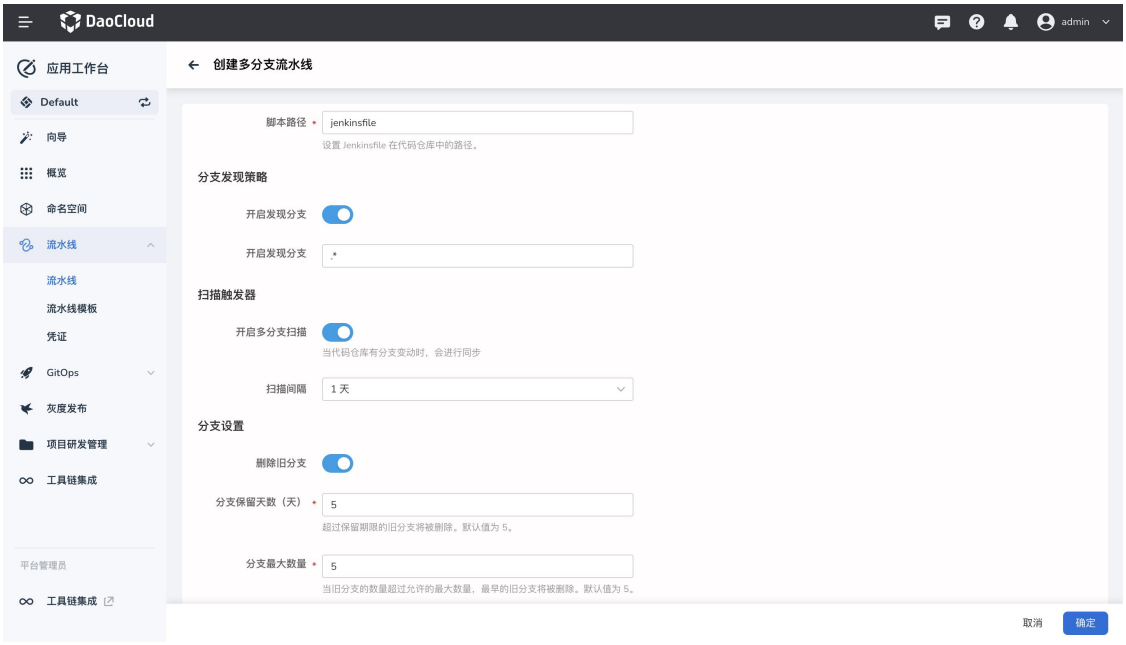
- 名称：流水线的名称。同一个工作空间下流水线名称必须唯一。
- 描述信息：用户描述当前流水线的作用。
- 代码仓库地址：填写远程代码仓库的地址。
- 凭证：对于私有仓库，需要提前创建仓库访问凭证并在此处选择该凭证。
- 脚本路径：Jenkinsfile 文件在代码仓库中的绝对路径。



mutilpipeline02

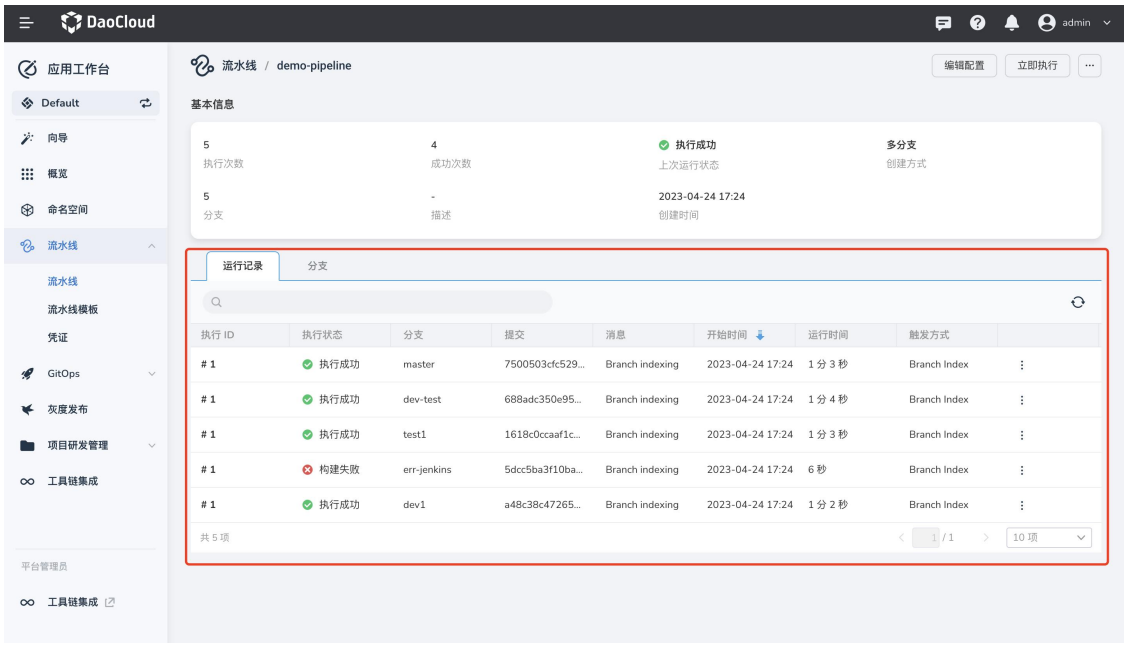
4. 参考下方说明填写分支发现策略、扫描触发器、分支设置、克隆配置信息。
   - 开启发现分支：默认值为 .* ，通过正则表达式来过滤分支。
   - 开启多分支扫描：开启后，一旦代码仓库有分支变动，会进行同步。
   - 扫描间隔：根据预设的间隔来进行扫描代码仓库，来检查是否又更新。
   - 删除旧分支：开启后将根据策略删除旧分支及流水线
   - 保留天数：旧分支及旧分支的流水线保留的天数，到期后删除。

- 保留数量：旧分支及旧分支的流水线保留的数量。
- 浅克隆：开启后，拉取代码仓库的最新版本。支持设置克隆深度，一般默认为 1 用于加速拉取。
- 克隆超时：拉取代码时的最长等待时间。



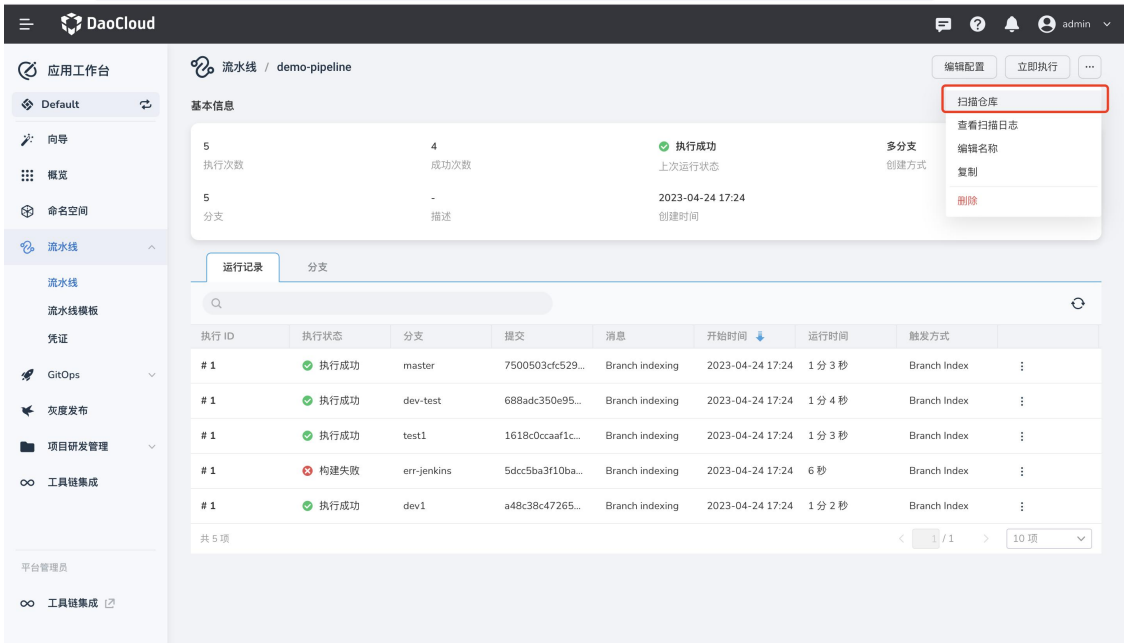
mutilpipeline03

5. 完成创建。确认所有参数输入完成后，点击确定按钮，完成多分枝流水线创建，自动返回流水线列表。



mutilpipeline04

6. 创建完成后，会自动触发执行符合条件的分支的对应的流水线。

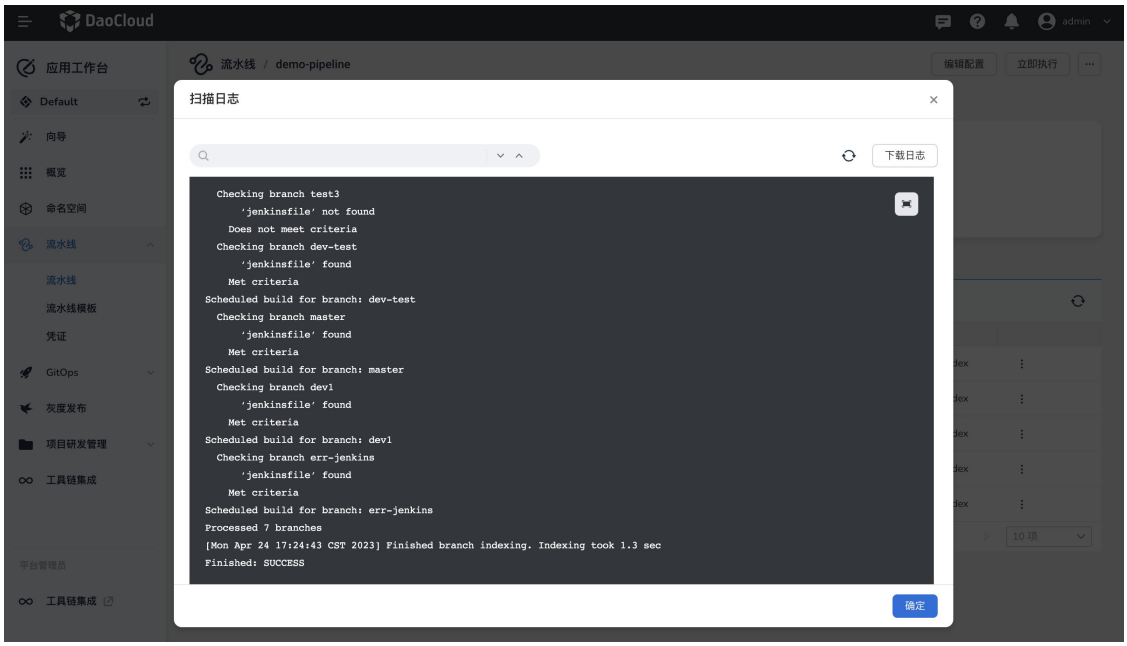


mutilpipeline05

其他操作

扫描仓库

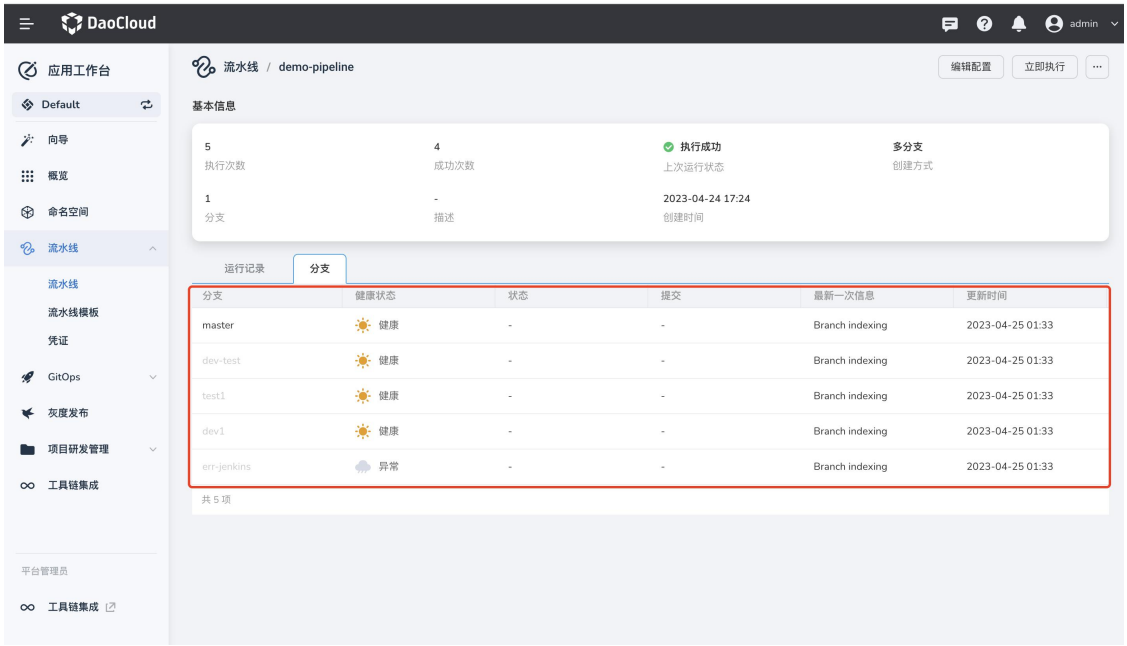
扫描仓库的目的是通过手动的方式触发发现代码仓库的新分支。



mutilpipeline06

查看扫描日志

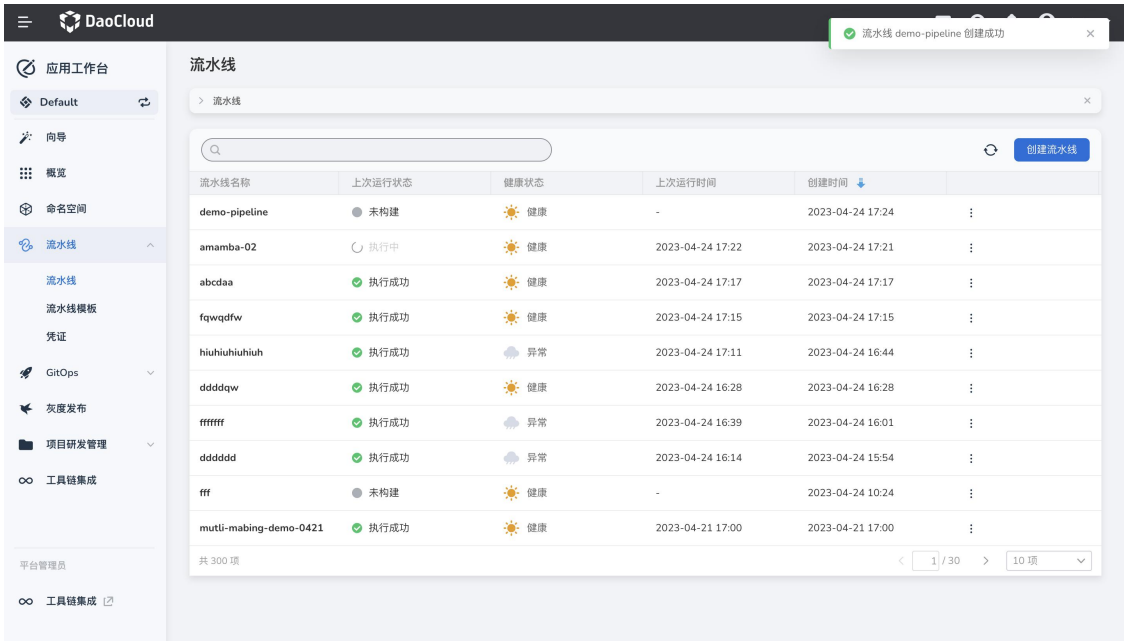
展示最新一次扫描代码仓库时发现分支的日志。



mutilpipeline07

查看分支

根据分支发现策略获取的分支信息，其中禁用 状态的分支代表，最新扫描结果中不符合分支发现的策略。



mutilpipeline08

集成工具链

DevOps 工具链是一组工具，使 DevOps 团队能够在整个产品生命周期中进行协作并解决关键的 DevOps 基础问题。

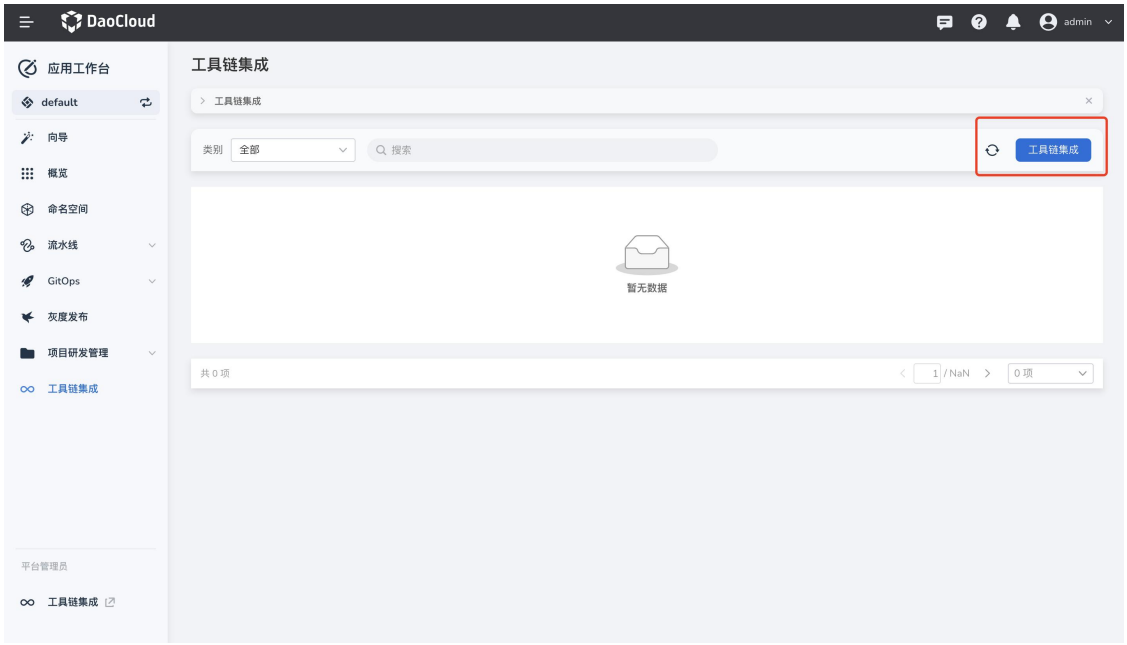
应用工作台支持两种视角的工具链集成：工作空间集成、管理员集成。其中管理员集成的实例支持分配到工作空间下，以供工作空间使用。

支持集成的工具链

工具链名称	描述	备注
GitLab	集成 GitLab 仓库后，可在流水线中使用	–
Jira	通过在应用工作台中集成 Jira，从而支持对 Jira–>Issue 的追踪。	–
Jenkins	集成 Jenkins 后，所有的工作空间均将获得流水线能力进行构建。	仅支持管理员集成，且全平台仅能集成一次 Jenkins
SonarQube	集成 SonarQube 后，可以在流水线中定义代码质量扫描	–

操作步骤

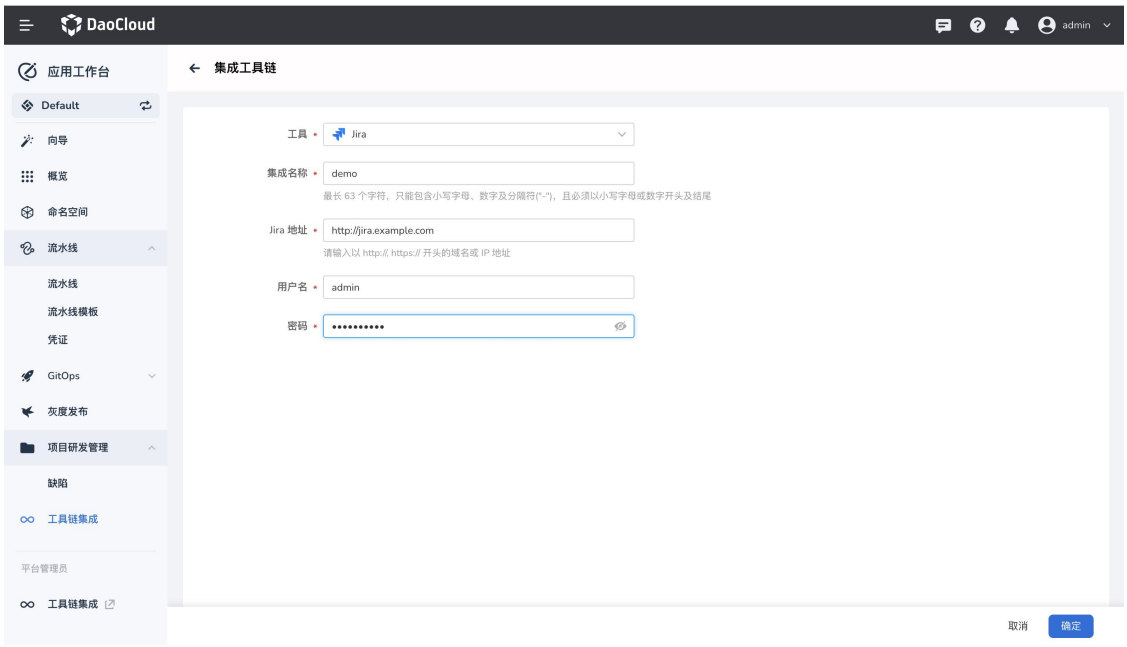
1. 进入工具链集成 页面，点击工具链集成按钮。



tool01

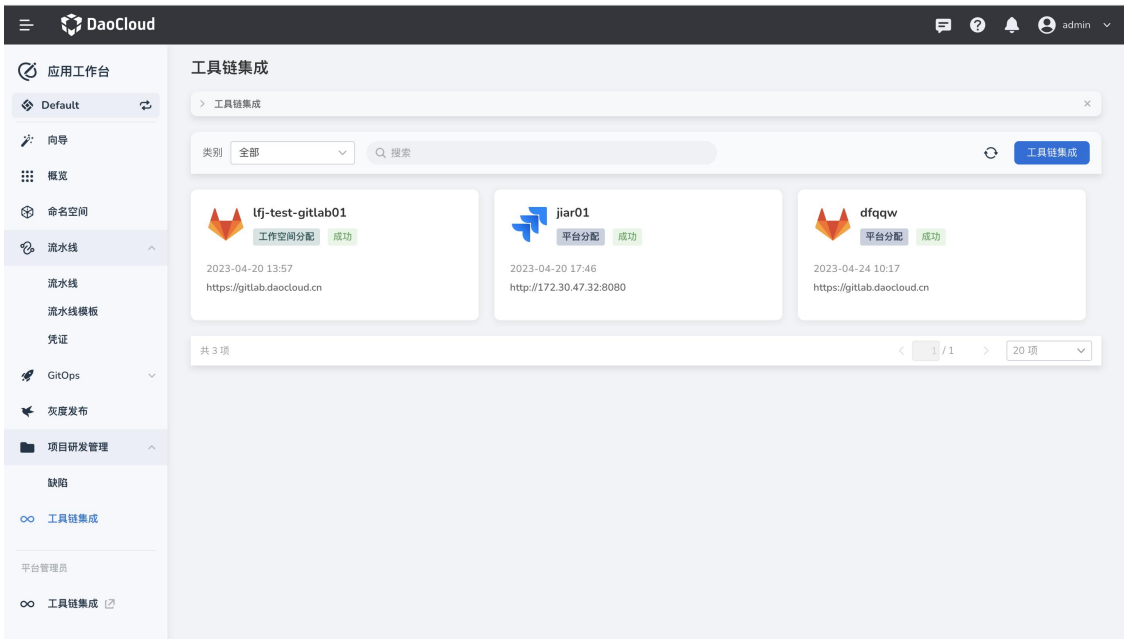
2. 参考以下说明，配置相关参数：

   - 工具：选择一个工具链类型进行集成。
   - 集成名称：集成工具的名称，不可重复。
   - 地址：可访问工具链的地址，以 http://, https:// 开头的域名或 IP 地址。
   - 用户名和密码：可以登录工具链的用户和密码、



tool02

3. 单击确定，集成成功返回到工具链列表页面。



tool03

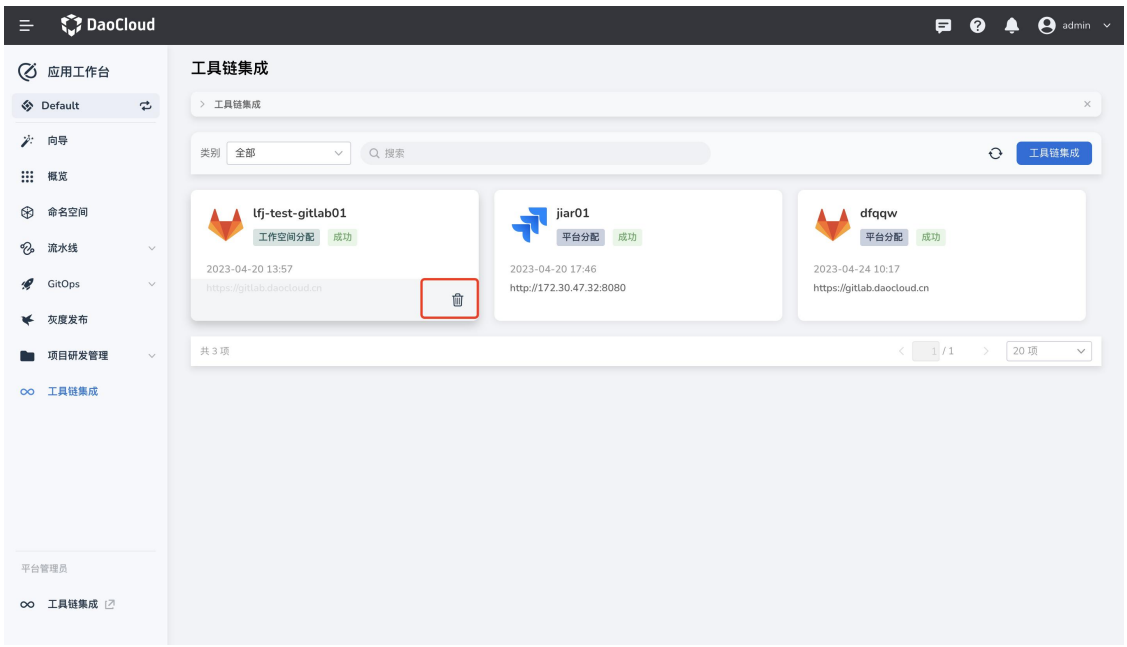
管理工具链实例

对集成进来的工具链进行管理，分为两种：工作空间集成的工具链、管理员集成的工具链

解除集成

工作空间

对于工作空间分配的工具链实例，支持解除集成操作：



tool04

对于平台分配的工具链实例，不支持解除集成操作：

管理员

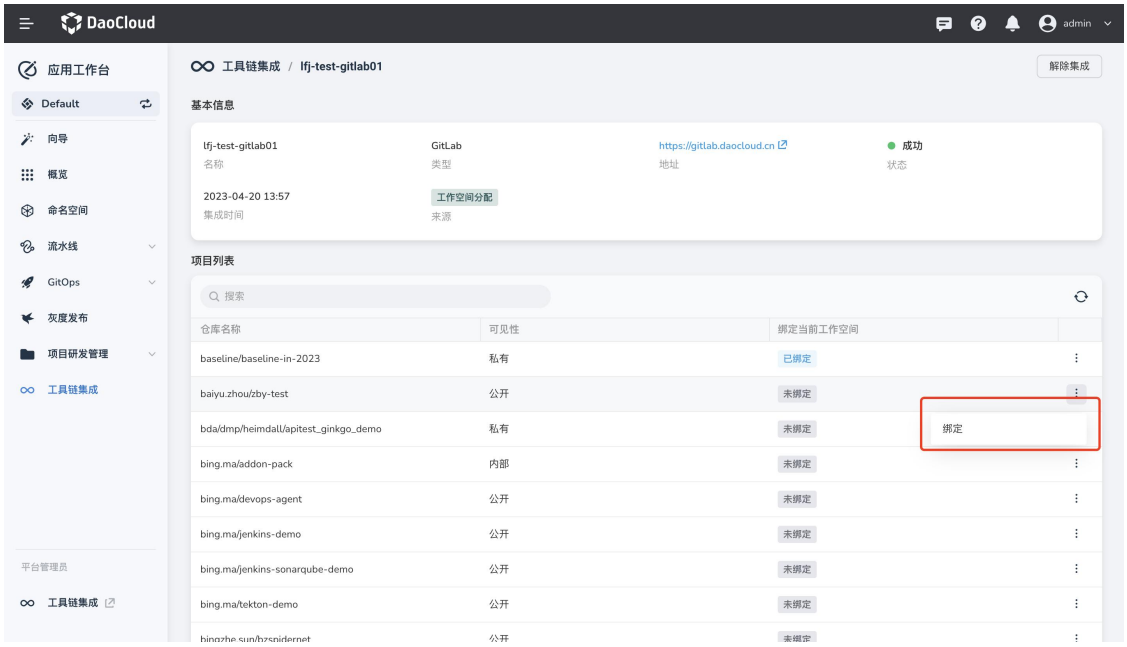
解除集成后，分配给工作空间下的实例也会自动被删除。

分配项目

工作空间

对于工作空间分配的工具链实例，支持绑定实例下的项目到当前工作空间下并进行使用。

另外已绑定的项目支持解除绑定。

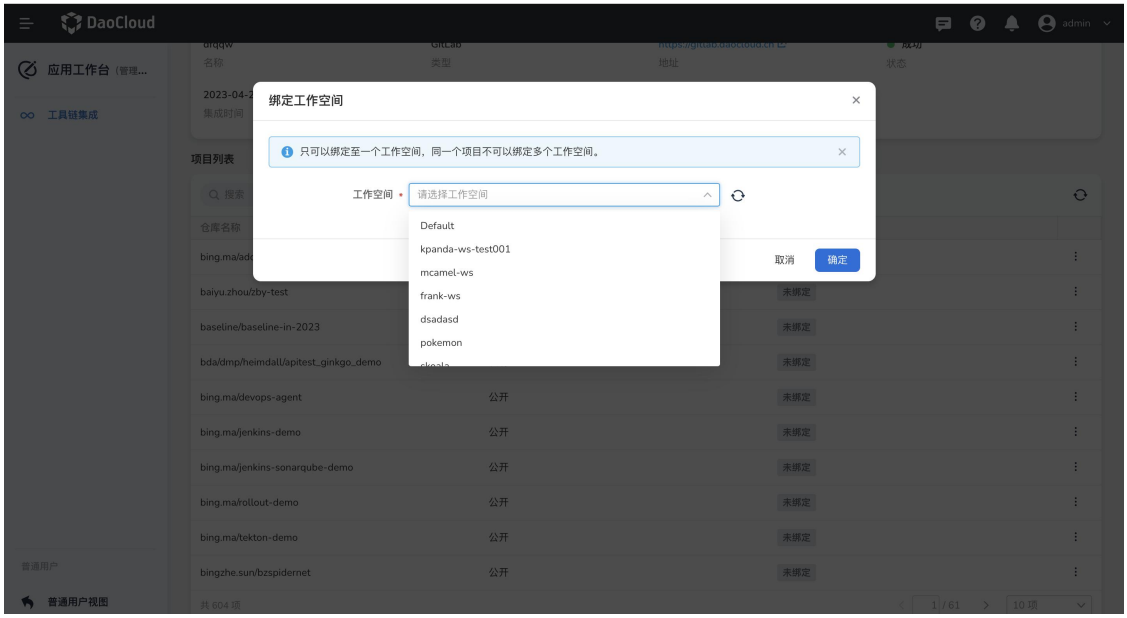


tool05

对于平台分配的工具链实例，仅支持查看，不支持绑定操作。

管理员

支持分配项目到工作空间，分配成功后，会在工作空间下生成一条实例，并且项目可以被该工作空间使用。


DaoCloud
admin
应用工作台（管理...
工具链集成
普通用户
普通用户视图
名称
类型
地址
状态
集成时间
项目列表
搜索
仓库名称
绑定工作空间
只可以绑定至一个工作空间，同一个项目不可以绑定多个工作空间。
工作空间 *
请选择工作空间
Default
kpanda-ws-test001
mcamel-ws
frank-ws
dsadasd
pokemon
取消
确定
baiyu.zhou/zby-test
baseline/baseline-in-2023
bda/dmp/heimdall/apitest_ginkgo_demo
bing.ma/devops-agent
bing.ma/jenkins-demo
bing.ma/jenkins-sonarqube-demo
bing.ma/rollout-demo
bing.ma/tekton-demo
bingzhe.sun/bzspidernet
公开
未绑定
共 604 项
1 / 61
10 项

tool06

在 Jenkins 中使用自定义工具链

应用工作台运行流水线的镜像内置了常用的工具，包括：make、wget、gcc 等[15]。应用工作台支持集成用户自定义的工具或者是特定版本的工具，可用于以下场景：

- 升级/降级到特定版本以修复工具的 Bug;
- 安装流水线运行依赖的工具（例如：yq、curl、kustomize 等）；
- 提前准备代码依赖的包以加快编译；

应用工作台支持通过 Volume 挂载或构建自定义镜像两种方式添加自定义工具链。

通过 Volume 挂载

第一种方式是使用 init 容器和 volumeMount 的方式拷贝工具到 agent 容器里。通过修改 jenkins casc 的 configmap，修改容器的默认行为。以 go 为例，下面这个例子中，init 容器使用另一个版本的 Helm 覆盖了 agent 中自带的 Helm：

```
jenkins.yaml: |
  jenkins:
    mode: EXCLUSIVE
    ...
    clouds:
      - kubernetes:
          templates:
            - name: go
```

[15] 完整列表可以参考：https://docs.daocloud.io/amamba/user-guide/pipeline/config/agent/#label_1

```
    label: go
    yaml: |
      spec:
        volumes:
        # 1. Define an emptyDir volume which will hold the custom binaries
        - emptyDirVolume:
            memory: true
            name: custom-tools
        # 2. Use an init container to download/copy custom binaries into the emptyDir
        initContainers:
        - args:
          - wget -qO- https://storage.googleapis.com/kubernetes-helm/helm-v2.12.3-linux-amd64.tar.gz
            | tar -xvzf - &&
            mv linux-amd64/helm /custom-tools/
          command:
          - sh
          - -c
          image: alpine:3.8
          name: download-tools
          volumeMounts:
          - mountPath: /custom-tools
            name: custom-tools
        # 3. Volume mount the custom binary to the bin directory (overriding the existing version)
        containers:
        - name: go
          volumeMounts:
          - mountPath: /usr/local/bin/helm
```

```
      name: custom-tools
      subPath: helm
```

创建自定义镜像

第二种方式是 BYOI (Build Your Own Image)，指通过自定义镜像的方式，将自定义工具链打包到镜像中。这种方式的好处是不需要在每个流水线中都下载工具，但是需要维护一个自定义镜像。下面是一个例子：

```
ARG RUNTIME
ARG REGISTRY_REPO
FROM $REGISTRY_REPO/amamba/jenkins-agent/builder-base:latest$RUNTIME

# Install tools needed for your repo-server to retrieve & decrypt secrets, render manifests
# (e.g. curl, awscli, gpg, sops)
RUN apt-get update && \
    apt-get install -y \
        curl \
        awscli \
        gpg && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* && \
    curl -o /usr/local/bin/sops -L https://github.com/mozilla/sops/releases/download/3.2.0/sops-3.2.0.linux && \
    chmod +x /usr/local/bin/sops
```

在创建流水线的时候选择 Agent 类型选择 kubernetes，YAML 文件如下：

```
pipeline {
  agent {
    kubernetes {
      yaml '''
        apiVersion: v1
```

```
            kind: Pod
            metadata:
              labels:
                some-label: some-label-value
            spec:
              containers:
              - name: maven
                image: your-custom-tooling-image
                command:
                - cat
                tty: true
            '''
        retries 2
      }
    }
    stages {
      stage('Run maven') {
        steps {
          container('maven') {
            sh 'mvn -version'
          }
          container('busybox') {
            sh '/bin/busybox'
          }
        }
      }
    }
}
```

参考

- https://plugins.jenkins.io/kubernetes

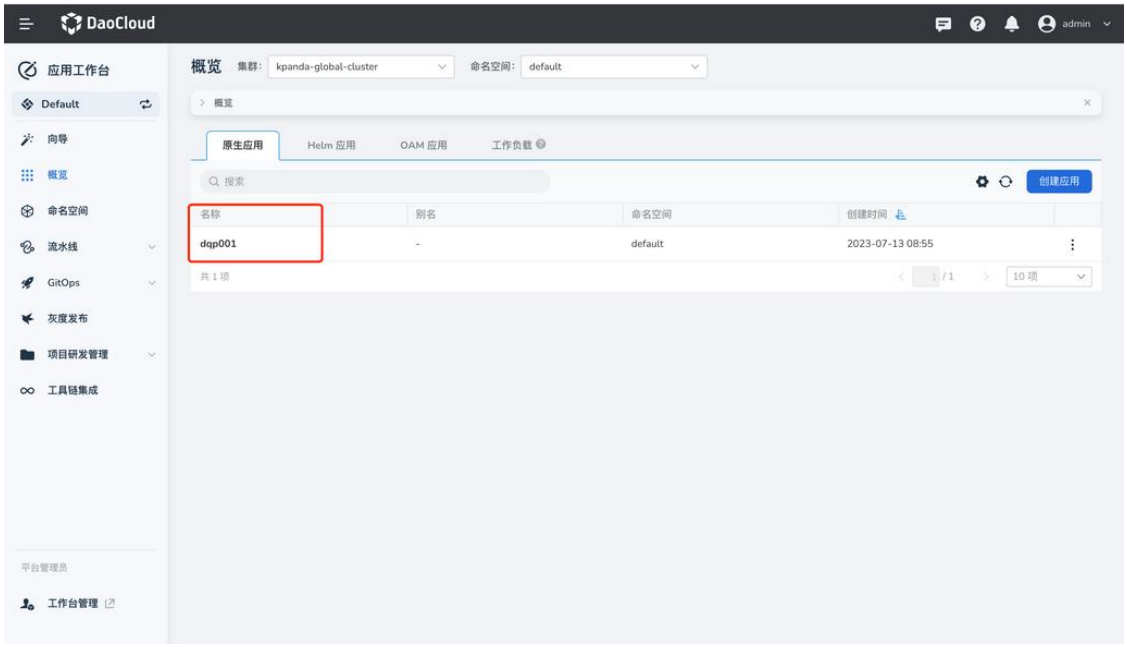
- https://argo-cd.readthedocs.io/en/stable/operator-manual/custom_tools/

原生应用管理

创建好原生应用之后，可以根据需要查看应用详情或更新应用配置。

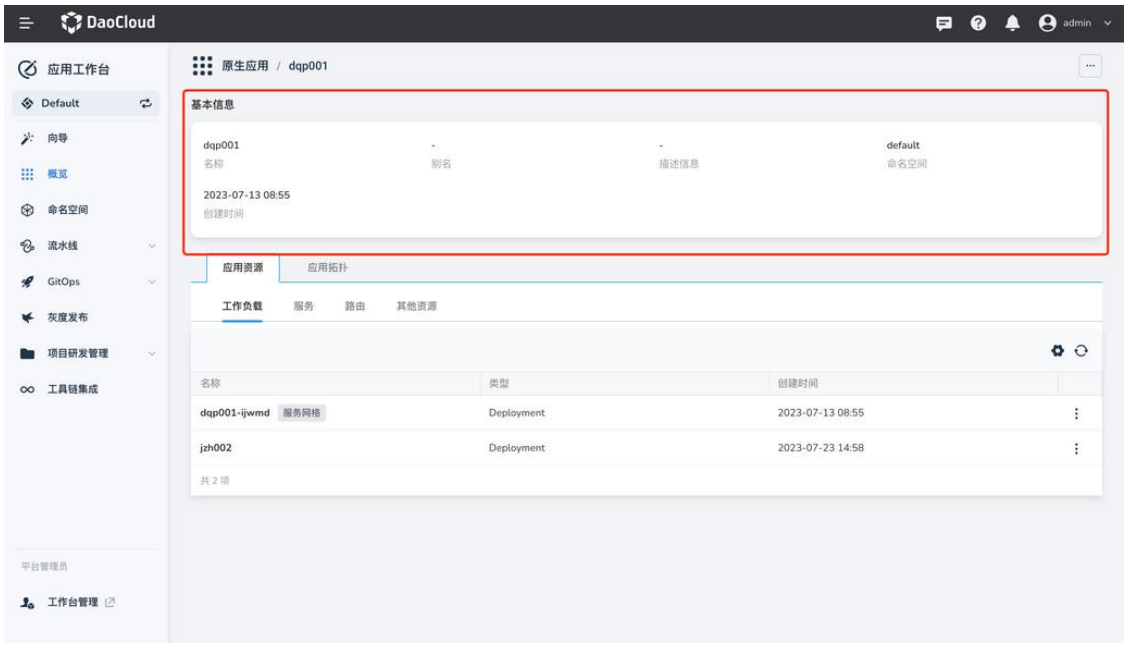
查看应用详情

在应用工作台–>概览页面，点击原生应用页签，点击原生应用的名称。



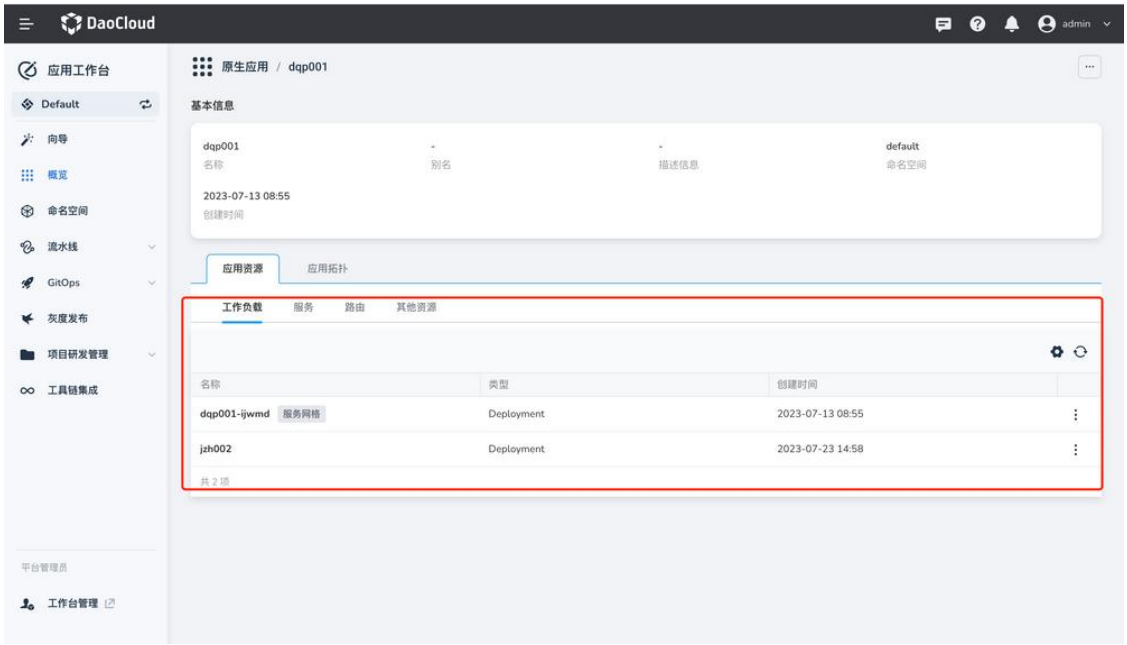
entry

- 查看应用名称、别名、描述信息、命名空间、创建时间等基本信息。



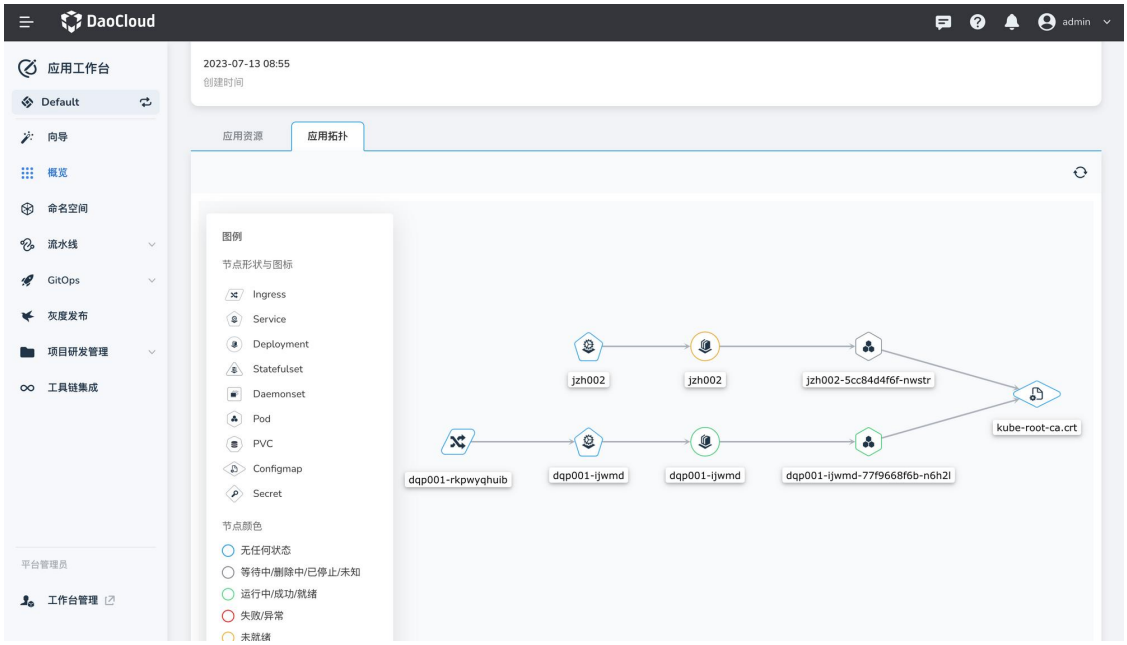
basic

- 点击应用资源页签，可以查看原生应用下的工作负载、服务、路由等 K8s 资源信息。支持对各种资源进行编辑、删除操作。



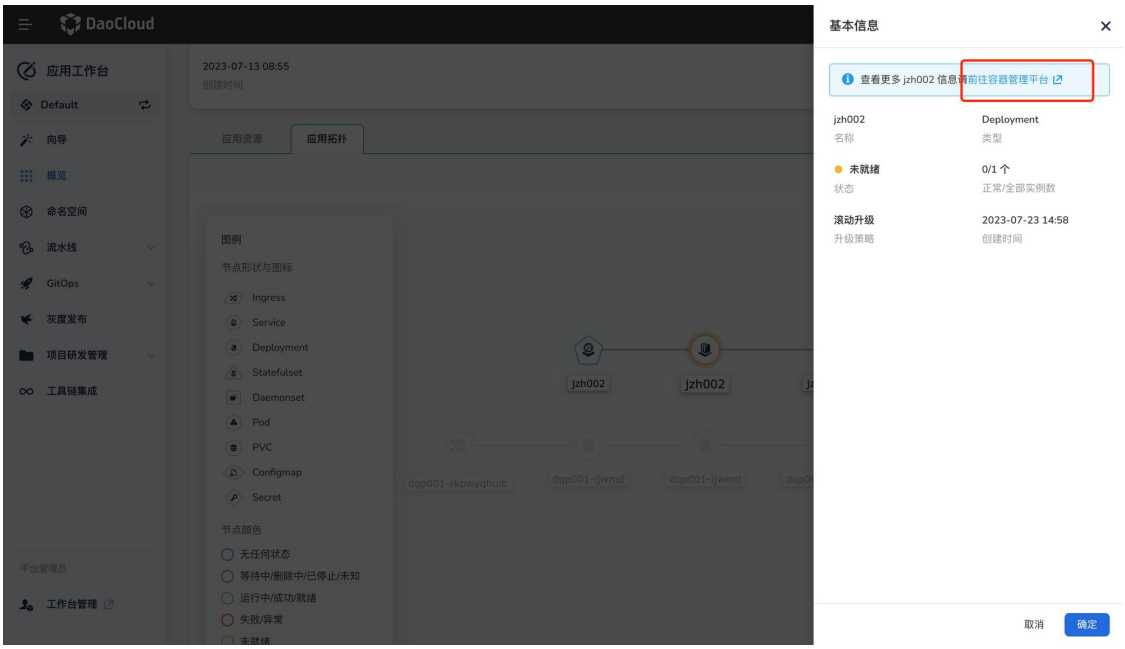
resource

- 点击应用拓扑页签，可以查看可视化的工作负载、容器、存储、配置与密钥等资源。



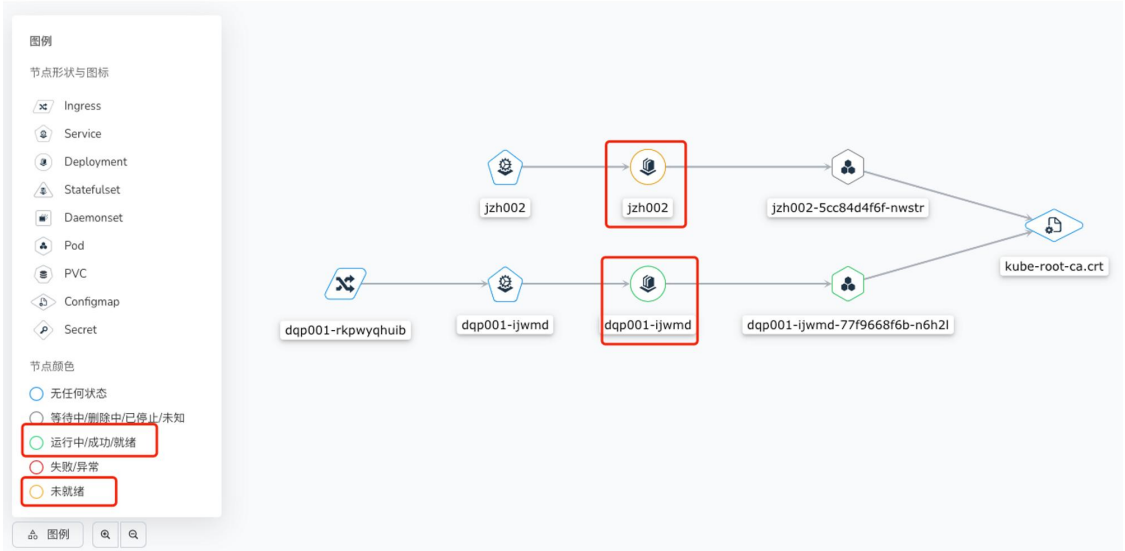
topology

  - 支持查看资源的基本信息，可以跳转到容器管理 模块查看资源的详情：



basic

- 对可视化拓扑的节点颜色进行特殊处理，一些支持存在状态的资源可通过节点颜色来判断当前资源的健康程度：

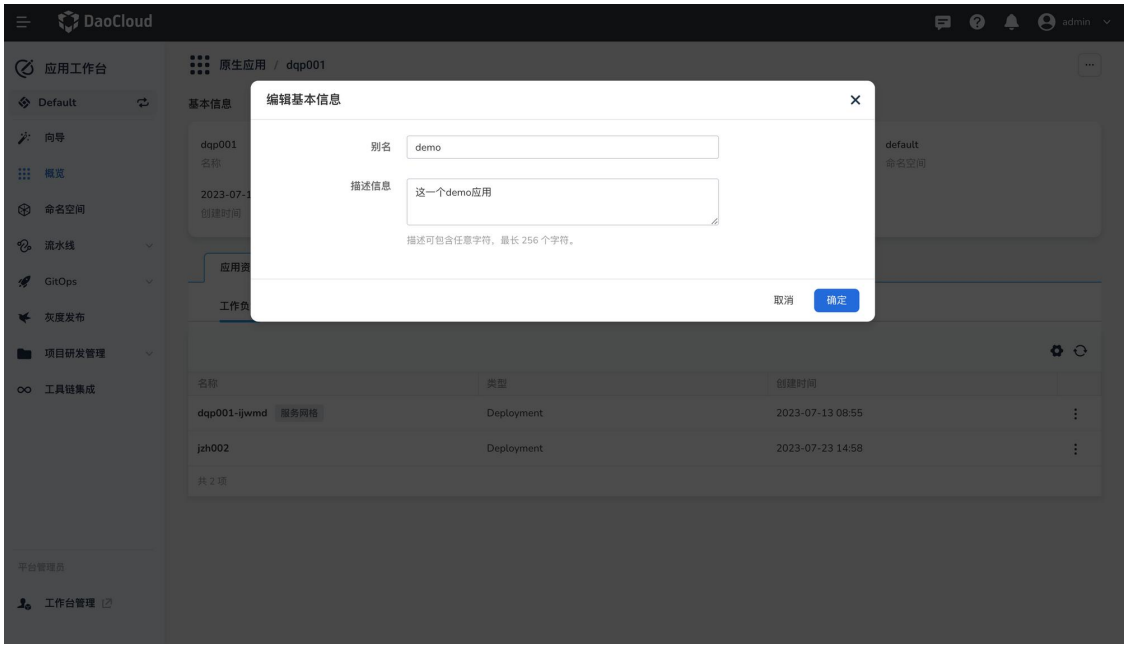


color

编辑原生应用基本信息

1. 点击原生应用的名称，然后在页面右上角点击　　选择编辑基本信息。

2. 根据需要设置别名，或补充描述信息描述信息。

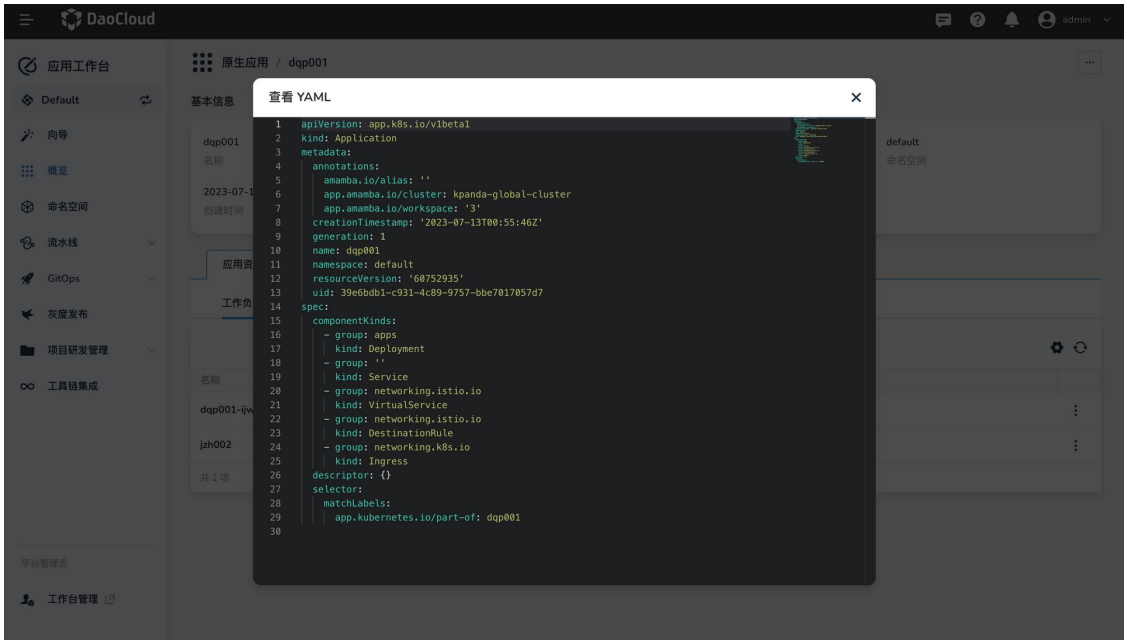


color

查看原生应用 YAML

1. 点击原生应用的名称，然后在页面右上角点击 选择查看 YAML。

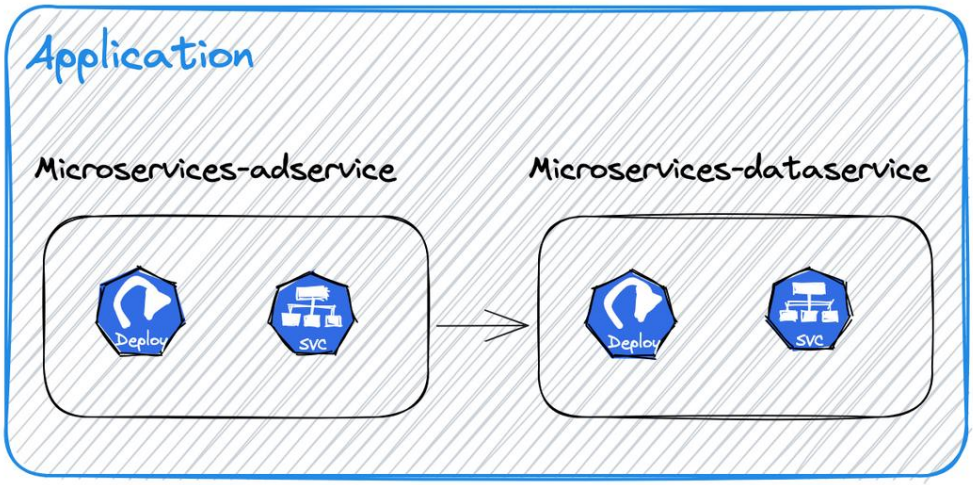
2. 查看原生应用的清单文件。


DaoCloud
admin
应用工作台
Default
向导
概览
命名空间
流水线
GitOps
灰度发布
项目研发管理
工具链集成
平台管理员
工作台管理
原生应用 / dqp001
基本信息
查看 YAML
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
annotations:
amamba.io/alias: ''
app.amamba.io/cluster: kpanda-global-cluster
app.amamba.io/workspace: '3'
creationTimestamp: '2023-07-13T00:55:46Z'
generation: 1
name: dqp001
namespace: default
resourceVersion: '60752935'
uid: 39e6bdb1-c931-4c89-9757-bbe7017057d7
spec:
componentKinds:
- group: apps
kind: Deployment
- group: ''
kind: Service
- group: networking.istio.io
kind: VirtualService
- group: networking.istio.io
kind: DestinationRule
- group: networking.k8s.io
kind: Ingress
descriptor: {}
selector:
matchLabels:
app.kubernetes.io/part-of: dqp001
dqp001
名称
default
命名空间
创建时间
名称
jzh002
共 2 项

color

通过原生应用部署传统微服务

原生应用旨在为客户提供由多个 Kubernets 资源关联组成的应用，并提供统一视图。本文介绍如何通过原生应用部署传统的微服务应用。本文用到的两个示例微服务应用名为 adservice 和 dataservice。



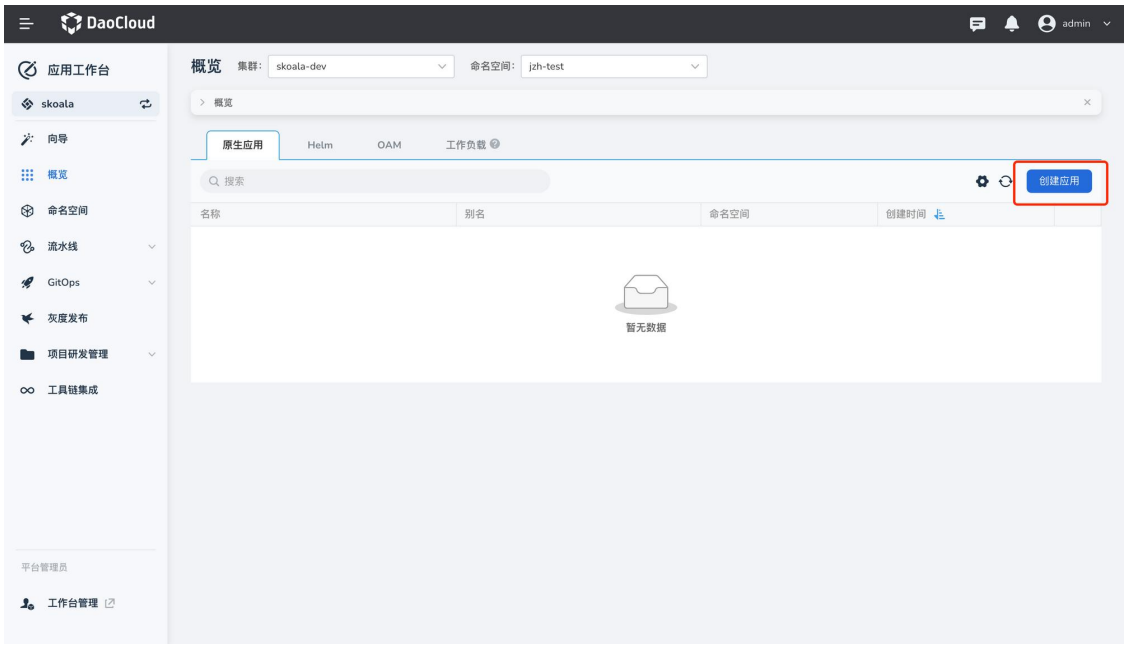
示意图

前提条件

- 当前工作空间下有创建好的托管 Nacos 实例，参考创建托管注册中心。
- 准备好需要部署的传统微服务的镜像，在本文中即 adservice 和 dataservice 的镜像。
- 需要改造传统微服务的代码，在代码中集成 Nacos 注册中心的 SDK。
- 如果要使用 Sentinel 治理传统微服务的流量，则还需要在代码中集成 Sentinel 的 Client。

创建原生应用

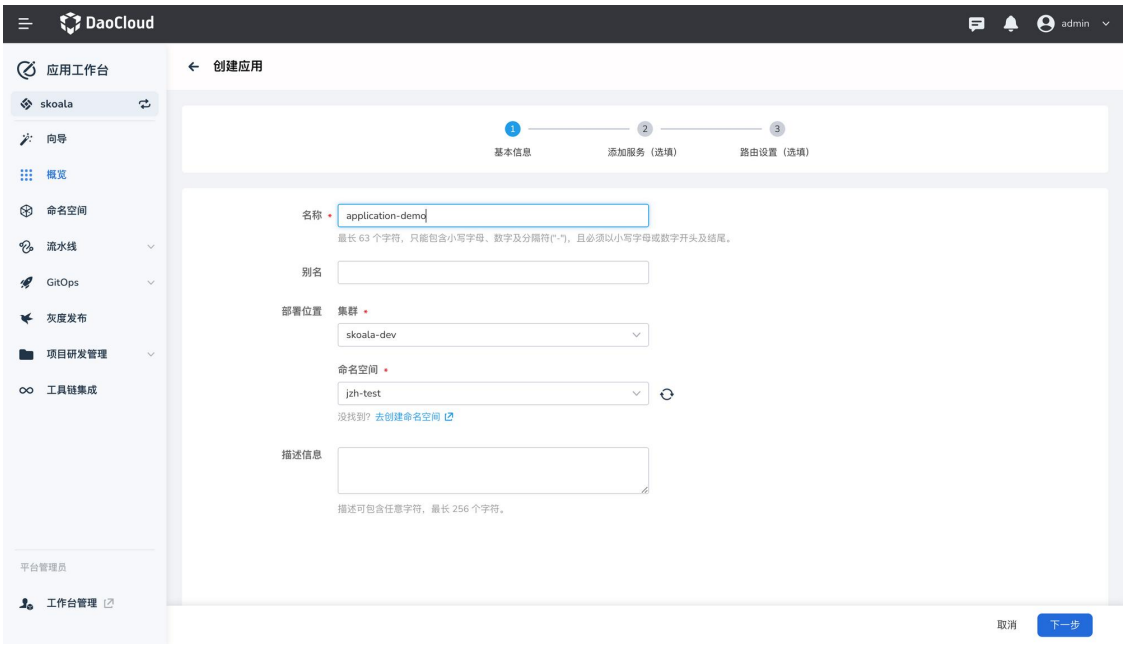
1. 在应用工作台 -> 概览页面中，点击原生应用页签，然后在右上角点击创建应用。



示意图

2. 参考以下说明填写基本信息，然后点击下一步。

   - 名称：原生应用的名称。
   - 别名：原生应用的别名。
   - 部署位置：选择将原生应用部署到哪个集群下的哪个命名空间。

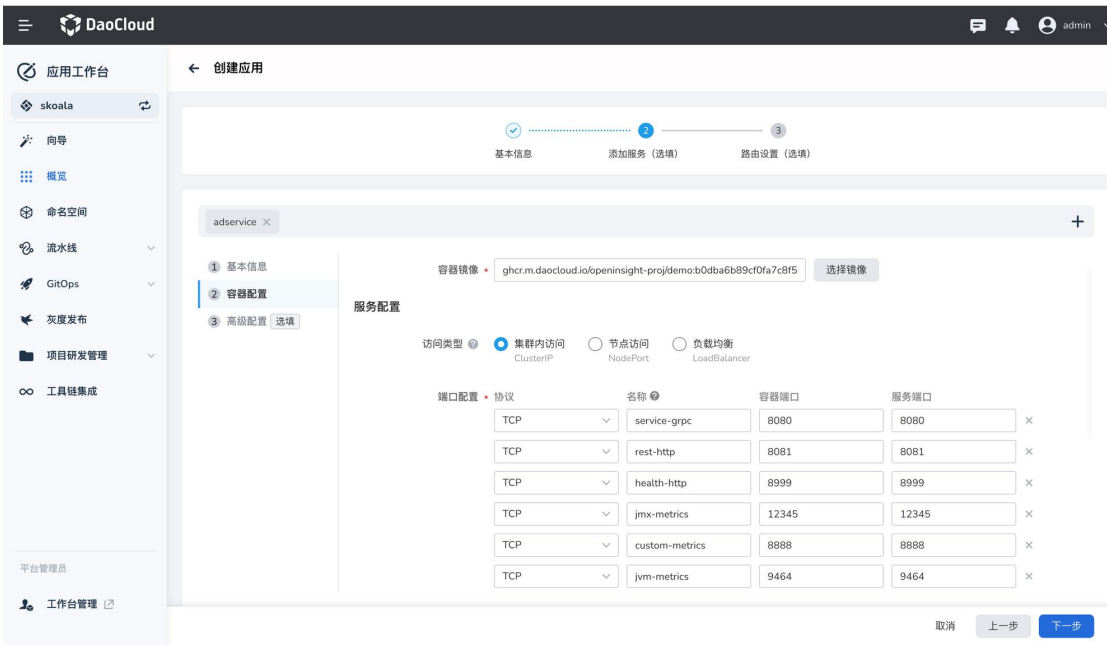


示意图

3. 参考以下说明添加传统微服务。

   添加微服务时，基本信息 和 容器配置 为必填信息，高级配置为选填信息。

   - 基本信息：设置微服务的名称、资源类型和实例数
   - 容器配置
     - 容器镜像：填写微服务的镜像地址
     - 服务配置：端口配置取决于服务代码。就此处的演示应用 adservice 而言，需要填写如下信息：

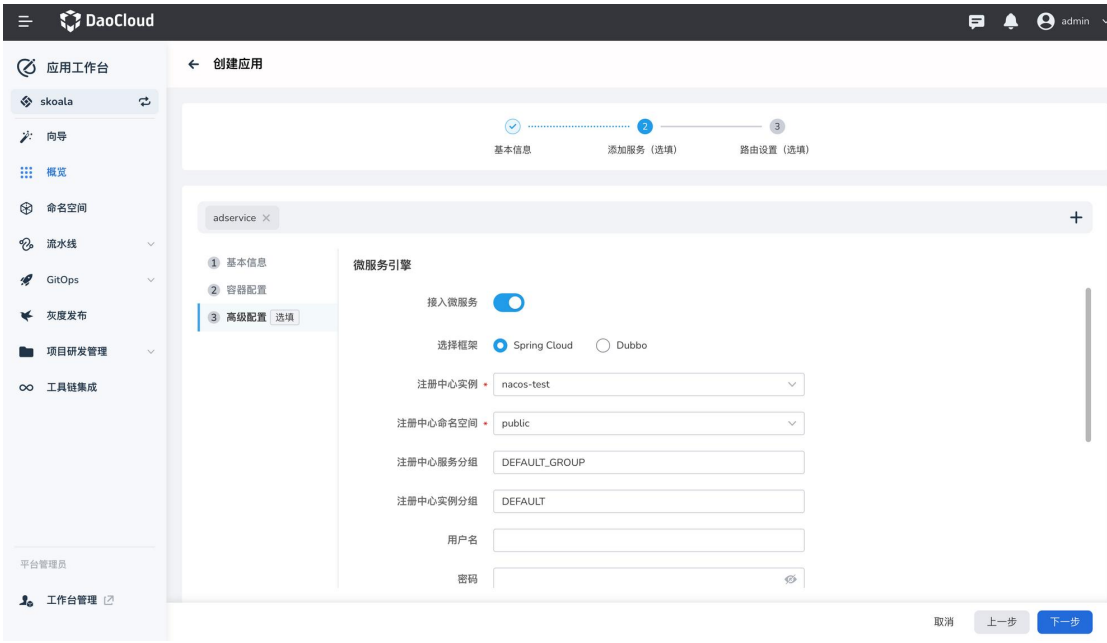


示意图

一 高级配置

- 接入微服务：即将所添加的传统微服务接入 DCE 5.0 的微服务引擎模块
- 选择框架：微服务使用的框架，例如 Spring Cloud 或 Dubbo
- 注册中心实例：选择将所添加的微服务接入哪个注册中心。可选项来自通过微服务引擎模块在当前工作空间下创建的托管 Nacos 注册中心实例。
- 注册中心命名空间：微服务应用的 Nacos 命名空间
- 注册中心服务分组：微服务应用的服务分组，即 Nacos 中的 Group 概念
- 用户名/密码：如果该注册中心实例被认证，则需要填写用户名密码

- 开启微服务治理：所选的注册中心实例应开启 Sentinel 或 Mesh 治理能力

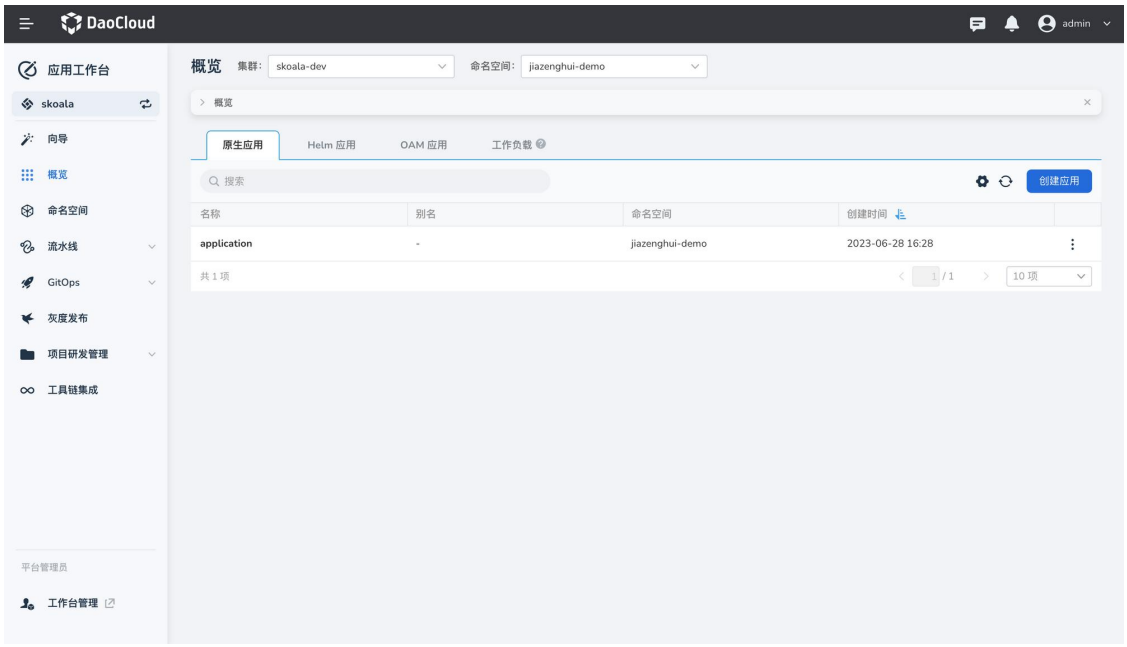


示意图

!!! note “如需添加更多微服务，点击屏幕右侧的 即可。”

![示意图](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/amamba/images/native06.png)

4. 根据需要配置路由并在右下角点击确定。

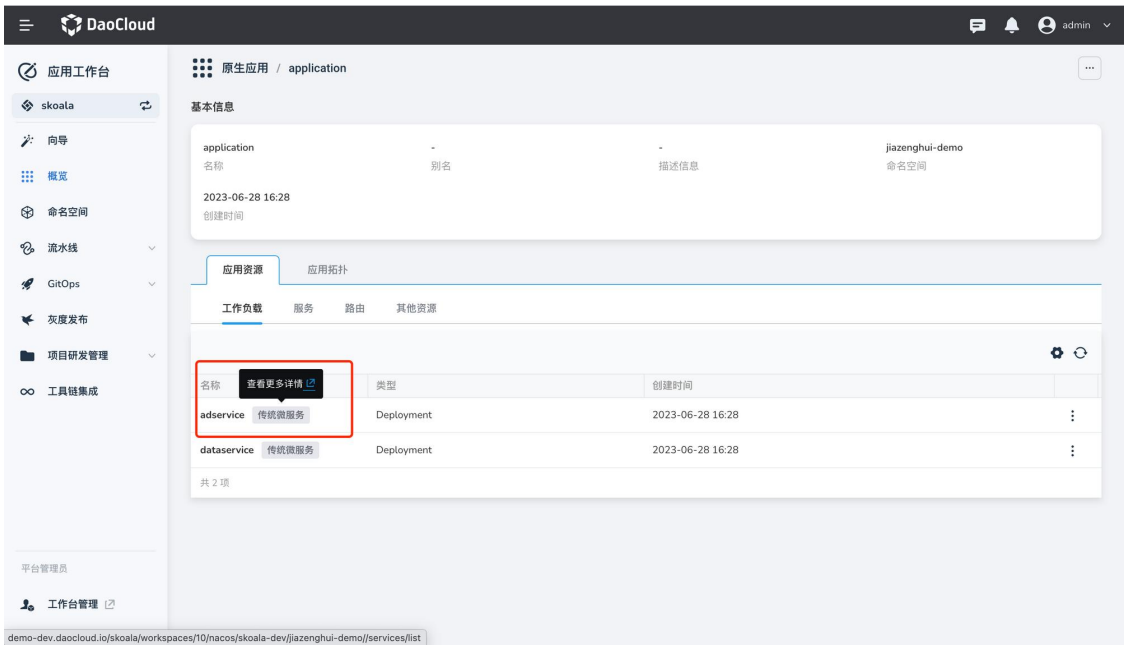


示意图

查看原生应用下的微服务

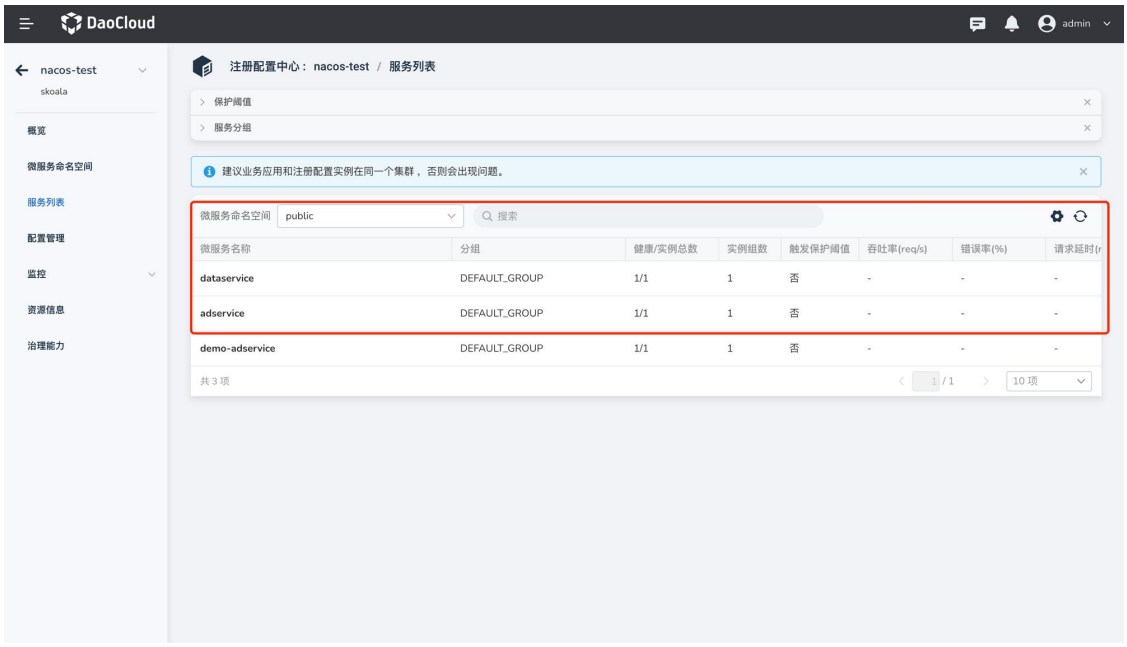
1. 应用创建成功后，点击应用名称，可以查看当前应用下的负载带有传统微服务标识。

   点击弹出的黑色链接可以跳转到微服务引擎下的服务详情页面。



示意图

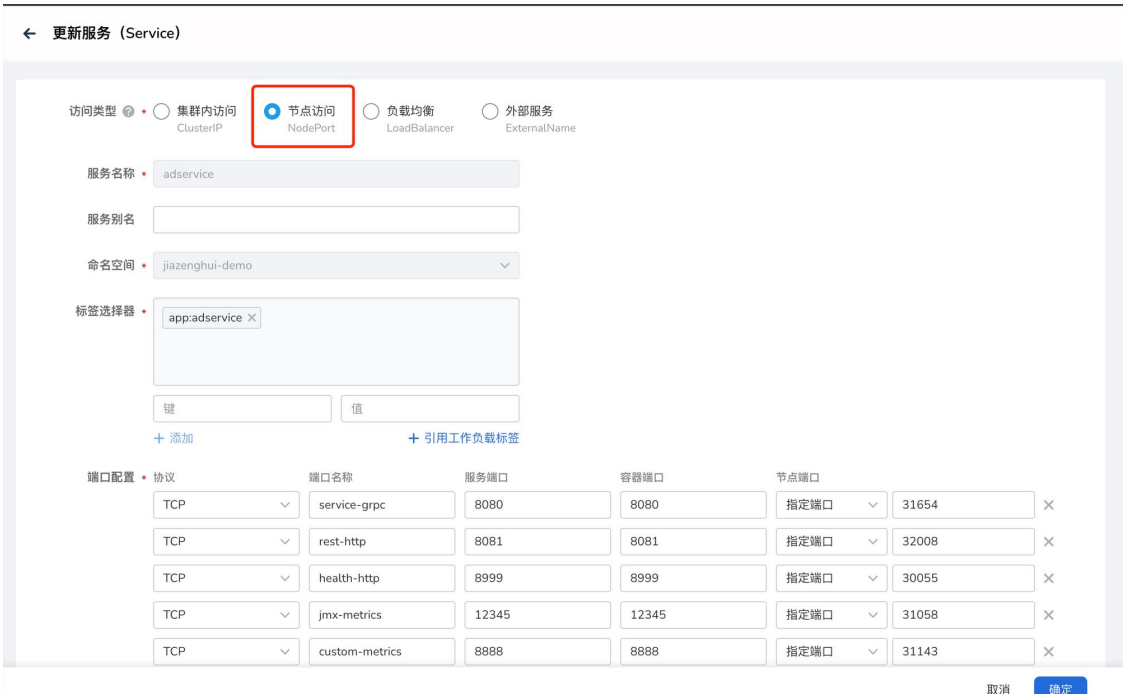
2. 也可以前往微服务引擎模块，在对应的工作空间和注册中心下查看所添加的服务。



示意图

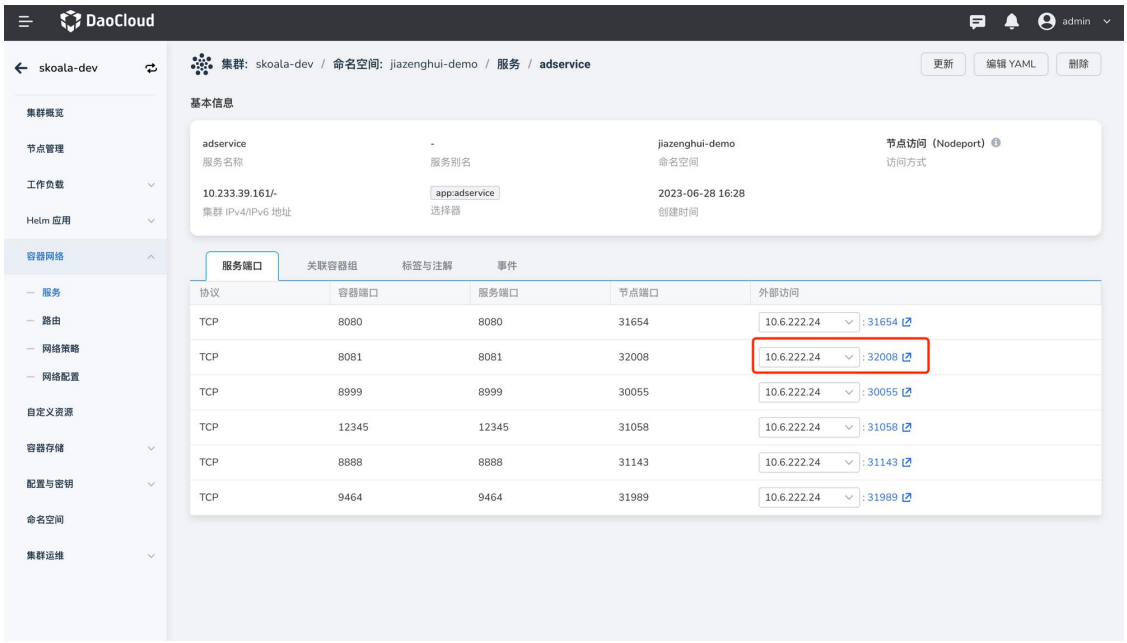
模拟服务调用

1. 首先在容器管理平台将 adservice 的服务的访问类型更改为 NodePort。



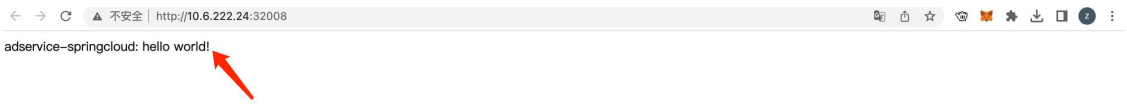
示意图

2. 在服务详情页面点击服务端口为 8081 的外部访问地址。



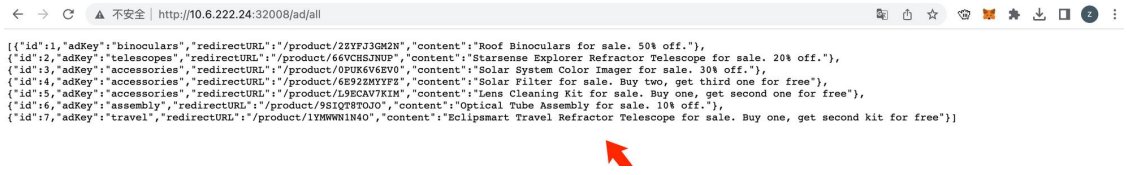
示意图

3. 浏览器新开标签出现如下页面，表示 adservice 部署成功。



示意图

4. 在 URL 地址后增加 /ad/all 进行访问，出现下图表示调用 dataservice 服务成功。



示意图

基于 Helm 模板部署 Helm 应用

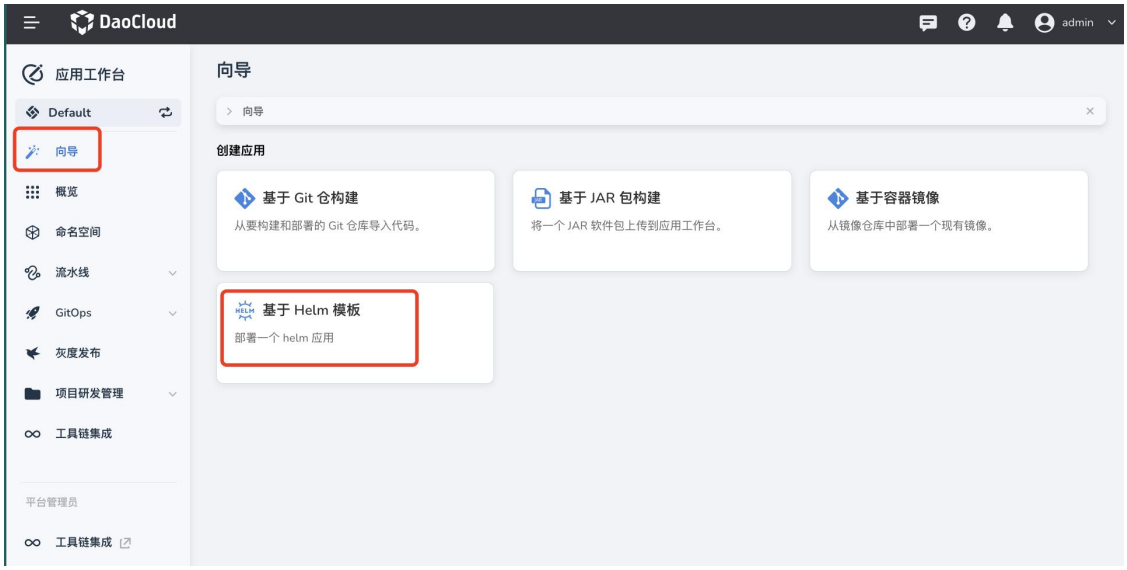
应用工作台支持通过 Git 仓库、Jar 包、容器镜像、Helm 模板等四种方式构建应用。本文介绍如何通过 Helm 模板部署 Helm 应用。

前提条件

- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 workspace edit 角色。 参考创建工作空间、用户和角色。

操作步骤

1. 进入应用工作台模块后，在左侧导航栏点击向导，然后选择基于 Helm 模板。

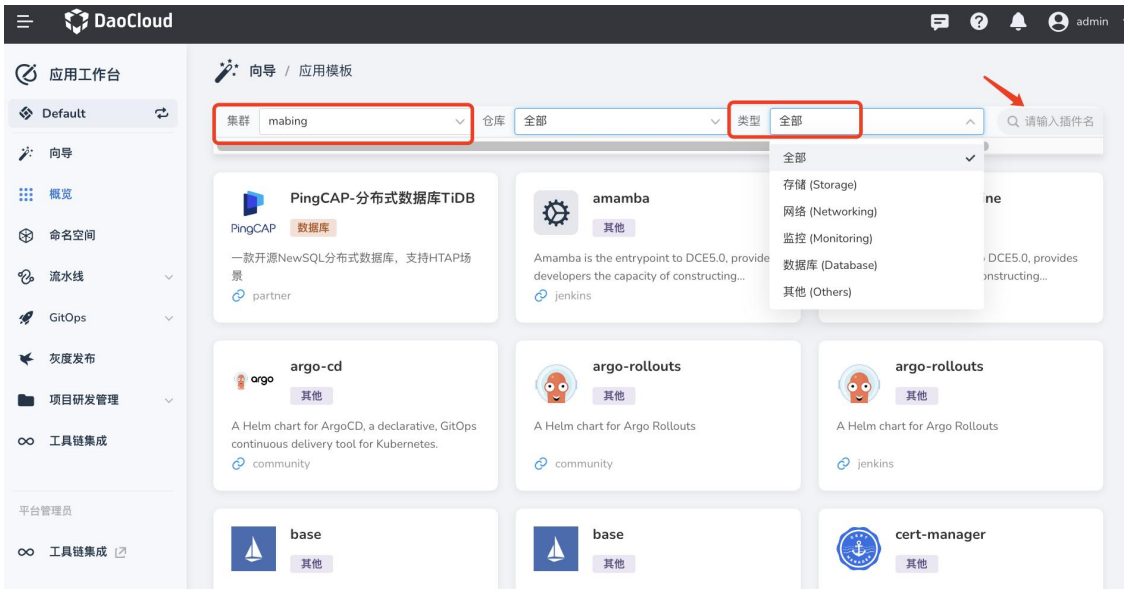


基于 helm

2. 在页面上方选择需要将应用部署在哪个集群，然后点击需要部署的 Helm Chart 卡片，例如“docker-registry”。

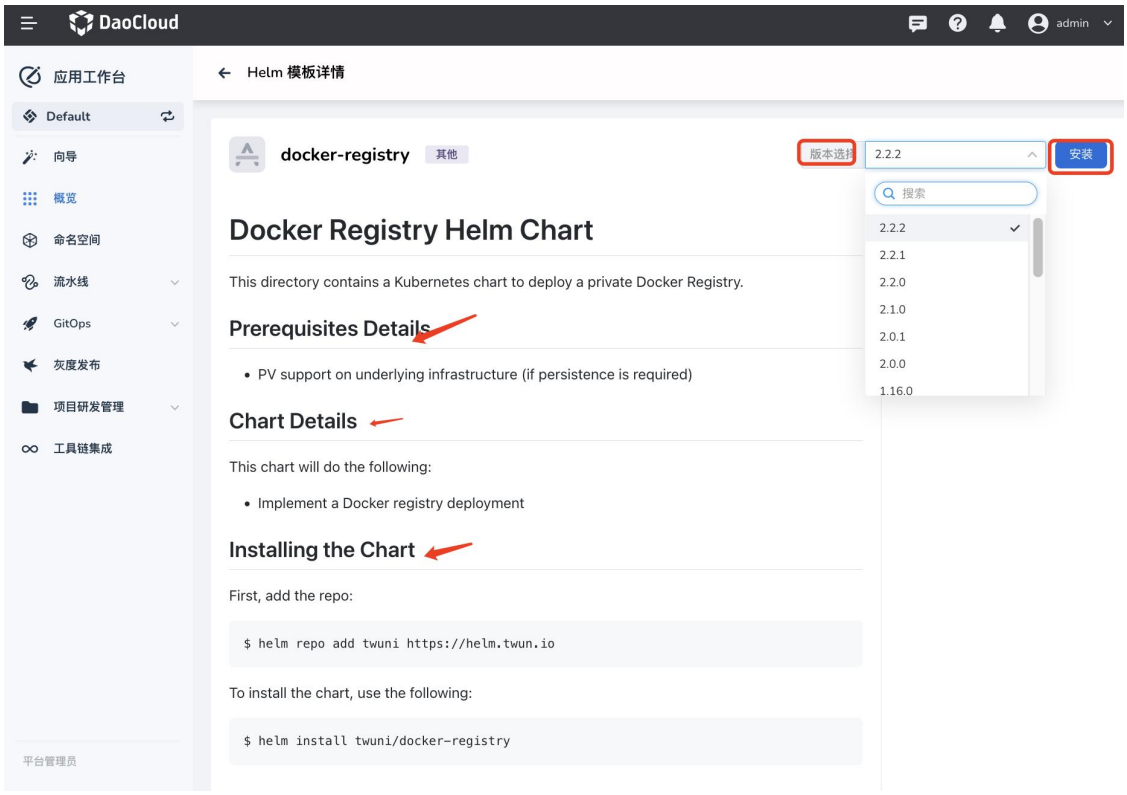
!!! note

- 点击`仓库`右侧的列表可以筛选 Helm 仓库
- 点击`类型`右侧的列表可以筛选 Helm Chart 的类型
- 也可以在在右侧的搜索框直接输入 Chart 名称快速找到需要部署的 Helm 应用



基本信息

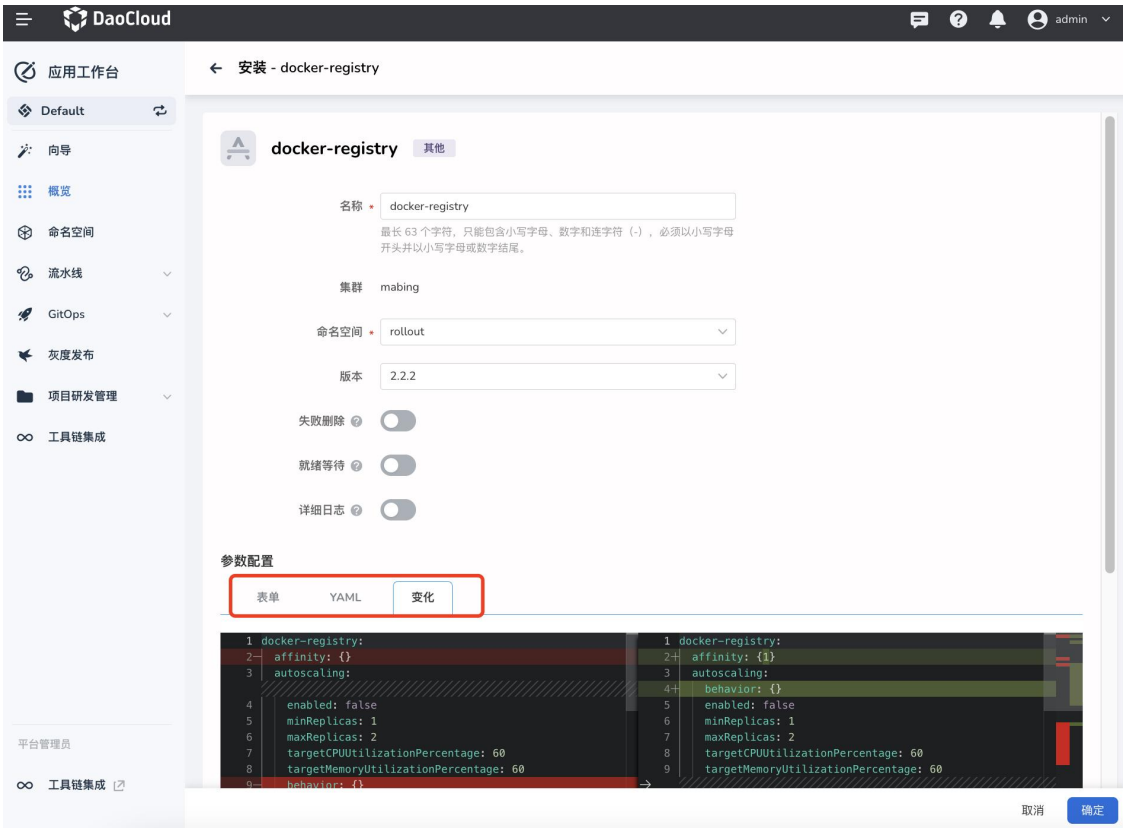
3. 阅读该应用的安装前提、参数配置说明等信息，在右上角选择需要安装版本，然后点击安装。



基本信息

4. 设置应用名称、命名空间等基本信息，然后在下方通过表单或 YAML 配置参数，最后在页面底部点击确定。

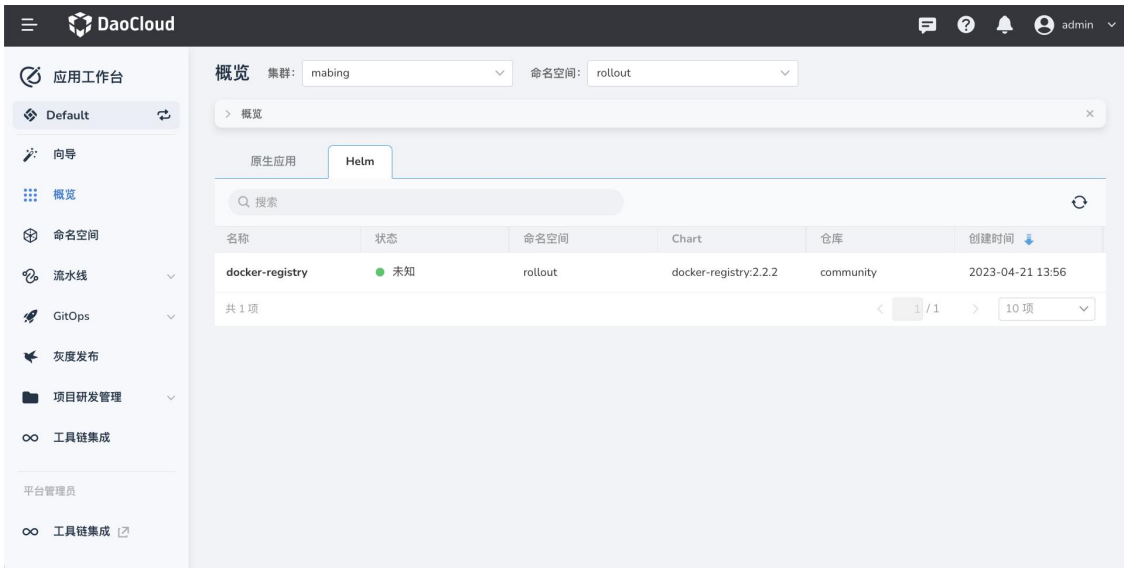
    - 就绪等待：启用后，将等待应用下的所有关联资源都处于就绪状态，才会标记应用安装成功。
    - 失败删除：如果插件安装失败，则删除已经安装的关联资源。开启后，将默认同步开启就绪等待。
    - 详情日志：开启后，将记录安装过程的详细日志。
    - 点击参数配置下方的变化页签可使用对比视图查看参数变更情况。



容器配置

5. 页面自动跳转到概览页面下的 Helm 应用列表，可查看当前集群下已经安装的 Helm 应用。

   点击应用名称可以跳转到容器管理模块查看应用详情。



高级配置

!!! note

如需更新删除该 Helm 应用，需要点击应用名称跳转到容器管理模块，在应用详情页面进行更新与删除以及更多操作。

![高级配置](https://docs.daocloud.io/daocloud-docs-images/docs/amamba/images/helm06.png)

基于 Jar 包部署 Java 应用

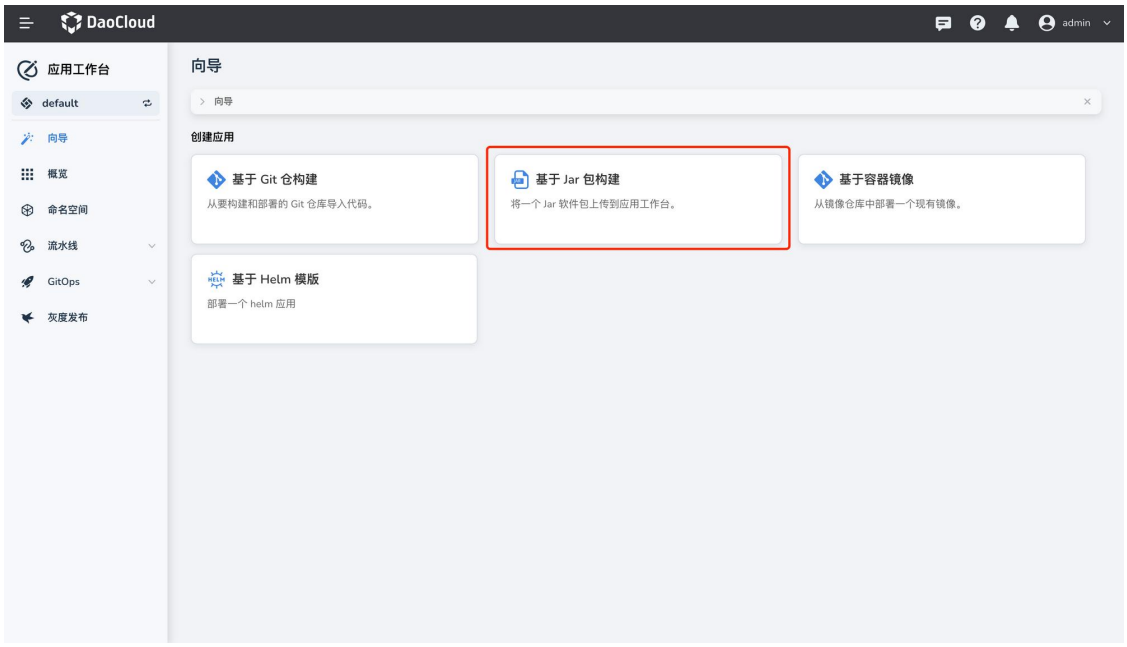
应用工作台支持通过 Git 仓库、Jar 包、容器镜像、Helm 模板等四种方式构建应用。本文介绍如何通过 Jar 文件部署 Java 应用程序。

前提条件

1. 创建一个工作空间和一个用户，该用户需加入该工作空间并具备 Workspace Editor 角色。
2. 创建访问镜像仓库的凭证，例如 registry。
3. 准备一个镜像仓库，例如 Harbor 仓库。

操作步骤

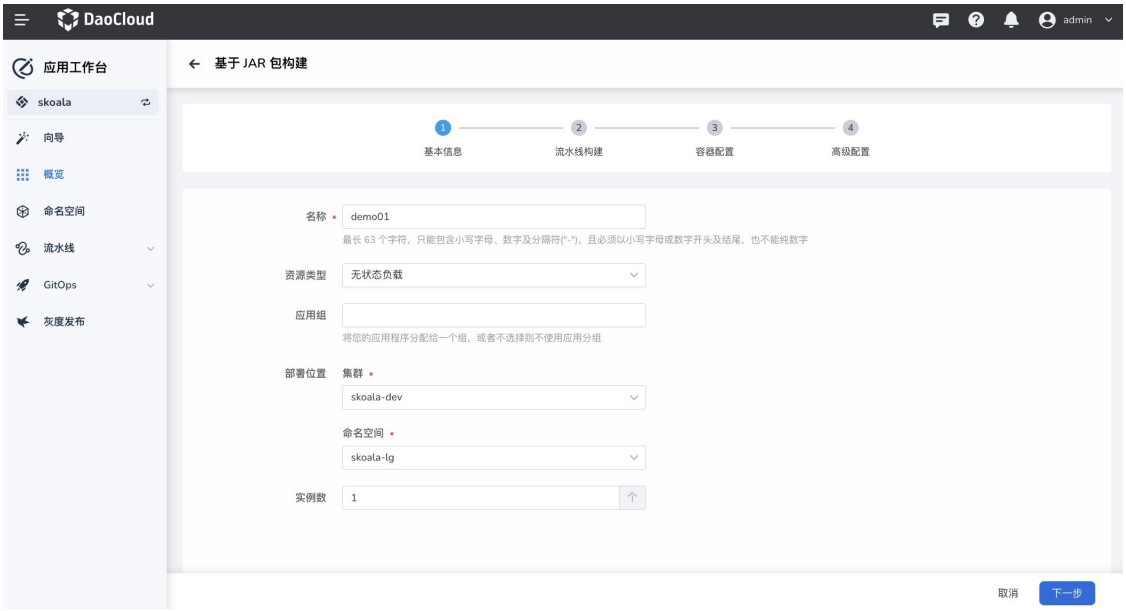
1. 在左侧导航栏点击向导，然后选择基于 Jar 包构建。



基于 jar 包

2. 参考下列要求填写基本信息，然后点击下一步。

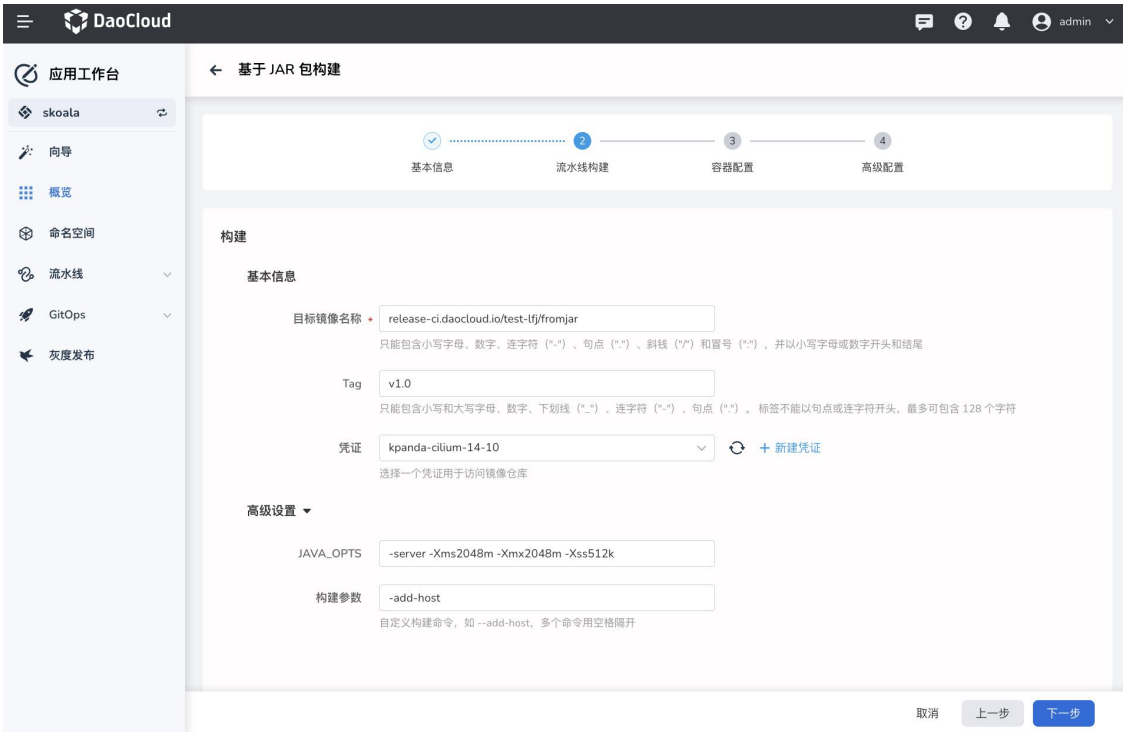
- 名称：最长 63 个字符，只能包含小写字母、数字及分隔符(“-”)，且必须以小写字母或数字开头及结尾。
- 资源类型：选择要创建的应用属于无状态负载，还是有状态负载。
- 应用组：为应用选择所属的分组。为空表示不对此应用进行分组。
- 部署位置：选择将应用部署在哪个集群下的哪个命名空间。只能选择当前工作空间下存在的集群。
- 实例数：为应用设置 Pod 的数量。



基本信息

3. 参考下列要求配置流水线，然后点击下一步。
   - 目标镜像名称：为目标镜像命名，需包含目标镜像的存储路径，例如 release-ci.daocloud.io/test-lfj/fromjar。
   - Tag：为目标镜像打标签，例如版本号 v1.0。
   - 凭证：选择访问镜像仓库的凭证，例如 registry-credential。

- JAVA_OPTS：用来设置 JVM 相关运行参数的变量，例如 –server –Xms2048m –Xmx2048m –Xss512k。
- 构建参数：构建参数会以 ––build–arg 的形式传递到 build 命令中，支持将上游制品下载地址、上游镜像下载地址设置为参数，也支持自定义任意参数。

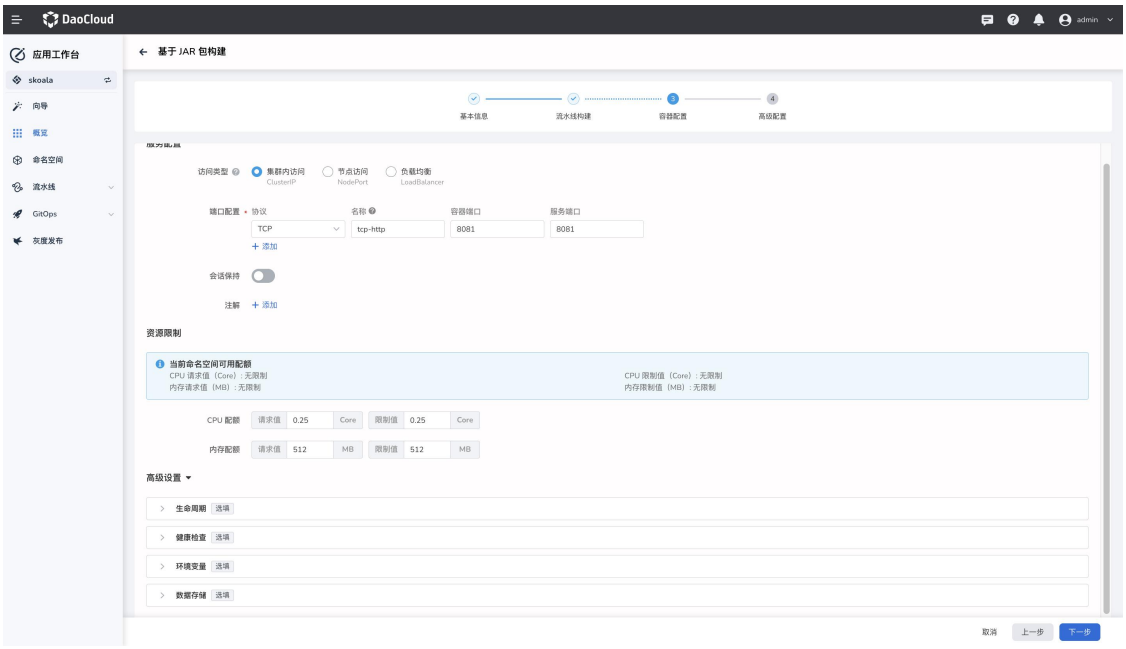


流水线构建

4. 参考下列要求填写容器配置，然后点击下一步。
   - 访问类型：支持通过 clusterIP 仅允许在集群内访问该应用，或者通过 NodePort 允许在集群外部访问，或者通过负载均衡器进行访问。
   - 端口配置：根据实际业务场景填写需要暴露的端口号。

     有关服务配置的更多详细说明，可参考创建服务。

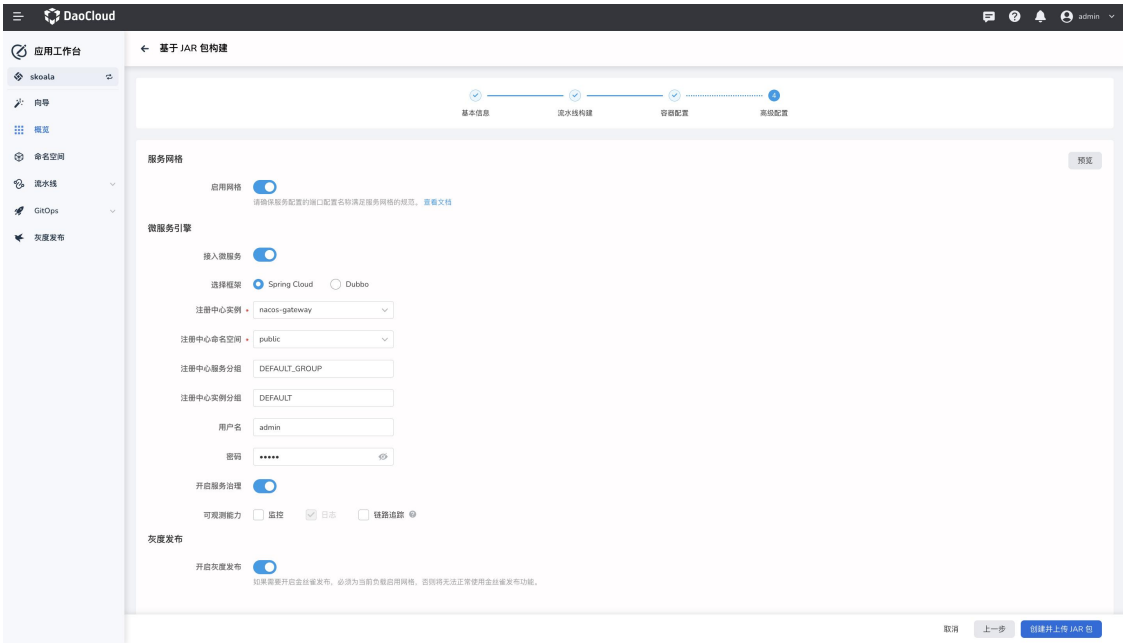
- 资源限制：CPU 和内存配额不得超出应用所在的命名空间在当前工作空间中的剩余资源。
- 生命周期：设置容器启动时、启动后、停止前需要执行的命令。详情可参考容器生命周期配置。
- 健康检查：用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考容器健康检查配置。
- 环境变量：配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考容器环境变量配置。
- 数据存储：配置容器挂载数据卷和数据持久化的设置。



容器配置

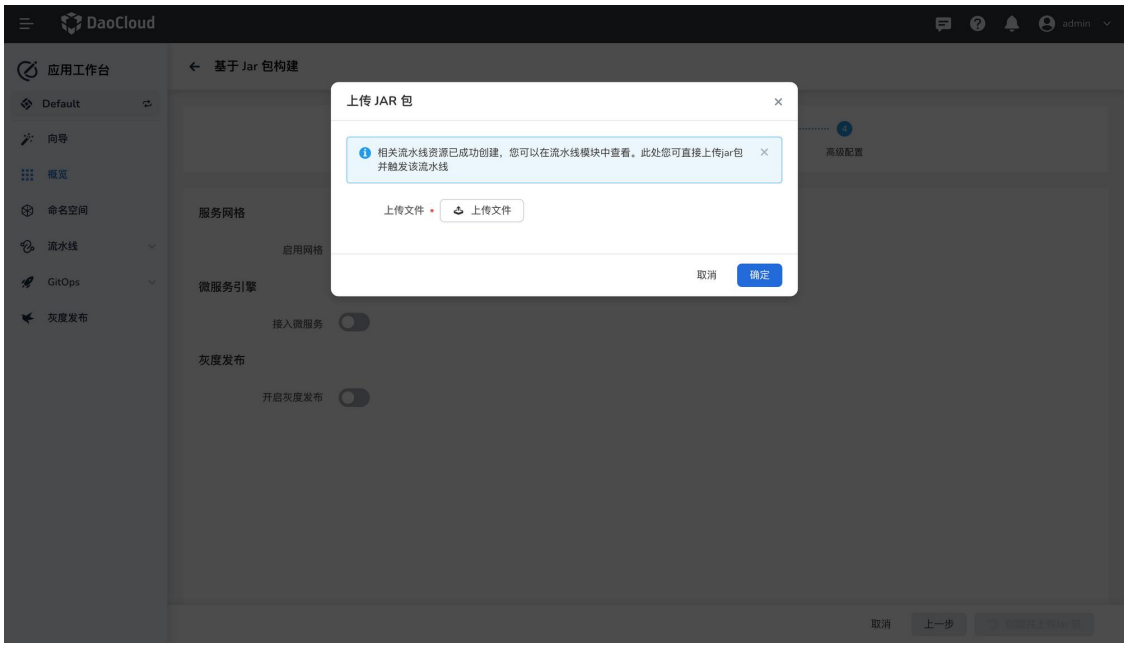
5. 参考下列说明选择是否开启高级功能，然后点击创建并上传 Jar 包。
   - 服务网格：选择是否启用 DCE 5.0 的服务网格模块来治理微服务流量。

- 微服务引擎：是否将新创建的应用接入 DCE 5.0 的微服务引擎模块。 > 有关微服务引擎的配置，可参考基于 Git 仓构建微服务应用。
- 灰度发布：选择是否开启灰度发布。有关灰度发布的更多内容，可参考金丝雀发布。



高级配置

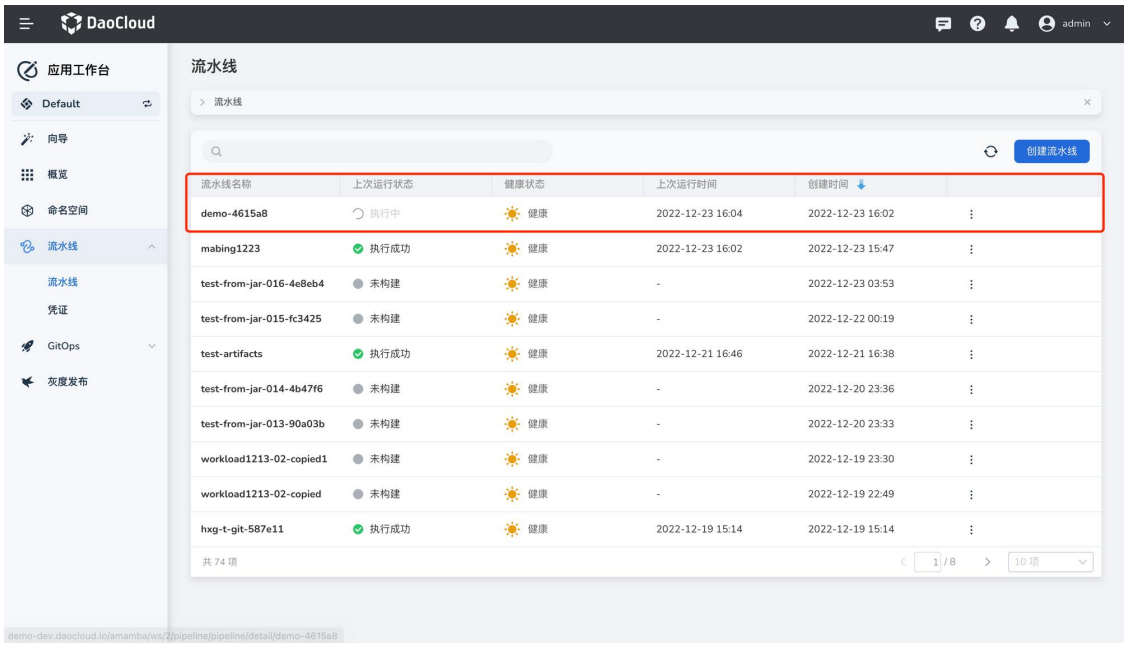
6. 选择需要上传的文件，点击确定 。



上传文件

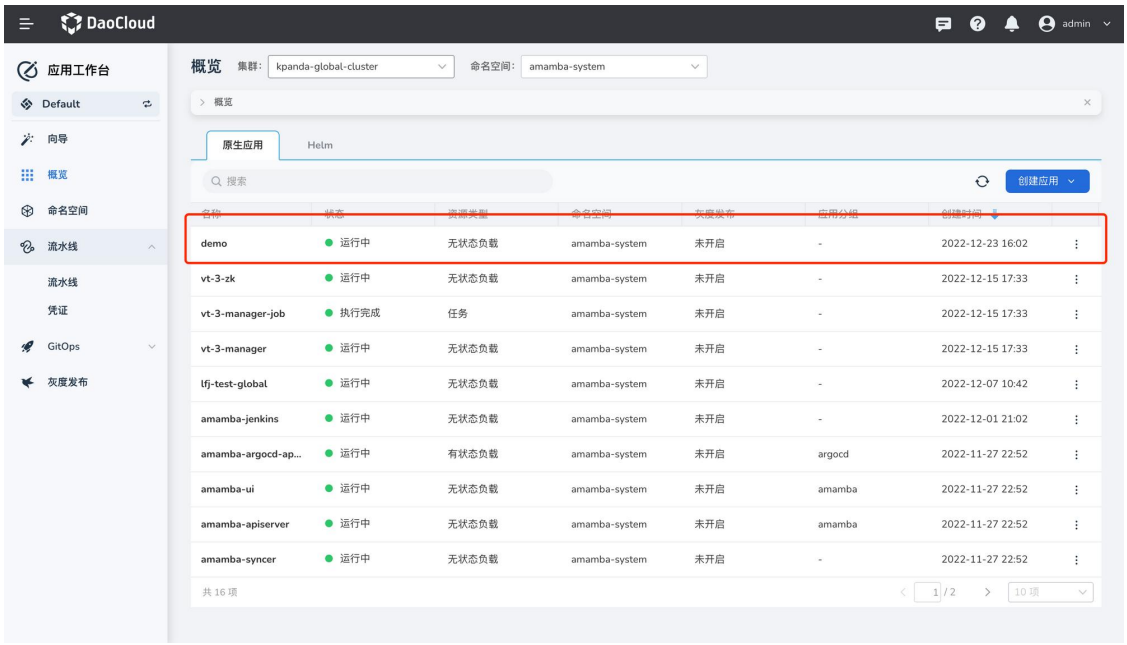
7. 创建成功后会触发运行对应的流水线，在左侧导航栏点击流水线可查看其运行状态。

   流水线的命名规则为“对应的应用名称–随机数”，例如通过流水线名称 demo–4615a8 即可得知对应的应用名为 demo。



运行流水线

8. 等待流水线执行成功后，在左侧导航栏点击概览，选择原生应用页签即可查看新建的应用。



创建成功

命名空间管理

命名空间是 Kubernetes 中用来隔离资源的一种抽象。

前提条件

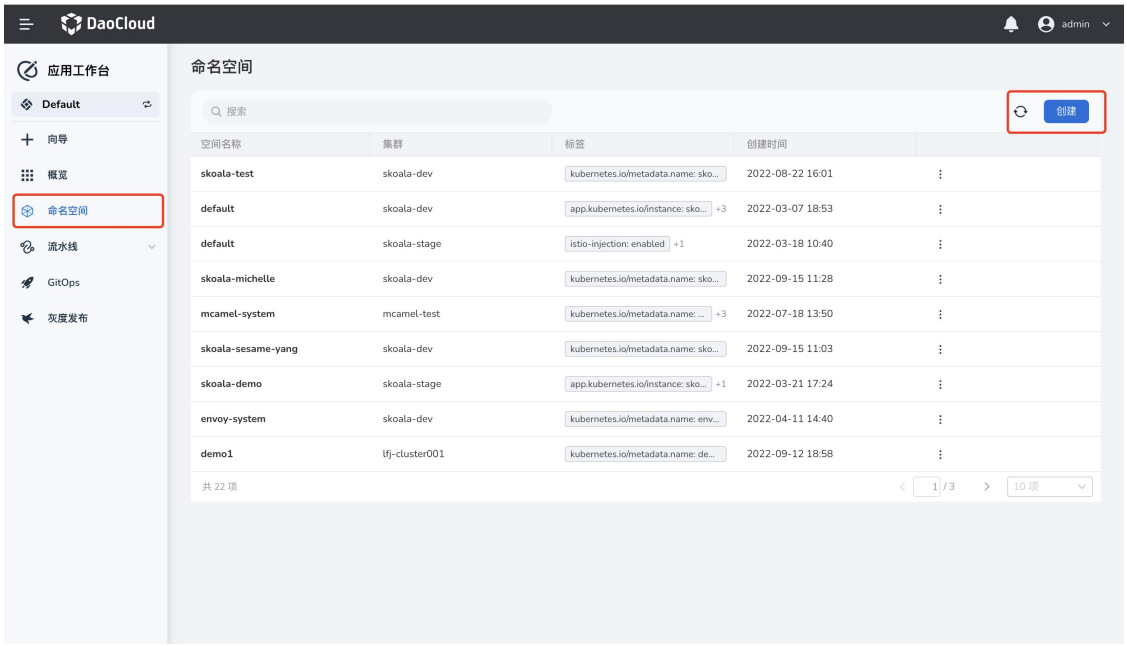
- 当前工作空间已经拥有集群资源，参考绑定资源。
- 当前用户授权为 Workspace Admin 角色，详情参考工作空间最佳实践。

创建命名空间

!!! note

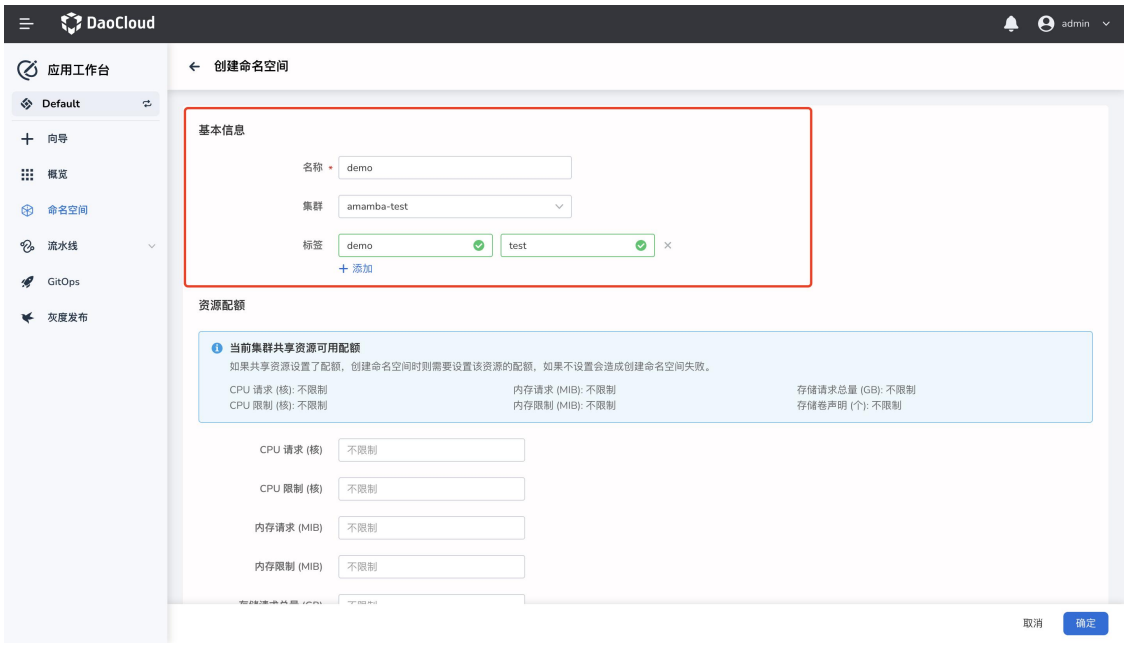
仅支持 `Workspace Admin` 创建命名空间，并设置命名空间配额，其他角色均不支持此操作。

1. 在左侧导航栏中点击命名空间，进入命名空间列表，点击右上角的创建。



namespace-listpng

2. 在创建命名空间页面中，配置命名空间的基本信息。



namespace-create01

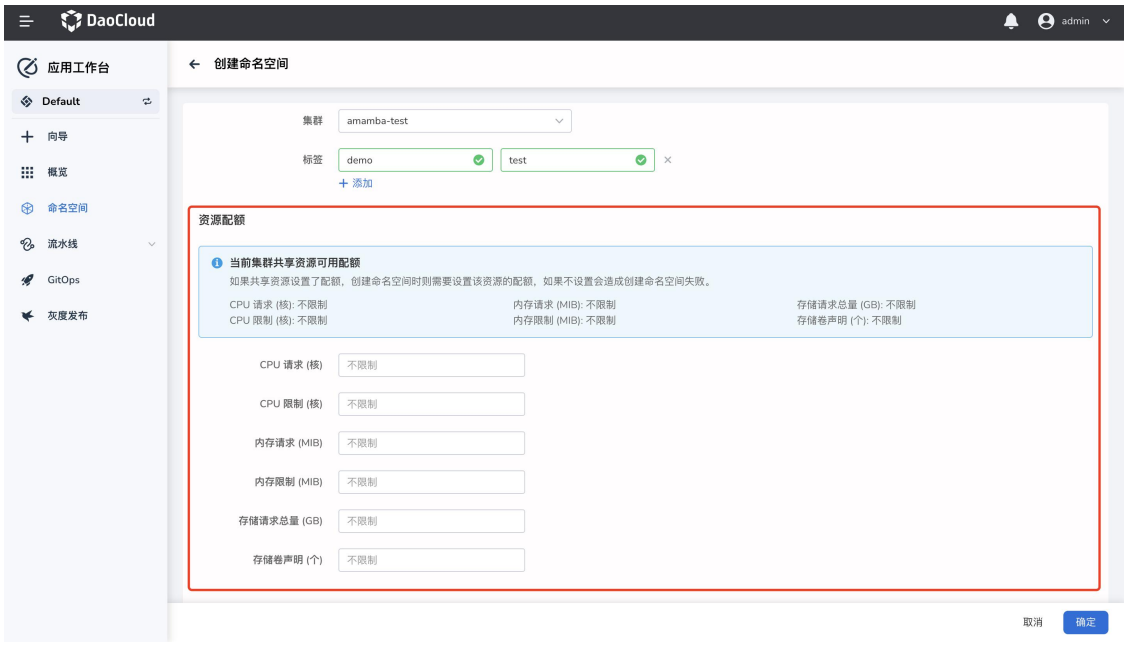
- 名称：设置命名空间的名称
- 集群：从当前工作空间下绑定的所有集群中选择一个集群
- 标签：设置命名空间的标签

3. 基本信息设置完成后，还需要配置命名空间的资源配额。

!!! note

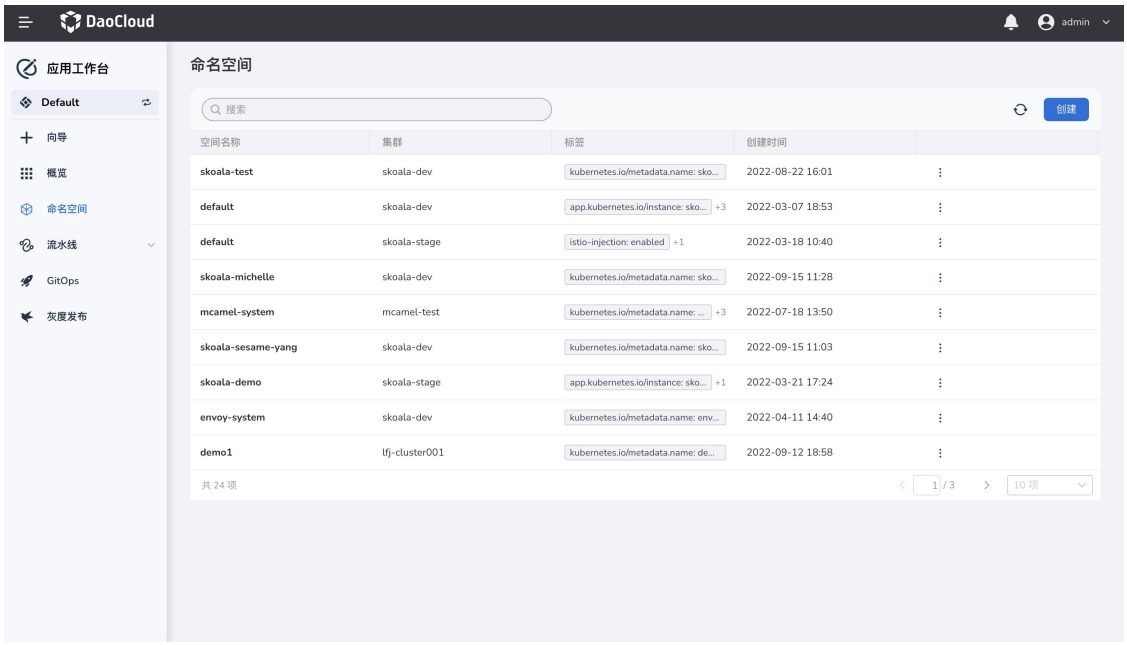
- 如果当前集群在工作空间设置了任何资源配额，则必须要在创建命名空间时为命名空间设置此资源配额。
- 设置命名空间的资源配额不能超过当前集群在该工作空间设置的资源使用上限。
- 资源配额的请求或限制字段为空意味不为当前命名空间设置任何配额。

目前工作空间中支持的资源配额有：CPU 请求、CPU 限制、内存请求、内存限制、存储请求总量、存储卷声明。其中 CPU、内存资源的限制值必须大于请求值。



namespace-create02

4. 点击确定完成资源创建。屏幕提示创建成功，返回命名空间列表页面。



namespace-list01

5. 点击列表右侧的 ：，可以在弹出菜单中选择配额管理、编辑标签、删除等操作。

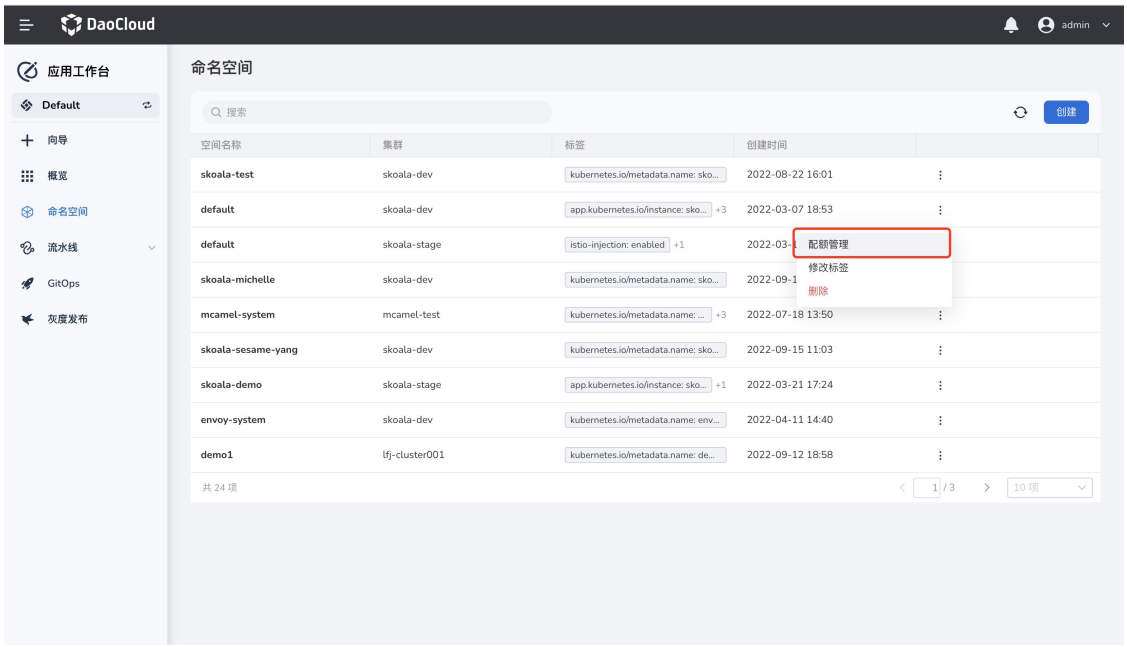
!!! warning

    如果删除了某个命名空间，会删除该命名空间下所有的资源，请谨慎操作。

命名空间配额

除了在创建命名空间中需要的 CPU 请求、CPU 限制、内存请求、内存限制、存储请求总量、存储卷声明资源配额，在命名空间配额管理中还可以设置其他资源配额，例如命名空间下的容器组、无状态负载、有状态负载、普通任务、定时任务等资源。

1. 在命名空间列表页面，选择某一个命名空间，点击配额管理。



namespace-quota

2. 在弹出的资源配额对话框中，可以看到目前命名空间的资源配额信息。在应用资源下点击添加，选择一个资源并设置配额，具体请参阅 Kubernetes 资源配额。



namespace-quota01

3.点击确定完成配额设置。

创建金丝雀发布任务

应用工作台基于开源项目 Argo Rollout 提供强大的灰度发布能力。灰度发布可以在不影响老版本的前提下，发布新的应用版本。按照预先定义的规则，渐进式地将流量切换到新版本，当新版本运行无问题后，最终自动将全部流量从老版本迁移至新版本。

前提条件

1. 创建一个工作空间和一个用户，该用户需加入该工作空间并具备 Workspace Editor 角色。

- 创建一个应用并开启灰度发布，可参考基于 Git 仓构建微服务应用、基于 Jar 包部署 Java 应用。
- 发布对象所在的集群已经安装了 Istio 和 Argo Rollout 两个组件。

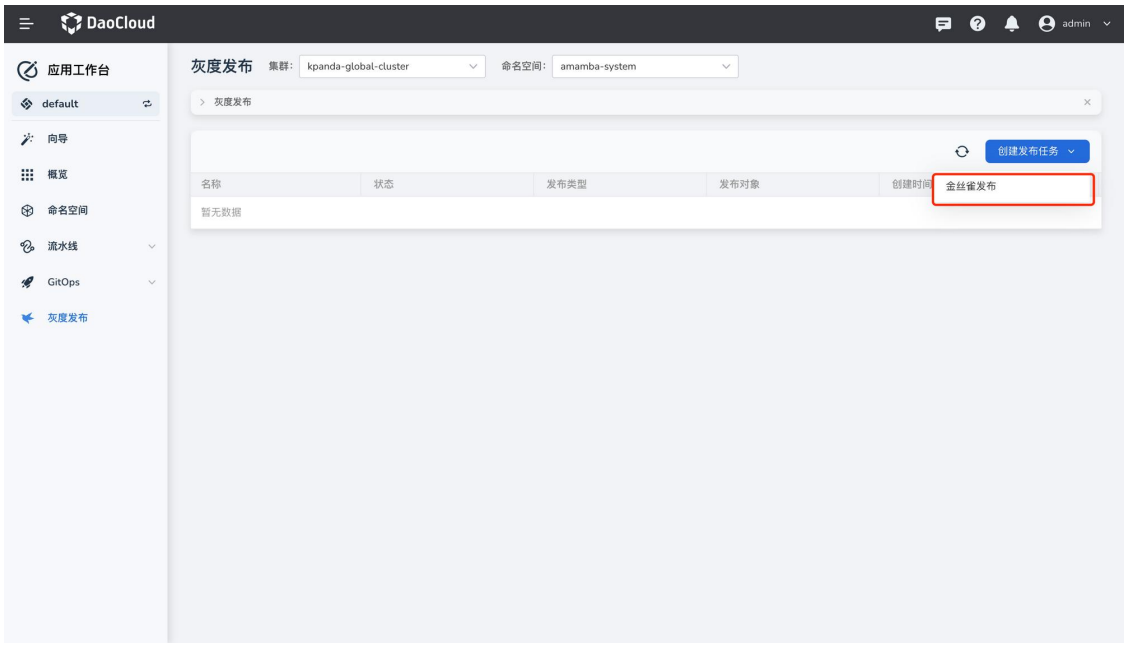
Istio 安装方式

您可以根据不同的业务场景需求，使用以下两种安装方案在集群中安装 Istio。

- 参考 Istio 官方安装指南
- 通过平台的服务网格模块创建专有网格

操作步骤

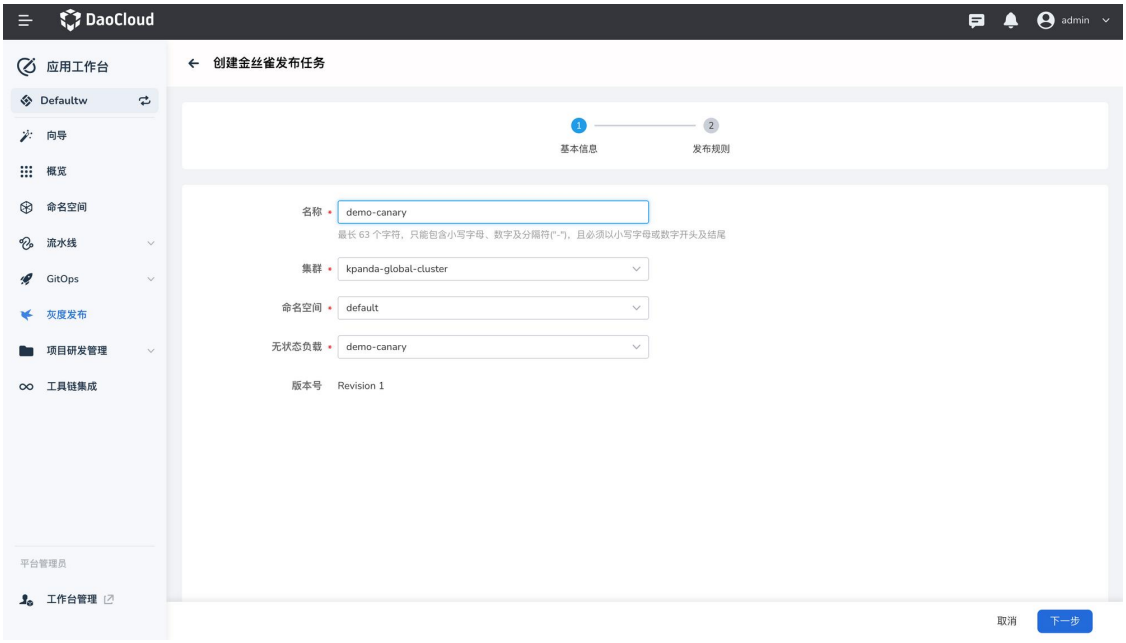
1. 进入应用工作台模块，在左侧导航栏点击 灰度发布，然后在页面右上角点击创建发布任务->金丝雀发布。



进入创建页面

2. 参考下列要求填写基本信息，然后点击下一步。

   - 名称：填写发布任务的名称。最长 63 个字符，只能包含小写字母、数字及分隔符(“-”)，且必须以小写字母或数字开头及结尾
   - 集群：选择发布对象所在的集群。需要确保该集群已经部署了 Istio 和 Argo Rollout。
   - 命名空间：选择发布对象所在的命名空间，且该命名空间中已经部署了的工作负载并且开启了服务网格。
   - 无状态负载：选择具体的发布对象。



基本信息

3. 参考下列说明配置发布规则。

   - 实例数量：进行灰度发布任务时应用的副本数量。
   - 流量管理类型： 支持副本数、Istio、Nginx。

===“基于副本数”

- 流量调度类型：仅支持基于权重的灰度发布

- 流量调度策略：

  - 当前阶段发布流量比例：每个流量周期内，为灰度版本增加的流量比例。
  - 到达流量比例后等待时间：为灰度版本增加流量周期，即等待多久后自动进入下一个灰度流量比例。

> 如果设置为 0 或不填写，发布任务执行到此步骤时会永久暂停。

![副本数](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/amamba/images/canary11.png)

===“基于 Istio”

– 流量调度类型：基于 Istio 支持基于权重、基于请求特征的灰度发布，目前仅支持基于权重的灰度发布。

– 流量调度策略：

– 当前阶段发布流量比例：每个流量周期内，为灰度版本增加的流量比例。
– 到达流量比例后等待时间：为灰度版本增加流量周期，即等待多久后自动进入下一个灰度流量比例。

> 如果设置为 0 或不填写，发布任务执行到此步骤时会永久暂停。

– 监控分析：开启后即可利用 Prometheus 的能力进行监控。根据预先定义的监控指标以及采集间隔，在整个发布过程中自动执行监控分析。如果不满足定义的规则，则自动回滚至旧版本，灰度发布失败。

![副本数](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/amamba/images/canary12.png)

===“基于 Nginx”

– 流量调度类型：基于 Nginx 仅支持基于权重的灰度发布。

– 服务：选择当前发布对象关联的 Service 对象

– 路由：选择当前发布对象关联的 Ingress 对象，用于配置流量调度策略/

– 流量调度策略：

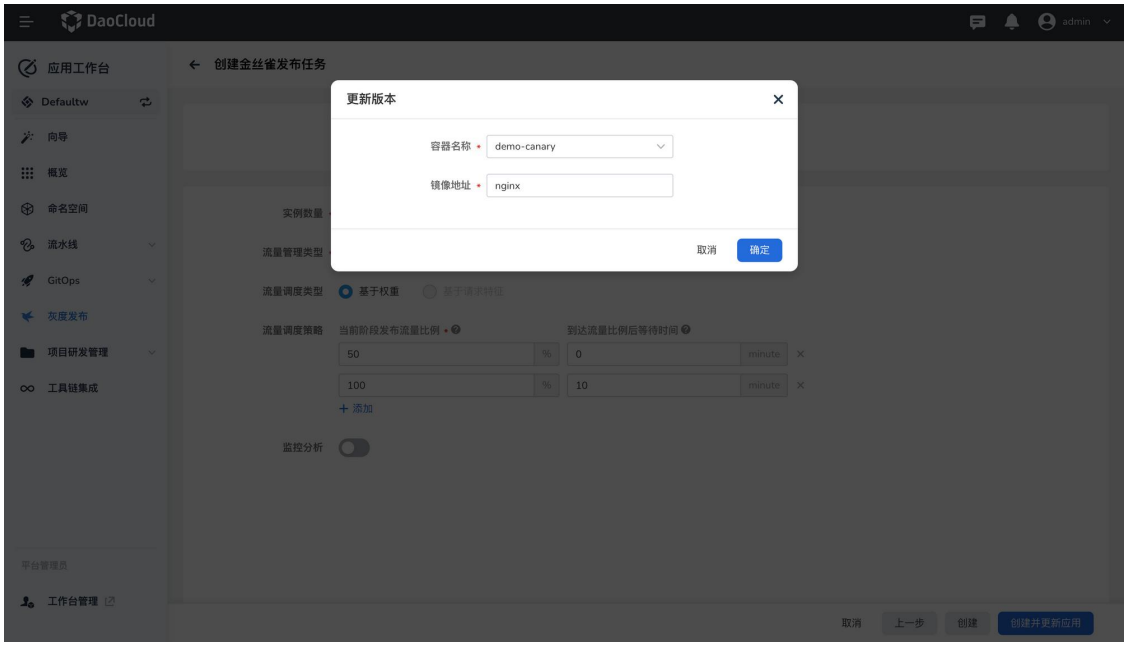
– 当前阶段发布流量比例：每个流量周期内，为灰度版本增加的流量比例。
– 到达流量比例后等待时间：为灰度版本增加流量周期，即等待多久后自动进入下一个灰度流量比例。

> 如果设置为 0 或不填写，发布任务执行到此步骤时会永久暂停。

![副本数](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/amamba/images/canary14.png)

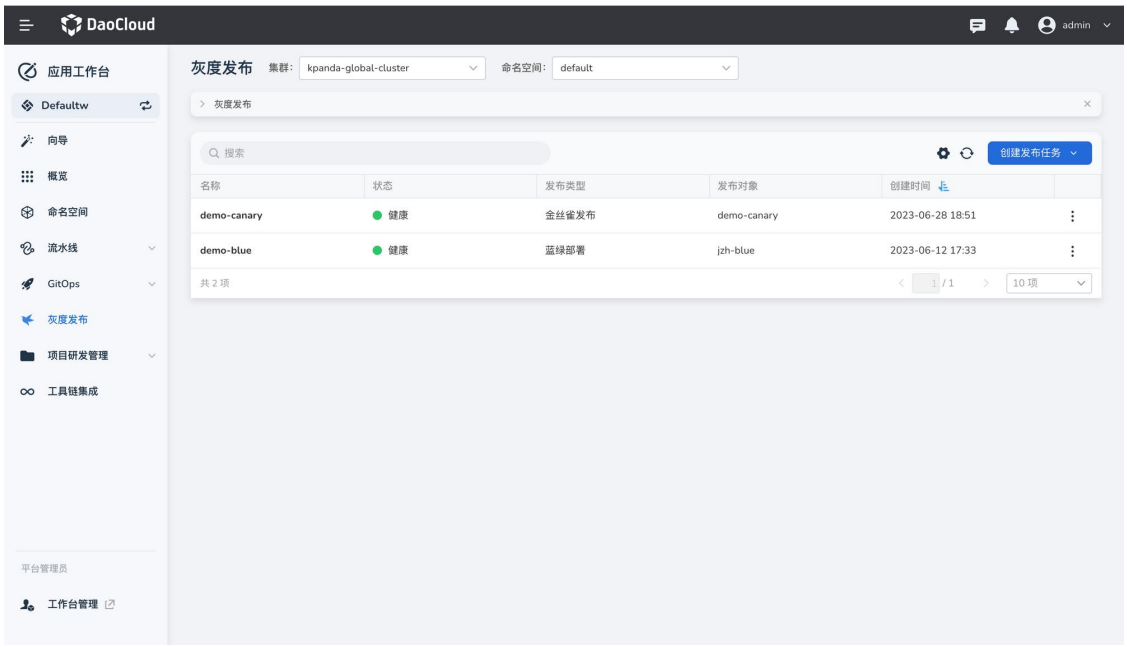
4. 在页面底部点击创建并更新应用，然后在弹框中设置灰度版本的镜像地址并点击确定。

   此时原工作负载的副本数会被设置为 0。



填写镜像地址

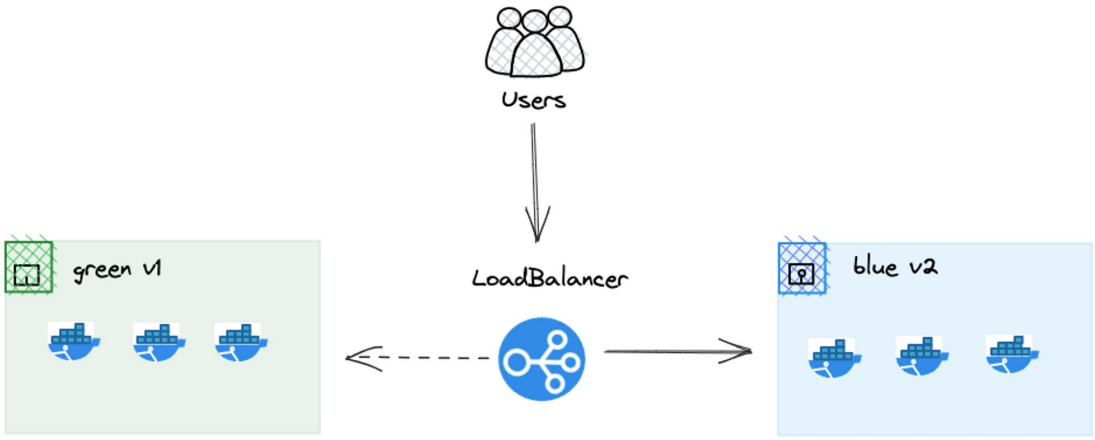
5. 系统自动跳转至灰度发布的任务列表页面，提示更新版本成功。


DaoCloud
admin
应用工作台
Defaultw
向导
概览
命名空间
流水线
GitOps
灰度发布
项目研发管理
工具链集成
平台管理员
工作台管理
灰度发布
集群:
kpanda-global-cluster
命名空间:
default
灰度发布
搜索
创建发布任务
名称
状态
发布类型
发布对象
创建时间
demo-canary
健康
金丝雀发布
demo-canary
2023-06-28 18:51
demo-blue
健康
蓝绿部署
jzh-blue
2023-06-12 17:33
共 2 项
1 / 1
10 项

返回列表

创建蓝绿发布任务

蓝绿部署，通过流量控制，可以将用户流量从旧版本逐渐迁移到新版本，同时旧版本会保持在线运行待机，一旦期间发生任何异常，就可以立马切换流量到旧版本，从而实现快速回滚。



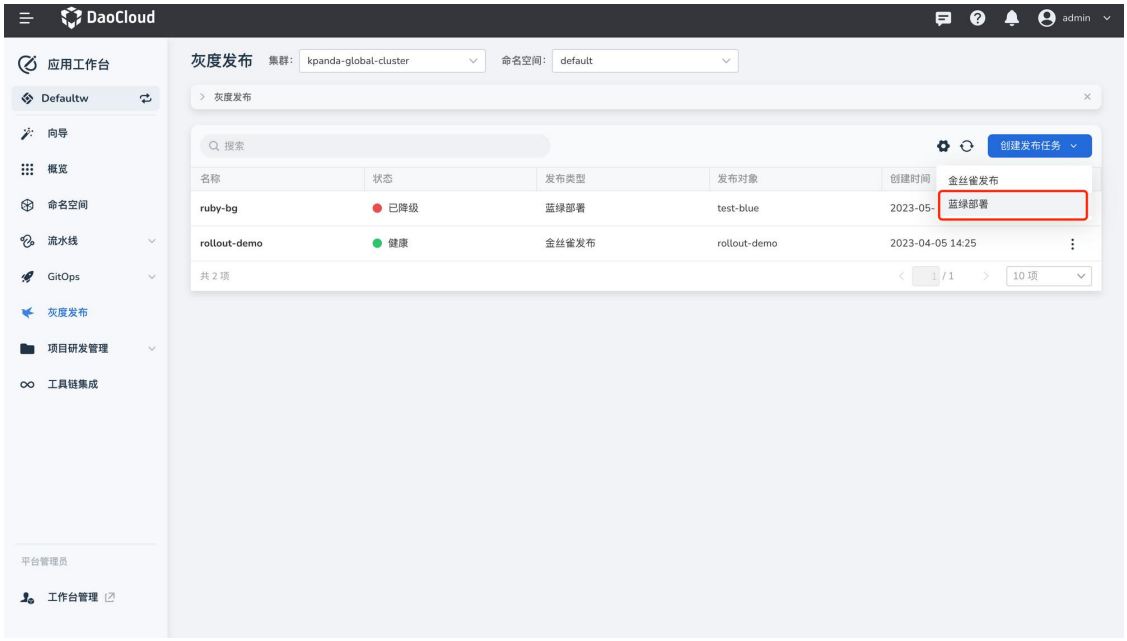
蓝绿发布

前提条件

- 创建一个工作空间和一个用户，该用户需加入该工作空间并具备 Workspace Editor 角色。
- 发布对象所在的集群已经安装了 Argo Rollout 组件。有关具体的安装方式，可参考管理 Helm 应用。

操作步骤

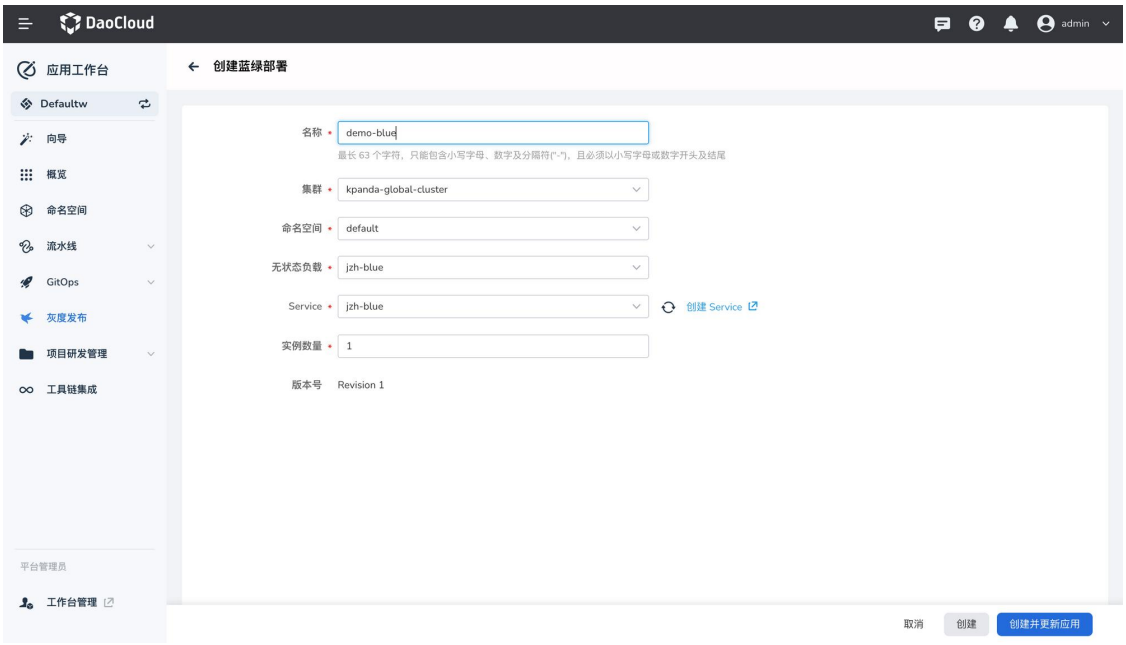
1. 进入应用工作台模块，在左侧导航栏点击 灰度发布，然后在页面右上角点击创建发布任务->蓝绿发布。



进入创建页面

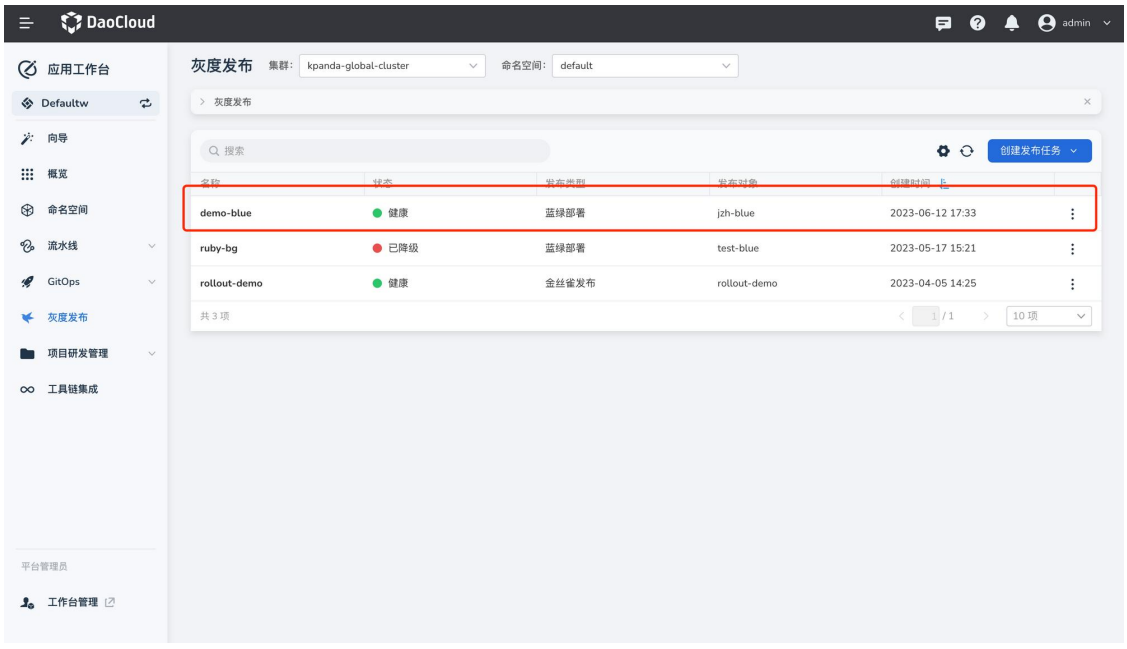
2. 参考下列要求填写基本信息，然后点击创建。

   - 名称：填写发布任务的名称。最长 63 个字符，只能包含小写字母、数字及分隔符(“-”)，且必须以小写字母或数字开头及结尾
   - 集群：选择发布对象所在的集群。需要确保该集群已经部署了 Istio 和 Argo Rollout。
   - 命名空间：选择发布对象所在的命名空间。
   - 无状态负载：选择具体的发布对象。
   - Service：选择发布对象的服务，该服务必须关联了该发布对象。



基本信息

3. 系统自动跳转至灰度发布的任务列表页面，提示创建成功，并成功在列表生成一条记录。



返回列表

Argo CD 应用状态说明

本页面介绍了 Argo CD 应用的健康状态、同步状态说明。

健康状态

健康状态	描述
健康	资源健康
已降级	资源已经被降级
进行中	资源还不健康，但仍在进行中，可能很快就会健康
暂停	资源被暂停并等待一些外部事件恢复（例如暂停的 CronJob 或暂停的部署）
未知	无法判断当前健康状态
丢失	资源已缺失

同步状态

同步状态	描述
已同步	集群中部署的资源与仓库中期望的状态一致
未同步	集群中部署的资源与仓库中期望的状态不一致
未知	无法判断当前同步状态

创建 Argo CD 应用

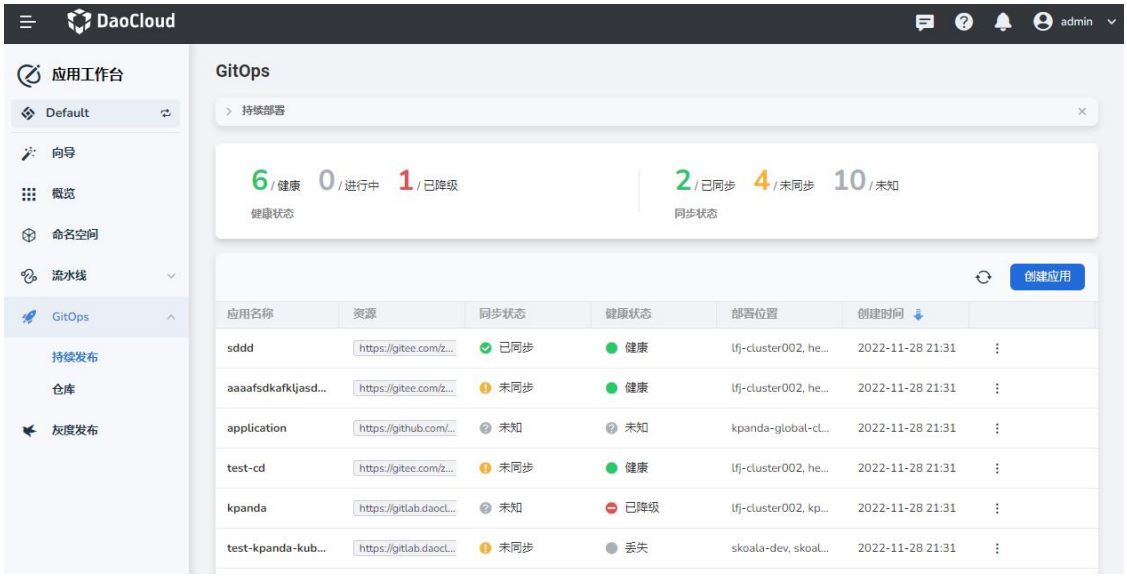
应用工作台基于开源软件 Argo CD 实现持续部署。本页面演示如何实现应用的持续部署。

前提条件

- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 workspace edit 角色。 参考创建工作空间、用户和角色。
- 准备一个 Git 仓库。

创建 Argo CD 应用

1. 在应用工作台 -> 持续发布页面中，点击创建应用按钮。



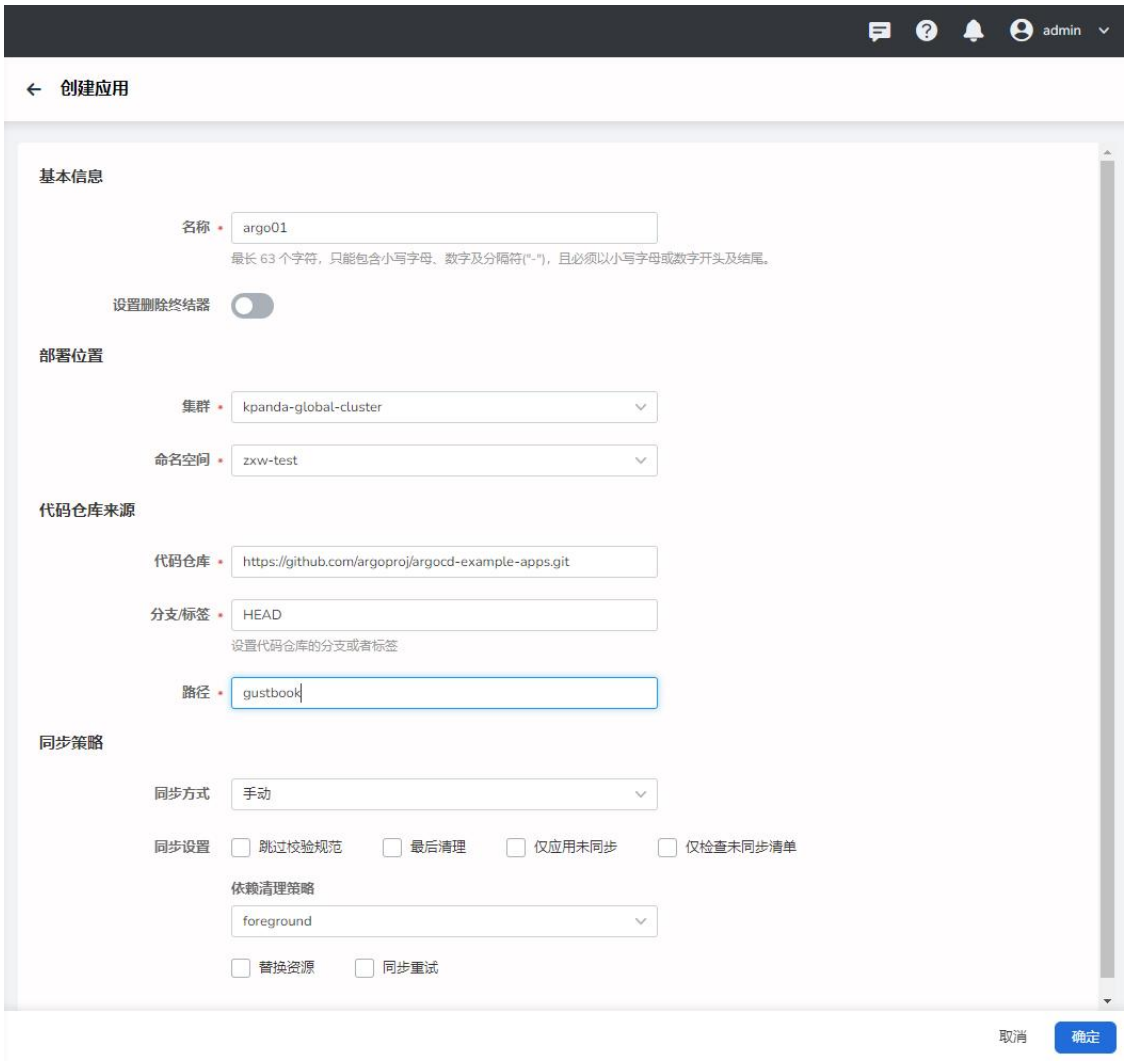
创建应用

2. 在创建应用页面中，分配配置基本信息、部署位置、代码仓库来源和同步策略后，点击确定。

    - 代码仓库来源：

- 代码仓库：选择已导入的代码仓库或者输入公开的代码仓库地址。例如：https://github.com/argoproj/argocd-example-apps.git
- 分支/标签：设置代码仓库的分支或标签，默认为 HEAD
- 路径：填写清单文件路径，例如 gustbook

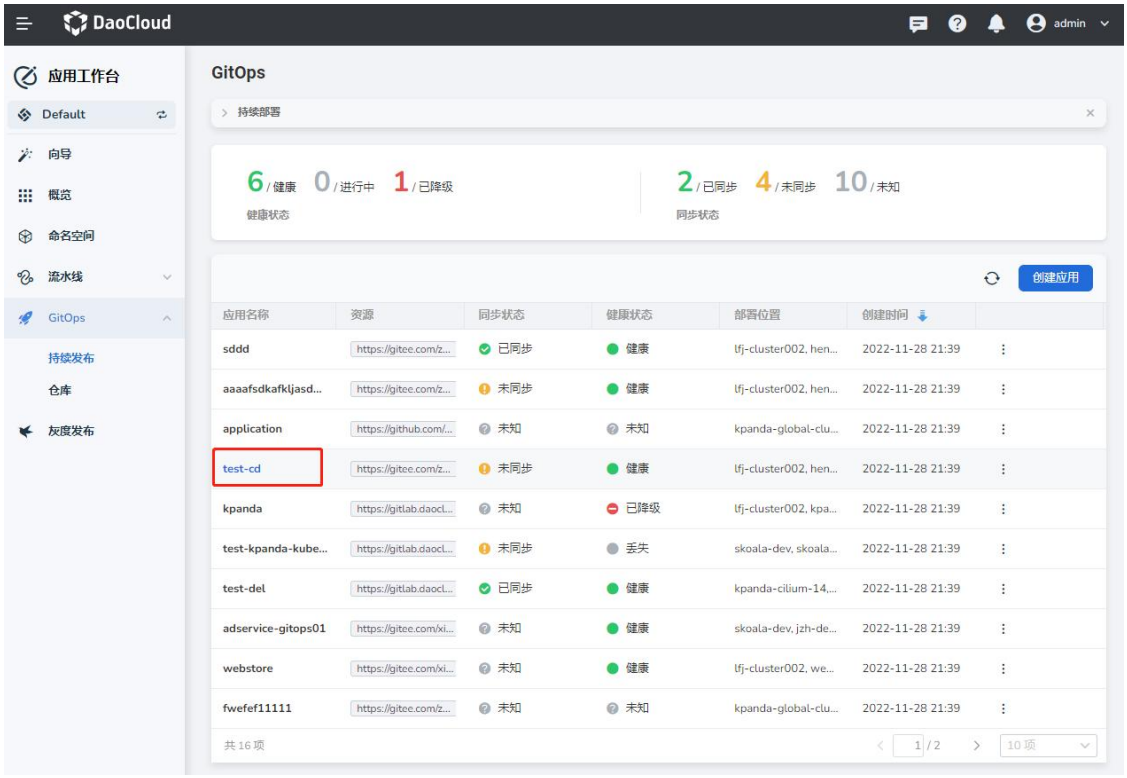
- 同步策略：
  - 手动同步：手动点击是否同步
  - 自动同步：自动检测代码仓库中清单文件的变化，一旦变化后就会立即同步应用资源至最新状态。并支持清理资源和自恢复选项：
    - 清理资源：同步时删除代码仓库不再定义的资源
    - 自恢复：确保与代码仓库中期望的状态同步
  - 同步设置：
    - 跳过校验规范：跳过应用清单文件规范校验
    - 最后清理：当所有的资源都已经同步且处于健康状态后，才会删除不存在的资源
    - 仅应用未同步：仅应用未同步的资源
    - 依赖清理策略：选择一个具体的清理策略：
      - foreground：先删除完所有依赖对象之后， 再删除属主对象
      - background：先删除属主对象之后，再删除所有依赖对象
      - orphan：删除属主对象之后，所有依赖对象仍然存在
    - 替换资源：是否需要替换已存在的资源
    - 同步重试：对应用同步重试参数化，支持设置重试最大次数、重试持续时间、重试持续最大时间、因子



创建应用

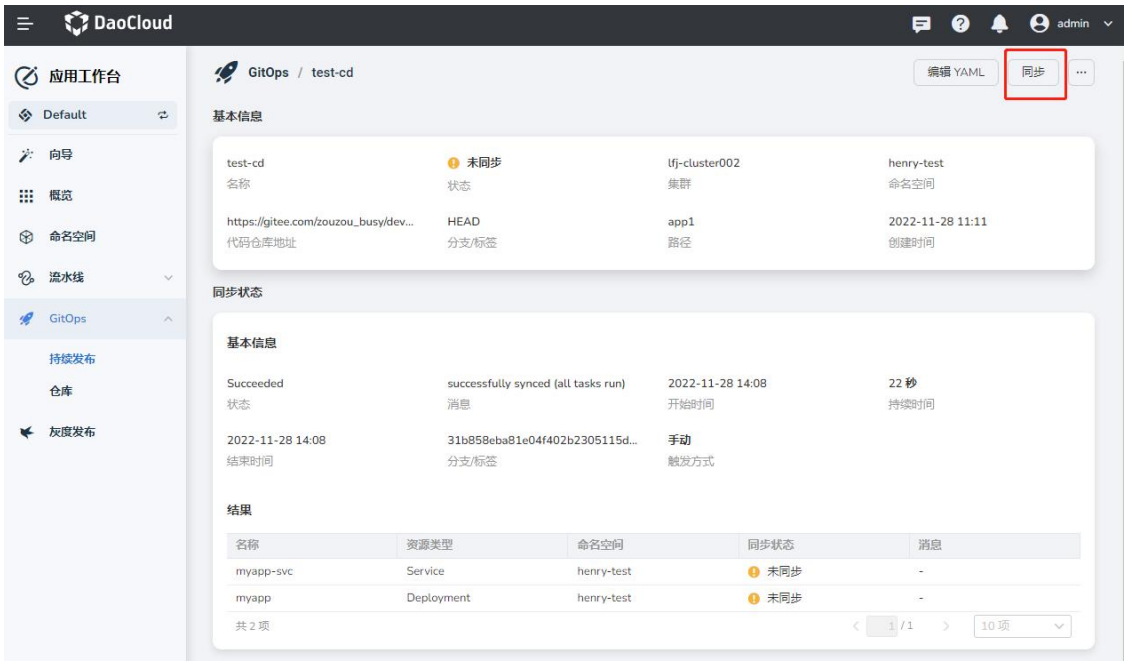
查看应用

1. 创建成功后，点击应用名称进入详情页面，可以查看应用详情。



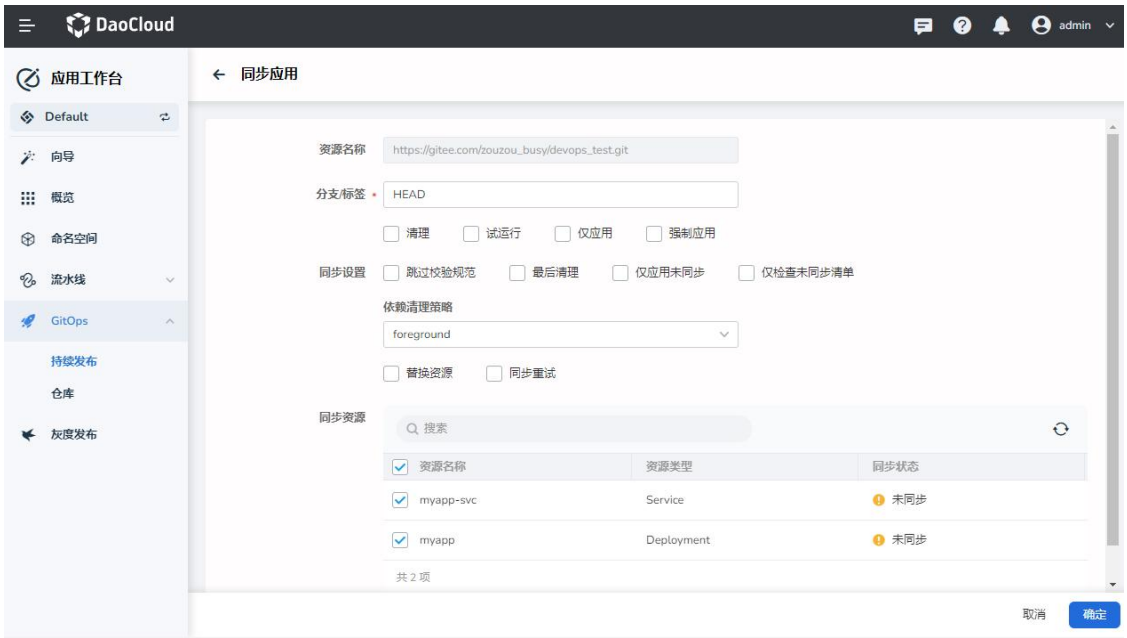
应用详情

2. 由于同步方式选择的是手动同步，所以我们需要手动进行同步，点击同步 。



手动同步

3. 同步过程中具体参数说明请参考手动同步应用，点击确定 。



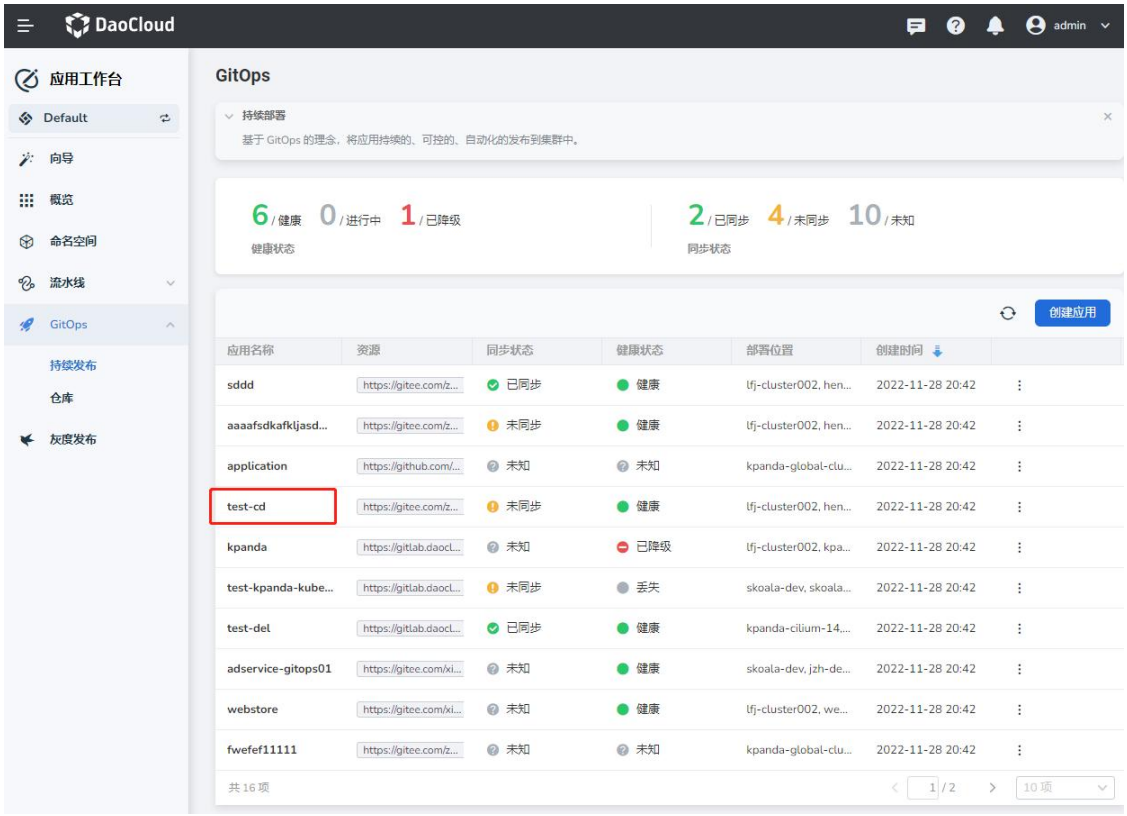
手动同步

4. 等待同步成功后，查看同步结果。

手动同步应用

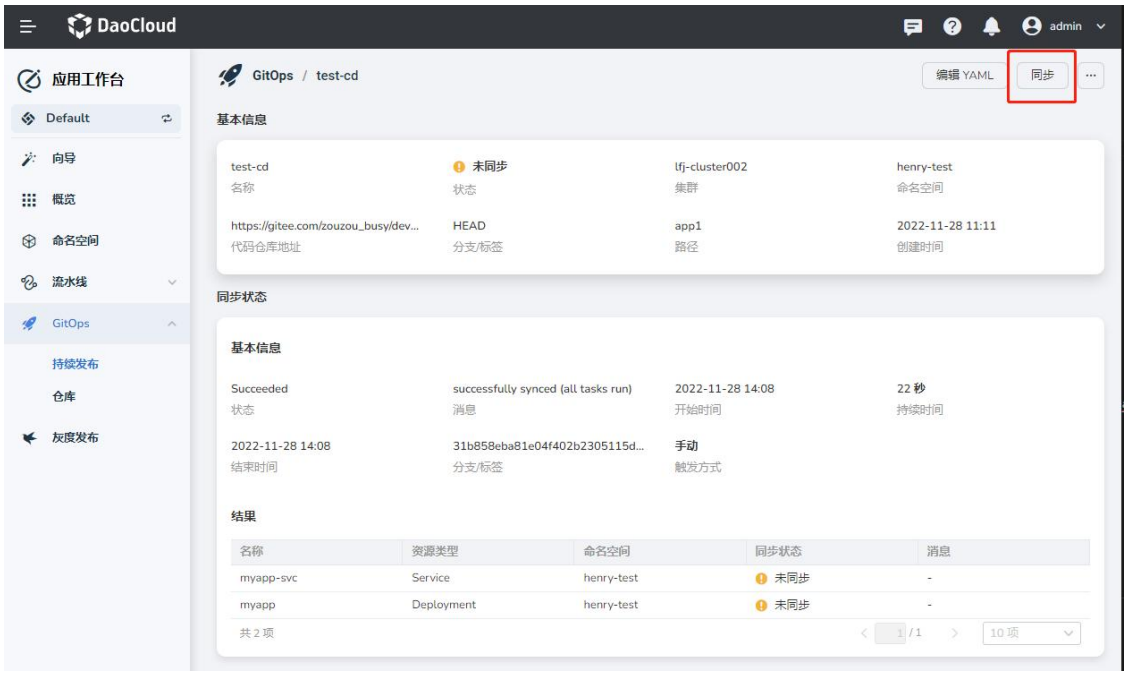
本页面演示如何手动同步持续部署的应用。

1. 在应用工作台 –> 持续发布页面中，点击一个同步状态为未同步的应用名称。



未同步

2. 在应用详情页面，点击同步按钮，进入同步页面：



同步

3. 在同步应用页面中，配置以下参数：

   - 资源名称：只读状态，不支持二次编辑
   - 分支/标签：设置同步的分支，如果同步方式为自动同步，更改后会同步报错
     - 清理：勾选后，如果清单文件中删除了一些资源，相关资源在同步时也会被清理，反之相关资源则会被跳过清理
     - 试运行：模拟同步过程
     - 仅应用：勾选后，会通过 kubectl apply 的方式部署资源
     - 强制应用：勾选后，如果部署资源时如果遇到冲突，会立即删除并重新创建该资源
   - 同步设置：同步设置的参数与创建时一致，具体参数说明请参考创建应用

   在同步资源区域，至少选择一个应用进行同步。例如一些资源已经被同步，只需要同步未同步的资源即可。



同步

4. 点击确定，等待同步成功后，查看同步结果。

使用流水线实现代码扫描

代码仓库中的源代码作为软件的最初原始形态，其安全缺陷是导致软件漏洞的直接根源。 因此，通过代码扫描分析发现源代码中的安全缺陷是降低软件潜在漏洞的一种重要方法。

例如，SonarQube 是一款代码自动审查工具，用于检测项目代码中的 bug 错误，提高测试覆盖率等。 它可以与项目中现有的工作流程集成，以便在项目分支和拉取请求之间进行连续的代码检查。

本文将介绍如何在流水线中集成 SonarQube 来实现代码扫描能力。

部署 SonarQube

如果您已有 SonarQube 环境，可以跳过此步骤，但需要保证当前集群可以访问到 SonarQube 服务器。

1. 通过 Helm 安装 SonarQube。

   ```
   helm repo add sonarqube <https://SonarSource.github.io/helm-chart-sonarqube>
   helm repo update
   kubectl create namespace sonarqube
   helm upgrade --install -n sonarqube sonarqube sonarqube/sonarqube --create-namespace --set service.type=NodePort
   ```

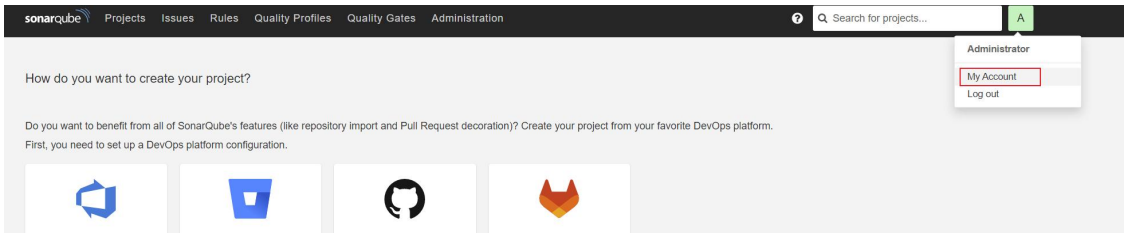
2. 查看对应 namespace 下的 Pod STATUS 都为 Running，表明 SonarQube 安装成功。

   ```
   kubectl get po
   NAME                            READY   STATUS    RESTARTS   AGE
   sonarqube-postgresql-0          1/1     Running   0          3h59m
   sonarqube-sonarqube-0           1/1     Running   0          3h59m
   ```

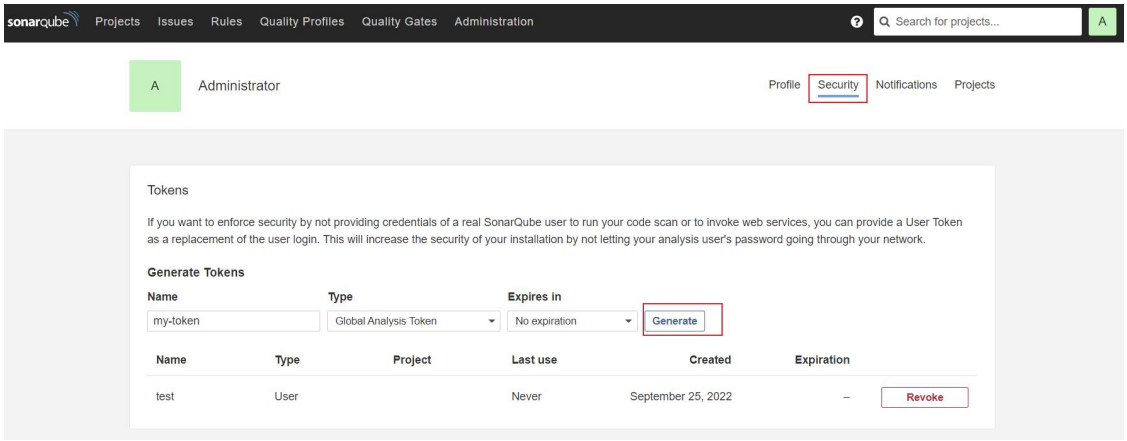
3. 查看 SonarQube 控制台的访问地址。通常访问地址为 http://<Node IP>:<NodePort>，账号和密码为 admin/admin。

```
export NODE_PORT=$(kubectl get --namespace amamba-system -o jsonpath="{.spec.ports[0].nodePort}" services sonarqube-sonarqube)
export NODE_IP=$(kubectl get nodes --namespace amamba-system -o jsonpath="{.items[0].status.addresses[0].address}")
echo http://$NODE_IP:$NODE_PORT
```

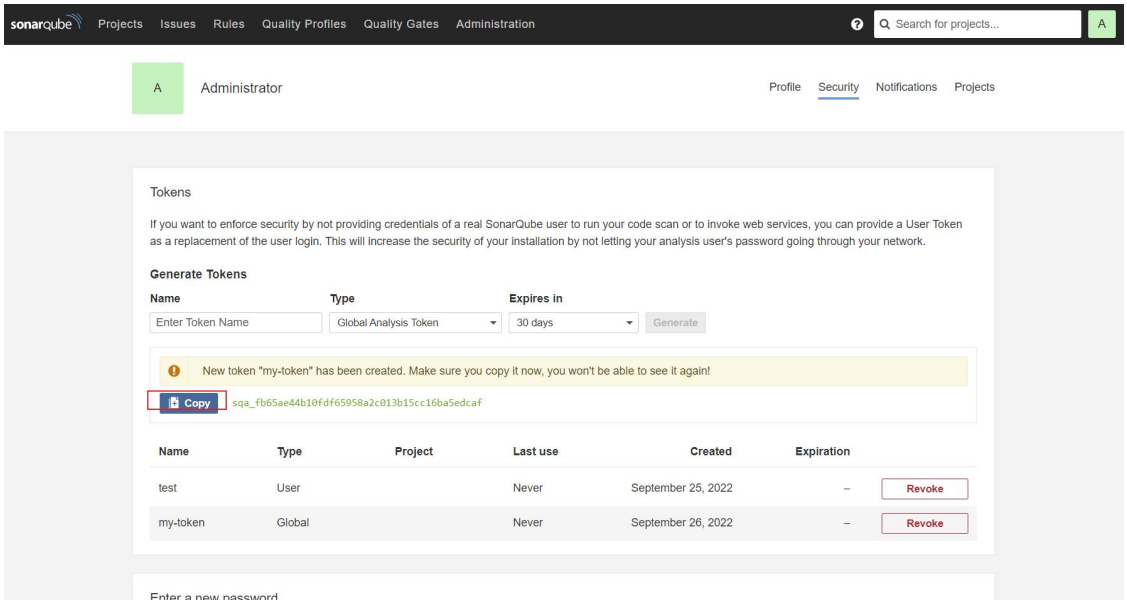
4. 在 SonarQube 生成管理员令牌（Token），操作路径为：My Account -> Profile -> Security -> Generate -> Copy



scan



scan

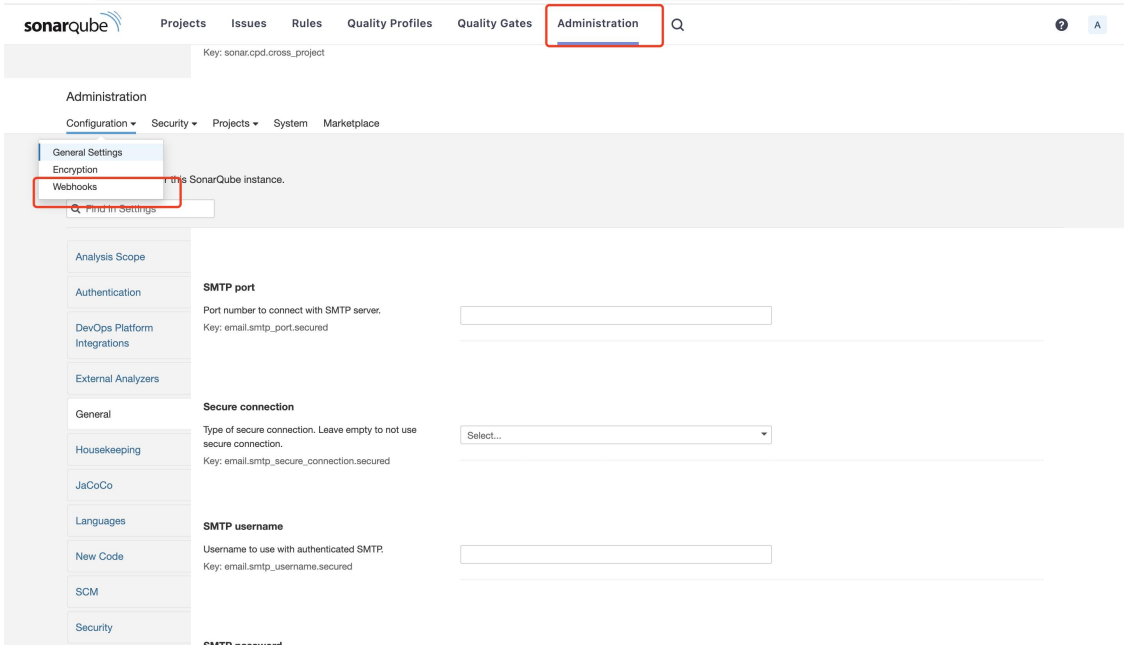


scan

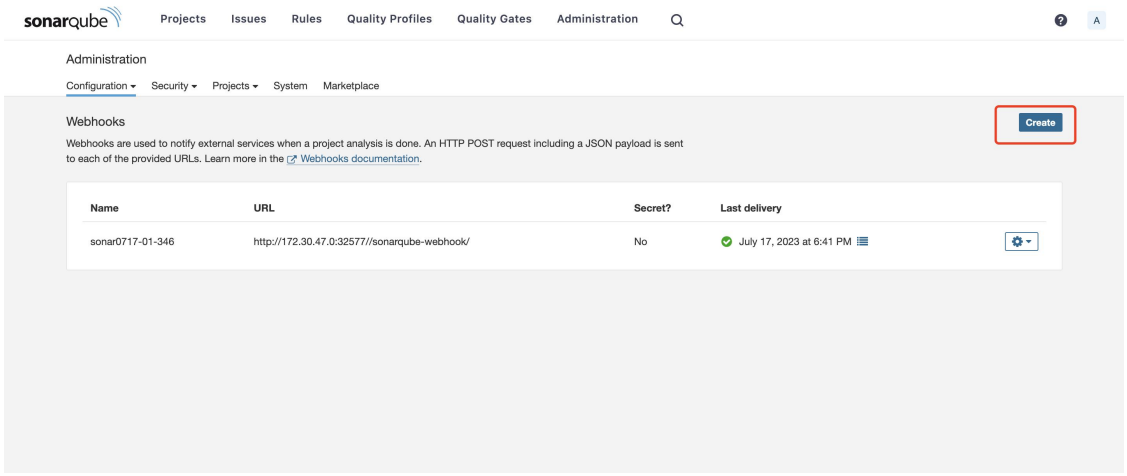
5. 在 SonarQube 里创建 Webhook 服务器：

    1. 操作路径为：Administration –> Configuration –> Webhooks。

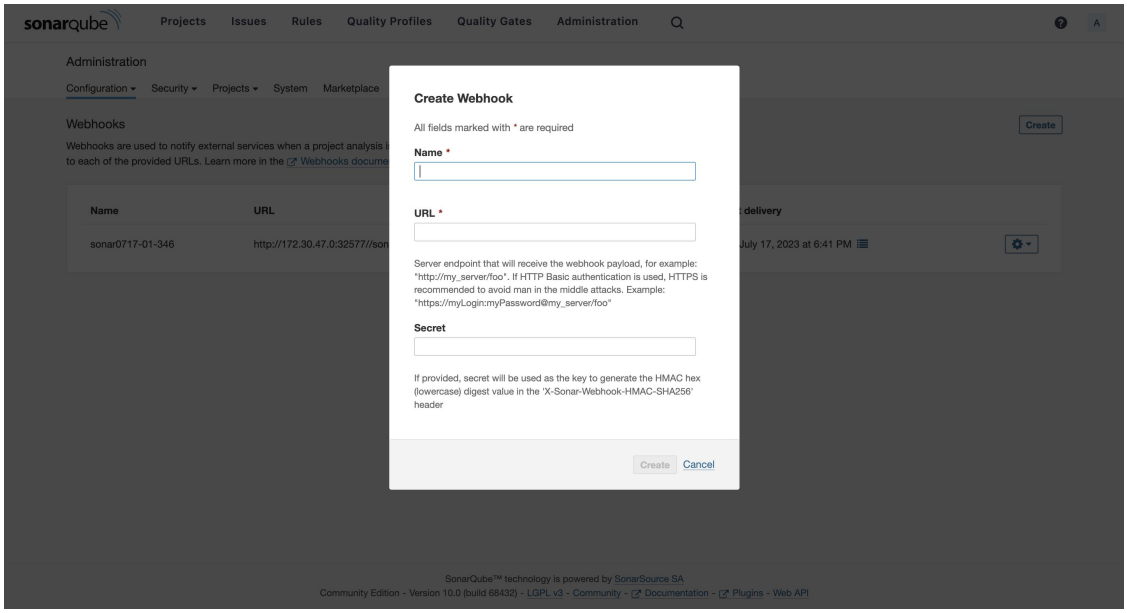
    2. 点击 Create，在弹出的对话框中输入 Name 和 Jenkins Console URL（即 SonarQube Webhook 地址，这个地址是前面获取的 SonarQube 地址+/sonarqube-webhook/）。


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Key: sonar.cpd.cross_project
Administration
Configuration
Security
Projects
System
Marketplace
General Settings
Encryption
Webhooks
this SonarQube instance.
Find in Settings
Analysis Scope
Authentication
DevOps Platform Integrations
External Analyzers
General
Housekeeping
JaCoCo
Languages
New Code
SCM
Security
SMTP port
Port number to connect with SMTP server.
Key: email.smtp_port.secured
Secure connection
Type of secure connection. Leave empty to not use secure connection.
Key: email.smtp_secure_connection.secured
Select...
SMTP username
Username to use with authenticated SMTP.
Key: email.smtp_username.secured

scan


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Administration
Configuration
Security
Projects
System
Marketplace
Webhooks
Create
Webhooks are used to notify external services when a project analysis is done. An HTTP POST request including a JSON payload is sent to each of the provided URLs. Learn more in the Webhooks documentation.
Name
URL
Secret?
Last delivery
sonar0717-01-346
http://172.30.47.0:32577//sonarqube-webhook/
No
July 17, 2023 at 6:41 PM

scan



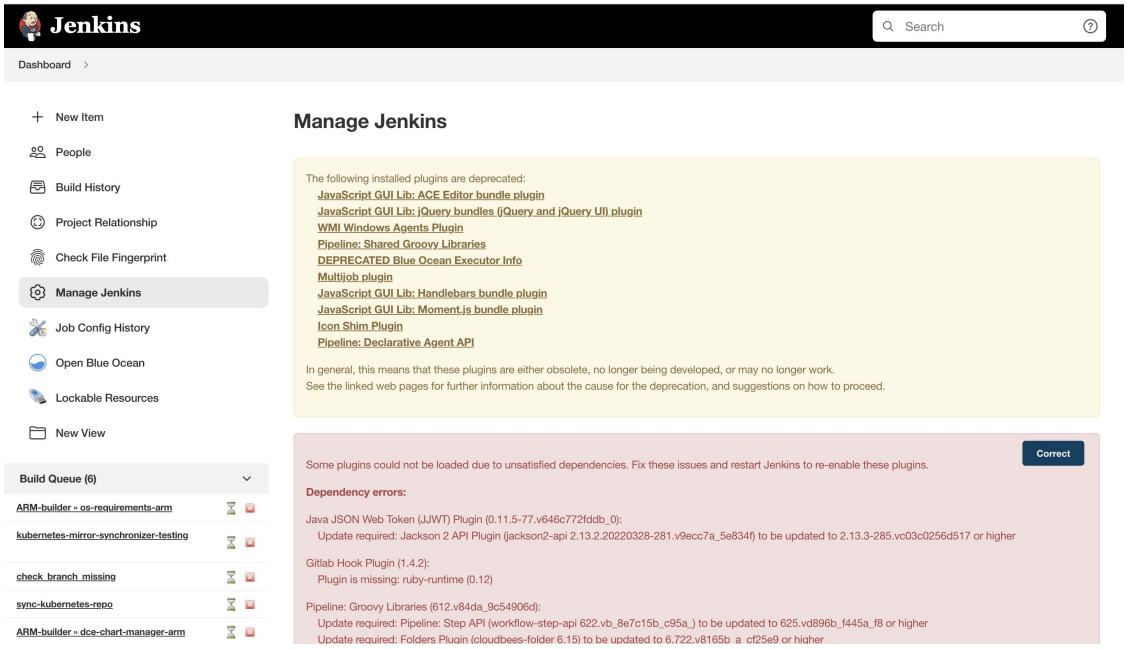
scan

6. 将 SonarQube 地址添加至 Jenkins，请确保可以互通：

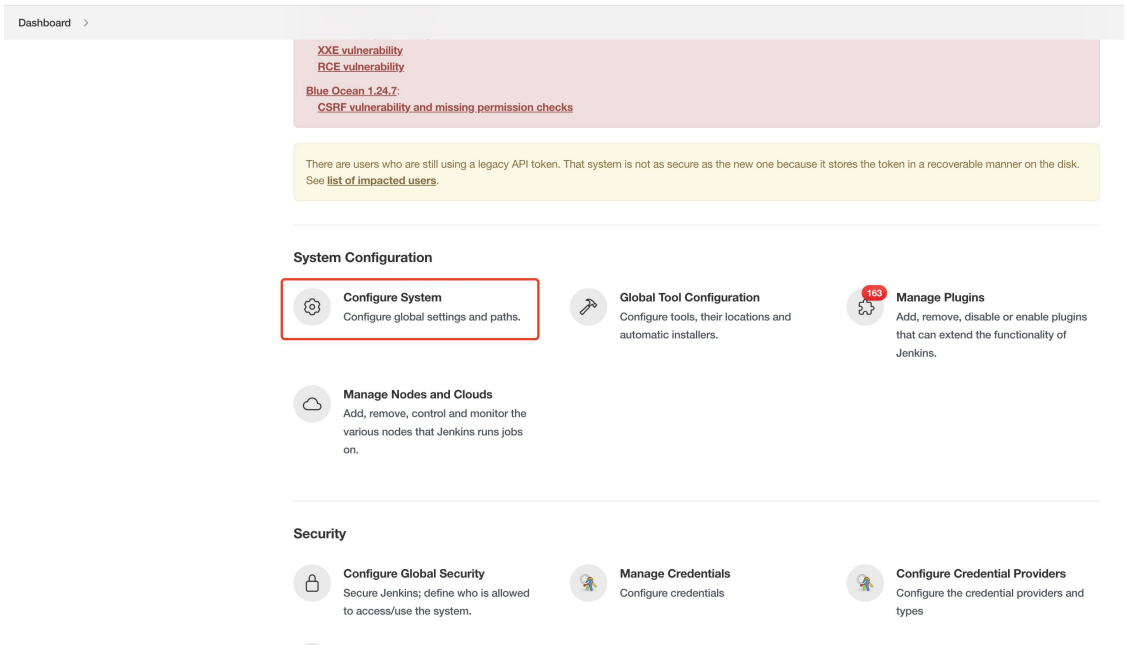
    1. 操作路径为 Manage Jenkins –> Configure System –> SonarQube servers –> Add SonarQube

    2. 在弹出的对话框中输入 Server URL 和 Server authentication token（即 SonarQube 地址，这个地址是前面获取的 SonarQube 地址 + 管理员令牌）。

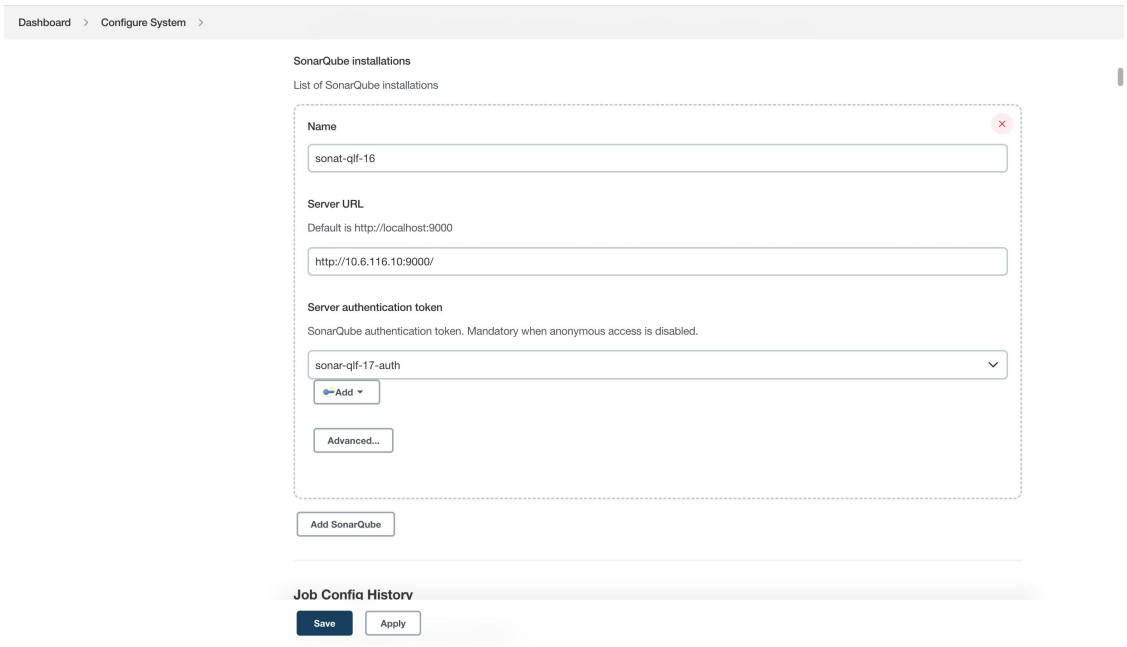
    3. 点击 Save 完成操作。


Jenkins
Search
Dashboard
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
Job Config History
Open Blue Ocean
Lockable Resources
New View
Build Queue (6)
ARM-builder » os-requirements-arm
kubernetes-mirror-synchronizer-testing
check_branch_missing
sync-kubernetes-repo
ARM-builder » dce-chart-manager-arm
Manage Jenkins
The following installed plugins are deprecated:
JavaScript GUI Lib: ACE Editor bundle plugin
JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin
WMI Windows Agents Plugin
Pipeline: Shared Groovy Libraries
DEPRECATED Blue Ocean Executor Info
Multijob plugin
JavaScript GUI Lib: Handlebars bundle plugin
JavaScript GUI Lib: Moment.js bundle plugin
Icon Shim Plugin
Pipeline: Declarative Agent API
In general, this means that these plugins are either obsolete, no longer being developed, or may no longer work.
See the linked web pages for further information about the cause for the deprecation, and suggestions on how to proceed.
Correct
Some plugins could not be loaded due to unsatisfied dependencies. Fix these issues and restart Jenkins to re-enable these plugins.
Dependency errors:
Java JSON Web Token (JJWT) Plugin (0.11.5-77.v646c772fddb_0):
Update required: Jackson 2 API Plugin (jackson2-api 2.13.2.20220328-281.v9ecc7a_5e834f) to be updated to 2.13.3-285.vc03c0256d517 or higher
Gitlab Hook Plugin (1.4.2):
Plugin is missing: ruby-runtime (0.12)
Pipeline: Groovy Libraries (612.v84da_9c54906d):
Update required: Pipeline: Step API (workflow-step-api 622.vb_8e7c15b_c95a_) to be updated to 625.vd896b_f445a_f8 or higher
Update required: Folders Plugin (cloudbees-folder 6.15) to be updated to 6.722.v8165b_a_cf25e9 or higher

scan


Dashboard
XXE vulnerability
RCE vulnerability
Blue Ocean 1.24.7:
CSRF vulnerability and missing permission checks
There are users who are still using a legacy API token. That system is not as secure as the new one because it stores the token in a recoverable manner on the disk.
See list of impacted users.
System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
163
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Security
Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.
Manage Credentials
Configure credentials
Configure Credential Providers
Configure the credential providers and types

scan



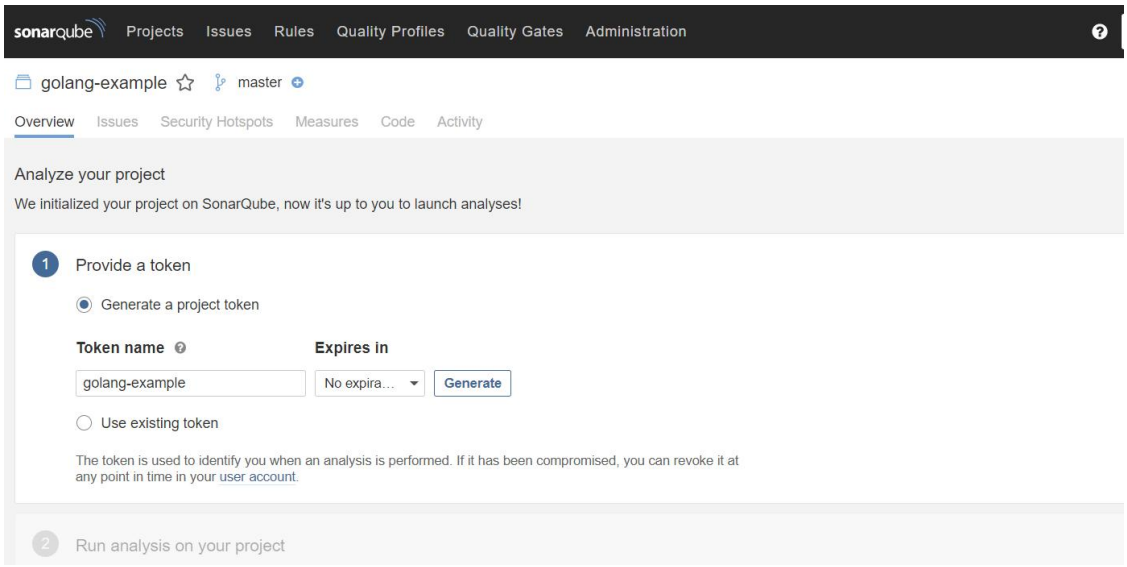
scan

!!! note

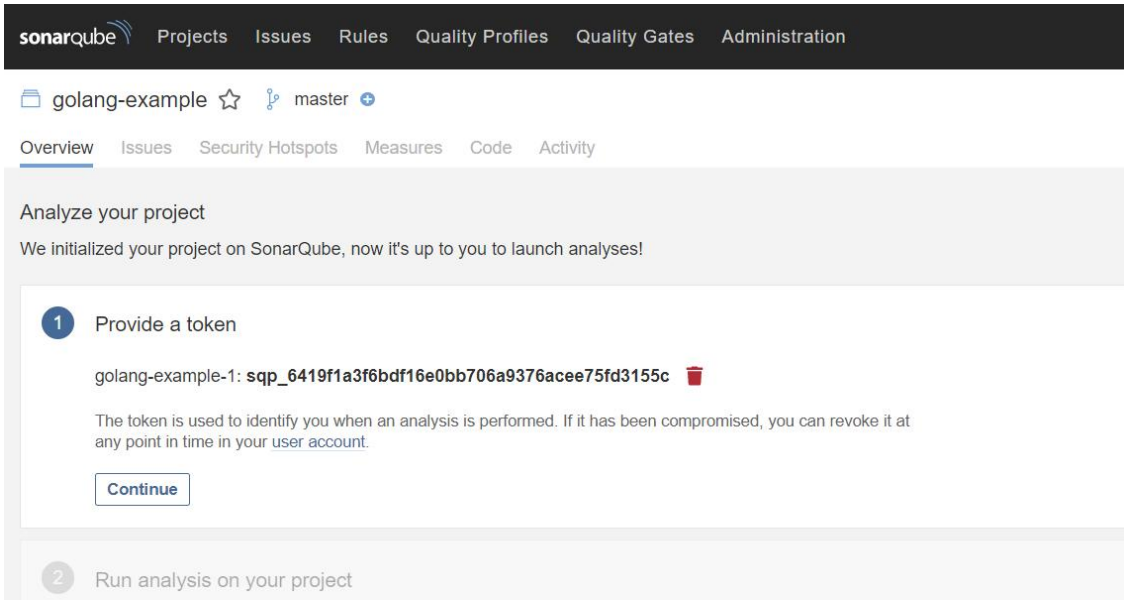
如何访问应用工作台部署的 Jenkins Dashboard?

- 前往容器管理->全局服务集群->无状态负载，在 amamba-system 下找到负载 amamba-jenkins，通过 NodePort 的方式暴露该服务。
- 默认用户名密码为 admin/Admin01

7. 为新项目创建 SonarQube Token，操作路径为 Create new project -> Set Up -> Generate -> Continue。



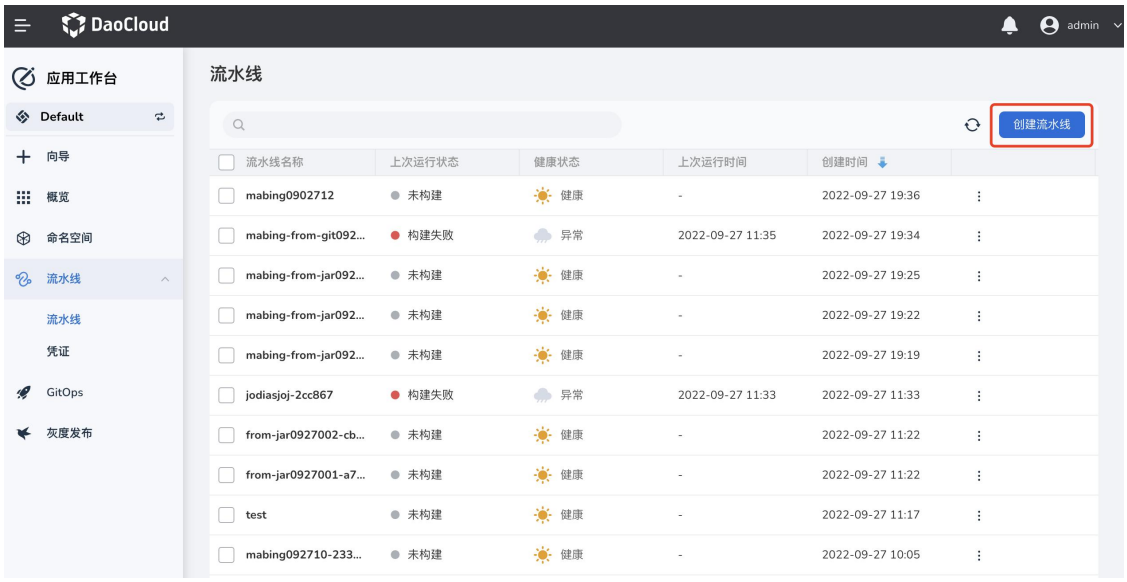
scan



scan

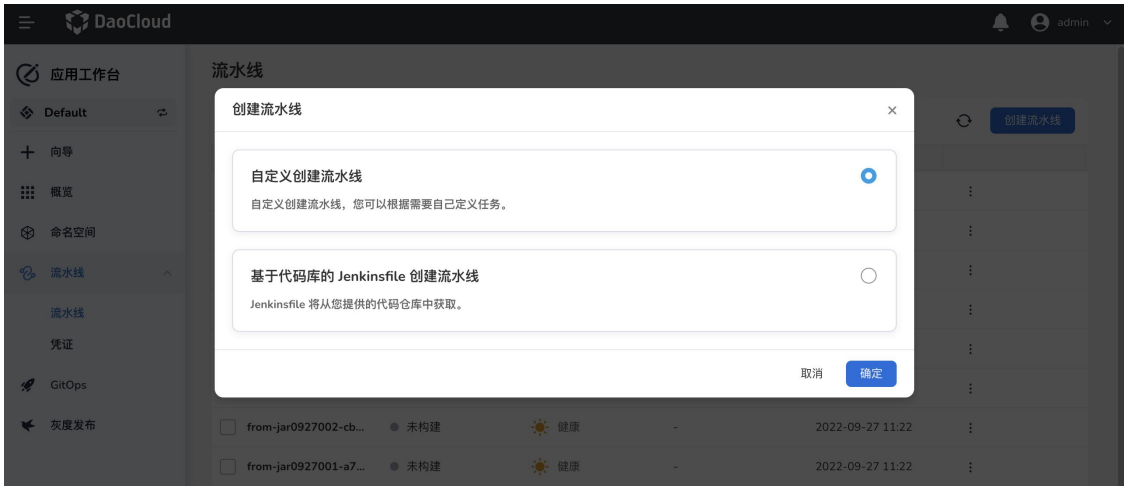
创建流水线

1. 在流水线页面，点击创建流水线。



scan

2. 选择自定义创建。



scan

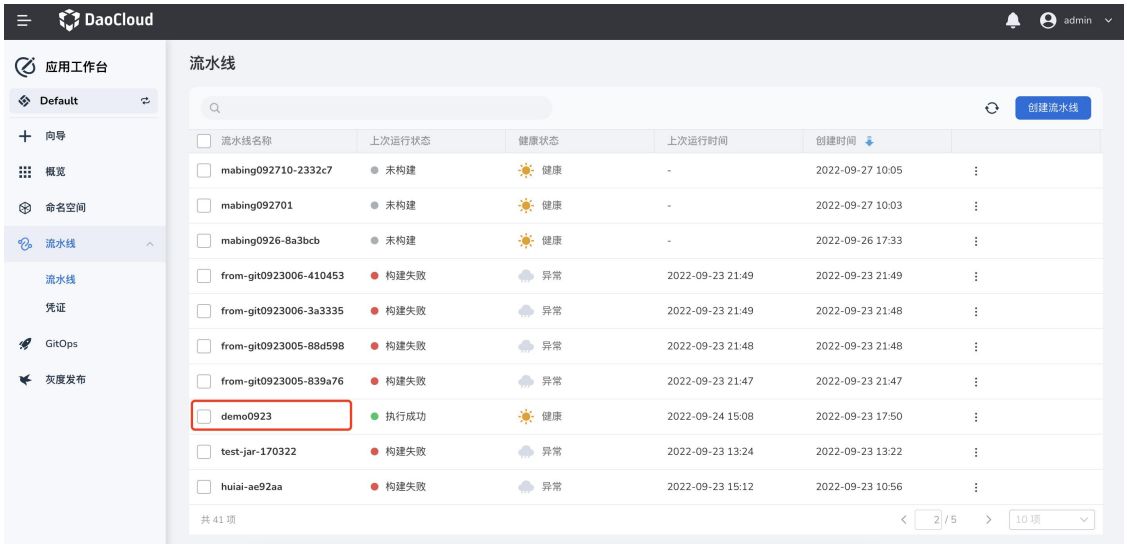
3. 输入名称，其他可使用默认值，点击确定。



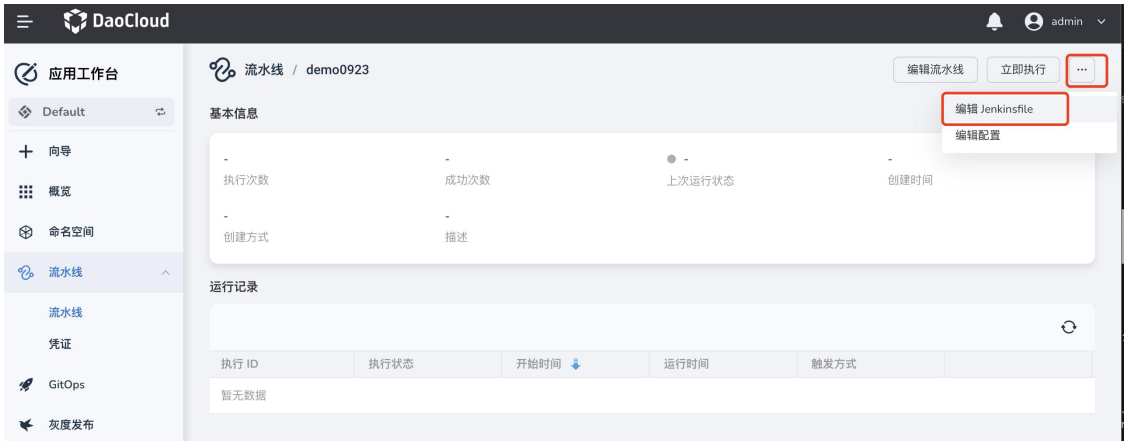
scan

编辑 Jenkinsfile

1. 点击一个流水线进入其详情页面，在右上角点击 … -> 编辑 Jenkinsfile。



scan



scan

2. 将以下 YAML 代码复制粘贴至 jenkinsfile。

```
pipeline {
agent {
    node {
    label 'go'
    }

}
stages {
    stage('git clone') {
    steps {
        git(credentialsId: 'mabing-gitlab', branch: 'master', url: 'https://gitlab.daocloud.cn/bing.ma/jenkins-sonarqube-demo.git')
    }
    }

    stage('unit test') {
    steps {
        container('go') {
            sh 'go test -json > test-report.out'
            sh 'go test -coverprofile=coverage.out'
```

```
        }
    }
    }

    stage("SonarQube analysis") {
    steps {
        container('go') {
        withSonarQubeEnv('demo-dev-sonarqube') {
            sh 'sonar-scanner -Dsonar.projectKey=golang-demo -Dsonar.sources=. -Dsonar.host.url=http://10.6.182.101:32313 -Dsonar.login=4b337cbbafd89ae9bca46a746cddab4c993d2a7a'
        }
        }
    }
    }

    stage("Quality Gate") {
    steps {
        container('go') {
        timeout(time: 1, unit: 'HOURS') {
            waitForQualityGate abortPipeline: false // 这里设置为 false，表示即使没有通过 sonarQube 的 QUALITY GATE，流水线也可以继续
        }
        }
    }
    }

    stage('build & push') {
    steps {
        container('go') {
        withCredentials([usernamePassword(credentialsId:'docker-credential',passwordVariable:'PASS',usernameVariable:'USER')]) {
```

```
                sh 'go build -o simple-http-server main/main.go'
                sh 'docker build -f Dockerfile . -t $registry/$project/$name:l
atest'
                sh 'docker login $registry -u $USER -p $PASS'
                sh 'docker push $registry/$project/$name:latest'
            }

            }

        }
        }

        stage('deply'){
        steps {
            container('go'){
            withCredentials([kubeconfigFile(credentialsId: 'kubeconfig-credentia
l', variable: 'KUBECONFIG')]) {
                sh 'kubectl apply -f deploy.yaml'
            }
            }
        }

        }

}
parameters {
    string(name: 'registry', defaultValue: 'release-ci.daocloud.io', descriptio
n: '')
    string(name: 'project', defaultValue: 'demo', description: '')
    string(name: 'name', defaultValue: 'http-hello', description: '')
```

```
}
}
```

!!! note

在上述代码中，

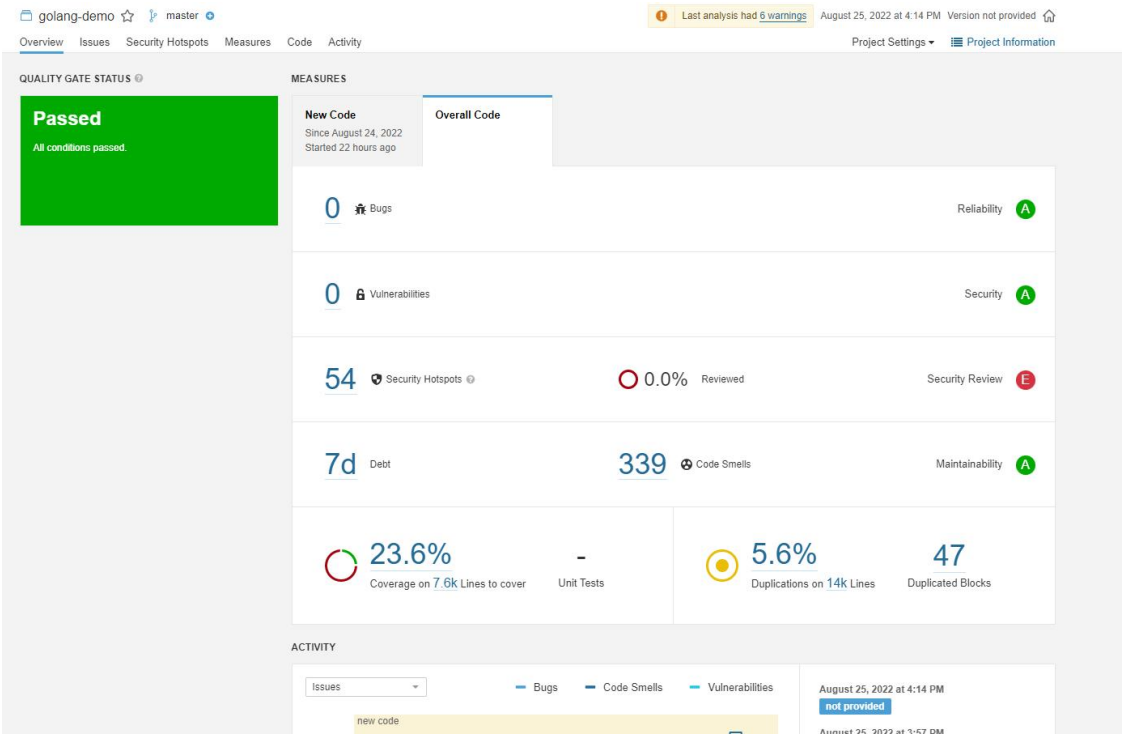
– waitForQualityGate abortPipeline: false，表示没有通过门禁质量检查也可以继续流水线。如果是 true，则表示放弃

– 对于 withSonarQubeEnv 下的 shell 语句，可以从上一步的 SonarQube 中复制

3. 保存后立即运行该流水线。

前往 SonarQube 查看代码扫描结果

等待流水线运行成功后，前往 SonarQube 查看代码扫描结果。



scan

基于 Argo Rollout 实现渐进式灰度发布

本文介绍如何基于开源的 Argo Rollout 实现渐进式灰度发布。

前提条件

- 示例中的镜像需要访问公网：argoproj/rollouts-demo:yellow 和 argoproj/rollouts-demo:blue。
- 仅适用于通过安装器以 metallb 方式部署出来的 DCE 5.0 平台。
- 使用灰度发布能力需要所在集群中安装 Istio 和 Argo Rollout 组件。

操作步骤

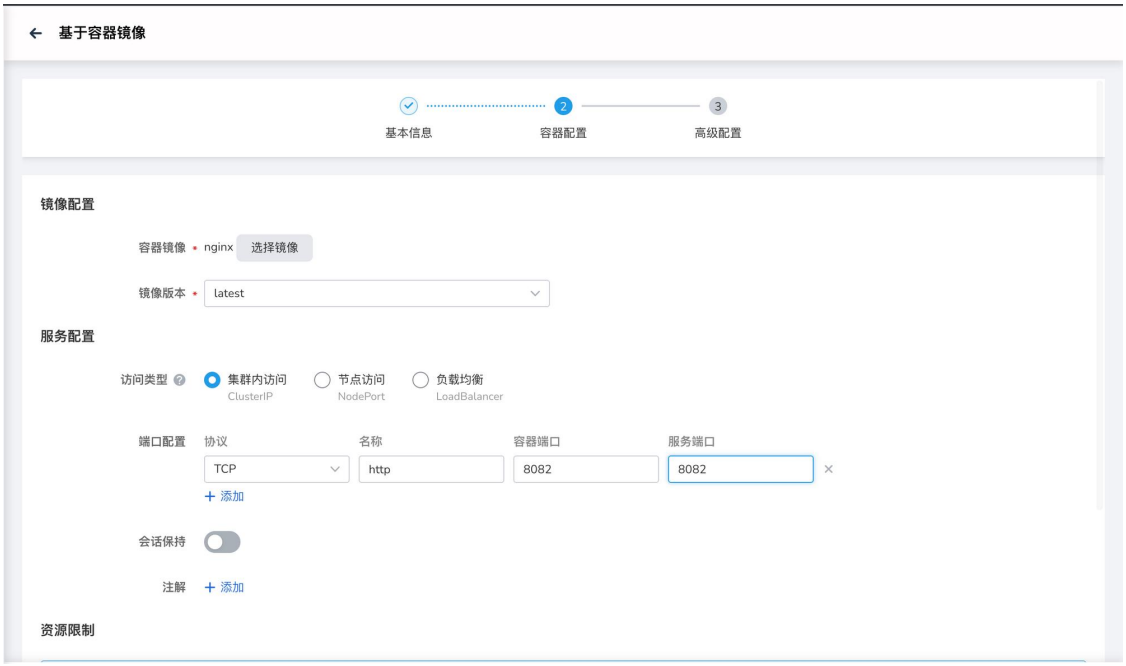
整个过程分为四个步骤：首先基于容器镜像构建应用，然后配置 Istio 相关资源，创建灰度发布任务，最后验证效果。

基于容器镜像构建应用

具体操作步骤可参考基于 Git 仓构建微服务应用、基于 Jar 包部署 Java 应用。

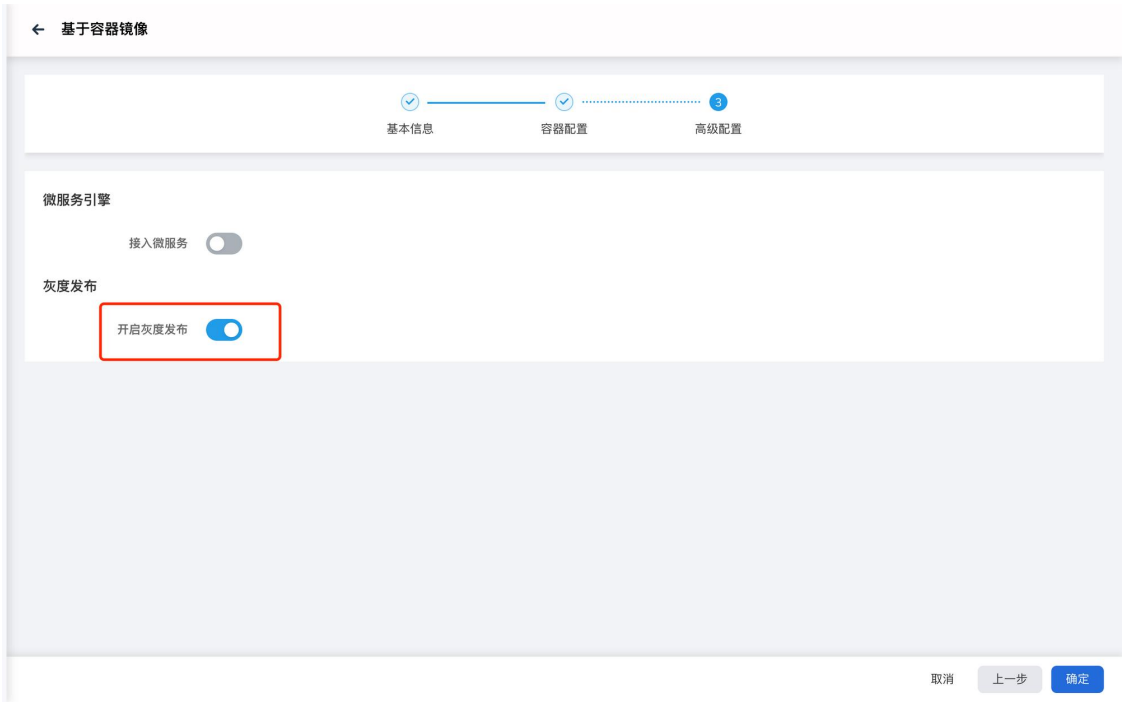
需要注意：

- 容器镜像为：argoproj/rollouts-demo:blue
- 服务端口：名称为 http、容器端口为 8082、服务端口为 8082。



容器配置

- 填写高级配置时需要开启灰度发布。



开启灰度发布

Istio 相关资源配置

在服务网格模块或者控制台创建以下资源。

1. 创建 Gateway

   ```yaml title="gateway.yaml" apiVersion: networking.istio.io/v1beta1 kind: Gateway metadata: name: rollout-demo namespace: rollout-demo # (1) spec: selector: istio: ingressgateway servers:

   - hosts:
     - '*' port: name: http number: 8082 protocol: HTTP ```

   1. 部署应用的命名空间

2. 部署 Gateway

   vi gateway.yaml
   kubectl apply -f gateway.yaml

3. 修改 VirtualService

   kubectl edit vs {#创建应用的名字} -n {#创建应用的命名空间}

   仅需修改提示的字段，其余字段无需修改。

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  creationTimestamp: "2022-11-07T10:46:23Z"
  generation: 84
  name: demo01
  namespace: default
  resourceVersion: "5741370"
  uid: 8109f754-aa9d-49f1-b8a9-d4daf5108032
spec:
  gateways:
```
```

```
  - rollout-demo  # (1)
  hosts:
  - '*'  # (2)
  http:
  - name: primary
    route:
    - destination:
        host: demo01
        subset: stable
      weight: 100
    - destination:
        host: demo01
        subset: canary
      weight: 0
```

1. 修改此处，需要新增 gateway，指向上一步创建的 gateway 名称
2. 修改此处，原来 host 为 vs 的名称，需要删除后更改为‘*’

4. 配置 istio-ingressgateway 网关

kubectl edit svc istio-ingressgateway -n istio-system

仅需修改提示的字段，其余字段无需修改。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: istio-ingressgateway
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: istio-ingressgateway
    app.kubernetes.io/version: 1.15.0
    helm.sh/chart: gateway-1.15.0
    istio: ingressgateway
```

```yaml
  name: istio-ingressgateway
  namespace: istio-system
  resourceVersion: "5775680"
  uid: 53bd7344-ba45-4547-b695-aca2c4dd713d
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 100.66.222.131
  clusterIPs:
  - 100.66.222.131
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: status-port
    nodePort: 32384
    port: 15021
    protocol: TCP
    targetPort: 15021
  # 新增以下内容
  - name: rollout-demo
    port: 8082
    protocol: TCP
    targetPort: 8082
  # ----------------
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  sessionAffinity: None
  type: LoadBalancer
```

5. 创建 AuthorizationPolicy 资源

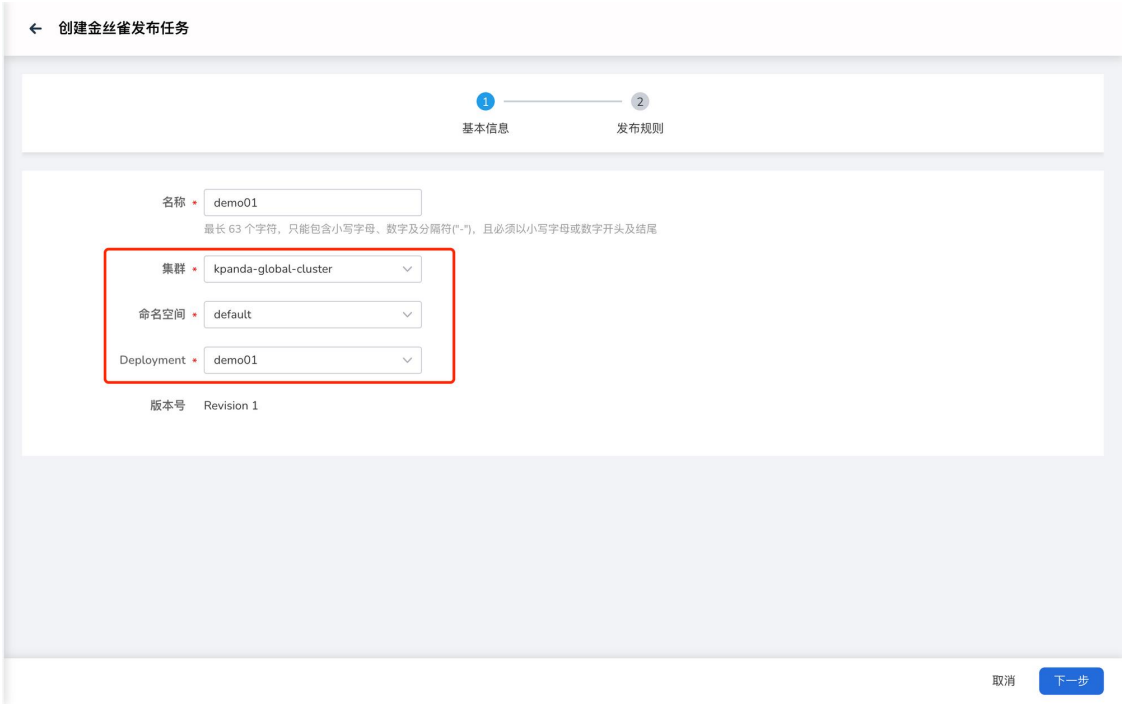
   此步骤的目的是在浏览器访问是可以正常访问到 js 相关的资源。

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: demo01
  namespace: istio-system
spec:
  rules:
  - to:
    - operation:
        paths:
        - /*
  - from:
    - source:
        requestPrincipals:
        - '*'
  selector:
    matchLabels:
      app: istio-ingressgateway
status:
  loadBalancer:
    ingress:
    - ip: 10.29.135.48
```

创建灰度发布任务

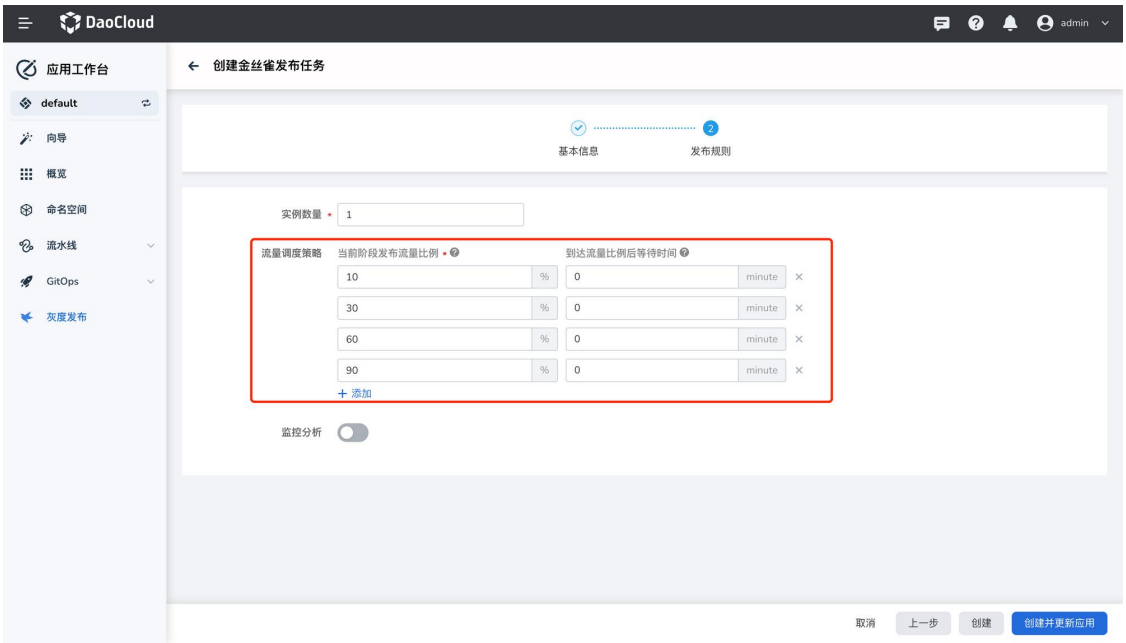
在应用工作台创建灰度发布任务。如需了解更详细的创建说明，可参考创建金丝雀发布任务。

1. 选择开启灰度发布的应用



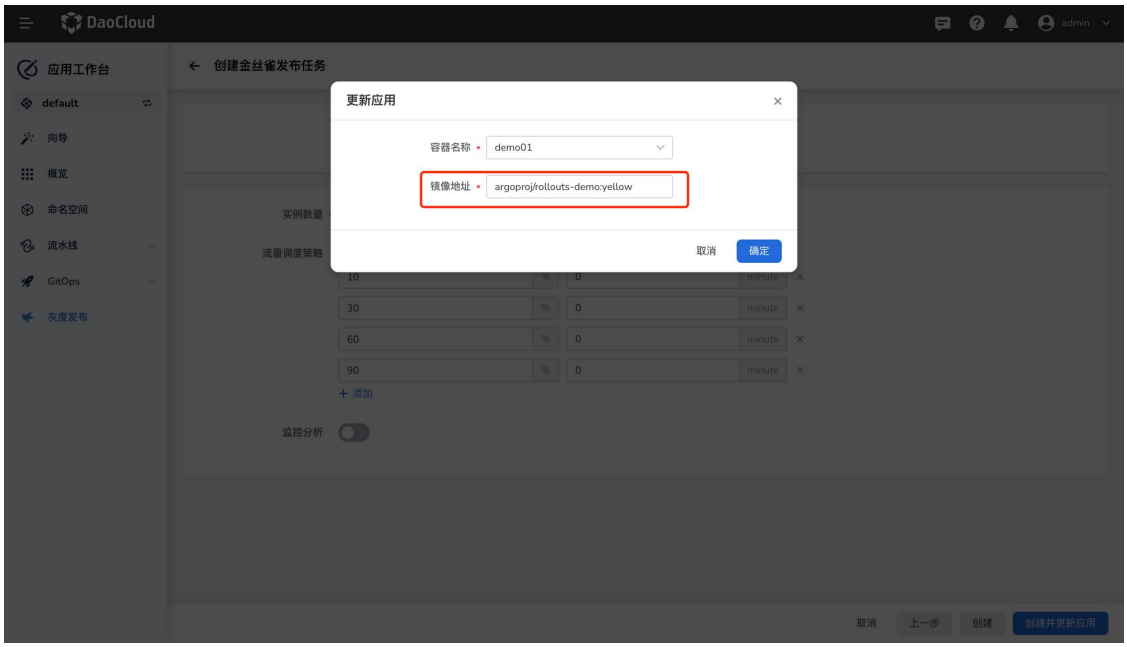
选择应用

2. 修改发布规则（为了更明显的演示效果）



修改规则

3. 点击创建并更新应用，在弹出的对话框中填写镜像地址：argoproj/rollouts-demo:yellow



填写镜像地址

验证效果

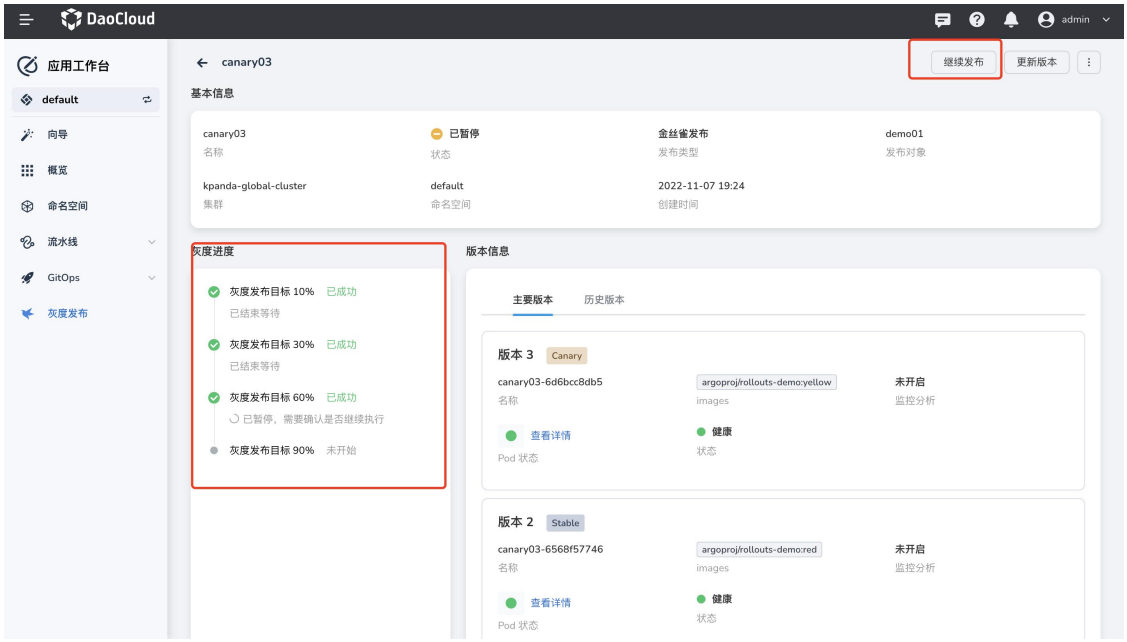
访问地址：http://{istio-ingressgateway LB IP}:8082，得到如下所示的访问效果。

此界面会并发调用 http://{istio-ingressgateway LB IP}:8082/color，将获取到颜色信息填充到方格中。 在灰度发布对象中，指定的颜色为 **blue、yellow**，会按照定义规则 1:9 的流量比进行展示。



效果示意

此时可以在应用工作台灰度发布模块选择继续发布应用来调整流量比例，直到最终成功发布。



调整比例

Jenkins 场景配置

Jenkins 分为 master 和 agent。master 主要用于存储配置，插件和协调，agent 和 master 之间通过 agent 这个 Pod 里的 jnlp 容器通信。流水线的工作都在 agent 上，是资源消耗大户。

本文推荐的配置是基于客户并行运行的所有流水线时不会导致整个集群资源崩溃，尤其是 jenkins master 不会出现被驱逐等异常情况。能够并行多少条流水线主要取决于实际 agent 的资源消耗和 K8s 集群资源的大小。

场景 1：并行 50 条流水线

master 配置

```
resources:
  requests:
    cpu: "2"
    memory: "2Gi"
  limits:
    cpu: "2"
    memory: "2Gi"
  JavaOpts: |-
        -XX:+PrintFlagsFinal -XX:MaxRAMPercentage=70.0
        -XX:MinHeapFreeRatio=8 -XX:MaxHeapFreeRatio=15
        -XX:MinRAMPercentage=20.0 -XX:-UseAdaptiveSizePolicy
        -XX:-ShrinkHeapInSteps
        -Dhudson.slaves.NodeProvisioner.initialDelay=20
        -Dhudson.slaves.NodeProvisioner.MARGIN=50
        -Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
        -Dhudson.model.LoadStatistics.clock=5000
        -Dhudson.model.LoadStatistics.decay=0.2
        -Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000
```

-Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true
-Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000
-Djenkins.install.runSetupWizard=false
-XX:+UseConcMarkSweepGC
-XX:+UseStringDeduplication -XX:+ParallelRefProcEnabled
-XX:+DisableExplicitGC -XX:+UnlockDiagnosticVMOptions
-XX:+UnlockExperimentalVMOptions
-javaagent:/otel-auto-instrumentation/javaagent.jar

agent 配置

请参照实际的流水线消耗资源设置，因为主要的资源消耗都是来自 agent。

```
resources:
  requests:
    cpu: "100m"
    memory: "256Mi"
  limits:
    cpu: "100m"
    memory: "128Mi"
```

场景 2：并行 100 条流水线

master 配置

```
resources:
  requests:
    cpu: "2"
    memory: "3Gi"
  limits:
    cpu: "2"
    memory: "3Gi"
  JavaOpts: |-
    -XX:+PrintFlagsFinal -XX:MaxRAMPercentage=70.0
    -XX:MinHeapFreeRatio=8 -XX:MaxHeapFreeRatio=15
```

-XX:MinRAMPercentage=20.0 -XX:-UseAdaptiveSizePolicy
-XX:-ShrinkHeapInSteps
-Dhudson.slaves.NodeProvisioner.initialDelay=20
-Dhudson.slaves.NodeProvisioner.MARGIN=50
-Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
-Dhudson.model.LoadStatistics.clock=5000
-Dhudson.model.LoadStatistics.decay=0.2
-Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000
-Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true
-Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000
-Djenkins.install.runSetupWizard=false
-XX:+UseConcMarkSweepGC
-XX:+UseStringDeduplication -XX:+ParallelRefProcEnabled
-XX:+DisableExplicitGC -XX:+UnlockDiagnosticVMOptions
-XX:+UnlockExperimentalVMOptions
-javaagent:/otel-auto-instrumentation/javaagent.jar

agent 配置

参照场景 1 的 agent 配置

场景 3：并行 200 条流水线

master 配置

```yaml
resources:
  requests:
    cpu: "2"
    memory: "3Gi"
  limits:
    cpu: "2"
    memory: "3Gi"
  JavaOpts: |-
    -XX:+PrintFlagsFinal -XX:MaxRAMPercentage=70.0
    -XX:MinHeapFreeRatio=8 -XX:MaxHeapFreeRatio=15
    -XX:MinRAMPercentage=20.0 -XX:-UseAdaptiveSizePolicy
```

-XX:-ShrinkHeapInSteps

-Dhudson.slaves.NodeProvisioner.initialDelay=20

-Dhudson.slaves.NodeProvisioner.MARGIN=50

-Dhudson.slaves.NodeProvisioner.MARGIN0=0.85

-Dhudson.model.LoadStatistics.clock=5000

-Dhudson.model.LoadStatistics.decay=0.2

-Dhudson.slaves.NodeProvisioner.recurrencePeriod=5000

-Dhudson.security.csrf.DefaultCrumbIssuer.EXCLUDE_SESSION_ID=true

-Dio.jenkins.plugins.casc.ConfigurationAsCode.initialDelay=10000

-Djenkins.install.runSetupWizard=false

-XX:+UseConcMarkSweepGC

-XX:+UseStringDeduplication -XX:+ParallelRefProcEnabled

-XX:+DisableExplicitGC -XX:+UnlockDiagnosticVMOptions

-XX:+UnlockExperimentalVMOptions

-javaagent:/otel-auto-instrumentation/javaagent.jar

agent 配置

参照场景 1 的 agent 配置

注意事项

- 当 jenkins pod 因为 OOM 重启时，建议加大 master 的内存。为了保证 Qos，建议 master 的内存和 cpu 的 request 和 limit 保持一致。
- 当流水线模块接口调用存在超时情况时，建议加大 master 的 CPU。
- 当 master 内存配置超过 4G 时，建议修改 JavaOpts 中的-XX:+UseConcMarkSweepGC 为-XX:+UseG1GC。

快速创建流水线

本节将通过编译、构建、部署来创建流水线，通过详细的步骤操作描述，帮助您快速创建一条流水线。

前提条件

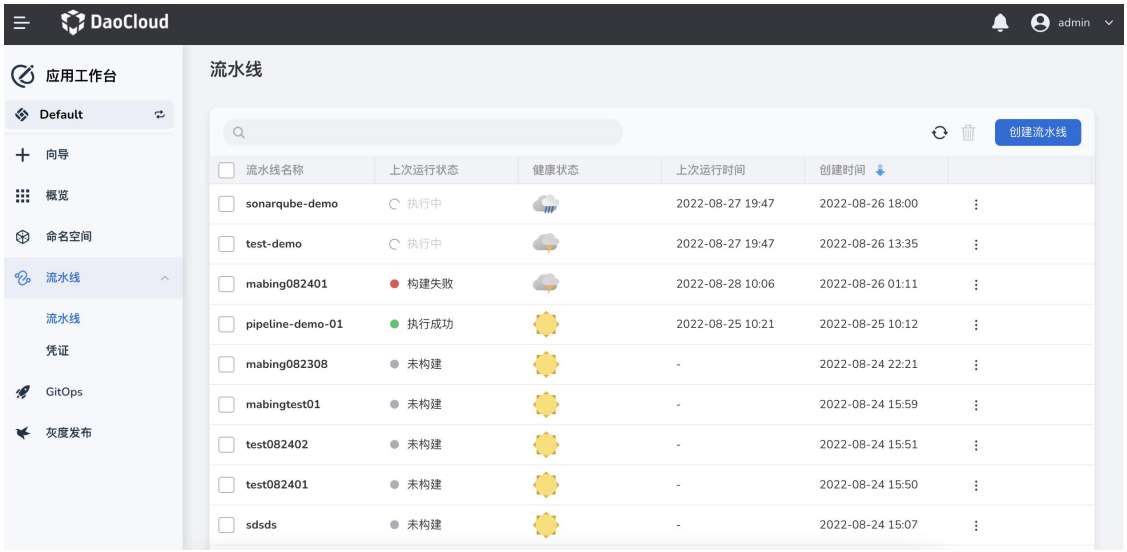
- 需创建一个工作空间和一个用户，该用户需加入该工作空间并赋予 workspace edit 角色。参考创建工作空间、用户和角色。
- 创建可以访问镜像仓库、集群的两个凭证，分别命名为：registry、kubeconfig。创建凭证的更多信息，请参考凭证管理。
- 准备一个 GitHub 仓库、DockerHub 仓库。

创建凭证

1. 在凭证页面创建两个凭证：
    - docker-credential：用户名和密码，用于访问镜像仓库。
    - demo-dev-kubeconfig：用于使用这个 kubeconfig 访问 Kubernetes 集群。
2. 创建完成后，可以在凭证列表页面看到凭证信息。

创建自定义流水线

1. 在流水线列表页点击创建流水线。



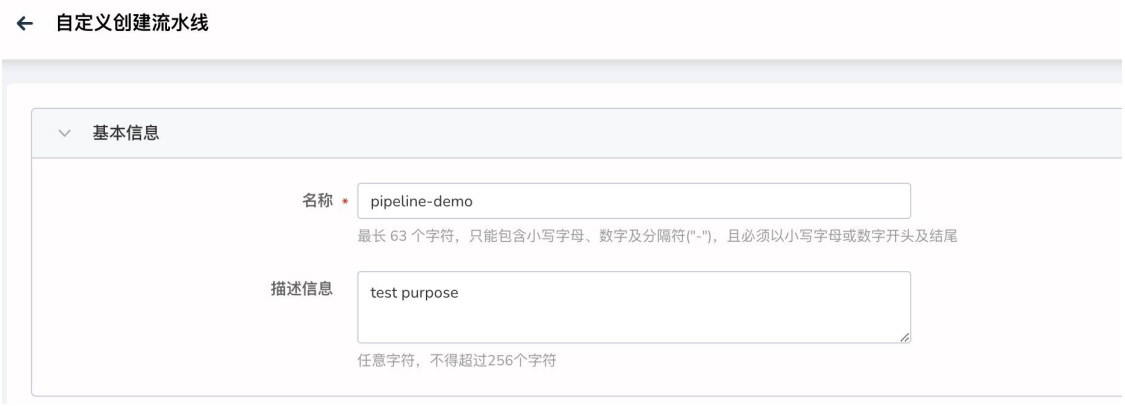
pipeline

2. 在弹出的对话框中，选择自定义创建流水线，点击确定。



pipeline

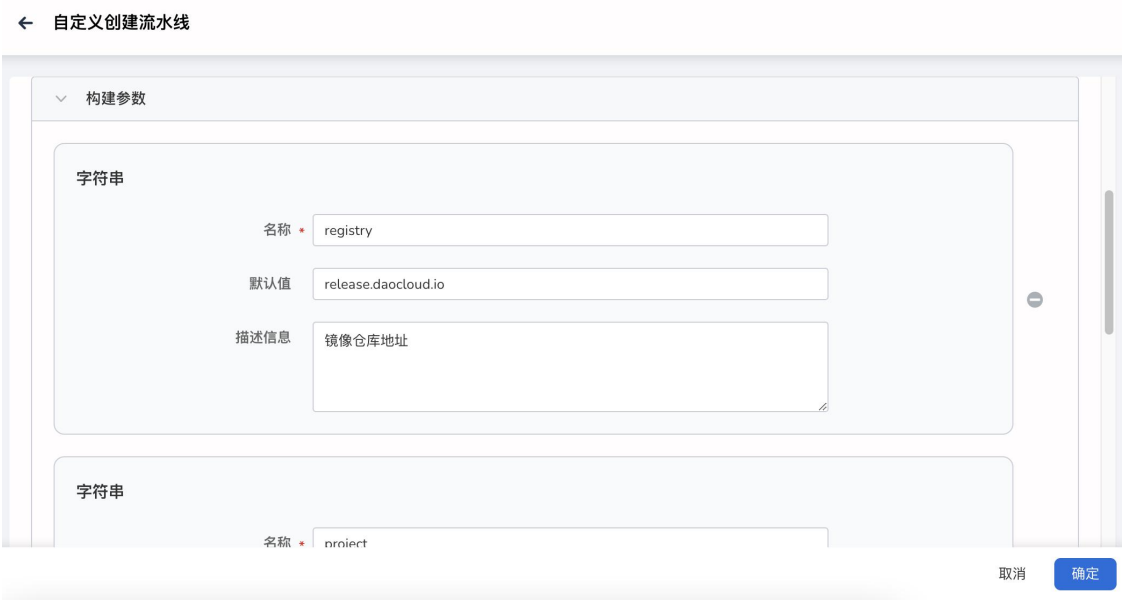
3. 进入自定义创建流水页面 ， 输入流水线名称 pipeline-demo。



pipeline

4. 在构建参数中添加三个字符串参数，这些参数将用于镜像构建的命令中。

   - registry：镜像仓库地址。示例值：release.daocloud.io。
   - project：镜像仓库中的项目名称。示例值：demo。
   - name：镜像的名称。示例值：http-hello。

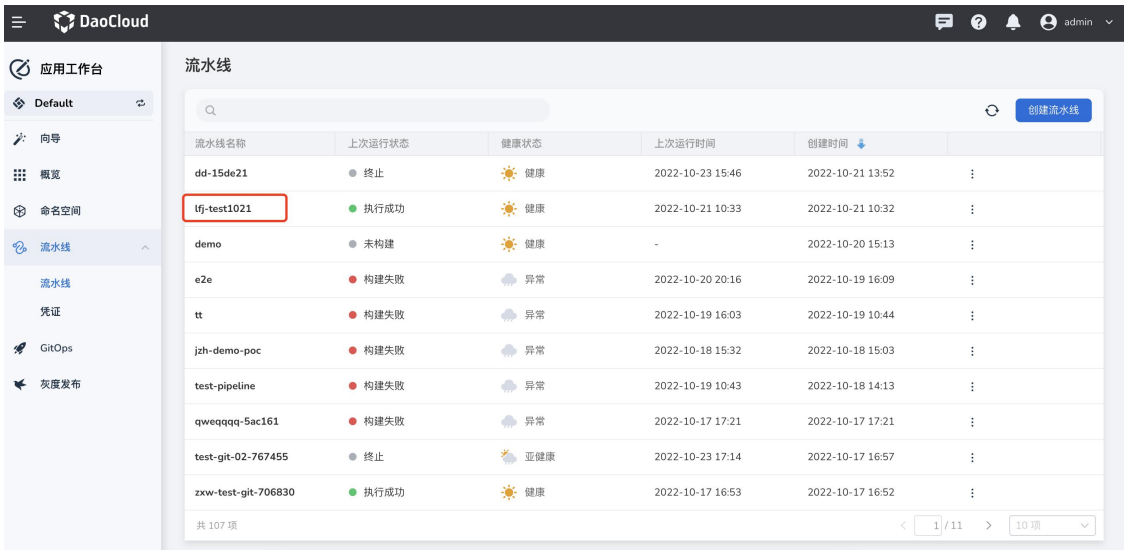


pipeline

5. 添加完成后，点击确定。

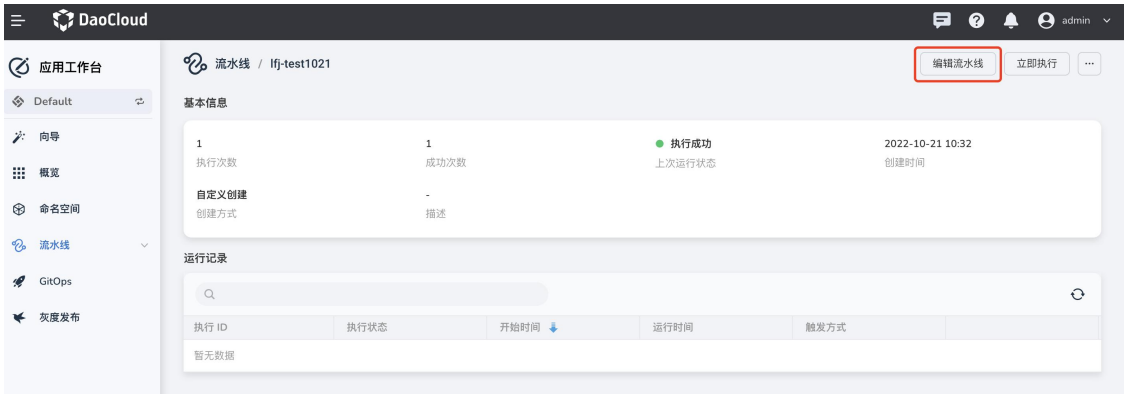
编辑流水线

1. 在流水线列表页面点击一个流水线的名称。



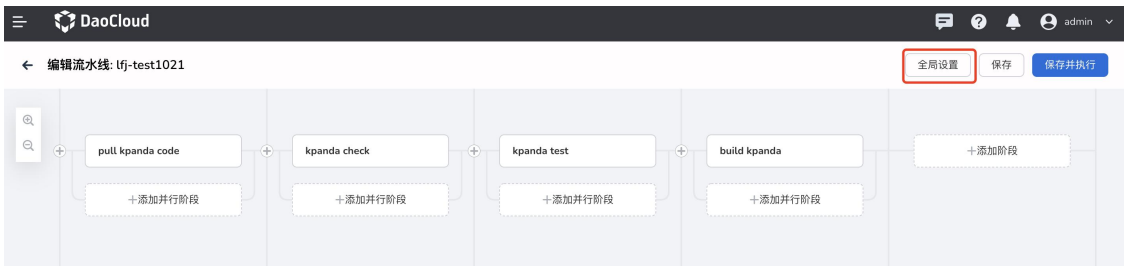
pipelisetting

2. 在右上角点击编辑流水线，



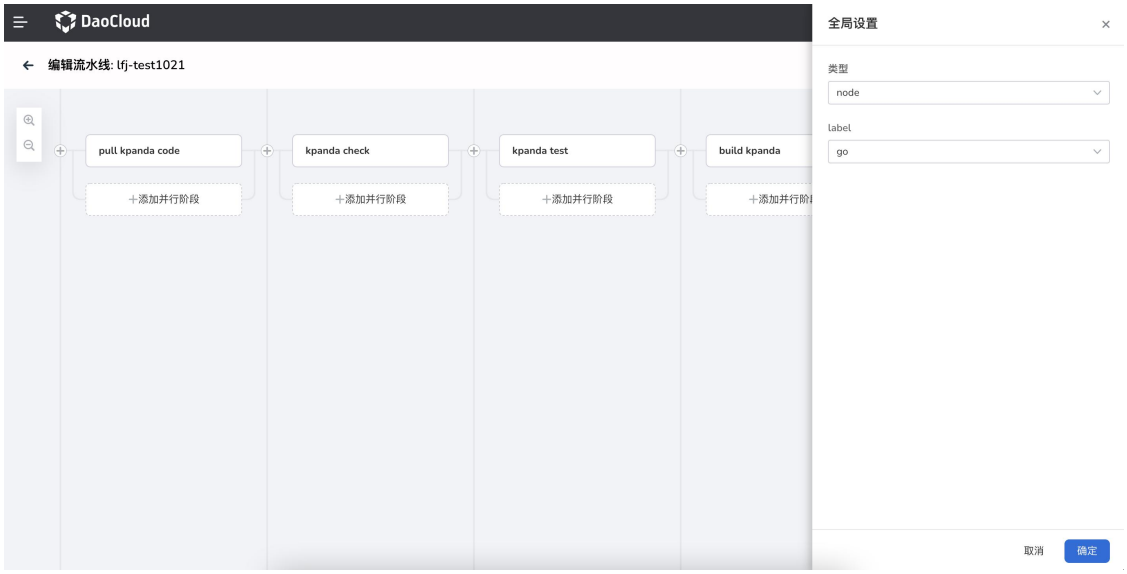
pipelisetting

3. 在右上角点击全局设置。



pipelisetting

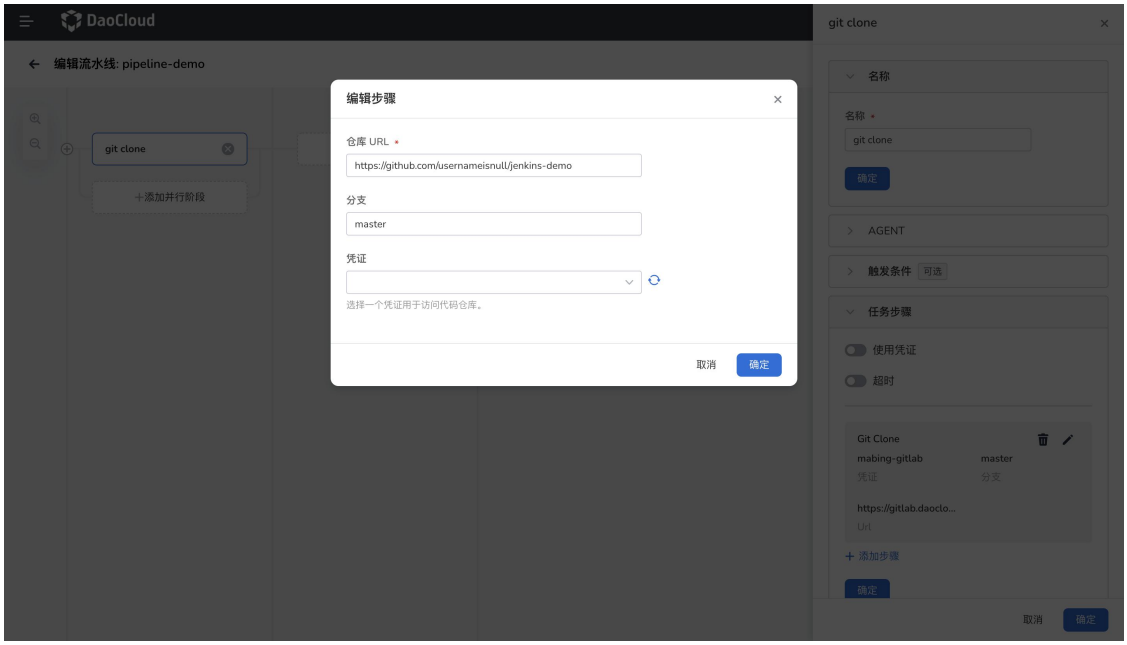
4. 类型设为 node，且 label 设为 go，点击确定。



pipelisetting

5. 添加阶段 – 拉取源代码。

   - 点击画布中的添加阶段。在右侧的阶段设置中设置名称：git clone。
   - 点击添加步骤，在弹出对话框中步骤类型下选择 git clone，配置相关参数：
     - 仓库 URL：输入 GitLab 仓库地址。
     - 分支：不填写默认为 master 分支。
     - 凭证：如果属于私有仓库，则需要提供一个凭证。

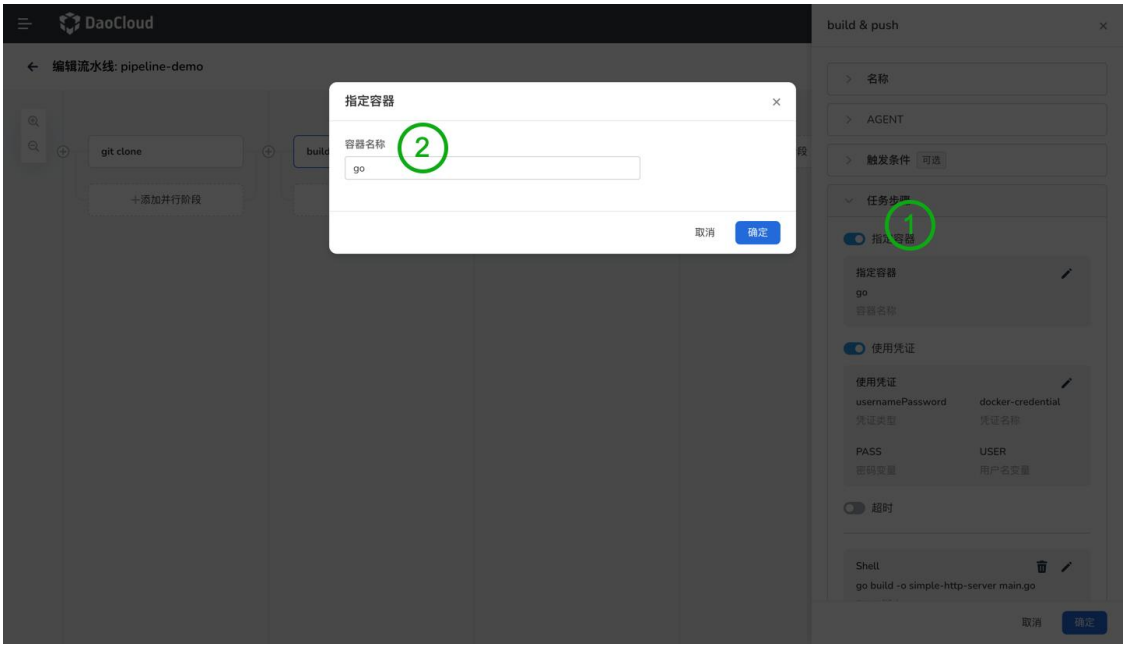


quickstart01

6. 添加阶段 – 构建并推送镜像。

   - 点击画布中的添加阶段。在右侧的阶段设置中设置名称：build & push。

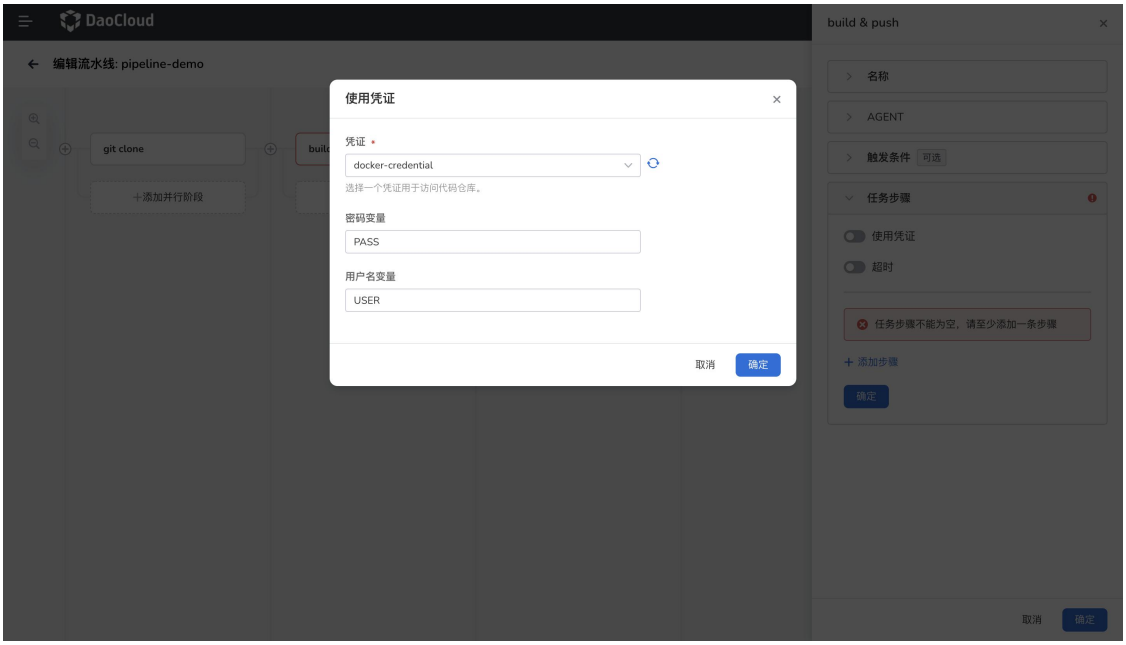
   - 在步骤模块中选择开启指定容器，在弹出的对话框中填写容器名称：go，然后点击确定。



container

- 在步骤模块中选择开启使用凭证，在弹出的对话框中填写相关参数，然后点击确定。

    - 凭证：选择创建的 Docker hub 凭证，用户访问镜像仓库。
    - 密码变量：PASS
    - 用户名变量：USER



quickstart02

- 点击添加步骤进行代码构建，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

  ```
  go build -o simple-http-server main.go
  ```

- 点击添加步骤以根据源码中的 Dockerfile 构建 Docker 镜像，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

  ```
  docker build -f Dockerfile . -t $registry/$project/$name:latest
  ```

- 点击添加步骤以登录镜像仓库，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

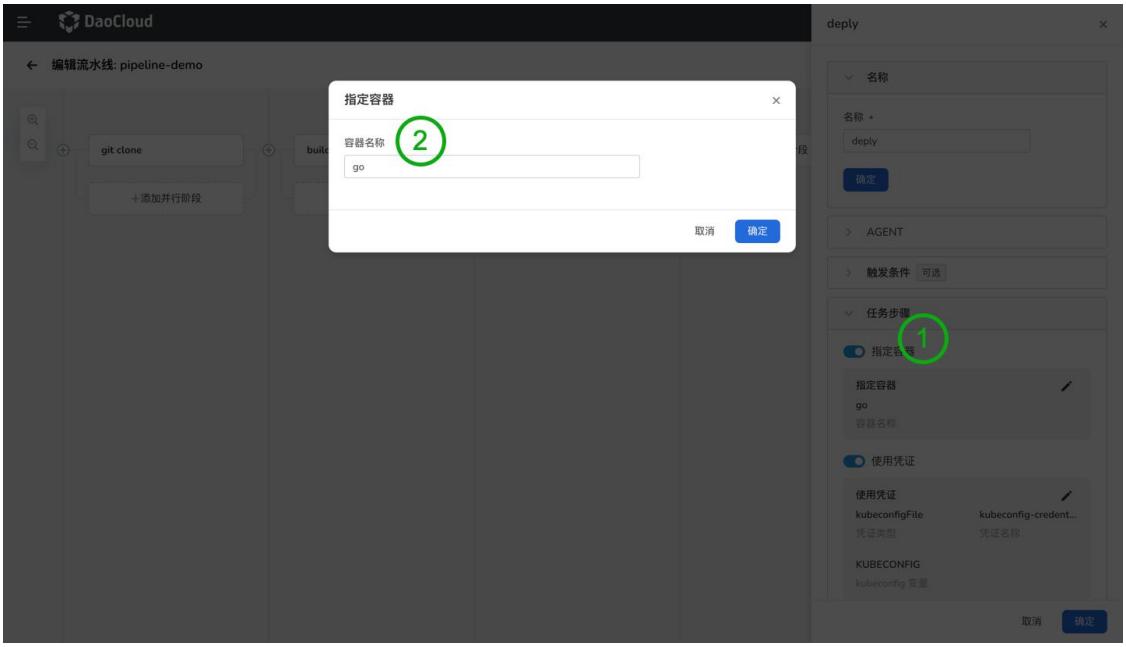
  ```
  docker login $registry -u $USER -p $PASS
  ```

- 点击添加步骤将镜像推送至镜像仓库中，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

  ```
  docker push $registry/$project/$name:latest
  ```

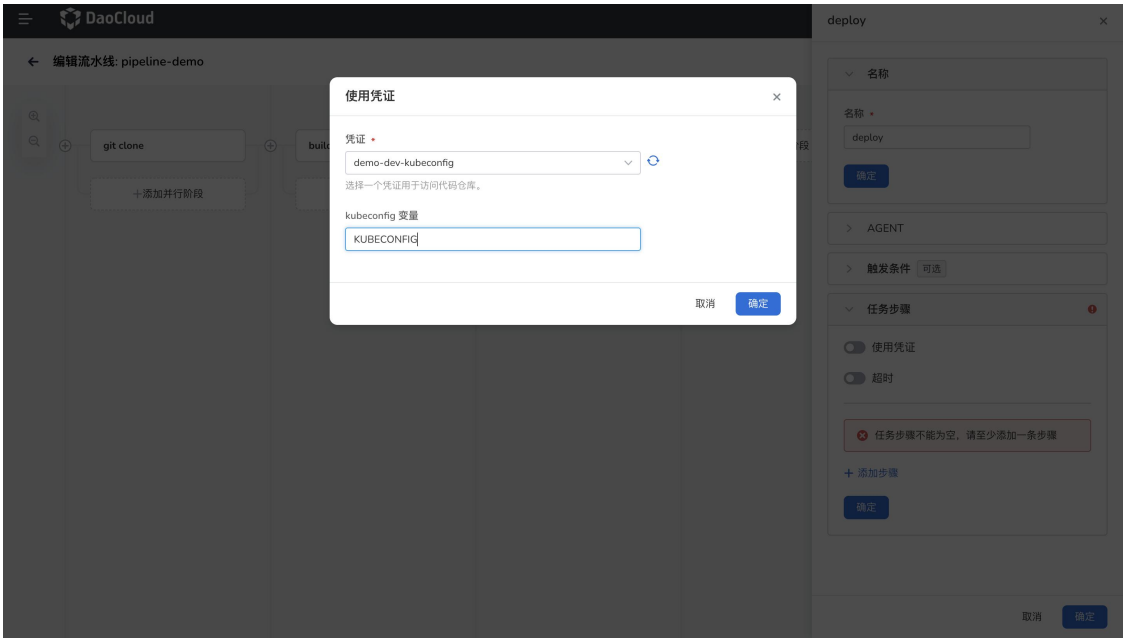
7. 添加阶段 – 部署至集群

   - 点击画布中的添加阶段。在右侧的阶段设置中设置名称：deploy。
   - 在步骤模块中选择开启指定容器，在弹出的对话框中填写容器名称：go，然后点击确定。



container2

   - 在步骤模块中选择开启使用凭证，在弹出的对话框中填写相关参数，然后点击确定。
     - 凭证：选择 kubeconfig 类型的凭证。
     - kubeconfig 变量：如果您使用的是 kubectl apply 的部署方式，变量值必须为 KUBECONFIG。



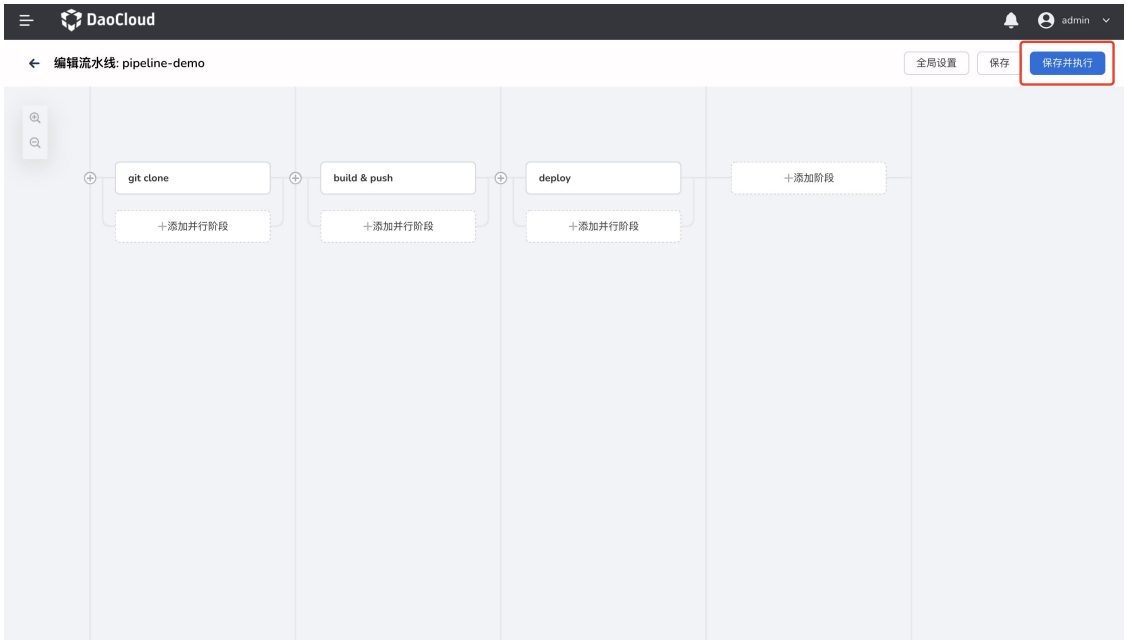
quickstart03

- 点击添加步骤以进行集群部署操作，在弹出的对话框中步骤类型下选择 shell，参并在命令行中输入以下命令，然后点击确定。

  kubectl apply -f deploy.yaml

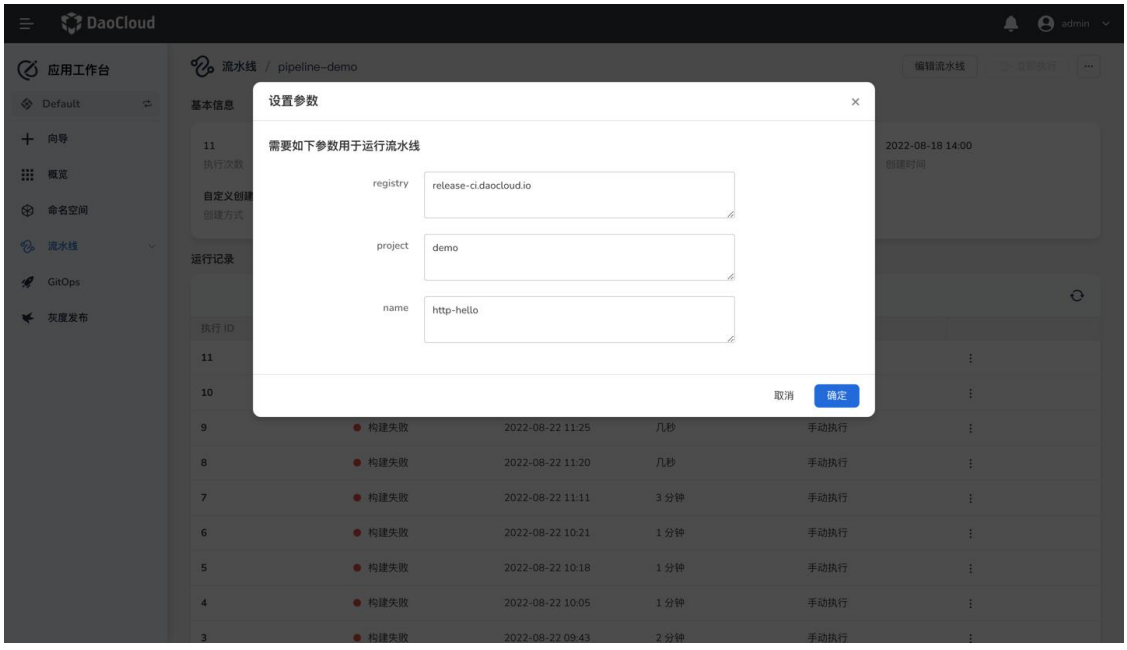
保存并执行流水线流水线

1. 完成上一步后点击保存并执行。



quickstart05

2. 在弹出的对话框中输入步骤二中的示例参数。点击确定即可成功运行该流水线。

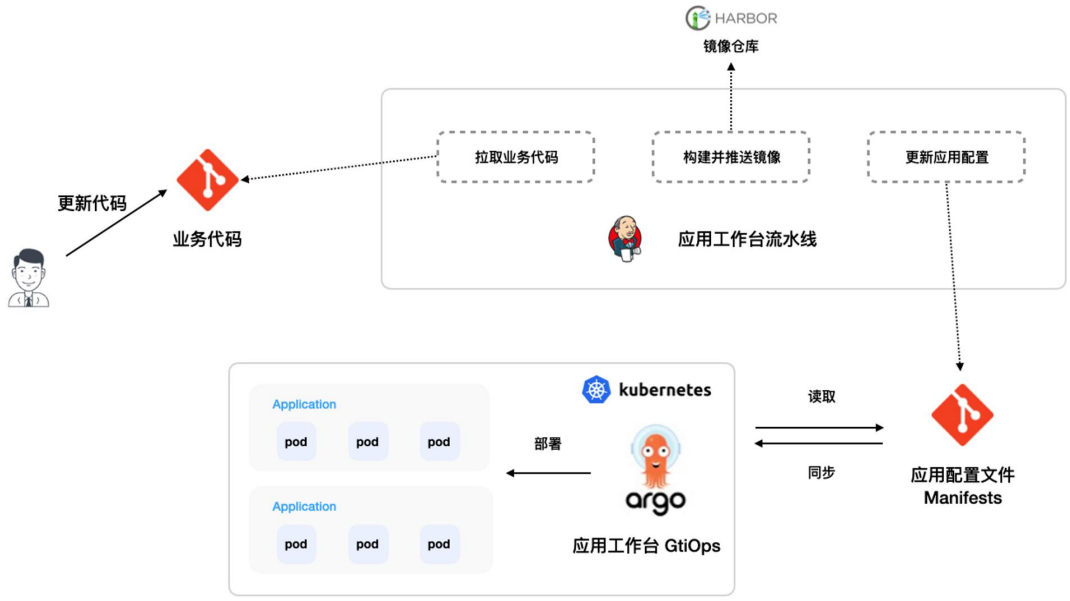


build-para

基于流水线和 GitOps 实现 CI/CD

本文介绍如何基于应用工作台的流水线与 GitOps 功能实现 CI/CD。

整体流程



cd:ci01

前提条件

基于整体流程，我们需要准备如下信息：

1. 准备两个代码仓库，其中一个作为存业务代码的仓库，一个作为应用的配置文件（yaml）仓库。

   可以参考 GitHub 中的配置：

   业务代码：<https://github.com/jzhupup/dao-2048.git>
   应用配置：<https://github.com/jzhupup/argocd-example-apps.git>

2. 准备一个 Harbor 镜像仓库

3. 准备访问上述三个仓库的凭证。此教程中使用的三个凭证分别命为 git-credentials、git-app-credentials、harbor-credentials

创建流水线

此教程主要包含的步骤有：拉取业务代码→构建镜像→更新应用配置文件

当更新应用配置文件步骤执行成功后，Argo CD 会监测变化，并会触发更新同步最新的配置文件部署到集群中。

1. 创建流水线的步骤可以参考: 创建流水线。

2. 创建成功后，选择该流水线操作：编辑 Jenkinsfile

   ??? note “点击查看流水线 Jenkinsfile 文件，可根据实际需要参数”

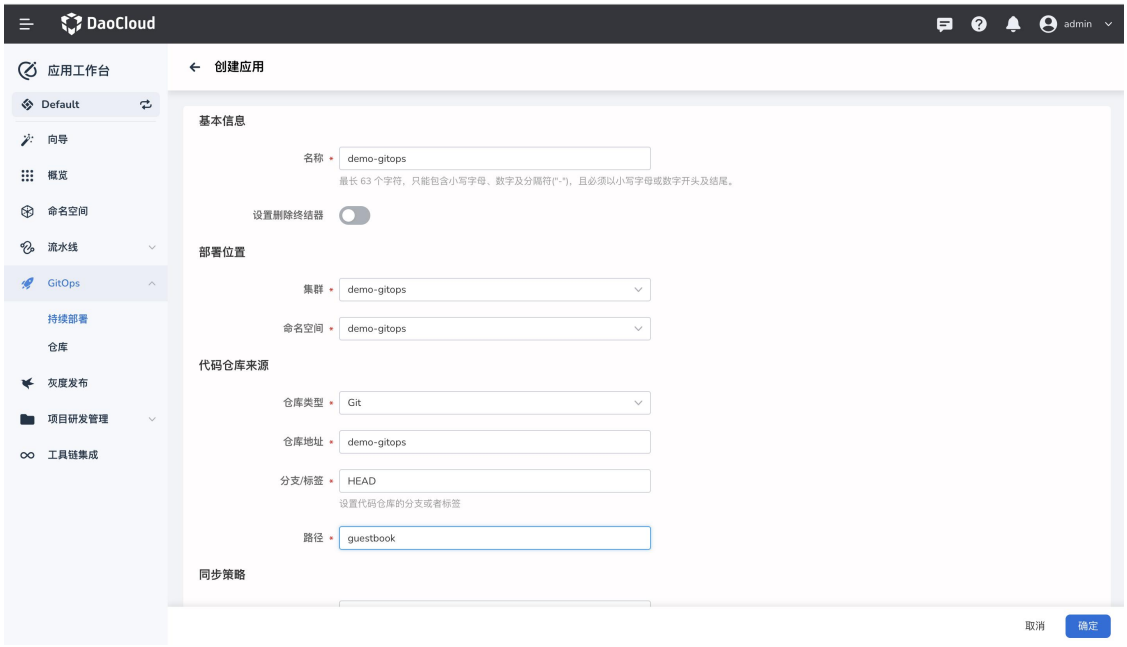
   ```yaml
   pipeline {
     agent {
       node {
         label 'maven'
       }
     }
     parameters {
       string(name: 'DOCKER_REPO', defaultValue: 'release-ci.daocloud.io/test-jzh/dao-2048', description: '镜像名称')
       string(name: 'DOCKER_IMAGE_VERSION', defaultValue: 'v2.0', description: '镜像版本')
     }
     stages {
       stage('git clone') {
         agent none
         steps {
           git(branch: 'main', credentialsId: ' git-credentials', url: '<http://172.30.40.32:9980/gitlab-instance-facdd89d/dao-2048.git>')
   ```

```
        }
      }
      stage('docker build & push') {
        agent none
        steps {
          container('maven') {
            withCredentials([usernamePassword(passwordVariable:'PASS',us
ernameVariable:'USER',credentialsId:'harbor-credentials')]) {
              sh 'docker login ${DOCKER_REPO} -u $USER -p $PASS'
              sh 'docker build -f Dockerfile -t ${DOCKER_REPO}:${DOCK
ER_IMAGE_VERSION} .'
              sh 'docker push ${DOCKER_REPO}:${DOCKER_IMAGE_VERSI
ON}'
            }
          }
        }
      }
      stage('update config') {
        agent none
        steps {
          withCredentials([usernamePassword(passwordVariable:'password',u
sernameVariable:'username',credentialsId:'git-app-credentials')]) {
            sh """
              git config --global user.name "root"
              git config --global user.email "test@gmail.com"
              git clone <http://${username}:${password}@172.30.40.32:998
0/gitlab-instance-facdd89d/argocd-example-apps.git>

              cd  ./argocd-example-apps
              sed -i "s#${DOCKER_REPO}.*#${DOCKER_REPO}:${DOCKER
_IMAGE_VERSION}#g" guestbook/dao-2048.yaml
              git add . && git commit -m "update image"
```

```
                git push <http://${username}:${password}@172.30.40.32:9980
/gitlab-instance-facdd89d/argocd-example-apps.git>
                """
            }
        }
    }
  }
}
```

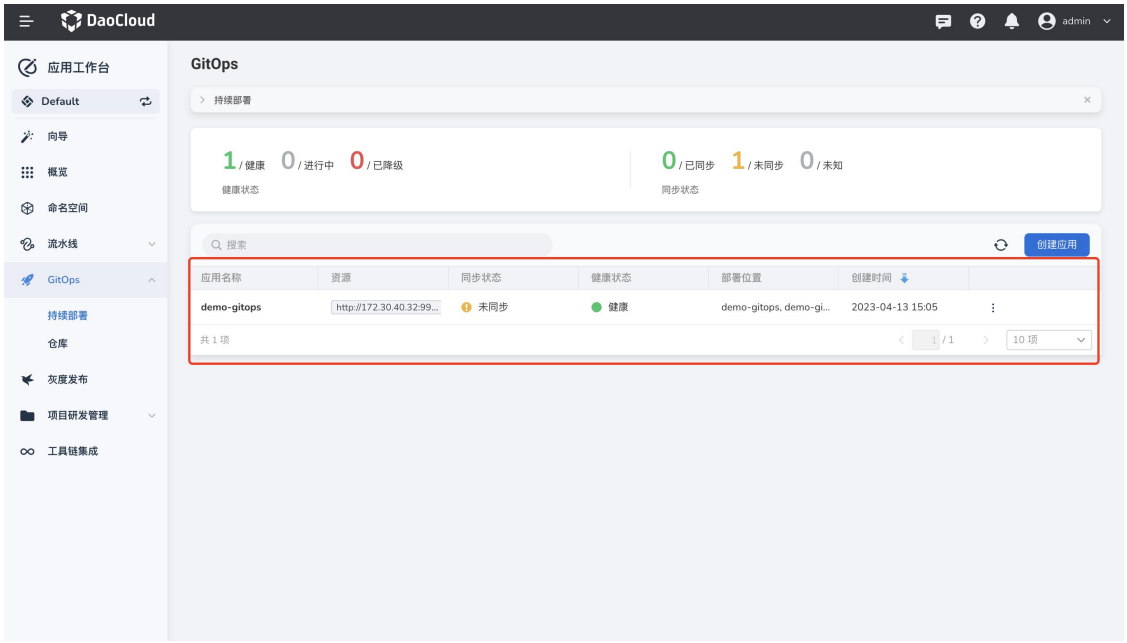
创建持续部署应用

1. 用 Https 方式导入 argocd-example-apps 仓库，参考步骤
2. 创建一个 GitOps 应用



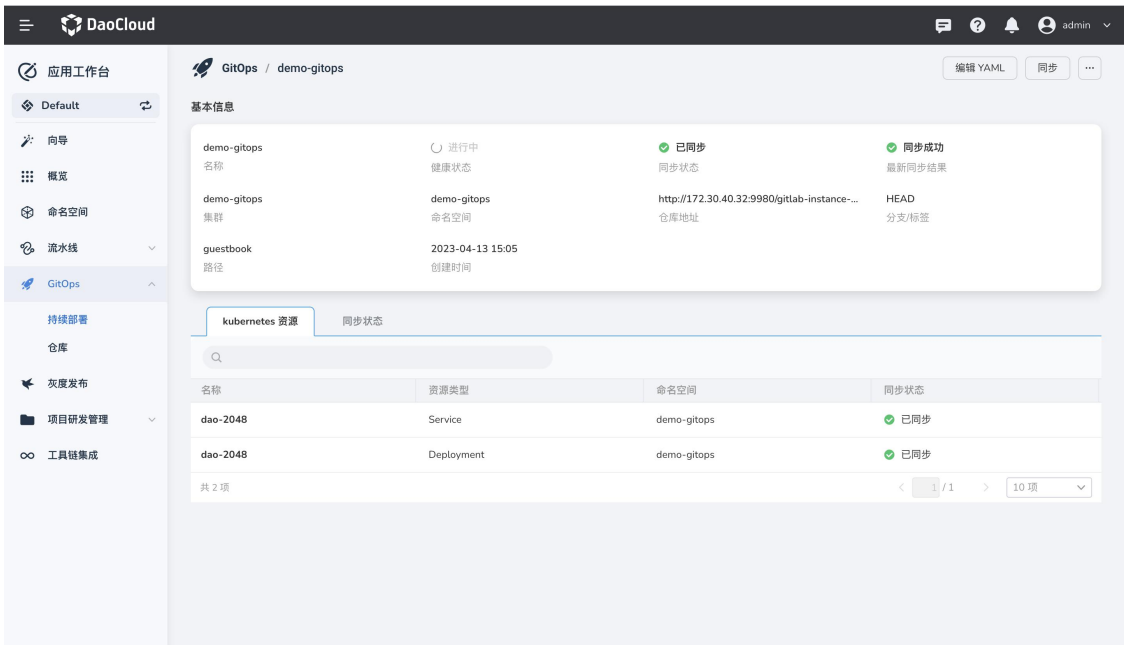
cd:ci02

3. 创建完成后，会自动生成一条记录，同步状态显示未同步



cd:ci03

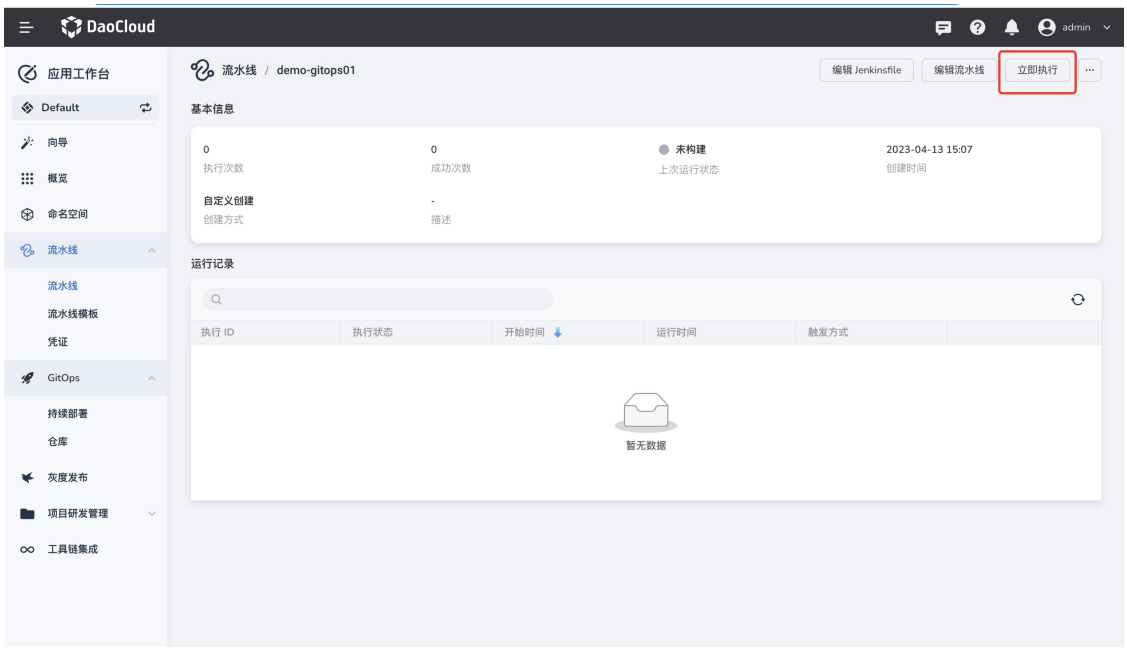
4. 点击同步，完成应用部署



cd:ci04

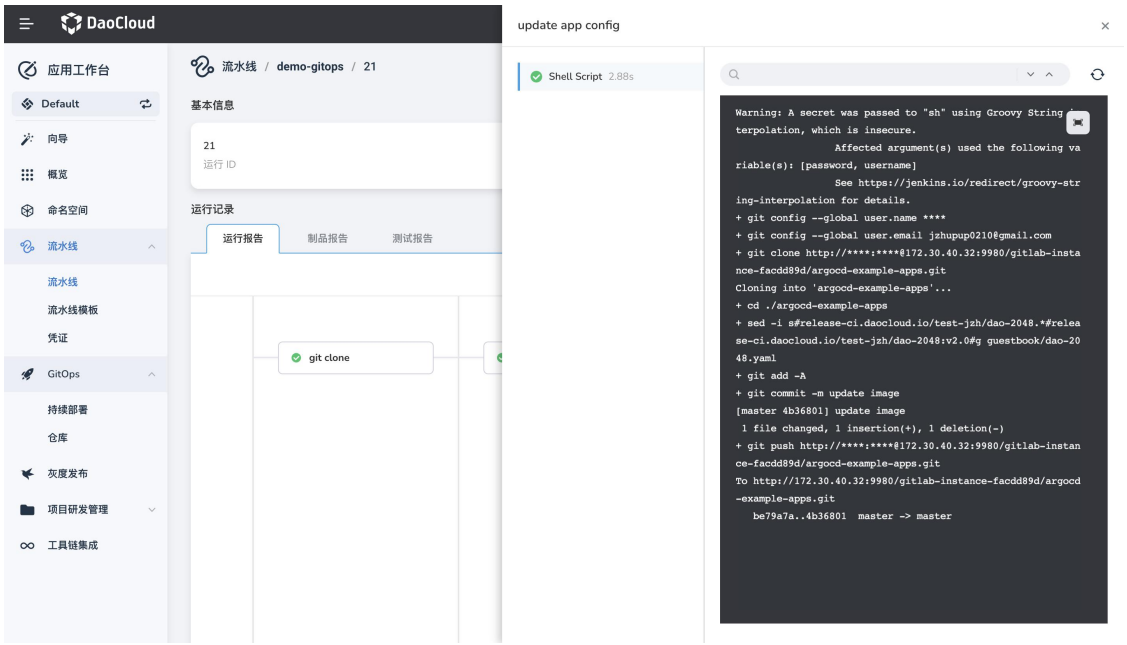
运行流水线触发 CI/CD

1. 选择上述创建的流水线，点击立即运行



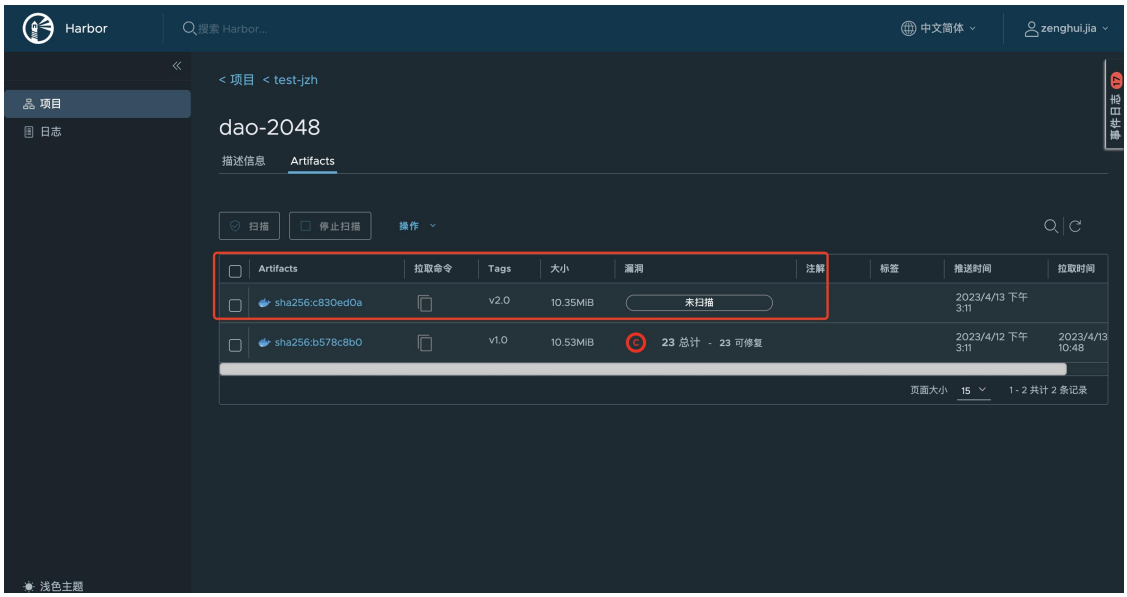
cd:ci05

2. 查看运行日志



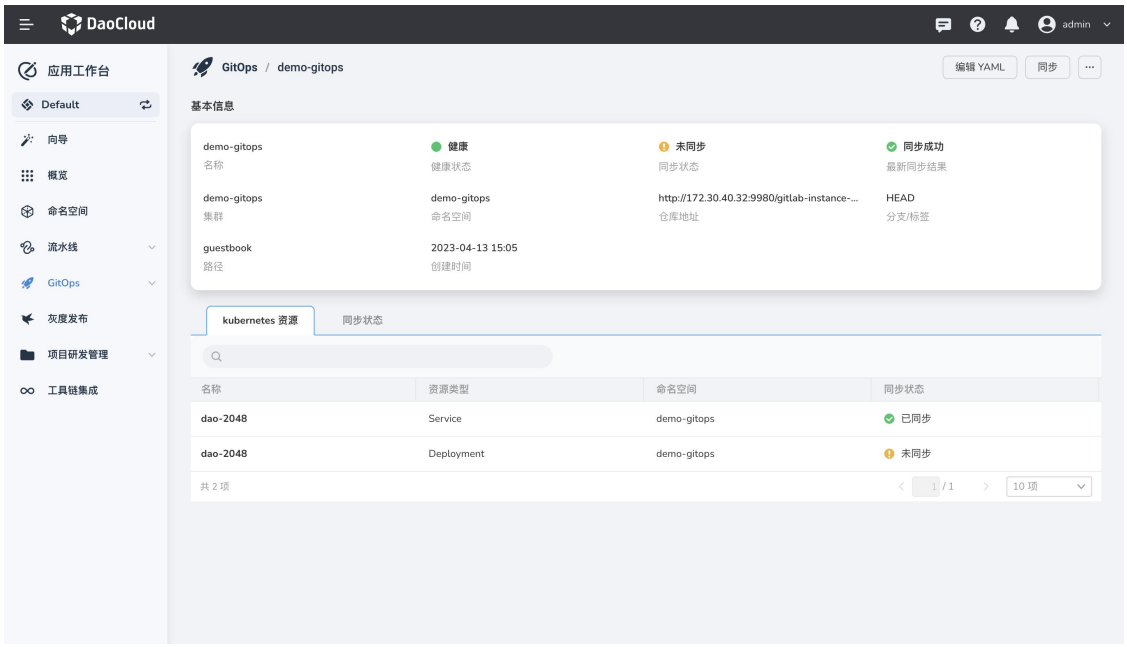
cd:ci06

3. 流水线运行成功后，验证镜像是否上传到 Harbor，Jenkinsfile 中定义的 tag 为 v2.0。

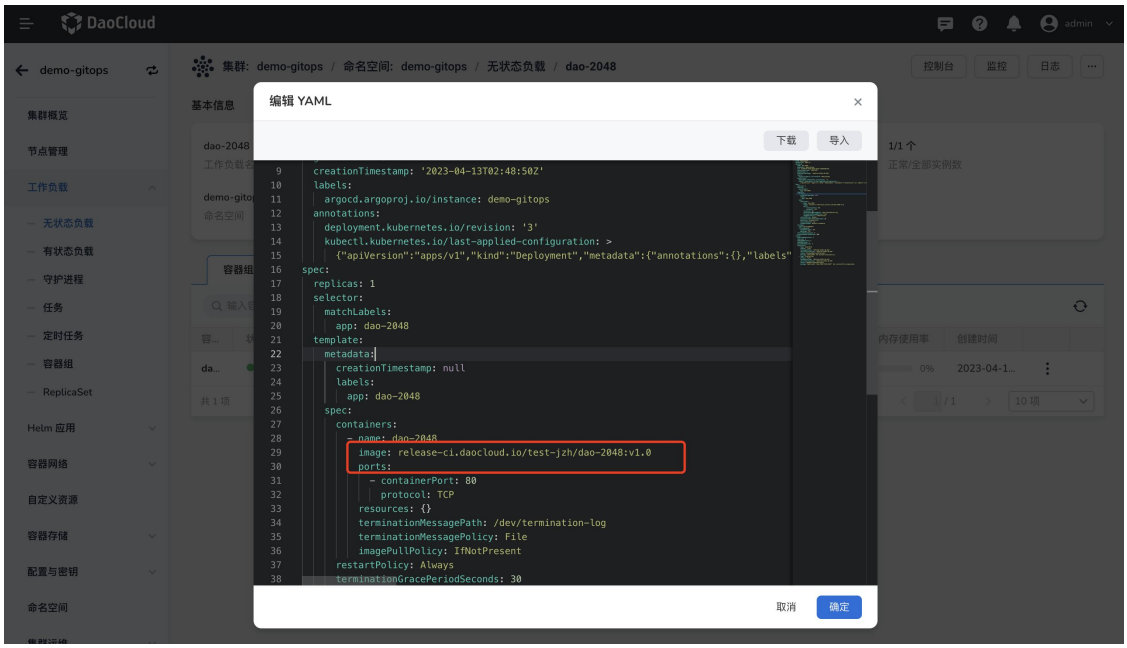


cd:ci07

4. 继续验证持续部署应用，发现处于未同步状态。看到 Deployment 资源未同步，并跳转到容器管理模块确认目前镜像版本。

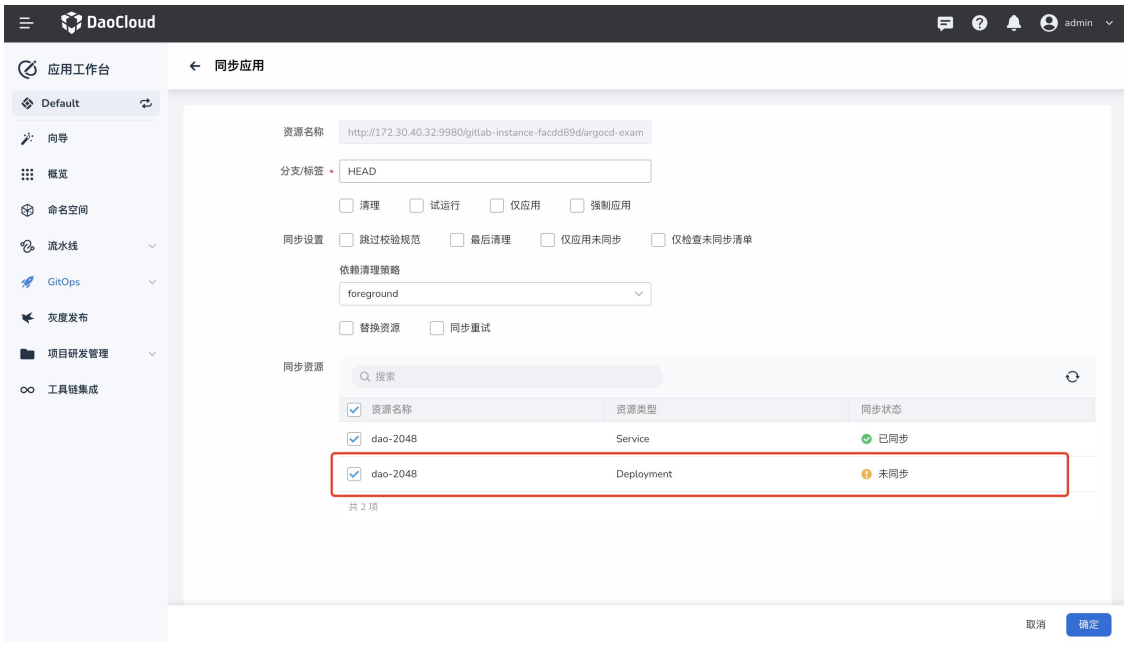


cd:ci08

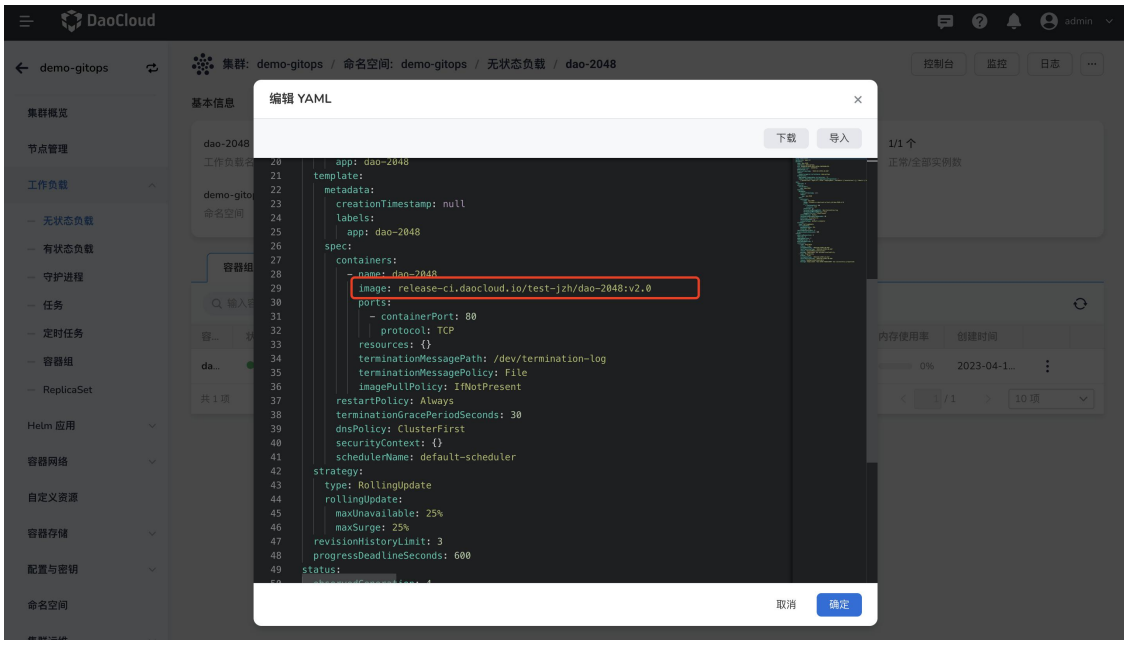


cd:ci09

5. 点击同步，等待同步成功后，查看 Deployment 资源，确认目前的镜像版本。



cd:ci11


DaoCloud
admin
demo-gitops
集群: demo-gitops / 命名空间: demo-gitops / 无状态负载 / dao-2048
控制台
监控
日志
集群概览
节点管理
工作负载
无状态负载
有状态负载
守护进程
任务
定时任务
容器组
ReplicaSet
Helm 应用
容器网络
自定义资源
容器存储
配置与密钥
命名空间
基本信息
编辑 YAML
下载
导入
app: dao-2048
template:
metadata:
creationTimestamp: null
labels:
app: dao-2048
spec:
containers:
- name: dao-2048
image: release-ci.daocloud.io/test-jzh/dao-2048:v2.0
ports:
- containerPort: 80
protocol: TCP
resources: {}
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
imagePullPolicy: IfNotPresent
restartPolicy: Always
terminationGracePeriodSeconds: 30
dnsPolicy: ClusterFirst
securityContext: {}
schedulerName: default-scheduler
strategy:
type: RollingUpdate
rollingUpdate:
maxUnavailable: 25%
maxSurge: 25%
revisionHistoryLimit: 3
progressDeadlineSeconds: 600
status:
取消
确定

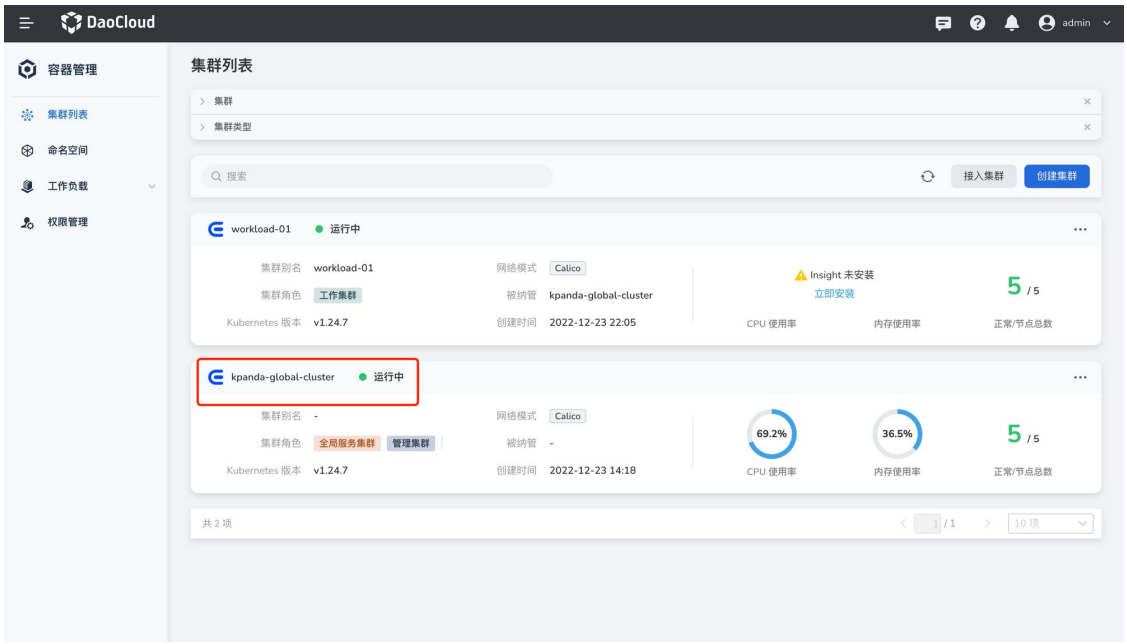
cd:ci10

在指定的节点上运行流水线

本文介绍如何在应用工作台中客户的流水线任务在指定的节点上运行。

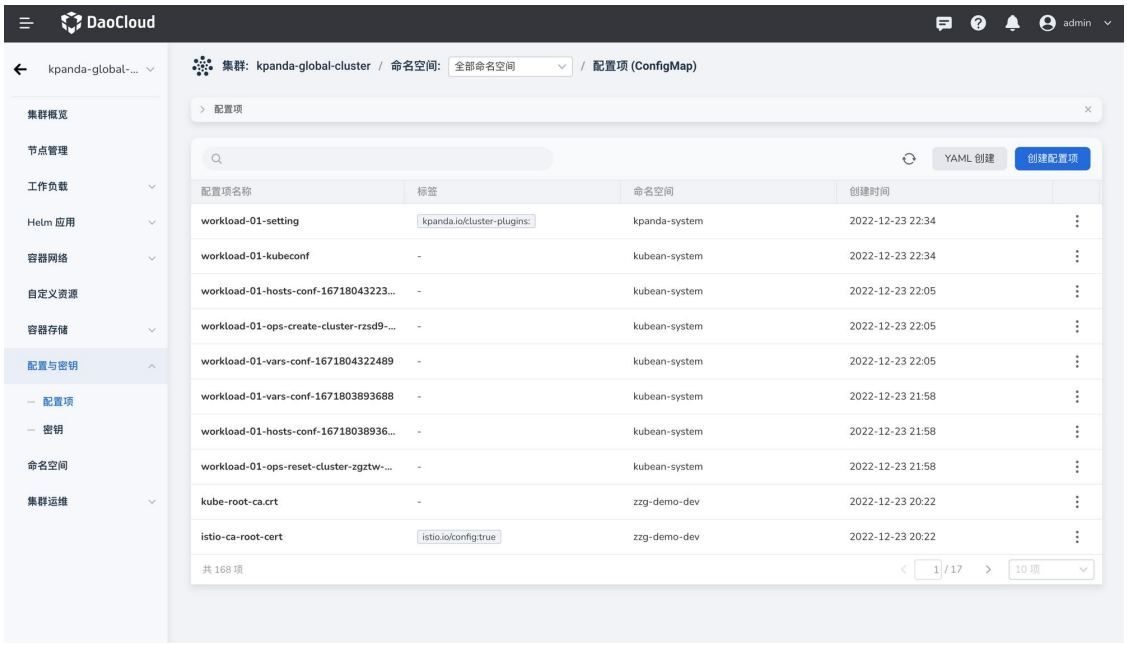
修改配置文件 jenkins-casc-config

1. 前往容器管理模块，进入目标集群的详情页面，例如 kpanda-global-cluster 集群。



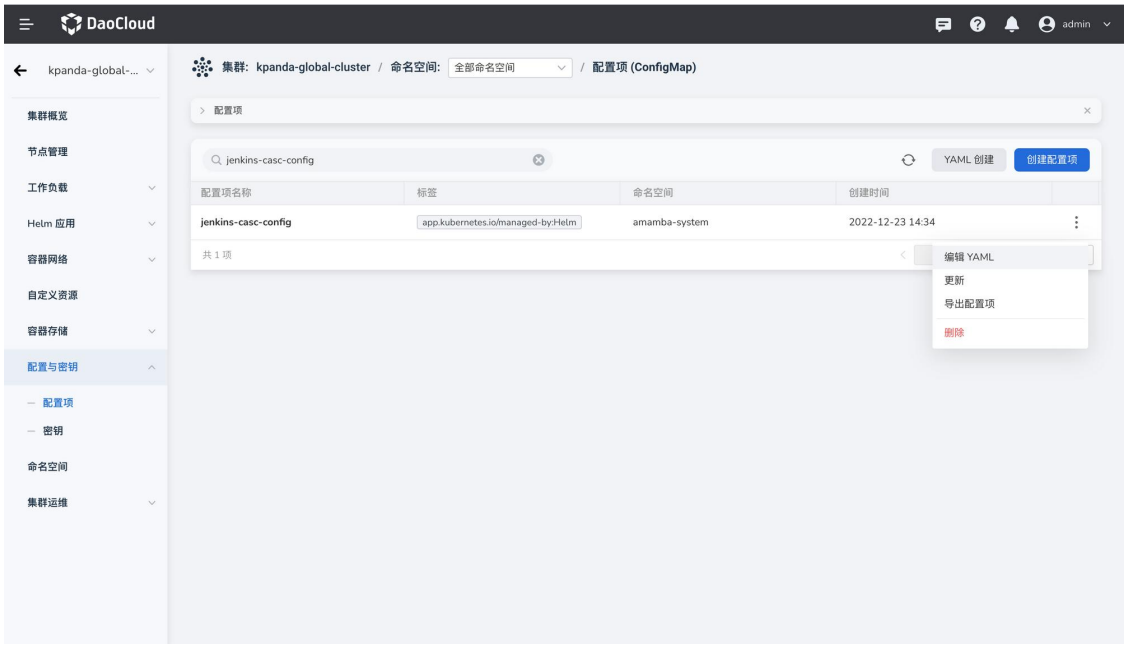
进入集群详情

2. 在左侧导航点击配置与密钥->配置项。



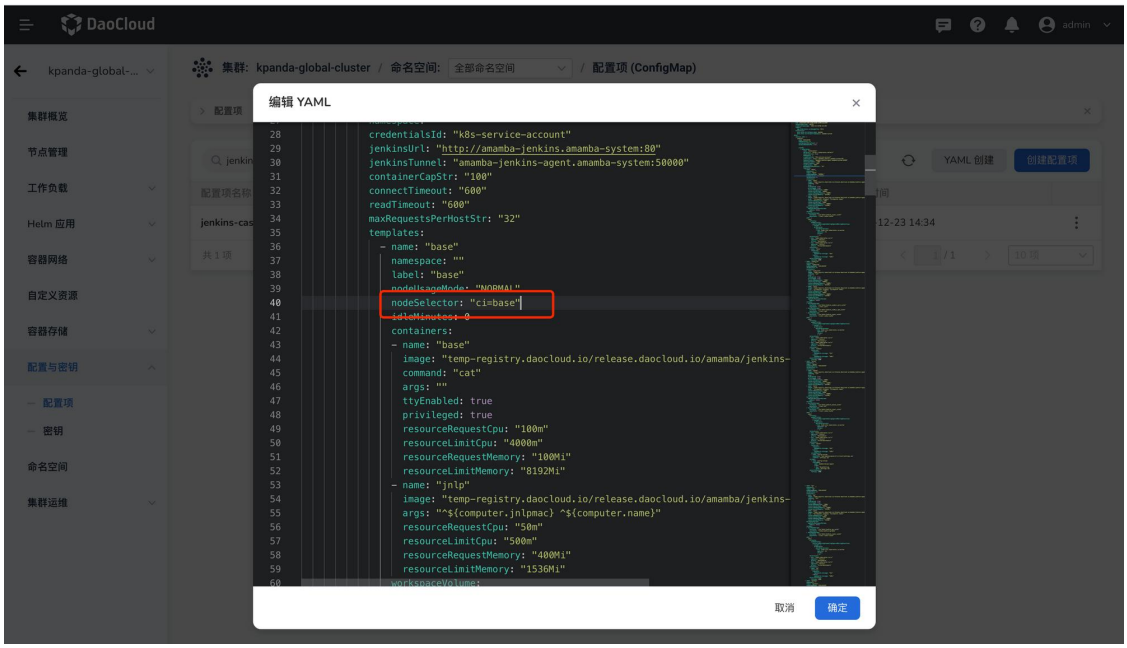
配置项

3. 搜索 jenkins-casc-config，在列表选择编辑 YAML 。



编辑 YAML

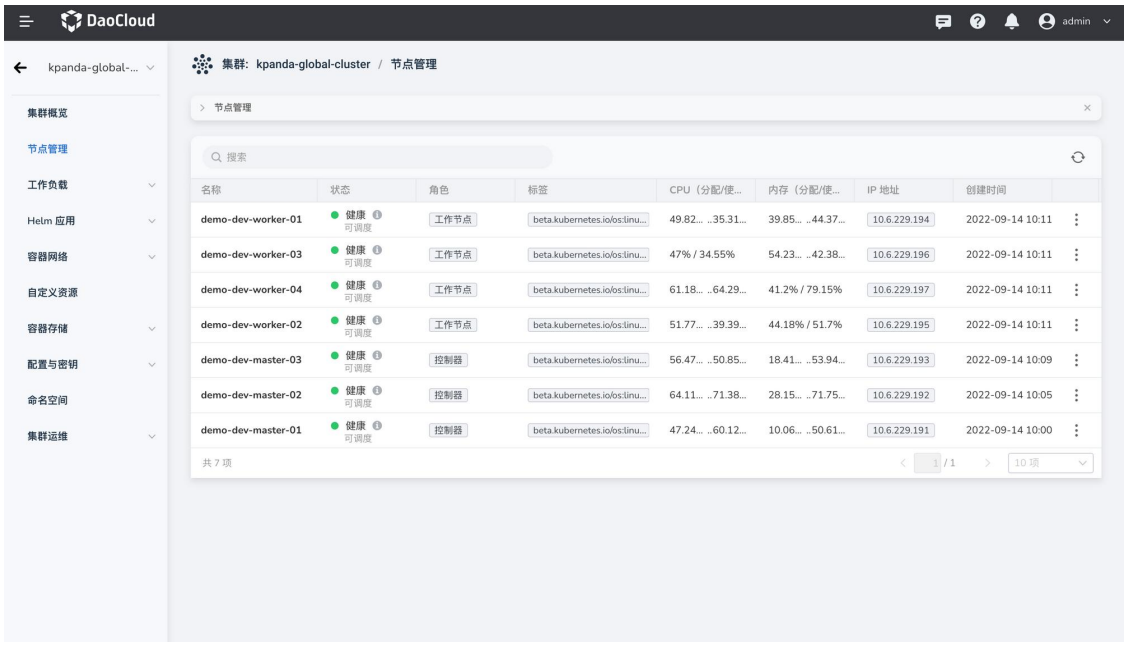
4. 在 YAML 配置项 jenkins.yaml 中的 jenkins.cloud.kubernetes.templates 位置下为某个具体的 Agent 添加 nodeSelector: "ci=base" ，点击确定保存更改。



添加 selector

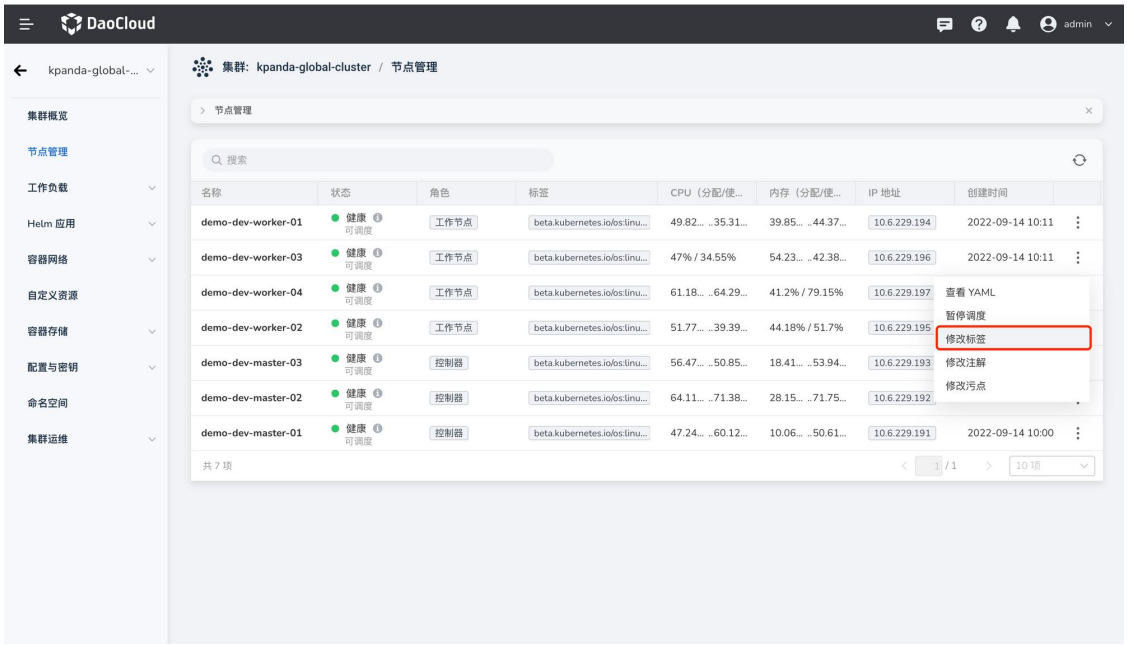
选择指定的节点添加标签

1. 进入容器管理模块，在 kpanda-global-cluster 集群的详情页面，在左侧导航点击节点管理。



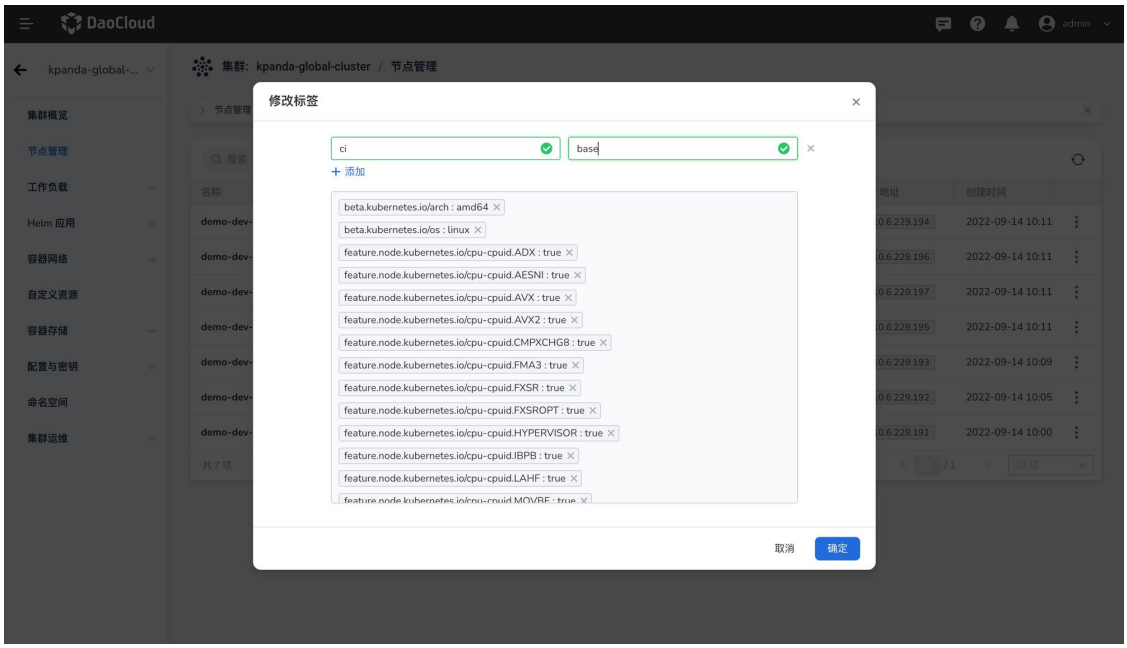
节点管理

2. 选择目标的工作节点（例如 demo-dev-worker-03），点击修改标签 。



修改标签

3. 添加 ci=base 标签，点击确定保存更改。

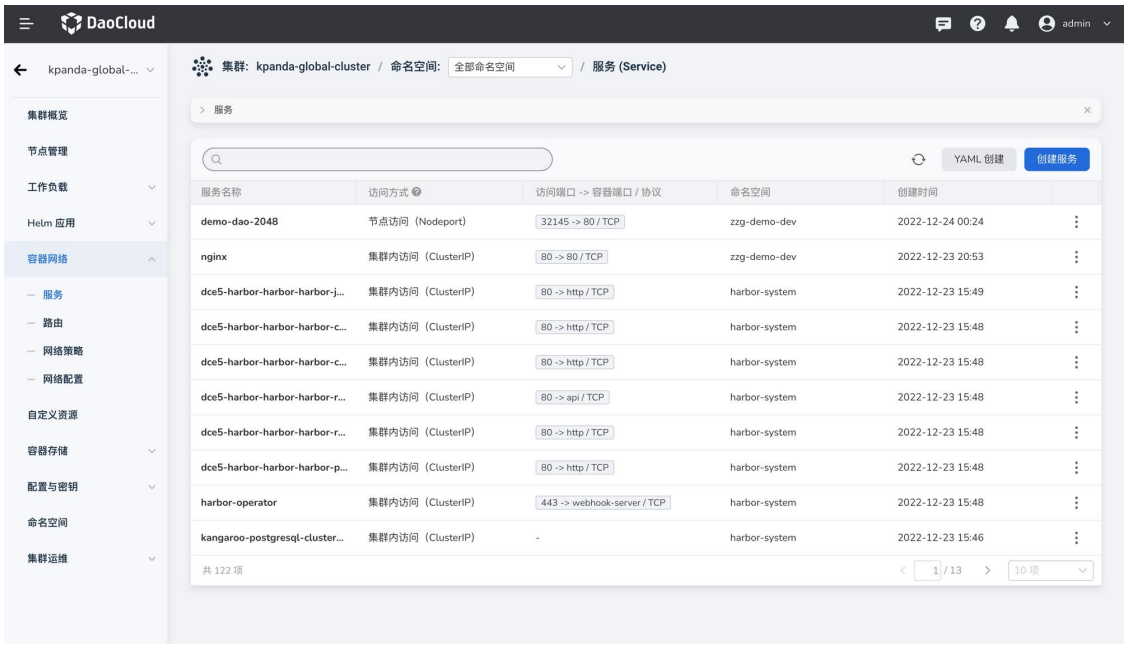


添加标签

访问 Jenkins Dashbord，重新加载配置

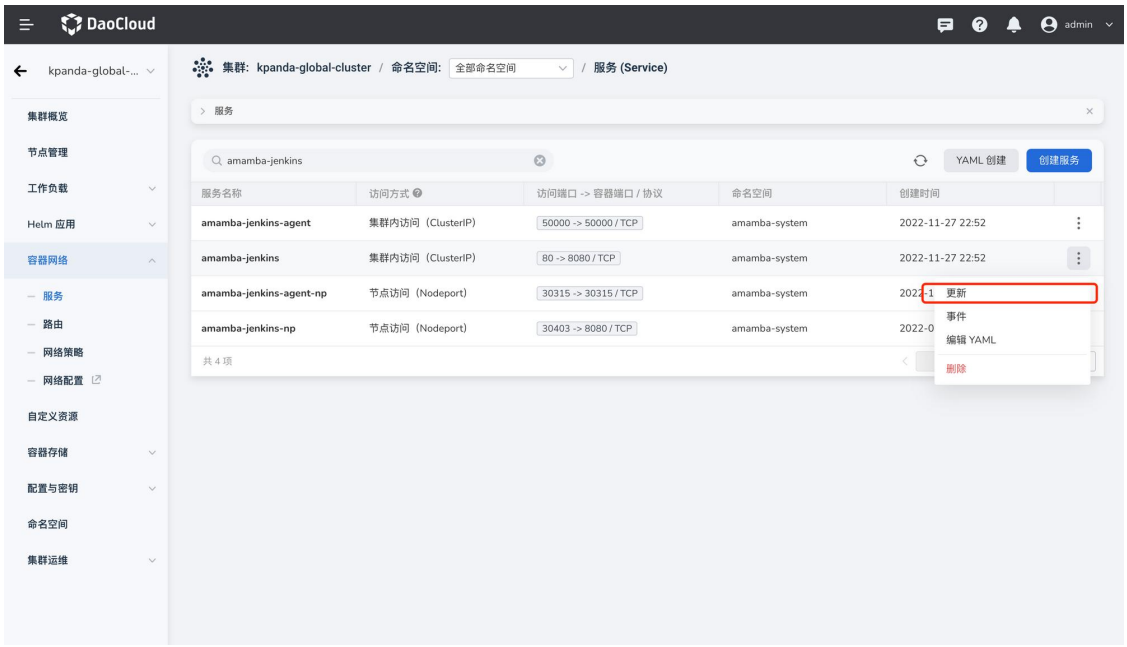
首先需要讲通过 NodePort 方式暴露 Jenkins Dashbord 的访问地址（其他暴露方式根据业务实际情况进行暴露）。

1. 进入容器管理模块，在 kpanda-global-cluster 集群页面，在左侧导航栏点击容器网络 ->服务。



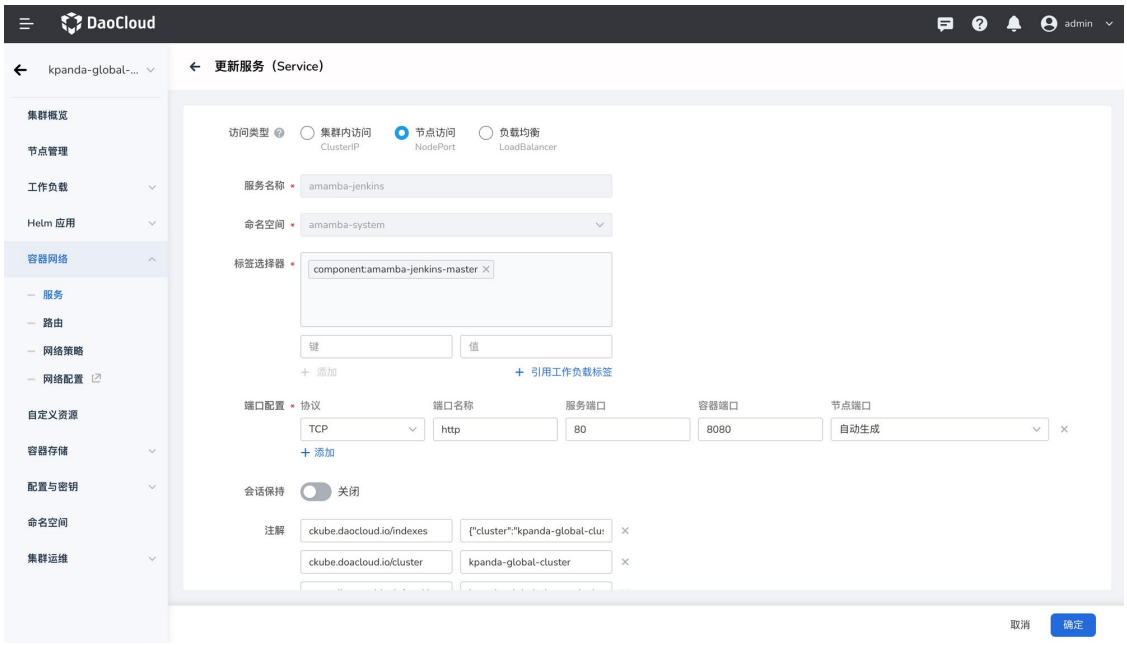
进入服务列表

2. 搜索 amamba–jenkins，在列表选择更新 。



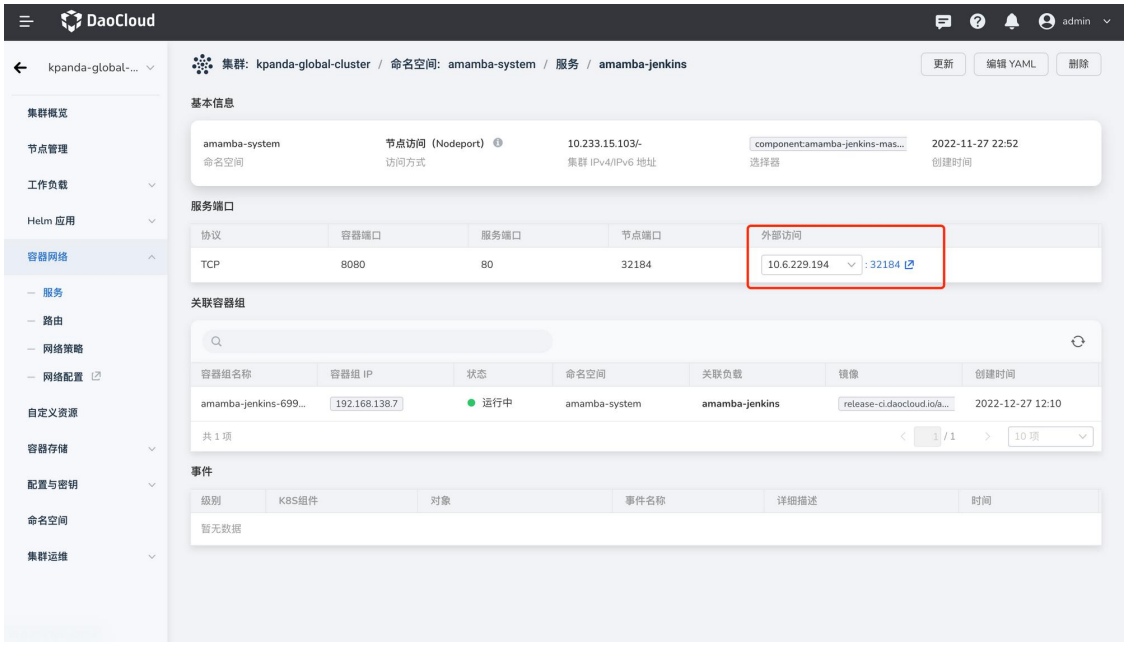
更新 jenkins

3. 将访问类型更改为 NodePort，节点端口选择自动生成即可。



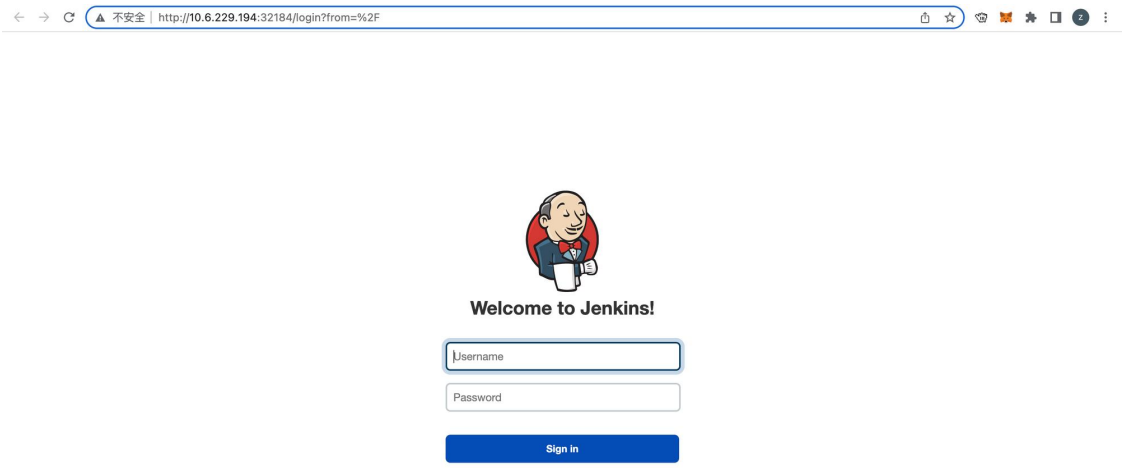
更改访问类型

4. 点击确定，然后返回到详情页面点击链接访问 Jenkins Dashboard。



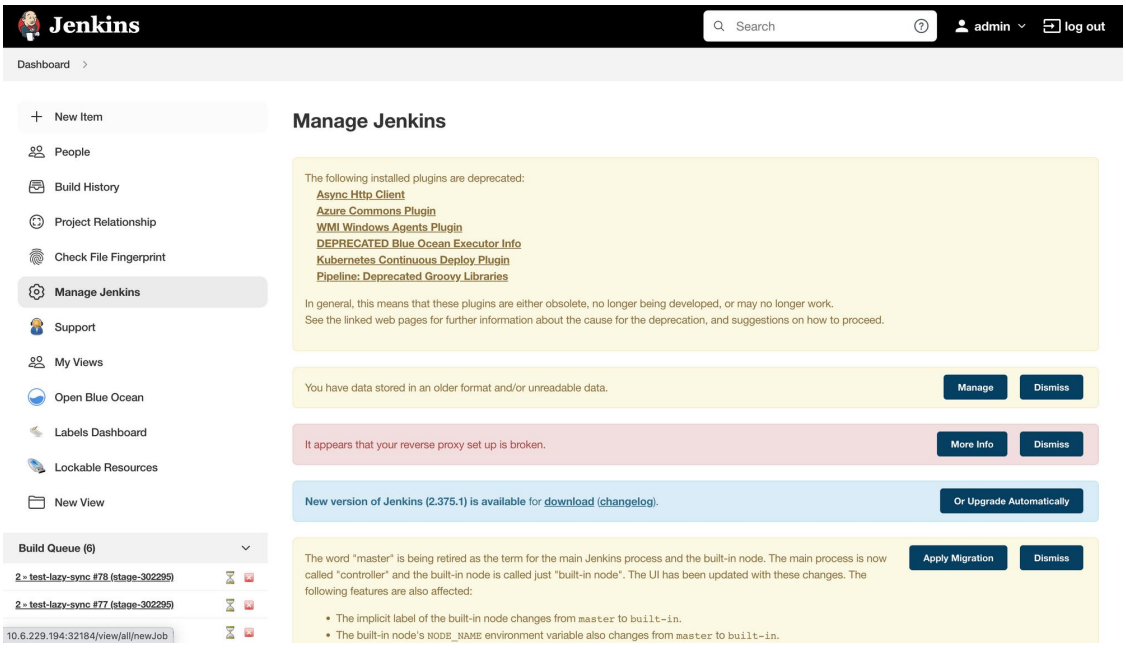
访问 jenkins dashboard

5. 输入账号/密码（默认为 admin/Admin01），进入到 Jenkins Dashboard 页面。



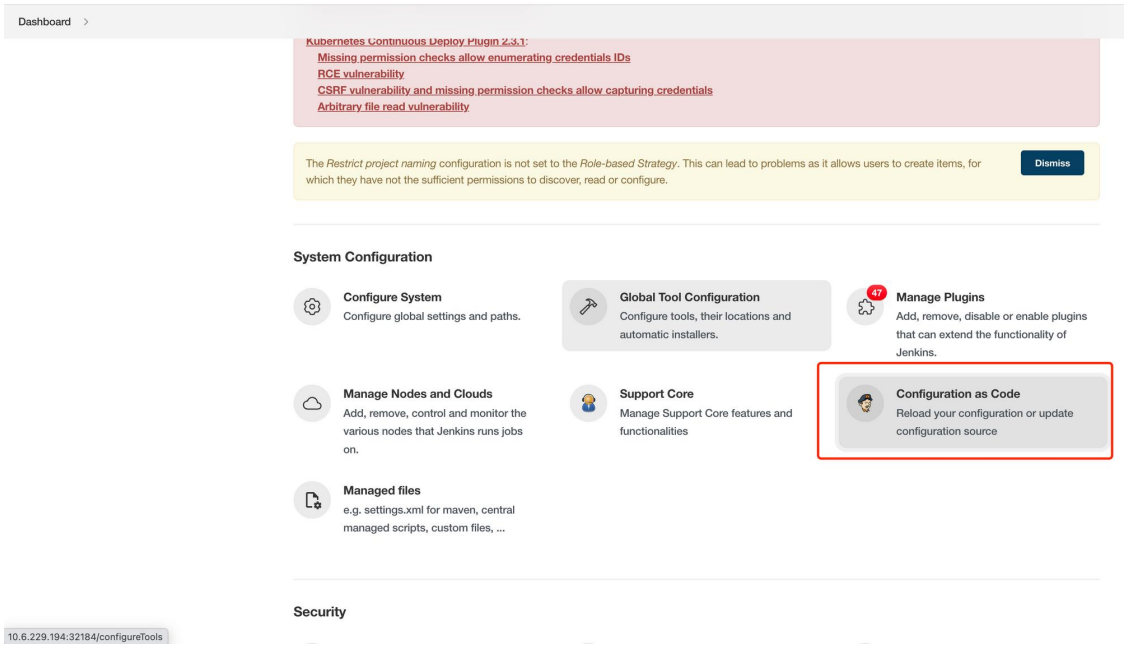
登录

6. 在左侧导航栏选择 Manage Jenkins 。



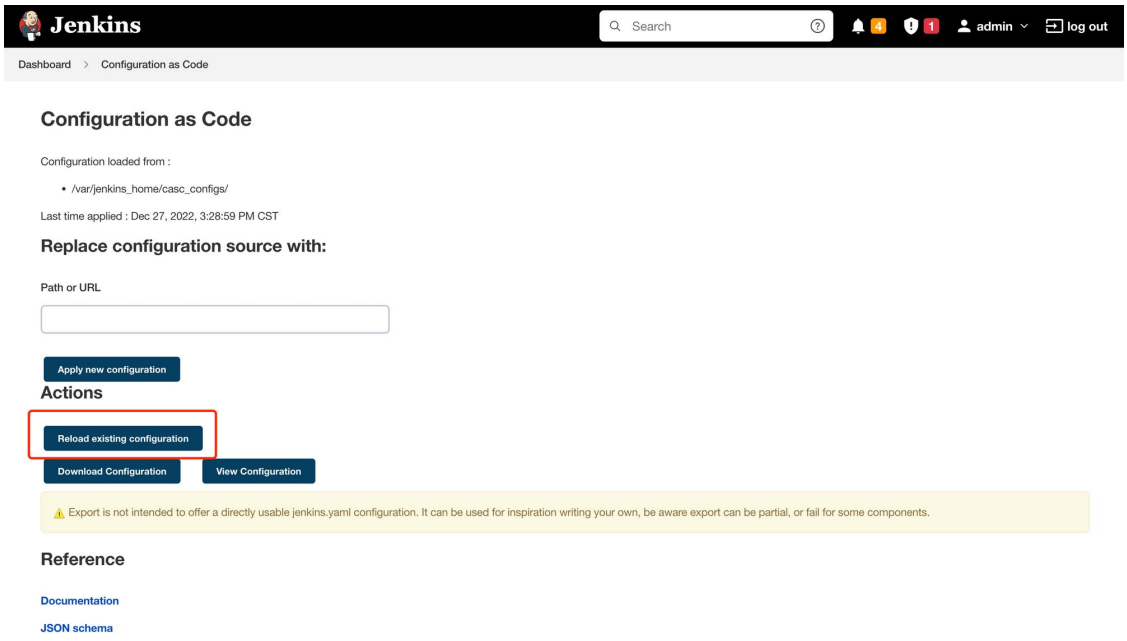
manage jennkins

7. 点击 Configuration as Code 。



configascode

8. 在 Configuration as Code 点击 Reload existing configuration 。如果点击后在当前页面没有任何提示，说明配置加载生效。



reload

运行流水线，检查是否在指定的节点上

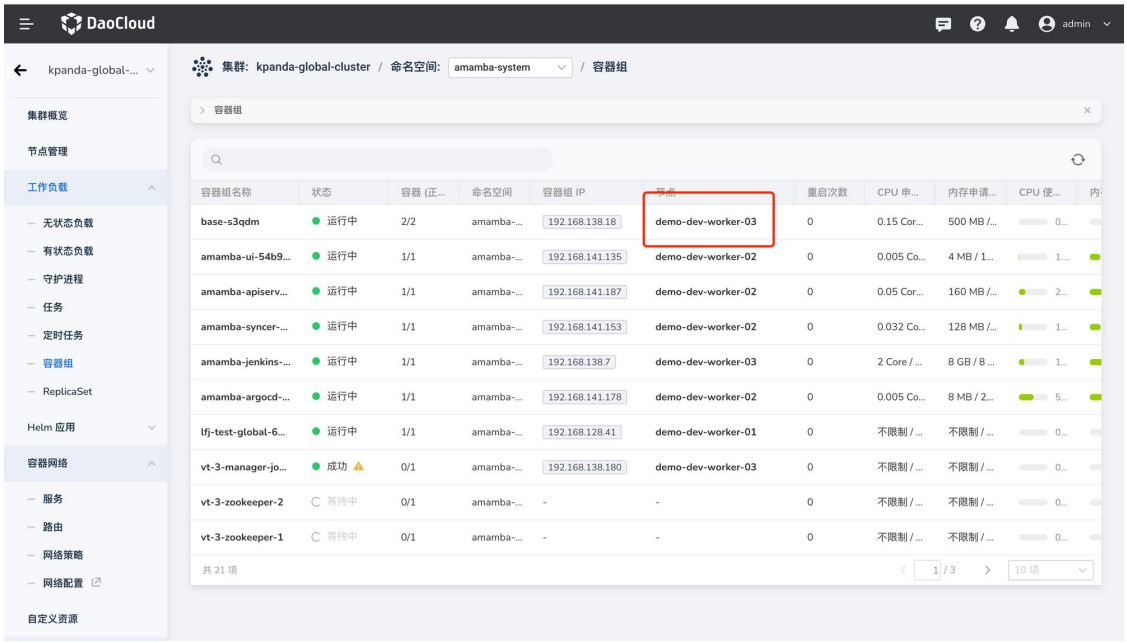
1. 在应用工作台 创建一个流水线任务，并编辑 Jenkinsfile 如下：

```
pipeline {
  agent {
    node {
      label 'base'
    }

  }
  stages {
    stage('Hello') {
      agent none
      steps {
        container('base') {
          echo 'Hello World'
          sh 'sleep 300'
        }

      }
    }

  }
}
```

!!! note

需要注意 agent 部分需要选择 label 为 base。因为在配置文件中只为 base 设置了指定节点，如果需要为其他的 agent 设置。重复上述操作即可。

2. 点击立即执行 该流水线，前往容器管理查看执行该任务的 Pod 运行的节点。



查看 pod 节点

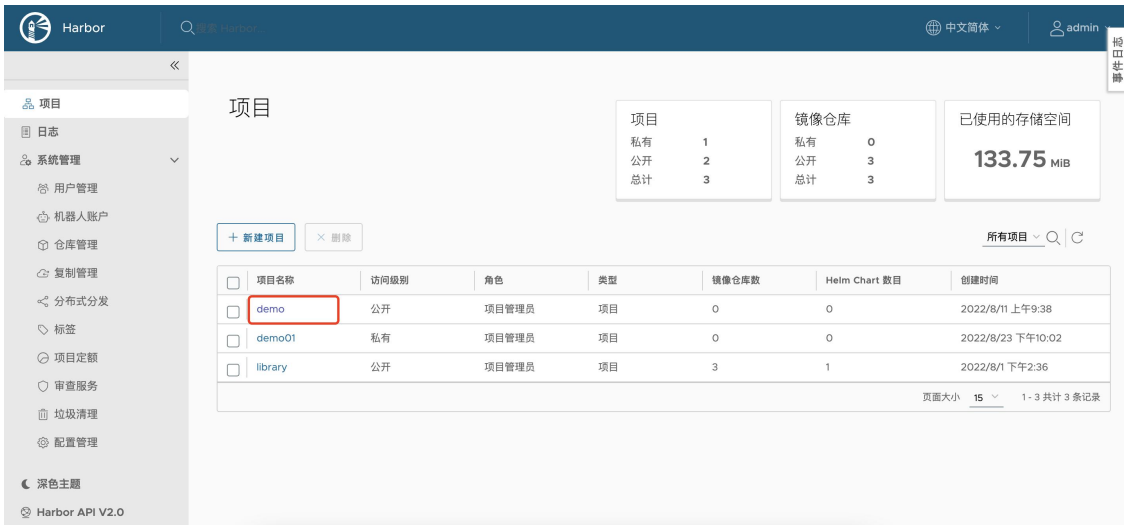
3. 可以看到执行该流水任务的 Pod 运行在了预期的 demo-dev-worker-03 节点上。

集成 Harbor 实现镜像安全扫描

本文将介绍如何在流水线中集成 Harbor 并实现镜像安全扫描。

在 Harbor 中开启自动扫描镜像

1. 登录 Harbor，点击一个具体的项目。



harbor

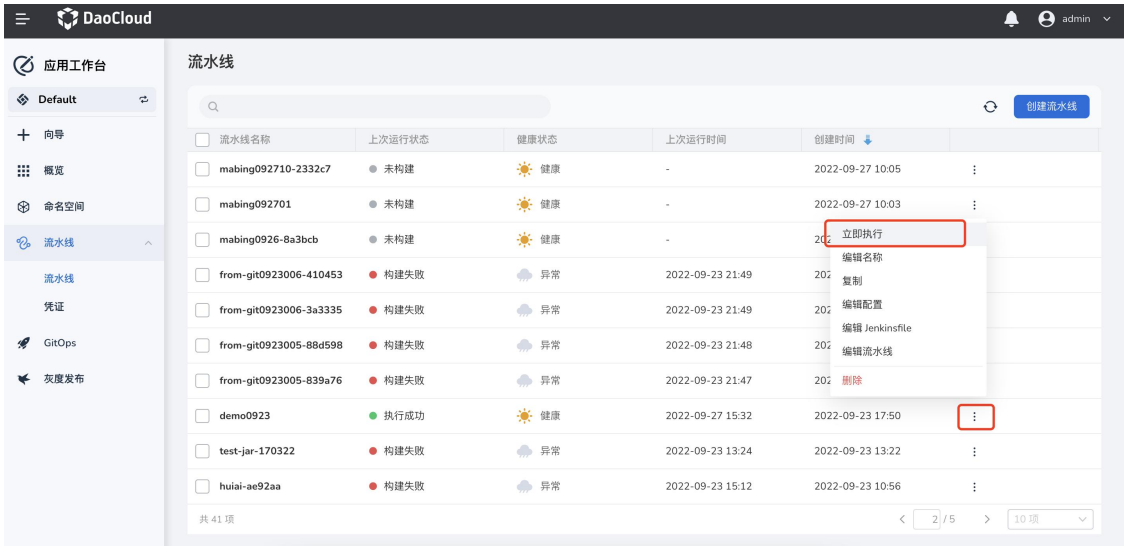
2. 选择配置管理页签，勾选自动扫描镜像。



harbor

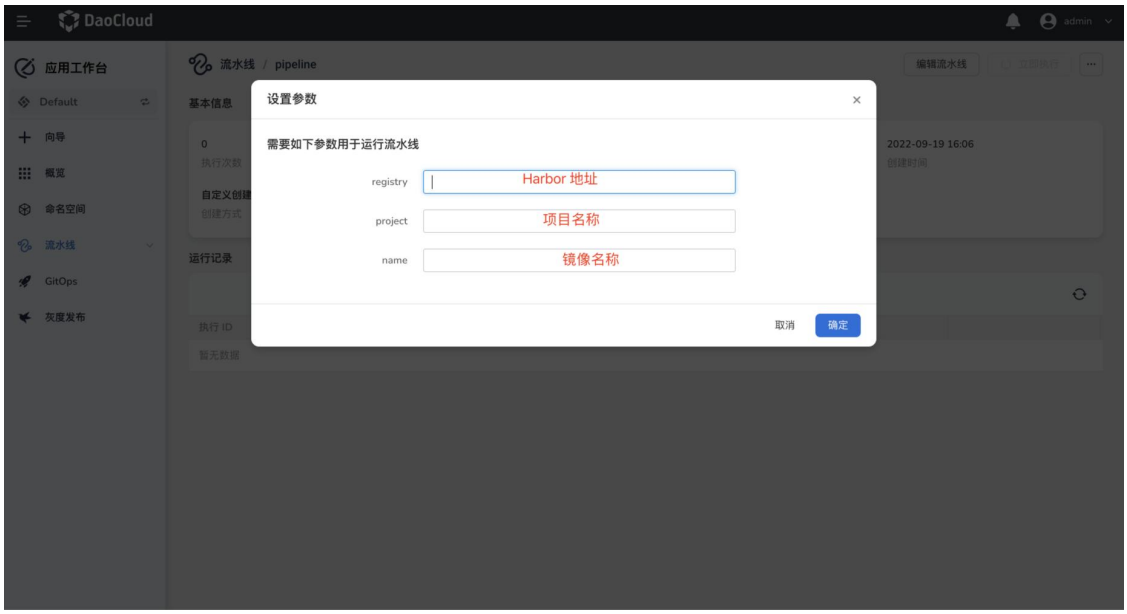
在应用工作台配置流水线

1. 在应用工作台，创建一条流水线，参考快速创建流水线，配置完毕后点击立即执行。



harbor

2. 在弹出的对话框中输入上述 Harbor 配置的项目中的镜像仓库地址。

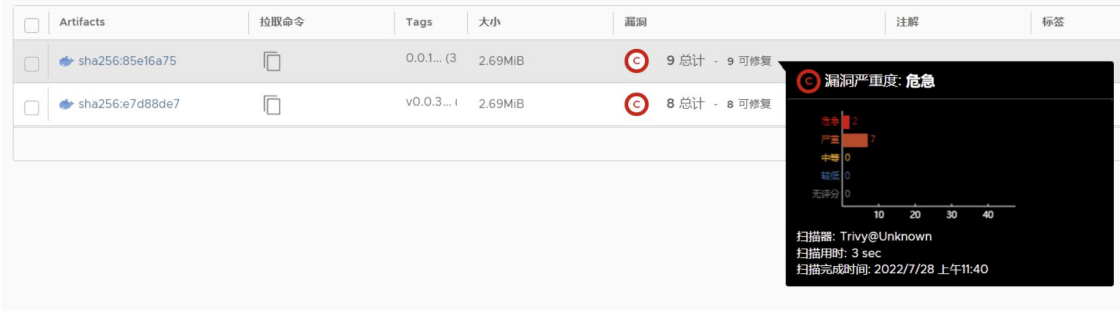


harbor

3. 等待流水线执行成功。

在 Harbor 中查看镜像安全扫描信息

在 Harbor 中依次访问 项目 → 镜像仓库，查看镜像的漏洞信息。



harbor

镜像仓库 Release Notes

本页列出镜像仓库 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-08-02

v0.10

- 新增 输出镜像仓库迁移/备份/恢复方案，已通过迁移 release-ci 仓库验证
- 新增 输出非安全镜像仓库登录最佳实践方案
- 新增 创建托管 Harbor 支持使用内部中间件 MINIO
- 新增 适配人大金仓的 PG 模式
- 优化 创建托管 harbor 时对 pg、redis 地址进行格式校验
- 优化 无权限置灰+提示优化
- 优化 集群解绑后的特殊情况处理

2023-07-02

v0.9

- 新增：创建托管 harbor 支持选择中间件 minio 实例
- 新增：支持 Ghippo 重点审计功能
- 优化：删除工作空间（WS）时校验该 WS 下的镜像仓库资源
- 优化：处理解绑集群后，托管 harbor 的异常状态
- 修复：修复 Ghippo workspace 资源无法解绑的问题
- 修复：修复 Sidecar 注入问题
- 修复：创建 harbor 选择 minio 时，无法推送镜像的问题

2023-06-05

v0.8

新功能

- harbor 类型仓库增加镜像描述信息

- 镜像空间列表增加实例状态
- 创建 harbor 支持随机生成 nodeport 端口号
- 创建 harbor 校验 redis、postgresql、s3 是否重复使用
- project 支持镜像回收功能

修复

- 修复离线部署托管 harbor 镜像地址拼接错误
- 修复 docker registry 类型 project 列表分页问题
- 修复 docker registry 类型 repository 搜索问题
- 修复机器人账号创建时已存在问题

优化

- 添加 job 清除逻辑
- 去除获取临时登录指令接口中 project 字段
- 登录指令优化
- 镜像扫描判断逻辑优化

2023-05-08

v0.7

新功能

- 支持页面配置多实例之间的镜像同步
- 在工作空间中支持关联 jfrog 类型仓库
- 支持在离线 ARM 环境下创建托管 harbor

修复

- 修复仓库集成 URL 修改后不能被更新的问题
- 修复 Project 从原生 Harbor 页面删除后逻辑不兼容情况

优化

- 优化对接 jfrog 接口性能到 2s 内

2023-04-07

v0.6.2

新功能

- 支持全局管理中的自定义角色，对用户授予不同的角色权限
- 支持生成临时登录指令。
- 创建托管 harbor 支持 cpu、memory 校验
- 创建托管 harbor 支持编辑镜像扫描器
- 创建托管 harbor 支持使用 S3 存储

修复

- 解决 harbor-operator helm 安装字节过大的问题

优化

- 优化镜像列表功能页面性能，单独提出 Project 列表页面

2023-03-15

v0.5

新功能

- 支持创建托管 harbor 支持 NodePort 方式暴露，并校验端口是否被占用
- 支持创建托管 harbor 支持 https 方式暴露
- 支持创建托管 harbor 支持开启 DCE 5.0 ODIC 接入，实现使用 DCE 5.0 用户登录 Harboor
- 支持创建托管 harbor 支持校验部署的集群是否安装 harbor-operator
- 支持创建托管 harbor 支持使用中间件模块部署的 redis 实例
- 支持创建托管 harbor 支持修改 Admin 密码、资源配额、Redis 实例、访问类型
- 支持创建托管 harbor 支持自动创建镜像扫描器
- 支持仓库集成，关联仓库校验用户密码以及用户权限
    - 支持仓库集成校验用户名密码正确，并且保证用户具备管理员权限
    - 支持关联仓库校验用户名密码正确

- 支持仓库集成列表新增模糊查询
- 支持页面编辑 Project 为公开或者私有
- 支持 harbor、docker registry 类型的仓库多架构镜像页面显示

修复

- 修复支持关联仓库接入 https 的 harbor、docker registry 仓库

2022-12-30

v0.4

新功能

- 支持基于 Harbor 创建托管镜像仓库
- 支持多副本部署
- 支持对镜像仓库的全生命周期管理
- 支持将托管 harbor 实例部署在平台的任意集群下的任意命名空间
- 支持平台接入和管理外部 Harbor、Docker Registry 镜像仓库
- 支持为平台工作空间（租户）单独分配私有镜像空间
- 支持创建公开/私有镜像空间
- 支持工作空间（租户）独立接入外部 Harbor、Docker Registry 镜像仓库
- 支持部署应用时通过镜像选择器选择镜像
- 支持镜像扫描

镜像仓库 FAQ

本页列出使用镜像仓库时常见的一些问题和解决办法。

DCE5.0 标准版本中不能使用中间件部署

DCE 5.0 标准版本中没有中间件，中间件属于白金版。

如何校验配置的中间件网络是否可连接

登录部署 Harbor 的目标集群，在任意节点中执行 ping 命令，测试是否能连接到中间件组件。

镜像空间列表看不到私有镜像

镜像仓库在 v0.7.0–v0.7.3、v0.8.0 版本系统存在一个 bug，会导致看不到私有镜像。

在使用中间件部署的 Minio 时

在使用中间件部署的 Minio 时，需要先手动通过 Minio 管理平台创建好 bucket。

仓库集成支持的 Harbor 最低版本

在仓库集成时因使用了 Harbor 的功能，对版本有一定要求，目前支持的已知最低版本为：2.4.0。更早的旧版本将不可用。

离线环境镜像扫描器失败

镜像扫描因为依赖漏洞数据，默认是去 CVE 官网获取漏洞数据。 如果是一个纯离线环境，则不能正常进行漏洞扫描，会执行失败。

```
2023-05-08T01:15:39Z [INFO] [/pkg/scan/job.go:387]: {
  "uuid": "c8ad3a89-eb2c-11ed-b4e3-fa9c7bd8892d",
  "name": "Trivy",
  "description": "The Trivy scanner adapter",
  "url": "http://trust-harbor-harbor-harbor-trivy",
  "disabled": false,
  "is_default": true,
  "health": "healthy",
  "auth": "",
  "access_credential": "[HIDDEN]",
  "skip_certVerify": false,
  "use_internal_addr": true,
  "adapter": "Trivy",
  "vendor": "Aqua Security",
  "version": "v0.35.0",
  "create_time": "2023-05-05T10:08:29.624234Z",
  "update_time": "2023-05-05T10:08:29.624238Z"
}
2023-05-08T01:15:39Z [INFO] [/pkg/scan/job.go:387]: {
  "registry": {
    "url": "http://trust-harbor-harbor-harbor-core:80",
    "authorization": "[HIDDEN]"
  },
  "artifact": {
    "namespace_id": 2,
    "repository": "test/dao-2048",
    "tag": "0506-1133",
    "digest": "sha256:09007877da4240967ab2808a6a82b99c45809f9096feldfaa8001490e660b5c0",
    "mime_type": "application/vnd.oci.image.manifest.v1+json"
  }
}
2023-05-08T01:15:39Z [INFO] [/pkg/scan/job.go:167]: Report mime types: [application/vnd.security.vulnerability.report; version=1.1]
2023-05-08T01:15:39Z [INFO] [/pkg/scan/job.go:224]: Get report for mime type: application/vnd.security.vulnerability.report; version=1.1
2023-05-08T01:15:41Z [INFO] [/pkg/scan/job.go:245]: Report with mime type application/vnd.security.vulnerability.report; version=1.1 is not ready yet, retry after 5 seconds
2023-05-08T01:15:46Z [ERROR] [/pkg/scan/job.go:294]: check scan report with mime type application/vnd.security.vulnerability.report; version=1.1: running trivy wrapper: running
trivy: exit status 1: 2023-05-08T01:15:58.339Z   [34mINFO [0m    Need to update DB
2023-05-08T01:15:58.340Z         [34mINFO [0m    DB Repository: ghcr.io/aquasecurity/trivy-db
2023-05-08T01:15:58.340Z         [34mINFO [0m    Downloading DB...
2023-05-08T01:15:58.448Z         [31mFATAL [0m   init error: DB error: failed to download vulnerability DB: OCI artifact error: OCI artifact error: OCI repository error: Get
"https://ghcr.io/v2/": read tcp 10.233.74.85:55994->20.205.243.164:443: read: connection reset by peer
: general response handler: unexpected status code: 500, expected: 200
```

trivy

创建托管 Harbor 时第一步集群校验通过后创建 Harbor 仍然出错

目前只校验了集群中是否有 CRD，没有校验 harbor-operator 服务，可能会出现不存在 harbor-operator 服务的情况，导致不能正确的创建 Harbor。

本地执行 docker login {ip} 之后报错

Error response from daemon: Get "https://{ip}/v2/": x509: cannot validate certificate for {ip} because it doesn't contain any IP SANs

出现这个错误是因为 registry 是 https 服务，是使用了非签名证书或者不安全证书，就会提示这个错误， 解决办法是在 /etc/docker/daemon.json 配置文件中 "insecure-registries" 加入对应的 IP。

```
"insecure-registries": [
  "{ip}",
  "registry-1.docker.io"
]
```

之后重启 systemctl restart docker。

创建托管 harbor 接入外部 PG、Redis，密码含有特殊字符 (!@#$%^&*) 之类的，服务启动失败

目前密码中不能有特殊字符，不然会出现服务启动失败的情况，可以使用大小写字母和数字组合的情况。

Harbor Operator 安装不成功

Harbor Operator 安装不成功需要检查这几点，cert-manager 是否安装成功，installCRDs 是否设置为 true。 安装 Harbor operator 的 helm 任务是否成功。

创建托管 Harbor 可以使用 redis cluster 模式吗

目前 Harbor 仍然不能使用 redis cluster 模式。

私有镜像在非镜像仓库模块能看到吗？

镜像仓库是严格按照 DEC 5.0 的权限来执行的，在镜像仓库中某个用户必须要属于某个租户， 才能看到当前租户下的私有镜像空间，否则即使管理员也不能看到。

私有镜像绑定工作空间后不能查询到

私有镜像绑定工作空间后程序需要异步执行很多逻辑，所以不会马上能看到。 这个过程会受到系统的影响，如果系统响应较快，则异步执行较快，1 分钟内能看到。最长应该不会超过 5 分钟。

托管 Harbor 创建后能访问了但是状态依然不健康

目前托管 Harbor 页面上的状态和仓库集成的状态是二合一的，当两个状态都为健康的时候才是健康， 因此可能出现托管 Harbor 已经可以访问了，但是状态依然不健康，这种情况需要等一个服务探测周期， 一个探测周期是 10 分钟，在一个周期后就会恢复如初。

创建的托管仓库状态为不健康



仓库不健康

- A1：用户输入的数据库、Redis、S3 存储等信息有误，导致无法连接，可通过查看日志文件进行排查。 现象主要是几个核心服务有 Pod 启动失败，可以通过查看日志进一步确认原因。

  kubectl –n kangaroo–lrf04 get pods

  NAME READY STATUS RESTARTS AGE
  trust–node–port–harbor–harbor–chartmuseum–57fdfb9cdc–qznwc 1/1 Running 0 20h
  trust–node–port–harbor–harbor–core–855f8df46c–cgqb9 1/1 Running 0 20h
  trust–node–port–harbor–harbor–jobservice–6b958dbc57–ks997 1/1 Running 0 20h
  trust–node–port–harbor–harbor–portal–5cf6bf659b–kj6gd 1/1 Running 0 20h
  trust–node–port–harbor–harbor–registry–5ccbf457c5–qrtx5 2/2 Running 0 20h
  trust–node–port–harbor–harbor–trivy–dbdc8945–xh6rv 1/1 Running 0 20h
  trust–node–port–nginx–deployment–677c74576–7kmh4 1/1 Running 0 20h

- A2：如果 A1 排查无误，排查 harborcluster 资源是否健康，如下命令查看 harborcluster 资源状态。

  kubectl –n kangaroo–lrf04 get harborclusters.goharbor.io

NAME PUBLIC URL STATUS
trust-node-port https://10.6.232.5:30010 healthy

- A3：如果 A2 排查无误，在 kpanda-global-cluster 集群上排查 registrysecrets.kangaroo.io 资源是否创建，以及 status 情况。

  提示: namespace 默认为 kangaroo-system。

  kubectl -n kangaroo-system get registrysecrets.kangaroo.io

  NAME AGE
  inte-bz-harbor-1 34d

  kubectl -n kangaroo-system describe registrysecrets.kangaroo.io inte-bz-harbor-1

!!! tip

- 上述 A1、A2 都在托管 Harbor 所在的集群上排查问题，目标集群通过如下页面路径查看：`仓库实例` -> `概览` -> `部署位置`
- 上述 A3 在 `kpanda-global-cluster` 集群上验证。

创建 Project 或上传镜像后发现页面上的镜像空间和可用存储未增加

这是因为 UI 页面上在托管 Harbor 首页、仓库集成详情中的统计信息是异步获取的数据，会有一定的延迟，最长延迟为 10 分钟。

仓库集成后但状态为不健康



仓库集成不健康

首先确认实例是否真的健康，如果实例不健康，则需要排查实例的问题； 如果实例健康，则通过在 kpanda-global-cluster 集群上排查 registrysecrets.kangaroo.io 资源是否创建，并排查 status 情况，这样可以初步确认问题所在。

提示：namespace 默认为 kangaroo-system。

```
kubectl -n kangaroo-system get registrysecrets.kangaroo.io

NAME                    AGE
trust-test-xjw          34d
```

```
kubectl -n kangaroo-system get registrysecrets.kangaroo.io trust-test-xjw -o yaml

apiVersion: kangaroo.io/v1alpha1
kind: RegistrySecret
metadata:
  name: trust-test-xjw
  namespace: kangaroo-system
spec:
  ....
status:
  state:
```

```
    lastTransitionTime: "2023-03-29T03:27:31Z"
    message: 'Get "https://harbor.kangaroo.daocloud.io": dial tcp: lookup harbor.
kangaroo.daocloud.io
      on 10.233.0.3:53: no such host'
    reason: RegistryHealthCheckFail
    status: "False"
    type: HealthCheckFail
```

仓库集成后，在镜像列表页面实例中不可查看

请确认仓库集成的资源是否健康，如果不健康是不会在镜像列表页面的实例列表中显示的。 确认方式请参考仓库集成后不健康的确认方法。

在 Kpanda 镜像选择器中选中一个私有 Project 镜像但部署时提示镜像拉取失败

- A1：能在镜像选择器中看到私有 Project 表明 Project 和 Workspace 已经进行了绑定， 此时需要去镜像部署的目标集群 namespace 中确认是否生成名为 registry-secret 的 secret。

  ```
  kubectl -n default get secret registry-secret
  ```

  ```
  NAME                TYPE                             DATA   AGE
  registry-secret     kubernetes.io/dockerconfigjson   1      78d
  ```

- A2：如果确认已经生成名为 registry-secret 的 secret，则需要确认 secret 中的 dockerconfigjson 是否正确。

  ```
  kubectl get secret registry-secret -o jsonpath='{.data.*}'| base64 -d | jq
  ```

  ```
  {
    "auths": {
      "127.0.0.1:5000": {
        "auth": "YWRtaW46SGFyYm9yMTIzNDU="
      }
  ```

```
  }
}
```

```
echo "YWRtaW46SGFyYm9yMTIzNDU=" | base64 -d
```

```
admin:Harbor12345
```

什么是镜像仓库

镜像仓库是一个支持多实例生命周期管理的云原生镜像托管服务，支持将镜像仓库实例部署至任意云原生基础环境，同时支持集成外部镜像仓库（Harbor Registry 和 Docker Registry）。 通过镜像仓库服务，您可以将私有镜像空间分配给一个或多个工作空间（租户）使用，确保私有镜像的安全性，也可以将镜像空间公开给所有 Kubernetes 命名空间使用，镜像仓库配合容器管理服务帮助用户快速部署应用。

功能特性

- 镜像仓库全生命周期管理

  通过托管 Harbor 提供镜像仓库的全生命周期管理，包括镜像仓库的创建、编辑、删除等。

- 租户化应用部署

  支持将镜像空间分配给一个或多个工作空间（租户）使用；支持工作空间（租户）独立关联外部镜像仓库。

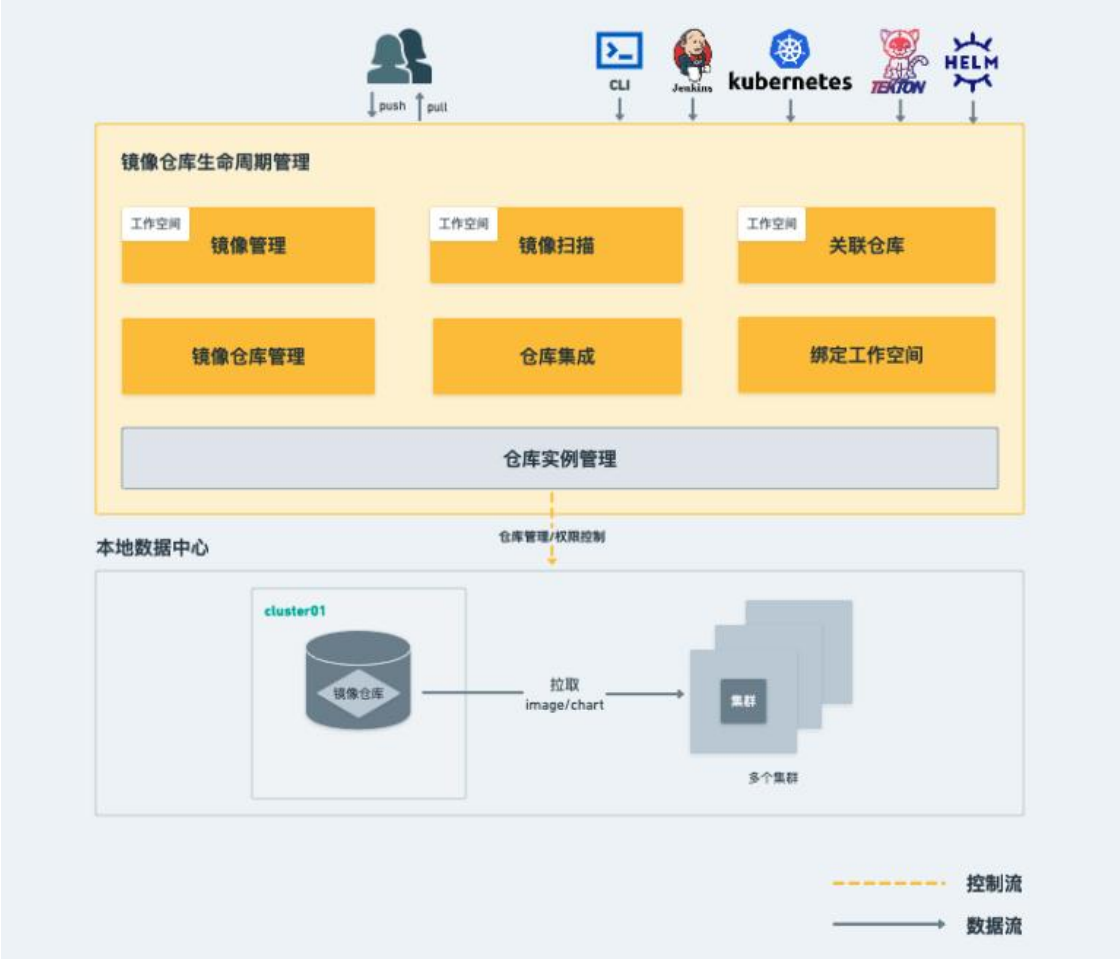
- 镜像扫描

  支持镜像扫描功能，识别镜像安全风险

- 镜像选择

  与容器管理模块联动，通过“选择镜像”功能快速选择镜像，完成应用部署。

产品逻辑架构



逻辑架构图

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

托管 Harbor

Harbor 是一个开源的镜像仓库服务，用于容器镜像、Helm Chart 等符合 OCI 标准的 Artifact 的安全托管及高效分发， 能够帮助您跨云原生计算平台（如 Kubernetes 和 Docker）一致且安全地管理 Artifact。 DCE 5.0 提供了基于 Harbor 的快捷部署能力，并通过与平台中的应用工作台、容器管理模块打通，与工作空间绑定等一系列便捷通道， 实现了一站式高可用、高性能、高效率的部署、管理、使用等全周期的镜像仓库服务。

产品功能

- 支持多副本部署实现高可用
- 支持将平台用户导入原生 Harbor 实例
- 提供原生 Harbor 实例入口，用户在平台 UI 的操作与在原生 Harbor 中的操作实时同步生效
- 支持使用平台自建数据库或接入外部数据库
- 支持使用平台自建 Redis 实例或接入外部 Redis 实例
- 支持指定内部存储或使用外部 S3 兼容对象存储

功能优势

- 多镜像仓库实例，满足开发、测试、生产等多种环境，多套镜像仓库的需求。
- 打破模块之间的调用壁垒，支持在应用工作台和容器管理模块部署应用时快速拉取镜像
- 提供统一的管理控制平面，管理员能够在同一界面对多个 Harbor 实例进行全生命周期管理。

操作步骤

1. 安装 Harbor Operator
2. 创建托管 Harbor 实例

Harbor 迁移指南

本文档将指导您将原始 Harbor 环境迁移至新的集群环境。此迁移案例模拟了一个复杂场景，原始 Harbor 使用 Localpath 存储， 新的环境使用 minio 对象存储，其中涉及到了 Harbor、PostgreSQL 和 Minio 的迁移过程。

环境准备

1. 原始 Harbor 环境：使用 Localpath 作为存储方式的 Harbor 容器环境。
2. 新集群环境：包含新的 PostgreSQL 和 Minio 存储环境。
3. 合理规划存储等资源
4. 待迁移 Harbor ：在迁移过程中，**将原始 Harbor 设置为只读模式**，确保数据的一致性和完整性。

步骤 1：迁移 PostgreSQL

在这一步骤中，我们将迁移原始 Harbor 所使用的 PostgreSQL 数据库至新的集群环境。

工具安装参见：PostgreSQL 官网

1. 使用 pg_dump 工具备份 Harbor 的 PostgreSQL 数据库：

   ```
   pg_dump --username=<原始数据库用户名> --host=<原始数据库主机> --port=<原始数据库端口> --format=plain --file=harbor_backup.sql <数据库名>
   ```

   请将尖括号中的参数替换为实际的值。例如，如果原始数据库用户名为 postgres，原始数据库主机为 locahost，端口为 32332，数据库名为 core，则命令应为：

   ```
   pg_dump --username=postgres --host=locahost --port=32332 --format=plain --file=harbor_backup.sql core
   ```

2. 将数据库备份文件 harbor_backup.sql 从原始 Harbor 环境传输到新的 PostgreSQL 环境。

3. 在新的 PostgreSQL 环境中恢复数据库：

   psql --host=<新数据库主机> --port=<新数据库端口> --username=<新数据库用户名> --dbname=<数据库名> --file=harbor_backup.sql

   请将尖括号中的参数替换为实际的值。例如，如果新数据库主机为 locahost，端口为 32209，新数据库用户名为 postgres，数据库名为 core，则命令应为：

   psql --host=locahost --port=32209 --username=postgres --dbname=core --file=harbor_backup.sql

完成以上步骤后，您已经成功将 Harbor 的 PostgreSQL 数据库迁移至新的集群环境。请确保在整个迁移过程中做好备份，并在迁移前做好充分的测试，以确保数据的安全性和业务的连续性。

步骤 2：迁移镜像数据至 Minio

在这一步骤中，我们将迁移原始 Harbor 存储的镜像数据至 Minio 存储系统。请按照以下步骤逐步进行：

1. 在新的集群环境中下载并安装 rclone 工具。rclone 是一个强大的命令行工具，可用于在不同对象存储之间进行数据同步和复制。 您可以从 rclone 官方网站（https://rclone.org/）获取安装说明。

2. 配置 rclone，设置连接信息以连接到 Minio 存储。打开终端或命令提示符，并执行命令 rclone config，这将引导您完成配置过程。 请根据提示，输入配置名称、Minio 存储类型以及访问密钥等信息。确保您已经正确设置了连接信息，并且可以成功连接到 Minio 存储桶。

3. 查看原始 Harbor 的 registry 服务所挂载的具体目录，这个目录存储着镜像文件。

4. 使用 rclone 进行数据迁移。在终端或命令提示符中执行以下命令

   rclone copy <原始 Harbor 镜像存储目录> <rclone 配置名称>:<Minio 存储桶名称>/harbor/images/

5. 请将尖括号中的参数替换为实际的值。例如，如果 rclone 配置名称为 minio，Minio 存储桶名称为 harbor- images， 原始 Harbor 镜像存储目录为/data/docker/registry/，则命令应为：

   rclone copy /data/docker/registry/ minio:harbor-images/harbor/images/

!!! note

数据迁移过程可能需要较长时间，具体耗时取决于镜像文件的大小和网络速度。请耐心等待数据迁移完成。

步骤 3：创建新的 Harbor 实例

在这一步中，我们将基于新的 PostgreSQL 和 Minio 环境创建一个全新的 Harbor 实例，并确保 Harbor 在新集群环境中正常运行。

1. 在新的集群环境中，使用新的 PostgreSQL 和 Minio 连接信息创建一个全新的 Harbor 实例。 您可以使用镜像仓库组件进行实例创建， 需要注意的是，创建时填写账号密码需要和原始环境中的保持一致，同时需要选择新环境中的数据库实例以及 minio 存储。
2. 确保新的 Harbor 实例已成功安装并配置。
3. 在新的 Harbor 实例中验证镜像数据是否已经成功迁移至 Minio 存储。
4. 确保新的 Harbor 实例在新集群环境中正常运行，并能够访问并提供镜像服务。
5. 测试新的 Harbor 实例，验证迁移的完整性和正确性。您可以尝试上传、下载镜像，并检查日志和事件等信息，确保 Harbor 正常工作。
6. 如果一切运行正常，恭喜！您已经成功完成了 Harbor 迁移至新的集群环境，并且现在的 Harbor 实例正常运行在 Minio 存储系统上。

请注意，在进行实际的迁移和创建新的 Harbor 实例之前，请确保做好充分的备份并在测试环境中测试迁移过程，以确保数据的安全性和业务的连续性。

创建托管 Harbor

前提条件：需要先安装好 Cert Manager 和 Harbor Operator 。

!!! note

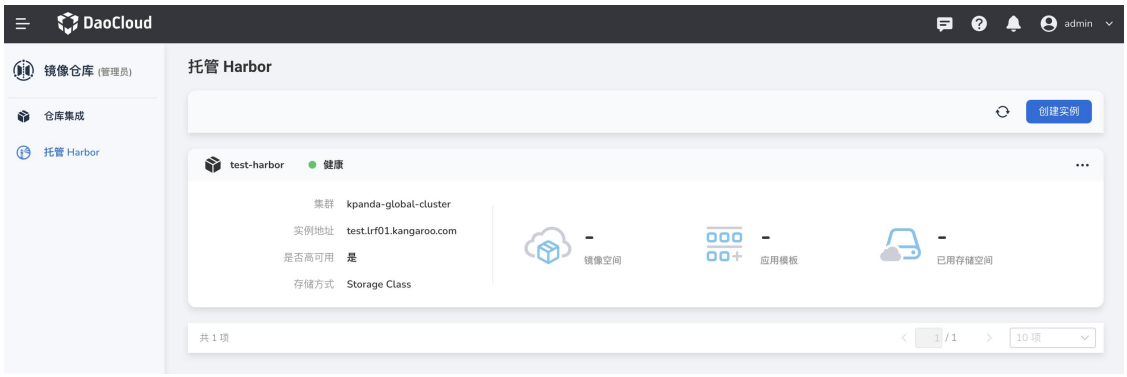
对于 Harbor 实例，除了接入管理员账号外，还可以接入机器人账号达到同样的接入效果。

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击镜像仓库 –> 托管 Harbor。



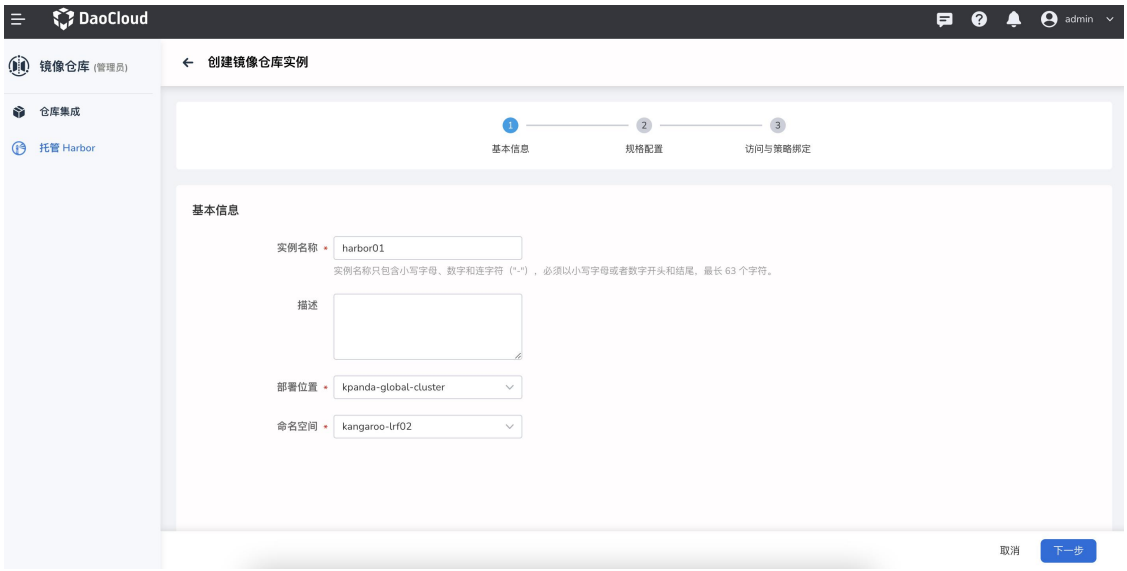
镜像仓库

2. 点击右上角的创建实例按钮。



创建实例

3. 填写实例名称，选择部署位置后点击下一步（若无部署位置可选，需先前往容器管理创建集群和命名空间）。



基本信息

4. 填写数据库、Redis 实例和镜像/Charts 存储信息后点击下一步（当前只支持接入外部数据库和 Redis 实例）。

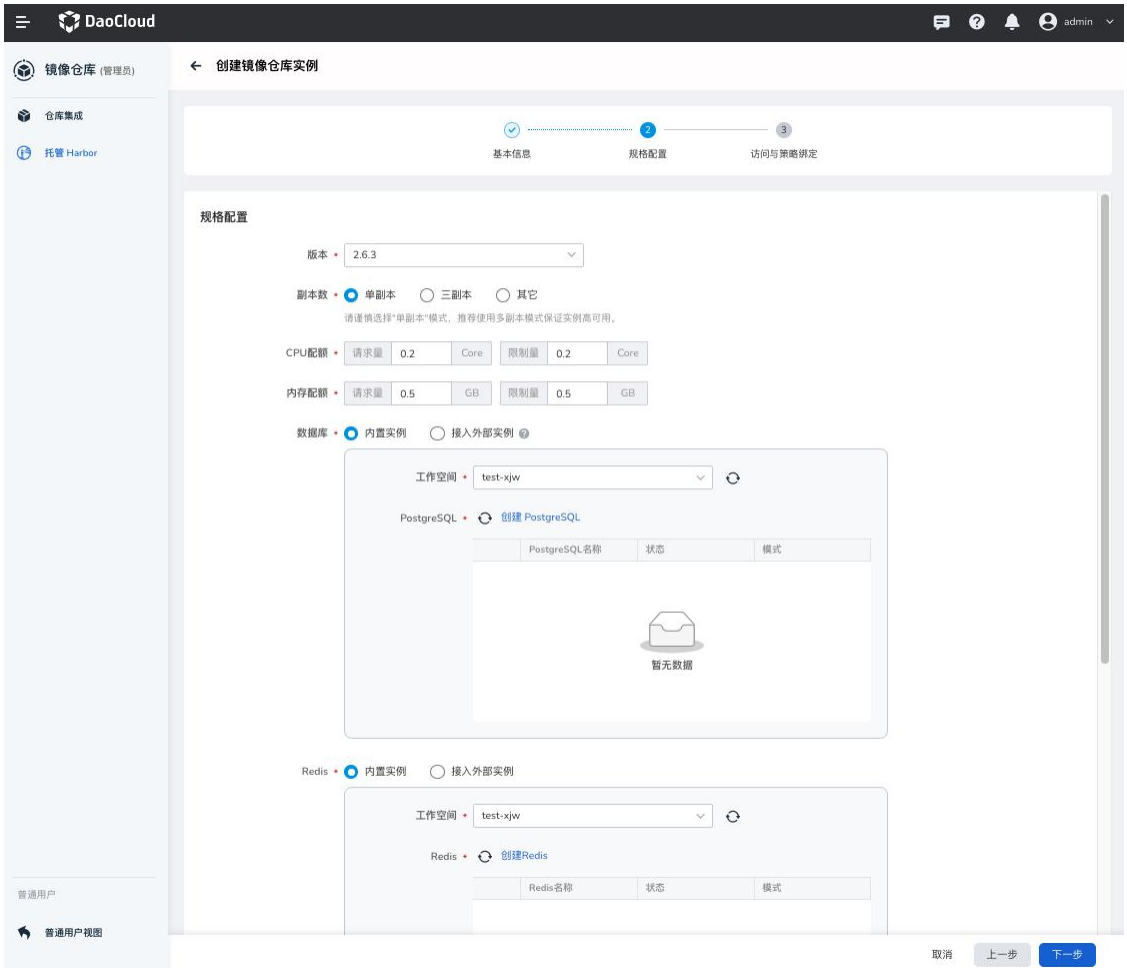
   系统会自动检测 PostgreSQL 和 Redis，如果没有可用的数据库，可以点击创建链接进行创建。

   数据库填写提示：

- 地址：postgres://{host}:{port}，例如 postgres://acid-minimal-cluster.default.svc:5432
- 用户名：填写连接数据库的用户名
- 密码：填写连接数据库的密码

Redis 填写分为单机和哨兵模式：

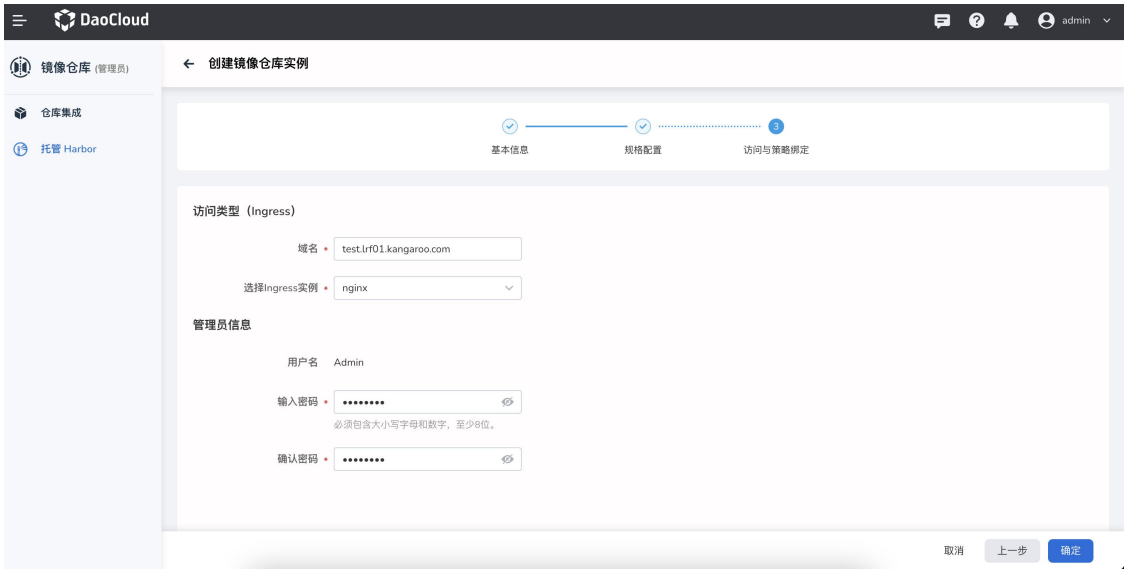
- 单机模式填写地址：redis://{host}:{port}，需要替换 host、port 两个参数。
- 哨兵模式填写地址：redis+sentinel://{host}:{port}?sentinelMasterId={master_id}，需要替换 host、port、master_id 三个参数。
- 密码：按需填写



规格配置

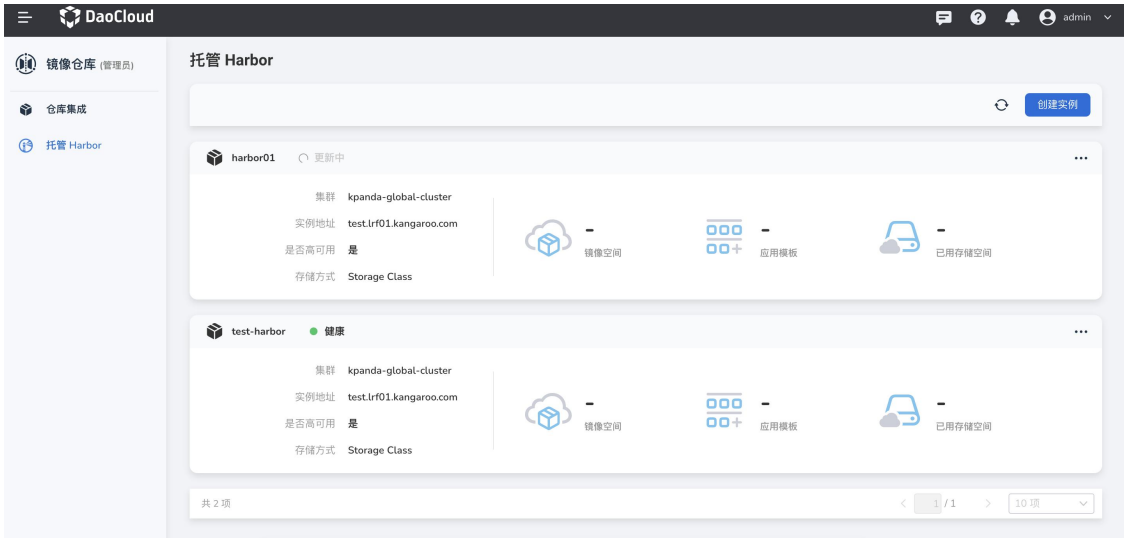
5. 填写域名，选择 Ingress 实例，输入管理员密码后点击确定（用户名/密码用于登录原生 Harbor 实例，请妥善保管密码）。

   域名填写提示: http://{host}，host 前面的 http:// 必须要带上。



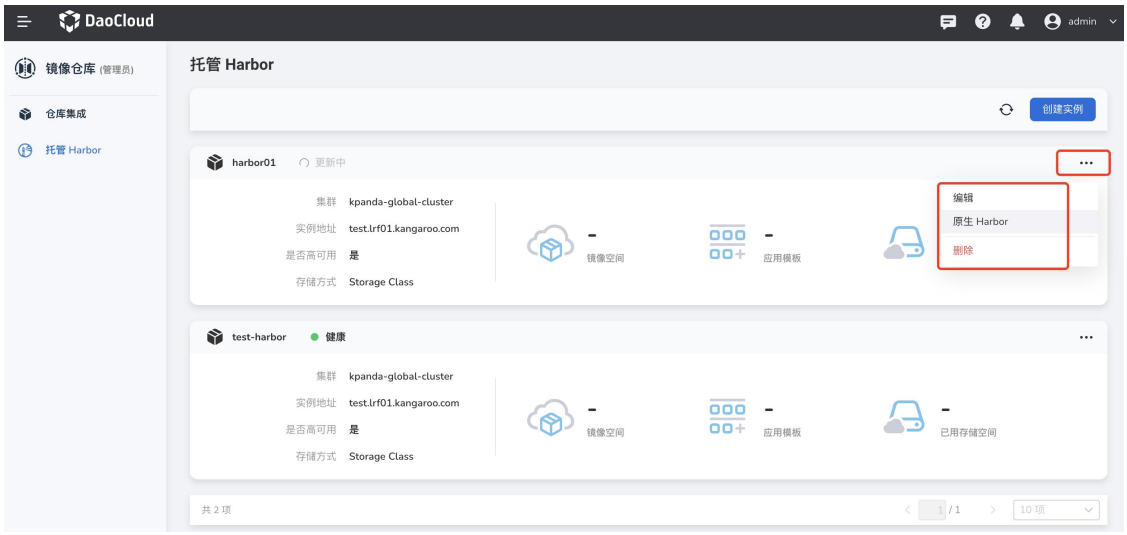
访问与策略绑定

6. 返回托管 Harbor 实例列表，新创建的实例默认位于第一个，等待状态从更新中变为健康，即可正常使用。



实例列表

7. 点击某个实例右侧的 ...，可以选择编辑、删除或进入原生 Harbor。



更多操作

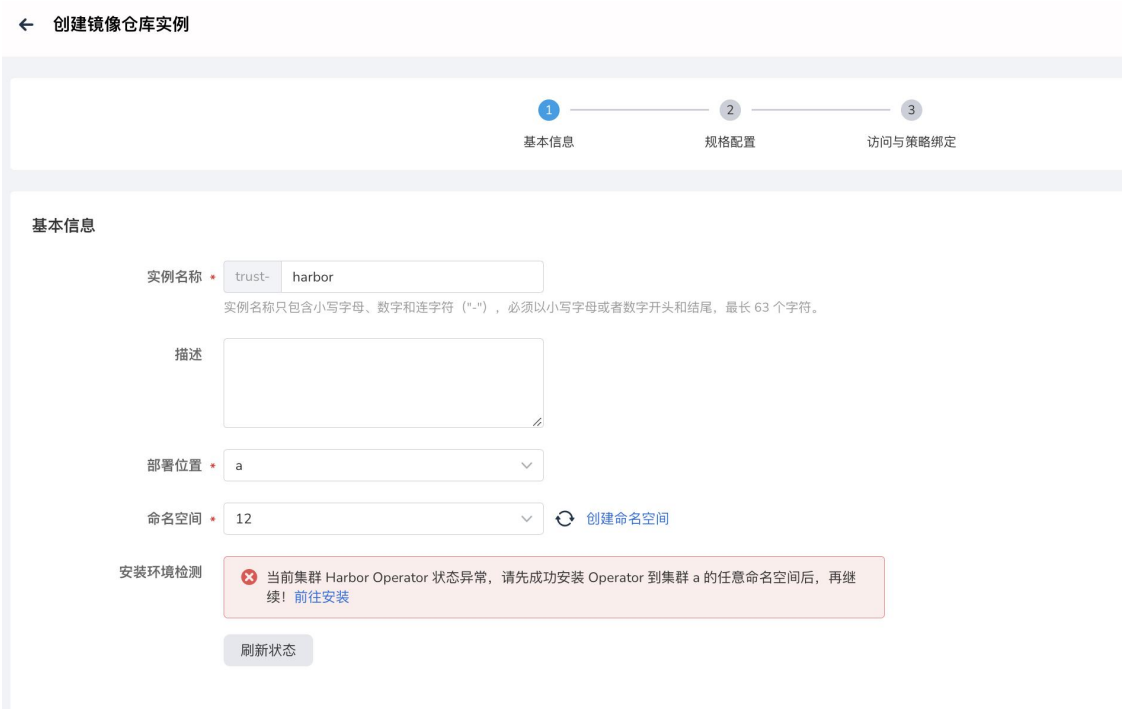
下一步：创建镜像空间

安装 Harbor Operator

托管 Harbor 使用的是 Harbor Operator 技术来进行 Harbor 创建、升级、删除等全生命周期管理。 在创建托管 Harbor 之前，需要先在容器管理中安装 Harbor Operator，版本要求最低 1.4.0。

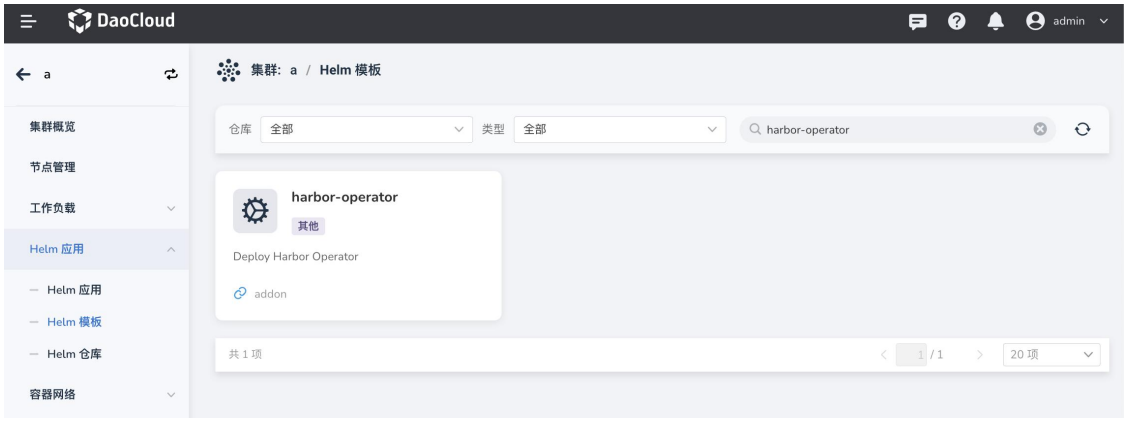
> 提示：Harbor Operator 依赖 Cert Manager，所以需要先安装好 Cert Manager。

1. 如果在创建 Harbor 实例时，出现以下异常提示，请点击前往安装。



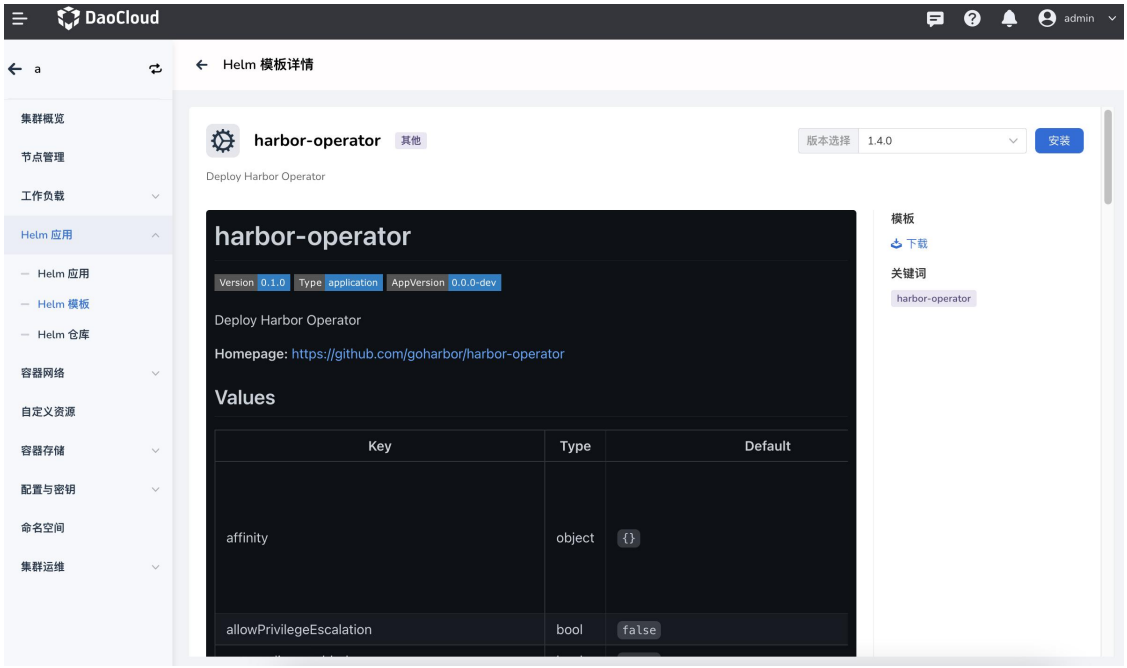
operator 异常

2. 进入容器管理的 Helm 应用 –> Helm 模板，找到并点击 harbor–operator 卡片。



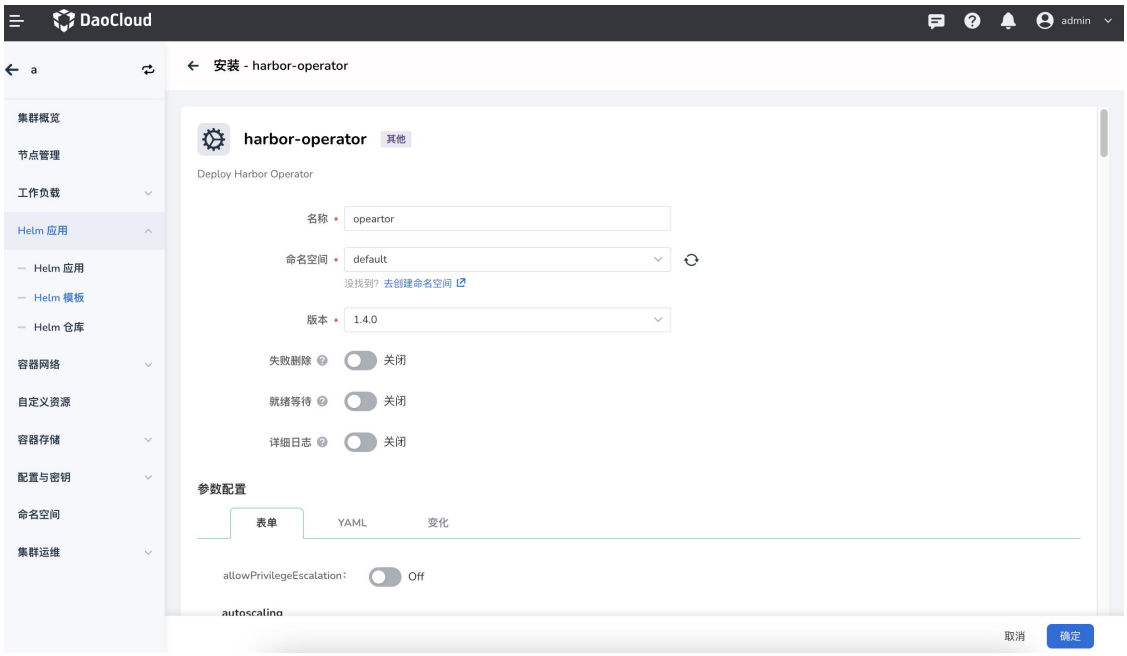
找到 operator

3. 选择版本，点击安装。



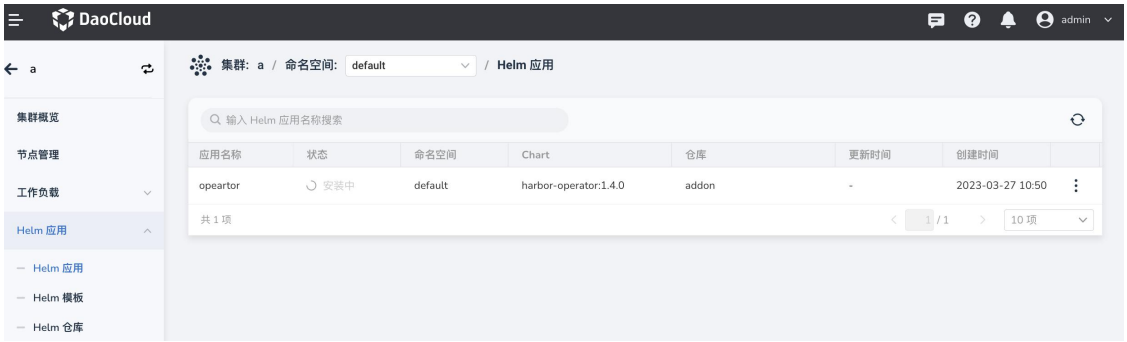
安装

4. 输入名称和命名空间后，点击确定，如果想要添加其他参数，请参考下一节的参数说明。



填表

5. 等待安装完成。



安装

参数值

Harbor Operator 在安装过程中有较多参数可以填写和控制，具体参数请参考如下表格内容：

其中 minio-operator.enabled、postgres-operator.enabled 和 redis-operator.enabled 只能为 false。

Key	Type	Default	Description

Key	Type	Default	Description
affinity	object	{}	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#affinity-v1-core For example: { "nodeAffinity": { "requiredDuringSchedulinglgnoredDuringExecution": { "nodeSelectorTerms": [{ "matchExpressions": [{ "key": "foo.bar.com/role", "operator": "In", "values": ["master"] }] }] } } }
allowPrivilegeEscalation	bool	false	Allow privilege escalation for the controller Pods
autoscaling.enabled	bool	false	Whether to enabled Horizontal Pod Autoscaling
autoscaling.maxReplicas	int	100	Maximum conroller replicas

Key	Type	Default	Description
autoscaling.minReplicas	int	1	Minimum conroller replicas
autoscaling.targetCPUUtilizationPercentage	int	80	CPU usage target for autoscaling
autoscaling.targetMemoryUtilizationPercentage	int	No target	Memory usage target for autoscaling
controllers.chartmuseum.maxReconcile	int	1	Max parallel reconciliation for ChartMuseum controller
controllers.common.classname	string	""	Harbor class handled by the operator. An empty class means watch all resources
controllers.common.networkPolicies	bool	false	Whether the operator should manage network policies
controllers.commo	bool	true	Whether the operator should watch children

Key	Type	Default	Description
n.watchChildren			
controllers.core.maxReconcile	int	1	Max parallel reconciliation for Core controller
controllers.harbor.maxReconcile	int	1	Max parallel reconciliation for Harbor controller
controllers.harborConfiguration.maxReconcile	int	1	Max parallel reconciliation for HarborConfiguration controller
controllers.harborcluster.maxReconcile	int	1	Max parallel reconciliation for HarborCluster controller
controllers.jobservice.maxReconcile	int	1	Max parallel reconciliation for JobService controller
controllers.notaryserver.maxReconcile	int	1	Max parallel reconciliation for NotaryServer controller

Key	Type	Default	Description
controllers.notarysigner.maxReconcile	int	1	Max parallel reconciliation for NotarySigner controller
controllers.portal.maxReconcile	int	1	Max parallel reconciliation for Portal controller
controllers.registry.maxReconcile	int	1	Max parallel reconciliation for Registry controller
controllers.registryctl.maxReconcile	int	1	Max parallel reconciliation for RegistryCtl controller
controllers.trivy.maxReconcile	int	1	Max parallel reconciliation for Trivy controller
deploymentAnnotations	object	{}	Additional annotations to add to the controller Deployment
fullnameOverride	string	""	
harborClass	string	""	Class name of the Harbor operator

Key	Type	Default	Description
image.pullPolicy	string	"IfNotPresent"	The image pull policy for the controller.
image.registry	string	"docker.io"	The image registry whose default is docker.io.
image.repository	string	"goharbor/harbor-operator"	The container registry whose default is the chart appVersion.
image.tag	string	"dev_master"	The image tag whose default is the chart appVersion.
imagePullSecrets	list	[]	Reference to one or more secrets to be used when pulling images https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/ For example: [{"name":"image-pull-secret"}]
installCRDs	bool	false	If true, CRD resources will be installed as part of the Helm chart. If enabled, when uninstalling CRD resources will be deleted causing all installed custom resources to be DELETED
leaderElection.nam	string	"kube-system"	The namespace used to store the ConfigMap for

Key	Type	Default	Description
espace			leader election
logLevel	int	4	Set the verbosity of controller. Range of 0 – 6 with 6 being the most verbose. Info level is 4.
minio-operator.enabled	bool	false	Whether to enabled MinIO Operator
nameOverride	string	""	
nodeSelector	object	{}	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#nodeselector-v1-core For example: [{ "matchExpressions": [{ "key": "kubernetes.io/e2e-az-name", "operator": "In", "values": ["e2e-az1", "e2e-az2"] }] }]
podAnnotations	object	{}	Additional annotations to add to the controller Pods
podLabel	object	{}	Additional labels to add to

Keys	Type	Default	Description
			the controller Pods
podSecurityContext	object	{"runAsNonRoot":true,"runAsUser":65532}	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#podsecuritycontext-v1-core For example: { "fsGroup": 2000, "runAsUser": 1000, "runAsNonRoot": true }
postgres-operator.configKubernetes.secret_name_template	string	"{username}.{cluster}.credentials"	
postgres-operator.enabled	bool	false	Whether to enabled Postgres operator
priorityClassName	string	""	priority class to be used for the harbor-operator pods
rbac.create	bool	true	Whether to install Role Based Access Control
redis-operator.	bool	false	Whether to enabled Redis Operator

Key	Type	Default	Description
enabled			
redis-operator.image.tag	string	"v1.2.0"	
replicaCount	int	1	Number of replicas for the controller
resources	object	{"limits":{"cpu":"500m","memory":"300Mi"},"requests":{"cpu":"300m","memory":"200Mi"}}	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#resourcerequirements-v1-core
service.port	int	443	Expose port for WebHook controller
service.type	string	"ClusterIP"	Service type to use
serviceAccount.annotations	object	{}	Annotations to add to the service account
serviceAccount.create	bool	true	Specifies whether a service account should be created
serviceAccount.name	string	""	The name of the service account to use. If not set and create is true, a name is generated using the fullname

Key	Type	Default	Description
			template
strategy	object	{}	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#deploymentstrategy-v1-apps For example: { "type": "RollingUpdate", "rollingUpdate": { "maxSurge": 0, "maxUnavailable": 1 } }
tolerations	list	[]	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#toleration-v1-core For example: [{ "key": "foo.bar.com/role", "operator": "Equal", "value": "master", "effect": "NoSchedule" }]
volumeMounts	list	[]	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-

Key	Type	Default	Description
			api/v1.18/#volumemount-v1-core For example: [{ "mountPath": "/test-ebs", "name": "test-volume" }]
volumes	list	[]	Expects input structure as per specification https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volume-v1-core For example: [{ "name": "test-volume", "awsElasticBlockStore": { "volumeID": "<volume-id>", "fsType": "ext4" } }]

下一步：创建托管 Harbor 实例

离线升级镜像仓库模块

本页说明从下载中心下载镜像仓库模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `kangaroo` 字样是镜像仓库模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

    !!! note

    该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

    ===“已安装 chart repo”

    若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

    ```yaml title="load-image.yaml"
    source:
      intermediateBundlesPath: kangaroo-offline # (1)
    target:
      containerRegistry: 10.16.10.111 # (2)
      containerRepository: release.daocloud.io/kangaroo # (3)
      repo:
        kind: HARBOR # (4)
    ```

```
    url: http://10.16.10.111/chartrepo/release.daocloud.io # (5)
    auth:
    username: "admin" # (6)
    password: "Harbor12345" # (7)
  containers:
    auth:
      username: "admin" # (8)
      password: "Harbor12345" # (9)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

“未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kangaroo-offline # (1)
target:
  containerRegistry: 10.16.10.111 # (2)
  containerRepository: release.daocloud.io/kangaroo # (3)
  repo:
```

```
      kind: LOCAL
      path: ./local-repo  # (4)
  containers:
    auth:
      username: "admin"  # (5)
      password: "Harbor12345"  # (6)
```

1. 到执行 charts-syncer 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径
2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar xvf kangaroo.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```shell
docker load -i images.tar
```

=== “containerd”

```shell
ctr image import images.tar
```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

=== “通过 helm repo 升级”

1. 检查全局管理 helm 仓库是否存在。

    ```shell
    helm repo list | grep kangaroo
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加全局管理的 helm 仓库。

```shell
heml repo add kangaroo http://{harbor url}/chartrepo/{project}
```

1. 更新全局管理的 helm 仓库。

```shell
helm repo update kangaroo # (1)
```

1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

1. 选择您想安装的全局管理版本（建议安装最新版本）。

```shell
helm search repo kangaroo/kangaroo --versions
```

```none
[root@master ~]# helm search repo kangaroo/kangaroo --versions
NAME                    CHART VERSION   APP VERSION   DESCRIPTION
kangaroo/kangaroo   0.9.0           v0.9.0        A Helm chart for Kangaroo
...
```

1. 备份 `--set` 参数。

在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
```

```
helm get values kangaroo -n kangaroo-system -o yaml > bak.yaml
```

1. 查看版本更新记录，如果 CRD 有更新，更新 kangaroo crds

```shell
helm pull kangaroo/kangaroo --version 0.9.0 && tar -zxf kangaroo-0.9.0.tgz
kubectl apply -f kangaroo/crds
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kangaroo kangaroo/kangaroo \
  -n kangaroo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
  --version 0.9.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

在升级全局管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values kangaroo -n kangaroo-system -o yaml > bak.yaml
```

1. 查看版本更新记录，如果 CRD 有更新，更新 kangaroo crds

```shell
kubectl apply -f ./crds
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kangaroo . \
  -n kangaroo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

仓库集成(管理员)

仓库集成(管理员)是集中管理平台镜像仓库的入口，既支持集成外部镜像仓库，如 Harbor Registry、Docker Registry； 又能够自动集成平台创建的托管 Harbor。仓库集成后，平台管理员可以通过将镜像空间与工作空间绑定的方式，将某个私有镜像空间分配给一个或者多个工作空间（工作空间下的命名空间）。 或者将镜像空间设置为公开，供平台所有命名空间使用。

主要功能

- 支持集成主流的镜像仓库，如 Harbor Registry、Docker Registry，帮助您集中管理平台级别的镜像仓库。
- 支持通过概览页，快速查看仓库地址、镜像空间数、存储使用量等数据。
- 支持创建并设置镜像空间状态为公开或私有。若镜像空间状态为公开，其下的镜像能够被平台所有命名空间使用。 若镜像空间状态为私有，则将镜像空间与一个或多个工作空间绑定后，只有被绑定的工作空间下的命名空间才能够使用该私有镜像，保证私有镜像的安全性。
- 自动集成托管 Harbor，平台创建托管 Harbor 实例后将被自动集成到集成仓库列表，进行统一的管理。

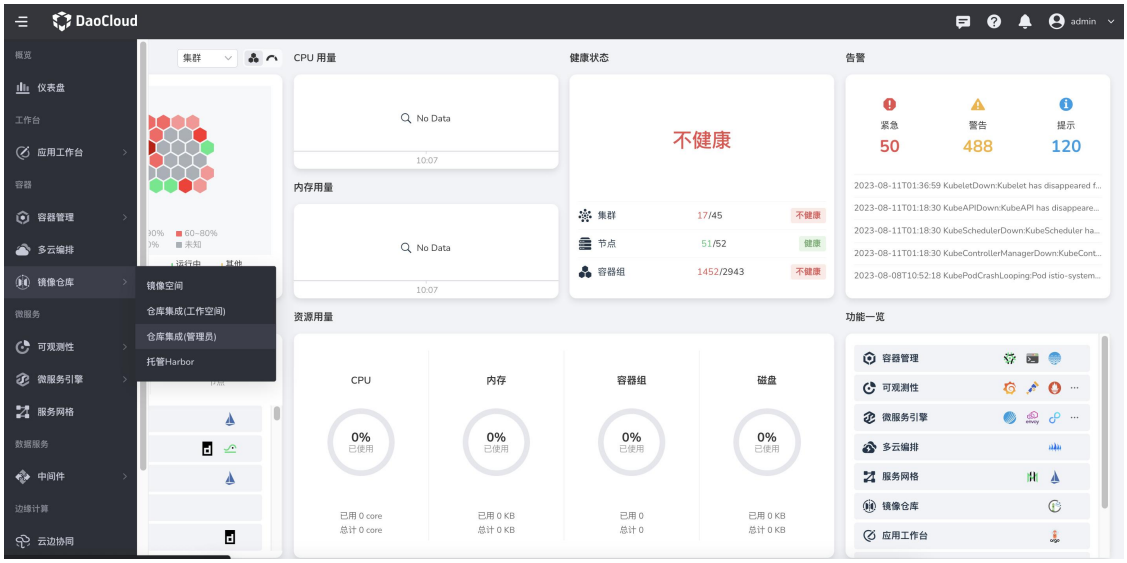
功能优势

- 统一管理入口，对集成镜像仓库和托管 Harbor 实例进行统一管理。
- 高安全性，私有镜像只能通过镜像空间与工作空间绑定的方式才能在部署应用时被拉到。
- 方便快捷，镜像空间一旦被设置为公开，则平台范围内的所有命名空间在部署应用时都能够拉取其下的公开镜像。
- 支持主流镜像仓库类型：Harbor Registry、Docker Registry。

操作步骤

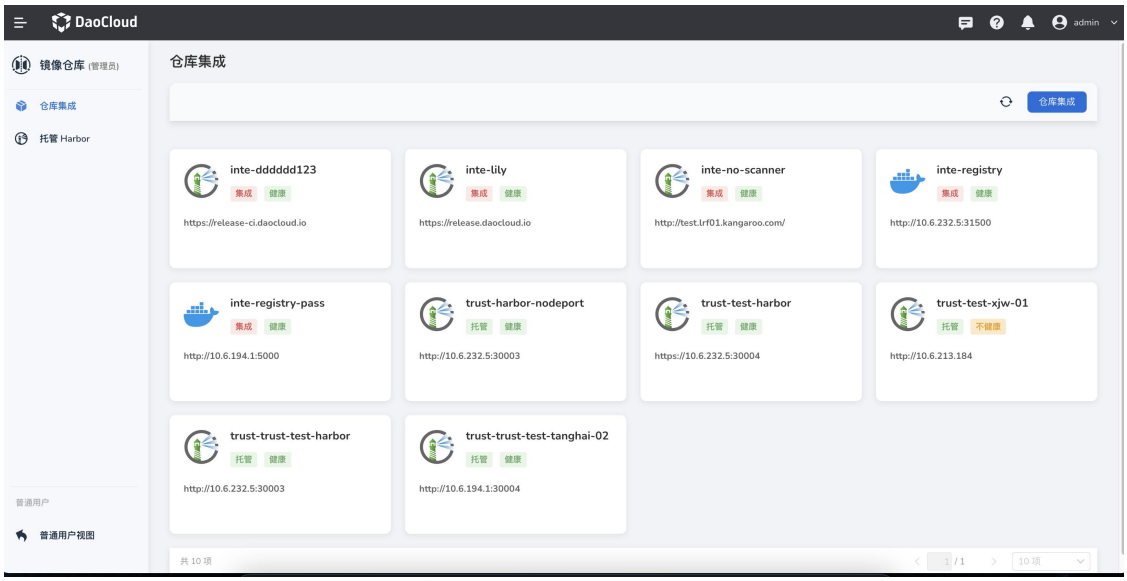
可以参阅视频教程熟悉以下操作步骤：

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击镜像仓库 –> 仓库集成(工作空间)。



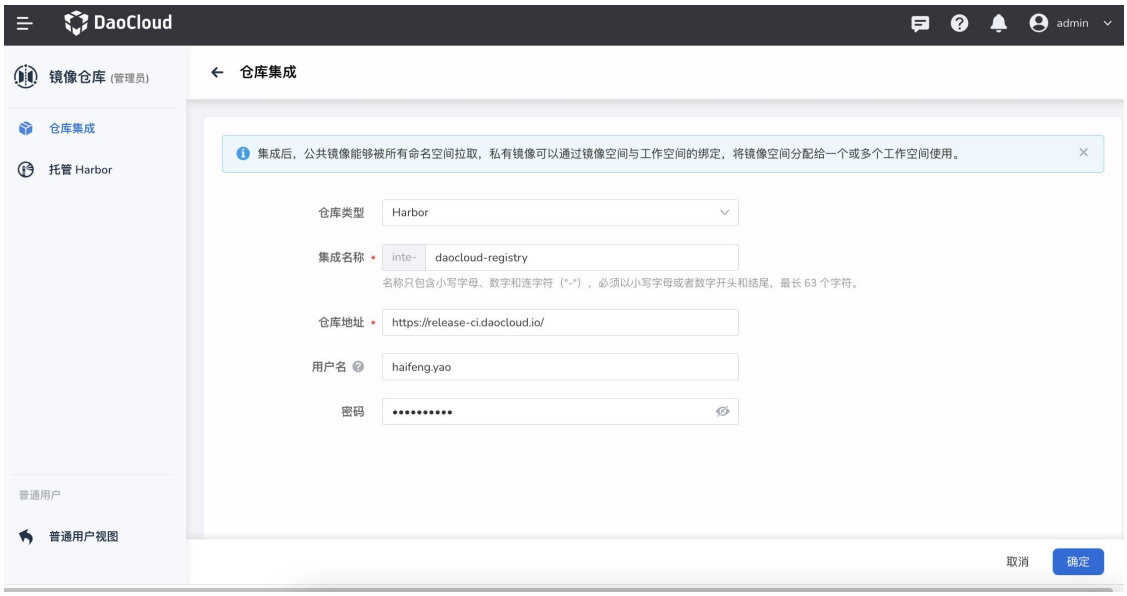
仓库集成

2. 点击右上角的仓库集成按钮。



点击按钮

3. 选择仓库类型，填写集成名称、仓库地址、用户名和密码，点击确定。

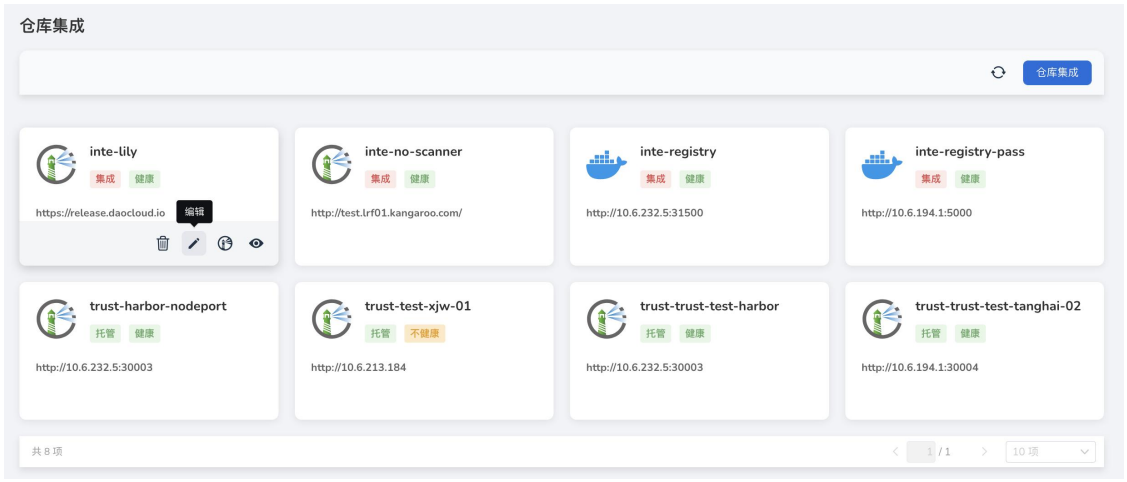


填写参数

!!! note

对于 Harbor 仓库，必须提供 Admin 级别的用户名/密码。

4. 返回仓库集成列表。集成的仓库将带有集成、健康或不健康等标签。光标悬浮到某个磁贴上，可以执行删除集成、编辑等操作。



更多操作

下一步：创建镜像空间

仓库集成(工作空间)

仓库集成(工作空间)是针对工作空间提供的便捷接入仓库功能。 工作空间管理员（Workspace Admin）可根据需要为该工作空间灵活接入镜像仓库，供工作空间成员使用。 接入后，成员在工作空间下的命名空间部署应用时可通过选择镜像按钮一键拉取工作空间下所有公开和私有的镜像，实现快速部署应用。

支持集成 3 种仓库：

- Harbor Registry：这是一个开源的企业级 Docker 镜像仓库，提供镜像存储、版本控制、访问控制和安全扫描等功能。 它专注于为企业环境提供高度可定制和安全的容器镜像管理解决方案。 Harbor Registry 支持跨多个容器编排平台进行集成，并具有丰富的权限管理和审计功能。
- Docker Registry：这是 Docker 官方提供的镜像仓库服务。它作为 Docker 生态系统的一部分， 用于存储、分发和管理 Docker 镜像。 Docker Registry 提供了基本的镜像存储和版本控制功能， 可以通过简单的 API 进行操作。
- JFrog Artifactory：这是一个通用的软件包管理和分发平台，支持各种不同的包类型，包括 Docker 镜像。 除了作为 Docker 镜像仓库外，Artifactory 还支持其他软件包格式（如 Maven、npm 等）的存储、分发和管理。 Artifactory 提供强大的权限控制、缓存和快速复制等功能，同时具备高度可扩展性和可定制性。 与上述两个镜像仓库相比，Artifactory 是一个较为全面的软件包管理平台，适用于跨多种包类型的工作负载。

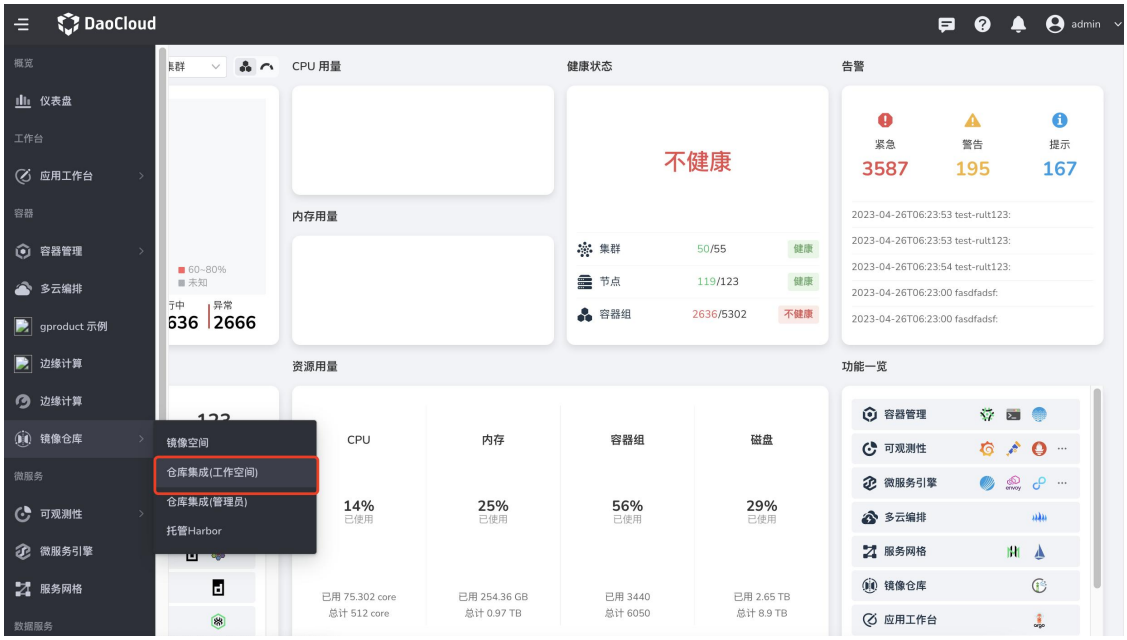
优势

- 灵活便捷：工作空间的管理员均可自主接入一个或多个 Harbor / Docker 类型的镜像仓库，供工作空间成员使用。
- 全局联动：接入后，在应用工作台部署应用时，可通过选择镜像按钮，一键选择仓库中的镜像，实现快速部署应用。

操作步骤

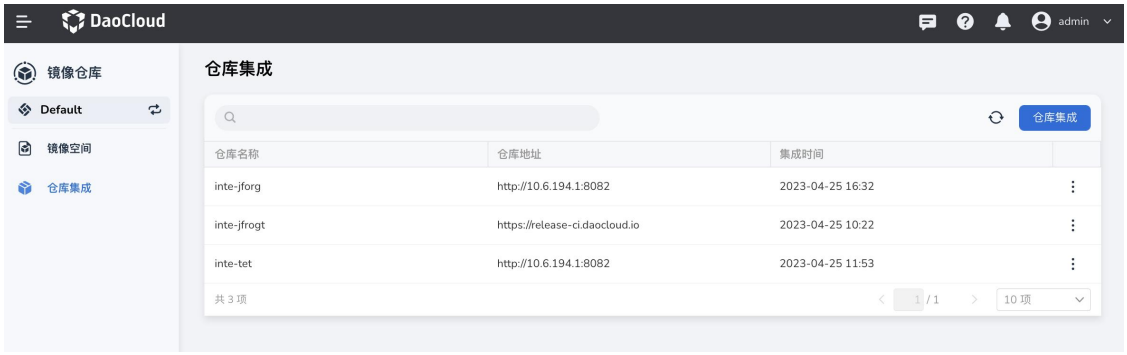
可以参阅视频教程熟悉以下操作步骤：

1. 使用具有 Workspace Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击镜像仓库 –> 仓库集成(工作空间)。



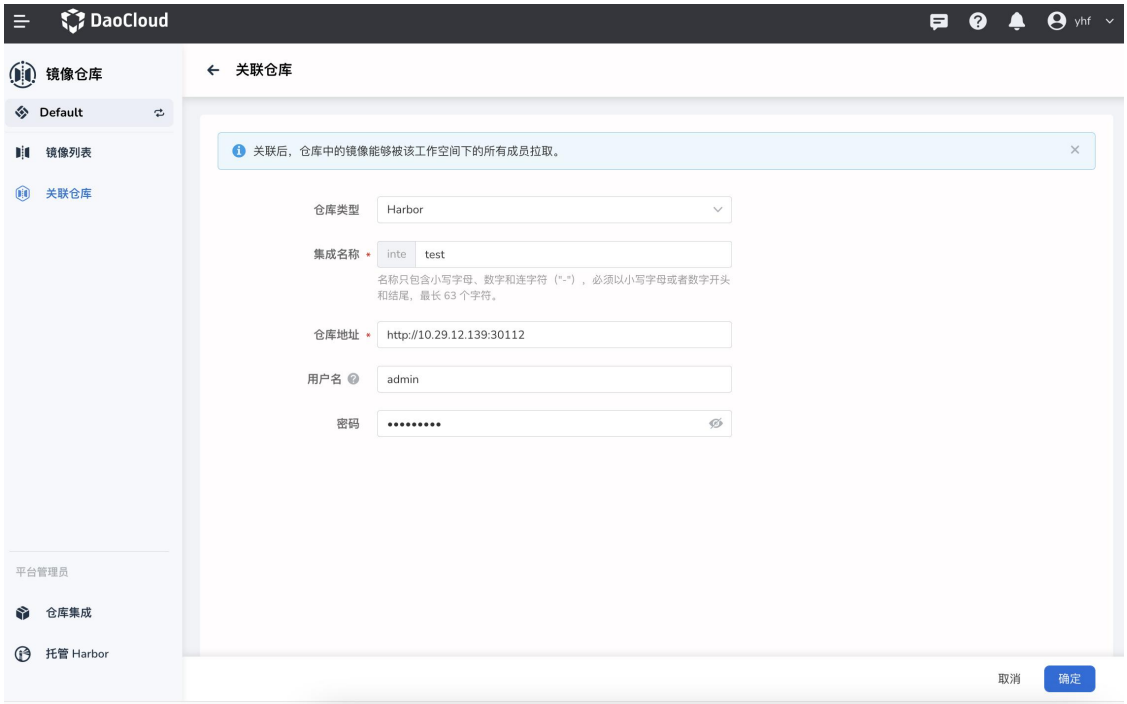
镜像仓库

2. 点击右上角的仓库集成按钮。



仓库集成

3. 填写表单信息后点击确定。



填写表单

!!! note

1. Docker Registry 镜像仓库若未设置密码可不填写，Harbor 仓库必须填写用户名/密码。
1. 有关实际操作演示，请参阅[仓库集成(工作空间)视频演示](../../videos/kangaroo.md)

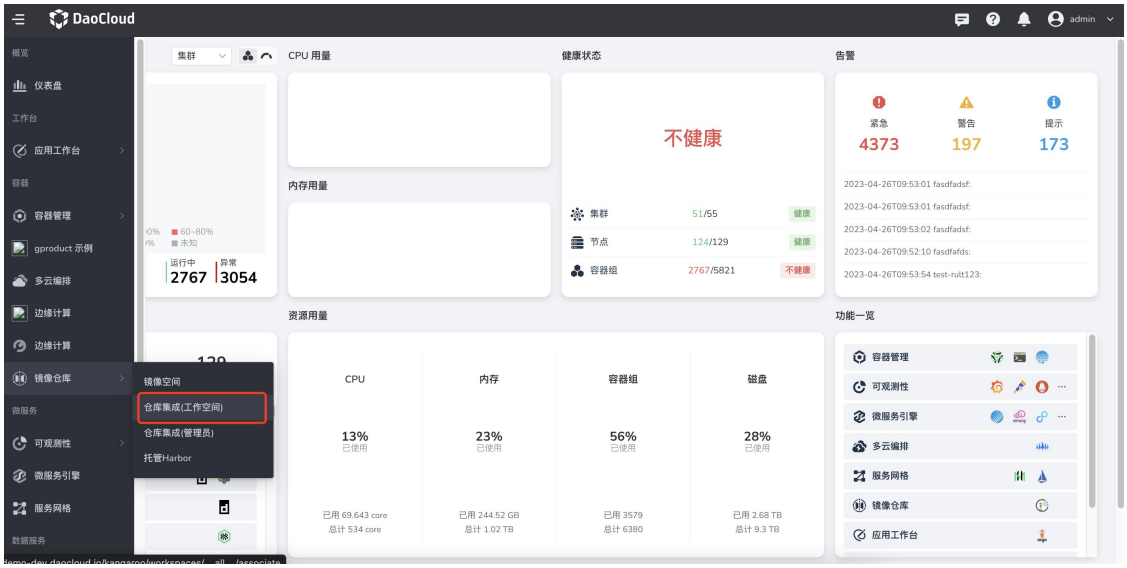
镜像空间

Harbor 提供了基于镜像空间（project）的镜像隔离功能。镜像空间分为公开和私有两种类型：

- 公开镜像仓库：所有用户都可以访问，通常存放公开的镜像，默认有一个公开镜像空间。
- 私有镜像仓库：只有授权用户才可以访问，通常存放镜像空间本身的镜像。

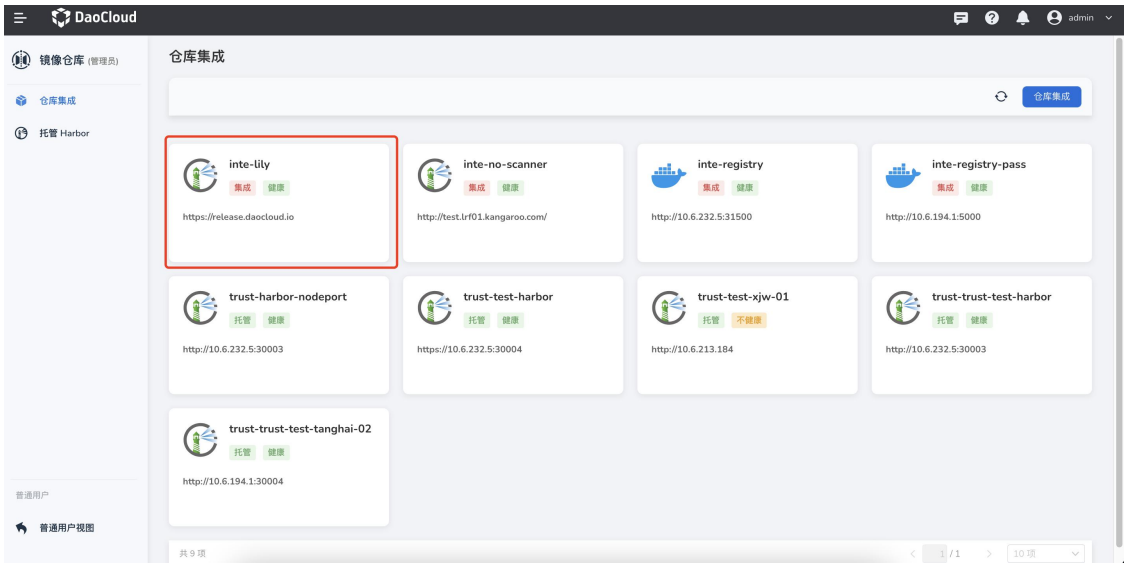
前提条件：已经创建或集成了一个外部 Harbor 仓库。

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击镜像仓库 -> 仓库集成(管理员)。



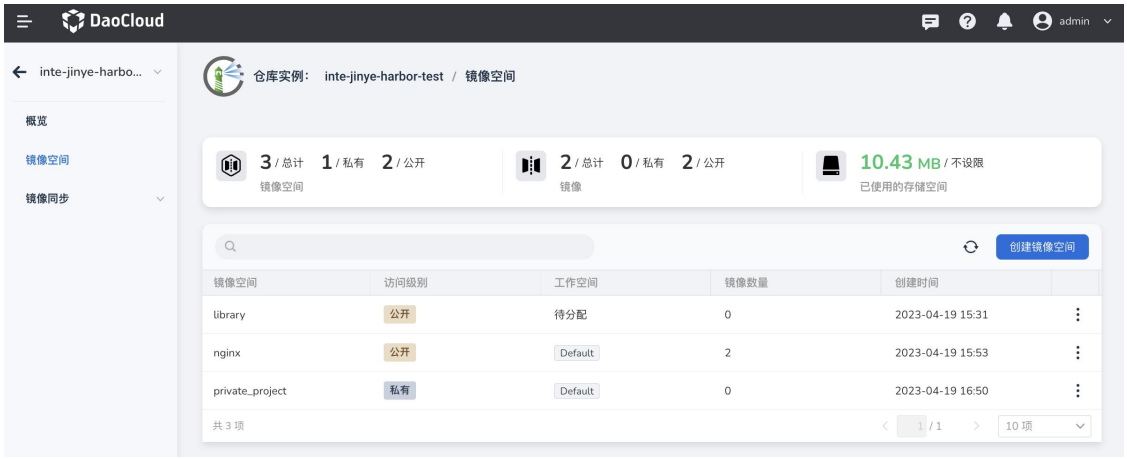
仓库集成

2. 点击某个仓库名称。



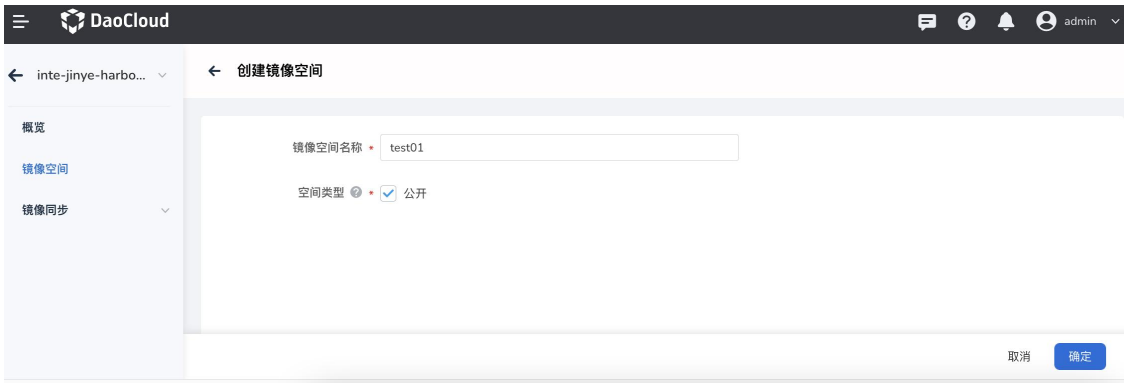
点击某个名称

3. 在左侧导航栏点击镜像空间，在右上角点击创建镜像空间。



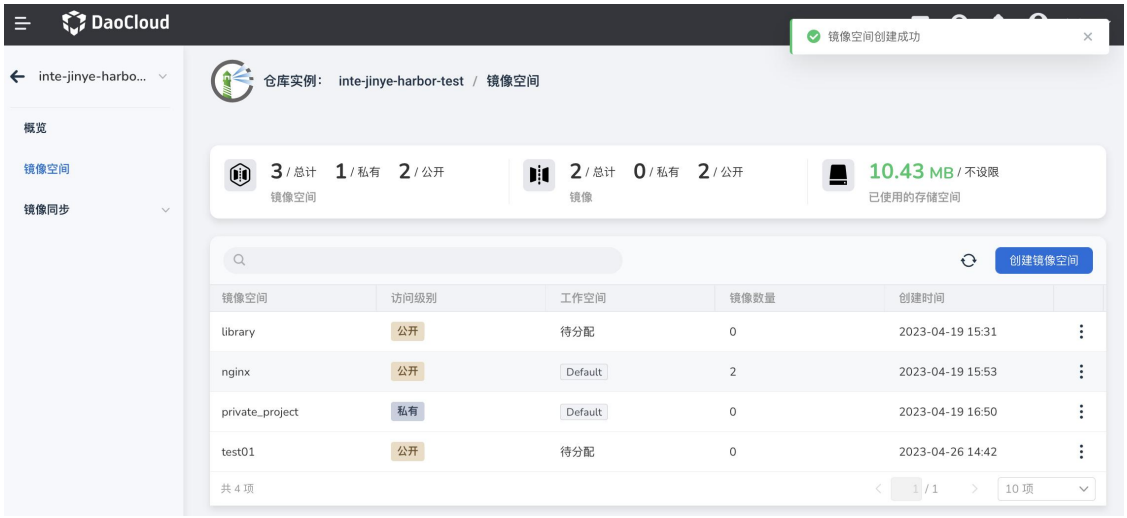
创建镜像空间

4. 填写镜像空间名称，勾选类型后点击确定。



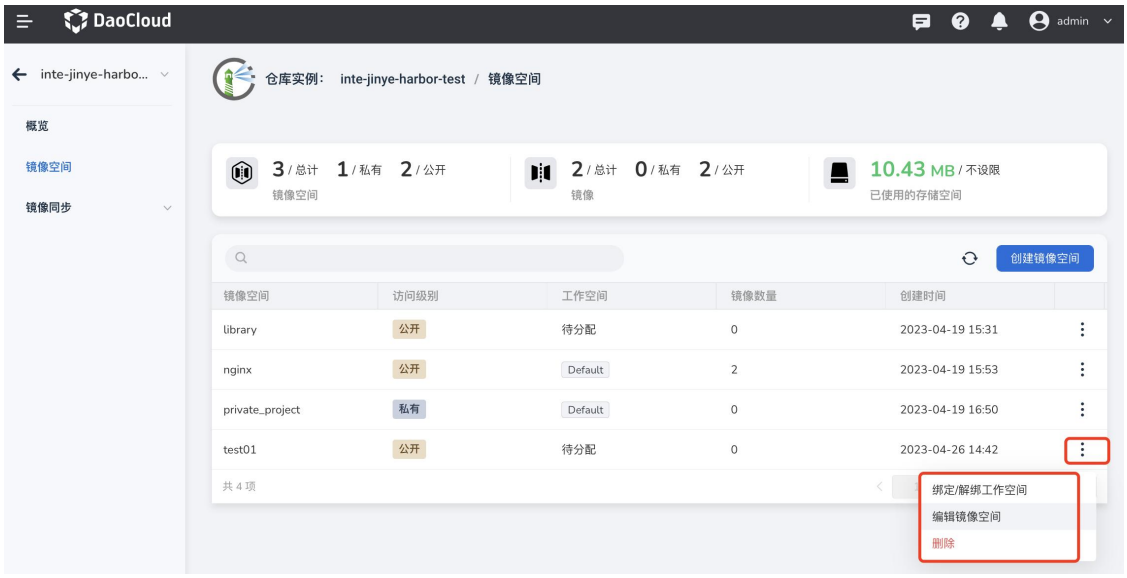
填写

5. 返回镜像空间列表，提示镜像空间创建成功。



成功

6. 找到刚创建的镜像空间，点击右侧的 ⋮，可以执行绑定/解绑工作空间、删除等操作。



其他操作

!!! info

– 若镜像空间状态为`公开`，则空间中的镜像能够被平台上的所有 Kubernetes 命名空间拉取使用；
– 若镜像空间状态为`私有`，则只有在管理员 Admin 将该镜像空间绑定到一个或多个工作空间（租户）后，才能被工作空间（租户）下的 Kubernetes 命名空间拉取使用。

下一步：绑定工作空间

绑定/解绑工作空间

镜像空间有两种类型：公开和私有。 公开镜像空间中的镜像为公开镜像，私有镜像空间中的镜像为私有镜像。

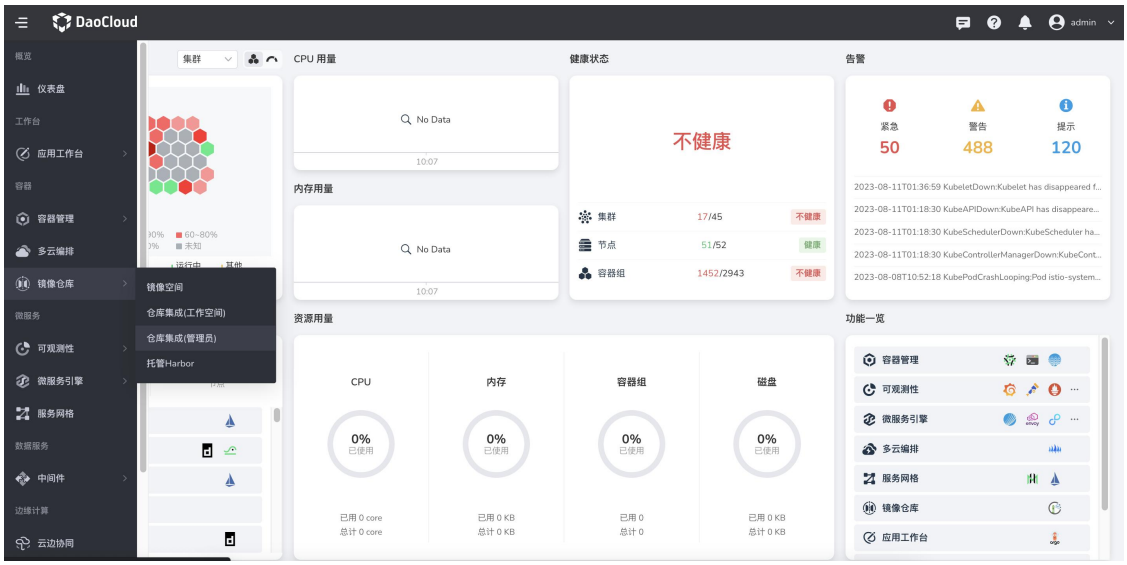
- 对于公开镜像，用户在容器管理模块部署应用时能够通过选择镜像按钮，选择镜像仓库 -> 仓库集成中所有公开镜像空间中的镜像部署应用，无需任何配置。
- 对于私有镜像，则需要管理员将私有镜像空间分配给工作空间（租户）后，才能够被工作空间下的成员使用，确保私有镜像的安全性。

前提条件：已经创建或集成了一个外部 Harbor 仓库，且已创建镜像空间。

绑定步骤

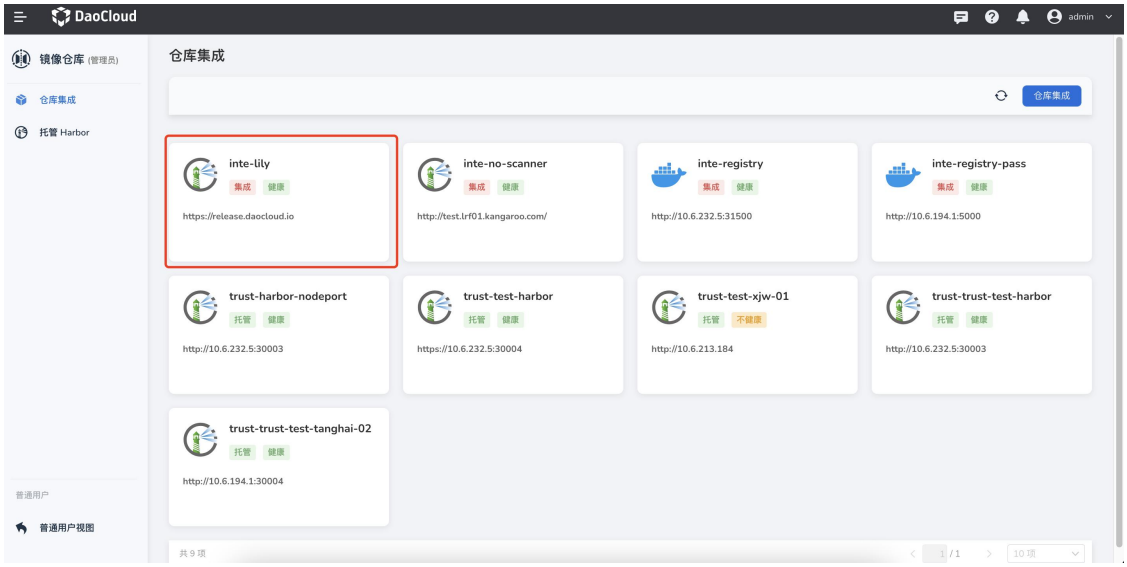
通常对私有的镜像空间，执行工作空间绑定，便于租户或成员使用其下的镜像。

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击镜像仓库 -> 仓库集成(管理员)。



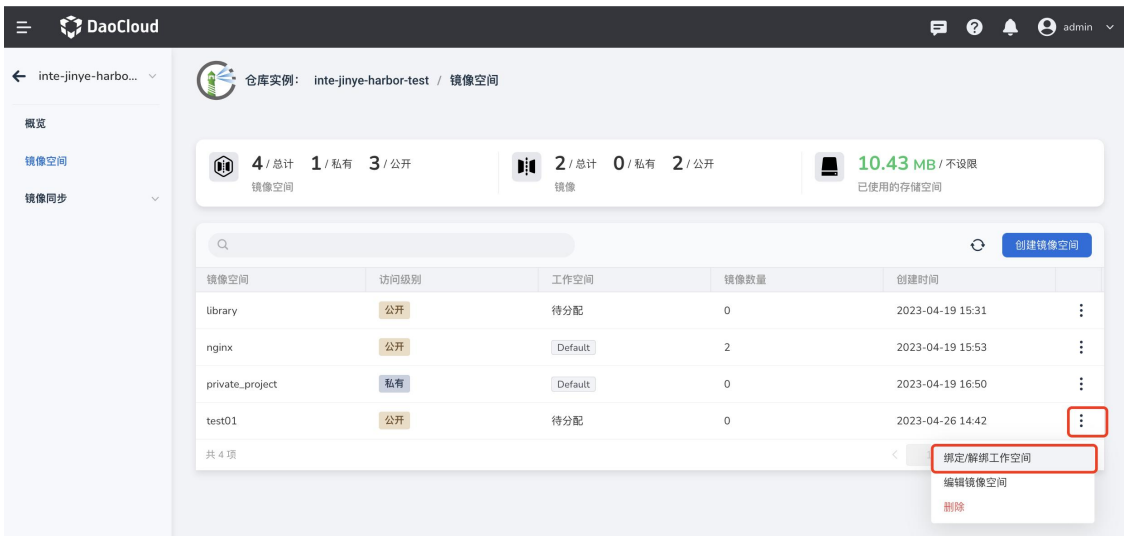
仓库集成

2. 点击某个仓库名称。



点击某个名称

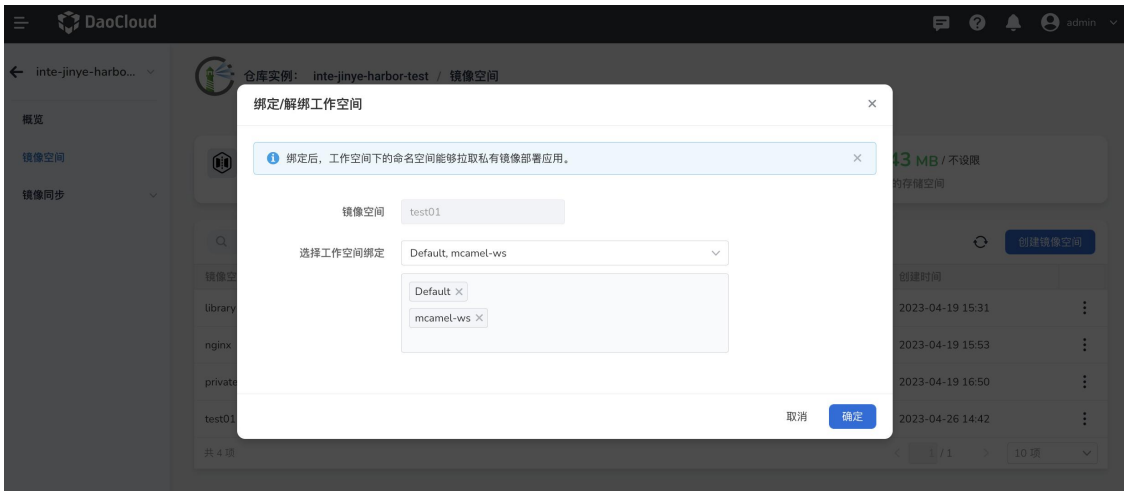
3. 在左侧导航栏点击镜像空间，在某个镜像空间最右侧，点击 ⋮，选择绑定/解绑工作空间。



绑定/解绑

若没有绑定任何工作空间，工作空间一栏将显示待分配。

4. 在弹窗中，选择一个或多个工作空间，点击确定。



绑定

若想取消某个工作空间的绑定，只需在已选中的工作空间列表中，点击 x。

5. 提示绑定/解绑工作空间成功，光标移到工作空间一栏，将显示所绑定的工作空间。



绑定

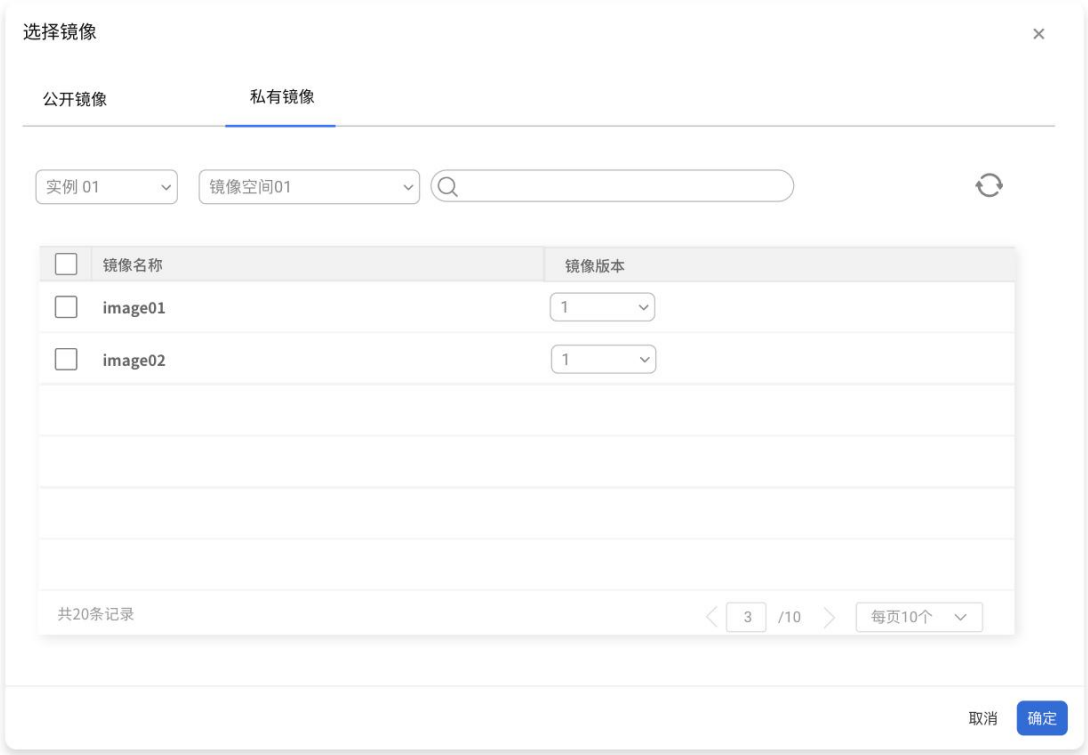
该工作空间（租户）下的成员将都能拉取这个镜像空间中的镜像。

FAQ

1. 在 Kubernetes 命名空间部署应用时，通过选择镜像按钮仍选择不到镜像空间中的镜像。

    - 排查该 Kubernetes 命名空间是否绑定了工作空间（需要绑定）。

- 排查该镜像空间是否绑定了 Kubernetes 命名空间所在的工作空间（需要绑定）。
- 排查该镜像空间状态是私有该是公开，切换如下 tab 查找。



镜像仓库

2. 将镜像空间分配给工作空间使用，与在工作空间下集成仓库有什么区别?

   平台管理员 Admin 可以统一管理，批量将一个镜像空间分配给多个工作空间使用，而无需分别关联。

   工作空间管理员 Workspace Admin 可以根据需要自己集成外部镜像仓库给成员使用，而无需全部依赖于平台管理员，使用起来更为灵活。

下一步：部署应用

镜像同步

镜像同步是指将两个或多个镜像仓库之间的镜像进行同步更新的过程。 在软件开发或系统管理中，这种同步更新方式常用于确保在多个服务器之间共享相同的软件或操作系统镜像， 以便在进行部署时能够确保一致性和减少工作量。通常情况下，通过镜像同步可以实时地将镜像内容同步到其他服务器上， 从而保证多个服务器上的镜像都是最新版本。

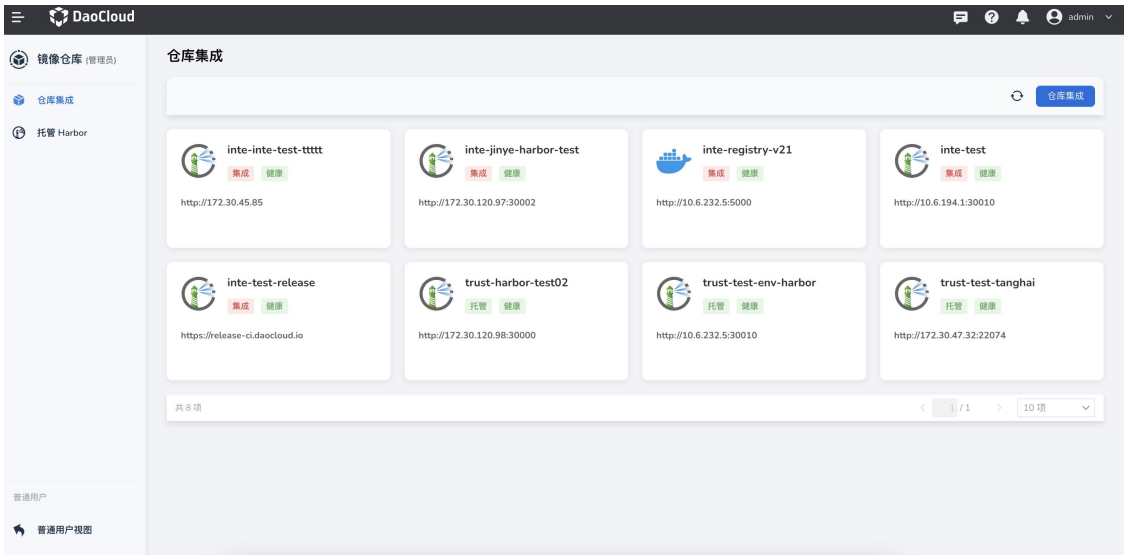
DCE 5.0 镜像仓库可以让用户创建同步规则，添加目标仓库等。

!!! note

创建同步规则和添加目标仓库需要管理员权限。

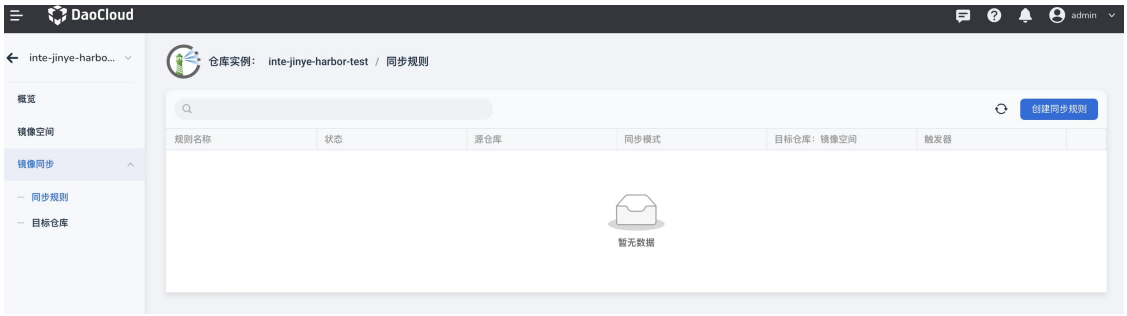
创建同步规则

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。



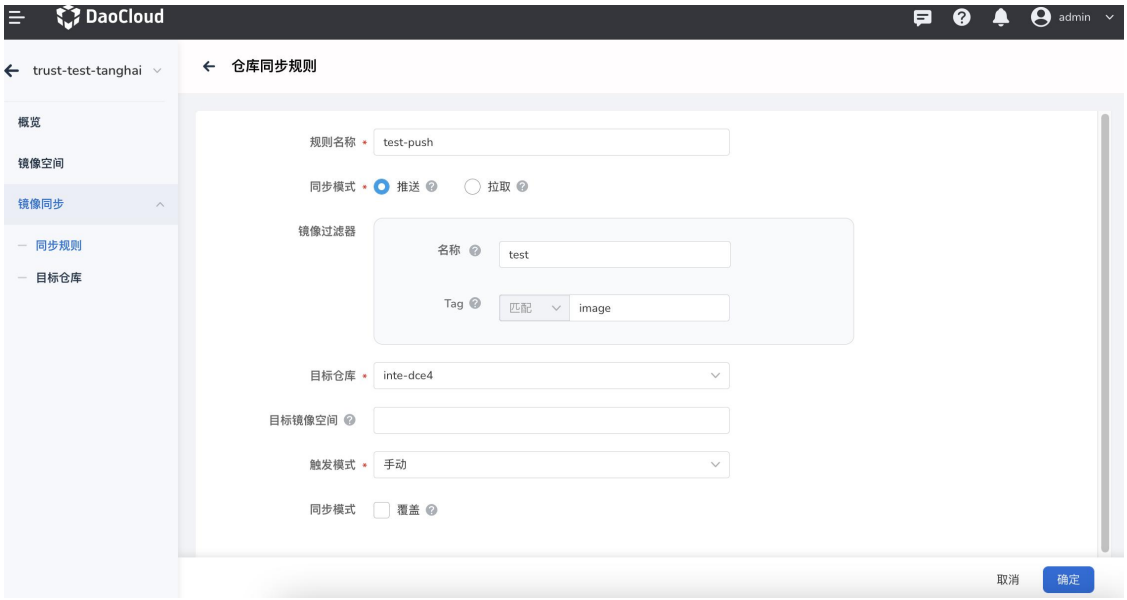
选择一个仓库

2. 在左侧导航栏，依次点击镜像同步 -> 同步规则，点击创建同步规则按钮。



点击按钮

3. 填写各项参数后点击确定。



配置参数

同步模式：

- 推送：将镜像同步到目标仓库
- 拉取：把目标仓库的镜像同步到当前仓库
- 覆盖：指定如果存在相同资源的名称，是否要覆盖 镜像过滤器：
- 名称：按名称过滤当前镜像空间下的资源，不填或 ** 将匹配所有资源。

- Tag：按 tag/version 过滤当前镜像空间下的资源，不填或 ** 将匹配所有资源。 目标镜像空间：如果不填写，镜像将被放到与源仓库相同的镜像空间中。

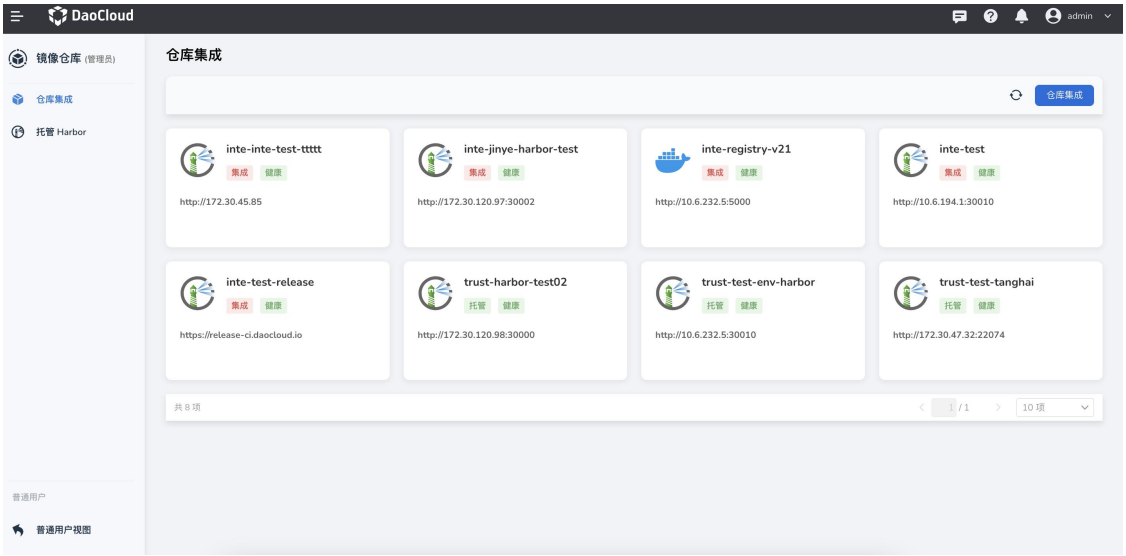
4. 返回同步规则列表，新建的规则默认处于启用状态。点击右侧的 ⋮，可以执行同步、编辑、禁用、删除等操作。



更多操作

添加目标仓库

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。



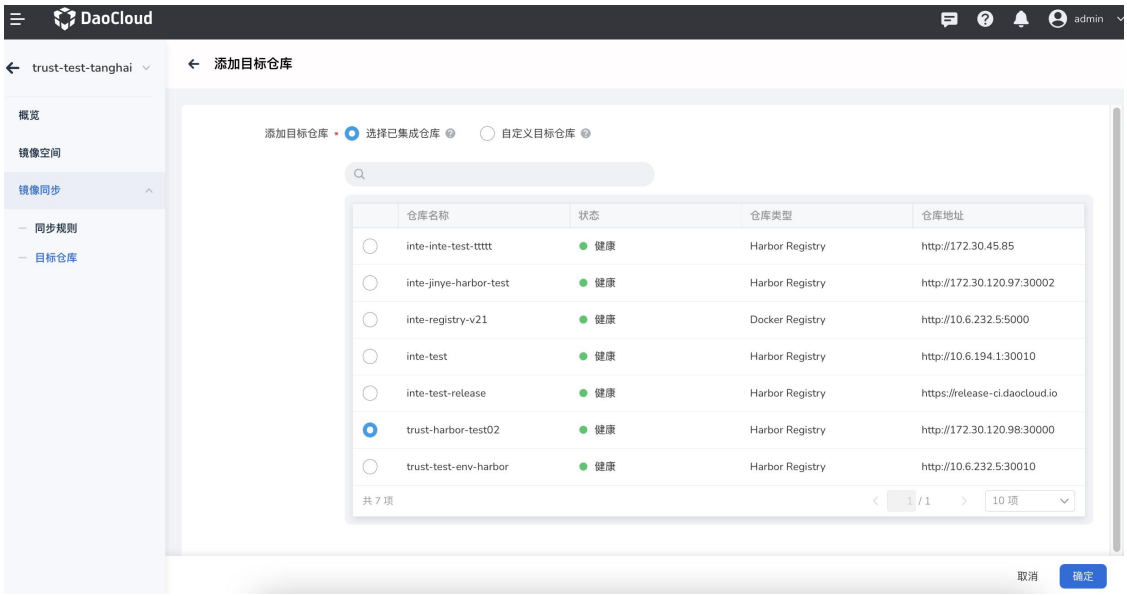
选择一个仓库

2. 在左侧导航栏，依次点击镜像同步 -> 目标仓库，点击添加目标仓库按钮。

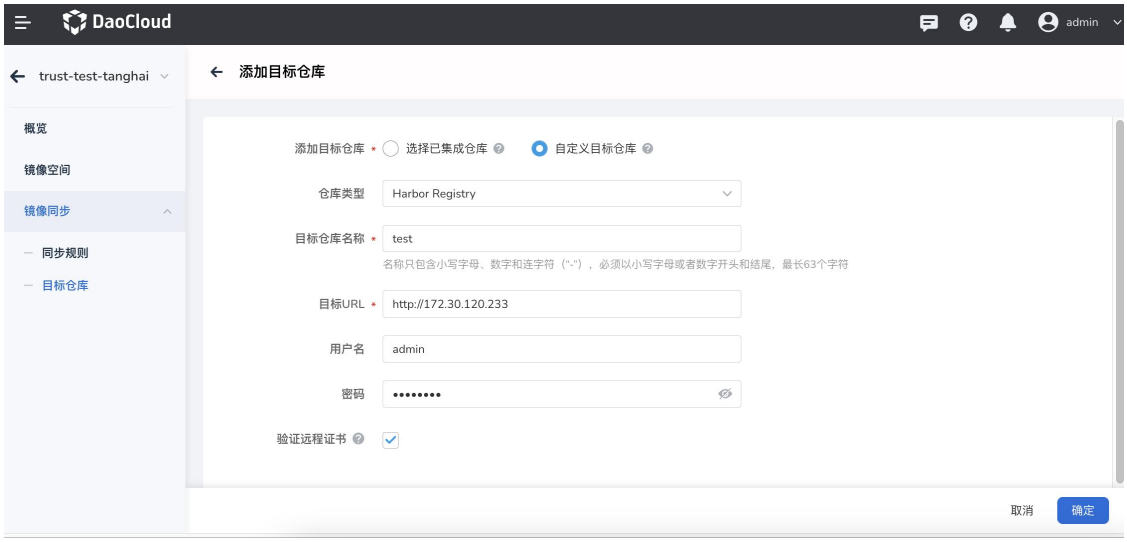


点击按钮

3. 可以点选已集成的仓库，或自定义仓库。



选择



自定义

4. 返回目标仓库列表，点击右侧的 ⋮，可以执行编辑、删除等操作。

推送镜像

创建托管 Harbor 和镜像空间后，您可以按照如下说明登录并将镜像推送的镜像仓库中； 或者登录原生 Harbor，查看原生 Harbor 在每个镜像空间（项目）下提供的引导。

推送方式一

前提：已创建托管 Harbor 和镜像空间

您可在本地构建新的容器镜像或从 DockerHub 上获取一个公开镜像用于测试。本文以 DockerHub 官方的 Nginx 最新镜像为例，在命令行工具中依次执行以下指令，即可推送该镜像。 请将 library 及 nginx 依次替换为您实际创建的镜像空间名称及镜像仓库名。

1. 登录镜像仓库

   docker login --username=<镜像仓库登录名> <镜像仓库地址>

   示例：docker login --username=admin http://test.lrf02.kangaroo.com

   在返回结果中输入镜像仓库密码（创建托管 Harbor 时设置的密码）。

2. 推送镜像

   执行以下命令，给镜像打标签。

   docker tag <镜像仓库名称>:<镜像版本号> <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>

   示例：docker tag nginx:latest http://test.lrf02.kangaroo.com/library/nginx:latest

   执行以下命令，推送镜像至镜像空间 library 中。

docker push <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>

示例：docker push http://test.lrf02.kangaroo.com/library/nginx:latest

3. 拉取镜像

   执行以下命令，拉取镜像。

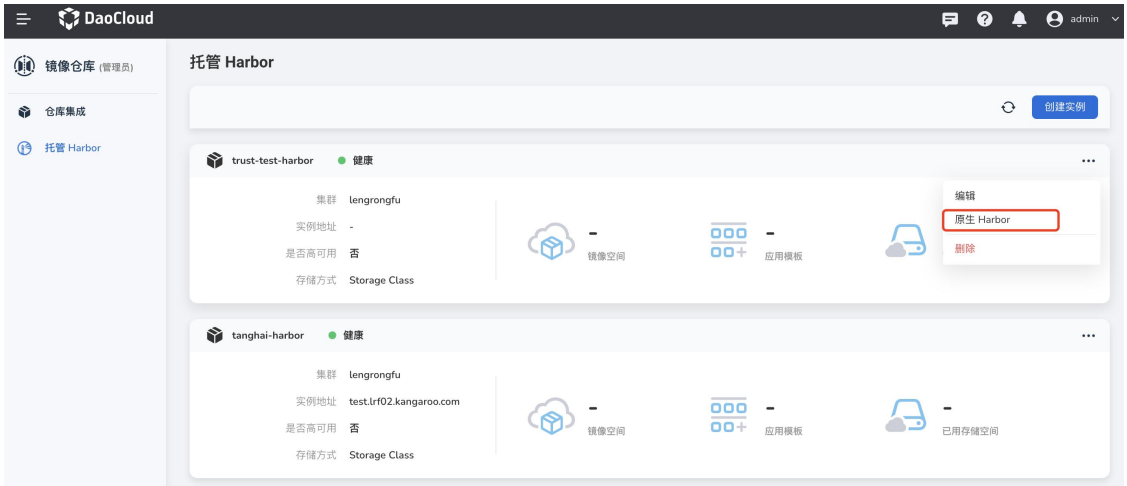
   docker pull <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>

   示例：docker pull http://test.lrf02.kangaroo.com/library/nginx:latest

推送方式二

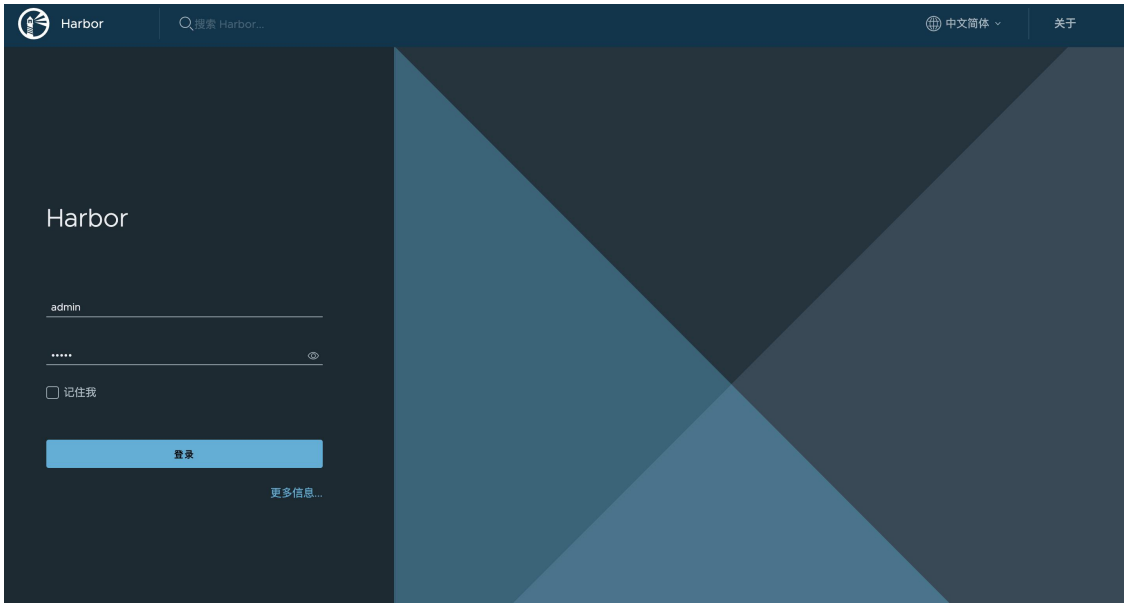
前提：已创建托管 Harbor 和镜像空间。

1. 在托管 Harbor 列表页面中，点击目标镜像仓库右侧的 ...，点击原生 Harbor，进入原生 Harbor 的登录页。



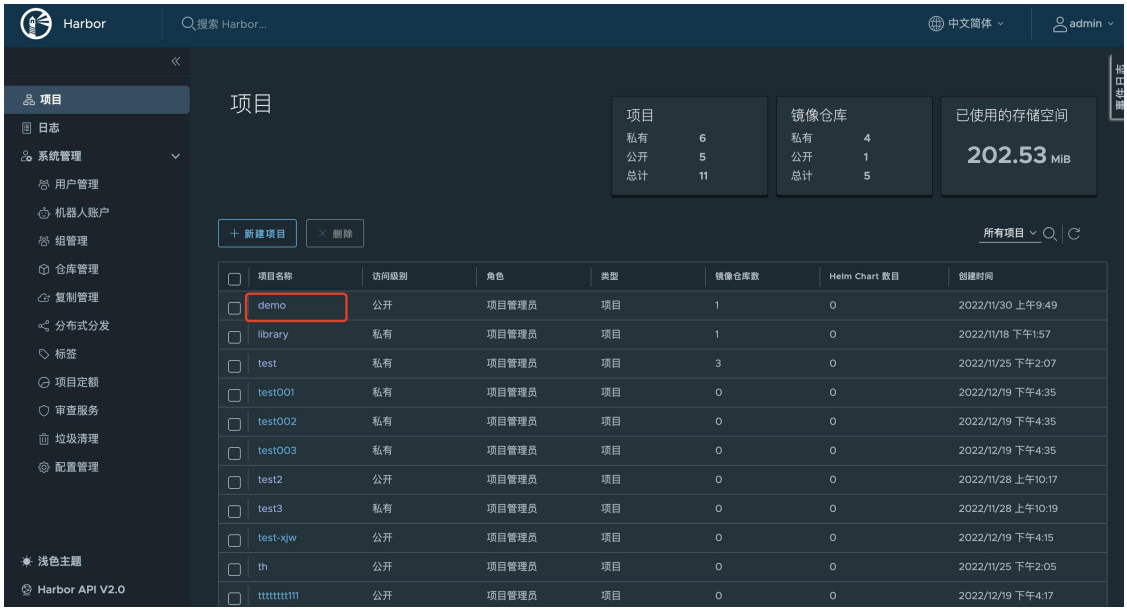
原生 Harbor

2. 输入创建托管 Harbor 时设置的用户名和密码进入原生 Harbor。



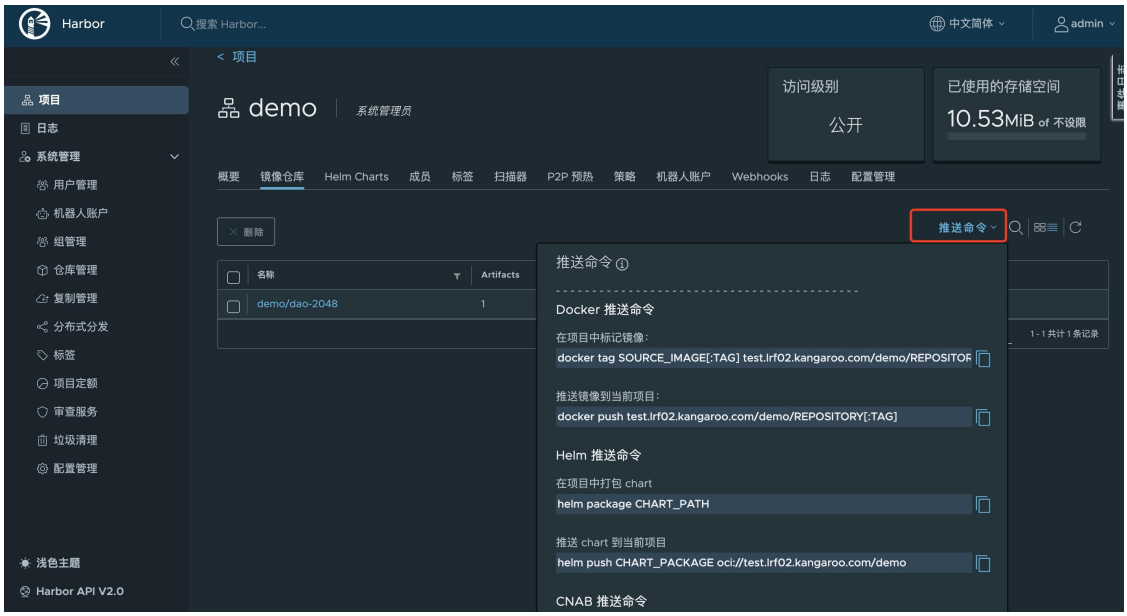
输入用户名和密码

3. 点击目标镜像空间（项目）的名称，进入镜像空间。



镜像空间

4. 点击右侧的推送命令，查看原生 Harbor 提供的推送命令。



查看推送命令

!!! tip

相对于[方式一](#_2)，原生 Harbor 的推送命令自动填入了镜像仓库地址和镜像空间名称。

Workspace Admin 快速使用镜像仓库为工作空间服务

在 DCE 5.0 中，Workspace Admin 能够通过仓库集成(工作空间)方式集成 Harbor 和 Docker Registry 两种外部镜像仓库。 集成后工作空间成员能够在镜像列表中看到仓库集成(工作空间)的所有镜像，以及在工作空间下的命名空间中部署应用时能够通过镜像选择器选择该仓库中的镜像进行部署。

假设您已经创建了一个外部 Harbor 或者 Docker Registry 镜像仓库，按照以下步骤可以将外部 Harbor 或者 Docker Registry 共享给工作空间成员使用。

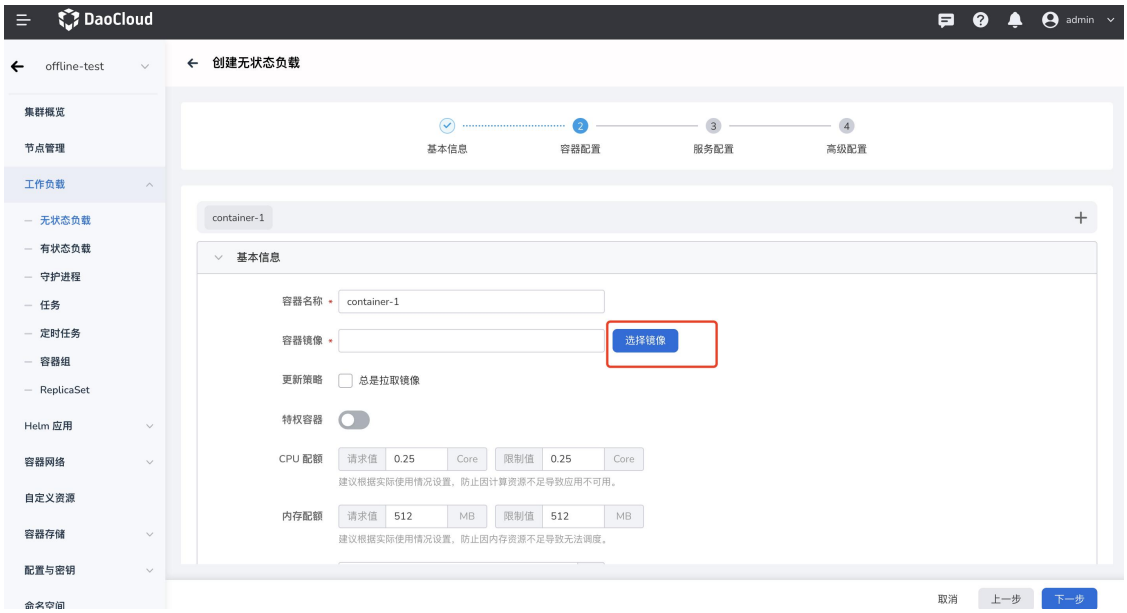
前提条件为：

1. 您是 Workspace Admin，并且该工作空间下绑定了一些命名空间
2. 您拥有一个或者多个外部镜像仓库（Harbor 或 Docker Registry）

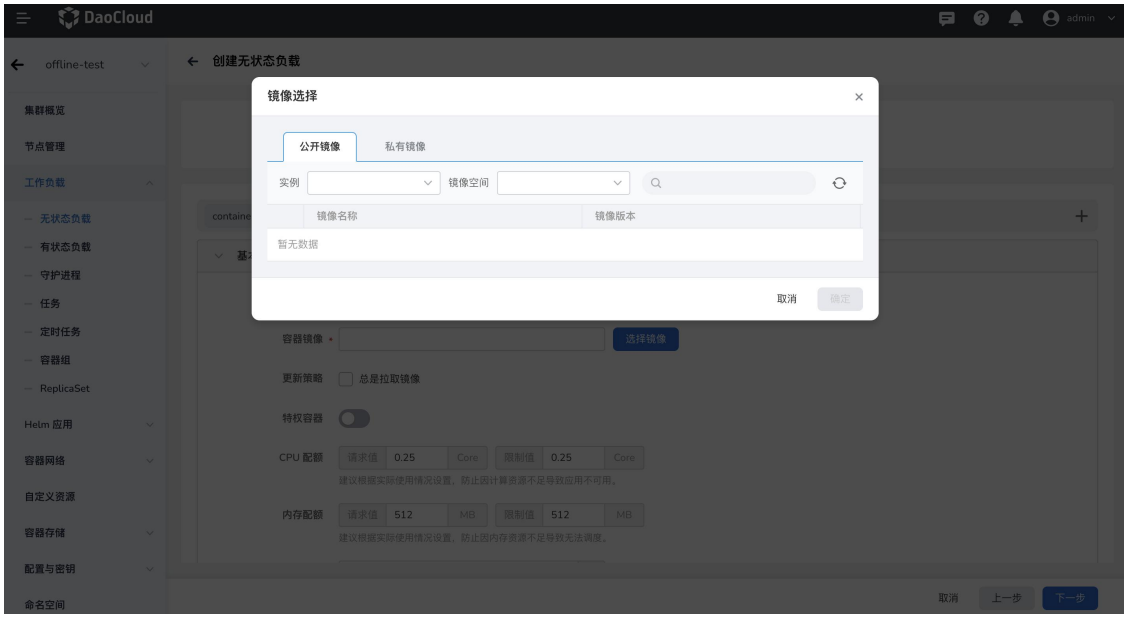
操作步骤为：

```
graph TB

associate[仓库集成] --> push[推送镜像] --> deploy[部署应用]

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class associate,push,deploy cluster;

click associate "https://docs.daocloud.io/kangaroo/related-registry/"
click push "https://docs.daocloud.io/kangaroo/quickstart/push/"
click deploy "https://docs.daocloud.io/kpanda/user-guide/workloads/create-deployment/"
```

预期结果：在该工作空间下的命名空间部署应用时能够通过镜像选择器，选择该镜像空间下的镜像进行部署应用。



按钮



镜像选择

管理外部镜像仓库

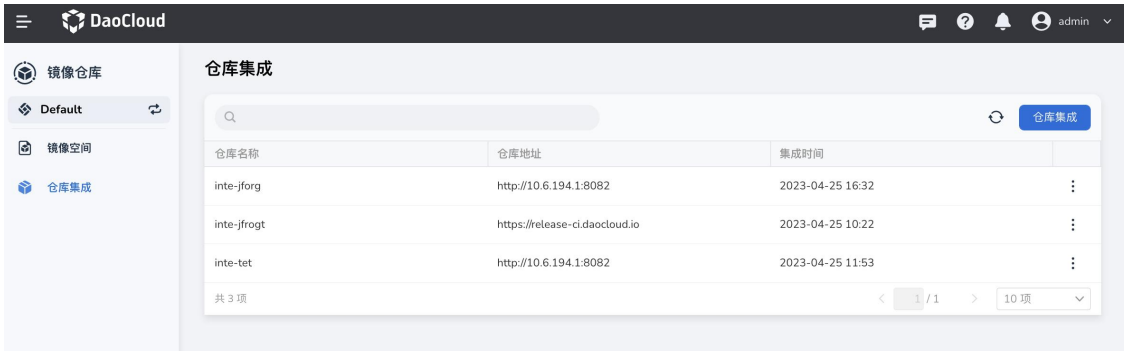
如果您有一个或多个 Harbor 或 Docker 镜像仓库，完全可以用 DCE 5.0 镜像仓库进行统一管理。视操作员的角色权限，可以使用两种方式：

- 仓库集成（工作空间）
- 仓库集成（管理员）

仓库集成(工作空间)

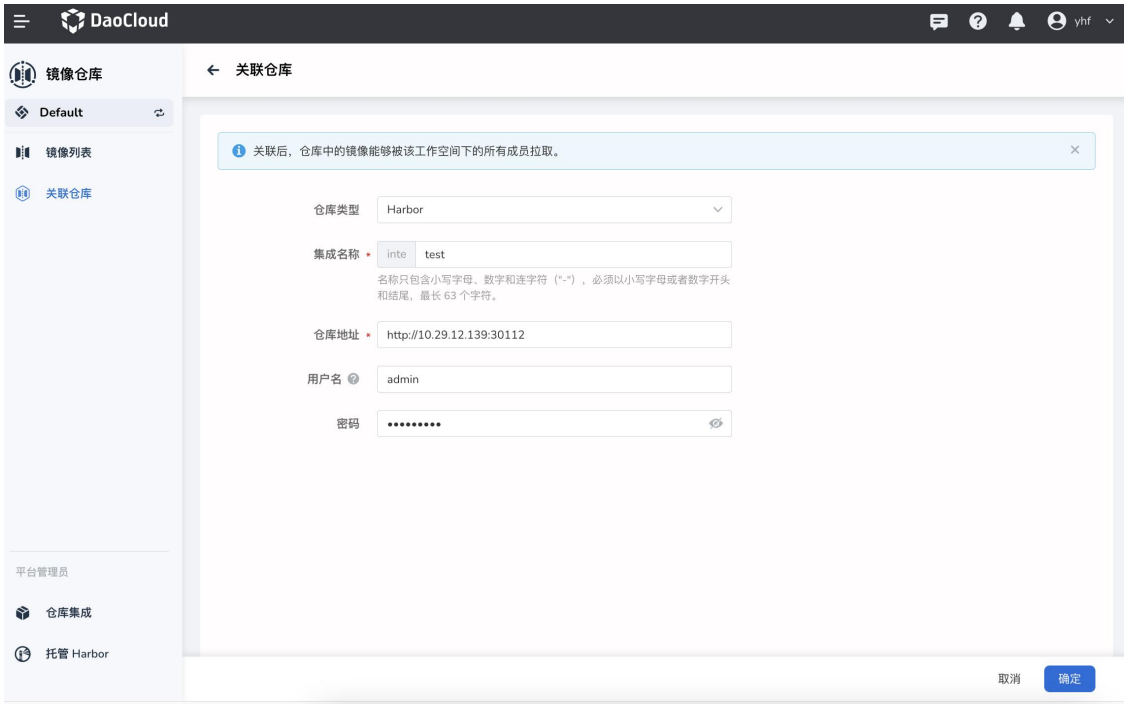
如果是 Workspace Admin，可以通过仓库集成(工作空间)功能，将现有镜像仓库关联到 DCE 平台，供工作空间的成员使用。简单的操作步骤如下：

1. 以 Workspace Admin 角色登录，从左侧导航栏点击仓库集成(工作空间)，点击右上角的仓库集成按钮。



仓库集成(工作空间)

2. 填写表单信息后点击确定。



填写表单

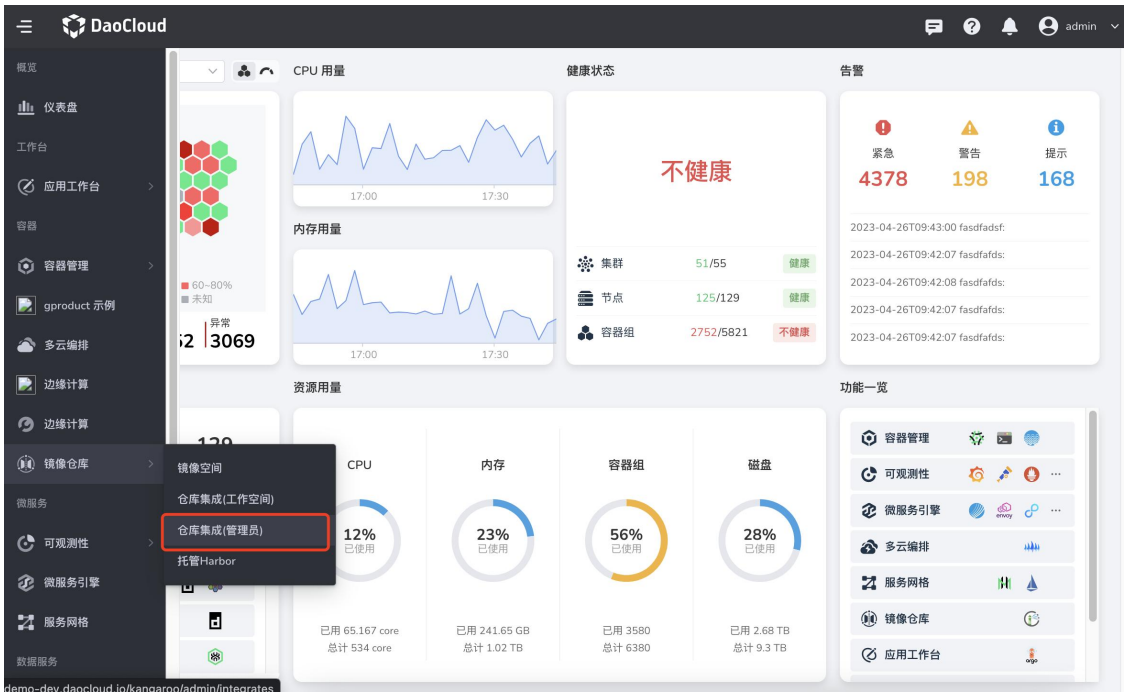
!!! note

1. Docker Registry 镜像仓库若未设置密码可不填写，Harbor 仓库必须填写用户名/密码。
2. 有关实际操作演示，请参阅[仓库集成视频演示](../../videos/kangaroo.md)。

仓库集成(管理员)

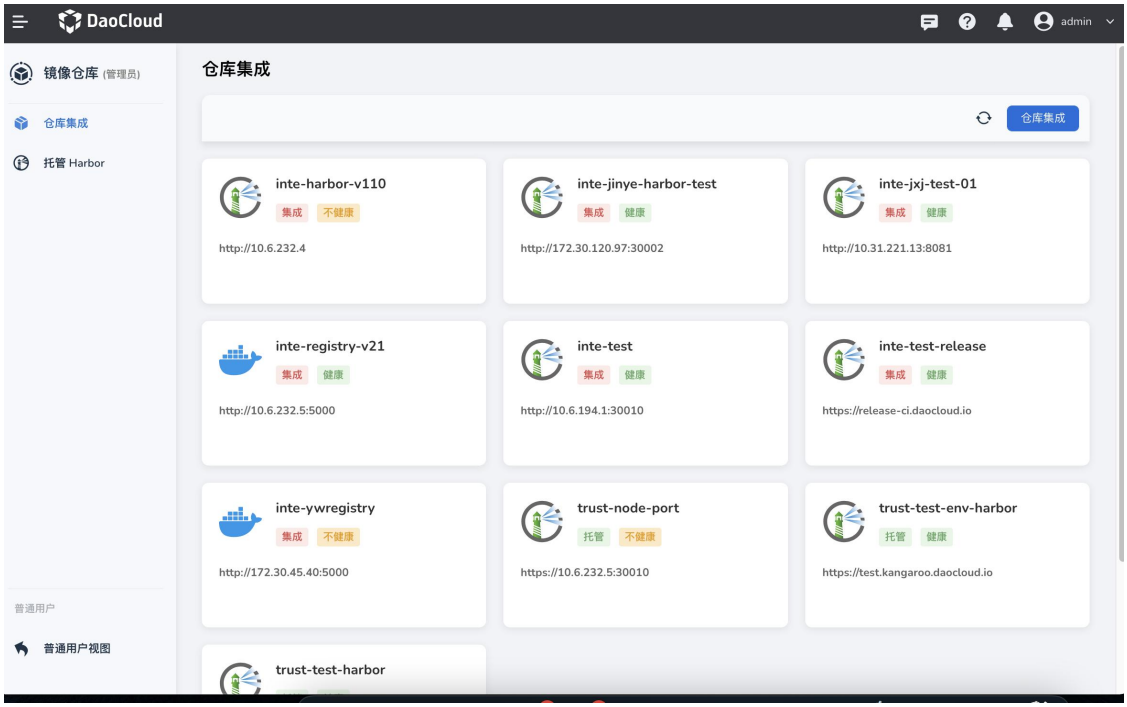
如果是 Admin（平台管理员），还可以通过仓库集成功能，将现有镜像仓库集成到 DCE 平台。 仓库集成是集中管理平台镜像仓库的入口，平台管理员可以通过将镜像空间与工作空间绑定的方式， 将某个私有镜像空间分配给一个或者多个工作空间（工作空间下的命名空间）使用。或者将镜像空间设置为公开，供平台所有命名空间使用。

1. 以 Admin 角色登录，在左侧导航栏点击仓库集成(管理员)。



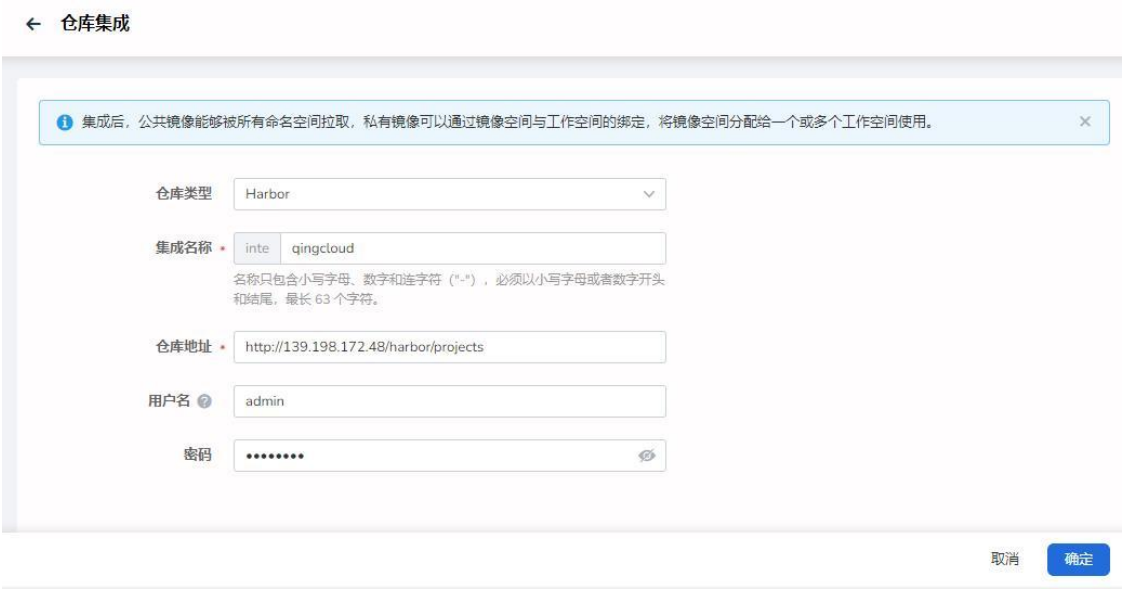
仓库集成

2. 点击右上角的仓库集成按钮。



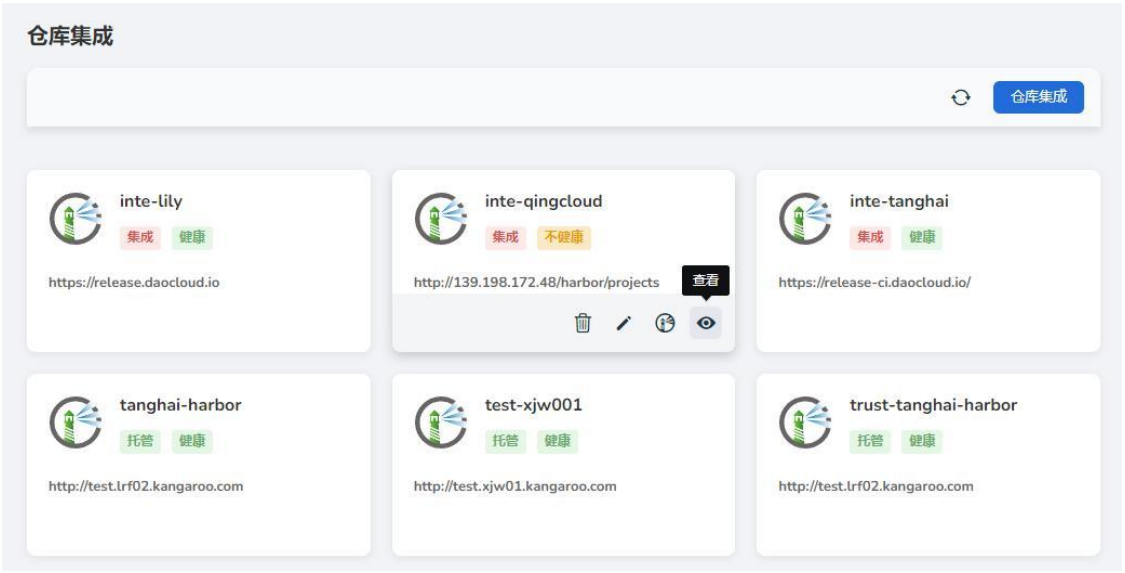
仓库集成

3. 选择仓库类型，填写集成名称、仓库地址、用户名和密码后点击确定。



填写表单

4. 在集成的仓库列表中，光标悬浮于某个仓库上，点击眼睛图标可查看概览。



查看概览

5. 概览页面显示了当前仓库的基本信息、统计信息，还在顶部提供了快速入门，方便管理镜像空间、工作空间、创建应用。


DaoCloud
admin
inte-qingcloud
概览
镜像空间
仓库实例： inte-qingcloud / 概览
更多功能请 前往原生 Harbor 实例 配置。Harbor实例的用户名和密码，请找该实例的创建人获取。
快速入门
1 创建镜像空间
创建镜像空间来管理和隔离镜像资源
2 绑定镜像空间与工作空间
通过绑定关系，将镜像空间分配给一个或多个工作空间使用。
3 部署应用
前往应用工作台通过镜像部署应用
基本信息
inte-qingcloud-2093
实例名称
http://139.198.172.48/harbor/proje...
实例地址
/
部署位置
其它
副本数
集成
类型
-
描述
-
集成人
2022-12-13 10:44
集成时间
统计信息
2
项目
1
镜像
0
Helm Charts
0 KB
已用存储
已用存储(KB)
最近 7 日
1
0.8

查看概览

离线环境开启镜像安全扫描

Trivy 会使用 admin 账号来拉取要扫描的镜像，在使用前需要确认可以使用 admin 进行 docker login。

下载离线镜像包

trivy 有两个版本，目前都需要下载版本 2，在 https://github.com/aquasecurity/trivy-db 项目中已经不提供版本 2 直接下载离线的 trivy-db 包了，被打包成了 oci 包。

如下使用 oras 工具来下载，先安装 oras，注意区分操作系统版本，如下是下载 linux 平台的：

```
export VERSION="1.0.0"
$ curl -LO "https://github.com/oras-project/oras/releases/download/v${VERSION}/oras_${VERSION}_linux_amd64.tar.gz"
$ mkdir -p oras-install/
$ tar -zxf oras_${VERSION}_*.tar.gz -C oras-install/
$ sudo mv oras-install/oras /usr/local/bin/
$ rm -rf oras_${VERSION}_*.tar.gz oras-install/
```

使用 oras 工具下载 trivy-db：

```
$ oras pull ghcr.io/aquasecurity/trivy-db:2
db.tar.gz
$ tar -zxf db.tar.gz
# 解压出来有两个文件
db/metadata.json
db/trivy.db
```

开启托管 Harbor 离线扫描

在托管 harbor 所在集群进行编辑

$ kubectl –n {namespace} edit harborclusters.goharbor.io {harbor–name}

主要修改 trivy offlineScan 和 skipUpdate 改为 true

trivy:

offlineScan: true

skipUpdate: true

也可以从 DCE 5.0 集群管理页面中进行修改



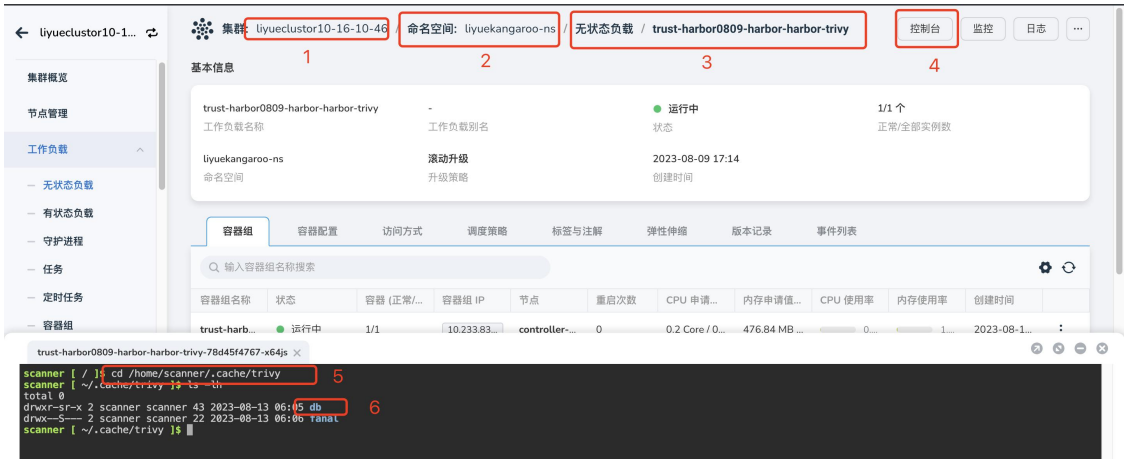
edit–harborcluster.png

1. 从集群列表进入某个集群，选择自定义资源
2. 选择 harborcluster 的资源
3. 进入托管 Harbor 所在 Namespace
4. 选择 YAML
5. 编辑 YAML：

```
trivy:
    offlineScan: true
    skipUpdate: true
```

上传 trivy.db 和 metadata.json 文件

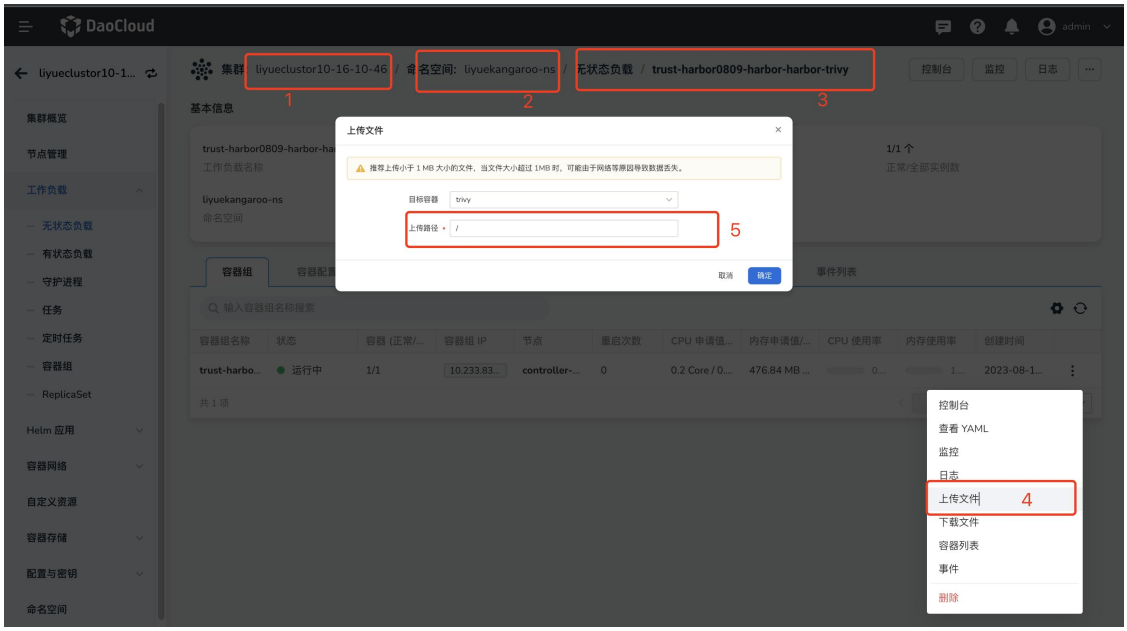
先在 trivy pod 中创建对应的目录 /home/scanner/.cache/trivy/db



upload-trivy-db.png

1. 从集群列表进入某个集群
2. 进入托管 Harbor 所在 Namespace
3. 进入 trivy 的工作负载
4. 点击控制台进入容器（如果有多个副本，每个副本都需要设置）
5. 进入容器后，执行 cd /home/scanner/.cache/trivy
6. 执行 mkdir db

创建好目录之后上传离线包



upload-trivy-offline-db.png

1. 从集群列表进入某个集群
2. 进入托管 Hharbor 所在 Namespace
3. 进入 trivy 的工作负载
4. 点击上传文件
5. 在弹窗中写入上传路径 /home/scanner/.cache/trivy/db，点击确认。
6. 进入选择文件页面，分别上传 trivy.db 和 metadata.json 文件。

托管 Harbor 应该选择什么访问类型

在创建托管 Harbor 时，需要选择访问类型。目前支持两种类型：Ingress 和 NodePort。 下表说明了这两种访问类型的优缺点；

Ingress

NodePort

优点

使用 Ingress 方便管理和使用，域名易记住，服务易迁移。

支持上传 HTTPS 证书，可以实现拉镜像不用配置非安全仓库。

不依赖任何组件，可快速启动服务进行 Demo 和试用。

NodePort 性能较高，没有额外的路由处理。

缺点

在私有网络环境需要配合 DNS 服务做域名解析，需要依赖基础网络设施。

Ingress 会对性能产生一定的损失，因为需要进行额外的路由处理。

管理不便，IP 和某一台宿主机强绑定，并且镜像地址变得不可迁移；

只能使用自签证书，在拉取镜像时需要配置非安全仓库。

!!! tip

– 在 **生产环境** 中，建议使用 Ingress 方式，因为 Ingress 可以使用 HTTPS 协议，
　可以使用自签证书或者购买证书，这样可以避免在拉取镜像时需要配置非安全仓库的问题。
– 在 **测试环境** 中，建议使用 NodePort 方式，因为 NodePort 方式不依赖任何组

件，

可快速启动服务进行 Demo 和试用。

Ingress

前提条件：

- 集群必须安装 Ingress 组件，Ingress 组件可以是 Nginx、Traefik、HAProxy 等，具体安装方法请参考 Ingress。
- 集群必须安装 DNS 服务，DNS 服务可以是 CoreDNS、KubeDNS、Bind 等，具体安装方法请参考 DNS。
- 集群必须安装 LoadBalancer 组件，LoadBalancer 组件可以是 MetalLB、F5、Nginx 等，具体安装方法请参考 LoadBalancer。

Ingress 流程

Ingress 可以从集群外部暴露 HTTP 和 HTTPS 路由到集群内部的服务，流量路由由 Ingress 资源上定义的规则控制。 在托管 Harbor 中 Ingress 使用流程如下图所示。

```
flowchart LR

subgraph cluster[Cluster]
    direction LR
    ingress[Ingress] -->|路由规则| harbor[Harbor Service]
    harbor --> pod1[Pod]
    harbor --> pod2[Pod]
end

client([Client]) -.->|以 LB 或 NodePort 暴露服务| ingress

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
```

```
class client plain;
class ingress,harbor,pod1,pod2 k8s
```

对于 **私有云** ，使用 Ingress 部署完托管 Harbor 之后，需要在 DNS 服务中添加域名解析， 将域名解析到 LoadBalancer 的 IP 地址上，这样用户就可以通过域名访问 Harbor 了； 添加 DNS 域名解析需要在 Kubernetes 集群内和 Kubernetes 集群外分别进行操作，具体操作步骤如下：

在 Kubernetes 集群内添加 DNS 域名解析

在 Kubernetes 集群内，需要在 CoreDNS 组件中添加 DNS 域名解析，具体操作步骤如下：

1. 进入 CoreDNS 组件的 ConfigMap 中，执行如下命令：

   ```
   kubectl –n kube–system edit configmap coredns
   ```

2. 进入 CoreDNS 组件的 Configmap 中后，执行如下命令，将 harbor.example.com 解析到 LoadBalancer 的 IP 地址上：

   ```
   hosts {
     10.1.1.1 harbor.example.com
     fallthrough
   }
   ```

   !!! note

   `harbor.example.com` 为用户自定义的域名，LoadBalancer 的 IP 地址为 Ingress 的 IP 地址。

3. 保存退出，会重新创建 CoreDNS Pod：

   ```
   kubectl get pod –n kube–system coredns–5c98db65d4–2t2l2
   ```

在 Kubernetes 集群外添加 DNS 域名解析

在集群外也是通过单独部署一个 CoreDNS 组件来实现 DNS 域名解析，然后配置域名。参照上述第 2 步。

如果用户的域名是公网域名，那就不需要做上述步骤，公网域名能被解析。

NodePort

NodePort 是一种访问 Kubernetes 集群中 Service 的方法，它会在每个 Node 上打开一个端口， 该端口会将流量转发到 Service 的端口上。NodePort 使用流程如下图所示。

```
flowchart LR

subgraph cluster[Cluster]
    direction LR
    harbor[Harbor Service]
    harbor --> pod1[Pod]
    harbor --> pod2[Pod]
end

client([Client]) -.->|以 NodePort 暴露服务| harbor

classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class client plain;
class ingress,harbor,pod1,pod2 k8s
```

登录非安全镜像仓库

当要从非安全的镜像仓库中进行 Pull、Push 时，会遇到 x509: certificate signed by unknown authority 错误提示； 这是由于镜像仓库是可能是 http 服务，或者 https 的证书是自签名的就会出现这个问题。

我从如下三种运行时来说明如何解决这个问题；我们以要登录 test.registry.com 为例。

Docker

在 /etc/docker/daemon.json 文件中加入如下配置：

```
{
  "insecure-registries": [
    "test.registry.com",
    "test.registry.com1"
  ]
}
```

修改之后重启 docker 即可：systemctl restart docker

containerd

containerd 配置非安全仓库有两种方法，一种是从 1.4 版本之后引入的新方法，一种是之前存在直到 2.0 版本的旧方法。

新方法

编辑 /etc/containerd/config.toml 文件中的 config_path 配置项，默认值是 /etc/containerd/certs.d。

```
[plugins."io.containerd.grpc.v1.cri".registry]
  config_path = "/etc/containerd/certs.d"
```

在 /etc/containerd/certs.d 目录下创建一个以 registry 命名的文件夹，并在其中创建一个 hosts.tomo 的文件。

```
mkdir /etc/containerd/certs.d/test.registry.com
```

如果 registry 是带端口号的也需要运行：

```
mkdir /etc/containerd/certs.d/127.0.0.1:5000
```

```
server = "http://test.registry.com"

[host."http://test.registry.com"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
```

配置后不用重启，可以直接使用。

旧方法

在 /etc/containerd/config.toml 文件中加入如下的配置：

```
[plugins."io.containerd.grpc.v1.cri".registry.configs."test.registry.com".tls]
  insecure_skip_verify = true
```

配置之后需要重启 containerd，system restart containerd。

CRI-O

修改 /etc/crio/crio.conf 配置文件：

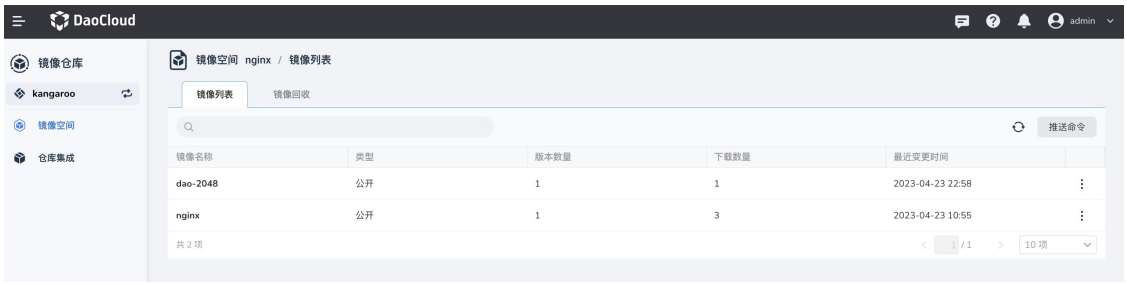
```
insecure_registries = ["test.registry.com"]
```

重启 crio: systemctl restart crio。

镜像列表

镜像列表以工作空间为维度，展示租户下所有可用的公开镜像和私有镜像。 镜像来源主要有两部分：

1. 第一部分是平台集成或托管的镜像仓库中全部的公开镜像，以及通过镜像空间与工作空间绑定而单独分配给该工作空间的私有镜像。
2. 第二部分是工作空间主动关联某镜像仓库而获得的全部公开或私有镜像。



镜像空间包含什么

主要功能

- 快速部署应用：镜像列表和应用工作台均以工作空间为维度，因此当您在应用工作台下选择同一工作空间部署应用时，可以通过选择镜像按钮一键获取该工作空间下镜像列表中所有可见的镜像，快速完成部署。
- 细粒度分配：管理员在平台集成或创建托管 Harbor 实例后，可以通过镜像空间与工作空间绑定的方式，将不同的镜像空间分配给不同的工作空间使用，实现细粒度分配。
- 镜像扫描：提供镜像扫描和分析功能，可视化查看漏洞等级及分布情况，提供扫描日志，便于追踪漏洞。
- 推送镜像：可通过平台提供的命令将镜像推送到有权限的镜像空间中。
- 层级等详情查看：对可见范围内的镜像提供镜像版本、层级信息、创建人、创建时间等详细信息的展示。

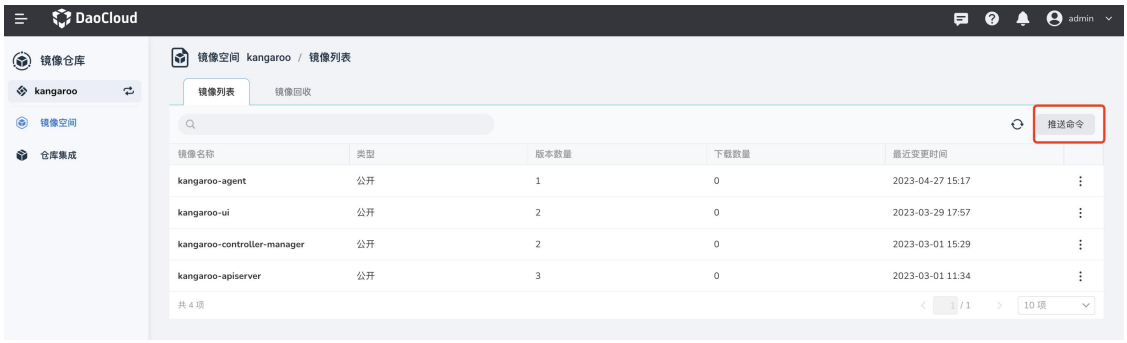
功能优势

- 安全可靠：通过镜像空间与工作空间绑定的方式实现细粒度的镜像分配，确保私有镜像只能被指定的工作空间拉取。
- 简单便捷：平台管理员创建托管 Harbor 或集成其他仓库后，若希望共享给所有工作空间及命名空间使用，只需要将镜像仓库公开即可。
- 灵活接入：除了平台级的仓库集成能力，在每个工作空间下还提供了仓库集成功能。集成后，该镜像仓库仅供该工作空间使用。
- 全局联动：应用工作台和容器管理针对镜像仓库提供了镜像选择器功能，在部署应用时可通过选择镜像按钮一键获取所有可见的镜像，快速完成部署。

推送命令

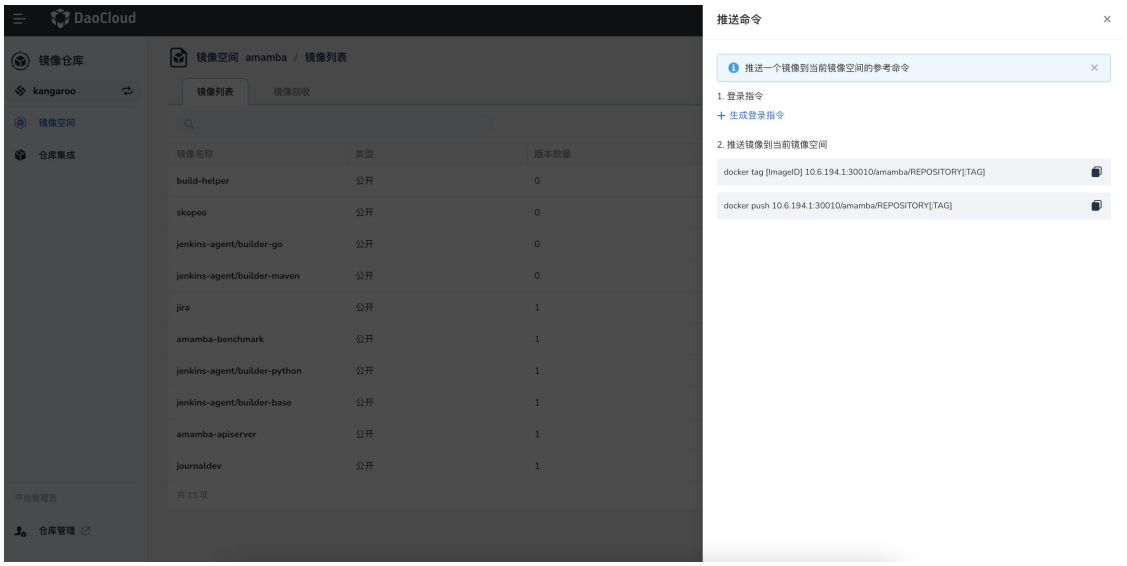
您可以推送一个镜像到当前镜像空间，并设置推送命令。

1. 在镜像列表页面上，点击右侧的推送命令按钮



点击按钮

2. 可以 + 生成登录指令后，查看推送镜像的命令。



推送命令

推送命令示例：

```
docker tag [ImageID] 10.6.194.1:30010/kangaroo/REPOSITORY[:TAG]
docker push 10.6.194.1:30010/kangaroo/REPOSITORY[:TAG]
```

下一步：Artifacts 和描述信息

镜像扫描

镜像被下载后可以直接使用，为用户提供了很多方便，但是不一定安全，可能被恶意植入后门。用户需要扫描下载的镜像，获知镜像安全信息。

在（DevOps）CI/CD 过程中，镜像被直接推送到镜像仓库，无法保证镜像的安全性，存在持续安全集成自动扫描的需求。

安全扫描是一种主动的防范措施，能有效避免黑客攻击行为，做到防患于未然，建议定时/手动扫描镜像。

（投产）容器到生产环境之后，生产环境对容器的安全性要求较高，需要容器的安全性得到保证，因此使用镜像运行容器前，一定要对镜像进行扫描，从而提高安全性。

最后扫描结果应提供更多有关纠正措施的指导。当用户收到容器镜像受到漏洞困扰的坏消息时，需要为自己找出修复方法，漏洞来自哪里？可以做些什么来解决问题?

镜像扫描特性

目前 DCE 5.0 的镜像仓库模块支持以下几种扫描镜像：

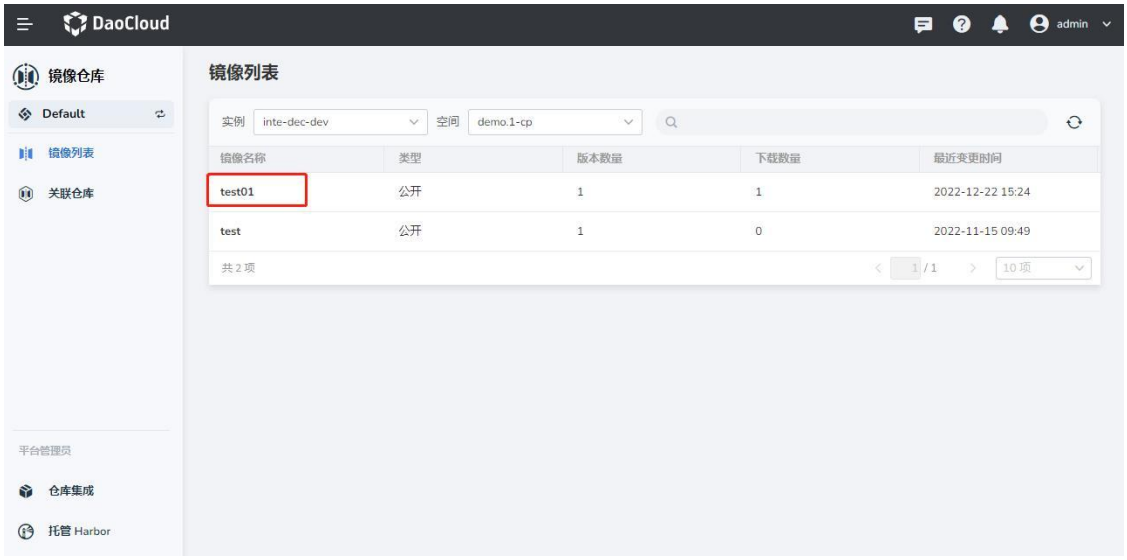
- 托管的 Harbor 仓库支持 Trivy 扫描。
- 原生的 Harbor 仓库支持 Clair 和 Trivy 扫描，具体取决于用户安装了什么插件。

用户在扫描镜像索引时，会同步扫描被索引的所有镜像，扫描结果为被索引镜像的扫描结果之和。

手动扫描镜像

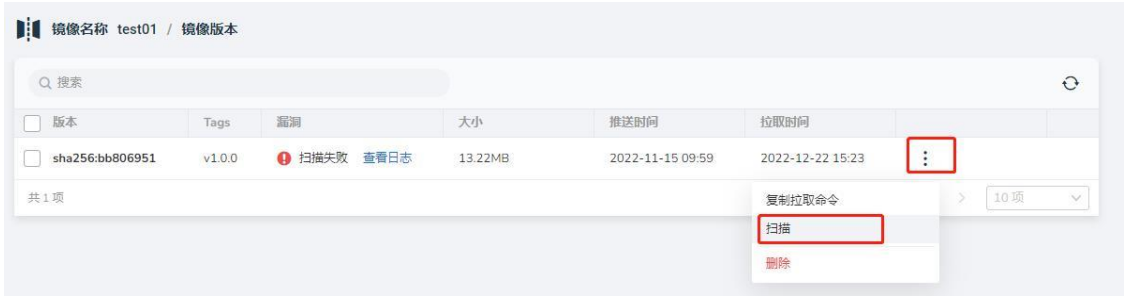
对于关联和集成的仓库，将出现在镜像列表中。您可以按需手动扫描某些镜像。

1. 进入镜像仓库的镜像列表中，选择一个实例和镜像空间，点击某一个镜像。



镜像列表

2. 在镜像详情列表中，点击列表右侧的 ⋮，在弹出菜单中选择扫描。



扫描

3. 系统开始扫描镜像，通常状态依次为排队中、扫描中、扫描完成。



排队中



扫描中

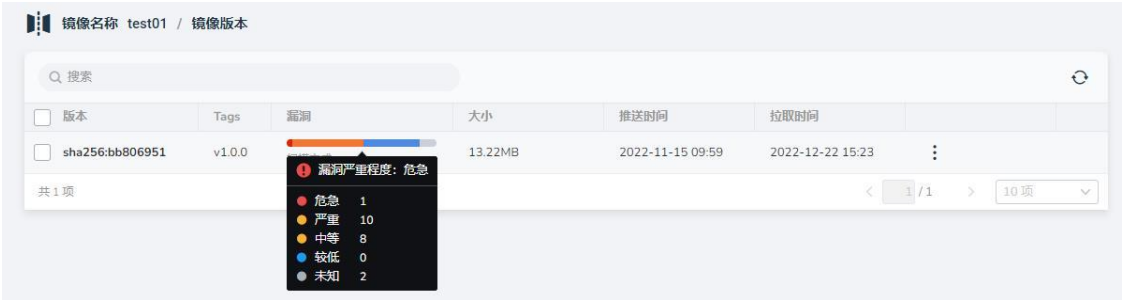


扫描完成

扫描状态包括：

- 未扫描：镜像从未被扫描过。
- 不支持：此镜像不支持扫描。
- 排队中：扫描任务已安排但尚未运行。
- 扫描中：扫描任务正在进行中，并显示进度条。
- 查看日志：扫描任务未能完成。点击查看日志以查看相关日志。
- 扫描完成：扫描任务成功完成。

4. 扫描完成后，将光标悬浮在扫描完成的比例条上，可以查看扫描详情。



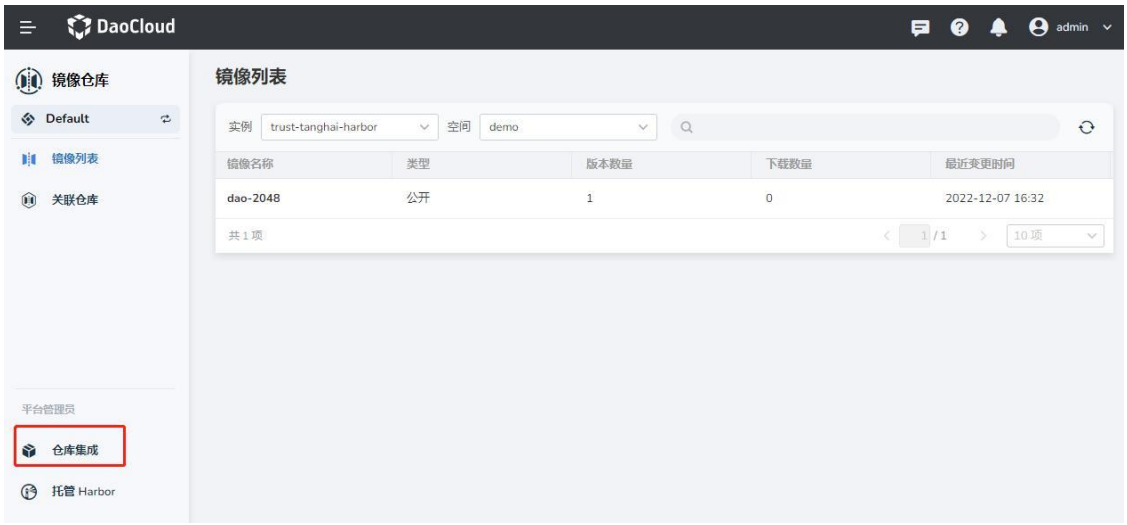
扫描完成

扫描原生 Harbor 镜像

集成的原生 Harbor 仓库，支持按 Clair 或 Trivy 进行扫描。

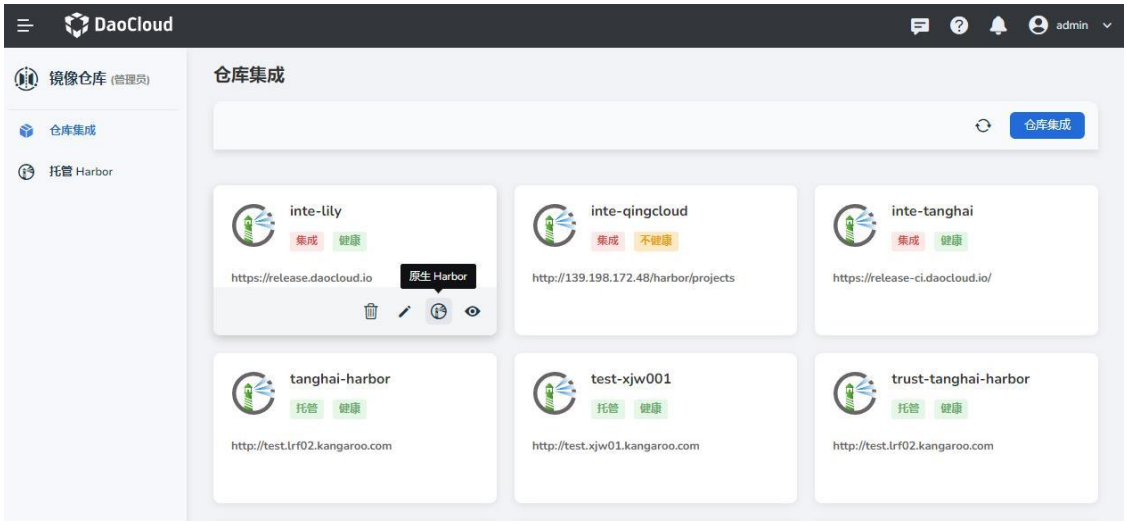
具体操作步骤为：

1. 以平台管理员登录镜像仓库，点击左侧底部的仓库集成。



仓库集成

2. 在集成的仓库列表中，光标悬浮于某个仓库上，点击原生 Harbor 图标。



仓库集成

3. 跳转到原生 Harbor，参阅扫描 Harbor 镜像。

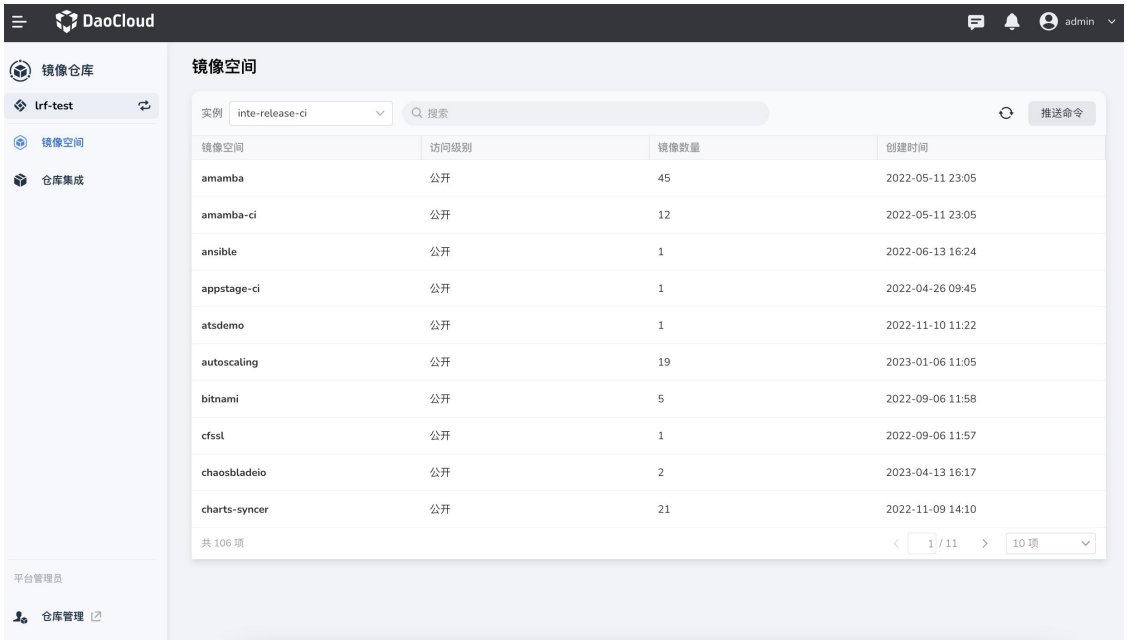
Artifacts 和描述信息

在普通用户视图中，您可以查看 Harbor 镜像空间的详情，包括 Artifacts 扫描结果和镜像的详细描述。您可以使用 Markdown 为每个镜像自定义描述信息。

Artifacts 详情

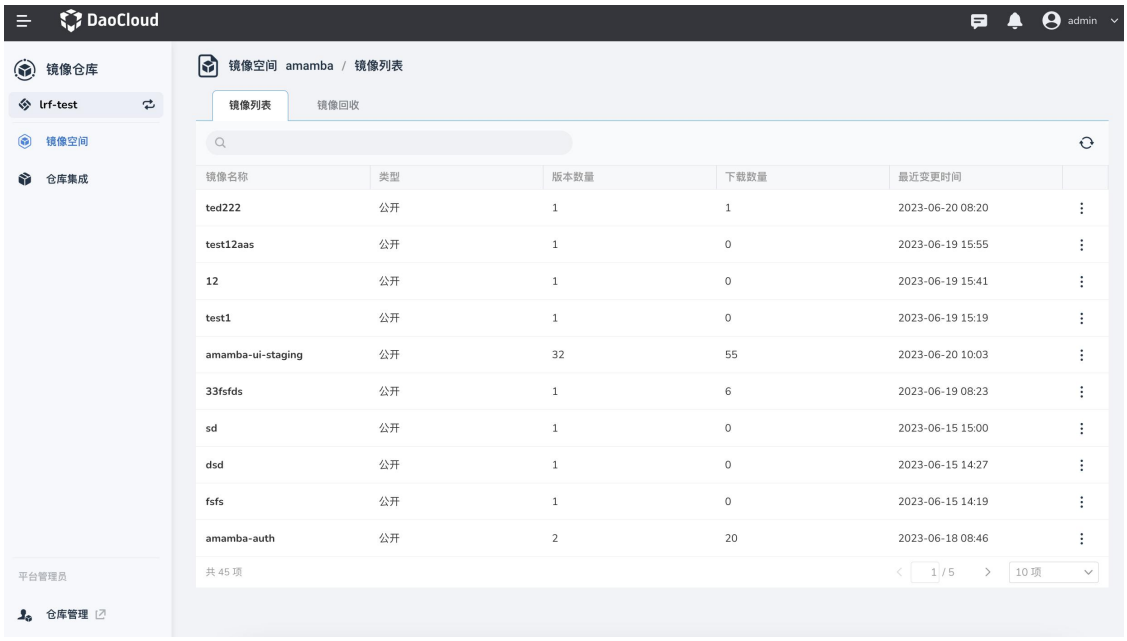
您可以参阅以下步骤，查看 Artifacts 的详情。

1. 选择一个 Harbor 实例，进入镜像空间页面，点击某个镜像空间的名称。



进入镜像空间

2. 进入镜像列表，点击某个镜像名称。



镜像列表

3. 默认显示 Artifacts 页签，点击某个版本，可以查看更多详情。



Artifacts

版本详情显示了当前 Artifact 的构建历史等信息。

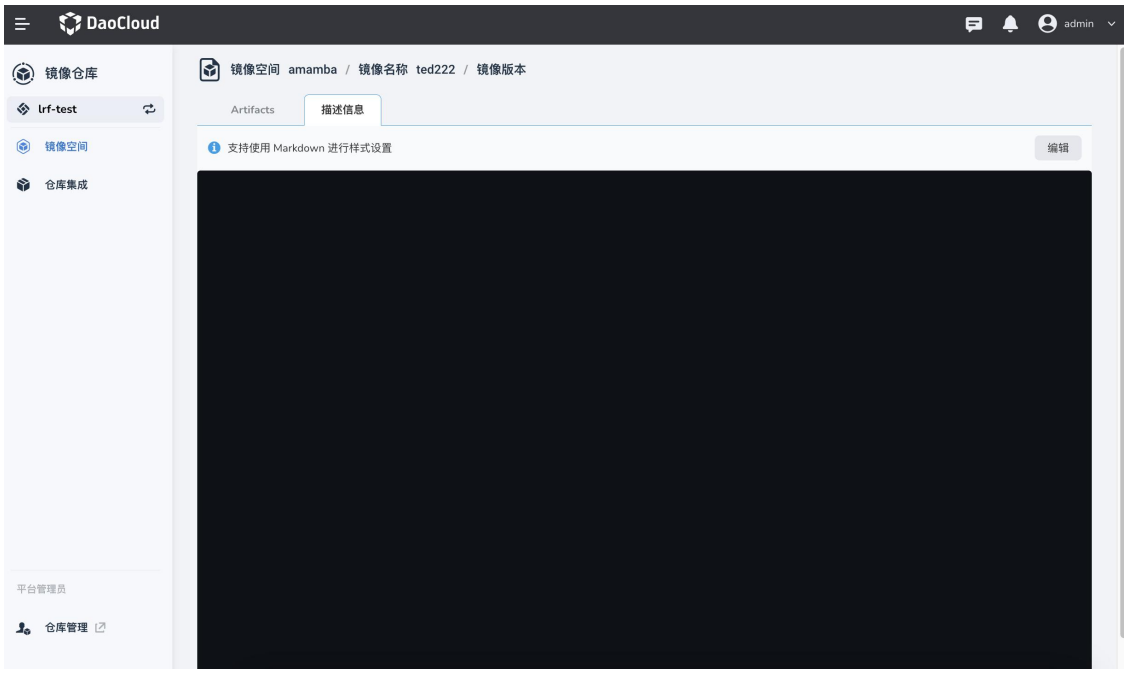


Artifacts

自定义描述信息

您可以参阅以下步骤，用 Markdown 自定义每个镜像的描述信息。

1. 接着上一节的步骤，点击描述信息页签，点击编辑按钮。



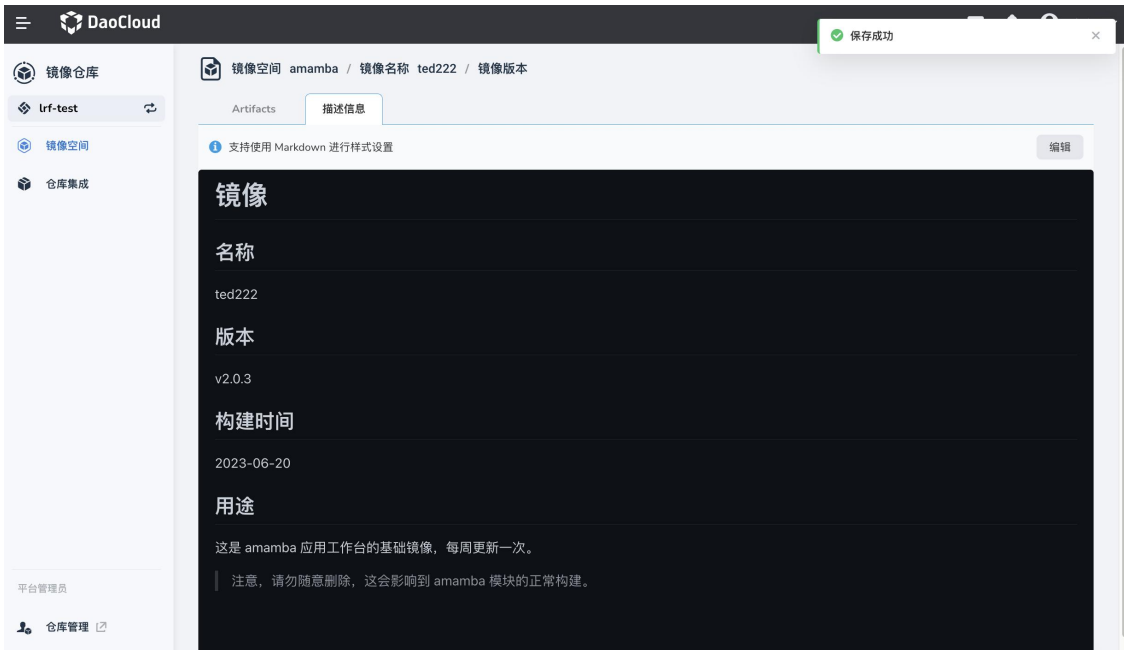
edit

2. 使用 Markdown 语法编写描述信息后，点击保存。



save

3. 最终效果如下图。



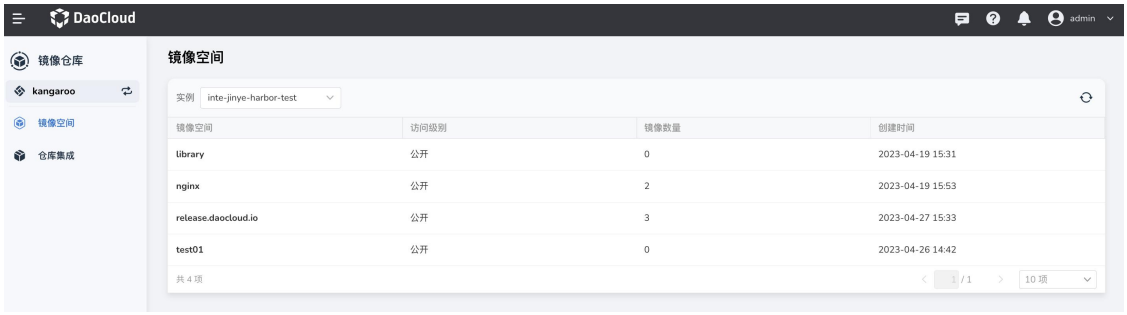
result

镜像空间

DCE 5.0 镜像仓库提供了基于镜像空间的镜像隔离功能。镜像空间分为公开和私有两种类型：

- 公开镜像仓库：所有用户都可以访问，通常存放公开的镜像，默认有一个公开镜像空间。
- 私有镜像仓库：只有授权用户才可以访问，通常存放镜像空间本身的镜像。

您可以选择不同的实例，查看其下的所有镜像空间。



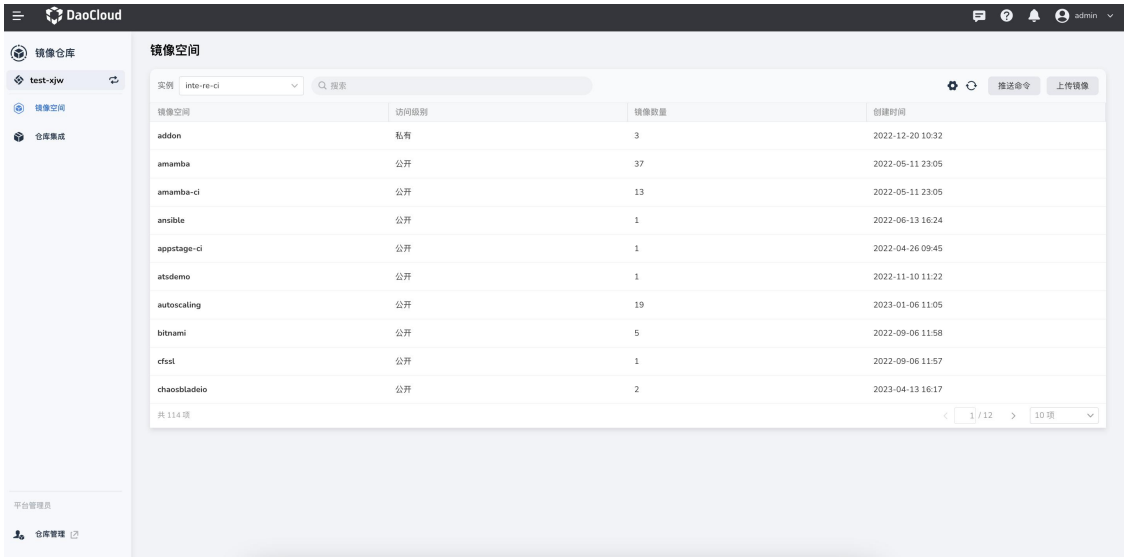
切换实例

点击某一个镜像空间的名称，可以查看当前空间内的镜像列表以及镜像回收规则。您可以为当前镜像空间创建镜像回收规则。所有镜像回收规则独立计算并且适用于所有符合条件的镜像，最多支持 15 条回收规则。



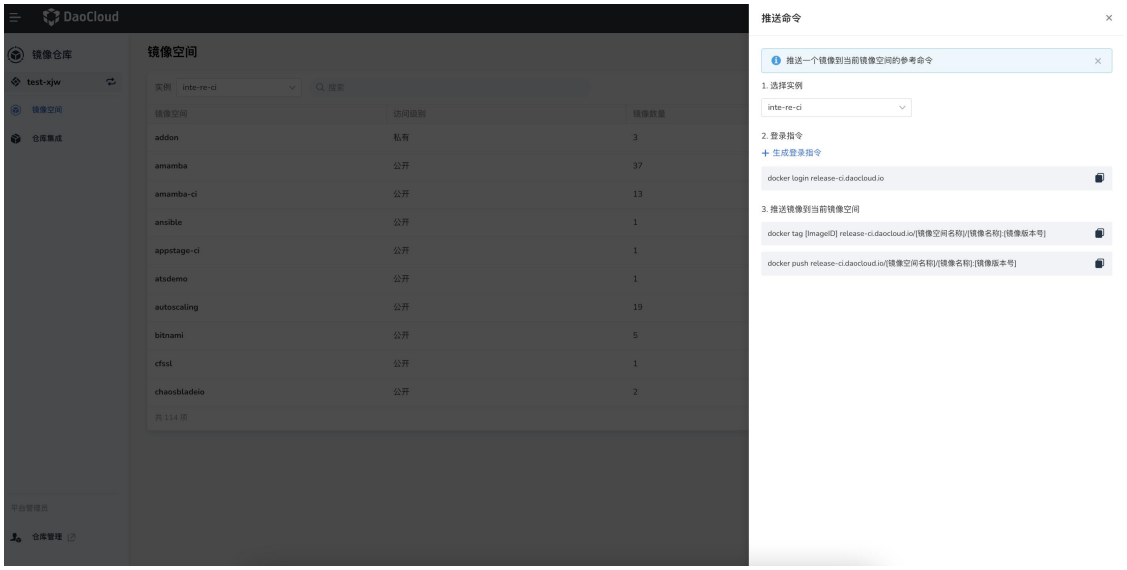
镜像空间包含什么

通过左侧导航栏的仓库集成(工作空间)进入镜像空间后，点击右侧的推送命令按钮。



按钮

可以查看将镜像推送到当前镜像空间的推送命令。



推送命令

点击上传镜像按钮，可以上传格式为 tar 或 tar.gz 的镜像文件。 注意不要超过 2GB，一个文件对应一个镜像，目前仅支持上传 1.11.2 及以上容器引擎客户端版本制作的镜像压缩包。



上传镜像

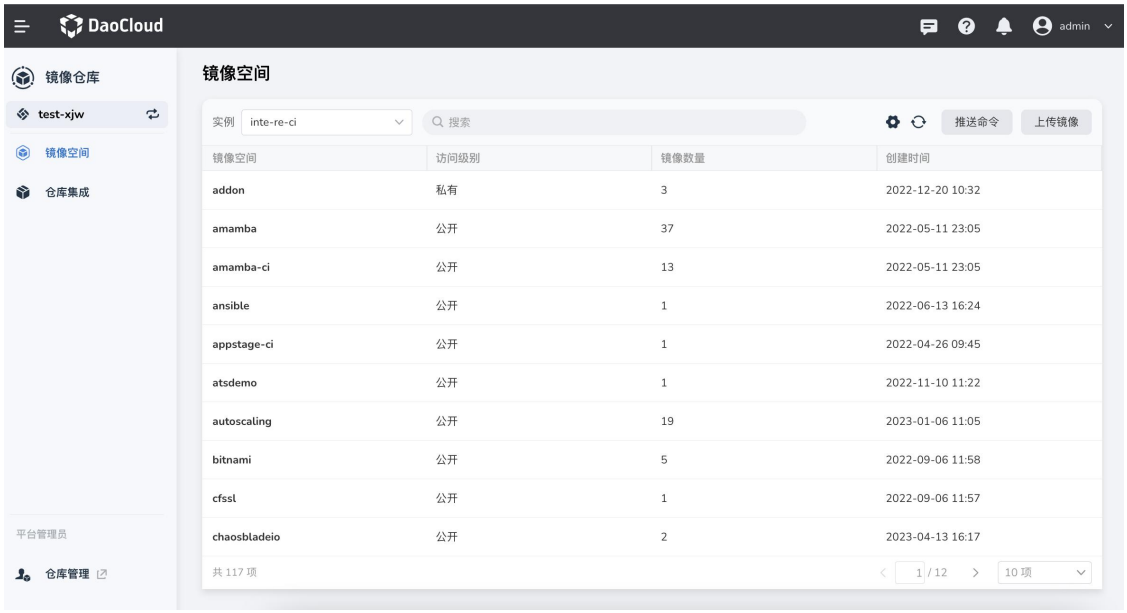
镜像回收

在镜像仓库中，许多镜像在一段时间或更新到一定版本后，旧版本的镜像可能不再需要，这些多余的镜像会消耗大量的存储容量。 如果您是 Workspace Admin，可对工作空间下所有镜像空间中的镜像进行管理。 您可以通过设置镜像回收规则来管理镜像仓库中的镜像，通过定时/手动的方式将一些镜像按照一定规则放入回收站。

这里的镜像回收指的是删除镜像并回收创建镜像所占用的资源。 当您不再需要一个镜像时，您可以将其删除，这将回收镜像占用的磁盘空间。 此过程被称为“镜像回收”。通过镜像回收，您可以释放磁盘空间以便在您的系统上运行其他操作，同时也可以保持系统的整洁和优化。

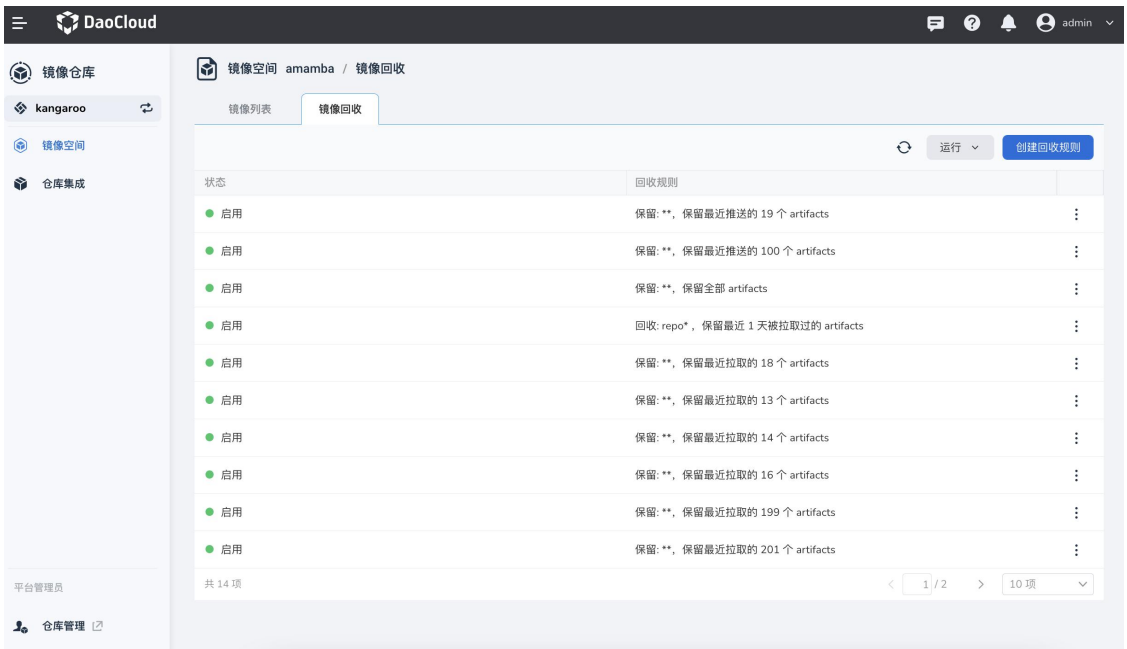
您可以为当前镜像空间创建镜像回收规则。 所有镜像回收规则独立计算并且适用于所有符合条件的镜像，目前 DCE 5.0 镜像仓库最多支持 15 条回收规则。

1. 使用具有 Workspace Admin 角色的用户登录 DCE 5.0，点击左侧导航栏镜像空间，点击列表中的某个名称。



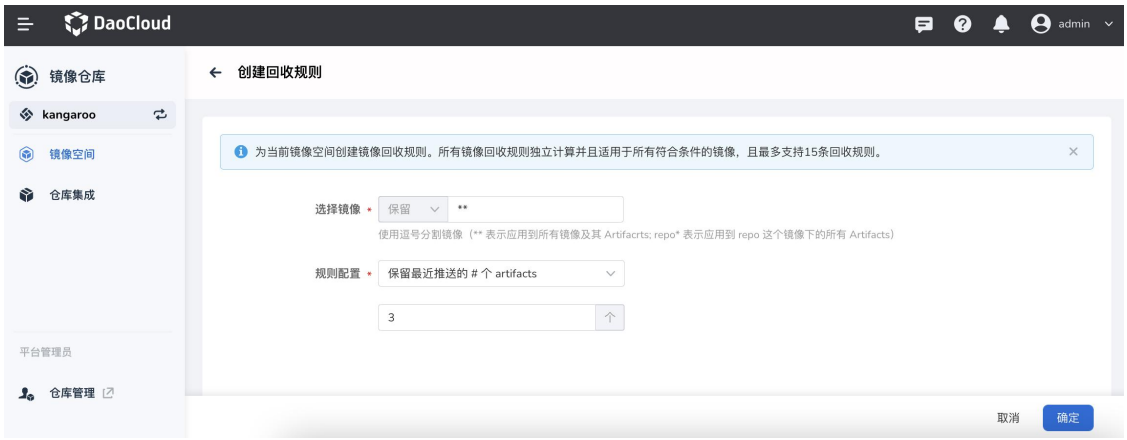
切换实例

2. 点击镜像回收页签，点击创建回收规则按钮（只有 Harbor 仓库才支持镜像回收）。



点击按钮

3. 按提示选择镜像，配置规则。



创建规则

4. 返回镜像回收列表，点击右侧的 ⋮，可以禁用、编辑或删除回收规则。

虚拟机容器 Release Notes

本页列出虚拟机容器的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-7-31

v0.1.0

新功能

- **新增** 支持通过集群展示虚拟机列表
- **新增** 支持通过容器镜像创建虚拟机
- **新增** 支持关机/启动虚拟机
- **新增** 支持删除虚拟机
- **新增** 支持通过控制台访问（VNC）虚拟机
- **新增** 支持查看虚拟机详情

什么是虚拟机容器 (Virtnest)

Virtnest 是一款强大的工具，通过容器镜像创建虚拟机（VM）。它能够在 Kubernetes 集群中快速部署和管理虚拟机， 并提供无缝控制虚拟机的启动、关机、配置等操作。

以下是使用 Virtnest 的大致步骤：

1. 构建所需的虚拟机容器镜像。
2. 将镜像推送到 Docker Registry 或其他镜像仓库中。
3. 使用镜像创建虚拟机实例。
4. 通过 VNC 或其他 Web 控制台方式访问虚拟机。
5. 查看虚拟机列表以及每个虚拟机的详细信息。
6. 执行虚拟机的创建、快照、关机、启动、SSH 访问、数据迁移等操作。

Virtnest 的优势

- 提升资源利用率：Virtnest 利用容器镜像作为虚拟机创建的基础，有效提高了资源利用率。 容器化的虚拟机可以独立运行应用和配置，同时共享主机内核和硬件资源，减少资源浪费。

- 快速部署和扩展：借助 Virtnest，在 k8s 集群中快速部署和扩展虚拟机。通过镜像和容器的集成， 减少了虚拟机启动时间，并实现根据需求动态调整虚拟机数量。这种灵活性提高了资源分配效率和系统性能。

- 简化管理和操作：Virtnest 简化在 k8s 集群中管理和操作虚拟机的流程。管理员可借助集群管理工具轻松管理虚拟机的启动、关机、配置和监控等任务。简化的管理过程降低了复杂性，减轻了管理负担。

- 灵活的应用部署：将虚拟机作为容器镜像部署在 k8s 集群中，为应用部署带来了极大的灵活性。 用户可以方便地将不同的应用和配置打包到容器镜

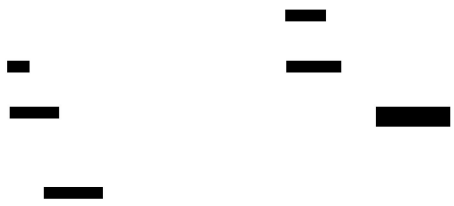
像中，并利用 Virtnest 快速部署到集群中。这种灵活性使应用部署更加敏捷和灵活。

- 与容器生态系统无缝集成：Virtnest 将虚拟机与容器相结合，无缝集成到 k8s 集群中已建立的容器生态系统中。 用户可以享受容器网络、服务发现、负载均衡等功能，并与其他容器化应用无缝协作。

借助 Virtnest，用户能够充分发挥 k8s 集群的潜力，高效管理虚拟化基础设施。我们致力于不断优化和完善 Virtnest 的容器化功能，为用户提供更好的虚拟机管理体验。

MirageDebug

MirageDebug：用于 Kubernetes 应用的本地远程调试工具，实现完全真实环境的调试。



MirageDebug

安装

安装 miragedebug

目前，安装 MirageDebug 需要 GO 运行环境，并可以使用以下方法进行安装：

```
go install github.com/miragedebug/miragedebug/cmd/mirage-debug@latest
```

使用方法

MirageDebug 服务器 - 后台服务

MirageDebug 服务器是一个后台服务，用于管理调试会话并提供有关调试会话的相关信息。

启动 MirageDebug 服务器

```
mirage-debug server
```

初始化调试应用程序

在项目根目录中执行以下命令，初始化调试应用程序，并按提示填写相关信息。

```
mirage-debug init
```

编写 IDE 配置文件

MirageDebug 可以自动生成不同 IDE 的调试配置文件，使本地调试变得简单。

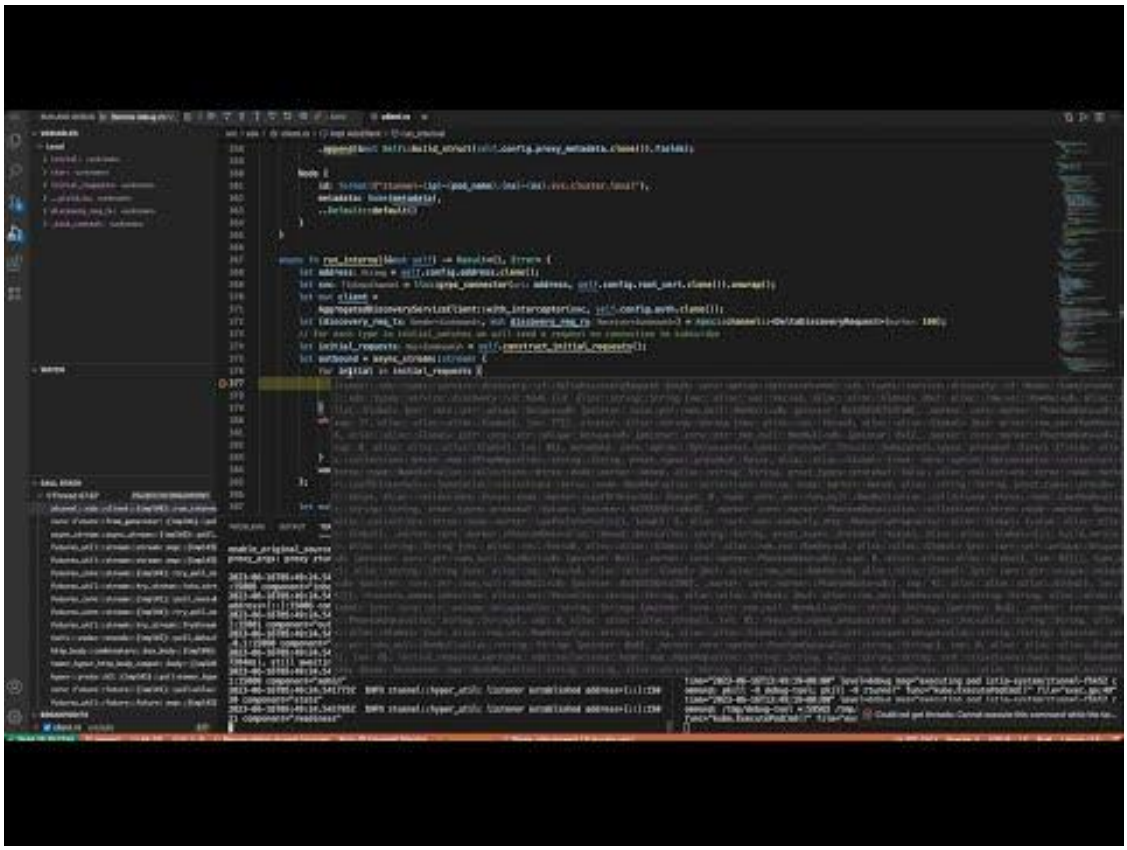
```
mirage-debug config <APPNAME>
```

开始调试

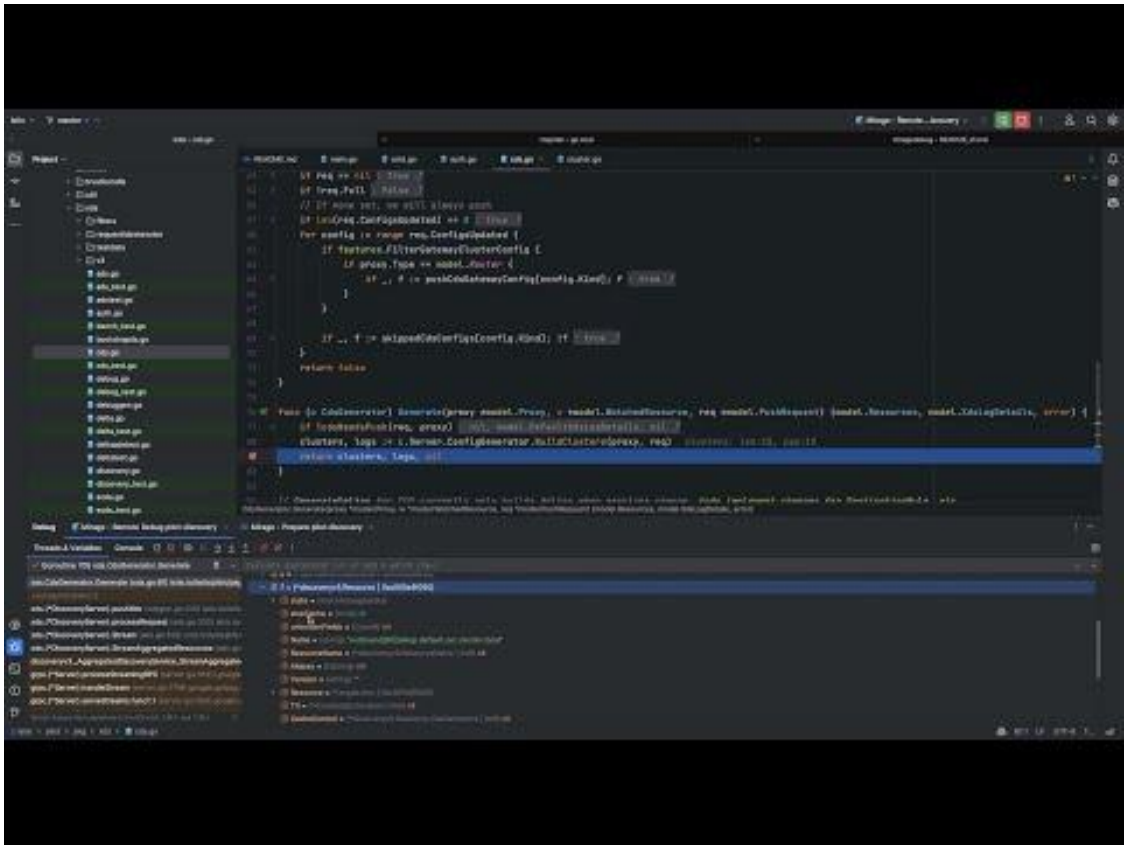
一旦配置好 IDE，就可以直接在 IDE 中开始调试了。

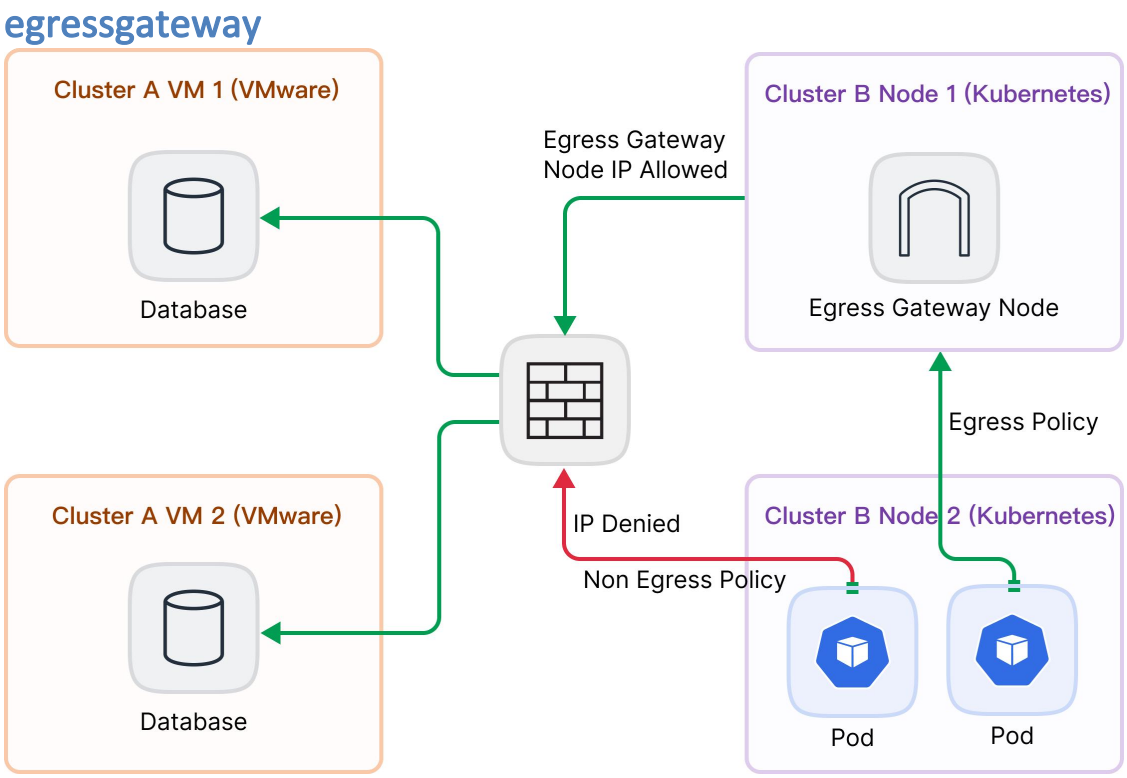
演示

在 Kubernetes 集群中使用 VSCode 调试 Rust 应用程序



在 Kubernetes 集群中使用 Goland 调试 istiod





background

从 2021 年开始，我们收到了以下反馈。

假设你有两个集群 A 和 B。集群 A 基于 VMWare，主要运行数据库工作负载，而集群 B 是一个 Kubernetes 集群。 集群 B 中的一些应用程序需要访问集群 A 中的数据库，网络管理员希望通过 Egress 网关来管理集群 Pod。

概述

该网关为 Kubernetes 集群提供网络 Egress 功能。

特点

- 解决 IPv4 IPv6 双栈连接问题
- 解决 Egress 节点的高可用性
- 允许过滤 Pods Egress 策略（*Destination CIDR*）

- 允许过滤 Egress 应用程序（*Pods*）
- 可在低内核版本中使用
- 支持多个 Egress 网关实例
- 支持命名空间 Egress IP
- 支持对 Egress 网关策略进行自动检测以获取集群流量
- 支持命名空间默认 Egress 实例

兼容性

- Calico

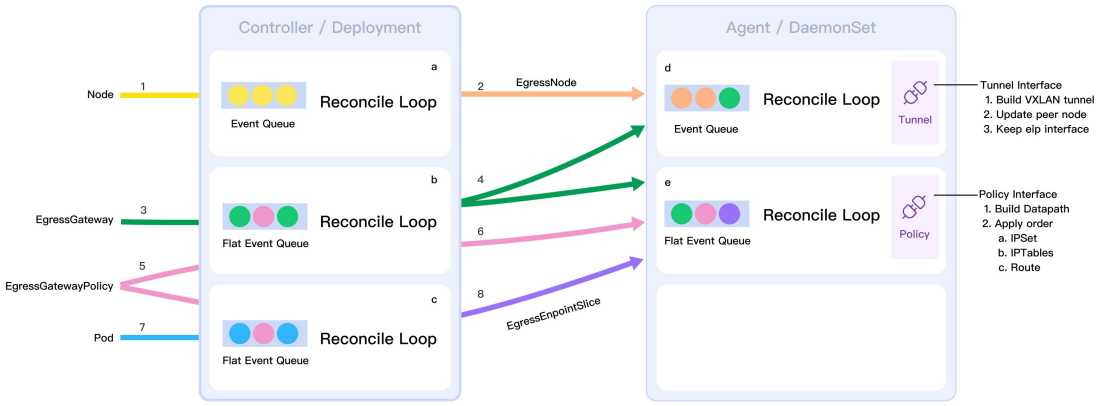
CRD

定义了一下 CRD：

- EgressNode
- EgressGateway
- EgressPolicy
- EgressClusterPolicy
- EgressEndpointSlice
- EgressClusterEndpointSlice
- EgressClusterInfo

你可以参考入门指南来搭建你的测试环境。

开发



develop

请参阅开发指南.

许可证

EgressGateway 在 Apache 许可证第 2 版下获得许可。有关完整的许可证文本， 请参见 LICENSE。

了解 egressgateway 社区

Ferry

Ferry 是一个 Kubernetes 多集群通信组件，旨在消除集群复杂环境的差异性，让用户就像在单集群一样通信，无需考虑多集群所在的网络环境等问题。

为什么需要 ferry

- 避免云厂商锁定
  - 多云互联互通
  - 多云无缝迁移
- 开箱即用
  - 提供命令行工具以方便安装和使用
  - 集中定义路由和穿透规则
- 无侵入
  - 不依赖 Kubernetes 版本
  - 不依赖任何 CNI 或网络环境
  - 无需修改现有环境
- 内网穿透
  - 只需要一个公共 IP

了解 ferry 社区

CloudTTY：一款专用于 Kubernetes 的 Cloud Shell Operator

CloudTTY 是专为 Kubernetes 云原生环境打造的 Web 终端和 Cloud Shell Operator。 通过 CloudTTY，您可以轻松用浏览器打开一个终端窗口，操控多云资源。

CloudTTY 意指云原生虚拟控制台，也称为 Cloud Shell（云壳）。 想象一下，在复杂多变的多云环境中，嵌入这样一层 Shell 来操纵整个云资源。 而这就是 CloudTTY 的设计初衷，我们希望在日趋复杂的 Kubernetes 容器云环境中，能够为开发者提供一个简单的 Shell 入口。

!!! info “TTY 的由来”

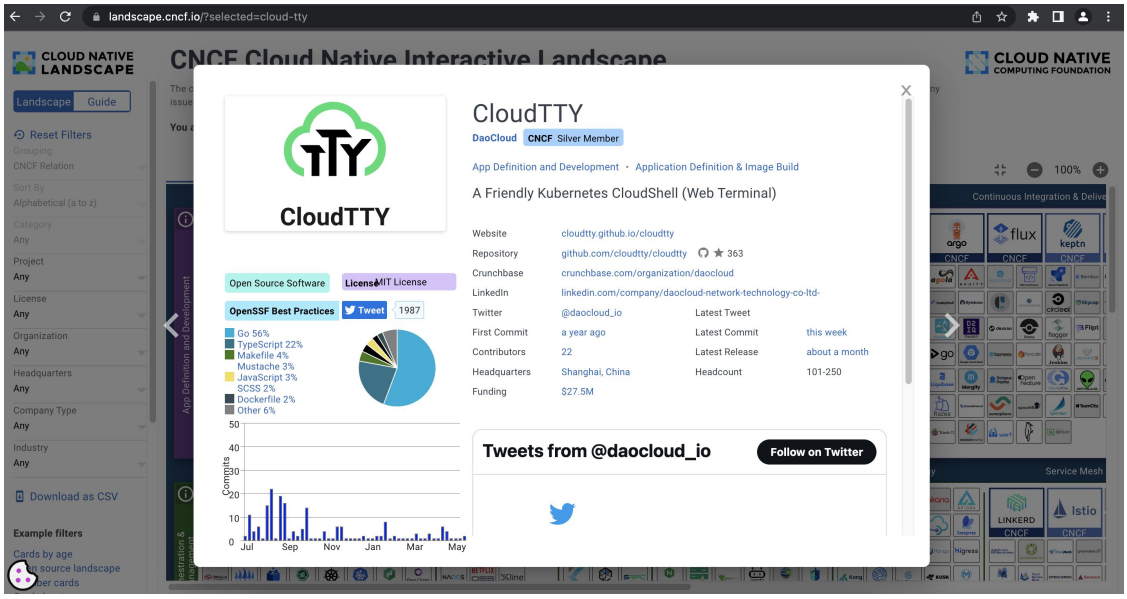
TTY 全称为 TeleTYpe，即电传打字机、电报机。
没错，就是那种很古老会发出滴滴答答声响、用线缆相连的有线电报机，那是一种只负责显示和打字的纯 IO 设备。
近些年随着虚拟化技术的飞速发展，TTY 常指虚拟控制台或虚拟终端。

为什么需要 CloudTTY?

目前，社区的 ttyd 等项目对 TTY 技术的探索已经达到了一定的深度，可以提供浏览器之上的终端能力。

但是在 Kubernetes 的场景下，这些 TTY 项目还需要更加云原生的能力拓展。如何让 ttyd 在容器内运行，如何通过 NodePort 等方式访问，如何用 CRD 的方式创建多个实例?

CloudTTY 提供了这些问题的解决方案，欢迎试用 CloudTTY ��！ CloudTTY 已入选 CNCF 全景图：



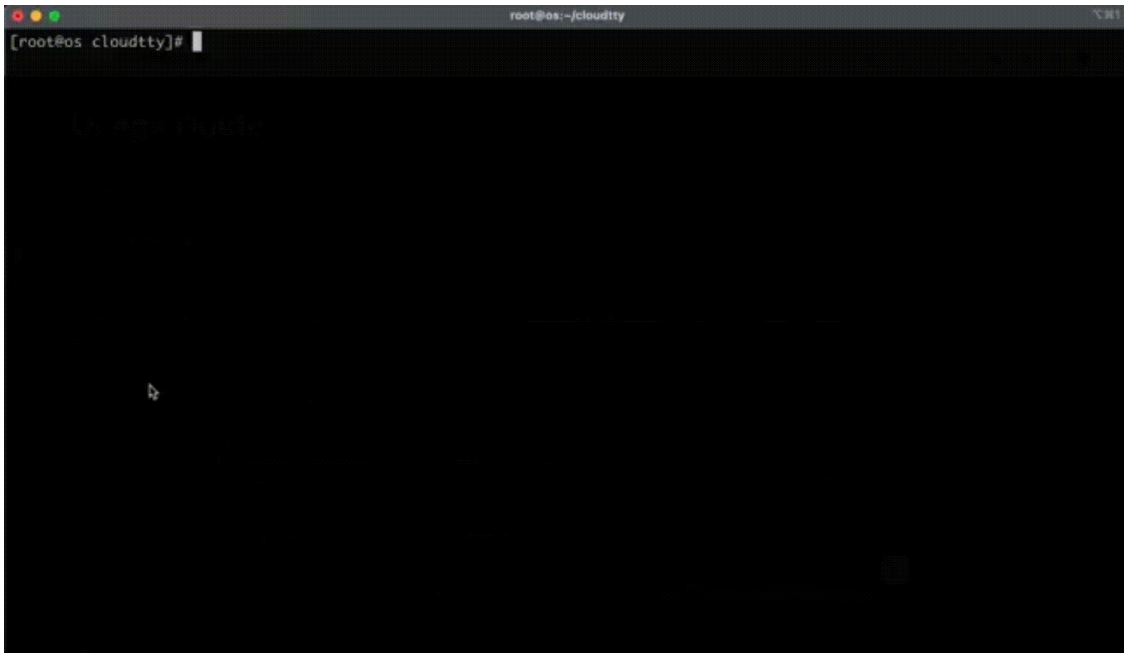
landscape

适用场景

CloudTTY 适用于以下几个场景。

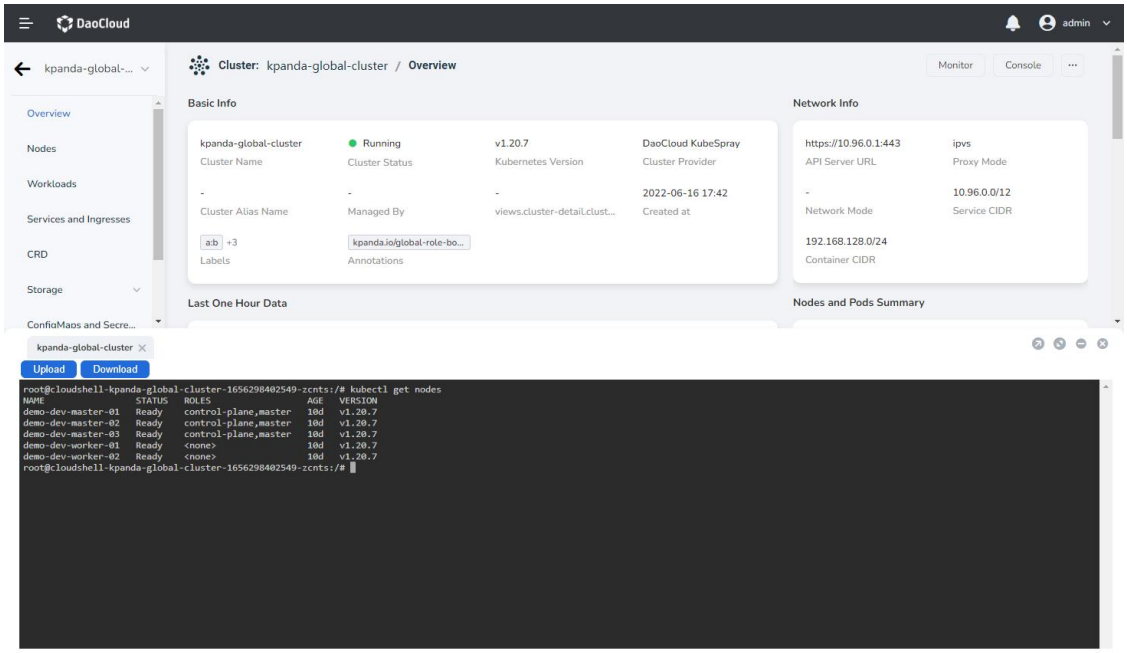
1. 如果企业正使用容器云平台来管理 Kubernetes，但由于安全原因，无法随意 SSH 到主机上执行 kubectl 命令，这就需要一种 Cloud Shell 能力。
2. 在浏览器网页上进入运行中的容器(kubectl exec)的场景。
3. 在浏览器网页上能够滚动展示容器日志的场景。

CloudTTY 的网页终端使用效果如下：



screenshot_gif

如果将 CloudTTY 集成到您自己的 UI 里面，最终效果 demo 如下:



demo_png

了解 CloudTTY 社区

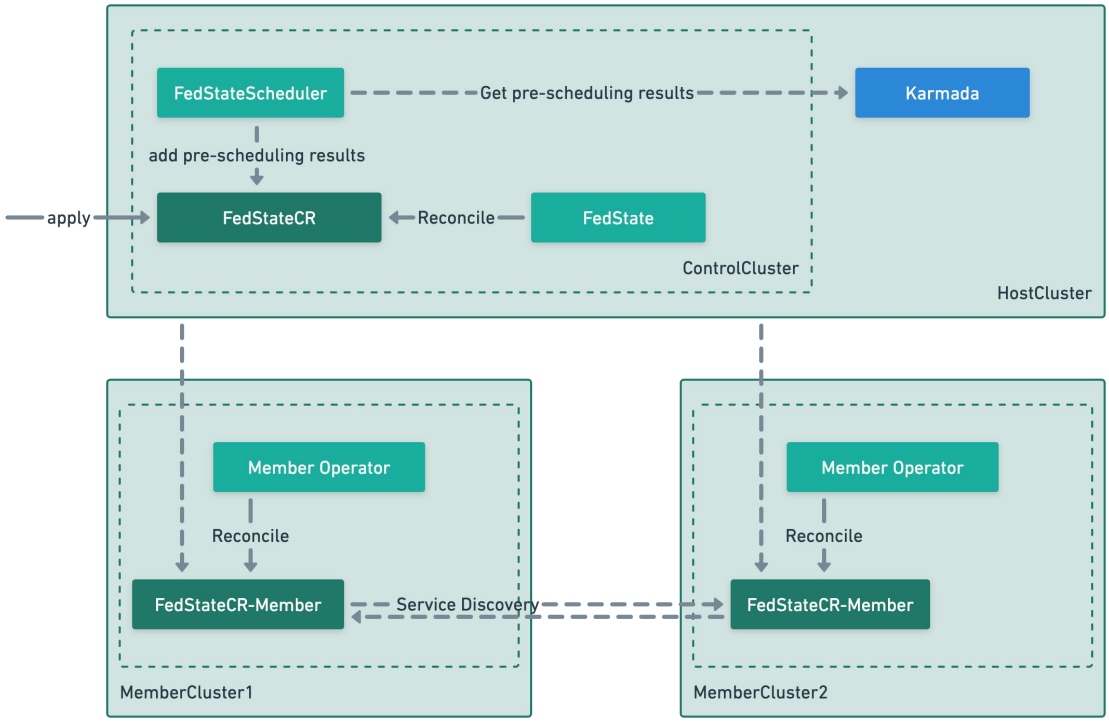
CloudTTY 已入选 CNCF 云原生全景图。

云原生联邦中间件 FedState

FedState 指的是 Federation Stateful Service，主要的设计目标是为了解决在多云、 多集群、多数据中心的场景下，有状态服务的编排、调度、部署和自动化运维等能力。

FedState 对需要部署在多云环境上的中间件、数据库等有状态的服务通过 Karmada 下发到各个成员集群，使其正常工作的同时并提供一些高级运维能力。

架构



架构图

组件说明：

- FedStateScheduler：多云有状态服务调度器，在 Karmada 调度器的基础上， 添加了一些与中间件服务相关的调度策略。

- FedState：多云有状态服务控制器主要负责按需配置各个管控集群与通过 Karmada 分发。
- Member Operator：表示部署在管控平面的有状态服务 Operator，FedState 内置了 Mongo Operator，后续会支持更多的有状态服务。
- FedStateCR：表示多云有状态服务实例的一个概念。
- FedStateCR-Member：表示多云有状态服务被下发到管控平面的实例。

参考文档

- 博文：介绍云原生联邦中间件 FedState
- Repo 地址: https://github.com/fedstate/fedstate

KLTS

KLTS 全称为 Kubernetes Long Term Support，主要使命是为 Kubernetes 早期版本提供长期免费的维护支持。

之所以需要维护早期版本，是因为在实际生产环境中，最新版本不一定是最好的，也不是最稳定的。 正常而言，Kubernetes 社区版本的维护周期只有一年左右，请参阅 Kubernetes 版本发行周期。 在社区停止维护后，KLTS 在接下来的三年内提供免费维护服务。

在实际生产中，为什么大多数企业选择采用早期的 Kubernetes 版本管控集群呢?

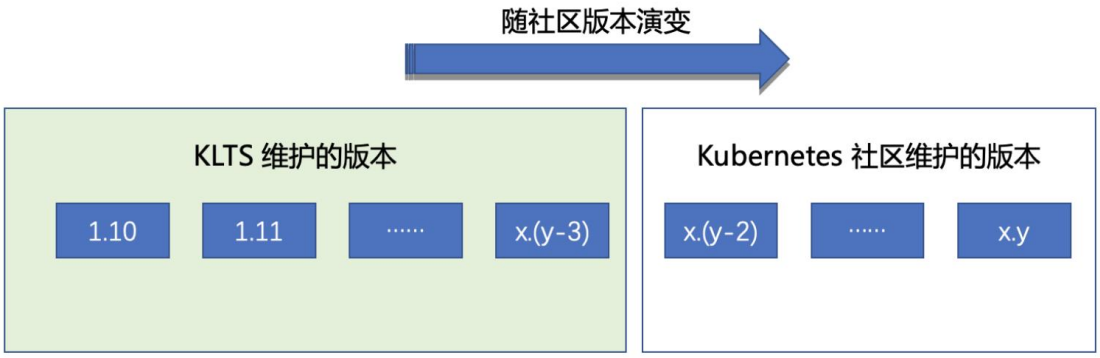
- 首先，升级频率高会带来变更风险，每次升级必须进行充分验证。 特别是金融行业的平台层变更周期通常比较长，因为一旦升级后的新版本存在 bug，就需要被迫回滚或快速响应升级至更新的版本，这样会造成不必要的成本支出。
- 其次，Kubernetes 升级后部分功能的替代方案还没有完全生产就绪，在生产环境中常会出现不兼容的状况。
- 最后，Kubernetes 社区仅支持小版本 +1 升级，不支持跨版本升级，因为跨版本升级经常会出现一些不可控的因素，造成更大的生产问题。

所以大多数企业的选择是沿用早期版本，不会贸然升级。 但 Kubernetes 社区只维护最新的 3 到 4 个版本，如何才能保证这些早期版本免受社区不定时发现的 CVE 漏洞和 bug 的袭扰呢? 这就是 KLTS 的价值所在！我们对早期版本提供长达 3 年的免费维护支持，积极修复早期版本的 CVE 安全漏洞和重大 bug。

KLTS 维护周期

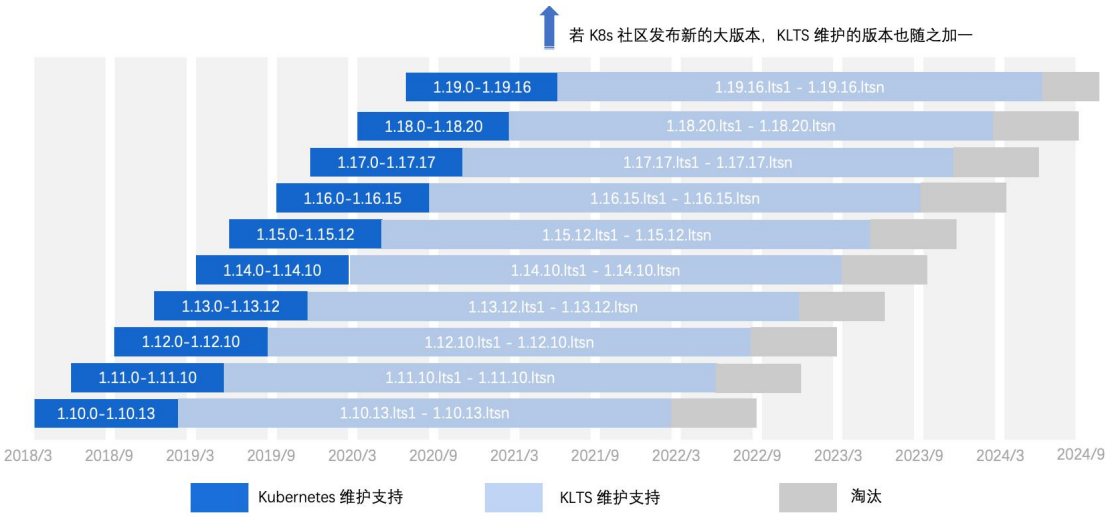
Kubernetes 版本号表示为 x.y.z，其中 x 是大版本号，y 是小版本号，z 是补丁版本，KLTS 提供的补丁版本号通常以 lts1、lts2 … ltsn 表示。为了方便表述，本节用前两位 x.y 描述 Kubernetes 版本号。

假设社区发布的最新 Kubernetes 版本为 x.y，根据社区版本维护声明，社区仅维护最近的三个版本，而 KLTS 目前维护从 1.10 起的近十个早期版本，如下图所示。



k8s 维护等版本

当 Kubernetes 社区发现可能影响生产的 CVE 新漏洞或 bug，受到影响的可能不止是社区正在维护的版本，还有之前已经停止维护、但企业仍在使用、且不能贸然升级的版本，KLTS 团队维护的正是这些社区放弃维护的版本。目前 KLTS 的版本维护周期如下：



klts 维护周期

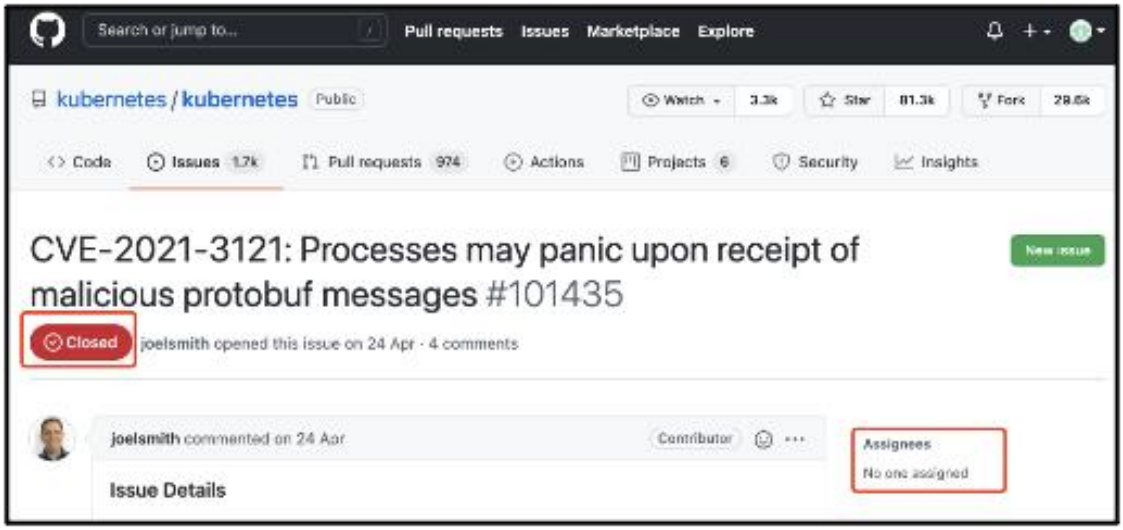
从上图可看出，Kubernetes 社区对某个版本的维护周期通常在一年左右，而 KLTS 可以在接下来的三年内提供长期维护，直至代码无法兼容，才会将相应版本淘汰。 进一步了解 Kubernetes 补丁发布周期。

KLTS 修复范围

有些高优先级的 CVE 或严重 Bug 存在于生产环境中会造成较大的安全隐患。CVE 安全问题是集群的生命线，KLTS 会优先修复中高级别的 CVE，其次会修复重大 Bug，确保生产环境稳定运行。

以 2021 年 1 月发现的 CVE-2021-3121 安全漏洞为例，CVSS 危急分数高达 7.5。 但截止 2021 年 9 月 Kubernetes 社区：

- 仅修复了 4 个版本：1.18、1.19、1.20、1.21
- 宣称“所有早期版本均有这个安全漏洞，建议用户立即停止使用早期版本”
- 拒绝修复早期版本漏洞的要求



cve 漏洞

KLTS 针对这一现状，默默修复了深受 CVE-2021-3121 安全漏洞影响的 8 个早期版本：

- v1.17.17
- v1.16.15
- v1.15.12
- v1.14.10
- v1.13.12
- v1.12.10
- v1.11.10
- v1.10.13

了解 KLTS 社区

查阅 KLTS 官网

Merbridge

Merbridge：使用 eBPF 加速您的服务网格，就像利用虫洞在网络世界中穿梭

只需要在 Istio 集群执行一条命令，即可直接使用 eBPF 代替 iptables 实现网络加速！

Merbridge 专为服务网格设计，使用 eBPF 代替传统的 iptables 劫持流量，能够让服务网格的流量拦截和转发能力更加高效。

eBPF (extended Berkeley Packet Filter) 技术可以在 Linux 内核中运行用户编写的程序，而且不需要修改内核代码或加载内核模块，目前广泛应用于网络、安全、监控等领域。相比传统的 iptables 流量劫持技术，基于 eBPF 的 Merbridge 可以绕过很多内核模块，缩短边车和服务间的数据路径，从而加速网络。Merbridge 没有对现有的 Istio 作出任何修改，原有的逻辑依然畅通。这意味着，如果您不想继续使用 eBPF，直接删除相关的 DaemonSet 就能恢复为传统的 iptables 方式，不会出现任何问题。

特性

Merbridge 的核心特性包括：

- 出口流量处理

  Merbridge 使用 eBPF 的 connect 程序，修改 user_ip 和 user_port 以改变连接发起时的目的地址，从而让流量能够发送到新的接口。为了让 Envoy 识别出原始目的地址，应用程序（包括 Envoy）会在收到连接之后调用 get_sockopts 函数，获取 ORIGINAL_DST。

- 入口流量处理

  入口流量处理与出口流量处理基本类似。需要注意的是，eBPF 是全局性的，不能在指定的命名空间生效。因此，如果对原本不是由 Istio 管理的

Pod 或者对外部的 IP 地址执行此操作，就会导致请求无法建立连接。为了解决此问题，Merbridge 设计了一个小的控制平面（以 DaemonSet 方式部署），通过 Watch 所有的 Pod，用类似于 kubelet 的方式获取当前节点的 Pod 列表，然后将已经注入 Sidecar 的 Pod IP 地址写入 local_pod_ips map。如果流量的目的地址不在该列表中，Merbridge 就不做处理，转而使用原来的逻辑。这样就可以灵活且便捷地处理入口流量。

- 同节点加速

  在 Istio 中，Envoy 使用当前 PodIP 加服务端口来访问应用程序。由于 PodIP 肯定也存在于 local_pod_ips ，所以请求就会被转发到 PodIP + 15006 端口。这样会造成无限递归，不能在 eBPF 获取当前命名空间的 IP 地址信息。因此，需要一套反馈机制：在 Envoy 尝试建立连接时仍然重定向到 15006 端口，在 sockops 阶段判断源 IP 和目的 IP 是否一致。如果一致，说明发送了错误的请求，需要在 sockops 丢弃该连接，并将当前的 ProcessID 和 IP 地址信息写入 process_ip map，让 eBPF 支持进程和 IP 的对应关系。下次发送请求时直接从 process_ip 表检查目的地址是否与当前 IP 地址一致。Envoy 会在请求失败时重试，且这个错误只会发生一次，后续的连接会非常快。

了解 Merbridge 社区

查阅 Merbridge 官网

Merbridge 已入选 CNCF 云原生全景图。

镜像加速站

很多镜像都在国外，比如 gcr。国内下载很慢，需要加速。 DaoCloud 为此提供了国内镜像加速，便于从国内拉取这些镜像。

使用方法

- 增加前缀（推荐）：

  k8s.gcr.io/coredns/coredns => m.daocloud.io/k8s.gcr.io/coredns/coredns

- 修改镜像仓库的前缀

  k8s.gcr.io/coredns/coredns => k8s-gcr.m.daocloud.io/coredns/coredns

支持的镜像列表

DaoCloud 目前收录了 600+ 国外镜像，方便国内用户拉取。 您可以随时提 PR，增加更多的镜像地址。

??? note “点击查看已收录的镜像”

```bash
docker.elastic.co/eck/eck-operator
docker.elastic.co/elasticsearch/elasticsearch
docker.elastic.co/kibana/kibana
docker.elastic.co/kibana/kibana-oss
docker.io/alpine
docker.io/alpine/helm
docker.io/amazon/aws-alb-ingress-controller
docker.io/amazon/aws-ebs-csi-driver
docker.io/apache/skywalking-java-agent
docker.io/apache/skywalking-oap-server
docker.io/apache/skywalking-ui
docker.io/aquasec/kube-bench
```

docker.io/aquasec/kube-hunter
docker.io/aquasec/trivy
docker.io/arey/mysql-client
docker.io/bitnami/bitnami-shell
docker.io/bitnami/contour
docker.io/bitnami/elasticsearch
docker.io/bitnami/elasticsearch-curator
docker.io/bitnami/elasticsearch-exporter
docker.io/bitnami/envoy
docker.io/bitnami/grafana
docker.io/bitnami/grafana-operator
docker.io/bitnami/kubeapps-apis
docker.io/bitnami/kubeapps-apprepository-controller
docker.io/bitnami/kubeapps-dashboard
docker.io/bitnami/kubeapps-kubeops
docker.io/bitnami/kubectl
docker.io/bitnami/kubernetes-event-exporter
docker.io/bitnami/mariadb
docker.io/bitnami/minideb
docker.io/bitnami/nginx
docker.io/bitnami/postgresql
docker.io/bitnami/wordpress
docker.io/bitpoke/mysql-operator
docker.io/bitpoke/mysql-operator-orchestrator
docker.io/bitpoke/mysql-operator-sidecar-5.7
docker.io/bitpoke/mysql-operator-sidecar-8.0
docker.io/busybox
docker.io/byrnedo/alpine-curl
docker.io/caddy
docker.io/calico/apiserver
docker.io/calico/cni
docker.io/calico/csi

docker.io/calico/kube-controllers
docker.io/calico/node
docker.io/calico/node-driver-registrar
docker.io/calico/pod2daemon-flexvol
docker.io/calico/typha
docker.io/cdkbot/hostpath-provisioner-amd64
docker.io/cdkbot/registry-amd64
docker.io/centos
docker.io/cfmanteiga/alpine-bash-curl-jq
docker.io/cfssl/cfssl
docker.io/cilium/json-mock
docker.io/clickhouse/clickhouse-server
docker.io/clickhouse/integration-helper
docker.io/cloudnativelabs/kube-router
docker.io/coredns/coredns
docker.io/csiplugin/snapshot-controller
docker.io/curlimages/curl
docker.io/datawire/ambassador
docker.io/datawire/ambassador-operator
docker.io/debian
docker.io/directxman12/k8s-prometheus-adapter
docker.io/docker
docker.io/elastic/filebeat
docker.io/envoyproxy/envoy
docker.io/envoyproxy/envoy-distroless
docker.io/envoyproxy/nighthawk-dev
docker.io/f5networks/f5-ipam-controller
docker.io/f5networks/k8s-bigip-ctlr
docker.io/fabulousjohn/kafka-manager
docker.io/falcosecurity/event-generator
docker.io/falcosecurity/falco-driver-loader
docker.io/falcosecurity/falco-exporter

docker.io/falcosecurity/falco-no-driver
docker.io/falcosecurity/falcosidekick
docker.io/falcosecurity/falcosidekick-ui
docker.io/fellah/gitbook
docker.io/flannelcni/flannel-cni-plugin
docker.io/flant/shell-operator
docker.io/fluent/fluent-bit
docker.io/fluent/fluentd
docker.io/fortio/fortio
docker.io/foxdalas/kafka-manager
docker.io/frrouting/frr
docker.io/goharbor/chartmuseum-photon
docker.io/goharbor/harbor-core
docker.io/goharbor/harbor-db
docker.io/goharbor/harbor-exporter
docker.io/goharbor/harbor-jobservice
docker.io/goharbor/harbor-portal
docker.io/goharbor/harbor-registryctl
docker.io/goharbor/nginx-photon
docker.io/goharbor/notary-server-photon
docker.io/goharbor/notary-signer-photon
docker.io/goharbor/redis-photon
docker.io/goharbor/registry-photon
docker.io/goharbor/trivy-adapter-photon
docker.io/golang
docker.io/grafana/grafana
docker.io/grafana/tempo
docker.io/halverneus/static-file-server
docker.io/haproxy
docker.io/honkit/honkit
docker.io/integratedcloudnative/ovn4nfv-k8s-plugin
docker.io/istio/citadel

docker.io/istio/examples-bookinfo-details-v1
docker.io/istio/examples-bookinfo-productpage-v1
docker.io/istio/examples-bookinfo-ratings-v1
docker.io/istio/examples-bookinfo-reviews-v1
docker.io/istio/examples-bookinfo-reviews-v2
docker.io/istio/examples-bookinfo-reviews-v3
docker.io/istio/examples-helloworld-v1
docker.io/istio/examples-helloworld-v2
docker.io/istio/galley
docker.io/istio/install-cni
docker.io/istio/kubectl
docker.io/istio/mixer
docker.io/istio/operator
docker.io/istio/pilot
docker.io/istio/proxyv2
docker.io/istio/sidecar_injector
docker.io/jaegertracing/all-in-one
docker.io/jaegertracing/jaeger-agent
docker.io/jaegertracing/jaeger-collector
docker.io/jaegertracing/jaeger-es-index-cleaner
docker.io/jaegertracing/jaeger-es-rollover
docker.io/jaegertracing/jaeger-operator
docker.io/jaegertracing/jaeger-query
docker.io/jaegertracing/spark-dependencies
docker.io/java
docker.io/jboss/keycloak
docker.io/jenkins/jnlp-slave
docker.io/jimmidyson/configmap-reload
docker.io/joosthofman/wget
docker.io/joseluisq/static-web-server
docker.io/jujusolutions/juju-db
docker.io/jujusolutions/jujud-operator

docker.io/k8scloudprovider/cinder-csi-plugin
docker.io/karmada/karmada-agent
docker.io/karmada/karmada-aggregated-apiserver
docker.io/karmada/karmada-controller-manager
docker.io/karmada/karmada-descheduler
docker.io/karmada/karmada-scheduler
docker.io/karmada/karmada-scheduler-estimator
docker.io/karmada/karmada-search
docker.io/karmada/karmada-webhook
docker.io/kedacore/keda
docker.io/kedacore/keda-metrics-apiserver
docker.io/kennethreitz/httpbin
docker.io/keyval/otel-go-agent
docker.io/kindest/base
docker.io/kindest/haproxy
docker.io/kindest/node
docker.io/kiwigrid/k8s-sidecar
docker.io/kubeedge/cloudcore
docker.io/kubeovn/kube-ovn
docker.io/kuberhealthy/dns-resolution-check
docker.io/kuberhealthy/kuberhealthy
docker.io/kubernetesui/dashboard
docker.io/kubernetesui/dashboard-amd64
docker.io/kubernetesui/metrics-scraper
docker.io/library/alpine
docker.io/library/busybox
docker.io/library/caddy
docker.io/library/centos
docker.io/library/debian
docker.io/library/docker
docker.io/library/golang
docker.io/library/haproxy

docker.io/library/java
docker.io/library/mariadb
docker.io/library/mysql
docker.io/library/nats-streaming
docker.io/library/nginx
docker.io/library/node
docker.io/library/openjdk
docker.io/library/percona
docker.io/library/perl
docker.io/library/phpmyadmin
docker.io/library/postgres
docker.io/library/python
docker.io/library/rabbitmq
docker.io/library/redis
docker.io/library/registry
docker.io/library/traefik
docker.io/library/ubuntu
docker.io/library/wordpress
docker.io/library/zookeeper
docker.io/longhornio/backing-image-manager
docker.io/longhornio/csi-attacher
docker.io/longhornio/csi-node-driver-registrar
docker.io/longhornio/csi-provisioner
docker.io/longhornio/csi-resizer
docker.io/longhornio/csi-snapshotter
docker.io/longhornio/longhorn-engine
docker.io/longhornio/longhorn-instance-manager
docker.io/longhornio/longhorn-manager
docker.io/longhornio/longhorn-share-manager
docker.io/longhornio/longhorn-ui
docker.io/mariadb
docker.io/merbridge/merbridge

docker.io/metallb/controller
docker.io/metallb/speaker
docker.io/minio/console
docker.io/minio/kes
docker.io/minio/logsearchapi
docker.io/minio/mc
docker.io/minio/minio
docker.io/minio/operator
docker.io/mirantis/k8s-netchecker-agent
docker.io/mirantis/k8s-netchecker-server
docker.io/mirrorgooglecontainers/defaultbackend-amd64
docker.io/mirrorgooglecontainers/hpa-example
docker.io/moby/buildkit
docker.io/multiarch/qemu-user-static
docker.io/mysql
docker.io/n8nio/n8n
docker.io/nacos/nacos-server
docker.io/nats-streaming
docker.io/neuvector/controller
docker.io/neuvector/enforcer
docker.io/neuvector/manager
docker.io/neuvector/scanner
docker.io/neuvector/updater
docker.io/nfvpe/multus
docker.io/nginx
docker.io/nginxdemos/hello
docker.io/node
docker.io/oamdev/cluster-gateway
docker.io/oamdev/kube-webhook-certgen
docker.io/oamdev/terraform-controller
docker.io/oamdev/vela-apiserver
docker.io/oamdev/vela-core

docker.io/oamdev/vela-rollout
docker.io/oamdev/velaux
docker.io/oliver006/redis_exporter
docker.io/openebs/admission-server
docker.io/openebs/linux-utils
docker.io/openebs/m-apiserver
docker.io/openebs/node-disk-manager
docker.io/openebs/node-disk-operator
docker.io/openebs/openebs-k8s-provisioner
docker.io/openebs/provisioner-localpv
docker.io/openebs/snapshot-controller
docker.io/openebs/snapshot-provisioner
docker.io/openjdk
docker.io/openpolicyagent/gatekeeper
docker.io/openstorage/stork
docker.io/openzipkin/zipkin
docker.io/osixia/openldap
docker.io/otel/demo
docker.io/otel/opentelemetry-collector
docker.io/otel/opentelemetry-collector-contrib
docker.io/percona
docker.io/perl
docker.io/phpmyadmin
docker.io/phpmyadmin/phpmyadmin
docker.io/portainer/portainer-ce
docker.io/postgres
docker.io/prom/alertmanager
docker.io/prom/mysqld-exporter
docker.io/prom/node-exporter
docker.io/prom/prometheus
docker.io/python
docker.io/rabbitmq

docker.io/rabbitmqoperator/cluster-operator
docker.io/rancher/helm-controller
docker.io/rancher/k3d-tools
docker.io/rancher/k3s
docker.io/rancher/kubectl
docker.io/rancher/local-path-provisioner
docker.io/redis
docker.io/redislabs/redisearch
docker.io/registry
docker.io/sonobuoy/cluster-inventory
docker.io/sonobuoy/kube-bench
docker.io/sonobuoy/sonobuoy
docker.io/sonobuoy/systemd-logs
docker.io/squidfunk/mkdocs-material
docker.io/swaggerapi/swagger-codegen-cli
docker.io/tgagor/centos-stream
docker.io/thanosio/thanos
docker.io/traefik
docker.io/ubuntu
docker.io/velero/velero
docker.io/victoriametrics/operator
docker.io/victoriametrics/vmagent
docker.io/victoriametrics/vmalert
docker.io/victoriametrics/vminsert
docker.io/victoriametrics/vmselect
docker.io/victoriametrics/vmstorage
docker.io/weaveworks/scope
docker.io/weaveworks/weave-kube
docker.io/weaveworks/weave-npc
docker.io/wordpress
docker.io/xueshanf/install-socat
docker.io/zenko/kafka-manager

docker.io/zookeeper
gcr.io/cadvisor/cadvisor
gcr.io/distroless/base
gcr.io/distroless/static
gcr.io/distroless/static-debian11
gcr.io/google-containers/pause
gcr.io/google.com/cloudsdktool/cloud-sdk
gcr.io/google_containers/hyperkube
gcr.io/heptio-images/ks-guestbook-demo
gcr.io/istio-release/app_sidecar_base_centos_7
gcr.io/istio-release/app_sidecar_base_centos_8
gcr.io/istio-release/base
gcr.io/istio-release/distroless
gcr.io/istio-release/iptables
gcr.io/istio-testing/app
gcr.io/istio-testing/build-tools
gcr.io/istio-testing/buildkit
gcr.io/istio-testing/dotdotpwn
gcr.io/istio-testing/ext-authz
gcr.io/istio-testing/fake-gce-metadata
gcr.io/istio-testing/fake-stackdriver
gcr.io/istio-testing/fuzz_tomcat
gcr.io/istio-testing/jwttool
gcr.io/istio-testing/kind-node
gcr.io/istio-testing/kindest/node
gcr.io/istio-testing/mynewproxy
gcr.io/istio-testing/myproxy
gcr.io/istio-testing/operator
gcr.io/istio-testing/pilot
gcr.io/istio-testing/proxyv2
gcr.io/k8s-staging-etcd/etcd
gcr.io/k8s-staging-gateway-api/admission-server

gcr.io/k8s-staging-kube-state-metrics/kube-state-metrics
gcr.io/k8s-staging-nfd/node-feature-discovery
gcr.io/k8s-staging-test-infra/krte
gcr.io/kaniko-project/executor
gcr.io/knative-releases/knative.dev/client/cmd/kn
gcr.io/knative-releases/knative.dev/eventing/cmd/apiserver_receive_adapter
gcr.io/knative-releases/knative.dev/eventing/cmd/controller
gcr.io/knative-releases/knative.dev/eventing/cmd/in_memory/channel_controller
gcr.io/knative-releases/knative.dev/eventing/cmd/in_memory/channel_dispatcher
gcr.io/knative-releases/knative.dev/eventing/cmd/mtbroker/filter
gcr.io/knative-releases/knative.dev/eventing/cmd/mtbroker/ingress
gcr.io/knative-releases/knative.dev/eventing/cmd/mtchannel_broker
gcr.io/knative-releases/knative.dev/eventing/cmd/mtping
gcr.io/knative-releases/knative.dev/eventing/cmd/webhook
gcr.io/knative-releases/knative.dev/net-istio/cmd/controller
gcr.io/knative-releases/knative.dev/net-istio/cmd/webhook
gcr.io/knative-releases/knative.dev/net-kourier/cmd/kourier
gcr.io/knative-releases/knative.dev/serving/cmd/activator
gcr.io/knative-releases/knative.dev/serving/cmd/autoscaler
gcr.io/knative-releases/knative.dev/serving/cmd/controller
gcr.io/knative-releases/knative.dev/serving/cmd/default-domain
gcr.io/knative-releases/knative.dev/serving/cmd/domain-mapping
gcr.io/knative-releases/knative.dev/serving/cmd/domain-mapping-webhook
gcr.io/knative-releases/knative.dev/serving/cmd/queue
gcr.io/knative-releases/knative.dev/serving/cmd/webhook
gcr.io/kuar-demo/kuard-amd64
gcr.io/kubebuilder/kube-rbac-proxy
gcr.io/kubecost1/cost-model
gcr.io/kubecost1/frontend
gcr.io/tekton-releases/github.com/tektoncd/dashboard/cmd/dashboard
gcr.io/tekton-releases/github.com/tektoncd/operator/cmd/kubernetes/operator
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/controller

gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/entrypoint
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/imagedigestexporter
gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/webhook
gcr.io/tekton-releases/github.com/tektoncd/results/cmd/api
gcr.io/tekton-releases/github.com/tektoncd/results/cmd/watcher
gcr.io/tekton-releases/github.com/tektoncd/triggers/cmd/controllers
gcr.io/tekton-releases/github.com/tektoncd/triggers/cmd/webhook
ghcr.io/aquasecurity/trivy
ghcr.io/aquasecurity/trivy-db
ghcr.io/clusterpedia-io/clusterpedia/apiserver
ghcr.io/clusterpedia-io/clusterpedia/clustersynchro-manager
ghcr.io/daocloud/ckube
ghcr.io/daocloud/dao-2048
ghcr.io/dependabot/dependabot-core
ghcr.io/dependabot/dependabot-core-development
ghcr.io/dexidp/dex
ghcr.io/dtzar/helm-kubectl
ghcr.io/ferryproxy/ferry/ferry-controller
ghcr.io/ferryproxy/ferry/ferry-tunnel
ghcr.io/fluxcd/helm-controller
ghcr.io/fluxcd/kustomize-controller
ghcr.io/fluxcd/notification-controller
ghcr.io/fluxcd/source-controller
ghcr.io/helm/chartmuseum
ghcr.io/hwameistor/admission
ghcr.io/hwameistor/apiserver
ghcr.io/hwameistor/drbd-reactor
ghcr.io/hwameistor/drbd9-bionic
ghcr.io/hwameistor/drbd9-focal
ghcr.io/hwameistor/drbd9-jammy
ghcr.io/hwameistor/drbd9-rhel7

ghcr.io/hwameistor/drbd9-rhel8
ghcr.io/hwameistor/drbd9-shipper
ghcr.io/hwameistor/evictor
ghcr.io/hwameistor/local-disk-manager
ghcr.io/hwameistor/local-storage
ghcr.io/hwameistor/scheduler
ghcr.io/hwameistor/self-signed
ghcr.io/jaredtan95/openinsight
ghcr.io/jaredtan95/tailing-sidecar
ghcr.io/jaredtan95/tailing-sidecar-operator
ghcr.io/k8snetworkplumbingwg/multus-cni
ghcr.io/k8snetworkplumbingwg/sriov-cni
ghcr.io/k8snetworkplumbingwg/sriov-network-device-plugin
ghcr.io/klts-io/kubernetes-lts/coredns
ghcr.io/klts-io/kubernetes-lts/etcd
ghcr.io/klts-io/kubernetes-lts/kube-apiserver
ghcr.io/klts-io/kubernetes-lts/kube-controller-manager
ghcr.io/klts-io/kubernetes-lts/kube-proxy
ghcr.io/klts-io/kubernetes-lts/kube-scheduler
ghcr.io/klts-io/kubernetes-lts/pause
ghcr.io/kube-vip/kube-vip
ghcr.io/kubean-io/kubean-operator
ghcr.io/kubean-io/kubespray
ghcr.io/kubean-io/spray-job
ghcr.io/open-telemetry/demo
ghcr.io/open-telemetry/opentelemetry-operator/autoinstrumentation-dotnet
ghcr.io/open-telemetry/opentelemetry-operator/autoinstrumentation-java
ghcr.io/open-telemetry/opentelemetry-operator/autoinstrumentation-nodejs
ghcr.io/open-telemetry/opentelemetry-operator/autoinstrumentation-python
ghcr.io/open-telemetry/opentelemetry-operator/opentelemetry-operator
ghcr.io/openfaas/basic-auth
ghcr.io/openfaas/faas-netes

ghcr.io/openfaas/gateway
ghcr.io/openfaas/queue-worker
ghcr.io/openinsight-proj/openinsight
ghcr.io/openinsight-proj/opentelemetry-demo-helm-chart/adservice
ghcr.io/openinsight-proj/opentelemetry-demo-helm-chart/sentinel
ghcr.io/projectcontour/contour
ghcr.io/pterodactyl/yolks
ghcr.io/scholzj/zoo-entrance
ghcr.io/spidernet-io/cni-plugins/meta-plugins
ghcr.io/spidernet-io/spiderdoctor-agent
ghcr.io/spidernet-io/spiderdoctor-controller
ghcr.io/spidernet-io/spiderpool/spiderpool-agent
ghcr.io/spidernet-io/spiderpool/spiderpool-base
ghcr.io/spidernet-io/spiderpool/spiderpool-controller
ghcr.io/sumologic/tailing-sidecar
ghcr.io/sumologic/tailing-sidecar-operator
quay.io/argoproj/argo-events
quay.io/argoproj/argo-rollouts
quay.io/argoproj/argocd
quay.io/argoproj/argocd-applicationset
quay.io/argoproj/argocli
quay.io/argoproj/argoexec
quay.io/argoproj/kubectl-argo-rollouts
quay.io/argoproj/workflow-controller
quay.io/argoprojlabs/argocd-image-updater
quay.io/brancz/kube-rbac-proxy
quay.io/calico/apiserver
quay.io/calico/cni
quay.io/calico/ctl
quay.io/calico/kube-controllers
quay.io/calico/node
quay.io/calico/pod2daemon-flexvol

quay.io/calico/typha
quay.io/cilium/certgen
quay.io/cilium/cilium
quay.io/cilium/cilium-etcd-operator
quay.io/cilium/cilium-init
quay.io/cilium/clustermesh-apiserver
quay.io/cilium/hubble-relay
quay.io/cilium/hubble-ui
quay.io/cilium/hubble-ui-backend
quay.io/cilium/json-mock
quay.io/cilium/operator
quay.io/cilium/operator-generic
quay.io/cilium/startup-script
quay.io/containers/skopeo
quay.io/coreos/etcd
quay.io/coreos/flannel
quay.io/datawire/ambassador-operator
quay.io/external_storage/cephfs-provisioner
quay.io/external_storage/local-volume-provisioner
quay.io/external_storage/nfs-client-provisioner
quay.io/external_storage/rbd-provisioner
quay.io/fluentd_elasticsearch/elasticsearch
quay.io/fluentd_elasticsearch/fluentd
quay.io/goswagger/swagger
quay.io/grafana-operator/grafana_plugins_init
quay.io/iovisor/bcc
quay.io/jaegertracing/jaeger-operator
quay.io/jetstack/cert-manager-cainjector
quay.io/jetstack/cert-manager-controller
quay.io/jetstack/cert-manager-ctl
quay.io/jetstack/cert-manager-webhook
quay.io/k8scsi/csi-attacher

quay.io/k8scsi/csi-node-driver-registrar
quay.io/k8scsi/csi-provisioner
quay.io/k8scsi/csi-resizer
quay.io/k8scsi/csi-snapshotter
quay.io/k8scsi/livenessprobe
quay.io/k8scsi/snapshot-controller
quay.io/keycloak/keycloak
quay.io/kiali/kiali
quay.io/kiwigrid/k8s-sidecar
quay.io/kubespray/kubespray
quay.io/l23network/k8s-netchecker-agent
quay.io/l23network/k8s-netchecker-server
quay.io/metallb/controller
quay.io/metallb/speaker
quay.io/minio/minio
quay.io/mongodb/mongodb-agent
quay.io/mongodb/mongodb-kubernetes-operator
quay.io/mongodb/mongodb-kubernetes-operator-version-upgrade-post-start-ho
ok
quay.io/mongodb/mongodb-kubernetes-readinessprobe
quay.io/nmstate/kubernetes-nmstate-handler
quay.io/nmstate/kubernetes-nmstate-operator
quay.io/opstree/redis
quay.io/opstree/redis-exporter
quay.io/opstree/redis-operator
quay.io/piraeusdatastore/drbd-reactor
quay.io/piraeusdatastore/drbd9-centos7
quay.io/piraeusdatastore/piraeus-client
quay.io/piraeusdatastore/piraeus-csi
quay.io/piraeusdatastore/piraeus-ha-controller
quay.io/piraeusdatastore/piraeus-operator
quay.io/prometheus-operator/prometheus-config-reloader

quay.io/prometheus-operator/prometheus-operator
quay.io/prometheus/alertmanager
quay.io/prometheus/blackbox-exporter
quay.io/prometheus/node-exporter
quay.io/prometheus/prometheus
quay.io/prometheuscommunity/elasticsearch-exporter
quay.io/spotahome/redis-operator
quay.io/strimzi/jmxtrans
quay.io/strimzi/kafka
quay.io/strimzi/kafka-bridge
quay.io/strimzi/kaniko-executor
quay.io/strimzi/maven-builder
quay.io/strimzi/operator
quay.io/submariner/submariner
quay.io/submariner/submariner-gateway
quay.io/submariner/submariner-globalnet
quay.io/submariner/submariner-networkplugin-syncer
quay.io/submariner/submariner-operator
quay.io/submariner/submariner-operator-index
quay.io/submariner/submariner-route-agent
quay.io/tigera/operator
registry.k8s.io/addon-resizer
registry.k8s.io/build-image/debian-iptables
registry.k8s.io/build-image/go-runner
registry.k8s.io/build-image/kube-cross
registry.k8s.io/cluster-api/cluster-api-controller
registry.k8s.io/cluster-api/kubeadm-bootstrap-controller
registry.k8s.io/cluster-api/kubeadm-control-plane-controller
registry.k8s.io/conformance
registry.k8s.io/coredns
registry.k8s.io/coredns/coredns
registry.k8s.io/cpa/cluster-proportional-autoscaler

registry.k8s.io/cpa/cluster-proportional-autoscaler-amd64
registry.k8s.io/cpa/cluster-proportional-autoscaler-arm64
registry.k8s.io/debian-base
registry.k8s.io/dns/k8s-dns-node-cache
registry.k8s.io/etcd
registry.k8s.io/etcd/etcd
registry.k8s.io/ingress-nginx/controller
registry.k8s.io/ingress-nginx/e2e-test-runner
registry.k8s.io/ingress-nginx/kube-webhook-certgen
registry.k8s.io/kube-apiserver
registry.k8s.io/kube-apiserver-amd64
registry.k8s.io/kube-controller-manager
registry.k8s.io/kube-controller-manager-amd64
registry.k8s.io/kube-proxy
registry.k8s.io/kube-proxy-amd64
registry.k8s.io/kube-registry-proxy
registry.k8s.io/kube-scheduler
registry.k8s.io/kube-scheduler-amd64
registry.k8s.io/kube-state-metrics/kube-state-metrics
registry.k8s.io/kueue/kueue
registry.k8s.io/kwok/kwok
registry.k8s.io/metrics-server
registry.k8s.io/metrics-server-amd64
registry.k8s.io/metrics-server/metrics-server
registry.k8s.io/metrics-server/metrics-server-amd64
registry.k8s.io/nfd/node-feature-discovery
registry.k8s.io/node-problem-detector/node-problem-detector
registry.k8s.io/node-test
registry.k8s.io/node-test-amd64
registry.k8s.io/pause
registry.k8s.io/prometheus-adapter/prometheus-adapter
registry.k8s.io/sig-storage/csi-attacher

registry.k8s.io/sig-storage/csi-node-driver-registrar
registry.k8s.io/sig-storage/csi-provisioner
registry.k8s.io/sig-storage/csi-resizer
registry.k8s.io/sig-storage/csi-snapshotter
registry.k8s.io/sig-storage/livenessprobe
registry.k8s.io/sig-storage/local-volume-provisioner
registry.k8s.io/sig-storage/nfs-subdir-external-provisioner
registry.k8s.io/sig-storage/snapshot-controller
registry.opensource.zalan.do/acid/logical-backup
registry.opensource.zalan.do/acid/pgbouncer
registry.opensource.zalan.do/acid/postgres-operator
registry.opensource.zalan.do/acid/spilo-14
```

了解镜像社区
```

开源社区

!!! info

「DaoCloud 道客」开源生态启航!

- :fontawesome-solid-tty:{ .lg .middle } **CloudTTY** :simple-cncf: 已入选 CNCF Landscape

---

迄今为止，社区首个开源的 Kubernetes 网页版控制台，非常好用 ��

:octicons-arrow-right-24: 了解更多

- :simple-wikipedia:{ .lg .middle } **Clusterpedia** :simple-cncf: 已入选 CNCF Sandbox 和 Landscape

---

Kubernetes 多集群资源一站式百科全书，已入选 CNCF 沙箱孵化，**中国移动** 等已部署至生产

:octicons-arrow-right-24: 了解更多

- :simple-amazonapigateway:{ .lg .middle } **egressgateway**

为 Kubernetes 集群提供网络 Egress 功能的一种网关

:octicons-arrow-right-24: 了解更多

- :material-run-fast:{ .lg .middle } **Fast**

基于 eBPF 实现的 Kubernetes CNI。目前已实现了以下两个功能：

- 基于 eBPF 的网络基本连通性
- 管理集群容器的 IP 地址

:octicons-arrow-right-24: 了解更多

- :fontawesome-brands-trade-federation:{ .lg .middle } **FedState**

赋能多云、多集群、多数据中心场景下的自动运维

:octicons-arrow-right-24: 了解更多

- :material-ferry:{ .lg .middle } **Ferry**

Kubernetes 多集群通信组件，消除多集群复杂度，让您就像管理一个单集群

:octicons-arrow-right-24: 了解更多

- :material-storage-tank:{ .lg .middle } **HwameiStor** :simple-cncf: 已入选 CNCF Sandbox 和 Landscape

---

高可用的本地存储方案，更快、更强、更可靠，已入选 CNCF 全景图和沙箱孵化

:octicons-arrow-right-24: 了解更多

- :material-doctor:{ .lg .middle } **kdoctor**

通过压力注入的方式，实现对集群功能、性能的主动式巡检

:octicons-arrow-right-24: 了解更多

- :simple-kubernetes:{ .lg .middle } **KLTS**

对 k8s（最新版本 - 0.03）等 10 多个版本的持续维护

:octicons-arrow-right-24: 了解更多

- :peanuts:{ .lg .middle } **Kubean** :simple-cncf: 已入选 CNCF Landscape

---

容器化集群的全生命周期管理工具，正在申请 Sandbox

:octicons-arrow-right-24: 了解更多

- :material-forest:{ .lg .middle } **KWOK** :simple-cncf: K8s-sigs 项目之一

以最小资源模拟成千上万的 kubelet，已被 OpenAI、苹果、IBM、阿里云等用于测试环境

:octicons-arrow-right-24: 了解更多

- :fontawesome-solid-bridge:{ .lg .middle } **Merbridge** :simple-cncf: 已入选 CNCF Sandbox 和 Landscape

---

使用 eBPF 加速您的服务网格，就像利用虫洞在网络世界中穿梭，已入选 CNCF 全景图和 Sandbox

:octicons-arrow-right-24: 了解更多

- :material-debug-step-over:{ .lg .middle } **miragedebug**

在本地远程调试 K8s 应用，实现完全真实环境的调试

:octicons-arrow-right-24: 了解更多

- :spider:{ .lg .middle } **Spiderpool** :simple-cncf: 已入选 CNCF Landscape

---

云原生网络 IPAM 自动化管理软件，建议作为插件用于 Underlay CNI，已入选 CNCF Landscape

:octicons-arrow-right-24: 了解更多

- :speedboat:{ .lg .middle } **公开镜像加速**

很多镜像都在国外，比如 gcr。国内下载很慢，需要加速。

:octicons-arrow-right-24: 了解更多

!!! success

上述开源项目只是「DaoCloud 道客」技术的冰山一角，更多项目请拭目以待！

世界因技术而精彩，技术因分享而进步。

Think Different, Think Open Source!

Clusterpedia、HwameiStor 和 Merbridge 已入选 CNCF 沙箱。

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

KWOK (Kubernetes without Kubelet)



kwok

KWOK 是 DaoCloud 自主开源并被 Kubernetes 社区接纳和管理一个开源项目，目前 star 数 1600+，已被苹果、IBM、阿里云、腾讯云等运用于测试等场景。

KWOK 可以在几秒钟内搭建由数千个节点组成的集群。 在此场景下，所有模拟节点与真实的节点行为一样，所以整体上占用的资源很少，你可以轻松在一台笔记本电脑实现这点。

到目前为止，提供了两个工具：

- **kwok:** 模拟数千个假节点。
- **kwokctl:** 一个 CLI，用于简化创建和管理 KWOK 模拟的集群。

请参阅 https://kwok.sigs.k8s.io/ 了解更深入的信息。



管理集群

社区、讨论、贡献和支持

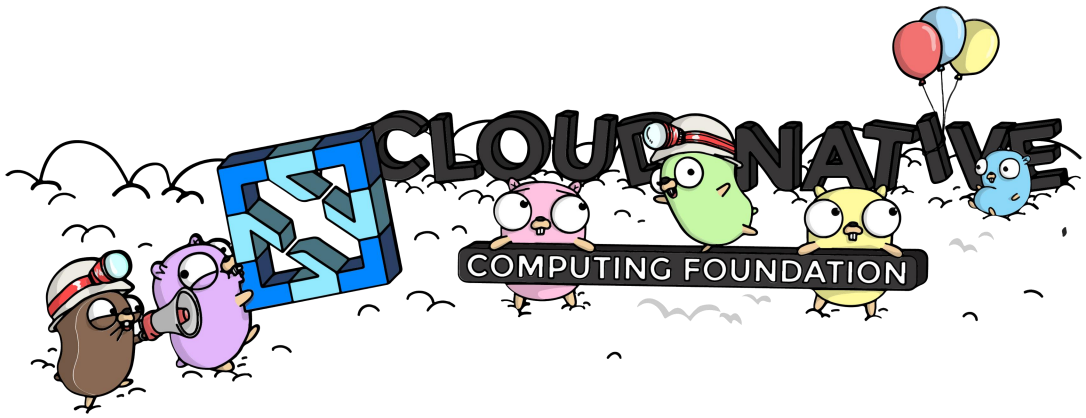
可以通过以下方式联系该项目的维护者：

- Slack
- 邮件列表

了解 KWOK 社区 KWOK 博文

本地存储解决方案

Hwameistor 是针对云原生有状态负载构建的高可用本地存储系统，隶属于 CNCF 全景图 -> 云原生存储 -> 运行时。



cncf gophers

HwameiStor 是一个 CNCF Sandbox 项目。

HwameiStor 是一款 Kubernetes 原生的容器附加存储（CAS）解决方案，将 HDD、SSD 和 NVMe 磁盘形成本地存储资源池进行统一管理，使用 CSI 架构提供分布式的本地数据卷服务，为有状态的云原生应用或组件提供数据持久化能力。



系统架构

HwameiStor 有 5 个组件：

- 本地磁盘管理器（Local Disk Manager, LDM）：简化管理节点上的磁盘，将磁盘抽象成一种可以被管理和监控的资源。这是一种 DaemonSet 对象，集群中每一个节点都会运行该服务，通过该服务检测存在的磁盘并将其转换成相应的 LocalDisk 资源。
- 本地存储（Local Storage, LS）旨在为应用提供高性能的本地持久化 LVM 存储卷。
- 调度器（scheduler）自动将 Pod 调度到配有 HwameiStor 存储卷的正确节点。使用调度器后，Pod 不必再使用 NodeAffinity 或 NodeSelector 字段来选择节点。调度器能处理 LVM 和 Disk 存储卷。
- 准入控制器（admission-controller）是一个 Webhook，可以自动验证哪个 Pod 使用 HwameiStor 卷，并将 schedulerName 修改为 hwameistor-scheduler。
- DRBD 安装器：通过 Linux 内核模块和相关脚本构成，用以构建高可用性的集群。其实现方式是通过网络来镜像整个设备，可以把它看作是一种网络 RAID。

了解 HwameiStor 社区

查阅 HwameiStor 官网

HwameiStor 位列 CNCF 云原生全景图。

kdoctor

kdoctor 是一个 kubernetes 数据面测试项目，通过压力注入的方式，实现对集群进行功能、性能的主动式巡检。

传统的集群巡检，通过采集指标、日志、应用状态等信息来确认集群和应用的状态，实现被动式巡检。 但是在一些特殊场景下，这种方式可能不能实现预期的巡检目的、时效性、集群范围， 运维人员就需要采用手动方式给集群注入一些压力，进行主动式巡检，当集群规模很大、巡检频率高或巡检流程复杂时， 手工方式难以持久实施。这些场景包括：

- 部署大规模集群后，希望确认所有节点间 POD 的网络连通性，避免某个节点存在网络故障，发现网络中是否存在偶发丢包问题， 而通信渠道非常多，包括 pod IP、clusterIP、nodePort、loadbalancer ip、ingress ip，甚至是 Pod 多网卡、双栈 IP
- 希望主动检测所有节点间上的 POD 能够正常访问 coredns 服务，希望确认 coredns 服务的资源配置和副本数量正确，其服务性能能欧支持预期的最大访问量
- 磁盘是易耗品，例如 etcd 等应用对磁盘性能是比较敏感的，在日常运维工作中，管理员希望周期地确认所有节点的本地磁盘是正常的， 文件读写的吞吐量和延时是符合预期的
- 给某个服务主动注入压力，它可能是镜像仓库、mysql 或者 api-server，以配合 BUG 复现，或确认服务性能

kdoctor 是一个 kubernetes 数据面测试项目，来源于生产运维过程中的实践场景，通过压力注入的方式， 实现对集群功能、性能的主动式巡检。kdoctor 可以应用于:

- 生产环境的部署检查、日常运维等场景，能避免了人工巡检的工作负担。

- 能应用 E2E 测试、bug 复现、混沌测试等，减少编程工作。

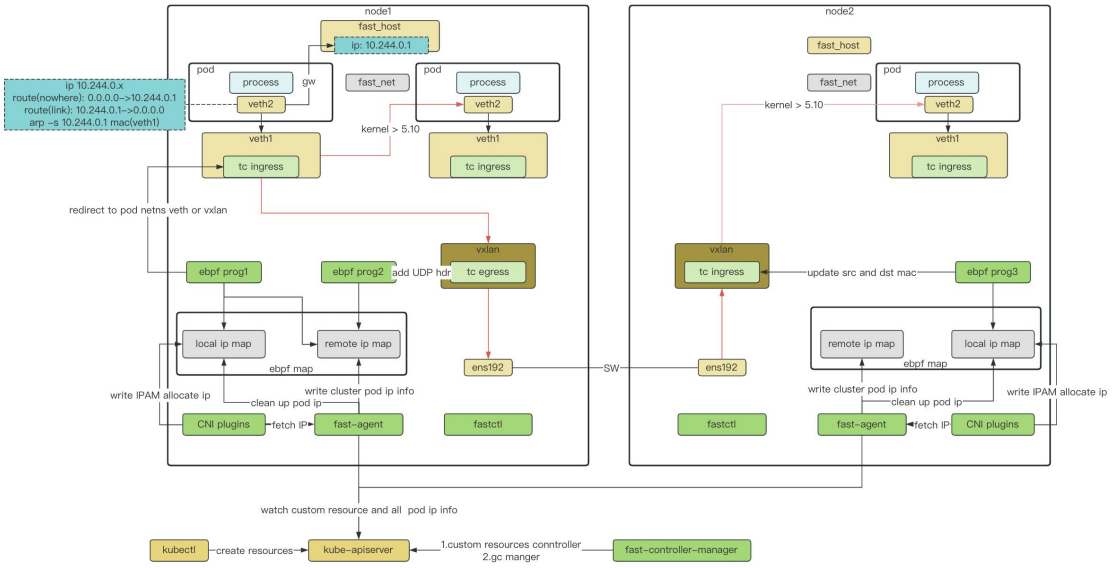
了解 kdoctor 社区

Fast

Fast 是基于 eBPF 实现的 Kubernetes CNI。目前，已经实现了以下两个功能：

- 基于 eBPF 的网络基本连通性
- 管理集群容器的 IP 地址

架构



fast

组件：

- fast-cni
  - 实现 CNI 功能
  - 访问 fast-agent 获取 Pod IP，并将其存储到节点本地的 eBPF map 中
  - 在 NIC 上附加 eBPF 程序
- fast-agent
  - 实现 IP 分配的接口

    - 获取集群的 Pod IP，并将信息存储到集群的 eBPF map 中
    - 初始化 eBPF map
- fast-controller-manager
    - 自定义资源控制
    - 执行 GC 管理，防止 IP 泄漏
- fast-ctl
    - 用于 eBPF map 的节点命令行工具

快速入门

单集群 IPAM

在覆盖网络场景中，固定集群容器的 IP 地址

1. 创建 ips 对象

```
cat << EOF | kubectl create -f -
apiVersion: sample.fast.io/v1alpha1
kind: Ips
metadata:
  name: sample-ips
spec:
  subnet: 10.244.0.1/32
  ips:
    - 10.244.10.0-10.244.90.0
EOF
```

2. 创建 Deployment，并通过注释与 ips 对象关联

```
cat << EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
```

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      annotations:
        fast.io/ips: sample-ips
      labels:
        app: busybox
    spec:
      containers:
        - name: busybox
          image: busybox:latest
          imagePullPolicy: IfNotPresent
          command:
            - sleep
            - "10000"
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
EOF
```

3. 查看容器的 IP 地址

   容器的 IP 地址被固定在 ips 定义的范围内

```
[root@york-master ~]# kubectl get po -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP
              NODE            NOMINATED NODE   READINESS GATES
busybox-757455cf7-4p9fq     1/1     Running   0          4s    10.244.1
```

```
0.1    10-29-15-49    <none>              <none>
busybox-757455cf7-f5lr9    1/1    Running    0          4s    10.244.10.
0    10-29-15-50    <none>              <none>
```

网络连通性测试

1. 创建另一个应用程序

```
cat << EOF | kubectl create -f -
apiVersion: sample.fast.io/v1alpha1
kind: Ips
metadata:
  name: default-ips
spec:
  subnet: 10.244.0.1/32
  ips:
    - 10.244.100.0-10.244.200.250
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: busybox1
  template:
    metadata:
      labels:
        app: busybox1
    spec:
      containers:
        - name: busybox1
```

```
        image: busybox:latest
        imagePullPolicy: IfNotPresent
        command:
          - sleep
          - "10000"
        ports:
          - name: http
            containerPort: 80
            protocol: TCP
EOF
```

创建后，共有四个容器

```
[root@york-master ~]# kubectl get po -o wide
NAME                          READY   STATUS    RESTARTS   AGE
  IP              NODE            NOMINATED NODE   READINESS GATE
S
busybox-757455cf7-4p9fq       1/1     Running   0          2m11s   10.
244.10.1        10-29-15-49   <none>           <none>
busybox-757455cf7-f5lr9       1/1     Running   0          2m11s   10.2
44.10.0         10-29-15-50   <none>           <none>
busybox1-5f98488b8d-2xc49     1/1     Running   0          10s     10.
244.100.0       10-29-15-50   <none>           <none>
busybox1-5f98488b8d-v42dq     1/1     Running   0          10s     10.
244.100.1       10-29-15-49   <none>           <none>
```

2. 同一节点的容器网络连接

可以看到网络是可达的

```
[root@york-master ~]# kubectl exec -it busybox-757455cf7-4p9fq -- pin
g 10.244.100.1
PING 10.244.100.1 (10.244.100.1): 56 data bytes
64 bytes from 10.244.100.1: seq=0 ttl=64 time=0.398 ms
```

```
64 bytes from 10.244.100.1: seq=1 ttl=64 time=0.117 ms
64 bytes from 10.244.100.1: seq=2 ttl=64 time=0.098 ms
64 bytes from 10.244.100.1: seq=3 ttl=64 time=0.111 ms
64 bytes from 10.244.100.1: seq=4 ttl=64 time=0.111 ms
^C
--- 10.244.100.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.098/0.167/0.398 ms
```

3. 不同节点的容器网络连接

   同样可以看到网络是可达的

```
[root@york-master ~]# kubectl exec -it busybox-757455cf7-4p9fq -- pin
g 10.244.100.0
PING 10.244.100.0 (10.244.100.0): 56 data bytes
64 bytes from 10.244.100.0: seq=0 ttl=64 time=1.029 ms
64 bytes from 10.244.100.0: seq=1 ttl=64 time=0.593 ms
64 bytes from 10.244.100.0: seq=2 ttl=64 time=0.614 ms
64 bytes from 10.244.100.0: seq=3 ttl=64 time=0.522 ms
64 bytes from 10.244.100.0: seq=4 ttl=64 time=0.704 ms
^C
--- 10.244.100.0 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.522/0.692/1.029 ms
```

接下来的计划

更多内容即将到来。欢迎提交 issue 和提交 PR。

集群检索百科全书 Clusterpedia

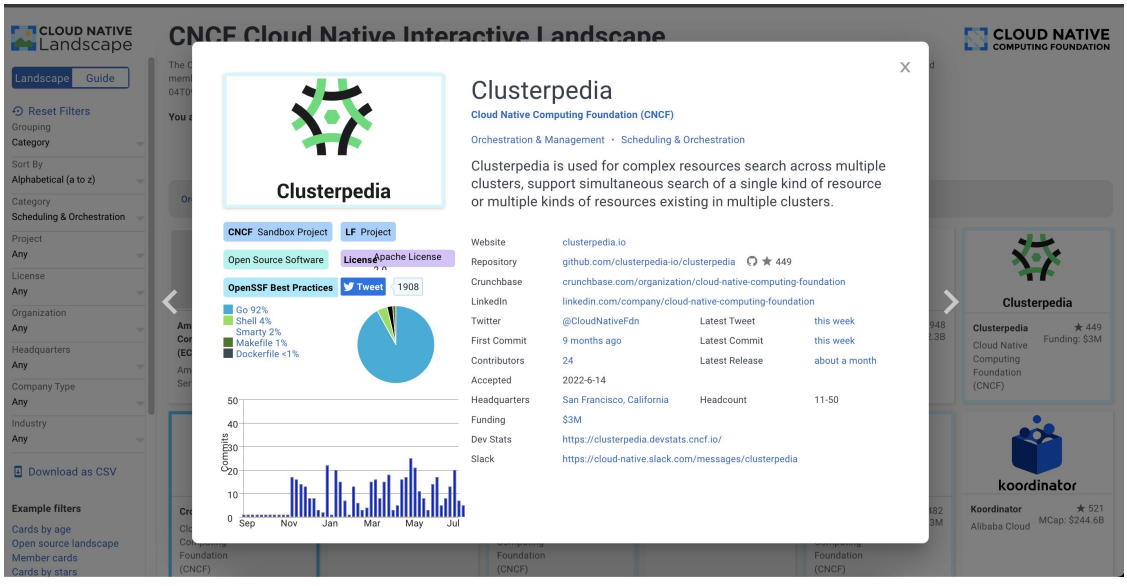
Clusterpedia 这个名字的灵感来自 Wikipedia（维基百科），我们希望这个项目像维基一样能够轻松检索容器化集群中所有资源，成为集群管理员的趁手工具，可以方便地同步、检索和控制多集群资源。

Clusterpedia 兼容 Kubernetes OpenAPI，实现了多集群资源的同步，提供了更强大的搜索功能，可以快速、轻松、有效地获取集群内所有资源信息。

Clusterpedia 的能力不仅是搜索和查看，而且还可以简单地控制资源，就像维基百科支持编辑条目一样来编辑资源。

Clusterpedia 入选云原生全景图

在 CNCF 最新发布的云原生全景图（Cloud Native Landscape）中，Clusterpedia 入选 Orchestration & Management（编排与管理）层的 Scheduling & Orchestration（调度与编排）象限，成为 CNCF 推荐的云原生多集群复杂检索工具。



landscape

参见 CNCF 全景图。

!!! info “关于 CNCF”

CNCF 全称 Cloud Native Computing Foundation（云原生计算基金会），隶属于 Linux 基金会，成立于 2015 年 12 月，是非营利性组织，致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术，普及云原生应用。

云原生全景图由 CNCF 从 2016 年 12 月开始维护，汇总了社区成熟和使用范围较广、具有最佳实践的产品和方案，并加以分类，为企业构建云原生体系提供参考，在云生态研发、运维领域具有广泛影响力。

Clusterpedia 发行版本

查阅 Clusterpedia 官网

Clusterpedia 已先后入选 CNCF 云原生全景图 和 CNCF 沙箱孵化。

Spiderpool

Spiderpool 是一个免费开源的 IPAM 管理软件，已加入 CNCF 全景图。

Spiderpool 专门设计用于 Underlay 底层网络，解决在集群内外通信时的复杂 IP 分配问题，网络管理员可以利用 Spiderpool 划分 ippool（IP 资源池）准确管理集群内外的每个 IP。

目前社区已经有一些 IPAM 插件，比如 whereabout、kube-ipam、static、dhcp、host-local，但是无法解决复杂的 Underlay 网络问题，这就是我们决定开发 Spiderpool IPAM 管理软件的原因。

Spiderpool 适用于任何能够对接第三方 IPAM 的 CNI 插件，尤其适合于一些缺失 IPAM 的 CNI，例如 SR-IOV、MacVLAN、IPVLAN、OVS-CNI 等。 而一些 Overlay CNI 自带了 IPAM 组件，对这款 Spiderpool IPAM 管理软件的需求相对较低。

了解 Spiderpool 社区

查阅 Spiderpool 官网

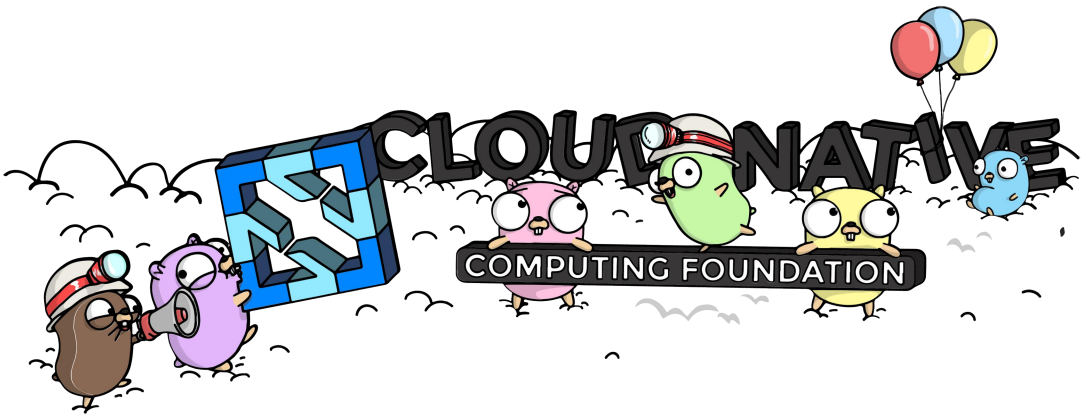
Spiderpool 固定 IP 场景视频

Spiderpool 已入选 CNCF 云原生全景图。

Kubean

Kubean 是一个基于 Kubespray 构建的集群生命周期管理工具。

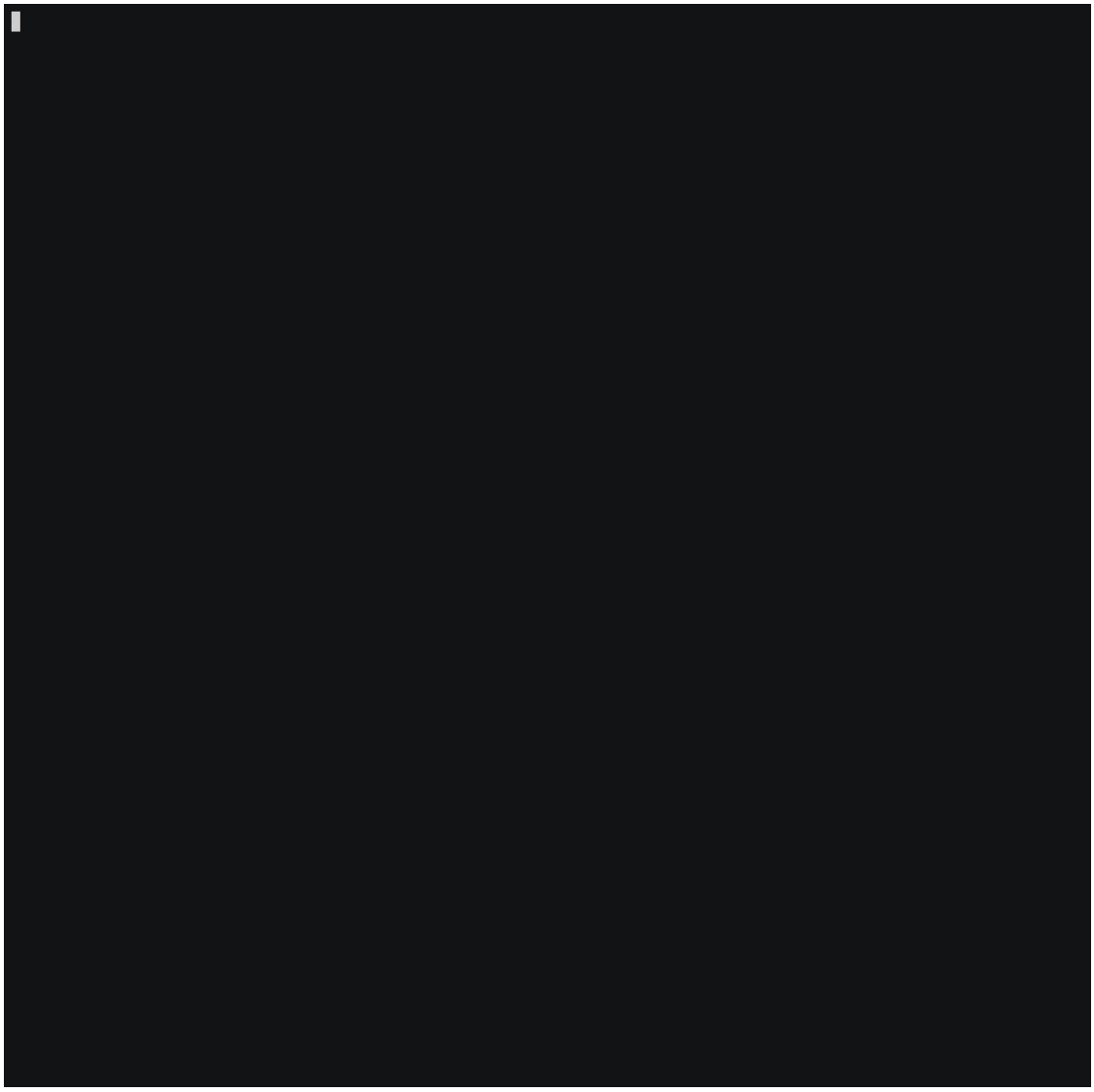
- 简单易用：通过声明式 API 实现 Kubean 和 K8s 集群强劲生命周期管理的部署。
- 支持离线：每个版本都会发布离线包（os-pkgs、镜像、二进制包）。你不必担心如何收集所需的资源。
- 兼容性：支持多架构交付：AMD、ARM；常见的 Linux 发行版；以及基于鲲鹏构建的麒麟操作系统。
- 可扩展性：允许使用原生 Kubespray 自定义集群。



cncf gophers

Kubean 是一个 CNCF Landscape 项目。

安装演示如下图：



quick_start_image

参阅 Kubean 安装视频演示

Kubean 发行版本 了解 Kubean 社区

Kubean 位列 CNCF 云原生全景图。

优势

容器管理模块具有如下优势：

集群的统一管理

- 支持不同集群统一纳管，支持所有特定版本范围内的任意 Kubernetes 集群纳入容器管理范围，实现云上、云下、多云、混合云容器云平台的统一管理，规避厂商锁定。
- 可通过 Web 界面一键式完成 Kubernetes 集群的平滑升级。
- 统一 Web 界面实现集群创建、集群节点的扩缩容等管理能力。
- 一键式证书轮滚，自建集群可在 Web 界面实现证书轮滚。
- 支持超大规模集群的统一纳管。

应用生产就绪

- 一站式应用分发，可通过镜像、YAML、Helm 进行应用分发，并实现跨云/跨集群的统一管理。
- 应用高可用，支持应用的分布式部署，支持单点故障流量的自动切换。
- 丰富的监控指标，实现应用的全方位监控，提前预警应用流量高峰，应用故障。

策略的统一下发

- 支持以命名空间或集群粒度制定网络策略、配额策略、资源限制策略、灾备策略、安全策略等策略，并支持以集群/命名空间为粒度进行策略下发。

安全可靠

- 自建集群默认为高可用部署模式，保障您的业务高可用。当节点出现故障、自然灾害时，应用可持续运行，保障生产环境的高可用，实现业务应用系统不中断。

- 跨地域应用高可用，支持跨地域部署不同的容器集群，能够同时将业务部署在不同地域的多个云容器集群上，帮助应用实现多地域的流量分发。 当某个云容器集群发生故障、机房宕机或者自然灾害时，通过统一流量分发的机制，自动将业务流量切换到其他云容器平台上，保障应用的高可用。
- 完善的用户权限体系，整合 Kubernetes RBAC 权限体系，支持为不同用户设置不同粒度权限。

异构兼容

- 提供高度自动化和具备弹性的异构多云支撑能力，适配 x86 以及信创云体系架构。
- 支持 x86、ARM 架构混合集群统一部署，统一管理和支撑应用运行，保障应用间网络互通。

开放兼容

- 基于原生的 Kubernetes 和 Docker 技术，完全兼容 Kubernetes API 和 Kubectl 命令。
- 提供了丰富的插件体系，扩充云容器集群功能，例如网络插件 Multus、Cillum、Contour 等组件。

容器管理 Release Notes

本页列出容器管理的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-8-01

v0.20.0

新功能

- **新增** Helm 应用界面支持查看 Helm 操作日志。
- **新增** 工作集群支持接入异构节点。
- **新增** 创建集群支持批量导入节点。
- **新增** 容器存储支持创建 NFS 类型的数据卷。
- **新增** 新增 vGPU 支持，支持自动识别节点 CPU、支持新增为负载配置 CPU 配额。

优化

- **优化** 优化集群接入逻辑，当接入集群二次接入新管理平台时，需要预先清理旧管理平台的数据冗余才能被接入，关于集群接入的更多细节，请参考卸载/解除接入集群。
- **优化** 升级 clusterpedia 到 v0.7.0。
- **优化** 优化基于权限的页面交互，无权限用户将无法进入无资源权限的页面。
- **优化** 接入节点支持配置内核调优等高级参数配置。
- **优化** Insight 组件安装检测机制。

修复

- **修复** 修复 helm 任务一直处在安装中、卸载中的问题。
- **修复** 创建集群节点检查内核版本检测错误问题。
- **修复** 创建集群插件无法自定义命名空间的问题。

- **修复** 更新密钥默认增加 ca.crt 数据的问题。

2023-7-06

v0.19.0

新功能

- **新增** 兼容 openAnolis / oracle linux 操作系统部署工作集群。
- **新增** 离线环境创建集群支持自动添加 jfrog 的认证信息。
- **新增** 创建工作负载新增环境变量规则校验。
- **新增** 新增边缘负载和服务。
- **新增** 新增双栈、系统内核作为节点前置检查项。
- **新增** 创建工作负载新增 secretKey/configmapKey 作为配置项挂载在容器内。

优化

- **优化** 优化 Helm 仓库刷新机制。
- **优化** 优化部分 I8N 英文翻译界面。

修复

- **修复** 修复在创建集群时，填写自定义参数，如果 value 为 0 或者 1，会被错误转换为 true 或者 false 的问题。
- **修复** 修复在离线环境创建集群时，无法写入 containerd 账号密码配置的问题。
- **修复** 修复对 1.26 及以上版本的集群进行升级时，由于 kubernetes 镜像仓库更改的原因，导致集群升级失败的问题。
- **修复** 修复命名空间级用户无法使用 StorageClasses 创建 PV 相关问题。
- **修复** 修复创建路由时指定命名空间不生效的问题。
- **修复** 修复集群升级后，日期返回错误问题。

2023-6-03

v0.18.1

优化

- **优化** 安装集群设置自定义参数时不限最大长度。

2023-5-28

v0.18.0

新功能

- **新增** 巡检报告下载。
- **新增** 对接高优先级操作全局审计日志。
- **新增** 连接 Minio 的超时处理。

优化

- **优化** CloudShell 由用 ConfigMap 进行 KubeConfig 挂载改为用 Secret 进行 KubeConfig 挂载。
- **优化** 创建备份策略集群下拉列表新增过滤创建了备份策略的集群的开关。

修复

- **修复** etcdbrctl 镜像离线化。
- **修复** 镜像选择器无法选择镜像。
- **修复** 创建集群时的 Repo 地址渲染。

2023-04-28

v0.17.0

新功能

- **新增** 巡检报告下载
- **新增** 查看 ETCD 备份日志
- **新增** 创建集群支持启用 Flannel、Kube-ovn 网络插件
- **新增** 创建集群启用 Cilium 双栈网络
- **新增** 创建集群支持自动识别节点 OS 类型

- **新增** Headless、External 类型的服务
- **新增** 离线环境下升级工作集群的 kubernetes 版本
- **新增** 集群级资源备份
- **新增** 使用私有密钥创建工作负载
- **新增** 配置 Helm job 的默认资源限制
- **新增** 使用 hwameistor 创建 PVC

优化

- **优化** 应用备份集群状态
- **优化** 负载详情内负载状态和负载下容器组状态不匹配的问题
- **优化** 离线模式下节点检查接口
- **优化** 多云应用的展示方式

修复

- **修复** 更新 helm 应用配置丢失的问题
- **修复** 使用 yaml 创建多种类型资源由于 ns 不一致导致创建失败的问题
- **修复** 使用麒麟操作系统无法选择 Docker 19.03 运行时，导致创建集群失败的问题
- **修复** 英文界面的错误翻译

2023-04-04

v0.16.0

新功能

- **新增** 使用界面查询 PVC 事件。
- **新增** 创建任务支持配置 backofflimit、completions、parallelism、activeDeadlineSeconds 等参数。
- **新增** 集成自研开源存储组件 Hwameistor，支持在 容器存储 模块查看本地存储资源概览等信息。
- **新增** 新增集群巡检功能，支持对集群进行秒级巡检（Alpha）。

- **新增** 新增应用备份功能，支持界面化快速对应用进行备份和恢复（Alpha）。
- **新增** 新增平台备份功能，支持对 ETCD 数据进行备份和恢复（Alpha）。
- **新增** 支持 Ghippo 的自定义角色管理集群。

优化

- **优化** Kpanda 卸载自建集群的流程，以此避免因用户误操作导致集群被删除。
- **优化** 界面创建集群失败后重新创建集群的用户体验，支持用户基于失败前的配置快速重新安装集群。
- **优化** 优化了当一个 命名空间下存在多个 Quota 资源时的处理逻辑。对多个 Quota 进行了聚合处理
- **优化** 工作负载详情内服务访问方式的信息展示，支持快速对负载服务进行访问。
- **优化** 优化 Helm 仓库刷新机制，默认不开启自动刷新

修复

- **修复** Loadblance 地址无法访问的问题。
- **修复** 执行卸载集群操作失败的问题。
- **修复** 接入集群超过 64 个字符导致集群无法获取的问题。
- **修复** 修复离线环境集群无法展示集群插件的问题。
- **修复** Global 集群无法更新配置的问题。
- **修复** 创建集群时，第一次节点检查失败，无法再次执行节点检查的问题。
- **修复** 创建/更新工作负载的环境变量不生效的问题。

2023-02-27

v0.15

新功能

- **新增** 对 PV(Persistent Volumes) 的产品化支持，支持在创建 PVC 时选择已有数据卷。

- **新增** 使用 kubernetes 无网络 CNI 创建集群的能力。
- **新增** 支持负载、配置、服务等资源中文名称。
- **新增** 通过 YAML 创建工作负载支持同时创建多种类型资源。
- **新增** 工作负载的暂停、启动功能。

优化

- **优化** 集群详情页，集群切换的使用体验。
- **优化** 工作负载状态显示，增加已停止（Stopped）状态。
- **优化** 工作负载增加手动扩缩容窗口，简化用户手动扩缩负载流程。
- **优化** 接入集群无法接入 DCE4.X 集群问题。

修复

- **修复** 修复了创建集群时，DNS 配置强制要求用户填写 upstream DNS 的问题。
- **修复** 修复了工作负载版本记录排序混乱问题。
- **修复** 通过 Helm 升级 Kubean 无效的问题。
- **修复** 创建集群执行节点检查失败后再次检查，上次的异常提示未消失的问题。
- **修复** 创建工作负载，镜像拉取失败问题。
- **修复** 定时的备份策略，无法执行立即执行操作的问题。
- **修复** 修改无资源限制的工作负载时，UI 会自动添加资源限制问题。
- **修复** 当工作空间没有与任何用户进行绑定时，往这个工作空间添加命名空间失败的问题。
- **修复** 修复绑定、解绑命名空间会导致命名空间注解消失的问题。
- **修复** 修复创建集群使用 kube-vip 策略不生效的问题。
- **修复** 创建集群设置 ntp servers 为空时，将清空主机已有 ntp 地址问题。

2022-12-29

v0.14

新功能

- **新增** Helm 模板支持展示中文名称和模板供应商。
- **新增** CronHPA，支持定时伸缩工作负载。
- **新增** VPA（垂直伸缩），支持手动/自动两种方式修改资源请求值，实现工作负载垂直伸缩。
- **新增** Namespace 独享主机功能。
- **新增** 存储池（StrogeClass）支持授权给特定命名空间独享或共享。
- **新增** 创建工作负载支持展示当前命名空间剩余资源配额。
- **新增** 节点连通性检查功能。
- **新增** 新增镜像选择器，支持创建工作负载时选择镜像仓库内的镜像。
- **新增** 应用备份与恢复功能。

优化

- **优化** 集群卸载流程，增加集群删除保护开关。
- **优化** 通过 YAML 创建资源时支持同时创建多个资源。
- **优化** 工作负载增加手动扩缩容窗口，简化用户手动扩缩负载流程。
- **优化** 服务（Service）访问方式体验，支持服务快速访问和展示节点、负载均衡地址。
- **优化** 文件上传下载支持选择特定容器。
- **优化** 支持不同 OS 系统离线安装。
- **优化** 离线环境下创建集群——节点配置支持选择节点操作系统和修改离线 Yum 源。
- **优化** YAML 编辑器未填写 Namespace 字段，支持自动补齐为 Default。
- **优化** 集群升级界面交互体验。
- **优化** 使用 Helm 创建应用时，提供 Namespace 快速创建入口。

修复

- **修复** 无法使用密码新增节点的问题。
- **修复** 获取 Token 方式接入的集群 kubeconfig 错误问题。
- **修复** 授予权限时无法获取完整的用户和用户组。
- **修复** Bindingsync 组件不正常时解绑工作空间原始权限存在问题
- **修复** Workspace Resync 无法正确将多余权限删除的问题。
- **修复** 删除中的 Namespace 还可以被选择的问题。
- **修复** 创建密钥，密钥数据单行显示的问题。

2022-11-29

v0.13

新功能

- **新增** Replicatsets 产品化：
  - 支持使用 WEB 终端（CloudTTY）管理 Replicatsets。
  - 支持查看 Replicatsets 监控、日志、Yaml、事件、容器。
  - 支持查看 Replicatsets 详情。
  - 联动**应用工作台**，由灰度发布管理 Replicatsets 全生命周期。
- **新增** Pod 详情页面。
- **新增** 命名空间详情页。
- **新增** 使用 WEB 终端 上传文件至容器内及从 Pod 内下载文件至本地。
- **新增** 工作负载基于自定义指标弹性伸缩，更加贴近用户实际业务弹性扩缩容需求。

优化

- **优化** 部署集群支持：
  - 使用 cilium CNI 部署集群。
  - 使用不同用户名、密码、SSH 端口的节点创建集群。
- **优化** Pod 列表支持查看容器组总数和运行中数量，以及支持查看容器类型。

- **优化** 工作负载增加手动扩缩容窗口，简化用户手动扩缩负载流程。
- **优化** 容器日志支持查看 init container 和 ephemeral container，提供更友好的运维体验。
- **优化** 节点详情，注解 vaule 值未正确展示问题。
- **优化** 操作提示反馈，给予用户操作以正确的反馈。

修复

- **修复** 因创建命名空间和绑定工作空间强耦合导致创建命名空间失败的问题。
- **修复** 更新路由规则无法修改转发策略的路径前缀问题。
- **修复** 创建工作负载界面同时创建 Services 不生效的问题。
- **修复** 更新服务异常报错问题。
- **修复** 无法接入 AWS 集群问题。
- **修复** 使用 WS Admin 用户绑定资源组后用户列表不同步问题。
- **修复** 配置详情页，PageSize=50 时，ListClusterConfigMaps 接口 异常报错问题。

2022-10-28

v0.10

新功能

- **新增** NetworkPolicy 策略管理功能，包括创建、更新、删除 NetworkPolicy 策略，以及 NetworkPolicy 策略详情展示，帮助用户为 Pod 配置进网络出入流量策略
- **新增** 工作负载支持多网卡配置和支持 IP Pool 展示，满足用户为工作负载配置单独配置多网卡需求
- **新增** 集群创建失败后支持查看创建过程的操作日志，帮助用户快速定位故障
- **新增** 有状态工作负载支持使用动态数据卷模板

- **新增** 创建集群、创建 Secret、创建 Ingress、编辑命名空间配额的信息校验，帮助引导用户输入正确的配置参数，降低用户创建任务失败体验

优化

- **优化** 集群下拉列表支持展示集群状态，优化用户在创建集群选择被纳管集群、创建命名空间选择目标集群、集群授权选择目标集群的使用体验
- **优化** 在 helm 应用中安装 insight-agent 插件，支持自动获取并填充全局服务集群的 Insight-server 相关地址
- **优化** Helm 模板图标为空时的默认图标
- **优化** 创建集群时选择网络模式为 None，以允许用户在集群创建完成后再安装网络插件
- **优化** 集群操作信息架构：
  - 将集群列表和集群概览页面的集群升级操作，调整至集群详情内的集群运维功能下
  - 当某个管理集群在集群列表内移除后，基于这个管理集群创建的集群将在界面隐藏　集群升级、纳管节点、删除节点操作

修复

- **修复** 资源切换时，所选命名空间自动转换为全部命名空间的问题

容器管理功能发布动态

本文介绍 DCE 5.0 容器管理相关内容的最新动态。

2023 年 4 月

功能

描述

版本

文档

新增界面查询 PVC 事件

支持在图形界面上直观地查阅集群中所有 PVC 事件的级别、相关 K8s 组件、操作对象、事件名称、发生的时间等信息

0.16.0

数据卷声明（PVC）

Job 新增 backofflimit、completions、parallelism、activeDeadlineSeconds 等参数

backoffLimit：指定标记此 Job 失败之前的重试次数。默认值为 6。

completions：指定 Job 应该运行并预期成功完成的 Pod 个数。设置为 nil 意味着任何 Pod 的成功都标识着所有 Pod 的成功， 并允许 parallelism 设置为任何正值。设置为 1 意味着并行性被限制为 1，并且该 Pod 的成功标志着任务的成功。

parallelism：指定 Job 应在任何给定时刻预期运行的 Pod 个数上限。当 (.spec.completions – .status.successful) < .spec.parallelism 时， 即当剩余的工作小于最大并行度时，在稳定状态下运行的 Pod 的实际数量将小于此数量。

activeDeadlineSeconds：系统尝试终止 Job 之前 Job 可以持续活跃的秒数，这个时间长度是相对于 startTime 的； 字段值必须为正整数。如果 Job 被挂起， 则当 Job 再次恢复时，此计时器会被停止并重置。

0.16.0

创建 Job

集成自主开源的存储组件 HwameiStor

支持在容器存储中查看本地存储资源概览等信息

0.16.0

存储池

新增集群巡检功能

支持对集群进行秒级巡检，目前这是 Alpha 特性

0.16.0

集群最近操作

新增应用备份功能

支持在图形界面上快速备份和恢复应用，目前这是 Alpha 特性

0.16.0

备份工作负载

新增平台备份功能

支持对 etcd 数据进行备份和恢复，目前这是 Alpha 特性

0.16.0

ETCD 备份还原

新增全局管理的自定义角色管理集群

支持通过全局管理创建自定义角色来管理集群

0.16.0

创建自定义角色

有关容器管理完整的功能列表，请查阅容器管理功能总览。

基本概念

容器管理相关的基本概念如下。

集群 Cluster

集群指容器运行所需要的云资源组合，关联了若干云服务器节点。您可以在集群中运行您的应用程序。基于容器管理模块，可以创建若干集群或者接入若干 Kubernetes 标准集群。

节点 Node

每一个节点对应一台虚拟机/物理服务器，所有的容器应用运行在集群的节点上。容器平台上的系统组件默认运行在控制器节点上，用于管理节点上运行的容器实例。集群中的节点数量可以进行扩缩容，节点类型分为：控制器节点（Controller）及工作节点（Worker）。

容器组 Pod

容器组是 Kubernetes 部署应用或服务的最小的基本单位。一个容器组可以封装一个或多个应用容器、存储资源、一个独立的网络 IP，这些容器是相对紧密的耦合在一起的。

容器 Container

容器是通过容器镜像部署运行的实例，容器将应用程序从底层的主机设施中解耦，因此在不同的云或 OS 环境中应用程序的部署部署更容易。

工作负载 Workload

工作负载是在 Kubernetes 上运行的应用程序。

无论工作负载是单一组件还是由多个组件构成，在 Kubernetes 中都可以在一组 Pod 中运行这些工作负载。

在 Kubernetes 提供若干种内置的工作负载资源：

- **无状态服务**（Deployment）：容器组之间完全独立、功能相同，具有弹性扩缩、滚动升级等特性。常用来部署无状态应用实现快速的扩缩，相较于有状态服务，实例数量可以灵活扩缩。例如 Nginx、WordPress。请参考创建无状态服务。
- **有状态服务**（StatefulSet）：容器组之间不完全独立，具有稳定的持久化存储和网络标示，以及有序的部署、扩缩和删除等特性。因为容器可以在不同主机间迁移，所以在主机上并不会保存数据，通过将存储卷挂载在容器上，从而实现有状态（Statefulset）服务的数据持久化，例如 mysql-HA、etcd。请参考创建有状态服务。
- **守护进程服务**（DaemonSet）：容器组之间完全独立，保证在分配的节点中持续执行后台任务，而无需用户干预。守护进程（DaemonSet）服务在每个节点创建一个 Pod，您可以选择部署的一个特定节点。守护进程示例包括像 Fluentd 之类的日志收集器和监控服务。请参考创建守护进程服务。
- **普通任务**（Job）：普通任务是一次性运行的短任务，部署完成后即可执行。使用场景为在创建工作负载前，执行普通任务，将镜像上传至镜像仓库。请参考创建普通任务。
- **定时任务**（CronJob）：定时任务是按照指定时间周期运行的短任务。使用场景为在某个固定时间点，为所有运行中的节点做时间同步。请参考创建定时任务。

服务与容器间的关系

一个服务由一个或多个容器组（Pod）组成。一个容器组由一个或多个容器组成，每个容器都对应一个容器镜像。对于无状态工作负载，容器组都是完全相同的。

应用模板

标准模板的统一资源管理和调度，并进行相关功能扩展。您可以基于应用模板管理和部署社区标准应用模板，以及自定义业务应用模板。

镜像 Image

容器镜像是一个容器应用打包的标准格式的模板，用于创建容器。 Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

例如：一个镜像可以包含一个完整的 Ubuntu 操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。

命名空间 Namespace

命名空间是对一组资源和对象的抽象整合。在同一个集群内可创建不同的命名空间，不同命名空间中的数据彼此隔离。使得它们既可以共享同一个集群的服务，也能够互不干扰。例如：

- 可以将开发环境、测试环境的业务分别放在不同的命名空间。
- 常见的 Pod、Service、Replication Controller 和 Deployment 等都属于某一个命名空间（默认是 Default），而 Node、PersistentVolume 等则不属于任何命名空间。

服务 Service

Service 是将运行在一组 Pod 上的应用程序公开为网络服务的抽象方法。

使用 Kubernetes，您无需修改应用程序即可使用不熟悉的服务发现机制。Kubernetes 为 Pod 提供自己的 IP 地址和一组 Pod 的单个 DNS 名称，并且可以在它们之间进行负载平衡。

Kubernetes 允许指定一个所需类型的 Service，该类型的取值以及行为如下：

- ClusterIP：集群内访问。通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部访问，这也是默认的 ServiceType。
- NodePort：节点访问。通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个

ClusterIP 服务会自动创建。通过请求 <NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务。

- LoadBalancer：负载均衡。使用云提供商的负载均衡器，可以向外部暴露服务。外部的负载均衡器可以路由到 NodePort 服务和 ClusterIP 服务。

七层负载均衡 Ingress

Ingress 是为进入集群的请求提供路由规则的集合，可以给 Service 提供集群外部访问的 URL、负载均衡、SSL 终止、HTTP 路由等。

网络策略 NetworkPolicy

NetworkPolicy 提供了基于策略的网络控制，用于隔离应用并减少攻击面。它使用标签选择器模拟传统的分段网络，并通过策略控制它们之间的流量以及来自外部的流量。

配置项 ConfigMap

ConfigMap 用于保存配置非机密性的数据保存到键值对中。使用时， 容器组可以将其用作环境变量、命令行参数或者存储卷中的配置文件。

ConfigMap 将您的环境配置信息和容器镜像解耦，便于应用配置的修改。

密钥 Secret

Secret 类似于 Configmap，但用于保存机密数据（如密码、Token、密钥等）的配置信息，Secret 将敏感信息和容器镜像解耦，不需要在应用程序代码中包含机密数据。

标签 Label

标签其实是一对 key/value，被关联到对象上，比如 Pod。标签的使用倾向于能够标示对象的特点，并且对用户而言是有意义的，但是标签对内核系统是没有直接意义的。标签可以在创建对象时指定，也可以在对象创建后指定。

选择器 LabelSelector

Label Selector 是 Kubernetes 核心的分组机制，通过 Label Selector 客户端/用户能够识别一组有共同特征或属性的资源对象。

注解 Annotation

Annotation 可以将 Kubernetes 资源对象关联到任意的非标识性元数据，可以通过注解检索到这些元数据。

Annotation 与 Label 类似，也使用 Key/Value 键值对的形式进行定义。

存储卷 PersistentVolume

PersistentVolume (PV) 和 PersistentVolumeClaim (PVC) 提供了方便的持久化卷：PV 提供网络存储资源，而 PVC 申领存储资源。

存储声明 PersistentVolumeClaim

PV 是存储资源，而 PersistentVolumeClaim (PVC) 是对 PV 的申领请求。PVC 跟 Pod 类似：Pod 消费 Node 资源，而 PVC 消费 PV 资源；Pod 能够请求 CPU 和内存资源，而 PVC 请求特定大小和访问模式的数据卷。

弹性扩缩 HPA

Horizontal Pod Autoscaling，简称 HPA，是 Kubernetes 中实现 Pod 水平自动扩缩的功能。Kubernetes 集群可以通过 Replication Controller 的扩缩机制完成服务的扩容或缩容，实现具有扩缩性的服务。

亲和性与反亲和性

亲和性和反亲和性扩展了您可以定义的约束类型。使用亲和性与反亲和性的一些好处有：

- 亲和性、反亲和性的表现能力更强。nodeSelector 只能选择拥有所有指定标签的节点。亲和性、反亲和性为您提供对选择逻辑的更强控制能力。
- 您可以标明某规则是“软需求”或者“偏好”，这样调度器在无法找到匹配节点时仍然调度该 Pod。

- 您可以使用节点上（或其他拓扑域中）运行的其他 Pod 的标签来实施调度约束，而不是只能使用节点本身的标签。这个能力让您能够定义规则允许哪些 Pod 可以被放置在一起。

在应用没有容器化之前，原先一个虚机上会装多个组件，进程间会有通信。但在做容器化拆分的时候，往往直接按进程拆分容器，比如业务进程一个容器，监控日志处理或者本地数据放在另一个容器，并且有独立的生命周期。 这时如果它们分布在网络中两个较远的节点，请求经过多次转发，其性能会很差。

- 亲和性：可以实现就近部署，增强网络能力实现通信上的就近路由，减少网络的损耗。例如应用 A 与应用 B 两个应用频繁交互，所以有必要利用亲和性让两个应用的尽可能的靠近，甚至在一个节点上，以减少因网络通信而带来的性能损耗。

- 反亲和性：主要是出于高可靠性考虑，尽量分散实例，某个节点故障的时候，对应用的影响只是 N 分之一或者只是一个实例。例如当应用采用多副本部署时，有必要采用反亲和性让各个应用实例打散分布在各个节点上，以提高 HA 能力。

节点亲和性 NodeAffinity

通过选择标签的方式，可以限制容器组被调度到特定的节点上。

节点反亲和性 NodeAntiAffinity

通过选择标签的方式，可以限制容器组不被调度到特定的节点上。

容器组负载亲和性 PodAffinity

指定工作负载部署在相同节点。用户可根据业务需求进行工作负载的就近部署，容器间通信就近路由，减少网络消耗。

容器组反亲和性 PodAntiAffinity

指定工作负载部署在不同节点。同个工作负载的多个实例反亲和部署，减少宕机影响；互相干扰的应用反亲和部署，避免干扰。

资源配额 Resource Quota

资源配额（Resource Quota）是用来限制用户资源用量的一种机制。

资源限制 Limit Range

默认情况下，Kubernetes 中所有容器都没有任何 CPU 和内存限制。
LimitRange 用来给命名空间增加一个资源限制，包括最小、最大和默认资源。在容器组创建时，强制执行使用 limits 的参数分配资源。

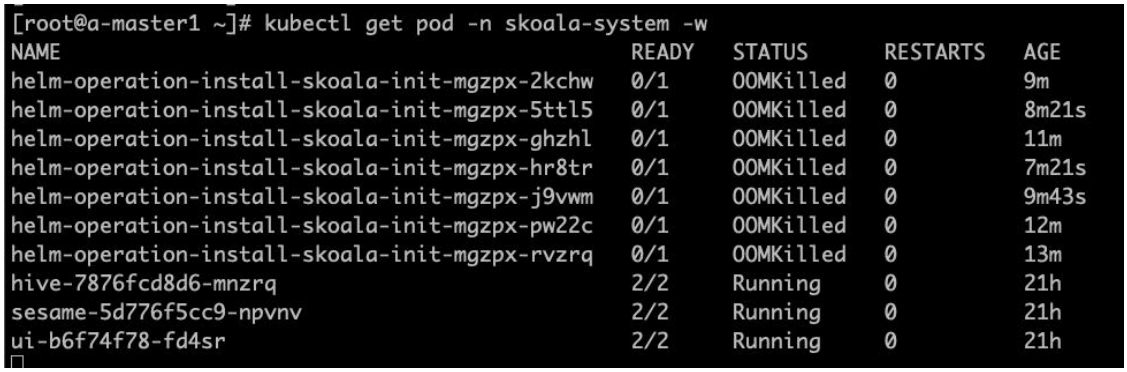
环境变量

环境变量是指容器运行环境中设定的一个变量，您可以在创建容器模板时设定不超过 30 个的环境变量。环境变量可以在工作负载部署后修改，为工作负载提供了极大的灵活性。

常见问题

本页面列出了一些在容器管理中可能遇到的常见问题，为您提供便利的故障排除解决方案。

1. Helm 应用安装失败，提示“OOMKilled”



失败情况

如图所示，容器管理会自动创建启动一个 Job 负责具体应用的安装工作，在 v0.6.0 版本中由于 job resources 设置不合理，导致 OOM,影响应用安装。该 bug 在 0.6.1 版本中已经被修复。如果是升级到 v0.6.1 的环境，仅仅会在新创建、接入的集群中生效，已经存在的集群需要进行手动调整，方能生效。

??? note“调整脚本”

– 以下脚本均在全局服务集群中执行
– 找到对应集群，本文以 skoala-dev 为例,获取对应的 skoala-dev-setting configmap
– 更新 configmap 之后即可生效

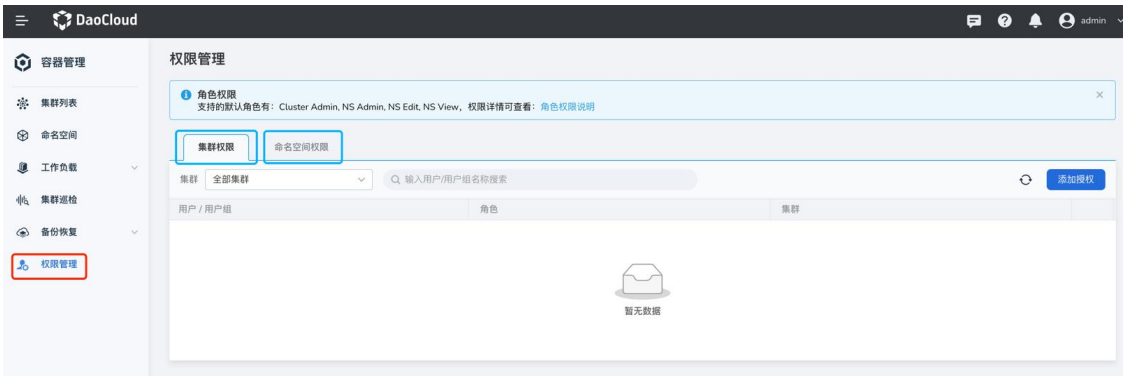
```shell
kubectl get cm -n kpanda-system skoala-dev-setting -o yaml
```

```
apiVersion: v1
data:
clusterSetting: '{"plugins":[{"name":1,"intelligent_detection":true},{"name":2,"enabled":true,"intelligent_detection":true},{"name":3},{"name":6,"intelligent_detection":true},{"name":7,"intelligent_detection":true},{"name":8,"intelligent_detection":true},{"name":9,"intelligent_detection":true}],"network":[{"name":4,"enabled":true,"intelligent_detection":true},{"name":5,"intelligent_detection":true},{"name":10},{"name":11}],"addon_setting":{"helm_operation_history_limit":100,"helm_repo_refresh_interval":600,"helm_operation_base_image":"release-ci.daocloud.io/kpanda/kpanda-shell:v0.0.6","helm_operation_job_template_resources":{"limits":{"cpu":"50m","memory":"120Mi"},"requests":{"cpu":"50m","memory":"120Mi"}}},"clusterlcm_setting":{"enable_deletion_protection":true},"etcd_backup_restore_setting":{"base_image":"release.daocloud.io/kpanda/etcdbrctl:v0.22.0"}}'
kind: ConfigMap
    metadata:
    labels:
        kpanda.io/cluster-plugins: ""
    name: skoala-dev-setting
    namespace: kpanda-system
    ownerReferences:
    - apiVersion: cluster.kpanda.io/v1alpha1
        blockOwnerDeletion: true
        controller: true
        kind: Cluster
        name: skoala-dev
        uid: f916e461-8b6d-47e4-906e-5e807bfe63d4
    uid: 8a25dfa9-ef32-46b4-bc36-b37b775a9632

    修改 clusterSetting-> helm_operation_job_template_resources 到合适的值即可，v0.6.1 版本对应的值为 cpu: 100m,memory: 400Mi
    ```
```
```

2. 容器管理模块和全局管理模块的权限问题

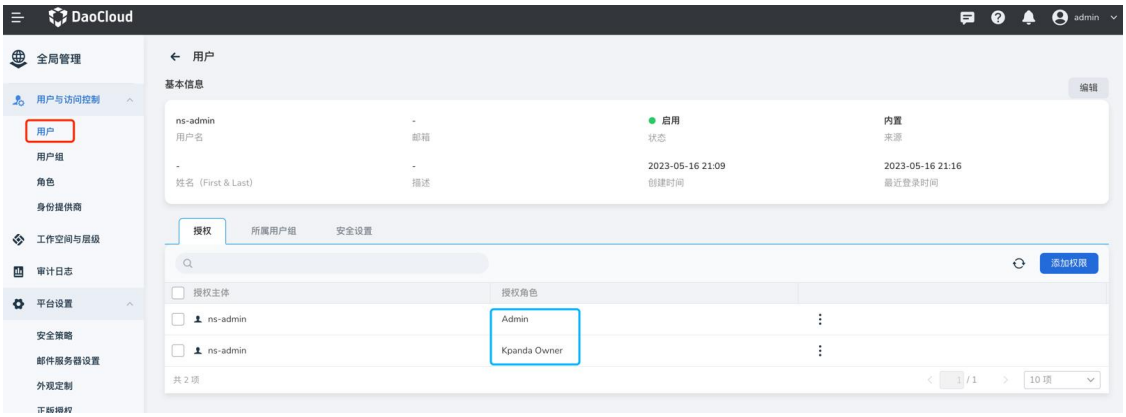
   经常有用户会问，为什么我这个用户可以看到这个集群，或者为什么我看不到这个集群，我们应该如何排查相关的权限问题？分为以下三种情况：

   - 容器管理模块的权限分为集群权限、命名空间权限。如果绑定了用户，那该用户就可以查看到相对应的集群及资源。具体权限说明，可以参考集群权限说明。



容器管理权限

   - 全局管理模块中用户的授权：使用 admin 账号，进入全局管理 -> 用户与访问控制 -> 用户菜单，找到对应用户。在授权所属用户组标签页，如果有类似 Admin、Kpanda Owner 等拥有容器管理权限的角色，那即使在容器管理没有绑定集群权限或命名空间权限，也可以看到全部的集群，可以参考用户授权文档说明

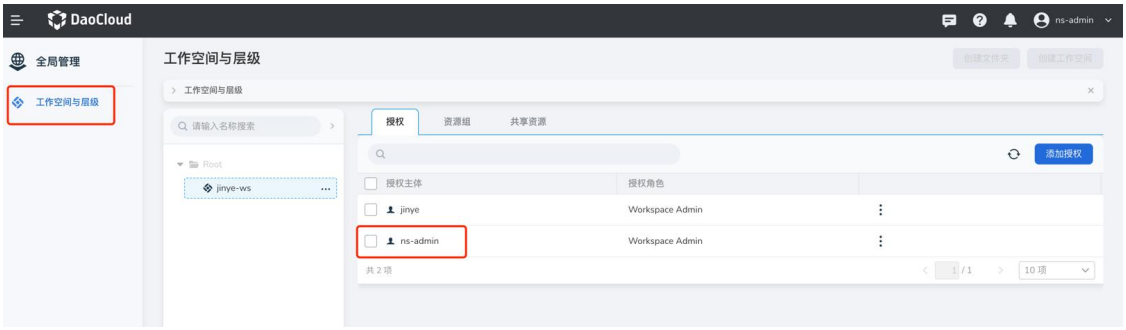


全局管理 用户授权

- 全局管理模块中工作空间的绑定：使用账号进入全局管理 -> 工作空间与层级，可以看到自己的被授权的工作空间，点击工作空间名称

  a.如果该工作空间单独授权给自己，就可以在授权标签页内看到自己的账号，然后查看资源组或共享资源标签页，如果资源组绑定了命名空间或共享资源绑定了集群，那该账号就可以看到对应的集群

  b.如果是被授予了全局管理相关角色，那就无法授权标签页内看到自己的账号，也无法在容器管理模块中看到工作空间所绑定的集群资源



全局管理工作空间的绑定

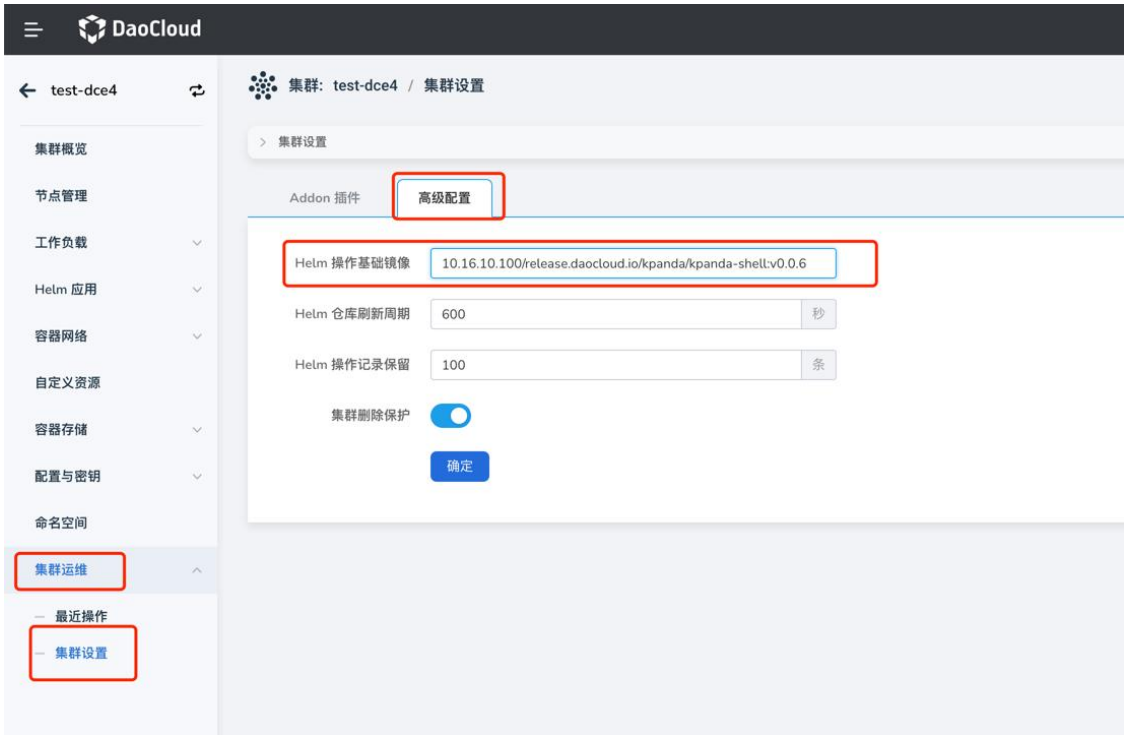
3. Helm 安装应用时，无法拉取 kpanda-shell 镜像

   使用离线安装后，接入的集群安装 helm 应用经常会遇到拉取 kpanda-shell 镜像失败，如图：



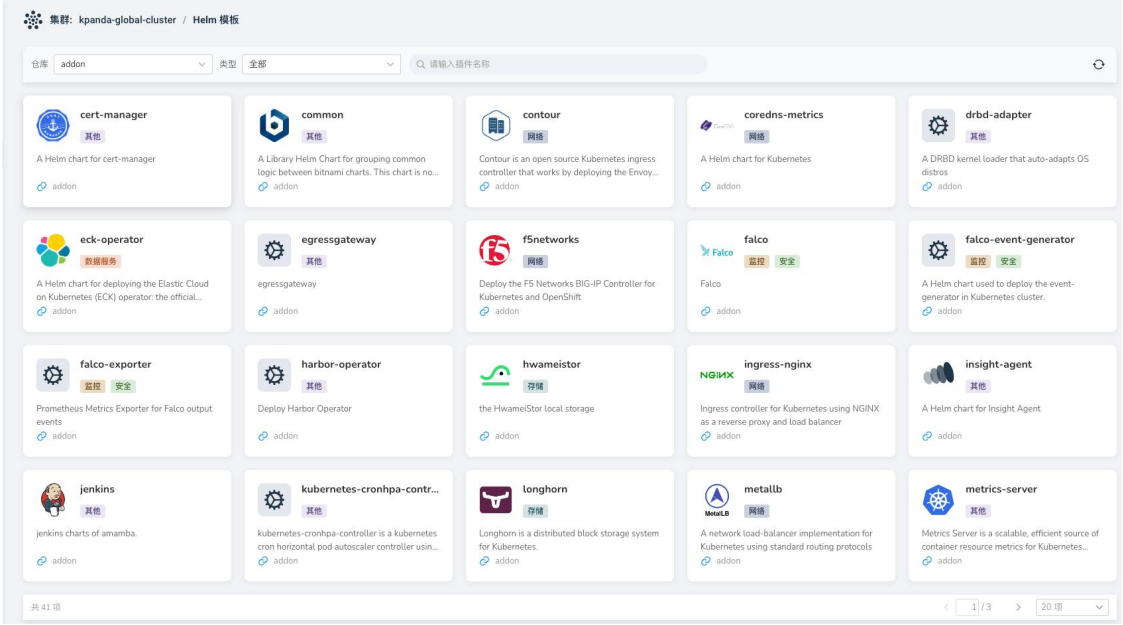
拉取镜像失败

此时，只需要去集群运维-集群设置页面，高级配置标签页，修改 Helm 操作基础镜像为一个可以被该集群正常拉取到的 kpanda-shell 的镜像即可。



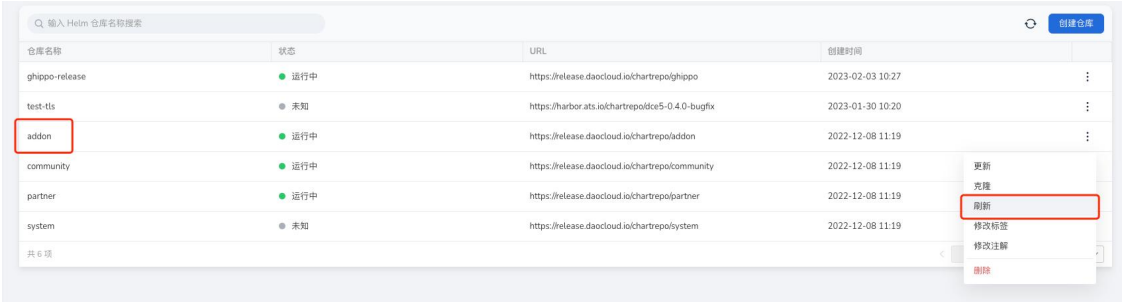
修改镜像

4. Helm Chart 界面未显示最新上传到对应 Helm Repo 的 Chart，如图：



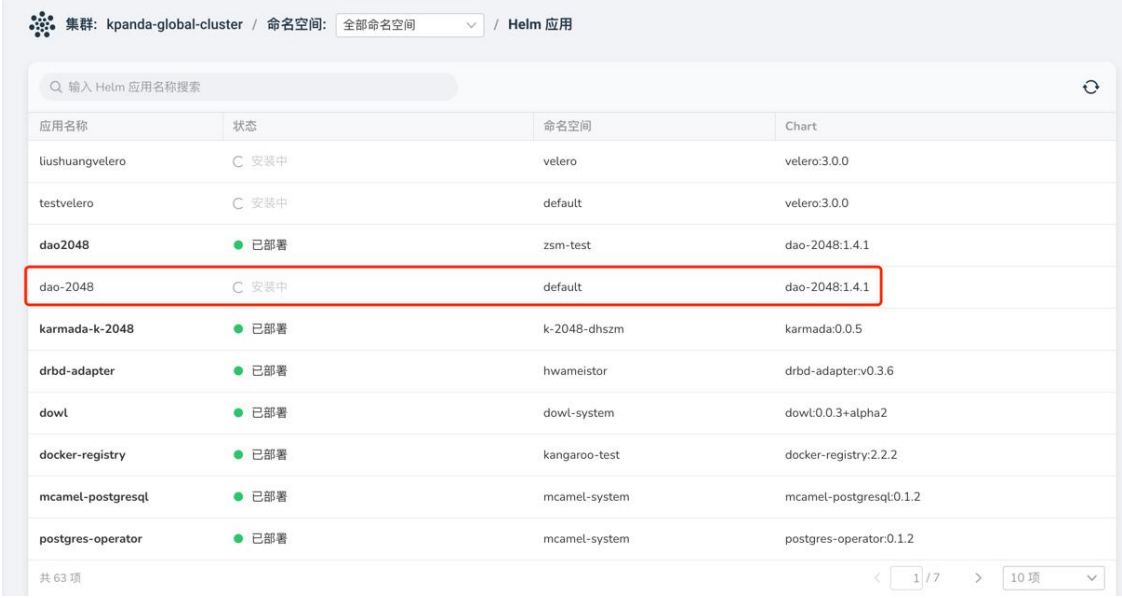
模板

此时，只需要去 Helm 仓库刷新对应的 Helm 仓库即可。



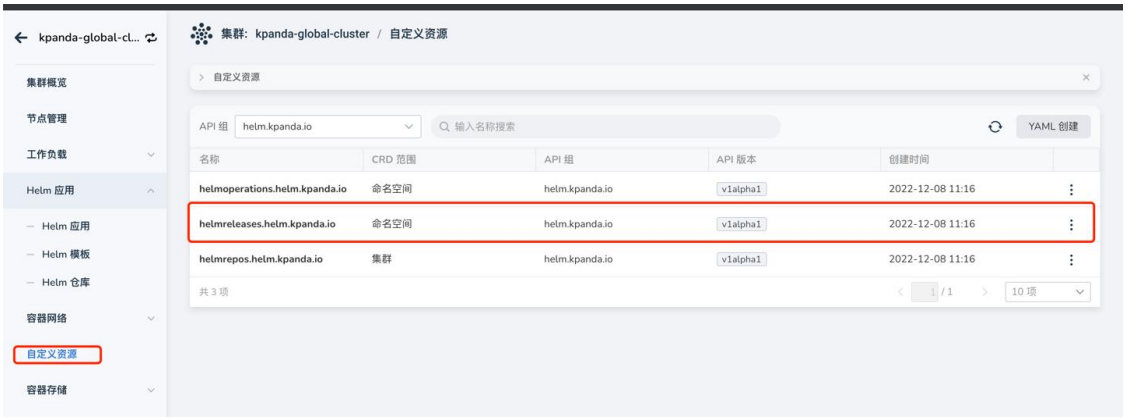
刷新仓库

5. Helm 安装应用失败时卡在安装中无法删除应用重新安装，如图：

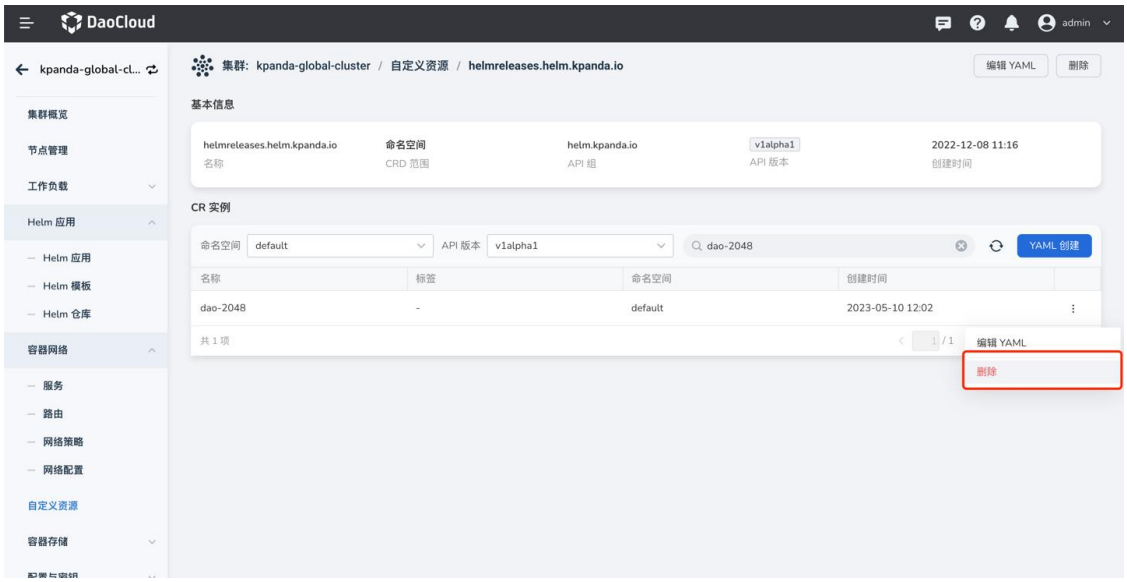


删除失败

此时，只需要去自定义资源页面，找到 helmreleases.helm.kpanda.io CRD，然后找到对应的 helmreleases CR 删除即可。



找到 CR



删除 CR

6. 工作负载–删除节点亲和性等调度策略后，调度异常，如图：



调度异常

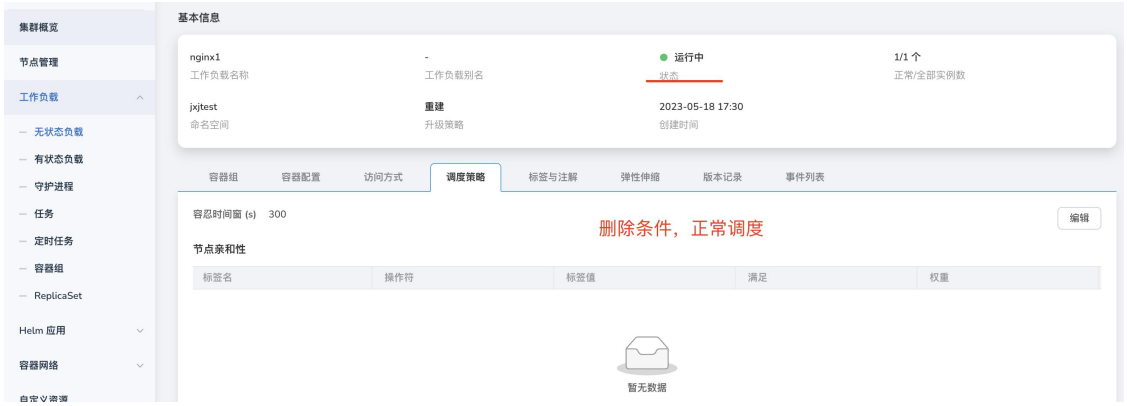
此时，可能是因为策略没有删除干净，点击编辑，删除所有策略。



编辑

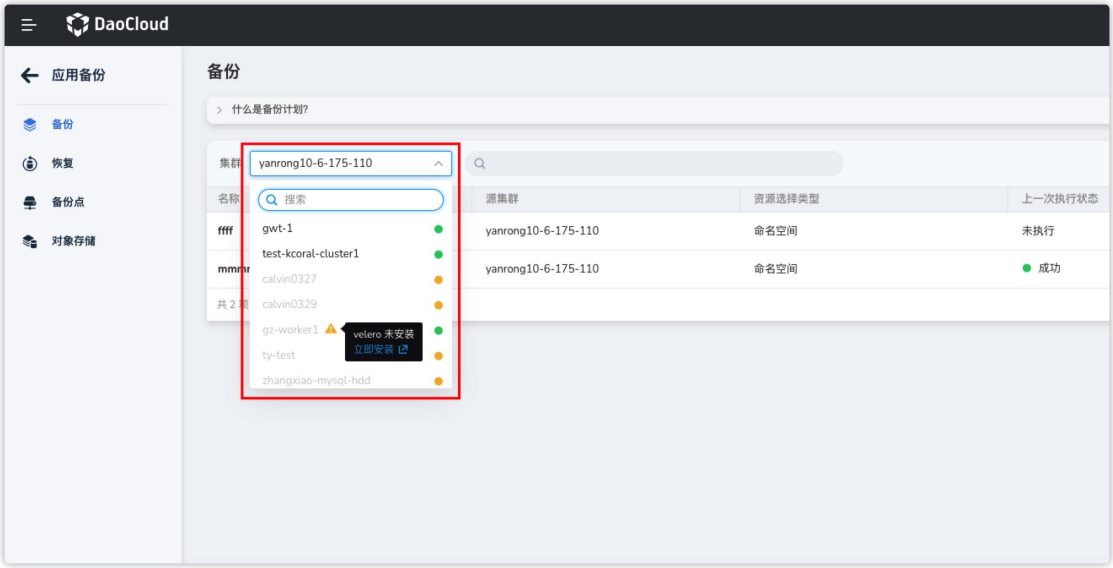


删除



正常调度

7. Kcoral 检测工作集群 Velero 状态的逻辑是什么?

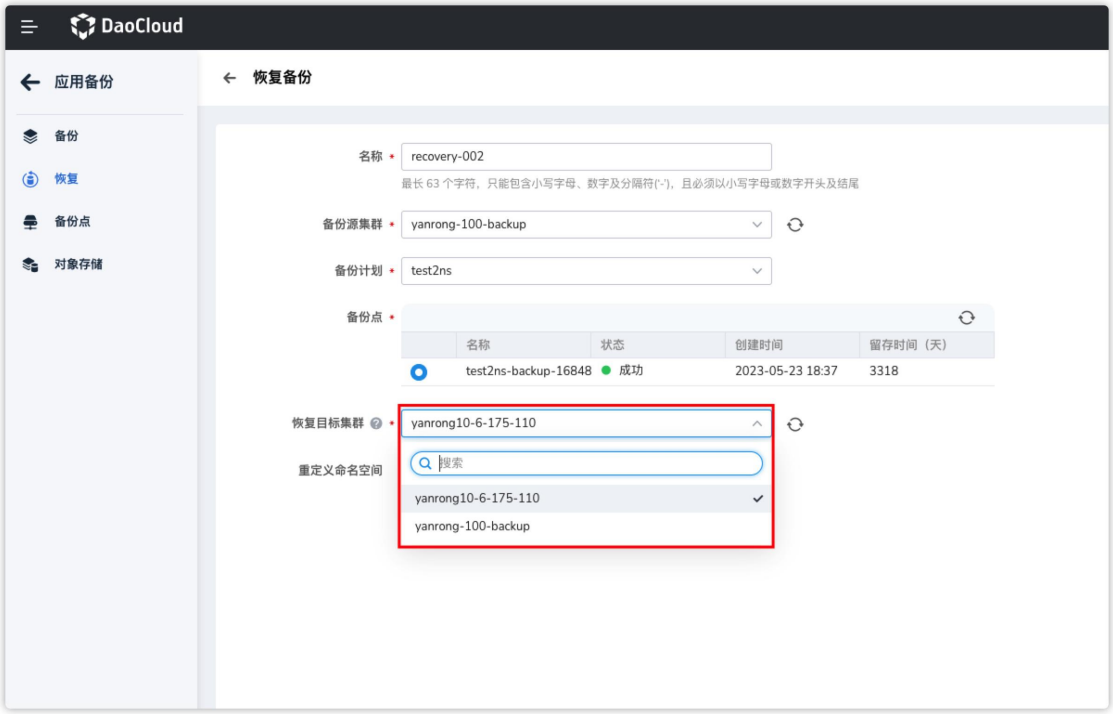


检测

- 工作集群在 velero 命名空间下安装了标准的 velero 组件
- velero 控制面 velero deployment 处于运行状态，并达到期望的副本数
- velero 数据面 node agent 处于运行状态，并达到期望副本数
- velero 成功连接到目标 MinIO（BSL 状态为 Available）

8. 在跨集群备份还原的时候，Kcoral 如何获取可用集群?

在通过 Kcoral 跨集群备份还原应用的时候，在恢复页面中，Kcoral 会帮助用户筛选可以执行跨集群还原的集群列表，逻辑如下：



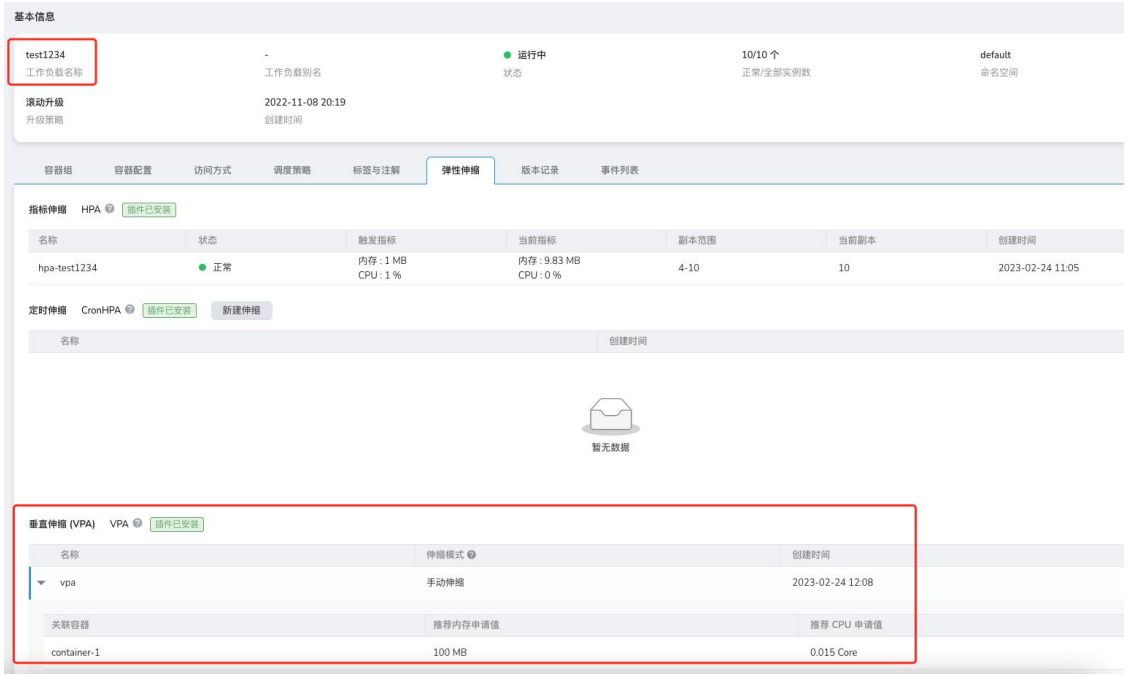
筛选

- 过滤未安装 Velero 的集群列表
- 过滤 Velero 状态异常的集群列表
- 获取与目标集群对接了相同 MinIO 和 Bucket 的集群列表并返回

所以只要对接了相同的 MinIO 和 Bucket，Velero 处于运行状态，就可以跨集群备份（需要有写入权限）和还原。

9. 卸载 VPA，HPA，CronHPA 之后，为什么对应弹性伸缩记录依然存在?

   虽然通过 Helm Addon 市场中把对应组件卸载，但是应用弹性伸缩界面相关记依然在，如下图所示:



编辑

这是 helm uninstall 的一个问题，它并不会卸载对应的 CRD，因此导致数据残留，此时我们需要手动卸载对应的 CRD，完成最终清理工作。

10. 为什么低版本集群的控制台打开异常?

    在 kubernetes 低版本（v1.18 以下）的集群中，打开控制台出现 csr 资源请求失败。打开控制台的时候，会根据当前登录用户在目标集群中通过 csr 资源申请证书，如果集群版本太低或者没有开启此功能 controller，会导致证书申请失败，从而无法连接到目标集群。

    申请证书流程请参考：https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/

    解决方案：

    - 如果集群版本大于 v1.18，请检查 kube-controller-manager 是否开启 csr 功能，确保以下的 controller 是否正常开启

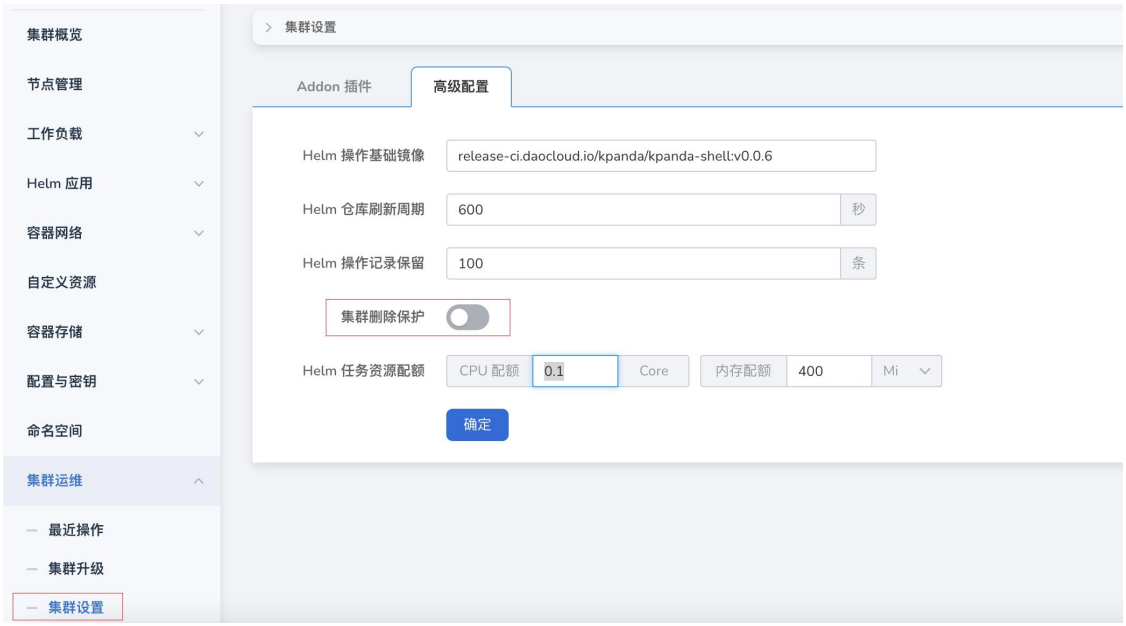
      ttl-after-finished,bootstrapsigner,csrapproving,csrcleaner,csrsigning

- 低版本集群目前解决方案只有升级版本

11. 如何重置创建的集群?

    创建的集群分为两种情况：

    - 创建失败的集群：在创建集群的过程中，因为参数设置错误导致集群创建失败，这种情况可以在安装失败的集群选择重试，然后重新设置参数重新创建。
    - 已经成功创建的集群：这种集群可以先卸载集群，然后重新创建集群。卸载集群需要关闭集群保护的功能才能卸载集群。



关闭集群保护


输入集群名称搜索
接入集群
创建集群
chenwen-test001
运行中
集群别名 chenwen-test001
集群角色 工作集群
Kubernetes 版本 v1.25.4
网络模式 Calico
被纳管 kpanda-global-cluster
创建时间 2023-05-31 16:52
8.2%
CPU 使用率
22.7%
内存使用率
控制台
监控
基础配置
下载 kubeconfig
解除接入
卸载集群
cilium02-qp
运行中
集群别名 -
集群角色 接入集群
接入时间 2023-05-31 14:53
网络模式 Cilium
Kubernetes 版本 v1.25.4
Insight 未安装
立即安装
CPU 使用率
内存使用率
1 / 1
正常/节点总数

卸载集群

什么是容器管理？

容器管理是基于 Kubernetes 开源技术构建的面向云原生应用的容器管理平台，基于原生多集群架构，解耦底层基础设施平台，实现多云与多集群统一化管理，简化企业的应用上云流程，降低运维管理和人力成本。便捷创建 Kubernetes 集群，帮助企业快速搭建企业级的容器云平台。容器管理模块的主要功能如下：

集群管理

- 集群的统一纳管，支持特定版本范围内的任意 Kubernetes 集群纳入容器管理范围，实现云上、云下、多云、混合云容器云平台的统一管理。
- 集群的快速部署，基于 DaoCloud 自主开源项目 Kubean 支持通过 Web UI 界面快速部署企业级的 Kubernetes 集群，快速搭建企业级容器云平台，适配物理机和虚拟机底层环境。
- 一键式集群升级，一键升级 Kubernetes 版本，统一管理系统组件升级。
- 集群高可用，内置集群容灾、备份能力，保障业务系统在主机故障、机房中断、自然灾害等情况下可恢复，提高生产环境的稳定性，降低业务中断风险。
- 集群的全生命周期管理，实现自建云原生集群的全生命周期管理。
- 开放式 API 能力，提供原生的 Kubernetes OpenAPI 能力。

应用管理

- 一站式部署，解耦底层 Kubernetes 平台，一站式部署和运维业务应用，实现应用的全生命周期管理。
- 应用负载的弹性伸缩，支持应用负载的手动/自动扩缩容，支持横向伸缩、纵向伸缩、以及定时伸缩，从容应对流量高峰。
- 应用的全生命周期，支持应用查看、更新、删除、回滚、事件查看以及升级等全生命周期管理。

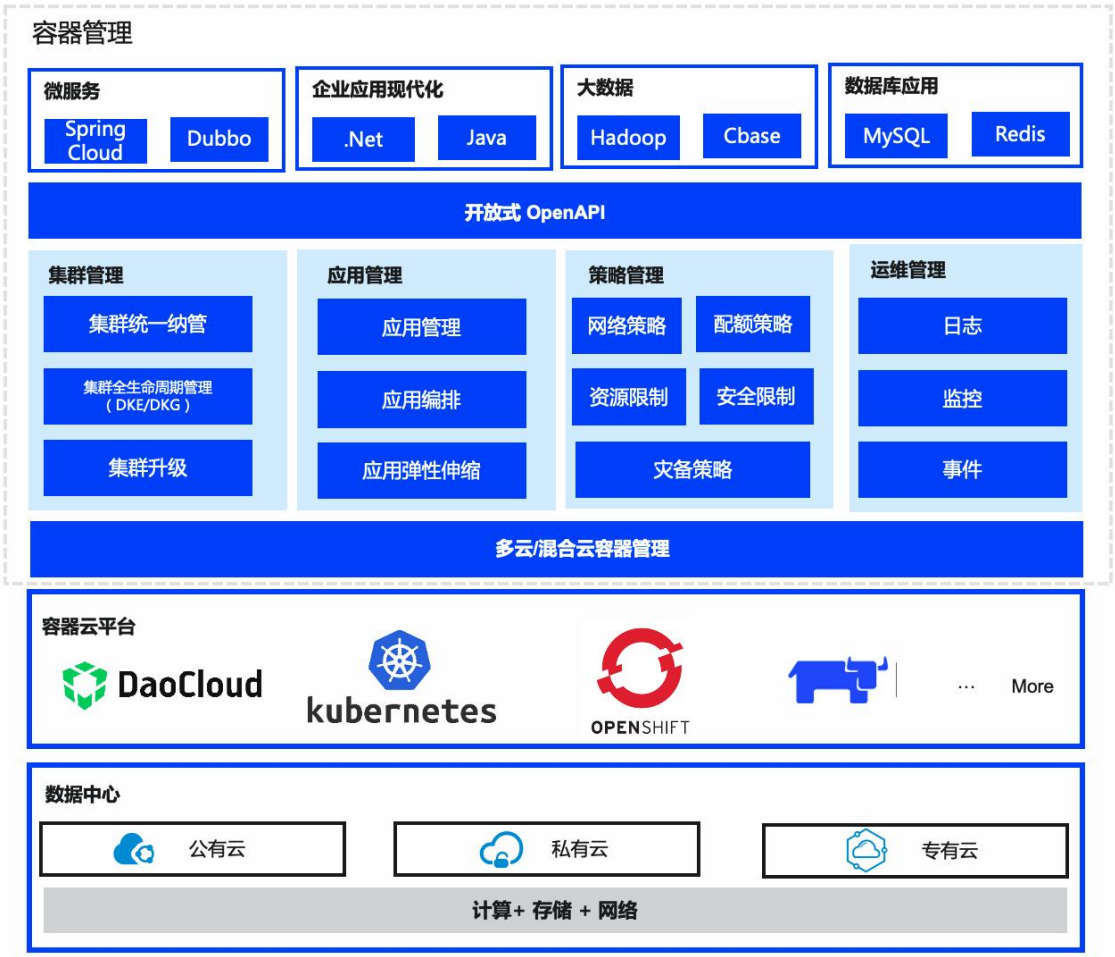
- 跨集群负载统一管理能力。

策略管理

支持以命名空间或集群粒度制定网络策略、配额策略、资源限制策略、灾备策略、安全策略。

- 网络策略，支持以命名空间或集群粒度制定网络策略，限定容器组与网络平上网络”实体“通信规则。
- 配额策略，支持以命名空间或集群粒度设定配额策略，限制集群内的命名空间的资源使用。
- 资源限制策略，支持以命名空间或集群粒度设定资源限制策略，约束对应命名空间内应用对资源的使用。
- 灾备策略，支持以命名空间或集群粒度设定灾备策略，实现以命名空间为维度进行容灾备份，保障集群的安全性。
- 安全策略，支持以命名空间或集群粒度设定安全策略，为 Pod 定义不同的隔离级别。

产品逻辑架构



逻辑架构图

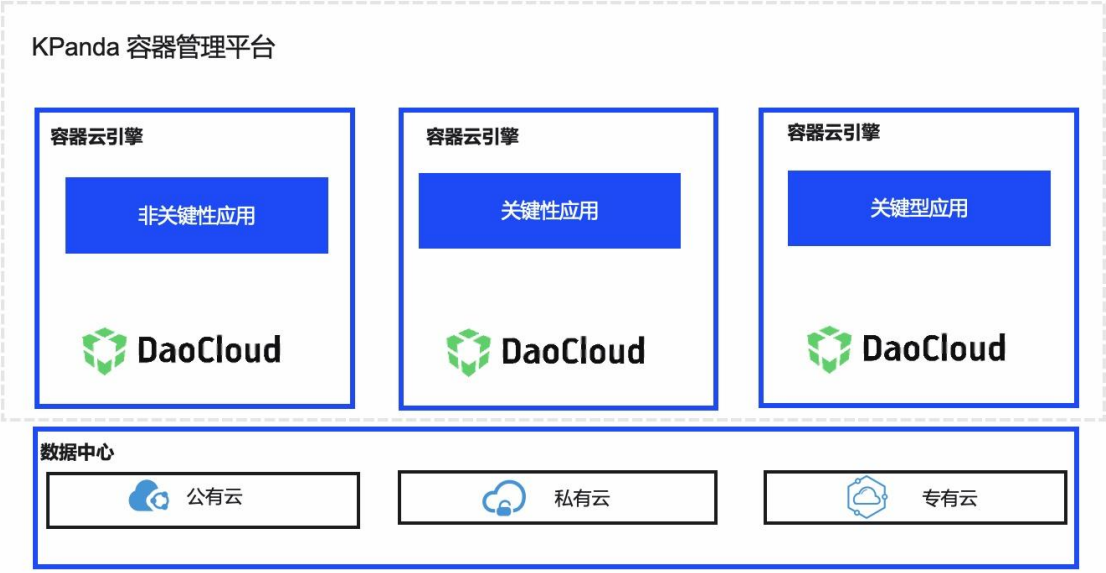
下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

适用场景

应用的多云部署及跨云容灾

多云组合部署

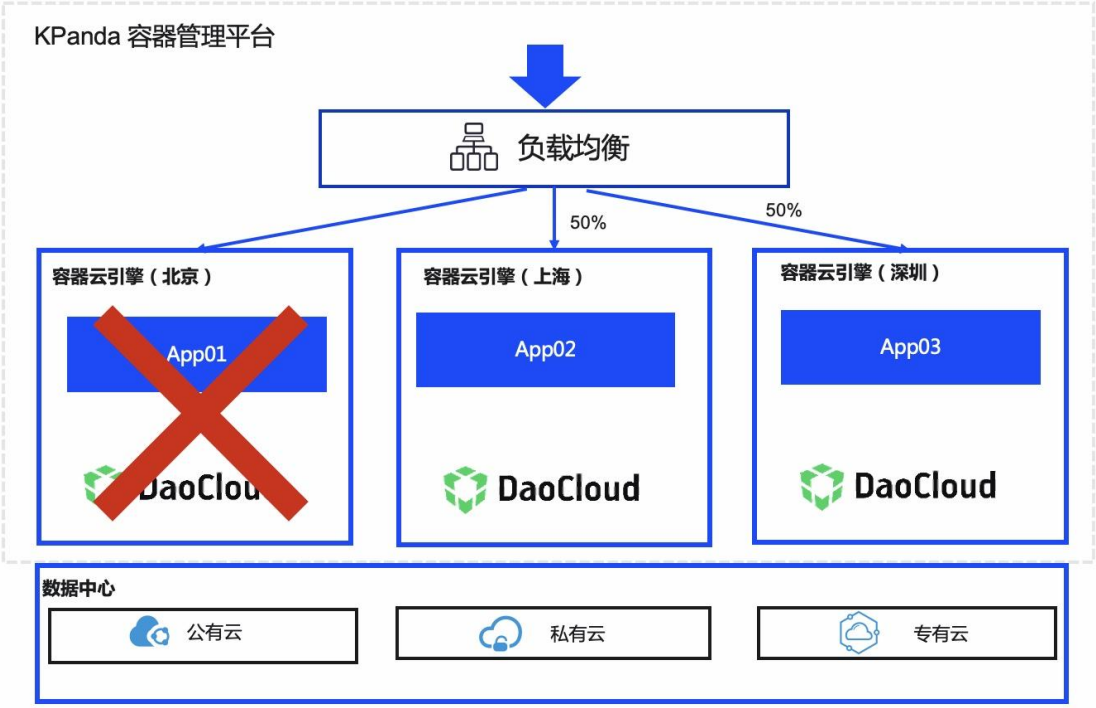
根据不同性价比的云平台组合，结合不同应用场景，进行多云组合，降低总体成本。对于金融等安全性要求高的行业用户，基于业务数据的安全性和敏感性要求，将部分业务应用部署在私有云环境中，而将非敏感性应用部署在云上集群中，并进行统一管理。



场景一

跨云容灾备份

为保证业务高可用，同时将业务部署在不同地域的多个云容器平台上，帮助应用实现多地域的流量分发，并且实现跨云应用在同一平台管理，减少运维成本。另外当某个云容器平台发生故障时，通过统一流量分发的机制，自动的将业务流量切换到其他云容器平台上。



场景二

跨云集群统一管理

跨云集群统一纳管

无论基于不同基础设施环境（公有云，私有云和混合云），或者不同容器云厂商搭建的 Kubernetes 平台，容器管理模块可统一纳管。

降低因不同基础设施、不同云提供商带来的额外管理成本，统一管理流程，降低成本。

跨云集群统一运维

不同的云容器提供商基于不同的 Kubernetes 发行版为客户提供不同云平台监控，实现统一平台监控运维难度大，无法便捷地了解整体运行状况，掌控故障处理进程和结果，基于容器管理模块，可基于集群的统一纳管实现一体化监控服务，实现多维度、跨云的统一监控运维。



场景三

弹性扩缩应对流量高峰

集群的弹性扩缩容

应对企业客户应对流量高峰，当业务应用部署所在 Kubernetes 容器平台资源已经达到阈值，需要结合底层 IaaS 弹性扩缩容能力，自动弹性扩缩容集群节点，从而应对超大流量的业务高峰，并在业务低谷期进行缩容，节约成本。

应用的弹性扩缩容

电商客户遇到双 11 等促销、限时秒杀等活动中，业务访问量迅速上升，需及时扩容应用计算资源，并根据用户设定的扩缩容策略自动调整，业务容器组数量随业务访问量上升而增加，随业务访问量下降而减少。

容器管理功能总览

本页列出了容器管理的主要功能特性。

集群全生命周期管理

- 集群的统一纳管：支持所有特定版本范围内的任意 Kubernetes 集群纳入容器管理范围，实现云上、云下、多云、混合云容器云平台的统一管理。
- 集群的快速创建：
  - 基于 DaoCloud 自主开源项目 Kubean 支持通过 Web UI 界面快速部署企业级的 Kubernetes 集群，快速搭建企业级容器云台，适配物理机和虚拟机底层环境。
  - 支持接入/创建集群，帮助用户构建一站式基础设施管理平台。
  - 支持创建集群时指定运行时类型，支持 containerd、Docker 等多种运行时。
- 一键式集群升级：一键升级自建容器云平台的 Kubernetes 版本，统一管理系统组件升级。
- 集群高可用：内置集群容灾、备份能力，保障业务系统在主机故障、机房中断、自然灾害等情况下可恢复，提高生产环境的稳定性，降低业务中断风险。
- 证书热更新：支持 Web 界面一键式热更新 Kubernetes 证书。
- 节点管理：支持自建集群增删节点，保障集群能够满足业务需求。

云原生应用负载

- 应用全生命周期管理：支持基于 Kubernetes 原生的工作负载类型部署和管理能力，包括创建、配置、监控、扩容、升级、回滚、删除等全生命周期管理。
- 一站式应用负载创建：解耦底层 Kubernetes 平台，一站式创建和运维业务工作负载。

- 跨集群应用负载管理：跨集群负载统一管理、高效检索能力。
- 应用负载的扩缩容：通过界面可实现应用负载的手动/自动扩缩容能力，自定义扩缩容策略，应对流量高峰。
- 容器的生命周期设置：支持创建工作负载时设置回调函数、启动后参数、停止前参数，满足特定场景需求。
- 容器就绪检查、存活检查设置：支持部署应用时设定工作负载就绪检查及存活检查：
    - 工作负载就绪检查：用于检测用户业务是否就绪，如未就绪，则不转发流量至当前实例。
    - 工作负载存活检查：用户检测容器是否正常，如异常，集群会执行容器重启操作。
- 容器的环境变量设置：可为业务容器运行环境设定指定环境变量。
- 容器的自动调度：支持应用服务调度管理，自动按照主机资源用量进行容器调度，可以指定具体的部署主机，以及通过标签策略进行容器调度。
- 亲和性和反亲和性：支持定义容器组间调度的亲和性和反亲和性；以及容器组与节点间的亲和性和反亲和性，满足业务自定义调度需求。
- 容器安全用户设置：支持设定容器运行用户，如以 Root 权限运行则填写 Root 用户 ID 0。
- 自定义资源（CRD）支持：支持自定义资源的创建、配置、删除等全生命周期管理。

服务与路由

服务（Service ）是 Kuberetes 原生资源，提供云原生的负载均衡能力，通过固定的 IP 地址和端口进行访问。目前支持的 Service 类型包括：

- 集群内访问（ClusterIP）：仅在集群内访问服务。
- 节点访问（NodePort）：使用节点 IP + 服务端口的方式进行访问。

命名空间管理

- 命名空间管理：支持命名空间创建、配额设置、资源限制设置等。

- 跨集群命名空间管理：支持跨集群命名空间统一管理及高效检索能力，实现多云场景、灾备场景下命名空间的管理能力。

容器存储

- 数据卷管理：支持本地存储、文件存储、块存储，通过 CSI 能力接入，并提供给应用工作负载使用。
- 数据卷动态创建：可支持存储池实现动态创建数据卷。

策略管理

- 策略统一管理和下发：支持以命名空间或集群粒度制定网络策略、配额策略、资源限制策略、灾备策略、安全策略等策略，并支持以集群/命名空间为粒度进行策略下发。
- 网络策略：支持以命名空间或集群粒度制定网络策略，限定容器组与网络平上网络”实体“通信规则。
- 配额策略：支持以命名空间或集群粒度设定配额策略，限制集群内的命名空间的资源使用。
- 资源限制策略：支持以命名空间或集群粒度设定资源限制策略，约束对应命名空间内应用使用资源限制。
- 灾备策略：支持以命名空间或集群粒度设定灾备策略，实现以命名空间为维度进行容灾备份，保障集群的安全性。
- 安全策略：支持以命名空间或集群粒度设定安全策略，为 Pod 定义不同的隔离级别。

扩展插件

提供了丰富的系统插件，扩充云容器集群功能。扩展插件包括：DNS、HPA 等。

权限管理

支持命名空间授权，通过权限设置可以让不同的用户或用户组拥有操作指定命名空间下不同的 Kubernetes 资源的权限。

集群运维

- 全方位集群监控：全方位覆盖集群、节点的指标监控及告警，实时了解和查看集群和节点状态，及时实施运维措施，保障业务连续性。
- 开放式 API：提供原生的 Kubernetes OpenAPI 能力。
- CloudShell 访问集群：支持通过 CloudShell 连接集群并通过 Kubectl 访问集群。

参阅容器管理功能发布动态。

离线升级容器管理模块

本页说明从下载中心下载容器管理模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `kpanda` 字样是容器管理模块的内部开发代号。

从下载的安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

方式一：使用 chart-syncer 同步镜像

使用 chart-syncer 可以将您下载的安装包中的 chart 及其依赖的镜像包上传至安装器部署 DCE 时使用的镜像仓库和 helm 仓库。

首先找到一台能够连接镜像仓库和 helm 仓库的节点（如火种节点），在节点上创建 load-image.yaml 配置文件，填入镜像仓库和 helm 仓库等配置信息。

1. 创建 load-image.yaml

   !!! note

   该 YAML 文件中的各项参数均为必填项。

   ===“已添加 Helm repo”

   若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

   ```yaml title="load-image.yaml"
   source:
     intermediateBundlesPath: kpanda # 节点上执行 load-image.yaml 文件的
   ```

路径。

```
  target:
    containerRegistry: 10.16.10.111 # 镜像仓库地址
    containerRepository: release.daocloud.io/kpanda # 镜像仓库路径
    repo:
      kind: HARBOR # Helm Chart 仓库类别
      url: http://10.16.10.111/chartrepo/release.daocloud.io # Helm 仓库地址
      auth:
      username: "admin" # 镜像仓库用户名
      password: "Harbor12345" # 镜像仓库密码
    containers:
      auth:
        username: "admin" # Helm 仓库用户名
        password: "Harbor12345" # Helm 仓库密码
```

===“未添加 Helm repo”

若当前节点上未添加 helm repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kpanda # 节点上执行 load-image.yaml 文件的路径。
target:
  containerRegistry: 10.16.10.111 # 镜像仓库 url
  containerRepository: release.daocloud.io/kpanda # 镜像仓库路径
  repo:
    kind: LOCAL
    path: ./local-repo # chart 本地路径
  containers:
    auth:
```

```
        username: "admin" # 镜像仓库用户名
        password: "Harbor12345" # 镜像仓库密码
```

2. 执行同步镜像命令。

    charts-syncer sync --config load-image.yaml

方式二：使用 Docker 或 containerd 加载镜像

解压并加载镜像文件。

1. 解压 tar 压缩包。

    tar xvf kpanda.bundle.tar

    解压成功后会得到 3 个文件：

    - hints.yaml
    - images.tar
    - original-chart

2. 从本地加载镜像到 Docker 或 containerd。

    === “Docker”

    ```shell
    docker load -i images.tar
    ```

    === “containerd”

    ```shell
    ctr image import images.tar
    ```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

===“通过 helm repo 升级”

1. 检查容器管理 helm 仓库是否存在。

    ```shell
    helm repo list | grep kpanda
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加容器管理的 helm 仓库。

    ```shell
    heml repo add kpanda http://{harbor url}/chartrepo/{project}
    ```

1. 更新容器管理的 helm 仓库。

    ```shell
    helm repo update kpanda
    ```

1. 选择您想安装的容器管理版本（建议安装最新版本）。

    ```shell
    helm search repo kpanda/kpanda --versions
    ```

    ```none
    [root@master ~]# helm search repo kpanda/kpanda --versions
    NAME                    CHART VERSION   APP VERSION   DESCRIPTION
    kpanda/kpanda   0.20.0          v0.20.0       A Helm chart for kpanda
    ...
    ```

1. 备份 `--set` 参数。

    在升级容器管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

    ```shell
    helm get values kpanda -n kpanda-system -o yaml > bak.yaml
    ```

1. 更新 kpanda crds

    ```shell
    helm pull kpanda/kpanda --version 0.21.0 && tar -zxf kpanda-0.21.0.tgz
    kubectl apply -f kpanda/crds
    ```

1. 执行 `helm upgrade`。

    升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像

仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kpanda kpanda/kpanda \
  -n kpanda-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.21.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

   在升级容器管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values kpanda -n k pan da-system -o yaml > bak.yaml
   ```

1. 更新 kpanda crds

   ```shell
   kubectl apply -f ./crds
   ```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kpanda . \
  -n kpanda-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

应用使用 MIG GPU

本节介绍应用如何使用 MIG GPU 资源。

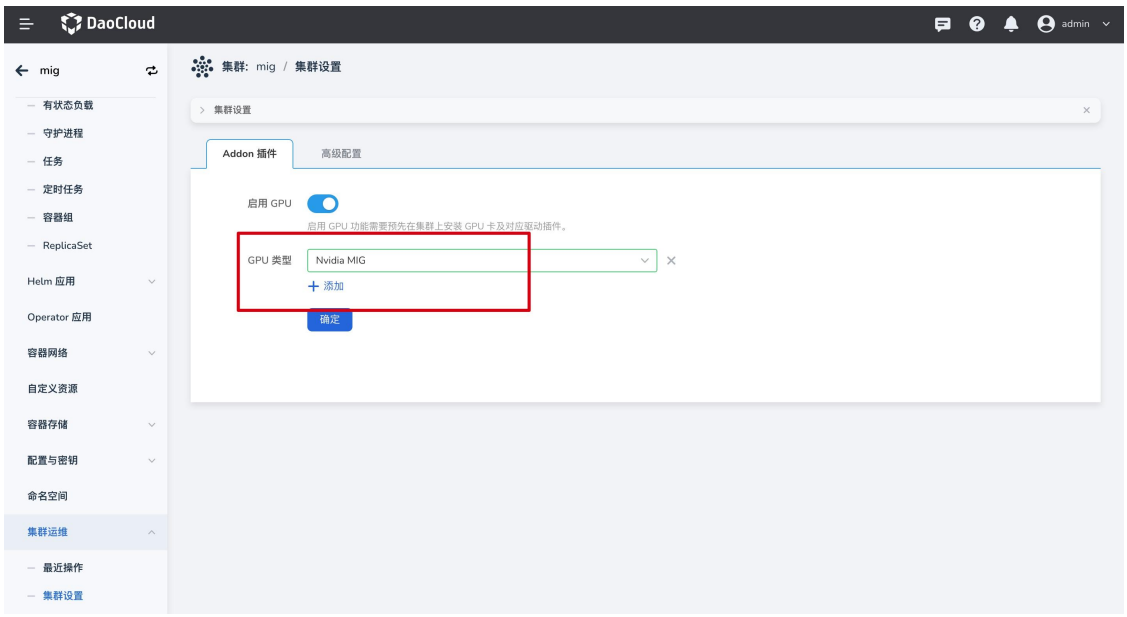
前提条件

- 已安装 Nvidia Driver，可参考 Nvidia GPU 驱动安装
- 已开启 MIG 能力 GPU Operator 创建 MIG
- 集群节点已插入 GPU MIG 卡

通过界面使用 MIG GPU

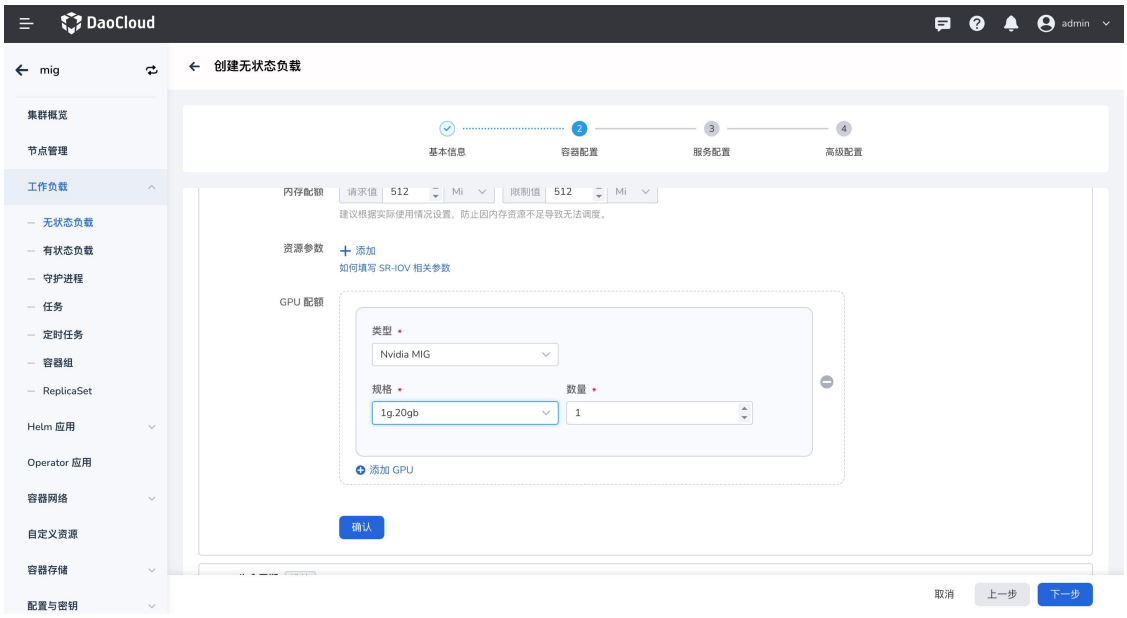
1. 确认集群是否已识别 GPU 卡类型

   进入集群详情 -> 集群设置 -> Addon 设置，查看是否已正确识别，自动识别频率为 10 分钟 。



   gpu

2. 通过镜像部署应用可选择并使用 Nvidia MIG 资源。



mig02

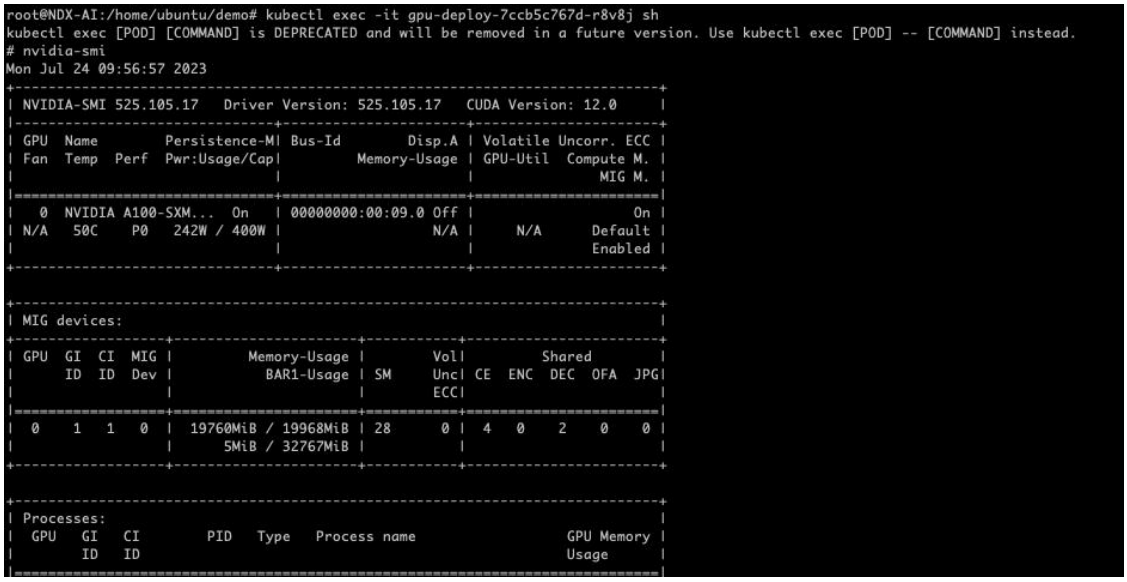
- 开启 MIG Single 模式时，添加的资源信息应为：

```
resources:
  limits:
  nvidia.com/gpu: 2
```

- 开启 MIG Mixed 模式时，添加的资源信息应为：

```
resources:
  limits:
  nvidia.com/mig-4g.20gb: 1 # 通过 nvidia.com/mig-<slice_count>g.<memory_size>gb 的资源类型公开各个 MIG 设备
```

3. 进入容器后可以查看只使用了一个 MIG 设备。


root@NDX-AI:/home/ubuntu/demo# kubectl exec -it gpu-deploy-7ccb5c767d-r8v8j sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
nvidia-smi
Mon Jul 24 09:56:57 2023
NVIDIA-SMI 525.105.17 Driver Version: 525.105.17 CUDA Version: 12.0
GPU Name Persistence-M Bus-Id Disp.A Volatile Uncorr. ECC
Fan Temp Perf Pwr:Usage/Cap Memory-Usage GPU-Util Compute M.
MIG M.
0 NVIDIA A100-SXM... On 00000000:00:09.0 Off On
N/A 50C P0 242W / 400W N/A N/A Default
Enabled
MIG devices:
GPU GI CI MIG Memory-Usage Vol Shared
ID ID Dev BAR1-Usage SM Unc CE ENC DEC OFA JPG
ECC
0 1 1 0 19760MiB / 19968MiB 28 0 4 0 2 0 0
5MiB / 32767MiB
Processes:
GPU GI CI PID Type Process name GPU Memory
ID ID Usage

mig03

GPU Operator 创建 MIG（多实例 GPU）

NVIDIA 当前提供两种在 Kubernetes 节点上公开 MIG 设备的策略：

- **Single 模式**，节点仅在其所有 GPU 上公开单一类型的 MIG 设备，节点上的所有 GPU 必须：
  - 属于同一个型号（例如 A100-SXM-40GB），只有同一型号 GPU 的 MIG Profile 才是一样的
  - 启用 MIG 配置，需要重启机器才能生效
  - 为在所有产品中公开“完全相同”的 MIG 设备类型，创建相同的 GI 和 CI
- **Mixed 模式**，节点在其所有 GPU 上公开混合 MIG 设备类型。请求特定的 MIG 设备类型需要设备类型提供的计算切片数量和内存总量。
  - 节点上的所有 GPU 必须：属于同一产品线（例如 A100-SXM-40GB）
  - 每个 GPU 可启用或不启用 MIG，并且可以自由配置任何可用 MIG 设备类型的混合搭配。
  - 在节点上运行的 k8s-device-plugin 将：
    - 使用传统的 nvidia.com/gpu 资源类型公开任何不处于 MIG 模式的 GPU
    - 使用遵循架构 nvidia.com/mig-g.gb 的资源类型公开各个 MIG 设备

注意：当前 GPU Operator 仅支持在线方式部署。

前提条件

- 确认集群节点上具有对应型号的 GPU 卡（NVIDIA H100、 A100 和 A30 Tensor Core GPU）
- 当前集群可联通网络

开启 GPU MIG Single 模式

1. 通过 Operator 开启 MIG Single 模式

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
helm upgrade -i  gpu-operator -n gpu-operator -n gpu-operator --create-namespace nvidia/gpu-operator --set migStrategy=single --set node-feature-discovery.image.repository=k8s.m.daocloud.io/nfd/node-feature-discovery --set driver.version=525-5.15.0-78-generic # 通过 set 指定 MIG 模式为 Single
```

2. 给对应节点(已插入对应 GPU 卡节点)打上 切分规格

```
kubectl label nodes {node} nvidia.com/mig.config="all-1g.10gb" --overwrite
```

3. 查看配置结果

```
kubectl get node 10.206.0.17 -o yaml|grep nvidia.com/mig.config
```

开启 GPU MIG Mixed 模式

1. 通过 Operator 开启 MIG Mixed 模式

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
helm install --generate-name --set migStrategy=mixed --set allowDefaultNamespace=true  nvidia/nvidia-device-plugin --set node-feature-discovery.image.repository=k8s.m.daocloud.io/nfd/node-feature-discovery # 通过 set 指定 MIG 模式为 Mixed
```

2. 设置 config.yaml 文件，此 config 文件中设置 MIG GI 实例切分规格。

```
version: v1
mig-configs:
  all-disabled:
    - devices: all
```

```
    mig-enabled: false
all-enabled:
  - devices: all
    mig-enabled: true
    mig-devices: {}
all-1g.10gb:
  - devices: all
    mig-enabled: true
    mig-devices:
      1g.5gb: 7
all-1g.10gb.me:
  - devices: all
    mig-enabled: true
    mig-devices:
      1g.10gb+me: 1
all-1g.20gb:
  - devices: all
    mig-enabled: true
    mig-devices:
      1g.20gb: 4
all-2g.20gb:
  - devices: all
    mig-enabled: true
    mig-devices:
      2g.20gb: 3
all-3g.40gb:
  - devices: all
    mig-enabled: true
    mig-devices:
      3g.40gb: 2
all-4g.40gb:
  - devices: all
```

```
      mig-enabled: true
      mig-devices:
        4g.40gb: 1
  all-7g.80gb:
    - devices: all
      mig-enabled: true
      mig-devices:
        7g.80gb: 1
  all-balanced:
    - device-filter: ["0x233110DE", "0x232210DE", "0x20B210DE", "0x20B
510DE", "0x20F310DE", "0x20F510DE"]
      devices: all
      mig-enabled: true
      mig-devices:
        1g.10gb: 2
        2g.20gb: 1
        3g.40gb: 1
  custom-config:      # 设置后会按照设置规格切分 CI 实例
    - devices: all
      mig-enabled: true
      mig-devices:
        3g.40gb: 2
```

3. 在上述的 config.yaml 中设置 custom-config，设置后会按照规格切分 CI 实例

```
custom-config:
  - devices: all
    mig-enabled: true
    mig-devices:
      1c.3g.40gb: 6
```

4. 按照修改后的 config.yaml 自定义设置：

```
kubectl create configmap -n gpu-operator custome-mig-parted-config --from-file=config.yaml
helm install gpu-operator -n gpu-operator --create-namespace nvidia/gpu-operator --set migManager.config.name=mig-config --set node-feature-discovery.image.repository=k8s.m.daocloud.io/nfd/node-feature-discovery
...
```

或者

```
...sh
helm upgrade -i gpu-operator nvidia/gpu-operator -n gpu-operator --set mig.strategy=mixed --set migManager.config.name=custome-mig-parted-config --set node-feature-discovery.image.repository=k8s.m.daocloud.io/nfd/node-feature-discovery
```

5. 设置完成后给对应的节点打上切分规格

```
kubectl label nodes {node} nvidia.com/mig.config="custom-config" --overwrite
```

6. 查看配置结果

```
kubectl get node 10.206.0.17 -o yaml|grep nvidia.com/mig.config
```

设置完成后，在确认部署应用时即可使用 GPU MIG 资源。

MIG 相关命令

GI 相关命名：

子命令	说明
nvidia-smi mig -lgi	查看创建 GI 实例列表
nvidia-smi mig -dgi -gi {Instance ID}	删除指定的 GI 实例
nvidia-smi mig -lgip	查看 GI 的 profile
nvidia-smi mig -cgi {profile id}	通过指定 profile 的 ID 创建 GI

CI 相关命令：

子命令	说明
nvidia-smi mig -lcip { -gi {gi Instance ID}}	查看 CI 的 profile，指定 -gi 可以查看特定 GI 实例可以创建的 CI
nvidia-smi mig -lci	查看创建的 CI 实例列表
nvidia-smi mig -cci {profile id} -gi {gi instance id}	指定的 GI 创建 CI 实例
nvidia-smi mig -dci -ci {ci instance id}	删除指定 CI 实例

GI+CI 相关命令：

子命令	说明
nvidia-smi mig -i 0 -cgi {gi profile id} -C {ci profile id}	直接创建 GI + CI 实例

Nvidia GPU 驱动安装

GPU 驱动安装

提供了手动安装和使用 gpu-operator 两种安装方式，使用 gpu-operator 安装会简化很多，推荐使用 gpu-operator 模式。

在 K8s 上使用 GPU 需要安装相关的驱动和程序，其中需要涉及到如下几个步骤：

- □ 安装 GPU 物理设备驱动
- □ 安装 cuda toolkit 工具
- □ 修改 container-runtime
- □ 安装 device-plugin

手动安装

1. 物理设备驱动安装

   - 添加 PPA 镜像源

     ```
     add-apt-repository ppa:graphics-drivers/ppa
     apt-get update
     ```

   - 查看可用的驱动版本

     ```
     ubuntu-drivers devices
     ```

```
vendor   : NVIDIA Corporation
driver   : nvidia-driver-510-server - distro non-free
driver   : nvidia-driver-515-open - distro non-free
driver   : nvidia-driver-510 - distro non-free
driver   : nvidia-driver-515 - distro non-free
driver   : nvidia-driver-470-server - distro non-free
driver   : nvidia-driver-450-server - distro non-free
driver   : nvidia-driver-470 - distro non-free
driver   : nvidia-driver-515-server - distro non-free
driver   : nvidia-driver-520 - distro non-free
driver   : nvidia-driver-520-open - distro non-free recommended
driver   : xserver-xorg-video-nouveau - distro free builtin
```

Alt text

- 选择合适的版本进行安装
- 安装建议版本：

  ubuntu-drivers autoinstall

- 安装指定版本：

  apt-get install nvidia-driver-XXX

- 安装 CUDA

  # https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#ubuntu
  $ apt-get install cuda

- 安装完成后重启

  nvidia-smi
  nvcc -V

- 查看驱动版本

  $ cat /proc/driver/nvidia/version
  NVRM version: NVIDIA UNIX x86_64 Kernel Module 525.105.17 Tue Mar 28 18:02:59 UTC 2023

2. 安装 cuda toolkit

   - https://github.com/NVIDIA/nvidia-container-toolkit
   - https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html
   - 运行 nvidia- container-toolkit 的条件：
     - 操作系统内核版本 > 3.10
     - Docker >= 19.03 或者使用 Containerd.
     - NVIDIA GPU 卡架构在 3.0 版本之上
     - NVIDIA Linux drivers >= 418.81.07
   - 下面以 containerd 来演示如何安装：
     - 设置 nvidia-container-runtime 到 /etc/containerd/config.toml 配置中，然后重启

```
[plugins."io.containerd.grpc.v1.cri".containerd]
default_runtime_name = "nvidia"
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
                SystemdCgroup = true
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]
            privileged_without_host_devices = false
            runtime_engine = ""
            runtime_root = ""
            runtime_type = "io.containerd.runc.v1"
            [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]
                BinaryName = "/usr/bin/nvidia-container-runtim
```

e"

SystemdCgroup = true

- 设置包仓库 nvidia-container-toolkit

  $ distribution=$(. /etc/os-release;echo IDVERSION_ID) && curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-key add - && curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

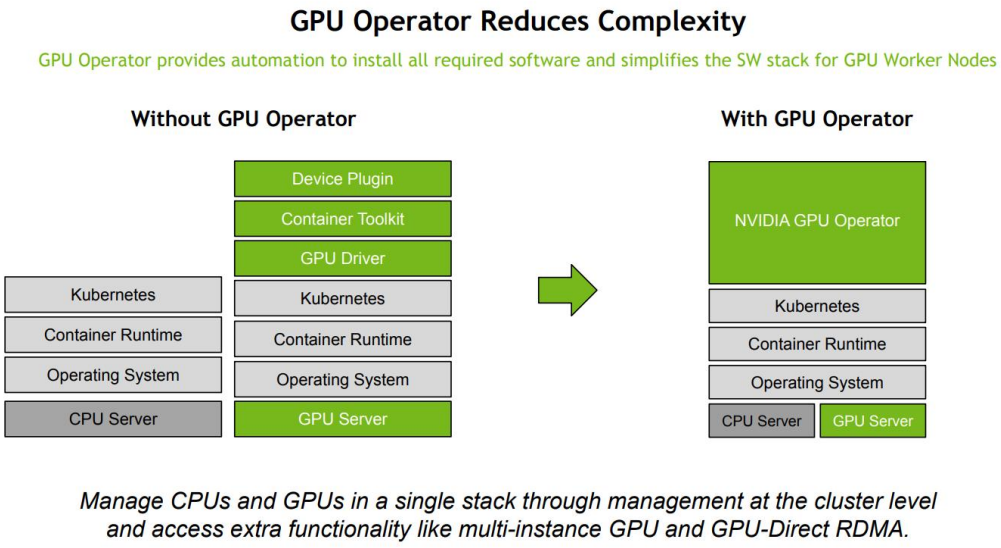
- 安装:

  sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit

3. 安装 device-plugin

   kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.14.0/nvidia-device-plugin.yml

gpu-operator 安装



Manage CPUs and GPUs in a single stack through management at the cluster level and access extra functionality like multi-instance GPU and GPU-Direct RDMA.

Alt text

GPU Operator 能够在 K8s 基础上进行 GPU 相关底层依赖的的操作，这其中就包括 GPU 驱动、Container ToolKit、 Device Plugin 资源自动上报等等能力，再也不用额外关注驱动这些糟心的事情了，理论上只要 GPU 卡插好，装上 K8s， 然后通过 GPU Operator 就能使用 所有的能力了。

- 检查 GPU 卡正常挂载

  ```
  $ lspci | grep -i nvidia
  1b:00.0 VGA compatible controller: NVIDIA Corporation GP102 [TITAN Xp] (rev a1)
  ```

- 确定 kernel 版本（这一步很重要，nvidia driver 镜像版本需要和节点内核版本强一致，包括小版本号）

  内核对应的驱动版本是否存在请去如下网站确认：
  https://catalog.ngc.nvidia.com/orgs/nvidia/containers/driver/tags

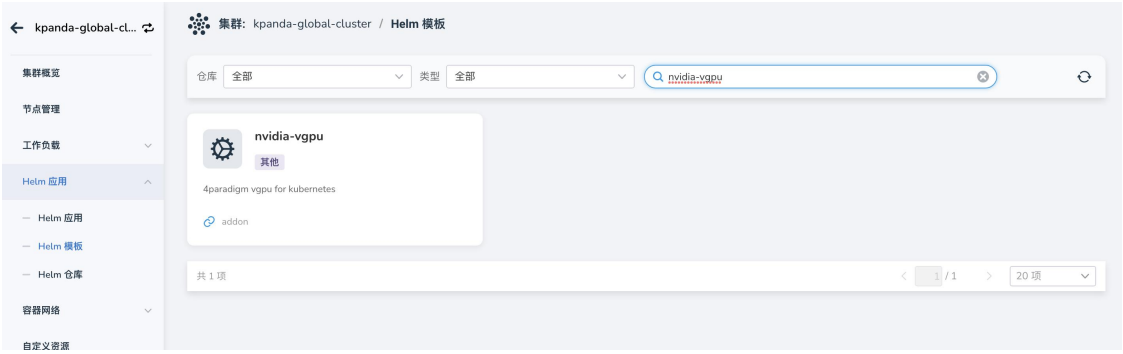
  ```
  $ hostnamectl | grep Kernel
  Kernel: Linux 5.15.0-75-generic
  ```

- GPU Operator 安装

  ```
  $ helm repo add nvidia https://helm.ngc.nvidia.com/nvidia && helm repo update
  $ helm install --wait --generate-name \
      -n gpu-operator --create-namespace \
      nvidia/gpu-operator \
      --set driver.version=525-5.15.0-69-generic
  ```

安装 vGPU 驱动

上述两个方案安装之后 pod 使用 GPU 时只能使用整卡资源，不能使用 vGPU 功能，如果需要使用 vGPU 需要在 DCE 5.0 容器管理的 Helm 仓库中查找 nvidia-vgpu 模块，并进行安装。


kpanda-global-cl...
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
容器网络
自定义资源
集群: kpanda-global-cluster / Helm 模板
仓库
全部
类型
全部
nvidia-vgpu
nvidia-vgpu
其他
4paradigm vgpu for kubernetes
addon
共 1 项
1 / 1
20 项

Alt text

NVIDIA 多实例 GPU(MIG) 概述

NVIDIA 多实例 GPU（Multi-Instance GPU，简称 MIG）是 NVIDIA 在 H100，A100，A30 系列 GPU 卡上推出的一项新特性， 旨在将一块物理 GPU 分割为多个 GPU 实例，以提供更细粒度的资源共享和隔离。MIG 最多可将一块 GPU 划分成七个 GPU 实例，使得一个 物理 GPU 卡可为多个用户提供单独的 GPU 资源，以实现最佳 GPU 利用率。

这个功能使得多个应用程序或用户可以同时共享 GPU 资源，提高了计算资源的利用率，并增加了系统的可扩展性。

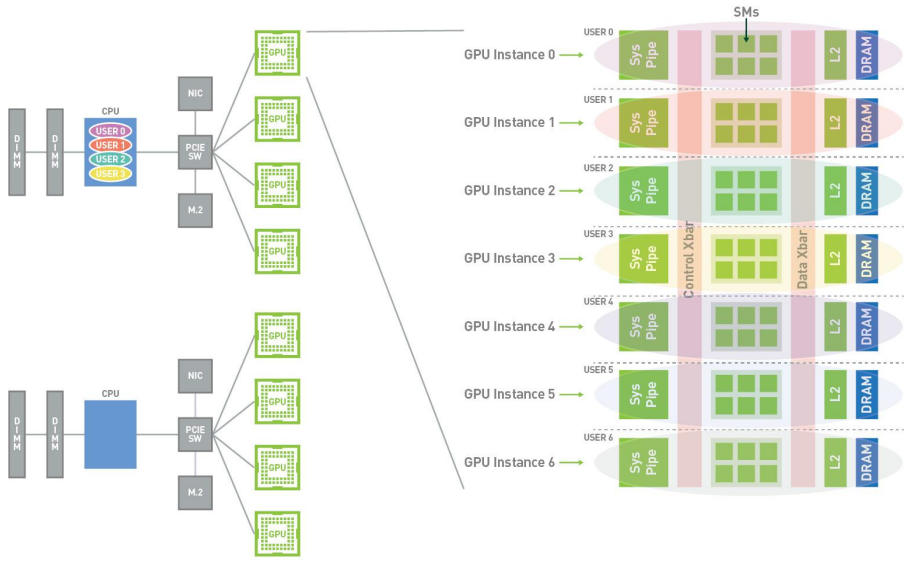
通过 MIG，每个 GPU 实例的处理器在整个内存系统中具有独立且隔离的路径——芯片上的交叉开关端口、L2 高速缓存组、内存控制器和 DRAM 地址总线都唯一分配给单个实例。

这确保了单个用户的工作负载能够以可预测的吞吐量和延迟运行，并具有相同的二级缓存分配和 DRAM 带宽。MIG 可以划分可用的 GPU 计算资源（包括流多处理器或 SM 和 GPU 引擎，如复制引擎或解码器）进行分区，以便为不同的客户端（如虚拟机、容器或进程）提供定义的服务质量（QoS）和故障隔离）。 MIG 使多个 GPU 实例能够在单个物理 GPU 上并行运行。

MIG 允许多个 vGPU（以及虚拟机）在单个 GPU 实例上并行运行，同时保留 vGPU 提供的隔离保证。有关使用 vGPU 和 MIG 进行 GPU 分区的详细信息，请参阅 NVIDIA Multi-Instance GPU and NVIDIA Virtual Compute Server。

MIG 使用场景

如下是一个 MIG 的概述图，可以看出 MIG 将一张物理 GPU 卡虚拟化成了 7 个 GPU 实例，这些 GPU 实例能够可以被多个 User 使用。



img

重要概念：

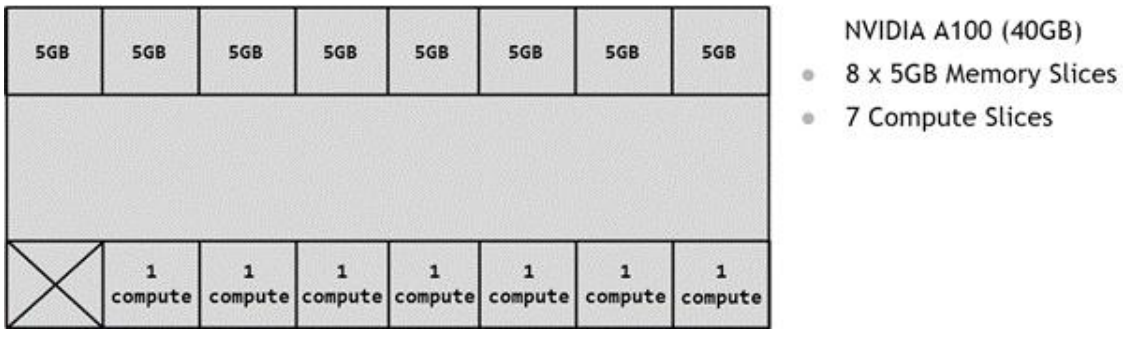
- SM：流式多处理器（Streaming Multiprocessor），GPU 的核心计算单元，负责执行图形渲染和通用计算任务。每个 SM 包含一组 CUDA 核心，以及共享内存、寄存器文件和其他资源，可以同时执行多个线程。每个 MIG 实例都拥有一定数量的 SM 和其他相关资源，以及被划分出来的显存。
- GPU Memory Slice：GPU 内存切片，GPU 内存切片是 GPU 内存的最小部分，包括相应的内存控制器和缓存。GPU 内存切片大约是 GPU 内存资源总量的八分之一，包括容量和带宽。
- GPU SM Slice：GPU SM 切片是 GPU 上 SM 的最小计算单位。在 MIG 模式下配置时，GPU SM 切片大约是 GPU 中可用 SMS 总数的七分之一。

- GPU Slice：GPU 切片是 GPU 中由单个 GPU 内存切片和单个 GPU SM 切片组合在一起的最小部分。
- GPU Instance：GPU 实例 （GI） 是 GPU 切片和 GPU 引擎（DMA、NVDEC 等）的组合。GPU 实例中的任何内容始终共享所有 GPU 内存切片和其他 GPU 引擎，但它的 SM 切片可以进一步细分为计算实例（CI）。GPU 实例提供内存 QoS。每个 GPU 切片都包含专用的 GPU 内存资源，这些资源会限制可用容量和带宽，并提供内存 QoS。每个 GPU 内存切片获得总 GPU 内存资源的 1/8，每个 GPU SM 切片获得 SM 总数的 1/7。
- Compute Instance：GPU 实例的计算切片可以进一步细分为多个计算实例（CI），其中 CI 共享父 GI 的引擎和内存，但每个 CI 都有专用的 SM 资源。

GPU 实例（GI）

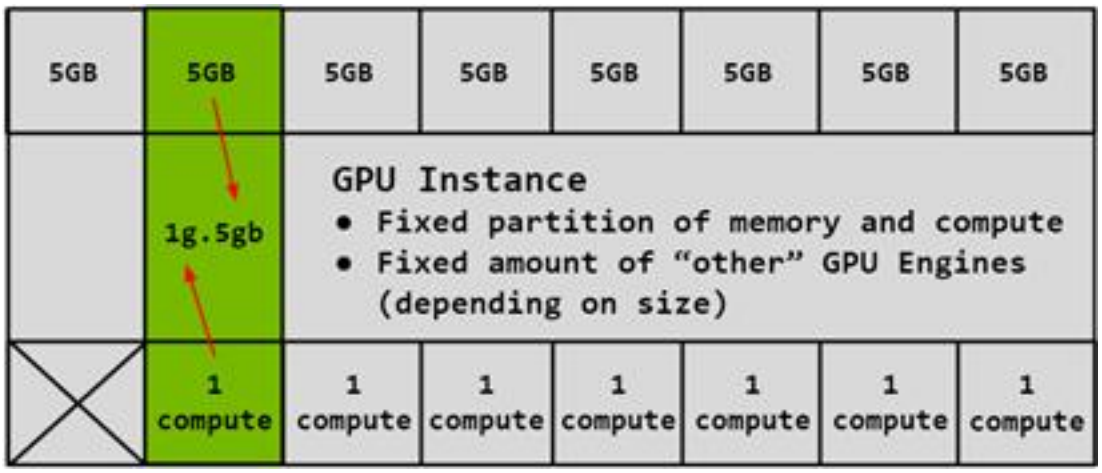
本节介绍如何在 GPU 上创建各种分区。将使用 A100-40GB 作为示例演示如何对单个 GPU 物理卡上进行分区。

GPU 的分区是使用内存切片进行的，因此可以认为 A100-40GB GPU 具有 8x5GB 内存切片和 7 个 GPU SM 切片，如下图所示，展示了 A100 上可用的内存切片。



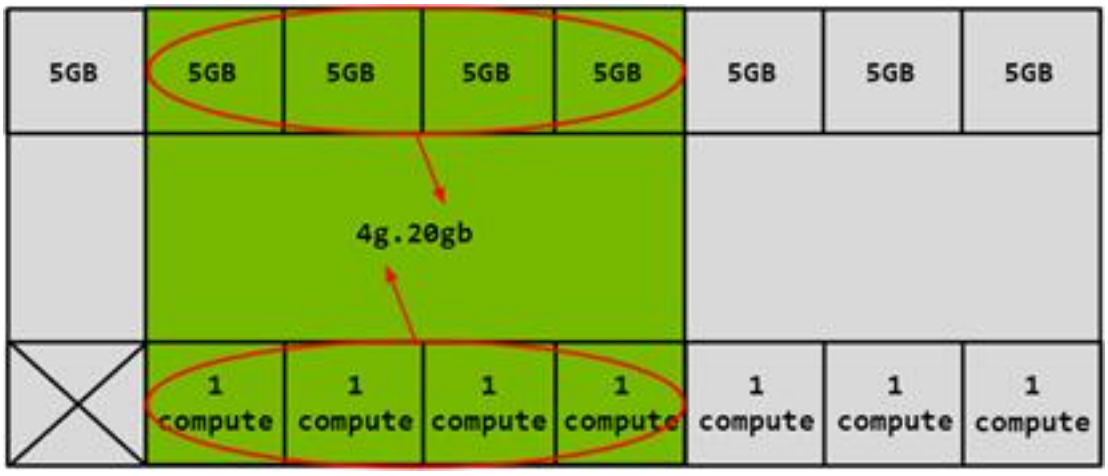
img

如上所述，创建 GPU 实例 （GI） 需要将一定数量的内存切片与一定数量的计算切片相结合。在下图中，一个 5GB 内存切片与 1 个计算切片相结合，以创建 1g.5gb GI 配置文件：



img

同样，4x5GB 内存切片可以与 4x1 计算切片结合使用以创建 4g.20gb 的 GI 配置文件：

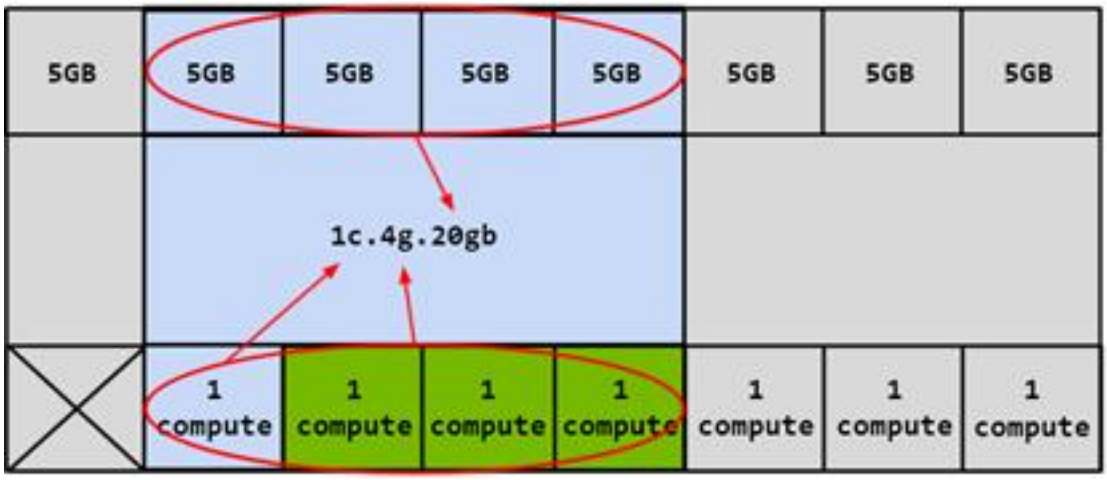


img

计算实例（CI）

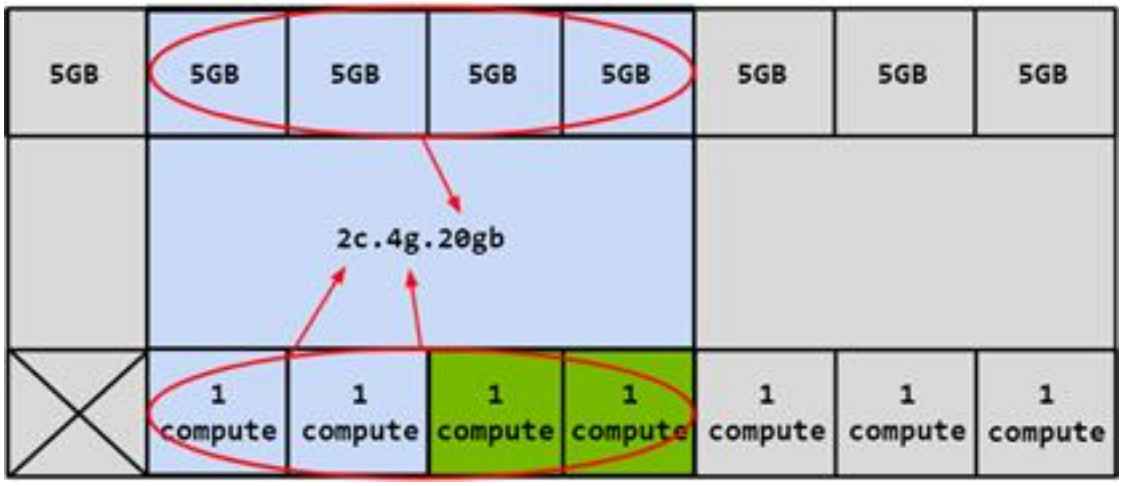
GPU 实例的计算切片(GI)可以进一步细分为多个计算实例（CI），其中 CI 共享父 GI 的引擎和内存，但每个 CI 都有专用的 SM 资源。使用上面的相同

4g.20gb 示例，可以创建一个 CI 以仅使用第一个计算切片的 1c.4g.20gb 计算配置，如下图蓝色部分所示：



img

在这种情况下，可以通过选择任何计算切片来创建 4 个不同的 CI。还可以将两个计算切片组合在一起以创建 2c.4g.20gb 的计算配置）：



img

除此之外，还可以组合 3 个计算切片以创建计算配置文件，或者可以组合所有 4 个计算切片以创建 3c.4g.20gb、4c.4g.20gb 计算配置文件。合并所有 4 个计算切片时，配置文件简称为 4g.20gb。

支持 MIG 特性的 GPU 卡

从 NVIDIA Ampere 一代开始的 GPU（即具有计算能力 >= 8.0）的 GPU）支持 MIG。下表提供了支持 MIG 的 GPU 列表：

GPU 型号	**架构**	**微架构**	**计算能力**	**内存大小**	**最大 GI 实例数**
H100–SXM5 H100–SXM5	Hopper	GH100	9.0	80GB	7
H100–PCIE H100–PCIE	Hopper	GH100	9.0	80GB	7
A100–SXM4 A100–SXM4	NVIDIA Ampere	GA100	8.0	40GB	7
A100–SXM4 A100–SXM4	NVIDIA Ampere	GA100	8.0	80GB	7
A100–PCIE A100–PCIE	NVIDIA Ampere	GA100	8.0	40GB	7
A100–PCIE A100–PCI	NVIDIA Ampere	GA100	8.0	80GB	7
A30	NVIDIA Ampere	GA100	8.0	24GB	4

执行集群巡检

巡检配置创建完成后，如果启用了定时巡检配置，则会在指定时间自动执行巡检。如未启用定时巡检配置，则需要手动触发巡检。

此页介绍如何手动执行集群巡检。

前提条件

- 在容器管理模块中接入或创建集群
- 已创建巡检配置
- 所选集群处于运行中状态且已经在集群中安装了 insight 组件

操作步骤

执行巡检时，支持勾选多个集群进行批量巡检，或者仅对某一个集群进行单独巡检。

===“批量巡检”

1. 在容器管理模块的一级导航栏点击`集群巡检`，然后在页面右侧点击`巡检`。

   ![start](../../images/inspect04.png)

2. 勾选需要巡检的集群，然后在页面底部点击`确定`即可。

   - 若选择多个集群进行同时巡检，系统将根据不同集群的巡检配置进行巡检。
   - 如未设置集群巡检配置，将使用系统默认配置。

     ![start](../../images/inspect05.png)

===“单独巡检”

1. 进入集群巡检页面。
2. 在对应巡检配置的右侧点击 ` ` 更多操作按钮，然后在弹出的菜单中选择`巡检`即可。



查看巡检报告

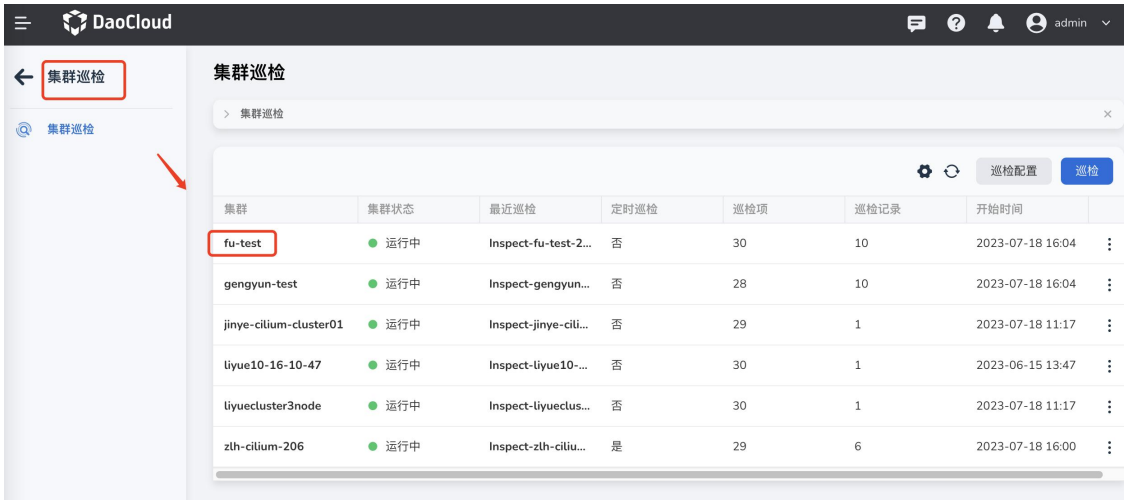
巡检执行完成后，可以查看巡检记录和详细的巡检报告。

前提条件

- 已经创建了巡检配置
- 已经执行过至少一次巡检

操作步骤

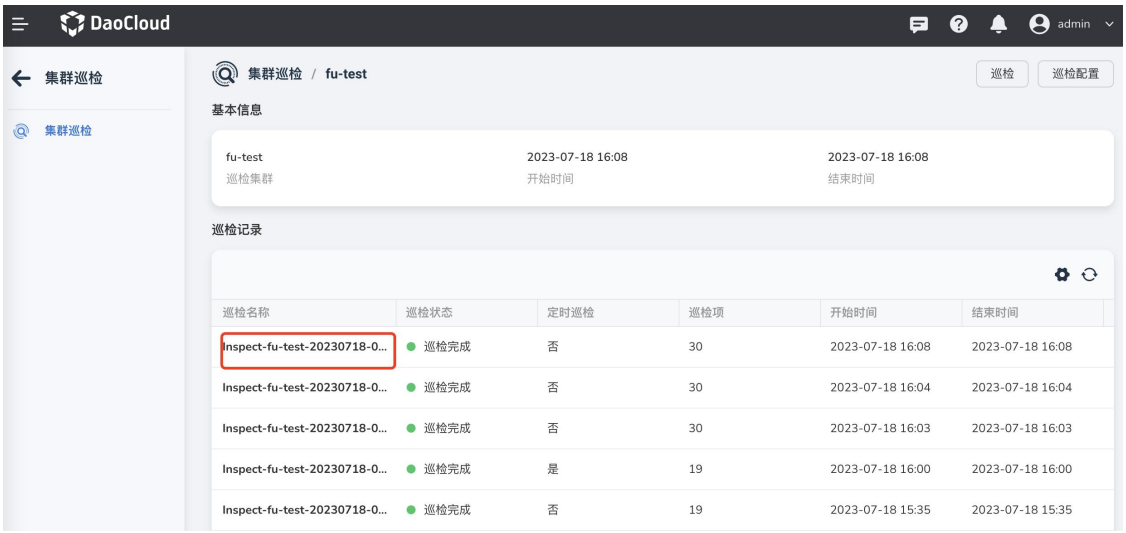
1. 进入集群巡检页面，点击目标巡检集群的名称。



start

2. 点击想要查看的巡检记录名称。

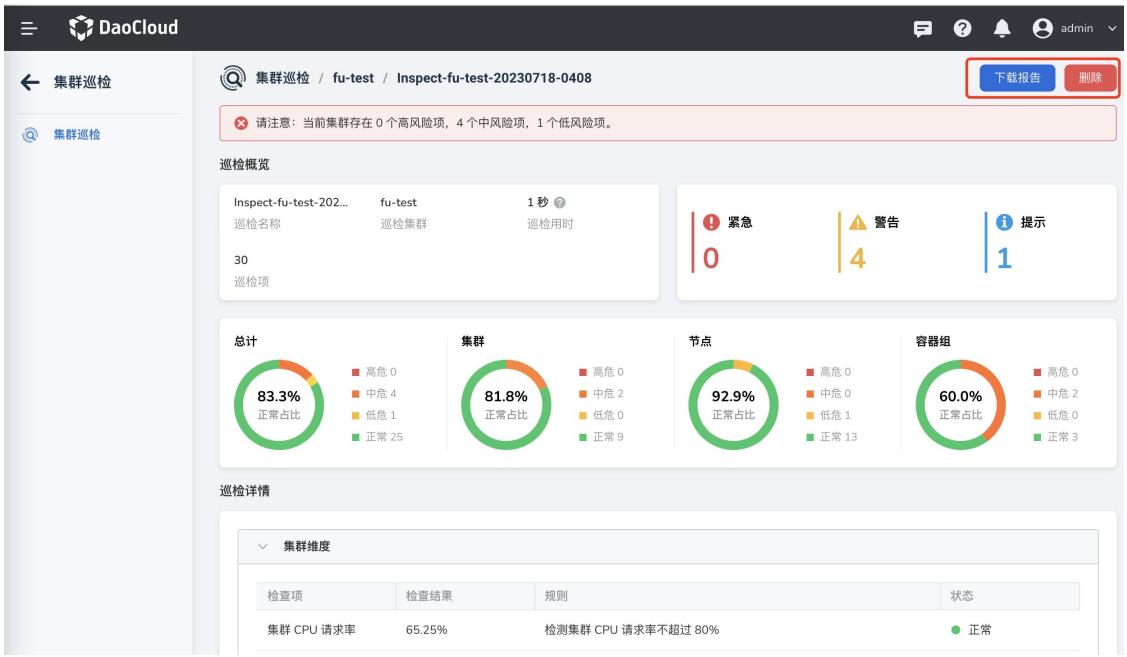
    - 每执行一次巡检，就会生成一条巡检记录。
    - 当巡检记录超过巡检配置中设置的最大保留条数时，从执行时间最早的记录开始删除。



start

3. 查看巡检的详细信息，根据巡检配置可能包括集群资源概览、系统组件的运行情况等。

   在页面右上角可以下载巡检报告或删除该项巡检报告。



start

集群巡检

集群巡检可以通过自动或手动方式，定期或随时检查集群的整体健康状态，让管理员获得保障集群安全的主动权。基于合理的巡检计划，这种主动自发的集群检查可以让管理员随时掌握集群状态，解决了之前出现故障时只能被动排查问题的困境，做到了事先监控、提前防范。

DCE 5.0 容器管理模块提供的集群巡检功能，支持从集群维度、节点维度、容器组维度进行自定义巡检项，巡检结束后会自动生成可视化的巡检报告。

- 集群维度：检查集群中系统组件的运行情况，包括集群状态、资源使用情况，以及控制节点特有的巡检项等，例如 kube-apiserver 和 etcd 的状态。
- 节点维度：包括控制节点和工作节点通用的检查项，例如节点资源使用情况、句柄数、PID 状态、网络状态。
- 容器组维度：检查 Pod 的 CPU 和内存使用情况、运行状态、PV 和 PVC 的状态等。

如需了解或执行安全方面的巡检，可参考 DCE 5.0 支持的安全扫描类型。

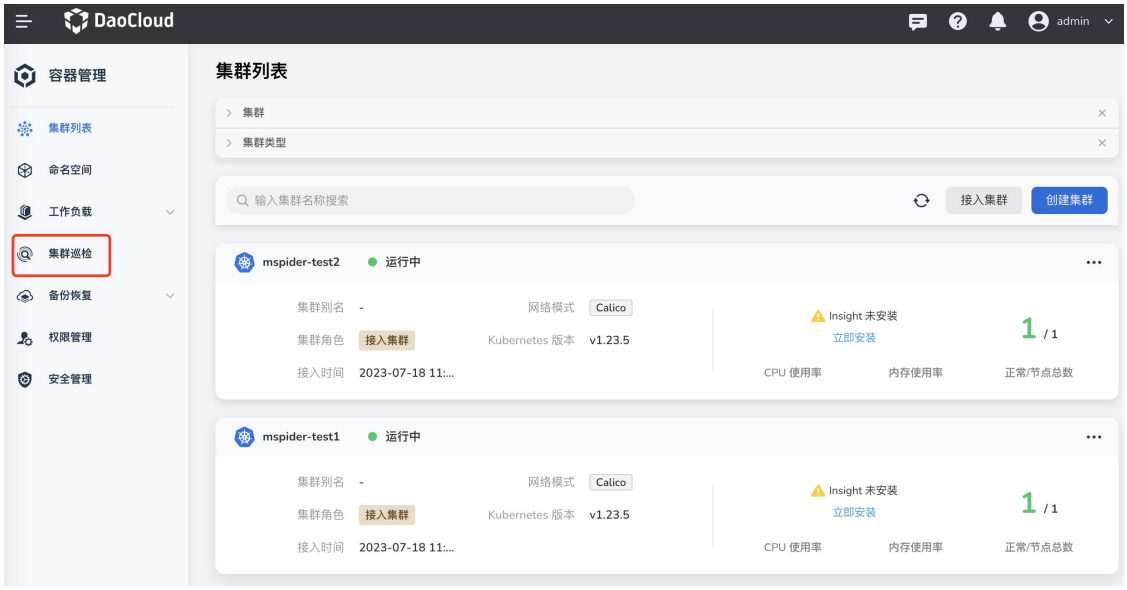
创建巡检配置

DCE 5.0 容器管理模块提供集群巡检功能，支持从集群维度、节点维度、容器组维度进行巡检。

- 集群维度：检查集群中系统组件的运行情况，包括集群状态、资源使用情况，以及控制节点特有的巡检项等，例如 kube-apiserver 和 etcd 的状态。
- 节点维度：包括控制节点和工作节点通用的检查项，例如节点资源使用情况、句柄数、PID 状态、网络状态。
- 容器组维度：检查 Pod 的 CPU 和内存使用情况、运行状态、PV 和 PVC 的状态等。

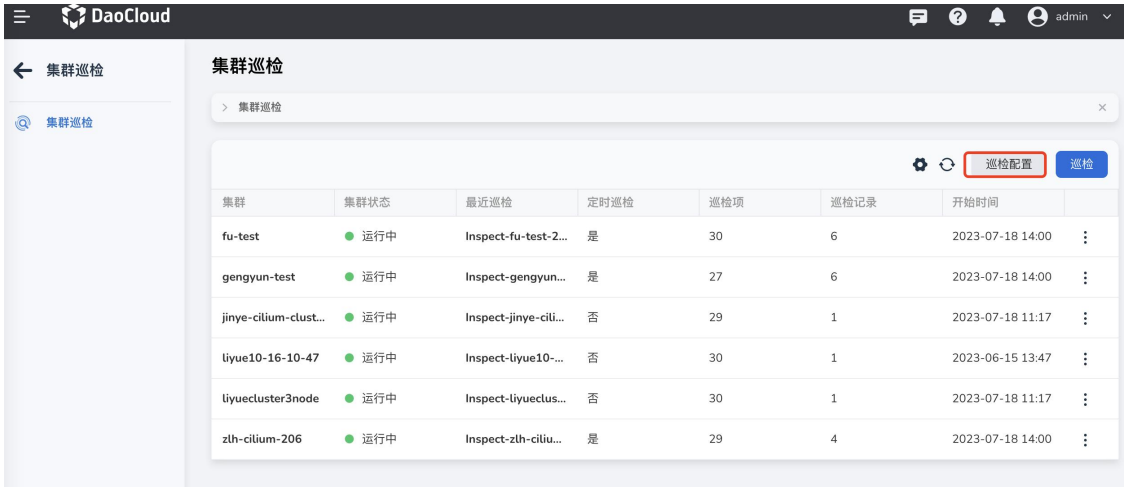
下面介绍如何创建巡检配置。

1. 在左侧导航栏点击集群巡检。



nav

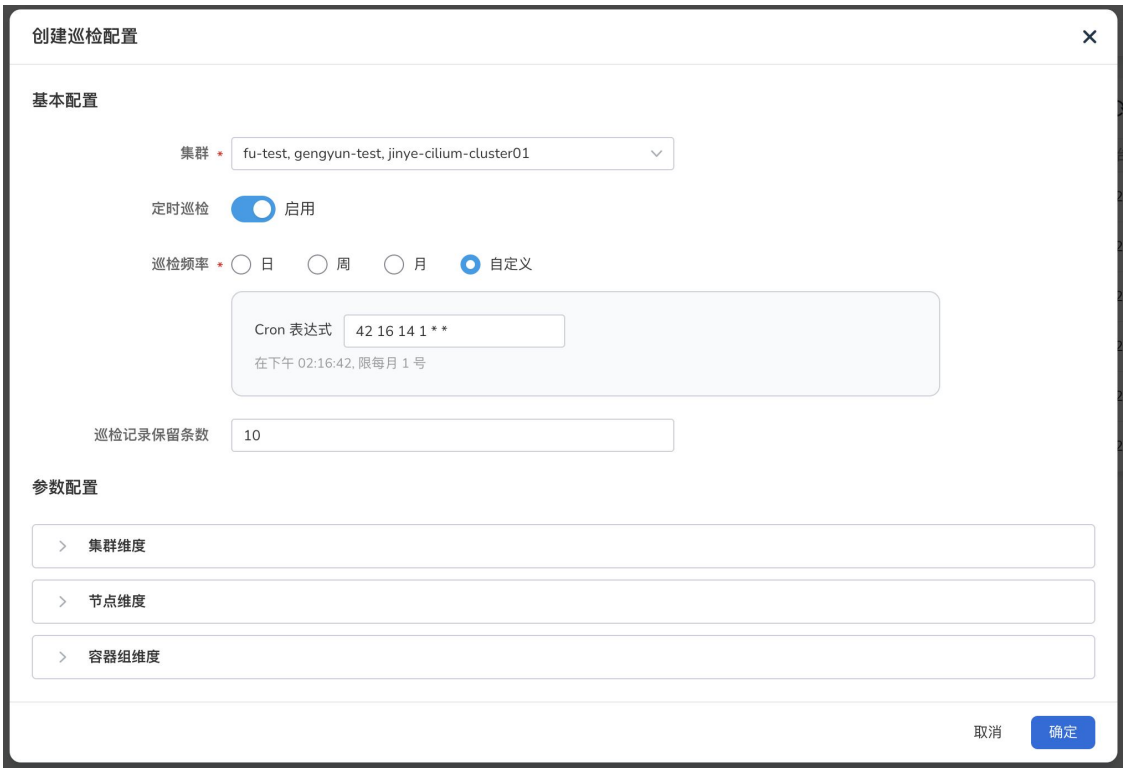
2. 在页面右侧点击巡检配置。



create

3. 参考以下说明填写巡检配置，然后在页面底部点击确定即可。

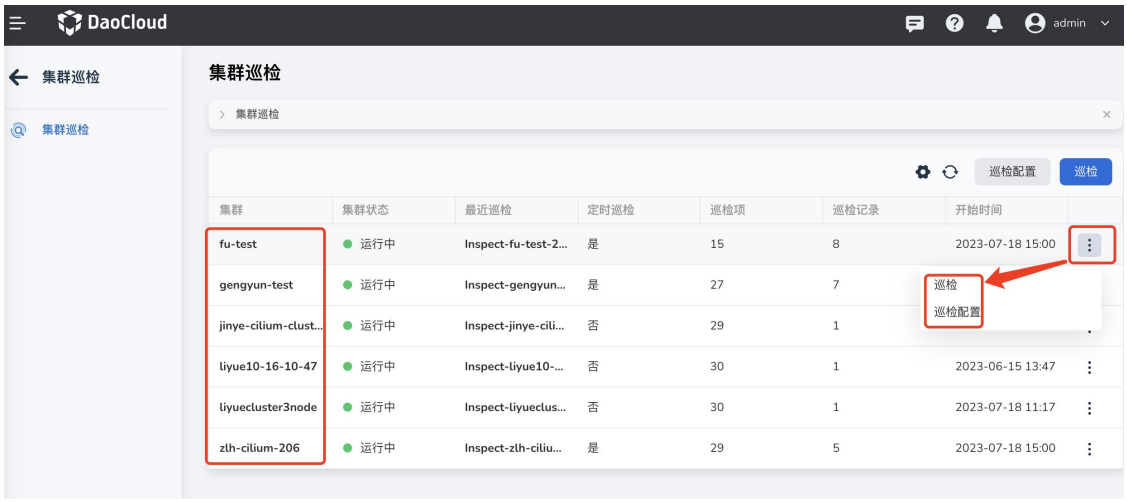
   - 集群：下拉选择要对哪些集群进行巡检。**如果选择多个集群，则自动生成多个巡检配置（仅巡检的集群不一致，其他配置都完全一致）**
   - 定时巡检：启用后可根据事先设置的巡检频率定期自动执行集群巡检
   - 巡检频率：设置自动巡检的周期，例如每周二上午十点。支持自定义 CronExpression，可参考 Cron 时间表语法
   - 巡检记录保留条数：累计最多保留多少条巡检记录，包括所有集群的巡检记录
   - 参数配置：参数配置分为集群维度、节点维度、容器组维度三部分，可以根据场景需求启用或禁用某些巡检项。



basic

巡检配置创建完成后，会自动显示在巡检配置列表中。在配置右侧点击更多操作按钮可以立即执行巡检或修改巡检配置。

- 点击巡检可以根据该配置立即执行一次巡检。
- 点击巡检配置可以修改巡检配置。



basic

!!! note

- 巡检配置创建完成后，如果启用了`定时巡检`配置，则会在指定时间自动执行巡检。
- 如未启用`定时巡检`配置，则需要手动[触发巡检](inspect.md)。

离线升级集群巡检模块

本页说明从下载中心下载集群巡检模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `kocllie` 字样是集群巡检模块的内部开发代号。

从下载的安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

方式一：使用 chart-syncer 同步镜像

使用 chart-syncer 可以将您下载的安装包中的 chart 及其依赖的镜像包上传至安装器部署 DCE 时使用的镜像仓库和 helm 仓库。

首先找到一台能够连接镜像仓库和 helm 仓库的节点（如火种节点），在节点上创建 load-image.yaml 配置文件，填入镜像仓库和 helm 仓库等配置信息。

1. 创建 load-image.yaml

   !!! note

   该 YAML 文件中的各项参数均为必填项。

   ===“已添加 Helm repo”

   若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

   ```yaml title="load-image.yaml"
   source:
     intermediateBundlesPath: kocllie # 节点上执行 load-image.yaml 文件的
   ```

路径。

```
 target:
    containerRegistry: 10.16.10.111 # 镜像仓库地址
    containerRepository: release.daocloud.io/kocllie # 镜像仓库路径
    repo:
      kind: HARBOR # Helm Chart 仓库类别
      url: http://10.16.10.111/chartrepo/release.daocloud.io # Helm 仓库地址
      auth:
      username: "admin" # 镜像仓库用户名
      password: "Harbor12345" # 镜像仓库密码
    containers:
      auth:
        username: "admin" # Helm 仓库用户名
        password: "Harbor12345" # Helm 仓库密码
```

===“未添加 Helm repo”

若当前节点上未添加 helm repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kocllie # 节点上执行 load-image.yaml 文件的路径。
target:
  containerRegistry: 10.16.10.111 # 镜像仓库 url
  containerRepository: release.daocloud.io/kocllie # 镜像仓库路径
  repo:
    kind: LOCAL
    path: ./local-repo # chart 本地路径
  containers:
    auth:
```

```
    username: "admin" # 镜像仓库用户名
    password: "Harbor12345" # 镜像仓库密码
```

2. 执行同步镜像命令。

   charts-syncer sync --config load-image.yaml

方式二：使用 Docker 或 containerd 加载镜像

解压并加载镜像文件。

1. 解压 tar 压缩包。

   tar xvf kocllie.bundle.tar

   解压成功后会得到 3 个文件：

   - hints.yaml
   - images.tar
   - original-chart

2. 从本地加载镜像到 Docker 或 containerd。

   === “Docker”

   ```shell
   docker load -i images.tar
   ```

   === “containerd”

   ```shell
   ctr image import images.tar
   ```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

===“通过 helm repo 升级”

1. 检查集群巡检 helm 仓库是否存在。

    ```shell
    helm repo list | grep kocllie
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加集群巡检的 helm 仓库。

    ```shell
    heml repo add kocllie http://{harbor url}/chartrepo/{project}
    ```

1. 更新集群巡检的 helm 仓库。

    ```shell
    helm repo update kocllie
    ```

1. 选择您想安装的集群巡检版本（建议安装最新版本）。

    ```shell
    helm search repo kocllie/kocllie --versions
    ```

    ```none
    [root@master ~]# helm search repo kocllie/kocllie --versions
    NAME                    CHART VERSION   APP VERSION   DESCRIPTION
    kocllie/kocllie   0.20.0          v0.20.0         A Helm chart for kocllie
    ...
    ```

1. 备份 `--set` 参数。

    在升级集群巡检版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

    ```shell
    helm get values kocllie -n kocllie-system -o yaml > bak.yaml
    ```

1. 更新 kocllie crds

    ```shell
    helm pull kocllie/kocllie --version 0.6.0 && tar -zxf kocllie-0.6.0.tgz
    kubectl apply -f kocllie/crds
    ```

1. 执行 `helm upgrade`。

    升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像

仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kocllie kocllie/kocllie \
  -n kocllie-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.6.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

   在升级集群巡检版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values kocllie -n k pan da-system -o yaml > bak.yaml
   ```

1. 更新 kocllie crds

   ```shell
   kubectl apply -f ./crds
   ```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kocllie . \
  -n kocllie-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

创建 HPA

DaoCloud Enterprise 5.0 支持 Pod 资源基于指标进行弹性伸缩（Horizontal Pod Autoscaling, HPA）。 用户可以通过设置 CPU 利用率、内存用量及自定义指标指标来动态调整 Pod 资源的副本数量。 例如，为工作负载设置基于 CPU 利用率指标弹性伸缩策略后，当 Pod 的 CPU 利用率超过/低于您设置的指标阈值，工作负载控制器将会自动增加/较少 Pod 副本数。

本文将介绍如何为工作负载配置基于内置指标和自定义指标的弹性伸缩。

!!! note

1. HPA 仅适用于 Deployment 和 StatefulSet，每个工作负载只能创建一个 HPA。
2. 如果基于 CPU 利用率创建 HPA 策略，必须预先为工作负载设置配置限制（Limit），否则无法计算 CPU 利用率。
3. 如果同时使用内置指标和多种自定义指，HPA 会根据多项指标分别计算所需伸缩副本数，取较大值（但不会超过设置 HPA 策略时配置的最大副本数）进行弹性伸缩。

内置指标弹性伸缩策略

系统内置了 CPU 和内存两种弹性伸缩指标以满足用户的基础业务使用场景。

前提条件

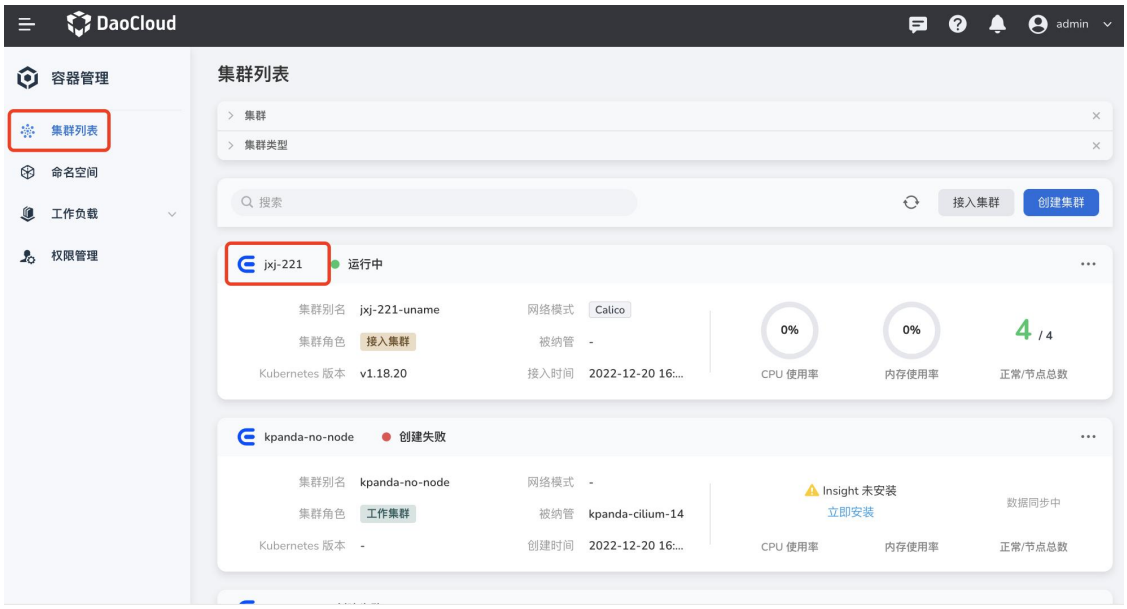
在为工作负载配置内置指标弹性伸缩策略之前，需要满足以下前提条件：

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建、无状态工作负载的创建或有状态工作负载的创建。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 已完成 metrics-server 插件安装。

操作步骤

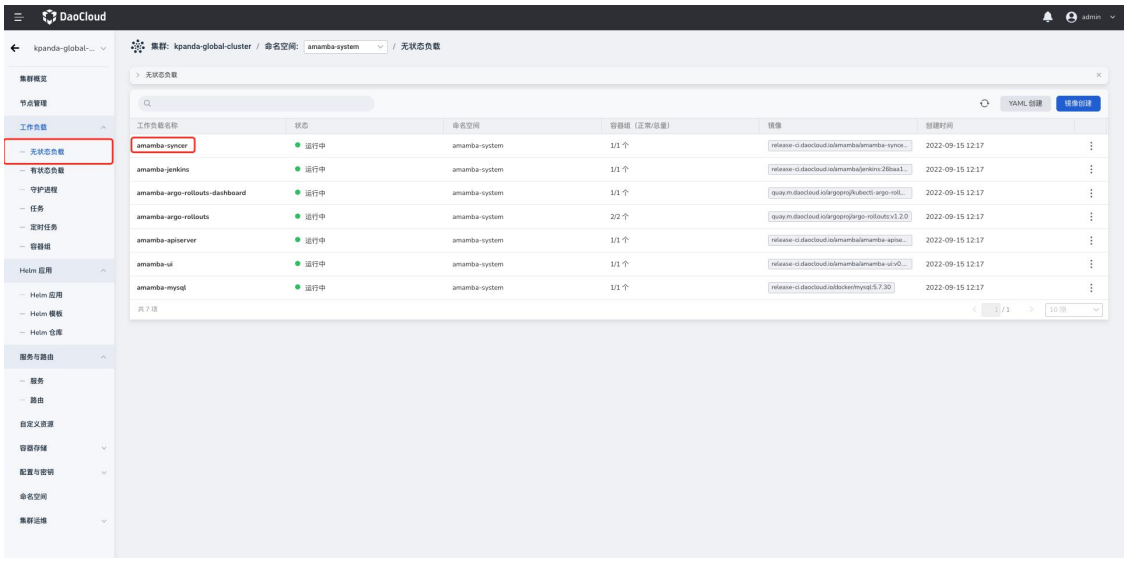
参考以下步骤，为工作负载配置内置指标弹性伸缩策略。

1. 点击左侧导航栏上的集群列表进入集群列表页面。点击一个集群名称，进入集群详情页面。



集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载进入工作负载列表后，点击一个负载名称，进入工作负载详情页面。



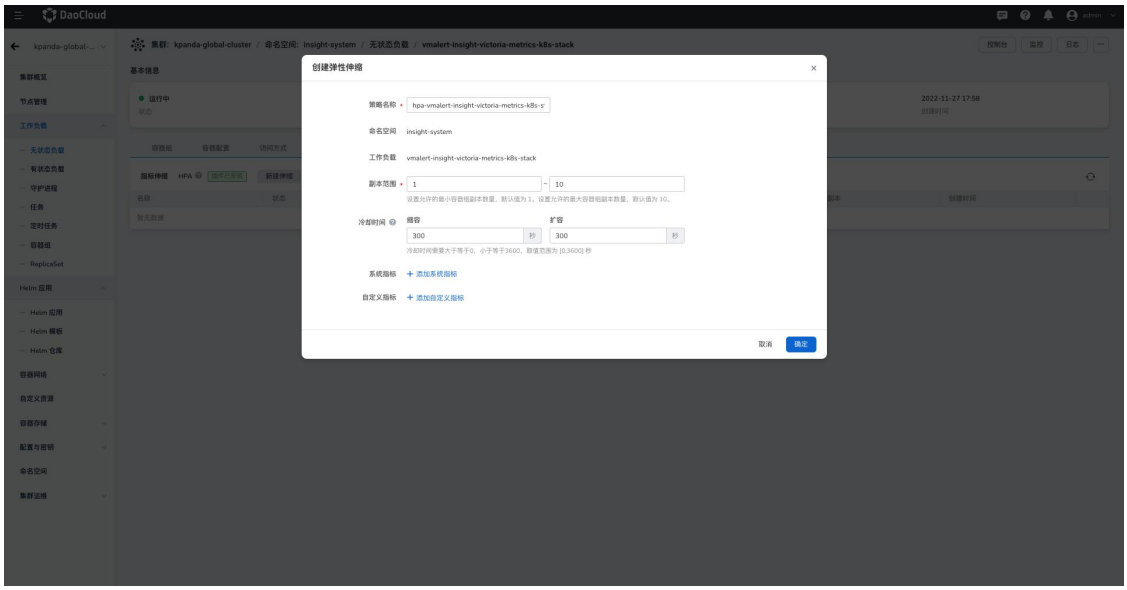
工作负载

3. 点击弹性伸缩页签，查看当前集群的弹性伸缩配置情况。



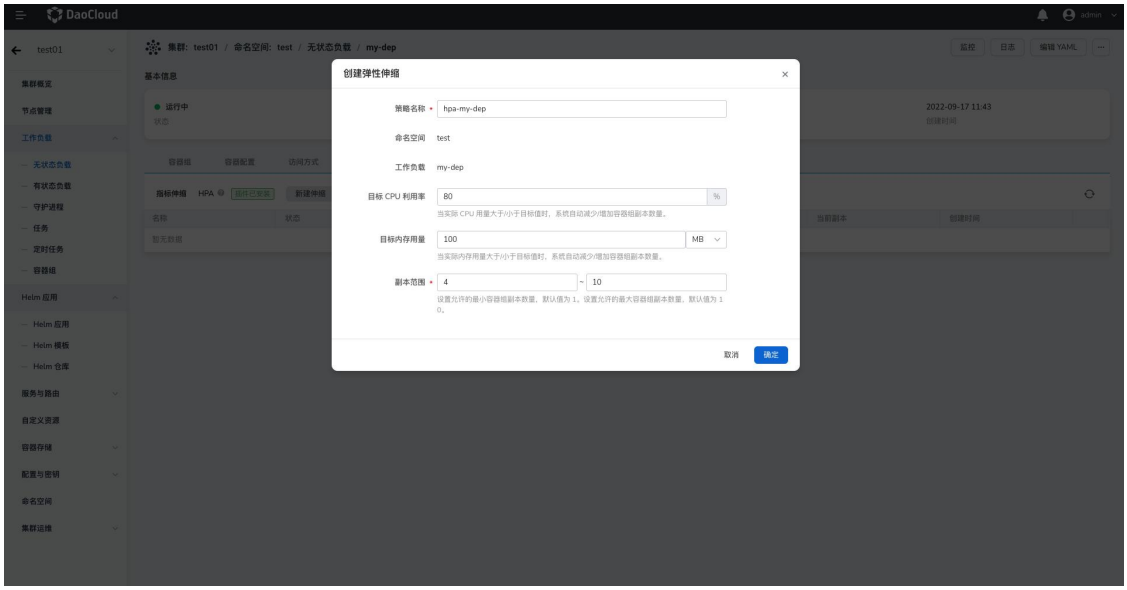
弹性伸缩

4. 确认集群已安装了 metrics-server 插件，且插件运行状态为正常后，即可点击新建伸缩按钮。



新建伸缩

5. 创建自定义指标弹性伸缩策略参数。



工作负载

- 策略名称：输入弹性伸缩策略的名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 hpa-my-dep。
- 命名空间：负载所在的命名空间。

- 工作负载：执行弹性伸缩的工作负载对象。
- 目标 CPU 利用率：工作负载资源下 Pod 的 CPU 使用率。计算方式为：工作负载下所有的 Pod 资源 / 工作负载的请求（request）值。当实际 CPU 用量大于/小于目标值时，系统自动减少/增加 Pod 副本数量。
- 目标内存用量：工作负载资源下的 Pod 的内存用量。当实际内存用量大于/小于目标值时，系统自动减少/增加 Pod 副本数量。
- 副本范围：Pod 副本数的弹性伸缩范围。默认区间为为 1 – 10。

6. 完成参数配置后，点击确定按钮，自动返回弹性伸缩详情页面。点击列表右侧的 ⋮，可以执行编辑、删除操作，还可以查看相关事件。



工作负载

自定义指标弹性伸缩策略

当系统内置的 CPU 和内存两种指标不能够满足您业务的实际需求时，您可以通过配置 ServiceMonitoring 来添加自定义指标，并基于自定义指标实现弹性伸缩。

前提条件

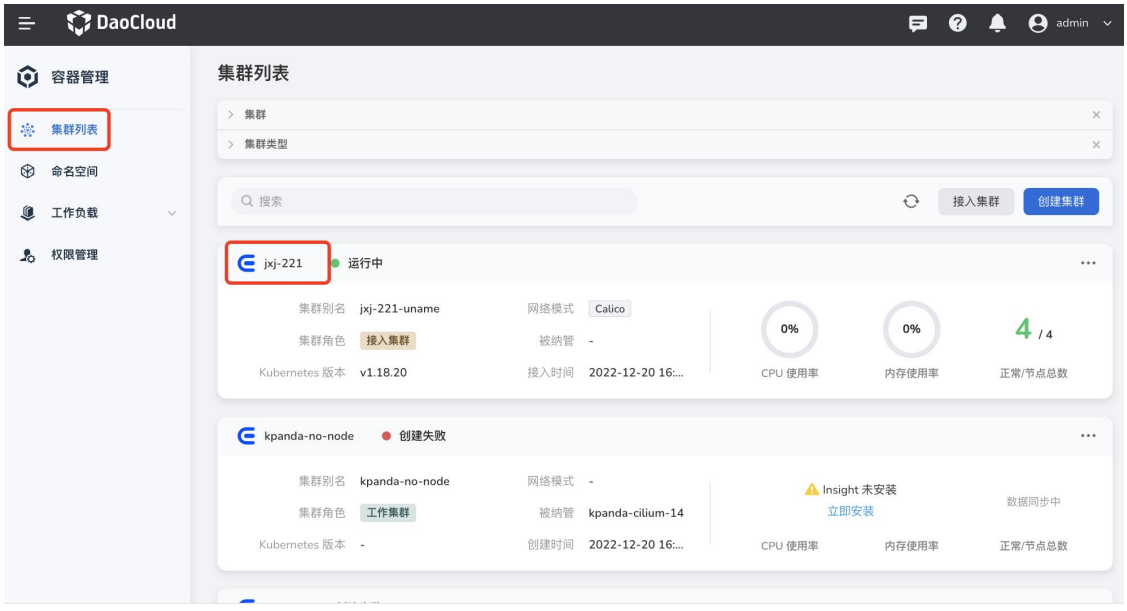
在为工作负载配置自定义指标弹性伸缩策略之前，需要满足以下前提条件：

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建、无状态工作负载的创建或有状态工作负载的创建。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 已完成 metrics-server 插件安装。
- 已完成 Insight 插件的安装。
- 已完成 Prometheus-adapter 插件的安装。

操作步骤

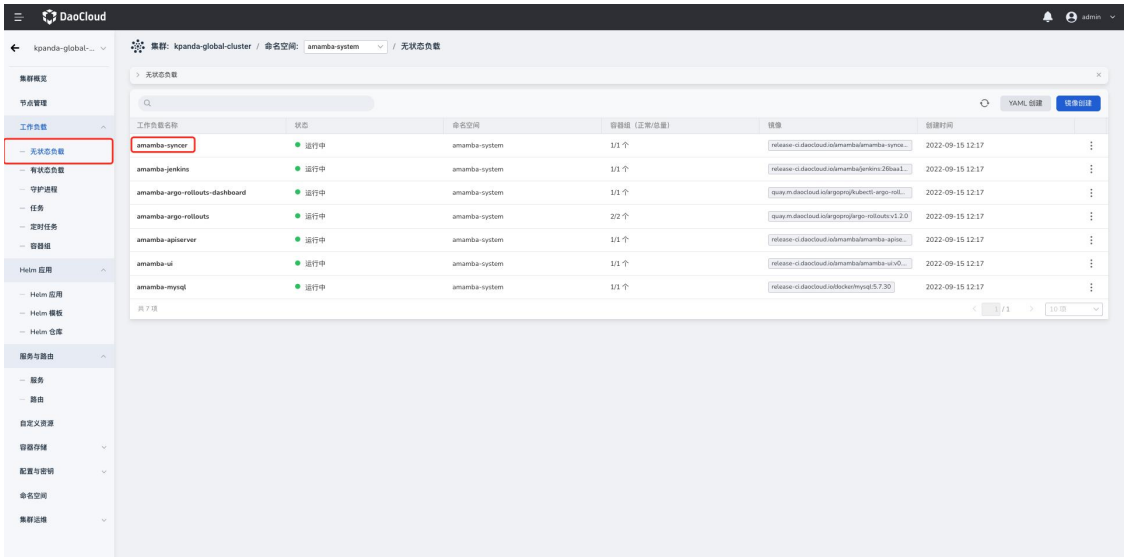
参考以下步骤，为工作负载配置指标弹性伸缩策略。

1. 点击左侧导航栏上的集群列表进入集群列表页面。点击一个集群名称，进入集群详情页面。



集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载进入工作负载列表后，点击一个负载名称，进入工作负载详情页面。



工作负载

3. 点击弹性伸缩页签，查看当前集群的弹性伸缩配置情况。

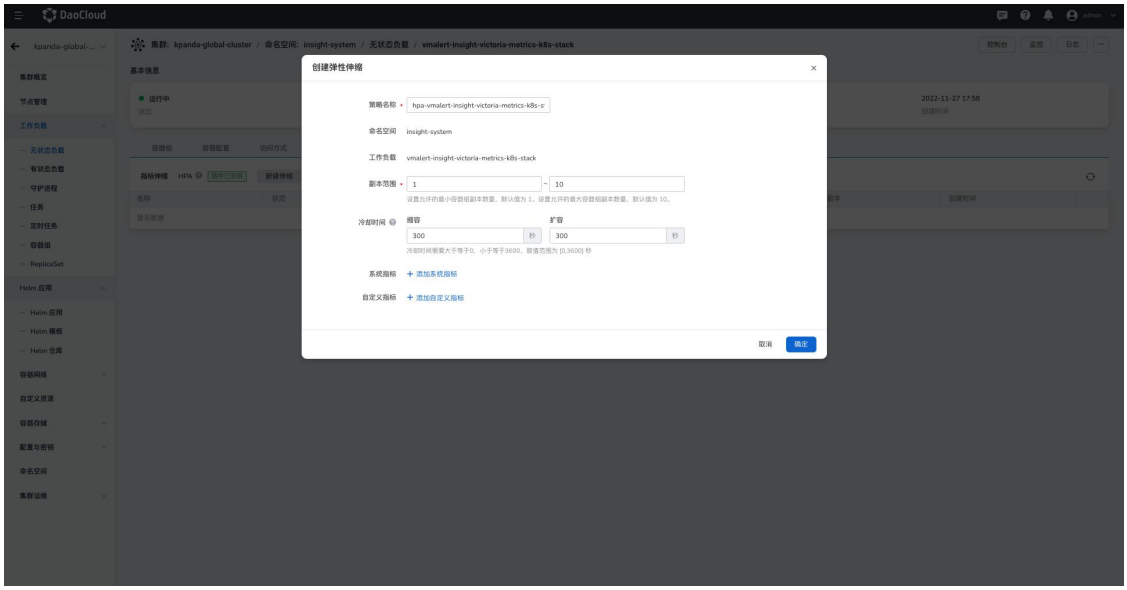


弹性伸缩配置

4. 确认集群已安装了 metrics-server、Insight、Prometheus-adapter 插件且插件运行状态为正常后，即可点击新建伸缩按钮。

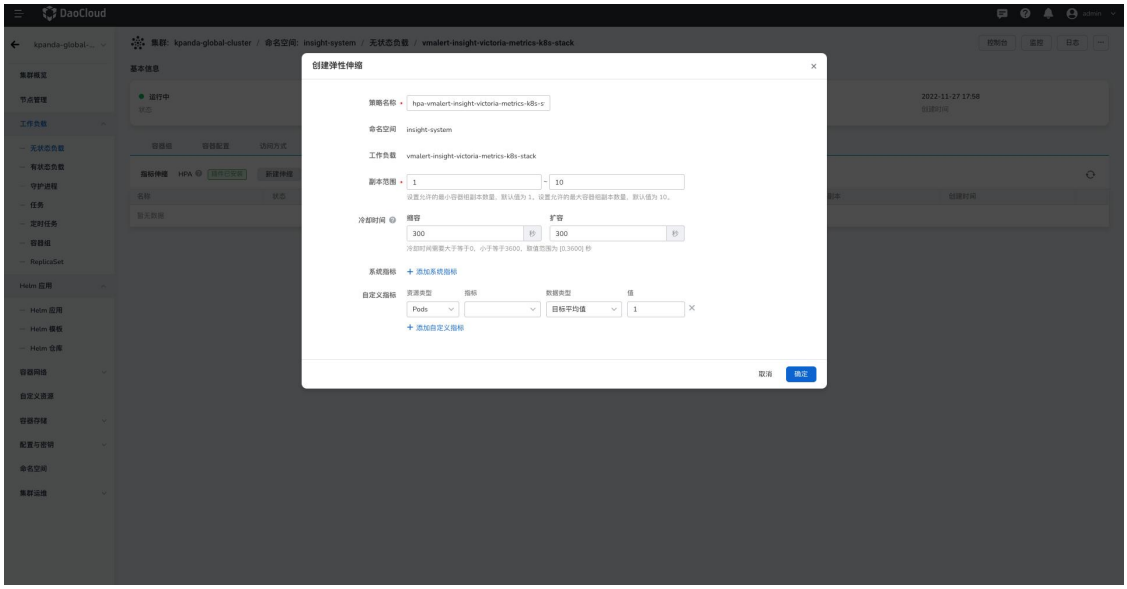
!!! note

如果相关插件未安装或插件处于异常状态，您在页面上将无法看见创建自定义指标弹性伸缩入口。



新建伸缩

5. 创建自定义指标弹性伸缩策略参数。



伸缩策略参数

- 策略名称：输入弹性伸缩策略的名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 hpa-my-dep。
- 命名空间：负载所在的命名空间。
- 工作负载：执行弹性伸缩的工作负载对象。
- 资源类型：进行监控的自定义指标类型，包含 Pod 和 Service 两种类型。
- 指标：使用 ServiceMonitoring 创建的自定义指标名称或系统内置的自定义指标名称。
- 数据类型：用于计算指标值的方法，包含目标值和目标平均值两种类型，当资源类型为 Pod 时，只支持使用目标平均值。

创建 VPA

容器垂直扩缩容策略（Vertical Pod Autoscaler, VPA）通过监控 Pod 在一段时间内的资源申请和用量，计算出对该 Pod 而言最适合的 CPU 和内存请求值。使用 VPA 可以更加合理地为集群下每个 Pod 分配资源，提高集群的整体资源利用率，避免集群资源浪费。

DCE 5.0 支持通过容器垂直扩缩容策略（Vertical Pod Autoscaler, VPA），基于此功能可以根据容器资源的使用情况动态调整 Pod 请求值。DCE 5.0 支持通过手动和自动两种方式来修改资源请求值，您可以根据实际需要进行配置。

本文将介绍如何为工作负载配置 Pod 垂直伸缩。

!!! warning

使用 VPA 修改 Pod 资源请求会触发 Pod 重启。由于 Kubernetes 本身的限制， Pod 重启后可能会被调度到其它节点上。

前提条件

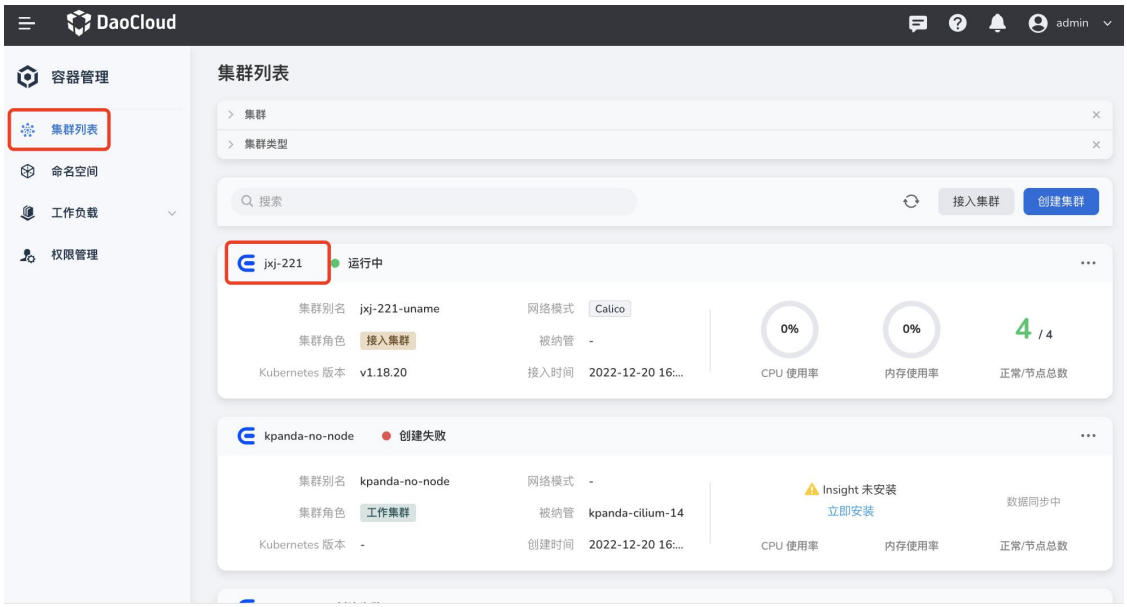
为工作负载配置垂直伸缩策略之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间、用户、无状态工作负载或有状态工作负载。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 当前集群已经安装 metrics-server 和 VPA 插件。

操作步骤

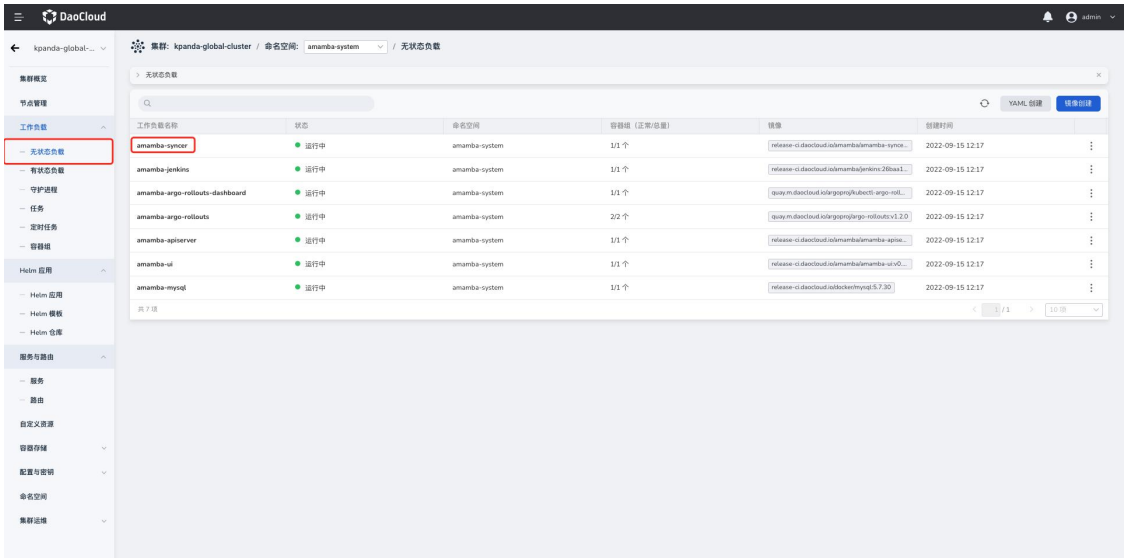
参考以下步骤，为工作负载配置内置指标弹性伸缩策略。

1. 在集群列表中找到目前集群，点击目标集群的名称。



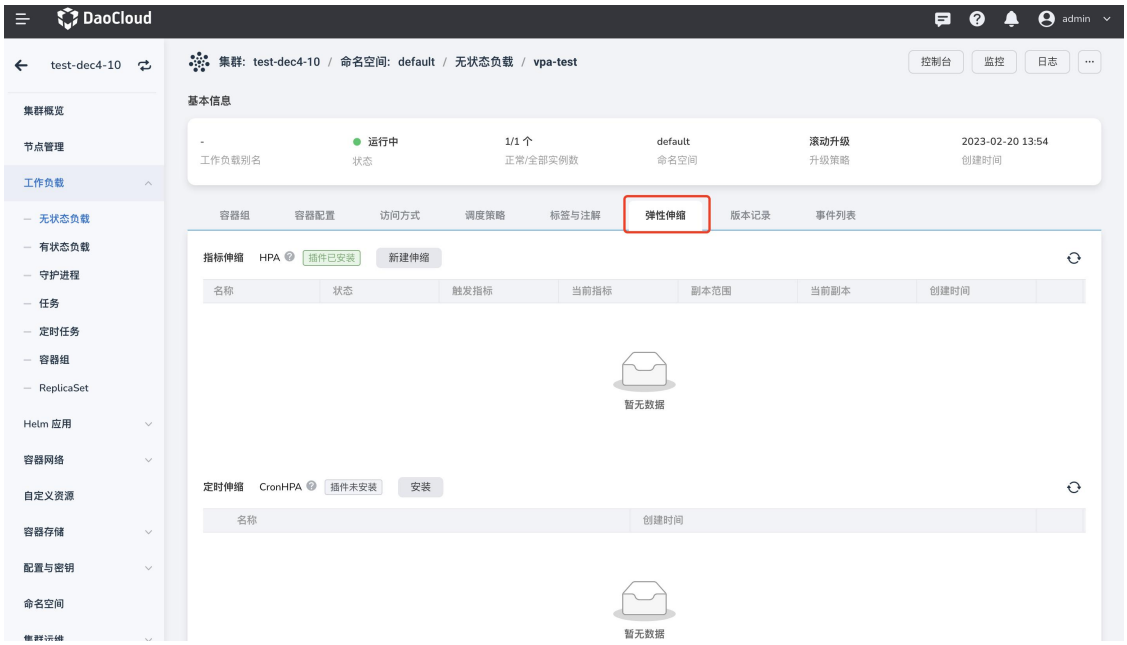
集群详情

2. 在左侧导航栏点击工作负载，找到需要创建 VPA 的负载，点击该负载的名称。



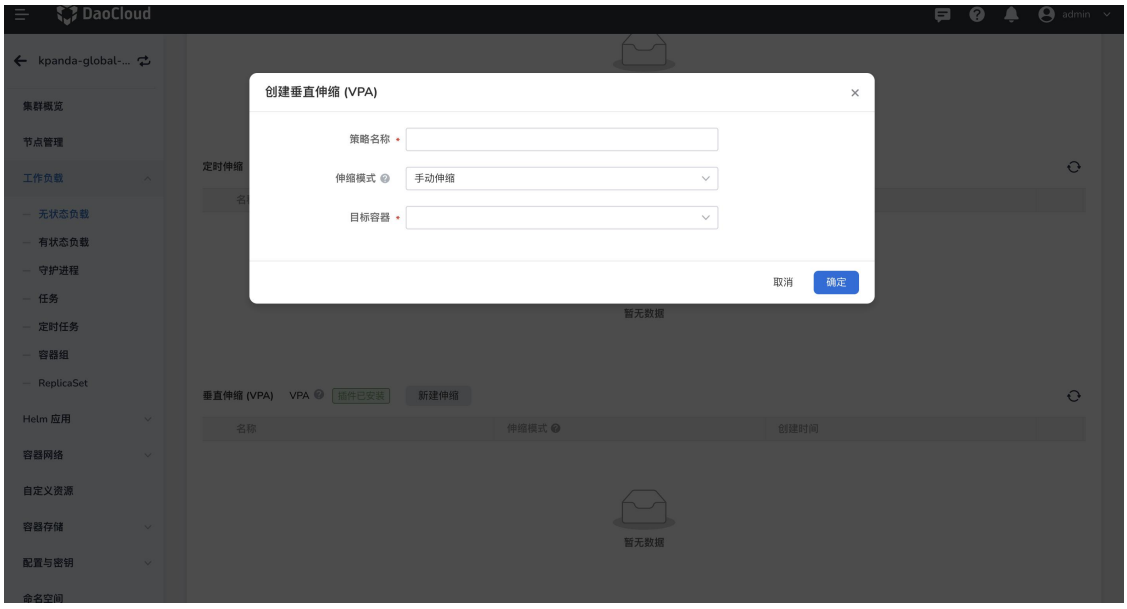
工作负载

3. 点击弹性伸缩页签，查看当前集群的弹性伸缩配置，确认已经安装了相关插件并且插件是否运行正常。



垂直伸缩

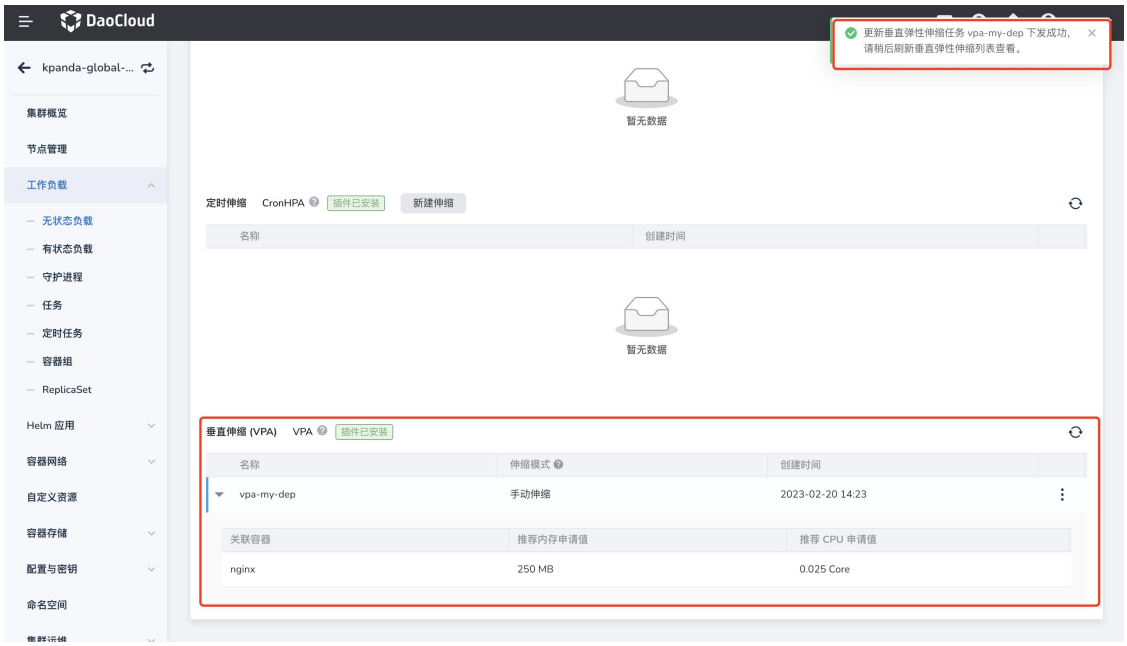
4. 点击新建伸缩按钮，并配置 VPA 垂直伸缩策略参数。



新建伸缩

- 策略名称：输入垂直伸缩策略的名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 vpa-my-dep。
- 伸缩模式：执行修改 CPU 和内存请求值的方式，目前垂直伸缩支持手动和自动两种伸缩模式。
  - 手动伸缩：垂直伸缩策略计算出推荐的资源配置值后，需用户手动修改应用的资源配额。
  - 自动伸缩：垂直伸缩策略自动计算和修改应用的资源配额。
- 目标容器：选择需要进行垂直伸缩的容器。

5. 完成参数配置后，点击确定按钮，自动返回弹性伸缩详情页面。点击列表右侧的 ⋮，可以执行编辑、删除操作。



工作负载

安装 metrics-server 插件

metrics-server 是 Kubernetes 内置的资源使用指标采集组件。 您可以通过配置弹性伸缩（HPA）策略来实现工作负载资源自动水平伸缩 Pod 副本。

本节介绍如何安装 metrics-server。

前提条件

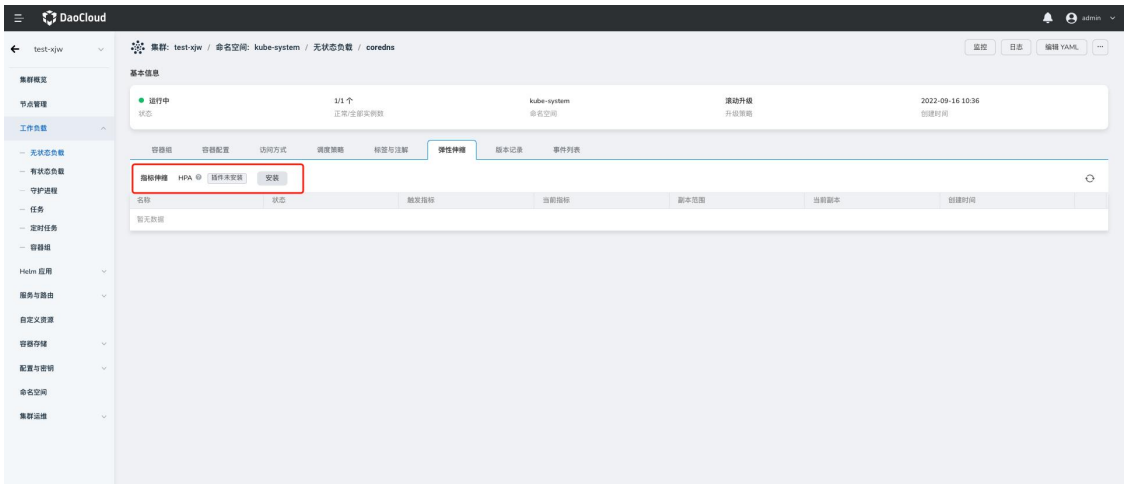
安装 metrics-server 插件前，需要满足以下前提条件：

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。

操作步骤

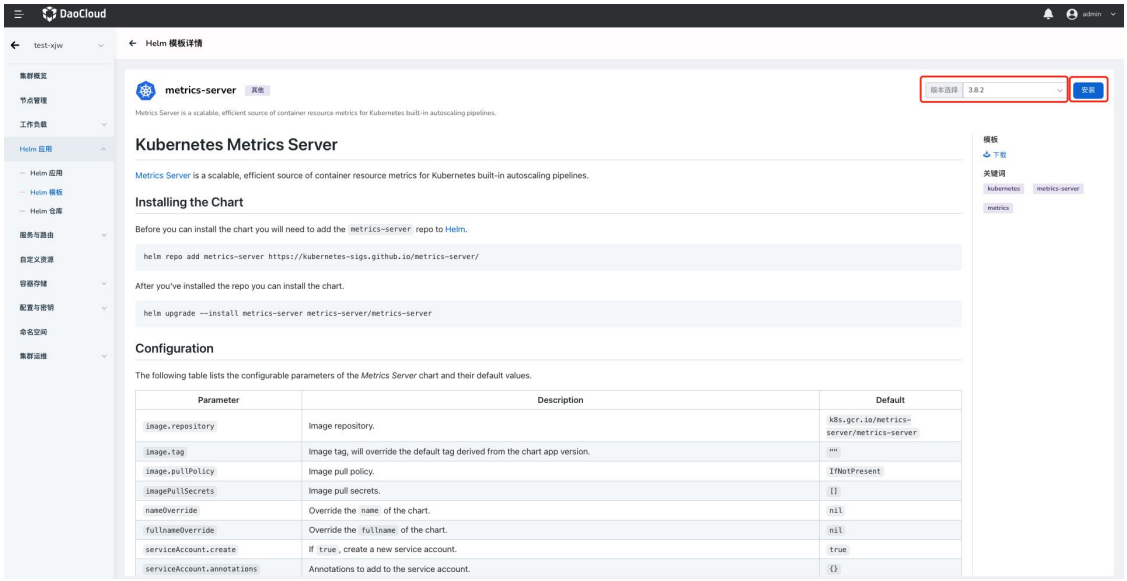
请执行如下步骤为集群安装 metrics-server 插件。

1. 在工作负载详情下的弹性伸缩页面，点击 安装 按钮，进入 metrics-server 插件安装界面。



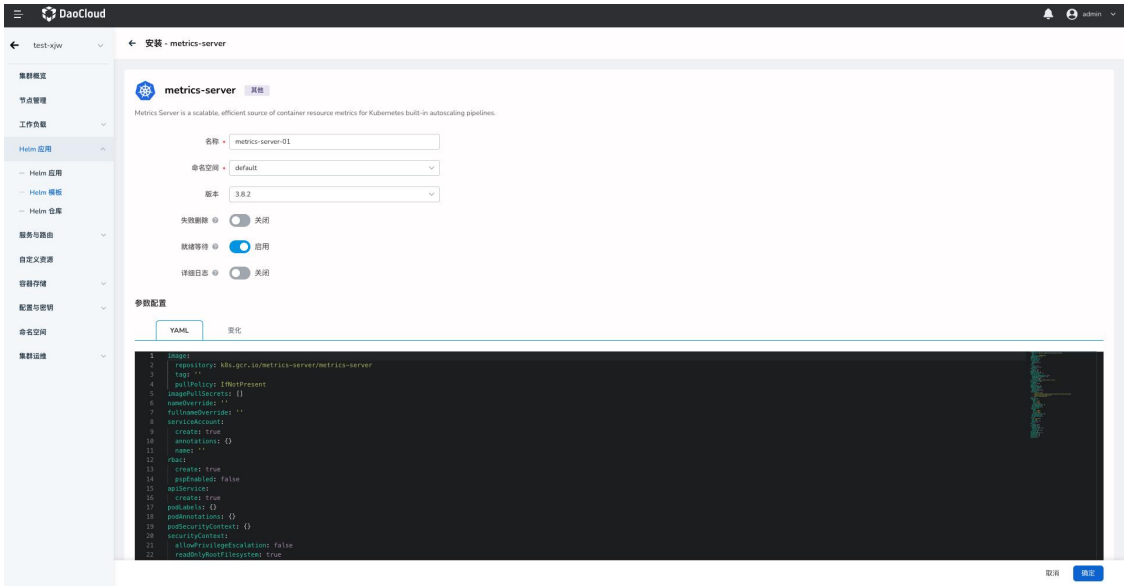
工作负载

2. 阅读 metrics-server 插件相关介绍，选择版本后点击安装按钮。本文将以 3.8.2 版本为例进行安装，推荐您安装 3.8.2 及更高版本。



工作负载

3. 在安装配置界面配置基本参数。



工作负载

- 名称：输入插件名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”）,且必须以小写字母或数字开头及结尾，例如 metrics-server-01。
- 命名空间：选择插件安装的命名空间，此处以 default 为例。
- 版本：插件的版本，此处以 3.8.2 版本为例。
- 就绪等待：启用后，将等待应用下所有关联资源处于就绪状态，才会标记应用安装成功。
- 失败删除：开启后，将默认同步开启就绪等待。如果安装失败，将删除安装相关资源。
- 详情日志：开启安装过程日志的详细输出。

!!! note

开启`就绪等待`和/或`失败删除`后，应用需要经过较长时间才会被标记为`运行中`状态。

4. 高级参数配置

- 如果集群网络无法访问 k8s.gcr.io 仓库，请尝试修改 repositort 参数为 repository: k8s.m.daocloud.io/metrics-server/metrics-server。
- 安装 metrics-server 插件还需提供 SSL 证书。如需绕过证书校验，需要在 defaultArgs: 处添加 - --kubelet-insecure-tls 参数。

??? note“推荐使用如下参数来替换参数配置内的默认 YAML 内容”

```yaml
image:
  repository: k8s.m.daocloud.io/metrics-server/metrics-server # 将仓库源地址修改为 k8s.m.daocloud.io
  tag: ''
  pullPolicy: IfNotPresent
imagePullSecrets: []
nameOverride: ''
fullnameOverride: ''
serviceAccount:
  create: true
  annotations: {}
  name: ''
rbac:
  create: true
  pspEnabled: false
apiService:
  create: true
podLabels: {}
podAnnotations: {}
podSecurityContext: {}
securityContext:
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
  runAsNonRoot: true
```

```
  runAsUser: 1000
priorityClassName: system-cluster-critical
containerPort: 4443
hostNetwork:
  enabled: false
replicas: 1
updateStrategy: {}
podDisruptionBudget:
  enabled: false
  minAvailable: null
  maxUnavailable: null
defaultArgs:
  - '--cert-dir=/tmp'
  - '--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname'
  - '--kubelet-use-node-status-port'
  - '--metric-resolution=15s'
  - --kubelet-insecure-tls # 绕过证书校验
args: []
livenessProbe:
  httpGet:
    path: /livez
    port: https
    scheme: HTTPS
  initialDelaySeconds: 0
  periodSeconds: 10
  failureThreshold: 3
readinessProbe:
  httpGet:
    path: /readyz
    port: https
    scheme: HTTPS
  initialDelaySeconds: 20
```

```
    periodSeconds: 10
    failureThreshold: 3
  service:
    type: ClusterIP
    port: 443
    annotations: {}
    labels: {}
  metrics:
    enabled: false
  serviceMonitor:
    enabled: false
    additionalLabels: {}
    interval: 1m
    scrapeTimeout: 10s
  resources: {}
  extraVolumeMounts: []
  extraVolumes: []
  nodeSelector: {}
  tolerations: []
  affinity: {}
```

5. 点击确定按钮，完成 metrics-server 插件的安装，之后系统将自动跳转至 Helm 应用列表页面，稍等几分钟后，为页面执行刷新操作，即可看到刚刚安装的应用。

!!! note

删除 `metrics-server` 插件时，在 `Helm 应用`列表页面才能彻底删除该插件。如果仅在工作负载页面删除 `metrics-server`，这只是删除了该应用的工作负载副本，应用本身仍未删除，后续重新安装该插件时也会提示错误。

安装 vpa 插件

容器垂直扩缩容策略（Vertical Pod Autoscaler, VPA）能够让集群的资源配置更加合理，避免集群资源浪费。vpa 则是实现容器垂直扩缩容的关键组件。

本节介绍如何安装 vpa 插件。

为了使用 VPA 策略，不仅需要安装 `vpa` 插件，还要[安装 `metrics-server` 插件](install-metrics-server.md)。

前提条件

安装 vpa 插件之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。

操作步骤

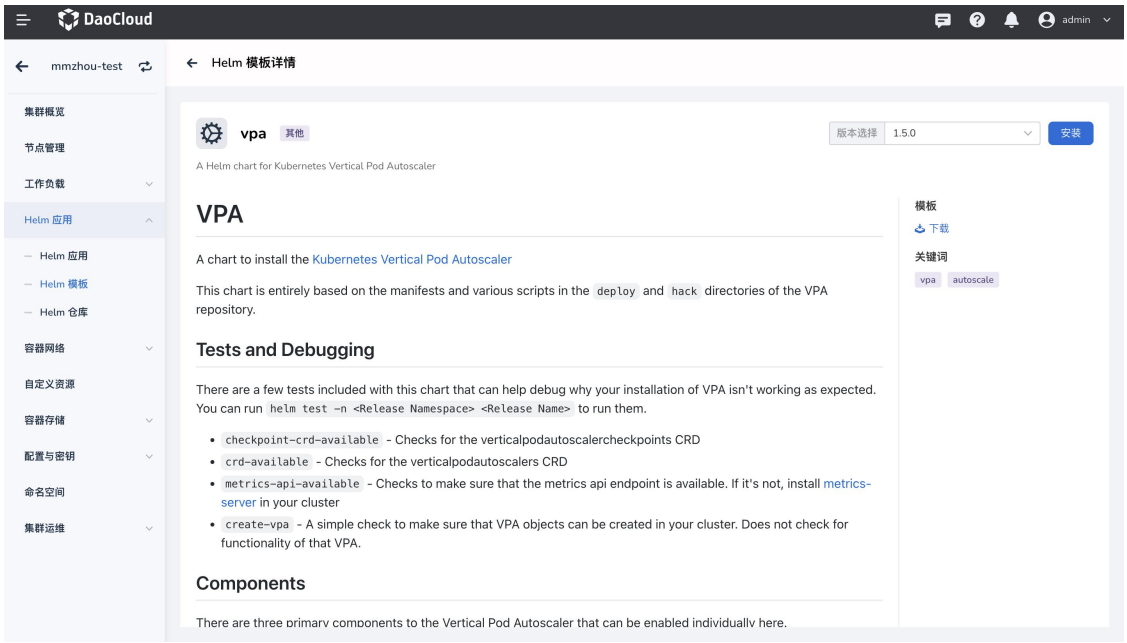
参考如下步骤为集群安装 vpa 插件。

1. 在集群列表页面找到需要安装此插件的目标集群，点击该集群的名称，然后在左侧点击工作负载 -> 无状态工作负载，点击目标工作负载的名称。
2. 在工作负载详情页面，点击弹性伸缩页签，在 VPA 右侧点击 安装。



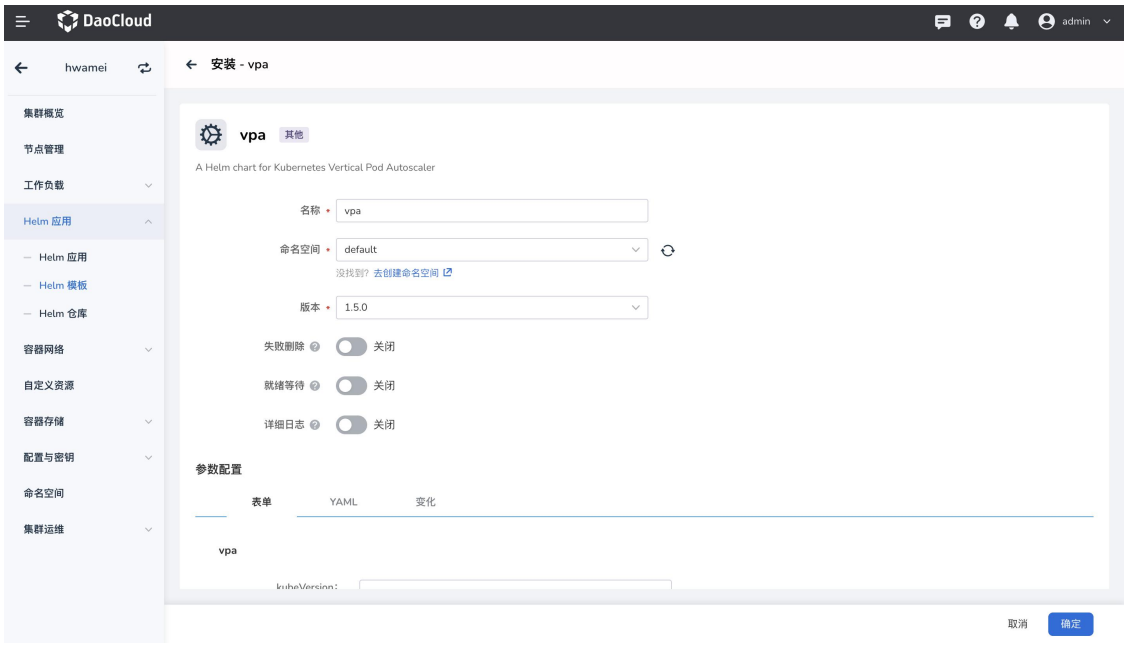
工作负载

3. 阅读该插件的相关介绍，选择版本后点击安装按钮。推荐安装 1.5.0 或更高版本。



工作负载

4. 查看以下说明配置参数。



工作负载

- 名称：输入插件名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 kubernetes-cronhpa-controller。
- 命名空间：选择将插件安装在哪个命名空间，此处以 default 为例。
- 版本：插件的版本，此处以 1.5.0 版本为例。
- 就绪等待：启用后，将等待应用下的所有关联资源都处于就绪状态，才会标记应用安装成功。
- 失败删除：如果插件安装失败，则删除已经安装的关联资源。开启后，将默认同步开启就绪等待。
- 详情日志：开启后，将记录安装过程的详细日志。

!!! note

开启`就绪等待`和/或`失败删除`后，应用需要经过较长时间才会被标记为“运行中”状态。

5. 在页面右下角点击确定，系统将自动跳转至 Helm 应用列表页面。稍等几分钟后刷新页面作，即可看到刚刚安装的应用。

!!! warning

如需删除 vpa 插件，应在 Helm 应用列表页面才能将其彻底删除。

如果在工作负载的弹性伸缩页签下删除插件，这只是删除了该插件的工作负载副本，插件本身仍未删除，后续重新安装该插件时也会提示错误。

6. 回到工作负载详情页面下的弹性伸缩页签，可以看到界面显示 插件已安装。现在可以开始创建 VPA 策略了。



工作负载

离线升级安全管理模块

本页说明从下载中心下载安全管理模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `dowl` 字样是安全管理模块的内部开发代号。

从下载的安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

方式一：使用 chart-syncer 同步镜像

使用 chart-syncer 可以将您下载的安装包中的 chart 及其依赖的镜像包上传至安装器部署 DCE 时使用的镜像仓库和 helm 仓库。

首先找到一台能够连接镜像仓库和 helm 仓库的节点（如火种节点），在节点上创建 load-image.yaml 配置文件，填入镜像仓库和 helm 仓库等配置信息。

1. 创建 load-image.yaml

   !!! note

   该 YAML 文件中的各项参数均为必填项。

   ===“已添加 Helm repo”

   若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

   ```yaml title="load-image.yaml"
   source:
     intermediateBundlesPath: dowl # 节点上执行 load-image.yaml 文件的路
   ```

径。

```
  target:
    containerRegistry: 10.16.10.111 # 镜像仓库地址
    containerRepository: release.daocloud.io/dowl # 镜像仓库路径
    repo:
      kind: HARBOR # Helm Chart 仓库类别
      url: http://10.16.10.111/chartrepo/release.daocloud.io # Helm 仓库地址
      auth:
      username: "admin" # 镜像仓库用户名
      password: "Harbor12345" # 镜像仓库密码
    containers:
      auth:
        username: "admin" # Helm 仓库用户名
        password: "Harbor12345" # Helm 仓库密码
```

===“未添加 Helm repo”

若当前节点上未添加 helm repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: dowl # 节点上执行 load-image.yaml 文件的路径。
target:
  containerRegistry: 10.16.10.111 # 镜像仓库 url
  containerRepository: release.daocloud.io/dowl # 镜像仓库路径
  repo:
    kind: LOCAL
    path: ./local-repo # chart 本地路径
  containers:
    auth:
```

```
        username: "admin" # 镜像仓库用户名
        password: "Harbor12345" # 镜像仓库密码
```

2. 执行同步镜像命令。

   charts-syncer sync --config load-image.yaml

方式二：使用 Docker 或 containerd 加载镜像

解压并加载镜像文件。

1. 解压 tar 压缩包。

   tar xvf dowl.bundle.tar

   解压成功后会得到 3 个文件：

   - hints.yaml
   - images.tar
   - original-chart

2. 从本地加载镜像到 Docker 或 containerd。

   === “Docker”

   ```shell
   docker load -i images.tar
   ```

   === “containerd”

   ```shell
   ctr image import images.tar
   ```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

===“通过 helm repo 升级”

1. 检查安全管理 helm 仓库是否存在。

    ```shell
    helm repo list | grep dowl
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加安全管理的 helm 仓库。

    ```shell
    heml repo add dowl http://{harbor url}/chartrepo/{project}
    ```

1. 更新安全管理的 helm 仓库。

    ```shell
    helm repo update dowl
    ```

1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

1. 选择您想安装的安全管理版本（建议安装最新版本）。

   ```shell
   helm search repo dowl/dowl --versions
   ```

   ```none
   [root@master ~]# helm search repo dowl/dowl --versions
   NAME                    CHART VERSION   APP VERSION   DESCRIPTION
   dowl/dowl   0.4.0           v0.4.0        A Helm chart for dowl
   ...
   ```

1. 备份 `--set` 参数。

   在升级安全管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values dowl -n dowl-system -o yaml > bak.yaml
   ```

1. 更新 dowl crds

   ```shell
   helm pull dowl/dowl --version 0.4.0 && tar -zxf dowl-0.4.0.tgz
   kubectl apply -f dowl/crds
   ```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade dowl dowl/dowl \
  -n dowl-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.4.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

在升级安全管理版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell
helm get values dowl -n k pan da-system -o yaml > bak.yaml
```

1. 更新 dowl crds

```shell
kubectl apply -f ./crds
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade dowl . \
  -n dowl-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

安全扫描类型

DCE 5.0 容器管理提供三种扫描类型：

- 合规性扫描：基于 CIS Benchmark 对集群节点进行安全扫描。
- 权限扫描：检查 Kubernetes 集群中的安全问题和合规性问题，记录并验证对 Kubernetes API 的授权访问、对象更改、事件和其它活动。
- 漏洞扫描：扫描 Kubernetes 集群中可能存在漏洞和风险，例如未经授权的访问、敏感信息泄露、弱身份验证、容器逃逸等。

合规性扫描

合规性扫描的对象是集群节点。扫描结果中会列出扫描项以及扫描结果，并针对未通过的扫描项给出修复建议。有关扫描时用到的具体安全规则，可参考 CIS Kubernetes Benchmark

检查不同类型的节点时，扫描的侧重点有所不同。

- 扫描控制平面节点（Controller）
    - 关注 API Server、controller-manager、scheduler、kubelet 等系统组件的安全性
    - 检查 Etcd 数据库的安全配置
    - 检查集群身份验证机制、授权策略和网络安全配置是否符合安全标准
- 扫描工作节点（Worker）
    - 检查 kubelet、Docker 等容器运行时的配置否符合安全标准
    - 检查容器镜像是否经过信任验证
    - 检查节点的网络安全配置否符合安全标准

!!! tip

使用合规性扫描时，需要先创建[扫描配置](cis/config.md)，然后基于该配置创建[扫描策略](cis/policy.md)。执行扫描策略之后，可以[查看扫描报告](cis/report.md)。

权限扫描

权限扫描侧重于权限问题引发的安全漏洞。权限扫描可以帮助用户识别 Kubernetes 集群中的安全威胁，标识哪些资源需要进行进一步的审查和保护措施。通过执行这些检查项，用户可以更清楚、更全面地了解自己的 Kubernetes 环境，确保集群环境符合 Kubernetes 的最佳实践和安全标准。

具体而言，权限扫描支持以下操作：

- 扫描集群中的所有节点的健康状态。
- 扫描集群组件的运行状况，如 kube-apiserver、kube-controller-manager、kube-scheduler 等。
- 扫描安全配置：检查 Kubernetes 的安全配置
    - API 安全：启用了不安全的 API 版本，是否设置了适当的 RBAC 角色和权限限制等
    - 容器安全：是否使用了不安全的 Image、是否开放了特权模式，是否设置了合适的安全上下文等
    - 网络安全：是否启用了合适的网络策略来限制流量，是否使用了 TLS 加密等
    - 存储安全：是否启用了适当的加密、访问控制等。
    - 应用程序安全：是否设置了必要的安全措施，例如密码管理、跨站脚本攻击防御等。
- 提供警告和建议：建议集群管理员执行的安全最佳实践，例如定期轮换证书、使用强密码、限制网络访问等。

!!! tip

使用合规性扫描时，需要先创建扫描策略。执行扫描策略之后，可以查看扫描报告。详情可参考[安全扫描](audit.md)。

漏洞扫描

漏洞扫描侧重于扫描潜在的恶意攻击和安全漏洞，例如远程代码执行、SQL 注入、XSS 攻击等，以及一些针对 Kubernetes 特定的攻击。最终的扫描报告会列出集群中存在的安全漏洞，并提出修复建议。

!!! tip

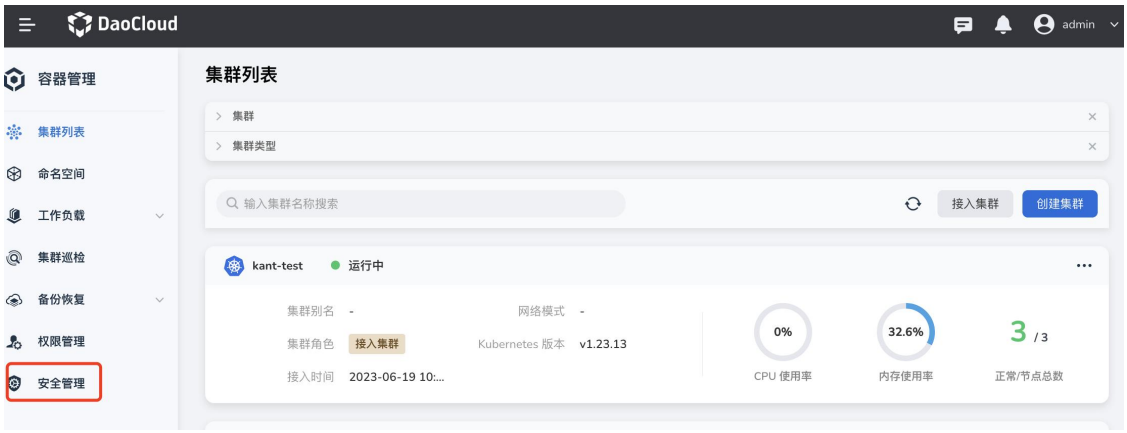
使用合规性扫描时，需要先创建扫描策略。执行扫描策略之后，可以查看扫描报告。详情可参考[漏洞扫描](hunter.md)。

权限扫描

为了使用权限扫描功能，需要先创建扫描策略，执行该策略之后会自动生成扫描报告以供查看。

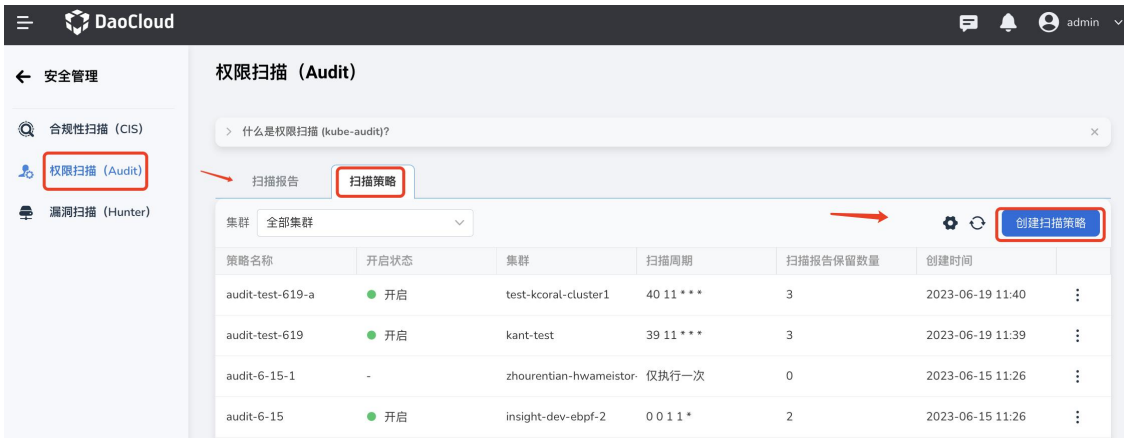
创建扫描策略

1. 在容器管理模块的首页左侧导航栏点击安全管理。



安全管理

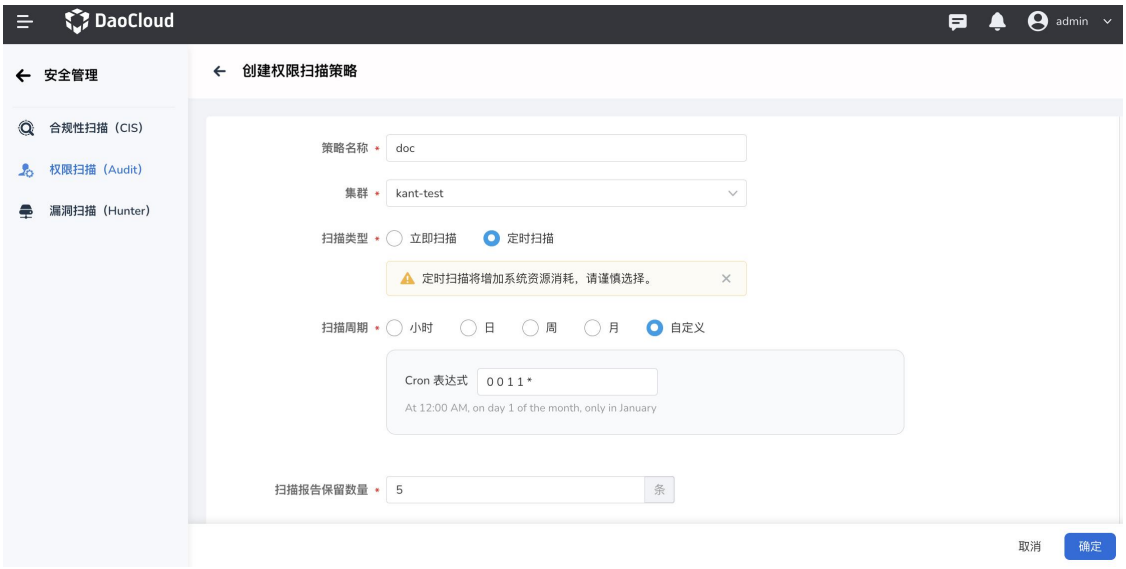
2. 在左侧导航栏点击权限扫描，点击扫描策略页签，在右侧点击创建扫描策略。



安全管理

3. 参考下列说明填写配置，最后点击确定即可。

- 集群：选择需要扫描哪个集群。可选的集群列表来自于容器管理模块中接入或创建的集群。如果没有想选的集群，可以去容器管理模块中接入或创建集群。
- 扫描类型：
  - 立即扫描：在扫描策略创建好之后立即执行一次扫描，后续不可以自动/手动再次执行扫描。
  - 定时扫描：通过设置扫描周期，自动按时重复执行扫描。
- 扫描报告保留数量：设置最多保留多少扫描报告。超过指定的保留数量时，从最早的报告开始删除。



安全管理

更新/删除扫描策略

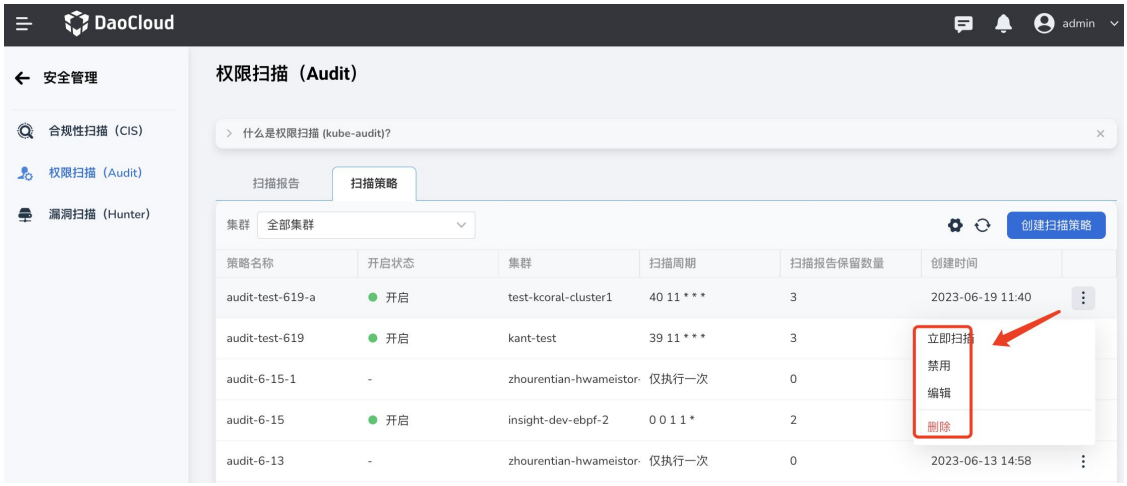
创建扫描策略之后，可以根据需要更新或删除扫描策略。

在扫描策略页签下，点击配置右侧的　　操作按钮：

- 对于周期性的扫描策略：
  - 选择立即执行意味着，在周期计划之外立即再扫描一次集群

- 选择禁用会中断扫描计划，直到点击启用才可以继续根据周期计划执行该扫描策略。
- 选择编辑可以更新配置，支持更新扫描配置、类型、扫描周期、报告保留数量，不可更改配置名称和需要扫描的目标集群。
- 选择删除可以删除该配置

- 对于一次性的扫描策略：仅支持删除操作。

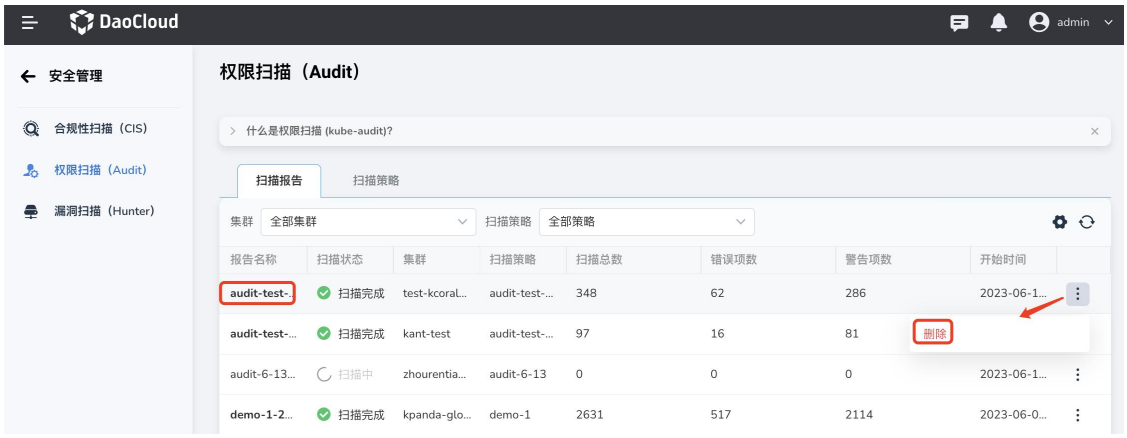


创建扫描配置

查看扫描报告

1. 在安全管理–>权限扫描–>扫描报告页签下，点击报告名称

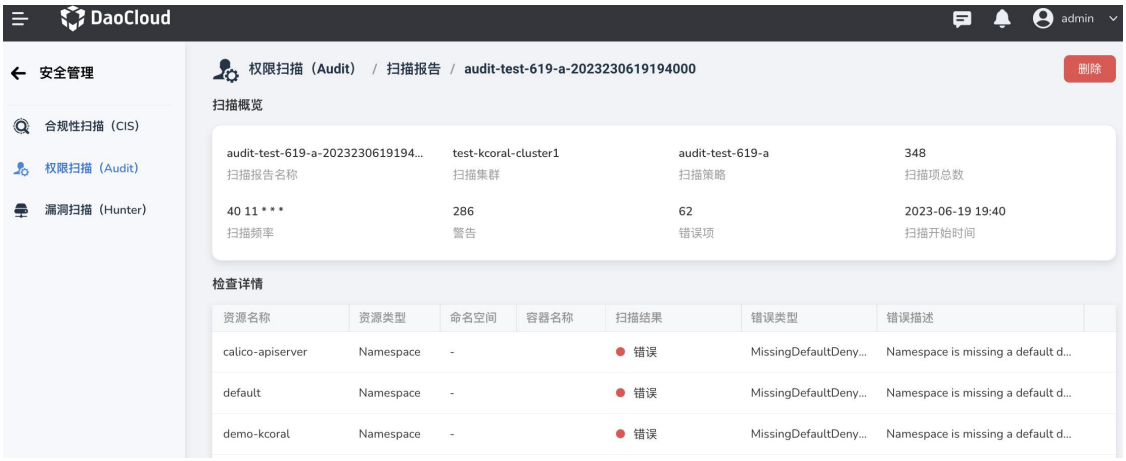
   在报告右侧点击删除可以手动删除报告。



创建扫描配置

2. 查看扫描报告内容，包括：

   - 扫描的目标集群
   - 使用的扫描策略
   - 扫描项总数、警告数、错误数
   - 在周期性扫描策略生成的扫描报告中，还可以查看扫描频率
   - 扫描开始的时间
   - 检查详情，例如被检查的资源、资源类型、扫描结果、错误类型、错误详情


DaoCloud
admin
安全管理
合规性扫描（CIS）
权限扫描（Audit）
漏洞扫描（Hunter）
权限扫描（Audit） / 扫描报告 / audit-test-619-a-2023230619194000
删除
扫描概览
audit-test-619-a-2023230619194...
扫描报告名称
test-kcoral-cluster1
扫描集群
audit-test-619-a
扫描策略
348
扫描项总数
40 11 * * *
扫描频率
286
警告
62
错误项
2023-06-19 19:40
扫描开始时间
检查详情
资源名称
资源类型
命名空间
容器名称
扫描结果
错误类型
错误描述
calico-apiserver
Namespace
-
错误
MissingDefaultDeny...
Namespace is missing a default d...
default
Namespace
-
错误
MissingDefaultDeny...
Namespace is missing a default d...
demo-kcoral
Namespace
-
错误
MissingDefaultDeny...
Namespace is missing a default d...

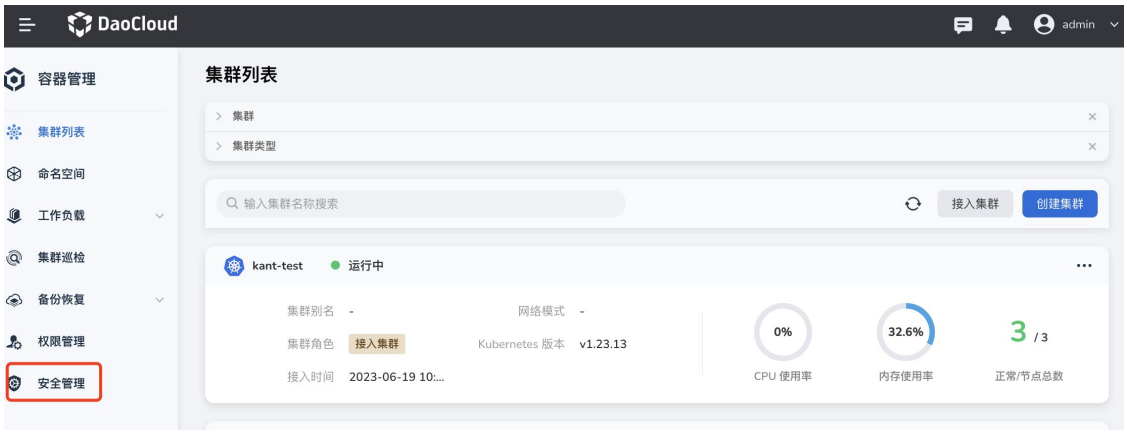
创建扫描配置

漏洞扫描

为了使用漏洞扫描功能，需要先创建扫描策略，执行该策略之后会自动生成扫描报告以供查看。

创建扫描策略

1. 在容器管理模块的首页左侧导航栏点击安全管理。



安全管理

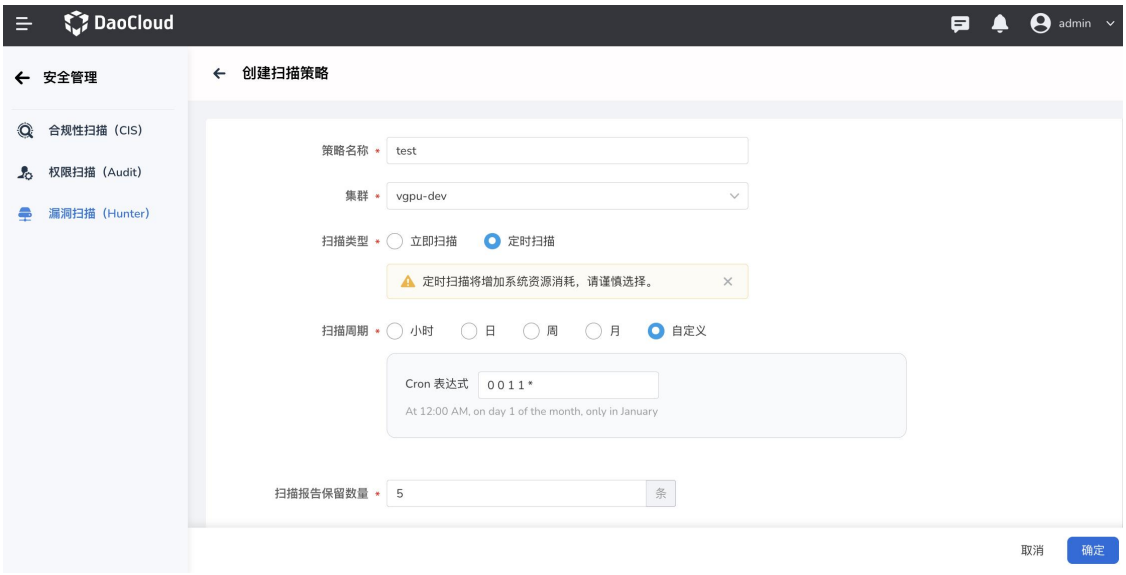
2. 在左侧导航栏点击漏洞扫描，点击扫描策略页签，在右侧点击创建扫描策略。



安全管理

3. 参考下列说明填写配置，最后点击确定即可。

- 集群：选择需要扫描哪个集群。可选的集群列表来自于容器管理模块中接入或创建的集群。如果没有想选的集群，可以去容器管理模块中接入或创建集群。
- 扫描类型：
  - 立即扫描：在扫描策略创建好之后立即执行一次扫描，后续不可以自动/手动再次执行扫描。
  - 定时扫描：通过设置扫描周期，自动按时重复执行扫描。
- 扫描报告保留数量：设置最多保留多少扫描报告。超过指定的保留数量时，从最早的报告开始删除。



安全管理

更新/删除扫描策略

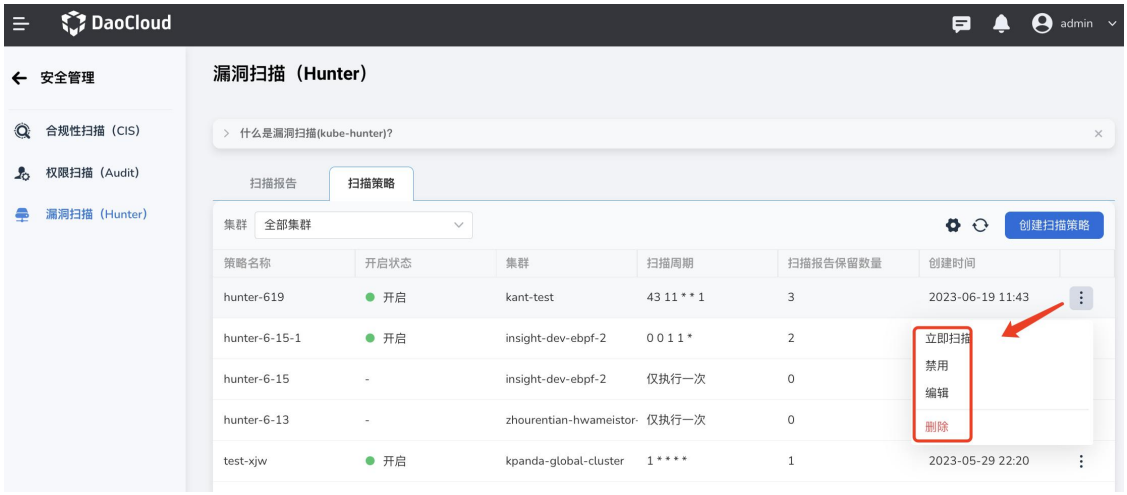
创建扫描策略之后，可以根据需要更新或删除扫描策略。

在扫描策略页签下，点击配置右侧的　　操作按钮：

- 对于周期性的扫描策略：
  - 选择立即执行意味着，在周期计划之外立即再扫描一次集群

- 选择禁用会中断扫描计划，直到点击启用才可以继续根据周期计划执行该扫描策略。
- 选择编辑可以更新配置，支持更新扫描配置、类型、扫描周期、报告保留数量，不可更改配置名称和需要扫描的目标集群。
- 选择删除可以删除该配置

- 对于一次性的扫描策略：仅支持删除操作。

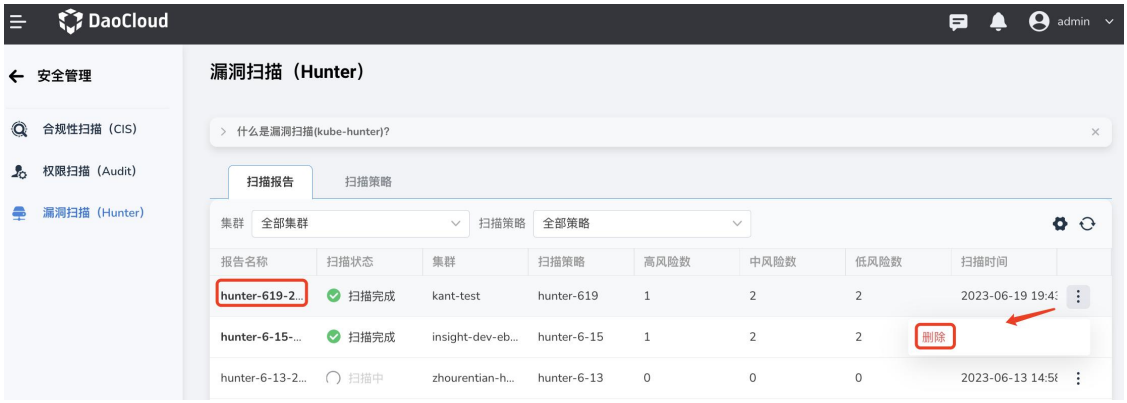


创建扫描配置

查看扫描报告

1. 在安全管理–>权限扫描–>扫描报告页签下，点击报告名称

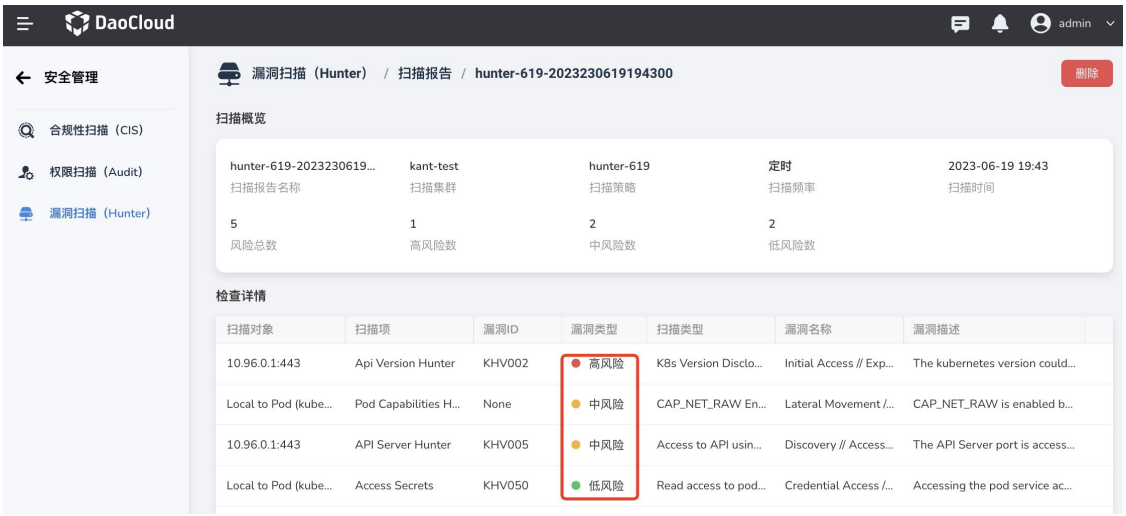
   在报告右侧点击删除可以手动删除报告。



创建扫描配置

2. 查看扫描报告内容，包括：

   - 扫描的目标集群
   - 使用的扫描策略
   - 扫描频率
   - 风险总数、高风险数、中风险数、低风险数
   - 扫描时间
   - 检查详情，例如漏洞 ID、漏洞类型、漏洞名称、漏洞描述等



创建扫描配置

hide:

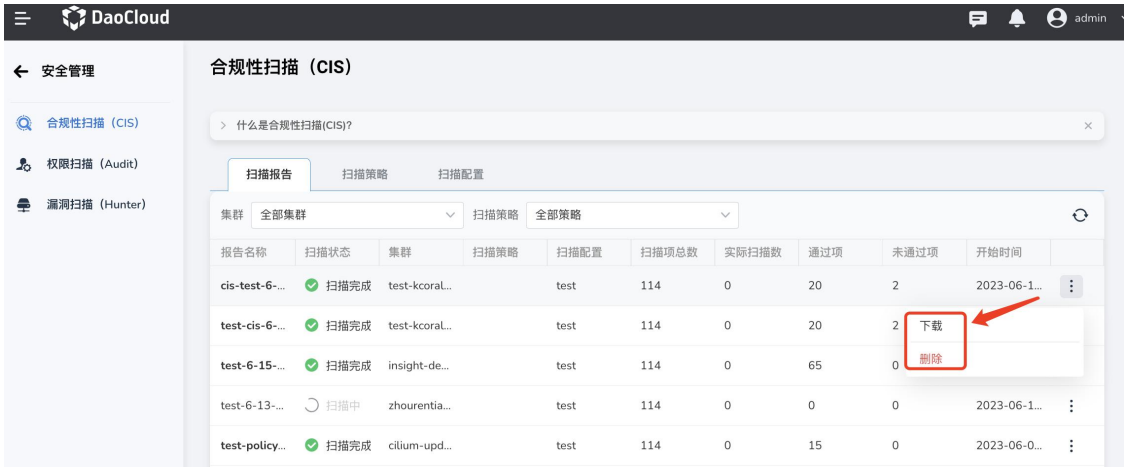
– toc

扫描报告

执行扫描策略之后会自动生成扫描报告。您可以在线查看扫描报告或将其下载到本地查看。

- 下载查看

  安全管理–>合规性扫描页面的扫描报告页签点击报告右侧的　　操作按钮选择下载。



报告列表截图

- 在线查看

  点击报告名称可以在线查看报告内容。具体包括：

  – 扫描的目标集群
  – 使用的扫描策略和扫描配置
  – 扫描开始时间
  – 扫描项总数、通过数与未通过数

- 对于未通过的扫描项给出对应的修复建议
- 对于通过的扫描项给出更安全的操作建议

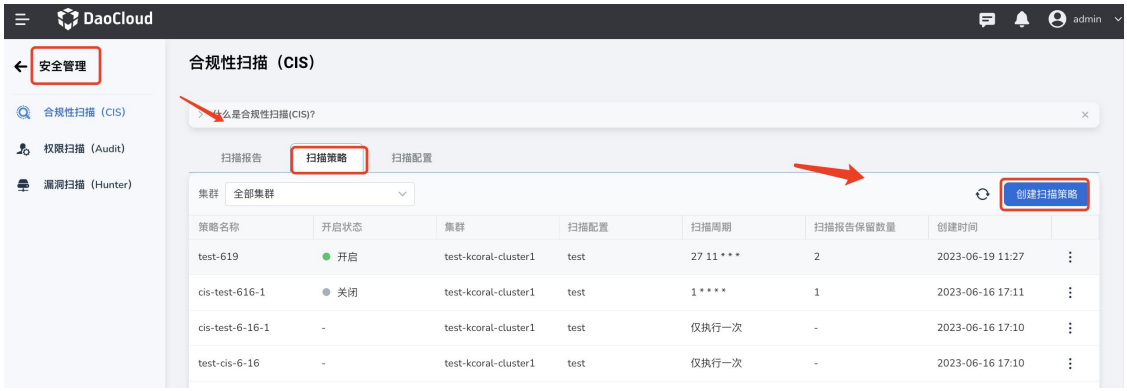
![报告列表截图](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/kpanda/user-guide/images/security10.png)

扫描策略

创建扫描策略

创建扫描配置之后，可以基于配置创建扫描策略。

1. 在安全管理–>合规性扫描页面的扫描策略页签下，在右侧点击创建扫描策略。

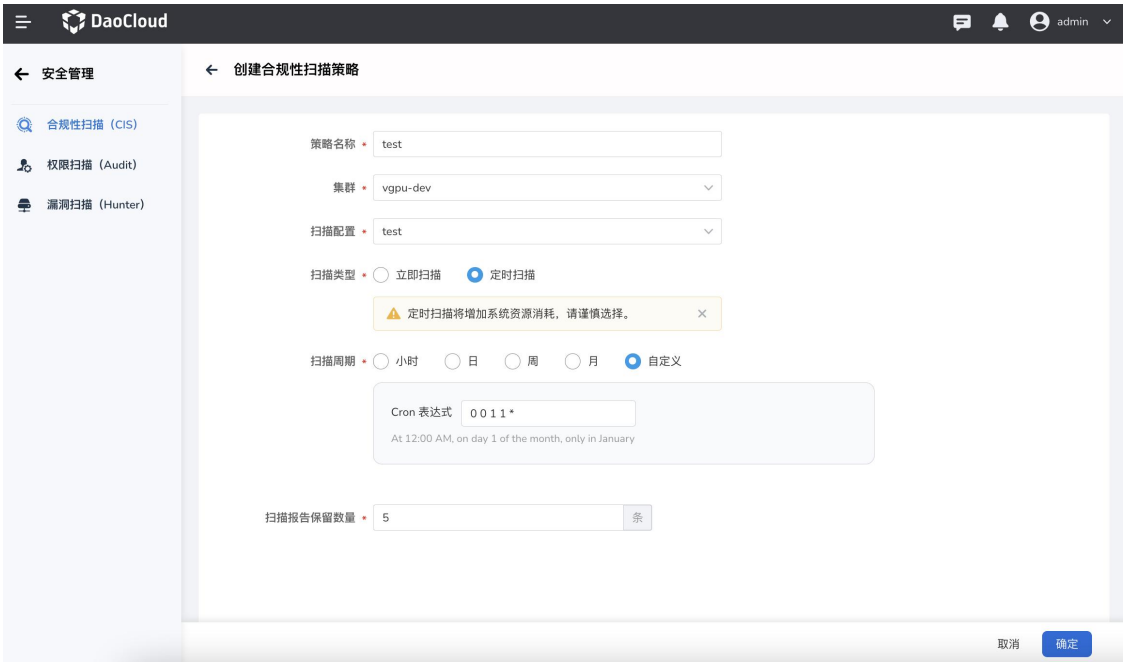


创建扫描配置

2. 参考下列说明填写配置后，点击确定。

   - 集群：选择需要扫描哪个集群。可选的集群列表来自于容器管理模块中接入或创建的集群。如果没有想选的集群，可以去容器管理模块中接入或创建集群。
   - 扫描配置：选择事先创建好的扫描配置。扫描配置规定了需要执行哪些具体的扫描项。
   - 扫描类型：
     - 立即扫描：在扫描策略创建好之后立即执行一次扫描，后续不可以自动/手动再次执行扫描。
     - 定时扫描：通过设置扫描周期，自动按时重复执行扫描。

- 扫描报告保留数量：设置最多保留多少扫描报告。超过指定的保留数量时，从最早的报告开始删除。



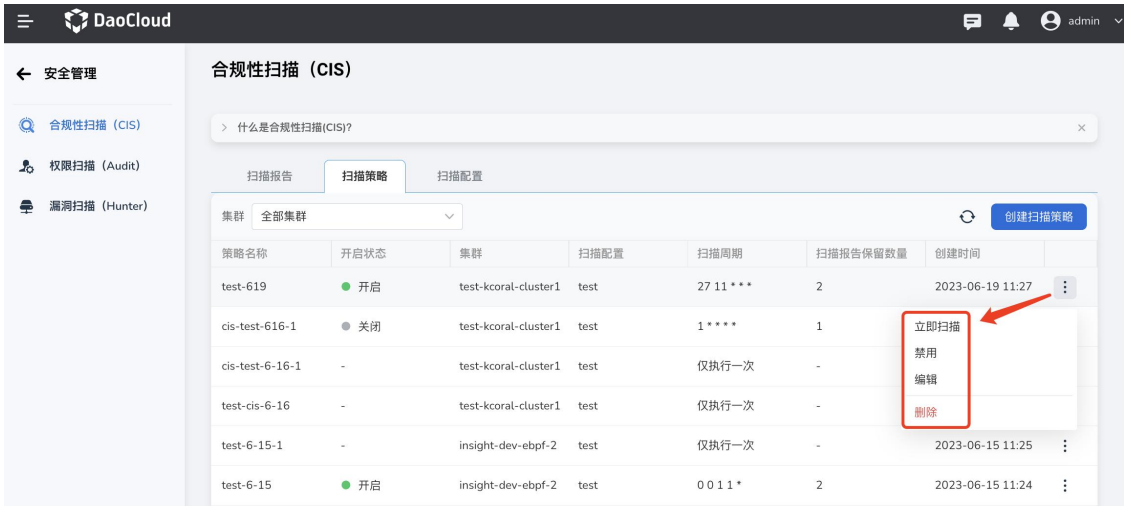
创建扫描配置

更新/删除扫描策略

创建扫描策略之后，可以根据需要更新或删除扫描策略。

在扫描策略页签下，点击配置右侧的　　操作按钮：

- 对于周期性的扫描策略：
  - 选择立即执行意味着，在周期计划之外立即再扫描一次集群
  - 选择禁用会中断扫描计划，直到点击启用才可以继续根据周期计划执行该扫描策略。
  - 选择编辑可以更新配置，支持更新扫描配置、类型、扫描周期、报告保留数量，不可更改配置名称和需要扫描的目标集群。
  - 选择删除可以删除该配置
- 对于一次性的扫描策略：仅支持删除操作。

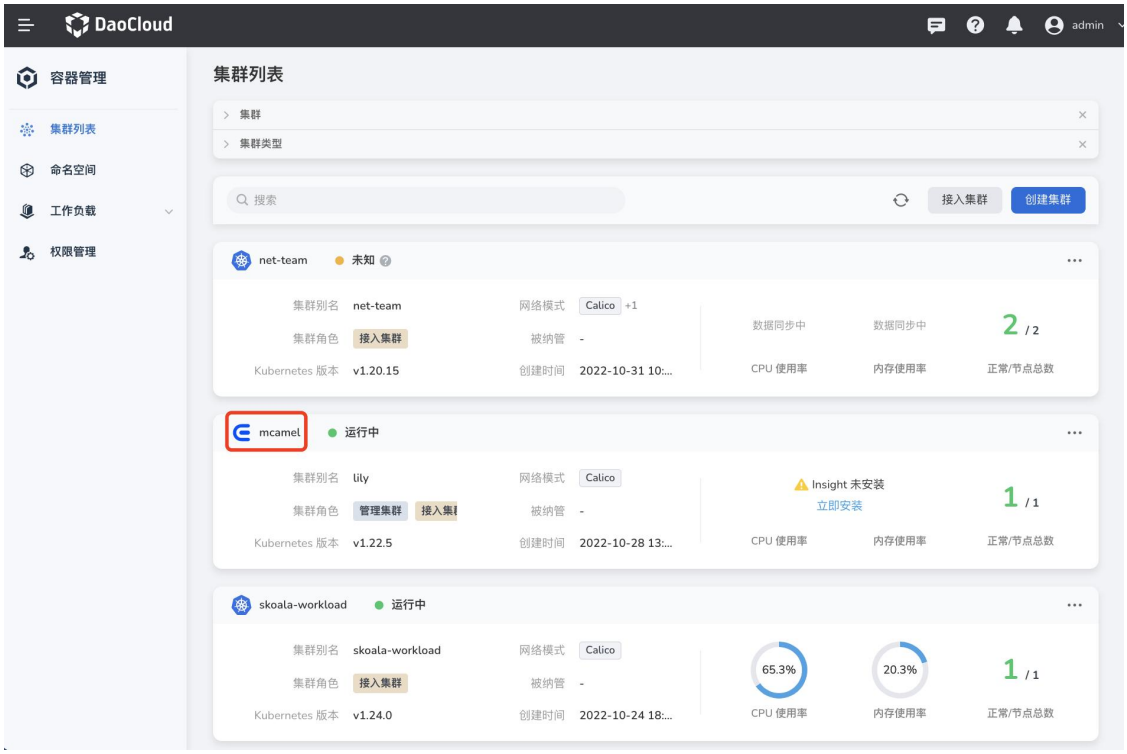

DaoCloud
admin
安全管理
合规性扫描（CIS）
权限扫描（Audit）
漏洞扫描（Hunter）
合规性扫描（CIS）
什么是合规性扫描(CIS)?
扫描报告
扫描策略
扫描配置
集群
全部集群
创建扫描策略
策略名称
开启状态
集群
扫描配置
扫描周期
扫描报告保留数量
创建时间
test-619
开启
test-kcoral-cluster1
test
27 11 * * *
2
2023-06-19 11:27
cis-test-616-1
关闭
test-kcoral-cluster1
test
1 * * * *
1
cis-test-6-16-1
-
test-kcoral-cluster1
test
仅执行一次
-
test-cis-6-16
-
test-kcoral-cluster1
test
仅执行一次
-
test-6-15-1
-
insight-dev-ebpf-2
test
仅执行一次
-
2023-06-15 11:25
test-6-15
开启
insight-dev-ebpf-2
test
0 0 1 1 *
2
2023-06-15 11:24
立即扫描
禁用
编辑
删除

创建扫描配置

节点详情

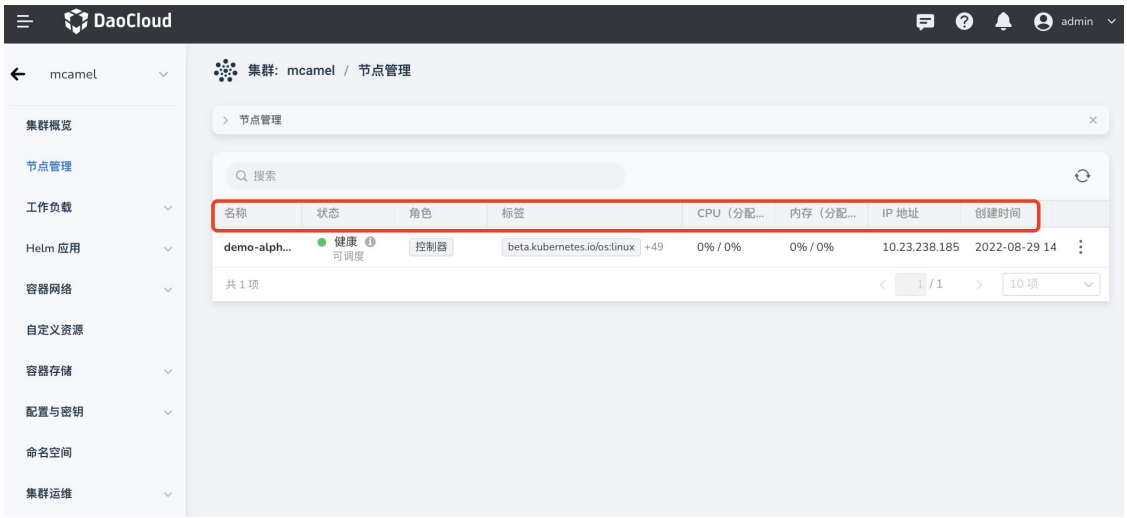
接入或创建集群之后，可以查看集群中各个节点的信息，包括节点状态、标签、资源用量、Pod、监控信息等。

1. 在集群列表页面点击目标集群的名称。



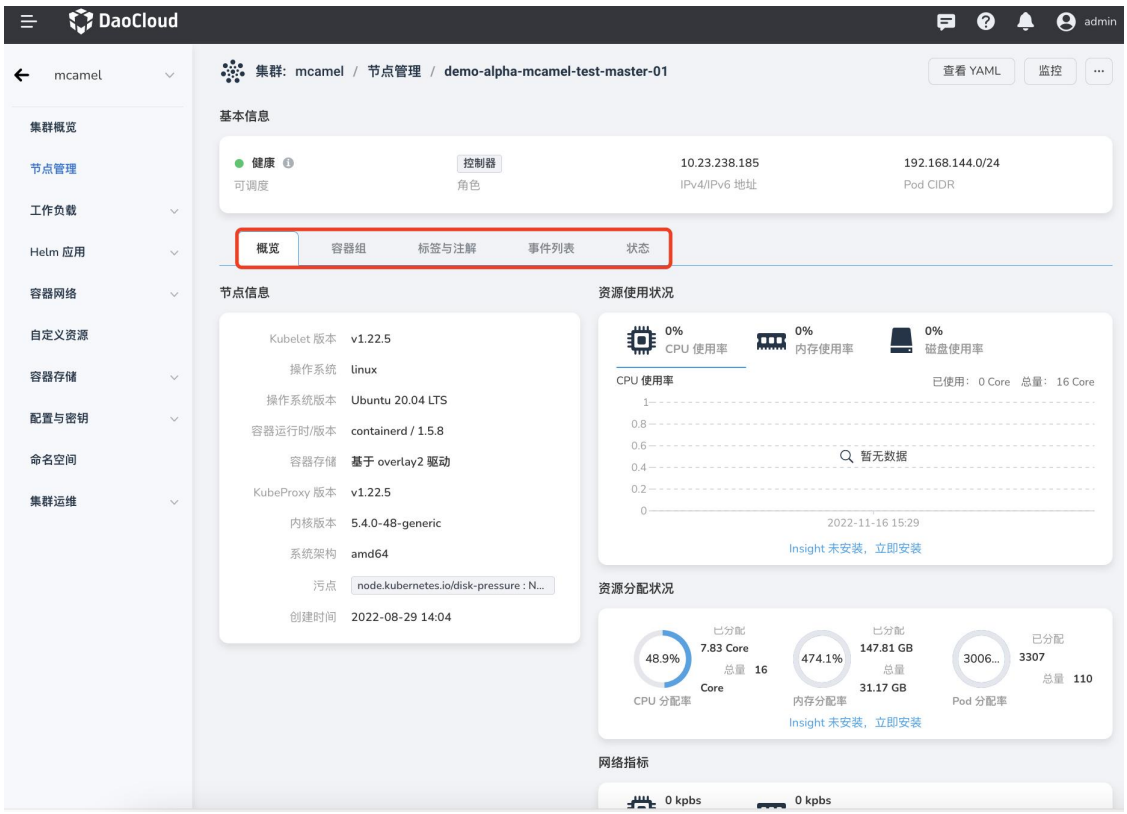
进入集群列表页面

2. 在左侧导航栏点击节点管理，可以查看节点状态、角色、标签、CPU/内存使用情况、IP 地址、创建时间。



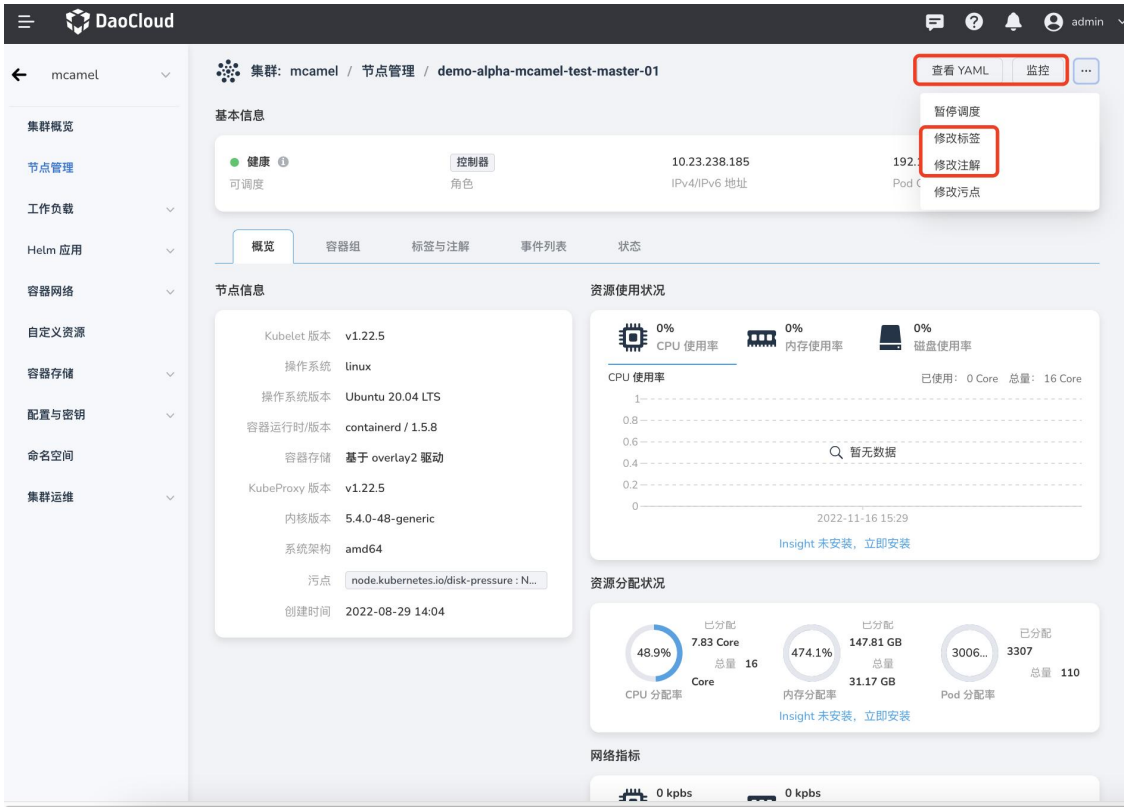
暂停调度

3. 点击节点名称，可以进入节点详情页面查看更多信息，包括概览信息、容器组信息、标签注解信息、事件列表、状态等。



节点管理

此外，还可以查看节点的 YAML 文件、监控信息、标签和注解等。



节点管理

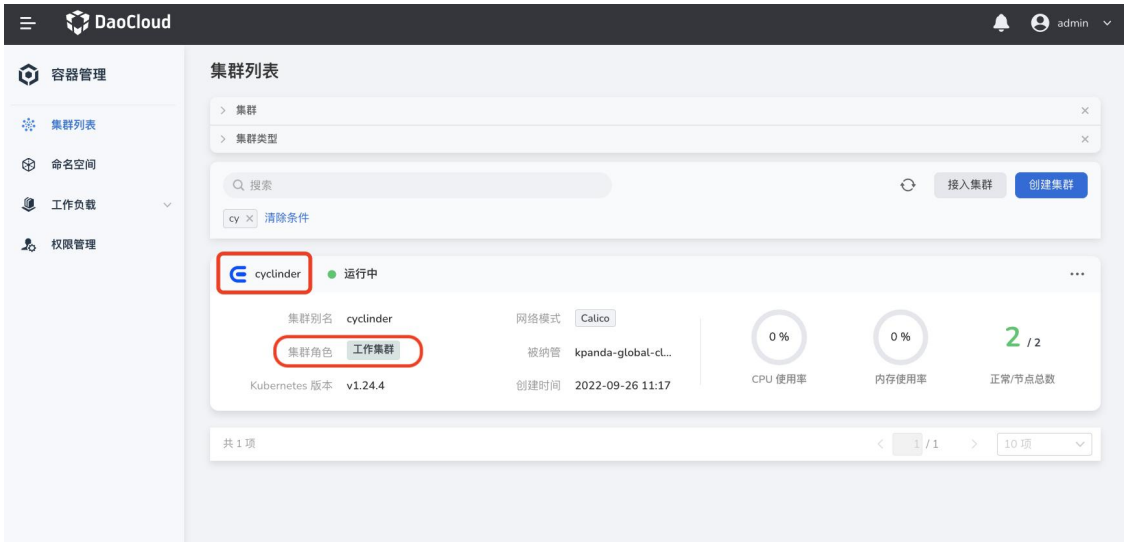
集群节点扩容

随着业务应用不断增长，集群资源日趋紧张，这时可以基于 kubean 对集群节点进行扩容。扩容后，应用可以运行在新增的节点上，缓解资源压力。

只有通过容器管理模块创建的集群才支持节点扩缩容，从外部接入的集群不支持此操作。本文主要介绍同种架构下工作集群的 **工作节点** 扩容，如需为集群增加控制节点或异构工作节点，请参阅：对工作集群的控制节点扩容、为工作集群添加异构节点、为全局服务集群的工作节点扩容。

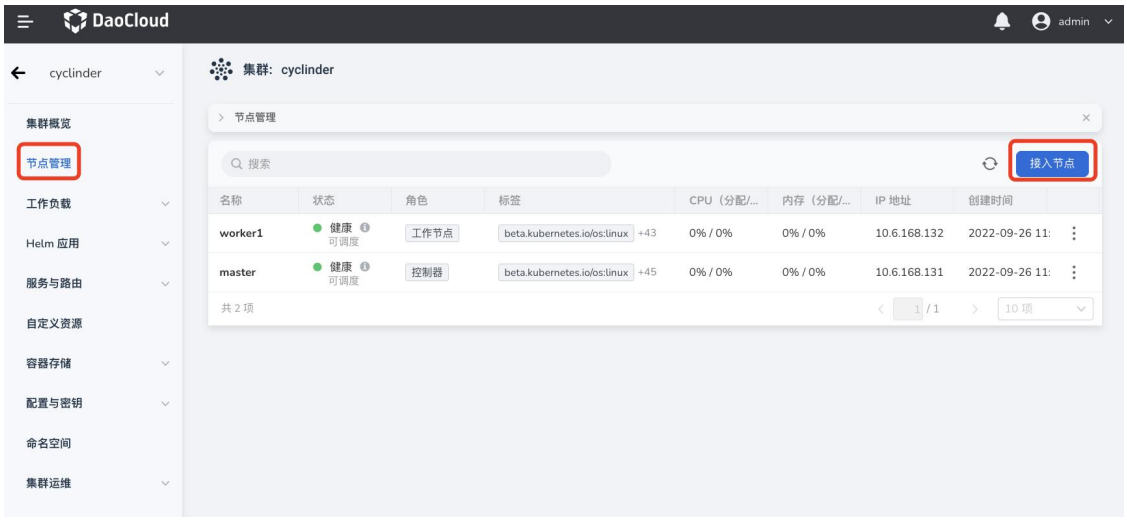
1. 在集群列表页面点击目标集群的名称。

   若集群角色中带有接入集群的标签，则说明该集群不支持节点扩缩容。



   进入集群列表页面

2. 在左侧导航栏点击节点管理，然后在页面右上角点击接入节点。



节点管理

3. 输入主机名称和节点 IP 并点击确定。

   点击 添加工作节点可以继续接入更多节点。



节点管理

!!! note

接入节点大约需要 20 分钟，请您耐心等待。

标签与注解

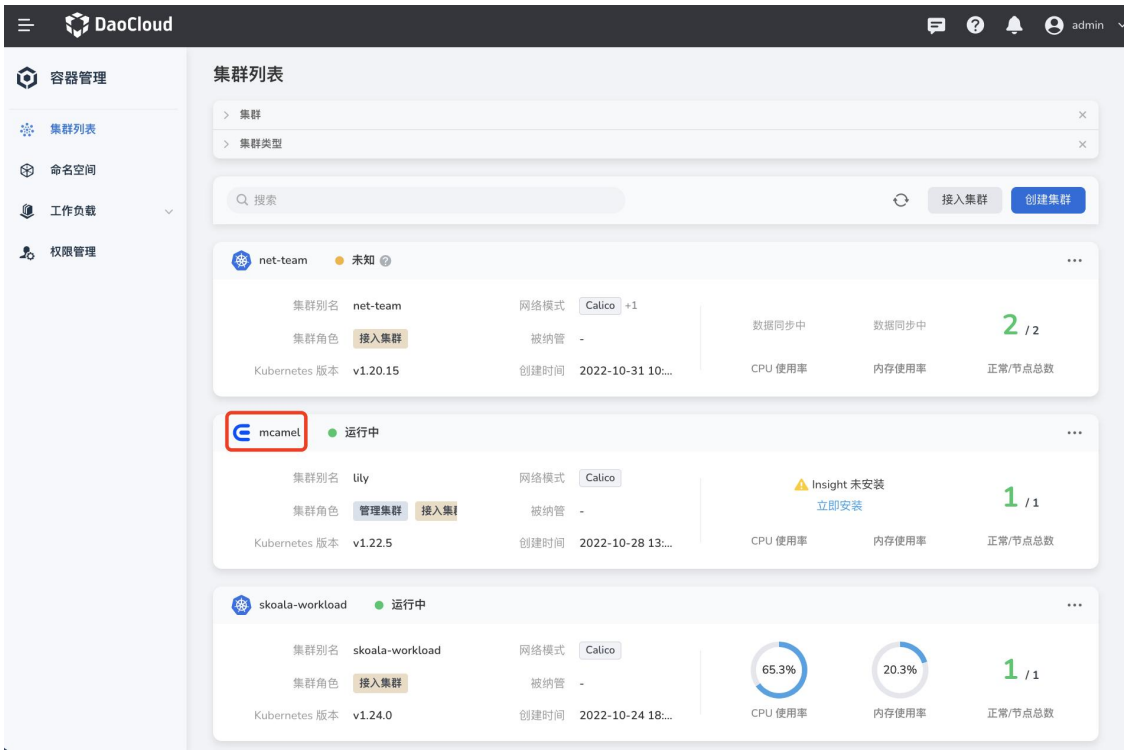
标签（Labels）是为 Pod、节点、集群等 Kubernetes 对象添加的标识性键值对，可结合标签选择器查找并筛选满足某些条件的 Kubernetes 对象。每个键对于给定对象必须是唯一的。

注解（Annotations）和标签一样，也是键/值对，但不具备标识或筛选功能。 使用注解可以为节点添加任意的元数据。 注解的键通常使用的格式为前缀（可选）/名称（必填），例如 nfd.node.kubernetes.io/extended-resources。 如果省略前缀，表示该注解键是用户私有的。

有关标签和注解的更多信息，可参考 Kubernetes 的官方文档标签和选择算符或注解。

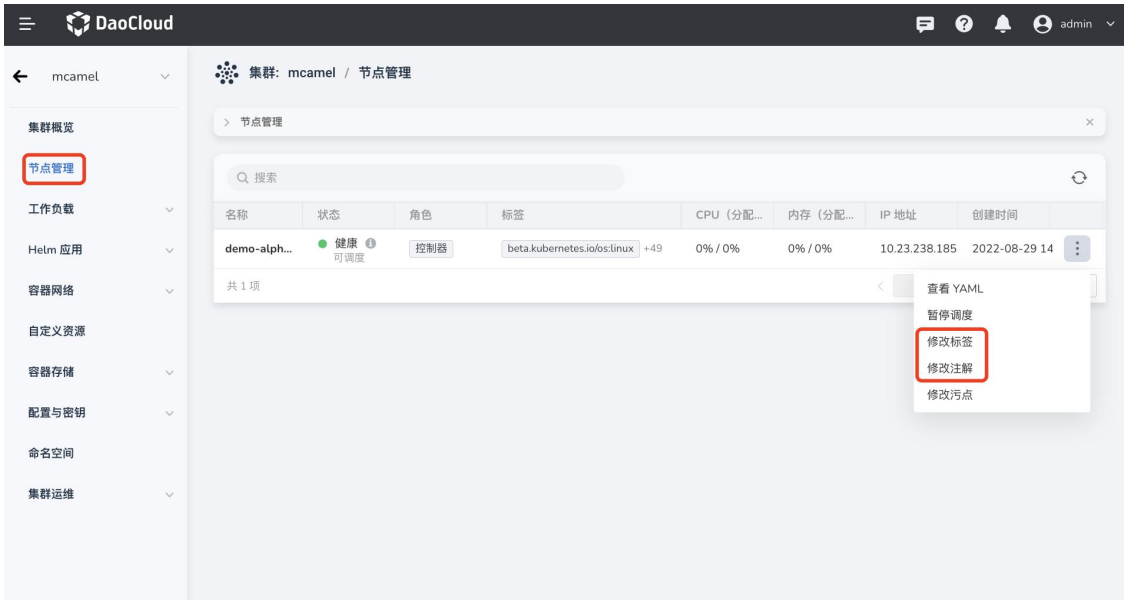
添加/删除标签与注解的步骤如下：

1. 在集群列表页面点击目标集群的名称。



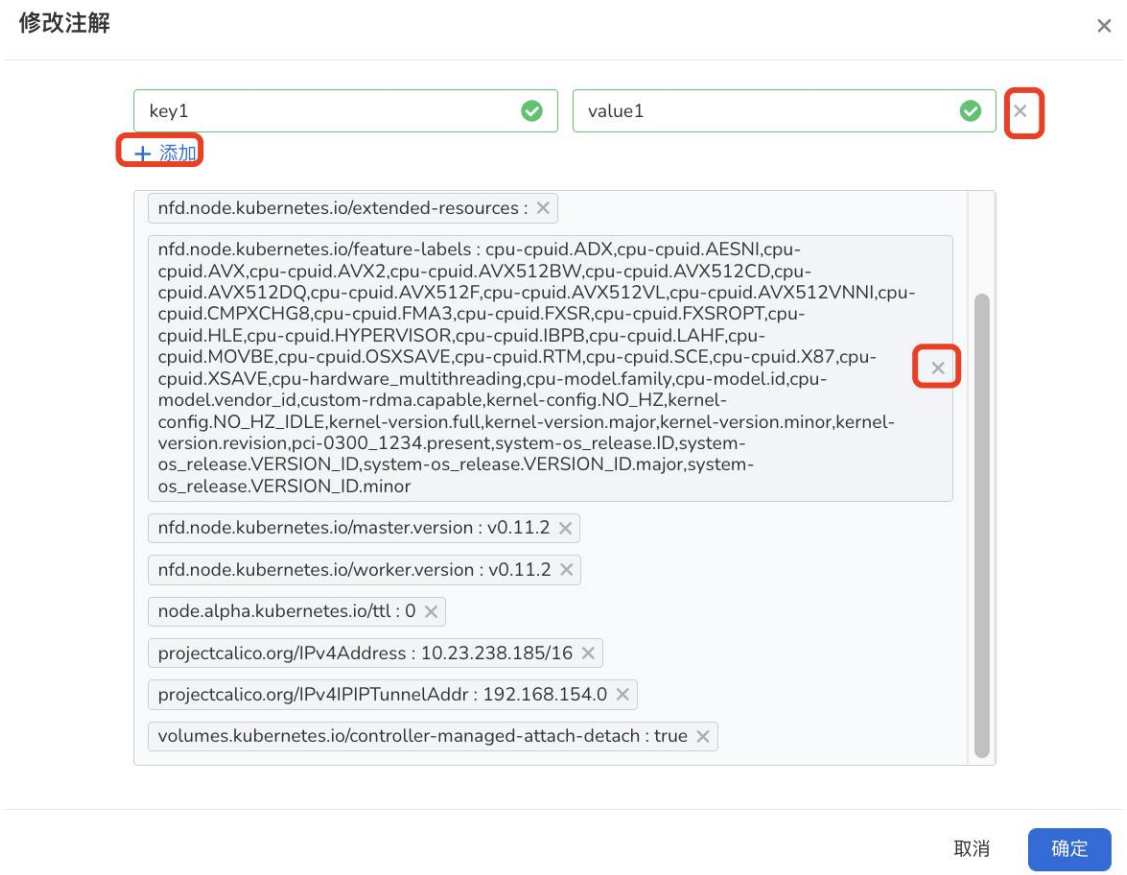
进入集群列表页面

2. 在左侧导航栏点击节点管理，在节点右侧点击　　操作图标，点击修改标签或修改注解。



暂停调度

3. 点击 添加可以添加标签或注解，点击 X 可以删除标签或注解，最后点击确定。

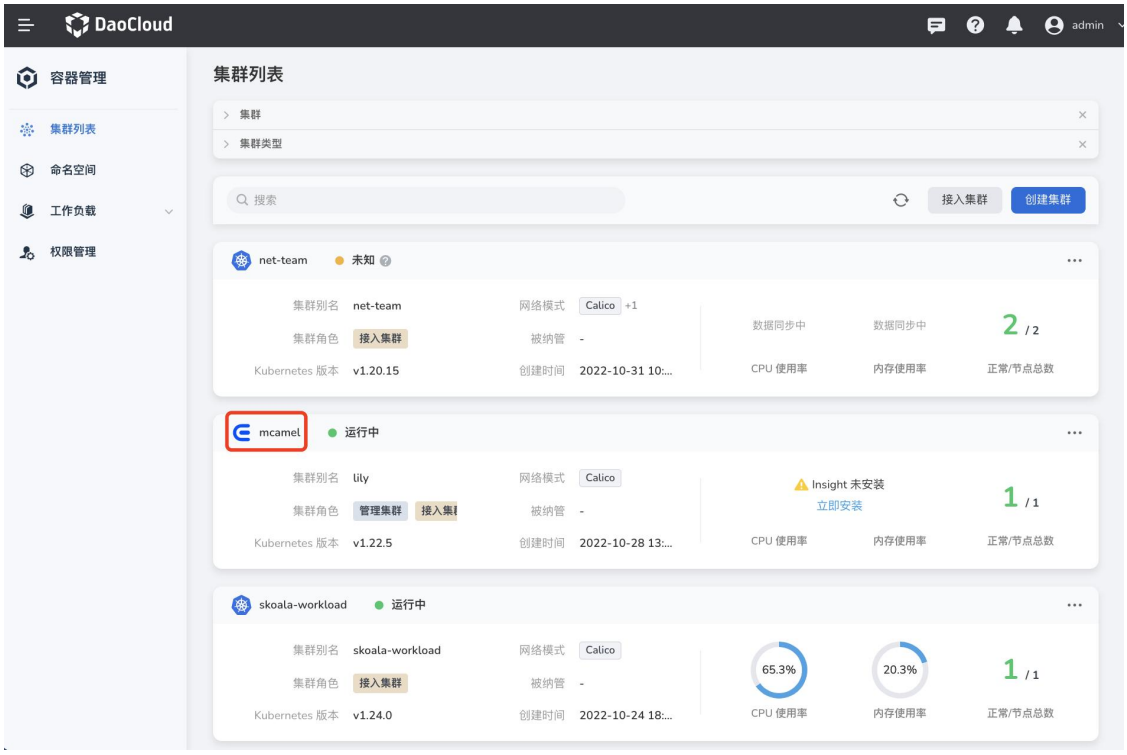


节点管理

节点调度

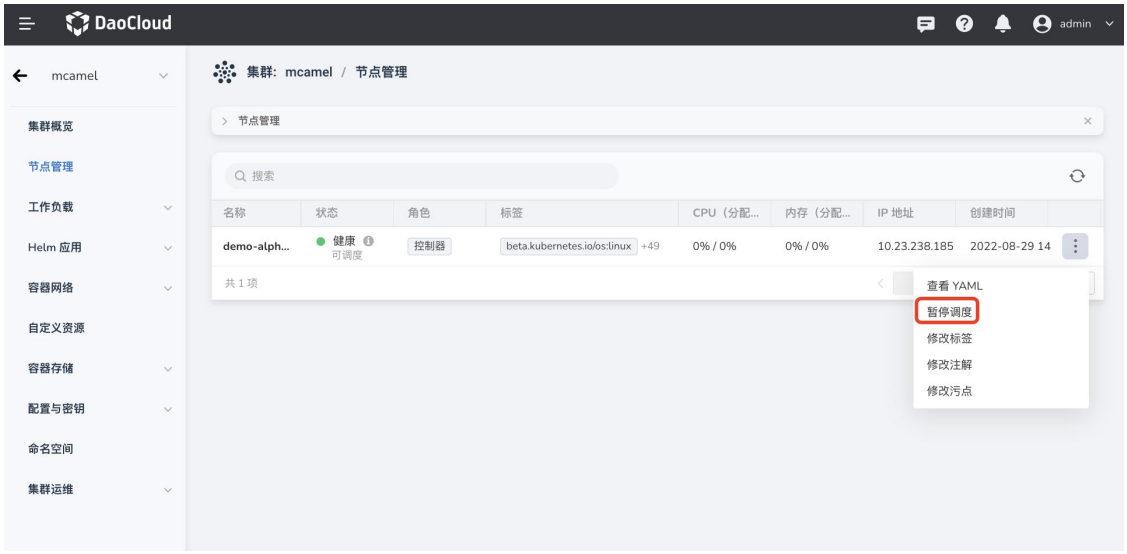
支持将节点暂停调度或恢复调度。暂停调度指，停止将 Pod 调度到该节点。恢复调度指，可以将 Pod 调度到该节点。

1. 在集群列表页面点击目标集群的名称。



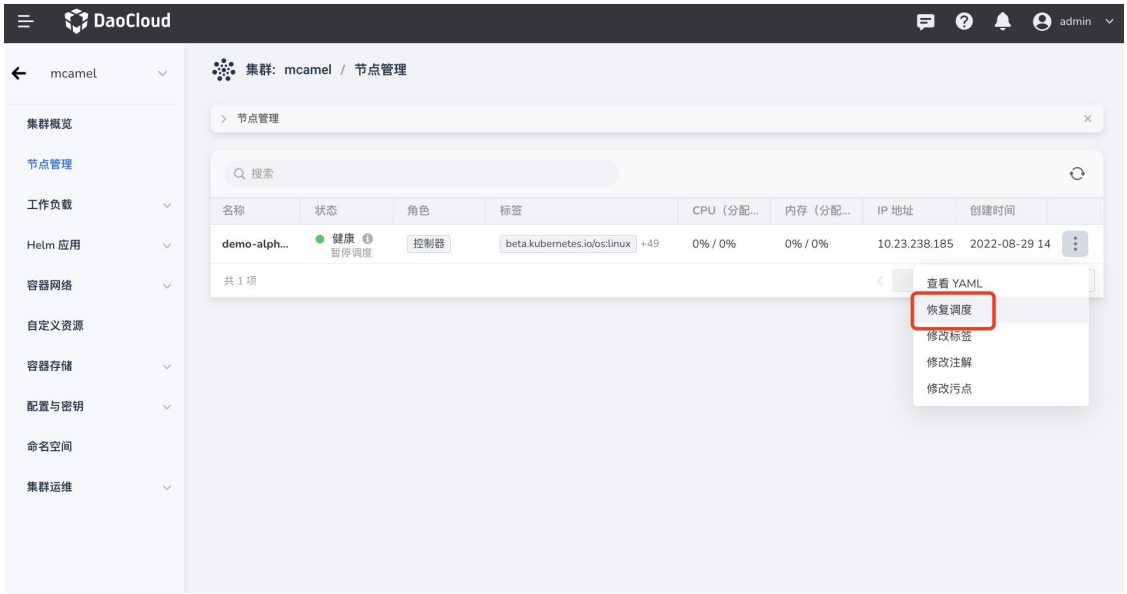
进入集群列表页面

2. 在左侧导航栏点击节点管理，在节点右侧点击　　操作图标，点击暂停调度按钮即可暂停调度该节点。



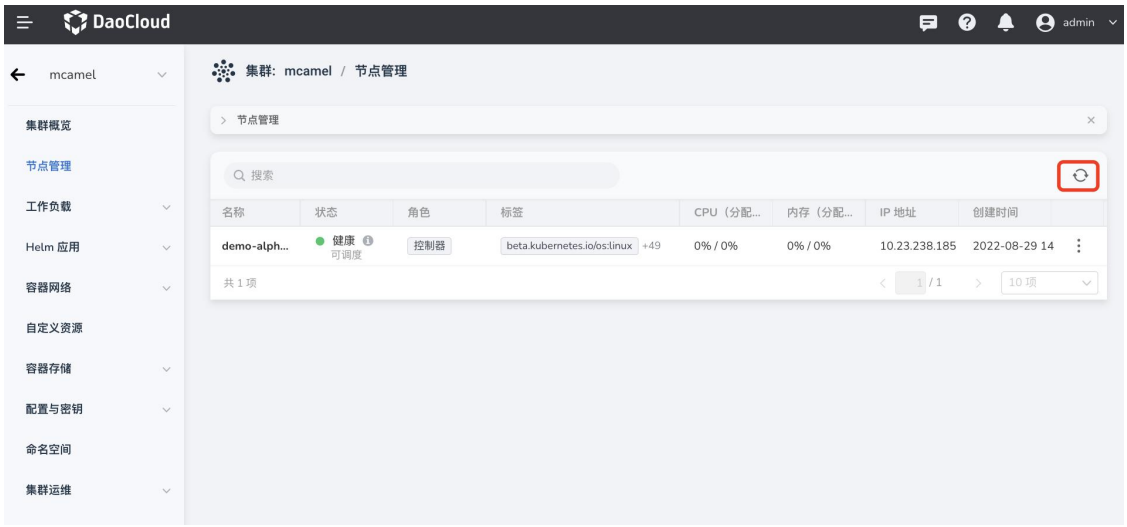
暂停调度

3. 在节点右侧点击　　操作图标，点击恢复调度按钮即可恢复调度该节点。



节点管理

节点调度状态可能因网络情况有所延迟，点击搜索框右侧的刷新图标可以刷新节点调度状态。


DaoCloud
admin
mcamel
集群概览
节点管理
工作负载
Helm 应用
容器网络
自定义资源
容器存储
配置与密钥
命名空间
集群运维
集群: mcamel / 节点管理
节点管理
搜索
名称
状态
角色
标签
CPU（分配...
内存（分配...
IP 地址
创建时间
demo-alph...
健康
可调度
控制器
beta.kubernetes.io/os:linux
+49
0% / 0%
0% / 0%
10.23.238.185
2022-08-29 14
共 1 项
1 / 1
10 项

节点管理

节点污点管理

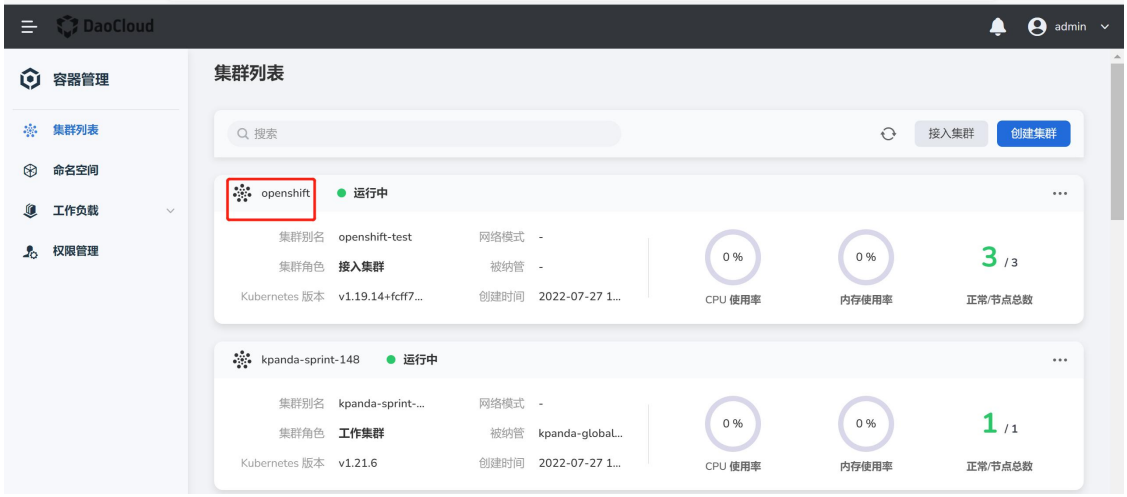
污点（Taint）能够使节点排斥某一类 Pod，避免 Pod 被调度到该节点上。 每个节点上可以应用一个或多个污点，不能容忍这些污点的 Pod 则不会被调度该节点上。 有关污点的更多详情，可参考 Kubernetes 官方文档污点和容忍度。

注意事项

1. 当前操作用户应具备 NS Edit 角色授权或其他更高权限。
2. 为节点添加污点之后，只有能容忍该污点的 Pod 才能被调度到该节点。

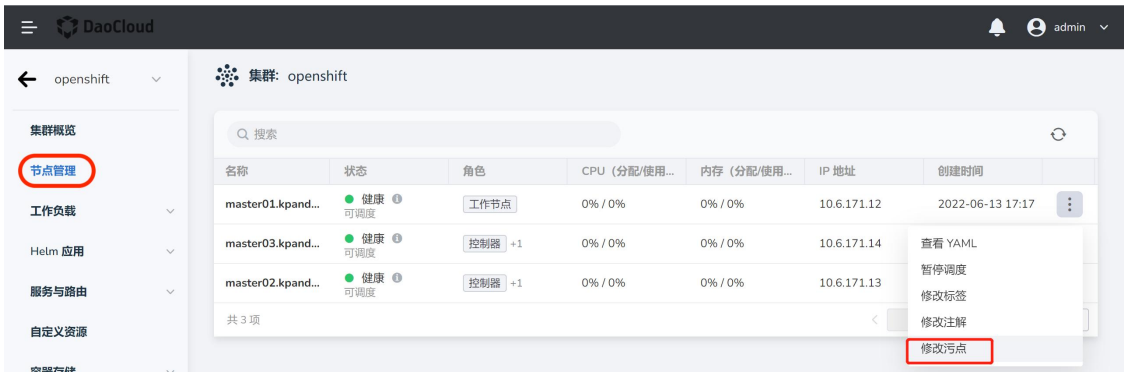
操作步骤

1. 在集群列表页找到目标集群，点击集群名称，进入集群概览页面。



点击集群名称

2. 在左侧导航栏，点击节点管理，找到需要修改污点的节点，点击右侧的操作图标并点击修改污点按钮。



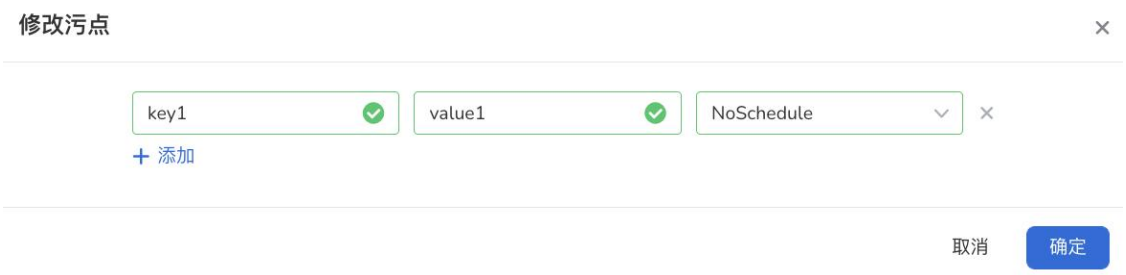
修改污点

3. 在弹框内输入污点的键值信息，选择污点效果，点击确定。

   点击　　添加可以为节点添加多个污点，点击污点效果右侧的 X 可以删除污点。

   目前支持三种污点效果：

   - NoSchedule：不能容忍某个污点的 Pod 不会被调度到存在该污点的节点上。
   - PreferNoSchedule：**尽量避免** 将不能容忍某个污点的 Pod 不会被调度到存在该污点的节点上。
   - NoExecute：保持现状。如果不能容忍某个污点的 Pod 在节点设置污点之前，已经运行在该节点上，也不会驱逐该 Pod。如果不能容忍某个污点的 Pod 在节点设置污点之前，还未运行在该节点上，则不会被调度到该节点。



修改污点

节点认证

使用 SSH 密钥认证节点

如果您选择使用 SSH 密钥作为待创建集群的节点认证方式，您需要按照如下说明配置公私钥。

1. 执行如下命令，在** 待建集群的管理集群中的任意节点 **上生成公私钥。

   ```
   cd /root/.ssh
   ssh-keygen -t rsa
   ```

2. 执行 ls 命令查看管理集群上的密钥是否创建成功，正确反馈如下：

   ```
   ls
   id_rsa  id_rsa.pub  known_hosts
   ```

   其中名为 id_rsa 的文件是私钥，名为 id_rsa.pub 的文件是公钥。

3. 执行如下命令，分别将公钥文件 id_rsa.pub 加载到待创建集群的所有节点上。

   ```
   ssh-copy-id -i /root/.ssh/id_rsa.pub root@10.0.0.0
   ```

   将上面命令中的 root@10.0.0.0 用户账号和节点 IP 替换为待创建集群的节点用户名和 IP。** 需要在待创建集群的每台节点都执行相同的操作 **。

4. 执行如下命令，查看步骤 1 所创建的私钥文件 id_rsa。

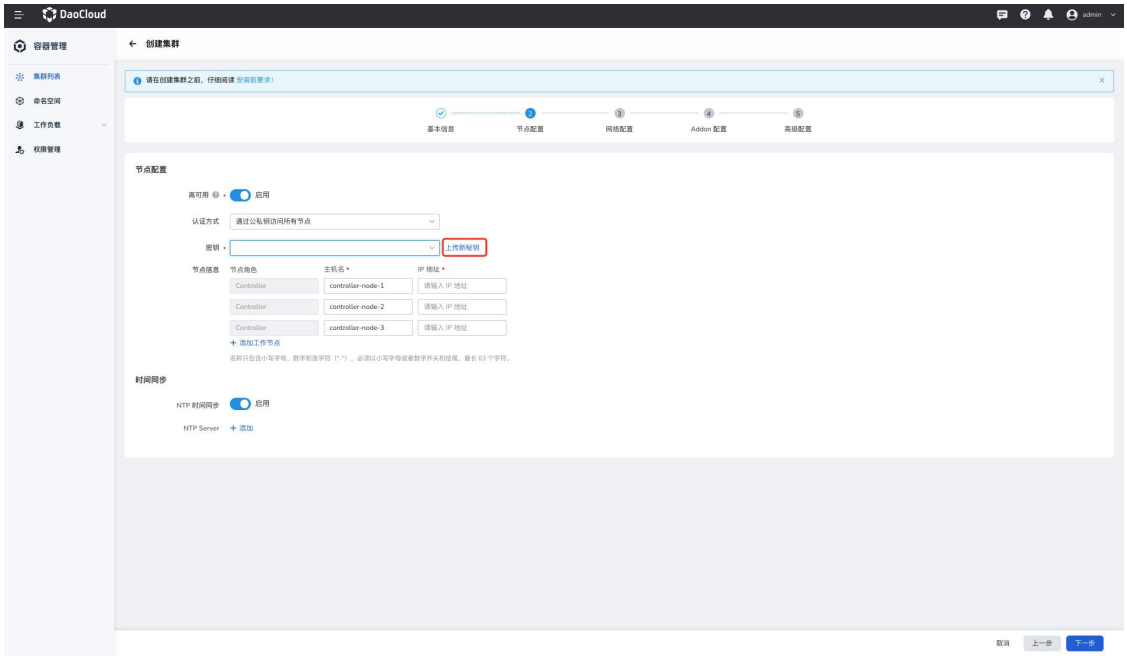
   ```
   cat /root/.ssh/id_rsa
   ```

   输出如下内容：

-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEA3UvyKINzY5BFuemQ+uJ6q+GqgfvnWwNC8HzZhpcMSjJ
y26MM
UtBEBJxy8fMi57XcjYxPibXW/wnd+32ICCycqCwByUmuXeCC1cjICQDqjcAvXa
e7
Y54IXGF7wm2IsMNwf0kjFEXjuS48FLDA0mGRaN3BG+Up5geXcHckg3K5LD8
kXFFx
dEmSIjdyw55NaUitmEdHzN7cIdfi6Z56jcV8dcFBgWKUx+ebiyPmZBkXToz6Gn
MF
rswzzZCl+G6Jb2xTGy7g7ozb4BoZd1IpSD5EhDanRrESVE0C5YuJ5zUAC0CvV
d1I
v67AK8Ko6MXToHp01/bcsvIM6cqgwUFXZKVeOwIDAQABAoIBAQCO36GQlo
3BEjxy
M2HvGJmqrx+unDxafliRe4nVY2AD515Qf4xNSzke4QM1QoyenMOwf446krQkJ
PK0
k+9nI6Xszby5gGCbK4BNFk8I6RaGPjZWeRx6zGUJf8avWJiPxx6yjz2esSC9Ri
R0
F0nmiiefVMyAfgv2/5++dK2WUFNNRKLgSRRpP5bRaD5wMzzxtSSXrUon6217
HO8p
3RoWsI51MbVzhdVgpHUNABcoa0rpr9svT6XLKZxY8mxpKFYjM0Wv2JIDABg3
kBvh
QbJ7kStCO3naZjKMU9UuSqVJs06cfIGYw7Or8/tABR3LErNQKPjkhAQqt0DX
w7Iw
3tKdTAJBAoGBAP687U7JAOqQkcphek2E/A/sbO/d37ix7Z3vNOy065STrA+Z
WMZn
pZ6Ui1B/oJpoZssnfvloz9sn559X0j67TljFALFd2ZGS0Fqh9KVCqDvfk+Vst1dq
+3r/yZdTOyswoccxkJiC/GDwZGK0amJWqvob39JCZhDAKIGLbGMmjdAHAo
GBAN5k
m1WGnni1nZ+3drylwgB6z1hWcnLTamzSET6KhSuo946ET0IRG9xtIheCx6dqIC
br
Vk1Y4NtRZjK/p/YGx59rDWf7E3I8ZMgR7mjieOcUZ4lUlA4l7ZlIW/2WZHW+nU
XO

Ti20fqJ8qSp4BUvOvuth1pz2GLUHe2/Fxjf7HlstAoGBAPHpPr9r+TfllPsJeRj2
6lzA3G8qWFRQfGRYjv0fjv0pA+Rlb1rzgP/l90g5+63G6Z+R4WdcxI/OJJNY1iu
G
uw9n/pFxm7U4JC990BPE6nj5iLz+clpNGYckNDBF9VG9vFSrSDLdaYkxoVNv
G/xJ
a9Na90H4lm7f3VewrPy310KvAoGAZr+mwNoEh5Kpc6xo8Gxi7aPP/mlaUVD6
X7Ki
gvmu02AqmC7rC4QqEiqTaONkaSXwGusqlWxJ3yp5hELmUBYLzszAEeV/s4z
Rp1oZ
g133LBRSTbHFAdBmNdqK6Nu+KGRb92980UMOKvZbliKDl+W6cbfvVu+gtKr
zTc3b
aevb4TUCgYEAnJAxyVYDP1nJf7bjBSHXQu1E/DMwbtrqw7dylRJ8cAzI7lxfSCe
z
7BYWq41PqVd9/zrb3Pbh2phiVzKe783igAlMqummcjo/kZyCwFsYBzK77max1j
F5
aPQsLbRS2aDz8klH6jHPZ/R+15EROmdtLmA7vlJZGerWWQR0dUU+XXA=

将私钥内容复制后填至界面密钥输入框。



SSH 认证

集群节点缩容

当业务高峰期结束之后，为了节省资源成本，可以缩小集群规模，卸载冗余的节点，即节点缩容。节点卸载后，应用无法继续运行在该节点上。

前提条件

- 当前操作用户具有 Cluster Admin 角色授权 。
- 只有通过容器管理模块创建的集群才支持节点扩缩容，从外部接入的集群不支持此操作。
- 卸载节点之前，需要暂停调度该节点，并且将该节点上的应用都驱逐至其他节点。
- 驱逐方式：登录控制器节点，通过 kubectl drain 命令驱逐节点上所有 Pod。安全驱逐的方式可以允许容器组里面的容器优雅地中止。

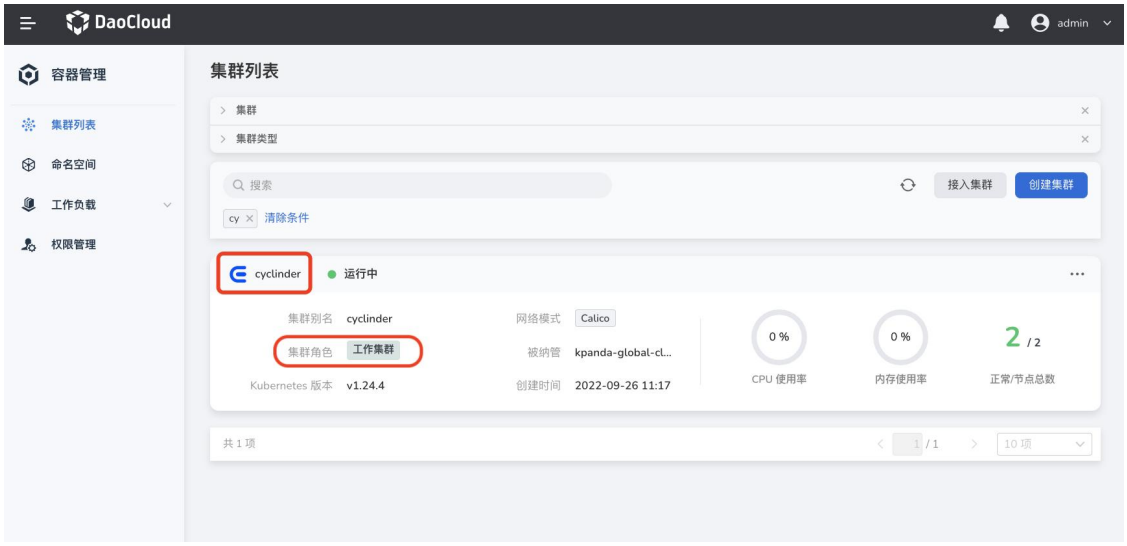
注意事项

1. 集群节点缩容时，只能逐个进行卸载，无法批量卸载。
2. 如需卸载集群控制器节点，需要确保最终控制器节点数为 **奇数**。
3. 集群节点缩容时不可下线 **第一个控制器** 节点。如果必须执行此操作，请联系售后工程师。

操作步骤

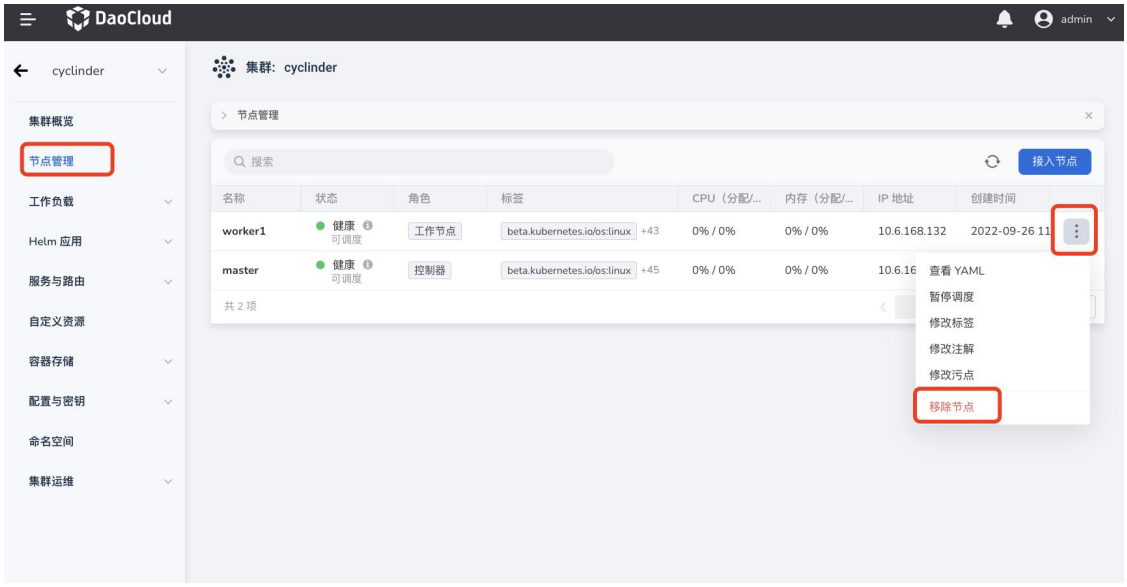
1. 在集群列表页面点击目标集群的名称。

   若集群角色中带有接入集群的标签，则说明该集群不支持节点扩缩容。



进入集群列表页面

2. 在左侧导航栏点击节点管理，找到需要卸载的节点，点击　　选择移除节点。



移除节点

3. 输入节点名称,并点击删除进行确认。


确定移除节点 worker1 吗?
确定移除节点 worker1 吗？移除后对应的数据将会全部丢失且无法恢复，请谨慎操作。
请输入worker1
worker1
删除
取消

移除节点

创建集群节点可用性检查

在创建集群或为已有集群添加节点时，请参阅下表，检查节点配置，以避免因节点配置错误导致集群创建或扩容失败。

检查项	描述
操作系统	参考支持的架构及操作系统
SELinux	关闭
防火墙	关闭
架构一致性	节点间 CPU 架构一致（如均为 ARM 或 x86）
主机时间	所有主机间同步误差小于 10 秒。
网络联通性	节点及其 SSH 端口能够正常被平台访问。
CPU	可用 CPU 资源大于 4 Core
内存	可用内存资源大于 8 GB

支持的架构及操作系统

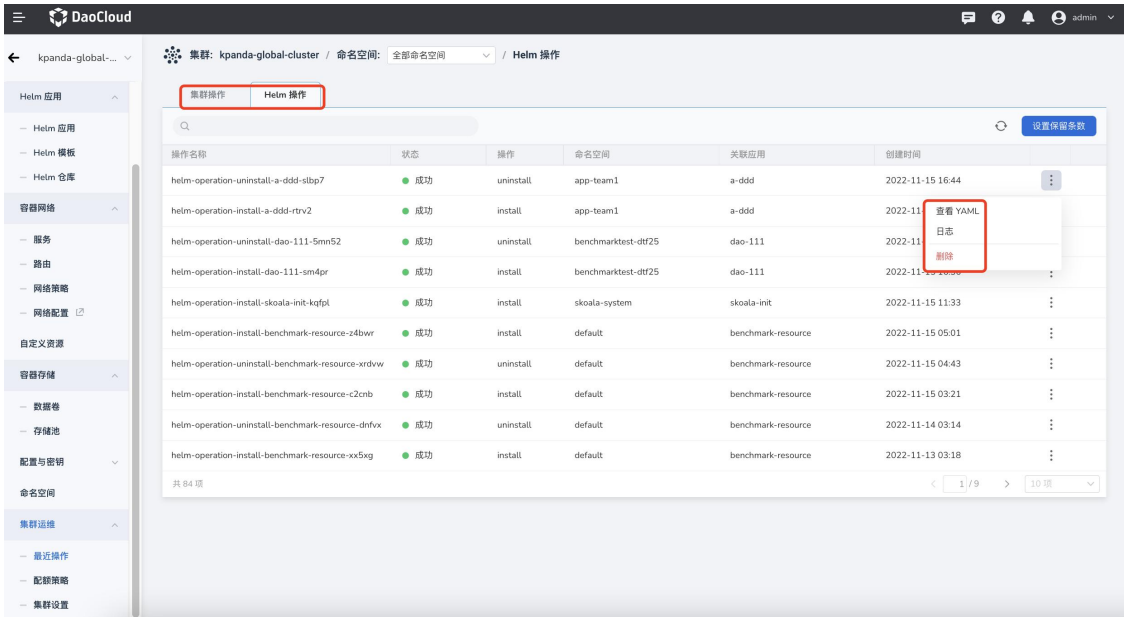
架构	操作系统	备注
ARM	Kylin Linux Advanced Server release V10 (Sword) SP2	推荐
ARM	UOS Linux	
ARM	openEuler	
x86	CentOS 7.x	推荐
x86	Redhat 7.x	推

架构	操作系统	备注
x86	Redhat 8.x	推荐
x86	Flatcar Container Linux by Kinvolk	
x86	Debian Bullseye, Buster, Jessie, Stretch	
x86	Ubuntu 16.04, 18.04, 20.04, 22.04	
x86	Fedora 35, 36	
x86	Fedora CoreOS	
x86	openSUSE Leap 15.x/Tumbleweed	
x86	Oracle Linux 7, 8, 9	
x86	Alma Linux 8, 9	
x86	Rocky Linux 8, 9	
x86	Amazon Linux 2	
x86	Kylin Linux Advanced Server release V10 (Sword) – SP2 海光	

架构	操作系统	备注
x86	UOS Linux 6	
x86	openEuler 6	

最近操作

在该页面可以查看最近的集群操作记录和 Helm 操作记录，以及各项操作的 YAML 文件和日志，也可以删除某一条记录。

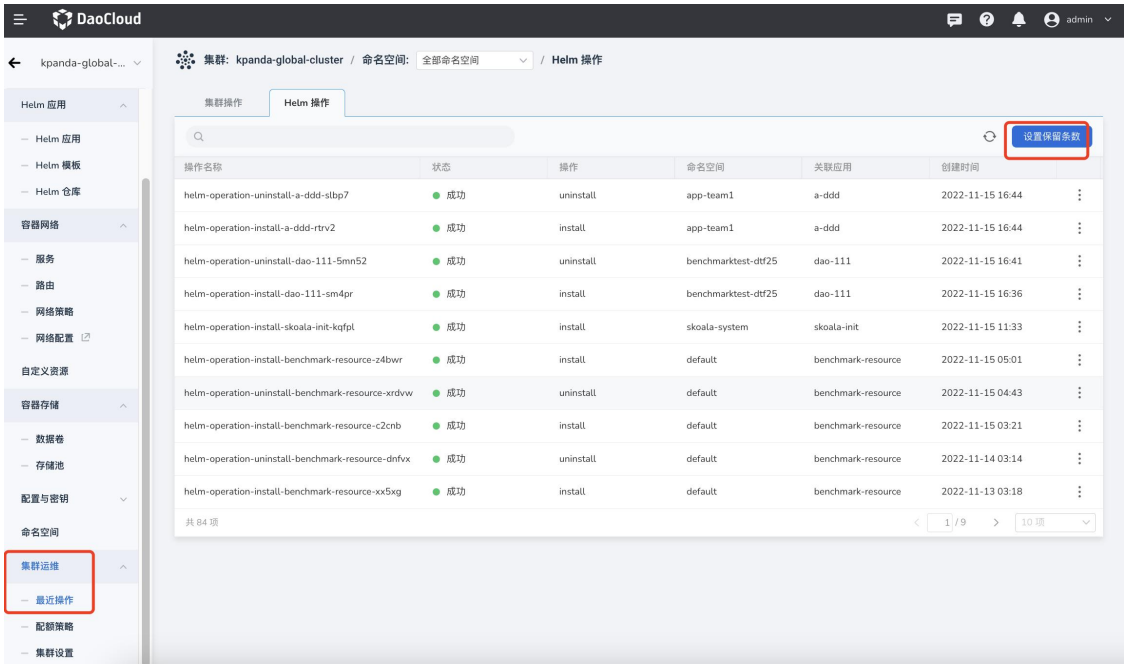


操作记录

设置 Helm 操作的保留条数：

系统默认保留最近 100 条 Helm 操作记录。若保留条数太多，可能会造成数据冗余，保留条数太少可能会造成您所需要的关键操作记录的缺失。需要根据实际情况设置合理的保留数量。具体步骤如下：

1. 点击目标集群的名称，在左侧导航栏点击最近操作–>Helm 操作–>设置保留条数。



保留条数

2. 设置需要保留多少条 Helm 操作记录，并点击确定。



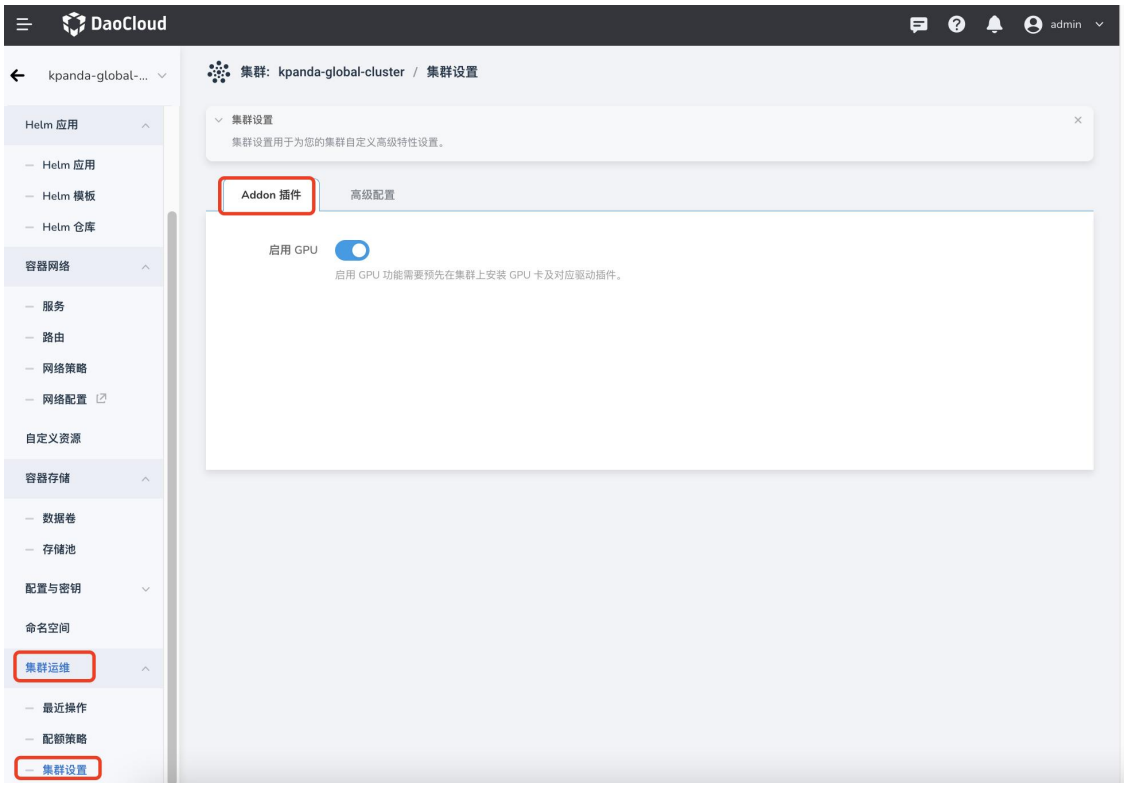
保留条数

集群设置

集群设置用于为您的集群自定义高级特性设置，包括是否启用 GPU、Helm 仓库刷新周期、Helm 操作记录保留等。

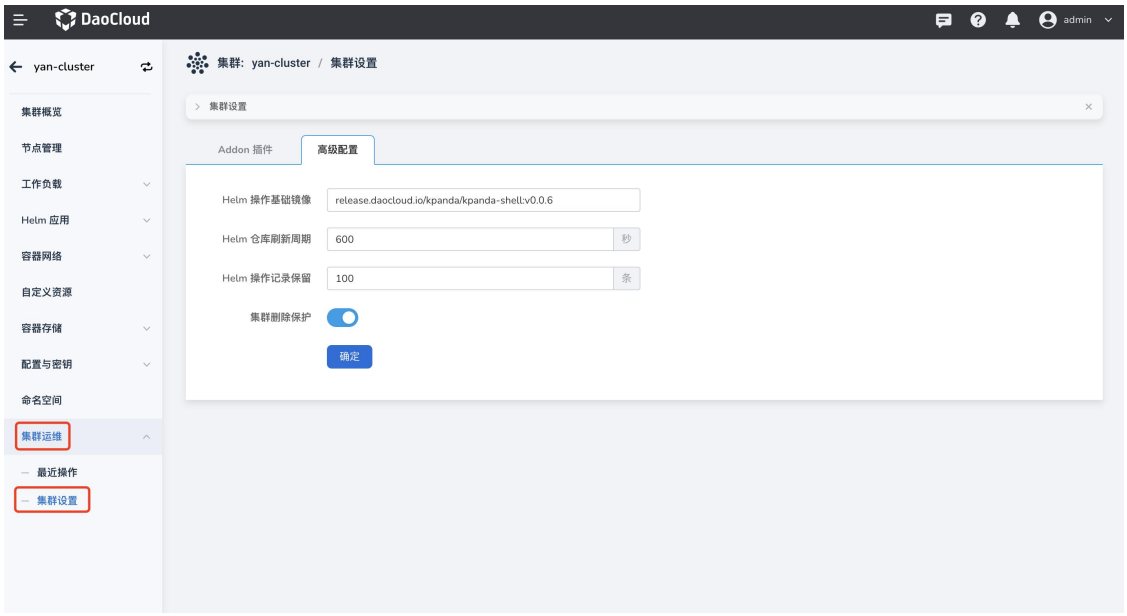
- 启用 GPU：需要预先在集群上安装 GPU 卡及对应驱动插件。

  点击目标集群的名称，在左侧导航栏点击最近操作–>集群设置–>Addon 插件。



配置 gpu

- Helm 操作基础镜像、仓库刷新周期、操作记录保留条数、是否开启集群删除保护（开启后集群将不能直接卸载）


DaoCloud
admin
yan-cluster
集群概览
节点管理
工作负载
Helm 应用
容器网络
自定义资源
容器存储
配置与密钥
命名空间
集群运维
最近操作
集群设置
集群: yan-cluster / 集群设置
集群设置
Addon 插件
高级配置
Helm 操作基础镜像
release.daocloud.io/kpanda/kpanda-shell:v0.0.6
Helm 仓库刷新周期
600
秒
Helm 操作记录保留
100
条
集群删除保护
确定

高级配置

2022-11-17

创建工作集群

在 DCE 5.0 容器管理模块中，集群角色分四类：全局服务集群、管理集群、工作集群、接入集群。其中，接入集群只能从第三方厂商接入，参见接入集群。

本页介绍如何创建工作集群，默认情况下，新建工作集群的工作节点 OS 类型和 CPU 架构需要与全局服务集群保持一致，如需使用区别于全局管理集群 OS 或架构的节点创建集群，参阅在 centos 管理平台上创建 ubuntu 工作集群进行创建。

推荐使用 DCE 5.0 支持的操作系统来创建集群，如您本地节点不在上述支持范围，可参考在非主流操作系统上创建集群进行创建。

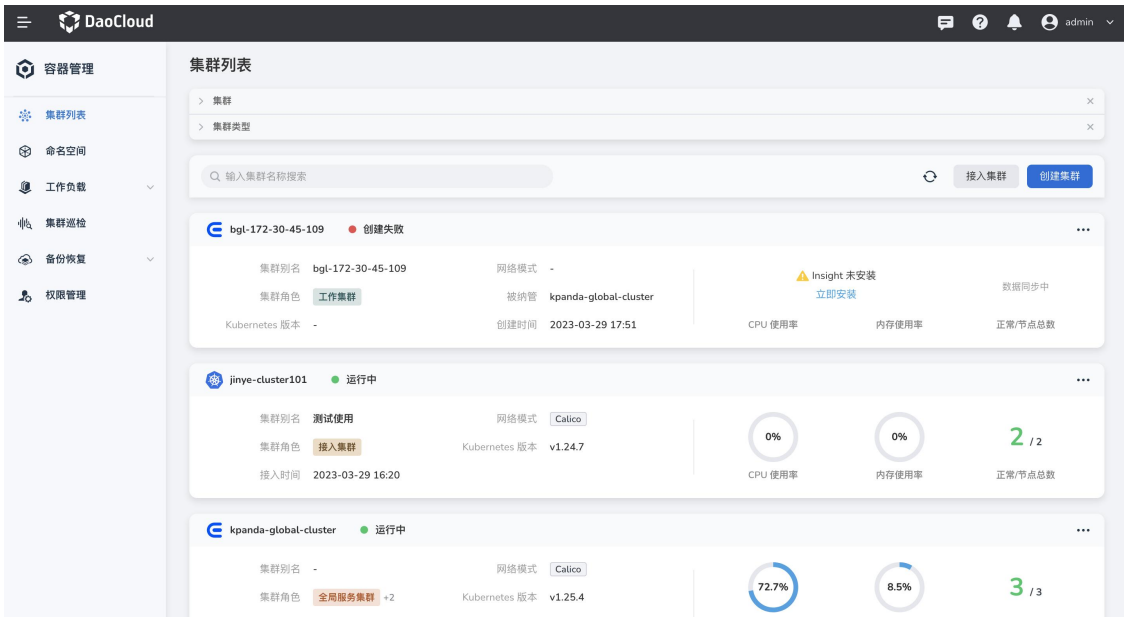
前提条件

创建集群之前需要满足一定的前提条件：

- 根据业务需求准备一定数量的节点，且节点 OS 类型和 CPU 架构一致。
- 推荐 Kubernetes 版本 1.24.7，具体版本范围，参阅 DCE 5.0 集群版本支持体系。
- 目标主机需要允许 IPv4 转发。如果 Pod 和 Service 使用的是 IPv6，则目标服务器需要允许 IPv6 转发。
- DCE 暂不提供对防火墙的管理功能，您需要预先自行定义目标主机防火墙规则。为了避免创建集群的过程中出现问题，建议禁用目标主机的防火墙。
- 参阅节点可用性检查

操作步骤

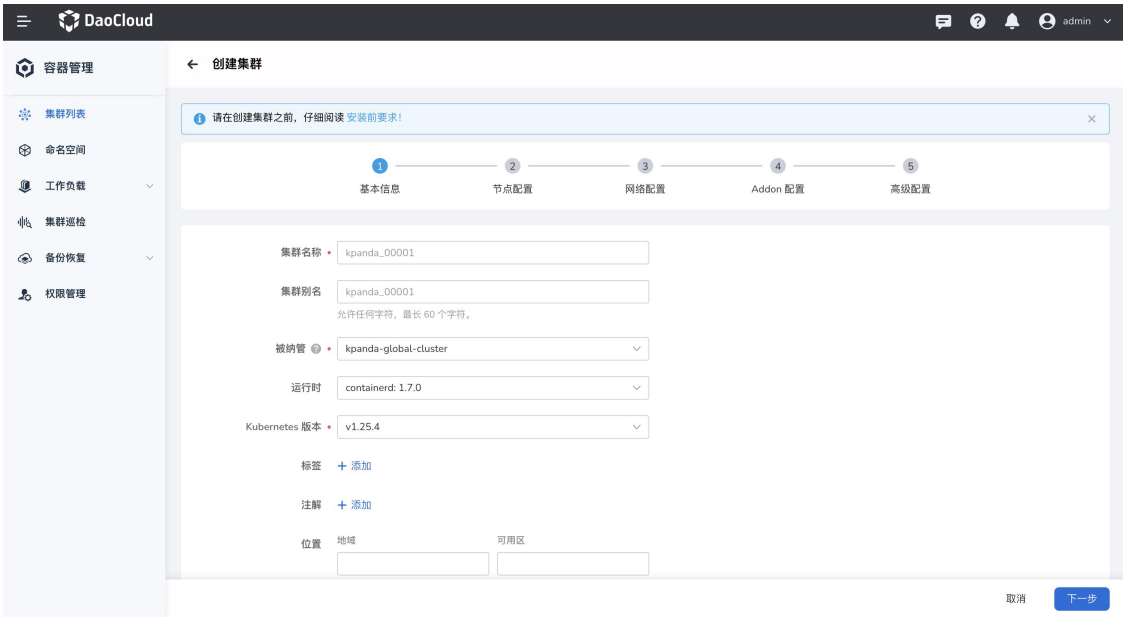
1. 在集群列表页面中，点击创建集群按钮。



创建集群按钮

2. 参考下列要求填写集群基本信息，并点击下一步。

    - 集群名称：名称只包含小写字母、数字和连字符（“-”），必须以小写字母或者数字开头和结尾，最长 63 个字符。
    - 被纳管：选择由哪个集群来管理此集群，例如在集群生命周期中创建、升级、节点扩缩容、删除集群等。
    - 运行时：选择集群的运行时环境，目前支持 containerd 和 docker，如何选择容器运行时。
    - Kubernetes 版本：支持 3 个版本跨度，具体取决于被纳管集群所支持的版本。



填写基本信息

3. 填写节点配置信息，并点击下一步。

    - 高可用：开启后需要提供至少 3 个控制器节点。关闭后，只提供 1 个控制器节点即可。

      生产环境中建议使用高可用模式。

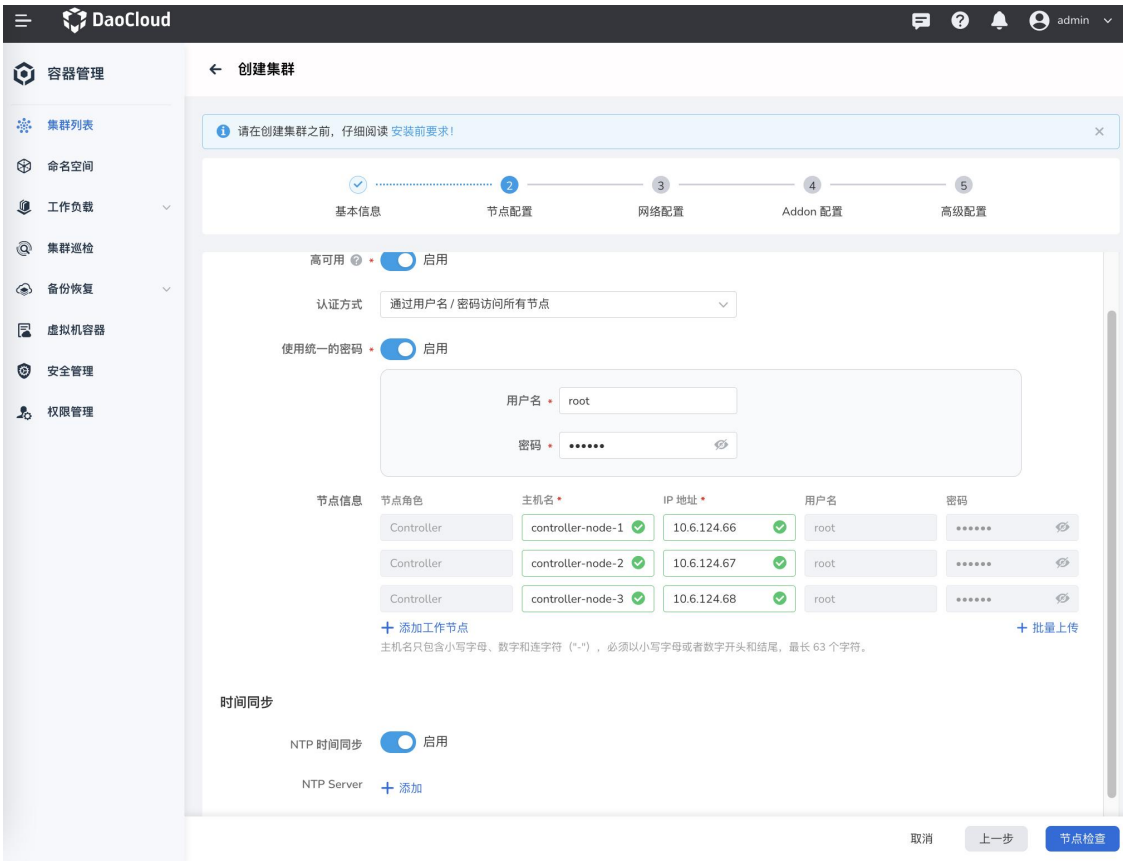
    - 认证方式：选择通过用户名/密码还是公私钥访问节点。

      如果使用公私钥方式访问节点，需要预先配置节点的 SSH 密钥。参阅使用 SSH 密钥认证节点。

    - 使用统一的密码：开启后集群中所有节点的访问密码都相同，需要在下方输入访问所有节点的统一密码。如果关闭，则可以为每个节点设置单独的用户名和密码。

    - 节点信息：填写节点名称和 IP 地址。

    - NTP 时间同步：开启后会自动同步各个节点上的时间，需要提供 NTP 服务器地址。



节点配置

4. 在页面底部点击节点检查。如果检查通过则继续下一步操作。如果检查未通过，则更新节点信息并再次执行检查。

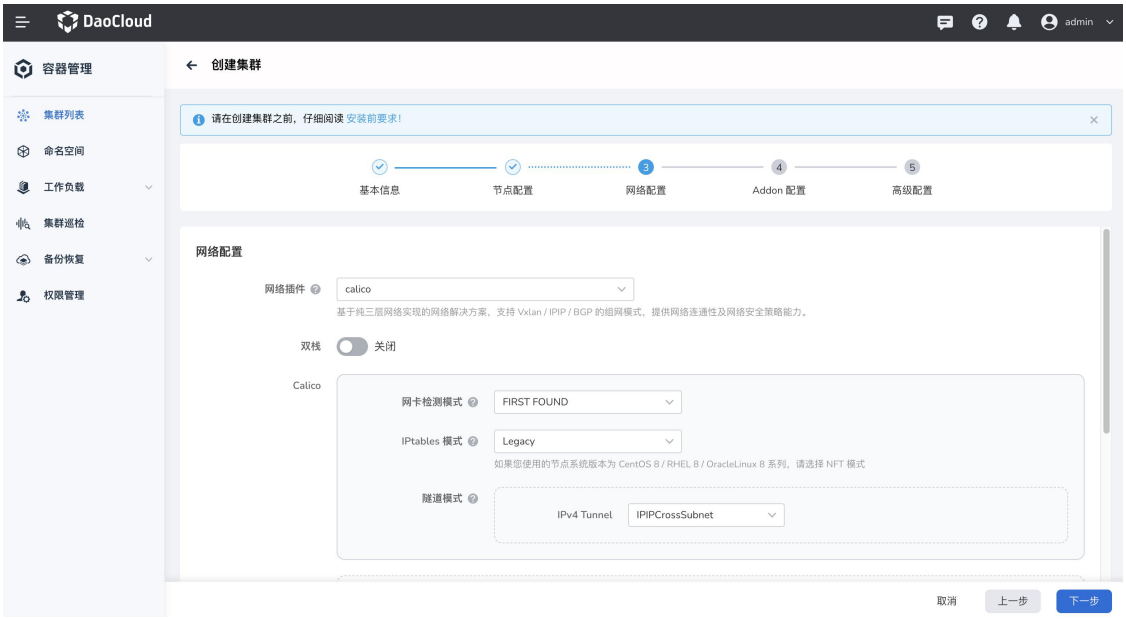
5. 填写网络配置信息，并点击下一步。

   - 网络插件：负责为集群内的 Pod 提供网络服务，**创建集群后不可更改网络插件**。支持 cilium 和 calico。选择 none 表示暂不安装网络插件。

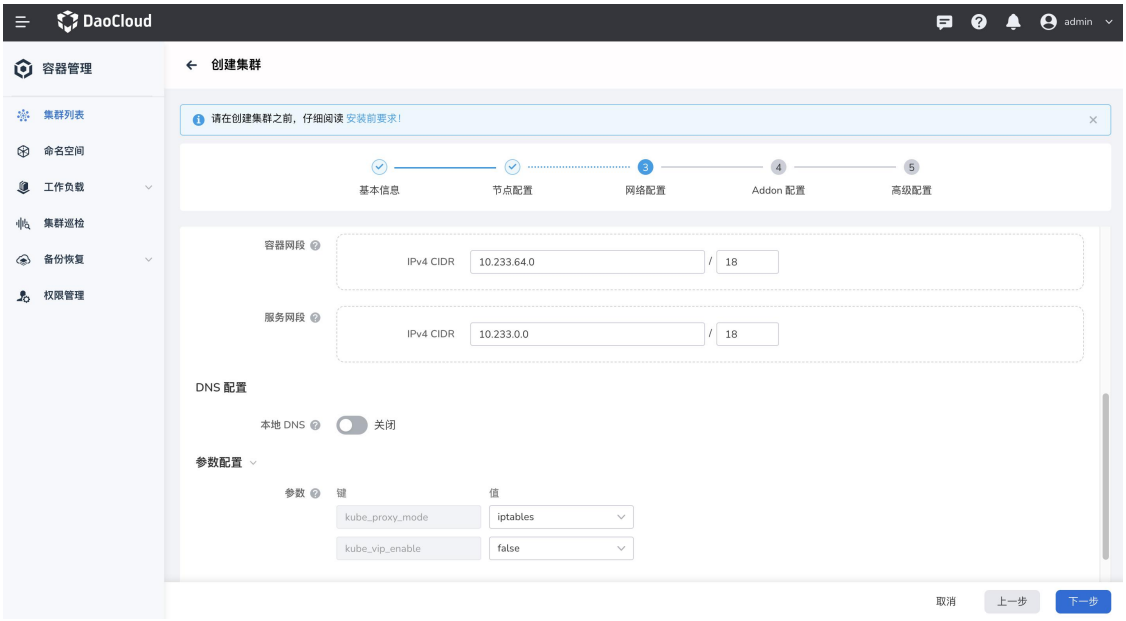
     有关网络插件的参数配置，可参考 cilium 安装参数配置或 calico 安装参数配置。

   - 容器网段：集群下容器使用的网段，决定集群下容器的数量上限。创建后不可修改。

- 服务网段：同一集群下容器互相访问时使用的 Service 资源的网段，决定 Service 资源的上限。创建后不可修改。

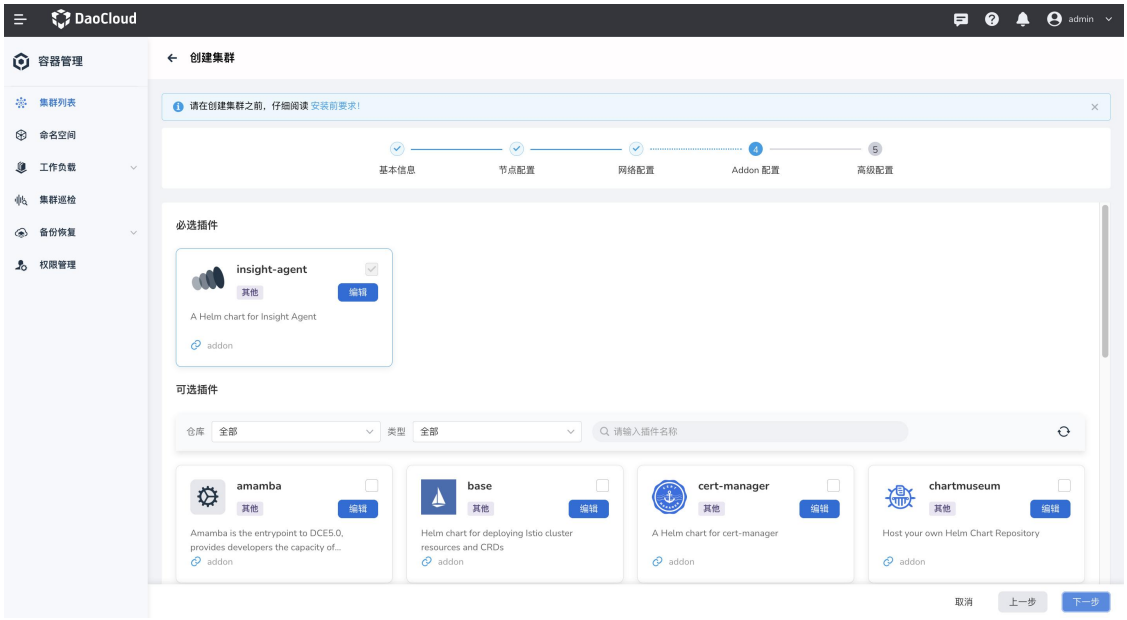


网络配置 1



网络配置 2

6. 填写插件配置信息，并点击下一步。

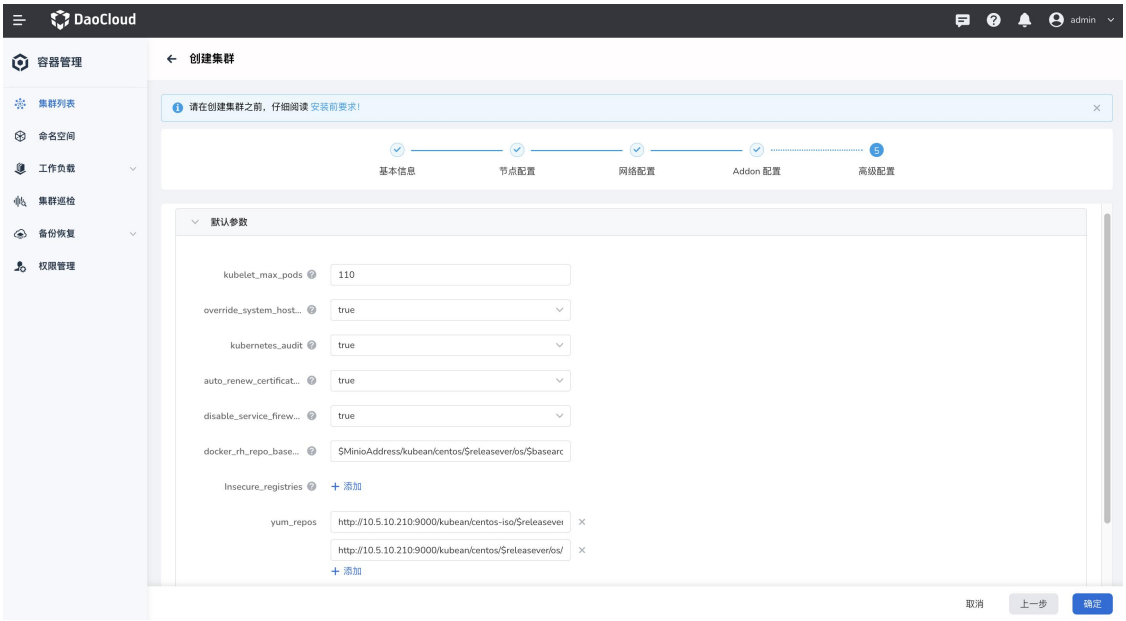


插件配置

7. 填写高级配置信息，并点击确定。

    - kubelet_max_pods：设置每个节点的最大 Pod 数量，默认为 110 个。
    - hostname_overide：重置主机名，建议使用默认值，采用系统默认生成的名称作为主机名称。
    - kubernetes_audit：Kubernetes 的审计日志，默认开启。
    - auto_renew_certificate：在每月第一个星期一自动更新 Kubernetes 控制平面证书，默认开启。
    - disable_firewalld&ufw：禁用防火墙，避免节点在安装过程中无法被访问。
    - Insecure_registries：私有镜像仓库配置。使用私有镜像仓库创建集群时，为了避免证书问题导致容器引擎拒绝访问，需要在这里填写私有镜像仓库地址，以绕过容器引擎的证书认证而获取镜像。

- yum_repos：填写 Yum 源仓库地址。



高级配置

!!! success

- 填写正确信息并完成上述步骤后，页面会提示集群正在创建中。
- 创建集群耗时较长，需要耐心等待。其间，可以点击`返回集群列表`按钮让安装过程后台运行。
- 如需查看当前状态，可点击`实时日志`。

![查看实时日志](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/create009.png)

!!! note

- 当集群出现未知状态时，表示当前集群已失联。
- 系统展示数据为失联前缓存数据，不代表真实数据。
- 同时失联状态下执行的任何操作都将不生效，请检查集群网络连通性或主机状态。

![未知状态](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/createnew07.png)

卸载/解除接入集群

通过 DCE 5.0 容器管理平台 **创建的集群** 支持卸载集群或解除接入操作，从其他环境直接 **接入的集群** 仅支持解除接入操作。

!!! Info

如果想彻底删除一个接入型的集群，需要前往创建该集群的原始平台操作。DCE 5.0 不支持删除接入型的集群。

在 DCE 5.0 平台中，卸载集群和解除接入的区别在于：

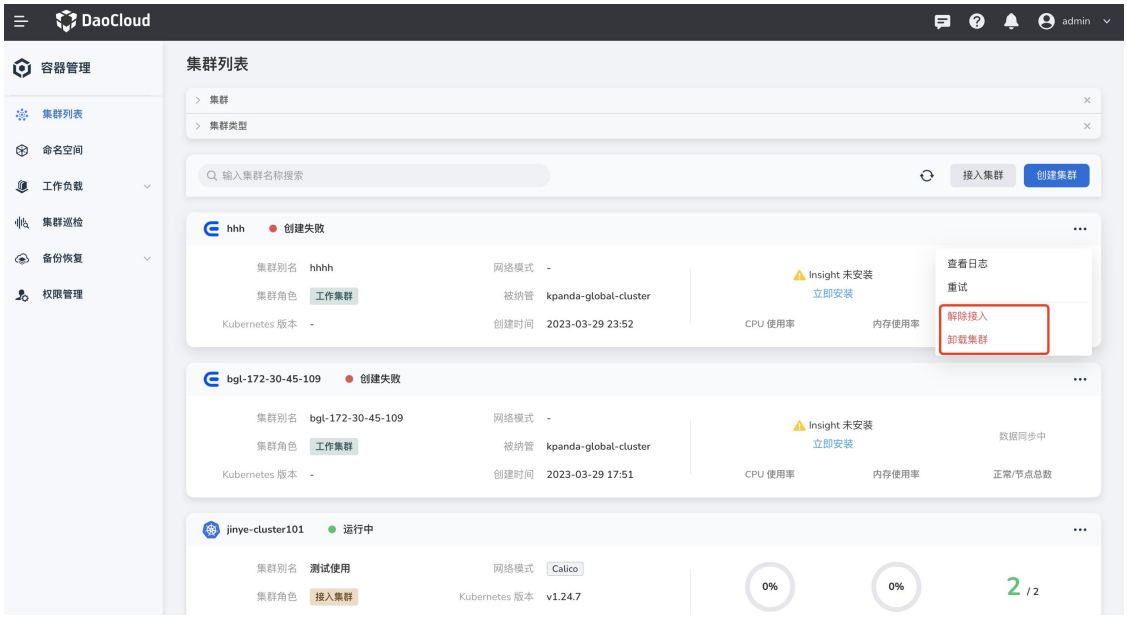
- 卸载集群 操作会销毁该集群，并重置集群下所有节点的数据。所有数据都将被销毁，建议做好备份。后期需要时必须重新创建一个集群。
- 解除接入 操作会将当前集群从平台中移除，不会摧毁集群，也不会销毁数据。

卸载集群

!!! note

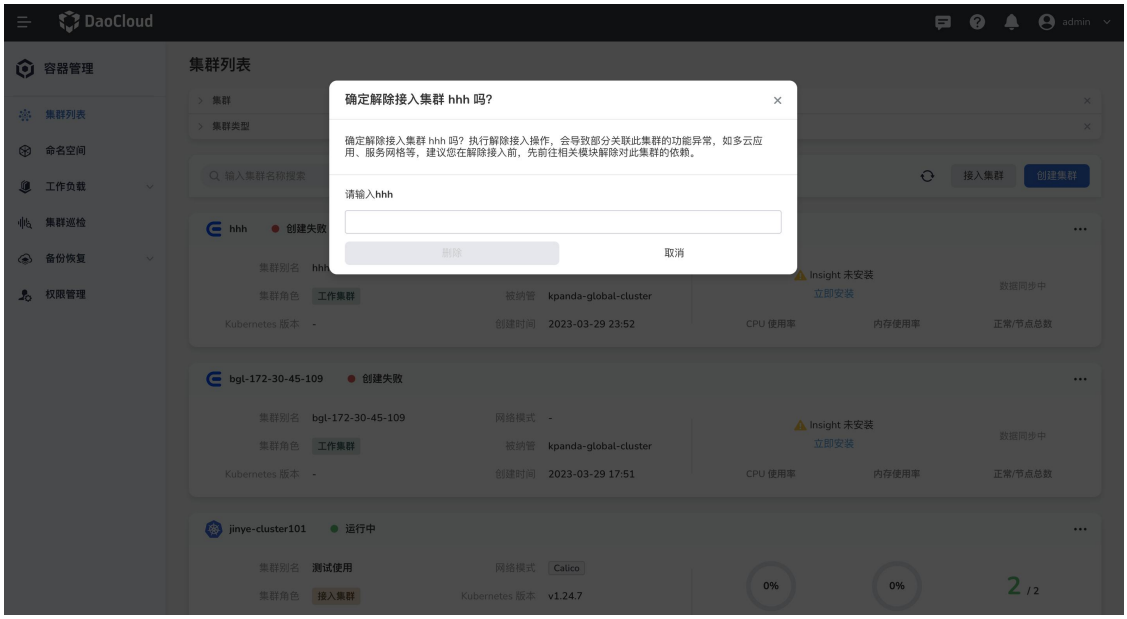
– 当前操作用户应具备 [Admin](../../../ghippo/user-guide/access-control/role.md) 或 [`Kpanda Owner`](../../../ghippo/user-guide/access-control/global.md) 权限才能执行卸载或解除接入的操作。
– 卸载集群之前，应该在`集群设置`->`高级配置`中关闭`集群删除保护`，否则不显示`卸载集群`的选项。
– `全局服务集群`不支持卸载或移除操作。

1. 在集群列表页找到需要卸载集群，点击右侧的 ... 并在下拉列表中点击卸载集群。



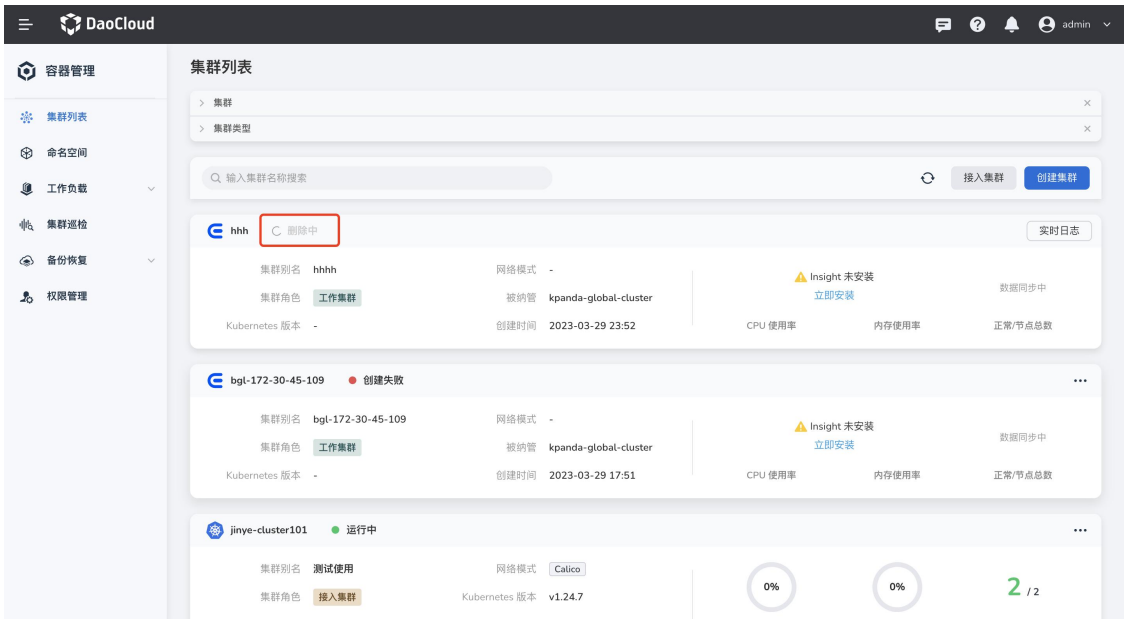
点击删除按钮

2. 输入集群名称进行确认，然后点击删除。



确认删除

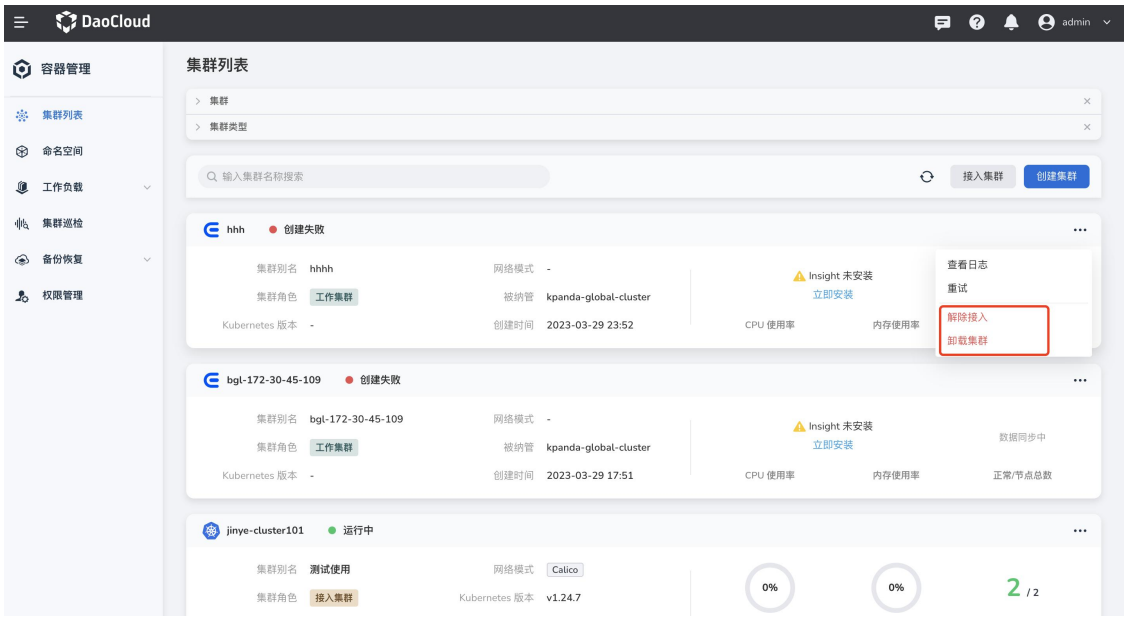
3. 返回集群列表页可以看到该集群的状态已经变成删除中。卸载集群可能需要一段时间，请您耐心等候。



删除中状态

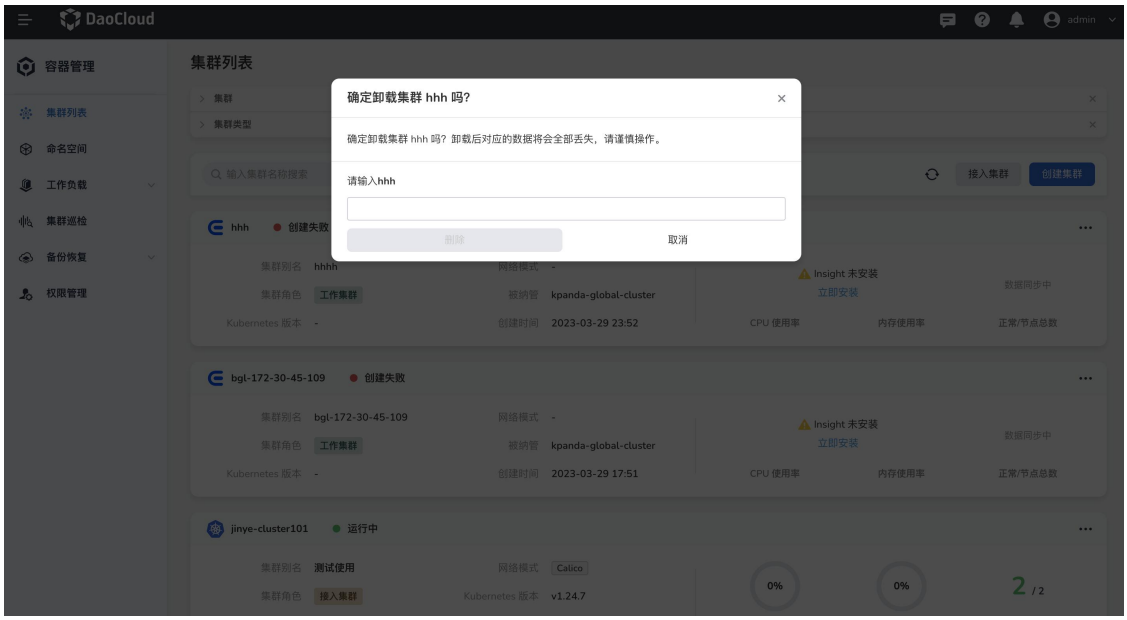
解除接入集群

1. 在集群列表页找到需要卸载集群，点击右侧的 … 并在下拉列表中点击解除接入。



点击删除按钮

2. 输入集群名称进行确认，然后点击删除。



确认删除

清理解除接入集群配置数据

集群被移除后，集群中原有的管理平台数据不会被自动清除，如需将集群接入至新管理平台则需要手动执行如下操作：

1. 删除 kpanda-system、insight-system 命名空间

   ```
   kubectl delete ns kpanda-system insight-system
   ```

2. 如果为当前集群启用了服务网格能力，请参考删除网格文档删除当前集群中的网格实例。

集群状态

容器管理模块支持纳管两种类型的集群：接入集群和自建集群。 关于集群纳管类型的更多信息，请参见集群角色。

这两种集群的状态如下所述。

接入集群

状态	描述
接入中（Joining）	集群正在接入
解除接入中（Removing）	集群正在解除接入
运行中（Running）	集群正常运行
未知（Unknown）	集群已失联，系统展示数据为失联前缓存数据，不代表真实数据，同时失联状态下执行的任何操作都将不生效，请检查集群网络连通性或主机状态。

自建集群

状态	描述
创建中（Creating）	集群正在创建
更新中（Updating）	更新集群 Kubernetes 版本
删除中（Deleting）	集群正在删除
运行中（Running）	集群正常运行
未知（Unknown）	集群已失联，系统展示数据为失联前缓存数据，不代表真实数据，同时失联状态下执行的任何操作都将不生效，请检查集群网络连通性或主机状态。
创建失败（Failed）	集群创建失败，请查看日志以获取详细失败原

状态	描述
	因

2022-11-17

集群升级

Kubernetes 社区每个季度都会发布一次小版本，每个版本的维护周期大概只有 9 个月。版本停止维护后就不会再更新一些重大漏洞或安全漏洞。手动升级集群操作较为繁琐，给管理人员带来了极大的工作负担。

本节将介绍如何在通过 Web UI 界面一键式在线升级工作集群 Kubernetes 版本，如需离线升级工作集群的 kubernetes 版本，请参阅工作集群离线升级指南进行升级。

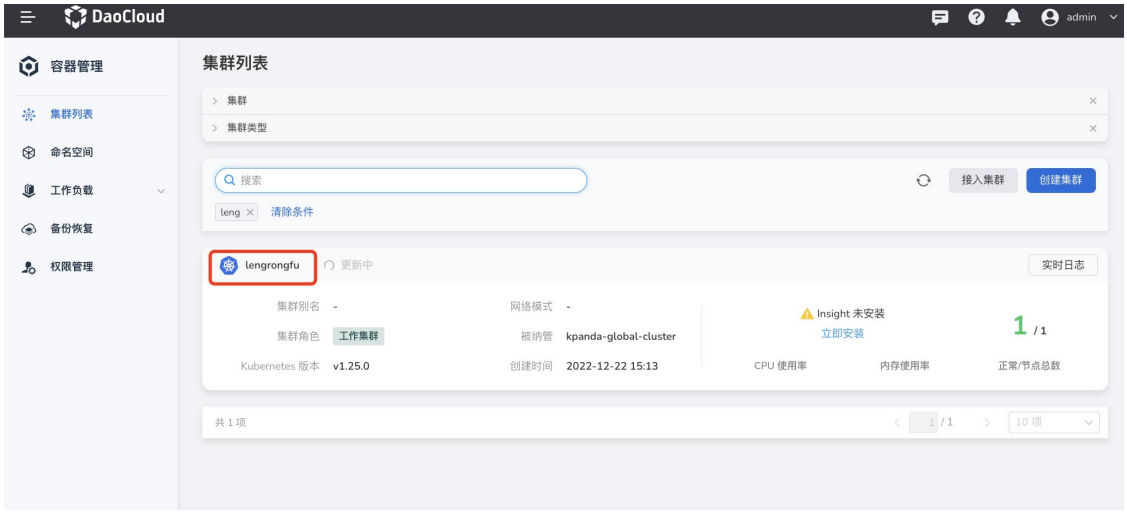
!!! danger

版本升级后将无法回退到之前的版本，请谨慎操作。

!!! note

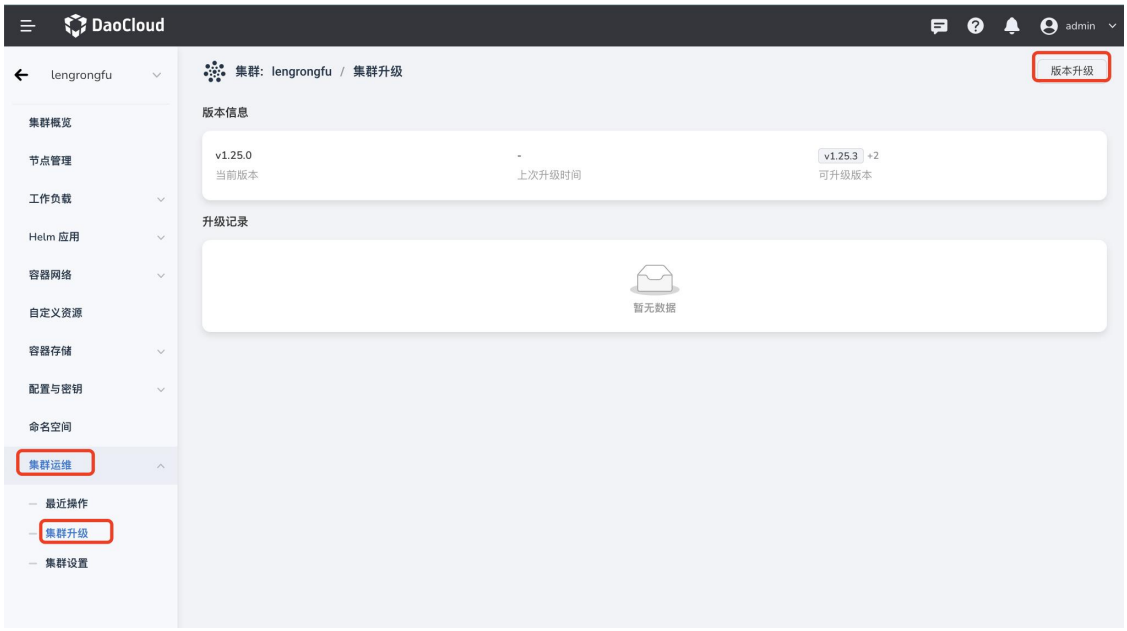
- Kubernetes 版本以 `x.y.z` 表示，其中 `x` 是主要版本，`y` 是次要版本，`z` 是补丁版本。
- 不允许跨次要版本对集群进行升级，例如不能从 1.23 直接升级到 1.25。
- **接入集群不支持版本升级。如果没有左侧导航栏没有`集群升级`，请查看该集群是否为接入集群。**
- 全局服务集群只能通过终端进行升级。
- 升级工作集群时，该工作集群的[管理集群](cluster-role.md#_3)应该已经接入容器管理模块，并且处于正常运行中。

1. 在集群列表中点击目标集群的名称。



升级集群

2. 然后在左侧导航栏点击集群运维–>集群升级，在页面右上角点击版本升级。



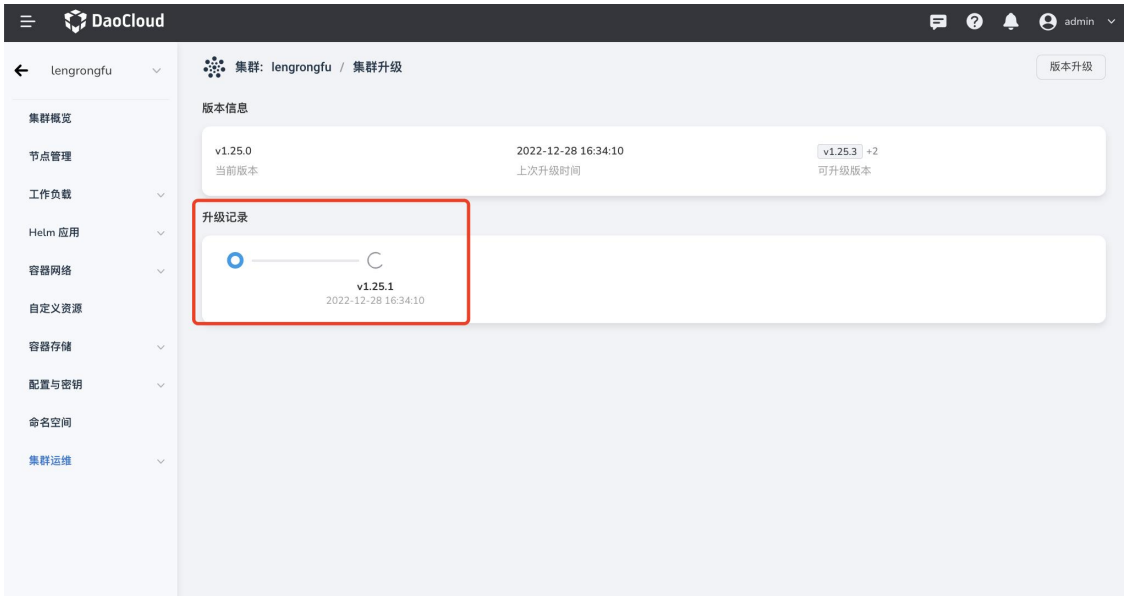
升级集群

3. 选择可升级的版本，输入集群名称进行确认。



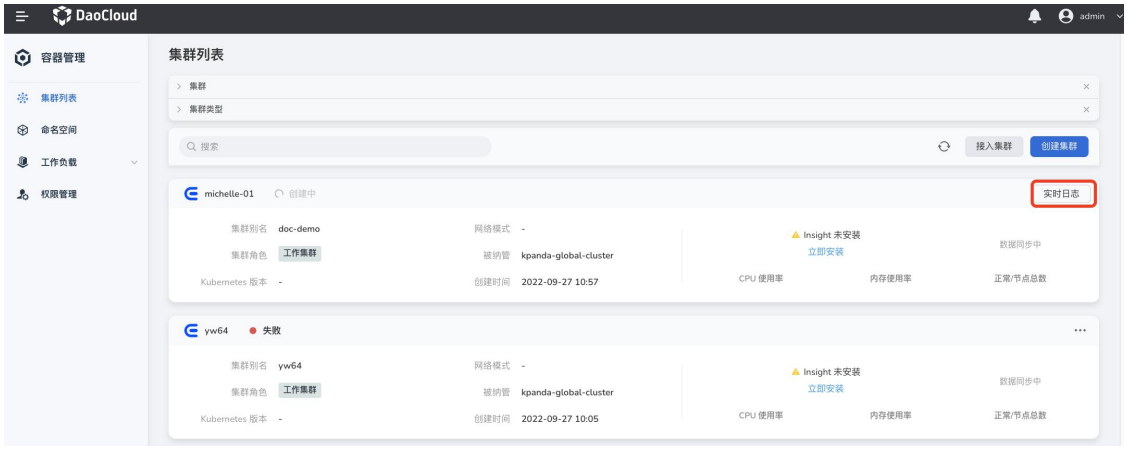
可升级版本

4. 点击确定后，可以看到集群的升级进度。



升级进度

5. 集群升级预计需要 30 分钟，可以点击实时日志按钮查看集群升级的详细日志。


DaoCloud
admin
容器管理
集群列表
命名空间
工作负载
权限管理
集群列表
集群
集群类型
搜索
接入集群
创建集群
michelle-01
创建中
实时日志
集群别名 doc-demo
集群角色 工作集群
Kubernetes 版本 -
网络模式 -
被纳管 kpanda-global-cluster
创建时间 2022-09-27 10:57
Insight 未安装
立即安装
CPU 使用率
内存使用率
数据同步中
正常/节点总数
yw64
失败
集群别名 yw64
集群角色 工作集群
Kubernetes 版本 -
网络模式 -
被纳管 kpanda-global-cluster
创建时间 2022-09-27 10:05
Insight 未安装
立即安装
CPU 使用率
内存使用率
数据同步中
正常/节点总数

实时日志

集群版本支持范围

在 DCE 5.0 平台中，接入型集群和自建集群采取不同的版本支持机制。

本文主要介绍自建集群的版本支持机制。

Kubernetes 社区支持 3 个版本范围，如 1.24、1.25、1.26。当社区新版本发布之后，支持的版本范围将会进行递增。 如社区最新的 1.27 版本已经发布，此时社区支持的版本范围是 1.25、1.26、1.27。

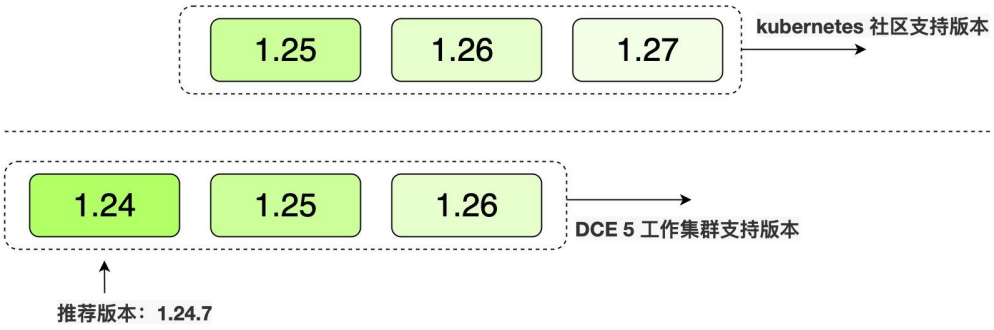
为了保障集群的安全和稳定性，在 DCE 5.0 中使用界面创建集群时支持的版本范围将始终比社区版本 **低一个版本**。

例如，社区支持的版本范围是 1.25、1.26、1.27，则在 DCE 5.0 中使用界面创建工作集群的版本范围是 1.24、1.25、1.26，并且会为用户推荐一个稳定的版本，如 1.24.7。

除此之外， DCE 5.0 中使用界面创建工作集群的版本范围与社区保持高度同步，当社区版本进行递增后，DCE 5.0 中使用界面创建工作集群的版本范围也会同步递增一个版本。

Kubernetes 版本发布说明

Kubernetes 社区版本范围	自建工作集群版本范围	自建工作集群推荐版本	DCE 5.0 版本	发布时间
1.251.261.27	1.241.251.26	**1.26.5**	V0.7.0	2023.05.09


1.25
1.26
1.27
kubernetes 社区支持版本
1.24
1.25
1.26
DCE 5 工作集群支持版本
推荐版本：1.24.7

版本支持机制

访问集群

使用 DCE 5.0 容器管理平台接入或创建的集群，不仅可以通过 UI 界面直接访问，也可以通过其他两种方式进行访问控制：

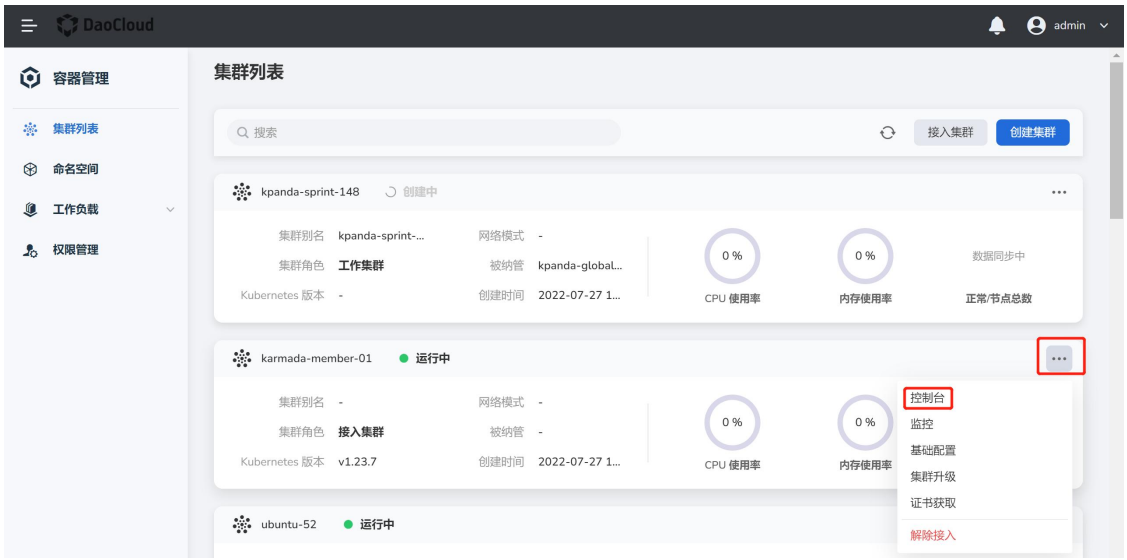
- 通过 CloudShell 在线访问
- 下载集群证书后通过 kubectl 进行访问

!!! note

访问集群时，用户应具有 [`Cluster Admin`](../permissions/permission-brief.md) 权限或更高权限。

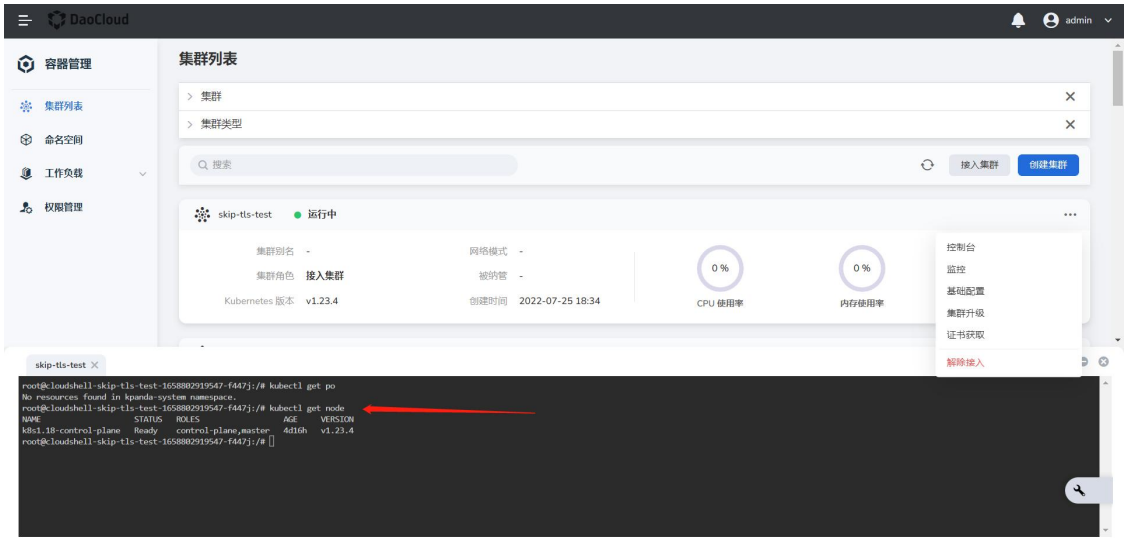
通过 CloudShell 访问

1. 在集群列表页选择需要通过 CloudShell 访问的集群，点击右侧的 ... 操作图标并在下拉列表中点击控制台。



调用 CloudShell 控制台

2. 在 CloudShell 控制台执行 kubectl get node 命令，验证 CloudShell 与集群的连通性。如图，控制台将返回集群下的节点信息。



验证连通性

现在，您可以通过 CloudShell 来访问并管理该集群了。

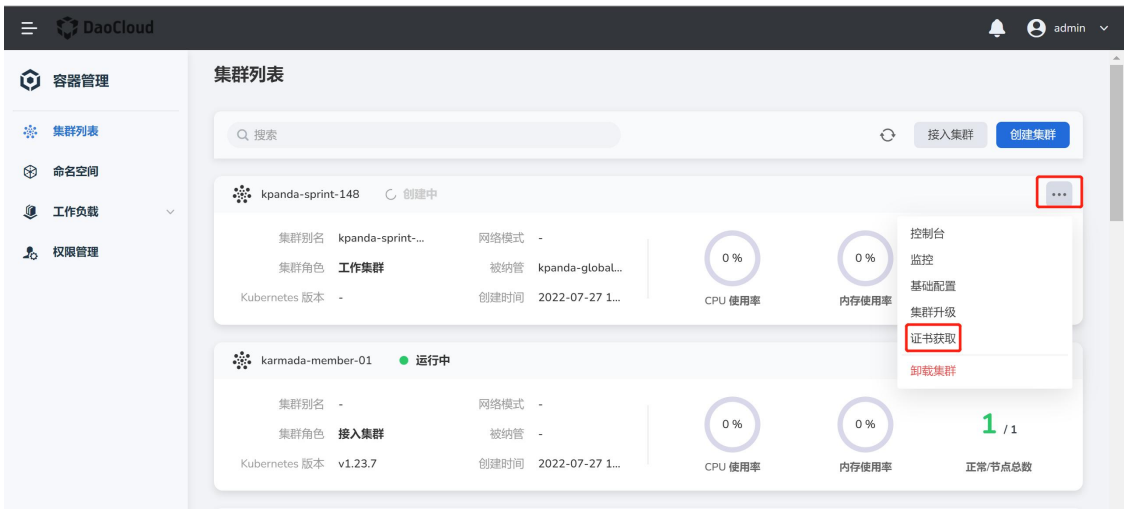
通过 kubectl 访问

通过本地节点访问并管理云端集群时，需要满足以下条件：

- 本地节点和云端集群的网络互联互通。
- 已经将集群证书下载到了本地节点。
- 本地节点已经安装了 kubectl 工具。关于详细的安装方式，请参阅安装 kubectl。

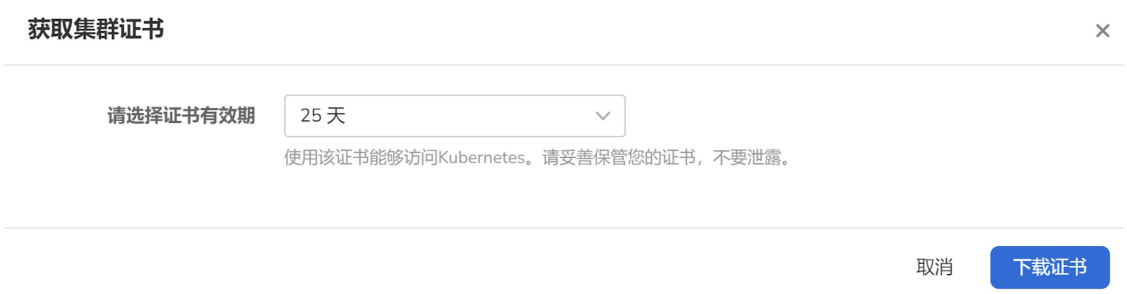
满足上述条件后，按照下方步骤从本地访问云端集群：

1. 在集群列表页选择需要下载证书的集群，点击右侧的 ...，并在弹出菜单中点击证书获取。



进入 下载证书页面

2. 选择证书有效期并点击下载证书。



下载证书

3. 打开下载好的集群证书，将证书内容复制至本地节点的 config 文件。

   kubectl 工具默认会从本地节点的 $HOME/.kube 目录下查找名为 config 的文件。该文件存储了相关集群的访问凭证，kubectl 可以凭该配置文件连接至集群。

4. 在本地节点上执行如下命令验证集群的连通性：

   ```
   kubectl get pod –n default
   ```

   预期的输出类似于:

```
NAME                                READY   STATUS    RESTARTS   AGE
dao-2048-2048-58c7f7fc5-mq7h4       1/1     Running   0          30h
```

现在您可以在本地通过 kubectl 访问并管理该集群了。

集群角色

DaoCloud Enterprise 5.0 基于集群的不同功能定位对集群进行了角色分类，帮助用户更好地管理 IT 基础设施。

全局服务集群

此集群用于运行 DCE 5.0 组件，例如容器管理、全局管理、可观测性、镜像仓库等。 一般不承载业务负载。

支持项	描述
K8s 版本	1.22+
操作系统	RedHat 7.6 x86/ARM, RedHat 7.9 x86, RedHat 8.4 x86/ARM, RedHat 8.6 x86;Ubuntu 18.04 x86, Ubuntu 20.04 x86;CentOS 7.6 x86/AMD, CentOS 7.9 x86/AMD
集群全生命周期管理	支持
K8s 资源管理	支持
云原生存储	支持
云原生网络	Calico、Cillium、Multus 和其它 CNI
策略管理	支持网络策略、配额策略、资源限制、灾备策略、安全策略

管理集群

此集群用于管理工作集群，一般不承载业务负载。

- 经典模式将全局服务集群和管理集群部署在不同的集群，适用于企业多数据中心、多架构的场景。
- 简约模式将管理集群和全局服务集群部署在同一个集群内。

支持的功能	描述
K8s 版本	1.22+
操作系统	RedHat 7.6 x86/ARM, RedHat 7.9 x86, RedHat 8.4 x86/ARM, RedHat 8.6 x86;Ubuntu 18.04 x86, Ubuntu 20.04 x86;CentOS 7.6 x86/AMD, CentOS 7.9 x86/AMD
集群全生命周期管理	支持
K8s 资源管理	支持
云原生存储	支持
云原生网络	Calico、Cillium、Multus 和其它 CNI
策略管理	支持网络策略、配额策略、资源限制、灾备策略、安全策略

工作集群

这是使用容器管理创建的集群，主要用于承载业务负载。该集群由管理集群进行管理。

支持的功能	描述
K8s 版本	支持 K8s 1.22 及以上版本
操作系统	RedHat 7.6 x86/ARM, RedHat 7.9 x86, RedHat 8.4 x86/ARM, RedHat 8.6 x86;Ubuntu 18.04 x86, Ubuntu 20.04 x86;CentOS 7.6 x86/AMD, CentOS 7.9 x86/AMD
集群全生命周期管理	支持
K8s 资源管理	支持
云原生存储	支持
云原生网络	Calico、Cillium、Multus 和其它 CNI
策略管理	支持网络策略、配额策略、资源限制、灾备策略、安全策略

接入集群

此集群用于接入已有的标准 K8s 集群，包括但不限于本地数据中心自建集群、公有云厂商提供的集群、私有云厂商提供的集群、边缘集群、信创集群、异构集群、Daocloud 不同发行版集群。 主要用于承担业务负载。

支持的功能	描述
K8s 版本	1.18+
支持友商	Vmware Tanzu、Amazon EKS、Redhat Openshift、SUSE Rancher、阿里 ACK、华为 CCE、腾讯 TKE、标准 K8s 集群、Daocloud DCE
集群全生命周期管理	不支持
K8s 资源管理	支持
云原生存储	支持
云原生网络	依赖于接入集群发行版网络模式
策略管理	支持网络策略、配额策略、资源限制、灾备策略、安全策略

!!! note

一个集群可以有多重集群角色，例如一个集群既可以是全局服务集群，也可以是管理集群或工作集群。

如何选择容器运行时

容器运行时是 kubernetes 中对容器和容器镜像生命周期进行管理的重要组件。kubernetes 在 1.19 版本中将 Containerd 设为默认的容器运行时，并在 1.24 版本中移除了 Dockershim 组件的支持。

因此相较于 Docker 运行时，我们更加 **推荐您使用轻量的 Containerd 作为您的容器运行时**，因为这已经成为当前主流的运行时选择。

除此之外，一些操作系统发行厂商对 Docker 运行时的兼容也不够友好，不同操作系统对运行时的支持如下表：

不同操作系统和支持的运行时对应关系

操作系统	支持的 Continerd 运行时版本	支持的 Docker 运行时版本
CentOS	1.5.5、1.5.7、1.5.8、1.5.9、1.5.10、1.5.11、1.5.12、1.5.13、1.6.0、1.6.1、1.6.2、1.6.3、1.6.4、1.6.5、1.6.6、1.6.7、1.6.8、1.6.9、1.6.10、1.6.11、1.6.12、1.6.13、1.6.14、1.6.15（默认）	18.09、19.03、20.10（默认）
RedHatOS	1.5.5、1.5.7、1.5.8、1.5.9、1.5.10、1.5.11、1.5.12、1.5.13、1.6.0、1.6.1、1.6.2、1.6.3、1.6.4、1.6.5、1.6.6、1.6.7、1.6.8、1.6.9、1.6.10、1.6.11、1.6.12、1.6.13、1.6.14、1.6.15（默认）	18.09、19.03、20.10（默认）

操作系统	支持的 Continerd 运行时版本	支持的 Docker 运行时版本
KylinOS	1.5.5、1.5.7、1.5.8、1.5.9、1.5.10、1.5.11、1.5.12、1.5.13、1.6.0、1.6.1、1.6.2、1.6.3、1.6.4、1.6.5、1.6.6、1.6.7、1.6.8、1.6.9、1.6.10、1.6.11、1.6.12、1.6.13、1.6.14、1.6.15 （默认）	19.03（仅 ARM 架构支持 ，在 X86 架构下不支持使用 Docker 作为运行时）

!!! note

在离线安装模式下，需要提前准备相关操作系统的运行时离线包。

接入集群

通过接入集群操作，能够对众多云服务平台集群和本地私有物理集群进行统一纳管，形成统一治理平台，有效避免了被厂商锁定风险，助力企业业务安全上云。

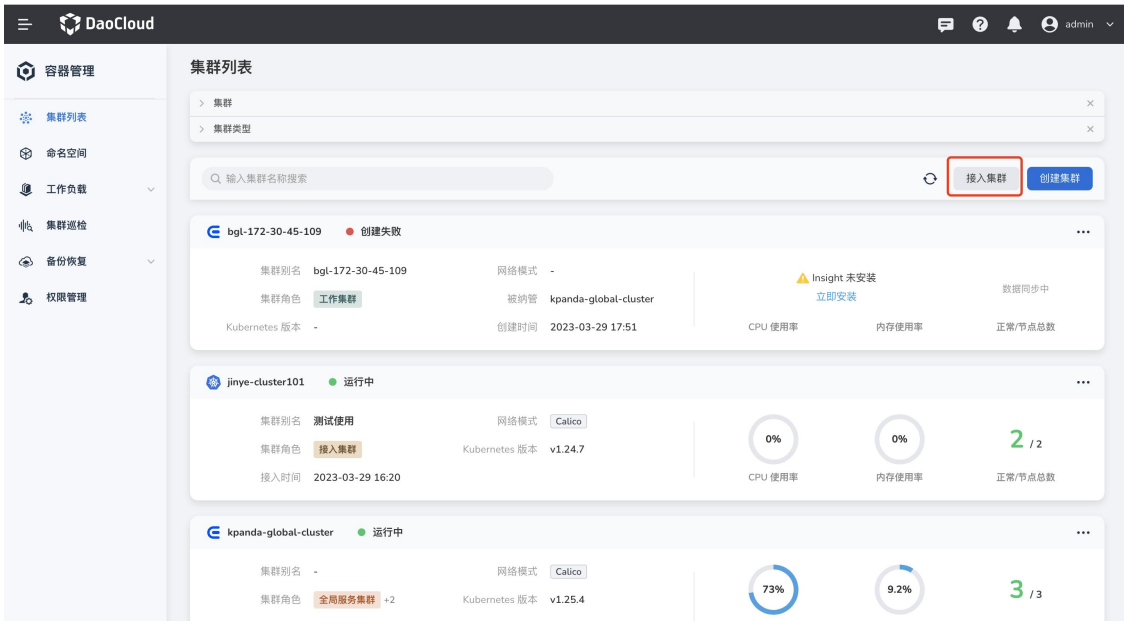
容器管理模块支持接入多种主流的容器集群，例如 DaoCloud KubeSpray, DaoCloud ClusterAPI, DaoCloud Enterprise 4.0、Redhat Openshift, SUSE Rancher, VMware Tanzu, Amazon EKS, Aliyun ACK, Huawei CCE, Tencent TKE,标准 Kubernetes 集群。

前提条件

- 准备一个待接入的集群，确保容器管理集群和待接入集群之间网络通畅，并且集群的 Kubernetes 版本 1.22+。
- 当前操作用户应具有 kpanda owner 或更高权限。

操作步骤

1. 进入集群列表页面，点击右上角的接入集群按钮。



接入集群

2. 填写基本信息。

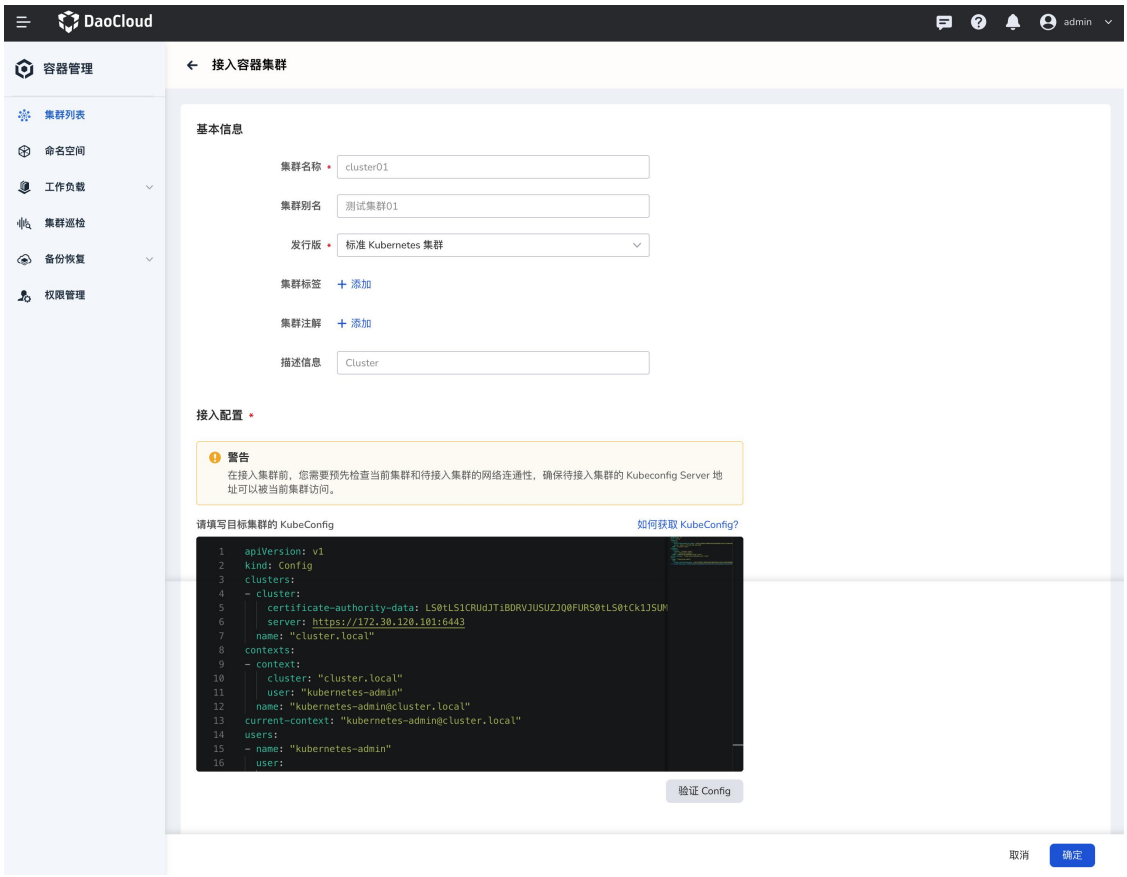
   - 集群名称：名称应具有唯一性，设置后不可更改。最长 63 个字符，只能包含小写字母、数字及分隔符(“-”)，且必须以小写字母或数字开头及结尾。
   - 集群别名：可输入任意字符，不超过 60 个字符。
   - 发行版：集群的发行厂商，包括市场主流云厂商和本地私有物理集群。

3. 填写目标集群的 KubeConfig，点击验证 Config，验证通过后才能成功接入集群。

   如果不知道如何获取集群的 KubeConfig 文件，可以在输入框右上角点击如何获取 kubeConfig 查看对应步骤。



4. 确认所有参数填写正确，在页面右下角点击确定。



接入集群

!!! note

– 新接入的集群状态为`接入中`，接入成功后变为`运行中`。

– 如果集群状态一直处于`接入中`，请确认接入脚本是否在对应集群上执行成功。有关集群状态的更多详情，请参考[集群状态](cluster-status.md)。

容器管理权限说明

容器管理权限基于全局权限管理以及 Kubernetes RBAC 权限管理打造的多维度权限管理体系。 支持集群级、命名空间级的权限控制，帮助用户便捷灵活地对租户下的 IAM 用户、用户组（用户的集合）设定不同的操作权限。

集群权限

集群权限基于 Kubernetes RBAC 的 ClusterRolebinding 授权，集群权限设置可让用户/用户组具备集群相关权限。 目前的默认集群角色为 Cluster Admin（不具备集群的创建、删除权限）。

Cluster Admin

Cluster Admin 具有以下权限：

1. 可管理、编辑、查看对应集群
2. 管理、编辑、查看 命名空间下的所有工作负载及集群内所有资源
3. 可授权用户为集群内角色（Cluster Admin、NS Admin、NS Edit、NS View）

该集群角色的 YAML 示例如下：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    kpanda.io/creator: system
  creationTimestamp: "2022-06-16T09:42:49Z"
  labels:
    iam.kpanda.io/role-template: "true"
  name: role-template-cluster-admin
```

```
  resourceVersion: "15168"
  uid: f8f86d42-d5ef-47aa-b284-097615795076
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

命名空间权限

命名空间权限是基于 Kubernetes RBAC 能力的授权，可以实现不同的用户/用户组对命名空间下的资源具有不同的操作权限(包括 Kubernetes API 权限，详情可参考：Kubernetes RBAC。目前容器管理的默认角色为：NS Admin、NS Edit、NS View。

NS Admin

NS Admin 具有以下权限：

1. 可查看对应命名空间
2. 管理、编辑、查看 命名空间下的所有工作负载，及自定义资源
3. 可授权用户为对应命名空间角色（NS Edit、NS View）

该集群角色的 YAML 示例如下：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```yaml
  annotations:
    kpanda.io/creator: system
  creationTimestamp: "2022-06-16T09:42:49Z"
  labels:
    iam.kpanda.io/role-template: "true"
  name: role-template-ns-admin
  resourceVersion: "15173"
  uid: 69f64c7e-70e7-4c7c-a3e0-053f507f2bc3
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

NS Edit

NS Edit 具有以下权限：

1. 可查看对应有权限的命名空间
2. 管理、编辑、查看 命名空间下的所有工作负载

??? note “点击查看集群角色的 YAML 示例”

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
```

```
    kpanda.io/creator: system
  creationTimestamp: "2022-06-16T09:42:50Z"
  labels:
    iam.kpanda.io/role-template: "true"
  name: role-template-ns-edit
  resourceVersion: "15175"
  uid: ca9e690e-96c0-4978-8915-6e4c00c748fe
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - persistentvolumeclaims/status
  - pods
  - replicationcontrollers
  - replicationcontrollers/scale
  - serviceaccounts
  - services
  - services/status
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - bindings
  - events
  - limitranges
  - namespaces/status
  - pods/log
  - pods/status
```

```
  - replicationcontrollers/status
  - resourcequotas
  - resourcequotas/status
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - controllerrevisions
  - daemonsets
  - daemonsets/status
  - deployments
  - deployments/scale
  - deployments/status
  - replicasets
  - replicasets/scale
  - replicasets/status
  - statefulsets
  - statefulsets/scale
  - statefulsets/status
  verbs:
  - '*'
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
```

```
  - horizontalpodautoscalers/status
  verbs:
  - '*'
- apiGroups:
  - batch
  resources:
  - cronjobs
  - cronjobs/status
  - jobs
  - jobs/status
  verbs:
  - '*'
- apiGroups:
  - extensions
  resources:
  - daemonsets
  - daemonsets/status
  - deployments
  - deployments/scale
  - deployments/status
  - ingresses
  - ingresses/status
  - networkpolicies
  - replicasets
  - replicasets/scale
  - replicasets/status
  - replicationcontrollers/scale
  verbs:
  - '*'
- apiGroups:
  - policy
  resources:
```

```
    - poddisruptionbudgets
    - poddisruptionbudgets/status
    verbs:
    - '*'
  - apiGroups:
    - networking.k8s.io
    resources:
    - ingresses
    - ingresses/status
    - networkpolicies
    verbs:
    - '*'
```

NS View

NS View 具有以下权限：

1. 可查看对应命名空间
2. 可查看对应命名空间下的所有工作负载，及自定义资源

??? note “点击查看集群角色的 YAML 示例”

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    kpanda.io/creator: system
  creationTimestamp: "2022-06-16T09:42:50Z"
  labels:
    iam.kpanda.io/role-template: "true"
  name: role-template-ns-view
  resourceVersion: "15183"
```

```
  uid: 853888fd-6ee8-42ac-b91e-63923918baf8
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - persistentvolumeclaims/status
  - pods
  - replicationcontrollers
  - replicationcontrollers/scale
  - serviceaccounts
  - services
  - services/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - bindings
  - events
  - limitranges
  - namespaces/status
  - pods/log
  - pods/status
  - replicationcontrollers/status
  - resourcequotas
  - resourcequotas/status
  verbs:
```

```
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - controllerrevisions
  - daemonsets
  - daemonsets/status
  - deployments
  - deployments/scale
  - deployments/status
  - replicasets
  - replicasets/scale
  - replicasets/status
  - statefulsets
  - statefulsets/scale
  - statefulsets/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - autoscaling
```

```
    resources:
    - horizontalpodautoscalers
    - horizontalpodautoscalers/status
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - batch
    resources:
    - cronjobs
    - cronjobs/status
    - jobs
    - jobs/status
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - extensions
    resources:
    - daemonsets
    - daemonsets/status
    - deployments
    - deployments/scale
    - deployments/status
    - ingresses
    - ingresses/status
    - networkpolicies
    - replicasets
    - replicasets/scale
    - replicasets/status
```

```
  - replicationcontrollers/scale
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  - poddisruptionbudgets/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  - ingresses/status
  - networkpolicies
  verbs:
  - get
  - list
  - watch
```

权限 FAQ

1. 全局权限和容器管理权限管理的关系？

   答：全局权限仅授权为粗粒度权限，可管理所有集群的创建、编辑、删除；而对于细粒度的权限，如单个集群的管理权限，单个命名空间的管理、编辑、删除权限，需要基于 Kubernetes RBAC 的容器管理权限进行实现。一般权限的用户仅需要在容器管理中进行授权即可。

2. 目前仅支持四个默认角色，后台自定义角色的 RoleBinding 以及 ClusterRoleBinding（Kubernetes 细粒度的 RBAC）是否也能生效?

   答：目前自定义权限暂时无法通过图形界面进行管理，但是通过 kubectl 创建的权限规则同样能生效。

集群和命名空间授权

容器管理基于全局权限管理及全局用户/用户组管理实现授权，如需为用户授予容器管理的最高权限（可以创建、管理、删除所有集群），请参见什么是用户与访问控制。

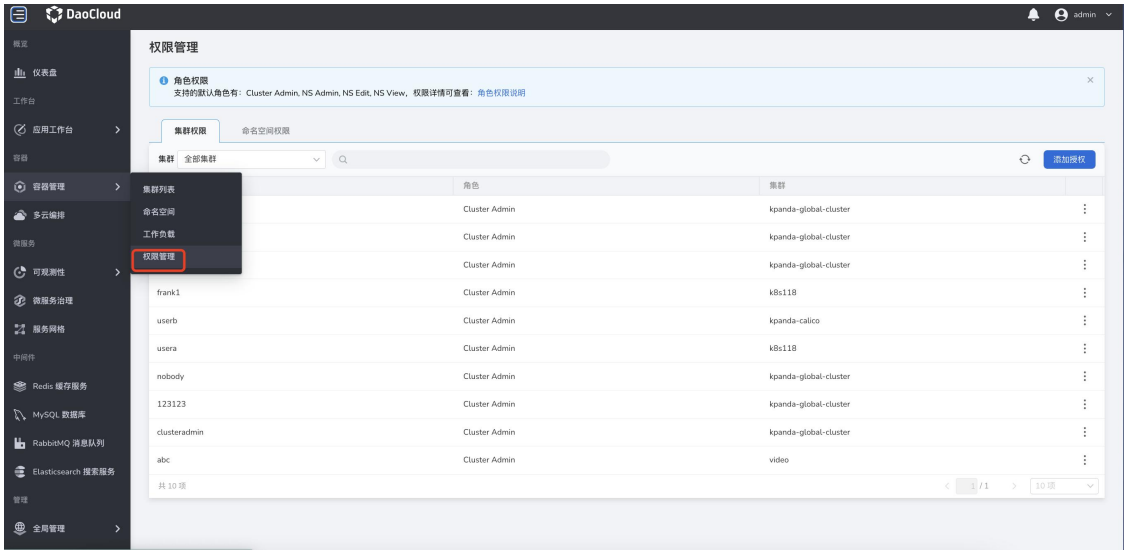
前提条件

给用户/用户组授权之前，请完成如下准备：

- 已在全局管理中创建了待授权的用户/用户组，请参考用户。
- 仅 Kpanda Owner 及当前集群的 Cluster Admin 具备集群授权能力。详情可参考权限说明。
- 仅 Kpanda Owner、当前集群的 Cluster Admin，当前命名空间的 NS Admin 具备命名空间授权能力。

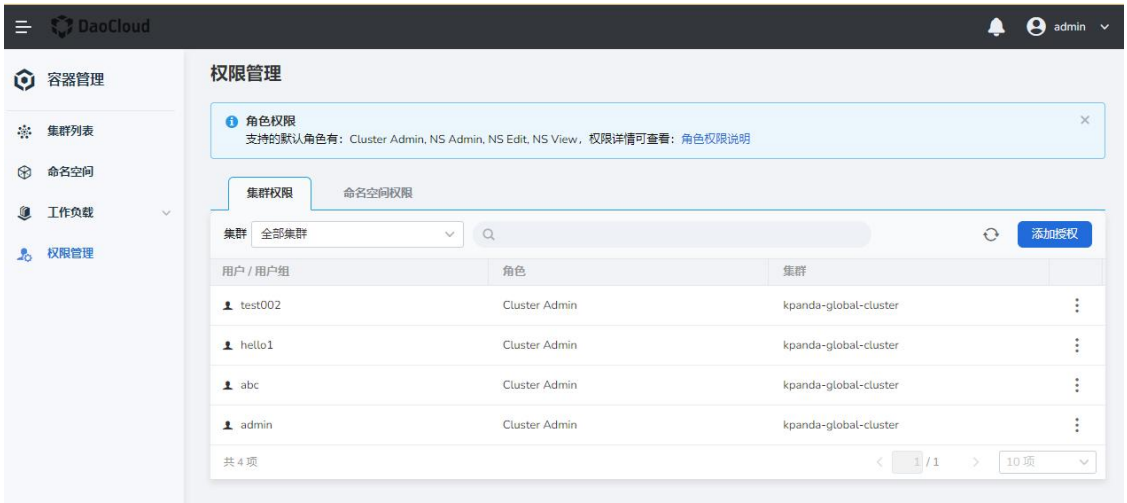
集群授权

1. 用户登录平台后，点击左侧菜单栏容器管理下的权限管理，默认位于集群权限页签。



集群权限

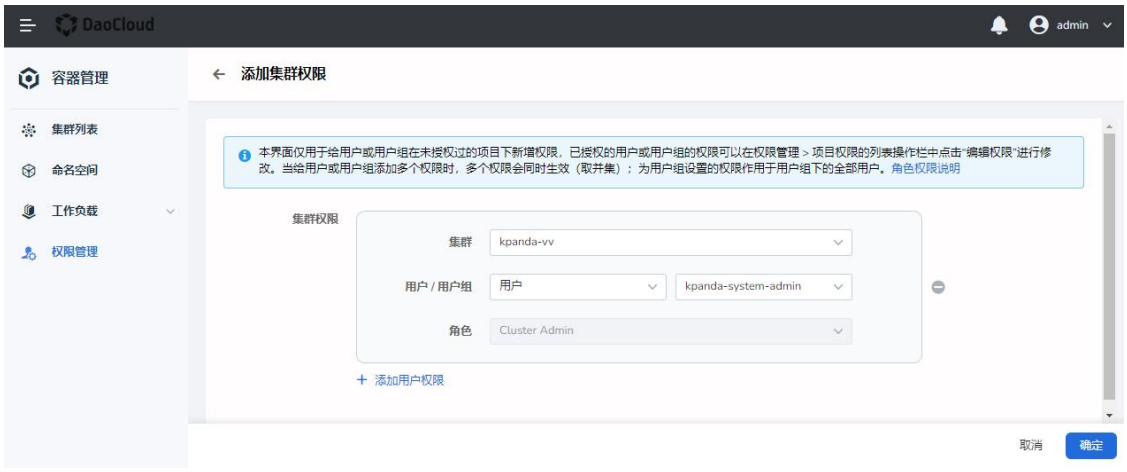
2. 点击添加授权按钮。



添加授权

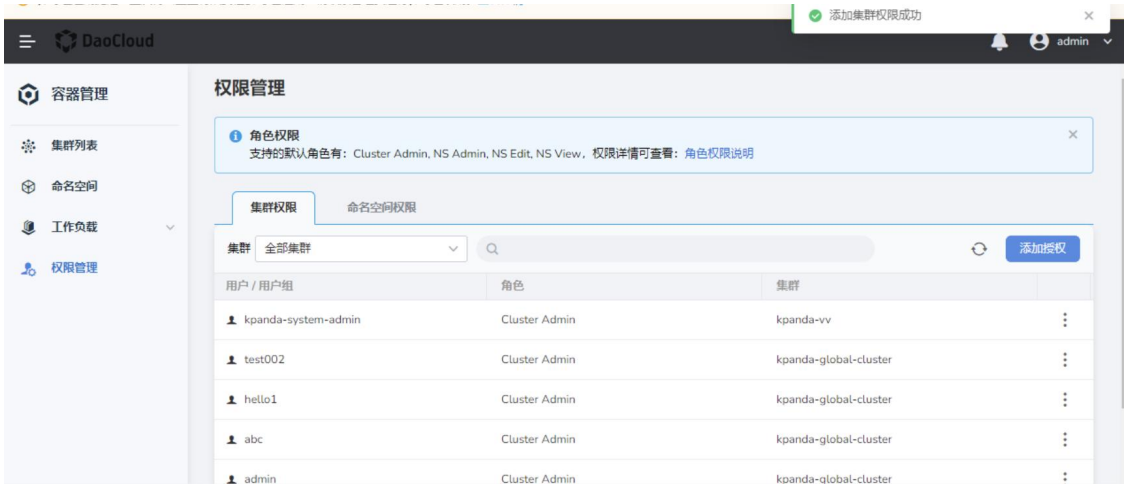
3. 在添加集群权限页面中，选择目标集群、待授权的用户/用户组后，点击确定。

目前仅支持的集群角色为 Cluster Admin，详情权限可参考权限说明。如需要给多个用户/用户组同时进行授权， 可点击添加用户权限进行多次添加。



添加集群权限

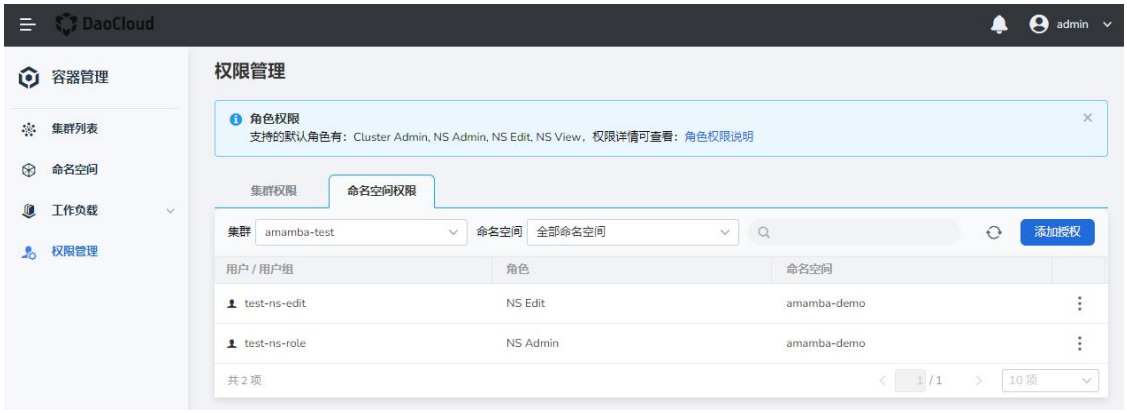
4. 返回集群权限管理页面，屏幕出现消息：添加集群权限成功。



添加成功

命名空间授权

1. 用户登录平台后，点击左侧菜单栏容器管理下的权限管理，点击命名空间权限页签。



命名空间权限

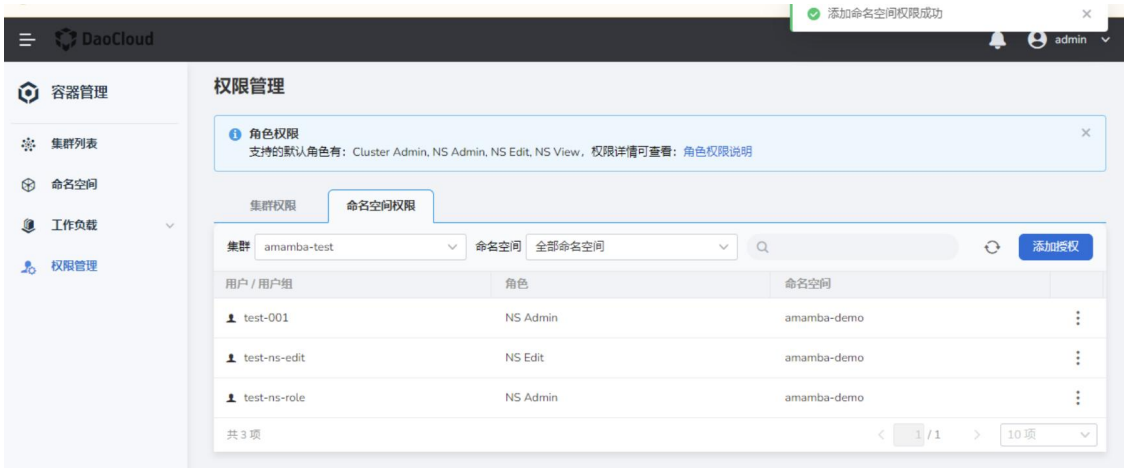
2. 点击添加授权按钮。在添加命名空间权限页面中，选择目标集群、目标命名空间，以及待授权的用户/用户组后，点击确定。

   目前支持的命名空间角色为 NS Admin、NS Edit、NS View，详情权限可参考权限说明。如需给多个用户/用户组同时进行授权，可点击添加用户权限进行多次添加。点击确定完成权限授权。



添加命名空间权限

3. 返回命名空间权限管理页面，屏幕出现消息：添加集群权限成功。



添加成功

!!! tip

后续如需删除或编辑权限，可点击列表右侧的 `⋮`，选择`编辑`或`删除`。

![编辑或删除](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/perm08.png)

数据卷(PV)

数据卷（PersistentVolume，PV）是集群中的一块存储，可由管理员事先制备，或使用存储类（Storage Class）来动态制备。PV 是集群资源，但拥有独立的生命周期，不会随着 Pod 进程结束而被删除。将 PV 挂载到工作负载可以实现工作负载的数据持久化。PV 中保存了可被 Pod 中容器访问的数据目录。

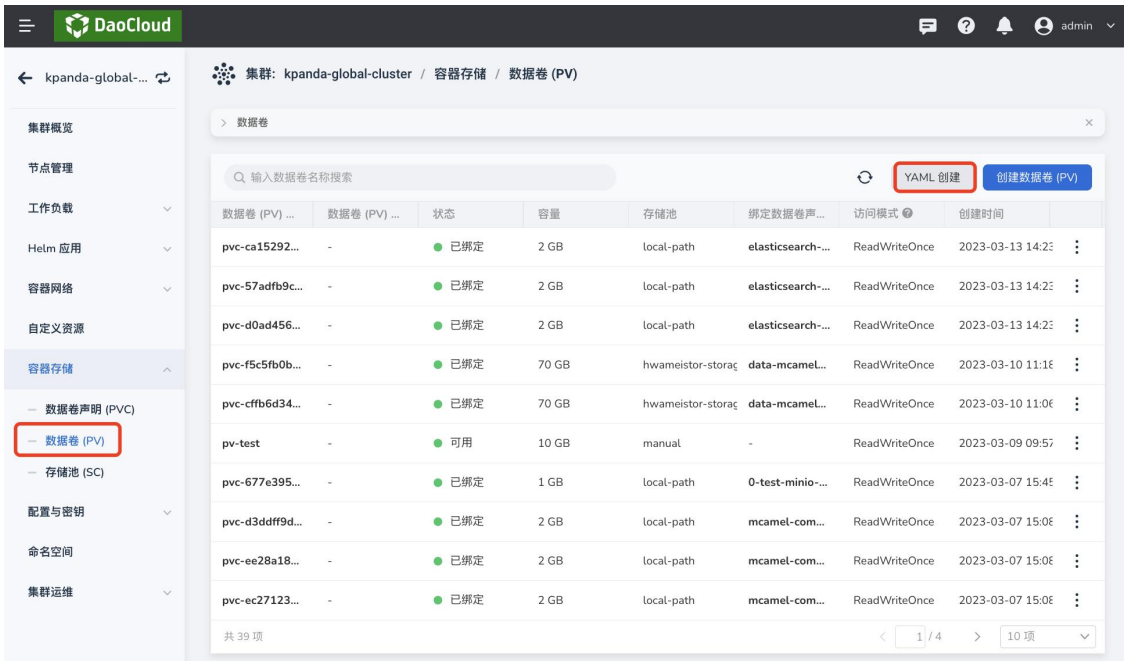
创建数据卷

目前支持通过 YAML 和表单两种方式创建数据卷，这两种方式各有优劣，可以满足不同用户的使用需求。

- 通过 YAML 创建步骤更少、更高效，但门槛要求较高，需要熟悉数据卷的 YAML 文件配置。
- 通过表单创建更直观更简单，根据提示填写对应的值即可，但步骤更加繁琐。

YAML 创建

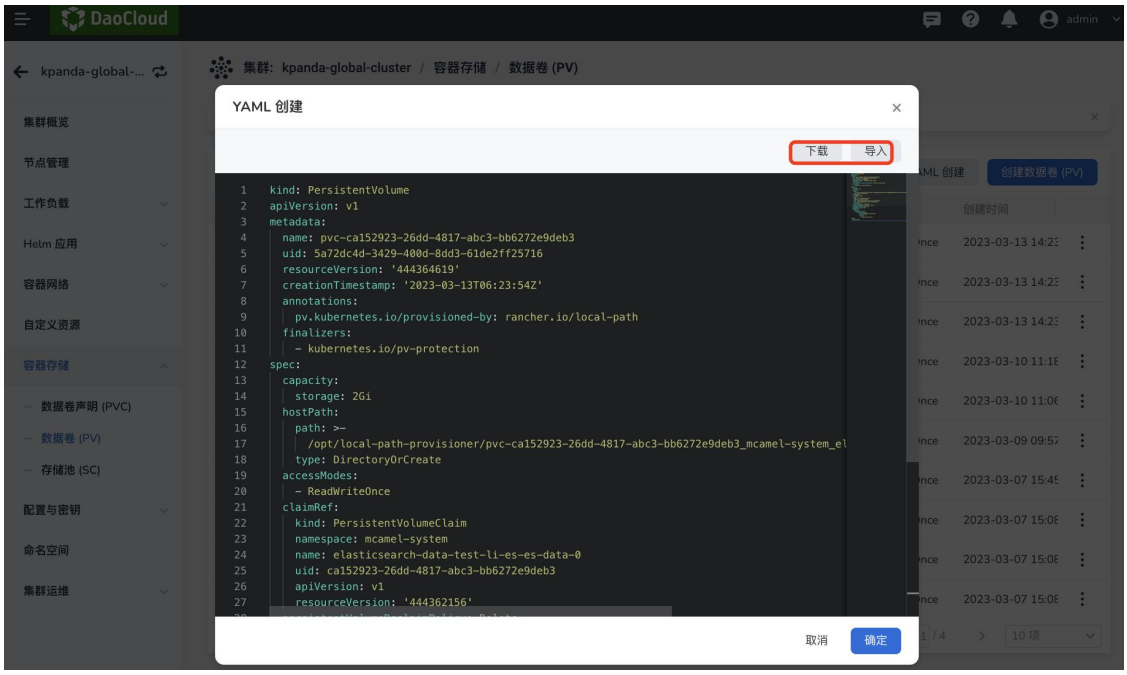
1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储->数据卷(PV)->YAML 创建。



路径

2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击确定。

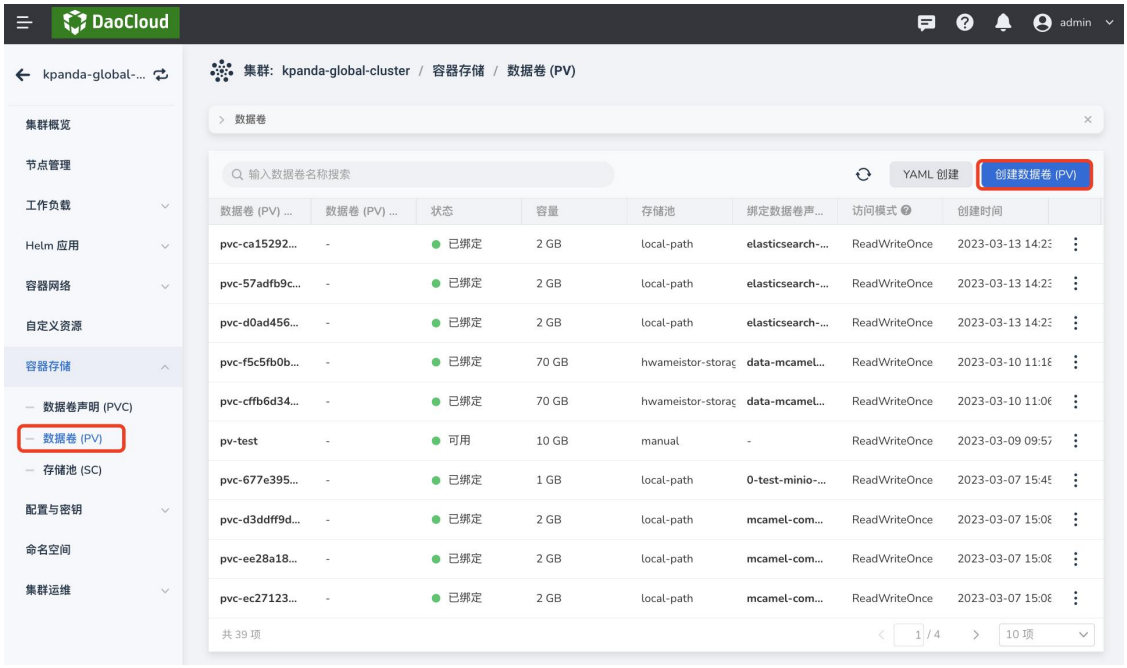
   支持从本地导入 YAML 文件或将填写好的文件下载保存到本地。



yaml

表单创建

1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>数据卷(PV)–>创建数据卷(PV)。



路径

2. 填写基本信息。

    - 数据卷名称、数据卷类型、挂载路径、卷模式、节点亲和性在创建之后不可更改。
    - 数据卷类型：有关卷类型的详细介绍，可参考 Kubernetes 官方文档卷。
        - Local：将 Node 节点的本地存储包装成 PVC 接口，容器直接使用 PVC 而无需关注底层的存储类型。Local 卷不支持动态配置数据卷，但支持配置节点亲和性，可以限制能从哪些节点上访问该数据卷。
        - HostPath：使用 Node 节点的文件系统上的文件或目录作为数据卷，不支持基于节点亲和性的 Pod 调度。
    - 挂载路径：将数据卷挂载到容器中的某个具体目录下。
    - 访问模式：

- ReadWriteOnce：数据卷可以被一个节点以读写方式挂载。
- ReadWriteMany：数据卷可以被多个节点以读写方式挂载。
- ReadOnlyMany：数据卷可以被多个节点以只读方式挂载。
- ReadWriteOncePod：数据卷可以被单个 Pod 以读写方式挂载。

- 回收策略：

  - Retain：不删除 PV，仅将其状态变为 released，需要用户手动回收。有关如何手动回收，可参考持久卷。
  - Recycle：保留 PV 但清空其中的数据，执行基本的擦除操作（rm –rf /thevolume/*）。
  - Delete：删除 PV 时及其中的数据。

- 卷模式：

  - 文件系统：数据卷将被 Pod 挂载到某个目录。如果数据卷的存储来自某块设备而该设备目前为空，第一次挂载卷之前会在设备上创建文件系统。
  - 块：将数据卷作为原始块设备来使用。这类卷以块设备的方式交给 Pod 使用，其上没有任何文件系统，可以让 Pod 更快地访问数据卷。

- 节点亲和性：

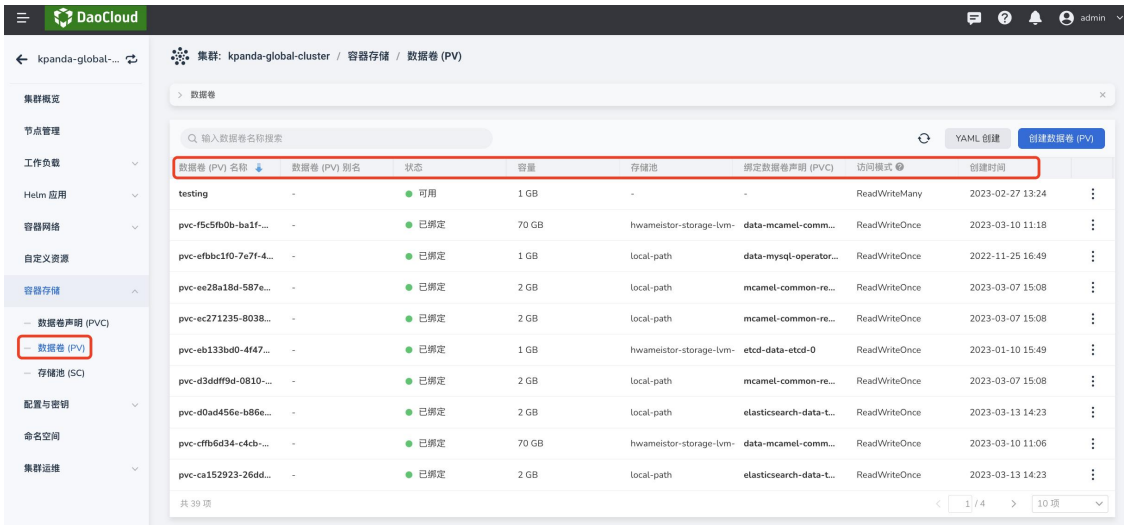


基本信息

查看数据卷

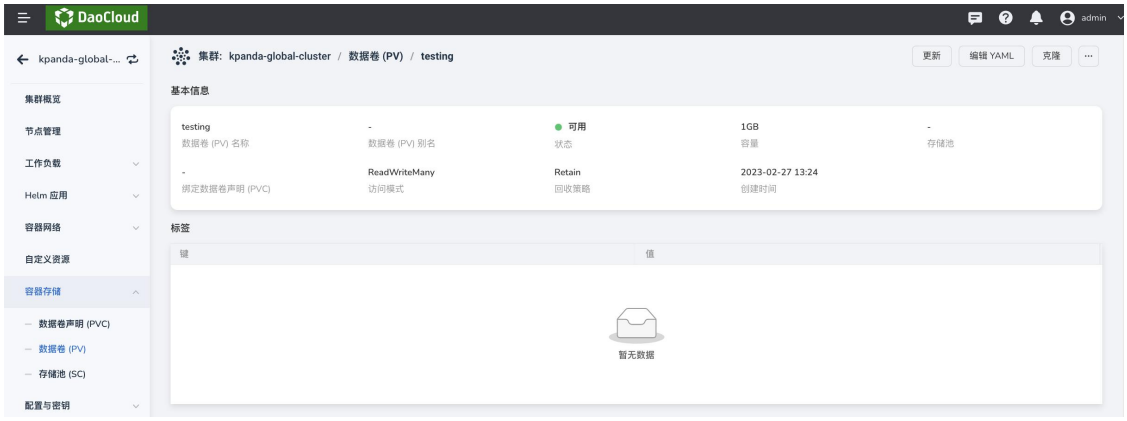
在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>数据卷(PV)。

- 该页面可以查看当前集群中的所有数据卷，以及各个数据卷的状态、容量、命名空间等信息。
- 支持按照数据卷的名称、状态、命名空间、创建时间进行顺序或逆序排序。



详情

- 点击数据卷的名称，可以查看该数据卷的基本配置、存储池信息、标签、注解等信息。



详情

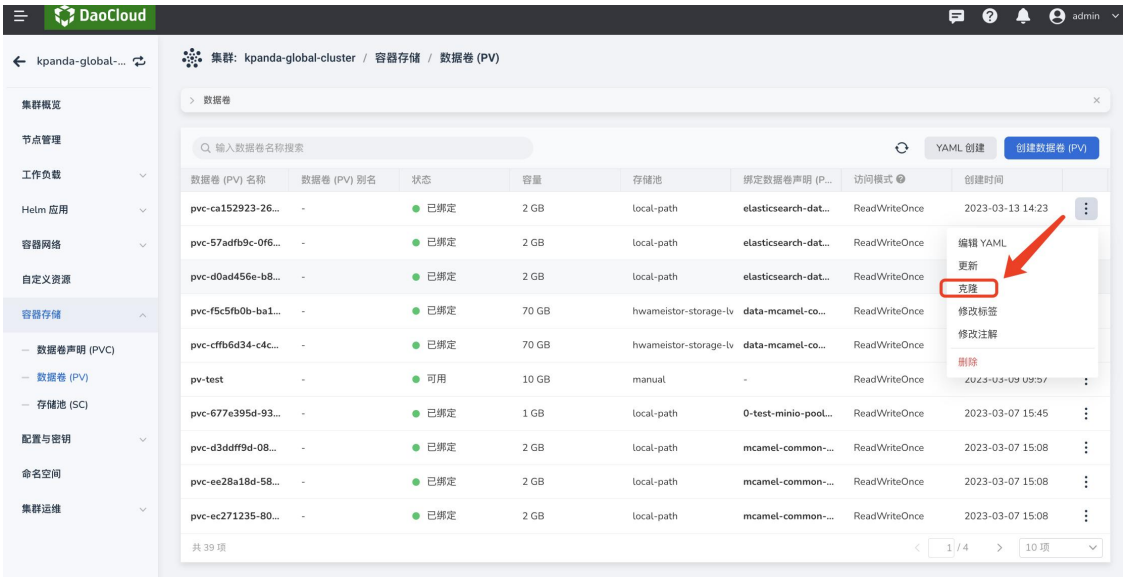
克隆数据卷

通过克隆数据卷，可以基于被克隆数据卷的配置，重新创建一个新的数据卷。

1. 进入克隆页面

    一 在数据卷列表页面，找到需要克隆的数据卷，在右侧的操作栏下选择克隆。

也可以点击数据卷的名称，在详情页面的右上角点击操作按钮选择克隆。



克隆

2. 直接使用原配置，或者按需进行修改，然后在页面底部点击确定。

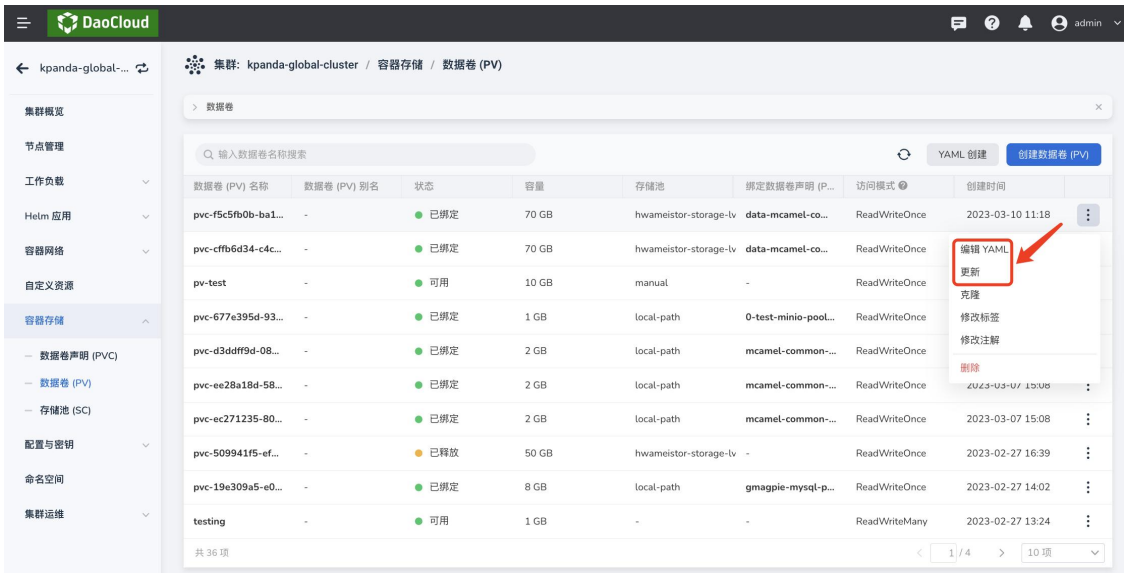
更新数据卷

有两种途径可以更新数据卷。支持通过表单或 YAML 文件更新数据卷。

!!! note

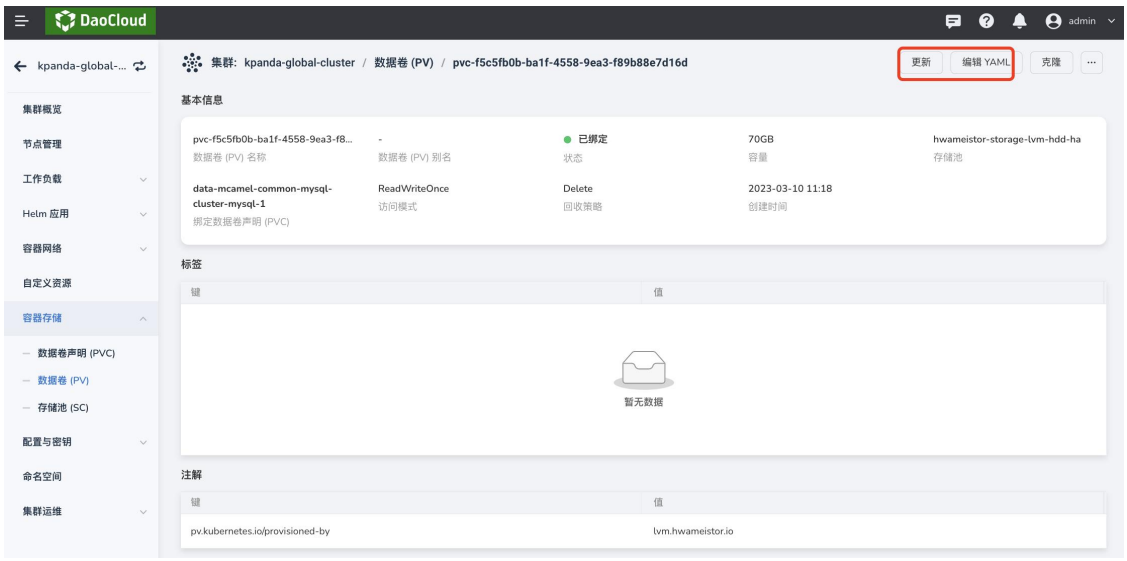
仅支持更新数据卷的别名、容量、访问模式、回收策略、标签和注解。

- 在数据卷列表页面，找到需要更新的数据卷，在右侧的操作栏下选择更新即可通过表单更新，选择编辑 YAML 即可通过 YAML 更新。



更新

- 点击数据卷的名称，进入数据卷的详情页面后，在页面右上角选择更新即可通过表单更新，选择编辑 YAML 即可通过 YAML 更新。

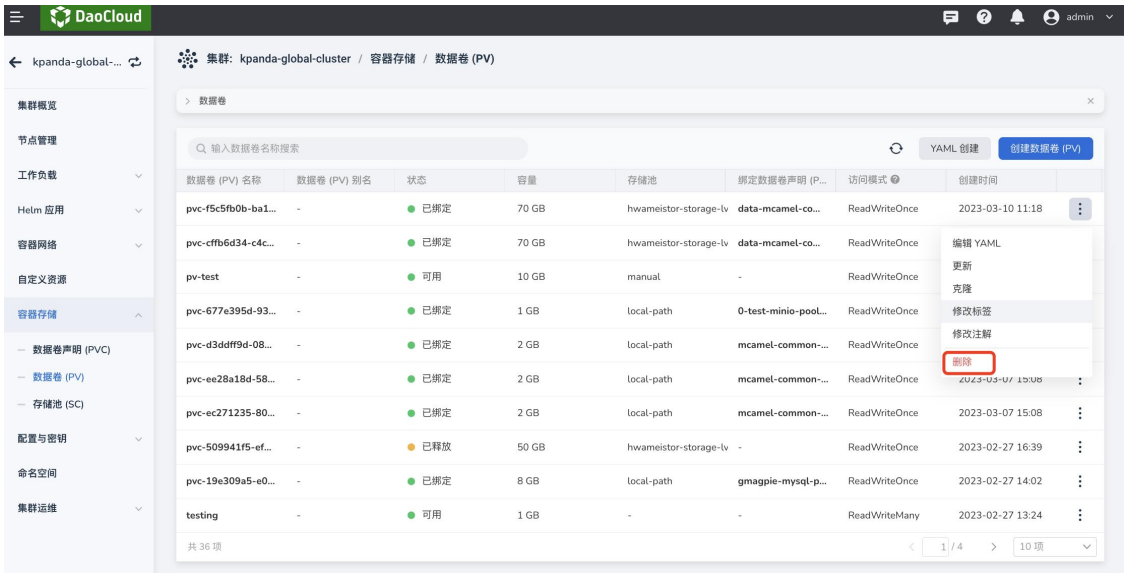


更新

删除数据卷

在数据卷列表页面，找到需要删除的数据，在右侧的操作栏下选择删除。

也可以点击数据卷的名称，在详情页面的右上角点击操作按钮选择删除。

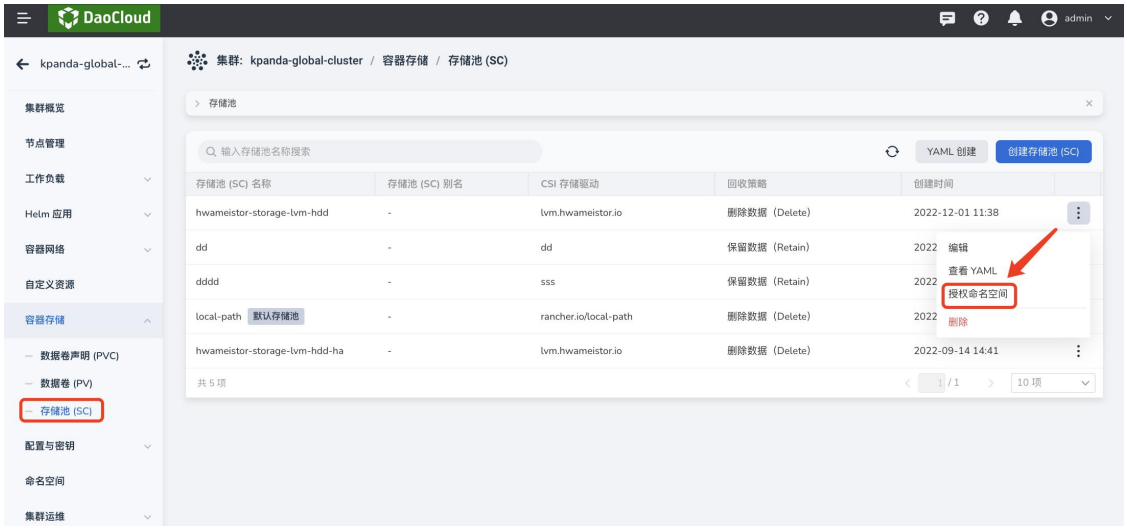

DaoCloud
admin
kpanda-global-...
集群概览
节点管理
工作负载
Helm 应用
容器网络
自定义资源
容器存储
数据卷声明 (PVC)
数据卷 (PV)
存储池 (SC)
配置与密钥
命名空间
集群运维
集群: kpanda-global-cluster / 容器存储 / 数据卷 (PV)
数据卷
输入数据卷名称搜索
YAML 创建
创建数据卷 (PV)
数据卷 (PV) 名称
数据卷 (PV) 别名
状态
容量
存储池
绑定数据卷声明 (P...
访问模式
创建时间
pvc-f5c5fb0b-ba1... - 已绑定 70 GB hwameistor-storage-lv data-mcamel-co... ReadWriteOnce 2023-03-10 11:18
pvc-cffb6d34-c4c... - 已绑定 70 GB hwameistor-storage-lv data-mcamel-co... ReadWriteOnce
pv-test - 可用 10 GB manual - ReadWriteOnce
pvc-677e395d-93... - 已绑定 1 GB local-path 0-test-minio-pool... ReadWriteOnce
pvc-d3ddff9d-08... - 已绑定 2 GB local-path mcamel-common-... ReadWriteOnce
pvc-ee28a18d-58... - 已绑定 2 GB local-path mcamel-common-... ReadWriteOnce
pvc-ec271235-80... - 已绑定 2 GB local-path mcamel-common-... ReadWriteOnce 2023-03-07 15:08
pvc-509941f5-ef... - 已释放 50 GB hwameistor-storage-lv - ReadWriteOnce 2023-02-27 16:39
pvc-19e309a5-e0... - 已绑定 8 GB local-path gmagpie-mysql-p... ReadWriteOnce 2023-02-27 14:02
testing - 可用 1 GB - - ReadWriteMany 2023-02-27 13:24
编辑 YAML
更新
克隆
修改标签
修改注解
删除
共 36 项
1 / 4
10 项

删除

共享存储池

DCE 5.0 容器管理模块支持将一个存储池共享给多个命名空间使用，以便提高资源利用效率。

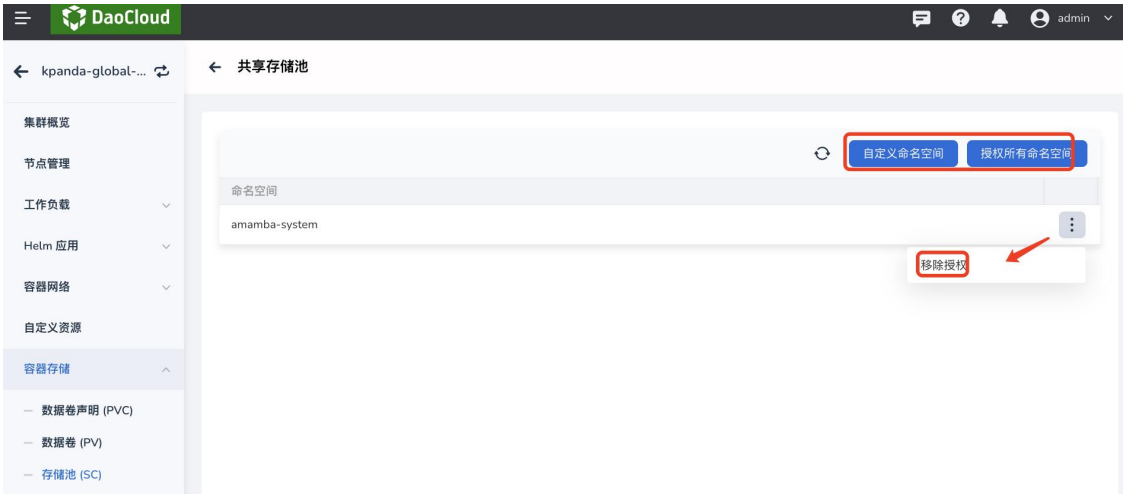
1. 在存储池列表中找到需要共享的存储池，在右侧操作栏下点击授权命名空间。



授权

2. 点击自定义命名空间可以逐一选择需要将此存储池共享到哪些命名空间。

    - 点击授权所有命名空间可以一次性将此存储池共享到当前集群下的所有命名空间。
    - 在列表右侧的操作栏下方点击移除授权，可以解除授权，停止将此存储池共享到该命名空间。


DaoCloud
admin
kpanda-global-...
集群概览
节点管理
工作负载
Helm 应用
容器网络
自定义资源
容器存储
数据卷声明 (PVC)
数据卷 (PV)
存储池 (SC)
共享存储池
自定义命名空间
授权所有命名空间
命名空间
amamba-system
移除授权

授权

存储池(SC)

存储池指将许多物理磁盘组成一个大型存储资源池，本平台支持接入各类存储厂商后创建块存储池、本地存储池、自定义存储池，然后为工作负载动态配置数据卷。

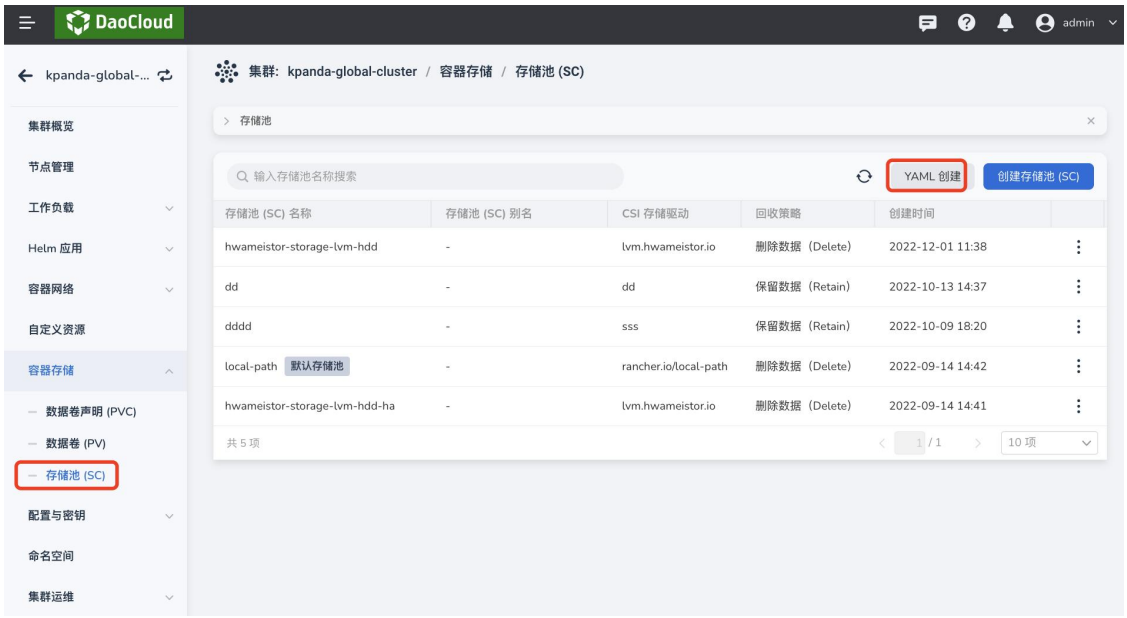
创建存储池(SC)

目前支持通过 YAML 和表单两种方式创建存储池，这两种方式各有优劣，可以满足不同用户的使用需求。

- 通过 YAML 创建步骤更少、更高效，但门槛要求较高，需要熟悉存储池的 YAML 文件配置。
- 通过表单创建更直观更简单，根据提示填写对应的值即可，但步骤更加繁琐。

YAML 创建

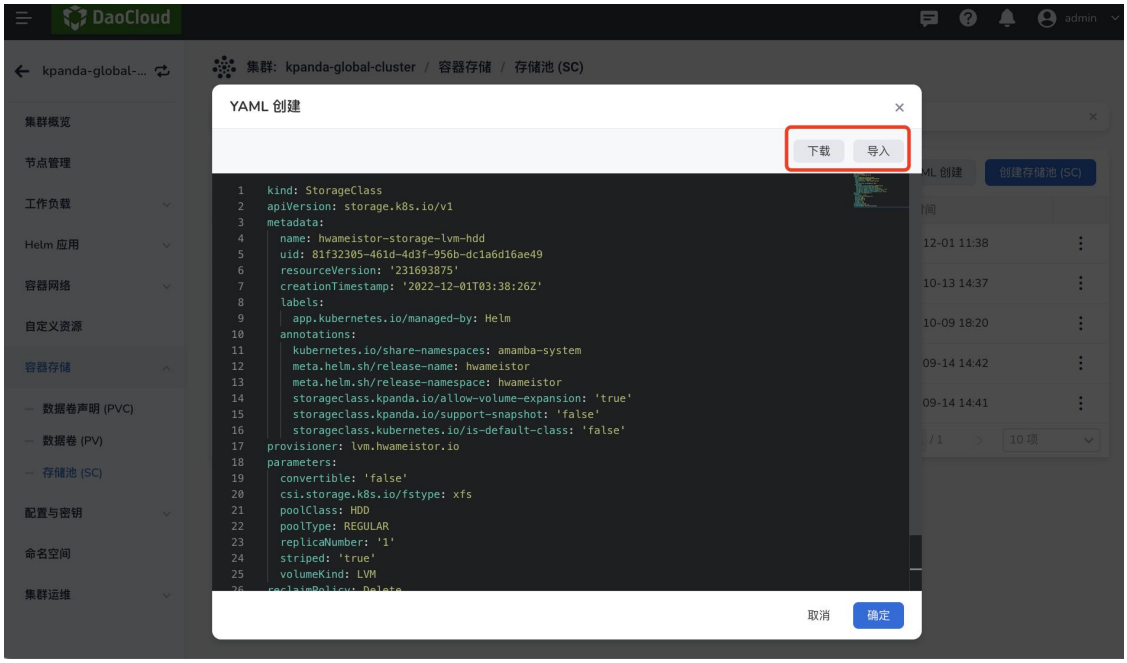
1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>存储池(SC)–>YAML 创建。



路径

2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击确定。

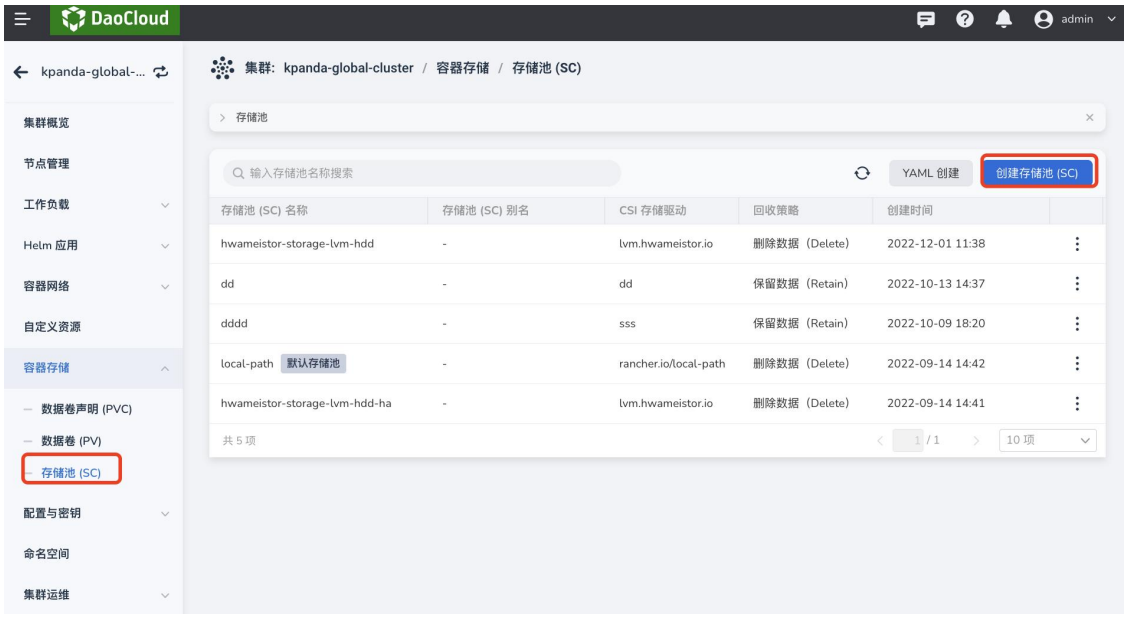
   支持从本地导入 YAML 文件或将填写好的文件下载保存到本地。



yaml

表单创建

1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>存储池(SC)–>创建存储池(SC)。



路径

2. 填写基本信息，然后在底部点击确定。

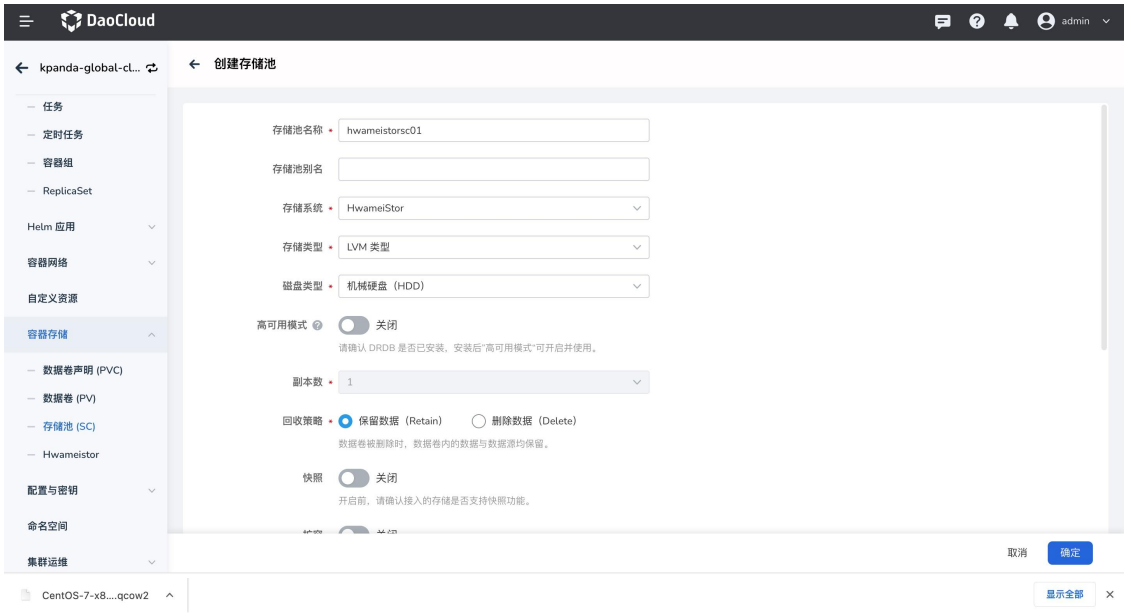
自定义存储系统

- 存储池名称、驱动、回收策略在创建后不可修改。
- CSI 存储驱动：基于标准 Kubernetes 的容器存储接口插件，需遵守存储厂商规定的格式，例如 rancher.io/local–path。
  - 有关如何填写不同厂商提供的 CSI 驱动，可参考 Kubernetes 官方文档存储类。
- 回收策略：删除数据卷时，保留数据卷中的数据或者删除其中的数据。

- 快照/扩容：开启后，基于该存储池的数据卷/数据卷声明才能支持扩容和快照功能，但 **前提是底层使用的存储驱动支持快照和扩容功能**。

Hwameistor 存储系统

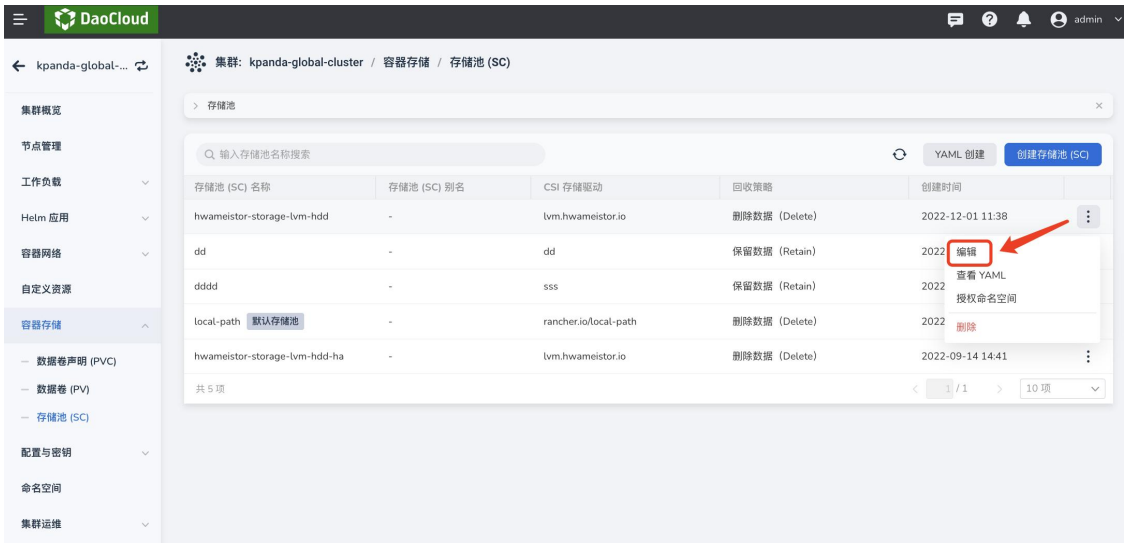
- 存储池名称、驱动、回收策略在创建后不可修改。
- 存储系统：Hwameistor 存储系统。
- 存储类型：支持 LVM，裸磁盘类型
  - LVM 类型：Hwameistor 推荐使用方式，可使用高可用数据卷，对应的的 CSI 存储驱动为： lvm.hwameistor.io。
  - 裸磁盘数据卷： 适用于高可用场景，无高可用能力，对应的 CSI 驱动为： hdd.hwameistor.io
- 高可用模式：使用高可用能力之前请确认 DRDB 组件已安装。开启高可用模式后，可将数据卷副本数设置为 1 和 2。 如需要可将数据卷副本从 1 Convert 成 1
- 回收策略：删除数据卷时，保留数据卷中的数据或者删除其中的数据。
- 快照/扩容：开启后，基于该存储池的数据卷/数据卷声明才能支持扩容和快照功能，但 **前提是底层使用的存储驱动支持快照和扩容功能**。



基本信息

更新存储池(SC)

在存储池列表页面，找到需要更新的存储池，在右侧的操作栏下选择编辑即可通过更新存储池。



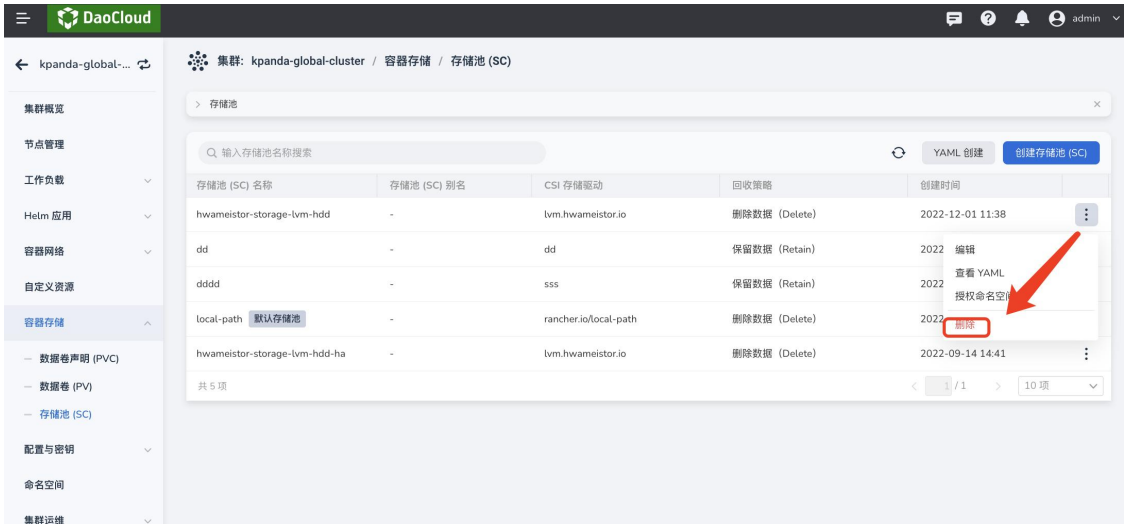
更新

!!! info

选择`查看 YAML` 可以查看该存储池的 YAML 文件，但不支持编辑。

删除存储池(SC)

在存储池列表页面，找到需要删除的存储池，在右侧的操作栏下选择删除。



删除

数据卷声明(PVC)

持久卷声明（PersistentVolumeClaim，PVC）表达的是用户对存储的请求。PVC 消耗 PV 资源，申领使用特定大小、特定访问模式的数据卷，例如要求 PV 卷以 ReadWriteOnce、ReadOnlyMany 或 ReadWriteMany 等模式来挂载。

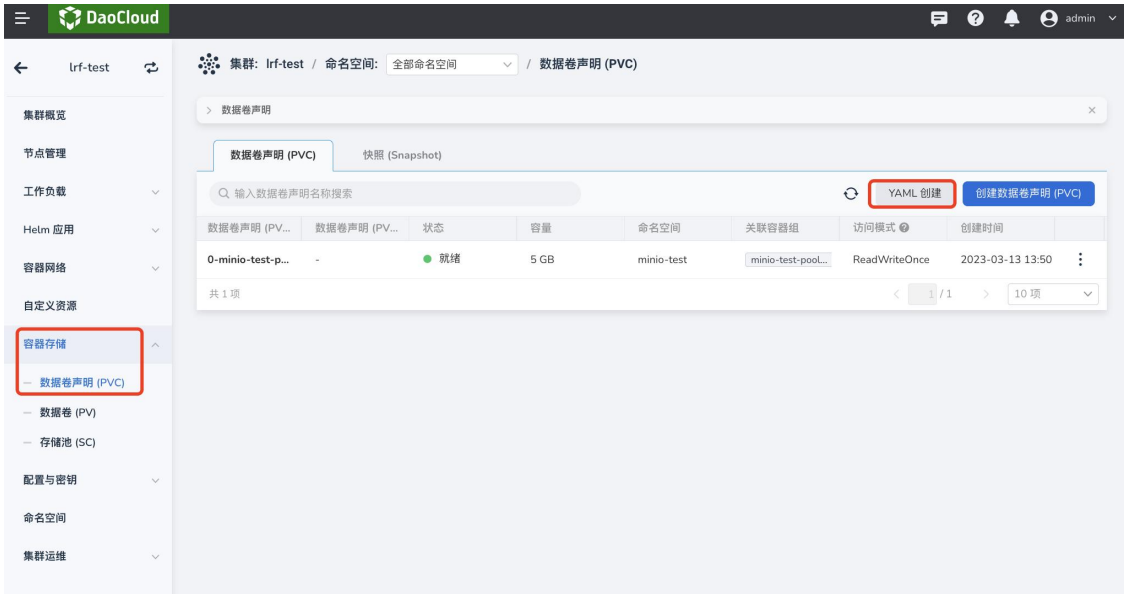
创建数据卷声明

目前支持通过 YAML 和表单两种方式创建数据卷声明，这两种方式各有优劣，可以满足不同用户的使用需求。

- 通过 YAML 创建步骤更少、更高效，但门槛要求较高，需要熟悉数据卷声明的 YAML 文件配置。
- 通过表单创建更直观更简单，根据提示填写对应的值即可，但步骤更加繁琐。

YAML 创建

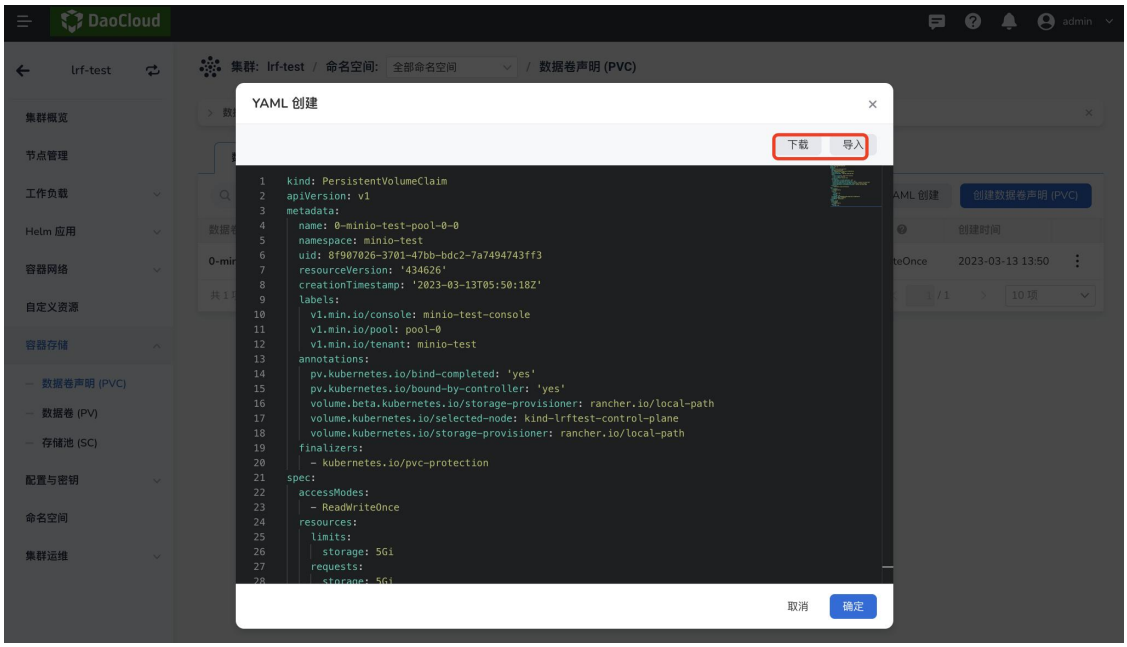
1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储->数据卷声明（PVC)->YAML 创建。



路径

2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击确定。

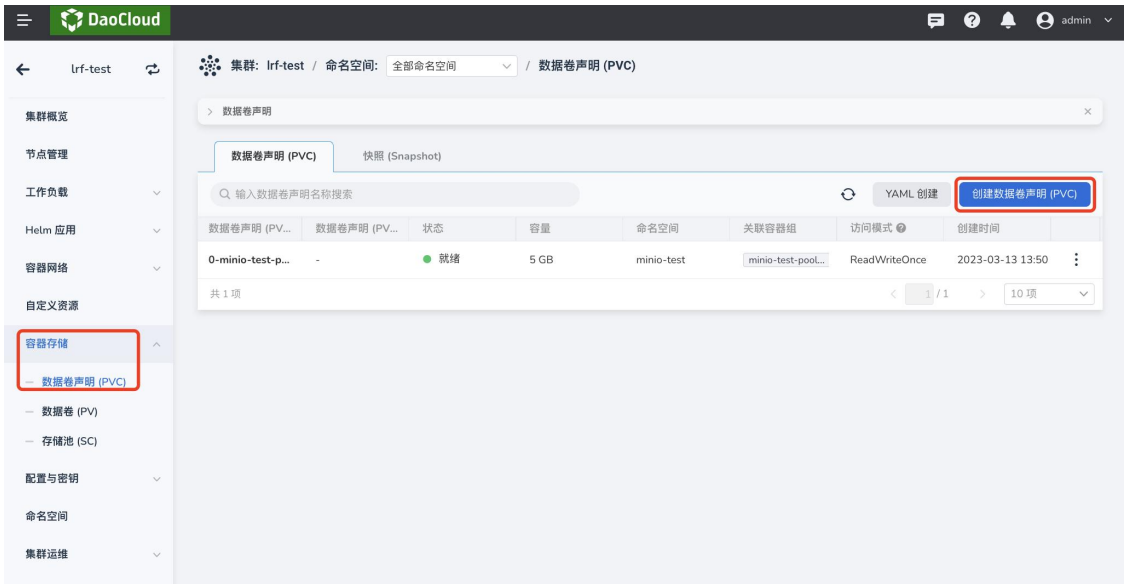
   支持从本地导入 YAML 文件或将填写好的文件下载保存到本地。



yaml

表单创建

1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>数据卷声明（PVC)–>创建数据卷声明（PVC)。



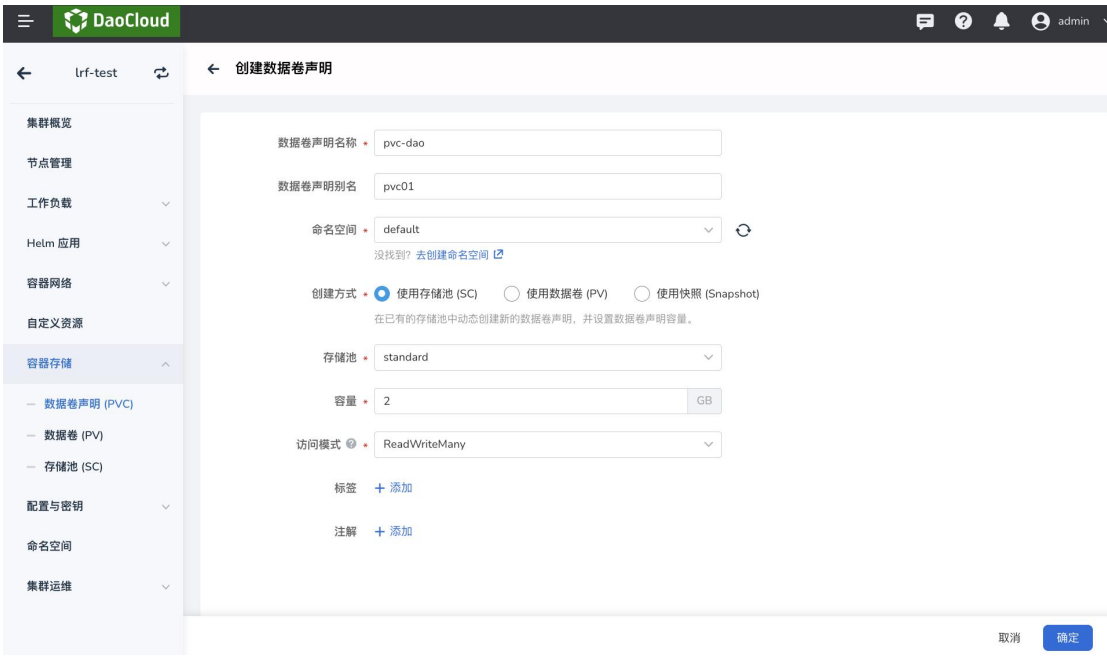
路径

2. 填写基本信息。

   - 数据卷声明的名称、命名空间、创建方式、数据卷、容量、访问模式在创建之后不可更改。
   - 创建方式：在已有的存储池或者数据卷中动态创建新的数据卷声明，或者基于数据卷声明的快照创建新的数据卷声明。

     基于快照创建时无法修改数据卷声明的容量，可以在创建完成后再进行修改。
   - 选择创建方式之后，在下拉列表中选择想要使用的存储池/数据卷/快照。
   - 访问模式：
     - ReadWriteOnce，数据卷声明可以被一个节点以读写方式挂载。

- ReadWriteMany，数据卷声明可以被多个节点以读写方式挂载。
- ReadOnlyMany，数据卷声明可以被多个节点以只读方式挂载。
- ReadWriteOncePod，数据卷声明可以被单个 Pod 以读写方式挂载。

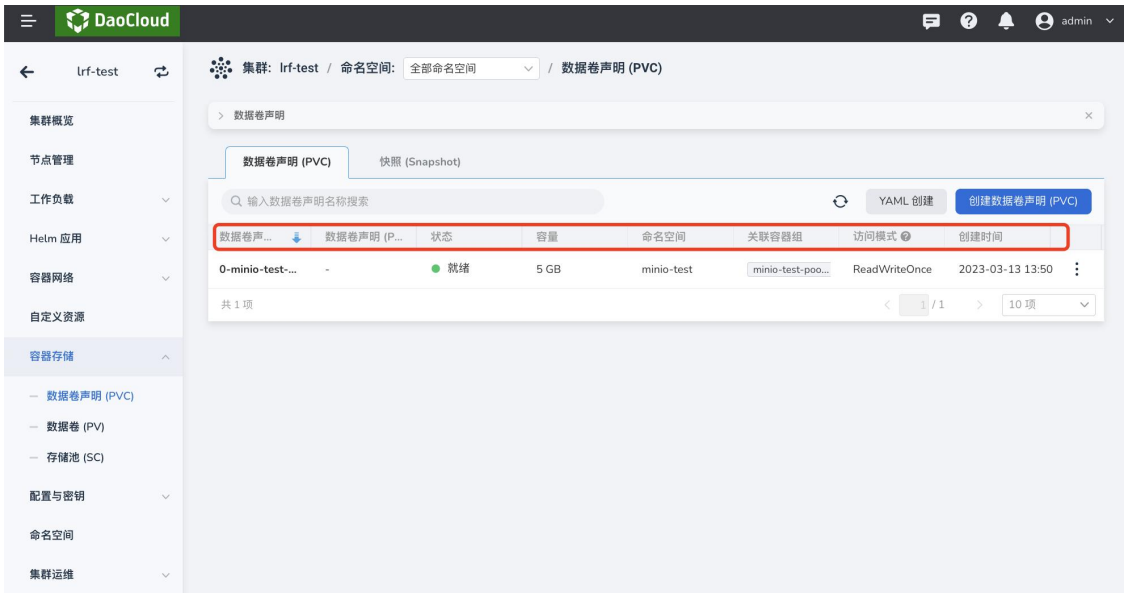


基本信息

查看数据卷声明

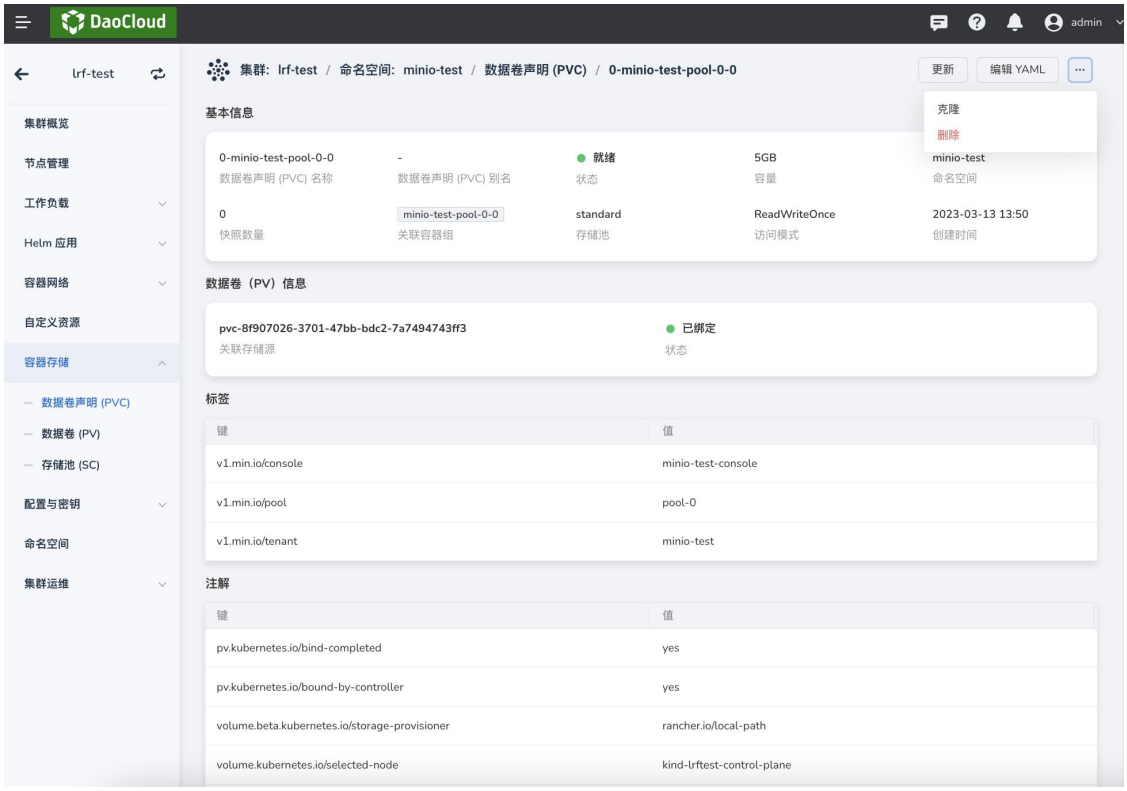
在集群列表中点击目标集群的名称，然后在左侧导航栏点击容器存储–>数据卷声明(PVC)。

- 该页面可以查看当前集群中的所有数据卷声明，以及各个数据卷声明的状态、容量、命名空间等信息。
- 支持按照数据卷声明的名称、状态、命名空间、创建时间进行顺序或逆序排序。



详情

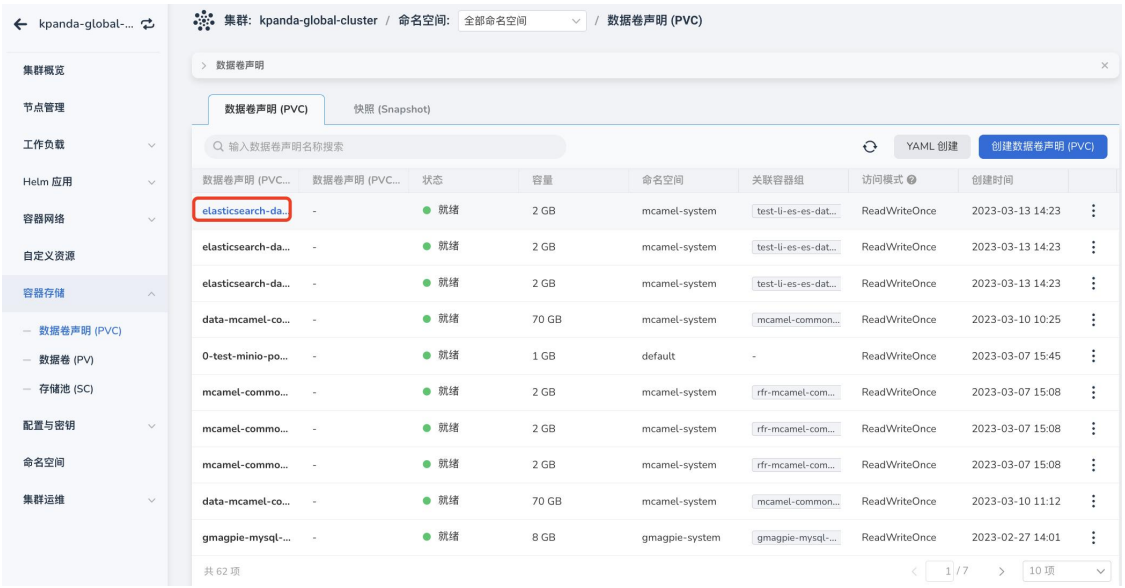
- 点击数据卷声明的名称，可以查看该数据卷声明的基本配置、存储池信息、标签、注解等信息。



详情

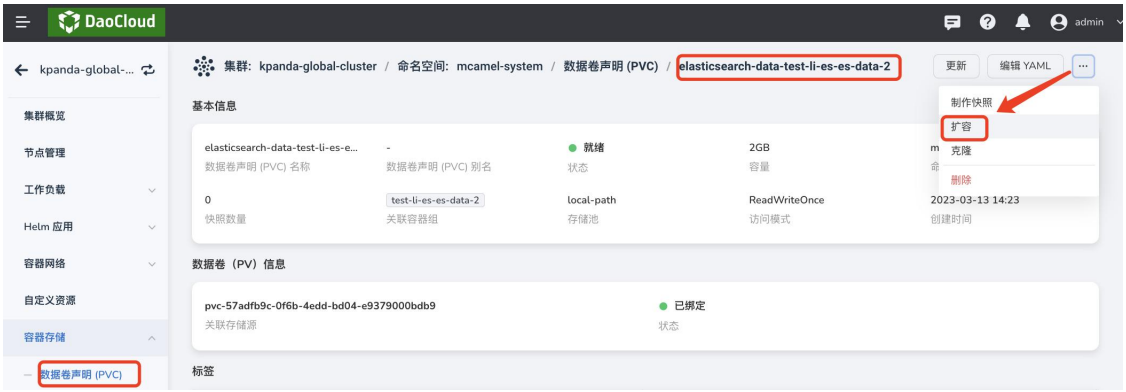
扩容数据卷声明

1. 在左侧导航栏点击容器存储–>数据卷声明(PVC)，找到想要调整容量的数据卷声明。



扩容

2. 点击数据卷声明的名称，然后在页面右上角点击操作按钮选择扩容。



扩容

3. 输入目标容量，然后点击确定。



克隆

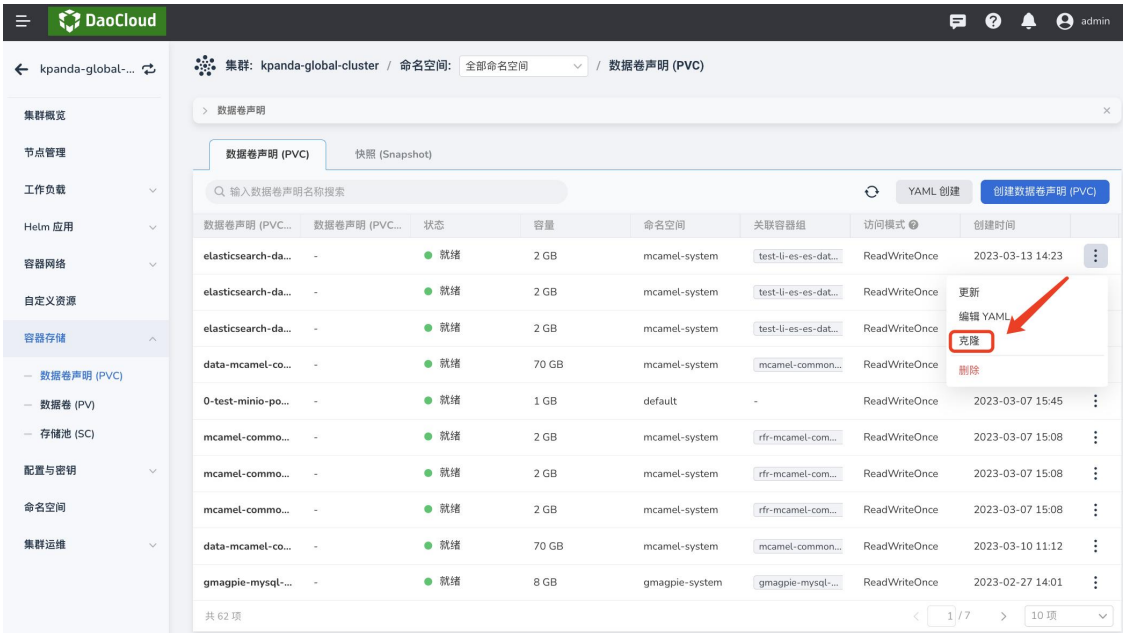
克隆数据卷声明

通过克隆数据卷声明，可以基于被克隆数据卷声明的配置，重新创建一个新的数据卷声明。

1. 进入克隆页面

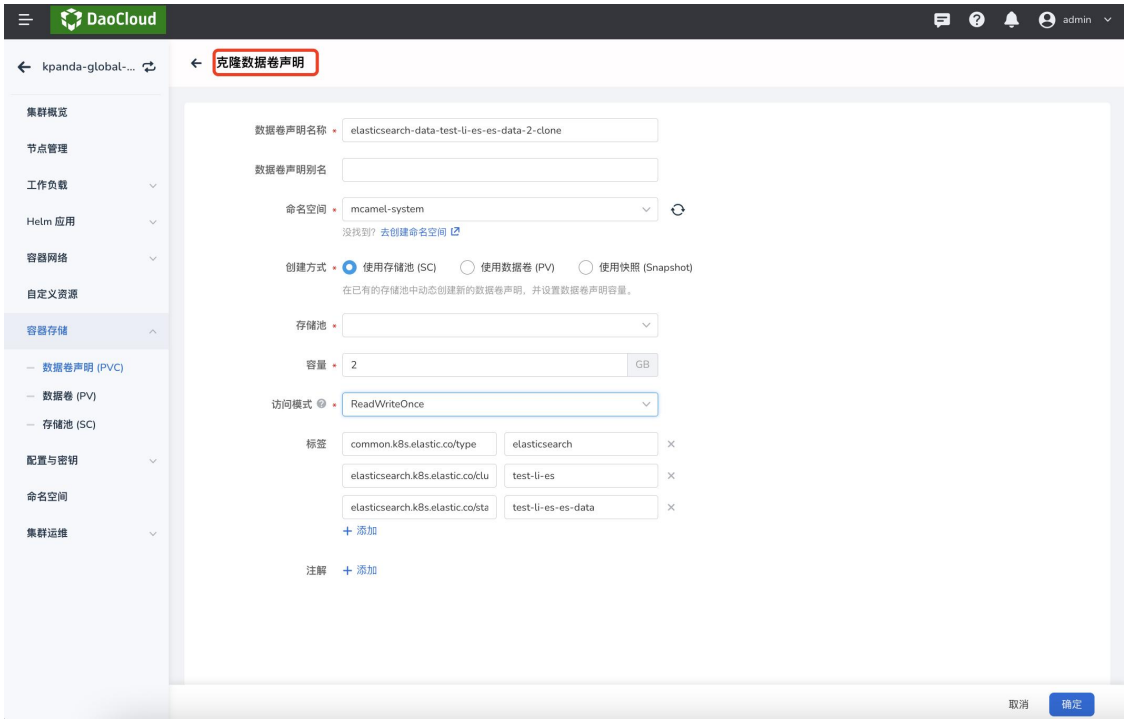
   - 在数据卷声明列表页面，找到需要克隆的数据卷声明，在右侧的操作栏下选择克隆。

     也可以点击数据卷声明的名称，在详情页面的右上角点击操作按钮选择克隆。



克隆

2. 直接使用原配置，或者按需进行修改，然后在页面底部点击确定。



克隆

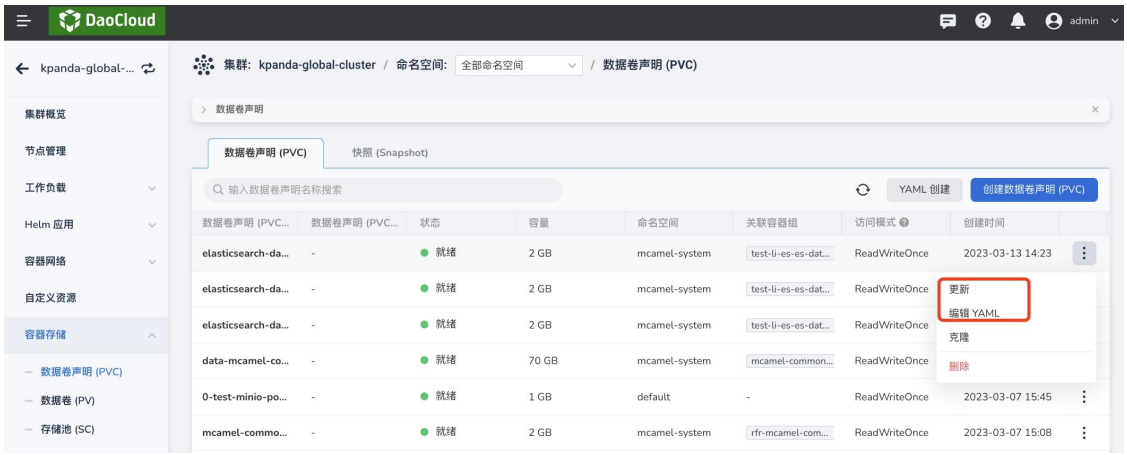
更新数据卷声明

有两种途径可以更新数据卷声明。支持通过表单或 YAML 文件更新数据卷声明。

!!! note

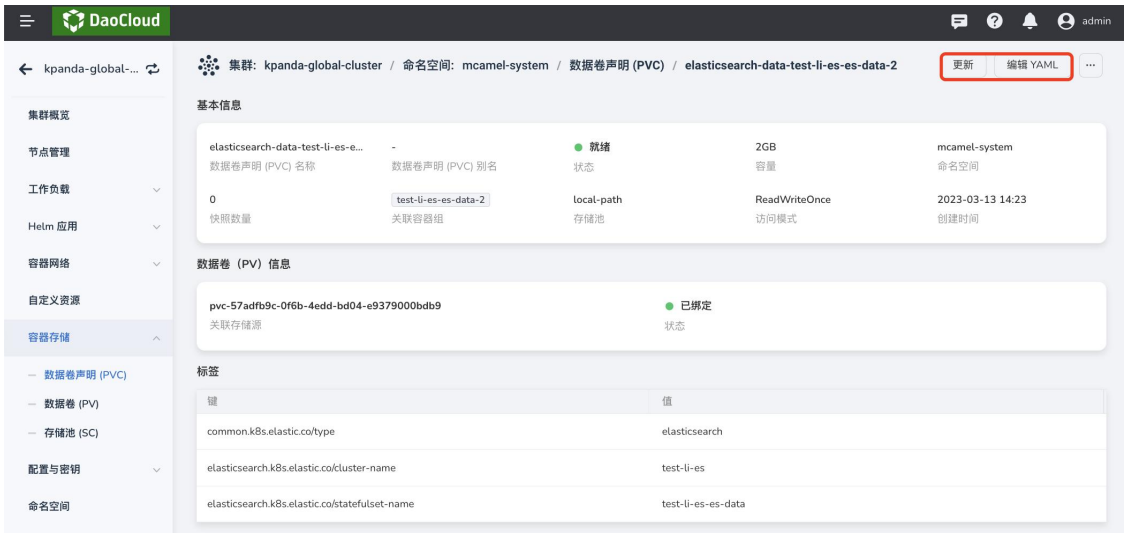
仅支持更新数据卷声明的别名、标签和注解。

- 在数据卷列表页面，找到需要更新的数据卷声明，在右侧的操作栏下选择更新即可通过表单更新，选择编辑 YAML 即可通过 YAML 更新。



更新

- 点击数据卷声明的名称，进入数据卷声明的详情页面后，在页面右上角选择更新即可通过表单更新，选择编辑 YAML 即可通过 YAML 更新。

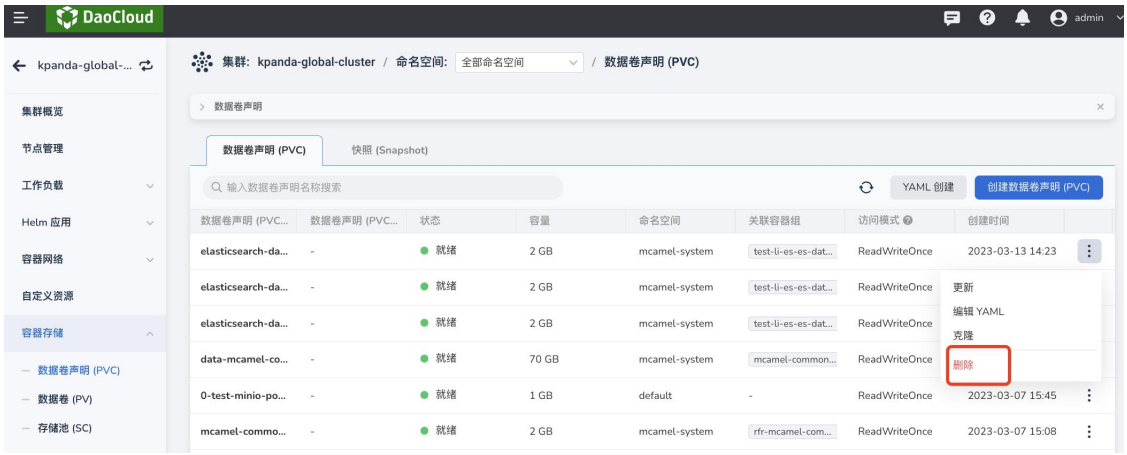


更新

删除数据卷声明

在数据卷声明列表页面，找到需要删除的数据，在右侧的操作栏下选择删除。

> 也可以点击数据卷声明的名称，在详情页面的右上角点击操作按钮选择删除。

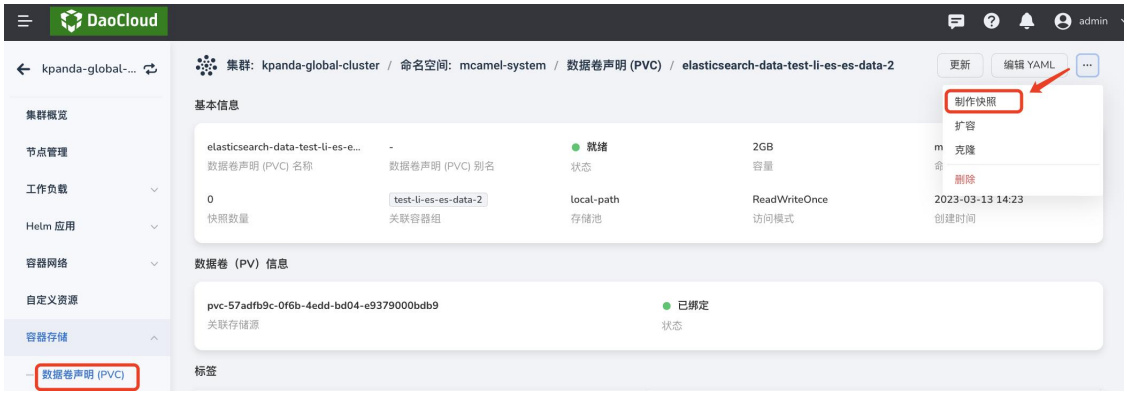


删除

常见问题

1. 如果列表中没有可选的存储池或数据卷，可以创建存储池或创建数据卷。

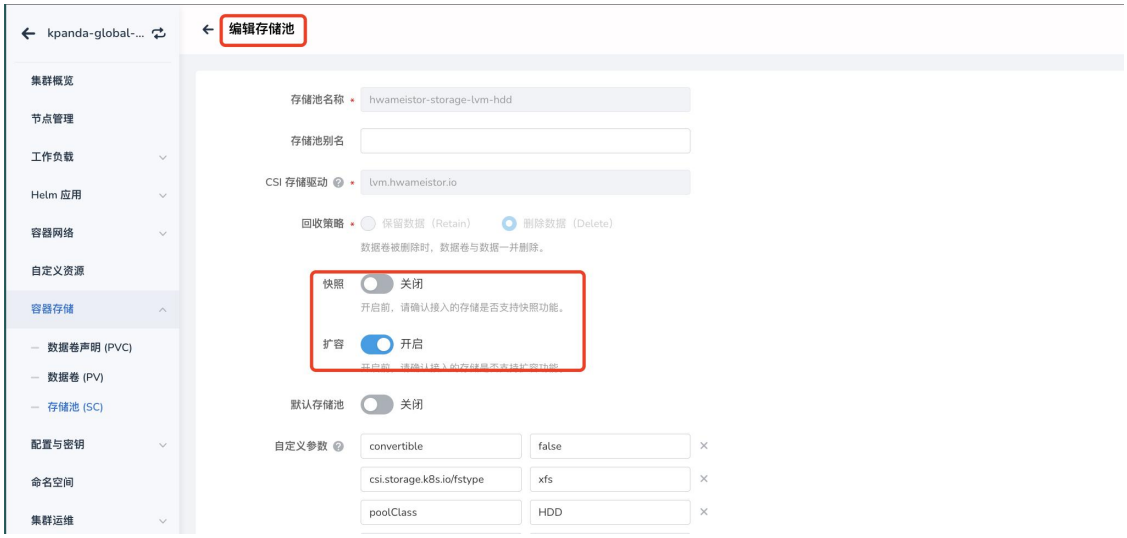
2. 如果列表中没有可选的快照，可以进入数据卷声明的详情页，在右上角制作快照。



制作快照

3. 如果数据卷声明所使用的存储池（SC）没有启用快照，则无法制作快照，页面不会显示“制作快照”选项。

4. 如果数据卷声明所使用的存储池（SC）没有开启扩容功能，则该数据卷不支持扩容，页面不会显示扩容选项。


kpanda-global-...
集群概览
节点管理
工作负载
Helm 应用
容器网络
自定义资源
容器存储
数据卷声明 (PVC)
数据卷 (PV)
存储池 (SC)
配置与密钥
命名空间
集群运维
编辑存储池
存储池名称 *
hwameistor-storage-lvm-hdd
存储池别名
CSI 存储驱动 *
lvm.hwameistor.io
回收策略 *
保留数据（Retain）
删除数据（Delete）
数据卷被删除时，数据卷与数据一并删除。
快照
关闭
开启前，请确认接入的存储是否支持快照功能。
扩容
开启
默认存储池
关闭
自定义参数
convertible
false
csi.storage.k8s.io/fstype
xfs
poolClass
HDD

开启快照

容器组安全策略

容器组安全策略指在 kubernetes 集群中，通过为指定命名空间配置不同的等级和模式，实现在安全的各个方面控制 Pod 的行为，只有满足一定的条件的 Pod 才会被系统接受。它设置三个等级和三种模式，用户可以根据自己的需求选择更加合适的方案来设置限制策略。

!!! note

一条安全模式仅能配置一条安全策略。同时请谨慎为命名空间配置 enforce 的安全模式，违反后将会导致 Pod 无法创建。

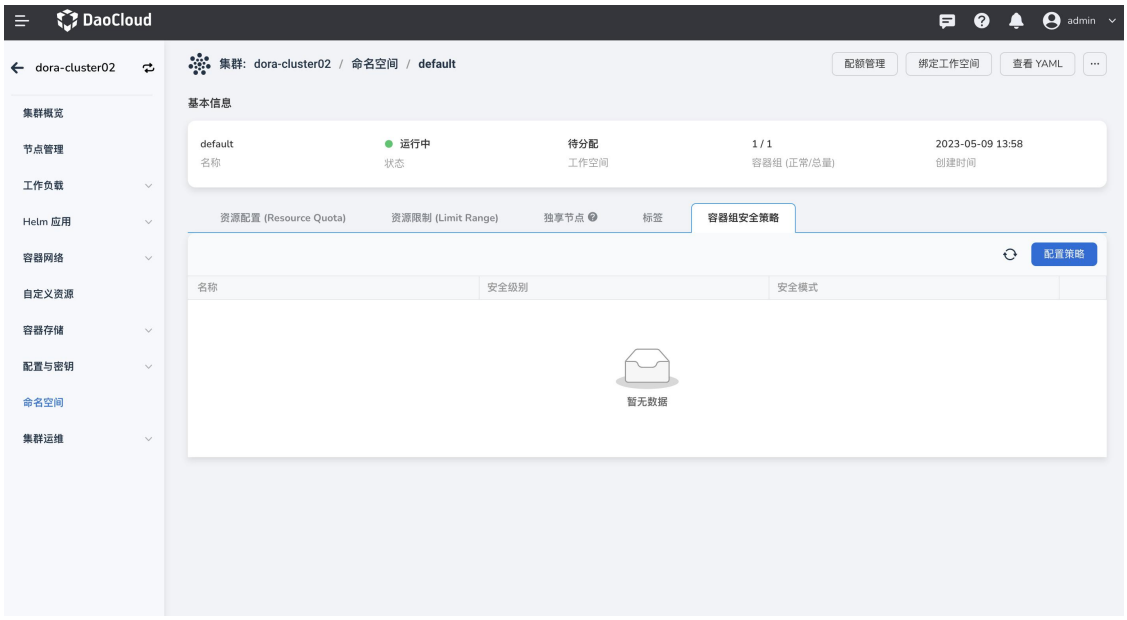
本节将介绍如何通过容器管理界面为命名空间配置容器组安全策略。

前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，集群的版本需要在 v1.22 以上，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建、用户的创建，并为用户授予 NS Admin 或更高权限，详情可参考命名空间授权。

为命名空间配置容器组安全策略

1. 选择需要配置容器组安全策略的命名空间，进入详情页。在容器组安全策略页面点击配置策略，进入配置页。



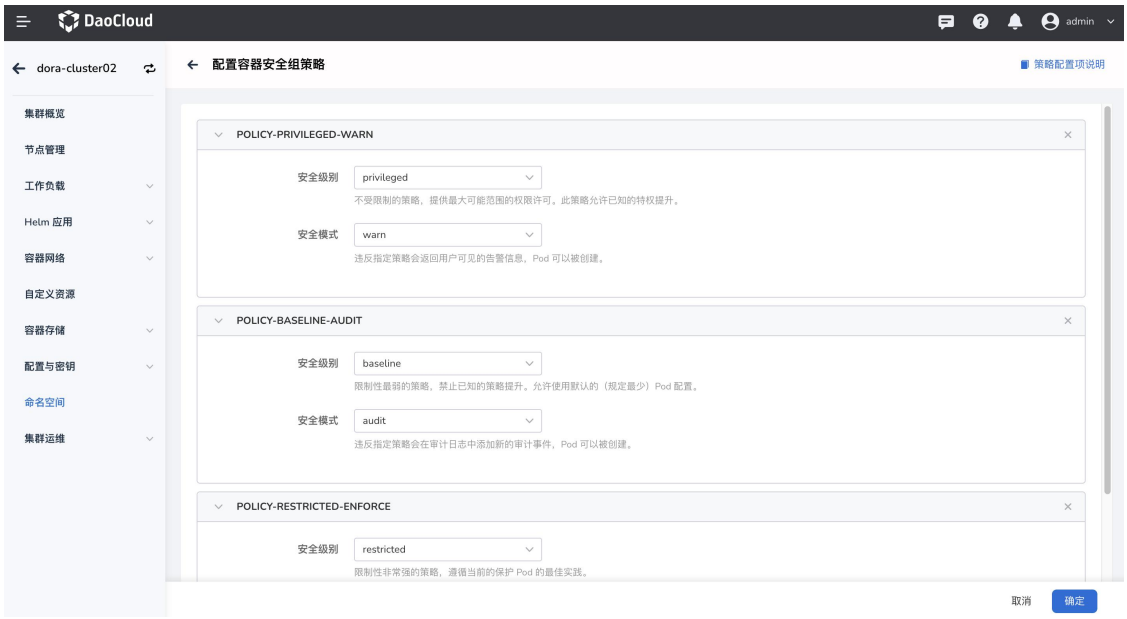
配置策略列表

2. 在配置页点击添加策略，则会出现一条策略，包括安全级别和安全模式，以下是对安全级别和安全策略的详细介绍。

安全级别	描述
Privileged	不受限制的策略，提供最大可能范围的权限许可。此策略允许已知的特权提升。
Baseline	限制性最弱的策略，禁止已知的策略提升。允许使用默认的（规定最少）Pod 配置。
Restricted	限制性非常强的策略，遵循当前的保护 Pod 的最佳实践。

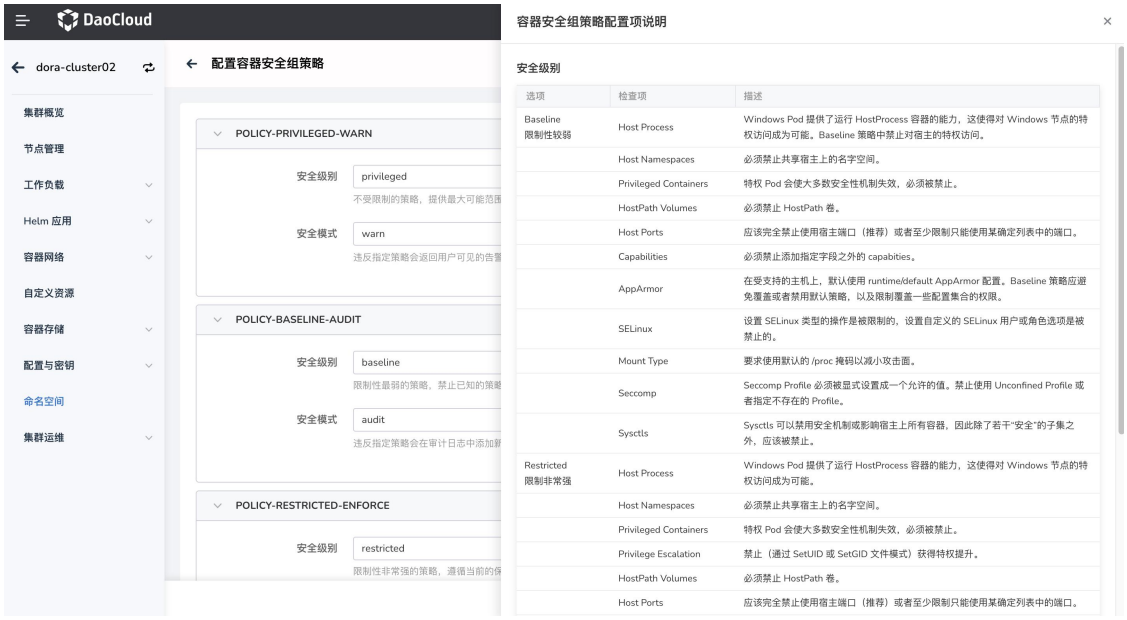
安全模式	描述
Audit	违反指定策略会在审计日志中添加新的审计事件，Pod 可以被创建。
Warn	违反指定策略会返回用户可见的告警信息，Pod 可以被创建。

安全模式	描述
Enforce	违反指定策略会导致 Pod 无法创建。



添加策略

3. 不同的安全级别对应不同的检查项，若您不知道该如何为您的命名空间配置，可以点击页面右上角的策略配置项说明查看详细信息。

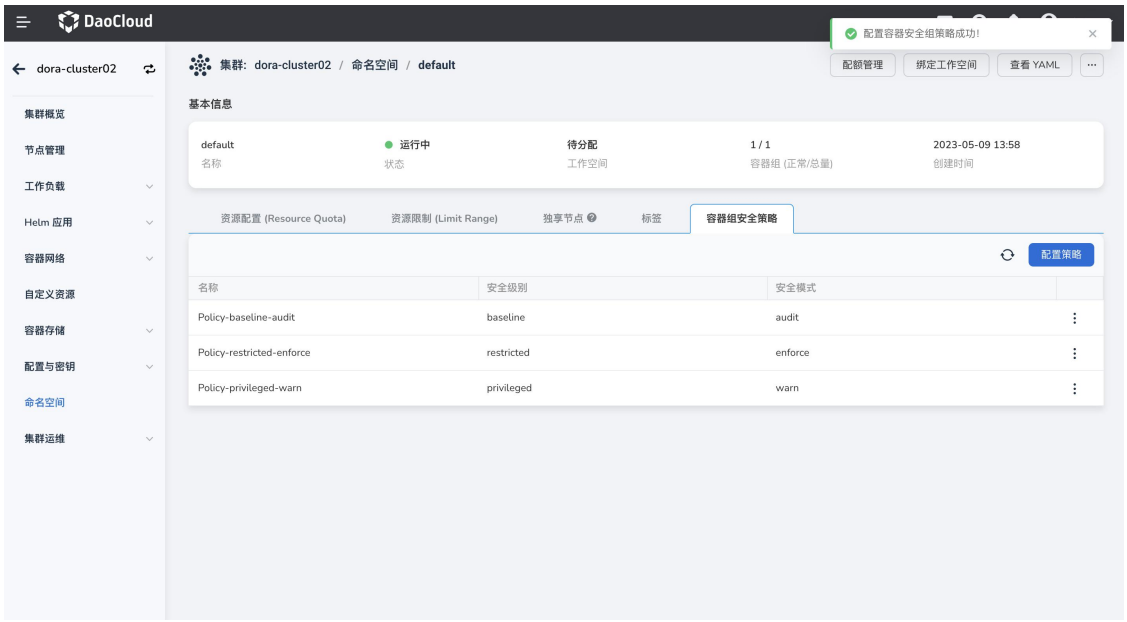


配置项说明 01



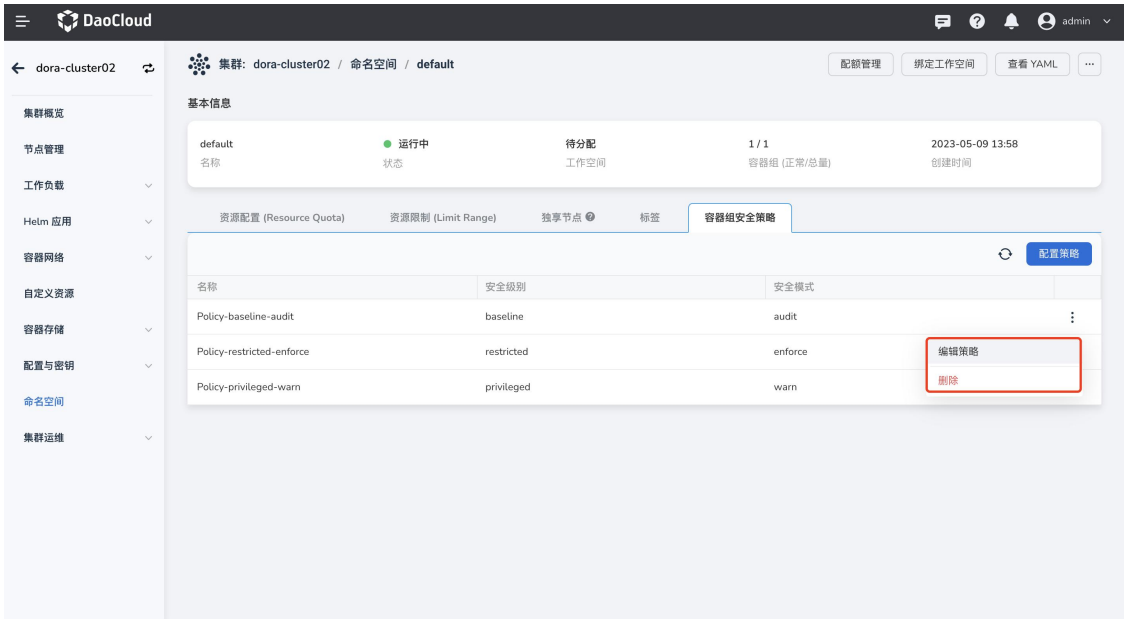
配置项说明 01

4. 点击确定，若创建成功，则页面上将出现您配置的安全策略。



创建成功

5. 点击操作还可以编辑或者删除您配置的安全策略。



操作

命名空间

命名空间是 Kubernetes 中用来进行资源隔离的一种抽象。一个集群下可以包含多个不重名的命名空间，每个命名空间中的资源相互隔离。有关命名空间的详细介绍，可参考命名空间。

本文将介绍命名空间的相关操作。

创建命名空间

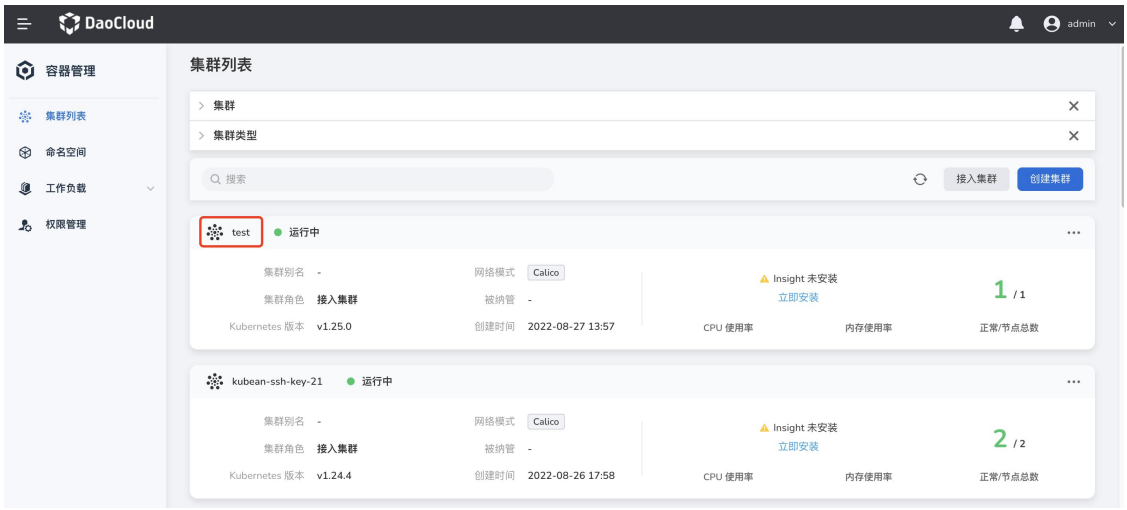
支持通过表单轻松创建命名空间，也支持通过编写或导入 YAML 文件快速创建命名空间。

!!! note

– 在创建命名空间之前，需要在容器管理模块[接入 Kubernetes 集群](../clusters/integrate-cluster.md)或者[创建 Kubernetes 集群](../clusters/create-cluster.md)。
– 集群初始化后通常会自动生成默认的命名空间 `default`。但对于生产集群而言，为便于管理，建议创建其他的命名空间，而非直接使用 `default` 命名空间。

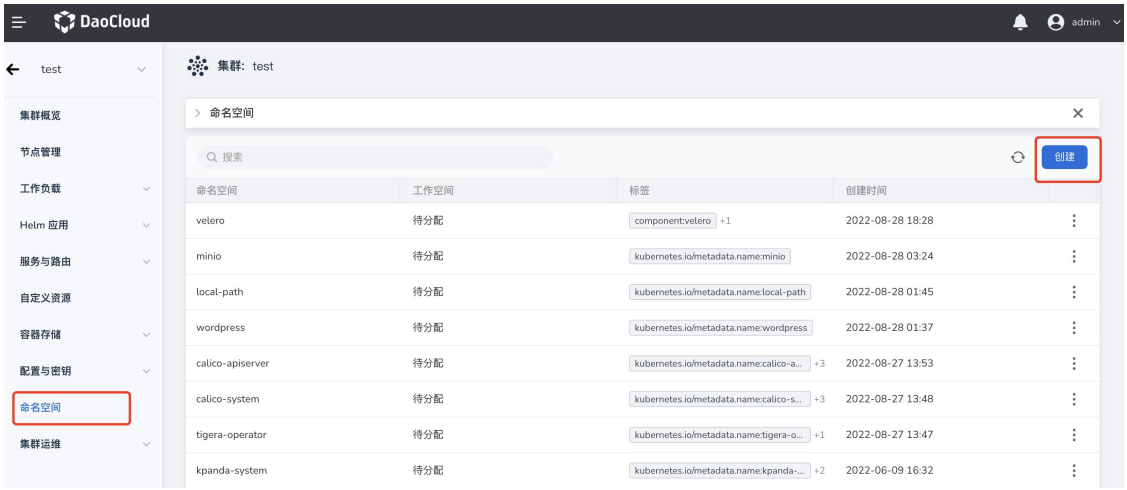
表单创建

1. 在集群列表页面点击目标集群的名称。



集群详情

2. 在左侧导航栏点击命名空间，然后点击页面右侧的创建按钮。



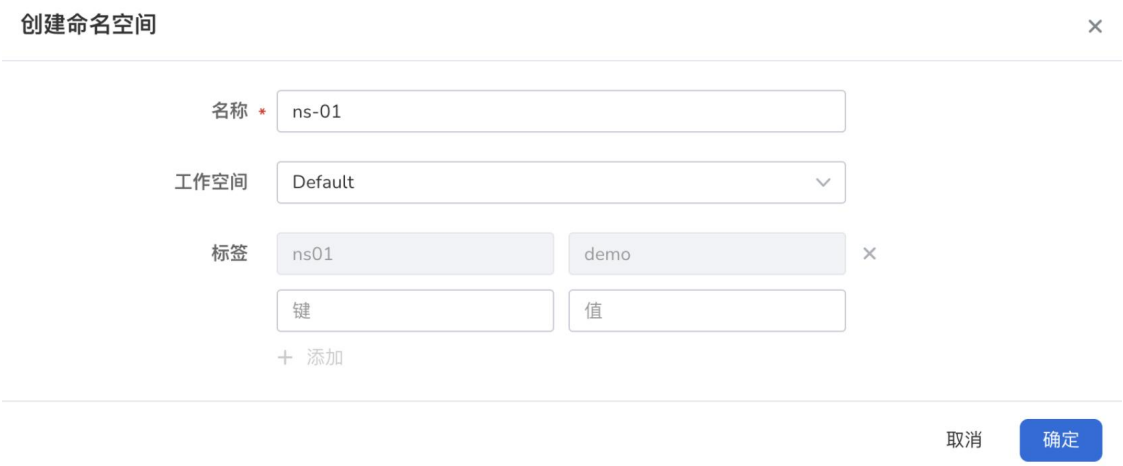
点击创建

3. 填写命名空间的名称，配置工作空间和标签（可选设置），然后点击确定。

   !!! info

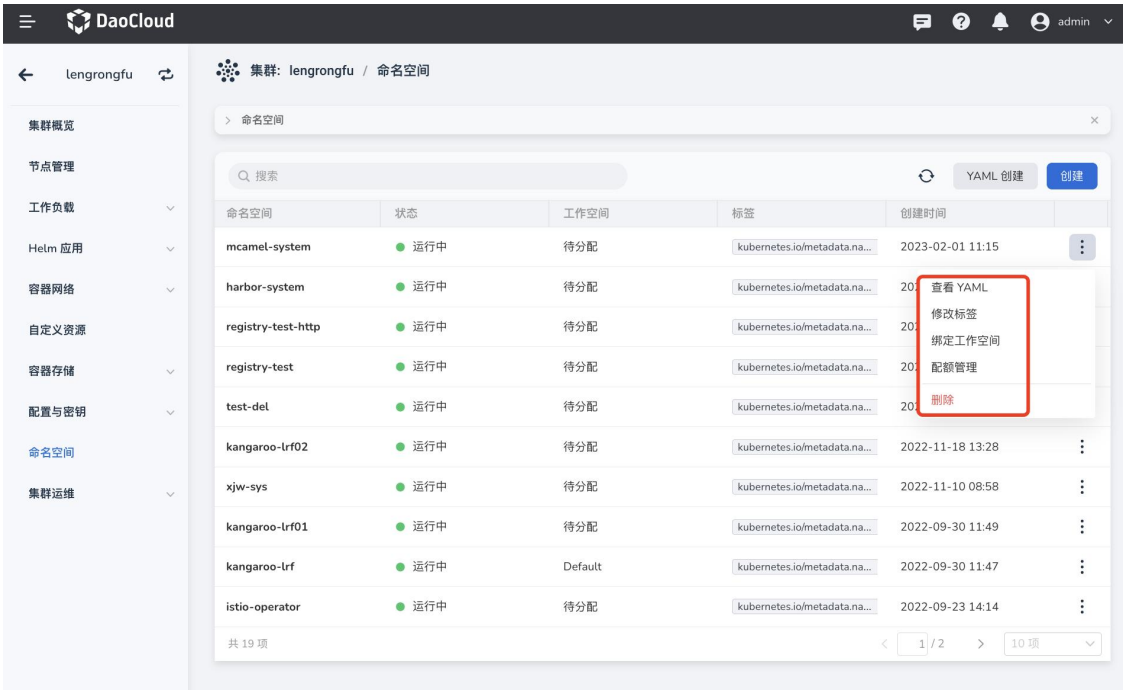
   – 命名空间绑定工作空间之后，该命名空间的资源就会共享给所绑定的工作空间。有关工作空间的详细说明，可参考[工作空间与层级](../../../ghippo/user-guide/workspace/workspace.md)。

– 命名空间创建完成后，仍然可以绑定/解绑工作空间。



填写表单

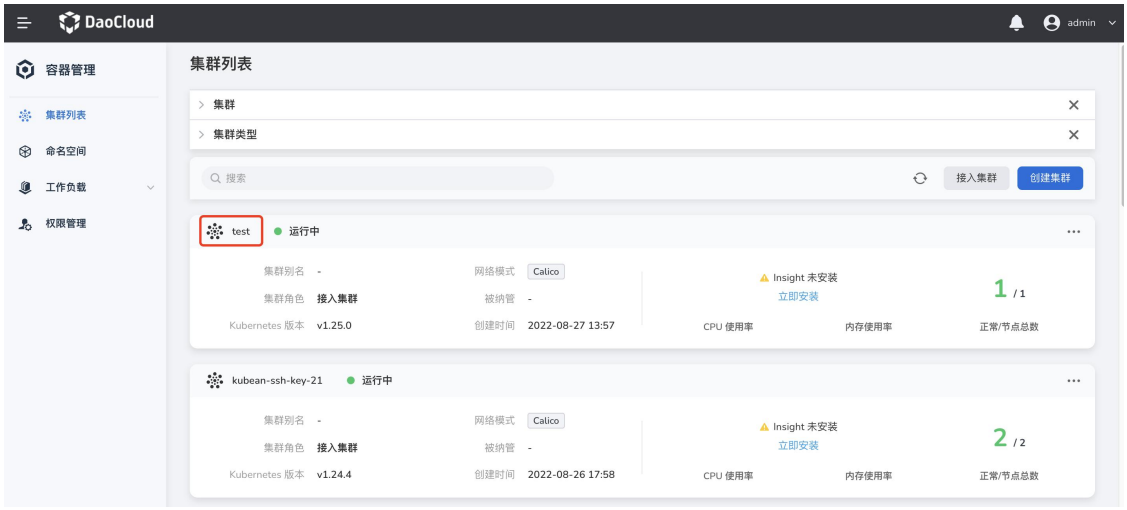
4. 点击确定，完成命名空间的创建。在命名空间列表右侧，点击 ⋮，可以从弹出菜单中选择更新、绑定/解绑工作空间、配额管理、删除等更多操作。



更多操作

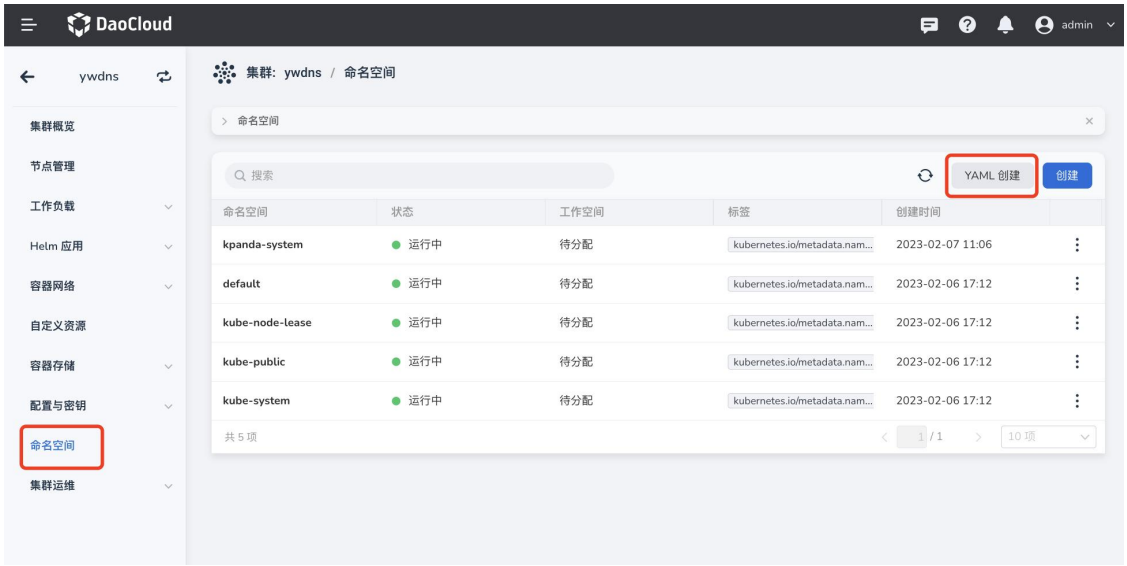
YAML 创建

1. 在集群列表页面点击目标集群的名称。



集群详情

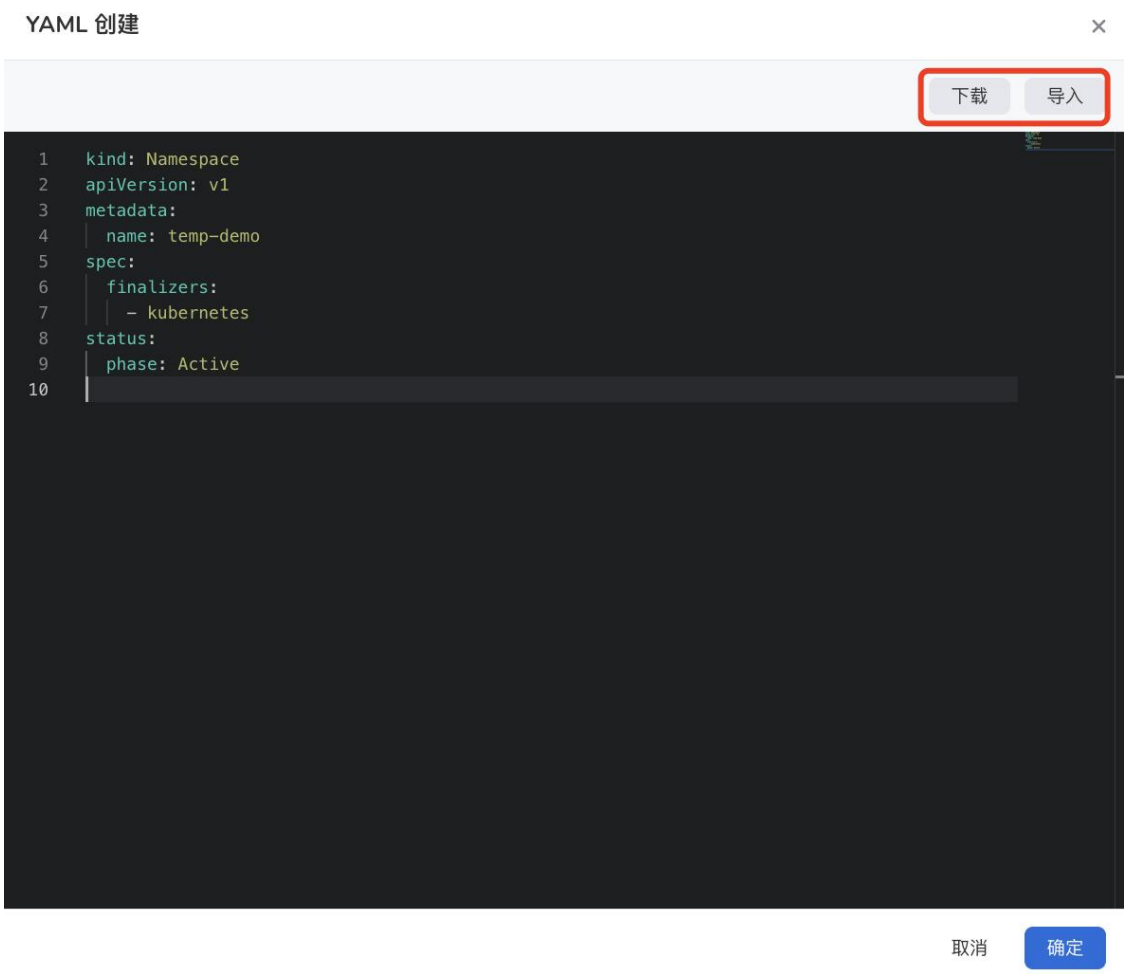
2. 在左侧导航栏点击命名空间，然后点击页面右侧的 YAML 创建按钮。



点击创建

3. 输入或粘贴事先准备好的 YAML 内容，或者从本地直接导入已有的 YAML 文件。

   输入 YAML 内容后，点击下载可以将该 YAML 文件保存到本地。



点击创建

4. 最后在弹框右下角点击确定即可。

启用命名空间独享节点

命名空间独享节点指在 kubernetes 集群中，通过污点和污点容忍的方式实现特定命名空间对一个或多个节点 CPU、内存等资源的独享。为特定命名空间配置独享节点后，其它非此命名空间的应用和服务均不能运行在被独享的节点上。使用独享节点可以让重要应用独享一部分计算资源，从而和其他应用实现物理隔离。

!!! note

在节点被设置为独享节点前已经运行在此节点上的应用和服务将不会受影响，依然会正常运行在该节点上，仅当这些 Pod 被删除或重建时，才会调度到其它非独享节点上。

准备工作

检查当前集群的 kube-apiserver 是否启用了 PodNodeSelector 和 PodTolerationRestriction 准入控制器。

使用命名空间独享节点功能需要用户启用 kube-apiserver 上的 PodNodeSelector 和 PodTolerationRestriction 两个特性准入控制器（Admission Controllers），关于准入控制器更多说明请参阅 kubernetes Admission Controllers Reference。

您可以前往当前集群下任意一个 Master 节点上检查 kube-apiserver.yaml 文件内是否启用了这两个特性，也可以在 Master 节点上执行执行如下命令进行快速检查：

```bash
[root@g-master1 ~]# cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep enable-admission-plugins

# 预期输出如下：
- --enable-admission-plugins=NodeRestriction,PodNodeSelector,PodTolerationRestriction
```

在 Global 集群上启用命名空间独享节点

由于 Global 集群上运行着 kpanda、ghippo、insight 等平台基础组件，在 Global 启用命名空间独享节点将可能导致当系统组件重启后，系统组件无法调度到被独享的节点上，影响系统的整体高可用能力。因此，**通常情况下，我们不推荐用户在 Global 集群上启用命名空间独享节点特性**。

如果您确实需要在 Global 集群上启用命名空间独享节点，请参考以下步骤进行开启：

1. 为 Global 集群的 kube-apiserver 启用了 PodNodeSelector 和 PodTolerationRestriction 准入控制器

   !!! note

   如果集群已启用了上述的两个准入控制器，请跳过此步，直接前往配置系统组件容忍。

   前往当前集群下任意一个 Master 节点上修改 kube-apiserver.yaml 配置文件，也可以在 Master 节点上执行执行如下命令进行配置：

   ```
   [root@g-master1 ~]# vi /etc/kubernetes/manifests/kube-apiserver.yaml

   # 预期输出如下：
   apiVersion: v1
   kind: Pod
   metadata:
       ......
   spec:
   containers:
   - command:
       - kube-apiserver
       ......
       - --default-not-ready-toleration-seconds=300
   ```

```
    - --default-unreachable-toleration-seconds=300
    - --enable-admission-plugins=NodeRestriction    #启用的准入控制器列表
    - --enable-aggregator-routing=False
    - --enable-bootstrap-token-auth=true
    - --endpoint-reconciler-type=lease
    - --etcd-cafile=/etc/kubernetes/ssl/etcd/ca.crt
    ......
```

找到 --enable-admission-plugins 参数，加入（以英文逗号分隔的）PodNodeSelector 和 PodTolerationRestriction 准入控制器。参考如下：

```
# 加入`,PodNodeSelector,PodTolerationRestriction`
- --enable-admission-plugins=NodeRestriction,PodNodeSelector,PodTolerationRestriction
```

2. 为平台组件所在的命名空间添加容忍注解

   完成准入控制器的开启后，您需要为平台组件所在的命名空间添加容忍注解，以保证平台组件的高可用。

   目前 DCE 5.0 的系统组件命名空间如下表：

命名空间	所包含的系统组件
kpanda-system	kpanda
hwameiStor-system	hwameiStor
istio-system	istio
metallb-system	metallb
cert-manager-system	cert-manager
contour-system	contour

命名空间	所包含的系统组件
kubean-system	kubean
ghippo-system	ghippo
kcoral-system	kcoral
kcollie-system	kcollie
insight-system	insight、insight-agent:
ipavo-system	ipavo
kairship-system	kairship
karmada-system	karmada
amamba-system	amamba、jenkins
skoala-system	skoala
mspider-system	mspider
mcamel-system	mcamel-rabbitmq、mcamel-elasticsearch、mcamel-mysql、mcamel-redis、mcamel-kafka、mcamel-minio、mcamel-postgresql
spidernet-system	spidernet
kangaroo-system	kangaroo
gmagpie-system	gmagpie
dowl-system	dowl

检查当前集群中所有命名空间是否存在上述的命名空间，执行如下命令，分别为每个命名空间添加注解：

scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "effect": "NoSchedule", "key": "ExclusiveNamespace"}]'。

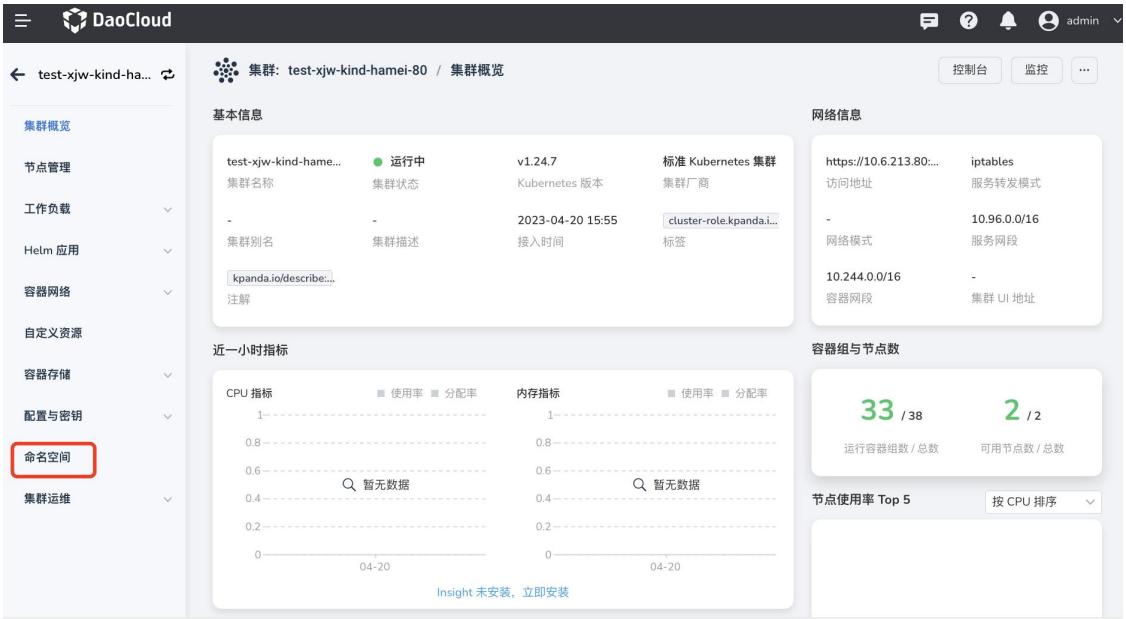
kubectl annotate ns <namespace-name> scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "effect": "NoSchedule", "key": "ExclusiveNamespace"}]'

请确保将 <namespace-name> 替换为要添加注解的平台命名空间名称。

3. 使用界面为命名空间设置独享节点

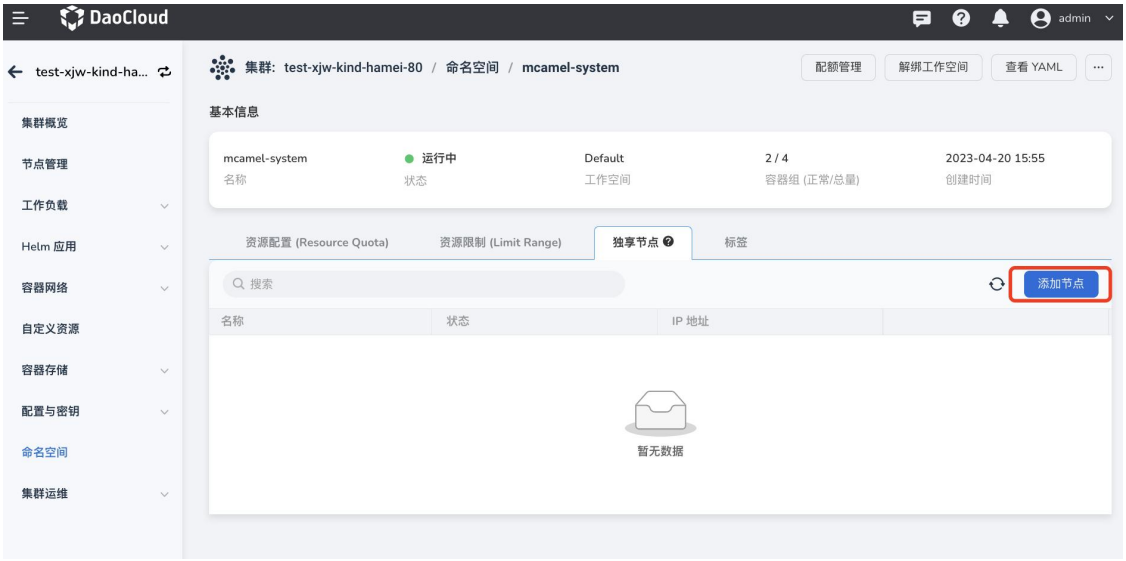
   当您确认集群 API 服务器上的 PodNodeSelector 和 PodTolerationRestriction 两个特性准入控制器已经开启后，请参考如下步骤使用 DCE 5.0 的 UI 管理界面为命名空间设置独享节点了。

   1. 在集群列表页面点击集群名称，然后在左侧导航栏点击命名空间。



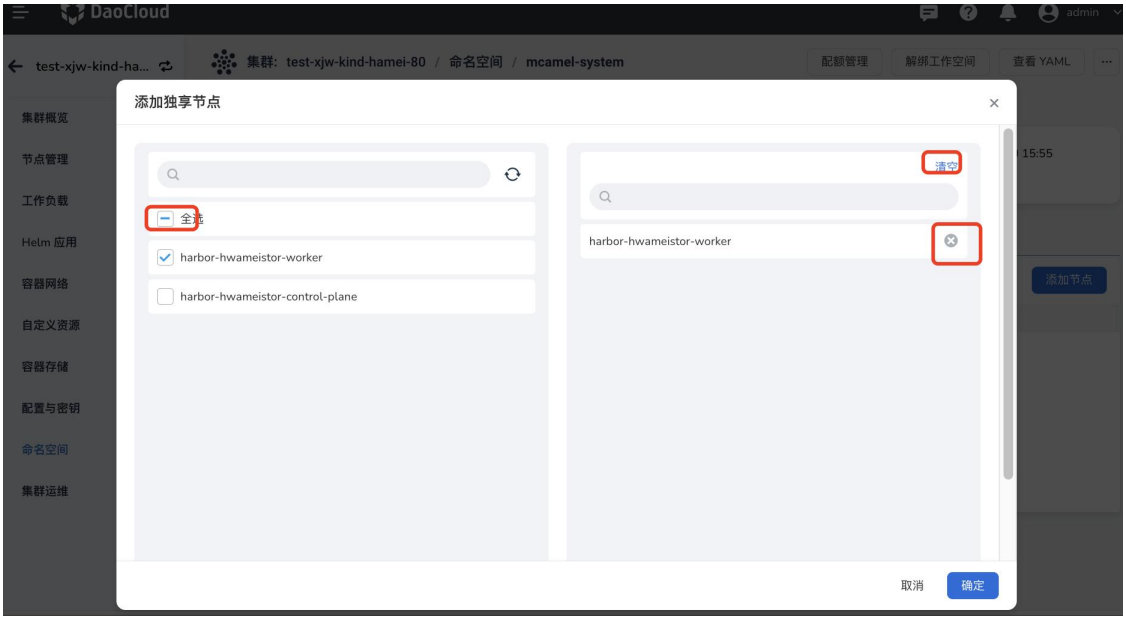
   命名空间

   2. 点击命名空间名称，然后点击独享节点 页签，在下方右侧点击添加节点。



命名空间

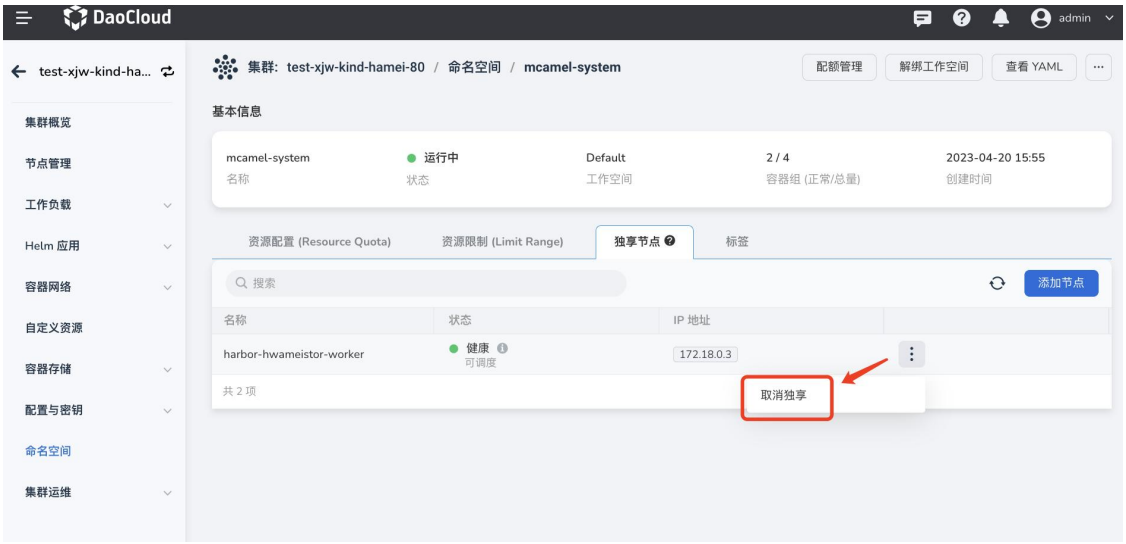
3. 在页面左侧选择让该命名空间独享哪些节点，在右侧可以清空或删除某个已选节点，最后在底部点击确定。



命名空间

4. 可以在列表中查看此命名空间的已有的独享节点，在节点右侧可以选择取消独享。

取消独享之后，其他命名空间下的 Pod 也可以被调度到该节点上。



命名空间

在 非 Global 集群上启用命名空间独享节点

在 非 Global 集群上启用命名空间独享节点，请参考以下步骤进行开启：

1. 为当前集群的 kube-apiserver 启用了 PodNodeSelector 和 PodTolerationRestriction 准入控制器

   !!! note

   如果集群已启用了上述的两个准入控制器，请跳过此步，直接前往界面为命名空间设置独享节点

   前往当前集群下任意一个 Master 节点上修改 kube-apiserver.yaml 配置文件，也可以在 Master 节点上执行执行如下命令进行配置：

   [root@g-master1 ~]# vi /etc/kubernetes/manifests/kube-apiserver.yaml

   # 预期输出如下：

   apiVersion: v1

```
kind: Pod
metadata:
    ......
spec:
containers:
- command:
    - kube-apiserver
    ......
    - --default-not-ready-toleration-seconds=300
    - --default-unreachable-toleration-seconds=300
    - --enable-admission-plugins=NodeRestriction    #启用的准入控制器列
表
    - --enable-aggregator-routing=False
    - --enable-bootstrap-token-auth=true
    - --endpoint-reconciler-type=lease
    - --etcd-cafile=/etc/kubernetes/ssl/etcd/ca.crt
    ......
```

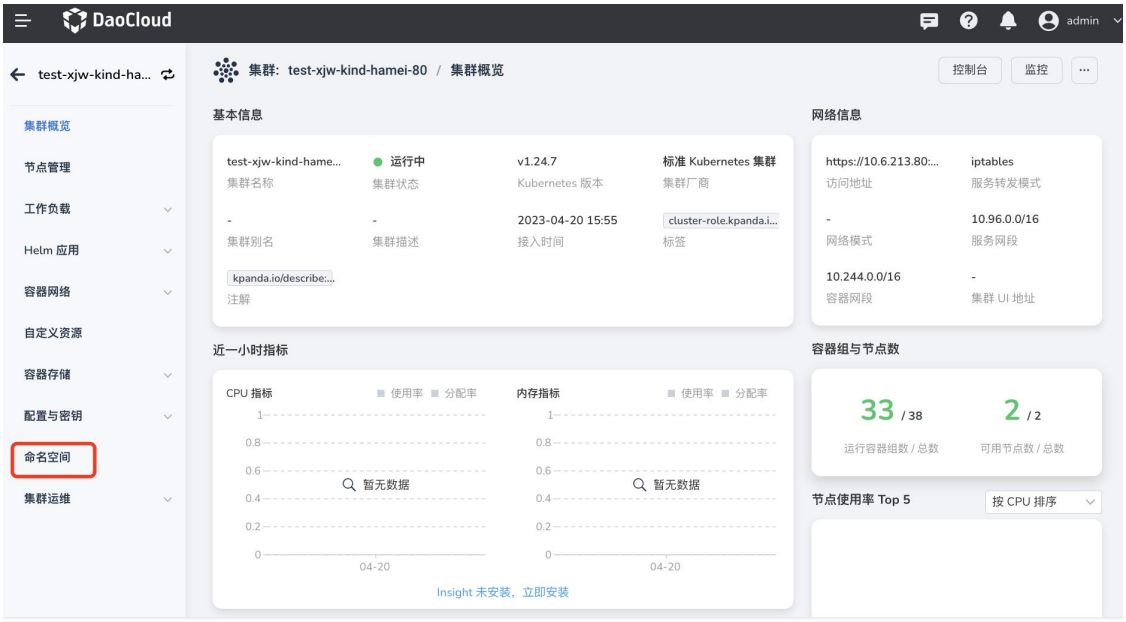
找到 --enable-admission-plugins 参数，加入（以英文逗号分隔的）PodNodeSelector 和 PodTolerationRestriction 准入控制器。参考如下：

```
# 加入`,PodNodeSelector,PodTolerationRestriction`
- --enable-admission-plugins=NodeRestriction,PodNodeSelector,PodTolera
tionRestriction
```

2. 使用界面为命名空间设置独享节点

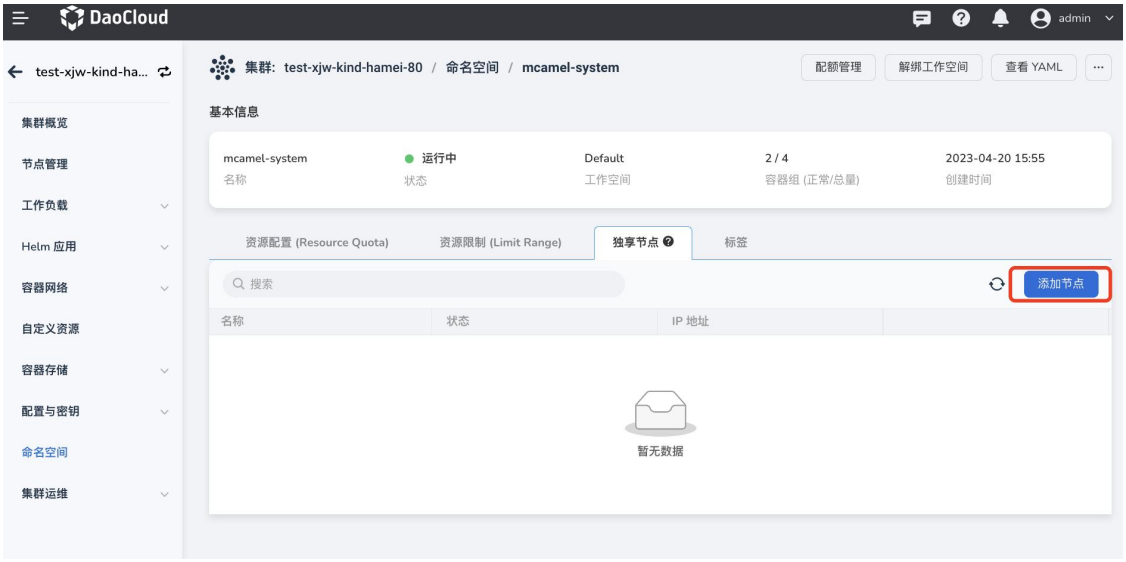
当您确认集群 API 服务器上的 PodNodeSelector 和 PodTolerationRestriction 两个特性准入控制器已经开启后，请参考如下步骤使用 DCE 5.0 的 UI 管理界面为命名空间设置独享节点了。

1. 在集群列表页面点击集群名称，然后在左侧导航栏点击命名空间。



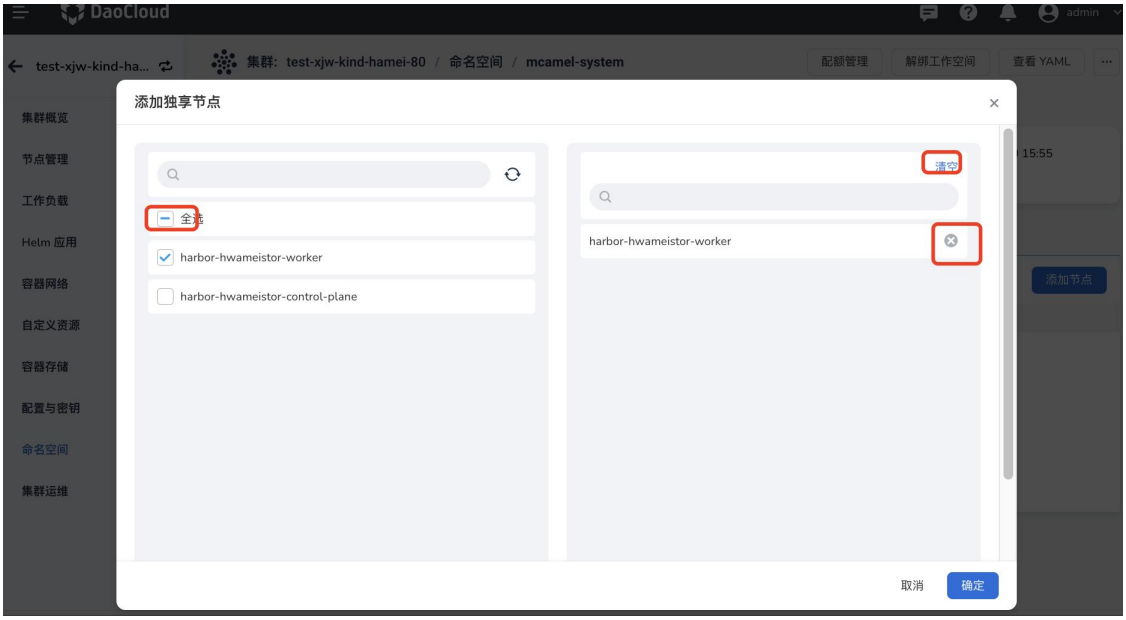
命名空间

2. 点击命名空间名称，然后点击独享节点 页签，在下方右侧点击添加节点。



命名空间

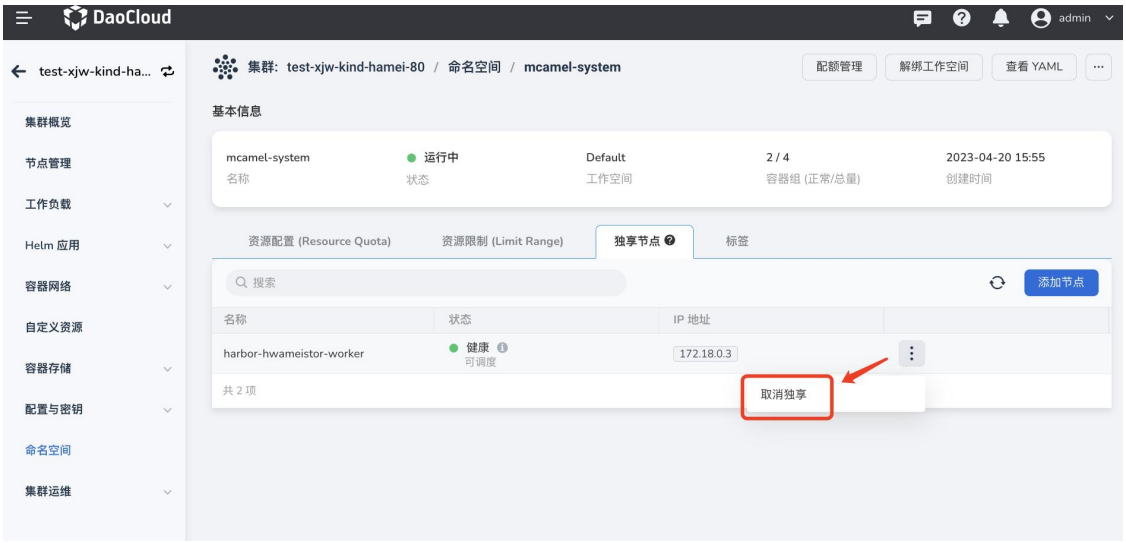
3. 在页面左侧选择让该命名空间独享哪些节点，在右侧可以清空或删除某个已选节点，最后在底部点击确定。



命名空间

4. 可以在列表中查看此命名空间的已有的独享节点，在节点右侧可以选择取消独享。

   取消独享之后，其他命名空间下的 Pod 也可以被调度到该节点上。



命名空间

3. 为需要高可用的组件所在的命名空间添加容忍注解（可选）

执行如下命令，需要高可用的组件所在的命名空间添加注解：scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "effect": "NoSchedule", "key": "ExclusiveNamespace"}]'。

kubectl annotate ns <namespace-name> scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "effect": "NoSchedule", "key": "ExclusiveNamespace"}]'

请确保将 <namespace-name> 替换为要添加注解的平台命名空间名称。

创建无状态负载（Deployment）

本文介绍如何通过镜像和 YAML 文件两种方式创建无状态负载。

无状态负载（Deployment）是 Kubernetes 中的一种常见资源，主要为 Pod 和 ReplicaSet 提供声明式更新，支持弹性伸缩、滚动升级、版本回退等功能。在 Deployment 中声明期望的 Pod 状态，Deployment Controller 会通过 ReplicaSet 修改当前状态，使其达到预先声明的期望状态。Deployment 是无状态的，不支持数据持久化，适用于部署无状态的、不需要保存数据、随时可以重启回滚的应用。

通过 DCE 5.0 的容器管理模块，可以基于相应的角色权限轻松管理多云多集群上的工作负载，包括对无状态负载的创建、更新、删除、弹性扩缩、重启、版本回退等全生命周期管理。

前提条件

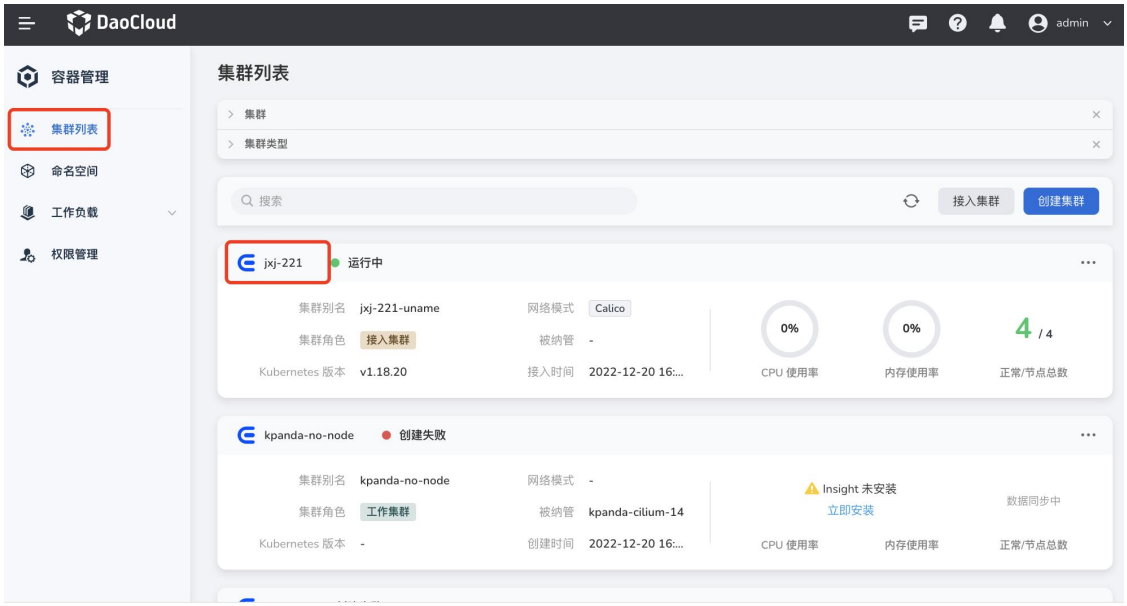
在使用镜像创建无状态负载之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

镜像创建

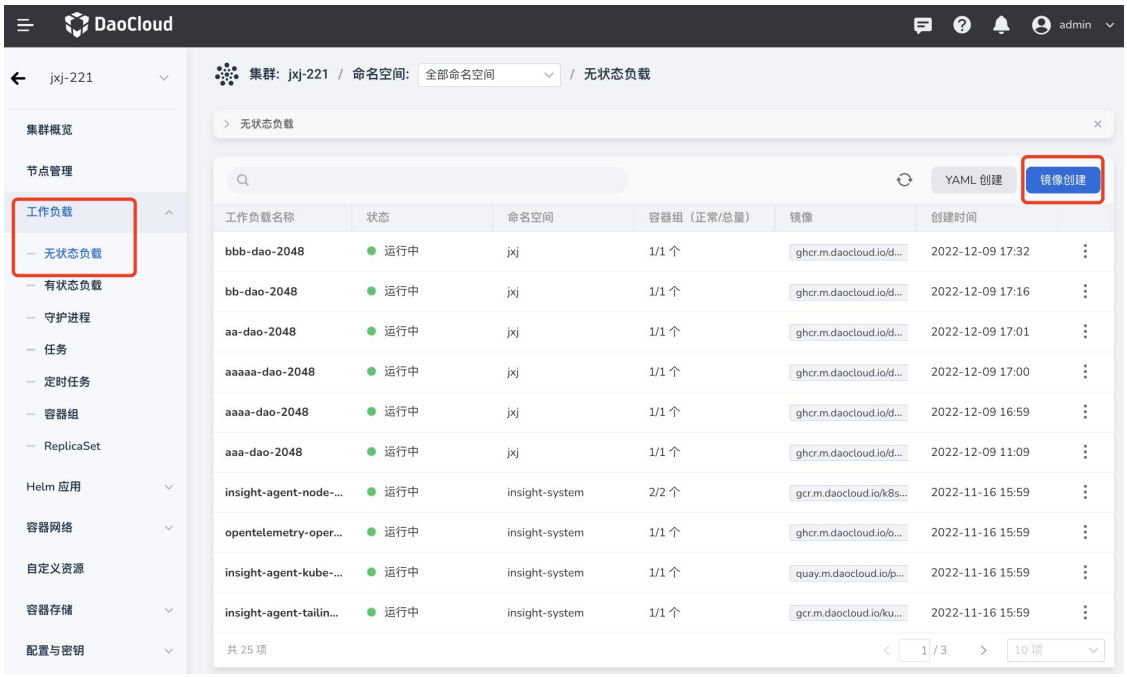
参考以下步骤，使用镜像创建一个无状态负载。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



集群详情

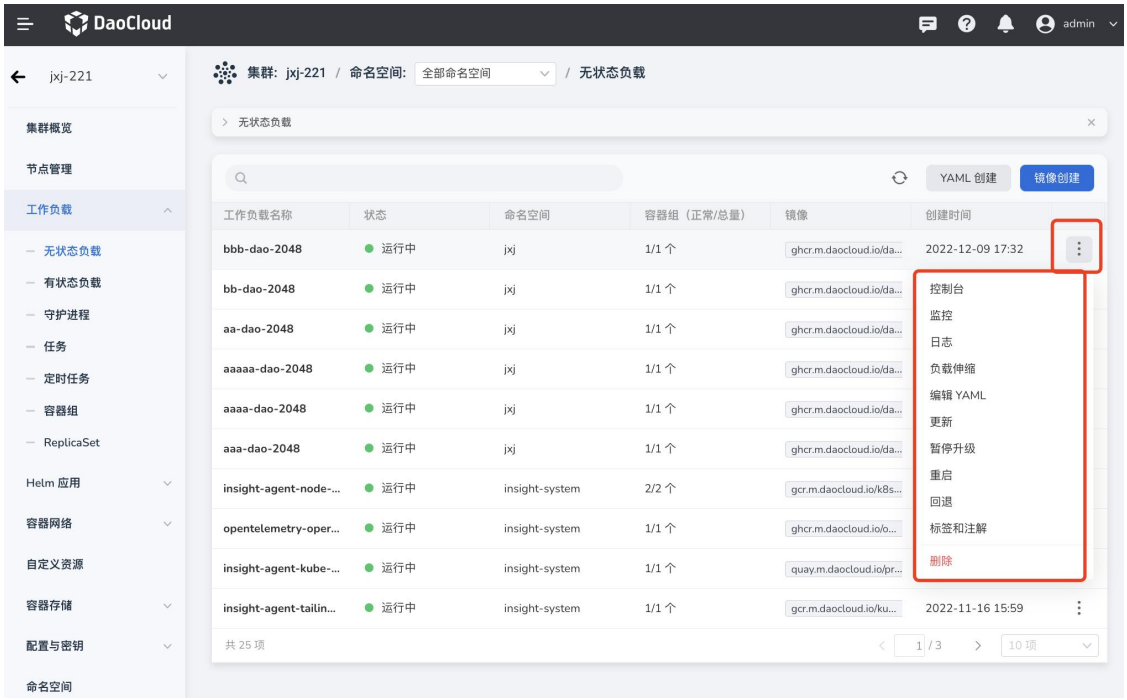
2. 在集群详情页面，点击左侧导航栏的工作负载 –> 无状态负载，然后点击页面右上角的镜像创建按钮。



工作负载

3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击确定完成创建。

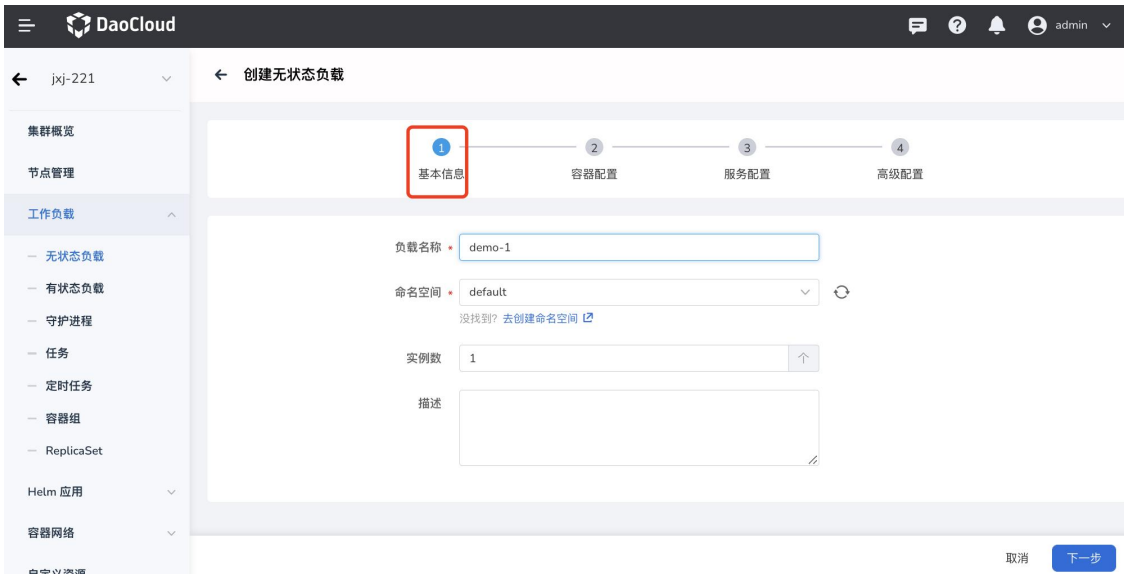
   系统将自动返回无状态负载列表。点击列表右侧的 ：，可以对负载执行执行更新、删除、弹性扩缩、重启、版本回退等操作。如果负载状态出现异常，请查看具体异常信息，可参考工作负载状态。



操作菜单

基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 deployment-01。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 命名空间：选择将新建的负载部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的命名空间。
- 实例数：输入负载的 Pod 实例数量，默认创建 1 个 Pod 实例。
- 描述：输入负载的描述信息，内容自定义。字符数不超过 512。



基本信息

容器配置

容器配置分为基本信息、生命周期、健康检查、环境变量、数据存储、安全设置六部分，点击下方的相应页签可查看各部分的配置要求。

> 容器配置仅针对单个容器进行配置，如需在一个容器组中添加多个容器，可点击右侧的 + 添加多个容器。

===“基本信息（必填）”

在配置容器相关参数时，必须正确填写容器的名称、镜像参数，否则将无法进入下一步。参考以下要求填写配置后，点击`确认`。

– 容器名称：最多包含 63 个字符，支持小写字母、数字及分隔符（“–”）。必须以小写字母或数字开头及结尾，例如 nginx–01。
– 容器镜像：输入镜像地址或名称。输入镜像名称时，默认从官方的 [DockerHub](https://hub.docker.com/）拉取镜像。接入 DCE 5.0 的[镜像仓库](../../../kangaroo/index.md)模块后，可以点击右侧的`选择镜像`来选择镜像。
– 更新策略：勾选`总是拉取镜像`后，负载每次重启/升级时都会从仓库重新拉取镜像。如果不勾选，则只拉取本地镜像，只有当镜像在本地不存在时才从镜像仓库重新拉取。更多详情可参考[镜像拉取策略](https://kubernetes.io/zh–cn/docs/concepts/containers/i

mages/#image-pull-policy)。
- 特权容器：默认情况下，容器不可以访问宿主机上的任何设备，开启特权容器后，容器即可访问宿主机上的所有设备，享有宿主机上的运行进程的所有权限。
- CPU/内存配额：CPU/内存资源的请求值（需要使用的最小资源）和限制值（允许使用的最大资源）。请根据需要为容器配置资源，避免资源浪费和因容器资源超额导致系统故障。默认值如图所示。
- GPU 独享：为容器配置 GPU 用量，仅支持输入正整数。GPU 配额设置支持为容器设置独享整张 GPU 卡或部分 vGPU。例如，对于一张 8 核心的 GPU 卡，输入数字 `8` 表示让容器独享整长卡，输入数字 `1` 表示为容器配置 1 核心的 vGPU。

> 设置 GPU 独享之前，需要管理员预先在集群节点上安装 GPU 卡及驱动插件，并在[集群设置](../clusterops/cluster-settings.md)中开启 GPU 特性。

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy05.png)

===“生命周期（选填）”

设置容器启动时、启动后、停止前需要执行的命令。详情可参考[容器生命周期配置](pod-config/lifecycle.md)。

![生命周期](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy06.png)

===“健康检查（选填）”

用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考[容器健康检查配置](pod-config/health-check.md)。

![健康检查](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy07.png)

===“环境变量（选填）”

配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考[容器环境变量配置](pod-config/env-variables.md)。

![环境变量](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy08.png)

===“数据存储（选填）”

配置容器挂载数据卷和数据持久化的设置。详情可参考[容器数据存储配置](pod-config/env-variables.md)。

![数据存储](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy09.png)

===“安全设置（选填）”

通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 `0` 表示使用 root 账号的权限。

![安全设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy10.png)

服务配置

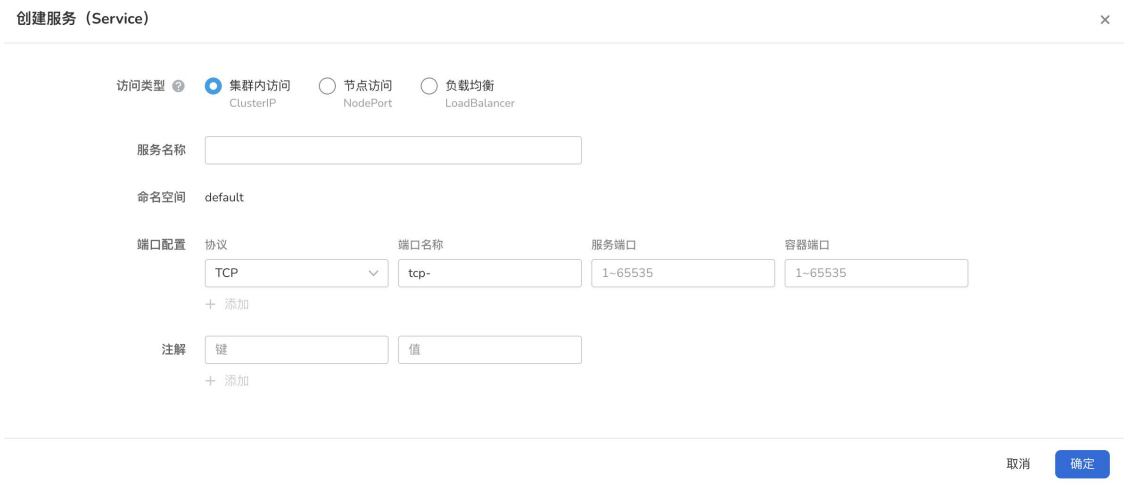
为无状态负载配置服务（Service），使无状态负载能够被外部访问。

1. 点击创建服务按钮。



服务配置

2. 参考创建服务，配置服务参数。



创建服务

3. 点击确定，点击下一步。

高级配置

高级配置包括负载的网络配置、升级策略、调度策略、标签与注解四部分，可点击下方的页签查看各部分的配置要求。

===“网络配置”

– 如在集群中部署了 [SpiderPool](../../../network/modules/spiderpool/index.md) 和 [Multus](../../../network/modules/multus-underlay/index.md) 组件，则可以在网络配置中配置容器网卡。详情参考[工作负载使用 IP 池](../../../network/modules/spiderpool/usage.md)。

– DNS 配置：应用在某些场景下会出现冗余的 DNS 查询。Kubernetes 为应用提供了与 DNS 相关的配置选项，能够在某些场景下有效地减少冗余的 DNS 查询，提升业务并发量。

– DNS 策略

  – Default：使容器使用 kubelet 的 `--resolv-conf` 参数指向的域名解析文件。该配置只能解析注册到互联网上的外部域名，无法解析集群内部域名，且不存在无效的 DNS 查询。
  – ClusterFirstWithHostNet：应用对接主机的域名文件。
  – ClusterFirst：应用对接 Kube-DNS/CoreDNS。
  – None：Kubernetes v1.9（Beta in v1.10）中引入的新选项值。设置为 None 之后，必须设置 dnsConfig，此时容器的域名解析文件将完全通过 dnsConfig 的配置来生成。

– 域名服务器：填写域名服务器的地址，例如 `10.6.175.20`。
– 搜索域：域名查询时的 DNS 搜索域列表。指定后，提供的搜索域列表将合并到基于 dnsPolicy 生成的域名解析文件的 search 字段中，并删除重复的域名。Kubernetes 最多允许 6 个搜索域。
– Options：DNS 的配置选项，其中每个对象可以具有 name 属性（必需）和 value 属性（可选）。该字段中的内容将合并到基于 dnsPolicy 生成的域名解析文件的 options 字段中，dnsConfig 的 options 的某些选项如果与基于 dnsPolicy 生成的域名解析文件的选项冲突，则会被 dnsConfig 所覆盖。
– 主机别名：为主机设置的别名。

![DNS 配置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy17.png)

===“升级策略”

– 升级方式：`滚动升级`指逐步用新版本的实例替换旧版本的实例，升级的过程中，业务流量会同时负载均衡分布到新老的实例上，因此业务不会中断。`重建升级`指先删除老版本的负载实例，再安装指定的新版本，升级过程中业务会中断。
– 最大无效 Pod 数：指定负载更新过程中不可用 Pod 的最大值或比率，默认 25%。如果等于实例数有服务中断的风险。
– 最大浪涌：更新 Pod 的过程中 Pod 总数超过 Pod 期望副本数部分的最大值或比率。默认 25%。
– 最大保留版本数：设置版本回滚时保留的旧版本数量。默认 10。
– Pod 可用最短时间：Pod 就绪的最短时间，只有超出这个时间 Pod 才被认为可用，默认 0 秒。
– 升级最大持续时间：如果超过所设置的时间仍未部署成功，则将该负载标记为失败。默认 600 秒。
– 缩容时间窗：负载停止前命令的执行时间窗（0–9,999 秒），默认 30 秒。

![升级策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy14.png)

===“调度策略”

– 容忍时间：负载实例所在的节点不可用时，将负载实例重新调度到其它可用节点的时间，默认为 300 秒。
– 节点亲和性：根据节点上的标签来约束 Pod 可以调度到哪些节点上。
– 工作负载亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 可以调度到哪些节点。
– 工作负载反亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 不可以调度到哪些节点。
– 拓扑域：即 topologyKey，用于指定可以调度的一组节点。例如，`kubernetes.io/os` 表示只要某个操作系统的节点满足 labelSelector 的条件就可以调度到该节点。

> 具体详情请参考[调度策略](pod-config/scheduling-policy.md)。

![调度策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy15.png)

===“标签与注解”

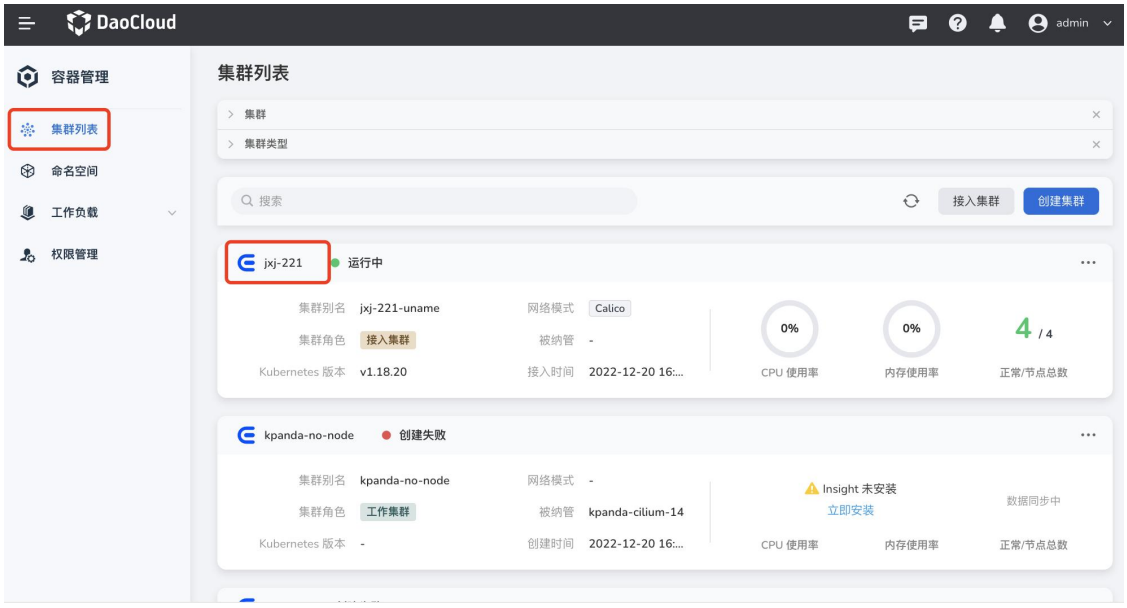
可以点击`添加`按钮为工作负载和容器组添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy16.png)

YAML 创建

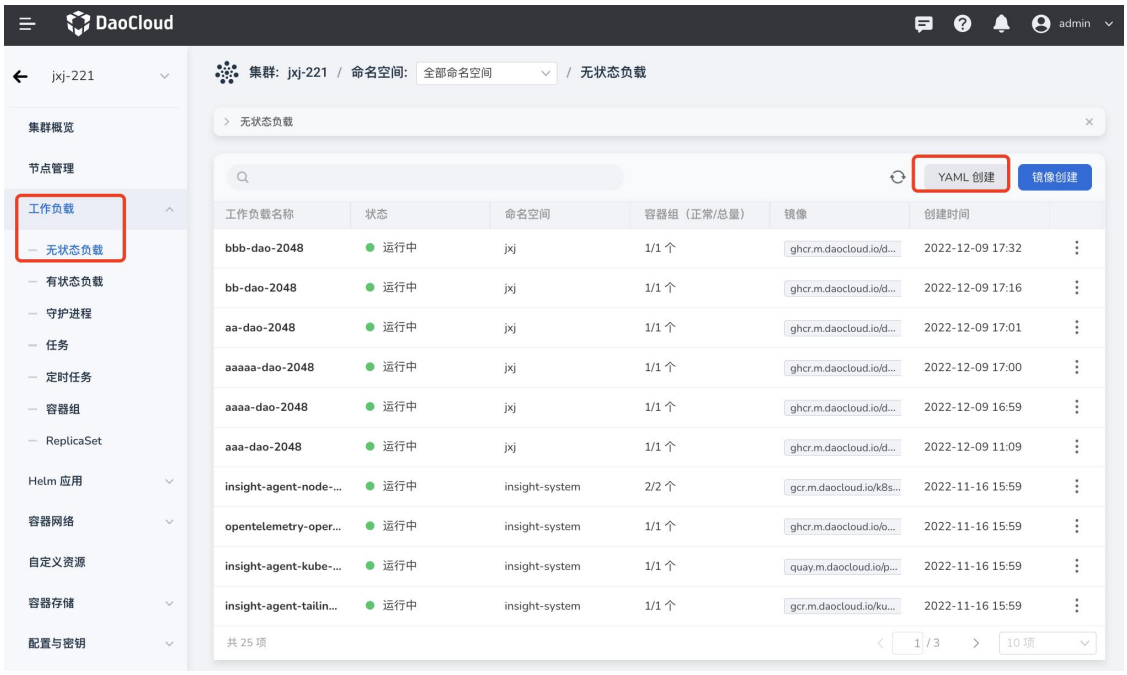
除了通过镜像方式外，还可以通过 YAML 文件更快速地创建创建无状态负载。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



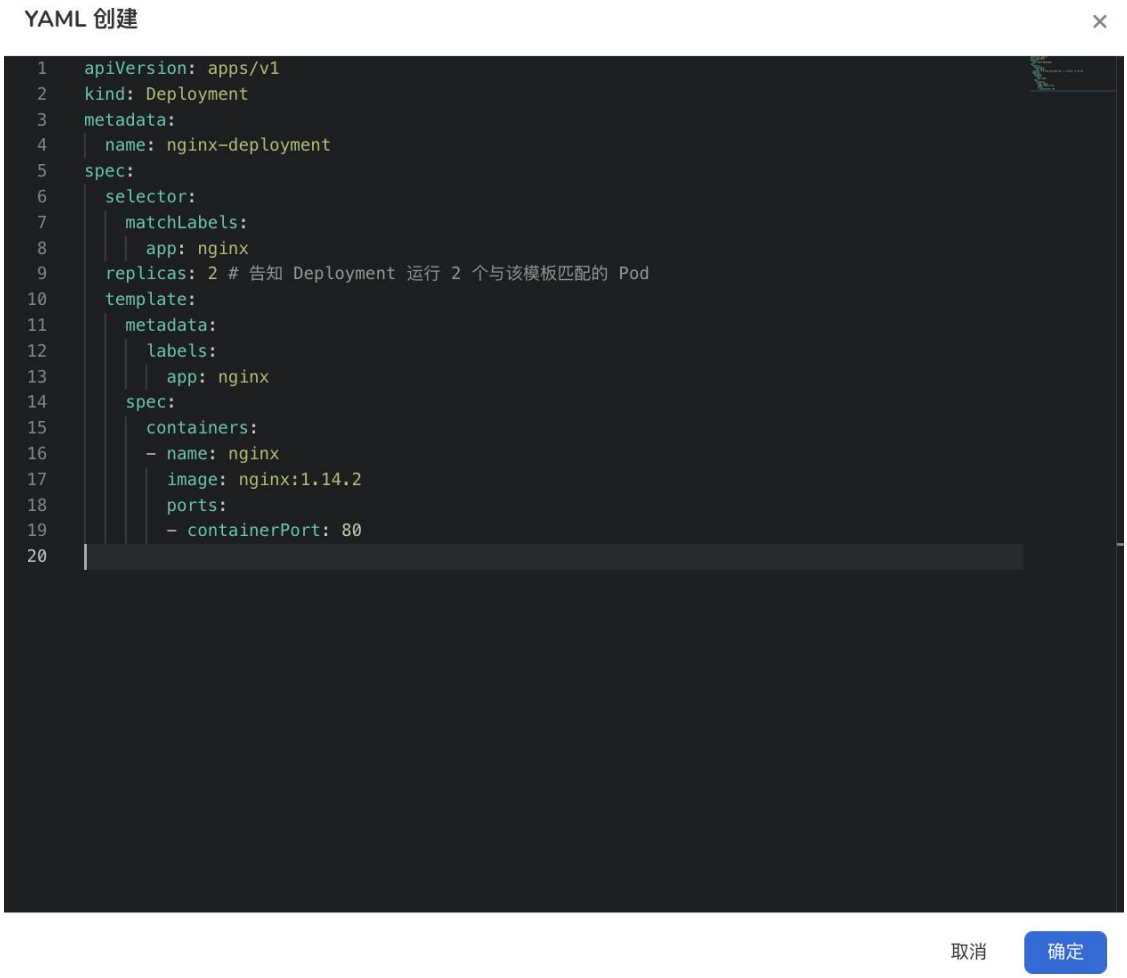
集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载 -> 无状态负载，然后点击页面右上角的 YAML 创建按钮。



工作负载

3. 输入或粘贴事先准备好的 YAML 文件，点击确定即可完成创建。



工作负载

??? note “点击查看创建无状态负载的 YAML 示例”

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # 告知 Deployment 运行 2 个与该模板匹配的 Pod
```

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

创建有状态负载（StatefulSet）

本文介绍如何通过镜像和 YAML 文件两种方式创建有状态负载（StatefulSet）。

有状态负载（StatefulSet）是 Kubernetes 中的一种常见资源，和无状态负载（Deployment）类似，主要用于管理 Pod 集合的部署和伸缩。二者的主要区别在于，Deployment 是无状态的，不保存数据，而 StatefulSet 是有状态的，主要用于管理有状态应用。此外，StatefulSet 中的 Pod 具有永久不变的 ID，便于在匹配存储卷时识别对应的 Pod。

通过 DCE 5.0 的容器管理模块，可以基于相应的角色权限轻松管理多云多集群上的工作负载，包括对有状态工作负载的创建、更新、删除、弹性扩缩、重启、版本回退等全生命周期管理。

前提条件

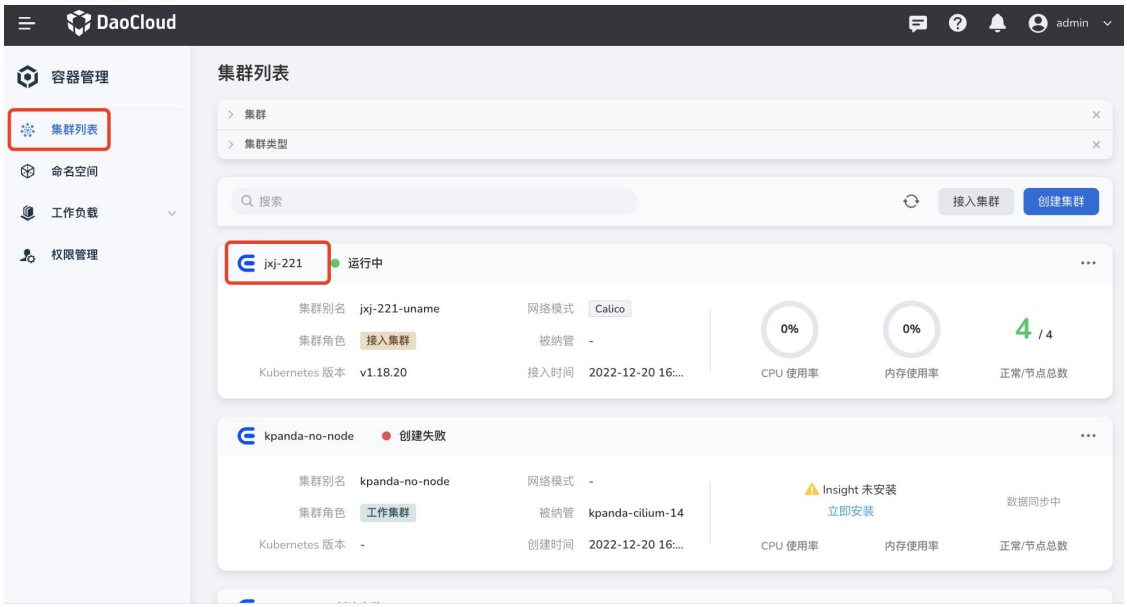
在使用镜像创建有状态负载之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

镜像创建

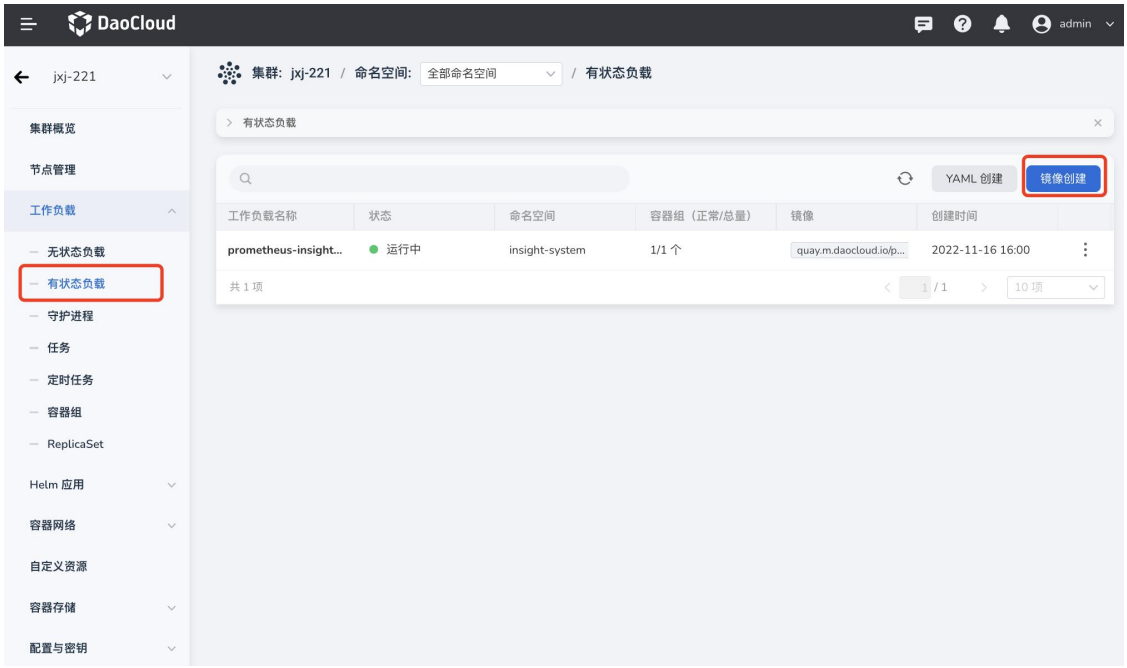
参考以下步骤，使用镜像创建一个有状态负载。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情。



集群详情

2. 点击左侧导航栏的工作负载 –> 有状态负载，然后点击右上角镜像创建按钮。

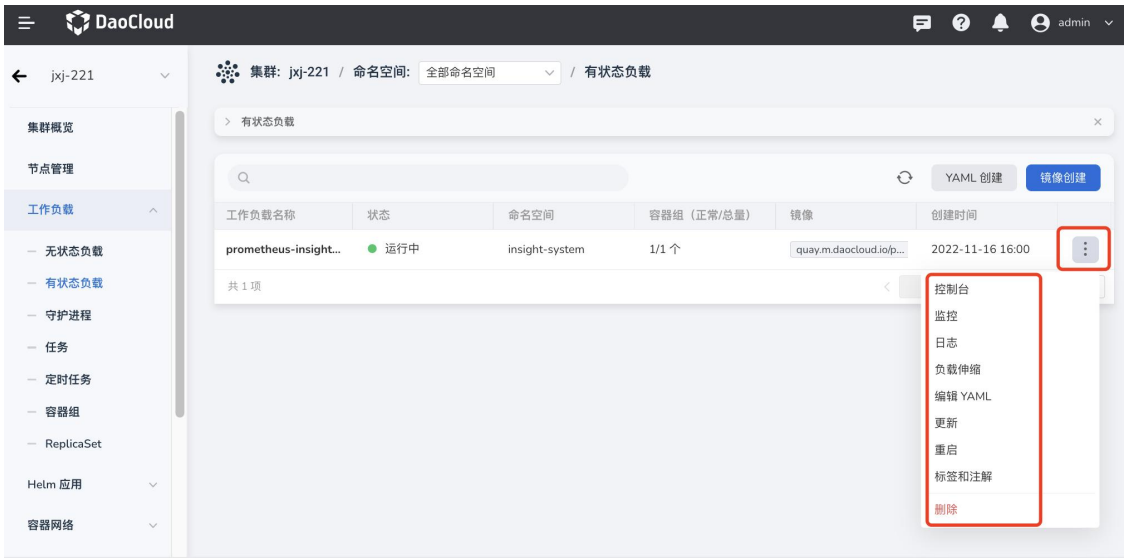


工作负载

3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击确定完成创建。

   系统将自动返回有状态工作负载列表，等待工作负载状态变为运行中。如果工作负载状态出现异常，请查看具体异常信息，可参考工作负载状态。

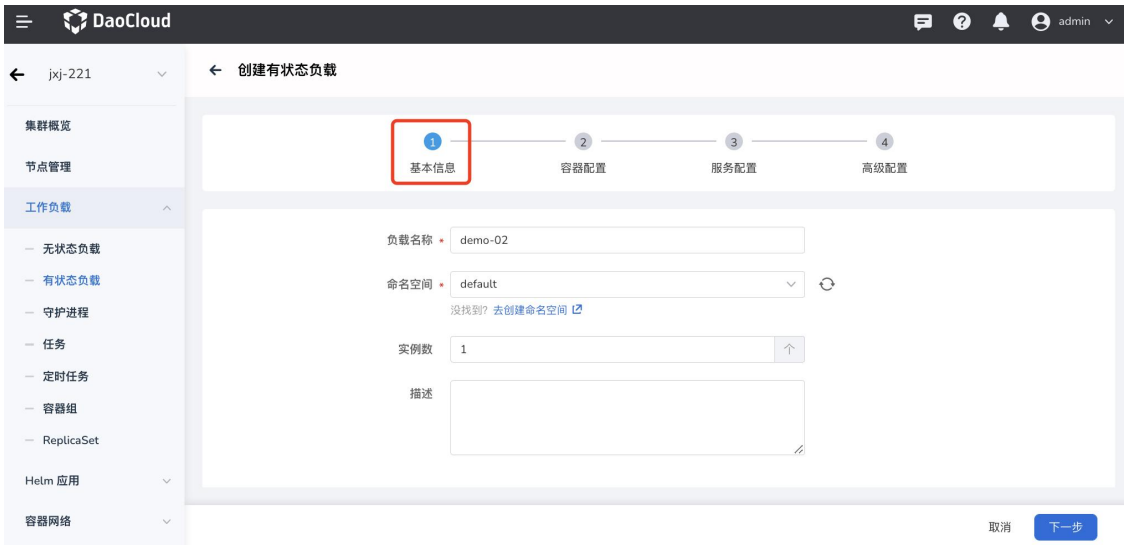
   点击新建工作负载列右侧的 ：，可以对工作负载执行执行更新、删除、弹性扩缩、重启、版本回退等操作。



操作菜单

基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 deployment-01。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 命名空间：选择将新建的负载部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的命名空间。
- 实例数：输入负载的 Pod 实例数量，默认创建 1 个 Pod 实例。
- 描述：输入负载的描述信息，内容自定义。字符数不超过 512。



基本信息

容器配置

容器配置分为基本信息、生命周期、健康检查、环境变量、数据存储、安全设置六部分，点击下方的相应页签可查看各部分的配置要求。

> 容器配置仅针对单个容器进行配置，如需在一个容器组中添加多个容器，可点击右侧的 + 添加多个容器。

===“基本信息（必填）”

在配置容器相关参数时，必须正确填写容器的名称、镜像参数，否则将无法进入下一步。参考以下要求填写配置后，点击`确认`。

– 容器名称：最多包含 63 个字符，支持小写字母、数字及分隔符（“–”）。必须以小写字母或数字开头及结尾，例如 nginx–01。
– 容器镜像：输入镜像地址或名称。输入镜像名称时，默认从官方的 [DockerHub](https://hub.docker.com/）拉取镜像。接入 DCE 5.0 的[镜像仓库](../../../kangaroo/index.md)模块后，可以点击右侧的`选择镜像`来选择镜像。
– 更新策略：勾选`总是拉取镜像`后，负载每次重启/升级时都会从仓库重新拉取镜像。如果不勾选，则只拉取本地镜像，只有当镜像在本地不存在时才从镜像仓库重新拉取。更多详情可参考[镜像拉取策略](https://kubernetes.io/zh–cn/docs/concepts/containers/i

mages/#image-pull-policy)。
- 特权容器：默认情况下，容器不可以访问宿主机上的任何设备，开启特权容器后，容器即可访问宿主机上的所有设备，享有宿主机上的运行进程的所有权限。
- CPU/内存配额：CPU/内存资源的请求值（需要使用的最小资源）和限制值（允许使用的最大资源）。请根据需要为容器配置资源，避免资源浪费和因容器资源超额导致系统故障。默认值如图所示。
- GPU 独享：为容器配置 GPU 用量，仅支持输入正整数。GPU 配额设置支持为容器设置独享整张 GPU 卡或部分 vGPU。例如，对于一张 8 核心的 GPU 卡，输入数字 `8` 表示让容器独享整长卡，输入数字 `1` 表示为容器配置 1 核心的 vGPU。

> 设置 GPU 独享之前，需要管理员预先在集群节点上安装 GPU 卡及驱动插件，并在[集群设置](../clusterops/cluster-settings.md)中开启 GPU 特性。

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/state11.png)

===“生命周期（选填）”

设置容器启动时、启动后、停止前需要执行的命令。详情可参考[容器生命周期配置](pod-config/lifecycle.md)。

![生命周期](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy06.png)

===“健康检查（选填）”

用于判断容器和应用的健康状态。有助于提高应用的可用性。详情可参考[容器健康检查配置](pod-config/health-check.md)。

![健康检查](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy07.png)

===“环境变量（选填）”

配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考[容器环境变量配置](pod-config/env-variables.md)。

![环境变量](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy08.png)

===“数据存储（选填）”

配置容器挂载数据卷和数据持久化的设置。详情可参考[容器数据存储配置](pod-config/env-variables.md)。

![数据存储](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy09.png)

===“安全设置（选填）”

通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 `0` 表示使用 root 账号的权限。

![安全设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy10.png)

服务配置

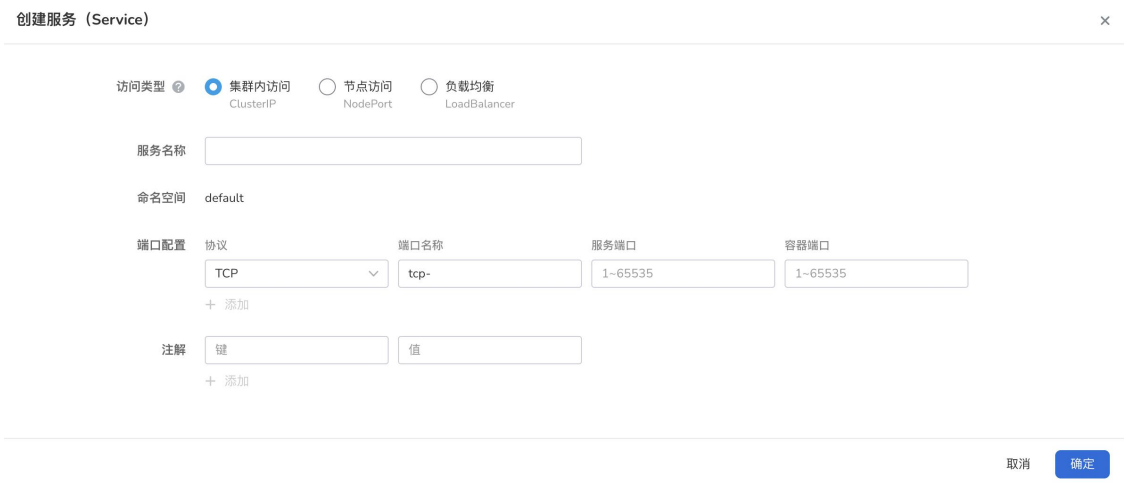
为有状态负载配置服务（Service），使有状态负载能够被外部访问。

1. 点击创建服务按钮。



服务配置

2. 参考创建服务，配置服务参数。



创建服务

3. 点击确定，点击下一步。

高级配置

高级配置包括负载的网络配置、升级策略、调度策略、标签与注解四部分，可点击下方的页签查看各部分的配置要求。

===“网络配置”

– 如在集群中部署了 [SpiderPool](../../../network/modules/spiderpool/index.md) 和 [Multus](../../../network/modules/multus-underlay/index.md) 组件，则可以在网络配置中配置容器网卡。详情参考[工作负载使用 IP 池](../../../network/modules/spiderpool/usage.md)。

– DNS 配置：应用在某些场景下会出现冗余的 DNS 查询。Kubernetes 为应用提供了与 DNS 相关的配置选项，能够在某些场景下有效地减少冗余的 DNS 查询，提升业务并发量。

– DNS 策略

  – Default：使容器使用 kubelet 的 `--resolv-conf` 参数指向的域名解析文件。该配置只能解析注册到互联网上的外部域名，无法解析集群内部域名，且不存在无效的 DNS 查询。
  – ClusterFirstWithHostNet：应用对接主机的域名文件。
  – ClusterFirst：应用对接 Kube-DNS/CoreDNS。
  – None：Kubernetes v1.9（Beta in v1.10）中引入的新选项值。设置为 None 之后，必须设置 dnsConfig，此时容器的域名解析文件将完全通过 dnsConfig 的配置来生成。

– 域名服务器：填写域名服务器的地址，例如 `10.6.175.20`。
– 搜索域：域名查询时的 DNS 搜索域列表。指定后，提供的搜索域列表将合并到基于 dnsPolicy 生成的域名解析文件的 search 字段中，并删除重复的域名。Kubernetes 最多允许 6 个搜索域。
– Options：DNS 的配置选项，其中每个对象可以具有 name 属性（必需）和 value 属性（可选）。该字段中的内容将合并到基于 dnsPolicy 生成的域名解析文件的 options 字段中，dnsConfig 的 options 的某些选项如果与基于 dnsPolicy 生成的域名解析文件的选项冲突，则会被 dnsConfig 所覆盖。
– 主机别名：为主机设置的别名。

![DNS 配置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy17.png)

===“升级策略”

– 升级方式：`滚动升级`指逐步用新版本的实例替换旧版本的实例，升级的过程中，业务流量会同时负载均衡分布到新老的实例上，因此业务不会中断。`重建升级`指先删除老版本的负载实例，再安装指定的新版本，升级过程中业务会中断。
– 最大无效 Pod 数：指定负载更新过程中不可用 Pod 的最大值或比率，默认 25%。如果等于实例数有服务中断的风险。
– 最大浪涌：更新 Pod 的过程中 Pod 总数超过 Pod 期望副本数部分的最大值或比率。默认 25%。
– 最大保留版本数：设置版本回滚时保留的旧版本数量。默认 10。
– Pod 可用最短时间：Pod 就绪的最短时间，只有超出这个时间 Pod 才被认为可用，默认 0 秒。
– 升级最大持续时间：如果超过所设置的时间仍未部署成功，则将该负载标记为失败。默认 600 秒。
– 缩容时间窗：负载停止前命令的执行时间窗（0–9,999 秒），默认 30 秒。

![升级策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy14.png)

===“容器管理策略”

Kubernetes v1.7 及其之后的版本可以通过 `.spec.podManagementPolicy` 设置 Pod 的管理策略，支持以下两种方式：

– `按序策略（OrderedReady）`：默认的 Pod 管理策略，表示按顺序部署 Pod，只有前一个 Pod 部署 成功完成后，有状态负载才会开始部署下一个 Pod。删除 Pod 时则采用逆序，最后创建的最先被删除。

– `并行策略（Parallel）`：并行创建或删除容器，和 Deployment 类型的 Pod 一样。StatefulSet 控制器并行地启动或终止所有的容器。启动或者终止其他 Pod 前，无需等待 Pod 进入 Running 和 ready 或者完全停止状态。 这个选项只会影响扩缩操作的行

为，不影响更新时的顺序。

![容器管理策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/state05.png)

===“调度策略”

- 容忍时间：负载实例所在的节点不可用时，将负载实例重新调度到其它可用节点的时间，默认为 300 秒。
- 节点亲和性：根据节点上的标签来约束 Pod 可以调度到哪些节点上。
- 工作负载亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 可以调度到哪些节点。
- 工作负载反亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 不可以调度到哪些节点。
- 拓扑域：即 topologyKey，用于指定可以调度的一组节点。例如，`kubernetes.io/os` 表示只要某个操作系统的节点满足 labelSelector 的条件就可以调度到该节点。

> 具体详情请参考[调度策略](pod-config/scheduling-policy.md)。

![调度策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy15.png)

===“标签与注解”

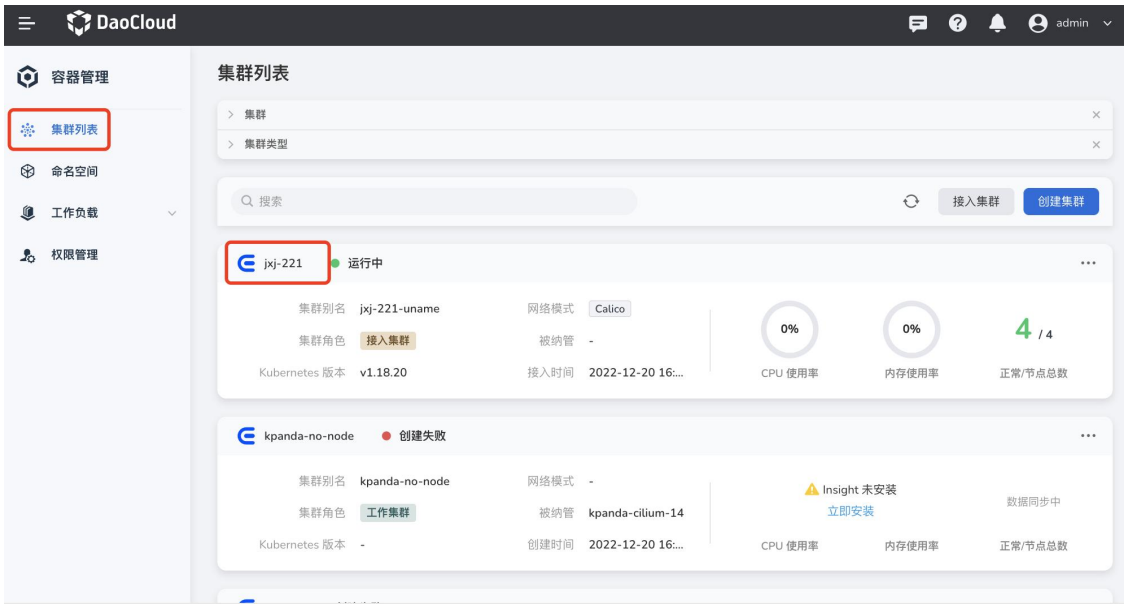
可以点击`添加`按钮为工作负载和容器组添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy16.png)

YAML 创建

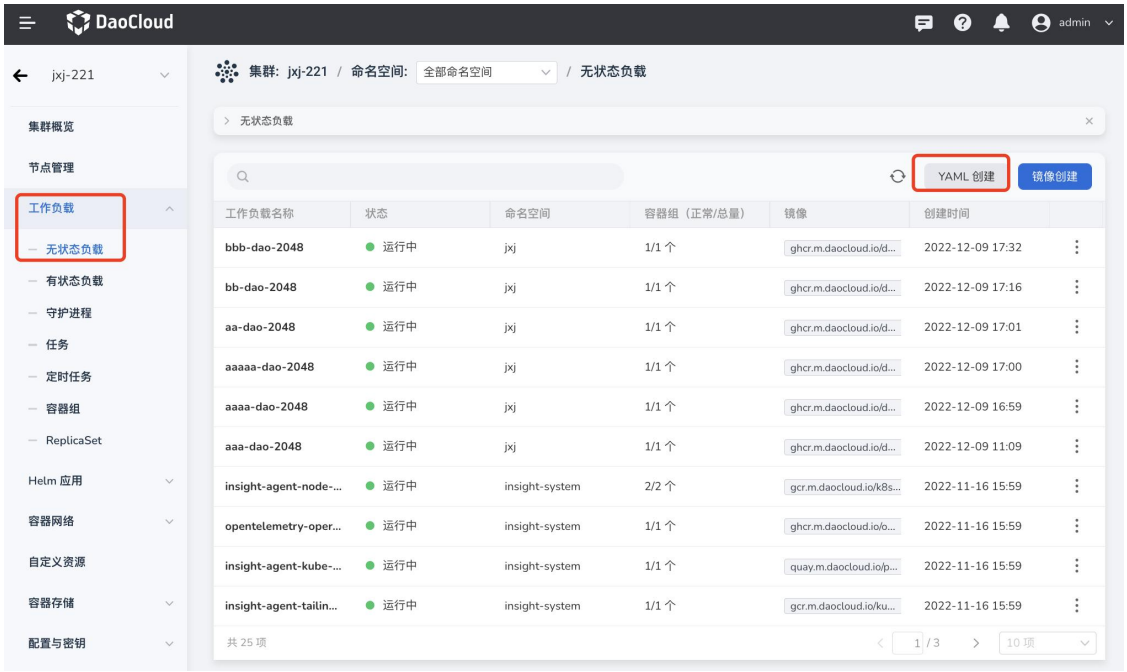
除了通过镜像方式外，还可以通过 YAML 文件更快速地创建创建有状态负载。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



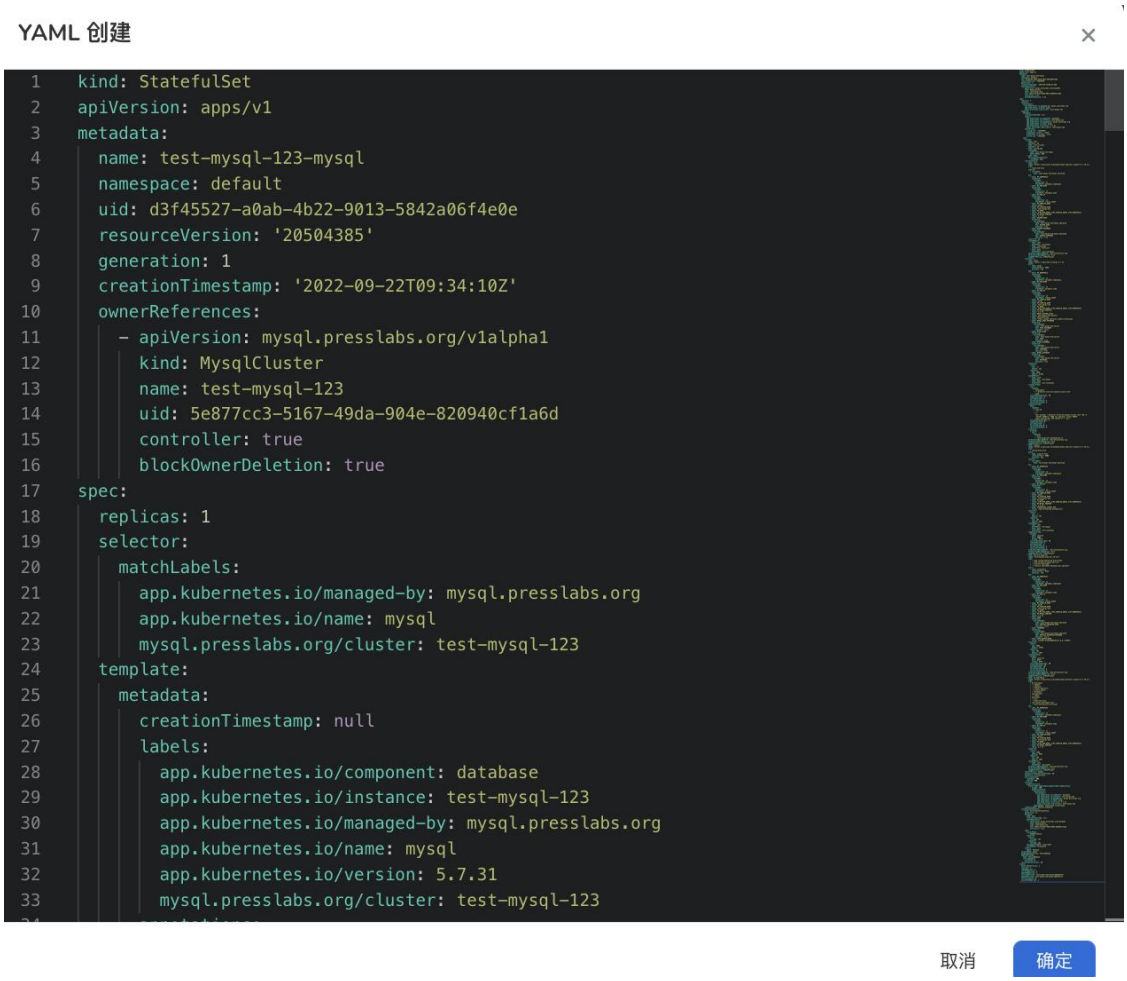
集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载 –> 有状态负载，然后点击页面右上角的 YAML 创建按钮。



工作负载

3. 输入或粘贴事先准备好的 YAML 文件，点击确定即可完成创建。



工作负载

??? note “点击查看创建有状态负载的 YAML 示例”

```yaml
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: test-mysql-123-mysql
  namespace: default
  uid: d3f45527-a0ab-4b22-9013-5842a06f4e0e
  resourceVersion: '20504385'
  generation: 1
  creationTimestamp: '2022-09-22T09:34:10Z'
```

```
  ownerReferences:
    - apiVersion: mysql.presslabs.org/v1alpha1
      kind: MysqlCluster
      name: test-mysql-123
      uid: 5e877cc3-5167-49da-904e-820940cf1a6d
      controller: true
      blockOwnerDeletion: true
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: mysql.presslabs.org
      app.kubernetes.io/name: mysql
      mysql.presslabs.org/cluster: test-mysql-123
  template:
    metadata:
      creationTimestamp: null
      labels:
        app.kubernetes.io/component: database
        app.kubernetes.io/instance: test-mysql-123
        app.kubernetes.io/managed-by: mysql.presslabs.org
        app.kubernetes.io/name: mysql
        app.kubernetes.io/version: 5.7.31
        mysql.presslabs.org/cluster: test-mysql-123
      annotations:
        config_rev: '13941099'
        prometheus.io/port: '9125'
        prometheus.io/scrape: 'true'
        secret_rev: '13941101'
    spec:
      volumes:
        - name: conf
```

```
    emptyDir: {}
  - name: init-scripts
    emptyDir: {}
  - name: config-map
    configMap:
      name: test-mysql-123-mysql
      defaultMode: 420
  - name: data
    persistentVolumeClaim:
      claimName: data
initContainers:
  - name: init
    image: docker.m.daocloud.io/bitpoke/mysql-operator-sidecar-5.7:v0.6.1
    args:
      - clone-and-init
    envFrom:
      - secretRef:
          name: test-mysql-123-mysql-operated
    env:
      - name: MY_NAMESPACE
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.namespace
      - name: MY_POD_NAME
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.name
      - name: MY_POD_IP
        valueFrom:
          fieldRef:
```

```
            apiVersion: v1
            fieldPath: status.podIP
        - name: MY_SERVICE_NAME
          value: mysql
        - name: MY_CLUSTER_NAME
          value: test-mysql-123
        - name: MY_FQDN
          value: $(MY_POD_NAME).$(MY_SERVICE_NAME).$(MY_NAMESPAC
E)
        - name: MY_MYSQL_VERSION
          value: 5.7.31
        - name: BACKUP_USER
          valueFrom:
            secretKeyRef:
              name: test-mysql-123-mysql-operated
              key: BACKUP_USER
              optional: true
        - name: BACKUP_PASSWORD
          valueFrom:
            secretKeyRef:
              name: test-mysql-123-mysql-operated
              key: BACKUP_PASSWORD
              optional: true
      resources: {}
      volumeMounts:
        - name: conf
          mountPath: /etc/mysql
        - name: config-map
          mountPath: /mnt/conf
        - name: data
          mountPath: /var/lib/mysql
      terminationMessagePath: /dev/termination-log
```

```
      terminationMessagePolicy: File
      imagePullPolicy: IfNotPresent
    containers:
      - name: mysql
        image: docker.m.daocloud.io/mysql:5.7.31
        ports:
          - name: mysql
            containerPort: 3306
            protocol: TCP
        env:
          - name: MY_NAMESPACE
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.namespace
          - name: MY_POD_NAME
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.name
          - name: MY_POD_IP
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: status.podIP
          - name: MY_SERVICE_NAME
            value: mysql
          - name: MY_CLUSTER_NAME
            value: test-mysql-123
          - name: MY_FQDN
            value: $(MY_POD_NAME).$(MY_SERVICE_NAME).$(MY_NAMESPAC
E)
```

```
    - name: MY_MYSQL_VERSION
      value: 5.7.31
    - name: ORCH_CLUSTER_ALIAS
      value: test-mysql-123.default
    - name: ORCH_HTTP_API
      value: http://mysql-operator.mcamel-system/api
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: test-mysql-123-secret
          key: ROOT_PASSWORD
          optional: false
    - name: MYSQL_USER
      valueFrom:
        secretKeyRef:
          name: test-mysql-123-secret
          key: USER
          optional: true
    - name: MYSQL_PASSWORD
      valueFrom:
        secretKeyRef:
          name: test-mysql-123-secret
          key: PASSWORD
          optional: true
    - name: MYSQL_DATABASE
      valueFrom:
        secretKeyRef:
          name: test-mysql-123-secret
          key: DATABASE
          optional: true
  resources:
    limits:
```

```
      cpu: '1'
      memory: 1Gi
    requests:
      cpu: 100m
      memory: 512Mi
  volumeMounts:
    - name: conf
      mountPath: /etc/mysql
    - name: data
      mountPath: /var/lib/mysql
  livenessProbe:
    exec:
      command:
        - mysqladmin
        - '--defaults-file=/etc/mysql/client.conf'
        - ping
    initialDelaySeconds: 60
    timeoutSeconds: 5
    periodSeconds: 5
    successThreshold: 1
    failureThreshold: 3
  readinessProbe:
    exec:
      command:
        - /bin/sh
        - '-c'
        - >-
          test $(mysql --defaults-file=/etc/mysql/client.conf -NB -e
          'SELECT COUNT(*) FROM sys_operator.status WHERE
          name="configured" AND value="1"') -eq 1
    initialDelaySeconds: 5
    timeoutSeconds: 5
```

```
    periodSeconds: 2
    successThreshold: 1
    failureThreshold: 3
  lifecycle:
    preStop:
      exec:
        command:
          - bash
          - /etc/mysql/pre-shutdown-ha.sh
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  imagePullPolicy: IfNotPresent
- name: sidecar
  image: docker.m.daocloud.io/bitpoke/mysql-operator-sidecar-5.7:v0.6.1
  args:
    - config-and-serve
  ports:
    - name: sidecar-http
      containerPort: 8080
      protocol: TCP
  envFrom:
    - secretRef:
        name: test-mysql-123-mysql-operated
  env:
    - name: MY_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
```

```
            apiVersion: v1
            fieldPath: metadata.name
      - name: MY_POD_IP
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: status.podIP
      - name: MY_SERVICE_NAME
        value: mysql
      - name: MY_CLUSTER_NAME
        value: test-mysql-123
      - name: MY_FQDN
        value: $(MY_POD_NAME).$(MY_SERVICE_NAME).$(MY_NAMESPAC
E)
      - name: MY_MYSQL_VERSION
        value: 5.7.31
      - name: XTRABACKUP_TARGET_DIR
        value: /tmp/xtrabackup_backupfiles/
    resources:
      limits:
        cpu: '1'
        memory: 1Gi
      requests:
        cpu: 10m
        memory: 64Mi
    volumeMounts:
      - name: conf
        mountPath: /etc/mysql
      - name: data
        mountPath: /var/lib/mysql
    readinessProbe:
      httpGet:
```

```
      path: /health
      port: 8080
      scheme: HTTP
    initialDelaySeconds: 30
    timeoutSeconds: 5
    periodSeconds: 5
    successThreshold: 1
    failureThreshold: 3
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  imagePullPolicy: IfNotPresent
- name: metrics-exporter
  image: prom/mysqld-exporter:v0.13.0
  args:
    - '--web.listen-address=0.0.0.0:9125'
    - '--web.telemetry-path=/metrics'
    - '--collect.heartbeat'
    - '--collect.heartbeat.database=sys_operator'
  ports:
    - name: prometheus
      containerPort: 9125
      protocol: TCP
  env:
    - name: MY_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
```

```
            fieldPath: metadata.name
      - name: MY_POD_IP
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: status.podIP
      - name: MY_SERVICE_NAME
        value: mysql
      - name: MY_CLUSTER_NAME
        value: test-mysql-123
      - name: MY_FQDN
        value: $(MY_POD_NAME).$(MY_SERVICE_NAME).$(MY_NAMESPAC
E)
      - name: MY_MYSQL_VERSION
        value: 5.7.31
      - name: USER
        valueFrom:
          secretKeyRef:
            name: test-mysql-123-mysql-operated
            key: METRICS_EXPORTER_USER
            optional: false
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: test-mysql-123-mysql-operated
            key: METRICS_EXPORTER_PASSWORD
            optional: false
      - name: DATA_SOURCE_NAME
        value: $(USER):$(PASSWORD)@(127.0.0.1:3306)/
    resources:
      limits:
        cpu: 100m
```

```
        memory: 128Mi
      requests:
        cpu: 10m
        memory: 32Mi
    livenessProbe:
      httpGet:
        path: /metrics
        port: 9125
        scheme: HTTP
      initialDelaySeconds: 30
      timeoutSeconds: 30
      periodSeconds: 30
      successThreshold: 1
      failureThreshold: 3
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    imagePullPolicy: IfNotPresent
  - name: pt-heartbeat
    image: docker.m.daocloud.io/bitpoke/mysql-operator-sidecar-5.7:v0.6.1
    args:
      - pt-heartbeat
      - '--update'
      - '--replace'
      - '--check-read-only'
      - '--create-table'
      - '--database'
      - sys_operator
      - '--table'
      - heartbeat
      - '--utc'
      - '--defaults-file'
      - /etc/mysql/heartbeat.conf
```

```
    - '--fail-successive-errors=20'
  env:
    - name: MY_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: MY_POD_IP
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: status.podIP
    - name: MY_SERVICE_NAME
      value: mysql
    - name: MY_CLUSTER_NAME
      value: test-mysql-123
    - name: MY_FQDN
      value: $(MY_POD_NAME).$(MY_SERVICE_NAME).$(MY_NAMESPAC
E)
    - name: MY_MYSQL_VERSION
      value: 5.7.31
  resources:
    limits:
      cpu: 100m
      memory: 64Mi
    requests:
      cpu: 10m
```

```
            memory: 32Mi
        volumeMounts:
          - name: conf
            mountPath: /etc/mysql
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      securityContext:
        runAsUser: 999
        fsGroup: 999
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - weight: 100
              podAffinityTerm:
                labelSelector:
                  matchLabels:
                    app.kubernetes.io/component: database
                    app.kubernetes.io/instance: test-mysql-123
                    app.kubernetes.io/managed-by: mysql.presslabs.org
                    app.kubernetes.io/name: mysql
                    app.kubernetes.io/version: 5.7.31
                    mysql.presslabs.org/cluster: test-mysql-123
                topologyKey: kubernetes.io/hostname
      schedulerName: default-scheduler
  volumeClaimTemplates:
    - kind: PersistentVolumeClaim
      apiVersion: v1
      metadata:
```

```
      name: data
      creationTimestamp: null
      ownerReferences:
        - apiVersion: mysql.presslabs.org/v1alpha1
          kind: MysqlCluster
          name: test-mysql-123
          uid: 5e877cc3-5167-49da-904e-820940cf1a6d
          controller: true
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        limits:
          storage: 1Gi
        requests:
          storage: 1Gi
      storageClassName: local-path
      volumeMode: Filesystem
    status:
      phase: Pending
  serviceName: mysql
  podManagementPolicy: OrderedReady
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  revisionHistoryLimit: 10
status:
  observedGeneration: 1
  replicas: 1
  readyReplicas: 1
  currentReplicas: 1
```

```
  updatedReplicas: 1
  currentRevision: test-mysql-123-mysql-6b8f5577c7
  updateRevision: test-mysql-123-mysql-6b8f5577c7
  collisionCount: 0
  availableReplicas: 1
```

创建定时任务（CronJob）

本文介绍如何通过镜像和 YAML 文件两种方式创建定时任务（CronJob）。

定时任务（CronJob）适用于于执行周期性的操作，例如备份、报告生成等。这些任务可以配置为周期性重复的（例如：每天/每周/每月一次），可以定义任务开始执行的时间间隔。

前提条件

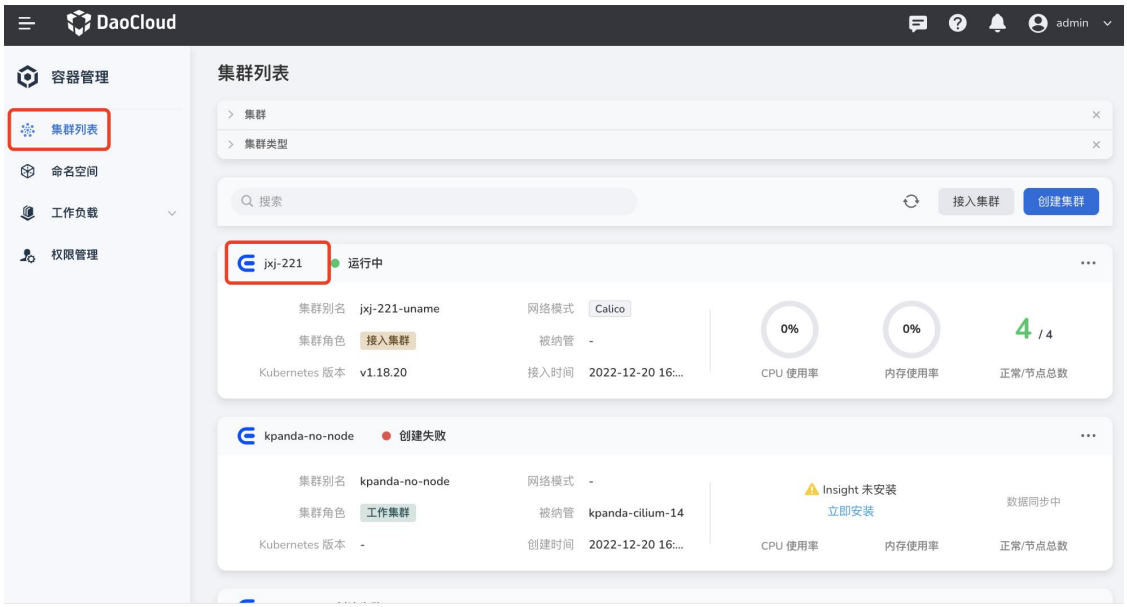
创建定时任务（CronJob）之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

镜像创建

参考以下步骤，使用镜像创建一个定时任务。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



集群详情

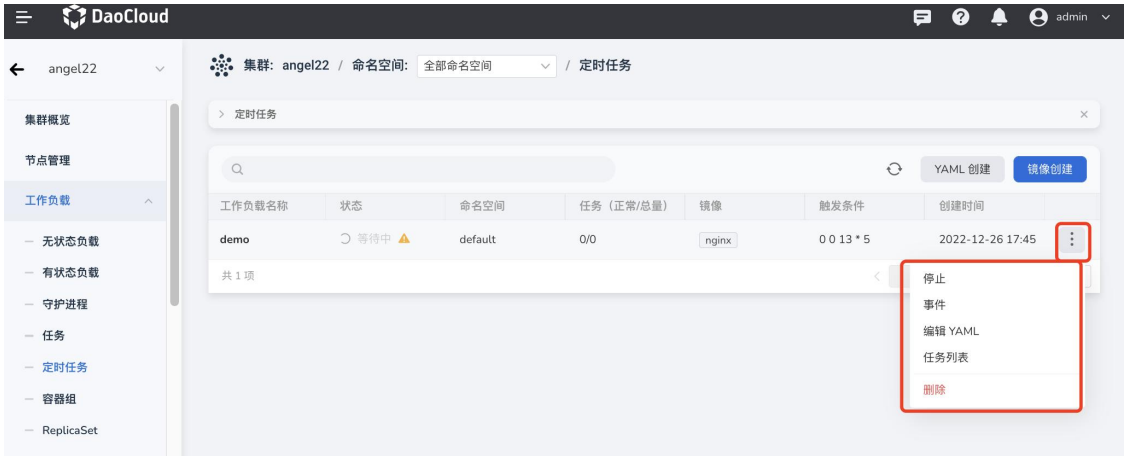
2. 在集群详情页面，点击左侧导航栏的工作负载 -> 定时任务，然后点击页面右上角的镜像创建按钮。



工作负载

3. 依次填写基本信息、容器配置、定时任务配置、高级配置后，在页面右下角点击确定完成创建。

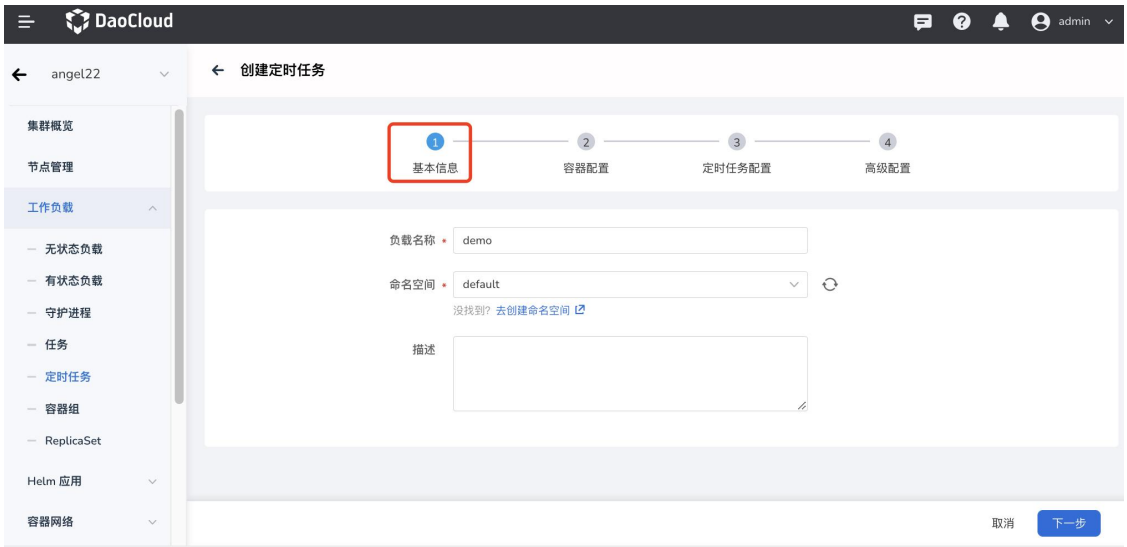
系统将自动返回定时任务列表。点击列表右侧的 ：，可以对定时任务执行执行更新、删除、重启等操作。



操作菜单

基本信息

在创建定时任务页面中，根据下表输入信息后，点击下一步。



基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。

- 命名空间：选择将新建的定时任务部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的命名空间。
- 描述：输入工作负载的描述信息，内容自定义。字符数量应不超过 512 个。

容器配置

容器配置分为基本信息、生命周期、健康检查、环境变量、数据存储、安全设置六部分，点击下方的相应页签可查看各部分的配置要求。

> 容器配置仅针对单个容器进行配置，如需在一个容器组中添加多个容器，可点击右侧的 + 添加多个容器。

===“基本信息（必填）”

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/cronjob03.png)

在配置容器相关参数时，必须正确填写容器的名称、镜像参数，否则将无法进入下一步。参考以下要求填写配置后，点击`确认`。

– 容器名称：最多包含 63 个字符，支持小写字母、数字及分隔符（“-”）。必须以小写字母或数字开头及结尾，例如 nginx-01。
– 容器镜像：输入镜像地址或名称。输入镜像名称时，默认从官方的 [DockerHub](https://hub.docker.com/) 拉取镜像。接入 DCE 5.0 的[镜像仓库](../../../kangaroo/index.md)模块后，可以点击右侧的`选择镜像`来选择镜像。
– 更新策略：勾选`总是拉取镜像`后，负载每次重启/升级时都会从仓库重新拉取镜像。如果不勾选，则只拉取本地镜像，只有当镜像在本地不存在时才从镜像仓库重新拉取。更多详情可参考[镜像拉取策略](https://kubernetes.io/zh-cn/docs/concepts/containers/images/#image-pull-policy)。
– 特权容器：默认情况下，容器不可以访问宿主机上的任何设备，开启特权容器后，容器即可访问宿主机上的所有设备，享有宿主机上的运行进程的所有权限。
– CPU/内存配额：CPU/内存资源的请求值（需要使用的最小资源）和限制值（允许使

用的最大资源）。请根据需要为容器配置资源，避免资源浪费和因容器资源超额导致系统故障。默认值如图所示。
– GPU 独享：为容器配置 GPU 用量，仅支持输入正整数。GPU 配额设置支持为容器设置独享整张 GPU 卡或部分 vGPU。例如，对于一张 8 核心的 GPU 卡，输入数字 `8` 表示让容器独享整长卡，输入数字 `1` 表示为容器配置 1 核心的 vGPU。

> 设置 GPU 独享之前，需要管理员预先在集群节点上安装 GPU 卡及驱动插件，并在[集群设置](../clusterops/cluster-settings.md)中开启 GPU 特性。

===“生命周期（选填）”

设置容器启动时、启动后、停止前需要执行的命令。详情可参考[容器生命周期配置](pod-config/lifecycle.md)。

![生命周期](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy06.png)

===“健康检查（选填）”

用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考[容器健康检查配置](pod-config/health-check.md)。

![健康检查](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy07.png)

===“环境变量（选填）”

配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考[容器环境变量配置](pod-config/env-variables.md)。

![环境变量](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy08.png)

===“数据存储（选填）”

配置容器挂载数据卷和数据持久化的设置。详情可参考[容器数据存储配置](pod-config/env-variables.md)。

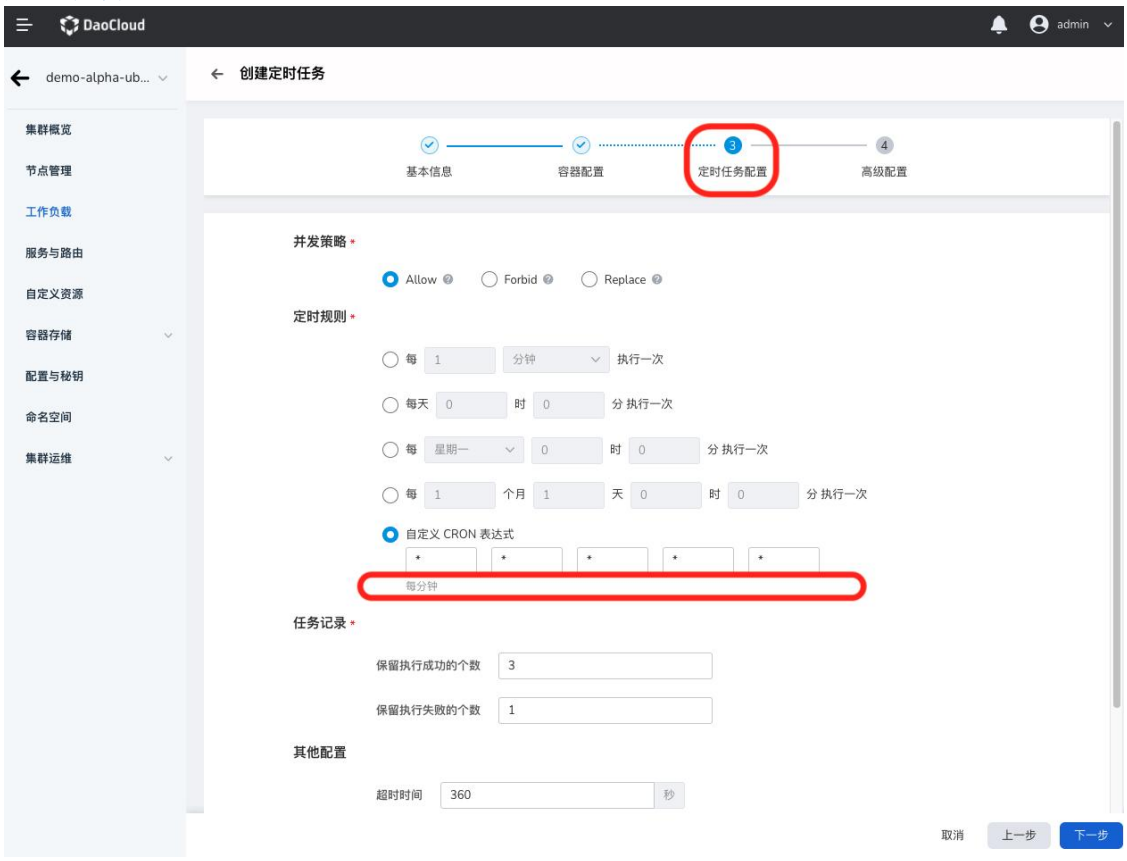
![数据存储](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy09.png)

===“安全设置（选填）”

通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 `0` 表示使用 root 账号的权限。

![安全设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy10.png)

定时任务配置



定时任务配置

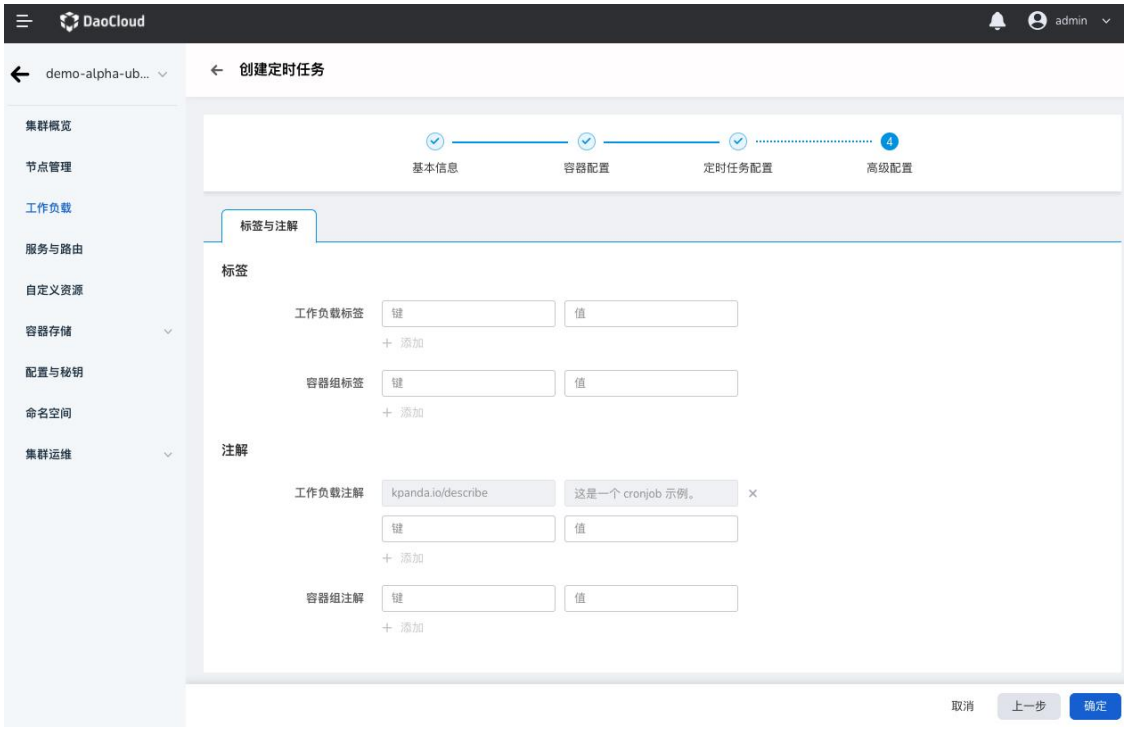
- 并发策略：是否允许多个 Job 任务并行执行。
  - Allow：可以在前一个任务未完成时就创建新的定时任务，而且多个任务可以并行。任务太多可能抢占集群资源。
  - Forbid：在前一个任务完成之前，不能创建新任务，如果新任务的执行时间到了而之前的任务仍未执行完，CronJob 会忽略新任务的执行。
  - Replace：如果新任务的执行时间到了，但前一个任务还未完成，新的任务会取代前一个任务。

  上述规则仅适用于同一个 CronJob 创建的多个任务。多个 CronJob 创建的多个任务总是允许并发执行。
- 定时规则：基于分钟、小时、天、周、月设置任务执行的时间周期。支持用数字和 * 自定义 Cron 表达式，**输入表达式后下方会提示当前表达式的含义**。有关详细的表达式语法规则，可参考 Cron 时间表语法。
- 任务记录：设定保留多少条任务执行成功或失败的记录。0 表示不保留。
- 超时时间：超出该时间时，任务就会被标识为执行失败，任务下的所有 Pod 都会被删除。为空时表示不设置超时时间。默认值为 360 s。
- 重试次数：任务可重试次数，默认值为 6。
- 重启策略：设置任务失败时是否重启 Pod。

高级配置

定时任务的高级配置主要涉及标签与注解。

可以点击添加按钮为工作负载实例 Pod 添加标签和注解。

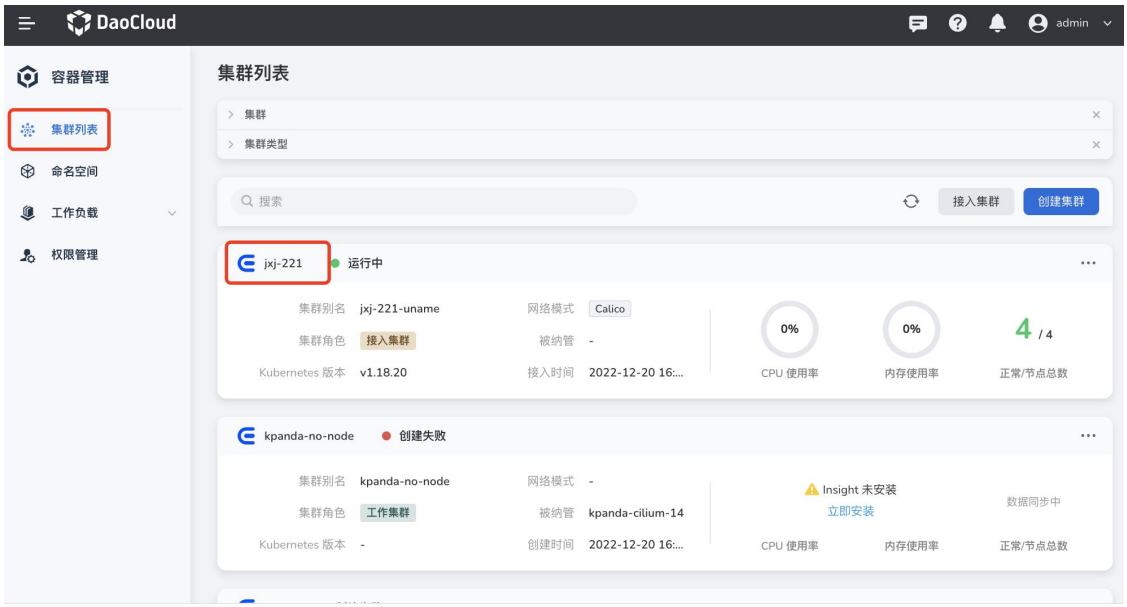


定时任务配置

YAML 创建

除了通过镜像方式外，还可以通过 YAML 文件更快速地创建创建定时任务。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



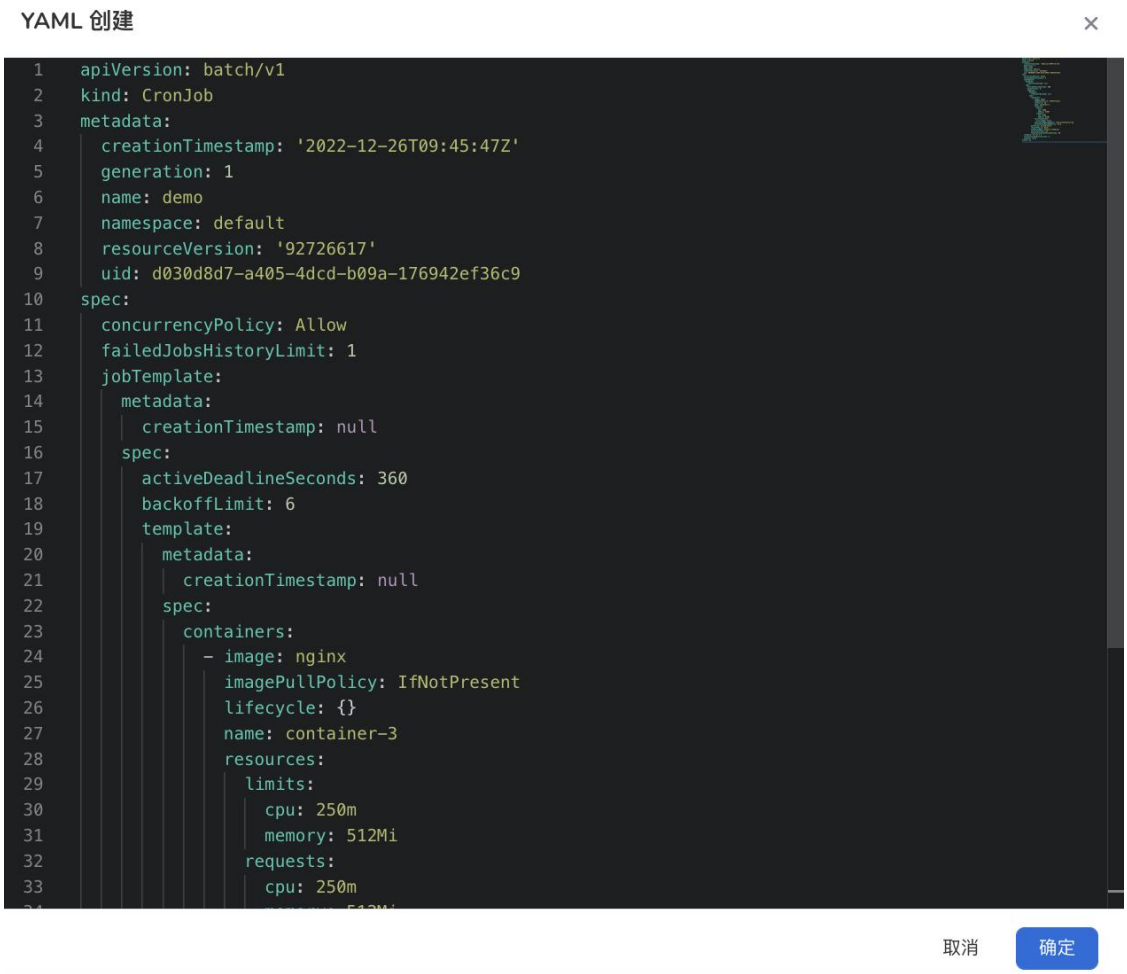
集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载 -> 定时任务，然后点击页面右上角的 YAML 创建按钮。



工作负载

3. 输入或粘贴事先准备好的 YAML 文件，点击确定即可完成创建。



工作负载

??? note“点击查看创建定时任务的 YAML 示例”

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  creationTimestamp: '2022-12-26T09:45:47Z'
  generation: 1
  name: demo
  namespace: default
  resourceVersion: '92726617'
  uid: d030d8d7-a405-4dcd-b09a-176942ef36c9
```

```
spec:
  concurrencyPolicy: Allow
  failedJobsHistoryLimit: 1
  jobTemplate:
    metadata:
      creationTimestamp: null
    spec:
      activeDeadlineSeconds: 360
      backoffLimit: 6
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
            - image: nginx
              imagePullPolicy: IfNotPresent
              lifecycle: {}
              name: container-3
              resources:
                limits:
                  cpu: 250m
                  memory: 512Mi
                requests:
                  cpu: 250m
                  memory: 512Mi
              securityContext:
                privileged: false
              terminationMessagePath: /dev/termination-log
              terminationMessagePolicy: File
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          schedulerName: default-scheduler
```

```
            securityContext: {}
            terminationGracePeriodSeconds: 30
    schedule: 0 0 13 * 5
    successfulJobsHistoryLimit: 3
    suspend: false
status: {}
```

创建守护进程(DaemonSet)

本文介绍如何通过镜像和 YAML 文件两种方式创建守护进程（DaemonSet）。

守护进程（DaemonSet）通过节点亲和性与污点功能确保在全部或部分节点上运行一个 Pod 的副本。对于新加入集群的节点，DaemonSet 自动在新节点上部署相应的 Pod，并跟踪 Pod 的运行状态。当节点被移除时，DaemonSet 则删除其创建的所有 Pod。

守护进程的常见用例包括：

- 在每个节点上运行集群守护进程。
- 在每个节点上运行日志收集守护进程。
- 在每个节点上运行监控守护进程。

简单起见，可以在每个节点上为每种类型的守护进程都启动一个 DaemonSet。如需更精细、更高级地管理守护进程，也可以为同一种守护进程部署多个 DaemonSet。每个 DaemonSet 具有不同的标志，并且对不同硬件类型具有不同的内存、CPU 要求。

前提条件

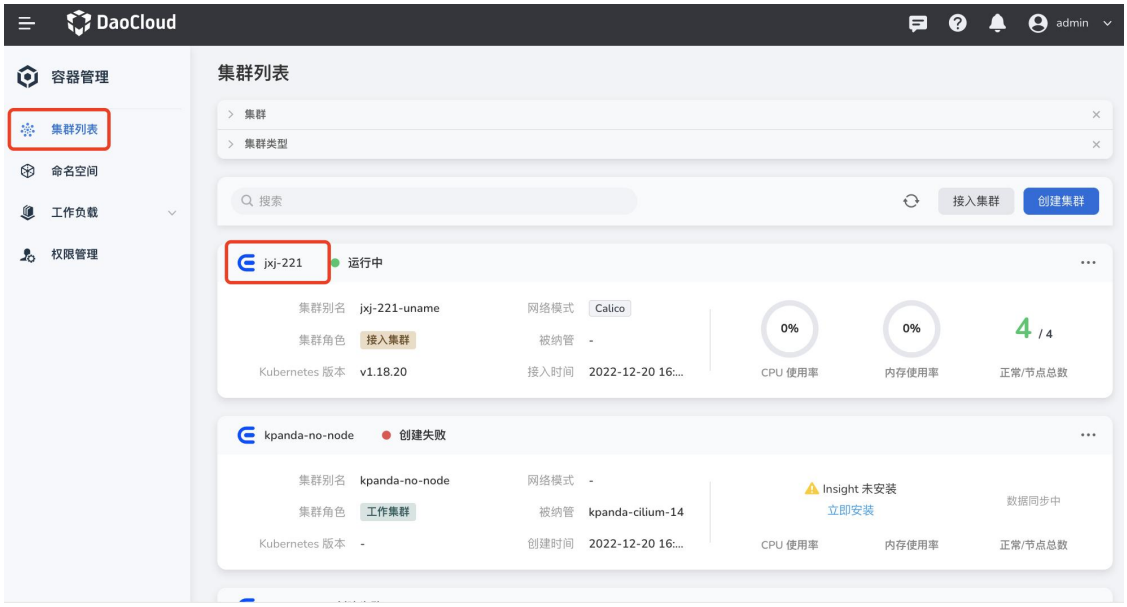
创建 DaemonSet 之前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

镜像创建

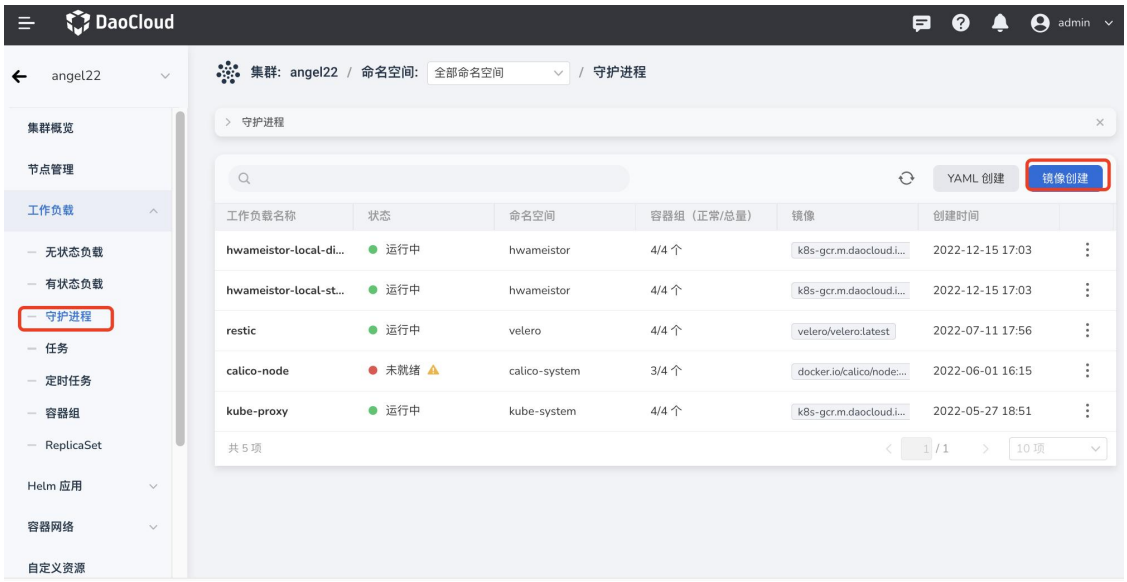
参考以下步骤，使用镜像创建一个守护进程。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



集群详情

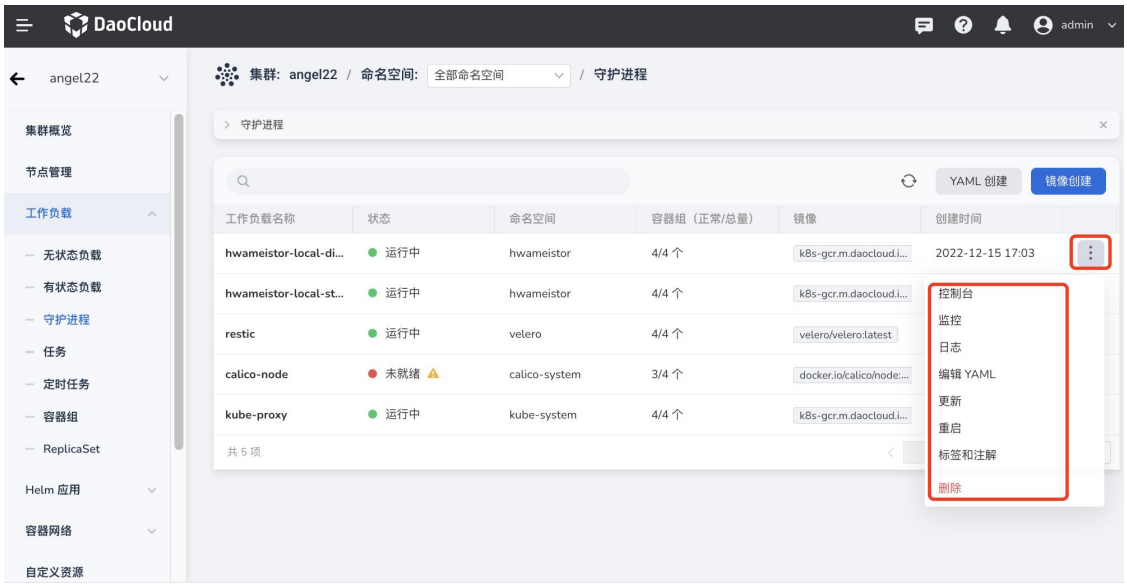
2. 在集群详情页面，点击左侧导航栏的工作负载 -> 守护进程，然后点击页面右上角的镜像创建按钮。



工作负载

3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击确定完成创建。

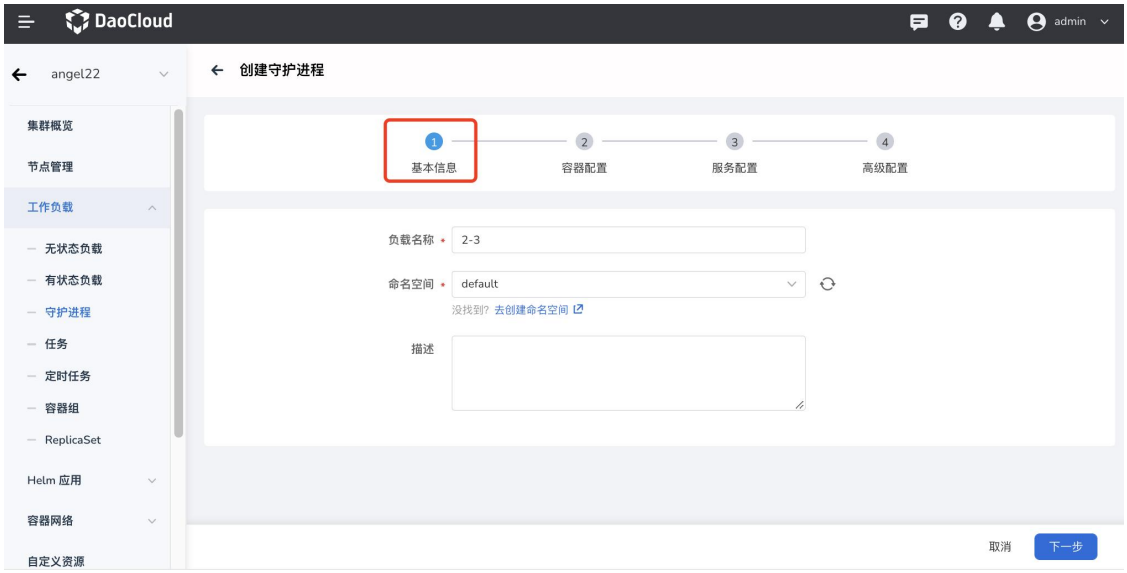
   系统将自动返回守护进程列表。点击列表右侧的 ：，可以对守护进程执行执行更新、删除、重启等操作。



操作菜单

基本信息

在创建守护进程页面中，根据下表输入信息后，点击下一步。



基本信息

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 命名空间：选择将新建的守护进程部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的命名空间。
- 描述：输入工作负载的描述信息，内容自定义。字符数量应不超过 512 个。

容器配置

容器配置分为基本信息、生命周期、健康检查、环境变量、数据存储、安全设置六部分，点击下方的相应页签可查看各部分的配置要求。

> 容器配置仅针对单个容器进行配置，如需在一个容器组中添加多个容器，可点击右侧的 + 添加多个容器。

===“基本信息（必填）”

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/daemon06.png)

在配置容器相关参数时，必须正确填写容器的名称、镜像参数，否则将无法进入下一步。参考以下要求填写配置后，点击`确认`。

- 容器名称：最多包含 63 个字符，支持小写字母、数字及分隔符（“-”）。必须以小写字母或数字开头及结尾，例如 nginx-01。
- 容器镜像：输入镜像地址或名称。输入镜像名称时，默认从官方的 [DockerHub](https://hub.docker.com/）拉取镜像。接入 DCE 5.0 的[镜像仓库](../../../kangaroo/index.md)模块后，可以点击右侧的`选择镜像`来选择镜像。
- 更新策略：勾选`总是拉取镜像`后，负载每次重启/升级时都会从仓库重新拉取镜像。如果不勾选，则只拉取本地镜像，只有当镜像在本地不存在时才从镜像仓库重新拉取。更多详情可参考[镜像拉取策略](https://kubernetes.io/zh-cn/docs/concepts/containers/images/#image-pull-policy)。
- 特权容器：默认情况下，容器不可以访问宿主机上的任何设备，开启特权容器后，容器即可访问宿主机上的所有设备，享有宿主机上的运行进程的所有权限。
- CPU/内存配额：CPU/内存资源的请求值（需要使用的最小资源）和限制值（允许使用的最大资源）。请根据需要为容器配置资源，避免资源浪费和因容器资源超额导致系统故障。默认值如图所示。
- GPU 独享：为容器配置 GPU 用量，仅支持输入正整数。GPU 配额设置支持为容器设置独享整张 GPU 卡或部分 vGPU。例如，对于一张 8 核心的 GPU 卡，输入数字 `8` 表示让容器独享整长卡，输入数字 `1` 表示为容器配置 1 核心的 vGPU。

> 设置 GPU 独享之前，需要管理员预先在集群节点上安装 GPU 卡及驱动插件，并在[集群设置](../clusterops/cluster-settings.md)中开启 GPU 特性。

===“生命周期（选填）”

设置容器启动时、启动后、停止前需要执行的命令。详情可参考[容器生命周期配置](pod-config/lifecycle.md)。

![生命周期](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy06.png)

===“健康检查（选填）”

用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考[容器健康检查配置](pod-config/health-check.md)。

![健康检查](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy07.png)

===“环境变量（选填）”

配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考[容器环境变量配置](pod-config/env-variables.md)。

![环境变量](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy08.png)

===“数据存储（选填）”

配置容器挂载数据卷和数据持久化的设置。详情可参考[容器数据存储配置](pod-config/env-variables.md)。

![数据存储](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy09.png)

===“安全设置（选填）”

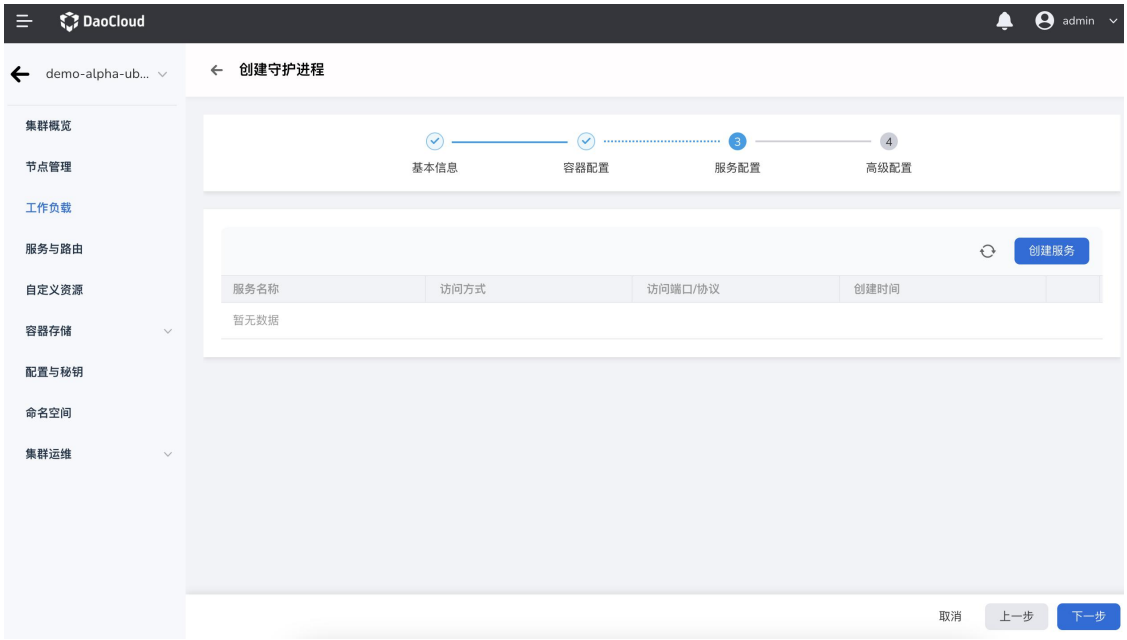
通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 `0` 表示使用 root 账号的权限。

![安全设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy10.png)

服务配置

为守护进程创建服务（Service），使守护进程能够被外部访问。

1. 点击创建服务按钮。



服务配置

2. 配置服务参数，详情请参考创建服务。



创建服务

3. 点击确定，点击下一步。

高级配置

高级配置包括负载的网络配置、升级策略、调度策略、标签与注解四部分，可点击下方的页签查看各部分的配置要求。

===“网络配置”

应用在某些场景下会出现冗余的 DNS 查询。Kubernetes 为应用提供了与 DNS 相关的配置选项，能够在某些场景下有效地减少冗余的 DNS 查询，提升业务并发量。

- DNS 策略

  - Default：使容器使用 kubelet 的 `--resolv-conf` 参数指向的域名解析文件。该配置只能解析注册到互联网上的外部域名，无法解析集群内部域名，且不存在无效的 DNS 查询。
  - ClusterFirstWithHostNet：应用对接主机的域名文件。
  - ClusterFirst：应用对接 Kube-DNS/CoreDNS。
  - None：Kubernetes v1.9（Beta in v1.10）中引入的新选项值。设置为 None 之后，必须设置 dnsConfig，此时容器的域名解析文件将完全通过 dnsConfig 的配置来生成。

- 域名服务器：填写域名服务器的地址，例如 `10.6.175.20`。
- 搜索域：域名查询时的 DNS 搜索域列表。指定后，提供的搜索域列表将合并到基于 dnsPolicy 生成的域名解析文件的 search 字段中，并删除重复的域名。Kubernetes 最多允许 6 个搜索域。
- Options：DNS 的配置选项，其中每个对象可以具有 name 属性（必需）和 value 属性（可选）。该字段中的内容将合并到基于 dnsPolicy 生成的域名解析文件的 options 字段中，dnsConfig 的 options 的某些选项如果与基于 dnsPolicy 生成的域名解析文件

的选项冲突，则会被 dnsConfig 所覆盖。
– 主机别名：为主机设置的别名。

===“升级策略”

![升级策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy14.png)

– 升级方式：`滚动升级`指逐步用新版本的实例替换旧版本的实例，升级的过程中，业务流量会同时负载均衡分布到新老的实例上，因此业务不会中断。`重建升级`指先删除老版本的负载实例，再安装指定的新版本，升级过程中业务会中断。
– 最大无效 Pod 数：指定负载更新过程中不可用 Pod 的最大值或比率，默认 25%。如果等于实例数有服务中断的风险。
– 最大浪涌：更新 Pod 的过程中 Pod 总数超过 Pod 期望副本数部分的最大值或比率。默认 25%。
– 最大保留版本数：设置版本回滚时保留的旧版本数量。默认 10。
– Pod 可用最短时间：Pod 就绪的最短时间，只有超出这个时间 Pod 才被认为可用，默认 0 秒。
– 升级最大持续时间：如果超过所设置的时间仍未部署成功，则将该负载标记为失败。默认 600 秒。
– 缩容时间窗：负载停止前命令的执行时间窗（0–9,999 秒），默认 30 秒。

===“调度策略”

![调度策略](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy15.png)

– 容忍时间：负载实例所在的节点不可用时，将负载实例重新调度到其它可用节点的时间，默认为 300 秒。
– 节点亲和性：根据节点上的标签来约束 Pod 可以调度到哪些节点上。
– 工作负载亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 可以调度到哪些节点。
– 工作负载反亲和性：基于已经在节点上运行的 Pod 的标签来约束 Pod 不可以调度到哪些节点。

– 拓扑域：即 topologyKey，用于指定可以调度的一组节点。例如，`kubernetes.io/os` 表示只要某个操作系统的节点满足 labelSelector 的条件就可以调度到该节点。

> 具体详情请参考[调度策略](pod-config/scheduling-policy.md)。

===“标签与注解”

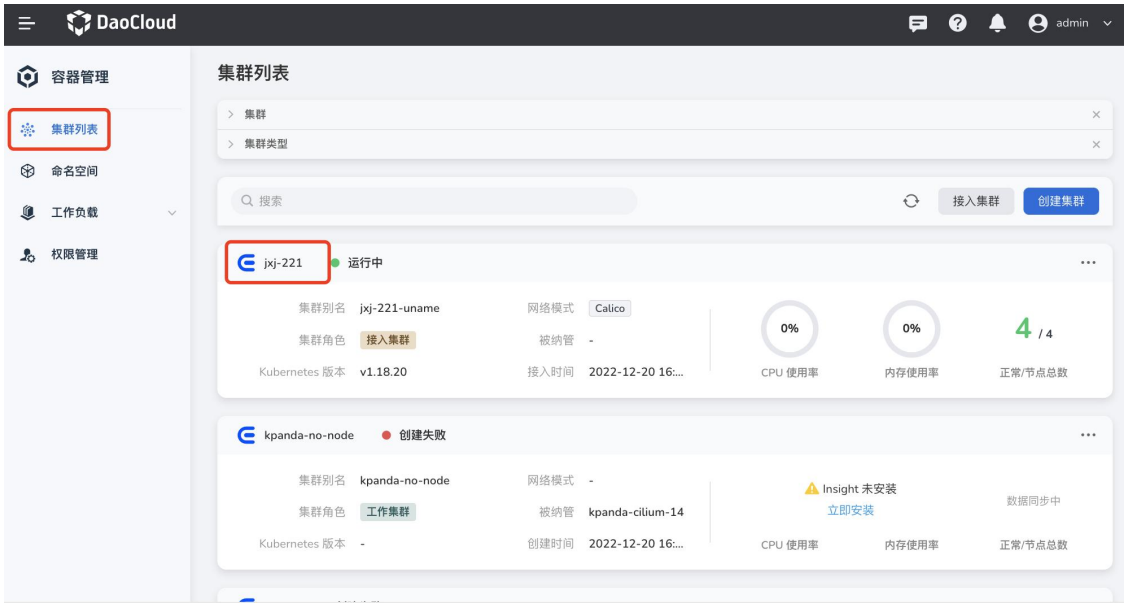
可以点击`添加`按钮为工作负载和容器组添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy16.png)

YAML 创建

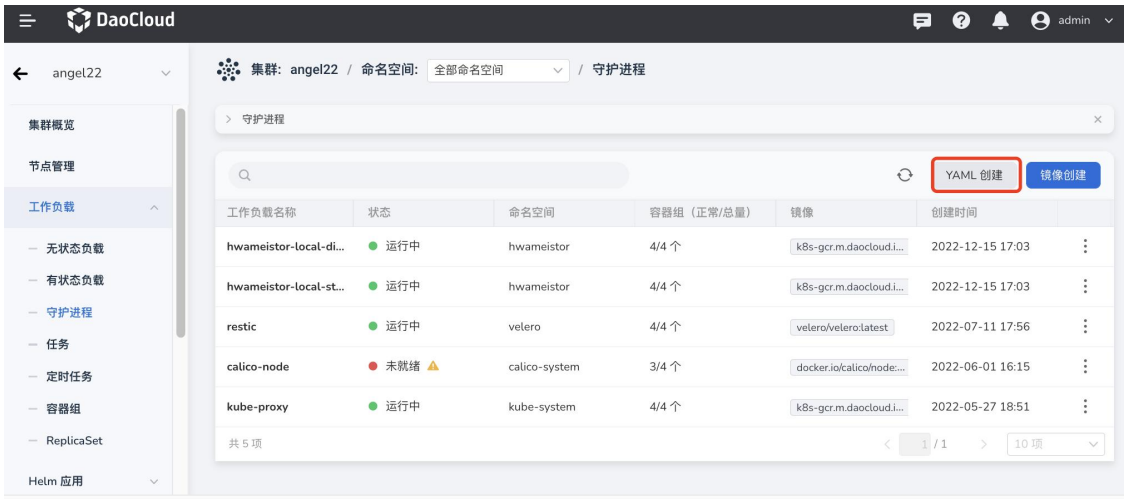
除了通过镜像方式外，还可以通过 YAML 文件更快速地创建创建守护进程。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载 -> 守护进程，然后点击页面右上角的 YAML 创建按钮。



工作负载

3. 输入或粘贴事先准备好的 YAML 文件，点击确定即可完成创建。



工作负载

??? note “点击查看创建守护进程的 YAML 示例”

```yaml
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: hwameistor-local-disk-manager
  namespace: hwameistor
  uid: ccbdc098-7de3-4a8a-96dd-d1cee159c92b
  resourceVersion: '90999552'
  generation: 1
  creationTimestamp: '2022-12-15T09:03:44Z'
  labels:
    app.kubernetes.io/managed-by: Helm
  annotations:
    deprecated.daemonset.template.generation: '1'
    meta.helm.sh/release-name: hwameistor
    meta.helm.sh/release-namespace: hwameistor
spec:
  selector:
    matchLabels:
      app: hwameistor-local-disk-manager
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: hwameistor-local-disk-manager
    spec:
      volumes:
        - name: udev
          hostPath:
            path: /run/udev
            type: Directory
        - name: procmount
```

```
      hostPath:
        path: /proc
        type: Directory
    - name: devmount
      hostPath:
        path: /dev
        type: Directory
    - name: socket-dir
      hostPath:
        path: /var/lib/kubelet/plugins/disk.hwameistor.io
        type: DirectoryOrCreate
    - name: registration-dir
      hostPath:
        path: /var/lib/kubelet/plugins_registry/
        type: Directory
    - name: plugin-dir
      hostPath:
        path: /var/lib/kubelet/plugins
        type: DirectoryOrCreate
    - name: pods-mount-dir
      hostPath:
        path: /var/lib/kubelet/pods
        type: DirectoryOrCreate
  containers:
    - name: registrar
      image: k8s-gcr.m.daocloud.io/sig-storage/csi-node-driver-registrar:v2.
5.0
      args:
        - '--v=5'
        - '--csi-address=/csi/csi.sock'
        - >-
          --kubelet-registration-path=/var/lib/kubelet/plugins/disk.hwameist
```

```
or.io/csi.sock
            env:
              - name: KUBE_NODE_NAME
                valueFrom:
                  fieldRef:
                    apiVersion: v1
                    fieldPath: spec.nodeName
            resources: {}
            volumeMounts:
              - name: socket-dir
                mountPath: /csi
              - name: registration-dir
                mountPath: /registration
            lifecycle:
              preStop:
                exec:
                  command:
                    - /bin/sh
                    - '-c'
                    - >-
                      rm -rf /registration/disk.hwameistor.io
                      /registration/disk.hwameistor.io-reg.sock
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
            imagePullPolicy: IfNotPresent
          - name: manager
            image: ghcr.m.daocloud.io/hwameistor/local-disk-manager:v0.6.1
            command:
              - /local-disk-manager
            args:
              - '--endpoint=$(CSI_ENDPOINT)'
              - '--nodeid=$(NODENAME)'
```

```
    - '--csi-enable=true'
  env:
    - name: CSI_ENDPOINT
      value: unix://var/lib/kubelet/plugins/disk.hwameistor.io/csi.sock
    - name: NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: WATCH_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    - name: POD_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: NODENAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
    - name: OPERATOR_NAME
      value: local-disk-manager
  resources: {}
  volumeMounts:
    - name: udev
      mountPath: /run/udev
    - name: procmount
      readOnly: true
```

```
          mountPath: /host/proc
        - name: devmount
          mountPath: /dev
        - name: registration-dir
          mountPath: /var/lib/kubelet/plugins_registry
        - name: plugin-dir
          mountPath: /var/lib/kubelet/plugins
          mountPropagation: Bidirectional
        - name: pods-mount-dir
          mountPath: /var/lib/kubelet/pods
          mountPropagation: Bidirectional
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
restartPolicy: Always
terminationGracePeriodSeconds: 30
dnsPolicy: ClusterFirst
serviceAccountName: hwameistor-admin
serviceAccount: hwameistor-admin
hostNetwork: true
hostPID: true
securityContext: {}
schedulerName: default-scheduler
tolerations:
  - key: CriticalAddonsOnly
    operator: Exists
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoSchedule
  - key: node-role.kubernetes.io/master
```

```
        operator: Exists
        effect: NoSchedule
      - key: node-role.kubernetes.io/control-plane
        operator: Exists
        effect: NoSchedule
      - key: node.cloudprovider.kubernetes.io/uninitialized
        operator: Exists
        effect: NoSchedule
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
  revisionHistoryLimit: 10
status:
  currentNumberScheduled: 4
  numberMisscheduled: 0
  desiredNumberScheduled: 4
  numberReady: 4
  observedGeneration: 1
  updatedNumberScheduled: 4
  numberAvailable: 4
```

创建任务（Job）

本文介绍如何通过镜像和 YAML 文件两种方式创建任务（Job）。

任务（Job）适用于执行一次性任务。Job 会创建一个或多个 Pod，Job 会一直重新尝试执行 Pod，直到成功终止的 Pod 达到一定数量。成功终止的 Pod 达到指定的数量后，Job 也随之结束。删除 Job 时会一同清除该 Job 创建的所有 Pod。暂停 Job 时删除该 Job 中的所有活跃 Pod，直到 Job 被继续执行。有关任务（Job）的更多介绍，可参考 Job。

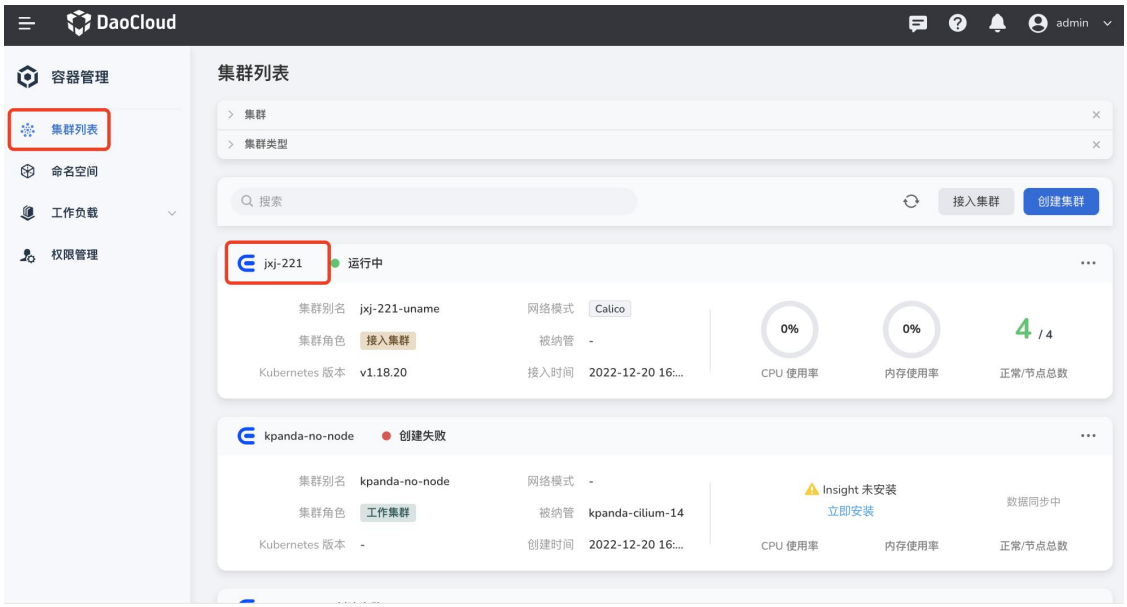
前提条件

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

镜像创建

参考以下步骤，使用镜像创建一个任务。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



集群详情

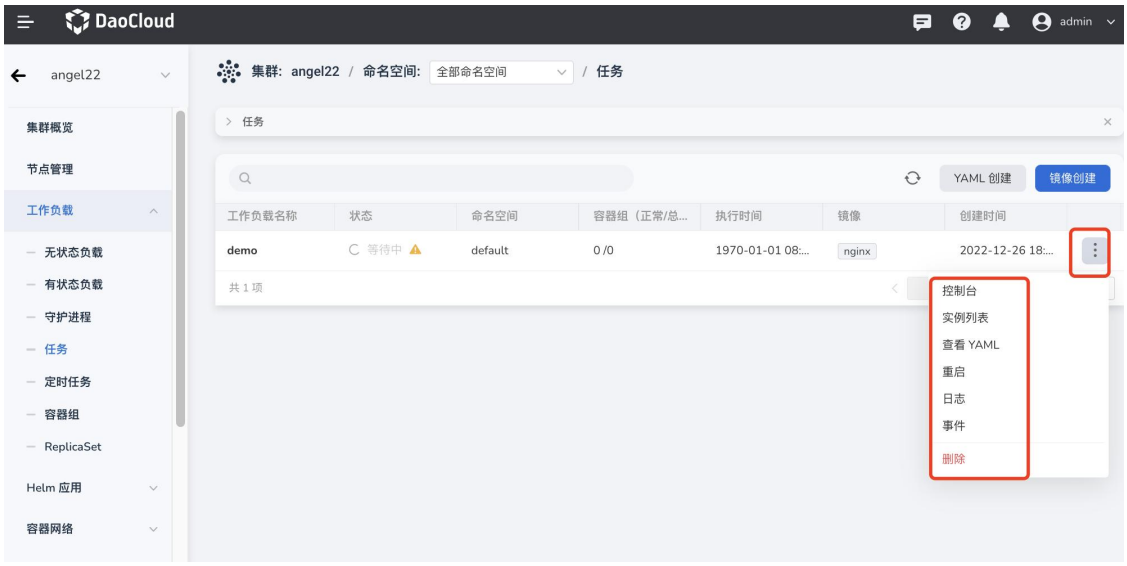
2. 在集群详情页面，点击左侧导航栏的工作负载 -> 任务，然后点击页面右上角的镜像创建按钮。



工作负载

3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击确定完成创建。

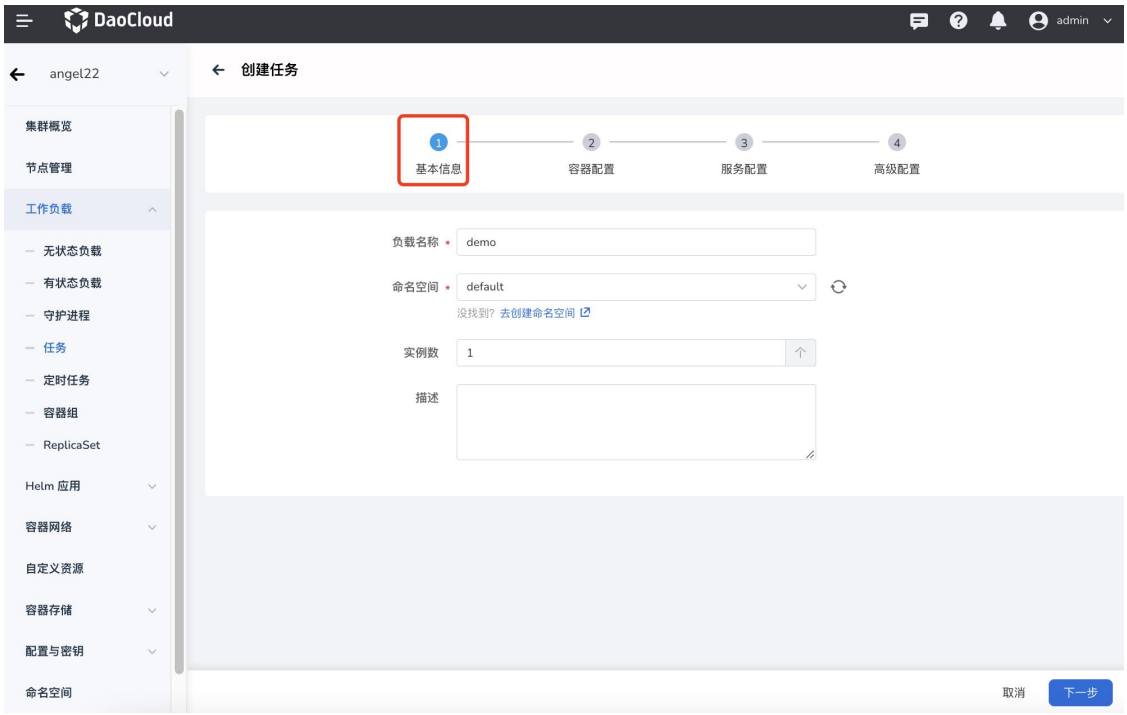
   系统将自动返回任务列表。点击列表右侧的 ：，可以对任务执行执行更新、删除、重启等操作。



操作菜单

基本信息

在创建任务页面中，根据下表输入基本信息后，点击下一步。



创建任务

- 负载名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内同一类型工作负载的名称不得重复，而且负载名称在工作负载创建好之后不可更改。
- 命名空间：选择将新建的任务部署在哪个命名空间，默认使用 default 命名空间。找不到所需的命名空间时可以根据页面提示去创建新的命名空间。
- 实例数：输入工作负载的 Pod 实例数量。默认创建 1 个 Pod 实例。
- 描述：输入工作负载的描述信息，内容自定义。字符数量应不超过 512 个。

容器配置

容器配置分为基本信息、生命周期、健康检查、环境变量、数据存储、安全设置六部分，点击下方的相应页签可查看各部分的配置要求。

> 容器配置仅针对单个容器进行配置，如需在一个容器组中添加多个容器，可点击右侧的 + 添加多个容器。

===“基本信息（必填）”

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/job02-1.png)

在配置容器相关参数时，必须正确填写容器的名称、镜像参数，否则将无法进入下一步。参考以下要求填写配置后，点击`确认`。

- 容器名称：最多包含 63 个字符，支持小写字母、数字及分隔符（“-”）。必须以小写字母或数字开头及结尾，例如 nginx-01。
- 容器镜像：输入镜像地址或名称。输入镜像名称时，默认从官方的 [DockerHub](https://hub.docker.com/）拉取镜像。接入 DCE 5.0 的[镜像仓库](../../../kangaroo/index.md)模块后，可以点击右侧的`选择镜像`来选择镜像。
- 更新策略：勾选`总是拉取镜像`后，负载每次重启/升级时都会从仓库重新拉取镜像。如果不勾选，则只拉取本地镜像，只有当镜像在本地不存在时才从镜像仓库重新拉取。更多详情可参考[镜像拉取策略](https://kubernetes.io/zh-cn/docs/concepts/containers/images/#image-pull-policy)。
- 特权容器：默认情况下，容器不可以访问宿主机上的任何设备，开启特权容器后，容器即可访问宿主机上的所有设备，享有宿主机上的运行进程的所有权限。
- CPU/内存配额：CPU/内存资源的请求值（需要使用的最小资源）和限制值（允许使用的最大资源）。请根据需要为容器配置资源，避免资源浪费和因容器资源超额导致系统故障。默认值如图所示。
- GPU 独享：为容器配置 GPU 用量，仅支持输入正整数。GPU 配额设置支持为容器设置独享整张 GPU 卡或部分 vGPU。例如，对于一张 8 核心的 GPU 卡，输入数字 `8` 表示让容器独享整长卡，输入数字 `1` 表示为容器配置 1 核心的 vGPU。

> 设置 GPU 独享之前，需要管理员预先在集群节点上安装 GPU 卡及驱动插件，并在[集群设置](../clusterops/cluster-settings.md)中开启 GPU 特性。

===“生命周期（选填）”

设置容器启动时、启动后、停止前需要执行的命令。详情可参考[容器生命周期配置](pod-config/lifecycle.md)。

![生命周期](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy06.png)

===“健康检查（选填）”

用于判断容器和应用的健康状态，有助于提高应用的可用性。详情可参考[容器健康检查配置](pod-config/health-check.md)。

![健康检查](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy07.png)

===“环境变量（选填）”

配置 Pod 内的容器参数，为 Pod 添加环境变量或传递配置等。详情可参考[容器环境变量配置](pod-config/env-variables.md)。

![环境变量](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy08.png)

===“数据存储（选填）”

配置容器挂载数据卷和数据持久化的设置。详情可参考[容器数据存储配置](pod-config/env-variables.md)。

![数据存储](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy09.png)

===“安全设置（选填）”

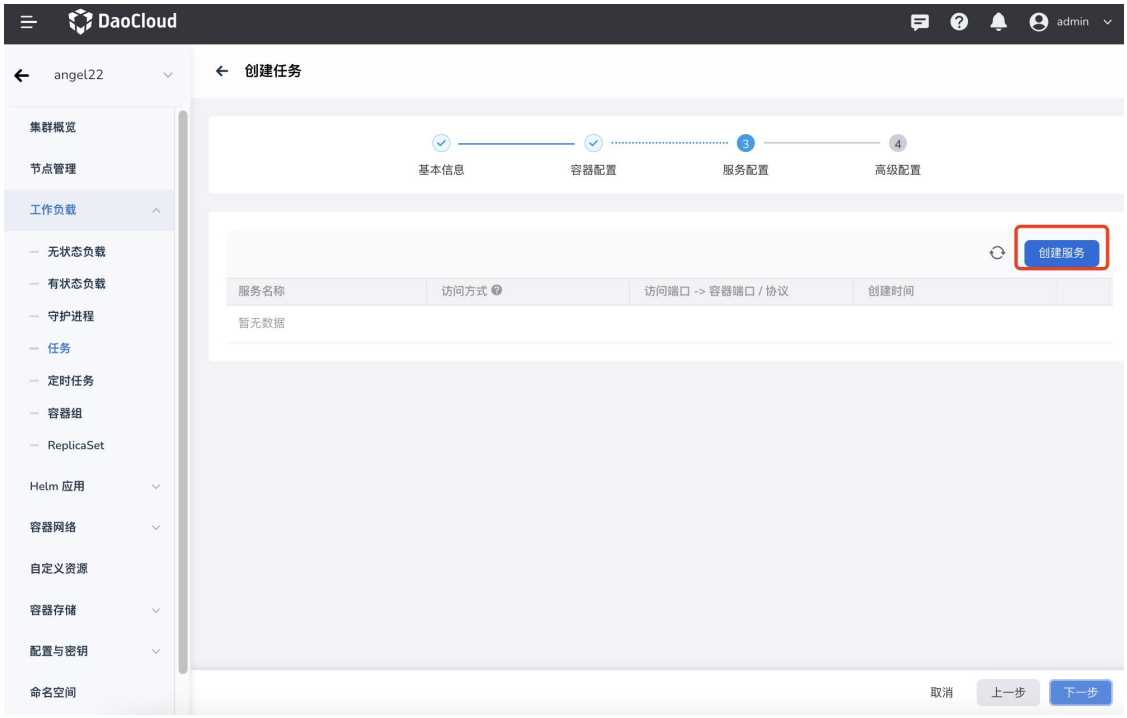
通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 `0` 表示使用 root 账号的权限。

![安全设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy10.png)

服务配置

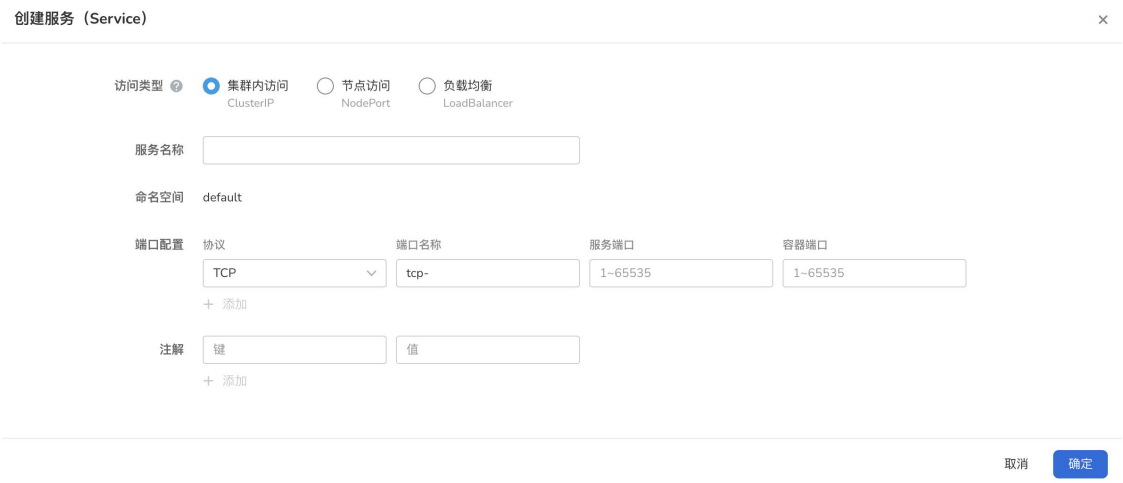
为 Job 配置服务（Service），使其能够被外部访问。

1. 点击创建服务按钮。



服务配置

2. 参考创建服务，配置服务参数。



创建服务

3. 点击确定，点击下一步。

高级配置

高级配置包括任务设置、标签与注解两部分。

===“任务设置”

![任务设置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/job03.png)

– 并行数：任务执行过程中允许同时创建的最大 Pod 数，并行数应不大于 Pod 总数。默认为 1。
– 超时时间：超出该时间时，任务会被标识为执行失败，任务下的所有 Pod 都会被删除。为空时表示不设置超时时间。
– 重启策略：设置失败时是否重启 Pod。

===“标签与注解”

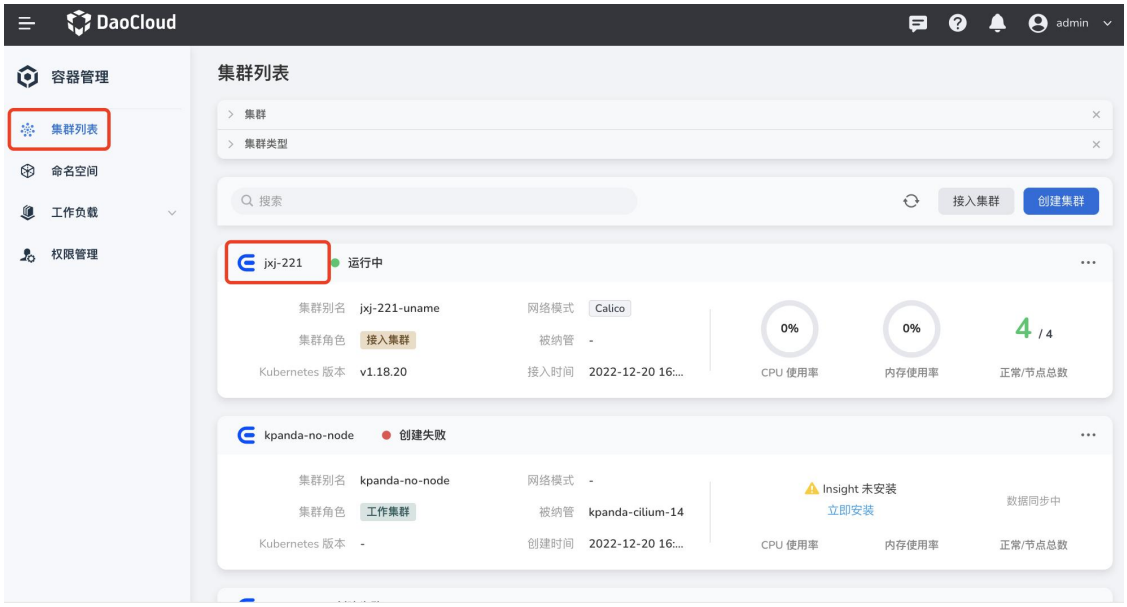
可以点击`添加`按钮为工作负载实例 Pod 添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/job04.png)

YAML 创建

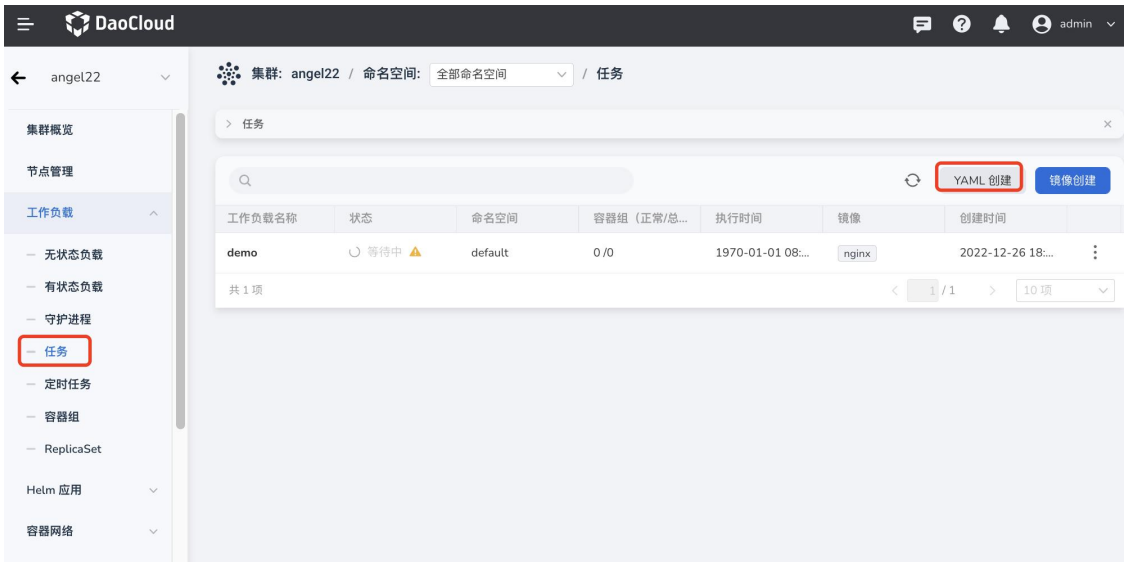
除了通过镜像方式外，还可以通过 YAML 文件更快速地创建创建任务。

1. 点击左侧导航栏上的集群列表，然后点击目标集群的名称，进入集群详情页面。



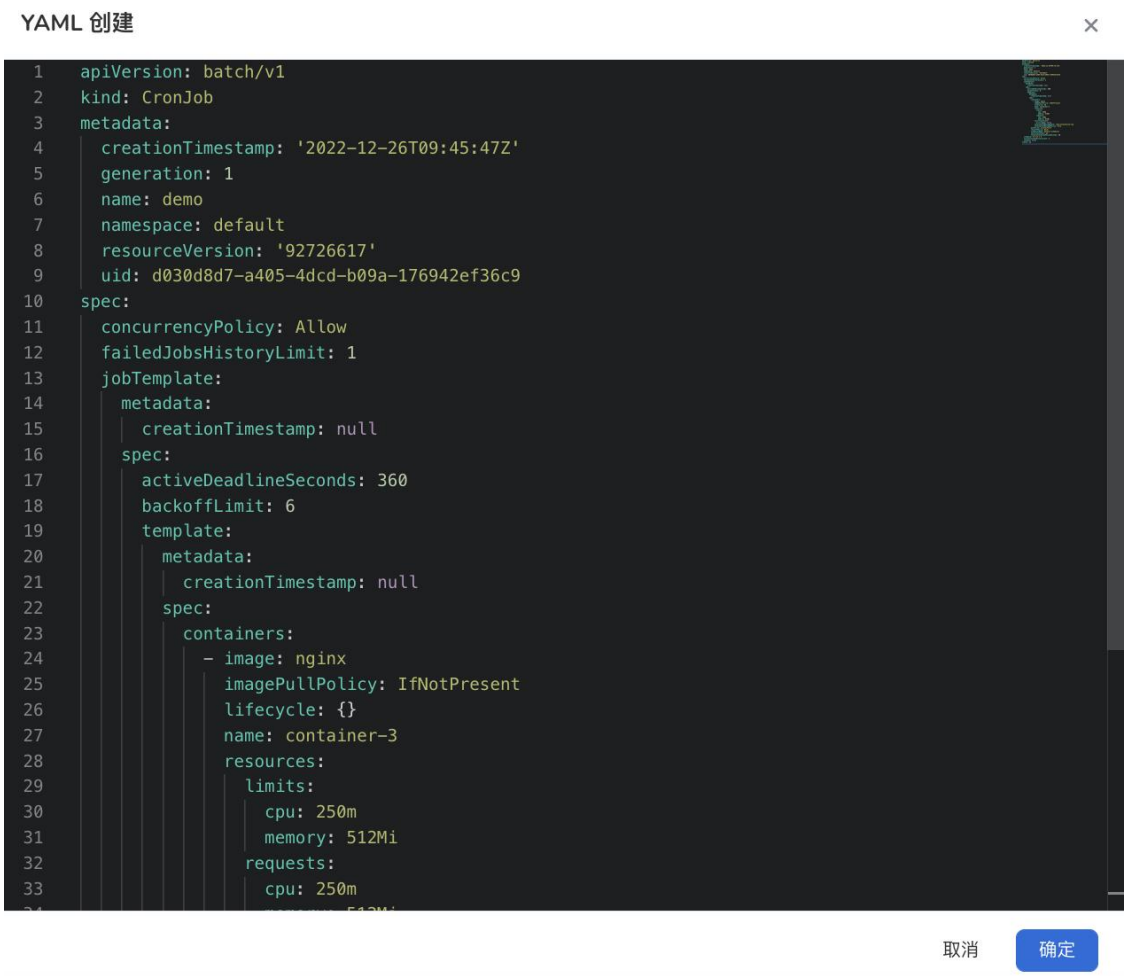
集群详情

2. 在集群详情页面，点击左侧导航栏的工作负载 -> 任务，然后点击页面右上角的 YAML 创建按钮。



工作负载

3. 输入或粘贴事先准备好的 YAML 文件，点击确定即可完成创建。



工作负载

??? note “点击查看创建任务的 YAML 示例”

```yaml
kind: Job
apiVersion: batch/v1
metadata:
  name: demo
  namespace: default
  uid: a9708239-0358-4aa1-87d3-a092c080836e
  resourceVersion: '92751876'
  generation: 1
  creationTimestamp: '2022-12-26T10:52:22Z'
```

```
  labels:
    app: demo
    controller-uid: a9708239-0358-4aa1-87d3-a092c080836e
    job-name: demo
  annotations:
    revisions: >-
      {"1":{"status":"running","uid":"a9708239-0358-4aa1-87d3-a092c080836e","
start-time":"2022-12-26T10:52:22Z","completion-time":"0001-01-01T00:00:00Z"}}
spec:
  parallelism: 1
  backoffLimit: 6
  selector:
    matchLabels:
      controller-uid: a9708239-0358-4aa1-87d3-a092c080836e
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: demo
        controller-uid: a9708239-0358-4aa1-87d3-a092c080836e
        job-name: demo
    spec:
      containers:
        - name: container-4
          image: nginx
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
```

```
            lifecycle: {}
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
            imagePullPolicy: IfNotPresent
            securityContext:
              privileged: false
        restartPolicy: Never
        terminationGracePeriodSeconds: 30
        dnsPolicy: ClusterFirst
        securityContext: {}
        schedulerName: default-scheduler
    completionMode: NonIndexed
    suspend: false
status:
    startTime: '2022-12-26T10:52:22Z'
    active: 1
```

工作负载状态

工作负载是运行在 Kubernetes 上的一个应用程序，在 Kubernetes 中，无论您的应用程序是由单个同一组件或是由多个不同的组件构成，都可以使用一组 Pod 来运行它。Kubernetes 提供了五种内置的工作负载资源来管理 Pod：

- 无状态工作负载
- 有状态工作负载
- 守护进程
- 任务
- 定时任务

您也可以通过设置自定义资源 CRD 来实现对工作负载资源的扩展。在第五代容器管理中，支持对工作负载进行创建、更新、扩容、监控、日志、删除、版本管理等全生命周期管理。

Pod 状态

Pod 是 Kuberneters 中创建和管理的、最小的计算单元，即一组容器的集合。这些容器共享存储、网络以及管理控制容器运行方式的策略。 Pod 通常不由用户直接创建，而是通过工作负载资源来创建。 Pod 遵循一个预定义的生命周期，起始于 Pending 阶段，如果至少其中有一个主要容器正常启动，则进入 Running，之后取决于 Pod 中是否有容器以失败状态结束而进入 Succeeded 或者 Failed 阶段。

工作负载状态

第五代容器管理模块依据 Pod 的状态、副本数等因素，设计了一种内置的工作负载生命周期的状态集，以让用户能够更加真实的感知工作负载运行情况。 由于不同的工作负载类型（比如无状态工作负载和任务）对 Pod 的管理机制不一致，因此，不同的工作负载在运行过程中会呈现不同的生命周期状态，具体如下表：

无状态负载、有状态负载、守护进程状态

状态	描述
等待中	1. 工作负载创建正在进行中，工作负载处于此状态。2. 触发升级或者回滚动作后，工作负载处于此状态。3. 触发暂停/扩缩容等操作，工作负载处在此状态。
运行中	负载下的所有实例都在运行中且副本数与用户预定义的数量一致时处于此状态。
删除中	执行删除操作时，负载处于此状态，直到删除完成。
异常	因为某些原因无法取得工作负载的状态。这种情况通常是因为与 Pod 所在主机通信失败。
未就绪	容器处于异常，pending 状态时，因未知错误导致负载无法启动时显示此状态

任务状态

状态	描述
等待中	任务创建正在进行中，工作负载处于此状态。
执行中	任务正在执行中，工作负载处于此状态。
执行完成	任务执行完成，工作负载处于此状态。
删除中	触发删除操作，工作负载处在此状态。
异常	因为某些原因无法取得 Pod 的状态。这种情况通常是因为与 Pod 所在主机通信失败。

定时任务状态

状态	描述
等待中	定时任务创建正在进行中，定时任务处于此状态。
已启动	创建定时任务成功后，正常运行或将已暂停的任务启动时定时任务处于此状态。

状态	描述
已停止	执行停止任务操作时，定时任务处于此状态。
删除中	触发删除操作，定时任务处在此状态。

当工作负载处于异常或未就绪状态时，您可以通过将鼠标移动到负载的状态值上，系统将通过提示框展示更加详细的错误信息。您也可以通过查看日志或事件来获取工作负载的相关运行信息。

配置容器生命周期

Pod 遵循一个预定义的生命周期，起始于 Pending 阶段，如果 Pod 内至少有一个容器正常启动，则进入 Running 状态。如果 Pod 中有容器以失败状态结束，则状态变为 Failed。以下 phase 字段值表明了一个 Pod 处于生命周期的哪个阶段。

值	描述
Pending（悬决）	Pod 已被系统接受，但有一个或者多个容器尚未创建亦未运行。这个阶段包括等待 Pod 被调度的时间和通过网络下载镜像的时间。
Running（运行中）	Pod 已经绑定到了某个节点，Pod 中的所有容器都已被创建。至少有一个容器仍在运行，或者正处于启动或重启状态。
Succeeded（成功）	Pod 中的所有容器都已成功终止，并且不会再重启。
Failed（失败）	Pod 中的所有容器都已终止，并且至少有一个容器是因为失败而终止。也就是说，容器以非 0 状态退出或者被系统终止。
Unknown（未知）	因为某些原因无法取得 Pod 的状态，这种情况通常是因为与 Pod 所在主机通信失败所致。

在 DCE 容器管理中创建一个工作负载时，通常使用镜像来指定容器中的运行环境。默认情况下，在构建镜像时，可以通过 Entrypoint 和 CMD 两个字段来定义容器运行时执行的命令和参数。如果需要更改容器镜像启动前、启动后、停止前的命令和参数，可以通过设置容器的生命周期事件命令和参数，来覆盖镜像中默认的命令和参数。

生命周期配置

根据业务需要对容器的启动命令、启动后命令、停止前命令进行配置。

参数	说明	举例值
启动命令	【类型】选填【含义】容器将按照启动命令进行启动。	
启动后命令	【类型】选填【含义】容器启动后出发的命令	
停止前命令	【类型】选填【含义】容器在收到停止命令后执行的命令。确保升级或实例删除时可提前将实例中运行的业务排水。	—

启动命令

根据下表对启动命令进行配置。

参数	说明	举例值
运行命令	【类型】必填【含义】输入可执行的命令，多个命令之间用空格进行分割，如命令本身带空格，则需要加（“”）。【含义】多命令时，运行命令建议用/bin/sh 或其他的 shell，其他全部命令作为参数来传入。	/run/server
运行	【类型】选填【含义】输入控制容器运行命令参数。	port=80

参数	说明	举例值
参数		80

启动后命令

DCE 提供命令行脚本和 HTTP 请求两种处理类型对启动后命令进行配置。您可以根据下表选择适合您的配置方式。

命令行脚本配置

参数	说明	举例值
运行命令	【类型】选填【含义】输入可执行的命令，多个命令之间用空格进行分割，如命令本身带空格，则需要加（“”）。【含义】多命令时，运行命令建议用/bin/sh 或其他的 shell，其他全部命令作为参数来传入。	/run/server
运行参数	【类型】选填【含义】输入控制容器运行命令参数。	port=80 80

停止前命令

DCE 提供命令行脚本和 HTTP 请求两种处理类型对停止前命令进行配置。您可以根据下表选择适合您的配置方式。

HTTP 请求配置

参数	说明	举例值
URL 路径	【类型】选填【含义】请求的 URL 路径。【含义】多命令时，运行命令建议用/bin/sh 或其他的 shell，其他全部命令作为参数来传入。	/run/server
端口	【类型】必填【含义】请求的端口。	port=80 80
节点地址	【类型】选填【含义】请求的 IP 地址，默认是容器所在的节点 IP。	-

调度策略

在 Kubernetes 集群中，节点也有标签。您可以手动添加标签。 Kubernetes 也会为集群中所有节点添加一些标准的标签。参见常用的标签、注解和污点以了解常见的节点标签。通过为节点添加标签，您可以让 Pod 调度到特定节点或节点组上。您可以使用这个功能来确保特定的 Pod 只能运行在具有一定隔离性，安全性或监管属性的节点上。

nodeSelector 是节点选择约束的最简单推荐形式。您可以将 nodeSelector 字段添加到 Pod 的规约中设置您希望目标节点所具有的节点标签。Kubernetes 只会将 Pod 调度到拥有指定每个标签的节点上。nodeSelector 提供了一种最简单的方法来将 Pod 约束到具有特定标签的节点上。亲和性和反亲和性扩展了您可以定义的约束类型。使用亲和性与反亲和性的一些好处有：

- 亲和性、反亲和性语言的表达能力更强。nodeSelector 只能选择拥有所有指定标签的节点。亲和性、反亲和性为您提供对选择逻辑的更强控制能力。
- 您可以标明某规则是“软需求”或者“偏好”，这样调度器在无法找到匹配节点时，会忽略亲和性/反亲和性规则，确保 Pod 调度成功。
- 您可以使用节点上（或其他拓扑域中）运行的其他 Pod 的标签来实施调度约束，而不是只能使用节点本身的标签。这个能力让您能够定义规则允许哪些 Pod 可以被放置在一起。

您可以通过设置亲和（affinity）与反亲和（anti-affinity）来选择 Pod 要部署的节点。

容忍时间

当工作负载实例所在的节点不可用时，系统将实例重新调度到其它可用节点的时间窗。默认为 300 秒。

节点亲和性（nodeAffinity）

节点亲和性概念上类似于 nodeSelector， 它使您可以根据节点上的标签来约束 Pod 可以调度到哪些节点上。 节点亲和性有两种：

- **必须满足：（requiredDuringSchedulingIgnoredDuringExecution）** 调度器只有在规则被满足的时候才能执行调度。此功能类似于 nodeSelector， 但其语法表达能力更强。您可以定义多条硬约束规则，但只需满足其中一条。
- **尽量满足：（preferredDuringSchedulingIgnoredDuringExecution）** 调度器会尝试寻找满足对应规则的节点。如果找不到匹配的节点，调度器仍然会调度该 Pod。您还可为软约束规则设定权重，具体调度时，若存在多个符合条件的节点，权重最大的节点会被优先调度。同时您还可以定义多条硬约束规则，但只需满足其中一条。

标签名

对应节点的标签，可以使用默认的标签也可以用户自定义标签。

操作符

- In：标签值需要在 values 的列表中
- NotIn：标签的值不在某个列表中
- Exists：判断某个标签是存在，无需设置标签值
- DoesNotExist：判断某个标签是不存在，无需设置标签值
- Gt：标签的值大于某个值（字符串比较）
- Lt：标签的值小于某个值（字符串比较）

权重

仅支持在“尽量满足”策略中添加，可以理解为调度的优先级，权重大的会被优先调度。取值范围是 1 到 100。

工作负载亲和性

与节点亲和性类似，工作负载的亲和性也有两种类型：

- **必须满足：（requiredDuringSchedulingIgnoredDuringExecution）** 调度器只有在规则被满足的时候才能执行调度。此功能类似于 nodeSelector， 但其语法表达能力更强。您可以定义多条硬约束规则，但只需满足其中一条。
- **尽量满足：（preferredDuringSchedulingIgnoredDuringExecution）** 调度器会尝试寻找满足对应规则的节点。如果找不到匹配的节点，调度器仍然会调度该 Pod。您还可为软约束规则设定权重，具体调度时，若存在多个符合条件的节点，权重最大的节点会被优先调度。同时您还可以定义多条硬约束规则，但只需满足其中一条。

工作负载的亲和性主要用来决定工作负载的 Pod 可以和哪些 Pod 部 署在同一拓扑域。例如，对于相互通信的服务，可通过应用亲和性调度，将其部署到同一拓扑域（如同一可用区）中，减少它们之间的网络延迟。

标签名

对应节点的标签，可以使用默认的标签也可以用户自定义标签。

命名空间

指定调度策略生效的命名空间。

操作符

- In：标签值需要在 values 的列表中
- NotIn：标签的值不在某个列表中
- Exists：判断某个标签是存在，无需设置标签值
- DoesNotExist：判断某个标签是不存在，无需设置标签值

拓扑域

指定调度时的影响范围。例如，如果指定为 kubernetes.io/Clustername 表示以 Node 节点为区分范围。

工作负载反亲和性

与节点亲和性类似，工作负载的反亲和性也有两种类型：

- **必须满足：（requiredDuringSchedulingIgnoredDuringExecution）** 调度器只有在规则被满足的时候才能执行调度。此功能类似于 nodeSelector， 但其语法表达能力更强。您可以定义多条硬约束规则，但只需满足其中一条。
- **尽量满足：（preferredDuringSchedulingIgnoredDuringExecution）** 调度器会尝试寻找满足对应规则的节点。如果找不到匹配的节点，调度器仍然会调度该 Pod。您还可为软约束规则设定权重，具体调度时，若存在多个符合条件的节点，权重最大的节点会被优先调度。同时您还可以定义多条硬约束规则，但只需满足其中一条。

工作负载的反亲和性主要用来决定工作负载的 Pod 不可以和哪些 Pod 部署在同一拓扑域。例如，将一个负载的相同 Pod 分散部署到不同的拓扑域（例如不同主机）中，提高负载本身的稳定性。

标签名

对应节点的标签，可以使用默认的标签也可以用户自定义标签。

命名空间

指定调度策略生效的命名空间。

操作符

- In：标签值需要在 values 的列表中
- NotIn：标签的值不在某个列表中
- Exists：判断某个标签是存在，无需设置标签值
- DoesNotExist：判断某个标签是不存在，无需设置标签值

拓扑域

指定调度时的影响范围。例如，如果指定为 kubernetes.io/Clustername 表示以 Node 节点为区分范围。

容器的健康检查

容器健康检查根据用户需求，检查容器的健康状况。配置后，容器内的应用程序入如果异常，容器会自动进行重启恢复。Kubernetes 提供了存活（Liveness）检查、就绪（Readiness）检查和启动（Startup）检查。

- **存活检查（LivenessProbe）** 可探测到应用死锁（应用程序在运行，但是无法继续执行后面的步骤）情况。 重启这种状态下的容器有助于提高应用的可用性，即使其中存在缺陷。
- **就绪检查（ReadinessProbe）** 可探知容器何时准备好接受请求流量，当一个 Pod 内的所有容器都就绪时，才能认为该 Pod 就绪。 这种信号的一个用途就是控制哪个 Pod 作为 Service 的后端。 若 Pod 尚未就绪，会被从 Service 的负载均衡器中剔除。
- **启动检查（StartupProbe）** 可以了解应用容器何时启动，配置后，可控制容器在启动成功后再进行存活性和就绪态检查， 确保这些存活、就绪探测器不会影响应用的启动。 启动探测可以用于对慢启动容器进行存活性检测，避免它们在启动运行之前就被杀掉。

存活和就绪检查

存活检查（LivenessProbe）的配置和就绪检查（ReadinessProbe）的配置参数相似， 唯一区别是要使用 readinessProbe 字段，而不是 livenessProbe 字段。

HTTP GET 参数说明：

参数	参数说明
路径（ Path）	访问的请求路径。如： 示例中的 /healthz 路径
端口(Port)	服务监听端口。 如： 示例中的 8080 端口
协议	访问协议，Http 或者 Https

参数	参数说明
延迟时间（initialDelaySeconds）	延迟检查时间，单位为秒，此设置与业务程序正常启动时间相关。例如，设置为 30，表明容器启动后 30 秒才开始健康检查，该时间是预留给业务程序启动的时间。
超时时间（timeoutSeconds）	超时时间，单位为秒。例如，设置为 10，表明执行健康检查的超时等待时间为 10 秒，如果超过这个时间，本次健康检查就被视为失败。若设置为 0 或不设置，默认超时等待时间为 1 秒。
超时时间（timeoutSeconds）	超时时间，单位为秒。例如，设置为 10，表明执行健康检查的超时等待时间为 10 秒，如果超过这个时间，本次健康检查就被视为失败。若设置为 0 或不设置，默认超时等待时间为 1 秒。
成功阈值（successThreshold）	探测失败后，被视为成功的最小连续成功数。默认值是 1，最小值是 1。存活和启动探测的这个值必须是 1。
最大失败次数（failureThreshold）	当探测失败时重试的次数。存活探测情况下的放弃就意味着重新启动容器。就绪探测情况下的放弃 Pod 会被打上未就绪的标签。默认值是 3。最小值是 1。

使用 HTTP GET 请求检查

YAML 示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
```

```yaml
containers:
- name: liveness
  image: k8s.gcr.io/liveness
  args:
  - /server
  livenessProbe:
    httpGet:
      path: /healthz   # 访问的请求路径
      port: 8080   # 服务监听端口
      httpHeaders:
      - name: Custom-Header
        value: Awesome
    initialDelaySeconds: 3   # kubelet 在执行第一次探测前应该等待 3 秒
    periodSeconds: 3    #kubelet 每隔 3 秒执行一次存活探测
```

按照设定的规则，Kubelet 向容器内运行的服务（服务在监听 8080 端口）发送一个 HTTP GET 请求来执行探测。如果服务器上 /healthz 路径下的处理程序返回成功代码，则 kubelet 认为容器是健康存活的。 如果处理程序返回失败代码，则 kubelet 会杀死这个容器并将其重启。返回大于或等于 200 并且小于 400 的任何代码都标示成功，其它返回代码都标示失败。 容器存活期间的最开始 10 秒中，/healthz 处理程序返回 200 的状态码。 之后处理程序返回 500 的状态码。

使用 TCP 端口检查

TCP 端口参数说明：

参数	参数说明
端口(Port)	服务监听端口。 如： 示例中的 8080 端口
延迟时间（initialDelaySeconds）	延迟检查时间，单位为秒，此设置与业务程序正常启动时间相关。例如，设置为 30，表明容器启动后 30 秒才开始健康检查，该时间是预留给业务程序启动的

参数	参数说明
	时间。
超时时间（timeoutSeconds）	超时时间，单位为秒。例如，设置为 10，表明执行健康检查的超时等待时间为 10 秒，如果超过这个时间，本次健康检查就被视为失败。若设置为 0 或不设置，默认超时等待时间为 1 秒。

对于提供 TCP 通信服务的容器，基于此配置，按照设定规则集群对该容器建立 TCP 连接，如果连接成功，则证明探测成功，否则探测失败。选择 TCP 端口探测方式，必须指定容器监听的端口。

YAML 示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
```

```
      port: 8080
    initialDelaySeconds: 15
    periodSeconds: 20
```

此示例同时使用就绪和存活探针。kubelet 在容器启动 5 秒后发送第一个就绪探测。 尝试连接 goproxy 容器的 8080 端口， 如果探测成功，这个 Pod 会被标记为就绪状态，kubelet 将继续每隔 10 秒运行一次检测。

除了就绪探测，这个配置包括了一个存活探测。 kubelet 会在容器启动 15 秒后进行第一次存活探测。 就绪探测会尝试连接 goproxy 容器的 8080 端口。 如果存活探测失败，容器会被重新启动。

执行命令检查

YAML 示例:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
```

```
      - /tmp/healthy
    initialDelaySeconds: 5 # kubelet 在执行第一次探测前等待 5 秒
    periodSeconds: 5   #kubelet 每 5 秒执行一次存活探测
```

periodSeconds 字段指定了 kubelet 每 5 秒执行一次存活探测，initialDelaySeconds 字段指定 kubelet 在执行第一次探测前等待 5 秒。按照设定规则，集群周期性的通过 kubelet 在容器内执行命令 cat /tmp/healthy 来进行探测。 如果命令执行成功并且返回值为 0，kubelet 就会认为这个容器是健康存活的。 如果这个命令返回非 0 值，kubelet 会杀死这个容器并重新启动它。

使用启动前检查保护慢启动容器

有些应用在启动时需要较长的初始化时间，需要使用相同的命令来设置启动探测，针对 HTTP 或 TCP 检测，可以通过将 failureThreshold * periodSeconds 参数设置为足够长的时间来应对启动需要较长时间的场景。

YAML 示例：

```
ports:
- name: liveness-port
  containerPort: 8080
  hostPort: 8080


livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10


startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
```

```
  failureThreshold: 30
  periodSeconds: 10
```

如上设置，应用将有最多 5 分钟（30 * 10 = 300s）的时间来完成启动过程，一旦启动探测成功，存活探测任务就会接管对容器的探测，对容器死锁作出快速响应。 如果启动探测一直没有成功，容器会在 300 秒后被杀死，并且根据 restartPolicy 来 执行进一步处置。

配置环境变量

环境变量是指容器运行环境中设定的一个变量，用于给 Pod 添加环境标志或传递配置等，支持通过键值对的形式为 Pod 配置环境变量。

DCE 容器管理在原生 Kubernetes 的基础上增加了图形化界面为 Pod 配置环境变量，支持以下几种配置方式：

- **键值对**（Key/Value Pair）：将自定义的键值对作为容器的环境变量
- **资源引用**（Resource）：将 Container 定义的字段作为环境变量的值，例如容器的内存限制、副本数等
- **变量/变量引用**（Pod Field）：将 Pod 字段作为环境变量的值，例如 Pod 的名称
- **配置项键值导入**（ConfigMap key）：导入配置项中某个键的值作为某个环境变量的值
- **密钥键值导入**（Secret Key）：使用来自 Secret 中的数据定义环境变量的值
- **密钥导入**（Secret）：将 Secret 中的所有键值都导入为环境变量
- **配置项导入**（ConfigMap）：将配置项中所有键值都导入为环境变量

任务参数说明

根据 .spec.completions 和 .spec.Parallelism 的设置，可以将任务（Job）划分为以下几种类型:

Job 类型	说明
非并行 Job	创建一个 Pod 直至其 Job 成功结束
具有确定完成计数的并行 Job	当成功的 Pod 个数达到 .spec.completions 时，Job 被视为完成
并行 Job	创建一个或多个 Pod 直至有一个成功结束

参数说明

RestartPolicy	创建一个 Pod 直至其成功结束
.spec.completions	表示 Job 结束需要成功运行的 Pod 个数，默认为 1
.spec.parallelism	表示并行运行的 Pod 的个数，默认为 1
spec.backoffLimit	表示失败 Pod 的重试最大次数，超过这个次数不会继续重试。
.spec.activeDeadlineSeconds	表示 Pod 运行时间，一旦达到这个时间，Job 即其所有的 Pod 都会停止。且 activeDeadlineSeconds 优先级高于 backoffLimit，即到达 activeDeadlineSeconds 的 Job 会忽略 backoffLimit 的设置。

以下是一个 Job 配置示例，保存在 myjob.yaml 中，其计算 π 到 2000 位并打印输出。

```
apiVersion: batch/v1
kind: Job                #当前资源的类型
```

```yaml
metadata:
  name: myjob
spec:
  completions: 50          # Job 结束需要运行 50 个 Pod，这个示例中就是打印π 50
次
  parallelism: 5          # 并行 5 个 Pod
  backoffLimit: 5          # 最多重试 5 次
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print  bpi(2000)"]
      restartPolicy: Never  #重启策略
```

相关命令

```
kubectl  apply  -f  myjob.yaml   #启动  job
kubectl  get  job  #查看这个 job
kubectl  logs  myjob-1122dswzs  查看 Job  Pod  的日志
```

ETCD 备份

ETCD 备份是以集群数据为核心的备份。在硬件设备损坏，开发测试配置错误等场景中，可以通过 ETCD 备份恢复集群数据。

本文介绍如何为集群制作 ETCD 备份。

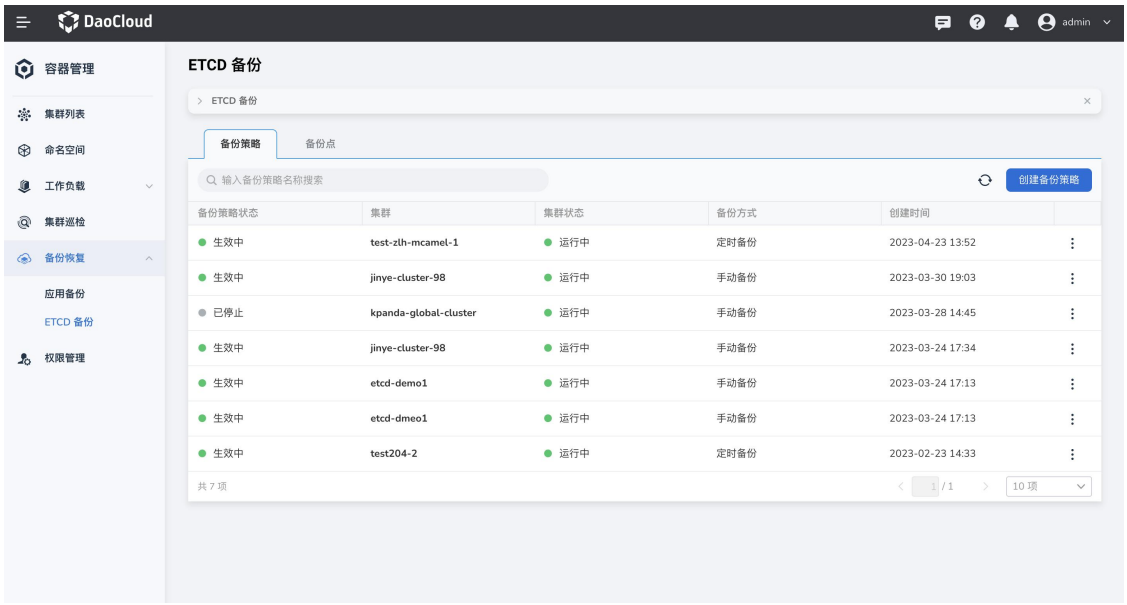
前提条件

- 接入或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建命名空间和用户，并为用户授予 NS Admin 或更高权限。详情可参考命名空间授权。
- 准备一个 MinIO 实例。建议通过 DCE 5.0 的 MinIO 中间件进行创建，具体步骤可参考 MinIO 对象存储。

创建 ETCD 备份

参照以下步骤创建 ETCD 备份。

1. 进入容器管理 -> 备份恢复 -> ETCD 备份，点击备份策略页签，然后在右侧点击创建备份策略。



备份策略列表

2. 参考以下说明填写基本信息。填写完毕后点击下一步，系统将自动校验 ETCD 的联通性，校验通过之后可以进行下一步。

    - 备份集群：选择需要备份哪个集群的 ETCD 数据，并在终端登录
    - ETCD 地址：格式为 https://${节点 IP}:${端口号}。

        - 在标准 Kubernetes 集群中，ETCD 的默认端口号为 2379。
        - 在 DCE 4.0 集群中，ETCD 的默认端口号为 12379。
        - 在公有云托管集群中，需要联系相关开发人员获取 ETCD 的端口号。这是因为公有云集群的控制面组件由云服务提供商维护和管理，用户无法直接访问或查看这些组件，也无法通过常规命令（如 kubectl）无法获取到控制面的端口等信息。

        ??? note “获取端口号的方式”

            (1) 在 `kube-system` 命名空间下查找 ETCD Pod

            ```shell
            ```

kubectl get po -n kube-system | grep etcd
```

(2) 获取 ETCD Pod 的 `listen-client-urls` 中的端口号

```shell
kubectl get po -n kube-system ${etcd_pod_name} -oyaml | grep listen-client-urls # 将 `etcd_pod_name` 替换为实际的 Pod 名称
```

预期输出结果如下，节点 IP 后的数字即为端口号:

```shell
- --listen-client-urls=https://127.0.0.1:2379,https://10.6.229.191:2379
```

- CA 证书：可通过如下命令查看证书，然后将证书内容复制粘贴到对应位置：

===“标准的 Kubernetes 集群”

```shell
cat /etc/kubernetes/ssl/etcd/ca.crt
```

===“DCE 4.0 集群”

```shell
cat /etc/daocloud/dce/certs/ca.crt
```
```

- Cert 证书：可通过如下命令查看证书，然后将证书内容复制粘贴到对应位置：

  ===“标准的 Kubernetes 集群”

  ```shell
  cat /etc/kubernetes/ssl/apiserver-etcd-client.crt
  ```

  ===“DCE 4.0 集群”

  ```shell
  cat /etc/daocloud/dce/certs/etcd/server.crt
  ```

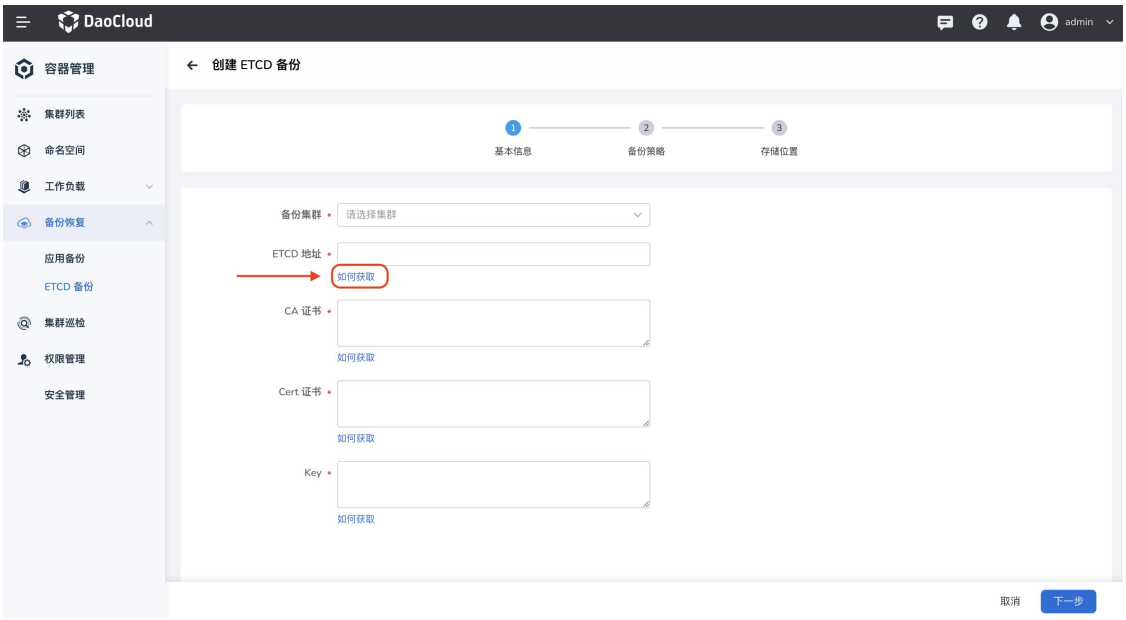
- Key：可通过如下命令查看证书，然后将证书内容复制粘贴到对应位置：

  ===“标准的 Kubernetes 集群”

  ```shell
  cat /etc/kubernetes/ssl/apiserver-etcd-client.key
  ```

  ===“DCE 4.0 集群”

  ```shell
  cat /etc/daocloud/dce/certs/etcd/server.key
  ```



创建基本信息

!!! note

点击输入框下方的`如何获取`可以在 UI 页面查看获取对应信息的方式。

3. 参考以下信息填写备份策略。

   - 备份方式：选择手动备份或定时备份

     - 手动备份：基于备份配置立即执行一次 ETCD 全量数据的备份。
     - 定时备份：按照设置的备份频率对 ETCD 数据进行周期性全量备份。

   - 备份链长度：最多保留多少条备份数据。默认为 30 条。

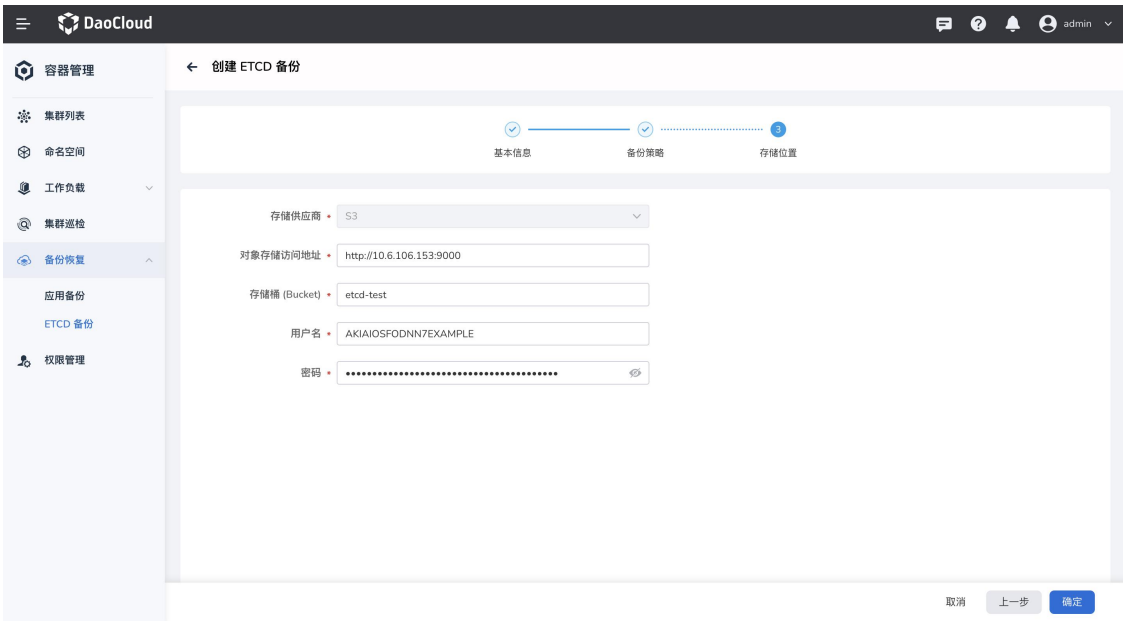
   - 备份频率：支持小时、日、周、月级别和自定义方式。



定时备份

4. 参考以下信息填写存储位置。

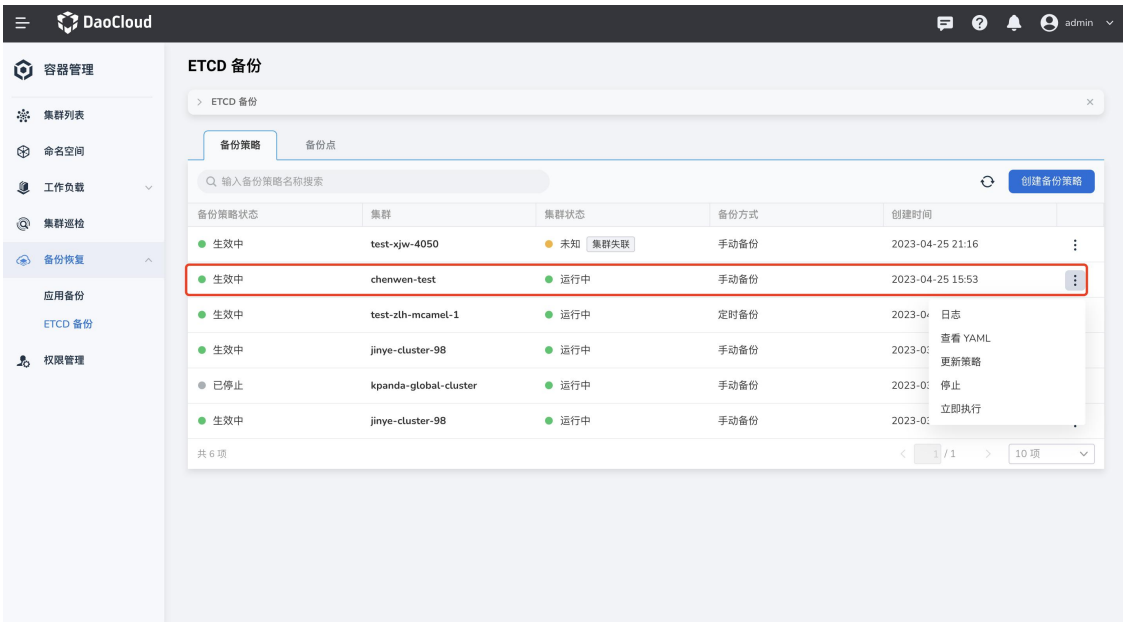
   - 存储供应商：默认选择 S3 存储
   - 对象存储访问地址：MinIO 的访问地址
   - 存储桶：在 MinIO 中创建一个 Bucket，填写 Bucket 的名称
   - 用户名：MinIO 的登录用户名
   - 密码：MinIO 的登录密码



存储位置

5. 点击确定后页面自动跳转到备份策略列表，可以查看目前创建好的所有策略。

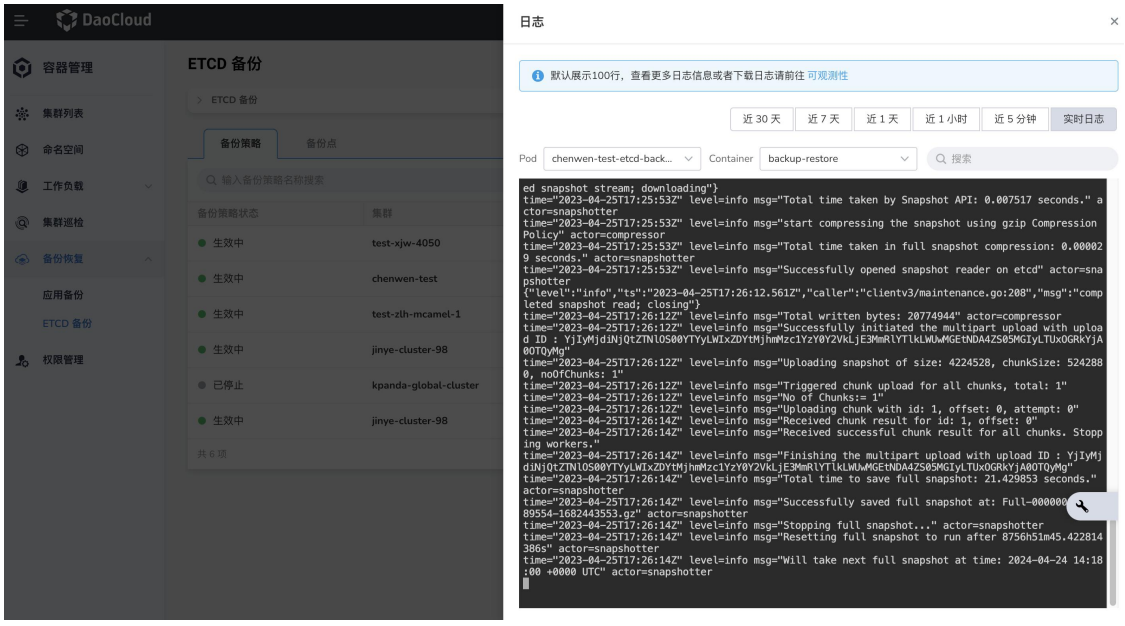
    - 在策略右侧点击　　操作按钮可以查看日志、查看 YAML、更新策略、停止策略、立即执行策略等。
    - 当备份方式为手动时，可以点击立即执行进行备份。
    - 当备份方式为定时备份时，则会根据配置的时间进行备份。



成功创建

查看备份策略日志

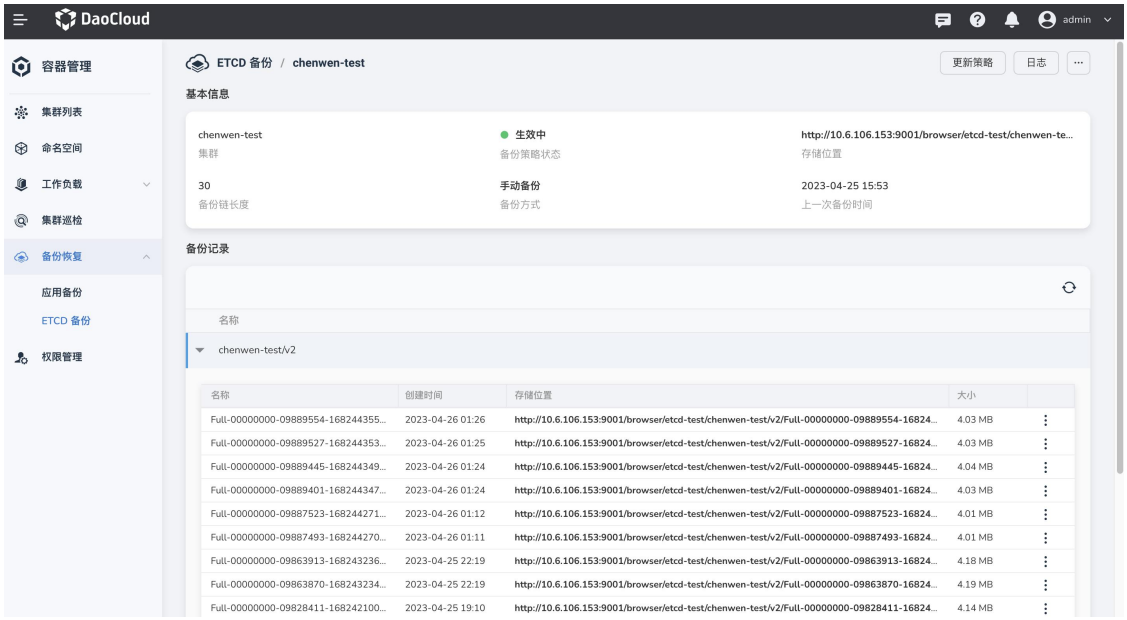
点击日志可以查看日志内容，默认展示 100 行。若想查看更多日志信息或者下载日志，可在日志上方根据提示前往可观测性模块。



查看日志

查看备份策略详情

进入容器管理 –> 备份恢复 –> ETCD 备份，点击备份策略页签，接着点击策略名称可以查看策略详情。

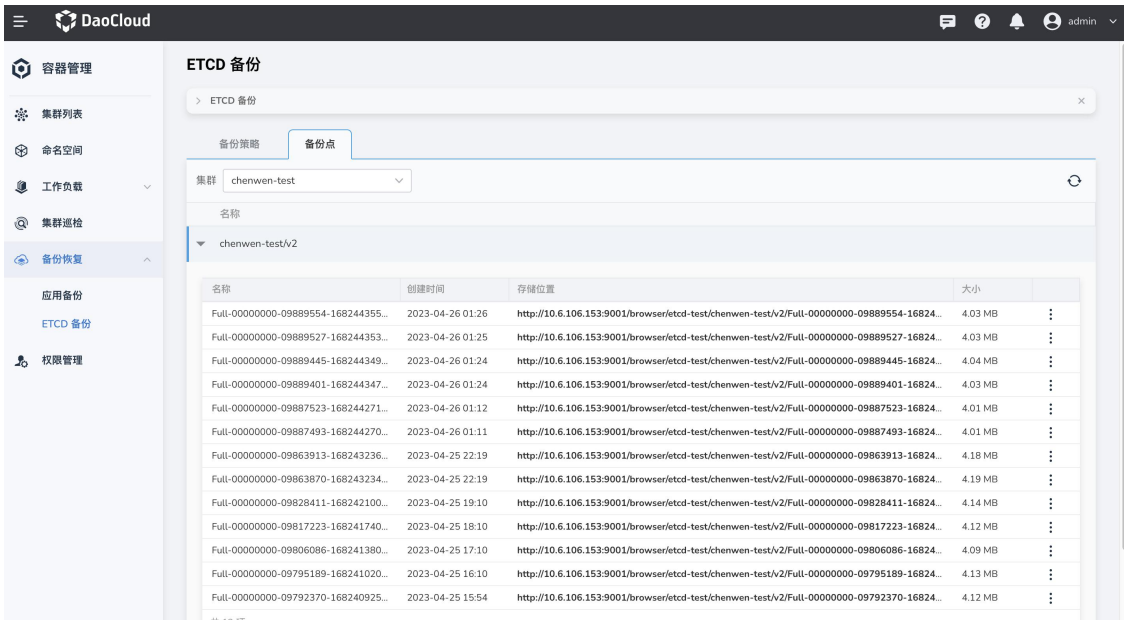


备份策略详情

查看备份点

1. 进入容器管理 –> 备份恢复 –> ETCD 备份，点击备份点页签。
2. 选择目标集群后，可以查看该集群下所有备份信息。

   每执行一次备份，对应生成一个备份点，可通过成功状态的备份点快速恢复应用。


DaoCloud
admin
容器管理
集群列表
命名空间
工作负载
集群巡检
备份恢复
应用备份
ETCD 备份
权限管理
ETCD 备份
ETCD 备份
备份策略
备份点
集群
chenwen-test
名称
chenwen-test/v2
名称
创建时间
存储位置
大小
Full-00000000-09889554-168244355... 2023-04-26 01:26 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09889554-16824... 4.03 MB
Full-00000000-09889527-168244353... 2023-04-26 01:25 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09889527-16824... 4.03 MB
Full-00000000-09889445-168244349... 2023-04-26 01:24 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09889445-16824... 4.04 MB
Full-00000000-09889401-168244347... 2023-04-26 01:24 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09889401-16824... 4.03 MB
Full-00000000-09887523-168244271... 2023-04-26 01:12 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09887523-16824... 4.01 MB
Full-00000000-09887493-168244270... 2023-04-26 01:11 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09887493-16824... 4.01 MB
Full-00000000-09863913-168243236... 2023-04-25 22:19 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09863913-16824... 4.18 MB
Full-00000000-09863870-168243234... 2023-04-25 22:19 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09863870-16824... 4.19 MB
Full-00000000-09828411-168242100... 2023-04-25 19:10 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09828411-16824... 4.14 MB
Full-00000000-09817223-168241740... 2023-04-25 18:10 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09817223-16824... 4.12 MB
Full-00000000-09806086-168241380... 2023-04-25 17:10 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09806086-16824... 4.09 MB
Full-00000000-09795189-168241020... 2023-04-25 16:10 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09795189-16824... 4.13 MB
Full-00000000-09792370-168240925... 2023-04-25 15:54 http://10.6.106.153:9001/browser/etcd-test/chenwen-test/v2/Full-00000000-09792370-16824... 4.12 MB

备份点

备份恢复

备份恢复分为备份和恢复两方面，实际应用时需要先备份系统在某一时点的数据，然后安全存储地备份数据。后续如果出现数据损坏、丢失、误删等事故，就可以基于之前的数据备份快速还原系统，**缩短故障时间，减少损失**。

- 在真实的生产环境中，服务可能分布式地部署在不同的云、不同区域或可用区，如果某一个基础设施自身出现故障，企业需要在其他可用环境中快速恢复应用。在这种情况下，跨云/跨集群的备份恢复显得非常重要。
- 在大规模系统中往往有很多角色和用户，权限管理体系复杂，操作者众多，难免有人误操作导致系统故障。在这种情况下，也需要能够通过之前备份的数据快速回滚系统，否则如果依赖人为排查故障、修复故障、恢复系统就会耗费大量时间，系统不可用时间越长，企业的损失越大。
- 此外，还有网络攻击、自然灾害、设备故障等各种因素也可能导致数据事故

因此，备份恢复非常重要，可以视之为维护系统稳定和数据安全的最后一道保险。

备份通常分为全量备份、增量备份、差异备份三种。DCE 5.0 目前支持全量备份和增量备份。

DCE 5.0 提供的备份恢复可以分为 **应用备份** 和 **ETCD 备份** 两种，支持手动备份，或基于 CronJob 定时自动备份。

- 应用备份

  应用备份指，备份集群中的某个工作负载的数据，然后将该工作负载的数据恢复到本集群或者其他集群。支持备份整个命名空间下的所有资源，也支持通过标签选择器过滤，仅备份带有特定标签的资源。

  应用备份支持跨集群备份有状态应用，具体步骤可参考 MySQL 应用及数据的跨集群备份恢复。

- ETCD 备份

  etcd 是 Kubernetes 的数据存储组件，Kubernetes 将自身的组件数据和其中的应用数据都存储在 etcd 中。因此，备份 etcd 就相当于备份整个集群的数据，可以在故障时快速将集群恢复到之前某一时点的状态。

  需要注意的是，目前仅支持将 etcd 备份数据恢复到同一集群（原集群）。如需了解与此相关的最佳实践，可参考 ETCD 备份还原。

应用备份

本文介绍如何在 DCE 5.0 中为应用做备份，本教程中使用的演示应用名为 dao-2048，属于无状态工作负载。

前提条件

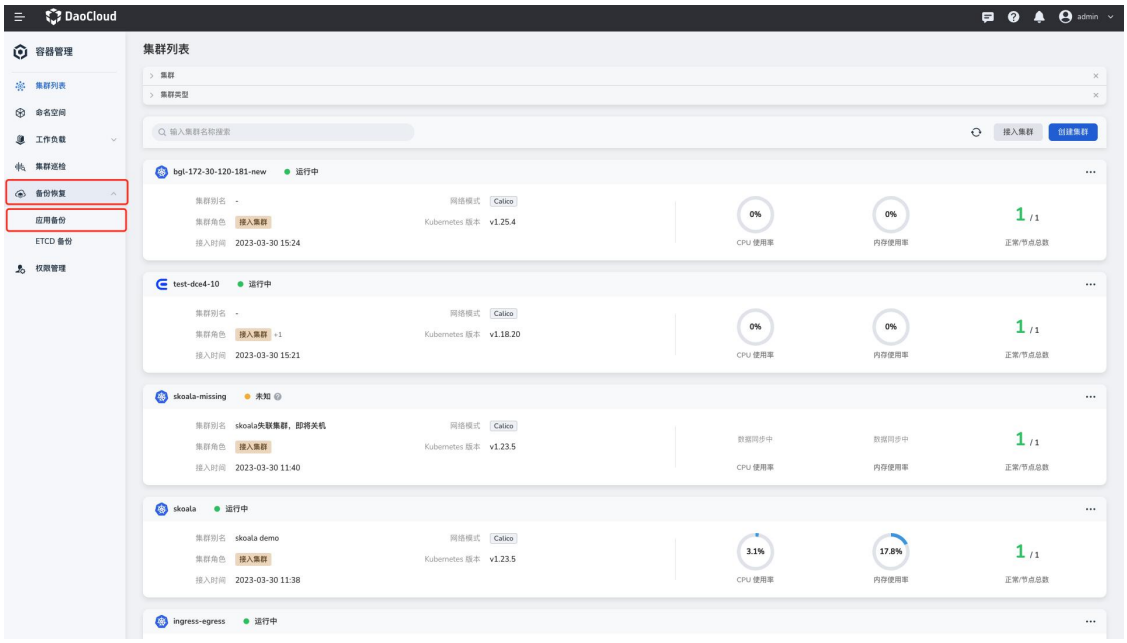
在对无状态工作负载进行备份前，需要满足以下前提条件：

- 在容器管理模块中接入 Kubernetes 集群或者创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 创建一个命名空间和用户。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。
- 安装 velero 组件，且 velero 组件运行正常。
- 创建一个无状态工作负载（本教程中的负载名为 dao-2048），并为无状态工作负载打上 app: dao-2048 的标签。

备份工作负载

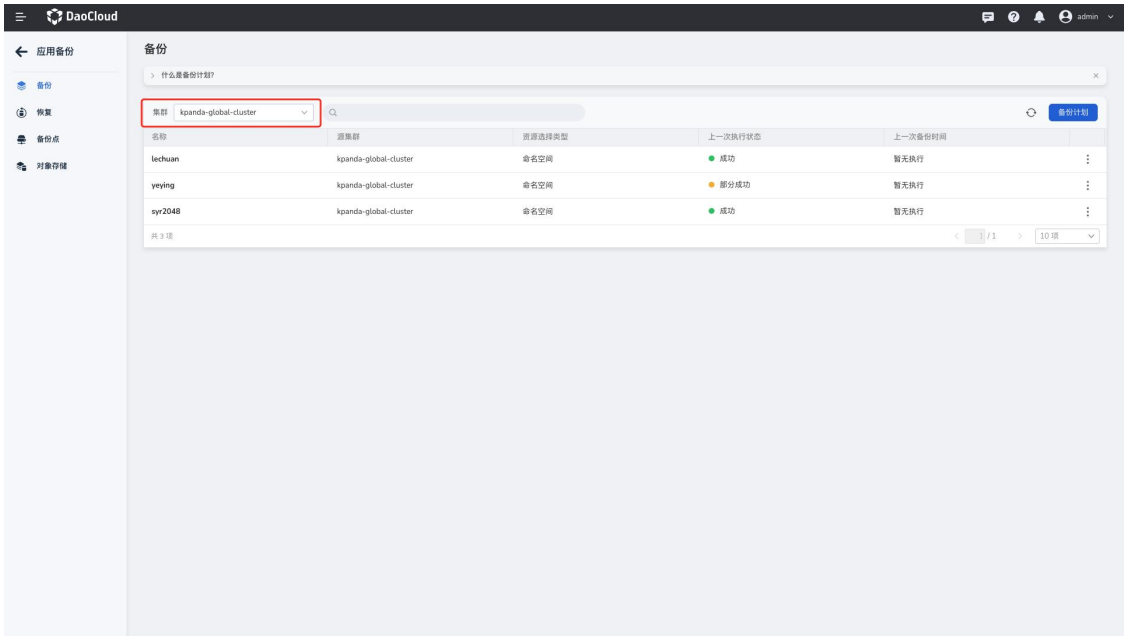
参考以下步骤，备份无状态工作负载 dao-2048。

1. 进入容器管理模块，点击左侧导航栏上的备份恢复 -> 应用备份，进入应用备份列表页面。



集群列表

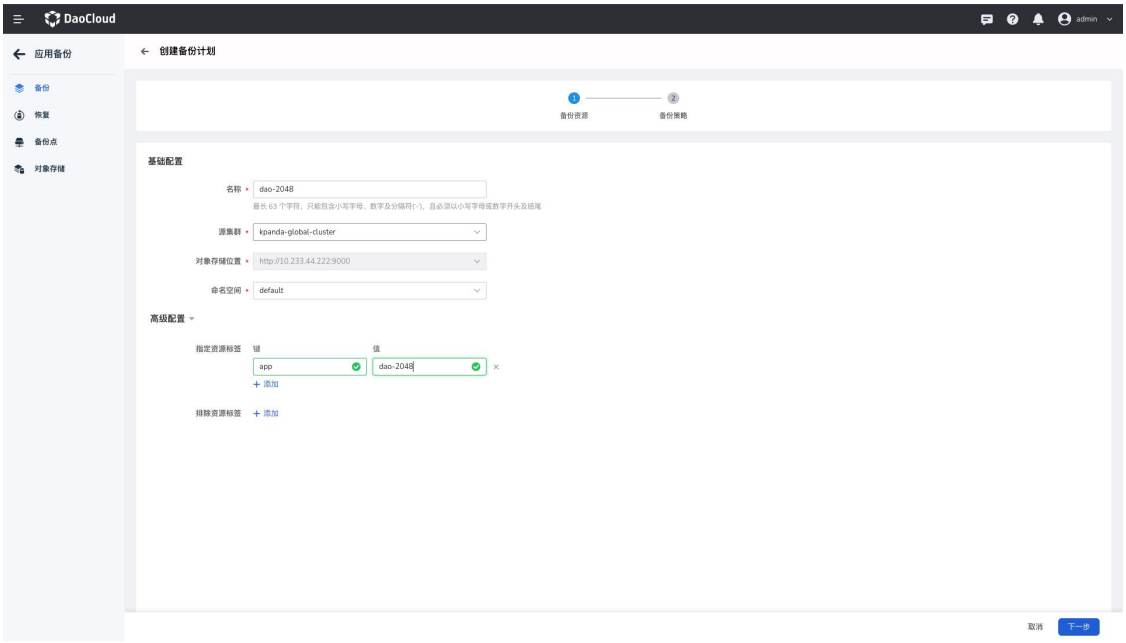
2. 在应用备份列表页面，选择已经安装了 velero 和 dao-2048 应用的集群。点击右上角的备份计划，新建一个备份集群。



应用备份

3. 参考下方说明填写备份配置。

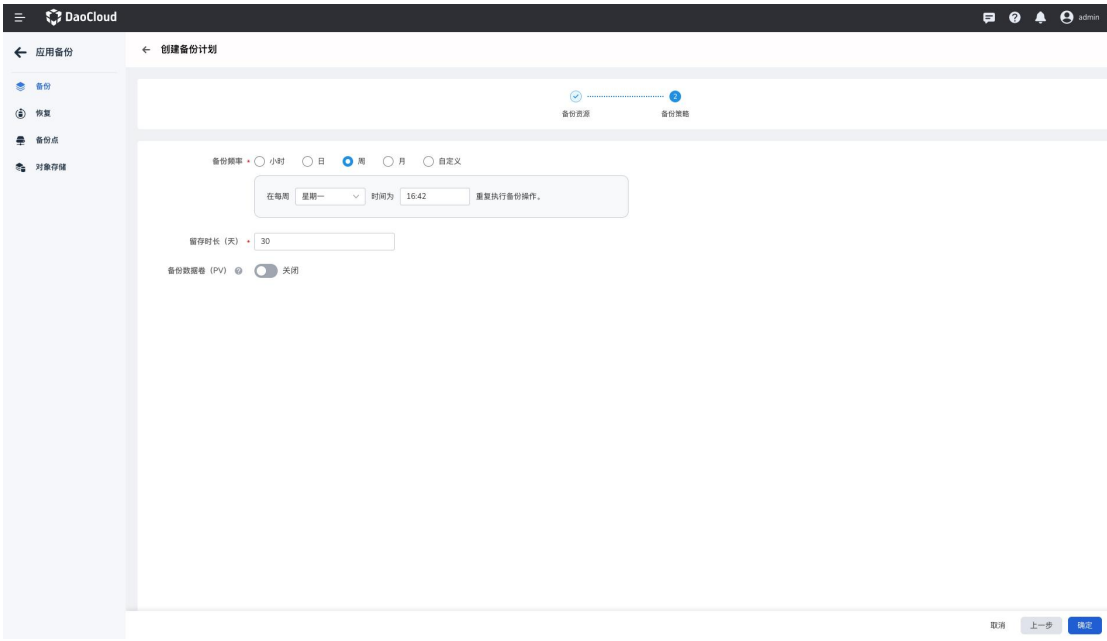
- 名称：新建备份计划的名称。
- 源集群：计划执行应用备份的集群。
- 对象存储位置：在源集群安装 velero 时配置的对象存储的访问路径。
- 命名空间：需要进行备份的命名空间，支持多选。
- 高级配置：根据资源标签对命名空间内的特定资源进行备份如某个应用，或者备份时根据资源标签对命名空间内的特定资源不进行备份。



操作菜单

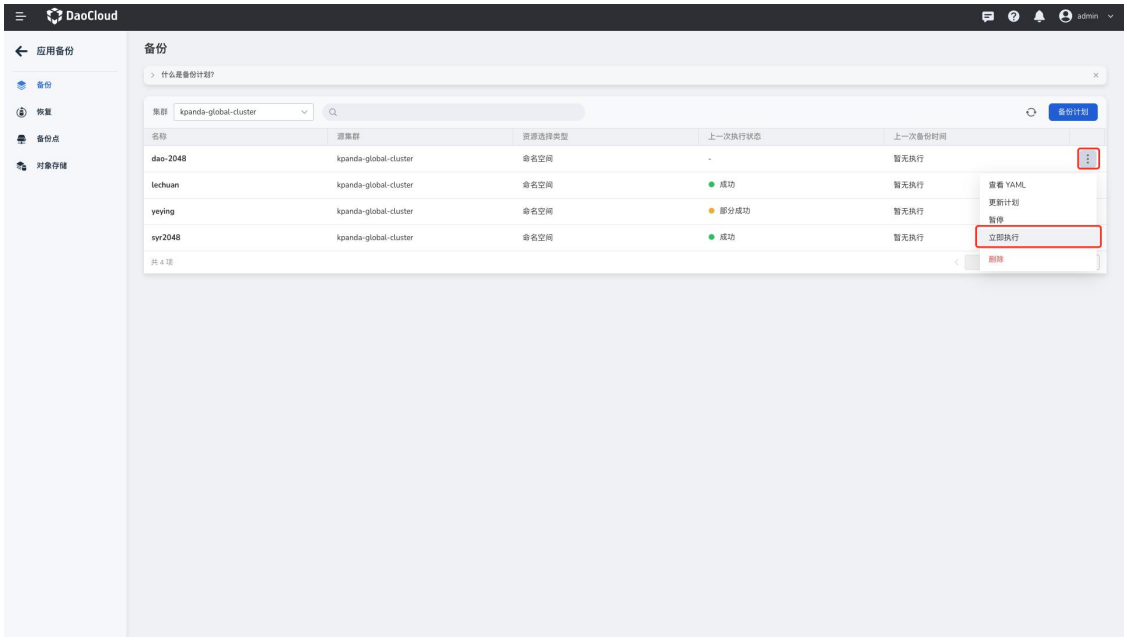
4. 参考下方说明设置备份执行频率，然后点击下一步。
   - 备份频率：基于分钟、小时、天、周、月设置任务执行的时间周期。支持用数字和 * 自定义 Cron 表达式，**输入表达式后下方会提示当前表达式的含义**。有关详细的表达式语法规则，可参考 Cron 时间表语法。

- 留存时长（天）：设置备份资源保存的时间，默认为 30 天，过期后将会被删除。
- 备份数据卷（PV）：是否备份数据卷（PV）中的数据，支持直接复制和使用 CSI 快照两种方式。
    - 直接复制：直接复制数据卷（PV）中的数据用于备份；
    - 使用 CSI 快照：使用 CSI 快照来备份数据卷（PV）。需要集群中有可用于备份的 CSI 快照类型。



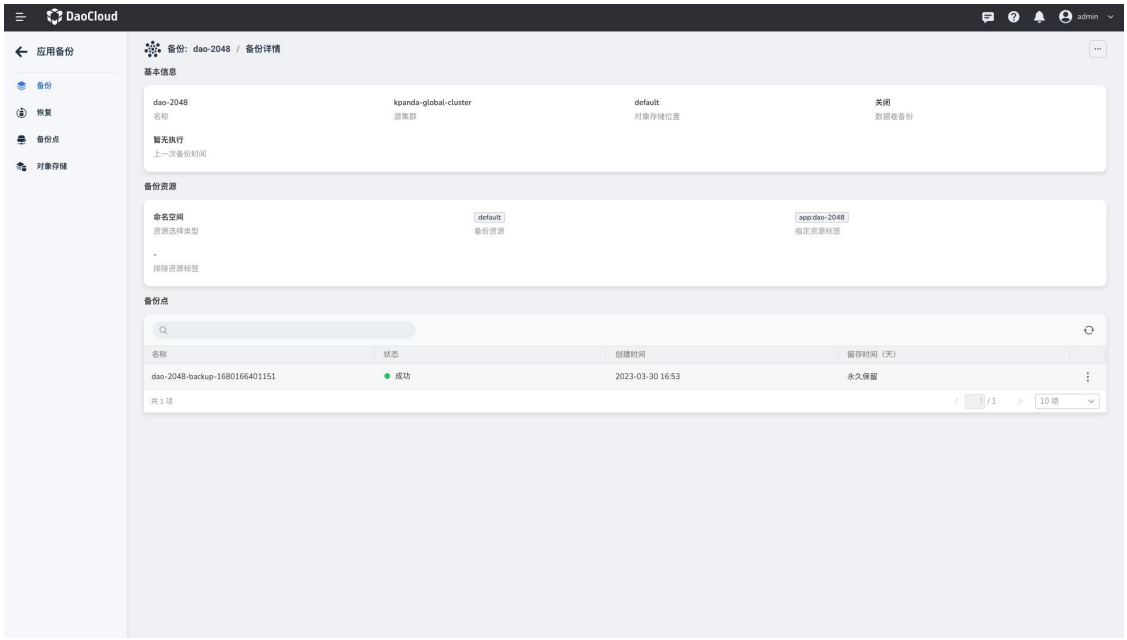
操作菜单

5. 点击 确定，页面会自动返回应用备份计划列表，找到新建的 dao-2048 备份计划，执行立即备份操作。



操作菜单

6. 此时集群的上一次执行状态将转变为备份中。等待备份完成后可以点击备份计划的名称，查看备份计划详情。



操作菜单

离线升级备份恢复模块

本页说明从下载中心下载备份恢复模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `kcoral` 字样是备份恢复模块的内部开发代号。

从下载的安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

方式一：使用 chart-syncer 同步镜像

使用 chart-syncer 可以将您下载的安装包中的 chart 及其依赖的镜像包上传至安装器部署 DCE 时使用的镜像仓库和 helm 仓库。

首先找到一台能够连接镜像仓库和 helm 仓库的节点（如火种节点），在节点上创建 load-image.yaml 配置文件，填入镜像仓库和 helm 仓库等配置信息。

1. 创建 load-image.yaml

   !!! note

   该 YAML 文件中的各项参数均为必填项。

   ===“已添加 Helm repo”

   若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

   ```yaml title="load-image.yaml"
   source:
     intermediateBundlesPath: kcoral # 节点上执行 load-image.yaml 文件的
   ```

路径。

```
  target:
    containerRegistry: 10.16.10.111 # 镜像仓库地址
    containerRepository: release.daocloud.io/kcoral # 镜像仓库路径
    repo:
      kind: HARBOR # Helm Chart 仓库类别
      url: http://10.16.10.111/chartrepo/release.daocloud.io # Helm 仓库地址
      auth:
      username: "admin" # 镜像仓库用户名
      password: "Harbor12345" # 镜像仓库密码
    containers:
      auth:
        username: "admin" # Helm 仓库用户名
        password: "Harbor12345" # Helm 仓库密码
```

===“未添加 Helm repo”

若当前节点上未添加 helm repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kcoral # 节点上执行 load-image.yaml 文件的路径。
target:
  containerRegistry: 10.16.10.111 # 镜像仓库 url
  containerRepository: release.daocloud.io/kcoral # 镜像仓库路径
  repo:
    kind: LOCAL
    path: ./local-repo # chart 本地路径
  containers:
    auth:
```

```
    username: "admin" # 镜像仓库用户名
    password: "Harbor12345" # 镜像仓库密码
```

2. 执行同步镜像命令。

   charts-syncer sync --config load-image.yaml

方式二：使用 Docker 或 containerd 加载镜像

解压并加载镜像文件。

1. 解压 tar 压缩包。

   tar xvf kcoral.bundle.tar

   解压成功后会得到 3 个文件：

   - hints.yaml
   - images.tar
   - original-chart

2. 从本地加载镜像到 Docker 或 containerd。

   === “Docker”

   ```shell
   docker load -i images.tar
   ```

   === “containerd”

   ```shell
   ctr image import images.tar
   ```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

===“通过 helm repo 升级”

1. 检查备份恢复 helm 仓库是否存在。

    ```shell
    helm repo list | grep kcoral
    ```

    若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

    ```none
    Error: no repositories to show
    ```

1. 添加备份恢复的 helm 仓库。

    ```shell
    heml repo add kcoral http://{harbor url}/chartrepo/{project}
    ```

1. 更新备份恢复的 helm 仓库。

    ```shell
    helm repo update kcoral
    ```

1. 选择您想安装的备份恢复版本（建议安装最新版本）。

    ```shell
    helm search repo kcoral/kcoral --versions
    ```

    ```none
    [root@master ~]# helm search repo kcoral/kcoral --versions
    NAME                    CHART VERSION   APP VERSION   DESCRIPTION
    kcoral/kcoral   0.20.0          v0.20.0         A Helm chart for kcoral
    ...
    ```

1. 备份 `--set` 参数。

    在升级备份恢复版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

    ```shell
    helm get values kcoral -n kcoral-system -o yaml > bak.yaml
    ```

1. 更新 kcoral crds

    ```shell
    helm pull kcoral/kcoral --version 0.5.0 && tar -zxf kcoral-0.5.0.tgz
    kubectl apply -f kcoral/crds
    ```

1. 执行 `helm upgrade`。

    升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像

仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kcoral kcoral/kcoral \
  -n kcoral-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version 0.5.0
```

===“通过 chart 包升级”

1. 备份 `--set` 参数。

   在升级备份恢复版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

   ```shell
   helm get values kcoral -n k pan da-system -o yaml > bak.yaml
   ```

1. 更新 kcoral crds

   ```shell
   kubectl apply -f ./crds
   ```

1. 执行 `helm upgrade`。

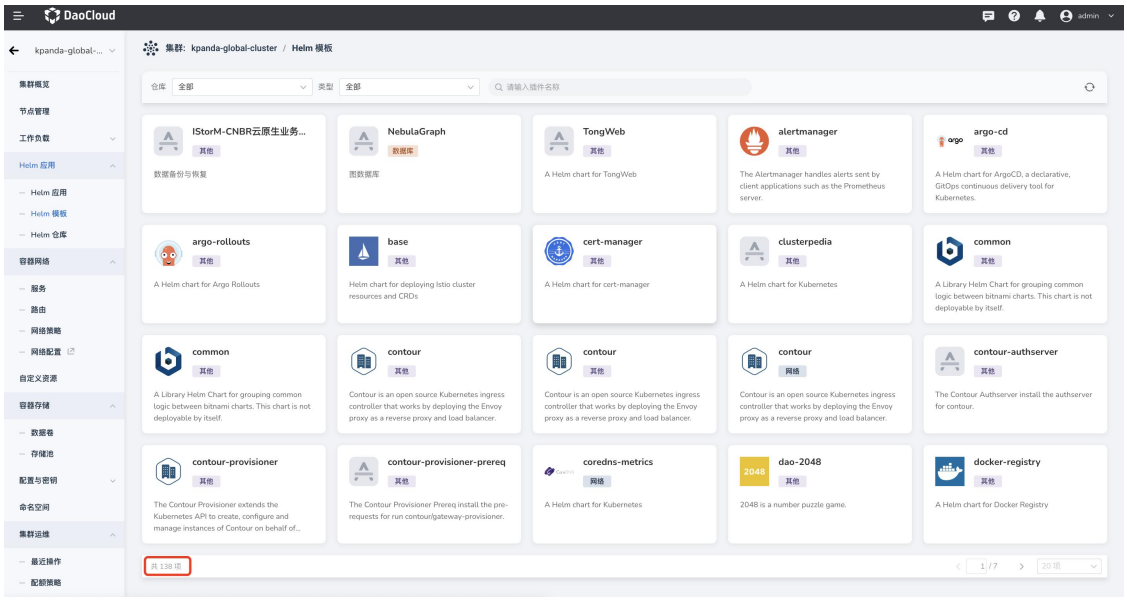
升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kcoral . \
  -n kcoral-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
```

Helm 模板

Helm 是 Kubernetes 的包管理工具，方便用户快速发现、共享和使用 Kubernetes 构建的应用。第五代容器管理模块提供了上百个 Helm 模板，涵盖存储、网络、监控、数据库等主要场景。借助这些模板，您可以通过 UI 界面快速部署、便捷管理 Helm 应用。此外，支持通过添加 Helm 仓库 添加更多的个性化模板，满足多样需求。



模板

关键概念：

使用 Helm 时需要了解以下几个关键概念：

- Chart：一个 Helm 安装包，其中包含了运行一个应用所需要的镜像、依赖和资源定义等，还可能包含 Kubernetes 集群中的服务定义，类似 Homebrew 中的 formula、APT 的 dpkg 或者 Yum 的 rpm 文件。**Chart 在 DCE 5.0 中称为 Helm 模板**。

- Release：在 Kubernetes 集群上运行的一个 Chart 实例。一个 Chart 可以在同一个集群内多次安装，每次安装都会创建一个新的 Release。**Release 在 DCE 5.0 中称为 Helm 应用**。
- Repository：用于发布和存储 Chart 的存储库。**Repository 在 DCE 5.0 中称为 Helm 仓库**。

更多详细信息，请前往 Helm 官网查看。

相关操作：

- 管理 Helm 应用，包括安装、更新、卸载 Helm 应用，查看 Helm 操作记录等。
- 管理 Helm 仓库，包括安装、更新、删除 Helm 仓库等。

管理 Helm 应用

容器管理模块支持对 Helm 进行界面化管理，包括使用 Helm 模板创建 Helm 实例、自定义 Helm 实例参数、对 Helm 实例进行全生命周期管理等功能。

本节将以 cert-manager 为例，介绍如何通过容器管理界面创建并管理 Helm 应用。

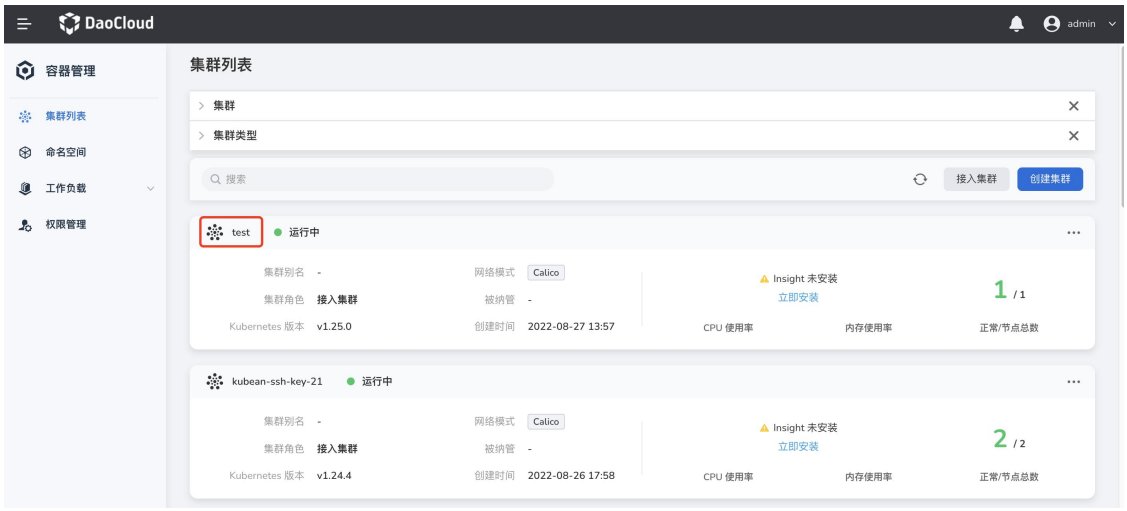
前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建、用户的创建，并为用户授予 NS Admin 或更高权限，详情可参考命名空间授权。

安装 Helm 应用

参照以下步骤安装 Helm 应用。

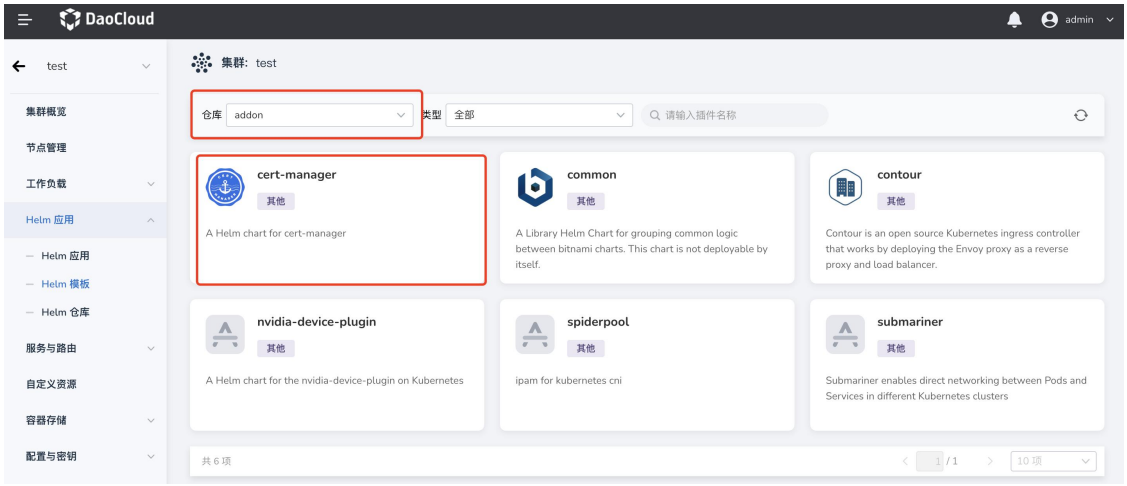
1. 点击一个集群名称，进入集群详情。



集群详情

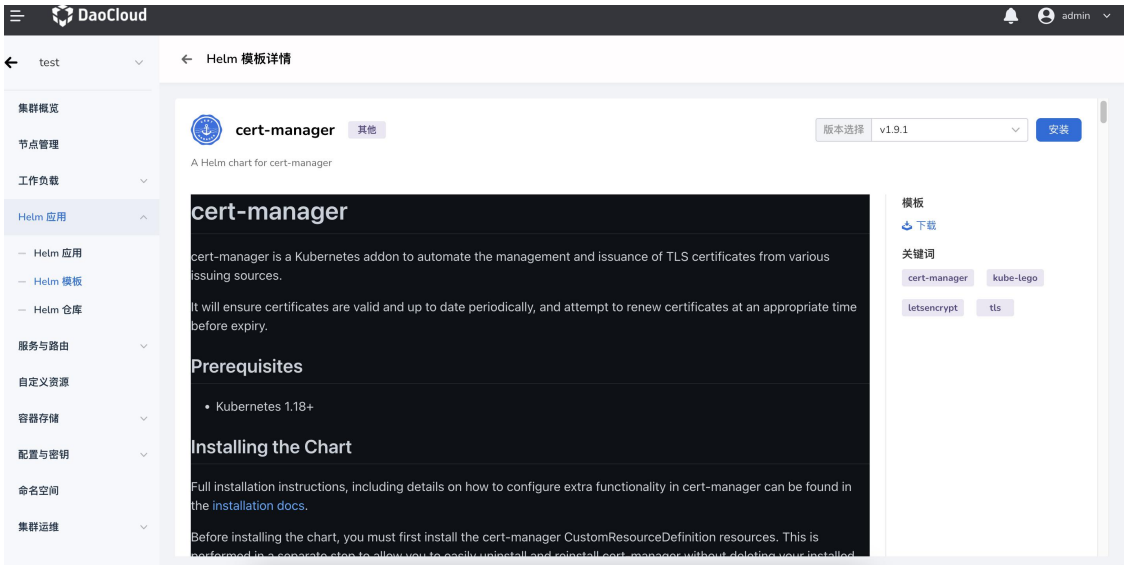
2. 在左侧导航栏，依次点击 Helm 应用 -> Helm 模板，进入 Helm 模板页面。

   在 Helm 模板页面选择名为 addon 的 Helm 仓库，此时界面上将呈现 addon 仓库下所有的 Helm chart 模板。 点击名称为 cert-manager 的 Chart。



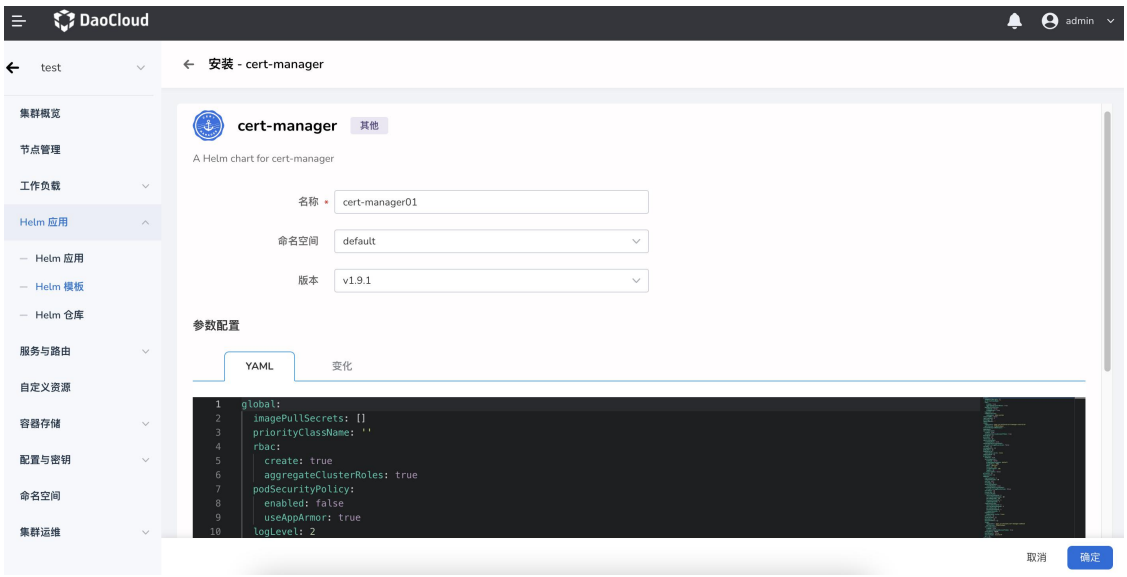
   找到 chart

3. 在安装页面，能够看到 Chart 的相关详细信息，在界面右上角选择需要安装的版本，点击安装按钮。此处选择 v1.9.1 版本进行安装。



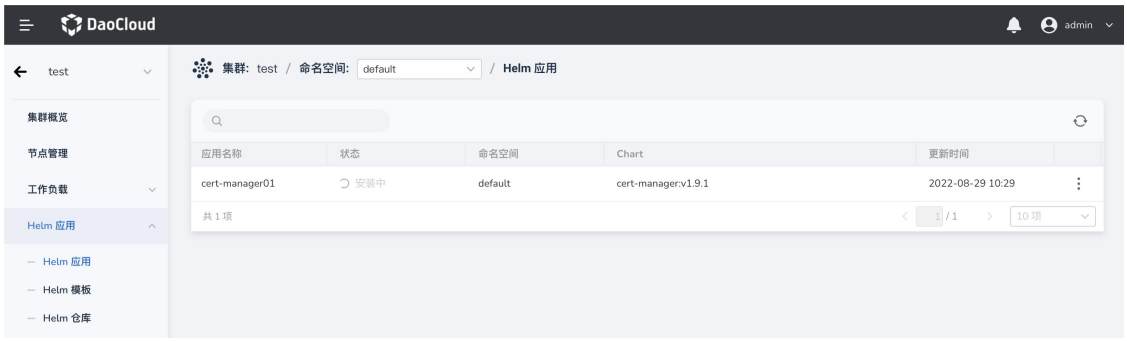
点击安装

4. 配置名称、命名空间及版本信息，也可以在下方的 **参数配置** 区域通过修改 YAML 来自定义参数。点击确定。



填写参数

5. 系统将自动返回 Helm 应用列表，新创建的 Helm 应用状态为安装中，等待一段时间后状态变为运行中。



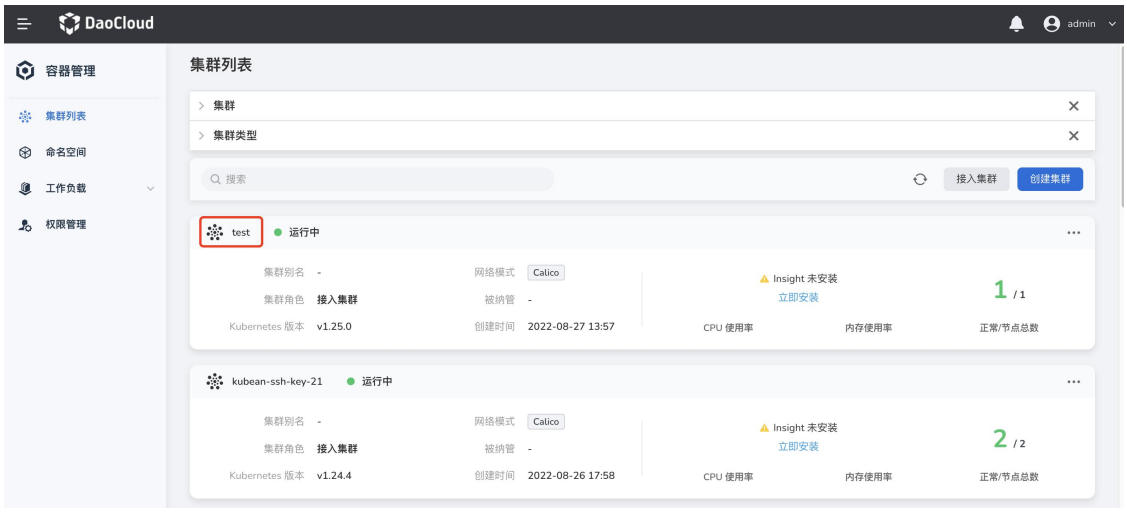
查看状态

更新 Helm 应用

当我们通过界面完成一个 Helm 应用的安装后，我们可以对 Helm 应用执行更新操作。注意：只有通过界面安装的 Helm 应用才支持使用界面进行更新操作。

参照以下步骤更新 Helm 应用。

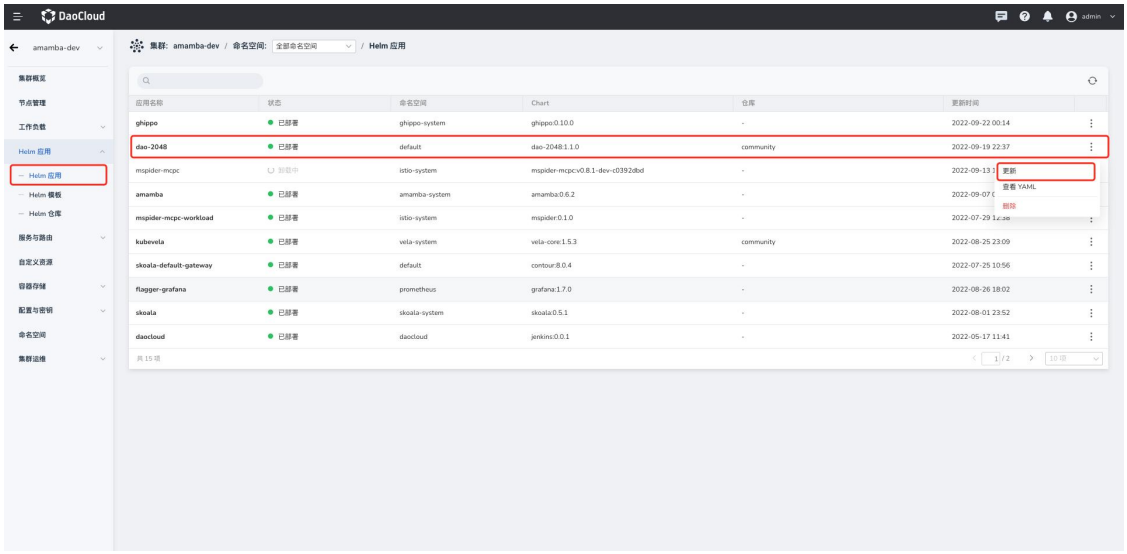
1. 点击一个集群名称，进入集群详情。



集群详情

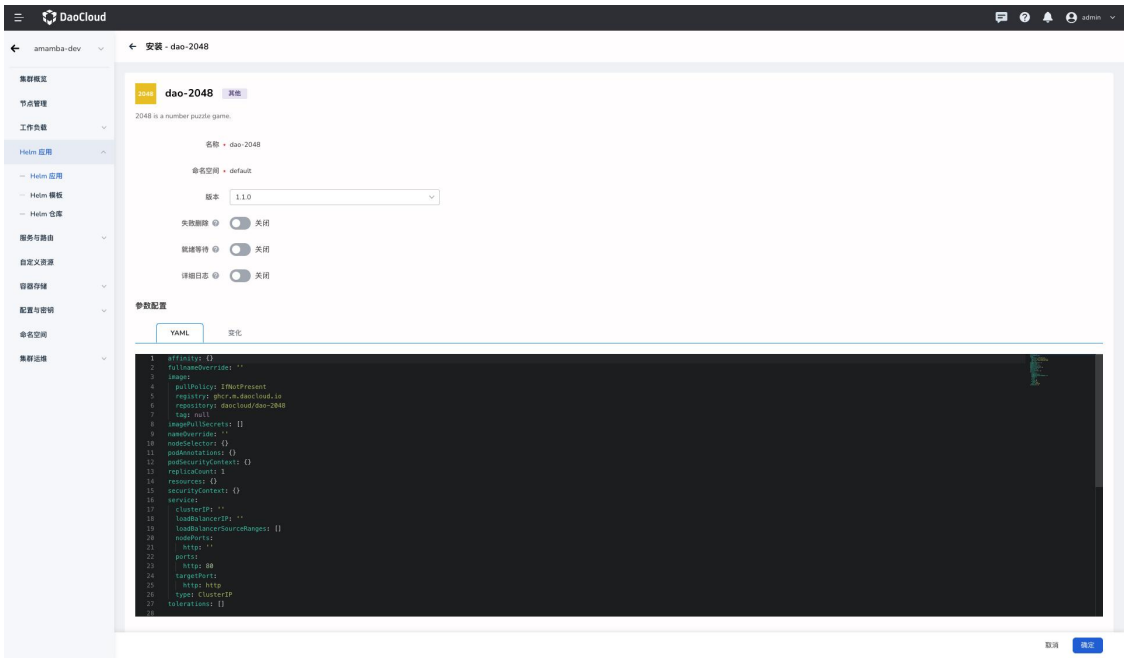
2. 在左侧导航栏，点击 Helm 应用，进入 Helm 应用列表页面。

   在 Helm 应用列表页选择需要更新的 Helm 应用，点击列表右侧的 ... 操作按钮，在下拉选择中选择更新操作。



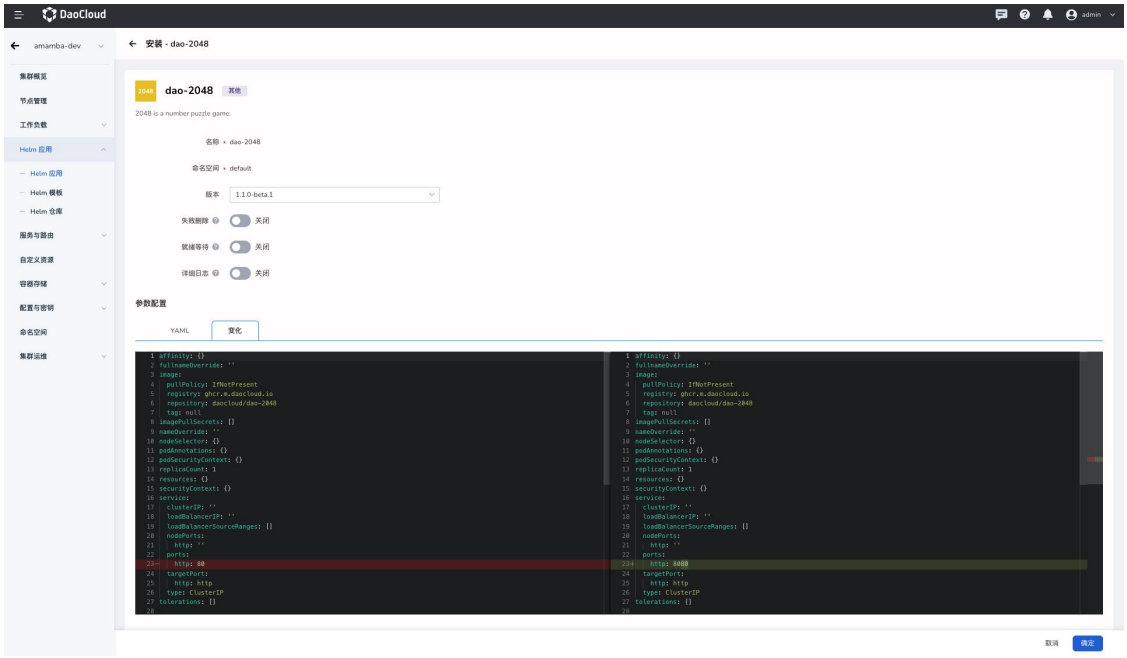
点击更新

3. 点击更新按钮后，系统将跳转至更新界面，您可以根据需要对 Helm 应用进行更新，此处我们以更新 dao-2048 这个应用的 http 端口为例。



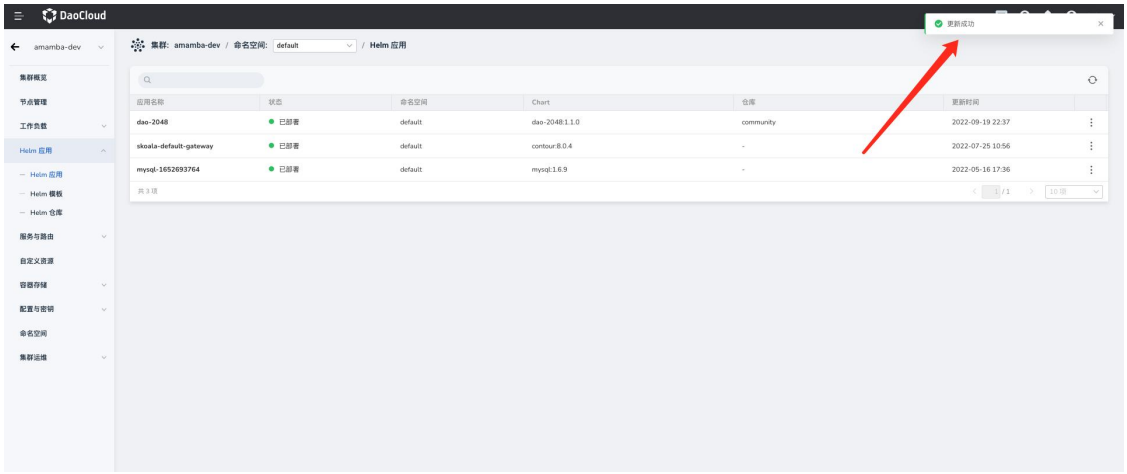
更新页面

4. 修改完相应参数后。您可以在参数配置下点击变化按钮，对比修改前后的文件，确定无误后，点击底部确定按钮，完成 Helm 应用的更新。



对比变化

5. 系统将自动返回 Helm 应用列表，右上角弹窗提示更新成功。

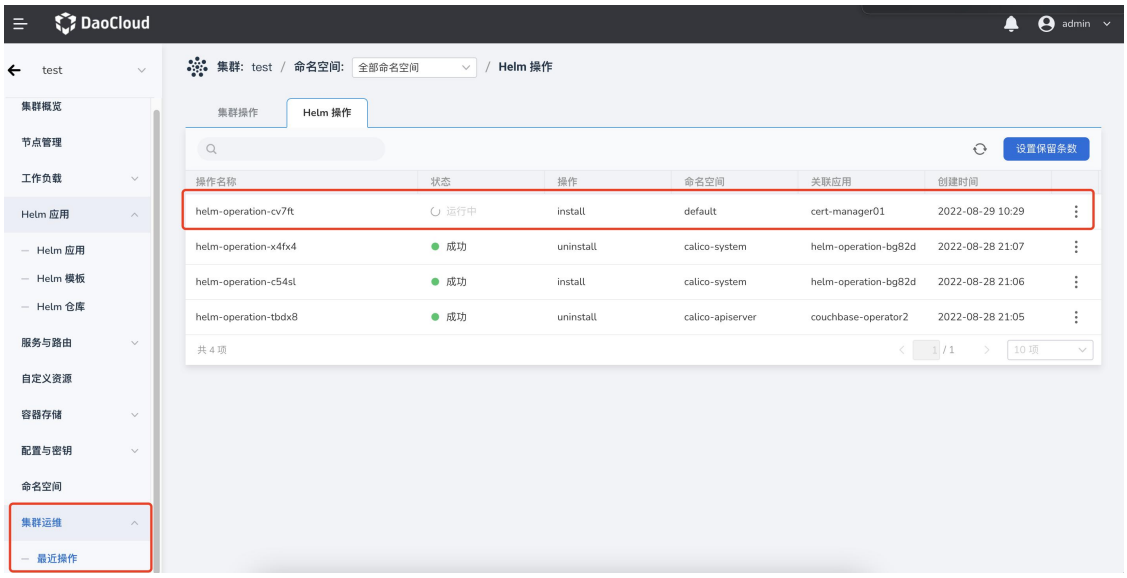


更新成功

查看 Helm 操作记录

Helm 应用的每次安装、更新、删除都有详细的操作记录和日志可供查看。

1. 在左侧导航栏，依次点击集群运维 -> 最近操作，然后在页面上方选择 Helm 操作标签页。每一条记录对应一次安装/更新/删除操作。



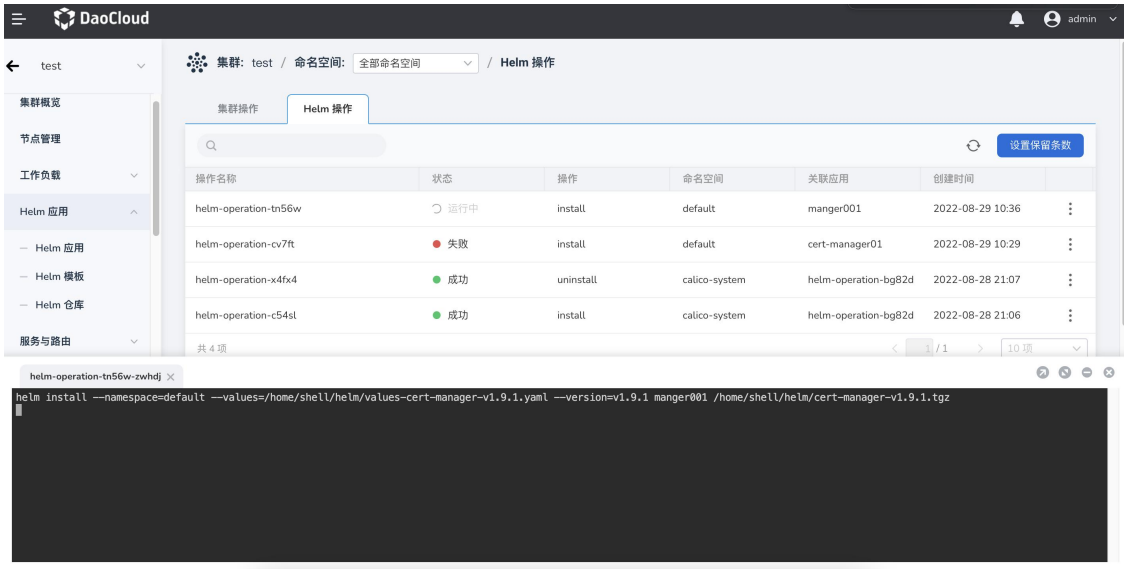
helm 操作

2. 如需查看每一次操作的详细日志：在列表右侧点击 ⋮，在弹出菜单中选择日志。



选择日志

3. 此时页面下方将以控制台的形式展示详细的运行日志。

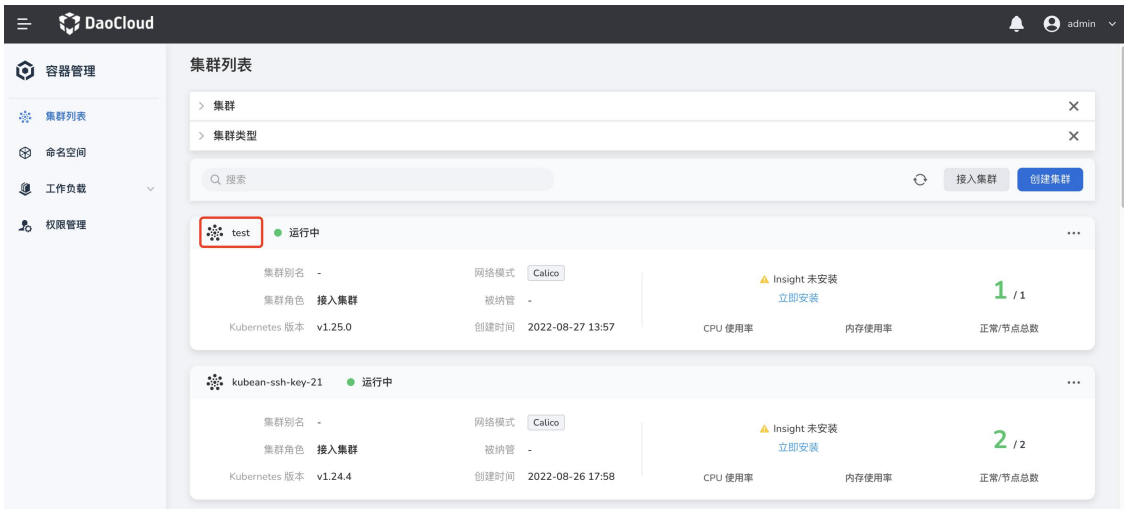


查看运行日志

删除 Helm 应用

参照以下步骤删除 Helm 应用。

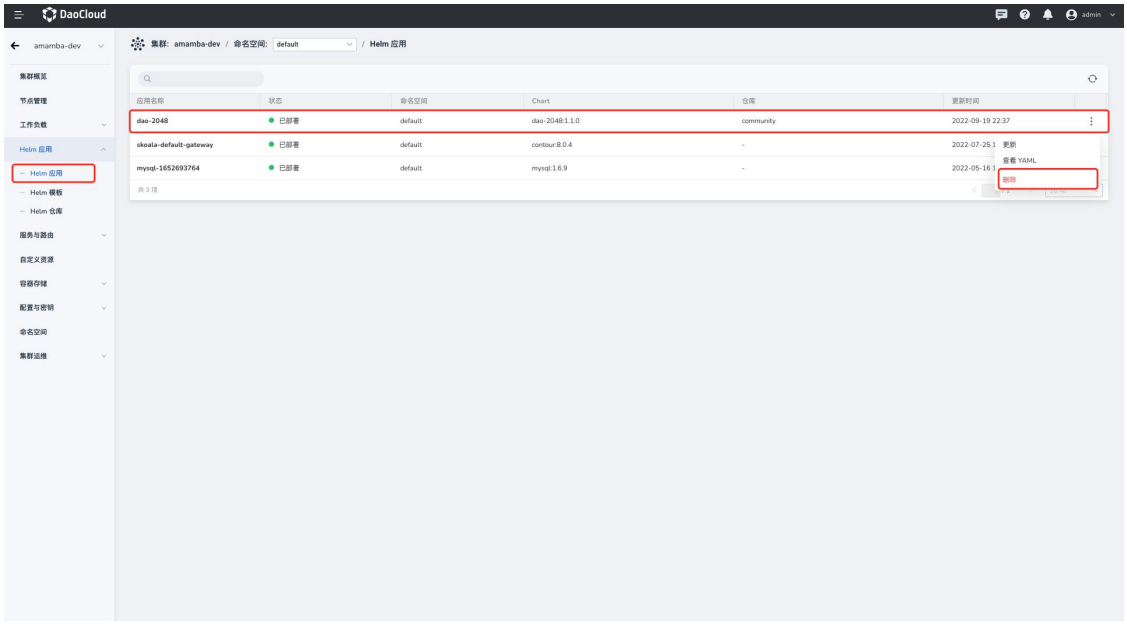
1. 找到待删除的 Helm 应用所在的集群，点击集群名称，进入集群详情。



集群详情

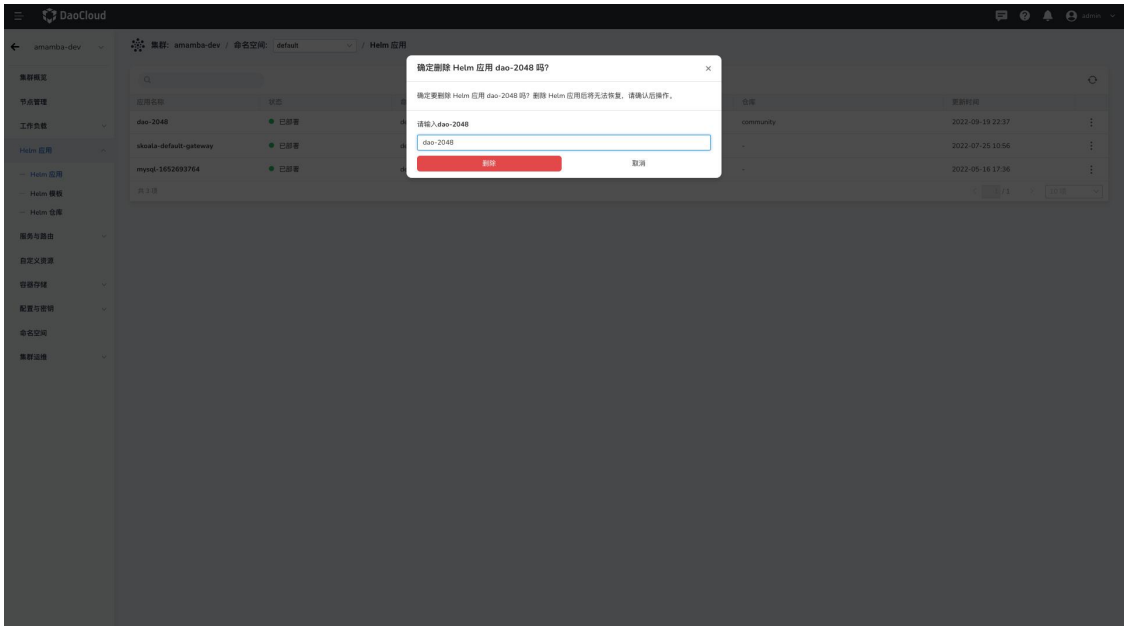
2. 在左侧导航栏，点击 Helm 应用，进入 Helm 应用列表页面。

在 Helm 应用列表页选择您需要删除的 Helm 应用，点击列表右侧的 ... 操作按钮，在下拉选择中选择删除。



点击删除

3. 在弹窗内输入 Helm 应用的名称进行确认，然后点击删除按钮。



确认删除

管理 Helm 仓库

Helm 仓库是用来存储和发布 Chart 的存储库。Helm 应用模块支持通过 HTTP(s) 协议来访问存储库中的 Chart 包。系统默认内置了下表所示的 4 个 Helm 仓库以满足企业生产过程中的常见需求。

仓库	描述	示例
partner	由生态合作伙伴所提供的各类优质特色 Chart	tidb
system	系统核心功能组件及部分高级功能所必需依赖的 Chart，如必需安装 insight-agent 才能够获取集群的监控信息	Insight
addon	业务场景中常见的 Chart	cert-manager
community	Kubernetes 社区较为热门的开源组件 Chart	Istio

除上述预置仓库外，您也可以自行添加第三方 Helm 仓库。本文将介绍如何添加、更新第三方 Helm 仓库。

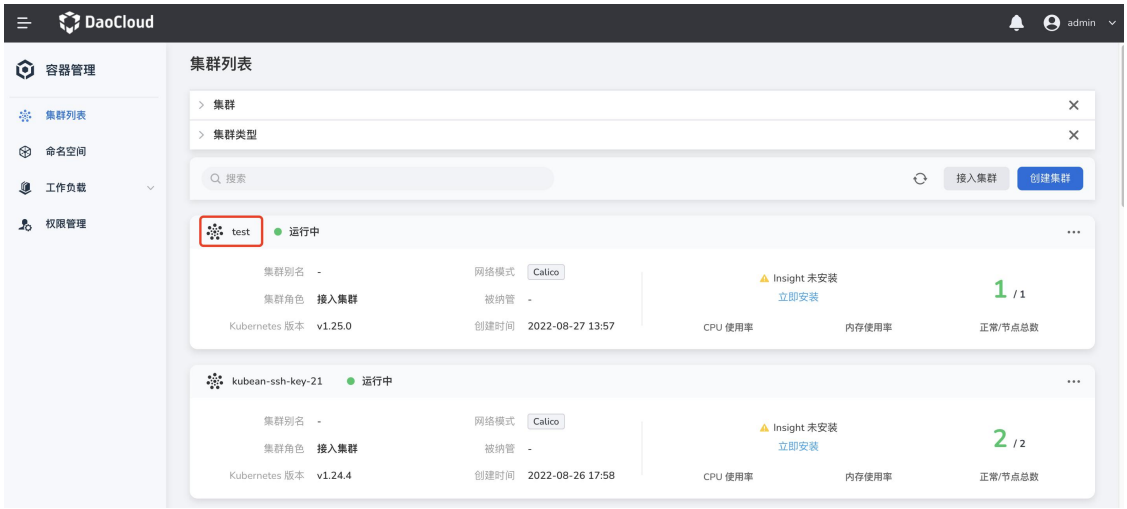
前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面
- 已完成一个命名空间的创建、用户的创建，并为用户授予 NS Admin 或更高权限 ，详情可参考命名空间授权。
- 如果使用私有仓库，当前操作用户应拥有对该私有仓库的读写权限。

引入第三方 Helm 仓库

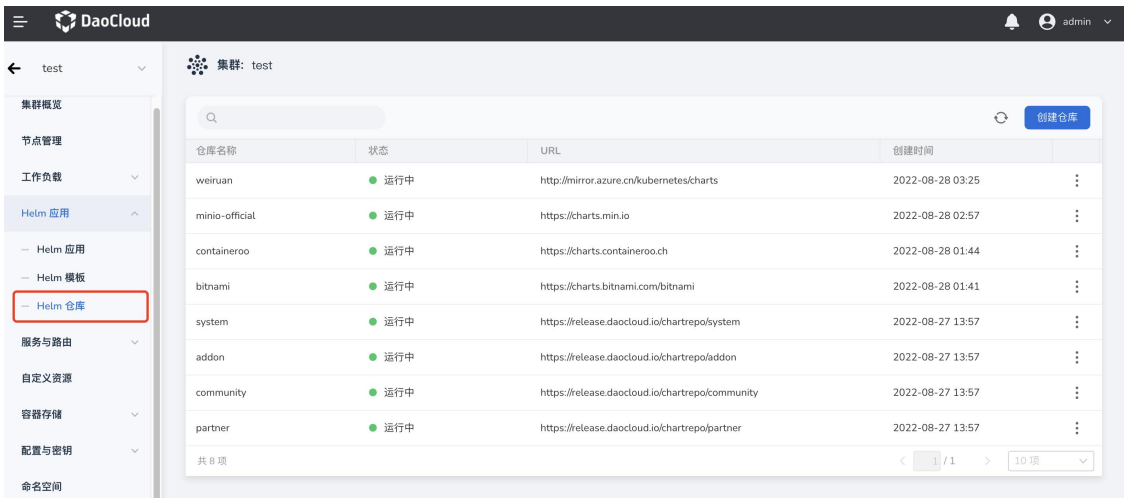
下面以 Kubevela 公开的镜像仓库为例，引入 Helm 仓库并管理。

1. 找到需要引入第三方 Helm 仓库的集群，点击集群名称，进入集群详情。



集群详情

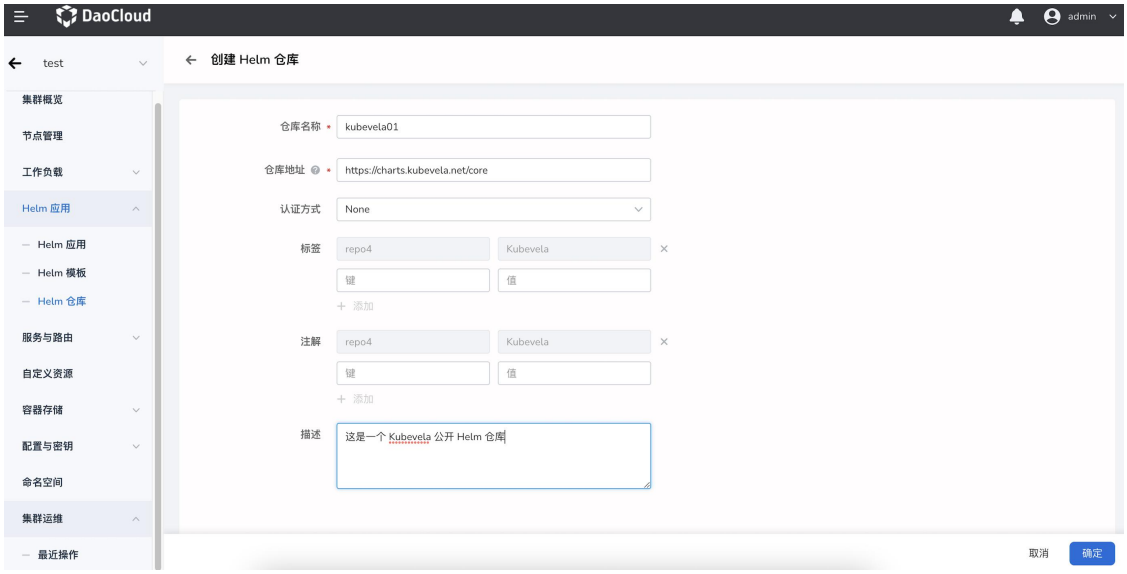
2. 在左侧导航栏，依次点击 Helm 应用 –> Helm 仓库，进入 Helm 仓库页面。



helm 仓库

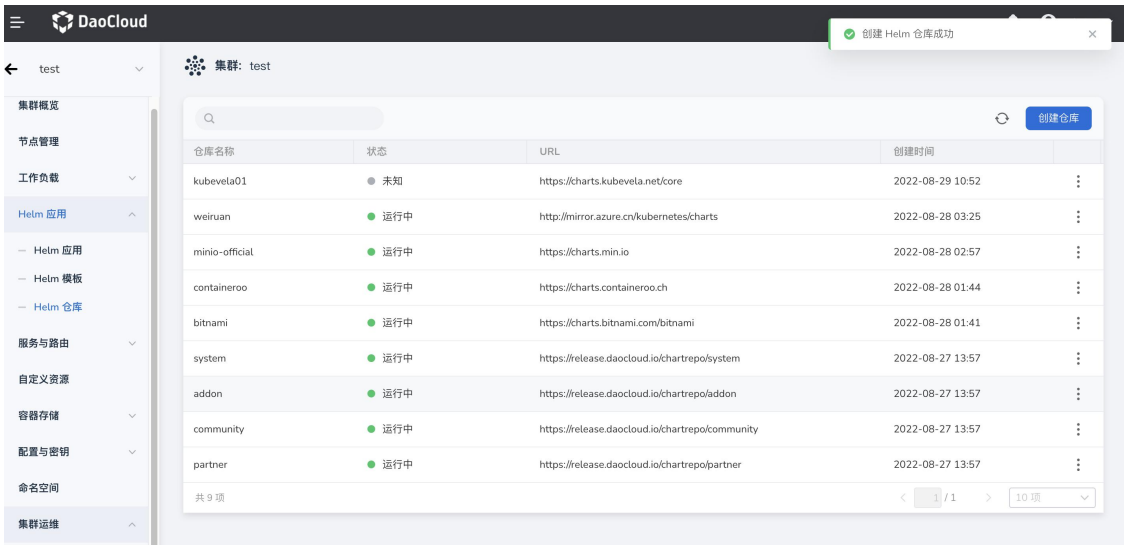
3. 在 Helm 仓库页面点击创建仓库按钮，进入创建仓库页面，按照下表配置相关参数。

- 仓库名称：设置仓库名称。最长 63 个字符，只能包含小写字母、数字及分隔符 -，且必须以小写字母或数字开头并结尾，例如 kubevela
- 仓库地址：用来指向目标 Helm 仓库的 http（s）地址。例如 https://charts.kubevela.net/core
- 认证方式：连接仓库地址后用来进行身份校验的方式。对于公开仓库，可以选择 None，私有的仓库需要输入用户名/密码以进行身份校验
- 标签：为该 Helm 仓库添加标签。例如 key: repo4；value: Kubevela
- 注解：为该 Helm 仓库添加注解。例如 key: repo4；value: Kubevela
- 描述：为该 Helm 仓库添加描述。例如：这是一个 Kubevela 公开 Helm 仓库



填写参数

4. 点击确定，完成 Helm 仓库的创建。页面会自动跳转至 Helm 仓库列表。

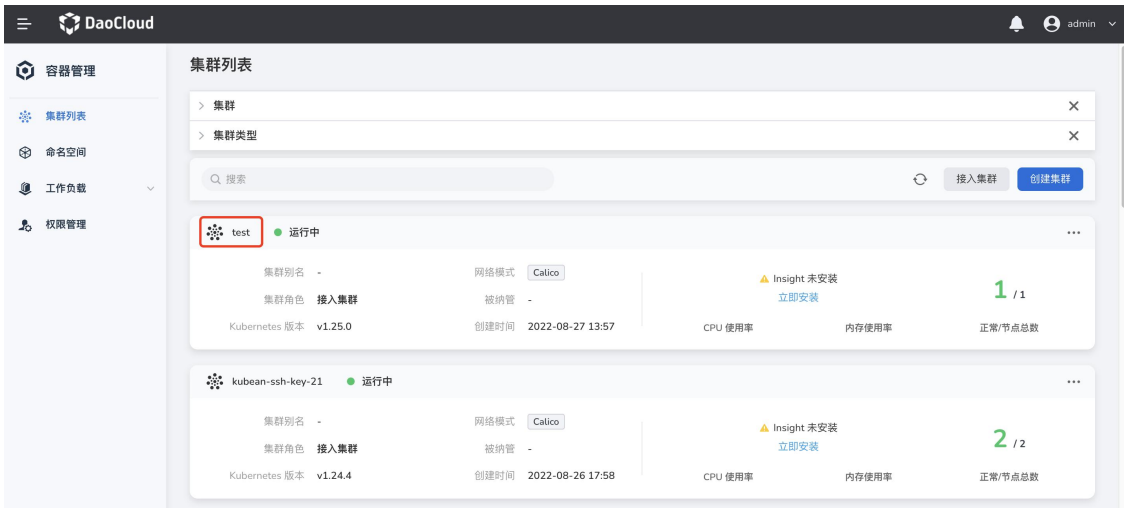


确定

更新 Helm 仓库

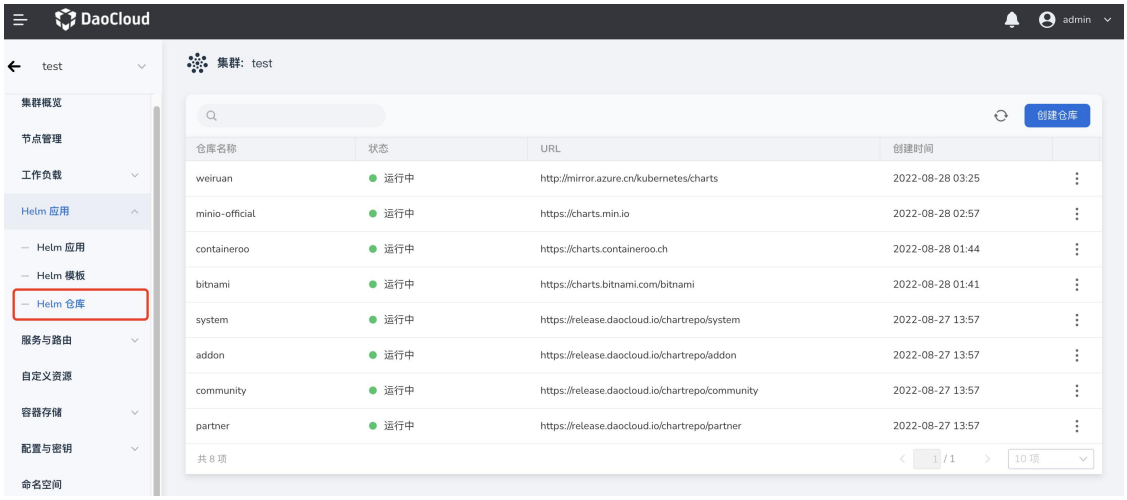
当 Helm 仓库的地址信息发生变化时，可以更新 Helm 仓库的地址、认证方式、标签、注解及描述信息。

1. 找到待更新仓库所在的集群，点击集群名称，进入集群详情。



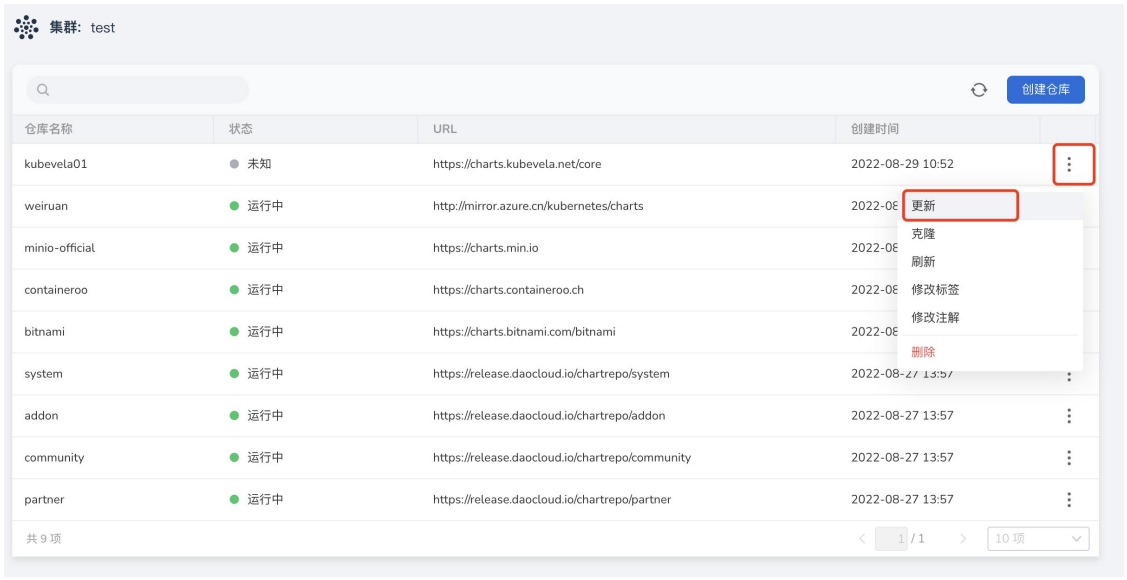
集群详情

2. 在左侧导航栏，依次点击 Helm 应用 –> Helm 仓库，进入 Helm 仓库列表页面。



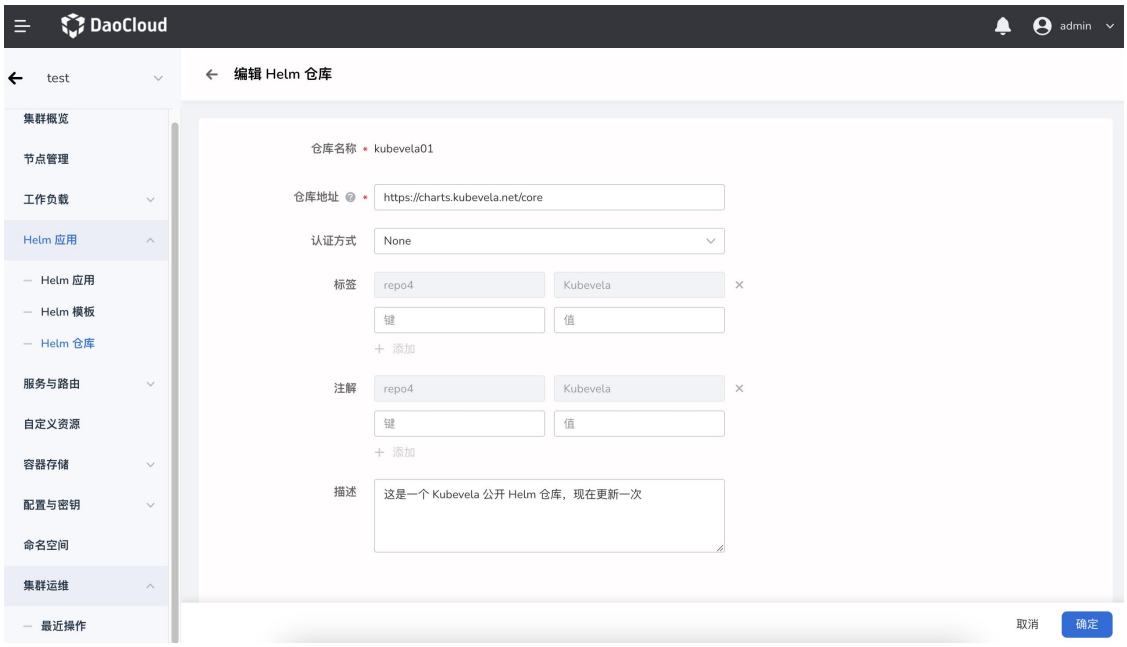
helm 仓库

3. 在仓库列表页面找到需要更新的 Helm 仓库，在列表右侧点击 ⋮ 按钮，在弹出菜单中点击更新。



点击更新

4. 在编辑 Helm 仓库页面进行更新，完成后点击确定。



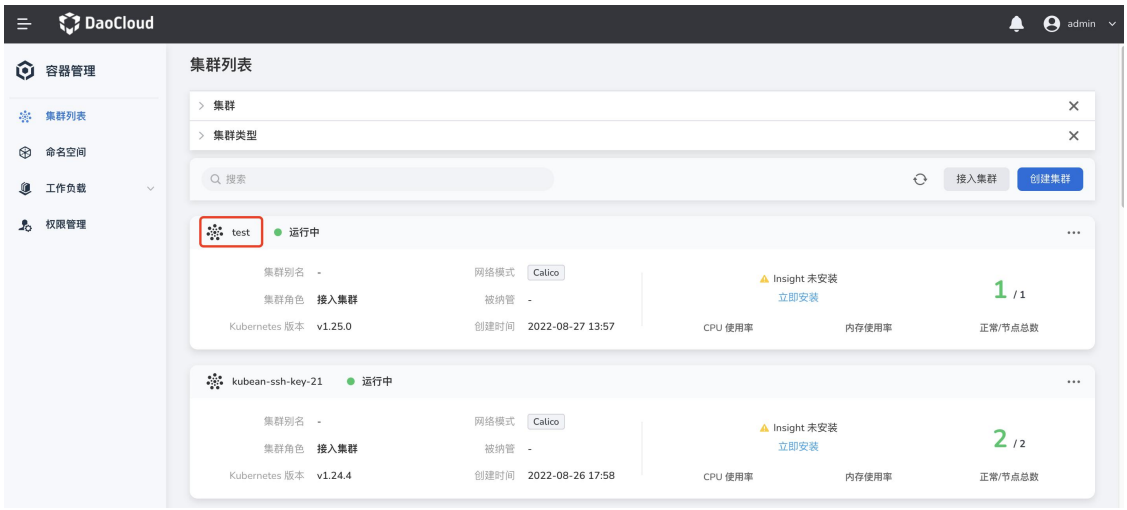
确定

5. 返回 Helm 仓库列表，屏幕提示更新成功。

删除 Helm 仓库

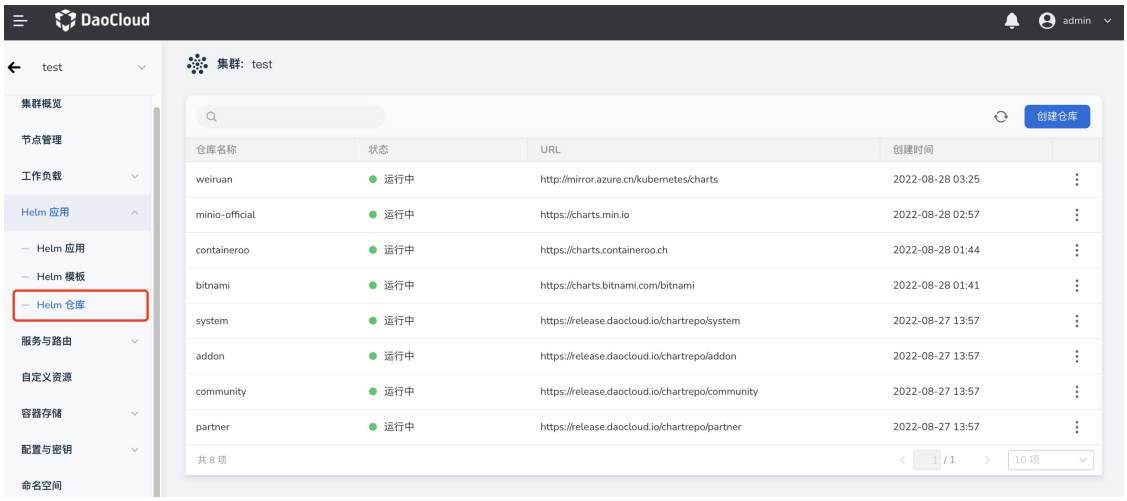
除了引入、更新仓库外，您也可以将不需要的仓库删除，包括系统预置仓库和第三方仓库。

1. 找到待删除仓库所在的集群，点击集群名称，进入集群详情。



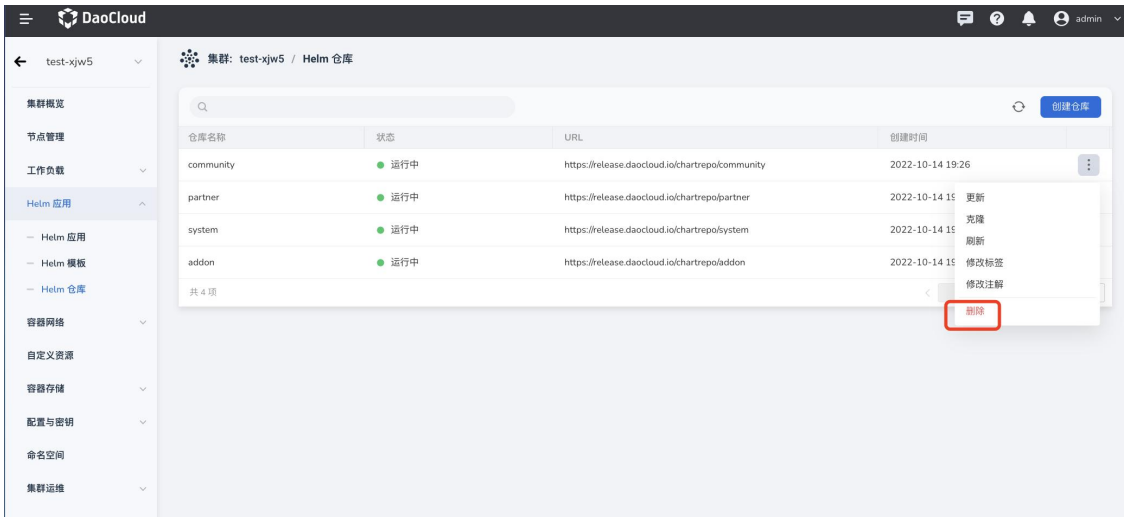
集群详情

2. 在左侧导航栏，依次点击 Helm 应用 –> Helm 仓库，进入 Helm 仓库列表页面。



helm 仓库

3. 在仓库列表页面找到需要更新的 Helm 仓库，在列表右侧点击 ⋮ 按钮，在弹出菜单中点击删除。



点击删除

4. 输入仓库名称进行确认，点击删除。



确认删除

5. 返回 Helm 仓库列表，屏幕提示删除成功。

创建服务（Service）

在 Kubernetes 集群中，每个 Pod 都有一个内部独立的 IP 地址，但是工作负载中的 Pod 可能会被随时创建和删除，直接使用 Pod IP 地址并不能对外提供服务。

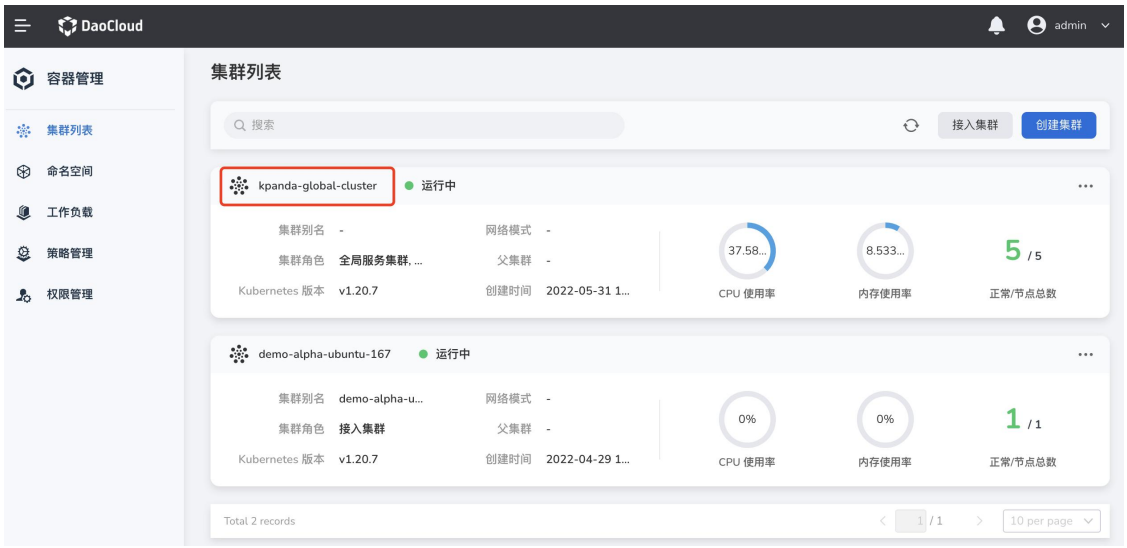
这就需要创建服务，通过服务您会获得一个固定的 IP 地址，从而实现工作负载前端和后端的解耦，让外部用户能够访问服务。同时，服务还提供了负载均衡（LoadBalancer）功能，使用户能从公网访问到工作负载。

前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes，且能够访问集群的 UI 界面。
- 已完成一个命名空间的创建、用户的创建，并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

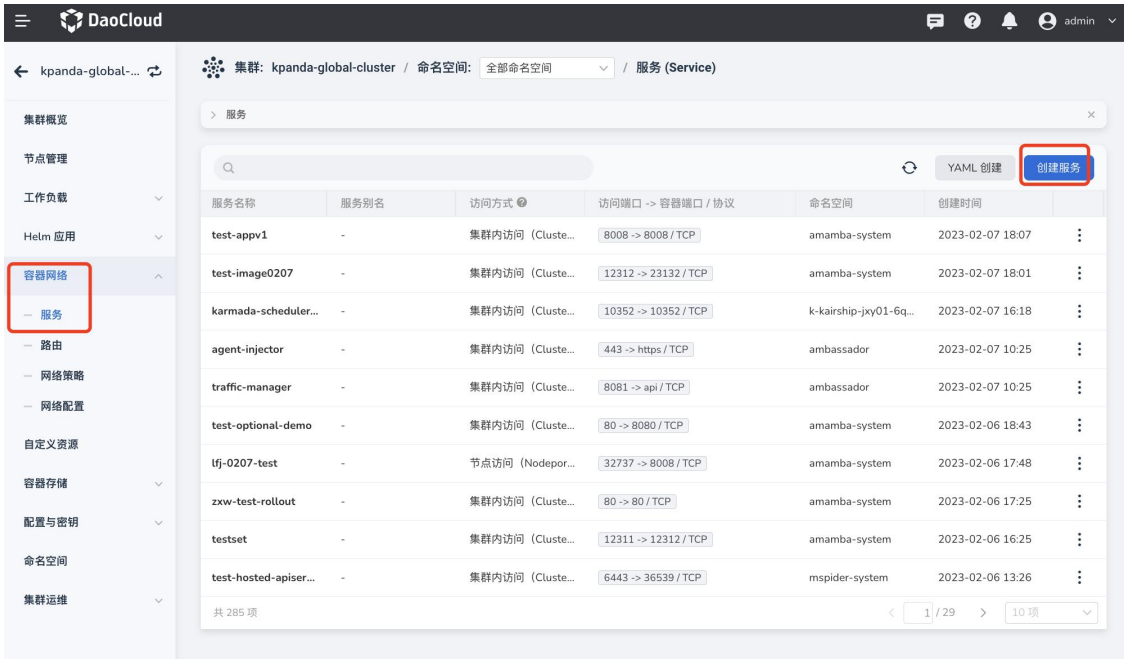
创建服务

1. 以 NS Edit 用户成功登录后，点击左上角的集群列表进入集群列表页面。在集群列表中，点击一个集群名称。



集群列表

2. 在左侧导航栏中，点击容器网络–>服务进入服务列表，点击右上角创建服务按钮。

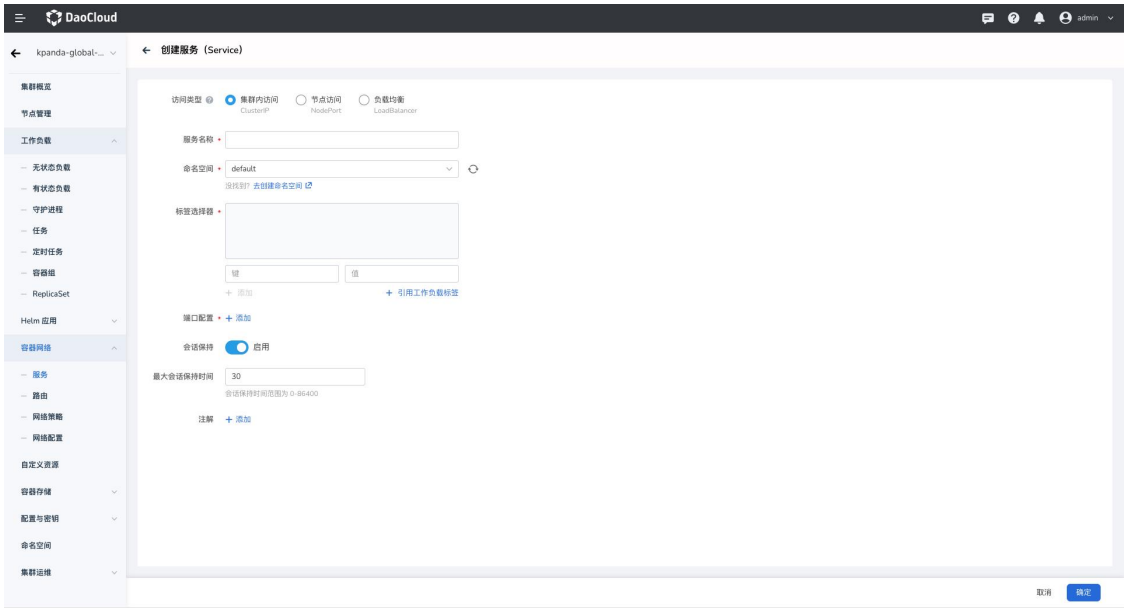


服务与路由

!!! tip

也可以通过 `YAML 创建`一个服务。

3. 打开创建服务页面，选择一种访问类型，参考以下三个参数表进行配置。



创建服务

创建 ClusterIP 服务

点选集群内访问（ClusterIP），这是指通过集群的内部 IP 暴露服务，选择此项的服务只能在集群内部访问。这是默认的服务类型。参考下表配置参数。

参数	说明	举例值
访问类型	【类型】必填【含义】指定 Pod 服务发现的方式，这里选择集群内访问（ClusterIP）。	ClusterIP
服务名称	【类型】必填【含义】输入新建服务的名称。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	Svc-01
命名空间	【类型】必填【含义】选择新建服务所在的命名空间。关于命名空间更多信息请参考命名空间概述。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文	default

参数	说明	举例值
	字母或数字结尾。	
标签选择器	【类型】必填【含义】添加标签，Service 根据标签选择 Pod，填写后点击“添加”。也可以引用已有工作负载的标签，点击引用负载标签，在弹出的窗口中选择负载，系统会默认将所选的负载标签作为选择器。	app:job01
端口配置	【类型】必填【含义】为服务添加协议端口，需要先选择端口协议类型，目前支持 TCP、UDP 两种传输协议额关于协议更多信息请参考协议概述。**端口名称**：输入自定义的端口的名称。**服务端口（port）**：Pod 对外提供服务的访问端口。**容器端口（targetport）**：工作负载实际监听的容器端口，用来对集群内暴露服务。	
会话保持	【类型】选填【含义】开启后，相同客户端的请求将转发至同一 Pod	开启
会话保持最大时长	【类型】选填【含义】开启会话保持后，保持的最大时长，默认为 30 秒	30 秒
注解	【类型】选填【含义】为服务添加注解	

创建 NodePort 服务

点选节点访问（NodePort），这是指通过每个节点上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到自动创建的 ClusterIP 服务。通过请求 <节点 IP>:<节点端口>，您可以从集群的外部访问一个 NodePort 服务。参考下表配置参数。

参数	说明	举例值
访问类型	【类型】必填【含义】指定 Pod 服务发现的方式，这里选择节点访问（NodePort）。	NodePort
服务名称	【类型】必填【含义】输入新建服务的名称。【注意】请	Svc-

参数	说明	举例值
	输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	01
命名空间	【类型】必填【含义】选择新建服务所在的命名空间。关于命名空间更多信息请参考命名空间概述。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	default
标签选择器	【类型】必填【含义】添加标签，Service 根据标签选择 Pod，填写后点击“添加”。也可以引用已有工作负载的标签，点击引用负载标签，在弹出的窗口中选择负载，系统会默认将所选的负载标签作为选择器。	
端口配置	【类型】必填【含义】为服务添加协议端口，需要先选择端口协议类型，目前支持 TCP、UDP 两种传输协议额关于协议更多信息请参考协议概述。**端口名称**：输入自定义的端口的名称。**服务端口（port）**：Pod 对外提供服务的访问端口。*默认情况下，为了方便起见，服务端口被设置为与容器端口字段相同的值。***容器端口（targetport）**：工作负载实际监听的容器端口。**节点端口（nodeport）**：节点的端口，接收来自于 ClusterIP 传输的流量。用来做外部流量访问的入口。	
会话保持	【类型】选填【含义】开启后，相同客户端的请求将转发至同一 Pod 开启后 Service 的 .spec.sessionAffinity 为 ClientIP，详情请参考：Service 的会话亲和性	开启
会话保持最大时长	【类型】选填【含义】开启会话保持后，保持的最大时长，默认超时时长为 30	30 秒

参数	说明	举例值
	秒.spec.sessionAffinityConfig.clientIP.timeoutSeconds 默认设置为 30 秒	
注解	【类型】选填【含义】为服务添加注解	

创建 LoadBalancer 服务

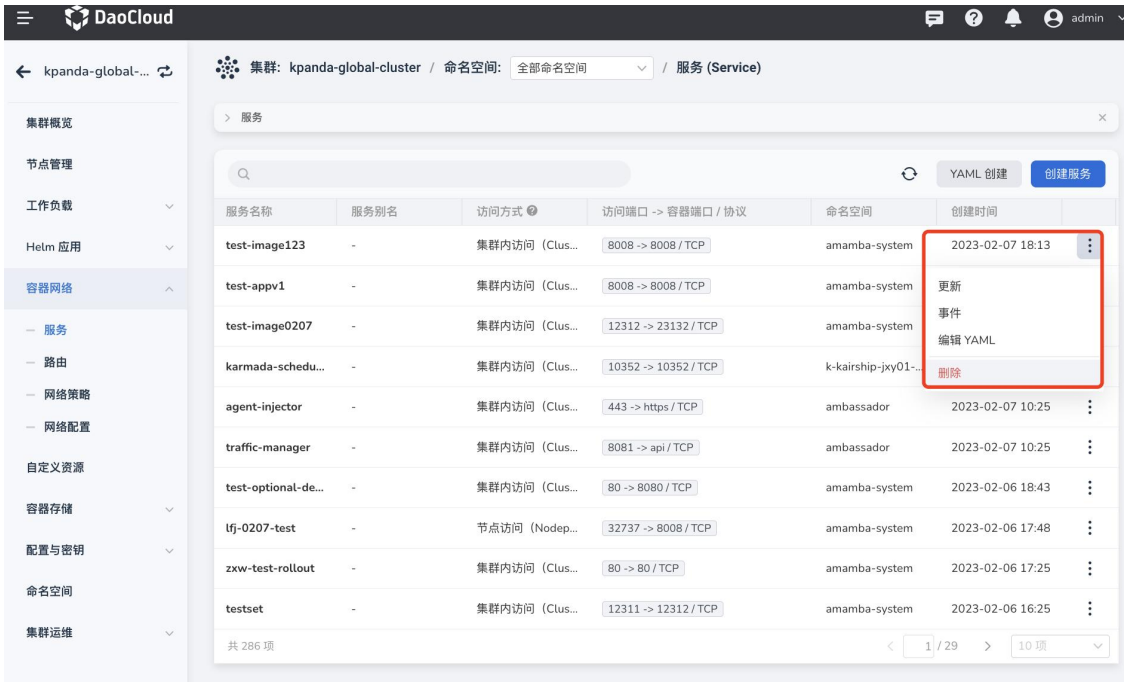
点选负载均衡（LoadBalancer），这是指使用云提供商的负载均衡器向外部暴露服务。 外部负载均衡器可以将流量路由到自动创建的 NodePort 服务和 ClusterIP 服务上。参考下表配置参数。

参数	说明	举例值
访问类型	【类型】必填【含义】指定 Pod 服务发现的方式，这里选择节点访问（NodePort）。	NodePort
服务名称	【类型】必填【含义】输入新建服务的名称。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	Svc-01
命名空间	【类型】必填【含义】选择新建服务所在的命名空间。关于命名空间更多信息请参考命名空间概述。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	default
外部流量策略	【类型】必填【含义】设置外部流量策略。**Cluster**：流量可以转发到集群中所有节点上的 Pod。**Local**：流量只发给本节点上的 Pod。【注意】请输入 4 到 63 个字符的字符串，可以包含小写英文字母、数字和中划线（-），并以小写英文字母开头，小写英文字母或数字结尾。	

参数	说明	举例值
标签选择器	【类型】必填【含义】添加标签，Service 根据标签选择 Pod，填写后点击“添加”。也可以引用已有工作负载的标签，点击引用负载标签，在弹出的窗口中选择负载，系统会默认将所选的负载标签作为选择器。	
负载均衡类型	【类型】必填【含义】使用的负载均衡类型，当前支持 MetalLB 和其他。	
MetalLB IP 池	【类型】必填【含义】选择的 负载均衡类型为 MetalLB 时，LoadBalancer Service 默认会从这个池中分配 IP 地址，并且通过 APR 宣告这个池中的所有 IP 地址，详情请参考：安装 MetalLB	
负载均衡地址	【类型】必填【含义】1.如使用的是公有云 CloudProvider，此处填写的为云厂商提供的负载均衡地址；2.如果上述负载均衡类型选择为 MetalLB ，默认从上述 IP 池中获取 IP ，如果不填则自动获取。	
端口配置	【类型】必填【含义】为服务添加协议端口，需要先选择端口协议类型，目前支持 TCP、UDP 两种传输协议额关于协议更多信息请参考协议概述。**端口名称**：输入自定义的端口的名称。**服务端口（port）**：Pod 对外提供服务的访问端口。默认情况下，为了方便起见，服务端口被设置为与容器端口字段相同的值。**容器端口（targetport）**：工作负载实际监听的容器端口。**节点端口（nodeport）**：节点的端口，接收来自于 ClusterIP 传输的流量。用来做外部流量访问的入口。	
注解	【类型】选填【含义】为服务添加注解	

完成服务创建

配置完所有参数后，点击确定按钮，自动返回服务列表。在列表右侧，点击 ⋮，可以修改或删除所选服务。



服务列表

使用配置项

配置项（ConfigMap）是 Kubernetes 的一种 API 对象，用来将非机密性的数据保存到键值对中，可以存储其他对象所需要使用的配置。 使用时， 容器可以将其用作环境变量、命令行参数或者存储卷中的配置文件。通过使用配置项，能够将配置数据和应用程序代码分开，为应用配置的修改提供更加灵活的途径。

!!! note

配置项并不提供保密或者加密功能。如果要存储的数据是机密的，请使用[密钥](use-secret.md)，或者使用其他第三方工具来保证数据的私密性，而不是用配置项。
此外在容器里使用配置项时，容器和配置项必须处于同一集群的命名空间中。

使用场景

您可以在 Pod 中使用配置项，有多种使用场景，主要包括：

- 使用配置项设置容器的环境变量
- 使用配置项设置容器的命令行参数
- 使用配置项作为容器的数据卷

设置容器的环境变量

您可以通过图形化界面或者终端命令行来使用配置项作为容器的环境变量。

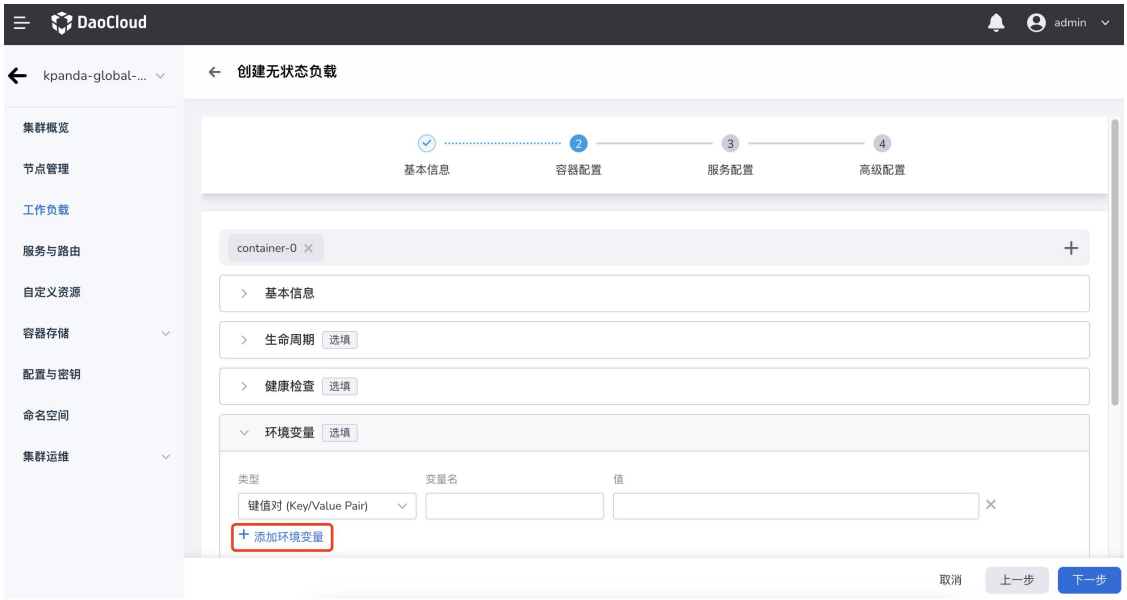
!!! note

配置项导入是将配置项作为环境变量的值；配置项键值导入是将配置项中某一参数作为环境变量的值。

图形化界面操作

通过镜像创建工作负载时，可以在环境变量界面通过选择配置项导入或配置项键值导入为容器设置环境变量。

1. 进入镜像创建工作负载页面中，在容器配置这一步中，选择环境变量配置，点击添加环境变量按钮。



添加环境变量

2. 在环境变量类型处选择配置项导入或配置项键值导入。

    - 当环境变量类型选择为配置项导入时，依次输入变量名、前缀名称、配置项的名称。
    - 当环境变量类型选择为配置项键值导入时，依次输入变量名、配置项名称、键的名称。

命令行操作

您可以在创建工作负载时将配置项设置为环境变量，使用 valueFrom 参数引用 ConfigMap 中的 Key/Value。

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod-1
spec:
```

```
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:                    # (1)
            configMapKeyRef:
              name: kpanda-configmap    # (2)
              key: SPECIAL_LEVEL        # (3)
  restartPolicy: Never
```

1. 使用 valueFrom 来指定 env 引用配置项的 value 值
2. 引用的配置文件名称
3. 引用的配置项 key

设置容器的命令行参数

您可以使用配置项设置容器中的命令或者参数值，使用环境变量替换语法 $(VAR_NAME) 来进行。如下所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod-3
spec:
  containers:
    - name: test-container
      image: busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TY
PE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
```

```
        valueFrom:
          configMapKeyRef:
            name: kpanda-configmap
            key: SPECIAL_LEVEL
      - name: SPECIAL_TYPE_KEY
        valueFrom:
          configMapKeyRef:
            name: kpanda-configmap
            key: SPECIAL_TYPE
  restartPolicy: Never
```

这个 Pod 运行后，输出如下内容。

```
Hello Kpanda
```

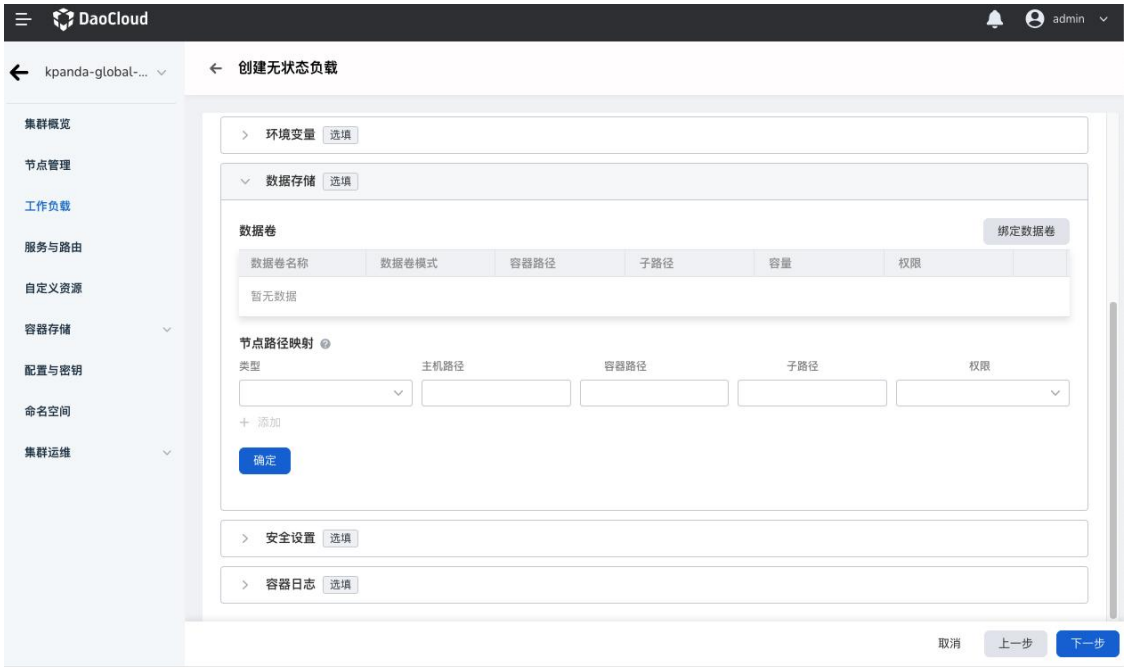
用作容器数据卷

您可以通过图形化界面或者终端命令行来使用配置项作为容器的环境变量。

图形化操作

在通过镜像创建工作负载时，您可以通过在数据存储界面选择存储类型为配置项，将配置项作为容器的数据卷。

1. 进入镜像创建工作负载页面中，在容器配置这一步中，选择数据存储配置，在节点路径映射列表点击添加按钮。



添加环境变量

2. 在存储类型处选择配置项，并依次输入容器路径、子路径等信息。

命令行操作

要在一个 Pod 的存储卷中使用 ConfigMap。

下面是一个将 ConfigMap 以卷的形式进行挂载的 Pod 示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
```

```
      readOnly: true
  volumes:
  - name: foo
    configMap:
      name: myconfigmap
```

如果 Pod 中有多个容器，则每个容器都需要自己的 volumeMounts 块，但针对每个 ConfigMap，您只需要设置一个 spec.volumes 块。

!!! note

将配置项作为容器挂载的数据卷时，配置项只能作为只读文件进行读取。

创建密钥

密钥是一种用于存储和管理密码、OAuth 令牌、SSH、TLS 凭据等敏感信息的资源对象。使用密钥意味着您不需要在应用程序代码中包含敏感的机密数据。

密钥使用场景：

- 作为容器的环境变量使用，提供容器运行过程中所需的一些必要信息。
- 使用密钥作为 Pod 的数据卷。
- 在 kubelet 拉取容器镜像时作为镜像仓库的身份认证凭证。

支持两种创建方式：

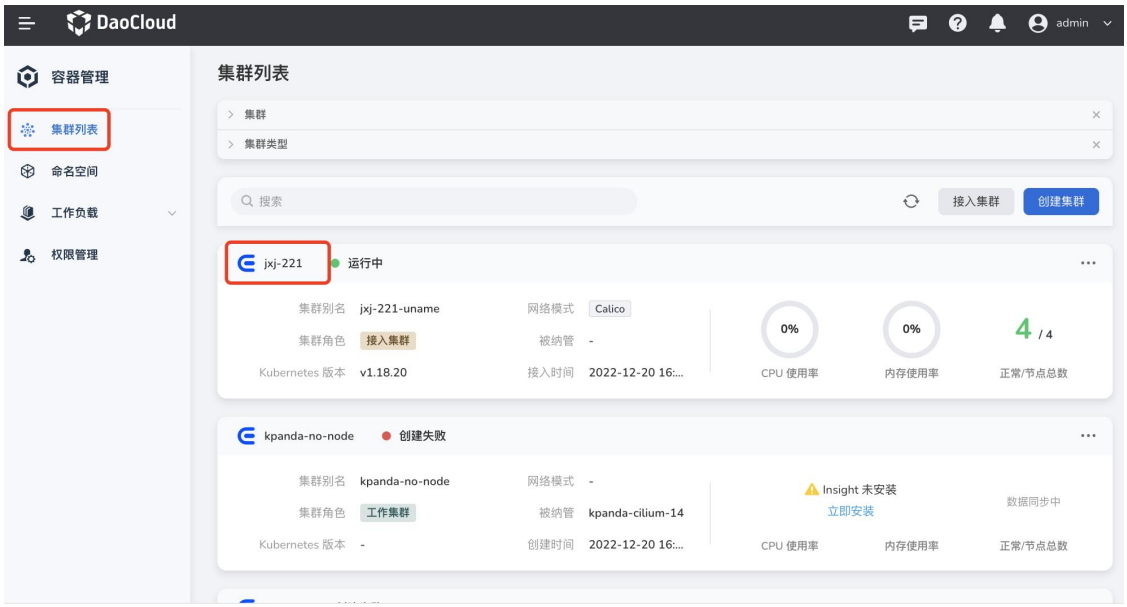
- 图形化表单创建
- YAML 创建

前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面
- 已完成一个命名空间的创建、用户的创建，并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。

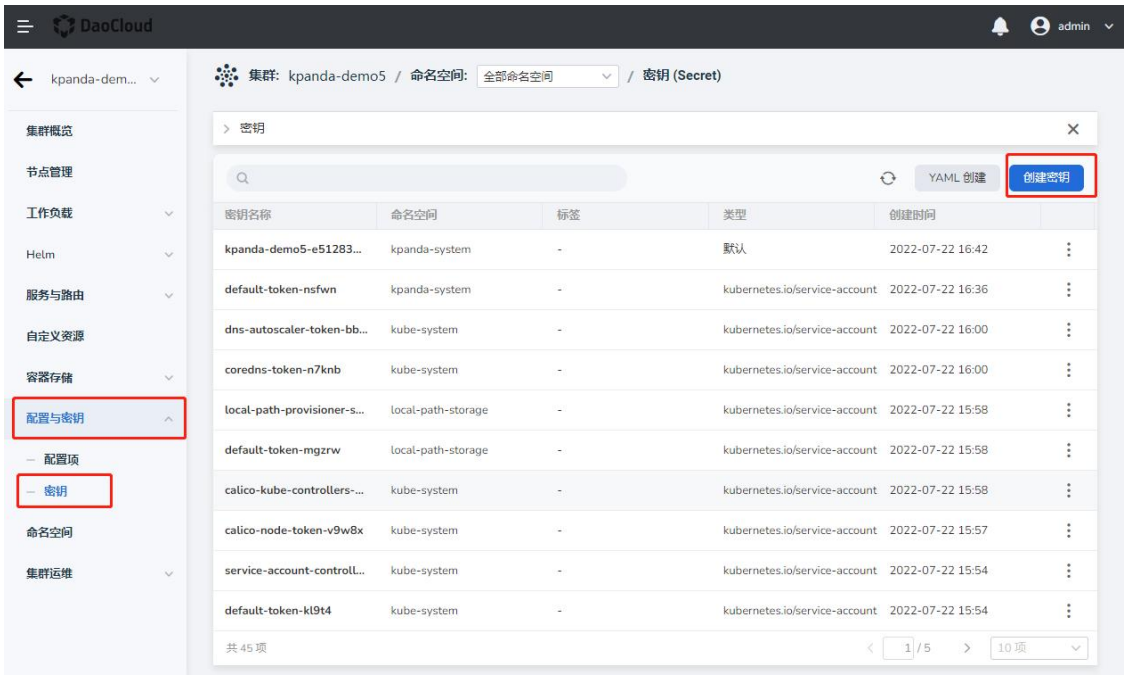
图形化表单创建

1. 在集群列表页面点击某个集群的名称，进入集群详情。



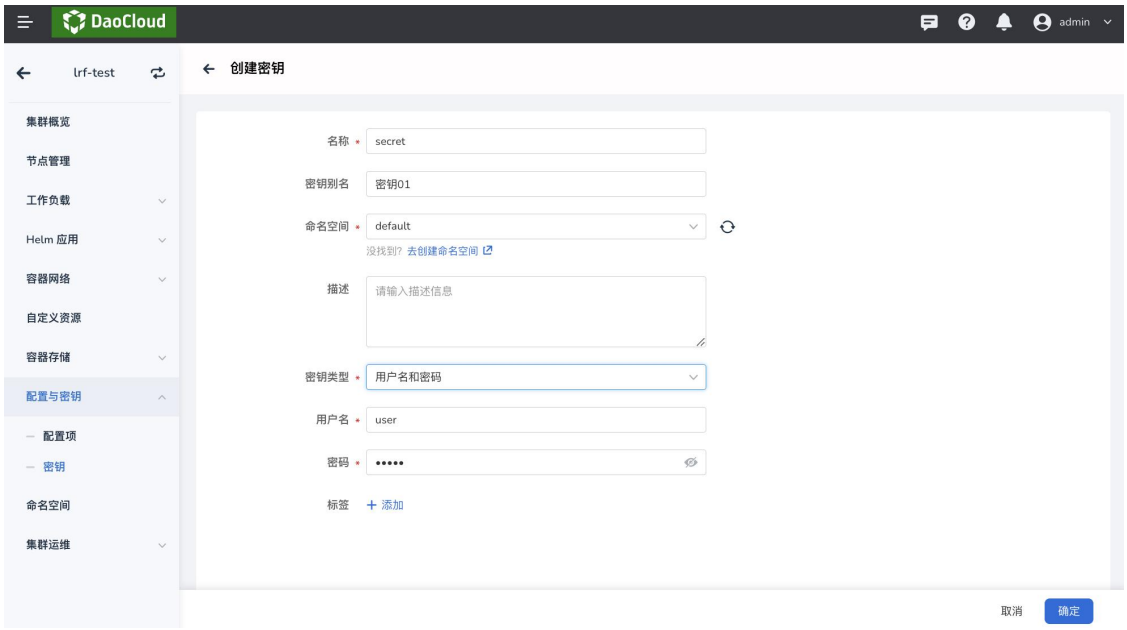
集群详情

2. 在左侧导航栏，点击配置与密钥–>密钥，点击右上角创建密钥按钮。



创建密钥

3. 在创建密钥页面中填写配置信息，点击确定。



创建密钥

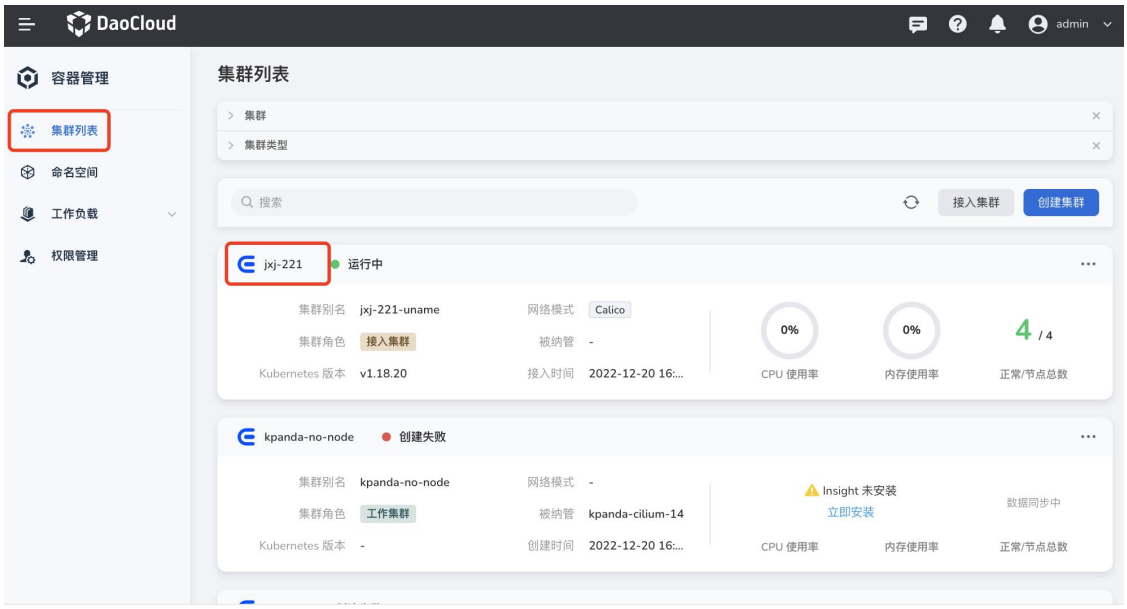
填写配置时需要注意：

- 密钥的名称在同一个命名空间中必须具有唯一性
- 密钥类型：
  - 默认（Opaque）：Kubernetes 默认的密钥类型，支持用户定义的任意数据。
  - TLS (kubernetes.io/tls)：用于 TLS 客户端或者服务器端数据访问的凭证。
  - 镜像仓库信息（kubernetes.io/dockerconfigjson)：用于镜像仓库访问的凭证。
  - 用户名和密码（kubernetes.io/basic-auth）：用于基本身份认证的凭证。
  - 自定义：用户根据业务需要自定义的类型。
- 密钥数据：密钥所存储的数据，不同数据需要填写的参数有所不同
  - 当密钥类型为默认（Opaque）/自定义：可以填入多个键值对数据。

- 当密钥类型为 TLS (kubernetes.io/tls)：需要填入证书凭证和私钥数据。证书是自签名或 CA 签名过的凭据，用来进行身份认证。证书请求是对签名的请求，需要使用私钥进行签名。
- 当密钥类型为镜像仓库信息 (kubernetes.io/dockerconfigjson)：需要填入私有镜像仓库的帐号和密码。
- 当密钥类型为用户名和密码（kubernetes.io/basic-auth）：需要指定用户名和密码。

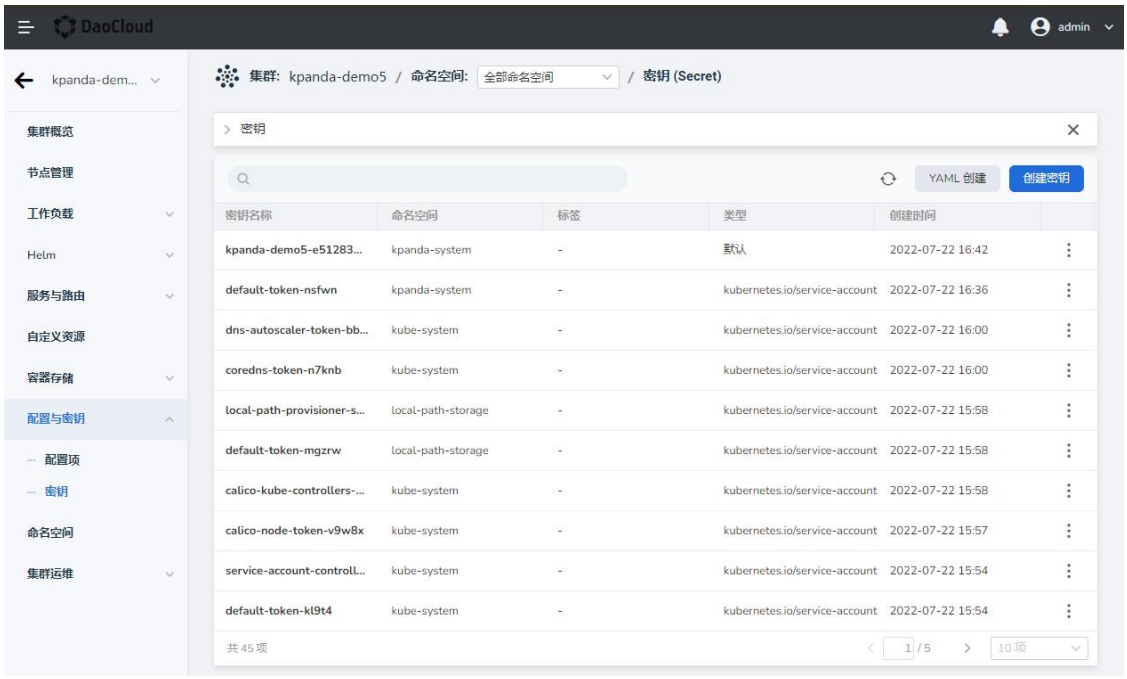
YAML 创建

1. 在集群列表页面点击某个集群的名称，进入集群详情。



集群详情

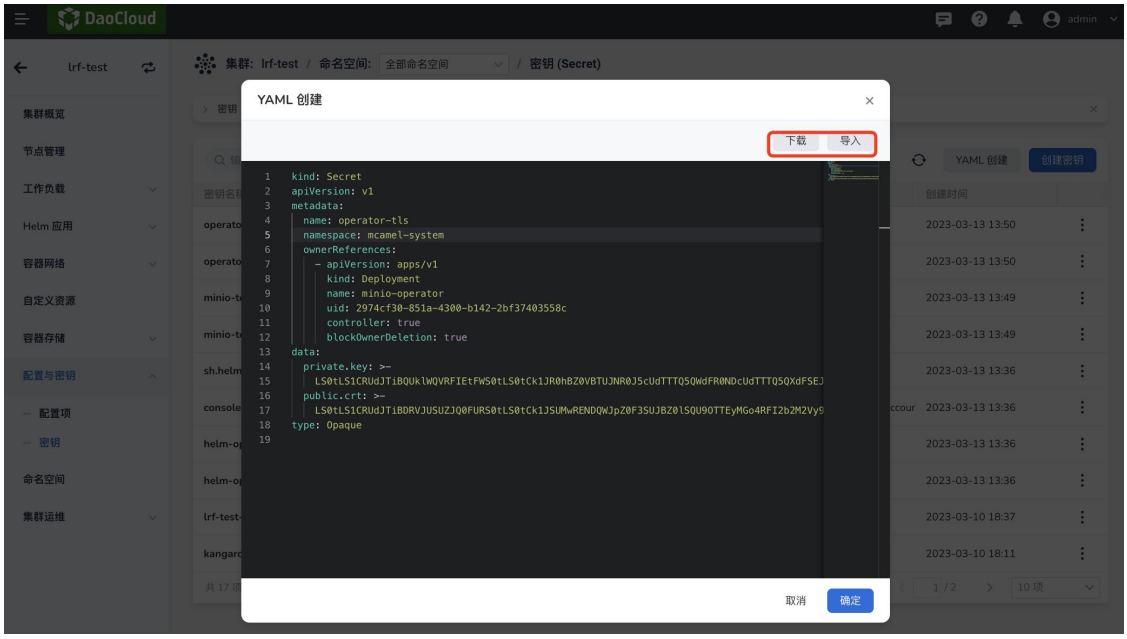
2. 在左侧导航栏，点击配置与密钥->密钥，点击右上角 YAML 创建按钮。



YAML 创建

3. 在 YAML 创建页面中填写 YAML 配置，点击确定。

支持从本地导入 YAML 文件或将填写好的文件下载保存到本地。



YAML 创建

密钥 YAML 示例

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secretdemo
type: Opaque
data:
  username: ******
  password: ******
```

下一步：使用密钥

创建配置项

配置项（ConfigMap）以键值对的形式存储非机密性数据，实现配置数据和应用代码相互解耦的效果。配置项可用作容器的环境变量、命令行参数或者存储卷中的配置文件。

!!! note

– 在配置项中保存的数据不可超过 1 MiB。如果需要存储体积更大的数据，建议挂载存储卷或者使用独立的数据库或者文件服务。

– 配置项不提供保密或者加密功能。如果要存储加密数据，建议使用[密钥](use-secret.md)，或者其他第三方工具来保证数据的私密性。

支持两种创建方式：

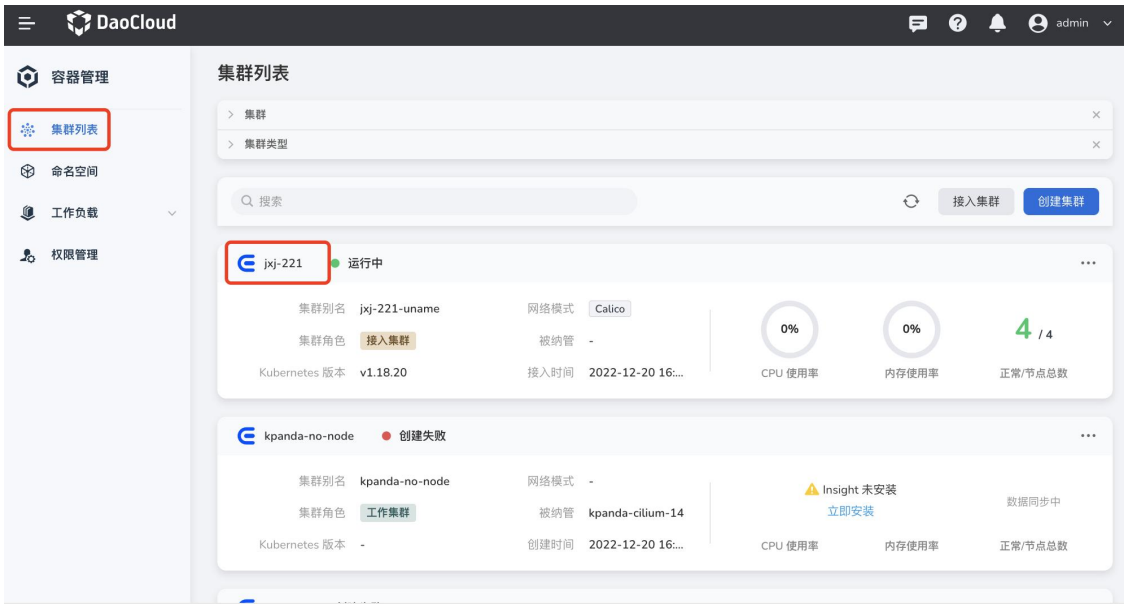
- 图形化表单创建
- YAML 创建

前提条件

- 容器管理模块已接入 Kubernetes 集群或者已创建 Kubernetes 集群，且能够访问集群的 UI 界面
- 已完成一个命名空间的创建、用户的创建，并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。

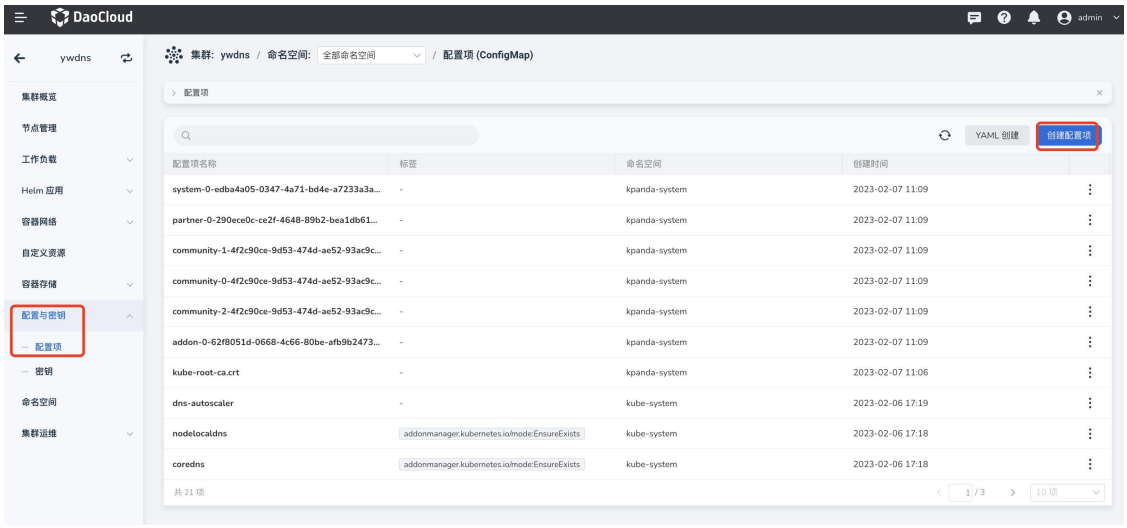
图形化表单创建

1. 在集群列表页面点击某个集群的名称，进入集群详情。



集群详情

2. 在左侧导航栏，点击配置与密钥–>配置项，点击右上角创建配置项按钮。

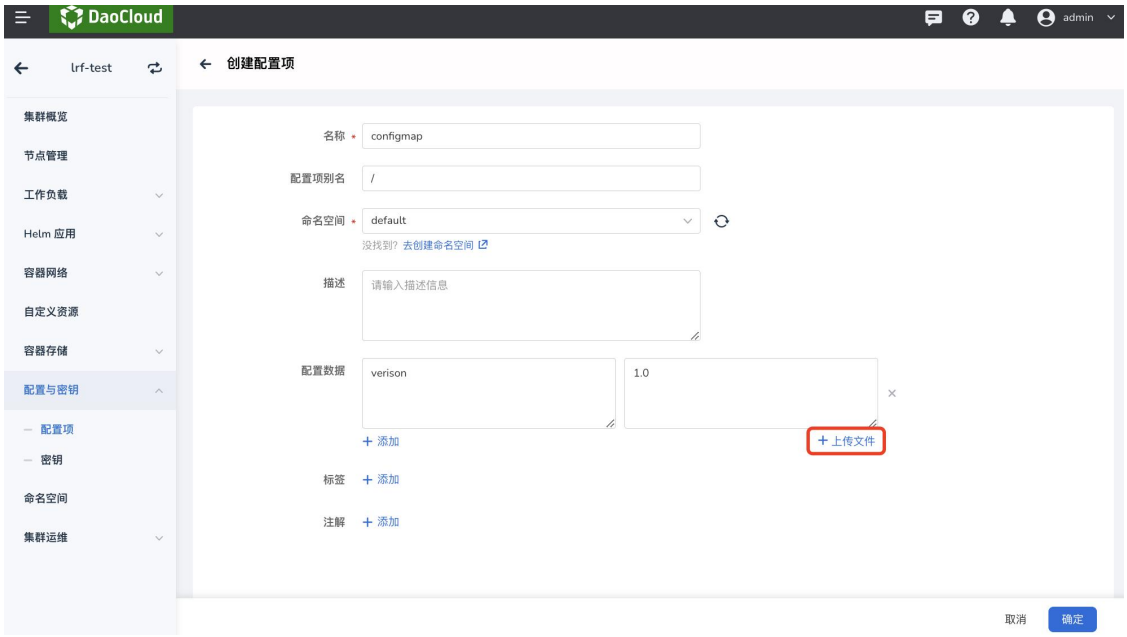


创建配置项

3. 在创建配置项页面中填写配置信息，点击确定。

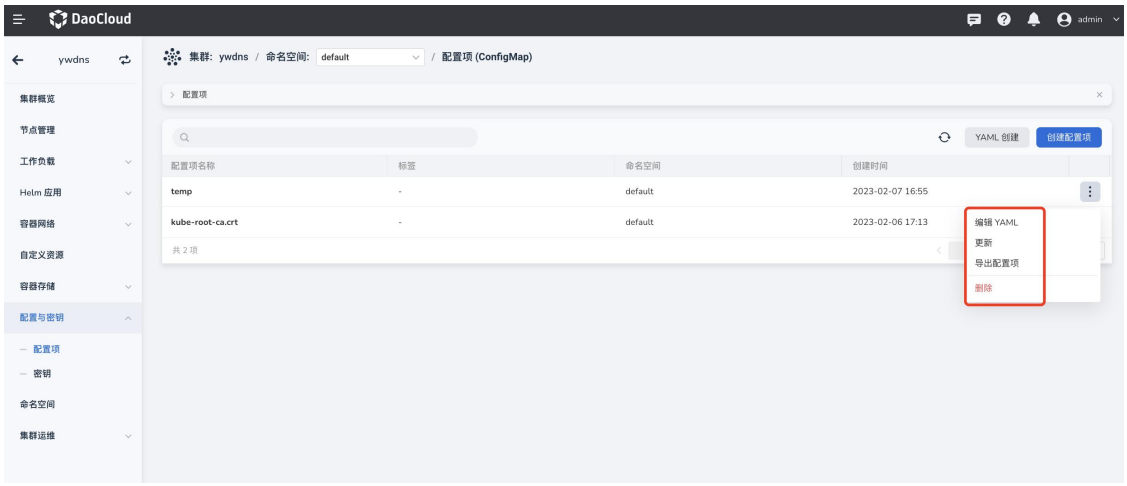
!!! note

点击`上传文件`可以从本地导入已有的文件，快速创建配置项。



创建配置项

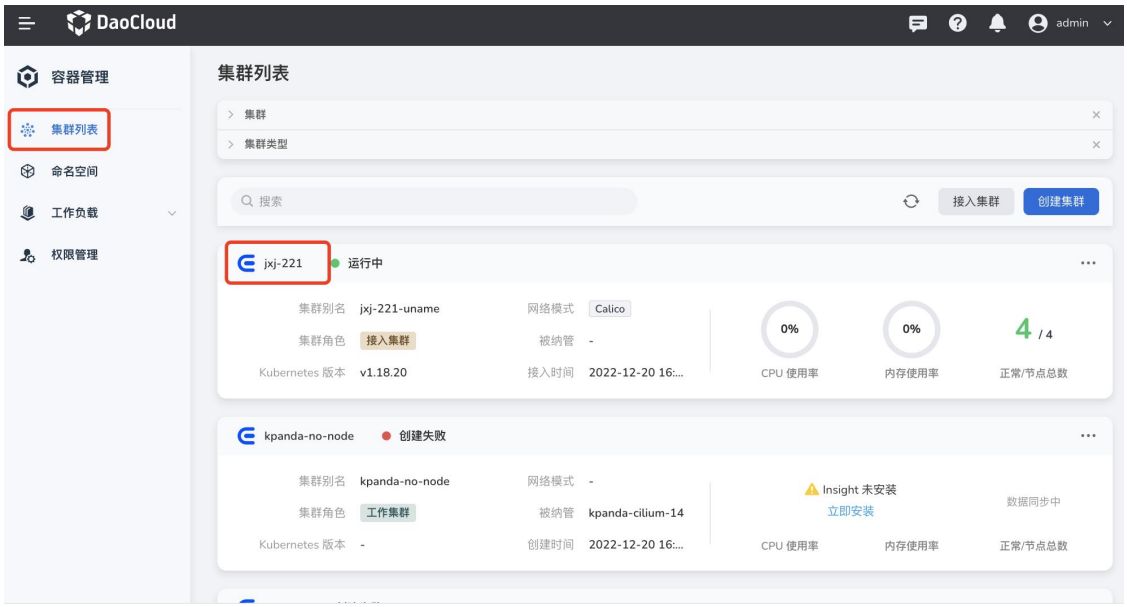
4. 创建完成后在配置项右侧点击更多可以，可以编辑 YAML、更新、导出、删除等操作。



创建配置项

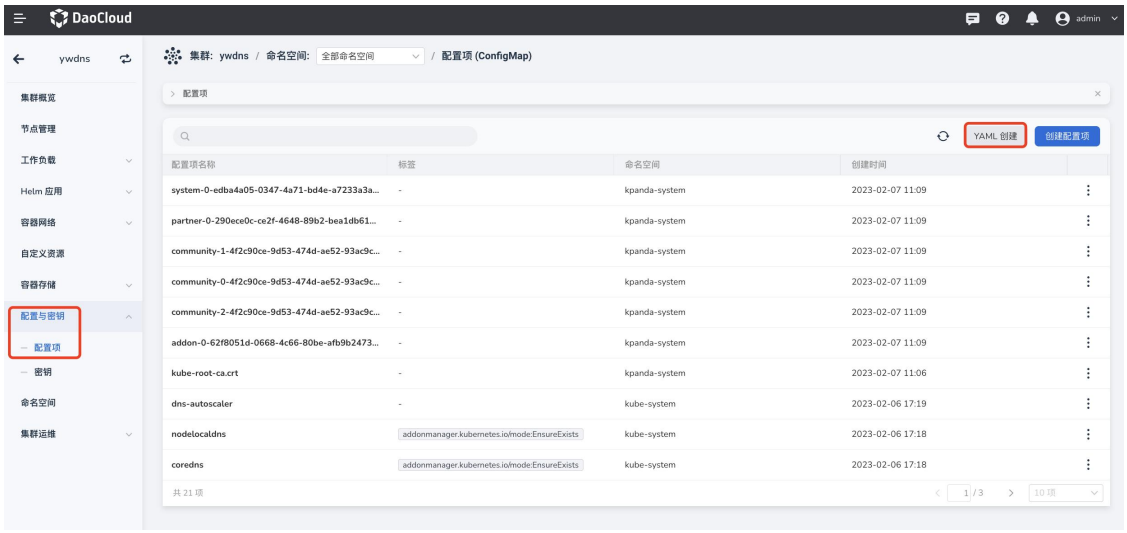
YAML 创建

1. 在集群列表页面点击某个集群的名称，进入集群详情。



集群详情

2. 在左侧导航栏，点击配置与密钥–>配置项，点击右上角 YAML 创建按钮。

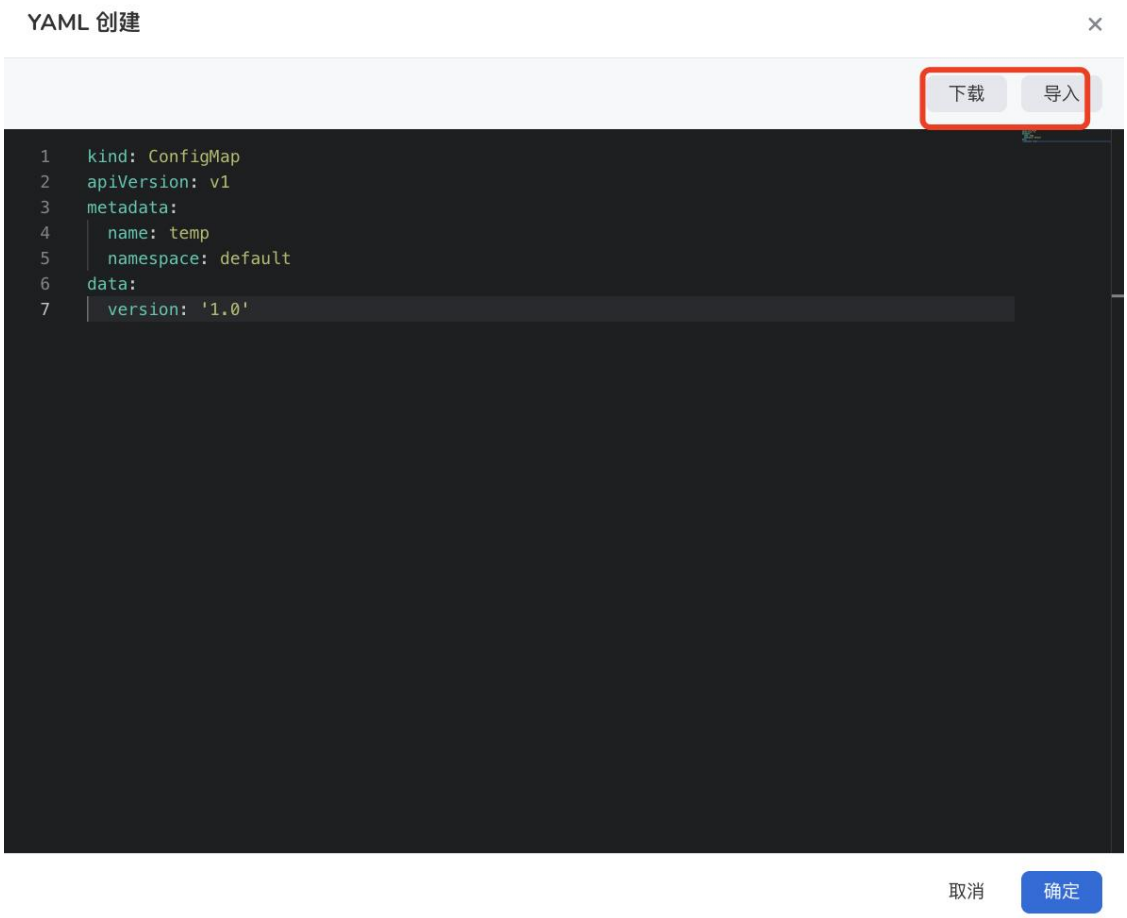


创建配置项

3. 填写或粘贴事先准备好的配置文件，然后在弹框右下角点击确定。

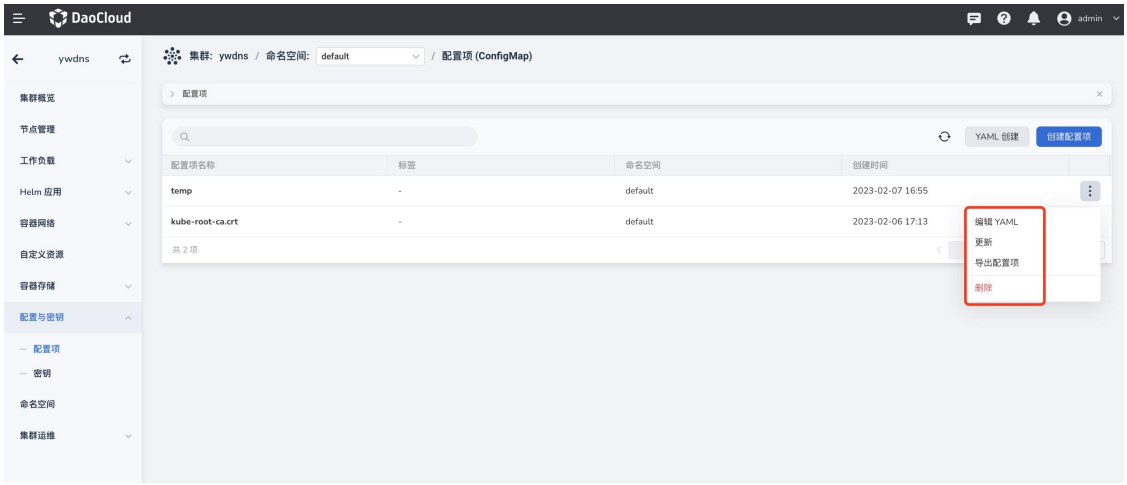
!!! note

- 点击`导入`可以从本地导入已有的文件，快速创建配置项。
- 填写数据之后点击`下载`可以将配置文件保存在本地。



创建配置项

4. 创建完成后在配置项右侧点击更多可以，可以编辑 YAML、更新、导出、删除等操作。



创建配置项

配置项 YAML 示例

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: kube-root-ca.crt
  namespace: default
  annotations:
data:
  version: '1.0'
```

下一步：使用配置项

使用密钥

密钥是一种用于存储和管理密码、OAuth 令牌、SSH、TLS 凭据等敏感信息的资源对象。使用密钥意味着您不需要在应用程序代码中包含敏感的机密数据。

使用场景

您可以在 Pod 中使用密钥，有多种使用场景，主要包括：

- 作为容器的环境变量使用，提供容器运行过程中所需的一些必要信息。
- 使用密钥作为 Pod 的数据卷。
- 在 kubelet 拉取容器镜像时用作镜像仓库的身份认证凭证使用。

使用密钥设置容器的环境变量

您可以通过图形化界面或者终端命令行来使用密钥作为容器的环境变量。

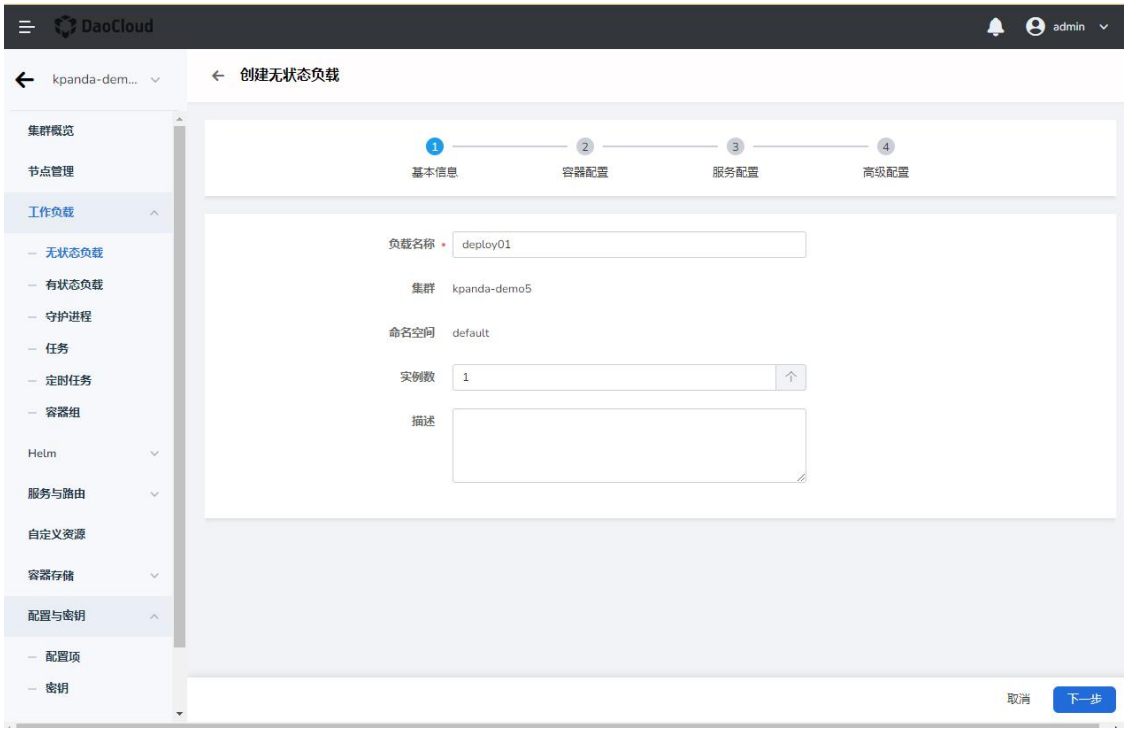
!!! note

密钥导入是将密钥作为环境变量的值；密钥键值导入是将密钥中某一参数作为环境变量的值。

图形界面操作

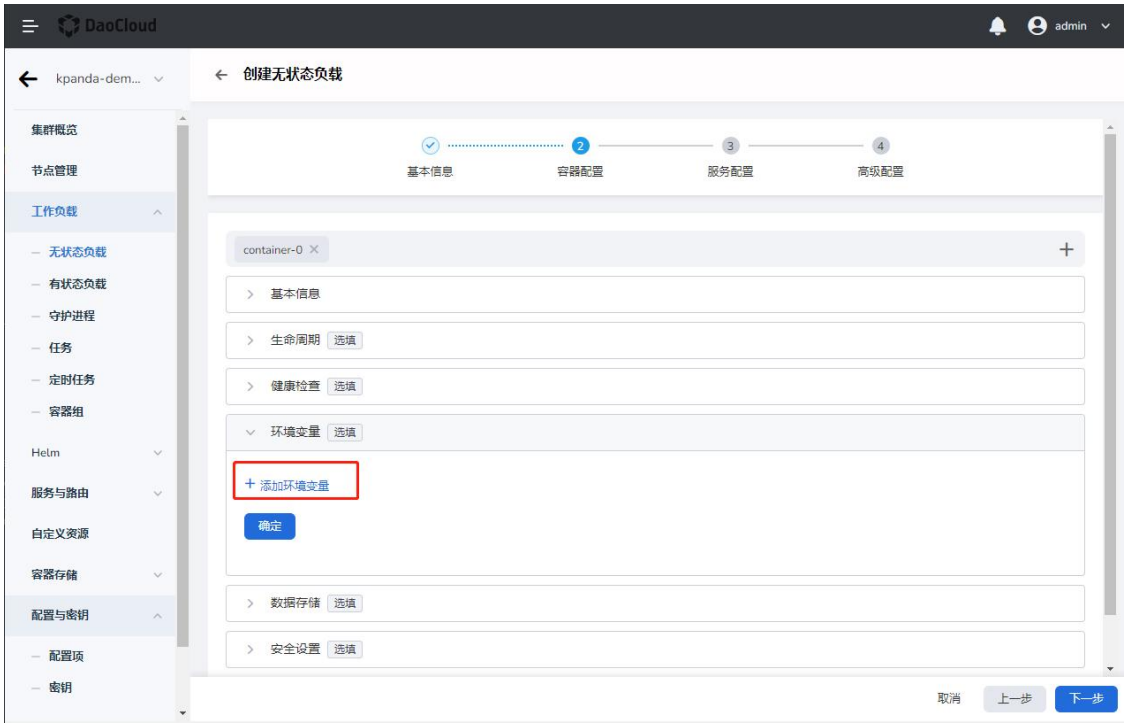
在通过镜像创建工作负载时，您可以在环境变量界面通过选择密钥导入或密钥键值导入为容器设置环境变量。

1. 进入镜像创建工作负载页面。



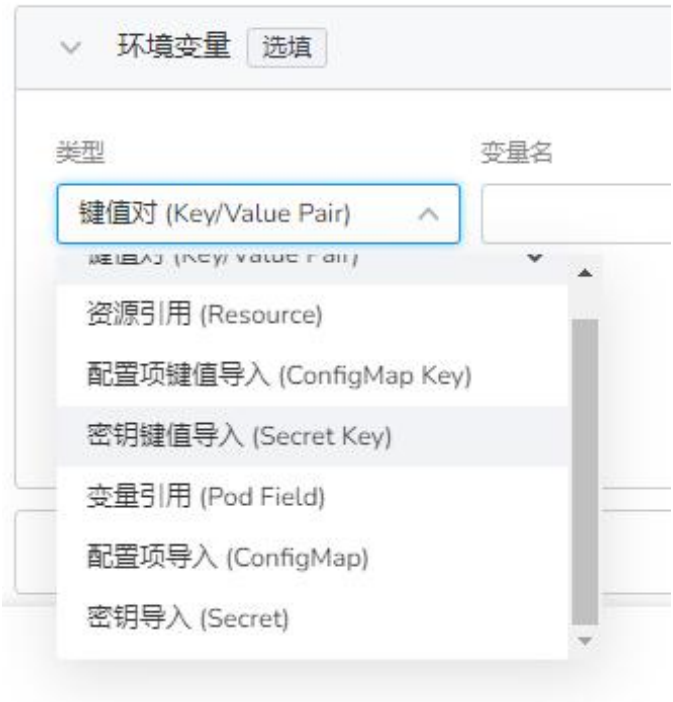
创建 deployment

2. 在容器配置选择环境变量配置，点击添加环境变量按钮。



添加环境变量

3. 在环境变量类型处选择密钥导入或密钥键值导入。



密钥导入

- 当环境变量类型选择为密钥导入时，依次输入变量名、前缀、密钥的名称。
- 当环境变量类型选择为密钥键值导入时，依次输入变量名、密钥、键的名称。

命令行操作

如下例所示，您可以在创建工作负载时将密钥设置为环境变量，使用 valueFrom 参数引用 Secret 中的 Key/Value。

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
```

```
    image: redis
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: username
            optional: false # (1)
      - name: SECRET_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: password
            optional: false # (2)
```

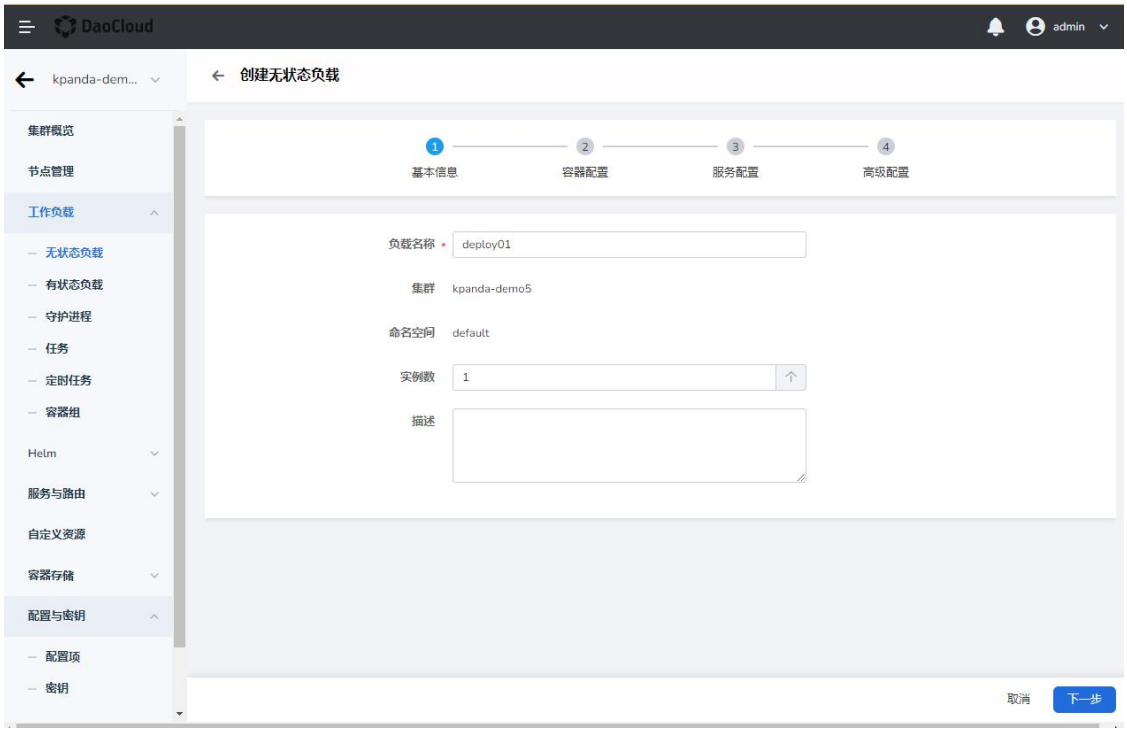
1. 此值为默认值；意味着“mysecret”，必须存在且包含名为“username”的主键
2. 此值为默认值；意味着“mysecret”，必须存在且包含名为“password”的主键

使用密钥作为 Pod 的数据卷

图形界面操作

在通过镜像创建工作负载时，您可以通过在数据存储界面选择存储类型为密钥，将密钥作为容器的数据卷。

1. 进入镜像创建工作负载页面。



创建 deployment

2. 在容器配置选择数据存储配置，在节点路径映射列表点击添加按钮。



创建 deployment

3. 在存储类型处选择密钥，并依次输入容器路径、子路径等信息。

命令行操作

下面是一个通过数据卷来挂载名为 mysecret 的 Secret 的 Pod 示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
      optional: false # (1)
```

1. 默认设置，意味着“mysecret”必须已经存在

如果 Pod 中包含多个容器，则每个容器需要自己的 volumeMounts 块，不过针对每个 Secret 而言，只需要一份 .spec.volumes 设置。

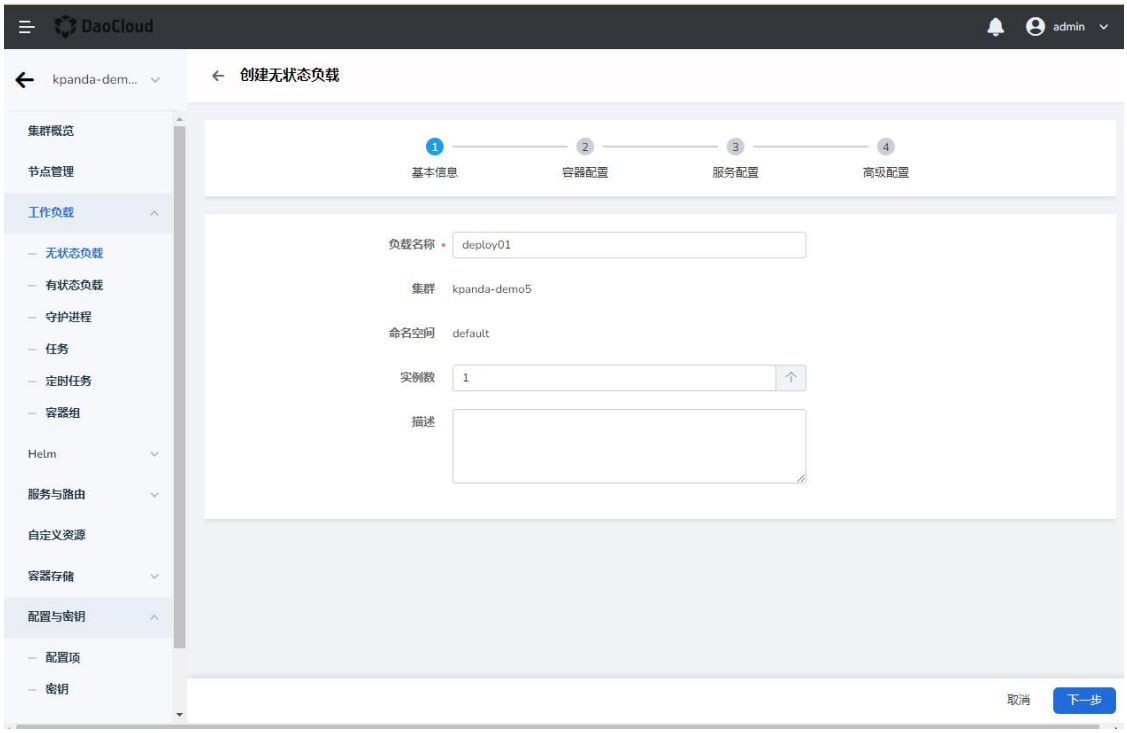
在 kubelet 拉取容器镜像时用作镜像仓库的身份认证凭证

您可以通过图形化界面或者终端命令行来使用密钥作为镜像仓库身份认证凭证。

图形化操作

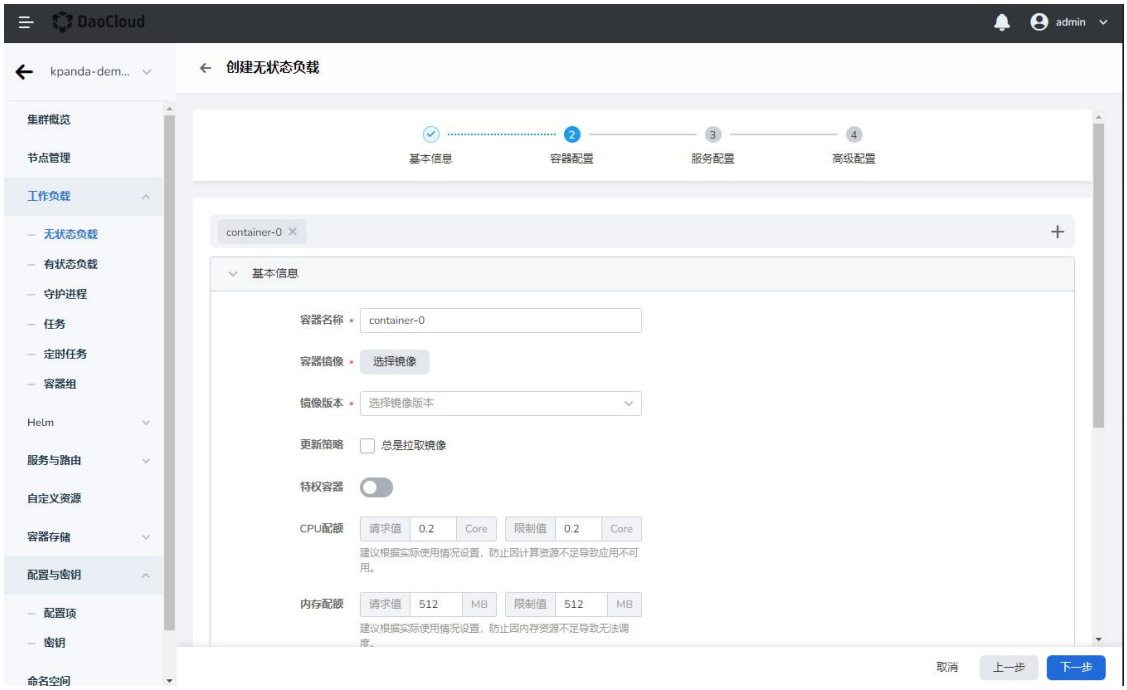
在通过镜像创建工作负载时，您可以通过在数据存储界面选择存储类型为密钥，将密钥作为容器的数据卷。

1. 进入镜像创建工作负载页面。



创建 deployment

2. 在第二步容器配置时选择基本信息配置，点击选择镜像按钮。



选择镜像

3. 在弹框的镜像仓库下拉选择私有镜像仓库名称。关于私有镜像密钥创建请查看创建密钥了解详情。



选择镜像

4. 输入私有仓库内的镜像名称，点击确定，完成镜像选择。

!!! note

创建密钥时，需要确保输入正确的镜像仓库地址、用户名称、密码并选择正确的镜像名称，否则将无法获取镜像仓库中的镜像。

ETCD 备份还原

使用 ETCD 备份功能创建备份策略，可以将指定集群的 etcd 数据定时备份到 S3 存储中。本文主要介绍如何将已经备份的数据还原到当前集群中。

!!! note

- DCE 5.0 ETCD 备份还原仅限于针对同一集群（节点数和 IP 地址没有变化）进行备份与还原。例如，备份了 A 集群 的 etcd 数据后，只能将备份数据还原到 A 集群中，不能还原到 B 集群。
- 对于跨集群的备份与还原，建议使用[应用备份还原](../user-guide/backup/deployment.md)功能。
- 首先创建备份策略，备份当前状态，建议参考[ETCD 备份](../user-guide/backup/etcd-backup.md)功能。

下面通过具体的案例来说明备份还原的整个过程。

环境信息

首先介绍还原的目标集群和 S3 存储的基本信息。这里以 Minlo 作为 S3 存储，整个集群有 3 个控制面（3 个 etcd 副本）。

IP	主机	角色	备注
10.6.212.10	host01	k8s-master01	k8s 节点 1
10.6.212.11	host02	k8s-master02	k8s 节点 2
10.6.212.12	host03	k8s-master03	k8s 节点 3
10.6.212.13	host04	minio	minio 服务

前提条件

安装 etcdbrctl 工具

为了实现 ETCD 数据备份还原，需要在上述任意一个 Kubernetes 节点上安装 etcdbrctl 开源工具。此工具暂时没有二进制文件，需要自行编译。编译方式请参考：https://github.com/gardener/etcd-backup-restore/blob/master/doc/development/local_setup.md#build。

安装完成后用如下命令检查工具是否可用：

```
etcdbrctl -v
```

预期输出如下:

```
INFO[0000] etcd-backup-restore Version: v0.23.0-dev
INFO[0000] Git SHA: b980beec
INFO[0000] Go Version: go1.19.3
INFO[0000] Go OS/Arch: linux/amd64
```

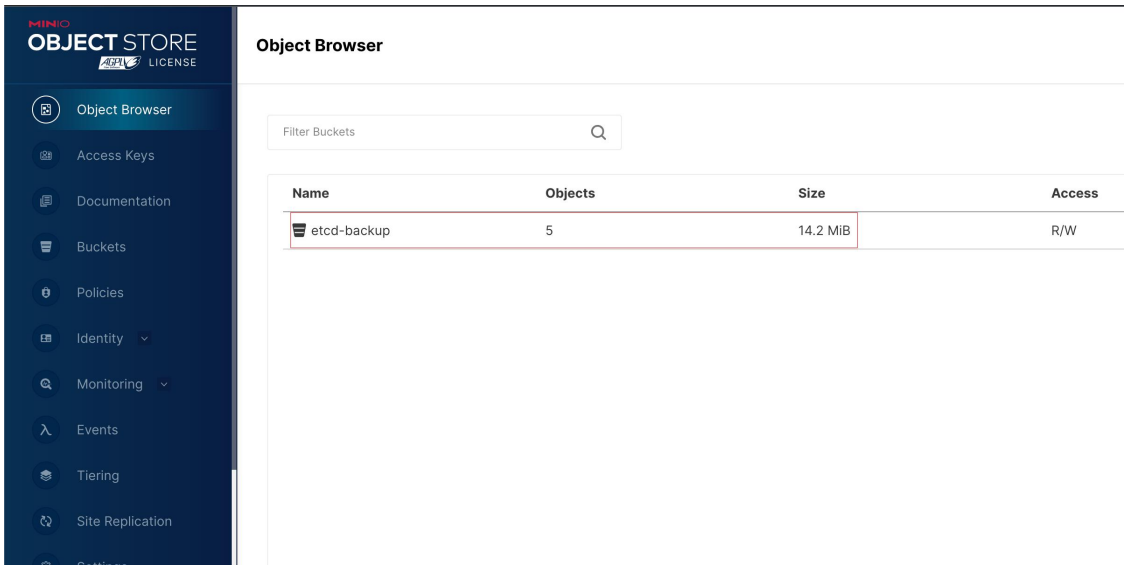
检查备份数据

还原之前需要检查下列事项：

- 是否已经在 DCE 5.0 中成功备份了数据
- 检查 S3 存储中备份数据是否存在

!!! note

DCE 5.0 的备份是全量数据备份，还原时将还原最后一次备份的全量数据。



minio

关闭集群

在备份之前，必须要先关闭集群。默认集群 etcd 和 kube-apiserver 都是以静态 Pod 的形式启动的。这里的关闭集群是指将静态 Pod manifest 文件移动到 /etc/kubernetes/manifest 目录外，集群就会移除对应 Pod，达到关闭服务的作用。

1. 首先删除之前的备份数据，移除数据并非将现有 etcd 数据删除，而是指修改 etcd 数据目录的名称。等备份还原成功之后再删除此目录。这样做的目的是，如果 etcd 备份还原失败，还可以尝试还原当前集群。此步骤每个节点均需执行。

   rm -rf /var/lib/etcd_bak

2. 然后需要关闭 kube-apiserver 的服务，确保 etcd 的数据没有新变化。此步骤每个节点均需执行。

   mv /etc/kubernetes/manifests/kube-apiserver.yaml /tmp/kube-apiserver.yaml

3. 同时还需要关闭 etcd 的服务。此步骤每个节点均需执行。

mv /etc/kubernetes/manifests/etcd.yaml /tmp/etcd.yaml

4. 确保所有控制平面的 kube-apiserver 和 etcd 服务都已经关闭。

5. 关闭所有的节点后，使用如下命令检查 etcd 集群状态。此命令在任意一个节点执行即可。

   endpoints 的值需要替换为实际节点名称

   ```
   etcdctl endpoint status --endpoints=controller-node-1:2379,controller-node-2:2379,controller-node-3:2379 -w table \
     --cacert="/etc/kubernetes/ssl/etcd/ca.crt" \
     --cert="/etc/kubernetes/ssl/apiserver-etcd-client.crt" \
     --key="/etc/kubernetes/ssl/apiserver-etcd-client.key"
   ```

   预期输出如下，表示所有的 etcd 节点都被销毁：

   ```
   {"level":"warn","ts":"2023-03-29T17:51:50.817+0800","logger":"etcd-client","caller":"v3@v3.5.6/retry_interceptor.go:62","msg":"retrying of unary invoker failed","target":"etcd-endpoints://0xc0001ba000/controller-node-1:2379","attempt":0,"error":"rpc error: code = DeadlineExceeded desc = latest balancer error: last connection error: connection error: desc = \"transport: Error while dialing dial tcp 10.5.14.31:2379: connect: connection refused\""}
   Failed to get the status of endpoint controller-node-1:2379 (context deadline exceeded)
   {"level":"warn","ts":"2023-03-29T17:51:55.818+0800","logger":"etcd-client","caller":"v3@v3.5.6/retry_interceptor.go:62","msg":"retrying of unary invoker failed","target":"etcd-endpoints://0xc0001ba000/controller-node-2:2379","attempt":0,"error":"rpc error: code = DeadlineExceeded desc = latest balancer error: last connection error: connection error: desc = \"transport: Error while dialing dial tcp 10.5.14.32:2379: connect: connection refused\""}
   Failed to get the status of endpoint controller-node-2:2379 (context deadline exceeded)
   {"level":"warn","ts":"2023-03-29T17:52:00.820+0800","logger":"etcd-client",
   ```

"caller":"v3@v3.5.6/retry_interceptor.go:62","msg":"retrying of unary invoker failed","target":"etcd-endpoints://0xc0001ba000/controller-node-1:2379","attempt":0,"error":"rpc error: code = DeadlineExceeded desc = latest balancer error: last connection error: connection error: desc = \"transport: Error while dialing dial tcp 10.5.14.33:2379: connect: connection refused\""}
Failed to get the status of endpoint controller-node-3:2379 (context deadline exceeded)
+----------+----+---------+---------+-----------+----------+-----------+------------+--------------------+--------+
| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+----------+----+---------+---------+-----------+----------+-----------+------------+--------------------+--------+
+----------+----+---------+---------+-----------+----------+-----------+------------+--------------------+--------+

还原备份

只需要还原一个节点的数据，其他节点的 etcd 数据就会自动进行同步。

1. 设置环境变量

   使用 etcdbrctl 还原数据之前，执行如下命令将连接 S3 的认证信息设置为环境变量：

   ```
   export ECS_ENDPOINT=http://10.6.212.13:9000 # (1)
   export ECS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE # (2)
   export ECS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY # (3)
   ```

   1. S3 存储的访问点
   2. S3 存储的用户名
   3. S3 存储的密码

2. 执行还原操作

执行 etcdbrctl 命令行工具执行还原，这是最关键的一步。

```
etcdbrctl restore --data-dir /var/lib/etcd/ --store-container="etcd-backup" \
  --storage-provider=ECS \
  --initial-cluster=controller-node1=https://10.6.212.10:2380 \
  --initial-advertise-peer-urls=https://10.6.212.10:2380
```

参数说明如下：

- —data-dir: etcd 数据目录，此目录必须跟 etcd 数据目录一致，etcd 才能正常加载数据。
- —store-container：S3 存储的位置，MinIO 中对应的 bucket，必须跟数据备份的 bucket 相对应。
- —initial-cluster：etcd 初始化配置, etcd 集群的名称必须跟原来一致。
- —initial-advertise-peer-urls：etcd member 集群之间访问地址。必须跟 etcd 的配置保持一致。

预期输出如下：

```
INFO[0000] Finding latest set of snapshot to recover from...
INFO[0000] Restoring from base snapshot: Full-00000000-00111147-1679991074  actor=restorer
INFO[0001] successfully fetched data of base snapshot in 1.241380207 seconds  actor=restorer
{"level":"info","ts":1680011221.2511616,"caller":"mvcc/kvstore.go:380","msg":"restored last compact revision","meta-bucket-name":"meta","meta-bucket-name-key":"finishedCompactRev","restored-compact-revision":110327}
{"level":"info","ts":1680011221.3045986,"caller":"membership/cluster.go:392","msg":"added member","cluster-id":"66638454b9dd7b8a","local-member-id":"0","added-peer-id":"123c2503a378fc46","added-peer-peer-urls":["https://10.6.212.10:2380"]}
```

INFO[0001] Starting embedded etcd server... actor=restorer

....

{"level":"info","ts":"2023-03-28T13:47:02.922Z","caller":"embed/etcd.go:565","msg":"stopped serving peer traffic","address":"127.0.0.1:37161"}
{"level":"info","ts":"2023-03-28T13:47:02.922Z","caller":"embed/etcd.go:367","msg":"closed etcd server","name":"default","data-dir":"/var/lib/etcd","advertise-peer-urls":["http://localhost:0"],"advertise-client-urls":["http://localhost:0"]}
INFO[0003] Successfully restored the etcd data directory.

!!! note "可以查看 etcd 的 YAML 文件进行对照，以免配置错误"

```shell
cat /tmp/etcd.yaml | grep initial-
- --experimental-initial-corrupt-check=true
- --initial-advertise-peer-urls=https://10.6.212.10:2380
- --initial-cluster=controller-node-1=https://10.6.212.10:2380
```

3. 以下命令在节点 01 上执行，为了恢复节点 01 的 etcd 服务。

   首先将 etcd 静态 Pod 的 manifest 文件移动到 /etc/kubernetes/manifests 目录下，kubelet 将会重启 etcd：

   mv /tmp/etcd.yaml /etc/kubernetes/manifests/etcd.yaml

   然后等待 etcd 服务启动完成以后，检查 etcd 的状态，etcd 相关证书默认目录：/etc/kubernetes/ssl。如果集群证书存放在其他位置，请指定对应路径。

   - 检查 etcd 集群列表:

```
etcdctl member list -w table \
--cacert="/etc/kubernetes/ssl/etcd/ca.crt" \
--cert="/etc/kubernetes/ssl/apiserver-etcd-client.crt" \
--key="/etc/kubernetes/ssl/apiserver-etcd-client.key"
```

预期输出如下：

```
+------------------+---------+-------------------+----
----------------------+-------------------------+----
--------+
|        ID        | STATUS  |        NAME        |        PEER
ADDRS        |        CLIENT ADDRS        | IS LEARNER |
+------------------+---------+-------------------+----
----------------------+-------------------------+----
--------+
| 123c2503a378fc46 | started | controller-node-1 | https://10.6.212.
10:2380 | https://10.6.212.10:2379 |        false |
+------------------+---------+-------------------+----
----------------------+-------------------------+----
--------+
```

- 查看 controller-node-1 状态:

```
etcdctl endpoint status --endpoints=controller-node-1:2379 -w tab
le \
--cacert="/etc/kubernetes/ssl/etcd/ca.crt" \
--cert="/etc/kubernetes/ssl/apiserver-etcd-client.crt" \
--key="/etc/kubernetes/ssl/apiserver-etcd-client.key"
```

预期输出如下：

```
+-----------------------+------------------+---------
+---------+-----------+-----------+-----------+------
------+-------------------+--------+
```

```
| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+----------------------+-----------------+---------+---------+-----------+-----------+-----------+------------------+--------+
| controller-node-1:2379 | 123c2503a378fc46 |  3.5.6 |  15 MB |      true |     false |         3 |       1200 |               1199 |        |
+----------------------+-----------------+---------+---------+-----------+-----------+-----------+------------------+--------+
```

4. 恢复其他节点数据

上述步骤已经还原了节点 01 的数据，若想要还原其他节点数据，只需要将 etcd 的 Pod 启动起来，让 etcd 自己完成数据同步。

- 在节点 02 和节点 03 都执行相同的操作：

  ```
  mv /tmp/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
  ```

- etcd member 集群之间的数据同步需要一定的时间，可以查看 etcd 集群状态，确保所有 etcd 集群正常：

  检查 etcd 集群状态是否正常:

  ```
  etcdctl member list -w table \
  --cacert="/etc/kubernetes/ssl/etcd/ca.crt" \
  --cert="/etc/kubernetes/ssl/apiserver-etcd-client.crt" \
  --key="/etc/kubernetes/ssl/apiserver-etcd-client.key"
  ```

  预期输出如下：

```
+------------------+---------+-------------------+----
--------------------+-----------------------+------
------+
|        ID        | STATUS  |    NAME           |       PEER
ADDRS        |       CLIENT ADDRS       | IS LEARNER |
+------------------+---------+-------------------+----
--------------------+-----------------------+------
------+
| 6ea47110c5a87c03 | started | controller-node-1 | https://10.5.14.31:
2380 | https://10.5.14.31:2379 |       false |
| e222e199f1e318c4 | started | controller-node-2 | https://10.5.14.3
2:2380 | https://10.5.14.32:2379 |       false |
| f64eeda321aabe2d | started | controller-node-3 | https://10.5.14.3
3:2380 | https://10.5.14.33:2379 |       false |
+------------------+---------+-------------------+----
--------------------+-----------------------+------
------+
```

检查 3 个 member 节点是否正常:

```
etcdctl endpoint status --endpoints=controller-node-1:2379,controll
er-node-2:2379,controller-node-3:2379 -w table \
--cacert="/etc/kubernetes/ssl/etcd/ca.crt" \
--cert="/etc/kubernetes/ssl/apiserver-etcd-client.crt" \
--key="/etc/kubernetes/ssl/apiserver-etcd-client.key"
```

预期输出如下:

```
+-----------------------+------------------+---------
+---------+-----------+-----------+-----------+------
------+-------------------+--------+
|     ENDPOINT          |        ID        | VERSION | DB SIZE
 | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT A
```

```
PPLIED INDEX | ERRORS |
+---------------------+-----------------+--------
+--------+----------+----------+----------+------
------+------------------+--------+
| controller-node-1:2379 | 6ea47110c5a87c03 |  3.5.6 |  88 MB |
      true |      false |          6 |     199008 |              1990
08 |          |
| controller-node-2:2379 | e222e199f1e318c4 |  3.5.6 |  88 MB |
     false |      false |          6 |     199114 |              19911
4 |          |
| controller-node-3:2379 | f64eeda321aabe2d |  3.5.6 |  88 MB |
     false |      false |          6 |     199316 |              19931
6 |          |
+---------------------+-----------------+--------
+--------+----------+----------+----------+------
------+------------------+--------+
```

恢复集群

等所有节点的 etcd 数据同步完成后，则可以将 kube-apiserver 进行重新启动，将整个集群恢复到可访问状态：

1. 重新启动 node1 的 kube-apiserver 服务

   mv /tmp/kube-apiserver.yaml /etc/kubernetes/manifests/kube-apiserver.yaml

2. 重新启动 node2 的 kube-apiserver 服务

   mv /tmp/kube-apiserver.yaml /etc/kubernetes/manifests/kube-apiserver.yaml

3. 重新启动 node3 的 kube-apiserver 服务

mv /tmp/kube-apiserver.yaml /etc/kubernetes/manifests/kube-apiserver.yaml

4. 等待 kubelet 将 kube-apiserver 启动后，检查还原的 k8s 数据是否正常：

   kubectl get nodes

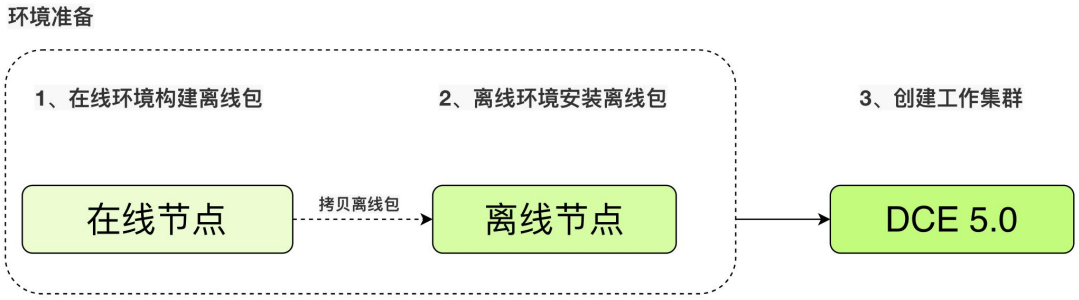
   预期输出如下：

NAME	STATUS	ROLES	AGE	VERSION
controller-node-1	Ready	<none>	3h30m	v1.25.4
controller-node-3	Ready	control-plane	3h29m	v1.25.4
controller-node-3	Ready	control-plane	3h28m	v1.25.4

在非主流操作系统上创建集群

本文介绍离线模式下如何在 **未声明支持的 OS** 上创建工作集群。DCE 5.0 声明支持的 OS 范围请参考 DCE 5.0 支持的操作系统

离线模式下在未声明支持的 OS 上创建工作集群，主要的流程如下图：



流程

接下来，本文将以 openAnolis 操作系统为例，介绍如何在非主流操作系统上创建集群。

前提条件

- 已经部署好一个 DCE 5.0 全模式，部署参考文档离线安装 DCE 5.0 商业版
- 至少拥有一台可以联网的同架构同版本的节点。

操作步骤

在线节点——构建离线包

找到一个和待建集群节点架构和 OS 均一致的在线环境，本文以 AnolisOS 8.8 GA 为例。执行如下命令，生成离线 os-pkgs 包。

```
# 下载相关脚本并构建 os packages 包
$ curl -Lo ./pkgs.yml https://raw.githubusercontent.com/kubean-io/kubean/main/build/os-packages/others/pkgs.yml
```

```
$ curl -Lo ./other_os_pkgs.sh https://raw.githubusercontent.com/kubean-io/kubean/main/build/os-packages/others/other_os_pkgs.sh && chmod +x other_os_pkgs.sh
$ ./other_os_pkgs.sh build #构建离线包
```

执行完上述命令后，预期将在当前路径下生成一个名为 **os-pkgs-anolis-8.8.tar.gz** 的压缩包。当前路径下文件目录大概如下：

```
.
├── other_os_pkgs.sh
├── pkgs.yml
└── os-pkgs-anolis-8.8.tar.gz
```

离线节点——安装离线包

将在线节点中生成的 other_os_pkgs.sh、pkgs.yml、os-pkgs-anolis-8.8.tar.gz 三个文件拷贝至离线环境中的待建集群的**所有**节点上。

登录离线环境中，任一待建集群的其中一个节点上，执行如下命令，为节点安装 os-pkg 包。

```
# 配置环境变量
$ export PKGS_YML_PATH=/root/workspace/os-pkgs/pkgs.yml # 当前离线节点 pkgs.yml 文件的路径
$ export PKGS_TAR_PATH=/root/workspace/os-pkgs/os-pkgs-anolis-8.8.tar.gz # 当前离线节点 os-pkgs-anolis-8.8.tar.gz 的路径
$ export SSH_USER=root # 当前离线节点的用户名
$ export SSH_PASS=dangerous # 当前离线节点的密码
$ export HOST_IPS='172.30.41.168' # 当前离线节点的 IP
$ ./other_os_pkgs.sh install #安装离线包
```

执行完成上述命令后，等待界面提示： All packages for node (X.X.X.X) have been installed 即表示安装完成。

前往界面创建集群

参考文档创建工作集群，创建 openAnolis 集群。

在 centos 管理平台上创建 ubuntu 工作集群

本文介绍如何在已有的 centos 管理平台上创建 ubuntu 工作集群。

!!! note

本文仅针对离线模式下，使用 DCE 5.0 平台创建工作集群，管理平台和待建工作集群的架构均为 AMD。创建集群时不支持异构（AMD 和 ARM 混合）部署，您可以在集群创建完成后，通过接入异构节点的方式进行集群混合部署管理。

前提条件

- 已经部署好一个 DCE 5.0 全模式，并且火种节点还存活，部署参考文档离线安装 DCE 5.0 商业版

操作步骤

下载并导入 ubuntu 相关离线包

请确保已经登录到火种节点！并且之前部署 DCE 5.0 时使用的 clusterConfig.yaml 文件还在。

下载 ubuntu 相关离线包

下载所需的 ubuntu OS package 包和 ISO 离线包

资源名	说明	下载地址
os-pkgs-ubuntu1804-v0.6.6.tar.gz	Ubuntu1804 OS-package 包	https://github.com/kubean-io/kubean/releases/download/v0.6.6/os-pkgs-ubuntu1804-v0.6.6.tar.gz
ISO 离线包	OS 包导入火种节点脚本	http://mirrors.melbourne.co.uk/ubuntu-releases/
import-iso	ISO 导入火种节点脚本	https://github.com/kubean-io/kubean/releases/download/v0.6.6/i

资源名	说明	下载地址
		mport_iso.sh

导入 os pckage 离线包至火种节点的 minio

解压 ubuntu os pckage 离线包

执行如下命令解压下载的 os pckage 离线包。

```
# 此处我们下载的 ubuntu os pckage 离线包
tar -xvf os-pkgs-ubuntu1804-v0.6.6.tar.gz
```

os package 解压后的文件内容如下：

```
os-pkgs
├── import_ospkgs.sh          # 该脚本用于导入 os packages 到 minio 文件服务
├── os-pkgs-amd64.tar.gz      # amd64 架构的 os packages 包
├── os-pkgs-arm64.tar.gz      # arm64 架构的 os packages 包
└── os-pkgs.sha256sum.txt     # os packages 包的 sha256sum 效验文件
```

导入 OS Package 至火种节点的 minio

执行如下命令，将 os packages 包到 minio 文件服务中:

```
MINIO_USER=rootuser MINIO_PASS=rootpass123 ./import_ospkgs.sh http://127.0.0.1:9000 os-pkgs-ubuntu1804-v0.6.6.tar.gz
```

!!! note

上述命令仅仅适用于火种节点内置的 Minio 服务，如果使用外部 Minio 请将“http://127.0.0.1:9000”替换为外部 Minio 的访问地址。“rootuser”和“rootpass123”是火种节点内置的 Minio 服务的默认账户和密码。“os-pkgs-ubuntu1804-v0.6.6.tar.gz“ 为所下载的 os package 离线包的名称。

导入 ISO 离线包至火种节点的 minio

执行如下命令，将 ISO 包到 minio 文件服务中:

MINIO_USER=rootuser MINIO_PASS=rootpass123 ./import_iso.sh http://127.0.0.1:9000 ubuntu-16.04.7-server-amd64.iso

!!! note

上述命令仅仅适用于火种节点内置的 Minio 服务，如果使用外部 Minio 请将“http://127.0.0.1:9000”替换为外部 Minio 的访问地址。“rootuser”和“rootpass123”是火种节点内置的 Minio 服务的默认账户和密码。“ubuntu-16.04.7-server-amd64.iso“ 为所下载的 ISO 离线包。

前往界面创建集群

参考文档创建工作集群，创建 ubuntu 集群。

为工作集群添加异构节点

本文介绍如何为 AMD 架构，操作系统为 CentOS 7.9 的工作集群添加 ARM 架构，操作系统为 Kylin v10 sp2 的工作节点

!!! note

本文仅针对离线模式下，使用 DCE 5.0 平台所创建的工作集群进行异构节点的添加，不包括接入的集群。

前提条件

- 已经部署好一个 DCE 5.0 全模式，并且火种节点还存活，部署参考文档离线安装 DCE 5.0 商业版
- 已经通过 DCE 5.0 平台创建好一个 AMD 架构，操作系统为 CentOS 7.9 的工作集群，创建参考文档创建工作集群

操作步骤

下载并导入离线包（以 ARM 架构、操作系统 Kylin v10 sp2 为例）

请确保已经登录到火种节点！并且之前部署 DCE 5.0 时使用的 clusterConfig.yaml 文件还在。

离线镜像包

!!! note

可以在[下载中心](https://docs.daocloud.io/download/dce5/)下载最新版本。

CPU 架构	版本	下载地址
AMD64	v0.8.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.8.0-amd64.tar

CPU 架构	版本	下载地址
ARM64	v0.8.0	https://qiniu-download-public.daocloud.io/DaoCloud_Enterprise/dce5/offline-v0.8.0-arm64.tar

下载完毕后解压离线包：

```
# 此处我们下载 arm64 架构的离线包
tar -xvf offline-v0.8.0-arm64.tar
```

ISO 离线包（Kylin v10 sp2）

CPU 架构	操作系统版本	下载地址
ARM64	Kylin Linux Advanced Server release V10 (Sword) SP2	申请地址：https://www.kylinos.cn/scheme/server/1.html 注意：麒麟操作系统需要提供个人信息才能下载使用，下载时请选择 V10 (Sword) SP2

osPackage 离线包 （Kylin v10 sp2）

其中 Kubean 提供了不同操作系统的 osPackage 离线包，可以前往 https://github.com/kubean-io/kubean/releases 查看。

操作系统版本	下载地址
Kylin Linux Advanced Server release V10 (Sword) SP2	https://files.m.daocloud.io/github.com/kubean-io/kubean/releases/download/v0.5.2/os-pkgs-kylinv10-v0.5.2.tar.gz

!!! note

osPackage 离线包的具体对应版本请查看离线镜像包中 `offline/sample/clusterConfig.yaml` 中对应的 kubean 版本

导入离线包至火种节点

执行 import-artifact 命令：

```
./offline/dce5-installer import-artifact -c clusterConfig.yaml \
    --target-arch=arm64 \
    --offline-path=/root/offline \
    --iso-path=/root/Kylin-Server-10-SP2-aarch64-Release-Build09-20210524.iso \
    --os-pkgs-path=/root/os-pkgs-kylinv10-v0.5.2.tar.gz
```

!!! note

参数说明：

- `-c clusterConfig.yaml` 指定之前部署 DCE5.0 时使用的 clusterConfig.yaml 文件
- `--target-arch` 指定架构，支持 arm64、amd64
- `--offline-path` 指定下载的离线镜像包文件地址
- `--iso-path` 指定下载的 ISO 操作系统镜像文件地址
- `--os-pkgs-path` 指定下载的 osPackage 离线包文件地址

导入命令执行成功后，会将离线包上传到火种节点的 Minio 中。

添加异构工作节点

请确保已经登录到 DCE5.0 的 Global 集群的管理节点上。

修改主机清单文件

主机清单文件示例：

===“新增节点前”

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
    name: ${cluster-name}-hosts-conf
    namespace: kubean-system
data:
    hosts.yml: |
        all:
        hosts:
            centos-master:
            ip: 10.5.14.122
            access_ip: 10.5.14.122
            ansible_host: 10.5.14.122
            ansible_connection: ssh
            ansible_user: root
            ansible_ssh_pass: ******
        children:
            kube_control_plane:
            hosts:
                centos-master:
            kube_node:
            hosts:
                centos-master:
            etcd:
            hosts:
                centos-master:
            k8s_cluster:
            children:
                kube_control_plane:
                kube_node:
```

===“新增节点后”

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
    name: ${cluster-name}-hosts-conf
    namespace: kubean-system
data:
    hosts.yml: |
        all:
        hosts:
            centos-master:
            ip: 10.5.14.122
            access_ip: 10.5.14.122
            ansible_host: 10.5.14.122
            ansible_connection: ssh
            ansible_user: root
            ansible_ssh_pass: ******
            # 添加异构节点信息
            kylin-worker:
            ip: 10.5.10.220
            access_ip: 10.5.10.220
            ansible_host: 10.5.10.220
            ansible_connection: ssh
            ansible_user: root
            ansible_ssh_pass: dangerous@2022
        children:
            kube_control_plane:
            hosts:
                centos-master:
            kube_node:
```

```
        hosts:
            centos-master:
            kylin-worker: # 添加新增的异构节点名称
        etcd:
        hosts:
            centos-master:
        k8s_cluster:
        children:
            kube_control_plane:
            kube_node:
```

按照上述的配置注释，添加新增的工作节点信息

```
# cluster-name 为工作集群的名称，通过容器管理创建集群时会默认生成
kubectl edit cm ${cluster-name}-hosts-conf -n kubean-system
```

通过 ClusterOperation.yml 新增扩容任务

示例：

```yaml
apiVersion: kubean.io/v1alpha1
kind: ClusterOperation
metadata:
  name: add-worker-node
spec:
  cluster: ${cluster-name} # 指定 cluster name
  image: ghcr.m.daocloud.io/kubean-io/spray-job:v0.5.0
  backoffLimit: 0
  actionType: playbook
  action: scale.yml
  extraArgs: --limit=kylin-worker
  preHook:
    - actionType: playbook
      action: ping.yml
```

```
    - actionType: playbook
      action: disable-firewalld.yml
    - actionType: playbook
      action: enable-repo.yml
      extraArgs: |
        -e "{repo_list: ["http://10.5.14.30:9000/kubean/kylin-iso/\$releasever/os/
\$basearch","http://10.5.14.30:9000/kubean/kylin/\$releasever/os/\$basearch"]}"
  postHook:
    - actionType: playbook
      action: cluster-info.yml
```

!!! note

- spec.image 镜像地址要与之前执行部署时的 job 其内镜像保持一致
- spec.action 设置为 scale.yml
- spec.extraArgs 设置为 --limit=g-worker
- pec.preHook 中的 enable-repo.yml 剧本参数，要填写相关 OS 的正确的 repo_list

按照上述的配置，创建并部署 join-node-ops.yaml

```
# 复制上述清单文件
vi join-node-ops.yaml
kubectl kubectl apply -f join-node-ops.yaml -n kubean-system
```

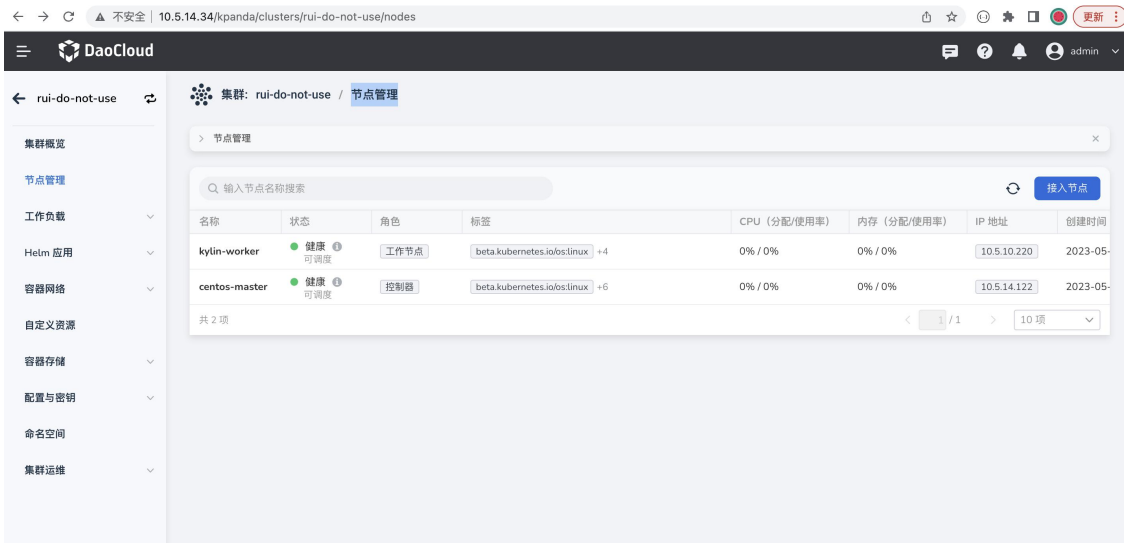
检查任务执行状态

```
kubectl -n kubean-system get pod | grep add-worker-node
```

了解缩容任务执行进度，可查看该 Pod 日志。

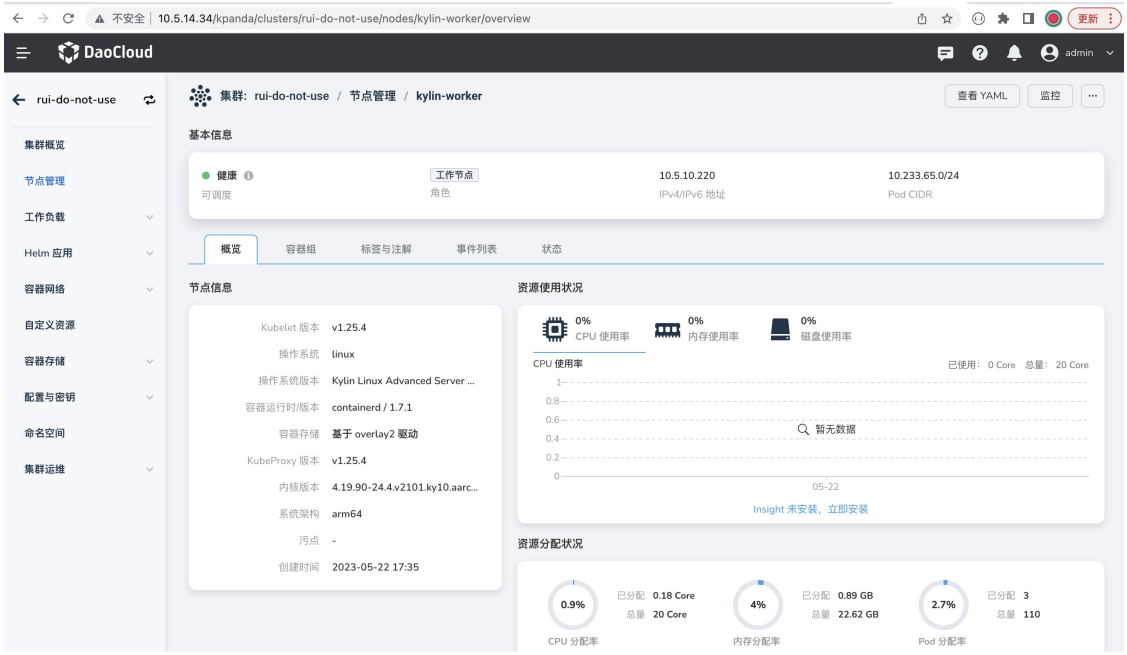
前往界面验证

1. 前往容器管理->集群->节点管理



节点管理

2. 点击新增的节点，查看详情



节点详情

工作集群离线升级指南

!!! note

本文仅针对离线模式下，使用 DCE 5.0 平台所创建的工作集群的 kubernetes 的版本进行升级，不包括其它 kubeneters 组件的升级。

概述

在离线场景下，用户可以通过制作增量离线包的方式对使用 DCE 5.0 平台所创建的工作集群的 kubernetes 的版本进行升级。整体的升级思路为：在联网节点构建离线包 → 将离线包导入火种节点 → 更新 Global 集群的 kubernetes 版本清单 → 使用平台 UI 升级工作集群的 kubernetes 版本 。

!!! note

目前支持构建的离线 kubernetes 版本如下：

– v1.26.3, v1.26.2, v1.26.1, v1.26.0, v1.25.8
– v1.25.7, v1.25.6, v1.25.5, v1.25.4, v1.25.3, v1.25.2, v1.25.1, v1.25.0,
– v1.24.12, v1.24.11, v1.24.10, v1.24.9, v1.24.8, v1.24.7, v1.24.6, v1.24.5, v1.24.4, v1.24.3, v1.24.2, v1.24.1, v1.24.0

在联网节点构建离线包

由于离线环境无法联网，用户需要事先准备一台能够**联网的节点**来进行增量离线包的构建，并且在这个节点上启动 Docker 服务。如何安装 Docker?

1. 检查联网节点的 Docker 服务运行状态

```
# 查看节点下 Docker 进程的运行状态
ps aux|grep docker

# 预期输出如下:
```

```
root       12341  0.5  0.2 654372 26736 ?        Ssl  23:45   0:00 /usr/
bin/docked
root       12351  0.2  0.1 625080 13740 ?        Ssl  23:45   0:00 dock
er-containerd --config /var/run/docker/containerd/containerd.toml
root       13024  0.0  0.0 112824   980 pts/0    S+   23:45   0:00 grep
 --color=auto docker
```

2. 在联网节点的 /root 目录下创建一个名为 manifest.yaml 的文件，命令如下：

   ```
   vi manifest.yaml
   ```

   manifest.yaml 内容如下：

   ```
   image_arch:
   - "amd64"
   kube_version:
   - "v1.25.5"
   - "v1.25.4"
   - "v1.24.8"
   - "v1.24.5"
   ```

   - image_arch 用于指定 CPU 的架构类型，可填入的参数为 amd64 和 arm64。
   - kube_version 用于指定需要构建的 kubernetes 离线包版本，可参考上文的支持构建的离线 kubernetes 版本。

3. 在 /root 目录下新建一个名为 /data 的文件夹来存储增量离线包。

   ```
   mkdir data
   ```

   执行如下命令，使用 ghcr.m.daocloud.io/kubean-io/airgap-patch:v0.4.8 镜像生成离线包。

更多关于 ghcr.m.daocloud.io/kubean-io/airgap-patch:v0.4.8 镜像的信息，请前往 kubean。

```
docker run --rm -v $(pwd)/manifest.yml:/manifest.yml -v $(pwd)/data:/data ghcr.m.daocloud.io/kubean-io/airgap-patch:v0.4.8
```

等待 docker 服务运行完成后，检查 /data 文件夹下的文件，文件目录如下：

```
data
└── v_offline_patch
    ├── amd64
    │   ├── files
    │   │   ├── import_files.sh
    │   │   └── offline-files.tar.gz #二进制文件
    │   └── images
    │       ├── import_images.sh
    │       └── offline-images.tar.gz #镜像文件
    └── kubeanofflineversion.cr.patch.yaml
```

将离线包导入火种节点

1. 将联网节点的 /data 文件拷贝至火种节点的 /root 目录下，在**联网节点**执行如下命令：

   ```
   scp -r data root@x.x.x.x:/root
   ```

   !!! note

   “x.x.x.x“ 为火种节点 IP 地址

2. 在**火种节点**上将 /data 文件内的镜像文件拷贝至火种节点内置的 docker resgitry 仓库。登录火种节点后执行如下命令：

   1. 进入镜像文件所在的目录

      ```
      cd data/v_offline_patch/amd64/images
      ```

2. 执行 import_images.sh 脚本将镜像导入火种节点内置的 docker resgitry 仓库。

   DEST_TLS_VERIFY=false ./import_images.sh 127.0.0.1:443

!!! note

上述命令仅仅适用于火种节点内置的 docker resgitry 仓库，如果使用外部仓库请使用如下命令：

```yaml
DEST_USER=${username} DEST_PASS=${password} DEST_TLS_VERIFY=false ./import_images.sh https://x.x.x.x:443
```

”https://x.x.x.x:443”为外部仓库的地址。
“DEST_USER=${username} DEST_PASS=${password}”外部仓库的用户名和密码参数。
如果外部仓库为免密，则可删除此参数。

3. 在火种节点上将/data 文件内的二进制文件拷贝至火种节点内置的 Minio 服务上。

   1. 进入二进制文件所在的目录

      cd data/v_offline_patch/amd64/files/

   2. 执行 import_files.sh 脚本将二进制文件导入火种节点内置的 Minio 服务上。

      MINIO_USER=rootuser MINIO_PASS=rootpass123 ./import_files.sh http://127.0.0.1:9000

!!! note

上述命令仅仅适用于火种节点内置的 Minio 服务，如果使用外部 Minio 请将“http://127.0.0.1:9000”替换为外部 Minio 的访问地址。
“rootuser”和“rootpass123”是火种节点内置的 Minio 服务的默认账户和密码。

更新 Global 集群的 kubernetes 版本清单

1. 将联网节点的 /data 文件内的 kubeanofflineversion.cr.patch 清单配置文件拷贝至 Global 集群内任一 **Master 节点**的 /root 目录下，请在**联网节点**执行如下命令：

   ```
   scp -r data/v_offline_patch/kubeanofflineversion.cr.patch.yaml root@x.x.x.x:/root
   ```

   !!! note

   “x.x.x.x“ 为 Global 集群内任一 Master 节点 IP 地址

2. 完成上一步后登录 Global 集群内任一 **Master 节点**执行清单配置文件，命令如下：

   ```
   kubectl apply -f kubeanofflineversion.cr.patch.yaml
   ```

使用平台 UI 升级工作集群的 kubernetes 版本

登录 DCE 5.0 的 UI 管理界面，参照集群升级文档对平台自建的工作集群进行升级。

为全局服务集群的工作节点扩容

本文将介绍离线模式下，如何手动为全局服务集群的工作节点进行扩容，默认情况下，不建议在平台部署后对全局服务集群进行扩容，请在平台部署前做好资源规划。

!!! note

注意：全局服务集群的控制节点不支持扩容。

前提条件

- 已经通过火种节点完成 DCE 平台的部署，并且火种节点上的 kind 集群运行正常。
- 必须使用平台 admin 权限的用户登陆。

获取火种节点上 kind 集群的 kubeconfig

1. 执行如下命令登陆火种节点：

   ssh root@火种节点 IP 地址

2. 在火种节点上执行如下命令获取 kind 集群的 CONTAINER ID：

   [root@localhost ~]# podman ps

   # 预期输出如下：

   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

   220d662b1b6a docker.m.daocloud.io/kindest/node:v1.26.2 2 weeks ago Up 2 weeks 0.0.0.0:443->30443/tcp, 0.0.0.0:8081->30081/tcp, 0.0.0.0:9000-9001->32000-32001/tcp, 0.0.0.0:36674->6443/tcp my-cluster-installer-control-plane

3. 执行如下命令，进入 kind 集群容器内：

   ```
   podman exec -it {CONTAINER ID} bash # {CONTAINER ID} 替换为您真实的容器 ID
   ```

4. 在 kind 集群容器内执行如下命令获取 kind 集群的 kubeconfig 配置信息：

   ```
   kubectl config view --minify --flatten --raw # {CONTAINER ID} 替换为您真实的容器 ID
   ```

   待控制台输出后，复制 kind 集群的 kubeconfig 配置信息，为下一步做准备。

在火种节点上 kind 集群内创建 cluster.kubean.io 资源

1. 使用 podman exec -it {CONTAINER ID} bash 命令进入 kind 集群容器内。

2. 复制并执行如下命令，在 kind 集群内执行，以创建 cluster.kubean.io 资源：

   ```
   kubectl apply -f - <<EOF
   apiVersion: kubean.io/v1alpha1
   kind: Cluster
   metadata:
   labels:
       clusterName: kpanda-global-cluster
   name: kpanda-global-cluster
   spec:
   hostsConfRef:
       name: my-cluster-hosts-conf
       namespace: kubean-system
   kubeconfRef:
       name: my-cluster-kubeconf
       namespace: kubean-system
   ```

```
  varsConfRef:
    name: my-cluster-vars-conf
    namespace: kubean-system
EOF
```

3. 在 kind 集群内执行如下命令，检验 cluster.kubean.io` 资源是否正常创建：

```
root@my-cluster-installer-control-plane:/# kubectl get clusters

# 预期输出如下
NAME                    AGE
kpanda-global-cluster   3s
my-cluster              16d
```

更新火种节点上的 kind 集群里的 containerd 配置

1. 执行如下命令，登陆全局服务集群的其中一个控制节点：

```
ssh root@全局服务集群控制节点 IP 地址
```

2. 在全局服务集群控制节点上执行如下命令，将控制节点的 containerd 配置文件 config.toml 复制到火种节点上：

```
scp /etc/containerd/config.toml root@{火种节点 IP}:/root
```

3. 在火种节点上执行如下命令，将 config.toml 配置文件复制到 kind 集群内：

```
cd /root
podman cp config.toml {CONTAINER ID}:/etc/containerd # {CONTAINER ID} 替换为您真实的容器 ID
```

4. 在 kind 集群内执行如下命令，重启 containerd 服务

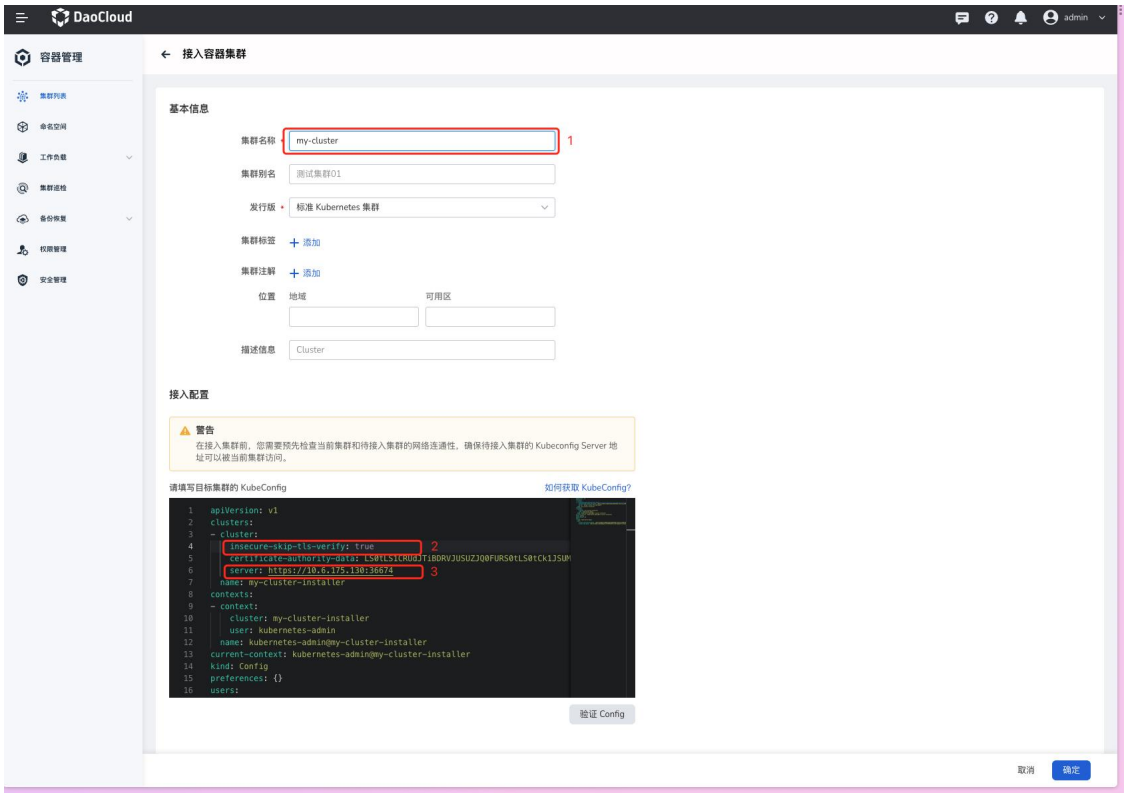
```
systemctl restart containerd
```

将 kind 集群接入 DCE 集群列表

1. 登陆 DCE 管理控制台，进入容器管理，在集群列表页右侧点击 接入集群按钮，进入接入集群页面。

2. 在接入配置处，填入并编辑刚刚复制的 kind 集群的 kubeconfig 配置。需要配置参数如下：

   - 集群名称：接入集群的名称，默认为 my-cluster。
   - insecure-skip-tls-verify: true：用以跳过 tls 验证，需要手动添加。
   - server：将默认的 https://my-cluster-installer-control-plane:6443 参数中的 IP 替换为火种节点的 IP； 6443 替换为 6443 端口在节点映射的端口。可执行 podman ps | grep 6443 命令查看。



img

3. 点击确认按钮，完成 kind 集群的接入。

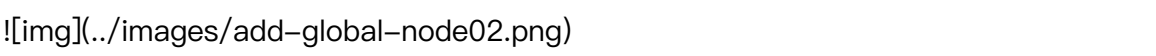
为全局服务集群添加标签

1. 登陆 DCE 管理控制台，进入容器管理，找到 kapnda-glabal-cluster 集群，在右侧操作列表找到基础配置操作按钮并进入基础配置界面。

2. 在基础配置页面，为全局服务集群添加的标签：kpanda.io/managed-by=my-cluster，如下图：

!!! note

注意：标签“kpanda.io/managed-by=my-cluster”中的 vaule 值为接入集群时指定的集群名称，默认为 "my-cluster"，具体依据您的实际情况。



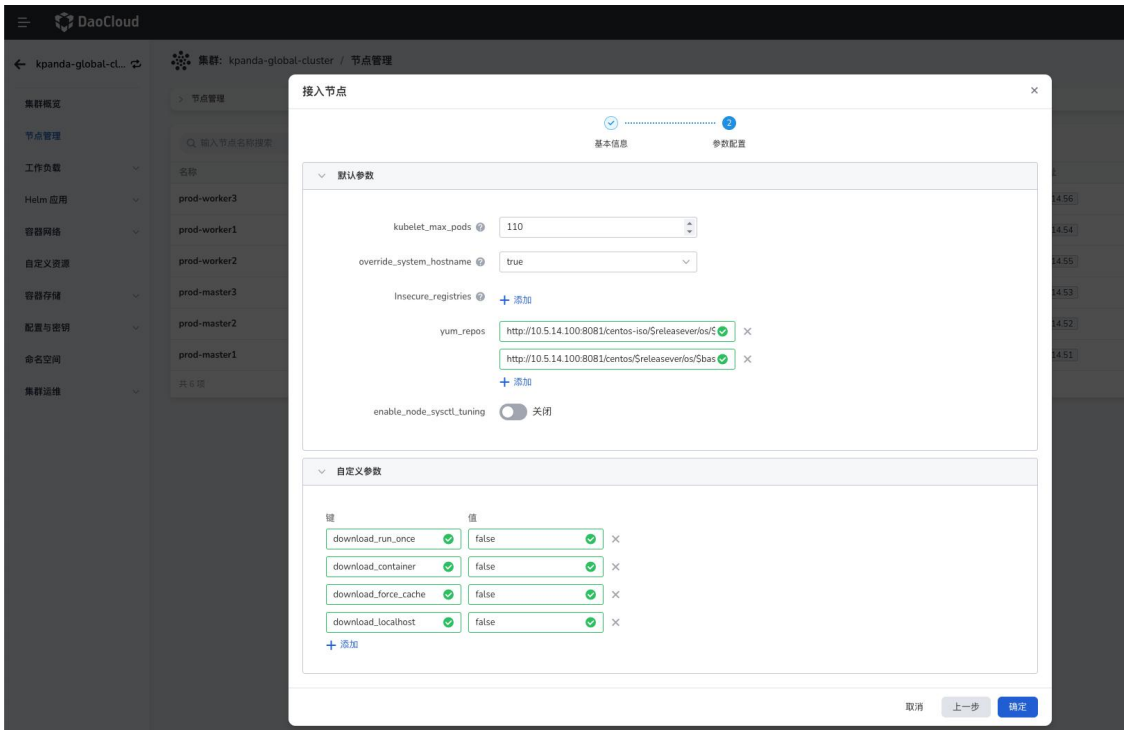
为全局服务集群添加节点

1. 进入全局服务集群节点列表页，在节点列表右侧找到接入节点按钮并点击进入节点配置页面。

2. 填入待接入节点的 IP 和认证信息。

3. 在自定义参数处添加如下自定义参数：

   download_run_once: false
   download_container: false
   download_force_cache: false
   download_localhost: false



img

4. 点击确认按钮，等待节点添加完成。

MySQL 应用及数据的跨集群备份恢复

本次演示将基于 DCE 5.0 的应用备份功能，实现一个有状态应用的跨集群备份迁移。

!!! note

当前操作者应具有 DCE 5.0 平台管理员的权限。

准备演示环境

准备两个集群

main-cluster 作为备份数据的源集群，recovery-cluster 集群作为需要恢复数据的目标集群。

集群	IP	节点
main-cluster	10.6.175.100	1 节点
recovery-cluster	10.6.175.110	1 节点

搭建 MinIO 配置

MinIO 服务器访问地址	存储桶	用户名	密码
http://10.7.209.110:9000	mysql-demo	root	dangerous

在两个集群部署 NFS 存储服务

!!! note

需要在 **源集群和目标集群** 上的所有节点上部署 NFS 存储服务。

1. 在两个集群中的所有节点安装 NFS 所需的依赖。

   ```
   yum install nfs-utils iscsi-initiator-utils nfs-utils iscsi-initiator-utils nfs-utils iscsi-initiator-utils -y
   ```

   预期输出

```
[root@g-master1 ~]# kubectl apply -f nfs.yaml
clusterrole.rbac.authorization.k8s.io/nfs-provisioner-runner created
clusterrolebinding.rbac.authorization.k8s.io/run-nfs-provisioner created
role.rbac.authorization.k8s.io/leader-locking-nfs-provisioner created
rolebinding.rbac.authorization.k8s.io/leader-locking-nfs-provisioner create
d
serviceaccount/nfs-provisioner created
service/nfs-provisioner created
deployment.apps/nfs-provisioner created
storageclass.storage.k8s.io/nfs created
```

2. 为 MySQL 应用准备 NFS 存储服务。

登录 main-cluster 集群和 recovery-cluster 集群的任一控制节点，使用 vi nfs.yaml 命令在节点上创建一个 名为 nfs.yaml 的文件，将下面的 YAML 内容复制到 nfs.yaml 文件。

nfs.yaml

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: nfs-provisioner-runner
namespace: nfs-system
rules:
- apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
- apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
```

```
    verbs: ["get", "list", "watch"]
- apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
- apiGroups: [""]
    resources: ["services", "endpoints"]
    verbs: ["get"]
- apiGroups: ["extensions"]
    resources: ["podsecuritypolicies"]
    resourceNames: ["nfs-provisioner"]
    verbs: ["use"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: run-nfs-provisioner
subjects:
- kind: ServiceAccount
    name: nfs-provisioner
    # replace with namespace where provisioner is deployed
    namespace: default
roleRef:
kind: ClusterRole
name: nfs-provisioner-runner
apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: leader-locking-nfs-provisioner
rules:
- apiGroups: [""]
```

```
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: leader-locking-nfs-provisioner
subjects:
- kind: ServiceAccount
    name: nfs-provisioner
    # replace with namespace where provisioner is deployed
    namespace: default
roleRef:
kind: Role
name: leader-locking-nfs-provisioner
apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: ServiceAccount
metadata:
name: nfs-provisioner
---
kind: Service
apiVersion: v1
metadata:
name: nfs-provisioner
labels:
    app: nfs-provisioner
spec:
ports:
    - name: nfs
    port: 2049
```

```
    - name: nfs-udp
      port: 2049
      protocol: UDP
    - name: nlockmgr
      port: 32803
    - name: nlockmgr-udp
      port: 32803
      protocol: UDP
    - name: mountd
      port: 20048
    - name: mountd-udp
      port: 20048
      protocol: UDP
    - name: rquotad
      port: 875
    - name: rquotad-udp
      port: 875
      protocol: UDP
    - name: rpcbind
      port: 111
    - name: rpcbind-udp
      port: 111
      protocol: UDP
    - name: statd
      port: 662
    - name: statd-udp
      port: 662
      protocol: UDP
  selector:
    app: nfs-provisioner
---
kind: Deployment
```

```yaml
apiVersion: apps/v1
metadata:
name: nfs-provisioner
spec:
selector:
    matchLabels:
    app: nfs-provisioner
replicas: 1
strategy:
    type: Recreate
template:
    metadata:
    labels:
        app: nfs-provisioner
    spec:
    serviceAccount: nfs-provisioner
    containers:
        - name: nfs-provisioner
        resources:
            limits:
            cpu: "1"
            memory: "4294967296"
        image: release.daocloud.io/velero/nfs-provisioner:v3.0.0
        ports:
            - name: nfs
            containerPort: 2049
            - name: nfs-udp
            containerPort: 2049
            protocol: UDP
            - name: nlockmgr
            containerPort: 32803
            - name: nlockmgr-udp
```

```
      containerPort: 32803
      protocol: UDP
    - name: mountd
      containerPort: 20048
    - name: mountd-udp
      containerPort: 20048
      protocol: UDP
    - name: rquotad
      containerPort: 875
    - name: rquotad-udp
      containerPort: 875
      protocol: UDP
    - name: rpcbind
      containerPort: 111
    - name: rpcbind-udp
      containerPort: 111
      protocol: UDP
    - name: statd
      containerPort: 662
    - name: statd-udp
      containerPort: 662
      protocol: UDP
  securityContext:
    capabilities:
      add:
        - DAC_READ_SEARCH
        - SYS_RESOURCE
  args:
    - "-provisioner=example.com/nfs"
  env:
    - name: POD_IP
      valueFrom:
```

```
            fieldRef:
              fieldPath: status.podIP
          - name: SERVICE_NAME
            value: nfs-provisioner
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        imagePullPolicy: "IfNotPresent"
        volumeMounts:
          - name: export-volume
            mountPath: /export
      volumes:
        - name: export-volume
          hostPath:
            path: /data
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: nfs
provisioner: example.com/nfs
mountOptions:
  - vers=4.1
```

3. 在两个集群的控制节点上执行 nfs.yaml 文件。

   kubectl apply -f nfs.yaml

4. 查看 NFS Pod 状态，等待其状态变为 running（大约需要 2 分钟）。

   kubectl get pod -n nfs-system -owide

   预期输出

```
[root@g-master1 ~]# kubectl get pod -owide
NAME                                READY   STATUS    RESTARTS   AGE     IP              NODE        NOMINATED NODE   READINESS GATES
nfs-provisioner-7dfb9bcc45-74ws2    1/1     Running   0          4m45s   10.6.175.100    g-master1   <none>           <none>
```

部署 MySQL 应用

1. 为 MySQL 应用准备基于 NFS 存储的 PVC，用来存储 MySQL 服务内的数据。

   使用 vi pvc.yaml 命令在节点上创建名为 pvc.yaml 的文件，将下面的 YAML 内容复制到 pvc.yaml 文件内。

   pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mydata
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: "1Gi"
  storageClassName: nfs
  volumeMode: Filesystem
```

2. 在节点上使用 kubectl 工具执行 pvc.yaml 文件。

   kubectl apply -f pvc.yaml

   预期输出

```
[root@g-master1 ~]# kubectl apply -f pvc.yaml
persistentvolumeclaim/mydata created
```

3. 部署 MySQL 应用。

使用 vi mysql.yaml 命令在节点上创建名为 mysql.yaml 的文件，将下面的 YAML 内容复制到 mysql.yaml 文件。

mysql.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mysql-deploy
  name: mysql-deploy
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: mysql-deploy
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
```

```
    app: mysql-deploy
  name: mysql-deploy
spec:
  containers:
  - args:
    - --ignore-db-dir=lost+found
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: dangerous
    image: release.daocloud.io/velero/mysql:5
    imagePullPolicy: IfNotPresent
    name: mysql-deploy
    ports:
    - containerPort: 3306
      protocol: TCP
    resources:
      limits:
        cpu: "1"
        memory: "4294967296"
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/lib/mysql
      name: data
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext:
    fsGroup: 999
  terminationGracePeriodSeconds: 30
  volumes:
  - name: data
```

```
      persistentVolumeClaim:
        claimName: mydata
```

4. 在节点上使用 kubectl 工具执行 mysql.yaml 文件。

   kubectl apply -f mysql.yaml

   预期输出

   ```
   [root@g-master1 ~]# kubectl apply -f mysql.yaml
   deployment.apps/mysql-deploy created
   ```

5. 查看 MySQL Pod 状态。

   执行 kubectl get pod | grep mysql 查看 MySQL Pod 状态，等待其状态变为 running（大约需要 2 分钟）。

   预期输出

   ```
   [root@g-master1 ~]# kubectl get pod |grep mysql
   mysql-deploy-5d6f94cb5c-gkrks         1/1       Running   0          2m53s
   ```

   !!! note

   - 如果 MySQL Pod 状态长期处于非 running 状态，通常是因为没有在集群的所有节点上安装 NFS 依赖。
   - 执行 `kubectl describe pod ${mysql pod 名称}` 查看 Pod 的详细信息。
   - 如果报错中有 `MountVolume.SetUp failed for volume "pvc-4ad70cc6-df37-4253-b0c9-8cb86518ccf8" : mount failed: exit status 32` 之类的信息，请分别执行 `kubectl delete -f nfs.yaml/pvc.yaml/mysql.yaml` 删除之前的资源后，重新从部署 NFS 服务开始。

6. 向 MySQL 应用写入数据。

   为了便于后期验证迁移数据是否成功，可以使用脚本向 MySQL 应用中写入测试数据。

1. 使用 vi insert.sh 命令在节点上创建名为 insert.sh 的脚本，将下面的 YAML 内容复制到该脚本。

insert.sh

```
#!/bin/bash

function rand(){
    min=$1
    max=$(($2-$min+1))
    num=$(date +%s%N)
    echo $(($num%$max+$min))
}

function insert(){
    user=$(date +%s%N | md5sum | cut -c 1-9)
    age=$(rand 1 100)

    sql="INSERT INTO test.users(user_name, age)VALUES('${user}', ${age});"
    echo -e ${sql}

    kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "${sql}"

}

kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "CREATE DATABASE IF NOT EXISTS test;"
kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "CREATE TABLE IF NOT EXISTS test.users(user_name VARCHAR(10) NOT NULL,age INT UNSIGNED)ENGINE=InnoDB DEFAULT CH
```

```
ARSET=utf8;"


 while true;do
      insert
      sleep 1
 done
```

2. 为 insert.sh 脚本添加权限并运行该脚本。

```
[root@g-master1 ~]# chmod +x insert.sh
[root@g-master1 ~]# ./insert.sh
```

预期输出

```
 mysql: [Warning] Using a password on the command line interface can be insecure.
 mysql: [Warning] Using a password on the command line interface can be insecure.
 INSERT INTO test.users(user_name, age)VALUES('dc09195ba', 10);
 mysql: [Warning] Using a password on the command line interface can be insecure.
 INSERT INTO test.users(user_name, age)VALUES('80ab6aa28', 70);
 mysql: [Warning] Using a password on the command line interface can be insecure.
 INSERT INTO test.users(user_name, age)VALUES('f488e3d46', 23);
 mysql: [Warning] Using a password on the command line interface can be insecure.
 INSERT INTO test.users(user_name, age)VALUES('e6098695c', 93);
 mysql: [Warning] Using a password on the command line interface can be insecure.
 INSERT INTO test.users(user_name, age)VALUES('eda563e7d', 63);
 mysql: [Warning] Using a password on the command line interface can be insecure.
```

INSERT INTO test.users(user_name, age)VALUES('a4d1b8d68', 17);
mysql: [Warning] Using a password on the command line interface can be insecure.

3. 在键盘上同时按下 control 和 c 暂停脚本的执行。

4. 前往 MySQL Pod 查看 MySQL 中写入的数据。

```
kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "SELECT * FROM test.users;"
```

预期输出

```
[root@g-master1 ~]# kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "SELECT * FROM test.users;"
mysql: [Warning] Using a password on the command line interface can be insecure.
user_name	age
dc09195ba	10
80ab6aa28	70
f488e3d46	23
e6098695c	93
eda563e7d	63
a4d1b8d68	17
ea47546d9	86
a34311f2e	47
740cefe17	33
ede85ea28	65
b6d0d6a0e	46
f0eb38e50	44
c9d2f28f5	72
8ddaafc6f	31
3ae078d0e	23
6e041631e	96
```

在两个集群安装 velero 插件

!!! note

需要在 **源集群和目标集群** 上均安装 velero 插件。

参考安装 velero 插件文档和下方的 MinIO 配置，在 main-cluster 集群和 recovery-cluster 集群上安装 velero 插件。

minio 服务器访问地址	存储桶	用户名	密码
http://10.7.209.110:9000	mysql-demo	root	dangerous

!!! note

安装插件时需要将 S3url 替换为此次演示准备的 MinIO 服务器访问地址，存储桶替换为 MinIO 中真实存在的存储桶。

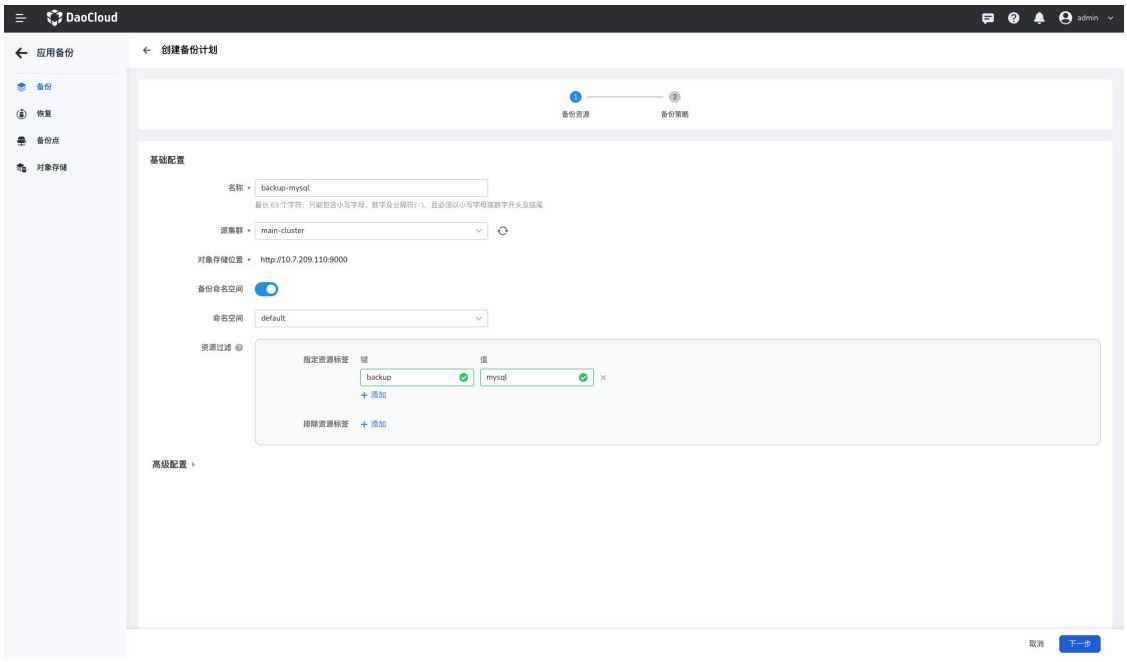
备份 MySQL 应用及数据

1. 为 MySQL 应用及 PVC 数据添加独有的标签：backup=mysql，便于备份时选择资源。

   kubectl label deploy mysql-deploy backup=mysql #为 `mysql-deploy` 负载添加标签
   kubectl label pod mysql-deploy-5d6f94cb5c-gkrks backup=mysql #为 mysql pod 添加标签
   kubectl label pvc mydata backup=mysql #为 mysql 的 pvc 添加标签

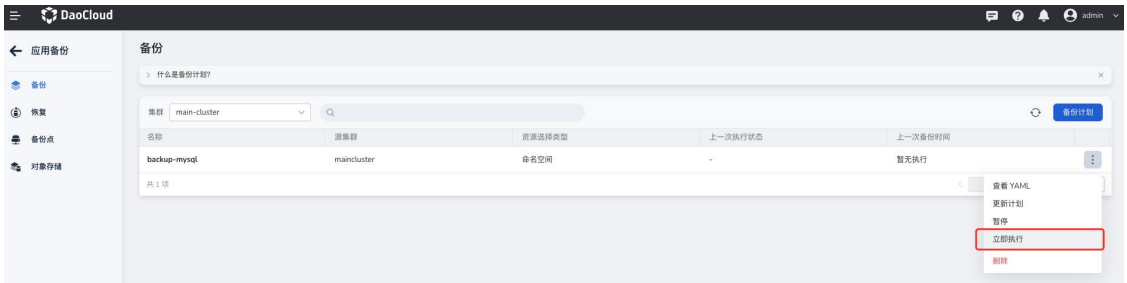
2. 参考应用备份中介绍的步骤，以及下方的参数创建应用备份。

   - 名称：backup-mysql（可以自定义）
   - 源集群： main-cluster
   - 命名空间：default
   - 资源过滤-指定资源标签：backup:mysql



img

3. 创建好备份计划之后页面会自动返回备份计划列表，找到新建的备份计划 backup-mysq，在计划点击更多操作按钮 ... 选择 立即执行 执行新建的备份计划。

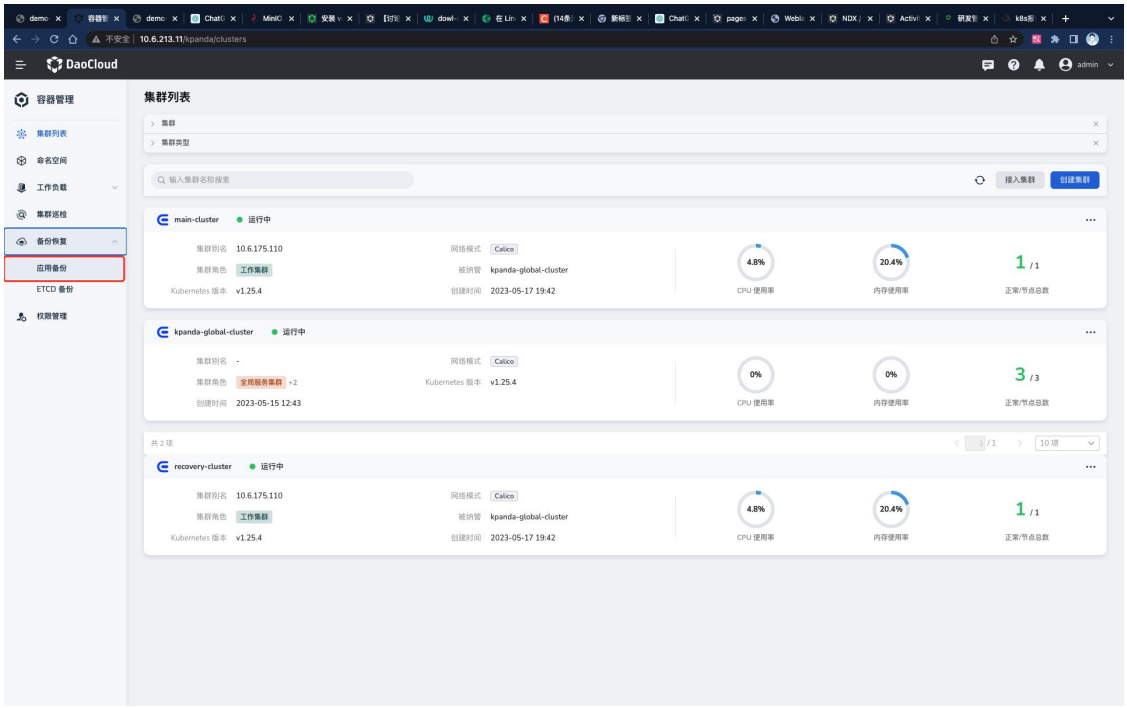


img

4. 等待备份计划执行完成后，即可执行后续操作。

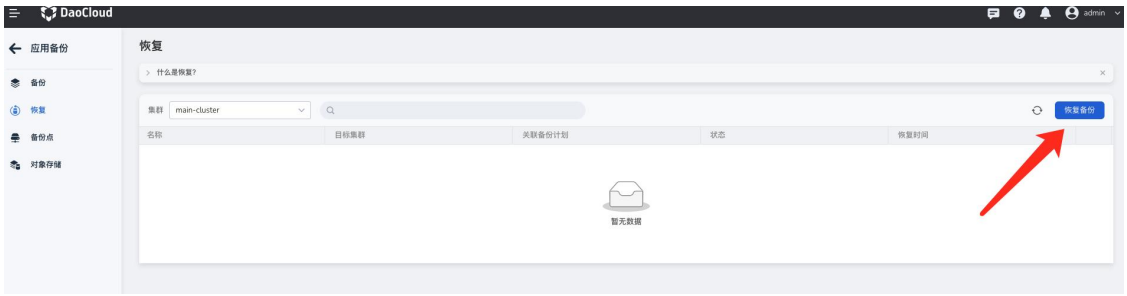
跨集群恢复 MySQL 应用及数据

1. 登录 DCE 5.0 平台，在左侧导航选择容器管理 -> 备份恢复 -> 应用备份。



img

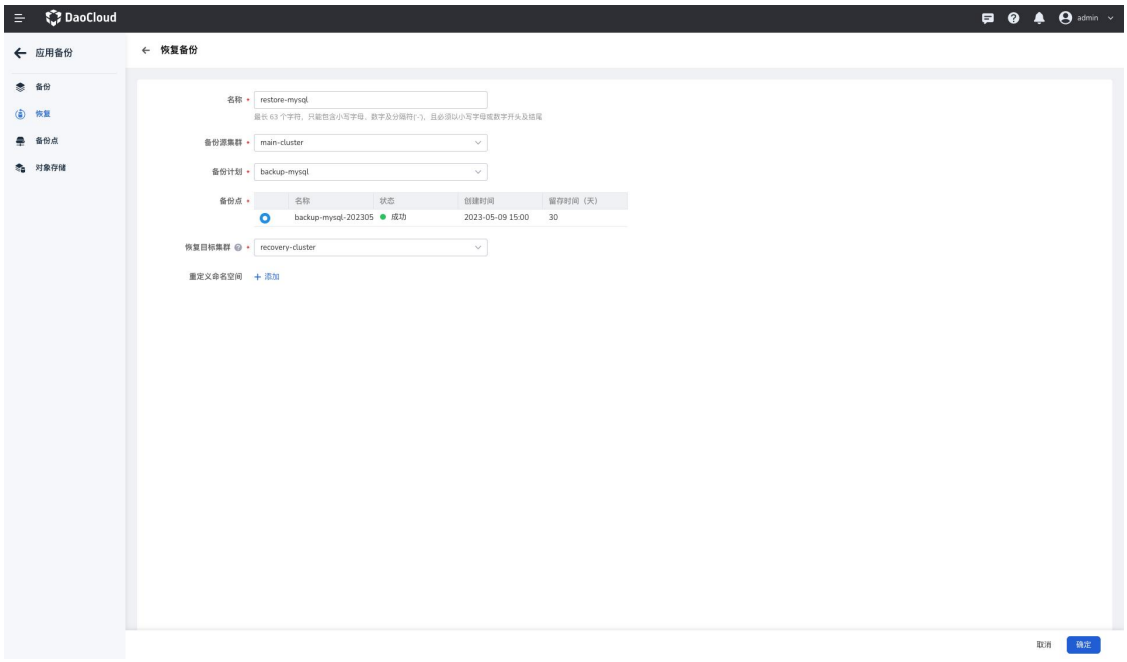
2. 在左侧功能栏选择 恢复，然后在右侧点击 恢复备份。



img

3. 查看以下说明填写参数：

   - 名称：restore-mysql（可以自定义）
   - 备份源集群：main-cluster
   - 备份计划：backup-mysql
   - 备份点：default
   - 恢复目标集群：recovery-cluster



img

4. 刷新备份计划列表，等待备份计划执行完成。

验证数据是否成功恢复

1. 登录 recovery-cluster 集群的控制节点，查看 mysql-deploy 负载是否已经成功备份到当前集群。

   kubectl get pod

   预期输出如下：

   ```
   NAME                                READY   STATUS    RESTARTS   AGE
   mysql-deploy-5798f5d4b8-62k6c       1/1     Running   0          24h
   ```

2. 检查 MySQL 数据表中的数据是否恢复成功。

   ```
   kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "SELECT * FROM test.users;"
   ```

预期输出如下： [root@g-master1 ~]# kubectl exec deploy/mysql-deploy -- mysql -uroot -pdangerous -e "SELECT * FROM test.users;" mysql: [Warning] Using a password on the command line interface can be insecure. user_name age dc09195ba 10 80ab6aa28 70 f488e3d46 23 e6098695c 93 eda563e7d 63 a4d1b8d68 17 ea47546d9 86 a34311f2e 47 740cefe17 33 ede85ea28 65 b6d0d6a0e 46 f0eb38e50 44 c9d2f28f5 72 8ddaafc6f 31 3ae078d0e 23 6e041631e 96

!!! success

可以看到，Pod 中的数据和 `main-cluster` 集群中 Pod 里面的数据一致。这说明已经成功地将 `main-cluster` 中的 MySQL 应用及其数据跨集群恢复到了 `recovery-cluster` 集群。

对工作集群的控制节点扩容

本文将以一个单控制节点的工作集群为例，介绍如何手动为工作集群的控制节点进行扩容，以实现自建工作集群的高可用。

!!! note

推荐在界面创建工作集群时即开启高可用模式，手动扩容工作集群的控制节点存在一定的操作风险，请谨慎操作

前提条件

- 已经通过 DCE 5.0 平台创建好一个工作集群，可参考文档创建工作集群。
- 工作集群的被纳管集群存在当前平台中，并且状态运行正常。

!!! note

被纳管集群：在界面创建集群时指定的用来管理当前集群，并为当前集群提供 kubernetes 版本升级、节点扩缩容、卸载、操作记录等能力的集群。

修改主机清单文件

1. 登录到容器管理平台，进入需要进行控制节点扩容的集群概览页面，在【基本信息】处，找到当前集群的 **被纳管集群**，点击被纳管集群的名称，进入被纳管集群的概览界面。

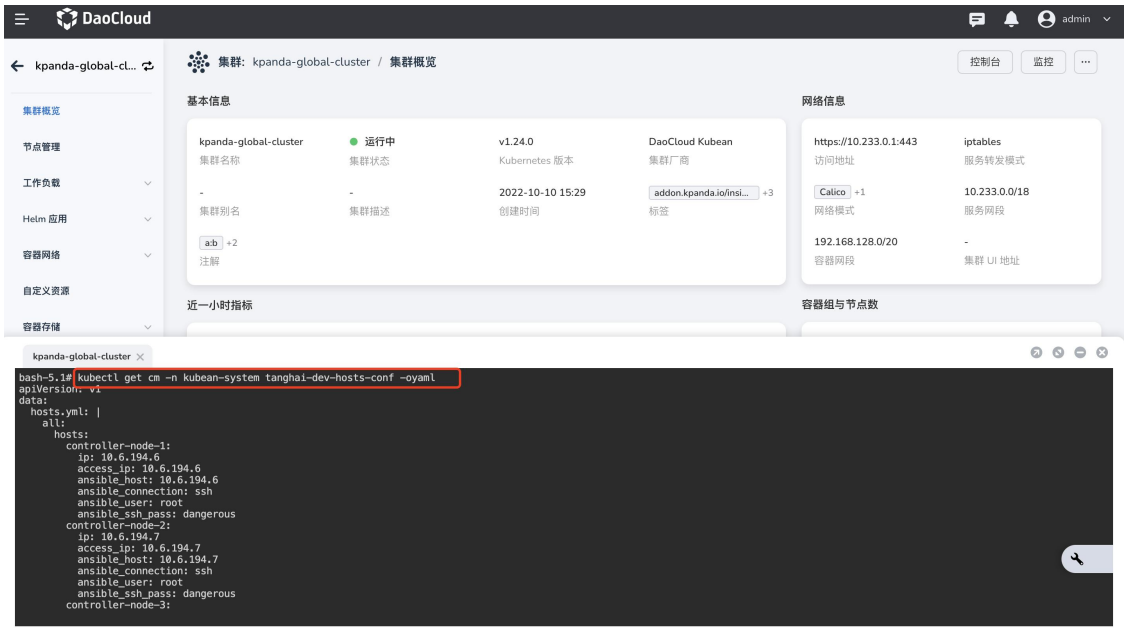


img

2. 在被纳管集群的概览界面，点击 **控制台**，打开云终端控制台，并执行如下命令，找到待扩容工作集群的主机清单文件。

   kubectl get cm -n kubean-system ${ClusterName}-hosts-conf -oyaml

   "${ClusterName}"：为待扩容工作集群的名称。



   img

3. 参考下方示例修改主机清单文件，新增控制节点信息。

   ===“修改前”

   ``` yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: tanghai-dev-hosts-conf
     namespace: kubean-system
   data:
     hosts.yml: |
       all:
   ```

```
    hosts:
      node1:
        ip: 10.6.175.10
        access_ip: 10.6.175.10
        ansible_host: 10.6.175.10
        ansible_connection: ssh
        ansible_user: root
        ansible_password: password01
    children:
      kube_control_plane:
        hosts:
          node1:
      kube_node:
        hosts:
          node1:
      etcd:
        hosts:
          node1:
      k8s_cluster:
        children:
          kube_control_plane:
          kube_node:
      calico_rr:
        hosts: {}
.....
```

===“修改后”

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: tanghai-dev-hosts-conf
  namespace: kubean-system
data:
  hosts.yml: |
    all:
      hosts:
        node1:
          ip: 10.6.175.10
          access_ip: 10.6.175.10
          ansible_host: 10.6.175.10
          ansible_connection: ssh
          ansible_user: root
          ansible_password: password01
        node2: # 新增控制节点 node2 内容
          ip: 10.6.175.20
          access_ip: 10.6.175.20
          ansible_host: 10.6.175.20
          ansible_connection: ssh
          ansible_user: root
          ansible_password: password01
        node3:
          ip: 10.6.175.30
          access_ip: 10.6.175.30
          ansible_host: 10.6.175.30
          ansible_connection: ssh
          ansible_user: root
          ansible_password: password01
      children:
        kube_control_plane:
          hosts:
            node1:
            node2: # 新增控制节点 node2 内容
```

```
        node3:  #  新增控制节点  node3  内容
    kube_node:
      hosts:
        node1:
        node2:  #  新增控制节点  node2  内容
        node3:  #  新增控制节点  node3  内容
    etcd:
      hosts:
        node1:
        node2:  #  新增控制节点  node2  内容
        node3:  #  新增控制节点  node3  内容
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
      hosts:  {}
```

重要参数：

- all.hosts.node1：原集群中已存在的主节点
- all.hosts.node2、all.hosts.node3：集群扩容待新增的控制节点
- all.children.kube_control_plane.hosts：集群中的控制节点组
- all.children.kube_node.hosts：集群中的工作节点组
- all.children.etcd.hosts：集群中的 ETCD 节点组

新增 ClusterOperation.yml 扩容任务

使用基于下面的 ClusterOperation.yml 模板，新增一个集群控制节点扩容任务 scale-master-node-ops.yaml。

yaml title="ClusterOperation.yml" apiVersion: kubean.io/v1alpha1 kind: ClusterOperation metadata: name: cluster1-online-install-ops spec: cluster:

${cluster-name} # 指定 cluster name image: ghcr.m.daocloud.io/kubean-io/spray-job:v0.4.6 # 指定 kubean 任务运行的镜像 backoffLimit: 0 actionType: playbook action: cluster.yml extraArgs: --limit=etcd,kube_control_plane -e ignore_assert_errors=yes preHook: - actionType: playbook action: ping.yml - actionType: playbook action: disable-firewalld.yml postHook: - actionType: playbook action: upgrade-cluster.yml extraArgs: --limit=etcd,kube_control_plane -e ignore_assert_errors=yes - actionType: playbook action: kubeconfig.yml - actionType: playbook action: cluster-info.yml

!!! note

- spec.image 镜像地址要与之前执行部署时的 job 其内镜像保持一致
- spec.action 设置为 scale.yml
- spec.extraArgs 设置为 --limit=g-worker
- pec.preHook 中的 enable-repo.yml 剧本参数，要填写相关 OS 的正确的 repo_list
- 如果一次性添加 Master（etcd）节点超过（包含）三个，需追加额外参数 -e etcd_retries=10 以增大 etcd node join 重试次数。

按照上述的配置，创建并部署 scale-master-node-ops.yaml。

```
# 复制上述清单文件

vi cale-master-node-ops.yaml

kubectl apply -f cale-master-node-ops.yaml -n kubean-system
```

执行完上述步骤，执行如下命令进行验证：

```
kubectl get node
```

Application contact emails

a@daocloud.io, b@daocloud.io

Project Summary

Spiderpool: An IP Address Management (IPAM) CNI plugin of Kubernetes for managing static IP addresses in underlay networks

Project Description

Why Spiderpool? Despite the increasing demand for IP Address Management (IPAM) solutions, there has yet to be a comprehensive, user-friendly, and intelligent open-source solution that meets the needs of underlay networks. That's where Spiderpool comes in - to eliminate the complexity of allocating IP addresses to underlay networks.

With the goal of making IP allocation as simple as some overlay-network CNI, Spiderpool offers a variety of features such as static application IP addresses, dynamic scalability of IP addresses, multi-NIC, and dual-stack support.

Overall, Spiderpool aims to simplify IPAM operations and make them more efficient for underlay networks. By providing a user-friendly interface and intelligent allocation features, it will allow network administrators to allocate and manage IP addresses with ease.

We believe that Spiderpool has the potential to be a game-changer for open-source enthusiasts who are looking for a comprehensive and reliable IPAM solution. With its advanced functionality and simplicity in mind, we are confident that Spiderpool will be a valuable addition to any underlay network.

Spiderpool can work well with any CNI project that is compatible with third-party IPAM plugins, such as:

- macvlan CNI
- vlan CNI
- ipvlan CNI
- SR-IOV CNI
- ovs CNI
- Multus CNI
- calico CNI
- weave CNI

Features

Spiderpool provides many exciting features to support:

- Multiple subnet objects

  The administrator can create multiple subnet objects that map to each underlay CIDR. This allows applications to be assigned IP addresses within different subnets, which helps to accommodate the complex planning of underlay networks. See an example for more details.

- Automatical ippool for applications that require static IP addresses

  In order to assign static IP addresses, some open source projects require hardcoding them in the application's annotations. However, this approach is prone to operational errors and manual conflicts with IP addresses, resulting in higher IP management costs when scaling applications.

  By contrast, Spiderpool can automatically create, delete, scale up and down a dedicated IP pool with static IP addresses for individual

applications, thereby minimizing the effort required for operations. This method also helps to manage IP addresses more efficiently, especially when applications need frequent scaling. By automating the process, it eliminates the need for manual intervention and reduces the overall cost of IP management.

  - For stateless applications, the IP address range can be automatically fixed and IP resources can be dynamically scaled according to the number of application replicas. See an example for more details.

  - For stateful applications, IP addresses can be automatically fixed for each pod, and the overall IP scaling range can be fixed as well. Also, IP resources can be dynamically scaled according to the number of application replicas. See an example for more details.

  - The dedicated IP pool can include some redundant IP addresses, which enable applications to perform a rolling update while creating new pods.

  - Support for third-party application controllers. See an example for details.

- Manual IP pooling is available for applications that require static IP addresses but where the administrator needs to manually specify the IP address. See an example for details.

- For applications not requiring static IP addresses, they can share an IP pool. See an example for details

- For an application with pods running on nodes accessing different underlay subnets, Spiderpool can assign IP addresses within different subnets. See an example for details.
- Multiple IP pools can be set for a pod for the usage of backup IP resources. See an example for details
- Set global reserved IPs that will not be assigned to pods, it can avoid to misuse IP addresses already used by other network hosts. See an example for details.
- Spiderpool can specify different subnets for each NIC when assigning multiple NICs to a pod with Multus CNI. See an example for details.
- IP pools can be shared by whole cluster or bound to a specific namespace. See an example for details.
- Spiderpool also provides an additional plugin veth to:
  - help some CNI addons to access clusterIP and perform pod-health check, such as macvlan CNI, vlan CNI, ipvlan CNI, SR-IOV CNI, ovs CNI. See an example for details.
  - help coordinate routes for each NIC in pods that have multiple NICs assigned by Multus. See an example for details.
- Private IPv4 addresses are scarce, but Spiderpool provides a reasonable IP recycling mechanism, especially for running new pods when nodes or old pods are abnormal. See an example for details.
- An administrator can specify customized route with Spiderpool. See an example for details.

- Efficient performance in assigning and releasing Pod IPs ensures smooth application release and disaster recovery for the cluster. See an example for details.
- All above features can work in ipv4-only, ipv6-only, and dual-stack scenarios. See example for details.

Org repo URL

https://github.com/spidernet-io

Project repo URL

https://github.com/spidernet-io/spiderpool

Additional repos

No response

Website URL

https://spidernet-io.github.io/spiderpool/

Roadmap

https://github.com/spidernet-io/spiderpool/blob/main/docs/develop/roadmap.md

Roadmap context

We are actively working on a longer-term roadmap which should be available in the next 30 days. For now we have a near-term roadmap.

Contributing Guide

https://github.com/spidernet-io/spiderpool/blob/main/docs/develop/contributing.md

Code of Conduct (CoC)

https://github.com/spidernet-io/spiderpool/blob/main/docs/develop/CODE-OF-CONDUCT.md

Maintainers file

https://github.com/spidernet-io/spiderpool/blob/main/MAINTAINERS.md

Why CNCF?

The Cloud Native Computing Foundation (CNCF) is the vendor-neutral, open-source hub of cloud-native computing that has become an essential project in promoting cloud-native technology. The Spiderpool community fully supports CNCF's vision.

- Being an ecosystem project of CNCF can increase awareness of the Spiderpool project and Spiderpool can enable more CNI solutions to benefit more people with underlying network needs.
- With the guidance of CNCF, Spiderpool can adhere to its requirements and set goals in the right direction.

Benefit to the Landscape

We believe that our community will contribute significantly to the advancements in CNI add-ons.

- Currently, there is no pure IPAM CNI project in the landscape. Spiderpool could help many of the main CNI addons to manage IP addresses, especially for CNIs without IPAM capabilities, such as vlan, ipvlan, SR-IOV, ovs, macvlan,
- In the CNI open-source community, there are few projects that help solve underlay network requirements. The IPAM needs for underlay networks are more complex than those for overlay networks. For example, pods may require static IP addresses for firewall reasons. Additionally, private IPv4 addresses are scarce, and Spiderpool must release IP addresses taken by abnormal pods to make them available for new pods. Therefore, Spiderpool is unique in that all its features are primarily designed for underlay networks. Its design is based on best practices in production environments.
- Before the Spiderpool project was launched, we had developed a similar private IPAM project. The idea and design of Spiderpool were based on our extensive experience in production environments. Spiderpool has been available for approximately one year, and it has proven to be both practical and stable.

Cloud Native 'Fit'

Spiderpool adheres to the CNI Specification and assigns IP addresses to the main CNI addons. Using a custom resource definition (CRD) and operator design, all requests and information are saved in the CR instance. This allows for effective management of IP address allocation, ensuring that addresses are used efficiently and conflicts are avoided.

Cloud Native 'Integration'

Spiderpool can assign IP addresses to any main CNI addon that can use third-party IPAM addons. It is specifically designed for underlay network use and works well in cooperation with CNI addons like macvlan, calico BGP mode, weave, vlan, ipvlan, SR-IOV, ovs, Multus.

Spiderpool offers extensive usage documentation that demonstrates how to solve underlay network problems by integrating with various CNI addons.

Spiderpool's documentation offers clear guidance on best practices for managing IP address allocation in complex container network environments. Users can learn how to set up Spiderpool with different CNI plugins and configure it to meet their specific needs, ensuring that they can effectively manage their container networks regardless of the container orchestration platform they are using.

Cloud Native Overlap

At this time, we do not believe that there is significant overlap between Spiderpool and other CNCF projects. However, we do address some of the same areas that other projects are investigating in the space of configuration management, scalable Kubernetes, and alternative object storage (e.g., Kine).

Spiderpool's focus on IP address allocation and management complements other container networking projects and can be used in conjunction with them to improve overall container network efficiency and reliability. By providing a flexible and scalable IPAM solution, Spiderpool helps to address common challenges in managing container networks at scale.

Similar projects

Whereabouts is an IPAM project, but not a landscape project. Spiderpool differs from Whereabouts in several ways:

- Spiderpool can automatically assign static IP addresses to applications, whereas Whereabouts cannot.
- For stateful applications, Spiderpool can automatically fix IP addresses for each pod, while Whereabouts does not support this feature.
- When an application has pods running on nodes accessing different underlay subnets, Spiderpool can assign IP addresses within different subnets, but Whereabouts does not have this capability.
- Spiderpool has subnet and ippool CRD storage, while Whereabouts does not.
- Spiderpool supports assigning IP addresses in different subnets for multi-NIC cases, but Whereabouts does not.
- Spiderpool supports customized pod routing, while Whereabouts does not.

Overall, while both projects are focused on IP address management, Spiderpool offers several features that distinguish it from Whereabouts and make it a useful tool for managing container networks in complex environments.

Product or Service to Project separation

N/A

可观测性

任务和实例

在 Insight 里，可以从中抓取采样值的端点称为实例（Instance），为了性能扩展而复制出来的多个这样的实例形成了一个任务（Job）。例如，下面的 api-server 任务有四个相同的实例：

job: *api-server*
instance 1: *1.2.3.4:5670*
instance 2: *1.2.3.4:5671*
instance 3: *5.6.7.8:5670*
istance 4: *5.6.7.8:5671*

Insight 抓取完采样值后，会自动给采样值添加下面的标签和值：

- job: 抓取所属任务
- instance: 抓取来源实例

另外每次抓取时，Insight 还会自动在以下时序里插入采样值：

- up{job="[job-name]", instance="instance-id"}：采样值为 1 表示实例健康，否则为不健康
- scrape_duration_seconds{job="[job-name]", instance="[instance-id]"}：采样值为本次抓取消耗时间
- scrape_samples_post_metric_relabeling{job="<job-name>", instance="<instance-id>"}：采样值为重新打标签后的采样值个数
- scrape_samples_scraped{job="<job-name>", instance="<instance-id>"}：采样值为本次抓取到的采样值个数

查询语言 PromQL

Insight 内置了一个强大的数据查询语言 PromQL。 Insight 是一个时间序列构成的数据库，所有保存在 Insight 里的数据都是按时间戳和值的序列顺序存放的，称为 Vector（向量）），因为是 NoSQL，相比关系型数据库 MySQL 能很好地支持大量数据写入。 从最新测试结果看，在硬件资源充足地情况下，Insight 单实例每秒可采集数十万条。每一次数据采集得到一个 Sample（样本），其由三部分组成：

- Metrics（指标）：包含了 Metrics name 以及 Labels。
- Timestamp（时间戳）：当前采样的时间，精确到毫秒。
- Value（采样值）：其类型为 float64 浮点数。

通过 PromQL 可以实现对监控数据的查询、聚合。同时 PromQL 也被应用于数据可视化（通过 Grafana）以及告警当中。通过 PromQL 可以轻松回答以下问题：

- 在过去一段时间中 95% 应用延迟时间的分布范围
- 预测在 4 小时后，磁盘空间占用大致会是什么情况
- CPU 占用率前 5 位的服务有哪些

数据模型

Insight 基本上将所有数据存储为时间序列。 为同一指标和同一组标签维度的值加上时间戳形成数据流。除了存储的时间序列，Insight 还可以为查询生成临时的时间序列结果。

- 指标名称和标签

  每个时间序列数据通过**指标名称**和**标签**键值对进行唯一的标识。

- 指标名称

指定了监测系统的常规功能（例如 http_requests_total– 收到的 HTTP 请求总数）。指标名称可以包含 ASCII 字母和数字以及下划线和英文冒号。它必须与正则表达式[a–zA–Z_:][a–zA–Z0–9_:]*匹配。

!!! note

英文冒号保留用于用户自定义的规则。不能用于 exporter 或 direct instrumentation。

Insight 采用标签实现多维度数据模型。相同指标名称的任何给定标签组合可以标识特定维度的指标（例如：所有使用 POST 方法到 /api/tracks 句柄的 HTTP 请求）。 这种查询语言允许根据这些维度进行过滤和聚合。更改任何标签值（包括添加或删除标签）将创建新的时间序列数据。

标签名称可以包含 ASCII 字母、数字以及下划线。必须与正则表达式 [a–zA–Z\u][a–zA–Z0–9\u]* 匹配。以 __ 开头的标签名称保留供内部使用。

标签值可以包含任何 Unicode 字符。

标签值为空被视为等同于标签不存在。

- 样本

  样本构成实际的时间序列数据。每个样品包括：float64 值和毫秒精度的时间戳

- 标记法

  给定指标名称和一组标签，时间序列数据通常使用以下标记法进行标识：

  <metric name>{<label name>=<label value>, ...}

  例如，指标名称为 api_http_requests_total 且标签为 method="POST" 和 handler="/messages" 的时间序列数据可以写为：

  api_http_requests_total{method="POST", handler="/messages"}

这与 OpenTSDB 使用的标记法相同。

指标类型

Insight 的度量指标有以下几种类型。

- 计数器（Counter）

  计数器是一种累计型的度量指标，它是一个**只能递增**的数值。计数器主要用于统计类似于服务请求数、任务完成数和错误出现次数这样的数据。

- 计量器（Gauge）

  计量器是一个**既可增又可减**的度量指标值。计量器主要用于测量类似于温度、内存使用量这样的瞬时数据。

- 直方图（Histogram）

  直方图对观察结果（通常是请求持续时间或者响应大小这样的数据）进行采样，并在可配置的桶中对其进行统计。有以下几种方式来产生直方图（假设度量指标为 <basename>）：

  - 按桶计数，相当于 <basename>_bucket{le="<upper inclusive bound>"}
  - 采样值总和，相当于<basename>_sum
  - 采样值总数，相当于 <basename>_count ，也等同于把所有采样值放到一个桶里来计数 <basename>_bucket{le="+Inf"}

  Histogram 可以理解为柱状图，典型的应用如：请求持续时间，响应大小。可以对观察结果采样，分组及统计。

- 汇总（Summary）

  类似于直方图，汇总也对观察结果进行采样。除了可以统计采样值总和和总数，它还能够按分位数统计。有以下几种方式来产生汇总（假设度量指标为 <basename>）：

- 按分位数，也就是采样值小于该分位数的个数占总数的比例小于 ϕ，相当于 <basename>{quantile="<ϕ>"}
- 采样值总和，相当于 <basename>_sum
- 采样值总数，相当于 <basename>_count

流量治理

服务网格的流量路由规则可以让您很容易的控制服务之间的流量和 API 调用。服务网格简化了服务级别属性的配置，比如熔断器、超时和重试，并且能轻松的设置重要的任务。 它还提供了开箱即用的故障恢复特性，有助于增强应用的健壮性，从而更好地应对被依赖的服务或网络发生故障的情况。

服务网格的流量管理模型源于和服务一起部署的 Envoy 代理。 网格内服务发送和接收的所有流量（data plane 流量）都经由 Envoy 代理，这让控制网格内的流量变得异常简单，而且不需要对服务做任何的更改。

为了在网格中导流，服务网格需要知道所有的 endpoint 在哪和属于哪个服务。为了定位到 service registry (服务注册中心)，服务网格会连接到一个服务发现系统。 例如，如果您在 Kubernetes 集群上安装了服务网格，那么它将自动检测该集群中的服务和 endpoint。

使用此服务注册中心，Envoy 代理可以将流量定向到相关服务。 大多数基于微服务的应用程序，每个服务的工作负载都有多个实例来处理流量，称为负载均衡池。默认情况下，Envoy 代理基于轮询调度模型在服务的负载均衡池内分发流量，按顺序将请求发送给池中每个成员，一旦所有服务实例均接收过一次请求后，重新回到第一个池成员。

服务网格基本的服务发现和负载均衡能力为您提供了一个可用的服务网格，但它能做到的远比这多的多。 在许多情况下，您可能希望对网格的流量情况进行更细粒度的控制。 作为 A/B 测试的一部分，您可能想将特定百分比的流量定向到新版本的服务，或者为特定的服务实例子集应用不同的负载均衡策略。 您可能还想对进出网格的流量应用特殊的规则，或者将网格的外部依赖项添加到服务注册中心。通过使用服务网格的流量管理 API 将流量配置添加到服务网格，就可以完成这些工作，甚至还可以拓展更多的能力。

和其他服务网格配置一样，这些 API 也使用 Kubernetes 的自定义资源定义（CRDs）来声明，您可以使用 YAML 进行配置。

流量治理策略

- 限流

  限流是流量限速（Rate Limit）的简称，指只允许指定的流量进入系统，超过阈值的流量将被拒绝服务、排队或等待、降级等。 限流是为了保障机器和整体业务的稳定性，避免系统雪崩。

- 降级

  当服务器压力剧增时，根据实际业务情况及流量，对非核心、非关键的服务有策略地不处理或简化处理，从而释放部分服务器资源以保证核心服务正常运作或高效运作。 当整个微服务架构整体的负载超出了预设上限或即将到来的流量预计会超过设定的阈值时，可以做降级处理。

- 熔断

  熔断这一概念来源于电子工程中的断路器（Circuit Breaker）。 在互联网系统中，当下游服务因访问压力过大而响应变慢或失败，超过设定的阈值时，上游服务会暂时切断对下游服务的调用，从而保护系统整体的可用性。这种牺牲局部，保全整体的措施就叫做熔断。

虚拟服务

此处介绍有关服务网格的虚拟服务。

虚拟服务（Virtual Service）和目标规则（Destination Rule）是服务网格流量路由功能的关键拼图。 虚拟服务让您配置如何在服务网格内将请求路由到服务，这基于服务网格和平台提供的基本的连通性和服务发现能力。 每个虚拟服务包含一组路由规则，服务网格按顺序评估它们，服务网格将每个给定的请求匹配到虚拟服务指定的实际目标地址。 您的网格可以有多个虚拟服务，也可以没有，取决于您的使用场景。

为什么使用虚拟服务？

虚拟服务在增强服务网格流量管理的灵活性和有效性方面，发挥着至关重要的作用，通过对客户端请求的目标地址与真实响应请求的目标工作负载进行解耦来实现。 虚拟服务同时提供了丰富的方式，为发送至这些工作负载的流量指定不同的路由规则。

为什么这如此有用？ 如果没有虚拟服务，Envoy 会在所有的服务实例中使用轮询的负载均衡策略分发请求。 您可以用对工作负载的了解来改善这种行为。例如，有些可能代表不同的版本。 这在 A/B 测试中可能有用，您可能希望在其中配置基于不同服务版本的流量百分比路由，或指引从内部用户到特定实例集的流量。

使用虚拟服务，您可以为一个或多个主机名指定流量行为。 在虚拟服务中使用路由规则，告诉 Envoy 如何发送虚拟服务的流量到适当的目标。 路由目标地址可以是同一服务的不同版本，也可以是完全不同的服务。

虚拟服务可以让您：

- 通过单个虚拟服务处理多个应用程序服务。

如果您的网格使用 Kubernetes，可以配置一个虚拟服务处理特定命名空间中的所有服务。 映射单一的虚拟服务到多个“真实”服务特别有用，可以在不需要客户适应转换的情况下，将单体应用转换为微服务构建的复合应用系统。 您的路由规则可以指定为“对这些 monolith.com 的 URI 调用转到 microservice A”等等。 您可以在下面的一个示例看到它是如何工作的。

- 和网关整合并配置流量规则来控制出入流量。

  在某些情况下，您还需要配置目标规则来使用这些特性，因为这是指定服务子集的地方。 在一个单独的对象中指定服务子集和其它特定目标策略，有利于在虚拟服务之间更简洁地重用这些规则。

虚拟服务示例

下面的虚拟服务根据请求是否来自特定的用户，把它们路由到服务的不同版本。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
```

```
      host: reviews
      subset: v3
```

hosts 字段

使用 hosts 字段列举虚拟服务的主机——即用户指定的目标或是路由规则设定的目标。 这是客户端向服务发送请求时使用的一个或多个地址。

```
hosts:
- reviews
```

虚拟服务主机名可以是 IP 地址、DNS 名称，或者依赖于平台的一个简称（例如 Kubernetes 服务的短名称），隐式或显式地指向一个完全限定域名（FQDN）。 您也可以使用通配符（“*”）前缀，让您创建一组匹配所有服务的路由规则。 虚拟服务的 hosts 字段实际上不必是服务网格服务注册的一部分，它只是虚拟的目标地址。这让您可以为没有路由到网格内部的虚拟主机建模。

路由规则

在 http 字段包含了虚拟服务的路由规则，用来描述匹配条件和路由行为。 它们把 HTTP/1.1、HTTP2 和 gRPC 等流量发送到 hosts 字段指定的目标（您也可以用 tcp 和 tls 片段为 TCP 和未终止的 TLS 流量设置路由规则）。 一个路由规则包含了指定的请求要流向哪个目标地址，具有 0 或多个匹配条件，取决于您的使用场景。

匹配条件

示例中的第一个路由规则有一个条件，因此以 match 字段开始。 在本例中，您希望此路由应用于来自 ”jason“ 用户的所有请求，所以使用 headers、end-user 和 exact 字段选择适当的请求。

```
- match:
    - headers:
        end-user:
          exact: jason
```

Destination

route 部分的 destination 字段指定了符合此条件的流量的实际目标地址。 与虚拟服务的 hosts 不同，destination 的 host 必须是存在于服务网格服务注册中心的实际目标地址，否则 Envoy 不知道该将请求发送到哪里。 可以是一个有代理的服务网格，或者是一个通过服务入口被添加进来的非网格服务。 本示例运行在 Kubernetes 环境中，host 名为一个 Kubernetes 服务名：

```
route:
- destination:
    host: reviews
    subset: v2
```

请注意，在该示例和本页其它示例中，为了简单，我们使用 Kubernetes 的短名称设置 destination 的 host。 在评估此规则时，服务网格会添加一个基于虚拟服务命名空间的域后缀，这个虚拟服务包含要获取主机的完全限定名的路由规则。在我们的示例中使用短名称也意味着您可以复制并在任何喜欢的命名空间中尝试它们。

只有在目标主机和虚拟服务位于相同的 Kubernetes 命名空间时才可以使用这样的短名称。 因为使用 Kubernetes 的短名称容易导致配置出错，我们建议您在生产环境中指定完全限定的主机名。

destination 片段还指定了 Kubernetes 服务的子集，将符合此规则条件的请求转入其中。 在本例中子集名称是 v2。您可以在目标规则章节中看到如何定义服务子集。

路由规则优先级

路由规则 按从上到下的顺序选择，虚拟服务中定义的第一条规则有最高优先级。本示例中，不满足第一个路由规则的流量均流向一个默认的目标，该目标在第二条规则中指定。 因此，第二条规则没有 match 条件，直接将流量导向 v3 子集。

```
- route:
  - destination:
```

```
      host: reviews
      subset: v3
```

我们建议提供一个默认的“无条件”或基于权重的规则（见下文）作为每一个虚拟服务的最后一条规则，如案例所示，从而确保流经虚拟服务的流量至少能够匹配一条路由规则。

路由规则的更多内容

正如上面所看到的，路由规则是将特定流量子集路由到指定目标地址的强大工具。您可以在流量端口、header 字段、URI 等内容上设置匹配条件。 例如，这个虚拟服务让用户发送请求到两个独立的服务：ratings 和 reviews，就好像它们是 http://bookinfo.com/ 这个更大的虚拟服务的一部分。 虚拟服务规则根据请求的 URI 和指向适当服务的请求匹配流量。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
    - bookinfo.com
  http:
  - match:
    - uri:
        prefix: /reviews
    route:
    - destination:
        host: reviews
  - match:
    - uri:
        prefix: /ratings
    route:
```

```
    - destination:
        host: ratings
...

  http:
  - match:
      sourceLabels:
        app: reviews
    route:
...
```

有些匹配条件可以使用精确的值，如前缀或正则。

您可以使用 AND 向同一个 match 块添加多个匹配条件，或者使用 OR 向同一个规则添加多个 match 块。 对于任何给定的虚拟服务也可以有多个路由规则。这可以在单个虚拟服务中使路由条件变得随您所愿的复杂或简单。

您也可以使用路由规则在流量上执行一些操作，例如：

- 添加或删除 header
- 重写 URL
- 为调用这一目标地址的请求设置重试策略

云原生网关

网关是介于客户端和服务器端之间的中间层，所有的外部请求都需要经过网关层。而云原生网关指的是基于云原生可声明式 API 概念发展而来的、与业务解耦的网关，例如 Zuul、Kong、Nginx、Spring Cloud Gateway、Envoy 等。 云原生网关最显著的特点是，可以通过声明的方式定义网关的运行时配置，可以通过控制面的声明自动生成配置。 云原生网关的功能和微服务网关基本类似，例如身份验证、路由、监控、负载均衡、缓存、服务升降级、静态响应处理、流量控制、日志、重试、熔断等。

- Sentinel

  Sentinel 是面向分布式服务架构的流量控制组件，主要以流量为切入点，从流量控制、熔断降级、系统自适应保护等多个维度来保障微服务的稳定性。 Sentinel 具有丰富的应用场景，承接了阿里巴巴近 10 年的双十一大促流量的核心场景。Sentinel 具有完备的实时监控，用户可以在控制台中看到接入应用的单台机器的秒级数据。 Sentinel 提供与其它开源框架 / 库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合，可以开箱即用。 此外，Sentinel 还能提供简单易用、完善的 SPI 扩展接口，用户可以通过实现扩展接口来快速地定制逻辑。

- Envoy

  Envoy 是面向服务架构的高性能网络代理，拥有强大的定制化能力。Envoy 是面向服务架构设计的 L7 代理和通信总线，核心是一个 L3/L4 网络代理。 Envoy 通常以 Sidecar 的方式运行在应用程序的周边，也可以作为网络的边缘代理来运行。 Envoy 的特性包括：进程外体系结构、L3/L4 过滤器体系结构、HTTP L7 过滤器体系结构、一流的 HTTP/2 支持、 HTTP/3 支持、HTTP L7 路由、gRPC 支持、服务发现和动态配置、健康检查、高级负载平衡、前端/边缘代理支持、一流的可观察性等。

- Contour

  Contour 将 Envoy 部署为反向代理和负载均衡器，从而充当 Kubernetes 的入口控制器。 Contour 支持动态配置更新，而且还引入了一个新的入口 API (HTTPProxy)。 该 API 通过自定义资源定义（CRD）实现，目标是扩展 Ingress API 的功能，提供更丰富的用户体验并解决原始设计中的缺陷。 Contour 架构灵活，可以部署为 Kubernetes Deployment 或 Daemonset。此外，Contour 还具有 TLS 证书授权特性，管理员可以安全地委派通配符证书访问。

微服务

微服务可以指一种通过多个小型服务组合来构建单个应用的架构风格，也可以指构成该应用的各个小型服务，即一些协同工作、小而自治的服务。 微服务架构具有技术异构、故障隔离、敏捷开发、独立部署，弹性扩缩、代码可复用等优势，因此越来越多的企业开始采用微服务架构开发应用。 但是，随着业务的不断扩展，业务逻辑越来越复杂，微服务的数量呈现出爆发式增长的态势。错综复杂的服务间调用关系、依赖关系影响着整个应用的稳定性和可用性，大大增加了运维的成本。因此，微服务治理工具应运而生。

- 传统微服务

  传统微服务指基于传统的微服务注册中心发展而来的微服务治理模式和技术体系。 传统的微服务框架主要有 Spring Cloud、Dubbo 等，传统的注册中心主要有 Eureka、Nacos、Zookeeper、Consul、etcd 等。

- 云原生微服务

  云原生微服务指基于 Kubernetes 发展而来的跨技术栈微服务治理模式，自无框架以来逐步向服务网格发展。 这种模式使用云原生的框架（如 Kubernetes、Istio、Linkered），云原生的注册中心（如 Kubernetes 注册中心和 Mesh 注册中心）和云原生的网关（如 Nginx、Envoy、Contour）。

- 微服务框架

  微服务框架可以分为侵入式和非侵入式两种。侵入式框架以 Spring Cloud 和 Dubbo 为代表，需要对旧代码进行改造。 非侵入式框架以 Istio 和 Linkerd 为代表，在原有基础上加入边车和服务网格，无需更改旧代码。

- Spring Cloud 框架

Spring Cloud 是一款基于 Spring Boot 的微服务框架。 Spring Cloud 将市面上成熟的、经过验证的微服务框架整合起来，在 Spring Boot 的基础上进行再封装，屏蔽复杂的配置和实现原理， 最终为开发人员提供了一套简单易懂、易部署和易维护的分布式系统开发工具包。 Spring Cloud 包括 Spring Cloud Config、Spring Cloud Bus 等 20 多个子项目，提供了服务治理、服务网关、智能路由、负载均衡、熔断器、监控跟踪、分布式消息队列、配置管理等领域的解决方案。

- Dubbo 框架

  Apache Dubbo 微服务开发框架提供了 RPC 通信与微服务治理两大关键能力。 这意味着，使用 Dubbo 开发的微服务将具备相互之间的远程发现与通信能力。 同时，利用 Dubbo 丰富的服务治理能力，可以实现诸如服务发现、负载均衡、流量调度等服务治理诉求。 Dubbo 是高度可扩展的，用户几乎可以在任意功能点去定制自己的实现，从而改变框架的默认行为以满足自己的业务需求。 Dubbo 目前支持 Consul、Nacos、ZooKeeper、Redis 等多种开源组件作为注册中心。

注册中心

注册中心相当于微服务架构中的”通讯录“，负责记录服务和服务地址的映射关系。 微服务在启动时，将自己的网络地址等信息注册到注册中心，注册中心存储这些数据。 服务消费者从注册中心查询服务提供者的地址，并通过该地址调用服务提供者的接口。 各个微服务与注册中心使用一定机制通信。如果注册中心与某微服务长时间无法通信，就会注销该实例。如果微服务网络地址发生变化时，会重新注册到注册中心。

传统的服务注册与发现

服务注册是指，微服务启动时通过集成的服务发现 SDK 向注册中心发送注册信息，注册中心在收到服务注册请求后将该服务的基本信息存储下来。 服务发现是指，在服务注册之后，如有服务消费者要调用该服务，调用方可以通过服务发现组件从注册中心查询目标微服务的地址列表，并通过获取到的服务地址列表，以某种负载策略向目标微服务发起调用。 传统的服务注册与发现主要由传统的注册中心提供支持，例如 Nacos、Eureka、Zookeeper 等。

- Eureka 注册中心

  Eureka 是 Spring Cloud 官方推荐使用的注册中心。 Eureka 主要适用于通过 Java 实现的分布式系统，或是与 JVM 兼容语言构建的系统。 但是，由于 Eureka 服务端的服务治理机制提供了完备的 RESTful API，所以也支持将非 Java 语言构建的微服务应用纳入 Eureka 的服务治理体系中来。 只是在使用其他语言平台的时候，需要自己来实现 Eureka 的客户端程序。 Eureka 集群采用非 Master/Slave 架构，集群中所有节点角色一致，数据写入集群任意一个节点后，再由该节点向集群内其他节点进行复制实现弱一致性同步。

- Nacos 注册中心

Nacos 注册中心是阿里巴巴推出的开源项目，可以轻松构建云原生应用程序和微服务平台。 Nacos 致力于发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，主要包括服务发现和服务健康监测、动态配置服务、动态 DNS 服务、服务及其元数据管理。 Nacos 支持几乎所有类型的服务，例如，Dubbo/gRPC 服务、Spring Cloud RESTful 服务或 Kubernetes 服务等。

- Zookeeper 注册中心

  Apache ZooKeeper 是开放源码的分布式应用程序协调组件，为分布式应用提供一致性服务的软件。 Zookeeper 提供的功能包括配置维护、域名服务、分布式同步、组服务等。 在微服务项目开发中，ZooKeeper 经常和 Dubbo 配合使用，充当服务注册中心。

云原生的服务注册与发现

- Kubernetes 注册中心

  在 Kubernetes 中能够为微服务提供内部服务注册发现的基础就是 Service 类型的资源。 Kubernetes 使用 DNS 作为服务注册表。 为了满足这一需要，每个 Kubernetes 集群都会在 kube-system 命名空间中用 Pod 的形式运行一个 DNS 服务（kube-dns/coredns），通常称之为集群 DNS。 服务注册过程是：

  1. 向 API Server 用 POST 方式提交一个新的 Service 定义。
  2. 该请求需要经过认证、鉴权以及其他的准入策略检查过程之后才会放行。
  3. Service 得到一个 ClusterIP（虚拟 IP 地址），并保存到集群数据仓库。
  4. 在集群范围内传播 Service 配置。
  5. 集群 DNS 服务得知该 Service 的创建，据此创建必要的 DNS 记录。 Service 对象注册到集群 DNS 之中后，就可以通过一个单一稳定的 IP 地址访问该 Service 中的 Pod。

Service Mesh 注册中心

Service Mesh 作为微服务的注册中心是基于 Kubernetes 实现的。 在 Kubernetes 集群环境中安装 Istio 后，Kubernetes 在创建 Pod 时， 就会通过 Kube-API 服务器调用控制面组件的 Sidecar-Injector 服务、自动修改应用程序的描述信息并将其注入 Sidecar， 之后在创建业务容器的 Pod 中同时创建 Sidecar 代理容器。SideCar 则会通过 xDS 协议与 Istio 控制面各组件连接。

登录无限循环，报错 401 或 403

出现这个问题原因为：ghippo-keycloak 连接的 Mysql 数据库出现故障，导致 OIDC Public keys 被重置

在全局管理 0.11.1 及以上版本，您可以参照以下步骤，使用 helm 更新全局管理配置文件即可恢复正常。

```
# 更新 helm 仓库
helm repo update ghippo

# 备份 ghippo 参数
helm get values ghippo -n ghippo-system -o yaml > ghippo-values-bak.yaml

# 获取当前部署的 ghippo 版本号
version=$(helm get notes ghippo -n ghippo-system | grep "Chart Version" | awk -F ': ' '{ print $2 }')

# 执行更新操作，使配置文件生效
helm upgrade ghippo ghippo/ghippo \
-n ghippo-system \
-f ./ghippo-values-bak.yaml \
--version ${version}
```

keycloak 无法启动

故障表现

MySQL 已就绪，无报错。在安装全局管理后 keycloak 无法启动（> 10 次）。

```
11:20:15,015 ERROR [org.keycloak.connections.jpa.updater.liquibase.LiquibaseJpaUpdaterProvider] (ServerServ
ice Thread Pool -- 68) Error has occurred while updating the database: liquibase.exception.MigrationFailedE
xception: Migration failed for change set META-INF/jpa-changelog-1.6.1.xml::1.6.1_from15::mposolda@redhat.c
om:
     Reason: liquibase.exception.DatabaseException: Error executing SQL ALTER TABLE keycloak.OFFLINE_USER_S
ESSION ADD PRIMARY KEY (USER_SESSION_ID, OFFLINE_FLAG): IJ031040: Connection is not associated with a manag
ed connection: org.jboss.jca.adapters.jdbc.jdk8.WrappedConnectionJDK8@1a4e6637
```

img

检查项

- 如果数据库是 MySQL，检查 keycloak database 编码是否是 UTF8。
- 检查从 keycloak 到数据库的网络，检查数据库资源是否充足，包括但不限于资源限制、存储空间、物理机资源。

解决步骤

```
09:39:24,207 ERROR [org.keycloak.connections.jpa.updater.liquibase.conn.DefaultLiquibaseConnectionProvider] (ServerService T
hread Pool -- 59) Change Set META-INF/jpa-changelog-4.0.0.xml::4.0.0-KEYCLOAK-5579-fixed::mposolda@redhat.com failed.  Error
: Table 'keycloak.CLIENT_TEMPLATE_ATTRIBUTES' doesn't exist [Failed SQL: ALTER TABLE keycloak.CLIENT_TEMPLATE_ATTRIBUTES DRO
P FOREIGN KEY FK_CL_TEMPL_ATTR_TEMPL]
09:39:24,208 ERROR [org.keycloak.connections.jpa.updater.liquibase.LiquibaseJpaUpdaterProvider] (ServerService Thread Pool -
- 59) Error has occurred while updating the database: liquibase.exception.MigrationFailedException: Migration failed for cha
nge set META-INF/jpa-changelog-4.0.0.xml::4.0.0-KEYCLOAK-5579-fixed::mposolda@redhat.com:
     Reason: liquibase.exception.DatabaseException: Table 'keycloak.CLIENT_TEMPLATE_ATTRIBUTES' doesn't exist [Failed SQL: A
LTER TABLE keycloak.CLIENT_TEMPLATE_ATTRIBUTES DROP FOREIGN KEY FK_CL_TEMPL_ATTR_TEMPL]
        at org.liquibase//liquibase.changelog.ChangeSet.execute(ChangeSet.java:619)
        at org.liquibase//liquibase.changelog.visitor.UpdateVisitor.visit(UpdateVisitor.java:51)
        at org.liquibase//liquibase.changelog.ChangeLogIterator.run(ChangeLogIterator.java:79)
        at org.liquibase//liquibase.Liquibase.update(Liquibase.java:214)
        at org.liquibase//liquibase.Liquibase.update(Liquibase.java:192)
        at org.liquibase//liquibase.Liquibase.update(Liquibase.java:188)
        at org.keycloak.keycloak-model-jpa@16.1.0//org.keycloak.connections.jpa.updater.liquibase.LiquibaseJpaUpdaterProvide
```

img

1. 检查 MySQL 资源占用是否到达 limit 限制
2. 检查 MySQL 中 database keycloak table 的数量是不是 92
3. 删除 keycloak database 并创建，提示 CREATE DATABASE IF NOT EXISTS keycloak CHARACTER SET utf8
4. 重启 keycloak Pod 解决问题

重启集群（虚拟机）istio-ingressgateway 无法启动？

报错提示如下图：



```
2022-09-06T10:08:21.729574Z     warning envoy config    gRPC config for type.googleapis.com/envoy.config.listener.v3.Listener rejected: Error adding/updating listener(s) 0.0.0.0_8080: Provider 'origins-0' in jwt_authn config has invalid local jwks: Jwks RSA [n] or [e] field is missing or has a parse error

2022-09-06T10:08:22.691699Z     warn    Envoy proxy is NOT ready: config received from XDS server, but was rejected: cds updates: 1 successful, 0 rejected; lds updates: 0 successful, 1 rejected
2022-09-06T10:08:24.684404Z     warn    Envoy proxy is NOT ready: config received from XDS server, but was rejected: cds updates: 1 successful, 0 rejected; lds updates: 0 successful, 1 rejected
2022-09-06T10:08:26.684665Z     warn    Envoy proxy is NOT ready: config received from XDS server, but was rejected: cds updates: 1 successful, 0 rejected; lds updates: 0 successful, 1 rejected
2022-09-06T10:08:28.684001Z     warn    Envoy proxy is NOT ready: config received from XDS server, but was rejected: cds updates: 1 successful, 0 rejected; lds updates: 0 successful, 1 rejected
2022-09-06T10:08:30.684783Z     warn    Envoy proxy is NOT ready: config received from XDS server, but was rejected: cds updates: 1 successful, 0 rejected
```

可能原因：RequestAuthentication CR 的 jwtsUri 地址无法访问， 导致 istiod 无法下发配置给 istio-ingressgateway（Istio 1.15 可以规避这个 bug：https://github.com/istio/istio/pull/39341/）

解决方法：

1. 备份 RequestAuthentication ghippo CR。

   kubectl get RequestAuthentication ghippo -n istio-system -o yaml > ghippo-ra.yaml

2. 删除 RequestAuthentication ghippo CR。

   kubectl delete RequestAuthentication ghippo -n istio-system

3. 重启 Istio。

   kubectl rollout restart deploy/istiod -n istio-system
   kubectl rollout restart deploy/istio-ingressgateway -n istio-system

4. 重新 apply RequestAuthentication ghippo CR。

   kubectl apply -f ghippo-ra.yaml

   !!! note

apply RequestAuthentication ghippo CR 之前，请确保 ghippo-apiserver 和 ghippo-keycloak 已经正常启动。

网格 Istio 版本升级

!!! warning “更新提示”

更新时尽量不要进行配置等相关变更，请合理安排升级计划。

服务网格支持三种网格类型，其中托管网格与专有网格由 DaoCloud 创建并进行托管，接入网格指接入客户已有的服务网格。

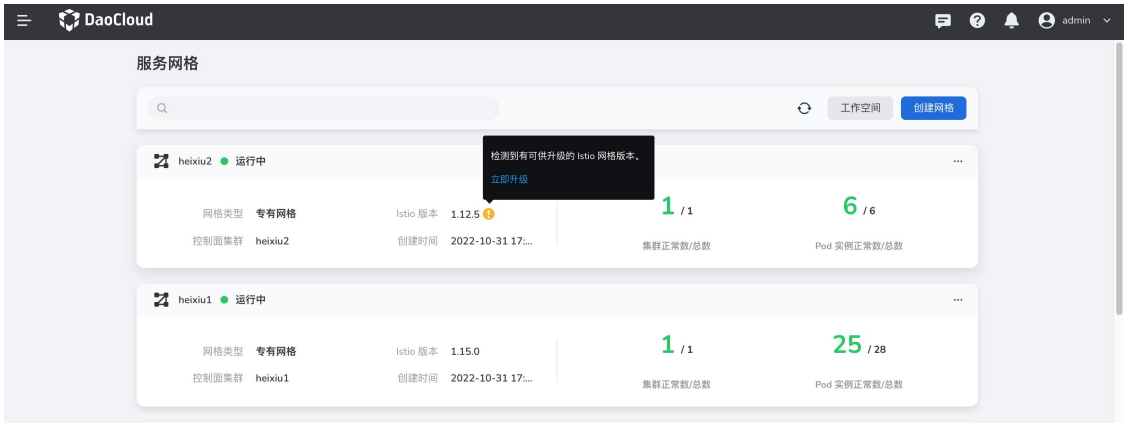
一键升级：托管网格与专有网格

DaoCloud 托管的网格类型，支持 Istio 版本可持续升级，DaoCloud 提供了原生版本和定制版本两类升级版本。

- 原生版本：社区原生 Istio，无任何定制化修改。
- 定制版本：基于 Istio 做了部分功能定制（后缀为：-mspider），例如集成 Merbridge 提高网格通信性能，支持 SpringCloud、Dubbo 等传统微服务的智能识别，支持边车热升级能力等。

这两类版本升级过程相同，但不支持不同类型的混合升级；所以，在创建网格实例时务必确认所需的版本。

DaoCloud 会持续提供 Istio 新版本适配工作，当系统检测存在新的 Istio 版本时，网格列表 会对可升级的网格实例进行提示（卡片会出现叹号提示图标），查看图标内容并点击立即升级按钮即可进入升级向导。



立即升级

Istio 升级向导包含选择目标版本、环境检测、执行升级三步，升级完成后网格即可立即上线运行。具体步骤如下：

1. **选择目标版本**：在列表中选择期望升级的版本，升级后将无法回滚至低版本，建议谨慎选择。



目标版本

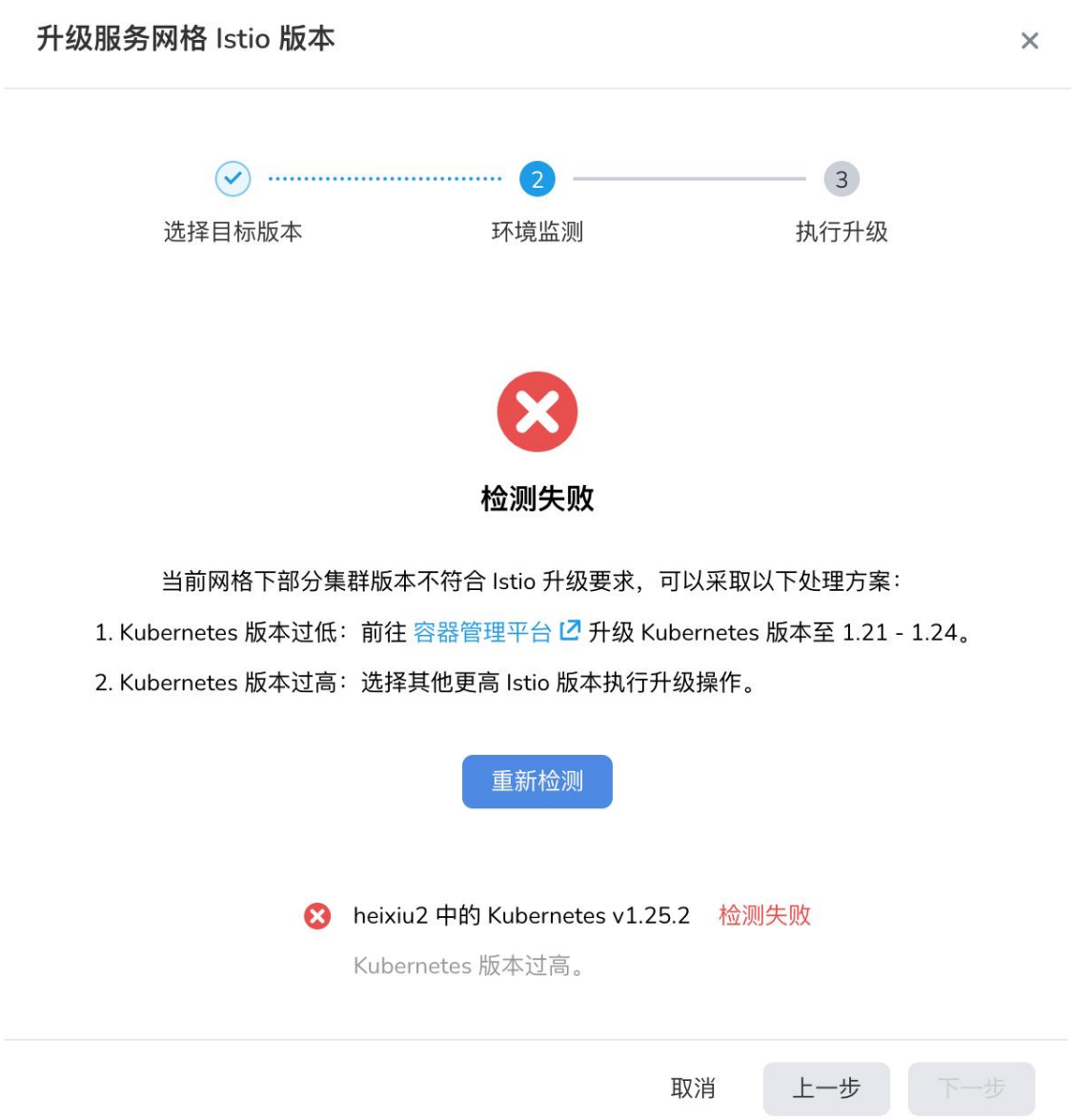
官方是不推荐的 Istio 对跨版本升级，建议按照阶梯升级的方式升级，例如 1.15.x 应该升级到 1.16.x， 不建议直接升级到大于 1.16.x 的版本。

2. **环境监测**：系统将基于所选的目标版本，检测网格下各集群（k8s）版本是否符合升级要求，如果符合要求，将激活下一步按钮，否则，需要用户处理环境问题.



环境检测

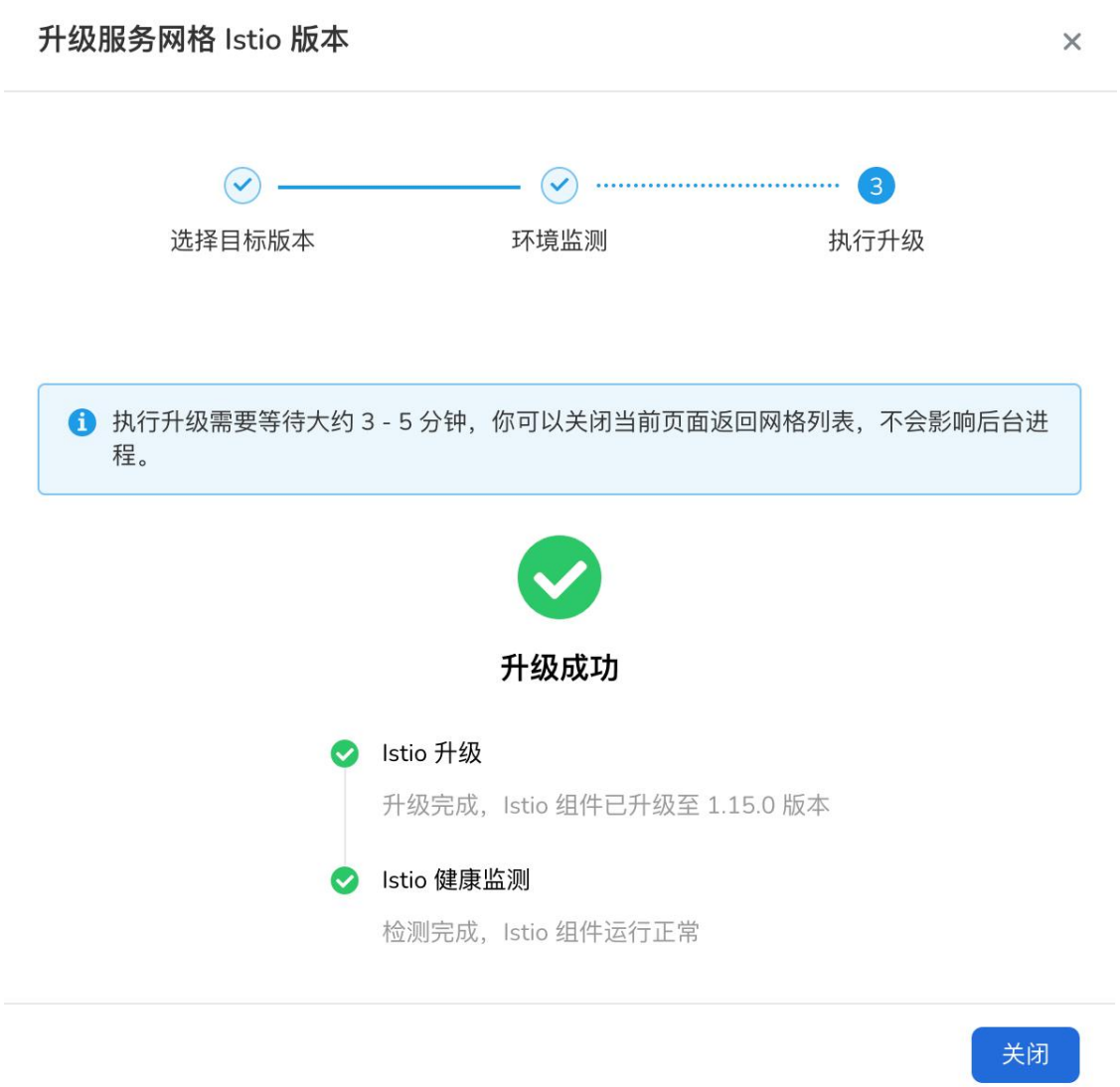
- 如果集群（k8s）版本过低，可以在容器管理平台先升级集群（k8s）版本后，点击重新检测按钮；
- 如果集群（k8s）版本过高，建议回退至“选择目标版本”选择其他更高版本的 Istio。



环境检测

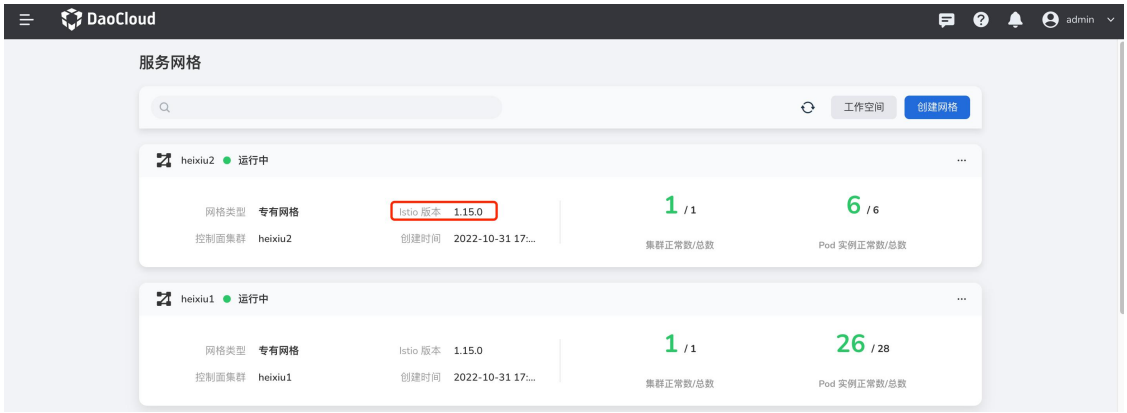
3. **执行升级**：环境检测通过后，将进入升级阶段，该过程包含升级和健康检测两阶段。

   - Istio 升级：Istio 镜像拉取和控制面组件升级
   - Istio 健康检测：Istio 控制面组件运行状态检查



执行升级

升级完成后，回到网格列表页面，可以看到网格的 Istio 版本已变更。



执行升级

!!! note

– 升级过程一旦开始将无法终止，升级期间建议不要对网格执行任何设置操作。
– 更直观的操作演示，可参阅[视频教程](../../videos/mspider.md)。

手工升级：接入网格

由于接入网格是由用户自行托管，其网格部署形式无法确定；所以需要用户自行进行，本文给出 Istio 官方推荐的几种升级建议，请根据实际部署方案操作。

- **金丝雀升级** 通过先运行一个金丝雀部署的新控制平面升级 Istio。
- **原地升级** 原地升级和回退。
- **使用 Helm 升级** 使用 Helm 升级 Istio 的说明。

!!! warning “注意事项”

尚未正式测试和推行一步升级多个版本（如 1.6.x 到 1.8.x），强烈建议通过阶梯升级的方式。

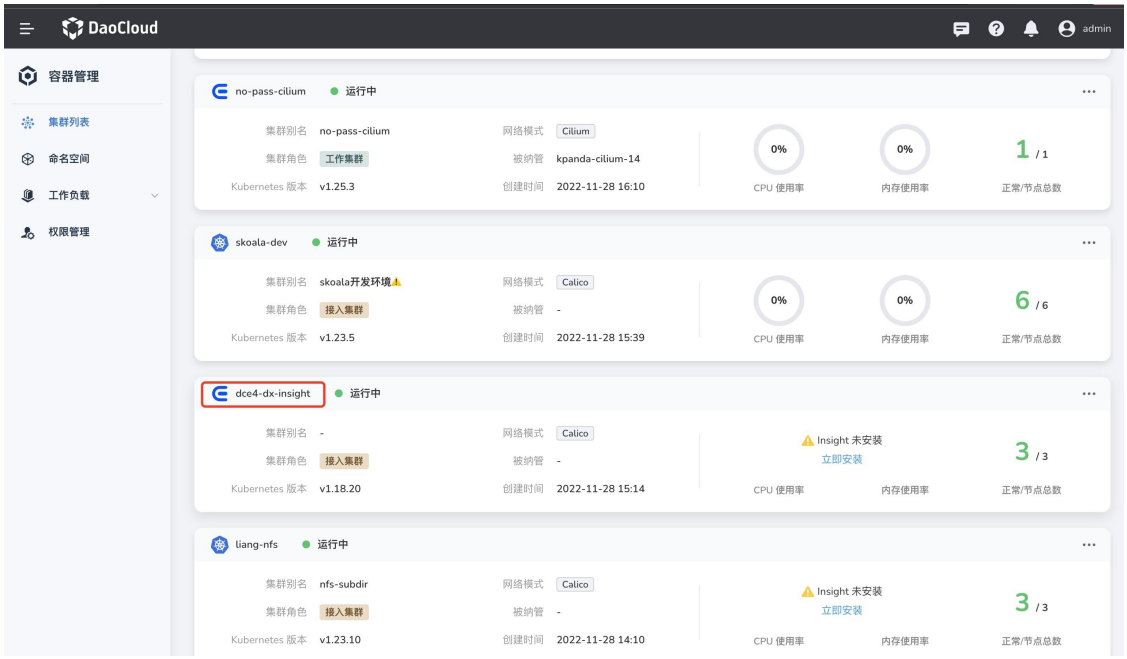
更多手工升级注意事项，请参考 Istio 官方文档：
https://istio.io/latest/docs/setup/upgrade/

2022-11-17

安装服务网格

请确认您的集群已成功接入容器管理平台，然后执行以下步骤安装服务网格。

1. 从左侧导航栏点击容器管理，进入集群列表，点击准备安装服务网格的集群名称。



安装采集器

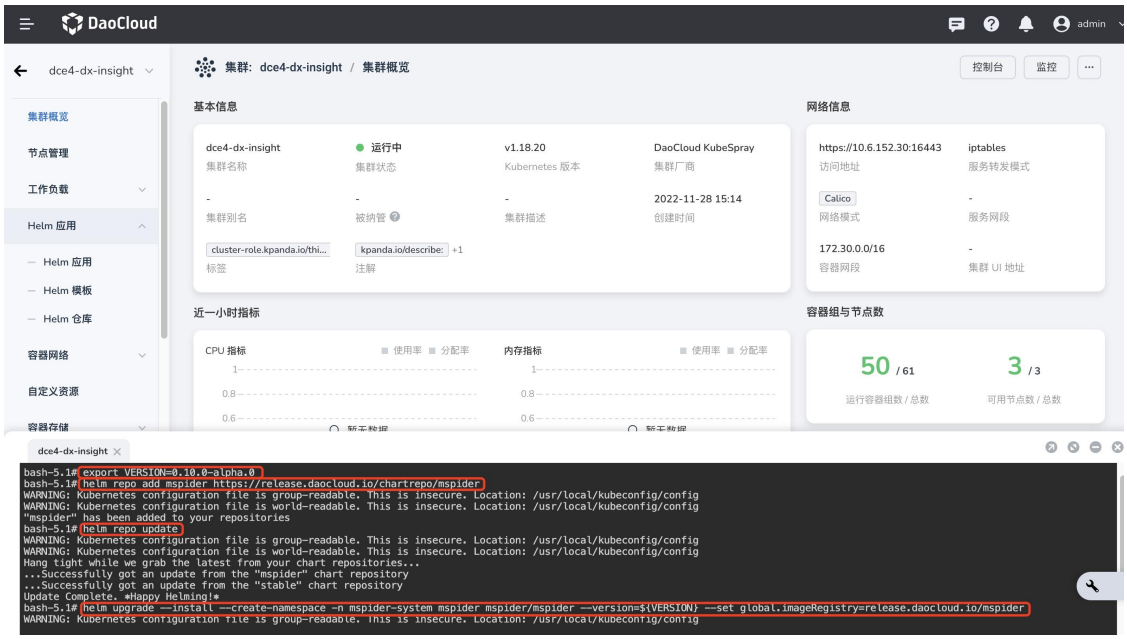
2. 在集群概览页面中点击控制台。



安装采集器

3. 自控制台中逐行输入如下命令（请修改对应的 VERSION 版本号）：

```
export VERSION=0.0.0-xxxx
helm repo add mspider https://release.daocloud.io/chartrepo/mspider
helm repo update
helm upgrade --install --create-namespace -n mspider-system mspider
mspider/mspider --version=${VERSION} --set global.imageRegistry=releas
e.daocloud.io/mspider
```

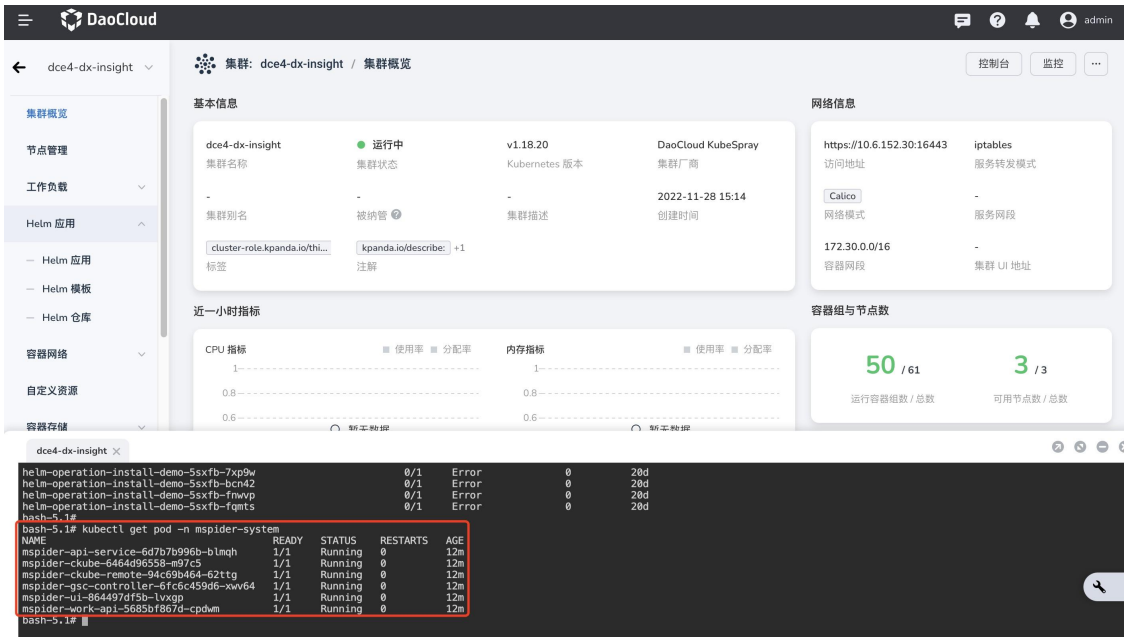


安装采集器

!!! note

请将 `0.0.0-xxx` 替换为计划安装的服务网格版本号。

4. 查看命名空间 mspider-system 下 Pod 信息，看到相关 Pod 已创建并运行，表示服务网格安装成功。



安装采集器

下一步：创建网格

边车版本升级

网格的 Istio 版本升级后，将触发边车升级功能。边车升级可分为热升级和重启升级两种升级方式。

- 热升级：道客订制版本 Istio 下边车可在不重启用户 Pod 的情况下完成边车升级，实现业务无中断；
- 重启升级：社区原生 Istio 或不满足热升级环境要求的订制版 Istio 下边车升级方式，需重启用户 Pod。

完成 Istio 版本升级后，进入工作负载边车管理界面，具备升级条件的工作负载将出现叹号提示信息，选中期望升级的工作负载，将出现边车升级按钮。



边车升级

点击边车升级按钮将进入边车升级版本向导，该向导分为环境检测、选择目标版本、执行升级三个步骤，在“热升级”和“重启升级”两种方式下，操作会有一定差别。

热升级

1. **环境检测**：该步骤中将检测集群环境是否满足热升级需求，检测项包括以下三项：
    - Istio 版本：是否是定制版本（版本后缀：–mspider）

- K8s 版本：是否符合热升级要求范围
- EphemeralContainer：是否已启用

以上三项满足后将在后续步骤中进入热升级流程。



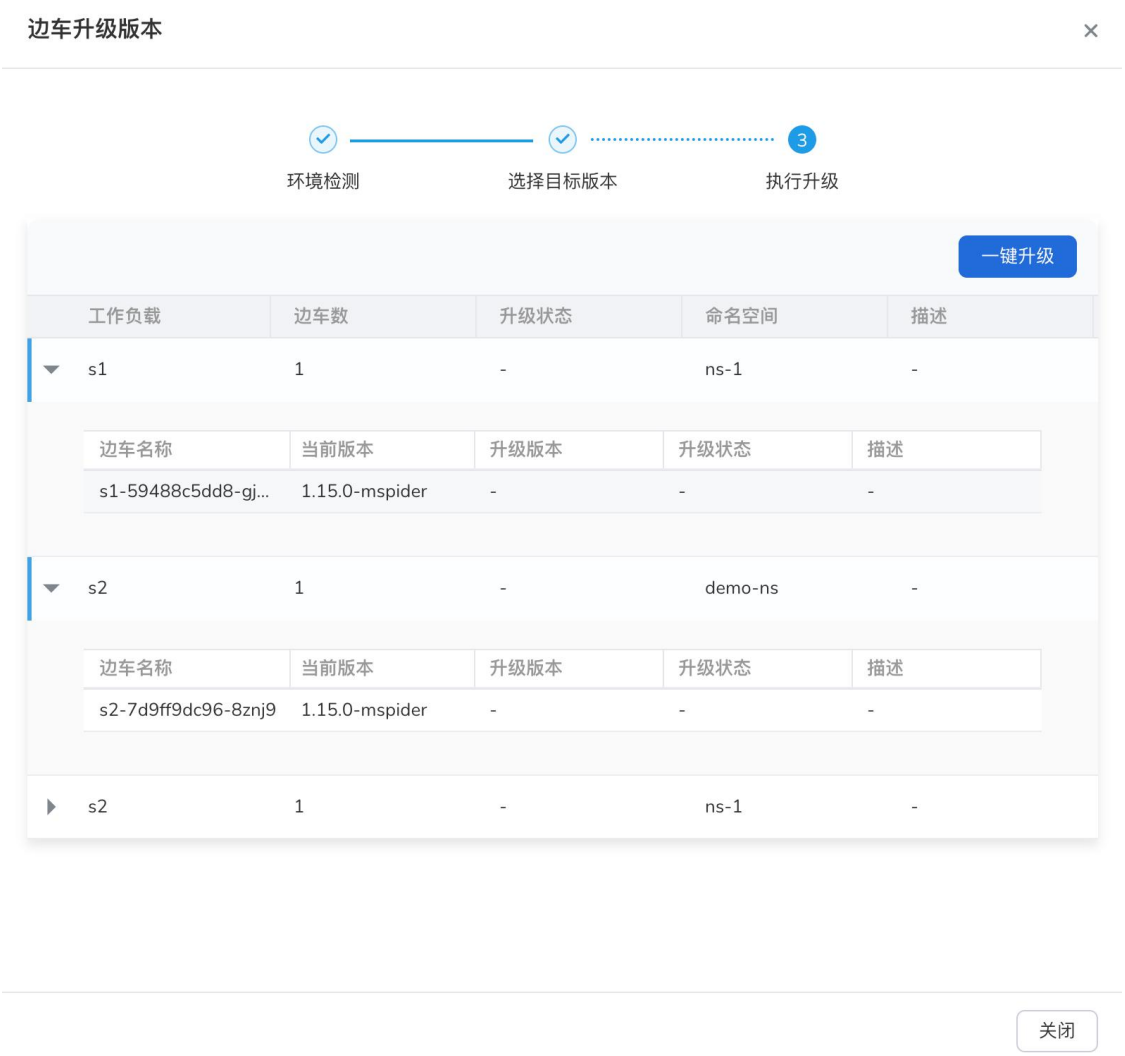
环境监测

2. **边车升级版本**：热升级流程中，可以选择期望升级的边车版本，默认为最新版本，如果选择了其他版本，相关 Pod 重启后也会自动升级至最新版本。



升级版本

3. **执行升级**：升级页面中展示了所选工作负载及相关边车信息，点击一键升级将启动升级过程。



执行升级

重启升级

1. **环境检测**：在检测阶段如果检测项不满足热升级要求，之后两个步骤将进入重启升级流程。



环境监测

2. **边车升级版本** 在重启升级流程中，无法选择版本，仅支持升级至最新版本。



升级版本

3. **执行升级**：升级页面中展示了所选工作负载的基本信息及边车版本信息，点击一键升级将立即重启 Pod，请务必谨慎操作。



执行升级



执行升级

!!! note

升级过程中关闭升级向导不会中断当前升级任务。

创建网格时找不到所属集群

原因分析

主要原因在于所属集群的 MeshCluster 状态不是“CLUSTER_RUNNING”，可登录 global 集群查看所属集群的 MeshCluster CRD 状态。

```
kubectl get meshcluster -n mspider-system
```

造成这种问题的情况有几种：

情况 1

如果先前未使用该所属集群创建过网格，执行上述命令未发现该所属集群的 meshcluster CRD， 可能是因为 gsc-controller 从容器管理（kpanda）同步集群异常。

情况 2

针对已创建的网格移除集群，执行上述命令。发现该集群的 meshcluster 状态可能处于如下状态之一：

- MANAGED_RECONCILING
- MANAGED_SUCCEED
- MANAGED_EVICTING
- MANAGED_FAILED

原因可能是正在清理资源或者存在 meshconfig 未清理干净。

解决方案

1. 针对情况 1，只需要重启 global 集群的 gsc-controller 即可。

   ```
   kubectl -n mspider-system delete pod $(kubectl -n mspider-system get pod -l app=mspider-gsc-controller -o 'jsonpath={.items.metadata.name}')
   ```

2. 针对情况 2，由于环境可能未清理干净。请确保所属集群(控制面集群)控制面组件清理干净，否则影响下一次网格创建。

    1. 删除未清理干净导致装态异常的 meshconfig

        ```
        kubectl delete meshcluster -n mspider-system ${clustername}
        ```

    2. 重启 gsc-controller 重新同步集群

        ```
        kubectl delete po -n mspider-system ${gsc-coontroller-xxxxxxxxxx}
        ```

创建的网格异常但无法删除

原因分析

网格处于失败状态，无法点击网格实例。 由于该网格中纳管了集群、创建了网格网关实例、或启用了边车注入，导致移除网格时检测总是失败，所以无法被正常删除。

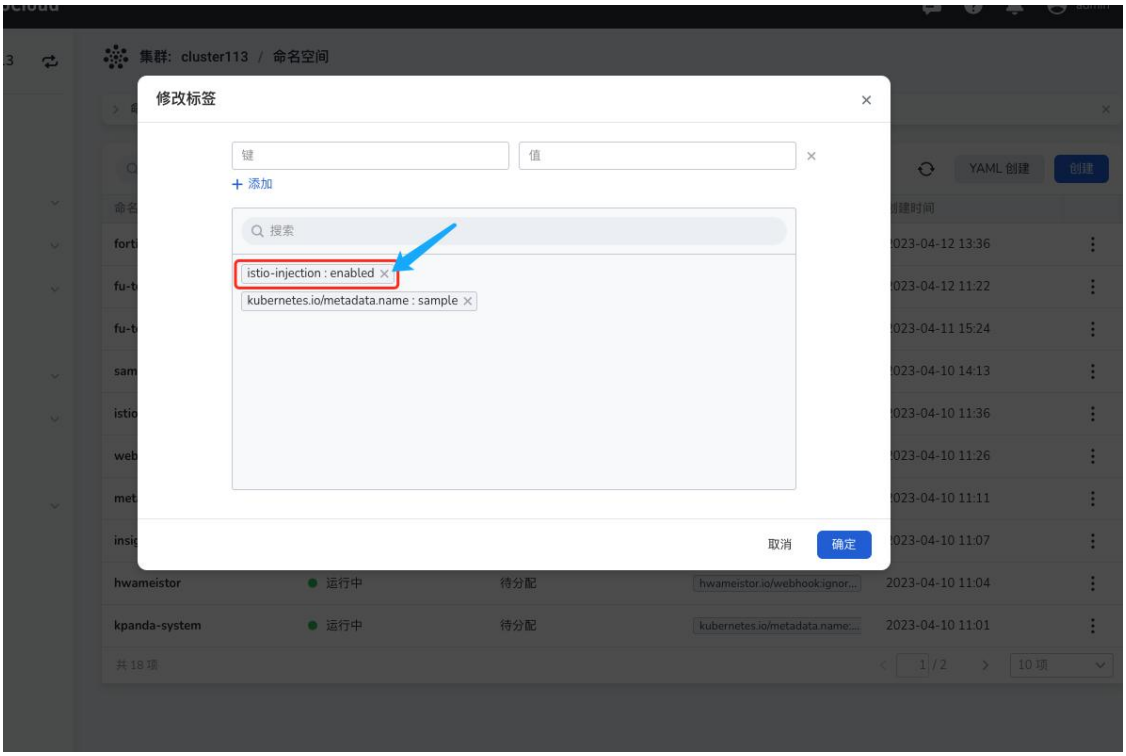
解决方案

建议排查具体网格失败的原因并解决，如果想要强制删除，请执行以下操作：

1. 禁用纳管集群的边车注入

    1. 禁用命名空间边车自动注入。

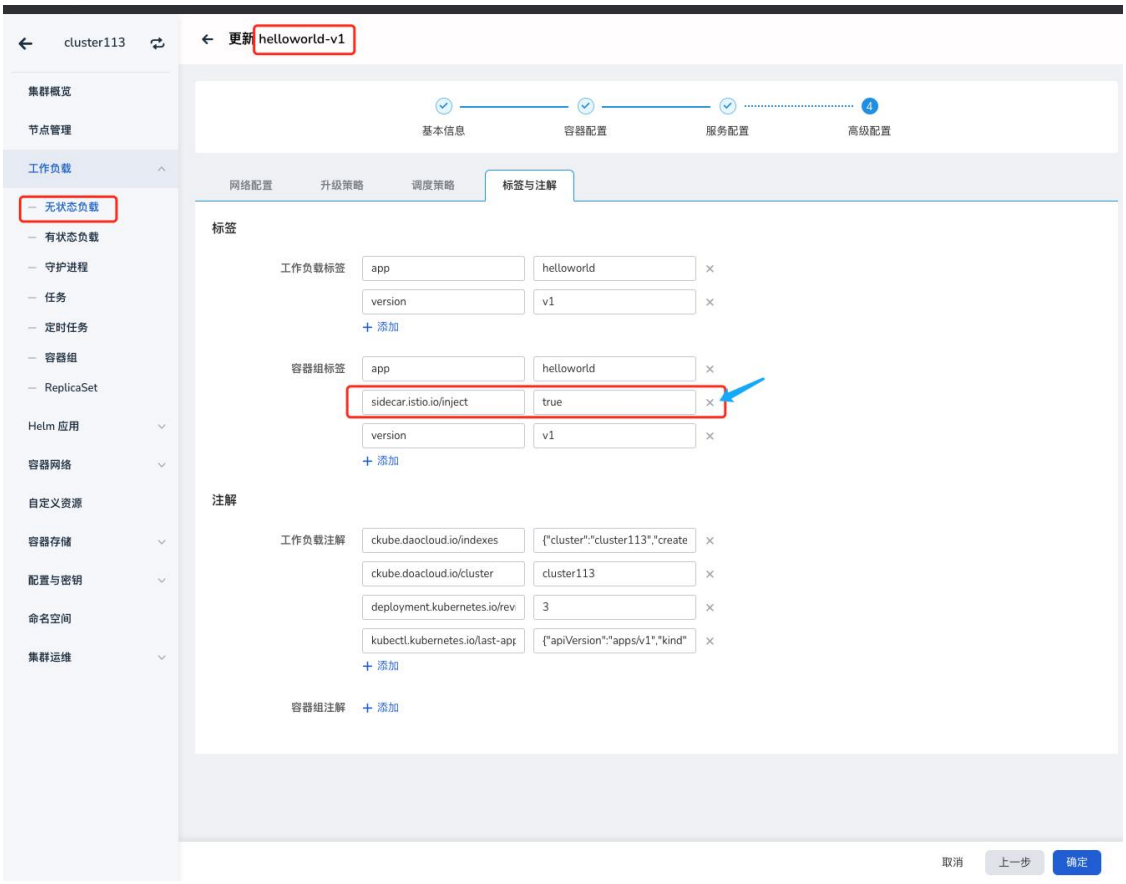
        在容器管理中，选择该集群 —> 命名空间 —> 修改标签 —> 移除 istio-injection: enabled 标签，重启该命名空间下的所有 Pod。



移除标签

2. 禁用工作负载边车注入：

   在容器管理中，选择该集群 —> 工作负载 —> 无状态负载 —> 标签与注解 —> 移除 sidecar.istio.io/inject: true 标签。

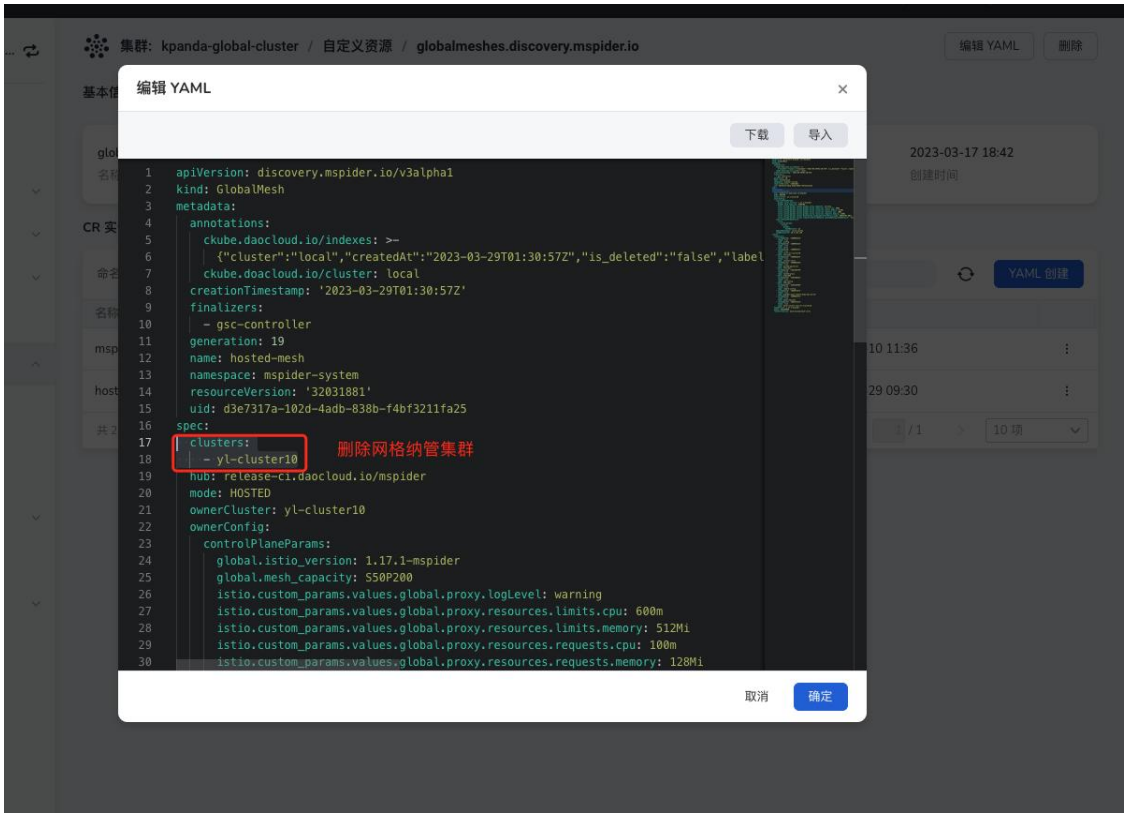


禁用边车注入

2. 删除创建的网格网关实例。

3. 移除集群。

   在容器管理中，选择 global 集群，自定义资源搜索 globalmeshes.discovery.mspider.io。 在 mspider-system 命名空间下选择要移除集群的网格，编辑 YAML：



编辑 yaml

4. 返回服务网格，删除该网格实例。

托管网格纳管集群时 istio-ingressgateway 异常

情况分析

当托管网格纳管工作负载集群时，常常会出现 istio-ingressgateway 组件不健康。

如下图：

```
[root@controller55 containerd]# kubectl get po -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
istio-ingressgateway-54b6f7c774-8djkv   0/1     Running   0          5m9s
istiod-66b59bcbf7-g8sr5                 1/1     Running   0          5m14s
```

不健康

首先查看 istio-ingressgateway 日志报错信息：

2023-04-25T12:18:04.657568Z info JWT policy is third-party-jwt

2023-04-25T12:18:04.657573Z info using credential fetcher of JWT type in cluster.local trust domain

2023-04-25T12:18:06.658680Z info Workload SDS socket not found. Starting Istio SDS Server

2023-04-25T12:18:06.658716Z info CA Endpoint istiod.istio-system.svc:15012, provider Citadel

2023-04-25T12:18:06.658747Z info Using CA istiod.istio-system.svc:15012 cert with certs: var/run/secrets/istio/root-cert.pem

2023-04-25T12:18:06.667981Z info Opening status port 15020

2023-04-25T12:18:06.694111Z info ads All caches have been synced up in 2.053719558s, marking server ready

2023-04-25T12:18:06.694864Z info xdsproxy Initializing with upstream address "istiod-remote.istio-system.svc:15012" and cluster "yw55"

2023-04-25T12:18:06.696385Z info sds Starting SDS grpc server

2023-04-25T12:18:06.696568Z info starting Http service at 127.0.0.1:15004

2023-04-25T12:18:06.698950Z info Pilot SAN: [istiod-remote.istio-syste

m.svc]
2023-04-25T12:18:06.705118Z info Starting proxy agent
2023-04-25T12:18:06.705177Z info starting
2023-04-25T12:18:06.705214Z info Envoy command: [-c etc/istio/proxy/envoy-rev.json --restart-epoch 0 --drain-time-s 45 --parent-shutdown-time-s 60 --local-address-ip-version v4 --file-flush-interval-msec 1000 --log-format %Y-%m-%dT%T.%fZ %l envoy %n %v -l warning --component-log-level misc:error]
2023-04-25T12:18:07.696708Z info cache generated new workload certificate latency=1.001557215s ttl=23h59m59.303308657s
2023-04-25T12:18:07.696756Z info cache Root cert has changed, start rotating root cert
2023-04-25T12:18:07.696785Z info ads XDS: Incremental Pushing:0 ConnectedEndpoints:0 Version:
2023-04-25T12:18:07.696896Z info cache returned workload trust anchor from cache ttl=23h59m59.303107754s
2023-04-25T12:19:07.664759Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 40s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"
2023-04-25T12:19:29.530922Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 62s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"
2023-04-25T12:19:51.228936Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 84s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"
2023-04-25T12:20:11.732449Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 104s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"

2023-04-25T12:20:41.426914Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 134s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"
2023-04-25T12:21:15.199447Z warning envoy config StreamAggregatedResources gRPC config stream to xds-grpc closed since 168s ago: 14, connection error: desc = "transport: Error while dialing dial tcp 10.233.48.75:15012: i/o timeout"

上面报错信息显示连接 10.233.48.75:15012 即 istiod-remote 的 service ip:15012 timeout !!!。

此时我们查看 istio-system 命名空间下 istiod-remote 的 endpoint。

kubectl get ep -n istio-system

NAME ENDPOINTS AGE
istio-eastwestgateway 36s
istio-ingressgateway 10m
istiod 10.233.97.220:15012,10.233.97.220:15010,10.233.97.220:15017 + 1 more... 10m
istiod-remote 10.233.95.141:15012,10.233.95.141:15017 10m

这里可以看出 istio-remote 分配的 endpoint 地址是 istiod-remote 10.233.95.141:15012,10.233.95.141:15017。如下图：

```
[root@controller-node-57 ~]# kubectl get po -n istio-system -owide
NAME                                      READY   STATUS    RESTARTS       AGE    IP             NODE              NOMINATED NODE   READINESS GATE
S
istio-eastwestgateway-7f55cd855c-xmndx    1/1     Running   0              97m    10.233.95.167  controller-node-57   <none>           <none>
istio-ingressgateway-5fb85b597f-tp7l5     1/1     Running   0              97m    10.233.95.143  controller-node-57   <none>           <none>
istiod-74b7855b7f-xm8mp                   1/1     Running   0              9h     10.233.95.146  controller-node-57   <none>           <none>
istiod-ywistio-hosted-bf4488c7d-v2hhc     1/1     Running   14 (28h ago)   29h    10.233.95.141  controller-node-57   <none>           <none>
mspider-mcpc-ckube-remote-d646f66cf-hmg56 1/1     Running   0              8h     10.233.95.189  controller-node-57   <none>           <none>
mspider-mcpc-mcpc-controller-5669fb4c8c-fhkzh 1/1 Running   0              8h     10.233.95.156  controller-node-57   <none>           <none>
ywistio-etcd-0                            1/1     Running   0              2d3h   10.233.95.178  controller-node-57   <none>           <none>
ywistio-etcd-1                            1/1     Running   0              28h    10.233.95.179  controller-node-57   <none>           <none>
ywistio-etcd-2                            1/1     Running   0              28h    10.233.95.142  controller-node-57   <none>           <none>
ywistio-hosted-apiserver-6b65bb46cf-f47jk 1/1     Running   0              8h     10.233.95.166  controller-node-57   <none>           <none>
[root@controller-node-57 ~]#
```

timeout

!!! note

工作负载集群接入托管网格时，istiod-remote 的 endpoint 地址分配的应该是 istiod-xxxx-hosted-lb service 的 loadBalancer IP:15012，
而这里却分配成控制面集群的 istiod-xxxx-hosted-xxxx Pod IP。

解决方案

更新网格基本信息配置中的控制面地址：

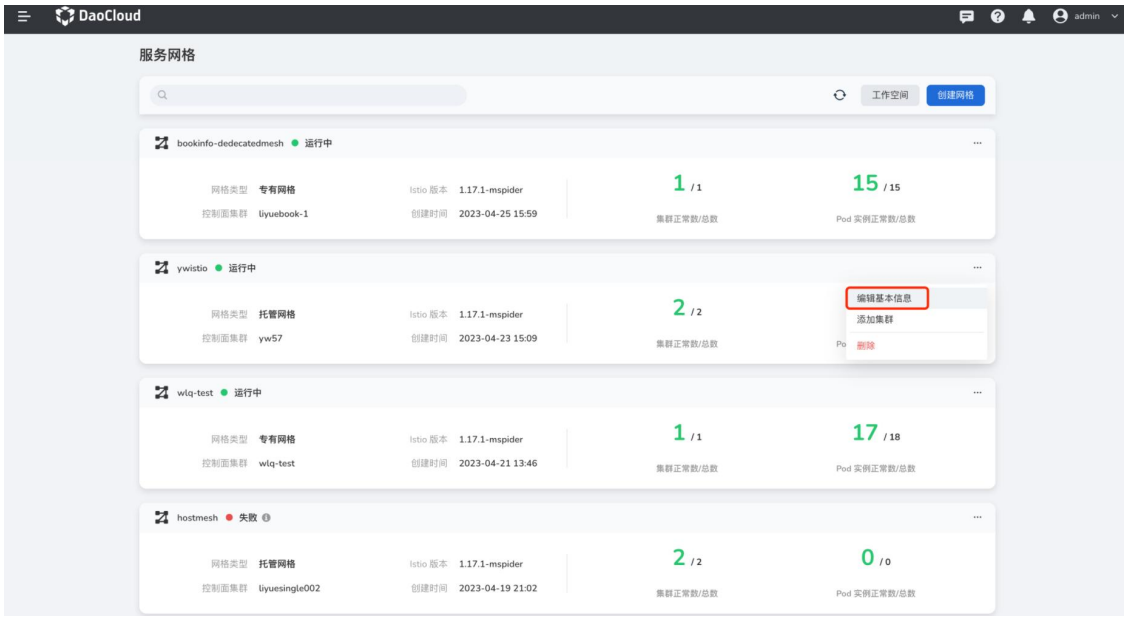
1. 登录控制面集群执行以下命令获取这个地址：

   kubectl get svc -n istio-system istiod-ywistio-hosted-lb -o "jsonpath={.status.loadBalancer.ingress[0].ip}"

```
[root@controller-node-57 ~]# kubectl get svc -n istio-system
NAME                        TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                                      AGE
istio-eastwestgateway       LoadBalancer   10.233.2.203    10.6.159.12   15443:31724/TCP                              108m
istio-ingressgateway        LoadBalancer   10.233.56.134   10.6.159.11   15021:32532/TCP,80:30422/TCP,443:30931/TCP   9h
istiod                      ClusterIP      10.233.59.72    <none>        15010/TCP,15012/TCP,443/TCP,15014/TCP        9h
istiod-remote               ClusterIP      10.233.56.14    <none>        15012/TCP,443/TCP                            9h
istiod-ywistio-hosted       ClusterIP      10.233.35.47    <none>        15010/TCP,15012/TCP,443/TCP,15014/TCP        30h
istiod-ywistio-hosted-lb    LoadBalancer   10.233.4.221    10.6.159.10   15012:31154/TCP,443:31674/TCP                30h
mspider-mcpc-ckube-remote   ClusterIP      10.233.31.154   <none>        80/TCP                                       2d4h
ywistio-etcd                ClusterIP      None            <none>        2379/TCP,2380/TCP                            2d4h
ywistio-hosted-apiserver    ClusterIP      10.233.21.90    <none>        6443/TCP                                     2d4h
```

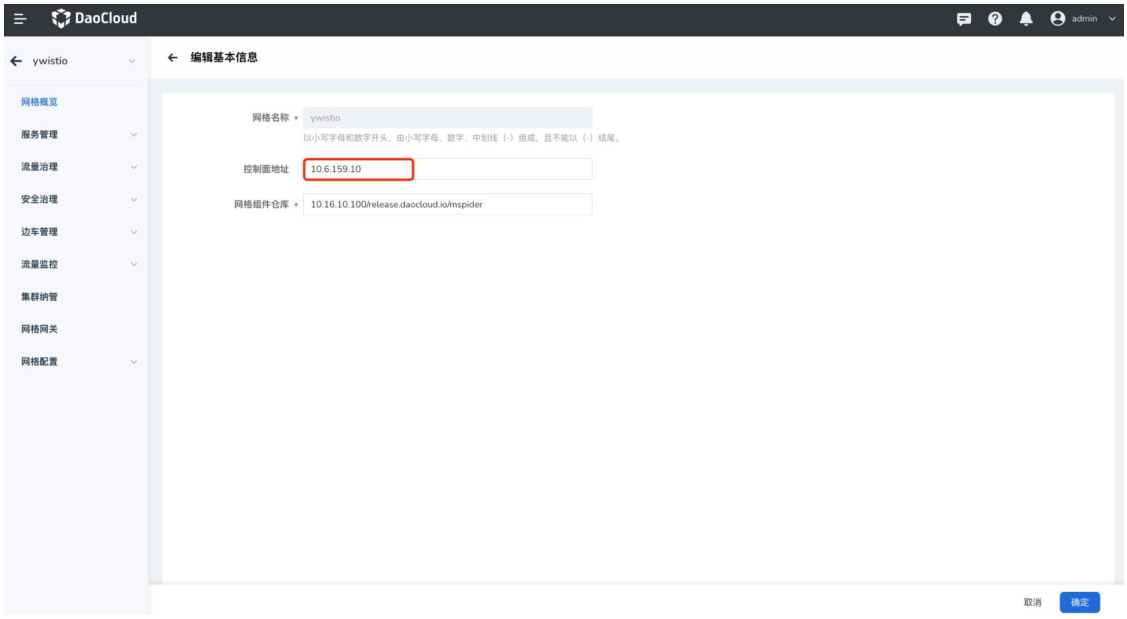
   获取地址

2. 点击右侧菜单，选择编辑基本信息。



基本信息

3. 填写控制面地址。



填写地址

4. 再次查看工作负载集群的 istio-ingressgateway，发现此时已经正常。

```
[root@controller55 containerd]# kubectl  get po -n istio-system -owide
NAME                                    READY   STATUS    RESTARTS   AGE   IP              NODE           NOMINATED NODE   READINESS GATES
istio-eastwestgateway-6748fd8965-jchml  1/1     Running   0          25m   10.233.97.223   controller55   <none>           <none>
istio-ingressgateway-7c6b5f9874-mrw58   1/1     Running   0          25m   10.233.97.222   controller55   <none>           <none>
istiod-66b59bcbf7-g8sr5                 1/1     Running   0          35m   10.233.97.220   controller55   <none>           <none>
[root@controller55 containerd]#
```

查看集群状态

5. 查看 istiod–remote endpoint 信息也正常。

```
[root@controller55 containerd]# kubectl  get ep -n istio-system
NAME                    ENDPOINTS                                                     AGE
istio-eastwestgateway   10.233.97.223:15443                                           27m
istio-ingressgateway    10.233.97.222:15021,10.233.97.222:8080,10.233.97.222:8443     36m
istiod                  10.233.97.220:15012,10.233.97.220:15010,10.233.97.220:15017 + 1 more...   36m
istiod-remote           10.6.159.10:15012,10.6.159.10:15017                           36m
[root@controller55 containerd]#
```

查看信息

创建网格时一直处于“创建中”

问题描述

mcpc-ckube-remote pod 一直 ContainerCreating。 mcpc-remote-kube-api-server configmap 等待很长时间没有创建。

```
root@yl-cluster1:~# kubectl get po -n istio-system
NAME                                          READY   STATUS              RESTARTS      AGE
hosted-mesh-hosted-apiserver-0                1/1     Running             0             17m
istiod-hosted-mesh-hosted-59f7bbfc7d-ldjpf    1/1     Running             0             17m
mspider-mcpc-ckube-remote-5447c5bcfc-25t7t    0/1     ContainerCreating   0             17m
mspider-mcpc-mcpc-controller-d7b76b945-vkfb7  0/1     CrashLoopBackOff    8 (76s ago)   17m
root@yl-cluster1:~# kubectl get cm -n istio-system
NAME                                  DATA   AGE
istio-hosted-mesh-hosted              2      17m
istio-sidecar-injector-hosted-mesh-hosted   2      17m
kube-root-ca.crt                      1      18m
mspider-mcpc                          1      17m
mspider-mcpc-ckube-remote             1      17m
root@yl-cluster1:~#
```

故障截图

日志

1. 查看 Pod 日志

   kubectl describe pod mspider-mcpc-ckube-remote-5447c5bcfc-25t7t -n istio-system

   Events:
   Type Reason Age From Message
   ---- ------ ---- ---- -------
   Normal Scheduled 18m default-scheduler Successfully assigned istio-system/mspider-mcpc-ckube-remote-5447c5bcfc-25t7t to yl-cluster1
   Warning FailedMount 6m59s kubelet Unable to attach or mount volumes: unmounted volumes=[remote-kube-api-serve

r], unattached volumes=[remote-kube-api-server kube-api-access-ljncs c
kube-config]: timed out waiting for the condition
Warning FailedMount 2m23s (x5 over 16m) kubelet Unable
to attach or mount volumes: unmounted volumes=[remote-kube-api-serve
r], unattached volumes=[ckube-config remote-kube-api-server kube-api-
access-ljncs]: timed out waiting for the condition
Warning FailedMount 105s (x16 over 18m) kubelet MountVol
ume.SetUp failed for volume "remote-kube-api-server" : configmap "mspi
der-mcpc-remote-kube-api-server" not found
Warning FailedMount 5s (x2 over 13m) kubelet Unable t
o attach or mount volumes: unmounted volumes=[remote-kube-api-serve
r], unattached volumes=[kube-api-access-ljncs ckube-config remote-kub
e-api-server]: timed out waiting for the condition

2. 查看 gsc controller 日志

??? note “点击查看详细日志”

```none
time="2022-12-27T08:08:10Z" level=error msg="unable to get livez for cl
uster hosted-mesh-hosted: client rate limiter Wait returned an error: cont
ext deadline exceeded - error from a previous attempt: read tcp 192.188.
110.245:51674->10.105.106.29:6443: read: connection reset by peer" func=
"mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-clust
er.go:191"
time="2022-12-27T08:08:10Z" level=error msg="cluster hosted-mesh-hos
ted reconcile api-server-healthy error: client rate limiter Wait returned an
error: context deadline exceeded - error from a previous attempt: read
tcp 192.188.110.245:51674->10.105.106.29:6443: read: connection reset by
peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1
212"
time="2022-12-27T08:08:13Z" level=error msg="unable to get livez for cl
uster hosted-mesh-hosted: client rate limiter Wait returned an error: cont
```

ext deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35842->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:13Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35842->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:16Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35874->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:16Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35874->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:19Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35902->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:19Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:35902->10.105.106.29:6443: read: connection reset by

peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:22Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:35940->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:22Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:35940->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:23Z" level=error msg="cluster cluster1-141 reconcile component-status error: deployment: istio-system/mspider-mcpc-mcpc-controller, error: {type: Available, reason: Deployment does not have minimum availability.;type: Progressing, reason: ReplicaSet \"mspider-mcpc-mcpc-controller-d7b76b945\" has timed out progressing.};deployment: istio-system/mspider-mcpc-ckube-remote, error: {type: Available, reason: Deployment does not have minimum availability.;type: Progressing, reason: ReplicaSet \"mspider-mcpc-ckube-remote-5447c5bcfc\" has timed out progressing.}" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:23Z" level=info msg="get mesh hosted-mesh's address: 0.0.0.0:30527" func="hosted-apiserver-proxy.(*proxyServer).GetMeshPort()" file="proxy.go:87"

time="2022-12-27T08:08:23Z" level=error msg="cluster cluster1-141 reconcile component-status error: deployment: istio-system/mspider-mcpc-mcpc-controller, error: {type: Available, reason: Deployment does not have minimum availability.;type: Progressing, reason: ReplicaSet \"mspider-mcpc-mcpc-controller-d7b76b945\" has timed out progressing.};deployment: i

stio-system/mspider-mcpc-ckube-remote, error: {type: Available, reason: Deployment does not have minimum availability.;type: Progressing, reason: ReplicaSet \"mspider-mcpc-ckube-remote-5447c5bcfc\" has timed out progressing.}" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:25Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:52050->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:25Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:52050->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

1.672128505418613e+09 ERROR Reconciler error {"controller": "meshcluster", "controllerGroup": "discovery.mspider.io", "controllerKind": "MeshCluster", "MeshCluster": {"name":"hosted-mesh-hosted","namespace":"mspider-system"}, "namespace": "mspider-system", "name": "hosted-mesh-hosted", "reconcileID": "e3583462-c271-4e58-ae00-18ba923362b8", "error": "Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again"}

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:326

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/

controller/controller.go:273
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:234
time="2022-12-27T08:08:25Z" level=error msg="update cluster status error: Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1241"
time="2022-12-27T08:08:28Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: EOF" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"
time="2022-12-27T08:08:28Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: EOF" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"
time="2022-12-27T08:08:31Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:52104->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"
time="2022-12-27T08:08:31Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:52104->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"
1.6721285114885132e+09 ERROR Reconciler error {"controller":

"meshcluster", "controllerGroup": "discovery.mspider.io", "controllerKind": "MeshCluster", "MeshCluster": {"name":"hosted-mesh-hosted","namespace":"mspider-system"}, "namespace": "mspider-system", "name": "hosted-mesh-hosted", "reconcileID": "b3abe28d-8ec4-4a69-8ecb-1e69e2f8c12d", "error": "Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again"}

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:326

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:273

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2

time="2022-12-27T08:08:31Z" level=error msg="update cluster status error: Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1241"

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:234

time="2022-12-27T08:08:34Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:49646->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:34Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned a

n error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:49646->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:37Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:49656->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"

time="2022-12-27T08:08:37Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded – error from a previous attempt: read tcp 192.188.110.245:49656->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"

time="2022-12-27T08:08:37Z" level=error msg="update cluster status error: Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1241"

1.6721285175748258e+09 ERROR Reconciler error {"controller": "meshcluster", "controllerGroup": "discovery.mspider.io", "controllerKind": "MeshCluster", "MeshCluster": {"name":"hosted-mesh-hosted","namespace":"mspider-system"}, "namespace": "mspider-system", "name": "hosted-mesh-hosted", "reconcileID": "4cc3c5be-dbe1-4bdc-a606-70ff2b7bc3f2", "error": "Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again"}

sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler

/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/

controller/controller.go:326
 sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
 /go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:273
 sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2
 /go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:234
 time="2022-12-27T08:08:39Z" level=error msg="mesh hosted-mesh reconcile remote-config error: create mesh hosted-mesh hosted ns error: an error on the server (\"unknown\") has prevented the request from succeeding (post namespaces)" func="global-mesh.(*Reconciler).Reconcile()" file="global-mesh.go:1198"
 time="2022-12-27T08:08:40Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:49676->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"
 time="2022-12-27T08:08:40Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:49676->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"
 time="2022-12-27T08:08:40Z" level=error msg="update cluster status error: Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1241"
 1.6721285206091342e+09 ERROR Reconciler error {"controller":

"meshcluster", "controllerGroup": "discovery.mspider.io", "controllerKind": "MeshCluster", "MeshCluster": {"name":"hosted-mesh-hosted","namespace":"mspider-system"}, "namespace": "mspider-system", "name": "hosted-mesh-hosted", "reconcileID": "31adec56-ca5b-4636-a74e-3ce4ae8b07af", "error": "Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again"}
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:326
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:273
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:234
time="2022-12-27T08:08:43Z" level=error msg="unable to get livez for cluster hosted-mesh-hosted: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:47316->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).checkAPIServerHealthy.func1()" file="mesh-cluster.go:191"
time="2022-12-27T08:08:43Z" level=error msg="cluster hosted-mesh-hosted reconcile api-server-healthy error: client rate limiter Wait returned an error: context deadline exceeded - error from a previous attempt: read tcp 192.188.110.245:47316->10.105.106.29:6443: read: connection reset by peer" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1212"
time="2022-12-27T08:08:43Z" level=error msg="update cluster status err

```
or: Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again" func="mesh-cluster.(*Reconciler).Reconcile()" file="mesh-cluster.go:1241"
1.6721285236628819e+09 ERROR Reconciler error {"controller": "meshcluster", "controllerGroup": "discovery.mspider.io", "controllerKind": "MeshCluster", "MeshCluster": {"name":"hosted-mesh-hosted","namespace":"mspider-system"}, "namespace": "mspider-system", "name": "hosted-mesh-hosted", "reconcileID": "908883a2-78a3-48f0-8a91-4f963903ae19", "error": "Operation cannot be fulfilled on meshclusters.discovery.mspider.io \"hosted-mesh-hosted\": the object has been modified; please apply your changes to the latest version and try again"}
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:326
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:273
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2
/go/pkg/mod/sigs.k8s.io/controller-runtime@v0.13.0/pkg/internal/controller/controller.go:234
```

原因分析

1. 情况 1：托管网格由于控制面集群没有提前部署 StorageClass 导致无法创建高可用 ETCD。

   xxxxx-etcd-0 一直 pending，etcd pvc 无法绑定 sc 导致 pvc pending， 进而导致 etcd pod 无法绑定 pvc。可以尝试以下步骤解决问题：

1. 部署 hwameistor 或者 localPath
2. 删除 istio-system 命名空间下 pending 的 pvc
3. 重启一下 xxx-etcd-0 pod，等待即可

!!! note
使用 hwameistor 完成部署后，每个节点必须存在一个空盘。您需要创建 LDC，然后检查 LSN LocalStorage_PoolHDD。

2. 情况 2：托管网格 istiod-xxxx-hosted-xxxx 组件异常

3. 情况 3：mspider-mcpc-ckube-remote-xxxx 组件异常，describe 出现如下报错：

Normal Scheduled 18m default-scheduler Successfully assigned istio-system/mspider-mcpc-ckube-remote-5447c5bcfc-25t7t to yl-cluster20
Warning FailedMount 6m59s kubelet Unable to attach or mount volumes: unmounted volumes=[remote-kube-api-server], unattached volumes=[remote-kube-api-server kube-api-access-ljncs ckube-config]: timed out waiting for the condition
Warning FailedMount 2m23s (x5 over 16m) kubelet Unable to attach or mount volumes: unmounted volumes=[remote-kube-api-server], unattached volumes=[ckube-config remote-kube-api-server kube-api-access-ljncs]: timed out waiting for the condition
Warning FailedMount 105s (x16 over 18m) kubelet MountVolume.SetUp failed for volume "remote-kube-api-server" : configmap "mspider-mcpc-remote-kube-api-server" not found
Warning FailedMount 5s (x2 over 13m) kubelet Unable to attach or mount volumes: unmounted volumes=[remote-kube-api-server], unattached volumes=[kube-api-access-ljncs ckube-config remote-kube-api-server]: timed out waiting for the condition

4. 情况 4：inotify watcher limit problems, remote-ckube 组件日志

panic: too many open files

解决方案

1. 情况 1：控制面集群提前部署 sc。
2. 情况 2：该组件异常可能控制面集群未部署 metalLB 导致网络不通，istiod-xxxx-hosed-lb 无法分配 endpoint。可在 addon 中为该集群部署 metalLB。
3. 情况 3：在移除原有托管网格后的环境中，再次创建托管网格的情况下，容易出现控制面还没有及时下发导致“mspider-mcpc-remote-kube-api-server”ConfigMap 未及时创建。可以重启一下 global 集群 gsc controller：

   ```
   kubectl -n mspider-system delete pod $(kubectl -n mspider-system get pod -l app=mspider-gsc-controller -o 'jsonpath={.items.metadata.name}')
   ```

4. 情况 4：修改 fs.inotify.max_user_instances = 65535

创建网格故障排查

本文将持续统计和梳理创建网格过程可能因环境或操作不规范引起的报错，以及服务网格使用过程中遇到某些报错的问题分析、解决方案。 若遇到服务网格的使用问题，请优先查看此排障手册。

- 托管模式：控制面和数据面分开，创建一个虚拟集群用来保存托管的 Istio CRD 资源

  创建托管网格时，会在托管控制面集群创建托管控制面虚拟集群 API Server 用来保存 Istio CRD 资源； 原则上一套 DCE 5.0 环境仅允许有一个托管网格。 DCE 5.0 的 global 集群含有全局管理所依赖的 Istio 相关组件， 不允许使用该集群创建托管模式网格，demo 时可以使用外接网格创建。

- 专有模式：控制面和数据面部署在一个集群
- 外接模式：所属集群必须含有 istio 相关组件

常见故障案例

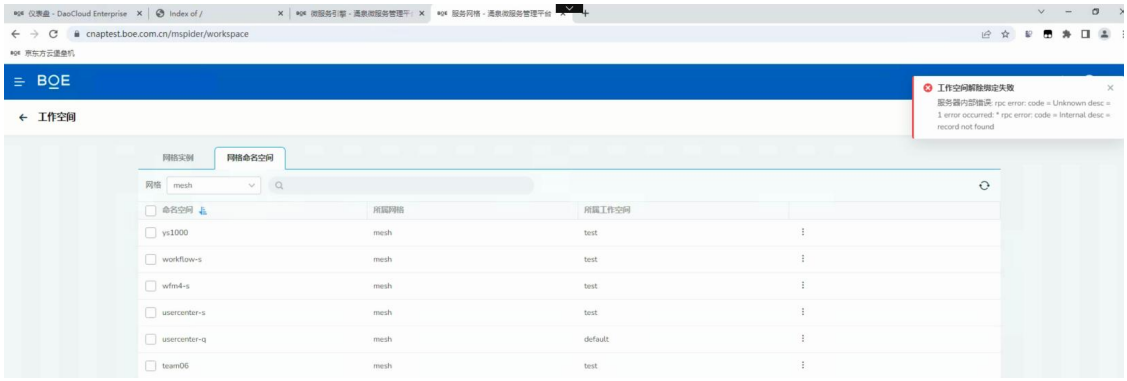
- 创建网格时找不到所属集群
- 创建网格时一直处于“创建中”，最终创建失败
- 创建的网格异常，但无法删除网格
- 托管网格纳管集群失败
- 托管网格纳管集群时 istio-ingressgateway 异常
- 网格空间无法正常解绑

网格空间无法正常解绑

问题描述

- 网格类型：托管网格
- Istio 版本：0.16.1-mspider

全局管理的工作空间改变名称后，网格空间无法正常解绑； 同样在全局管理中解除空间绑定后，服务网格依然显示绑定而且无法解绑，网格空间中被全局管理删除的空间无法解除绑定。



unbind error

分析

服务网格对工作空间做了缓存，需要清除缓存中的脏数据。

解决办法

1. 进入对应网格实例的部署在的 $ClusterName，找到这个 mesh-hosted-apiserver

   /kpanda/clusters/$ClusterName/namespaces/istio-system/pods/mesh-hosted-apiserver-0/containers

2. 使用控制台进入 pod 内部，然后调整下 kubectl 命令的权限

kubeadm init phase kubeconfig admin

alias kubectl="kubectl --kubeconfig=/etc/kubernetes/admin.conf --insurce-skip-tls-verify"

3. 使用下方命令移除对应 namespace 上的 annotation 注解

kubectl annotate ns $namespace controller.mspider.io/workspace-id- controller.mspider.io/workspace-name-

托管网格纳管集群失败

问题

- 托管网格接入新集群时 istio-ingressgateway 无法正常工作。

  ??? note “点击查看完整的错误日志”

  ```none
  [root@dce88 metallb]# kubectl describe pod istio-ingressgateway-b8b597c59-7gnwr -n istio-system
  Name:             istio-ingressgateway-b8b597c59-7gnwr
  Namespace:        istio-system
  Priority:         0
  Node:             dce88/10.6.113.100
  Start Time:       Fri, 24 Feb 2023 10:51:22 +0800
  Labels:           app=istio-ingressgateway
                    chart=gateways
                    heritage=Tiller
                    install.operator.istio.io/owning-resource=unknown
                    istio=ingressgateway
                    istio.io/rev=default
                    operator.istio.io/component=IngressGateways
                    pod-template-hash=b8b597c59
                    release=istio
                    service.istio.io/canonical-name=istio-ingressgateway
                    service.istio.io/canonical-revision=latest
                    sidecar.istio.io/inject=false
  Annotations:      prometheus.io/path: /stats/prometheus
                    prometheus.io/port: 15020
                    prometheus.io/scrape: true
                    sidecar.istio.io/inject: false
  ```

Status: Running
IP: 10.244.0.17
IPs:
IP: 10.244.0.17
Controlled By: ReplicaSet/istio-ingressgateway-b8b597c59
Containers:
istio-proxy:
Container ID: docker://40ef710d84433b108b58e8d286b46d6fde3f5ac2f6b7e80765d61aafb2d269da
Image: 10.16.10.120/release.daocloud.io/mspider/proxyv2:1.16.1-mspider
Image ID: docker-pullable://10.16.10.120/release.daocloud.io/mspider/proxyv2@sha256:4ac80ff62af5ebe7ebb5cb75d1b1f6791b55b56e1b45a0547197b26d9674fd22
Ports: 15021/TCP, 8080/TCP, 8443/TCP, 15090/TCP
Host Ports: 0/TCP, 0/TCP, 0/TCP, 0/TCP
Args:
proxy
router
--domain
$(POD_NAMESPACE).svc.cluster.local
--proxyLogLevel=warning
--proxyComponentLogLevel=misc:error
--log_output_level=default:info
State: Running
Started: Fri, 24 Feb 2023 10:51:25 +0800
Ready: False
Restart Count: 0
Limits:
cpu: 2
memory: 1Gi
Requests:

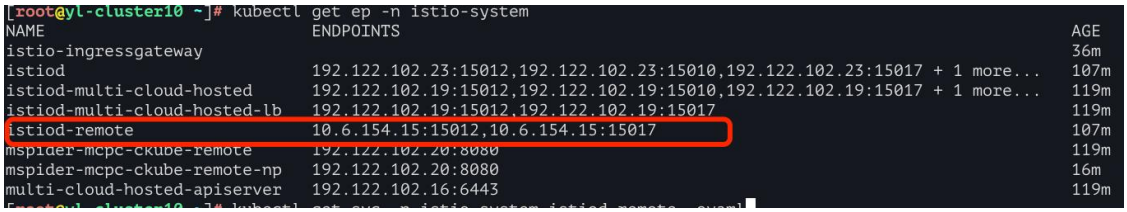
```
      cpu:          100m
      memory:     128Mi
    Readiness:   http-get http://:15021/healthz/ready delay=1s timeout=1
s period=2s #success=1 #failure=30
    Environment:
      JWT_POLICY:                    third-party-jwt
      PILOT_CERT_PROVIDER:           istiod
      CA_ADDR:                       istiod.istio-system.svc:15012
      NODE_NAME:                      (v1:spec.nodeName)
      POD_NAME:                      istio-ingressgateway-b8b597c
59-7gnwr (v1:metadata.name)
      POD_NAMESPACE:                 istio-system (v1:metadata.na
mespace)
      INSTANCE_IP:                   (v1:status.podIP)
      HOST_IP:                       (v1:status.hostIP)
      SERVICE_ACCOUNT:                (v1:spec.serviceAccountNam
e)
      ISTIO_META_WORKLOAD_NAME:      istio-ingressgateway
      ISTIO_META_OWNER:              kubernetes://apis/apps/v1/na
mespaces/istio-system/deployments/istio-ingressgateway
      ISTIO_META_MESH_ID:            fupan-mesh
      TRUST_DOMAIN:                  cluster.local
      ISTIO_META_UNPRIVILEGED_POD:   true
      ISTIO_META_DNS_AUTO_ALLOCATE:  true
      ISTIO_META_DNS_CAPTURE:        true
      ISTIO_META_CLUSTER_ID:         futest-2
    Mounts:
      /etc/istio/config from config-volume (rw)
      /etc/istio/ingressgateway-ca-certs from ingressgateway-ca-cert
s (ro)
      /etc/istio/ingressgateway-certs from ingressgateway-certs (ro)
      /etc/istio/pod from podinfo (rw)
```

/etc/istio/proxy from istio-envoy (rw)
/var/lib/istio/data from istio-data (rw)
/var/run/secrets/credential-uds from credential-socket (rw)
/var/run/secrets/istio from istiod-ca-cert (rw)
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-hpgkx (ro)
/var/run/secrets/tokens from istio-token (ro)
/var/run/secrets/workload-spiffe-credentials from workload-certs (rw)
/var/run/secrets/workload-spiffe-uds from workload-socket (rw)
Conditions:
Type Status
Initialized True
Ready False
ContainersReady False
PodScheduled True
Volumes:
workload-socket:
Type: EmptyDir (a temporary directory that shares a pod's lifetime)
Medium:
SizeLimit: <unset>
credential-socket:
Type: EmptyDir (a temporary directory that shares a pod's lifetime)
Medium:
SizeLimit: <unset>
workload-certs:
Type: EmptyDir (a temporary directory that shares a pod's lifetime)
Medium:
SizeLimit: <unset>

```
  istiod-ca-cert:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      istio-ca-root-cert
    Optional:  false
  podinfo:
    Type:  DownwardAPI (a volume populated by information about the
 pod)
    Items:
      metadata.labels -> labels
      metadata.annotations -> annotations
  istio-envoy:
    Type:      EmptyDir (a temporary directory that shares a pod's lif
etime)
    Medium:
    SizeLimit:  <unset>
  istio-data:
    Type:      EmptyDir (a temporary directory that shares a pod's lif
etime)
    Medium:
    SizeLimit:  <unset>
  istio-token:
    Type:                     Projected (a volume that contains injecte
d data from multiple sources)
    TokenExpirationSeconds:  43200
  config-volume:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      istio
    Optional:  true
  ingressgateway-certs:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  istio-ingressgateway-certs
    Optional:    true
```

```
    ingressgateway-ca-certs:
      Type:          Secret (a volume populated by a Secret)
      SecretName:    istio-ingressgateway-ca-certs
      Optional:      true
    kube-api-access-hpgkx:
      Type:                    Projected (a volume that contains injecte
d data from multiple sources)
      TokenExpirationSeconds:  3607
      ConfigMapName:           kube-root-ca.crt
      ConfigMapOptional:       <nil>
      DownwardAPI:             true
    QoS Class:                 Burstable
    Node-Selectors:            <none>
    Tolerations:               node.kubernetes.io/not-ready:NoExecute o
p=Exists for 300s
                               node.kubernetes.io/unreachable:NoExecute
op=Exists for 300s
    Events:
    Type     Reason     Age                  From               Messag
e
    ----     ------     ----                 ----               -----
--
    Normal   Scheduled  63s                  default-scheduler  Successf
ully assigned istio-system/istio-ingressgateway-b8b597c59-7gnwr to dce
88
    Normal   Pulled     62s                  kubelet            Container
 image "10.16.10.120/release.daocloud.io/mspider/proxyv2:1.16.1-mspider" a
lready present on machine
    Normal   Created    60s                  kubelet            Created
container istio-proxy
    Normal   Started    60s                  kubelet            Started c
ontainer istio-proxy
```

Warning Unhealthy 19s (x22 over 59s) kubelet Readines
s probe failed: Get "http://10.244.0.17:15021/healthz/ready": dial tcp 10.24
4.0.17:15021: connect: connection refused

```

- 托管网格的控制面和数据面一起部署的情况下，istiod-remote ep ip 分配错误。

  托管集群当作工作负载集群接入托管网格时，istiod-remote ep ip 分配为 metalLB IP，应该为 PodIP（mspider-mcpc-ckube-remote-xxx）




*istiod-remote*

## 原因分析

1. 情况一：被纳管集群没有安装 metalLB 导致集群网络不通，ingressgateway 无法正常分配 endpoint 一直 CrashLoopBackoff
2. 情况二：istio-remote 组件 endpoint 分配错误，执行 kubectl get ep -n istio-system 查看详情

## 解决方案

1. 情况一：addon 部署 metalLB
2. 情况二：检查
    1. 控制面集群 istio-remote ep: istio-${meshID}-hosted ${podIP}:15012/15017
    2. 工作负载集群 istio-remote ep: istiod-hosted-mesh-hosted-lb 的 ${loadBalancerIP}:15012/15017
```

DCE 4.0 接入问题追踪

本页列出一些服务网格接入 DCE 4.0 时常见的问题。

LimitRange 问题

问题描述

1. 接入 DCE 4.0 时报错。

   message: 'unable to retrieve Pods: Unauthorized'

2. 控制面集群。

   提示以下报错：

   mspider-mcpc-mcpc-controller-5bd6d54c4-df6kg CrashLoopBackOff

   查看控制面的日志：

   kubectl logs -n istio-system mspider-mcpc-mcpc-controller-5bd6d54c4-df6kg

   time="2022-10-19T06:35:49Z" level=error msg="Unable to get kube configs of multi clusters: Get \"http://mspider-mcpc-ckube-remote/api/v1/namespaces/istio-system/configmaps/mspider-mcpc-remote-kube-api-server?resourceVersion=dsm-cluster-dce4-mspider\": dial tcp: lookup mspider-mcpc-ckube-remote: i/o timeout" func="cmholder.NewConfigHolder()" file="holder.go:52"
   panic: unable to get kube configs of multi clusters: Get "http://mspider-mcpc-ckube-remote/api/v1/namespaces/istio-system/configmaps/mspider-mcpc-remote-kube-api-server?resourceVersion=dsm-cluster-dce4-mspider": dial tcp: lookup mspider-mcpc-ckube-remote: i/o timeout

   goroutine 1 [running]:

main.main()

/app/cmd/control-plane/mcpc/main.go:62 +0x694

3. rs: istio-operator-*** 报错：

message: 'pods "istio-operator-5fbbf5bbd-hf2q2" is forbidden: memory max limit

to request ratio per Container is 1, but provided ratio is 2.000000'

解决办法

需要手动在 istio-operator 和 isito-system 中设置 limit range，将超配比例设置成 0。

执行以下命令查看 istio-operator 命令空间的 limit range：

kubectl describe limits -n istio-operator dce-default-limit-range

执行以下命令查看 istio-system 命令空间的 limit range：

kubectl describe limits -n istio-system dce-default-limit-range

istiod 和 ingressgateway 一直处于 ContainerRuning 状态

原因分析：DCE 4.0 所使用的 Kubernetes 版本为 1.18，相对新版服务网格，其版本太低了。

需要手动设置 GlobalMesh 的参数 istio.custom_params.values.global.jwtPolicy: first-party-jwt。

```
#
apiVersion: discovery.mspider.io/v3alpha1
kind: GlobalMesh
metadata:
  creationTimestamp: "2022-10-19T06:23:27Z"
  finalizers:
  - gsc-controller
  generation: 3
  name: dce4
  namespace: mspider-system
  resourceVersion: "65705236"
  uid: 7b857644-d333-4135-8087-5eafcdcf56cb
spec:
  hub: release-ci.daocloud.io/mspider
  mode: DEDICATED
  ownerCluster: dce4-mspider
  ownerConfig:
    controlPlaneParams:
      global.istio_version: 1.15.0
      istio.custom_params.values.global.jwtPolicy: first-party-jwt
      global.mesh_capacity: S50P200
      istio.custom_params.values.global.proxy.resources.limits.cpu: "0"
      istio.custom_params.values.global.proxy.resources.limits.memory: "0"
      istio.custom_params.values.global.proxy.resources.requests.cpu: "0"
      istio.custom_params.values.global.proxy.resources.requests.memory: "0"
      istio.custom_params.values.sidecarInjectorWebhook.enableNamespacesByDefault: "false"
    deployNamespace: istio-system
```

params

!!! tip

DCE 4.0 在接入新版服务网格之前，需提前部署 coreDNS。

服务网格 Release Notes

本页列出服务网格各版本的 Release Notes，便于您了解各版本的演进路径和特性变化。

2023-08-31

v0.19.0

升级

- **新增** userinfo 接口，可以获取当前用户的权限信息。**注意：如果进入网格后，需要同时传入 mesh_id 参数才可以获取到精确的权限信息**。
- **新增** Mcpc Controller 支持虚拟机（自动创建 WorkloadShadow）。
- **新增** 网格创建虚拟机（VM）服务接口。
- **新增** 虚拟机（VM）离线安装包构建和上传脚本。
- **新增** 虚拟机（VM）绑定服务并生成配置接口。
- **新增** Grafana 路由面板添加 Reporter 参数。
- **新增** Mspider 虚拟机代理控制器（mspider-vm-agent）。
- **新增** generator 包，实现 ComponentAnalyzer ,传入 MeshCluster 或 GlobalMesh，即可获取对应组件状态。
- **优化** 网格是否可用升级接口，增加对权限的判断。
- **优化** 重写 gsc-controller 下的 mesh-cluster 中 reconcileComponentsStatus 。
- **升级** 前端版本至 v0.17.0。
- **升级** 支持的网格版本至 1.16.6, 1.17.5, 1.18.2。

修复

- **修复** 网格外的 ServiceEntry 筛选异常问题。
- **修复** 查询虚拟机工作负载实例时 workloadId 正则不兼容虚拟机工作负载 ID（{CLUSTER}-vm-{VM_APP}-{VM_IP}-{VM_NETWORK}）问题。

- **修复** 集群创建东西网关会强制覆盖网关系统默认的南北网关配置。
- **修复** 虚拟机服务自动创建了治理策略，并且修改 label 以后无法自动同步到 WorkloadShadow 问题。
- **修复** 多云互联创建网络分组，名称不可为数字开头的问题。
- **修复** 更新网格时导致自定义配置丢失的问题。
- **修复** 移除 Secret 的 label， 导致 work-api 组件空指针问题。
- **修复** 删除网格检测时包含系统命名空间导致网格删除失败。（会在删除时被我们卸载）。
- **修复** customMeshConfig 无法对 Operator 生效的问题。
- **修复** Operator 版本与实际安装 Istio 的版本不匹配的问题。
- **修复** Mspider 控制面 API Server 和 Work API Server 因为权限被禁用导致无法采集指标问题。
- **修复** Mspider 控制面和 MCPC Ckube 组件没有采集指标问题。
- **修复** OTEL 上报的 trace 数据中，k8s.pod.name 字段为 unknown 的问题。
- **修复** 首次执行虚拟机安装脚本时，存在 warning 告警。

2023-07-28

v0.18.0

升级

- **新增** Istio 版本列表会包含推荐版本。
- **新增** 是否加入互联网络池检测条件 CLUSTER_EXIST_NET_POOLS。
- **新增** 获取集群命名空间列表接口。
- **新增** filter_system_namespaces 支持过滤系统命名空间。
- **新增** 获取集群中工作负载边车注入列表接口。
- **新增** 监控拓扑为工作负载视图功能新增字段：graph_type，当前只支持 SERVICE_SCOPE 与 WORKLOAD_SCOPE，默认为 SERVICE_SCOPE。

- **新增** 删除网络分组检测新增检测条件，是否存在在网络互联池中（NET_EXISTS_NET_POOLS）。
- **新增** 边车工作负载列表支持通过 page.search 搜索工作实例标签 PodLabels。
- **新增** 查询多集群工作负载列表接口 /apis/mspider.io/v3alpha1/meshes/{mesh_id}/clusters/-/sidecar-management/workloads，同时也支持搜索 PodLabels。
- **新增** 服务诊断接口设计。
- **新增** 集群移除网格检测新增是否加入互联网络池检测条件 CLUSTER_EXIST_NET_POOLS 实现。
- **新增** 网格网关新建名称有效性检测接口实现。
- **新增** 获取集群命名空间列表新增字段 filter_system_namespaces 实现。
- **新增** 获取集群中工作负载边车注入列表新增字段 filter_system_namespaces 实现。
- **新增** 监控拓扑支持工作负载纬度。
- **新增** 服务诊断接口实现（可自动修复部分）。
- **新增** 需要手动修复的诊断项检测。
- **新增** 服务修复接口实现。
- **新增** 在 Trace Tags 中，新增 k8s.pod.name 标签，用于标记 Pod 的名称。
- **新增** 平台管理员及工作空间管理员可对工作空间下的网格进行多云网络互联的管理，
- **升级** Istio api 修复 WasmPlugin 无法设置优先级的问题。
- **升级** ckube 到 v1.3.5 解决服务网格列表可能为空的问题。

修复

- **修复** sidecar 和 sidecarResources 不一致的问题。
- **修复** 工作负载列表排序索引混乱问题。

- **修复** 修复 ListClusterNamespace 接口，支持按集群查询 namespace 信息。
- **修复** 从全局管理绑定网格资源可能无法在网格内部显示的问题。
- **修复** 无法修改 portName 为空的端口协议。
- **修复** Workload 缺少 Service 的诊断项的问题。
- **修复** 服务配置同步诊断永远失败的问题。
- **修复** 某些批量操作接口的审计日志不准确的问题。
- **修复** 解除接入的集群，仍然被尝试进行 livez 探测。
- **修复** 专有网格模式下，当命名空间注入状态发生变化时，工作负载的状态没有更新为正确的状态。
- **修复** leaderelection 没有成功同步的问题。
- **修复** 修复了在部分情况下，网格 Pod 总数统计出错的问题。
- **修复** 修正报错信息中的错别字，将 mot 改为 not 。
- **修复** 无法修复某些服务的 Sidecar 注入状态。
- **修复** 非 admin 用户情况下，接口 /apis/mspider.io/v3alpha1/clusters/{name}/components 和 /apis/mspider.io/v3alpha1/clusters/{name} 在 mesh_id 不传时将调用失败。
- **修复** Istio CRD 名称校验，统一为所有操作的名称首字母允许为数字。
- **修复** Graph 接口关于 label_selectors 提供的 reporter 注释有误。
- **修复** 服务诊断修复接口缺少消息体说明。
- **优化** Istio 资源接口中，namespaces 字段的描述。
- **优化** 网格控制面检测流程，忽略控制面集群。
- **优化** 不同角色的不同权限，与最新的权限设计保持一致。
- **优化** 权限设计，将多云网络互联权限从网格管理中独立出来。
- **优化** 统一 global.high_available 参数的含义。
- **优化** 统一 CHART.replicas 的用法，修改默认值为空。

2023-06-29

v0.17.0

升级

- **新增** 新增 mspider.io/mesh-gateway-name Label 规范，用于定义网格网关名称。
- **新增** 命名空间新增字段 injectionStatus，用于列表搜索。
- **新增** 在拓扑查询中，新增字段 label_selectors,条件查询拓扑结果。
- **新增** 边车工作负载信息新增字段 Labels 和 PodLabels。
- **新增** 服务工作负载信息新增字段 Labels 和 PodLabels。
- **新增** 集群信息新增组件信息（components）与 Istio 版本（meshVersion）字段。
- **新增** 集群列表与网格管理集群列表新增 include_components 字段，用于选择是否需要展示集群组件信息，例如 Insight 等外部组件。
- **新增** 为所有需要的接口，增加审计信息。
- **新增** 工作负载注入策略清除策略操作。
- **新增** 命名空间列表新增索引搜索实现。
- **新增** 审计日志能力。
- **新增** 工作负载注入策略清除能力。
- **新增** 集群列表新增组件状态实现。
- **新增** 新增 mspider.io/mesh-gateway-name Label 规范，用于定义网关网关名称。
- **新增** MCPC Controller 实现纳管服务自动注入能力。
- **新增** Ghippo 资源上报功能，按照规范自动创建并且更新 GProductResource 资源。
- **新增** MCPC Controller 新增配置 global.config.enableAutoInitPolicies，用于是否开启纳管服务自动初始化治理策略。
- **新增** MCPC Controller 新增配置 global.config.enableAutoInjectedSidecar，用于是否开启纳管服务自动注入策略。

- **新增** K8s 各版本兼容性测试。
- **优化** 新增 cache 优化查询集群 Insight Agent 状态接口过慢问题。
- **优化** 网格创建时对托管集群检测加强，避免创建冲突网格。
- **优化** 网格网关更新操作将忽略名称（Name）、命名空间（Namespace）、标签（Labels）更新，避免更新触发异常。
- **优化** Kpanda 集群 kubeconfig 同步方式。
- **优化** WorkloadShadow controller watcher 创建逻辑。
- **升级** 支持查询集群与集群组件，允许不传 MeshID，独立获取信息。
- **升级** go package istio.io/istio 到 v0.0.0-20230131034922-50fb2905d9f5 版本。
- **升级** CloudTTY 到 v0.5.3 版本。
- **升级** 前端版本至 v0.15.0 版本。

修复

- **修复** 创建集群东西网关的审计日志描述不准确的问题。
- **修复** 东西网关副本数不生效问题。
- **修复** 网格删除检测边车部分无效。
- **修复** 监控拓扑无法筛选命名空间。
- **修复** 监控拓扑服务错误率数据不准确问题。
- **修复** 修复网格不存在时应该返回 404 错误。
- **修复** Audit 存在 nil pointer 导致 panic 的问题。
- **修复** 审计日志中，Enum 类型的资源类型显示为数字的问题。
- **修复** GRPC 请求带用户认证信息时，行为和 HTTP 请求不一致的情况。
- **修复** 创建专有网格时，Telemetry 资源没有被正确创建。
- **修复** RegProxy watcher 不重建。
- **修复** traffic-lane 插件会给流量打上错误的标签，导致流量泳道无法工作。
- **修复** 删除网格时，没有正确清理集群配置。
- **修复** MCPC Controller 接入不健康集群，运行不正常的问题。

- **修复** 修复网格删除或者移除集群过程中，无法正常移除组件问题。
- **修复** GlobalMesh 和 MeshCluster 一直处于删除状态，无法强制删除的逻辑。
- **修复** 默认不注入的系统命名空间注入状态不正确的问题。
- **修复** 代理 HostedAPIServer 的时候，选择到了没就绪的 Pod，导致网格无法就绪。
- **修复** 网格删除时，Telemetry 删除失败导致网格删除流程失败的问题。因该资源在卸载 Istio / 移除 Hosted ApiServer 时会被清理，所以不需要在删除网格流程中进行删除。
- **修复** HostedAPIServer 因为权限问题无法再 OCP 上安装的问题。
- **修复** helm chart 中的描述信息。

移除

- **移除** 托管网格模式下，istiod-[meshID]-hosted-lb 服务中 443 端口。

2023-05-31

v0.16.2

升级

- **新增** Ckube 按需加载资源。
- **新增** IstioResource 字段：labels 与 annotations，能够更新 Label 与 Annotation。
- **新增** MeshCluster 中 ClusterProvider 同步实现。
- **新增** mspider.io/protected Label 定义，用于网格保护能力。
- **新增** 边车升级支持多工作负载能力，SidecarUpgrader 中 workloads 同时支持 workloadshadow.name 和 deployment.name。
- **新增** 工作负载类型改造成 string 实现。
- **新增** 工作负载相关接口新增字段 localized_name，展示工作负载名称。
- **新增** 工作负载注入策略清除能力

- **新增** 获取全局配置接口 /apis/mspider.io/v3alpha1/settings/global-configs。
- **新增** clusterPhase 字段，用于标记集群的状态（以前在 phase 字段中标记，现在剥离开）。
- **新增** clusterProvider 字段，用于标记集群提供商。
- **新增** 流量泳道 CRD 能力实现。
- **新增** 默认启用 Reg-Proxy 组件。
- **新增** 实现 Service 的 selector 字段输出。
- **新增** 通过给 Namespace 加 Network label 解决未注入 Sidecar 跨集群访问问题。
- **新增** 托管网格 hosted-apiserver 自定义参数配置能力。(该参数只有安装时生效，暂时不支持更新)，(更多参数请参考 helm 参数配置)：
- **新增** 网格控制面组件状态
- **新增** 网格网格查询接口新增 loadBalancerStatus 字段，用于描述实际分配的 LB 地址。
- **新增** 网格组件进度详情接口 /apis/mspider.io/v3alpha1/meshes/{mesh_id}/components-progress。
- **新增** 为控制面组件增加 HPA。
- **新增** 新增获取集群 StorageClass 接口定义。
- **新增** 新增获取集群安装组件接口（目前支持 Insight Agent）。
- **优化** 绑定/解绑工作空间的使用体验。
- **优化** 工作负载相关接口字段 workload_kind 类型从枚举优化为 string。
- **优化** 托管网格情况下，对于集群 k8s 版本检测：除包含工作集群外，也包含对控制面集群的版本检测。
- **升级** 升级 CloudTTY 到 0.5.3 版本。
- **升级** WorkloadShadow controller watcher 创建逻辑。

修复

- **修复** helm chart 中的描述信息
- **修复** RegProxy watcher 不重建。
- **修复** traffic-lane 插件会给流量打上错误的标签，导致流量泳道无法工作。
- **修复** 修复网格删除或者移除集群过程中，无法正常移除组件问题。
- **修复** 修复网格不存在时应该返回 404 错误。
- **修复** CloudShell 权限问题。
- **修复** MeshCluster Status RemotePilotAddress 无效数据没有及时清理。
- **修复** MeshCluster 无法被删除的问题。
- **修复** TrafficLane 的 FailedReason 中缺失内容。
- **修复** TrafficLaneActionsRequest 中的 action 字段缺失。
- **修复** 当存在不健康集群时，网格列表无法展示的问题。
- **修复** 当实例异常时，有效注入实例数不正确。
- **修复** 当一个服务被多个泳道选择时，WasmPlugin 无法创建多个的问题。
- **修复** 服务标签可能存在残留旧的工作负载。
- **修复** 服务列表无法获取工作负载有效的工作负载数，动态获取 ReadyReplicas 的类型解析错误。
- **修复** 工作负载发生变更时，无法将变更的状态同步到对应的 Service。
- **修复** 会对未接入网格的集群一直检查状态的问题。
- **修复** 集群状态不可搜索。
- **修复** 没有边车时，网格无法移除集群的情况。
- **修复** 网格名称正则表达式，不允许数字开头。
- **修复** 网格状态显示不正确，没有边车时有时仍然显示状态为正常。
- **修复** 修复网格网格的自动注入的模板不生效问题。
- **修复** 由于集群缺少默认值，导致非 Admin 用户无法获取流量拓扑。
- **修复** 自动注入服务策略空指针异常。

2023-04-27

v0.15.0

升级

- **新增** 引入 d2 作为绘图工具
- **新增** 一个新的 wasm 插件，用于根据 trace id，给请求加上不同的 header。
- **新增** 网格配置托管 Istiod 的 LoadBalancer Annotations 实现
- **新增** 网格网关配置服务 Annotations 实现
- **新增** 增加网格字段 load_balancer_annotations，支持自定义负载均衡 Annotations
- **新增** 在 mspider-api 中，手动执行 pipeline，设置 SYNC_OPENAPI_DOCS 为 key，既可触发上传文档站（提 PR）
- **新增** MCPC Controller 感知到 Service 的中存在 mspider.io/managed 的标签时，将触发自动创建该服务对应的治理策略。
- **新增** MCPC Controller 多工作负载类型支持。
- **新增** 健康检查功能，当某个网格 APIServer 代理出现无法连通的情况时，自动重建该代理， 防止 PortForward 自身逻辑不可靠（可能和 Istio Sidecar 有关）

修复

- **修复** 未兼容 grpcgateway-accept-language（等价与 HTTP 的 Accept-Language）Header，导致无法正确切换中英文模式模式。 当前同时兼容 Accept 与 Accept-Language 两种模式
- **修复** 同步 OpenAPI 时，由于 shadow clone 代码导致 upstream 无法 push 的问题
- **修复** 无法更新带 . 的 Istio 资源
- **修复** 在 1.17.1 版本中，istio-proxy 无法正常启动的问题
- **修复** ingress gateway 缺少 name 导致 merge 失败无法部署的问题
- **修复** 服务地址无法搜索的问题

- **修复** 无法更新带 . 的 Istio 资源 的问题
- **修复** 之前 controller 内存泄漏的问题。
- **修复** add/delete cluster 有时没有正确触发的逻辑。
- **修复** 了托管模式下 istio-system 可能被误删的情况。
- **修复** ingress gateway 缺少 name 导致 merge 失败无法部署的问题。
- **修复** 在开启全局注入情况下，更新网格可能导致 istio-operator pod 被注入，从而使网格创建失败的问题
- **修复** 服务与 WorkloadShadow 关联的两个部分没有清理逻辑，导致服务被错误绑定在工作负载上
- **修复** 一个导致虚拟集群被同步到 Mspider 导致接入失败的问题
- **优化** controller 命名空间、服务资源处理逻辑，减少频繁触发 workloadShadow 资源更新。
- **优化** workloadShadow 资源频繁获取/更新的问题，现在只对某些发生特定改变的资源进行 reconcile。
- **优化** 减少 pod 变更不断更新 WorkloadShadow
- **修复** relok8s 中 wasm 插件镜像地址拼写错误的问题。
- **修复** TrafficLane 默认 repository 错误。
- **优化** Helm 镜像渲染模板。镜像结构拆分为三个部分：registry/repository:tag

移除

- **移除** 同步全局集群到网格的逻辑
- **弃用** 弃用 Deployment Controller 的逻辑
- **弃用** ui.version 参数，改为 ui.image.tag 参数设定前端版本

2023-03-31

v0.14.3

升级

- **升级** 前端版本至 v0.12.2

修复

- **修复** 无法更新带 . 的 Istio 资源
- **修复** 在 1.17.1 版本中，istio-proxy 无法正常启动的问题
- **修复** ingress gateway 缺少 name 导致 merge 失败无法部署的问题。

2023-03-30

v0.14.0

新功能

- **新增** CloudShell 相关接口定义
- **新增** 更新服务的标签实现
- **新增** 服务列表和详情会返回 labels
- **新增** CloudShell 相关实现
- **新增** 服务列表支持对服务标签的查询
- **新增** 一个新的 API，用于更新服务的标签
- **新增** istio 1.17.1 支持
- **新增** 一个新的 etcd 高可用方案
- **新增** 场景化测试框架，用于测试场景化的功能
- **新增** 选择不同网格规模时，自动调整组件资源配置
- **新增** 自定义角色的实现，支持自定义角色的创建、更新、删除、绑定、解绑等操作

优化

- **优化** mcpc controller 启动逻辑，避免出现工作集群没有正确注册的情况
- **优化** WorkloadShadow 清理逻辑：由定时触发改造事件触发：控制器启动时、检测到工作集群发生变化时； WorkloadShadow 发生变化时，自健康检测，对应工作负载不存在时，触发清理逻辑
- **优化** mcpc controller 启动逻辑，避免出现工作集群没有正确注册的情况
- **升级** Insight api 升级到 v0.14.7 版本
- **升级** ckube 支持 labels 的复杂条件查询

- **移除** 移除 Helm 升级时间限制

修复

- **修复** 东西网关没 Ready 时界面无显示的问题
- **修复** 多云互联会自动注册东西网关 LB IP，可能导致内部网络异常（移除东西网关实例 label：topology.istio.io/network，该标签会将东西网关自动注册）
- **修复** 携带东西网关的集群迁移会出错问题（无法修改实例的 label，如果需要修改组件 label 只能删除组件再重建）
- **修复** 了一个导致 Mesh 绑定 workspace 服务失败（界面显示成功）的问题
- **修复** 由于异常导致虚拟集群中存在游离的命名空间，在启动 mcpc-controller 时增加自检并且清除行为
- **修复** 由于 controller 更新网格，导致 api 下发网格配置失败
- **修复** 创建托管网格时，ServicePort 的 TargetPort 未正确设置的问题
- **修复** GlobalMesh.Status.MeshVersion 错误覆盖问题
- **修复** mcpc-controller 无法开启 debug 模式问题
- **修复** mcpc-controller 无法触发集群删除事件
- **修复** 删除 Mesh 再重建同名的 Mesh 时，会导致 Mesh 无法正常创建的问题（hosted proxy 无法正常更新）
- **修复** mcpc controller 在某些情况下没有正确修改 istiod-remote 的 service 的问题

2023-02-28

v0.13.1

新功能

- **新增** 多云网络互联管理及相关接口
- **新增** 网格全局配置能力（网格系统配置、全局流量治理能力、全局边车注入设置）增强

- **新增** 支持 zookeeper 代理

优化

- **优化** Namespace 控制器加上 Cache，当集群中 Namespace 过多时会造成太多的重复的 Get 请求
- **优化** 减少正常模式下 ckube 的日志输出

修复

- **修复** 由容器管理平台移除集群导致的网格相关逻辑错误
- **修复** RegProxy 注入 Sidecar 会导致 Nacos 无法注册的问题
- **修复** Spring Cloud 服务识别错误的问题
- **修复** 没有同步容器管理平台 Cluster 状态信息
- **修复** 流量透传策略 – 流量筛选 IP 段配置未生效
- **修复** 流量透传策略 – 流量透传策略无法删除
- **修复** 获取网格配置时，由于配置路径不存在导致返回错误

2022-12-29

v0.12.0

新功能

- **新增** 流量透传功能的接口实现
- **新增** 对 istio 1.15.4、1.16.1 的支持

优化

- **优化** 命名空间边车管理页面中增加“未设置”边车策略，避免命名空间层面的边车策略对工作负载层面产生连带影响；
- **优化** 发布流水线参数
- **优化** 边车注入、边车资源限制逻辑、避免出现不同步现象

修复

- **修复** 网格升级后部分组件未更新的问题
- **修复** 网格移除后部分资源未清除的问题 – 边车资源同步不正确，改为通过 Pod 中的 istio-proxy 容器的实际边车资源

- **修复** 托管集群无法作为工作集群
- **修复** 当边车注入实例为 0/0 时注入情况显示不正确
- **修复** 取消边车注入后仍然显示边车相关信息

2022-11-30

v0.11.1

新功能

- **新增** 密钥管理相关 API
- **新增** 在虚拟服务、目标规则、网关规则列表
- **新增**治理策略标签及相关筛选能力
- **新增** 网格中集群健康状态检查能力
- **新增** otel sdk 接入
- **新增** secret 的多个接口实现
- **新增** 对 istio 1.15.3 的支持

优化

- **优化** 重工作负载边车注入接口，防止出现问题
- **优化** 网格服务监控面板默认展开
- **优化** 禁用网格全局监控面板子链接跳转，避免出现界面混乱现象
- **优化** 网格控制面处理流程，避免更新已经更新过的 object 导致冲突
- **优化** 网格的集群接入、移除逻辑

修复

- **修复** 当控制面 MSpider 升级之后，mcpc 等组件未更新的问题
- **修复** 获取集群资源错误
- **修复** 获取集群命名空间列表接口集群名称不存在时响应码为 200 情况
- **修复** 网格升级流程中前置条件判断有误的问题
- **修复** 网格升级未执行 k8s 版本限制的问题
- **修复** 边车管理接口工作负载边车资源配置失效的问题
- **修复** 因网格内集群的监控检测失败导致集群无法移除的问题

- **修复** 数据面的镜像没有被打包到 DCE 5.0 的离线包中
- **修复** Ckube 无法自动更新配置的问题

常见问题

1. 服务网格中的 Istio 是否可以单独安装?

   可以，但不建议。用户单独安装的 Istio 控制面组件可通过外接网格的方式接入服务网格，但相比于其他模式，只提供基础的 Istio 具备的功能。

2. 服务网格中的 Istio 组件版本是多少?

   有多个版本，用户可以在创建网格时选择。

3. 服务网格的 Istio 是否可以升级?

   可以，支持用户自主升级。

4. 网格中注入的边车是否可以升级?

   可以，同时支持边车热升级。

5. 第 4 代服务网格（DSM）是否可升级到第 5 代服务网格?

   不可以升级，但可以以外接网格的形式接入服务网格。建议采用重新安装的方式，获得最好的扩展性和后期维护。

6. 如果设置了不同的命名空间边车注入策略和工作负载边车注入策略，实际执行效果是什么样?

   将按以下规则顺序匹配，匹配到有效规则将不再执行之后的规则：

   - 如果二者有一个设置为禁用，则禁用边车注入
   - 如果二者有一个设置为启用，则启用边车注入
   - 如果二者都没有设置，将执行网格全局边车注入策略（values.sidecarInjectorWebhook.enableNamespacesByDefault）

详细情况可参考安装边车

7. 传统微服务加入服务网格有什么不同?

   传统微服务加入服务网格只需要将其对应的工作负载注入边车，以便网格能够对其进行管理。 基于不同的微服务框架以及网格的无侵入的特性，原先框架的治理策略等可能会有所改变，但无需用户自行实现。 当接入网格后，传统微服务即可获得网格的全部能力，包括流量治理、安全、可观测性等等。

什么是服务网格

服务网格是基于 Istio 开源技术构建的面向云原生应用的下一代服务网格。

服务网格是一种具备高性能、高易用性的全托管服务网格产品，通过提供完整的非侵入式的微服务治理方案，能够统一治理多云多集群的复杂环境， 以基础设施的方式为用户提供服务流量治理、安全性治理、服务流量监控、以及传统微服务（SpringCloud、Dubbo）接入。

DCE 5.0 的服务网格兼容社区原生 Istio 开源服务网格，提供原生 Istio 接入管理能力。在较高的层次上，服务网格有助于降低服务治理的复杂性，减轻开发运维团队的压力。

服务网格作为 DCE 5.0 产品的体系一员，无缝对接容器管理平台，可以为用户提供开箱即用的上手体验， 并作为基础设施为微服务引擎提供容器微服务治理支持，方便用户通过单一平台对各类微服务系统做统一管理。

产品优势

DCE 5.0 服务网格相比其他产品具备以下优势：

- 简单易用

  无需修改任何业务代码，也无需手动安装代理，只需开启服务网格功能，即可体验丰富的无侵入服务治理能力。

- 策略化的智能路由与弹性流量管理

  支持为服务配置负载均衡、服务路由、故障注入、离群检测等治理规则。结合一站式治理系统；提供实时的、可视化的微服务流量管理； 支持无侵入智能流量治理，应用无需任何改造，即可进行动态的智能路由和弹性流量管理。

  - 权重、内容、TCP/IP 等路由规则。

- HTTP 会话保持，满足业务处理持续性诉求。
- 限流、离群检测，实现服务间链路稳定、可靠。
- 网络长连接管理降低资源损耗，提升网络吞吐量。
- 服务安全认证：认证、鉴权、审计等，提供服务安全保障基石。

- 图形化应用全景拓扑，流量治理可视化

  服务网格提供了可视化的流量监控，包括链路信息、服务异常响应、超长响应时延等信息，通过图表以及拓扑等多样化形式全面展现业务运行情况。

  服务网格可以结合应用运维管理、应用性能管理服务，提供详细的微服务级流量监控、异常响应流量报告以及调用链信息，能够更快速、更准确的定位问题。

- 性能增强，可靠性增强

  服务网格控制面和数据面基于社区版本的基础上进行更可靠的加固和性能优化。

- 多云多集群、多基础设施

  提供免运维的托管控制面，提供多云多集群的全局统一的服务治理、安全和服务运行监控能力，还提供对容器和虚拟机（VM） 等多种基础设施的统一服务发现和管理。

- 协议扩展

  扩展 Dubbo 协议的支持。

- 传统 SDK 集成

  提供 Spring Cloud、Dubbo 等传统微服务 SDK 的集成解决方案，传统微服务 SDK 开发的业务无需大量代码改造即可快速迁移到云原生容网格运行环境上运行。

学习路径

服务网格的学习路径如下：

```
flowchart TD

    install([安装部署])
    install --> mesh[创建网格]
        subgraph mesh[创建网格]
            managed[托管网格]
            private[专有网格]
            external[外接网格]

        end

    mesh --> cluster[纳管集群]

    cluster --> inject[注入边车]

        subgraph inject[注入边车]
            namespace[命名空间注入]
            workload[工作负载注入]
        end


    inject -.-> service[服务管理]
    inject -.-> traffic[流量治理]
    inject -.-> security[安全治理]
    inject -.-> sidecar[边车管理]
    inject -.-> watch[流量监控]
    inject -.-> gateway[网格网关]
    inject -.-> upgrade[版本升级]
```

```
    service -.-> entry[服务条目<br>一键修复]
    traffic -.-> virtual[虚拟服务<br>目标规则<br>网关规则]
    security -.-> peer[对等身份认证<br>请求身份认证<br>授权策略]
    sidecar -.-> sidecarm[命名空间边车管理<br>工作负载边车管理<br>边车流量透传]
    watch -.-> watch2[流量监控<br>流量拓扑]
    upgrade -.-> upgrade1[Istio 升级<br>边车升级]

    classDef plain fill:#ddd,stroke:#fff,stroke-width:1px,color:#000;
    classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff;
    classDef cluster fill:#fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

    class mesh plain
    class install,service,gateway,traffic,watch,upgrade,security,entry,virtual,peer,cluster,sidecar,sidecarm,watch2,managed,private,external,namespace,workload,upgrade1 cluster

    click install "https://docs.daocloud.io/mspider/install/install/"
    click managed "https://docs.daocloud.io/mspider/user-guide/service-mesh/"
    click private "https://docs.daocloud.io/mspider/user-guide/service-mesh/"
    click external "https://docs.daocloud.io/mspider/user-guide/service-mesh/external-mesh/"
    click cluster "https://docs.daocloud.io/mspider/user-guide/cluster-management/join-clus/"
    click global "https://docs.daocloud.io/mspider/user-guide/sidecar-management/global-sidecar/"
    click namespace "https://docs.daocloud.io/mspider/user-guide/sidecar-management/ns-sidecar/"
    click workload "https://docs.daocloud.io/mspider/user-guide/sidecar-manage
```

ment/workload-sidecar/"

click gateway "https://docs.daocloud.io/mspider/user-guide/gateway-instance/create/"

click service "https://docs.daocloud.io/mspider/user-guide/service-list/"

click traffic "https://docs.daocloud.io/mspider/user-guide/traffic-governance/"

click security "https://docs.daocloud.io/mspider/user-guide/security/"

click watch "https://docs.daocloud.io/mspider/user-guide/traffic-monitor/"

click upgrade "https://docs.daocloud.io/mspider/user-guide/upgrade/istio-update/"

click entry "https://docs.daocloud.io/mspider/user-guide/service-list/service-entry/"

click virtual "https://docs.daocloud.io/mspider/user-guide/traffic-governance/virtual-service/"

click peer "https://docs.daocloud.io/mspider/user-guide/security/peer/"

click sidecar "https://docs.daocloud.io/mspider/user-guide/sidecar-management/ns-sidecar/"

click sidecarm "https://docs.daocloud.io/mspider/user-guide/sidecar-management/passthrough/"

click watch2 "https://docs.daocloud.io/mspider/user-guide/traffic-monitor/conn-topo/"

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

适用场景

服务网格适用的场景如下：

服务流量治理

流量治理是一个非常宽泛的话题，例如：

- 动态修改服务间访问的负载均衡策略，如根据某个请求特征做会话保持。
- 同一个服务有两个版本在线，将一部分流量切到某个版本上。
- 对服务进行保护，例如限制并发连接数、限制请求数、隔离故障服务实例等。
- 动态修改服务中的内容，或者模拟服务运行故障等。

服务网格可以提供非侵入的流量治理能力，无需修改任何业务代码就能实现这些服务治理功能。 根据服务的协议，提供策略化、场景化的网络连接管理。可以根据需要对特定服务的特定端口配置不同的治理规则。

场景优势

- 重试

  服务访问失败自动重试，从而提高总体访问成功率和质量。支持配置 HTTP 请求重试次数、重试超时时间和重试条件。

- 超时

  服务访问超时自动处理，快速失败，从而避免资源锁定和请求卡顿。支持配置 HTTP 请求超时时间。

- 连接池

  通过连接池管理，可以对四层协议配置 TCP 的最大连接数、连接超时时间、最大无响应次数、最短空闲时间和健康检查间隔； 可以对七层协议配

置 HTTP 最大请求数、最大重试次数、最大等待请求数、每连接最大请求数、连接最大空闲时间，从而防止一个服务的失败级联影响到整个应用。

- 离群检测

  通过离群检测配置实例被驱逐前的连续错误次数、驱逐间隔时长、最小驱逐时间、最大驱逐比例等参数，从而定期考察被访问的服务实例的工作情况。如果连续出现访问异常，则将服务实例标记为异常并进行隔离，在一段时间内不为其分配流量。 过一段时间后，被隔离的服务实例会再次被解除隔离，尝试处理请求。如果还不正常，则被隔离更长的时间。从而实现异常服务实例的故障隔离和自动故障恢复。

- 负载均衡

  配置各种负载均衡策略，如随机、轮询、最少连接，还可以配置一致性哈希将流量转发到特定的服务实例上。

- HTTP Header

  灵活增加、修改和删除指定 HTTP Header，包括将 HTTP 请求转发到目标服务之前对 Header 进行操作。 还可以在将 HTTP 响应回复给客户端前，对 Header 进行操作，以非侵入方式管理请求内容。

- 故障注入

  通过对选定的服务注入中断故障、延时故障来构造故障场景，无需修改代码即可进行故障测试。

端到端的透明安全

众所周知，将传统的单体应用拆分为一个个微服务固然带来了各种好处，例如：更好的灵活性、可扩缩性、重用性，但微服务也同样面临着特殊的安全需求：

- 为了抵御中间人攻击，需要用到流量加密。
- 为了提供灵活的服务访问控制，需要用到 TLS 和细粒度访问策略。

- 为了决定哪些人在哪些时间可以做哪些事，需要用到审计工具。

面对这些需求服务网格提供全面的安全解决方案，包括身份验证策略、透明的 TLS 加密以及授权和审计工具。

场景优势

- 非侵入安全

  服务网格是以一种安全基础设施的方式向用户提供透明的安全能力，让不涉及安全问题的代码安全运行，让不太懂安全的人可以开发和运维安全的服务， 不用修改业务代码就能提供服务访问安全。应用服务网格提供了一个透明的分布式安全层，并提供了底层安全的通信通道，管理服务通信的认证、授权和加密， 提供 Pod 到 Pod、服务到服务的通信安全。开发人员在这个安全基础设施层上只需专注于应用程序级别的安全性。

- 多集群安全

  在多集群场景下服务网格提供了全局的服务访问安全。多个集群的网格共享一套根证书，给数据面的服务实例分发密钥和证书对，并定期替换密钥证书， 根据需要撤销密钥证书。在服务间访问时，网格的数据面代理就会代理本地服务和对端进行双向认证、通道加密。 这里的双向认证的服务双方可以来自两个不同的集群，从而做到跨集群的透明的端到端双向认证。

- 细粒度授权

  在认证的基础上，就可以进行服务间的访问授权管理，可以控制某个服务，或者服务的一个特定接口进行授权管理。 如只开放给特定的一个 Namespace 下的服务，或者开放给某个特定的服务。 源服务和目标服务可以在不同的集群，甚至源服务的不同实例在不同的集群，目标服务的不同实例在不同的集群。

服务运行监控

运营容器化的基础设施带来了一系列新的挑战。我们需要增强容器、评估 API 端点的性能以及识别出基础设施中的有害部分。 服务网格可在不修改代码的情况下实现 API 增强，并且不会带来服务延迟。

服务网格可以为网格内的所有服务通信进行遥测，这种遥测技术提供了服务行为的可观察性，有助于运营商对其应用程序进行故障排除、维护和优化， 而不会给服务开发人员带来任何额外负担。通过服务网格，运营商可以全面了解被监控的服务如何与其他服务以及组件本身进行交互。

场景优势

- 非侵入监控数据采集

  在复杂应用的场景下，服务间的访问拓扑、调用链、监控等都是服务访问异常时进行定位定界的必要手段。 服务网格技术的一项重要能力就是以应用非侵入的方式提供这些监控数据的采集，用户只需关注自己的业务开发，无需额外关注监控数据的生成。

- 丰富性能监控能力

  基于网格生成服务访问数据，集成各种不同的性能监控服务，提供跨集群智能的服务运行管理。 包括跨集群的服务调用链、服务访问拓扑和服务运行健康状态、通过跨集群的全局视图来关联服务间的访问状况等。

功能总览

此处介绍服务网格支持的功能。

流量治理

- 七层连接池管理

  支持配置 HTTP 最大请求数、最大重试次数、最大等待请求数、每次连接最大请求数以及连接最长空闲时间。

- 四层连接池管理

  支持配置 TCP 最大连接数、连接超时时间、最大无响应次数、最短空闲时间以及健康检查间隔。

- 离群检测

  支持配置服务离群检测规则，包括实例被驱逐前的连续错误数、检查周期、基础隔离时间以及最大隔离实例比例。

- 重试

  支持配置 HTTP 重试次数、重试超时时间以及重试条件。

- 超时

  支持配置 HTTP 请求超时时间。

- 负载均衡

  支持配置随机调度、轮询调度、最少连接和一致性哈希多种负载均衡算法。

- HTTP Header

可以灵活添加、修改和删除指定 HTTP Header，包括将 HTTP 请求转发到目标服务之前对 Header 的操作，以及将 HTTP 响应回复给客户端前对 Header 的操作。

- 故障注入

  支持配置延时故障和中断故障。

安全

- 透明双向认证

  支持界面配置服务间的双向认证。

- 细粒度访问授权

  支持通过界面配置服务间的访问授权（后台 API 可以配置 Namespace 级别授权，授权将会给一个特定的接口）。

可观测性

- 流量拓扑

  支持查看网格应用流量拓扑，了解服务间依赖关系。

- 服务运行监控

  支持查看服务访问信息，包括服务和服务各个版本的 QPS 和延时等指标。

- 访问日志

  支持收集和检索服务的访问日志。

- 调用链

  支持非侵入调用链埋点，并可以通过检索调用链数据进行问题定界定位。

多集群模式

- 多集群配置统一管理

支持网格下多集群的网格配置、集群配置管理；支持不同集群不同粒度的边车注入策略，以及支持跨集群流量策略等对数据面配置的统一管理。

- 可扩展性

  支持一键接入、移除集群

网格数据面微服务框架

- Spring Cloud

  支持 Spring Cloud SDK 开发的服务无侵入式的接入网格，并统一管理。

- Dubbo

  支持 Dubbo SDK 开发的服务无侵入式的接入网格，并统一管理。

兼容性和扩展

- 版本兼容

  API 完全兼容通用服务网格。

- 插件支持

  支持 Tracing、Prometheus、Kiali、Grafana。

服务网格应用接入规范

为了您的应用能够接入网格，并体验到所有网格的能力，我们需要对应用做一些规范。 本文档将介绍服务网格的基本原理和结构，以及应用接入服务网格的规范和建议。

当然，如果您的应用在接入网格之后，没有按照预期运行，也可以按照此文档进行排查，确认应用配置是否符合规范。

引言

服务网格概述

服务网格（Service Mesh）是一种基础设施层，用于处理服务到服务之间的通信。 在一个微服务架构中，服务网格帮助构成其复杂拓扑，以便它可以更好地运行、观察、保护和配置微服务。 服务网格通常用于大规模的企业应用，其中包含大量的微服务。

我们的服务网格产品，基于 Istio 开源项目，提供了一套完整的服务网格解决方案， 包括对多云应用的支持、对多种开发语言的支持、对多种部署方式的支持、对多种应用场景的支持等。

规范目的和适用范围

此《服务网格应用接入规范》的目的在于提供一个具体的指南，以帮助开发者和团队理解服务网格的核心概念和工作流程， 并正确地将应用接入服务网格，确保服务稳定、安全、高效运行。

本规范主要适用于有一定微服务基础知识，希望进一步理解和使用服务网格技术的技术人员。 无论你是新接触服务网格，还是已经有一定了解，希望深化对接入过程的理解，本规范都会为你提供详细的指南和实用建议。

服务网格基本原理与结构

服务网格是一种用于处理服务到服务之间通信的基础设施层。它负责在微服务架构中的各个服务之间进行可靠、快速和安全的网络请求传输。 服务网格通过提供服务发现、负载均衡、故障恢复和服务访问策略等功能，帮助开发者专注于构建业务逻辑，而不必关心服务间通信的细节。

在服务网格架构中，网络通信被抽象化为服务间的请求和响应。这些请求和响应通过服务网格在服务间进行路由， 并由网格负责处理重试、超时、断路、身份验证等问题。开发者只需要声明他们希望如何处理请求和响应，服务网格则负责实现这些策略。

为了实现这些功能，服务网格通常由两部分组成：数据平面和控制平面。

- 数据平面由一组网络代理组成，它们在应用程序的每个服务实例中作为边车运行，处理进出该实例的网络通信。
- 控制平面管理和配置数据平面的代理，以及提供服务网格的管理功能，如流量管理、策略配置、服务发现、认证、监控和报告等。

因为引入了边车，所以我们需要对接入网格的应用做一些规范，确保应用能够正常运行。

服务接入规范与建议

接入流程概述

在将您的应用接入服务网格的过程中，需要注意以下几个步骤以确保接入的顺利进行：

- **了解服务网格**：在应用接入之前，开发者需要了解服务网格的基本概念、核心组件以及运作方式。 通过对服务网格的理解，可以帮助开发者更好地整合应用。
- **评估应用的适配性**：评估您的应用是否适合在服务网格环境下运行，例如是否兼容服务网格的通信协议、是否能在容器化环境下运行等。 此外，开

发者也需要理解服务网格带来的变化，例如流量的路由、负载均衡等都由服务网格控制。

- **应用的修改与优化**：根据服务网格的规范，可能需要对应用进行一些修改，如将健康检查、日志、跟踪等信息暴露出来， 以便服务网格进行监控和管理。同时，也要考虑如何处理服务网格在应用外部处理的功能，例如重试、超时控制等。
- **接入服务网格**：根据服务网格提供的接口或 SDK，进行代码级别的接入，这通常包括导入相关库， 初始化服务，设置服务发现、负载均衡策略等。
- **测试与优化**：在应用接入服务网格后，需要进行充分的测试，以确保应用在服务网格环境下的表现和性能。 此外，根据测试结果，可能需要对应用或服务网格的配置进行进一步的优化。

应用运行环境要求

由于在同一个 Pod 实例中引入了边车，所以在接入服务网格时，需要对应用的运行环境做一些要求，具体如下：

要求项	是否必须	要求值	说明
端口监听	是	应用不可以监听以下端口：	由于服务网格的数据平面需要监听这些端口，所以应用不可以监听这些端口。
Pod 标签	是	Pod app 标签的值必须与所关联的 Service 名称相同，注意不是如 Deploment 自己的标签，而是 Pod（通常位于 .spec.template.metadata.labels）的标签。	观测、流量泳道等功能都是基于 Pod 的 app 标签进行的，所以必须保证 app 标签的值与 Service 名称相同。

要求项	是否必须	要求值	说明
运行用户 uid	是	不能以 uid 为 1337 的用户运行	由于服务网格的数据平面需要以 uid 为 1337 的用户运行，边车的流量拦截不会处理这个用户的流量，所以应用不能以 uid 为 1337 的用户运行。
HostNetwork	是	Pod 不能以 HostNetwork 模式运行	HostNetwork 模式不受服务网格支持
DNSPolicy	是	Pod 的 DNSPolicy 建议为 ClusterFirst，且 ndots 建议设置成 5。	由于边车需要与控制面通讯，需要依赖 DNS 解析控制面地址
包含 Envoy 进程的应用 base-id	否	如果业务应用运行了 Envoy，需要增加 --base-id XXX 参数	因为边车使用了 Envoy，如果不加 --base-id 参数，两个 Envoy 无法共存

应用通讯规范

要求项	是否必须	要求值	说明
服务	否	服务之间访问，需要以服务	由于服务网格的数据平面需要以服务名的方式

要求项	要求值	说明
多集群访问方式	名的方式进行访问，或者 ClusterIP，不可以直接使用 Pod IP 或者 NodePort。	策略匹配，所以应用不可以直接使用 Pod IP 或者 NodePort，否则可能造成策略失效或者无法访问。
端口协议	正确配置 Service 的端口的协议，且在多集群模式下，确保服务在所有集群的配置保持一致（即同一个 ns 下的同名服务（Service），Service.spec 应当保持一致）。	可在 DCE 5.0 服务网格界面（服务管理-服务列表-地址信息）中修改具体端口的协议，或者按照 Istio 文档进行配置。错误的配置可能引起访问问题或者策略问题。

对接链路追踪

我们目前使用 OpenTelemetry 作为链路追踪的标准，并默认会将链路信息上报到可观测模块，如果您希望使用链路追踪功能，需要在应用中引入 OpenTelemetry 的 SDK，并在应用中配置好链路追踪的参数，具体可以参考 OpenTelemetry 官方文档。

或者如果想简单使用，可以简单的透传 W3C 标准的 TraceContext，详细可以参考 W3C TraceContext。 即： 将 request 请求中的 traceparent 和 tracestate 请求头发送到需要被请求的消息头中即可。

推荐配置

为了您的应用能够更好的运行在服务网格环境下，我们推荐您在应用中做如下配置：

健康检查

建议给您的应用配置健康检查，以便服务网格能够更好的监控您的应用的健康状态。具体可以参考 Kubernetes 官方文档。

重试机制

由于 Sidecar 机制的存在，可能存在 Sidecar 在服务之后启动的情况。在此期间， 您的服务可能无法访问外部请求，所以我们建议您在应用中配置重试机制，以便在 Sidecar 启动后，能够自动重试请求。

使用 HTTP 或者 GRPC 作为应用通讯协议

目前我们的服务网格能够非常好的支持 HTTP 和 GRPC 协议，建议您的应用也使用这两种协议， 如果您的应用是其他协议，如 BRPC，则会被降级成 TCP 协议，可能会导致一些功能无法使用。

默认情况下，服务网格会对服务间访问进行 TLS 加密（mTLS），所以我们也不建议在服务间的访问使用 HTTPS 协议（会被当做 TCP 协议处理）。

所以我们建议您的应用使用 HTTP 或者 GRPC 协议。

避免使用外置注册中心机制

虽然我们的产品支持使用注册中心的服务接入服务网格，但这并不是我们推荐的方式， 我们建议您的应用直接使用服务网格（Kubernetes 的 Service）的服务发现机制，以便能够更好的使用服务网格的功能。

也就是说，我们不推荐使用 Spring Cloud 这样的框架开发新的应用，但是这类存量的应用可以被我们产品支持。 如果是 Java 应用，我们推荐使用 Spring Boot 的方式开发应用。

服务对外暴露

默认情况下，接入服务网格的应用我们不推荐使用 NodePort 或者 LoadBalancer 等方式直接对外提供服务，这将导致部分服务网格的功能无法使用。

在目前的版本中，我们有两种方式可以对外提供服务：

1. 使用网格网关进行对外暴露。
2. 使用微服务引擎的云原生网关进行对外暴露。

对于微服务引擎的云原生网关使用方法，请参阅相关文档。

对于网格网关的使用方法，也请参阅相关文档。值得注意的是，我们建议将需要对外服务的 VirtualService 独立出来， 不要和其他（如 Service 同名的 VirtualService）的 VirtualService 混合在一起，以便能够更好的管理。

服务网格的控制面组件

组件名称	位置	描述	默认资源设置
mspider-ui	全局管理集群	服务网格界面	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
mspider-ckube	全局管理集群	Kubernetes API Server 的加速组件，用于调用全局集群相关的资源	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
mspider-ckube-remote	全局管理集群	用于调用远程集群的 Kubernetes， 聚合多集群资源，并且加速	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
mspider-gsc-controller	全局管理集群	服务网格管理组件，用于网格创建，网格配置等网格控制面生命周期管理，	requests: CPU: 未设置；内存: 未

组件名称	位置	描述	默认资源设置
		以及权限管理等 Mspider 控制面能力	设置 limits: CPU: 未设置；内存: 未设置
mspider-api-service	全局管理集群	为 Mspider 后台 API 交互，等控制行为提供接口	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
托管网格			
istiod-{meshID}-hosted	控制面集群	用于托管网格的策略管理	requests: CPU: 100m；内存: 100m limits: CPU: 未设置；内存: 未设置
mspider-mcpc-ckube-remote	控制面集群	调用远程的网格工作集群，加速并且聚合多集群资源	requests: CPU: 100m；内存: 50mlimits: CPU: 500m；内存: 500m
mspider-mcpc-mcpc-	控制面集	聚合网格多集群相关数据	requests:

组件名称	位置	描述	默认资源设置
controller	群	面信息	CPU: 100m；内存: 0 limits: CPU: 300m；内存: 1.56G
{meshID}-hosted-apiserver	控制面集群	托管控制面虚拟集群 API Server	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
{meshID}-etcd	控制面集群	托管控制面虚拟集群 etcd，用于托管网格的策略存储	requests: CPU: 未设置；内存: 未设置 limits: CPU: 未设置；内存: 未设置
istiod	工作集群	主要用于所在集群的边车生命周期管理	requests: CPU: 100；内存: 100 limits: CPU: 未设置；内存: 未设置

专有网格

组件名称	位置	描述	默认资源设置
istiod		用于策略创建、下发、边车生命周期管理的工作	requests: CPU: 100；内存: 100 limits: CPU: 未设置；内存: 未设置
mspider-mcpc-ckube-remote	工作集群	调用远程的网格工作集群	requests: CPU: 100m；内存: 50m limits: CPU: 500m；内存: 500m
mspider-mcpc-mcpc-controller	工作集群	收集集群数据面信息	requests: CPU: 100m；内存: 0 limits: CPU: 300m；内存: 1.56G
外接网格			
mspider-mcpc-ckube-remote	工作集群	调用远程的网格工作集群	requests: CPU: 100m；内存: 50m limits: CPU: 500m；内存:

组件名称	位置	描述	默认资源设置
			500m
mspider-mcpc-mcpc-controller	工作集群	收集集群数据面信息	requests: CPU: 100m；内存: 0 limits: CPU: 300m；内存: 1.56G

服务网格的各控制面组件预设资源设置如上表所示，用户可以在容器管理模块查找相应的工作负载，为工作负载自定义 CPU、内存资源。

系统架构

DCE 5.0 服务网格产品支持多种类型的网格实例管理能力，支持托管多集群网格进行云原生微服务治理能力；用户可将不同来源的集群接入托管网格中，进行统一的流量治理和安全治理等。

从整体架构上讲，服务网格产品可分为三个面：网格全局管理面、网格控制面、网格数据面



系统架构

- 网格全局管理面

  运行于独立的控制面集群 Global Service Cluster（以下简称 GSC），用于对多个服务网格实例进行统一管理，提供友好的管理与控制能力。 网格全局管理面提供了提供了友好的界面化操作和 YAML 原生资源编辑能力，统一的网格实例生命周期管理、权限隔离等。同时，也负责与 DCE 5.0 其它模块的对接的工作，以将减少其它模块的适配成本。 注意，网格全局管理面仅管理网格实例本身，不参与具体的服务治理事务，因此并不包含 Istio 相关组件。

- 网格控制面

  运行于 Mesh Control Plane Cluster（以下简称 MCPC），网格控制面本质上也是工作在用户的工作集群，同时具备网格控制面的角色。 Istio 等核心控制组件会安装在该集群，作为网格的核心控制平面，直接对本网格内的网格资源进行管理，进行统一的服务发现和治理策略维护，实际执行、下发各项流量策略和安全策略。 托管网格实例，支持纳管多工作集群，实现多集群的流量治理和安全治理等。专用网格实例，支持单集群的流量治理和安全治理等。

- 网格数据面

  实际业务应用服务和治理策略生效的工作集群，包含了 Istio 的基本组件，但不会作为控制面使用，仅是提供一个类似管理代理的模式，主要负责边车注入、证书转发、xDS 转发等业务，同步来自 MCPC 的策略、服务注册信息下发至集群内的所在业务 Sidecar。 当网格类型为托管网格时，网格控制面和数据面一般情况是指不同的 Kubernetes 集群，如果您需要对当前集群纳管且不需要同时管理多集群服务时，建议使用专用网格。

贯穿整体架构的拓展模块主要有：可观测模块、微服务治理模块、应用工作台

- 可观测模块

  服务网格的可观测能力，借助 DCE 5.0 可观测性模块处理，服务网格通过接口调用的方式获取流量指标信息绘制拓扑图，并为用户提供各种维度的微服务监控图表。

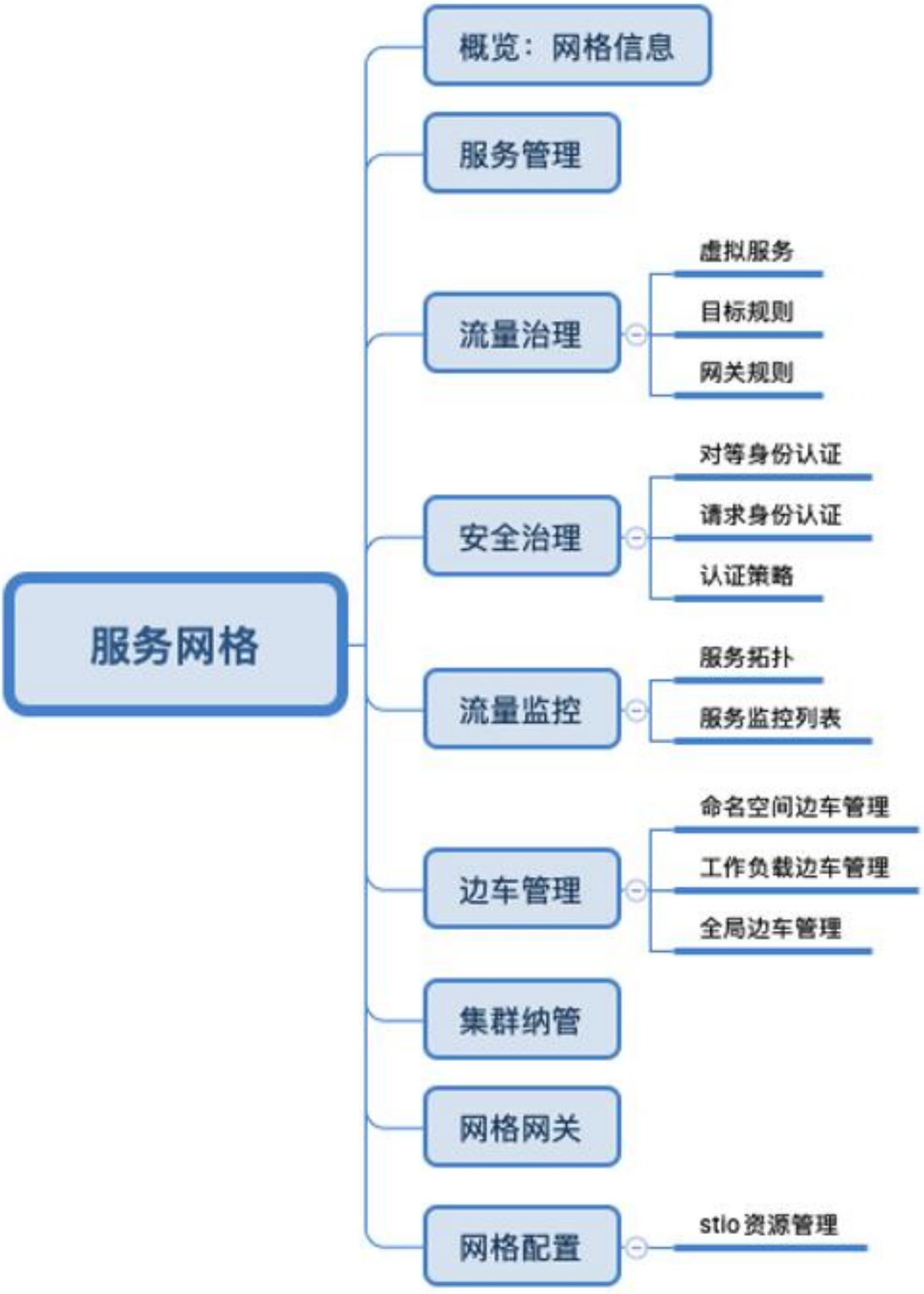
- 微服务治理模块

  微服务治理模块结合服务网格底座，完成对统一微服务治理的产品建设，赋能其云原生微服务治理和场景化微服务治理能力，方便用户对微服务做统一治理。

- 应用工作台

应用工作台是 DCE 5.0 的核心模块，用于提供给用户统一的应用编排、部署、流水线管理、CICD、应用监控、日志查询等能力。

功能架构


服务网格
概览：网格信息
服务管理
流量治理
虚拟服务
目标规则
网关规则
安全治理
对等身份认证
请求身份认证
认证策略
流量监控
服务拓扑
服务监控列表
边车管理
命名空间边车管理
工作负载边车管理
全局边车管理
集群纳管
网格网关
网格配置
stio资源管理

服务网格功能

如上图所示，服务网格提供了 9 个模块 12 个子模块功能，实现了多样化集群接入、多种模式的网格管理的能力。

系统命名空间

DCE 5.0 服务网格自带了一些系统命名空间，请勿将业务应用和服务部署到这些命名空间，这些命名空间及其作用如下所述。

分类	命名空间	作用
Istio 系统命名空间	istio-system	承载 Istio 控制平面组件及其资源
	istio-operator	部署和管理 Istio Operator
K8s 系统命名空间	kube-system	控制平面组件
	kube-public	集群配置和证书等
	kube-node-lease	监测和维护节点的活动
DCE 5.0 系统命名空间	amamba-system	应用工作台
	ghippo-system	全局管理
	insight-system	可观测性
	ipavo-system	首页仪表盘
	kairship-system	多云编排
	kant-system	云边协同
	kangaroo-system	镜像仓库
	kcoral-system	应用备份
	kubean-system	集群生命周期管理
	kpanda-system	容器管理
	local-path-storage	本地存储
	mcamel-system	中间件
	mspider-system	服务网格

分类	命名空间	作用
	skoala-system	微服务引擎
	spidernet-system	网络模块

UI 界面支持的功能

DCE 服务网格支持三种网格：托管网格、专有网格、外接网格。 在 UI 界面上，对这三种网格支持的功能如下。

网格管理

子类	托管网格	专有网格	外接网格
创建	√	√	√
配置	√	√	√
删除	√	√	√

网格概览

子类	托管网格	专有网格	外接网格
网格概览	√	√	√

集群管理

子类	托管网格	专有网格	外接网格
接入	√	×	×
移除	√	×	×

服务管理

子类	托管网格	专有网格	外接网格
服务列表	√	√	√
服务详细信息	√	√	√
服务编辑	√	√	√
服务条目（增删改查）	√	√	√

流量治理

子类	托管网格	专有网格	外接网格
虚拟服务（增删改查）	√	√	√
目标规则（增删改查）	√	√	√
网关规则（增删改查）	√	√	√

安全治理

子类	托管网格	专有网格	外接网格
对等身份认证（增删改查）	√	√	√
请求身份认证（增删改查）	√	√	√
授权策略（增删改查）	√	√	√

流量监控

子类	托管网格	专有网格	外接网格
拓扑	√	√	√
图表	√	√	√

边车管理

子类	托管网格	专有网格	外接网格
注入	√	√	√
卸载	√	√	√
资源限制	√	√	×

网格网关

子类	托管网格	专有网格	外接网格
创建	√	√	×

子类	托管网格	专有网格	外接网格
删除	√	√	×
设置	√	√	×

Istio 资源管理

子类	托管网格	专有网格	外接网格
创建	√	√	√
编辑	√	√	√
删除	√	√	√

OCP 环境接入服务网格配置方法

OCP (OpenShift Container Platform) 是 Red Hat 推出的容器平台。

本页说明服务网格接入 OCP 平台的操作步骤。

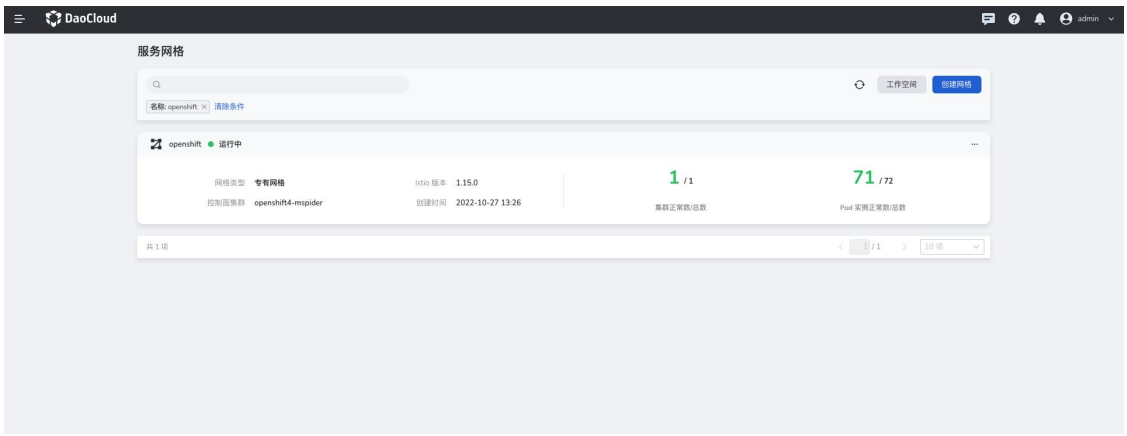
SCC 安全策略设置

在 Openshift 集群中，为服务网格增加命名空间的 privileged 用户权限，以 istio-operator、istio-system 两个命名空间为例：

oc adm policy add-scc-to-user privileged system:serviceaccount:istio-operator:istio-operator
oc adm policy add-scc-to-user privileged system:serviceaccount:istio-system:istio-system

接入 Openshift 集群

创建一个网格，接入 Openshift 集群。返回网格列表，发现 Openshift 集群已接入成功。



集群接入成功

但后端会报错：

COMMIT
2022-10-27T07:06:50.610621Z info Running command: iptables-restore --noflush /tmp/iptables-rules-1666854410610268141.txt1105821213
2022-10-27T07:06:50.616716Z error Command error output: xtables parameter problem: iptables-restore: unable to initialize table 'nat'

Error occurred at line: 1
Try `iptables-restore -h' or 'iptables-restore --help' for more information.
2022-10-27T07:06:50.616746Z error Failed to execute: iptables-restore --noflush /tmp/iptables-rules-1666854410610268141.txt1105821213, exit status 2

通过以下几步消除错误。

OCP 激活 iptables

修改 YAML

参考以下 YAML，根据实际环境修改部署：

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: dsm-init
  namespace: openshift-sdn
spec:
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dsm-init
  template:
    metadata:
      labels:
        app: dsm-init
        type: infra
    spec:
```

```
      containers:
      - command:
        - /bin/sh
        - -c
        - |
          #!/bin/sh
          set -x
          iptables -t nat -A OUTPUT -m tcp -p tcp -m owner ! --gid-owner
1337 -j REDIRECT --to-ports 15006
          iptables -t nat -D OUTPUT -m tcp -p tcp -m owner ! --gid-owner
1337 -j REDIRECT --to-ports 15006
          while true; do sleep 100d; done
        image: release.daocloud.io/mspider/proxyv2:1.15.0      # 修改 proxy 的镜像
地址
        name: dsm-init
        resources:
          requests:
            cpu: 100m
            memory: 20Mi
        securityContext:
          privileged: true
      dnsPolicy: ClusterFirst
      hostNetwork: true
      hostPID: true
      nodeSelector:
        kubernetes.io/os: linux
      priorityClassName: system-node-critical
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      serviceAccount: sdn
      serviceAccountName: sdn
```

添加参数

在 globalmesh YAML 中添加以下一行参数：

istio.custom_params.components.cni.enabled: "true"

!!! note

OpenShift 4.1 以上版本不再使用 iptables，转而使用 nftables。
因此需要安装 istio CNI 插件，否则在边车注入时会出现如下错误，即无法执行 iptables-resotre 命令。

```none
istio iptables-restore: unable to initialize table 'nat'
```

部署 istio-cni

```yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  components:
    cni:
      enabled: true
      namespace: istio-system
  values:
    sidecarInjectorWebhook:
      injectedAnnotations:
        k8s.v1.cni.cncf.io/networks: istio-cni
    cni:
      excludeNamespaces:
        - istio-system
      psp_cluster_role: enabled
      cniBinDir: /var/lib/cni/bin
      cniConfDir: /etc/cni/multus/net.d
```

```
cniConfFileName: istio-cni.conf
chained: false
```

兼容性测试报告 - OpenShift

基于 OpenShift，测试服务网格在创建专有网格时的兼容性。

本次测试报告以 OpenShift 4.11 为例，详细阐述了服务网格接入 OpenShift 专有网格模式时的兼容性测试过程。

测试目的

验证 DCE 5.0 服务网格上创建基于原生 istio，从控制面/数据面聚合 OpenShift 单集群模式的专有网格。

测试环境

服务网格环境

基于 demo-dev.daocloud.io

OpenShift 环境

准备了 OpenShift 3 个节点：

HostName	IP	Kubernetes 版本	Crictl 版本	OS 内核版本	OS Release
master01.mspider.openshift.io	10.6.157.111	v1.24.0+3882f8f	1.24.2	4.18.0-372.26.1.el8_6.x86_64	rhcos
master02.mspider.openshift.io	10.6.157.112	v1.24.0+3882f8f	1.24.2	4.18.0-372.26.1.el8_6.x86_64	rhcos
master03.mspider.openshift.io	10.6.157.113	v1.24.0+3882f8f	1.24.2	4.18.0-372.26.1.el8_6.x86	rhcos

HostName	IP	Kubernetes 版本	Crictl 版本	OS 内核版本	OS Release
				_64	

查看 3 个节点的状况：

oc get nodes -owide

输出类似于：

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME
master01.mspider.openshift.io Ready master.worker 7d3h v1.24.0+3882f8f 10.6.157.111 <none> Red Hat Enterprise Linux CoreOS 411.86.202209211811-0 (Ootpa) 4.18.0-372.26.1.el8_6.x86_64 cri-o://1.24.2-7.rhaos4.11.gitca400e0.el8
master02.mspider.openshift.io Ready master.worker 7d2h v1.24.0+3882f8f 10.6.157.112 <none> Red Hat Enterprise Linux CoreOS 411.86.202209211811-0 (Ootpa) 4.18.0-372.26.1.el8_6.x86_64 cri-o://1.24.2-7.rhaos4.11.gitca400e0.el8
master03.mspider.openshift.io Ready master.worker 7d2h v1.24.0+3882f8f 10.6.157.113 <none> Red Hat Enterprise Linux CoreOS 411.86.202209211811-0 (Ootpa) 4.18.0-372.26.1.el8_6.x86_64 cri-o://1.24.2-7.rhaos4.11.gitca400e0.el8

测试过程

通过容器管理接入 OpenShift

获取 OpenShift 集群的 kubeconfig (~/.kube/config)，在 demo-dev 环境的容器管理中接入集群。

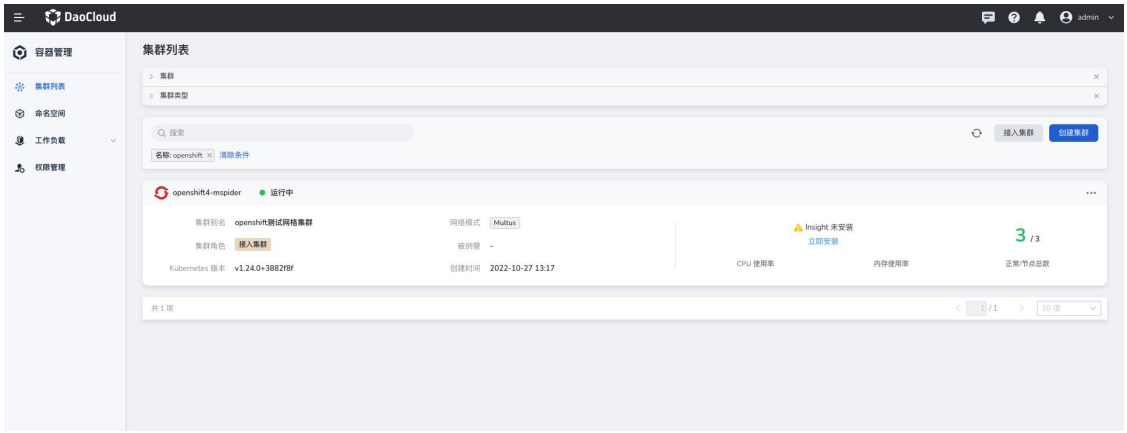
```
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://10.6.157.130:6443
  name: mspider
contexts:
- context:
    cluster: mspider
    user: admin
  name: admin
current-context: admin
preferences: {}
users:
- name: admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR
aekNDQWsrZ0F3SUJBZ0lJV0wzelp0MFErbGt3RFFZSktvWklodmNOQVFFTEJRQXdO
akVTTUJBR0ExVUUKQ3hNSmlzQmxibk5vYVdaME1TQXdIZ1lEVlFRREV4ZGhaRzFwY
mkxcmRXSmxZMjl1Wm1sbkxYTnBaMjVsY2pBZQpGdzB5TWpFd01qWXhNekE1TWpa
YUZ3MHpNakV3TWpNeE16QTVNalphTURBeEZ6QVZCZ05WQkFvVERuTjVjM1JsCmJ
UcHRZWE4wWlhKek1SVXdFd1lEVlFRREV3eHplWE4wWlcwNllXUnRhVzR3Z2dFaU1B
MEdDU3FHU0liM0RRRUlKQVFVQUE0SUJEd0F3Z2dFS0FvSUJBUUNtZ2RKcUNCTC
tGcmRmQmVseGU2dkNnbWpZMXJqN09rV1dDTHVUVWVJWgoyaWg1NHBWUVByUz
VkQVEvbUVaME1NcTVBSmQ4dE1maWpVcFlnQkU4d1pmeU0wZEM0d3h0empXWnZv
QWhSejZUCjFuRDFvd1RsZndJMDJjZ0lkZUpJSlJEa2tsV3JQUjVRZU0wbytLdWYrN1d
FUDVSSW82bWVxVEtuUHBuMmRyaWwKQUxPSFlvQWFiaklqVG51YUM2QWNlSzdyN
1VobGtQQ2pRK3dKdVl1cld5eW9HVFBTMlc4MVd6Mm9UNi9UQ05UVQpYTDhncWFo
a2dlWGpYZkl0bzYweVVpNnV3K1dMNk1oc0NhVlpXZG1TaDVRMzZZdTlSZEFKazdua
HlVR3JkNXRYCmplRm9LM0gyZjcycmwrOGdrcTdlVjNVYjRPY2pyU2VUUTlvNlY5Njh
wQzcxQWdNQkFBR2pmekl5TUE0R0ExVWQKRHdFQi93UUVBd0lGb0RBZEJnTlZIU1
VFRmpBVUJnZ3JCZ0VGQlFjREFRWUlLd1lCQlFVSEF3SXdEQVlEVlwVApBUUgvQkFJ
d0FEQWRCZ05WSFE0RUZnUVVEdW1uRWkxWjRlLytlcjJXcEY4bTBLODhGcE13SHdZ
```

RFZSMGpCQmd3CkZvQVVBM1VjTVdYYjh5aE54TVgrOUhBcWpneTBYMnN3RFFZSk
tvWklodmNOQVFFTEJRQURnZ0VCQUYvL1BvM3MKSWp5N2tva1p2OU1LUDJvMm9u
bWdhY0owNkllay9VbTVlbC8rMVE5WmhDeDJWSDNtWnlyME5kQ0M5Rm03OFovZw
poK3lDVi9hbGRXTnAxUXhyaWJUbGpscmNsek55L20vYWRmMUhKSGoram1mdEJFe
CtmNkVEdFFQZ3BUZUlCbk01CnhzMDFSUldwVTBKVnVsYU5RclphNHUyYVZnc3h3dj
YzdEEyeXlzblpxbGFqWktNZjU3TEhNTjBKdE10K3FyTDMKS1p1R1pqblVvcDk3SFh0VD
VBZ01jU1R1V0UyME9QR0tldFVuTVl3cldza2pvSlU5VWNSelBTQ1FGTWZCam53VQpN
LzNRM0lYTmNVay9mSDdxZk1DV2pGL29sNWNXMnhZOTN4ekF0bTVFWVpUei9VRk
4wOEFzS09qMS9hRWVBaHllCkFhclV5UHBEYTlwUkpJWT0KLS0tLS1FTkQgQ0VSVEl
GSUNBVEUtLS0tLQo=

client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb
3dJQkFBS0NBUUVBcG9lU2FnZ1MvaGEzWHdYcGNYdXJ3b0pvMk5hNCt6cEZsZ2k3
azFlaUdkb29lZUtWClVENjB1WFFFUDVoR2REREt1UUNYZkxUSDRvMUtXSUFSUE1H
WDhqTkhRdU1NYmM0MW1iNkFJVWMrazladzlhTUUKNVg4Q05ObklDSFhpU0NVUTV
KSlZxejBlVUhqTktQaXJuL3UxaEQrVVNLT3BucWt5cHo2WjluYTRwUUN6aDBmdwpH
bTR5STA1N21ndWdlSGl1NisxSVpaRHdvMFBzQ2JrZWExc3NxQmt6MHRsdk5WczlxR
St2MHdqVTFGeS9JS21vClpJSGw0MTN5TGFPdE1sSXVyc1BsaStqSWJBbWxXVm5aa
29lVU4rbUx2VVhRQ1pPNTRjbEJxM2ViVjQzaGFDdHgKOW4rOXE1ZnZJSkt1M2xkMU
crRG5JNjBuazBOditsZmV2S1F1OVFJREFRQUJBb0lCQUJ5dHo2Z2pxK0hlMTkydQpE
djlVNWNpaTNadzduN0RsNEladkNwL2RRcXhoUHdkL1YyaHk1SDNzMWE1K2MrUWZZ
MHRxSnExOEZkR1h0RzU1CjRlNHVlaFZsYjZpOW9xNW5EaVJsQTN5MzRMZG1BQjdP
N1ZENklwOURFNGtoaE5BWVVraU1TK1VxcWNZQ2lKTysKQVJHVk1UYU9lT1JHRERrZ
nUzSEMvcEhNOFJDNlB3dzJxV1BCazZJKzJWbHdGZTNDMUZGTGpHL0IxQkFPaUZk
YQpibUxkMU85cEt4QVMvODlBSzJYd0RBY1MwWk0zaDRTTkl5LzIvcVY0MHRHUFFh
aG5RWEJKcGpNRy9ZQWV0U3JkClVES3VnSkF6azVvT2Q0Z1pVSlNzZFYvK3lCQlJlc2
Zta3FpTTdxQXArZVpzbkxUUERYZy8zcDlCcXd2eHJUKzcKb3BGaFVTRUNnWUVBeV
EyZFZaRHZHcWljTzFXSVF1VFZYbFBsdjd4dTl2aklic1pxZWV3K1FlVXBnYkFoOXFQMA
pwbDJsaGtFR1VlZjZzeUdnb3BUTnRaci9lYzVCUXRSbzJGbzl4Z25QZkxTQ21ZMS9Fb
TQyd3UzUmhJNVNZOU1lCmZWYVFTeS9vRTRrQ3Y1ZmpTSmM2dkZ1aHVYTllFWkhr
aWRmSHFVaIFDazduRTErY2lxT01YNWNDZ1lFQTFBTSsKVitXWXlvb0JSaEVpck9vbV
F1U1l1d1lRcHhKaVF1UFBzUk9jR3VkRUFXU2V6SVVRV2RSTERETEl6cHhuUHhGQgpG
MHRrQnY4VmhrMk9uS0dRazAweVZnTlhnOHlvMnBlNk8zQ0w1QUdiMTdPMlBvWmJ4

Mkozc1JNa3Fuc015US92CkJVbUJ6YXVTTWEwVXprREFoU0xHaU01V253dDNMOEp
EaUE0ZWQxTUNnWUFWUFAzd3l3V25FRWFvc2VsWi82aFcKZFpCZ2g3eEZGSlMvd
HZBS2Z4MDRnc2Uycm05NENXdlBvemJZRHNobStiV2U5Sjl1YlQrcHZuelNuallncENX
TApMVVUzUlZRSXZWektjYnNKckdEV2lKN0lYT3h4SlJxMml4MkFVOGcxUUJUdFBsT
kJHTkNZa3lscldMYmw3RDV6CkhUczNNN3plU2VWNUUzR0s1Nm50Y3dLQmdRRRex
NQVNheHE4cjBFVlNPbS9xR2tuckNCeWlrVGNTZDVyMmtsQmwKVys3YkZkTm1KbUh
PanFSYUF3eWR0em9lVVdUZDl1SnZXZENXcC9lZ0RFcG1NSzFYanl5MEVhWFk2ZGJ
XRUYzcQpRM1crWWhCU2pLaFhpZnNCdm93Smg5ZzNEdEQxRFRFODI2TFJBdUtNZ
TEyYVFoS0FSaERSNGpiQUhJUHdvSlNLCkcwWDFnd0tCZ0J3cERlNmJWMGc2aEQz
N1VVOStpQnBTblVnWWNzLzZBVlJOR0k5RDhreFFlb1JESnNWT2ZVV1QKaERDWDF2
aUFpU0hEUStmSzlIZUhOWVk5Z05iRmdRakpDdlAxbHFRYU9lZnltUGliWUo5OVgxWn
Z5eWlqNjJkbQo1a3gvMnhwOGJPanM0U2hCY21JcGc5TjFZODl1UWtpTlJ4MEl0NUdF
RWJweFNXQ1U4TGEyCi0tLS0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg==

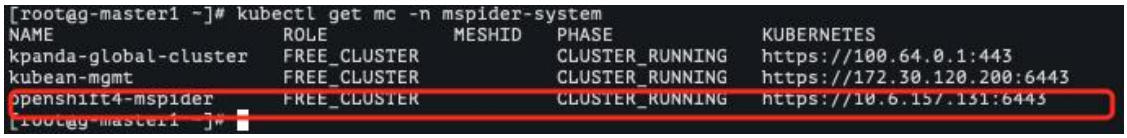
在容器管理中查看接入集群的状态：



接入 openshift

创建专有网格

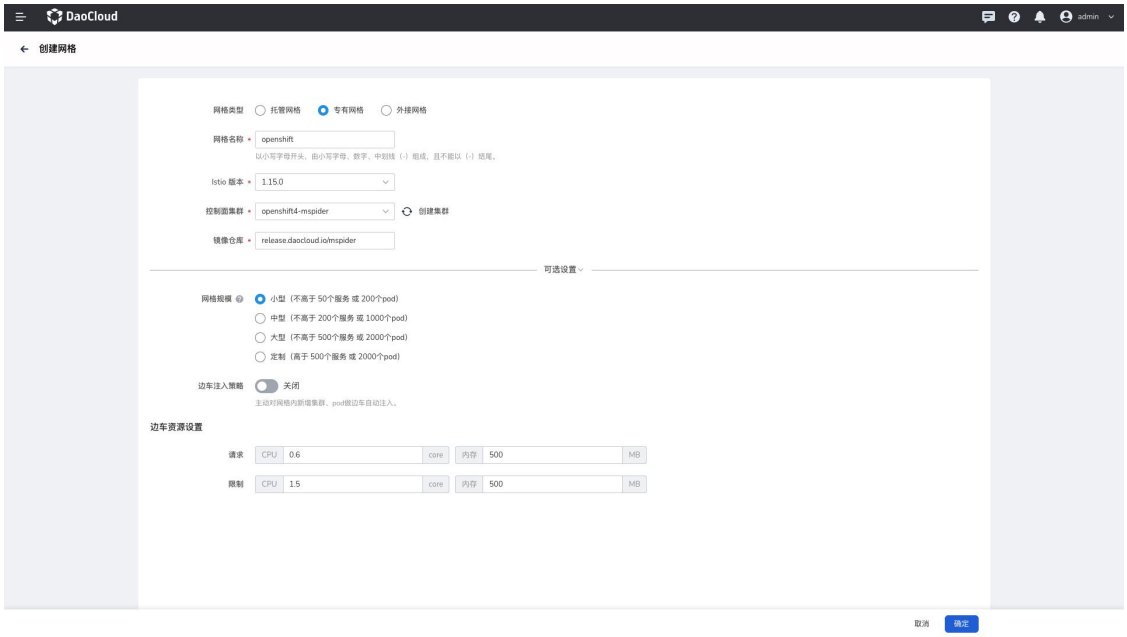
1. 登录 global-cluster 节点，查看服务网格同步容器管理接入集群的状态。



查看集群接入状态

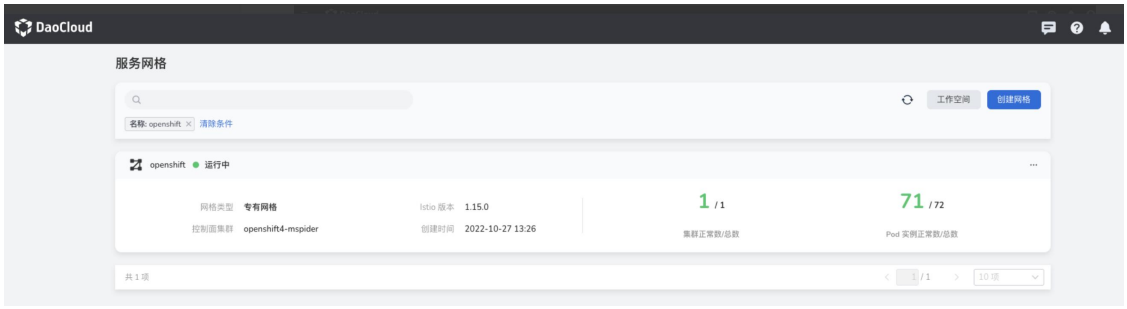
2. 完成创建网格时的配置后，点击确定。

   对于控制面集群，请选择通过容器管理接入的 openshift4-mspider 集群



创建网格

3. OpenShift 集群成功接入网格



成功接入网格

4. 查看各个组件是否健康

```
[root@msic ~]# oc get iop -n istio-system
NAME                        REVISION   STATUS    AGE
openshift-mspider-mcpc                 HEALTHY   73m
[root@msic ~]# oc get po -n istio-system
NAME                                          READY   STATUS    RESTARTS   AGE
istio-ingressgateway-86f8b95ff4-dm4fz         1/1     Running   0          6m33s
istiod-85b75c457b-fpxg5                       1/1     Running   0          6m22s
mspider-mcpc-ckube-remote-77c7fcb6b8-c72zm    1/1     Running   0          73m
mspider-mcpc-mcpc-controller-56498cb99-t5bpk  1/1     Running   0          73m
[root@msic ~]# oc get po -n istio-operator
NAME                              READY   STATUS    RESTARTS   AGE
istio-operator-58df7989b9-g4wc8   1/1     Running   0          2m36s
[root@msic ~]#
```

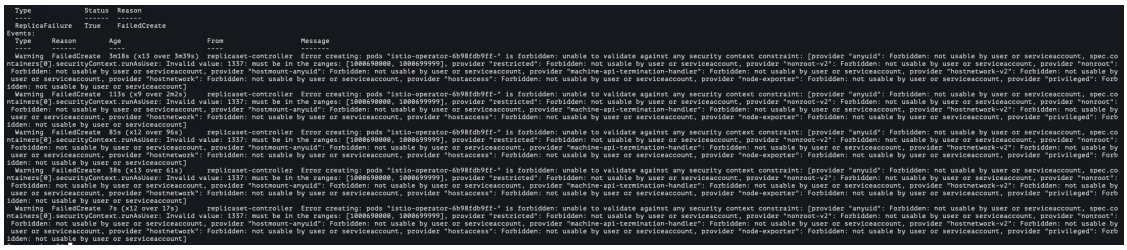
组件状况

```
[root@g-master1 ~]# kubectl get mc -n mspider-system
NAME                   ROLE                MESHID      PHASE              KUBERNETES
kpanda-global-cluster  FREE_CLUSTER                    CLUSTER_RUNNING    https://100.64.0.1:443
kubean-mgmt            FREE_CLUSTER                    CLUSTER_RUNNING    https://172.30.120.200:6443
openshift4-mspider     INTEGRATED_CLUSTER  openshift   MANAGED_SUCCEED    https://10.6.157.131:6443
[root@g-master1 ~]# kubectl get gm -n mspider-system
NAME        MODE        OWNERCLUSTER         DEPLOYNAMESPACE   PHASE       MESHVERSION
openshift   DEDICATED   openshift4-mspider   istio-system      SUCCEEDED   1.15.0
[root@g-master1 ~]#
```

组件状况

接入时遇到的问题

1. 提示错误 istio-operator RS CreateFailed



RS CreateFailed

原因分析：OpenShift（安全上下文约束）SCC 对创建 Pod 有权限限制

解决办法：增加 SCC 用户命名空间权限“privileged”

oc adm policy add-scc-to-user privileged -z istio-operator

oc adm policy add-scc-to-user privileged -z istio-system

oc adm policy add-scc-to-group privileged system:authenticated

重启 istio-operator rs 解决问题。

2. 提示错误 message: no running Istio pods in "istio-system"

原因分析：字面提示是 istio-system 命名空间下缺少 istiod/istio-ingressgateway service pod。 实际是网格全局边车资源限制中的内存资源参数格式错误；正确格式为 500Mi

```
[root@g-master1 ~]# kubectl get gm -n mspider-system openshift -oyaml
apiVersion: discovery.mspider.io/v3alpha1
kind: GlobalMesh
metadata:
  creationTimestamp: "2022-10-27T05:26:07Z"
  finalizers:
  - gsc-controller
  generation: 2
  name: openshift
  namespace: mspider-system
  resourceVersion: "820271"
  uid: a40318a9-5e39-47f2-9fa5-6ac0bfc50205
spec:
  hub: release.daocloud.io/mspider
  mode: DEDICATED
  ownerCluster: openshift4-mspider
  ownerConfig:
    controlPlaneParams:
      global.istio_version: 1.15.0
      global.mesh_capacity: S50P200
      istio.custom_params.values.global.proxy.resources.limits.cpu: "1.5"
      istio.custom_params.values.global.proxy.resources.limits.memory: "500"
      istio.custom_params.values.global.proxy.resources.requests.cpu: "0.6"
      istio.custom_params.values.global.proxy.resources.requests.memory: "500"
      istio.custom_params.values.sidecarInjectorWebhook.enableNamespacesByDefault: "false"
    deployNamespace: istio-system
status:
```

资源参数格式

解决办法：修改 yaml 文件，添加 Mi 单位

kubectl edit gm –n mspider-system openshift –oyaml

```
[root@g-master1 ~]# kubectl get gm -n mspider-system openshift -oyaml
apiVersion: discovery.mspider.io/v3alpha1
kind: GlobalMesh
metadata:
  creationTimestamp: "2022-10-27T05:26:07Z"
  finalizers:
  - gsc-controller
  generation: 3
  name: openshift
  namespace: mspider-system
  resourceVersion: "839985"
  uid: a40318a9-5e39-47f2-9fa5-6ac0bfc50205
spec:
  hub: release.daocloud.io/mspider
  mode: DEDICATED
  ownerCluster: openshift4-mspider
  ownerConfig:
    controlPlaneParams:
      global.istio_version: 1.15.0
      global.mesh_capacity: S50P200
      istio.custom_params.values.global.proxy.resources.limits.cpu: "1.5"
      istio.custom_params.values.global.proxy.resources.limits.memory: 500Mi
      istio.custom_params.values.global.proxy.resources.requests.cpu: "0.6"
      istio.custom_params.values.global.proxy.resources.requests.memory: 500Mi
      istio.custom_params.values.sidecarInjectorWebhook.enableNamespacesByDefault: "false"
    deployNamespace: istio-system
status:
```

修改单位

3. 注入边车时提示 pod Init:CrashLoopBackOff

   激活 ocp iptables

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: dsm-init
  namespace: openshift-sdn
spec:
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dsm-init
  template:
    metadata:
      labels:
        app: dsm-init
        type: infra
    spec:
      containers:
      - command:
        - /bin/sh
        - -c
        - |
          #!/bin/sh
          set -x
          iptables -t nat -A OUTPUT -m tcp -p tcp -m owner ! --gid-owner 1337 -j REDIRECT --to-ports 15006
          iptables -t nat -D OUTPUT -m tcp -p tcp -m owner ! --gid-owner 1337 -j REDIRECT --to-ports 15006
          while true; do sleep 100d; done
        image: release.daocloud.io/mspider/proxyv2:1.15.0
```

```
    name: dsm-init
    resources:
      requests:
        cpu: 100m
        memory: 20Mi
    securityContext:
      privileged: true
  dnsPolicy: ClusterFirst
  hostNetwork: true
  hostPID: true
  nodeSelector:
    kubernetes.io/os: linux
  priorityClassName: system-node-critical
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: sdn
  serviceAccountName: sdn
```

运行命令：

kubectl edit gm -n mspider-system openshift -oyaml

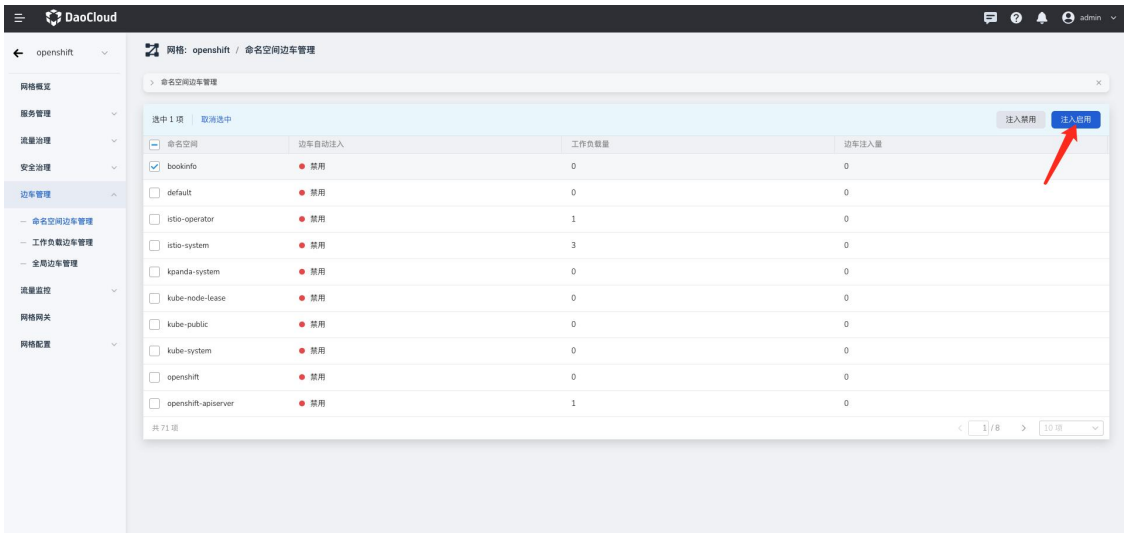
添加：istio.custom_params.components.cni.enabled: "true"

部署 bookinfo 应用测试

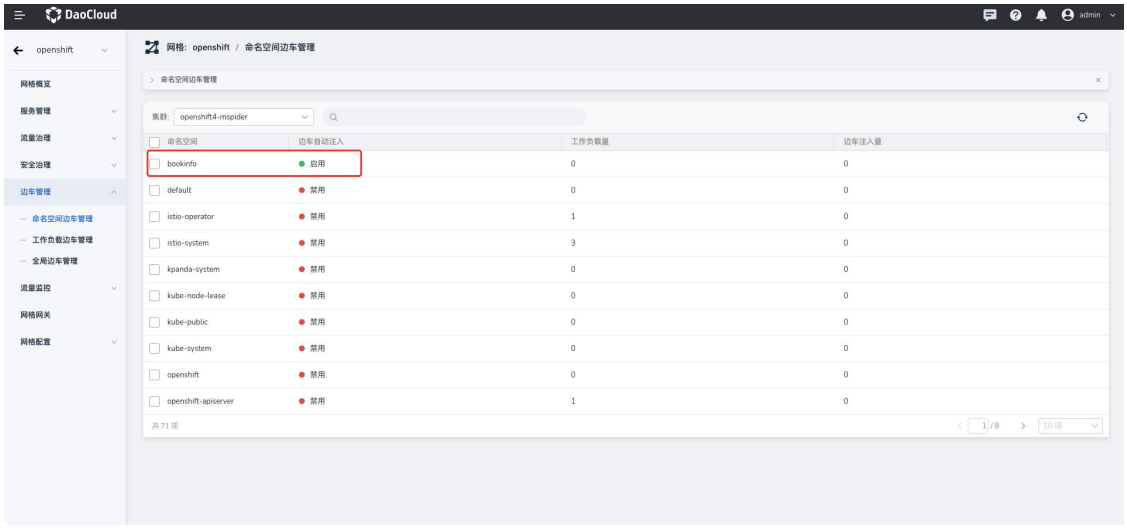
创建 bookinfo namespace，部署 bookinfo 应用时需要对命名空间添加 SCC。

oc adm policy add-scc-to-user privileged -z bookinfo

1. 启用命名空间边车注入



启用命名空间边车注入



查看命名空间边车

2. 终端查看 bookinfo 命名空间 Labels

   oc describe ns bookinfo |grep "istio-injection-enabled"

   屏幕输出类似于：

   Labels: istio-injection-enabled

3. 部署 bookinfo 应用

oc apply –f https://raw.githubusercontent.com/istio/istio/release–1.15/samples/bookinfo/platform/kube/bookinfo.yaml –n bookinfo



部署 bookinfo

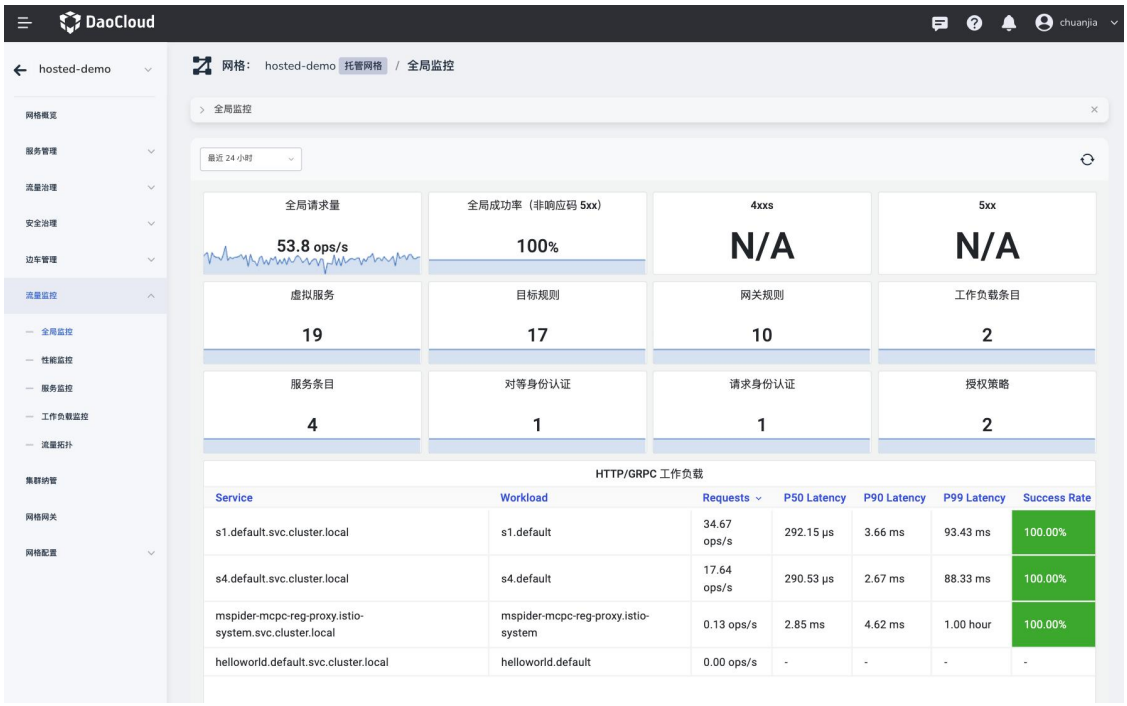
oc get pod –n bookinfo

2023-08-18

监控看板指标说明

本文用于补充介绍，网格实例中流量监控中的各个监控看板的指标的含义，方便用户查询和定义

全局监控



全局监控

分类	参数	参数介绍	计算方式
常规	全局请求总量	“Global Request Volume”代表整个服务网格中的请求流量总量，Ops/s 标识每秒执行的操作或请求的数量。在 Istio 中，这个度量通常用来评估整个服务网格的流	OPS/s = 总操作数 / 时间段（秒）

分类	参数	参数介绍量	计算方式
	全局成功率（非 5xx 响应）		全局成功率（非响应率 5xx） = (总请求 – 5xx 响应) / 总请求 * 100
	4xxs	4xxs 是客户端错误，表示请求可能有问题（例如，404 表示未找到）	ops/s = 每秒收到的 4xx 响应的数量
	5xxs	5xxs 是服务器错误，表示服务器未能正确处理有效请求（例如，500 表示内部服务器错误）	ops/s = 每秒收到的 5xx 响应的数量
	虚拟服务	统计一段时间范围内服务网格内的虚拟服务的净增长情况，通过比较添加和删除事件的数量，可以了解服务的动态变化，并可能有助于识别潜在的问题。	max(Virtual Services && and) – max(Virtual Services && delete)
	目标规则	统计一段时间范围内服务网格内的目标规则（Destination Rule）的净增长情况，通过比较添加和删除事件的数量，可以了解服务的动态变化，并可能有助于识别潜在的问题。	max(Destination Rule && and) – max(Destination Rule && delete)
	网关规则	统计一段时间范围内服务网格内的网关规则	max(Gateway && and) – max(Gateway && delete)

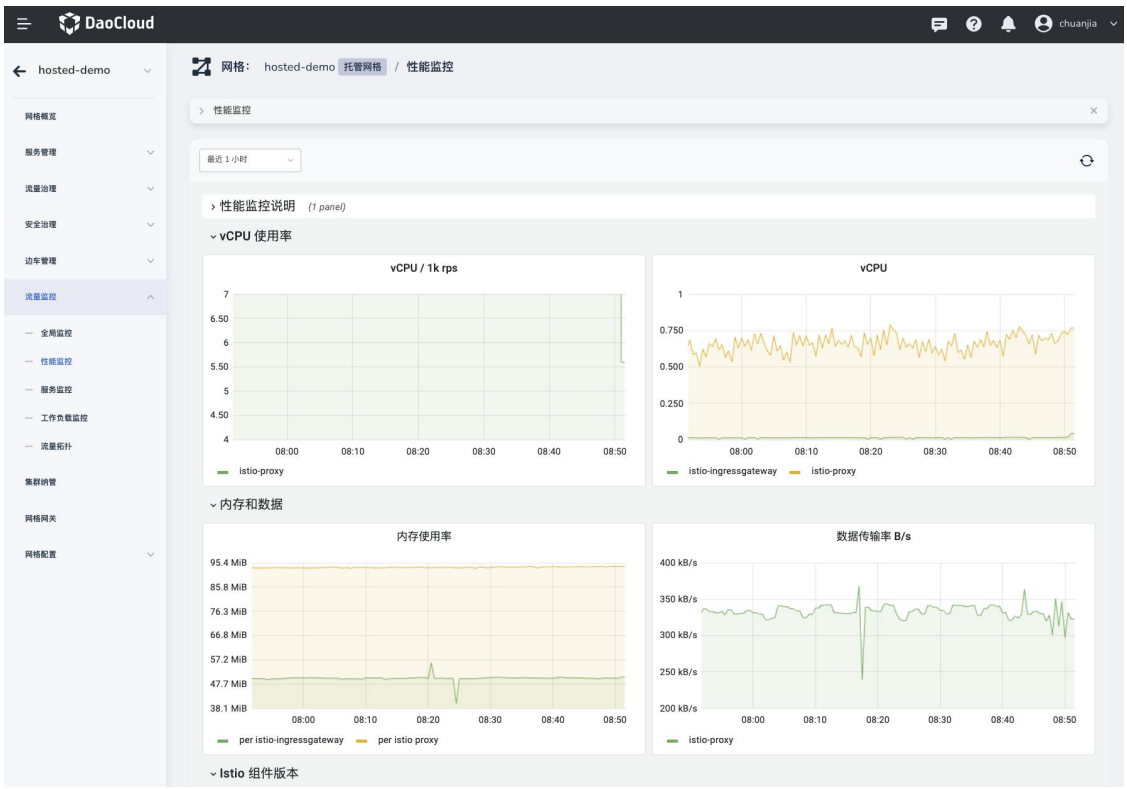
分类	参数	参数介绍	计算方式
		（Gateway）的净增长情况，通过比较添加和删除事件的数量，可以了解服务的动态变化，并可能有助于识别潜在的问题。	
	工作负载条目	统计一段时间内 WorkloadEntry 的增长趋势，通过观察 Workload 的变化，可以快速了解服务的动态变化，并可能有助于识别潜在的问题。	max(WorkloadEntry && and) – max(WorkloadEntry && delete)
	服务条目	统计一段时间内 Service Entries 的增长趋势，通过观察 Service Entries 的变化，可以快速了解服务的动态变化，并可能有助于识别潜在的问题。	max(Service Entries && and) – max(Service Entries && delete)
	对等认证策略	统计一段时间内 PeerAuthentication Policies 的增长趋势，通过观察 PeerAuthentication Policies 的变化，可以快速了解服务的动态变化，并可能有助于识别潜在的问题。	max(PeerAuthentication Policies && and) – max(PeerAuthentication Policies && delete)
	请求认证策略	统计一段时间内 RequestAuthentication Policies 的增长趋势，通过	max(RequestAuthentication Policies && and) – max(RequestAuthentic

分类	参数	参数介绍	计算方式
		观察 RequestAuthentication Policies 的变化，可以快速了解服务的动态变化，并可能有助于识别潜在的问题。	n Policies && delete)
	授权策略	统计一段时间内 Authorization Policies 的增长趋势，通过观察 Authorization Policies 的变化，可以快速了解服务的动态变化，并可能有助于识别潜在的问题。	max(Authorization Policies && and) – max(Authorization Policies && delete)
HTTP/ GRPC 工作负载	Service	服务名称，名称组合服务所在的命名空间和 Kubernetes 的 Service 访问信息	
	工作负载	工作负载名称，名称组合工作负载所在的命名空间	
	请求量	请求的数量，表示特定时间段内的请求总数	sum(rate(istio_requests_total{…}[1m]))
	P50 延迟	中位数延迟，50%的请求在的请求延迟在这个时间以下完成	histogram_quantile(0.50, sum(rate(…))
	P90 延迟	90%的请求在的请求延迟在这个时间以下完成	histogram_quantile(0.90, sum(rate(…))
	P99 延迟	99%的请求在的请求延迟在这个时间以下完成	histogram_quantile(0.99, sum(rate(…))

分类	参数	参数介绍	计算方式
	成功率	成功率，在查询时间段内成功响应（响应状态码不等于5xx）的请求所占的百分比，	sum(rate(… response_code!~“5.*“)) / sum(rate(…))
TCP 工作负载	Service	工作负载名称，从 Istio 提供的指标中获取 destination_service 标签，这个标签包含了服务的名称。	destination_workload
	工作负载	工作负载名称，从 Istio 提供的指标中获取 destination_workload 标签，这个标签包含了工作负载的名称。	destination_service
	发送字节数	每秒发送的字节量	通过 istio_tcp_sent_bytes_total 计算累计的 TCP 字节，使用 rate 计算对应的发送速率
	接收字节数	每秒接收的字节量	通过 istio_tcp_received_bytes_total 计算累计的 TCP 字节，使用 rate 计算对应的发送速率
基于版本的 Istio 组		Istio 组件构建版本的可视化展示，展示各个组件的版本分布，以及它们在不同集	sum(istio_build{mesh_id=“$mesh”}) by (component, tag, mesh_cluster)

分类	参数	参数介绍	计算方式
件		群中的部署情况。这对于了解 Istio 部署的健康状况和一致性非常有用。	

性能监控



性能监控

分类	参数	参数介绍	计算方式
VCPU 使用量	vCPU / 1k rps	展示 Istio 每千次请求（1k rps）所消耗的虚拟 CPU（vCPU）资源，主要查询了 istio-ingressgateway 和 istio-proxy。为了保障查询效率，Istio 限制仅当 istio-ingressgateway 请求	(sum(irate(container_cpu_usage_seconds_total{namespace!="istio-system",container="istio-proxy"}[1m]))/ (round(sum(irate(istio_requests_total[1m])),

分类	参数	参数介绍	计算方式
		大于 10 时才进行 istio-proxy 统计。	0.001)/1000))/(sum(irate(istio_requests_total{source_workload="istio-ingressgateway"}[1m])) >bool 10)
	vCPU	展示 Istio 中虚拟 CPU (vCPU) 的整体使用情况	
内存和数据	内存用量	展示 Istio 系统组件中的内存使用情况，统计单位为 bytes	sum(container_memory_working_set_bytes{pod=~"istio-ingressgateway-.*"}) / count(container_memory_working_set_bytes{pod=~"istio-ingressgateway-.*",container!="POD"})
	数据传输量 B/s	展示 Istio 系统组件的的每秒传输的字节数，统计单位为 Bps	sum(irate(istio_response_bytes_sum{source_workload="istio-ingressgateway", reporter=~"$reporter",destination_mesh_id="$mesh"}[1m]))
Istio 组件版本	Istio 组件版本	展示 Istio 组件的版本统计信息，图例格式将包括组件名称、标签和网格集群。	sum(istio_build{mesh_id="$mesh"}) by (component, tag, mesh_cluster) || 边车资源使用率 | 内存 | 展示边车容器"istio-proxy"的每分钟工作集字节的总和的变

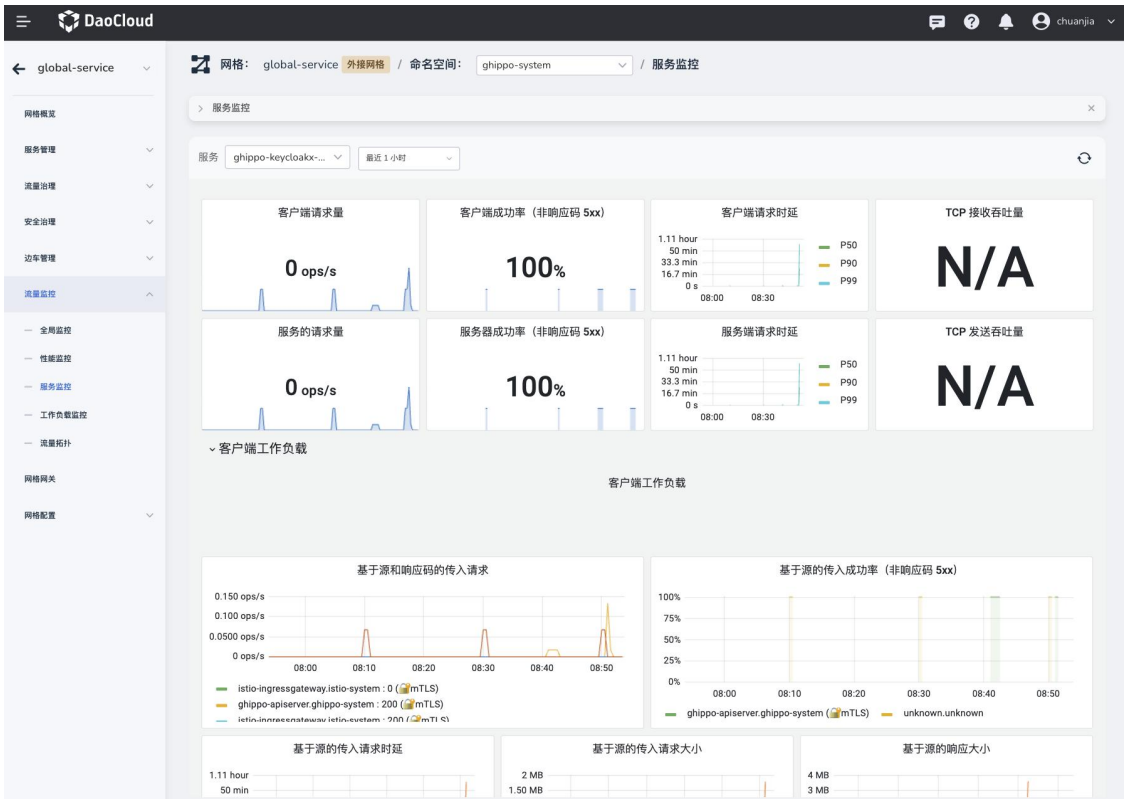
分类	参数	参数介绍	计算方式
			化，用于监视 Istio 代理容器的内存使用情况 | sum(container_memory_working_set_bytes{container="istio-proxy"}) | | | vCPU | 展示边车代理（Proxy）资源使用的虚拟 CPU（vCPU）统计信息，展示容器“istio-proxy”的 CPU 使用秒数的速率总和 | sum(rate(container_cpu_usage_seconds_total{container="istio-proxy"}[1m])) | | | 磁盘 | 展示边车代理（Proxy）资源使用的磁盘统计信息，展示容器“istio-proxy”的文件系统使用字节的总和 | sum(container_fs_usage_bytes{container="istio-proxy"}) | | Istiod 资源使用率 | 内存 | 展示 Istiod 服务的内存使用情况，提供了一个全面的视图： - 总计：Istiod 服务在 Kubernetes 中的总内存使用量 - 容器内存：Istiod 服务在 Kubernetes 中每个容

分类	参数	参数介绍	计算方式
			器的内存使用量 包括虚拟内存、常驻内存、堆内存和栈内存等不同类型的内存使用情况。 | 总计（Total (k8s)）："sum(container_memory_working_set_bytes{container=~\"discovery|istio-proxy\", pod=~\"istiod-.*\"})" 容器内存（{{ container }} (k8s)）："container_memory_working_set_bytes{container=~\"discovery|istio-proxy\", pod=~\"istiod-.*\"}" |||
			vCPU | 展示 Istiod 服务的虚拟 CPU（vCPU）使用情况，提供了一个全面的视图： – 总计： 显示 Istiod 服务在 Kubernetes 中的总 CPU 使用率 – 容器 CPU 使用率：显示 Istiod 服务在 Kubernetes 中每个容器的 CPU 使用率 – Pilot：显示 Istiod 的 pilot 组件的 CPU 使用情况 | 总计（Total (k8s)）："sum(rate(container_cpu_

分类	参数	参数介绍	计算方式
			usage_seconds_total{container=~\"discovery|istio-proxy\", pod=~\"istiod-.*\"}[1m]))" 容器 CPU 使用率（{{ container }} (k8s)）："sum(rate(container_cpu_usage_seconds_total{container=~\"discovery|istio-proxy\", pod=~\"istiod-.*\"}[1m])) by (container)" pilot: "irate(process_cpu_seconds_total{app=\"istiod\"}[1m])" ||| 磁盘 | 展示每个集群中 Istio 组件的磁盘使用情况，特别是与 discovery 和 istio-proxy 容器相关的文件系统使用情况。 | sum(process_open_fds{mesh_id="$mesh",app="istiod"}) by (mesh_cluster) container_fs_usage_bytes{container=~"discovery|istio-proxy", pod=~"istiod-.*"}
	Goroutines	展示每个集群中 Istio 组件的 Go 协程数量的趋势	sum(go_goroutines{mesh_id="$mesh",

分类	参数	参数介绍	计算方式
			app=“istiod”}) by (mesh_cluster)

服务监控



服务监控

分类	参数	参数介绍	计算方式
常规	客户端请求量	展示当前服务的客户端的每 5 分钟的请求操作量，如果结果为空值是会展示为 N/A；阈值数值超过 80% 时会展示为红色	round(sum(irate(istio_requests_total{reporter=“$reporter",destination_mesh_id="$mesh”,destination_service=“$service"}[5m])), 0.001) | | |
	客户端成功率（非 5xx 响应）	展示当前服务的客户端的每 5 分钟的成功请求率，并且提供了一种可视化	

分类	参数	参数介绍	计算方式
			方式来快速识别潜在的问题或趋势。 | sum(irate(istio_requests_total{reporter=~"$reporter",destination_mesh_id="$mesh",destination_service=~"$service",response_code!~"5.*"}[5m])) / sum(irate(istio_requests_total{reporter="$reporter",destination_mesh_id="$mesh",destination_service="$service"}[5m])) | | | 客户端请求时延 | 展示当前服务的客户端的请求用时情况，定义了三个目标，用于计算 P50、P90 和 P99 的持续时间。表达式分别计算了 50%、90% 和 99% 的持续时间百分位数。 | 示例：(histogram_quantile(0.50, sum(irate(istio_request_duration_milliseconds_bucket{reporter=~\"$reporter",destination_mesh_id="$mesh\",destination_service=~\"$service"}[1m])) by (le)) / 1000) or histogram_quantile(0.50,

分类	参数	参数介绍	计算方式
			sum(irate(istio_request_duration_seconds_bucket{reporter="$reporter\",destination_mesh_id=\"$mesh",destination_service="$service\"}[1m])) by (le)) \| \| \| TCP 接收字节数 \| 展示当前服务在 1 分钟内 TCP 接收字节的即时速率之和，如果匹配到 "null"，结果文本为 "N/A"，单位 Bps \| "sum(irate(istio_tcp_received_bytes_total{reporter=~\"$reporter",destination_mesh_id="$mesh\", destination_service=~\"$service"}[1m]))”
	服务的请求量	展示当前服务的请求量，按照时间展示趋势图，如果匹配到 “null”，结果文本为 “N/A”，单位 Ops	round(sum(irate(istio_requests_total{reporter=“destination”, destination_mesh_id=“$mesh",destination_service=~"$service”}[5m])), 0.001)
	服务器成功率（非 5xx 响应）	展示当前服务的非 5xx 响应的成功率，阈值设置表明，95%以下的成功率标记为红色，99%以下的成功率标记为橙色， 100%为绿色（百分比取值 2 位）	sum(irate(istio_requests_total{reporter=“destination”, destination_mesh_id=“$mesh",destination_service=~"$service”,response_code!

分类	参数	参数介绍	计算方式
			~“5.”}[5m])) / sum(irate(istio_requests_total{reporter=”destination”, destination_mesh_id=”$mesh", destination_mesh_id="$mesh”,destination_service=„$service"}[5m])) \|\|\| 服务端请求时延 \| 展示当前服务的服务器请求的延时，通过计算不同分位数的持续时间来提供对服务性能的深入了解。三个目标表达式分别计算了中位数、90%、和 99%的请求的延时，以提供从中位数到高端的性能概览 \| – histogram_quantile – istio_request_duration_milliseconds_bucket – istio_request_duration_seconds_bucket \|\|\| TCP 发送字节数 \| 展示当前服务在 1 分钟内 TCP 发送字节的即时速率之和，如果匹配到 "null"，结果文本为 "N/A"，单位 Bps \|\|\| 客户端工作负载 \| 基于源和响应码的传入请求 \| 展示按源工作负载和响应代码分类的传入请求，能够可视化地展示各种工作负载之间的交互情况，分别对具有和不具有互相认证 TLS 的连接进行计算，以提供对连接安全性的洞察。可以清晰地了解请求如何在不同的源和目的地之间分布。 \|\|\|\| 基于源的传入成功率（非 5xx 响应） \| 展示按源工作负载和命名空间分类的传入成功率，其中成功率是非 5xx 响应的百分比，分别对具有和不具有互相认证 TLS 的连接进行计算，以提供对连接成功率的视图。 \| sum(irate(istio_requests_total{reporter=~"$reporter”,

分类	参数	参数介绍	计算方式
			destination_mesh_id="$mesh",* *connection_security_policy="mutual_tls",* *destination_service=~"$service",response_code!”5*.”,* source_workload=~“$srcwl", source_workload_namespace=~"$srcns”}[5m])) by (source_workload, source_workload_namespace) / sum(irate(istio_requests_total{reporter=~“$reporter",destination_mesh_id="$mesh”, connection_security_policy=“mutual_tls”, destination_service=~“$service", source_workload=~"$srcwl”, source_workload_namespace=~“$srcns"}[5m])) by (source_workload, source_workload_namespace)||| 基于源的传入请求时延 | 展示按源工作负载和命名空间的请求耗时，分别

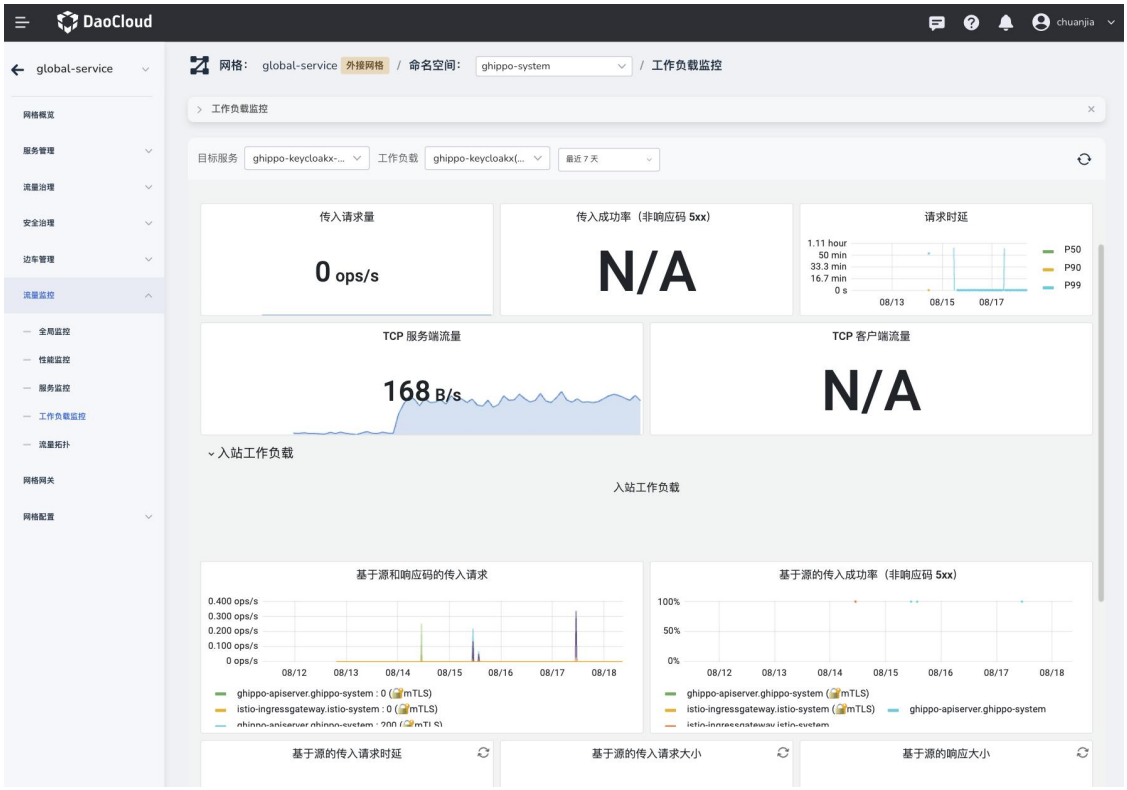
分类	参数	参数介绍	计算方式
		计算了 P50、P90、P95 和 P99 等分位数不同的请求持续时间。注意，将分位值除以 1000 转为为秒作为单位。 | 示例：	(histogram_quantile(0.50, sum(irate(istio_request_duration_milliseconds_bucket{reporter=~"$reporter”,destination_mesh_id=“$mesh", connection_security_policy="mutual_tls", destination_service=~"$service”, source_workload=~“$srcwl", source_workload_namespace=~"$srcns”}[1m])) by (source_workload, source_workload_namespace, le)) / 1000) or histogram_quantile(0.50, sum(irate(istio_request_duration_seconds_bucket{reporter=~“$reporter",destination_mesh_id="$mesh”, connection_security_polic

分类	参数	参数介绍	计算方式
			y=“mutual_tls”, destination_service=~“$service", source_workload=~"$srcwl”, source_workload_namespace=~“$srcns”}[1m])) by (source_workload, source_workload_namespace, le))
	基于源的传入请求大小	展示按源工作负载传入的请求大小，分别计算了 P50、P90、P95 和 P99 等分位数不同的请求大小。	
	基于源的响应大小	展示按源工作负载（source workload）展示了在 P50、P90、P95、P99 (��mTLS): 表示在启用了相互 TLS（mTLS）的情况下的响应大小的百分位数	
	接收来自传入 TCP 连接的字节数	展示当前服务通过 TCP 连接收到的字节数，并且展示在 mutual TLS 连接（标记为 ��mTLS）和非 mutual TLS 连接下从 TCP 连接收到的字节数的情况	
	发送到传	展示当前服务通过 TCP 连	

分类	参数	参数介绍	计算方式
	入 TCP 连接的字节数	接发送的字节数，并且展示在 mutual TLS 连接（标记为 ��mTLS）和非 mutual TLS 连接下从 TCP 连接收到的字节数的情况	
服务工作负载	基于目标负载和响应码的传入请求	展示按目的工作负载和响应代码分类的传入请求，能够可视化地展示各种工作负载之间的交互情况，分别对具有和不具有互相认证 TLS 的连接进行计算，以提供对连接安全性的洞察。可以清晰地了解请求如何在不同的源和目的地之间分布。	
	基于目标负载的传入成功率（非 5xx 响应）	展示按目的工作负载和命名空间分类的传入成功率，其中成功率是非 5xx 响应的百分比，分别对具有和不具有互相认证 TLS 的连接进行计算，以提供对连接成功率的视图。	
	基于服务负载的传入请求时延	展示按目的工作负载和命名空间的请求耗时，分别计算了 P50、P90、P95 和 P99 等分位数不同的请求持续时间。注意，将分位值除以 1000 转为为秒作为单	

分类	参数	参数介绍	计算方式
		位。	
	基于服务负载的传入请求大小	展示按目的工作负载传入的请求大小，分别计算了 P50、P90、P95 和 P99 等分位数不同的请求大小。	
	基于服务负载的响应大小	展示按目的工作负载（source workload）展示了在 P50、P90、P95、P99 (��mTLS): 表示在启用了相互 TLS（mTLS）的情况下的响应大小的百分位数	
	接收来自传入 TCP 连接的字节数	展示当前服务通过 TCP 连接收到的字节数，并且展示在 mutual TLS 连接（标记为 ��mTLS）和非 mutual TLS 连接下从 TCP 连接收到的字节数的情况	
	发送到传入 TCP 连接的字节数	展示当前服务通过 TCP 连接发送的字节数，并且展示在 mutual TLS 连接（标记为 ��mTLS）和非 mutual TLS 连接下从 TCP 连接收到的字节数的情况	

工作负载监控



工作负载监控

分类	参数	参数介绍	计算方式
常规	传入请求量	展示当前工作负载的传入请求量，单位为 Ops，如果接收到的数据为空（null），则会显示“N/A”	计算过去 5 分钟内的传入请求总数，其中包括特定的目的地工作负载、命名空间和集群。
	传入成功率（非 5xx 响应）	展示当前工作负载的每 5 分钟的成功请求率（不含 5xx 的请求），并且提供了一种可视化方式来快速识别潜在的问题或趋势。如果成功率低于 95%，将显示为红色，如果低于 99%，则显示为橙色	使用了两个分母和分子的查询来计算非 5xx 响应的百分比。分子计算与特定服务相关的非 5xx 响应，分母计算与该服务相关的所有请求

分类	参数	参数介绍	计算方式
	请求时延	展示当前工作负载的请求的时延，分别显示 P50、P90 和 P99 的请求持续时间，可以用于观察系统的性能，并快速识别潜在的瓶颈或延迟问题。	(histogram_quantile(0.50, sum(irate(istio_request_duration_milliseconds_bucket{reporter=“$reporter",destination_mesh_id="$mesh”,destination_workload=“$workload", destination_cluster=~"$dstcluster”, destination_workload_namespace=~“$namespace"}[1m])) by (le)) / 1000) or histogram_quantile(0.50, sum(irate(istio_request_duration_seconds_bucket{reporter=~"$reporter”,destination_mesh_id=“$mesh",destination_workload=~"$workload”, destination_cluster=~“$dstcluster", destination_workload_namespace=~"$namespace”}[1m])) by (le))
	TCP 服务端流量	展示当前工作负载的 TCP 服务器流量，注重统计作为目标工作负载、命名空间和集群，即作为 TCP 服务器，有助于了解网络负载和	destination_workload_namespace=~“$namespace", destination_workload=~"$workload”, destination_cluster=~“$ds

分类	参数	参数介绍	计算方式
		可能的瓶颈；并以每秒字节（Bps）为单位显示 TCP 发送和接收的字节总数。	tcluster"||| TCP 客户端流量 | 展示当前工作负载的 TCP 客户端的流量，注重统计作为源工作负载、命名空间和集群，即作为 TCP 服务器，有助于了解网络负载和可能的瓶颈；并以每秒字节（Bps）为单位显示 TCP 发送和接收的字节总数。 | source_workload_namespace=~"$namespace”, source_workload=~“$workload"|| 入站工作负载 | 基于源和响应码的传入请求 | 展示当前工作负载中，根据源工作负载和响应代码分类的传入请求，计算使用和不使用双向 TLS 连接的传入请求速率，并按源工作负载、源命名空间和响应代码分类 | 示例：round(sum(irate(istio_requests_total{connection_security_policy!="mutual_tls", destination_workload_namespace=~"$namespace”, destination_workload=~“$

分类	参数	参数介绍	计算方式
			workload", destination_cluster=~"$dstcluster”, reporter=~“$reporter",destination_mesh_id="$mesh”, source_workload=~“$srcwl", source_workload_namespace=~"$srcns”}[5m])) by (source_workload, source_workload_namespace, response_code), 0.001)
	基于源的传入成功率（非 5xx 响应）	展示传入成功请求的比率（非 5xx 响应），分别按使用和不使用双向 TLS 的连接进行分类，并进一步按源工作负载和源命名空间分组。	sum(irate(istio_requests_total{reporter=~“$reporter",destination_mesh_id="$mesh”, connection_security_policy=“mutual_tls”, destination_workload_namespace=~“$namespace", destination_workload=~"$workload”, destination_cluster=~“$dstcluster",response_code!~"5.*", source_workload=~"$srcwl”,

分类	参数	参数介绍	计算方式
			source_workload_namespace=~“$srcns"}[5m])) by (source_workload, source_workload_namespace) / sum(irate(istio_requests_total{reporter=~"$reporter”,destination_mesh_id=“$mesh", connection_security_policy="mutual_tls", destination_workload_namespace=~"$namespace”, destination_workload=~“$workload", destination_cluster=~"$dstcluster”, source_workload=~“$srcwl", source_workload_namespace=~"$srcns”}[5m])) by (source_workload, source_workload_namespace)
	基于源的传入请求时延	展示使用双向 TLS（标志为 ��mTLS）和未使用双向 TLS 的源工作负载的请求持续时间。通过各种百分位数	

分类	参数	参数介绍	计算方式
		P50、P90 和 P99，可以更好地理解不同工作负载下的性能表现	
	基于源的传入请求大小	展示来自不同来源的请求大小的图表，增加了用于计算不同分位数 P50、P90、P95 和 P99 的请求大小。	
	基于源的响应大小	展示响应大小的线型图表，可用于监控来自不同来源的响应大小，使用了 histogram_quantile 函数计算不同分位数的响应字节大小（例如 P50、P90、P95 和 P99）。	
	接收来自传入 TCP 连接的字节数	展示与传入 TCP 连接相关的字节接收情况，分别表示与 mTLS 和非 mTLS 的连接有关的数据。	
	发送到传入 TCP 连接的字节数	展示对发送到传入 TCP 连接的字节的可视化表示。通过与之前的图表结合，可以提供完整的对 TCP 连接的监控视图，同时比较启用和未启用 mTLS 的连接	
出站工作负载	基于目标和响应码的传出请	展示按目的地和响应代码的传出请求，基于 istio_requests_total 指	round(sum(irate(istio_requests_total{destination_principal=~“spiffe.*“,

分类	参数	参数介绍	计算方式
	求	标，，分别表示与 mTLS 和非 mTLS 的请求量数据。	source_workload_namespace=~”$namespace", source_workload=~"$workload”, reporter=“source”, destination_mesh_id=“$mesh", destination_service=~"$dstsvc”}[5m])) by (destination_service, response_code), 0.001)
	基于目标的传出成功率（非 5xx 响应）	展示传出成功请求的比率（非 5xx 响应），分别按使用和不使用双向 TLS 的连接进行分类，并进一步按源工作负载和源命名空间分组。	
	基于目标的传出请求时延	展示对应请求目标传出的请求时长的百分位，支持 50、P90、P95 和 P99。同时支持 mTLS 和非 mTLS 的的安全策略，单位：秒/毫秒。	
	基于目标的传出请求大小	展示在使用 mTLS 和不使用时传出的请求大小，支持支持 50、P90、P95 和 P99。表达式中使用了与 Istio、工作负载、命名空间和目标服务相关的标签进行	

分类	参数	参数介绍	计算方式
		过滤。	
	基于目标的响应大小	展示传出不同目的地的响应大小，分别计算了 P50、P90、P95 和 P99 的值，有的与 mutual_tls 的连接安全策略相关联。	
	传出 TCP 连接发送的字节数	展示通过 TCP 连接发出的字节。通过对比 mTLS（双向 TLS）和非 mTLS 连接，深入了解不同安全策略下的字节发送情况。	
	接收来自传出 TCP 连接的字节数	展示通过目的服务发出的外部 TCP 连接接收的字节量， 分别统计通过 mutual TLS 和不通过 mutual TLS 进行保护的连接的字节总量	示例： round(sum(irate(istio_tcp_sent_bytes_total{reporter=“source”, destination_mesh_id=“$mesh", connection_security_policy!="mutual_tls", source_workload_namespace=~"$namespace”, source_workload=~“$workload", destination_service=~"$dstsvc”}[1m])) by (destination_service), 0.001)

流量监控

流量监控图表采用了 Istio 的原生 Grafana 图表，目前采用了以下 4 个图表板块：

- 全局监控：网格全局的各类资源统计
- 性能监控：网格控制面和数据面组件性能展示，还包含基于组件版本的性能统计信息
- 服务监控：网格内所有注入边车的服务性能统计信息，并提供多种筛选方式，您可以从命名空间、请求来源、服务版本等多个维度筛选显示内容
- 工作负载监控：网格内所有注入边车的工作负载性能统计信息，并提供多种筛选方式，您可以从命名空间、所属服务、请求来源、服务版本等多个纬度筛选显示内容

还提供了流量拓扑，监测不同命名空间、采集源的流量，还增加了流量动画，方便查看动态的流量流转情况。



traffic topology

有关更多信息，请参阅 Istio 的 Grafana Dashboard 说明。

2023-08-18

监控图表阅读指南

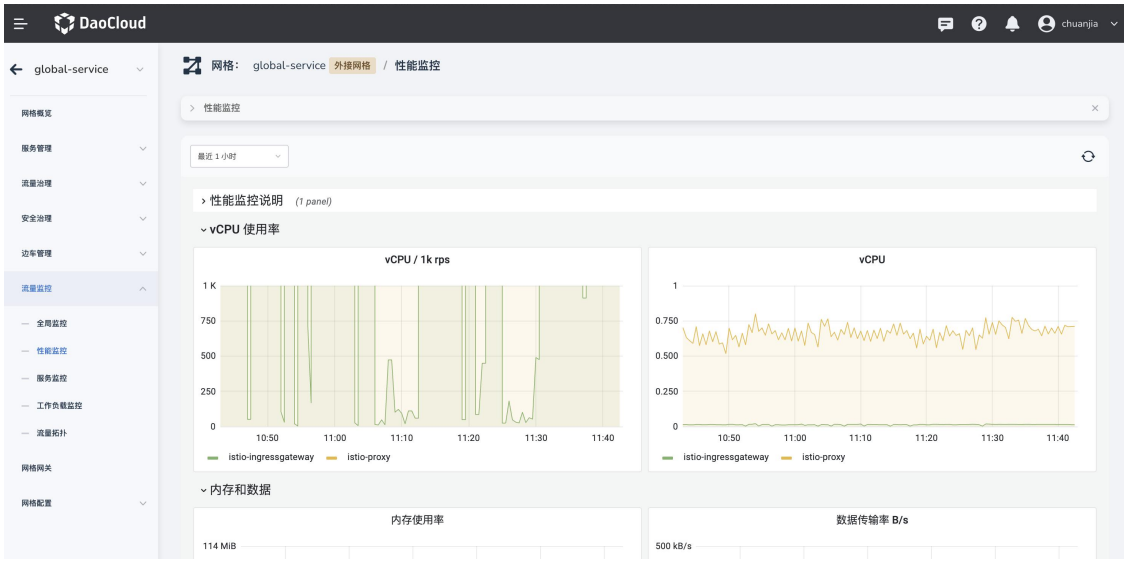
服务网格基于 Istio 开源的 Grafana Dashboard，为了每一个服务网格实例自动生成了流量监控看板。

主要根据以下几个维度进行呈现：

- 全局监控：主要展示网格实例的策略数量、服务与工作负载等全部的资源情况。
- 性能监控：重点展示网格实例的系统资源的性能情况、以及 Sidecar 资源消耗资源等。
- 服务监控：以服务作为主视角，展示服务的流量和工作负载的状态。
- 工作负载监控：以工作负载作为主视角，展示工作负载的资源消耗、业务负载等数据。

图表阅读指南

以性能监控为例，图表中提供了各个指标的监控数据，展示形式主要以数值、图表、表单等来呈现。



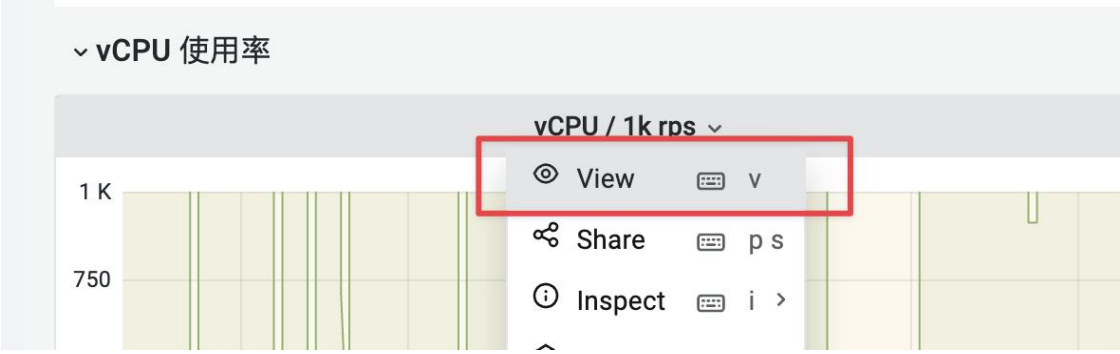
性能监控

图表上方的搜索条提供了监控看板的一些辅助检索字段，基于看板的不同，可用的检索字段也略有不同。

- 时间范围（通用）
- 刷新按钮（通用）
- 命名空间
- 服务名称
- 工作负载
- …

全屏查看指标

点击对应指标图表的标题，可以展开更多的指标操作，其中第一个就是方法查看，可通过键盘 **ESC** 返回。



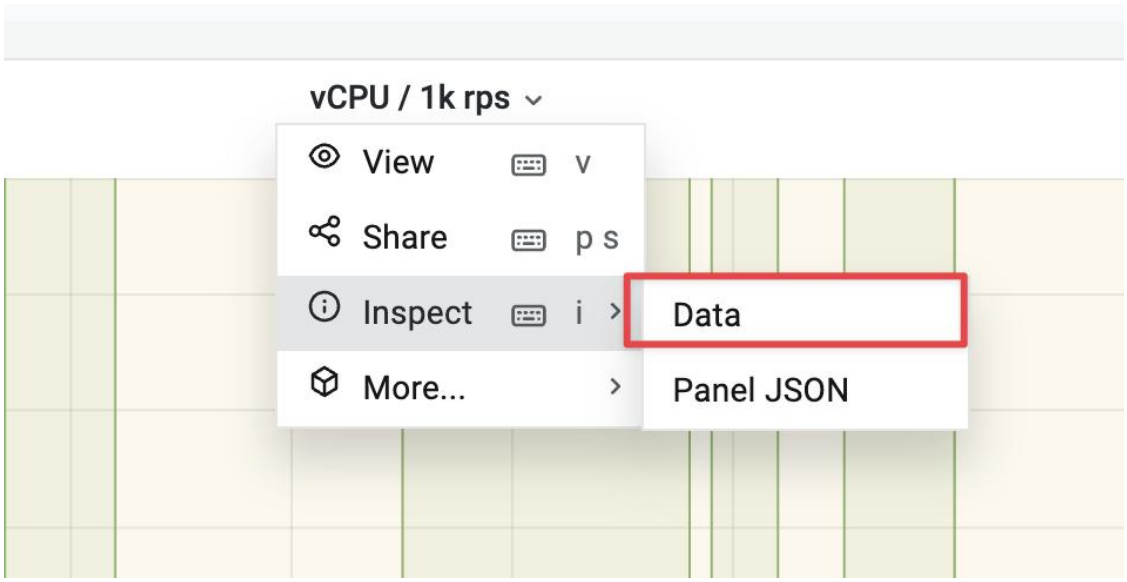
view menu



view big

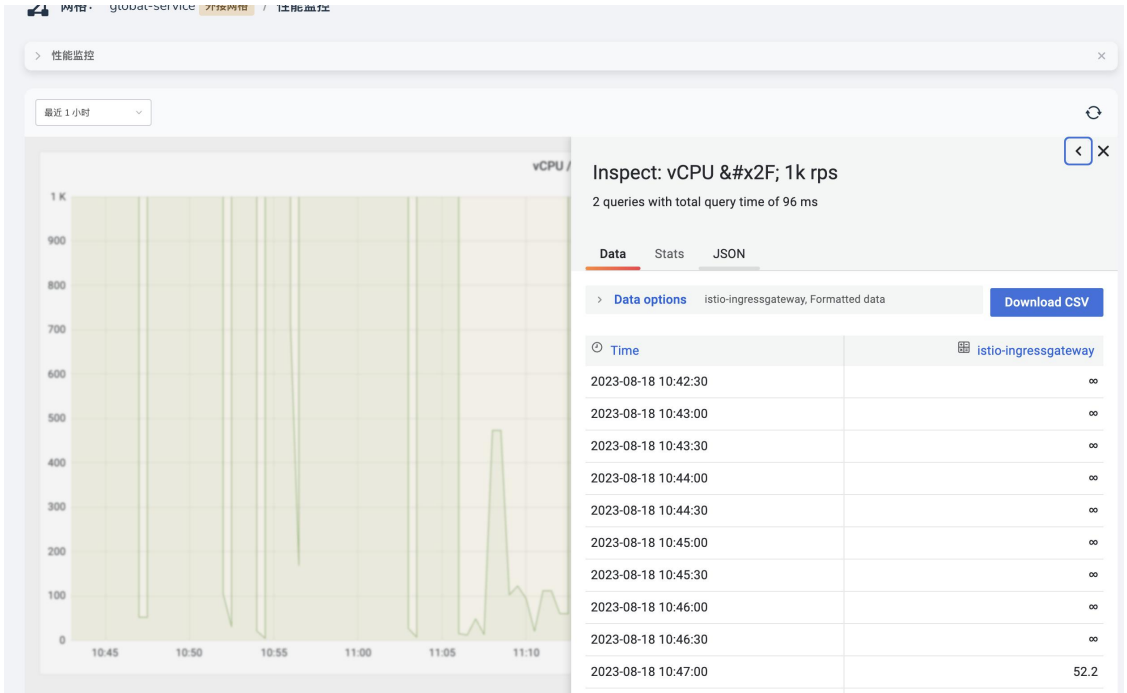
导出指标数据

如果希望查看图表对应的原始数据，可以点击指标图表的标题，然后选择 Inspect > Data，就可以用数据表的形式查看对应的数据。



inspect data

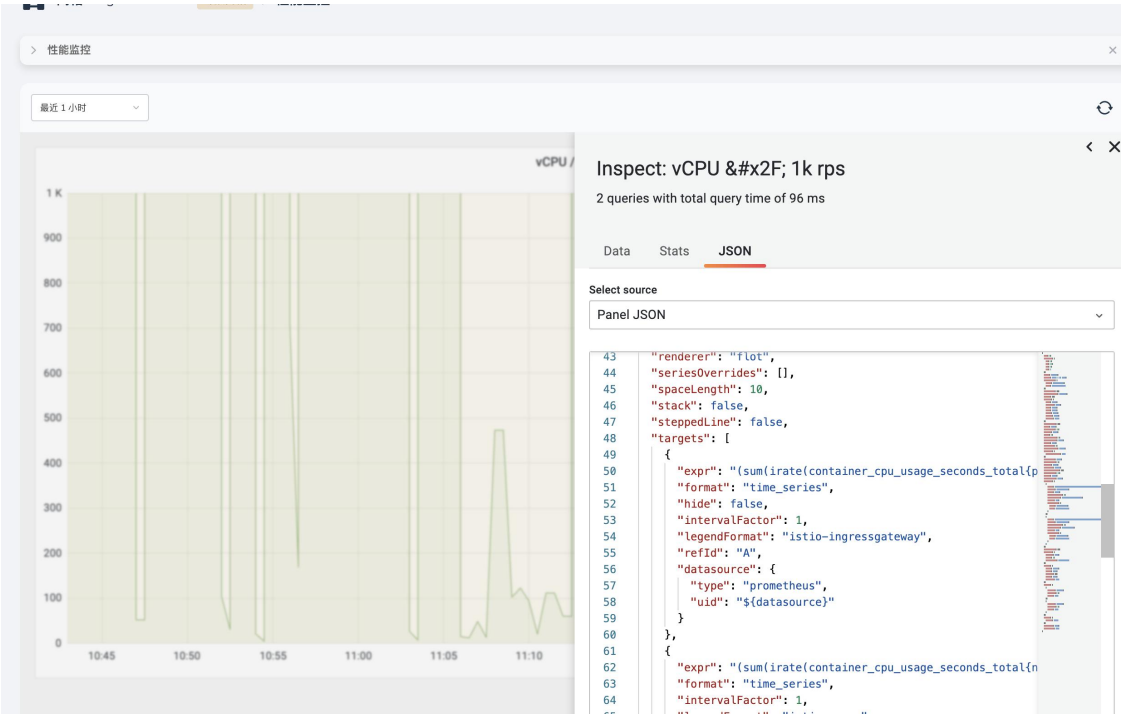
此时，我们可以进行数据的导出，支持 CSV 和 Excel 两种格式



data

查看指标的计算表达式

如果想要详细的了解改指标的计算公式，可以查看指标图表的 Json 结构体，在其中会给出指标的具体计算方式，操作路径如下：

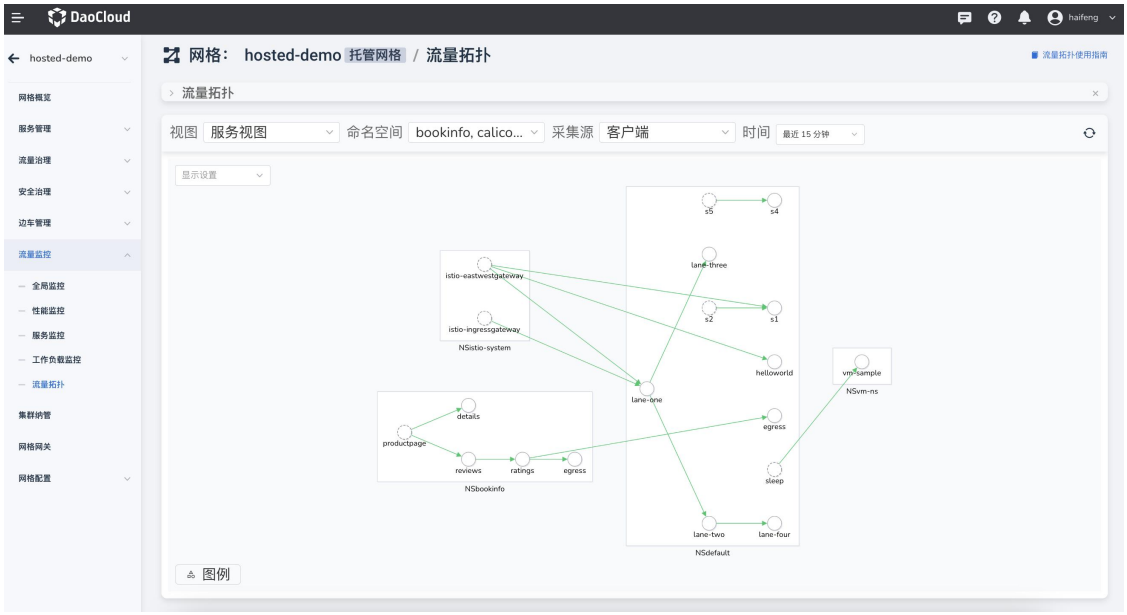


json

流量拓扑

DCE 5.0 服务网格提供给了动态流转的流量拓扑功能。

在左侧导航栏中，点击流量监控 -> 流量拓扑，您可以选择视图方式、命名空间、采集源、时间来查看服务的拓扑关系。



流量拓扑

显示设置

共有 3 个选项：

- 命名空间边界：按命名空间分区显示服务
- 显示空闲节点
- 开启动画：展示流量流转的动态方向



显示设置

服务拓扑图支持平移、缩放等操作。

图例

点击左下角的图例按钮，可以查看当前线条、圆圈、颜色所代表的含义。



图例

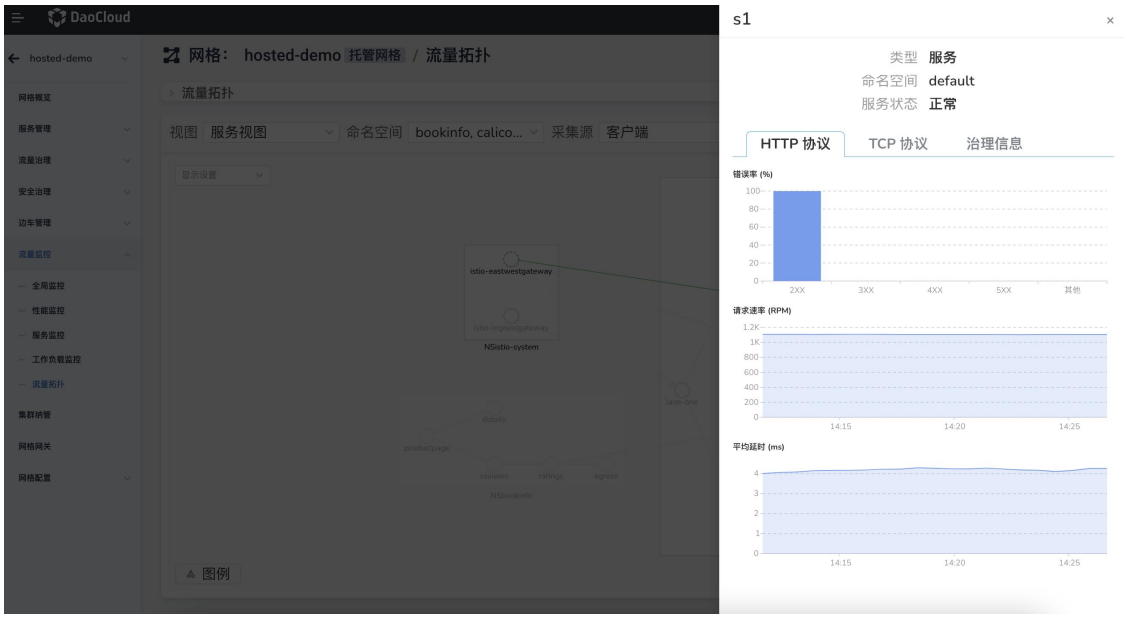
服务用圆圈表示，圆圈的颜色表示了该服务的健康状态：

- 健康（灰色）：错误率 = 0 并且 延时不超过 100 ms
- 警告（橙色）：0 < 错误率 <= 5% 或 100 ms < 延时 <= 200 ms
- 异常（红色）：错误率 > 5% 或延时 > 200 ms
- 未知（虚线）：未获得任何指标数据

服务指标数据信息

点击任意服务，会弹出一个侧边栏，基于协议类型展示服务相关指标：

- HTTP 协议：错误率（%）、请求速率（RPM）、平均延时（ms）
- TCP 协议：连接数（个）、接收吞吐量（B/S）、发送吞吐量（B/S）
- 治理信息：查看治理的虚拟服务、目标服务、网关等



侧边栏

2023-07-28

更改边车日志级别

边车日志，用于记录工作负载的边车的工作情况，通过控制日志级别，可以控制边车日志的输出，从而减少日志的输出，减少日志的存储和传输。

在部署网格实例时，DaoCloud 支持配置全局默认的边车日志级别，默认情况下所有工作负载的边车都会采用此配置。

边车日志级别配置

- 全局默认边车日志级别：默认情况下，全部边车日志的级别，可以在网格实例的边车信息中进行配置
- 临时修改工作负载边车日志级别：适用于单个工作负载的边车日志级别，可通过到工作负载的边车容器内临时修改

全局默认边车日志级别

!!! warning

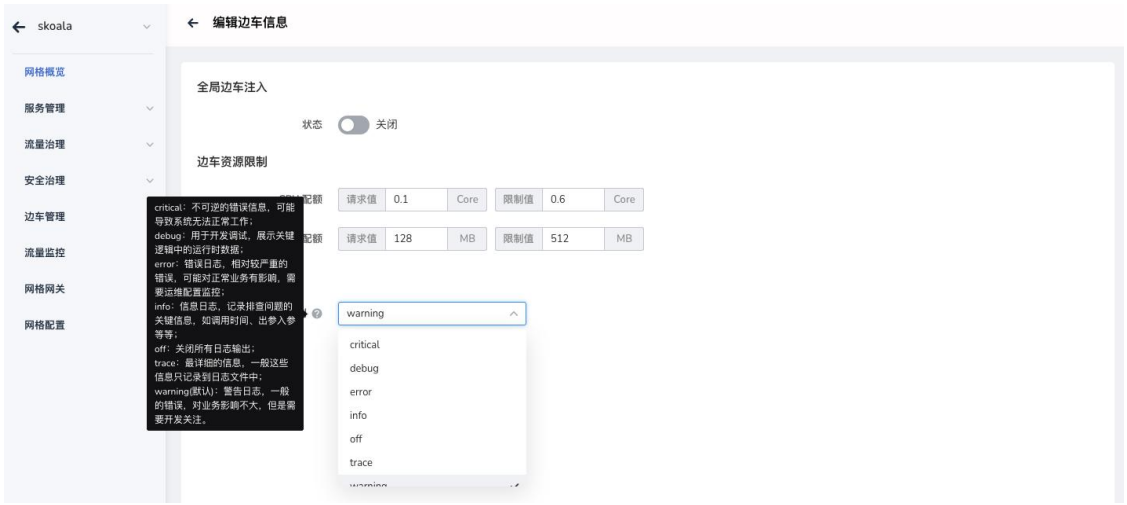
全局默认边车日志级别，需由网格管理员进行修改。

1. 登录 DaoCloud 控制台，进入网格实例详情页，点击边车信息菜单，进入边车信息修改页面



image

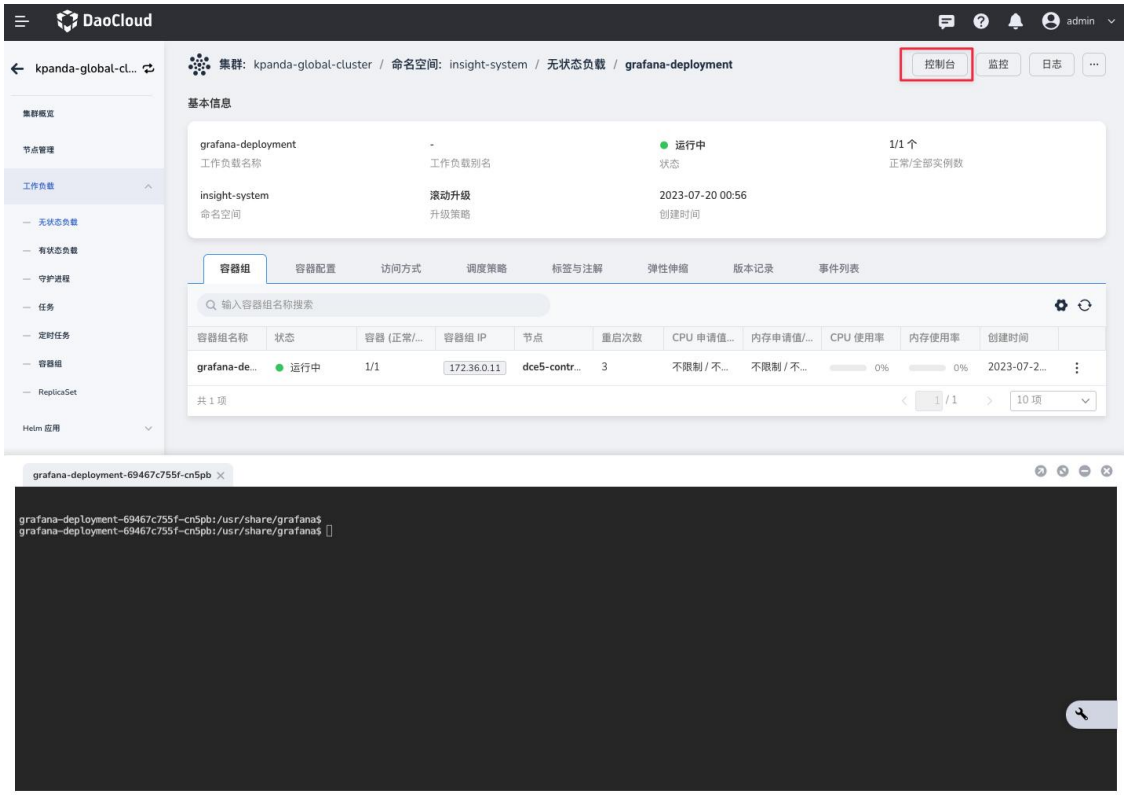
2. 在边车信息修改页面，可以修改全局默认边车日志级别，修改完成后，点击保存按钮，即可保存修改



image

临时修改工作负载边车日志级别

通常情况下我们在临时分析问题时，需要修改某个工作负载的边车日志级别，所以，我们支持通过到工作负载的边车容器内临时修改。



image

这里需要使用 Kubectl 来进行容器边车的配置更新，您需要访问集群控制台，并打开终端，执行如下命令：

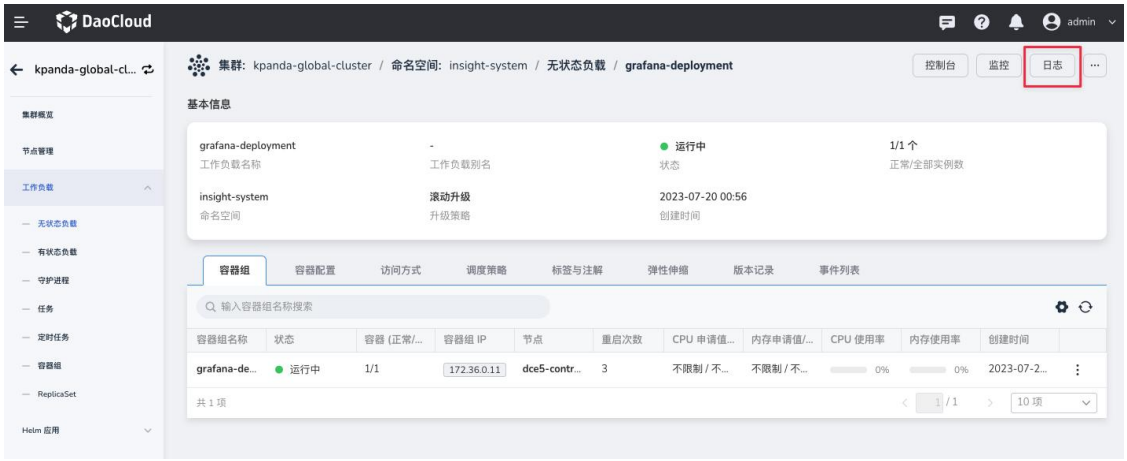
kubectl –n <namespace> exec –it <pod–name> –c istio–proxy –– curl –X POST localhost:15000/logging?level=<log level>

- <namespace>：工作负载所在的命名空间
- <pod–name>：工作负载的 Pod 名称
- <log level>：边车日志级别，可选值为：trace、debug、info、warning、error、critical、off 等
- istio–proxy：边车容器名称，不需要修改
- localhost:15000：边车容器的监听地址，不需要修改

例如，我们需要将 default 命名空间下的 productpage–v1–5b4f8f9b9f–8q9q2 工作负载的边车日志级别修改为 debug，则执行如下命令：

kubectl –n default exec –it productpage–v1–5b4f8f9b9f–8q9q2 –c istio–proxy –– curl –X POST localhost:15000/logging?level=debug

执行完成后，可以在页面点击查看日志，确认边车日志级别是否已经修改成功。



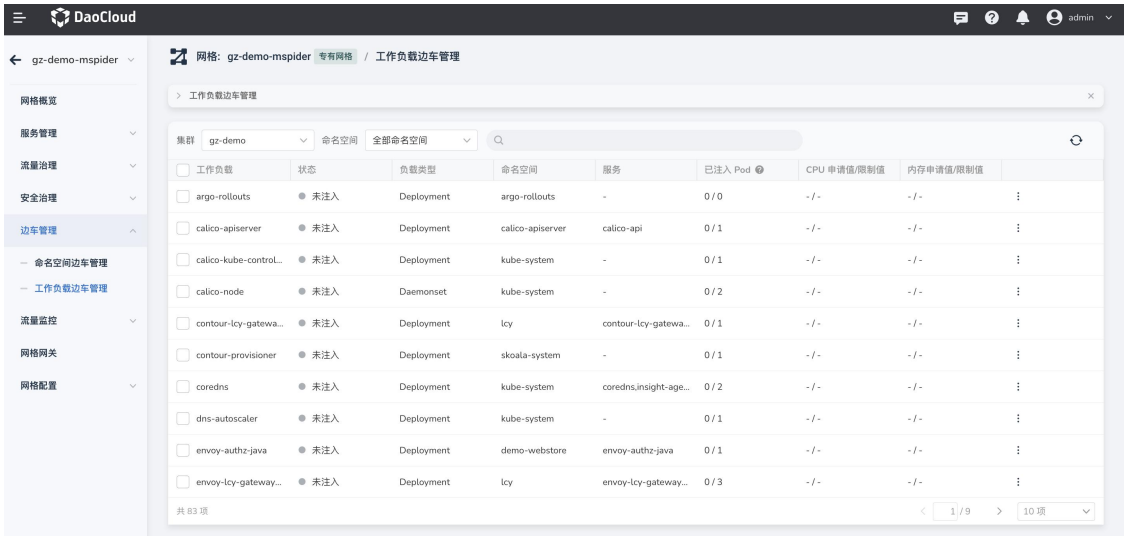
image

工作负载边车管理

您可以对工作负载执行边车注入的查看、启用、禁用等操作，还可以为工作负载设置资源限制。

查看边车注入信息

在左侧导航栏中，点击边车管理 -> 工作负载边车管理，选择一个集群后，可以查看该集群下所有工作负载及其边车注入状态、所属命名空间、资源限制等信息。



工作负载边车列表

各列的含义如下：

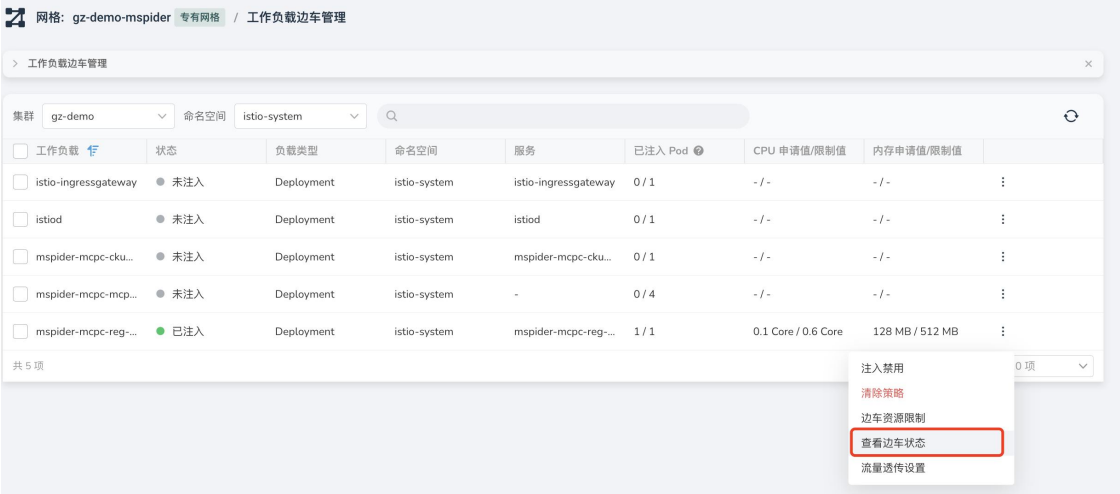
字段名称	含义
工作负载	所选集群下所有工作负载（不包含系统占用不可边车注入的工作负载）
状态	三个状态：已注入 – 已完成边车自动注入未注入 – 禁用边车自动注入待重启 – 所述命名空间的 istio-injection 已发生变化，但相关 Pod 还未重启
命名空间	该工作负载所属命名空间；

字段名称	含义
服务	该工作负载相关服务，该项内容可能有多个，可采用扩缩列表，您点击后浮动窗显示所有内容；
已注入 Pod	该工作负载下 Pod 的注入情况；格式：已注入 Pod/所有可注入 Pod 如果工作负载的注入状态为已注入，但有部分 Pod 没有注入，例如 3/5，该项将高亮显示，提醒您有注入失败的 Pod，需及时处理
CPU 申请值/限制值	包含请求资源和限制资源两个数值，如果您未进行资源设置，该项目为未设置。格式：请求 / 限制
内存申请值/限制值	包含请求资源和限制资源两个数值，如果您未进行资源设置，该项目为未设置。格式：请求 / 限制
操作	注入启用、清除策略、边车资源限制、查看边车状态和流量透传设置等操作菜单

当工作负载较多时，可以对工作负载名称进行排序，并通过搜索功能查找当前集群下的目标工作负载。

查看边车运行状态

如果一个工作负载已注入边车，点击最后一列的 ⋮，在弹出的菜单中选择查看边车状态。



边车运行状态

可以查看边车当前运行状态、资源申请值和限制值。

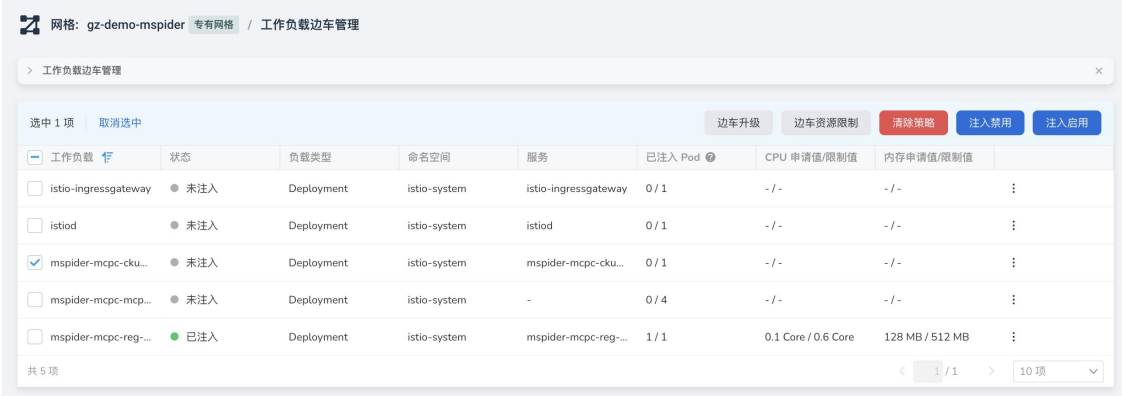


边车运行状态

启用边车注入

选择一个或多个工作负载后，可以启用边车自动注入功能。该操作会重启相关 Pod，因此请在执行该操作前确认 Pod 可以重启。具体操作步骤如下：

1. 选择一个或多个未启用边车注入的工作负载，点击右侧的注入启用。



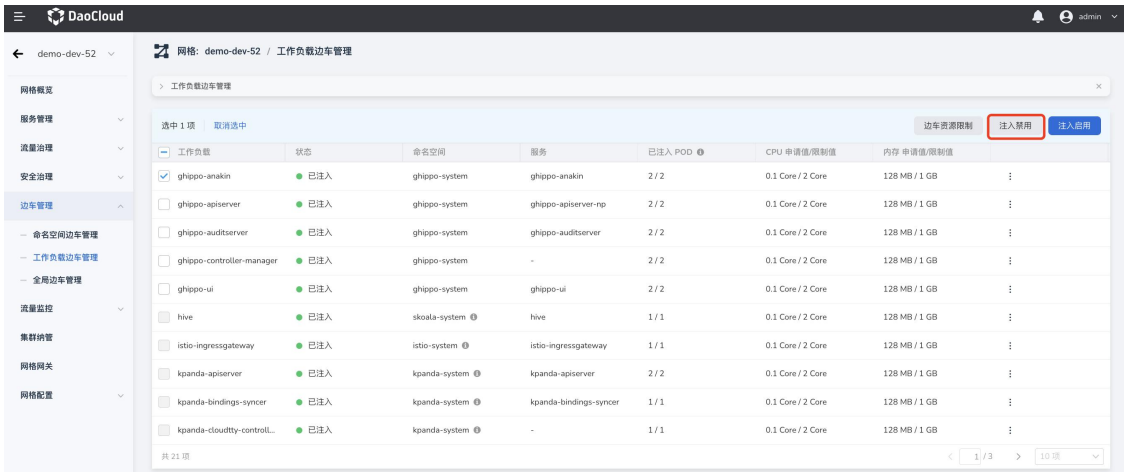
选择边车注入

2. 在弹出对话框中，确认选择的工作负载后，勾选立即重启，点击确定。



选择边车注入

3. 返回工作负载列表，所选工作负载的状态将发生变化，例如从未注入到已注入。 在工作负载完成重启后，将完成边车注入，相关注入进度可查看已注入 Pod 这一列。



边车注入成功

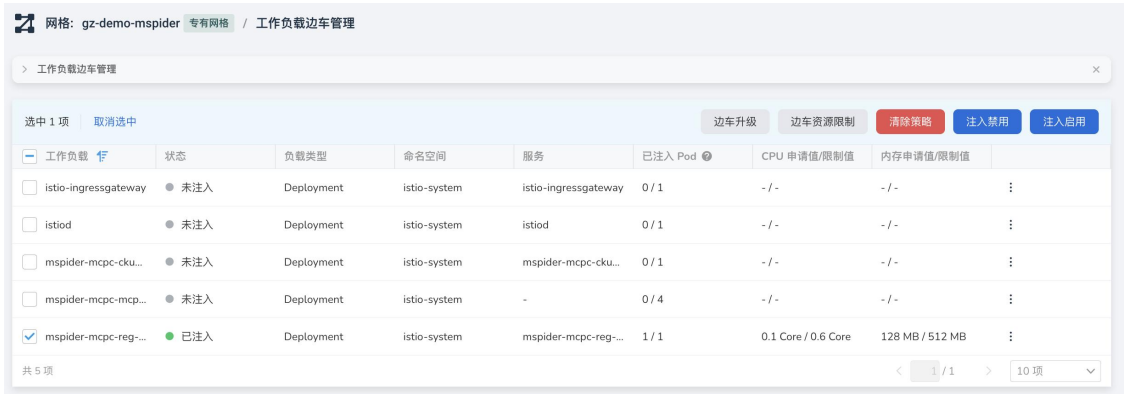
!!! note

如果工作负载所属命名空间已执行`注入启用/禁用`操作但未重启工作负载，则该工作负载将无法执行新的边车相关操作。
需先完成重启，才能执行新的边车操作。

禁用边车注入

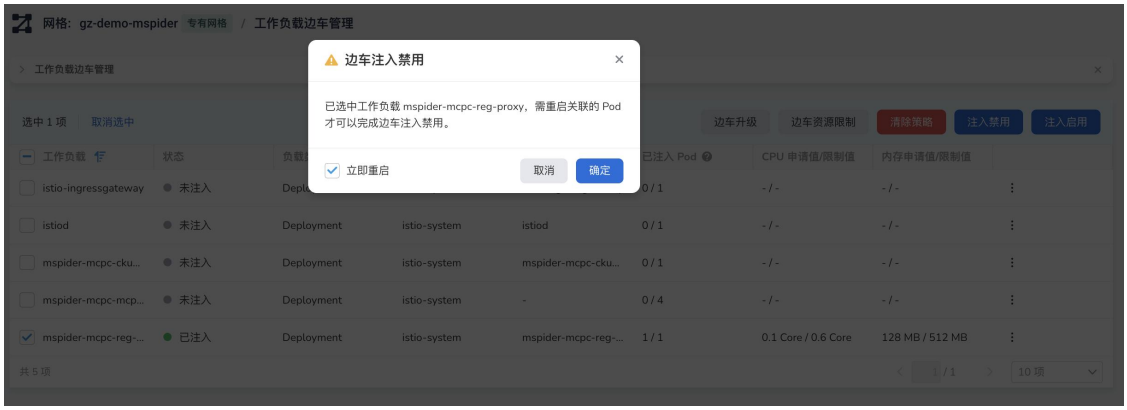
选择一个或多个工作负载后，可以禁用边车自动注入功能。该操作会重启相关 Pod，因此请在执行该操作前确认 Pod 可以重启。具体操作步骤如下：

1. 选择一个或多个已启用边车注入的工作负载，点击右侧的注入禁用。



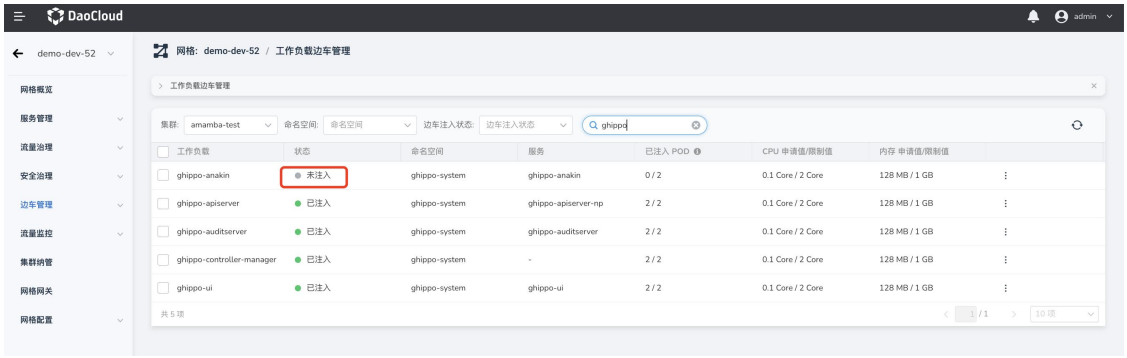
选择边车禁用

2. 在弹出对话框中，确认选择的工作负载数量是否正确，确认无误后，勾选立即重启，点击确定。



选择边车注入

3. 返回工作负载列表，可以看到所选工作负载的状态已变更为未注入。相关卸载进度可查看已注入 Pod 这一列。

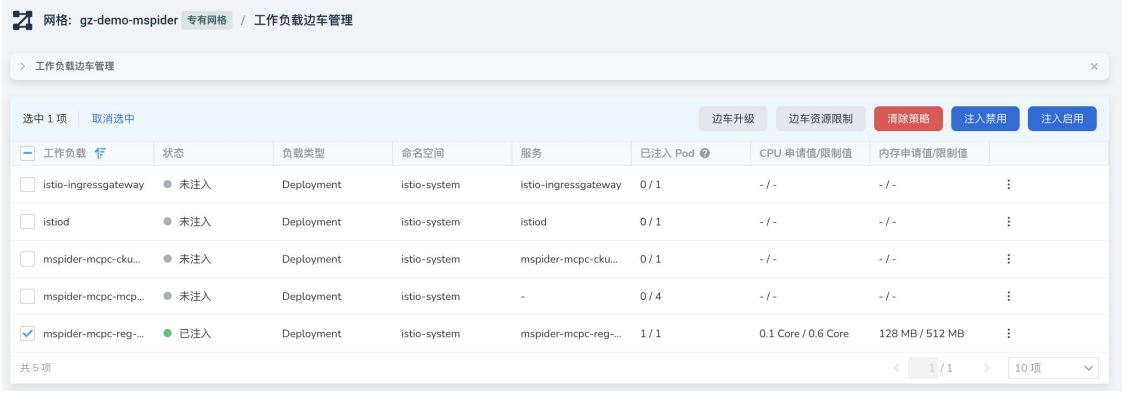


边车禁用成功

边车资源限制

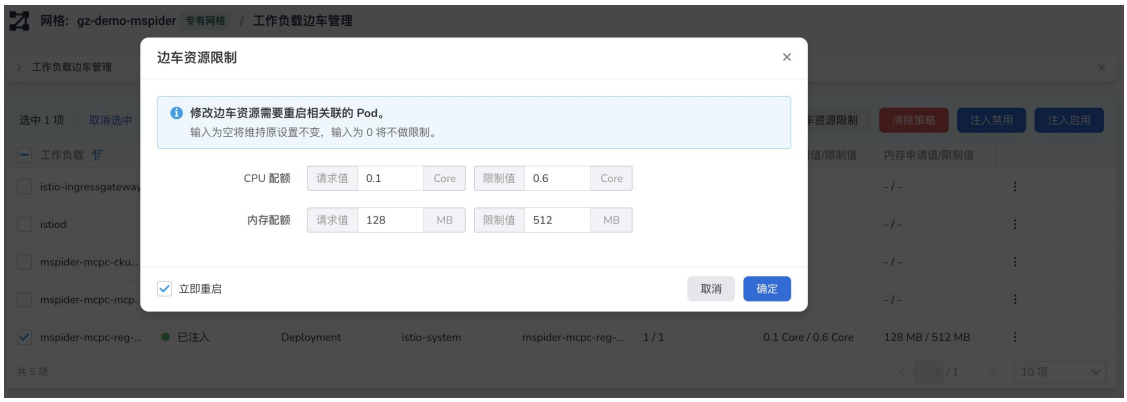
为了防止工作负载的资源被过量使用，可以为工作负载设置资源使用限制。该操作会重启相关 Pod，因此请在执行该操作前确认 Pod 可以重启。具体操作步骤如下：

1. 选择一个（或多个）已启用边车注入的工作负载，点击边车资源限制按钮。



边车资源限制

2. 弹出对话框中，分别设置 CPU/内存的请求值与限制值。选中立即重启，点击确定。



选择边车注入

3. 在工作负载边车管理列表中，可以看到指定工作负载的 CPU 申请值/限制值和内存申请值/限制值内数据已更新。

边车升级

在 DCE 5.0 服务网格中，边车指的是 Envoy 代理，用于实现服务网格中的流量控制和路由规则等功能。 边车升级是指将 Envoy 代理从旧版本升级到新版本。

需要升级边车的原因包括：

1. 安全更新：新版本可能修复了安全漏洞或其他安全问题，为了确保服务网格的安全性，需要将边车升级到最新版本。

2. 功能增强：新版本可能增加了一些新的功能或改进了一些功能，以提高服务网格的性能和可靠性。
3. 错误修复：新版本可能修复了一些错误或 Bug，以提高服务网格的稳定性和可靠性。
4. 版本过时：随着时间的推移，旧版本的 Envoy 代理可能会变得过时并且不再受支持，因此需要将其升级到最新版本以获得更好的支持和维护。

但是，在进行边车升级之前，需要进行充分的测试和验证，以确保升级过程不会对服务网格产生负面影响。

具体操作步骤，请参阅边车版本升级。

命名空间边车管理

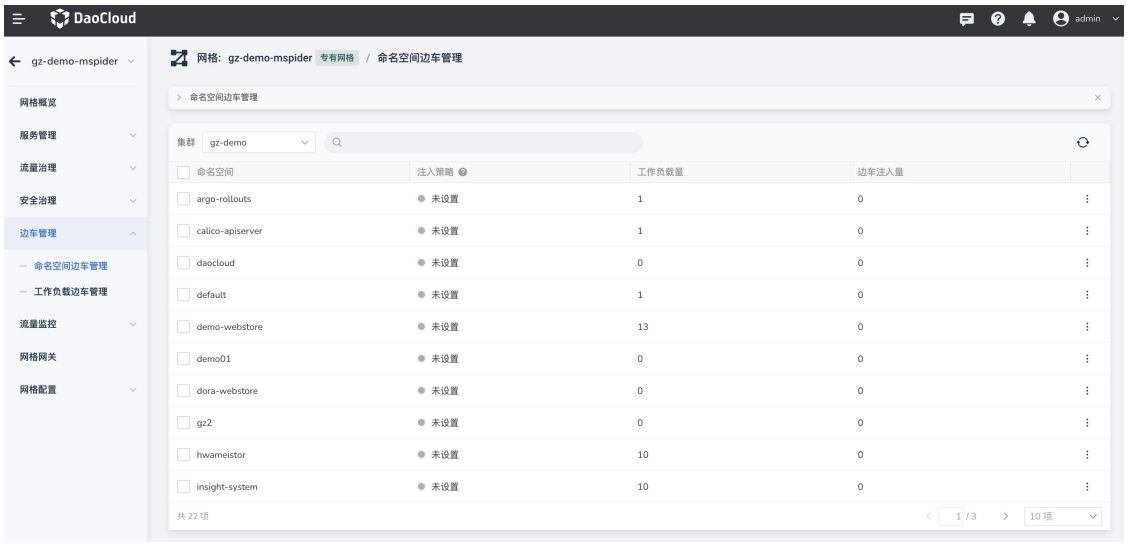
您可以勾选任意集群下的多个命名空间，启用、禁用边车注入或清除策略。

注意：如果某工作负载的边车注入设置为禁用，则该工作负载不会随所属命名空间启用注入。

查看边车注入信息

在左侧导航栏中，点击边车管理 –> 命名空间边车管理，可以查看对应网格下所有命名空间的边车状态。

当命名空间较多时，可以按命名空间的名称进行排序，并通过搜索功能进行查找。

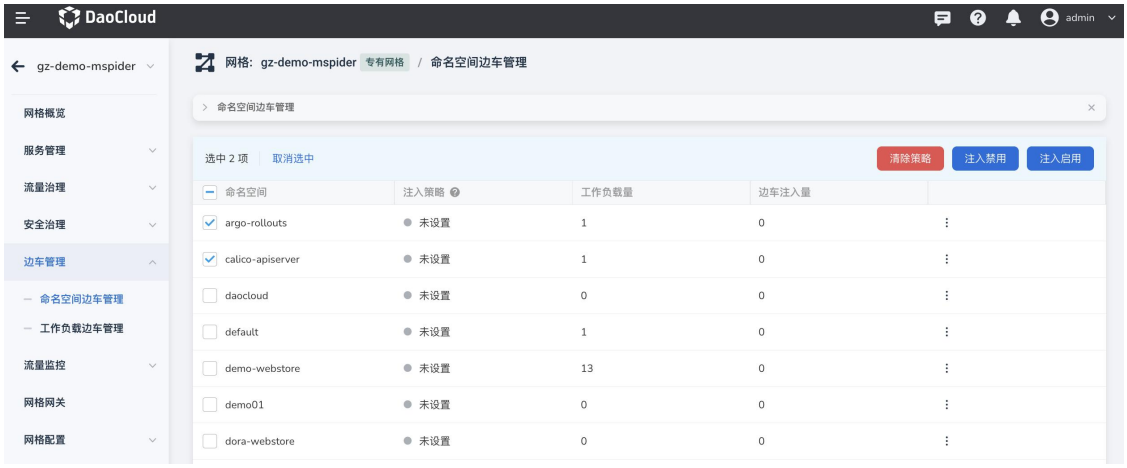


查看边车注入

启用边车注入

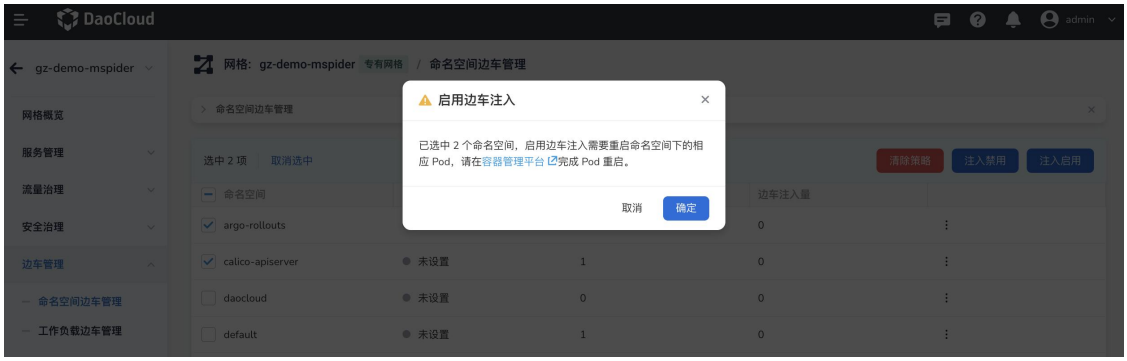
您可以勾选一个或多个命名空间，启用边车注入，具体步骤如下：

1. 勾选一个或多个未注入边车的命名空间，点击注入启用；



点击注入启用

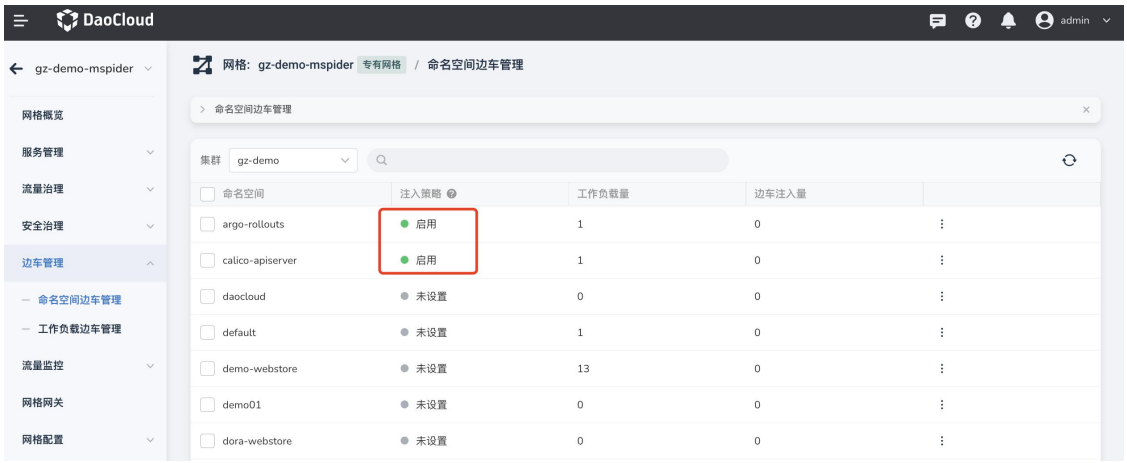
2. 在弹出的对话框中，确认所选的命名空间，点击确定。



确定

请按屏幕提示，重启对应的 Pod。

3. 返回命名空间的边车列表，可以看到刚刚所选命名空间的注入策略状态已变更为启用。在用户完成工作负载的重启后，将完成边车注入，相关注入进度可查看边车注入量一列。



启用

禁用边车注入

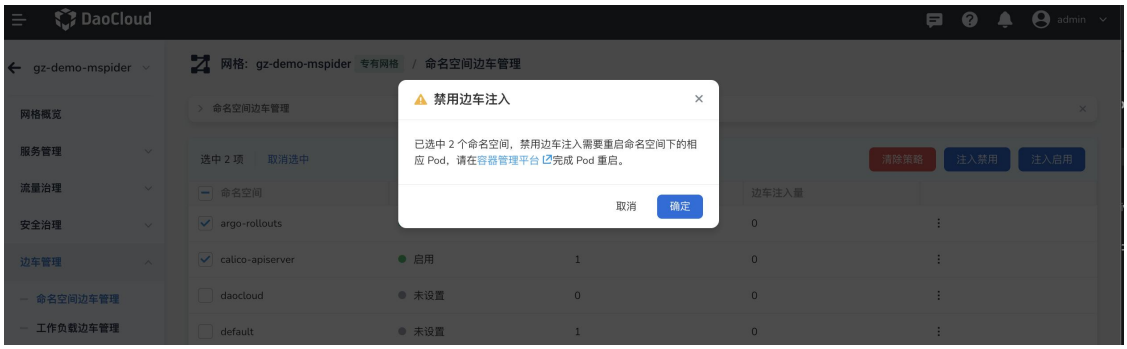
您可以选择一个或多个命名空间，禁用边车注入，具体步骤如下：

1. 勾选一个或多个已启用边车注入的命名空间，点击注入禁用；



注入禁用

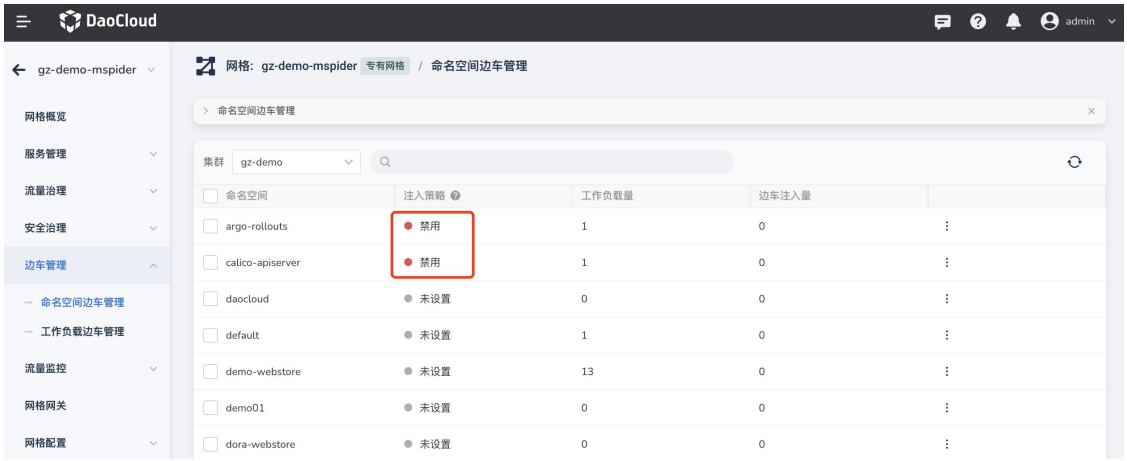
2. 在弹出的对话框中，确认所选的命名空间后，点击确定。



确定

请按屏幕提示，重启对应的 Pod。

3. 返回命名空间的边车列表，可以看到刚刚所选命名空间的注入策略状态已变更为禁用。 在用户完成工作负载的重启后，将完成边车禁用，相关卸载进度可查看边车注入量一列。



禁用

清除策略

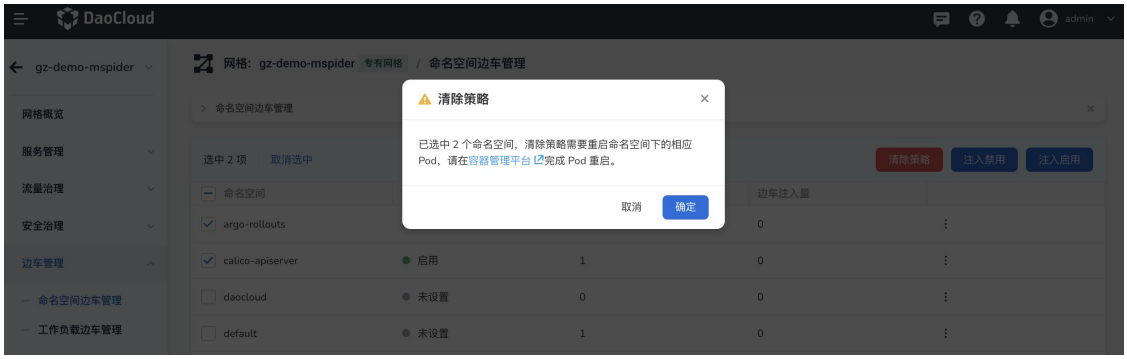
您可以选择一个或多个命名空间，清除命名空间层面的边车策略。 清除后，命名空间下的工作负载的边车状态将仅受工作负载边车管理控制，具体步骤如下：

1. 勾选一个或多个已启用边车注入的命名空间，点击清除策略按钮；



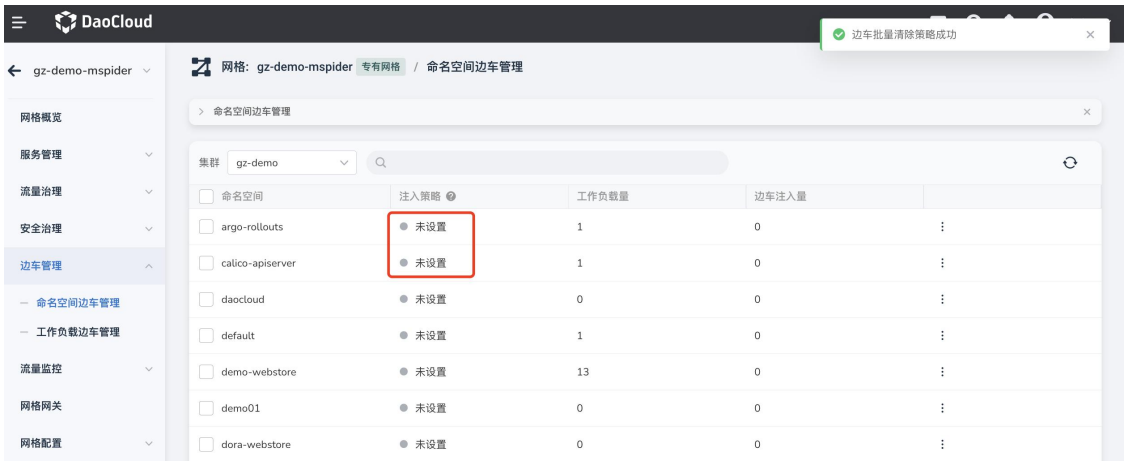
清除策略

2. 在弹出的对话框中，确认所选的命名空间后，点击确定。



确定

3. 返回命名空间的边车列表，可以看到刚刚所选命名空间的注入策略状态已变更为未设置， 此时用户可以在工作负载边车管理为特定工作负载设定边车注入策略。



未设置

下一步：工作负载边车管理

边车流量透传

流量透传（traffic passthrough）指的是工作负载的全部或部分上游、下游流量不经边车转发，直接发送至工作负载本身。

DCE 5.0 服务网格实现了对工作负载出站/入站流量的边车透传可控，可针对特定端口、IP 段实现拦截设置。

- 功能设置对象：工作负载
- 设置参数：端口、IP 段
- 流向：入站、出站

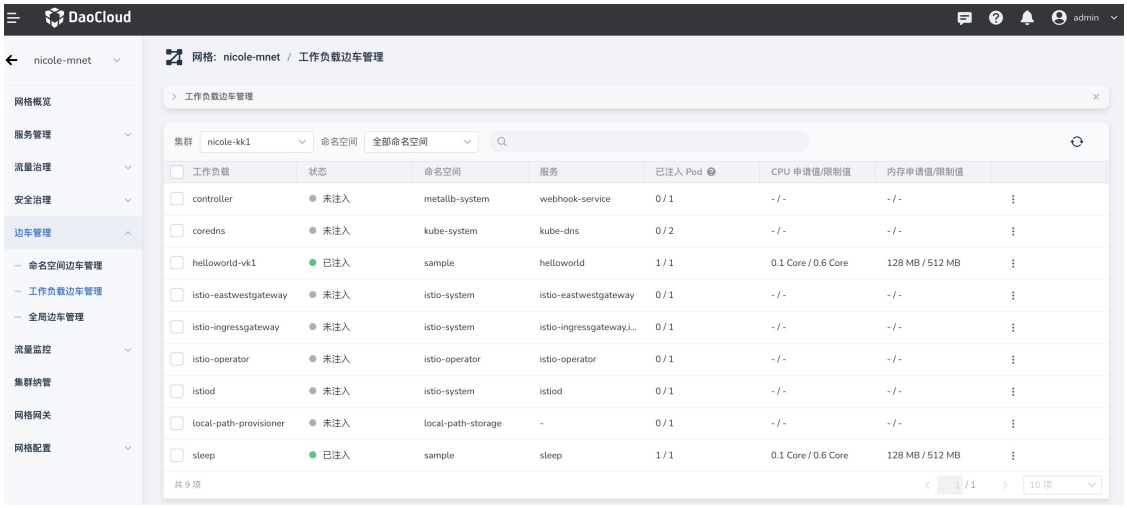
流量透传相关字段：

traffic.sidecar.istio.io/excludeInboundPorts
traffic.sidecar.istio.io/excludeOutboundPorts
traffic.sidecar.istio.io/excludeOutboundIPRanges

启用流量透传

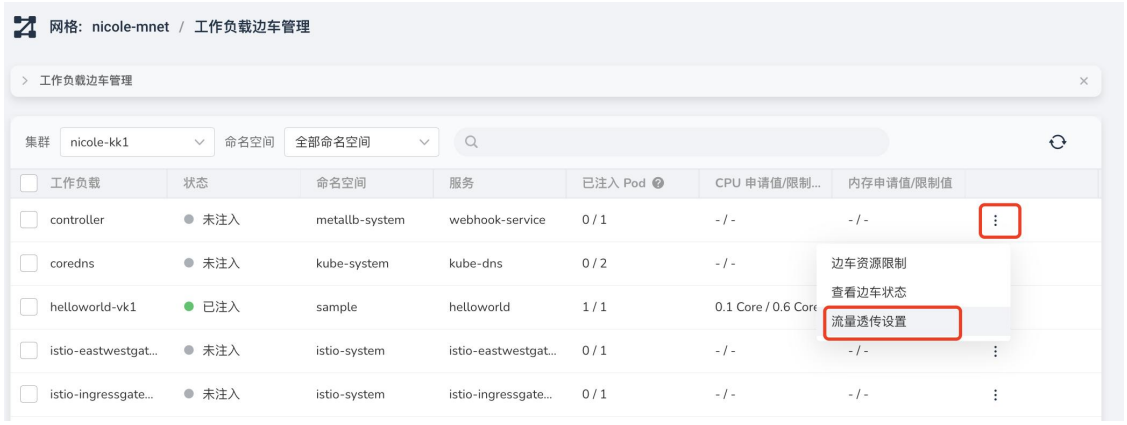
本节说明如何在 DCE 图形界面上启用/禁用流量透传。

1. 进入某个网格，点击边车管理 -> 工作负载边车管理。



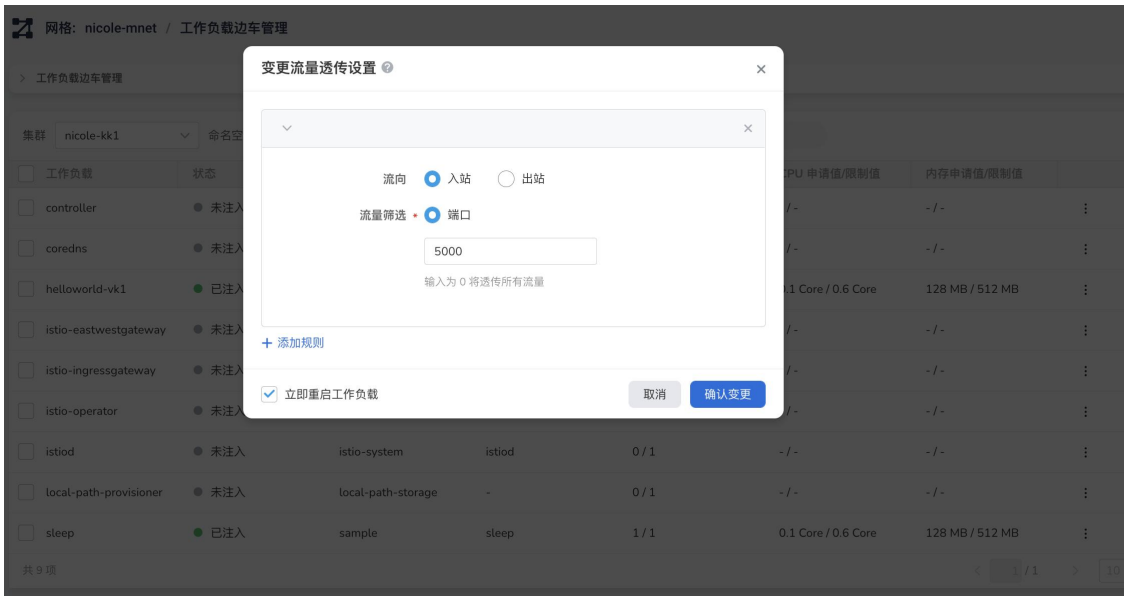
工作负载边车管理

2. 点击某个负载右侧的 ⋮，在弹出菜单中选择流量透传设置。



点击菜单项

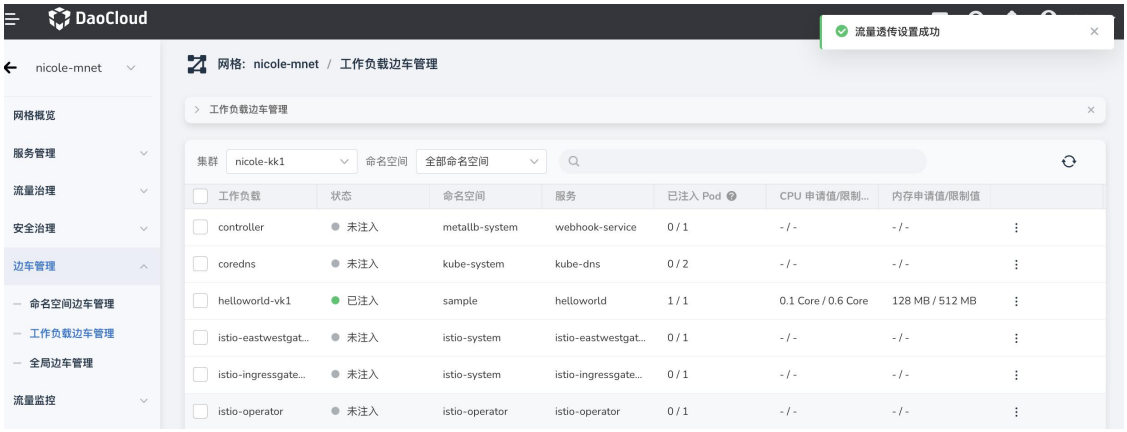
3. 设置流量透传的参数后，勾选立即重启工作负载，点击确认变更。



流量透传设置

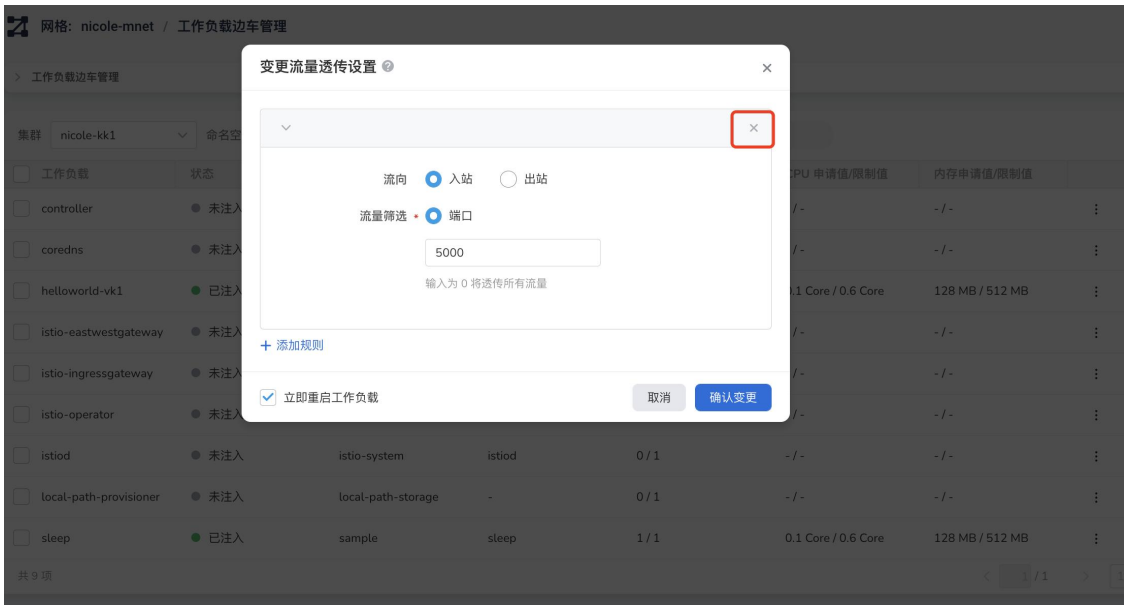
- 入站：仅支持端口，即从外部访问网格内负载的端口
- 出站：可设置目标的端口或 IP 段

4. 如果设置无误，右上角将出现流量透传设置成功的提示消息。您还可以查验流量透传效果。



成功设置

5. 如果流量透传已启用，上述第 3 步的流量透传设置弹窗将显示设置的参数，可点击右侧的 x，勾选立即重启工作负载，点击确认变更来禁用流量透传。



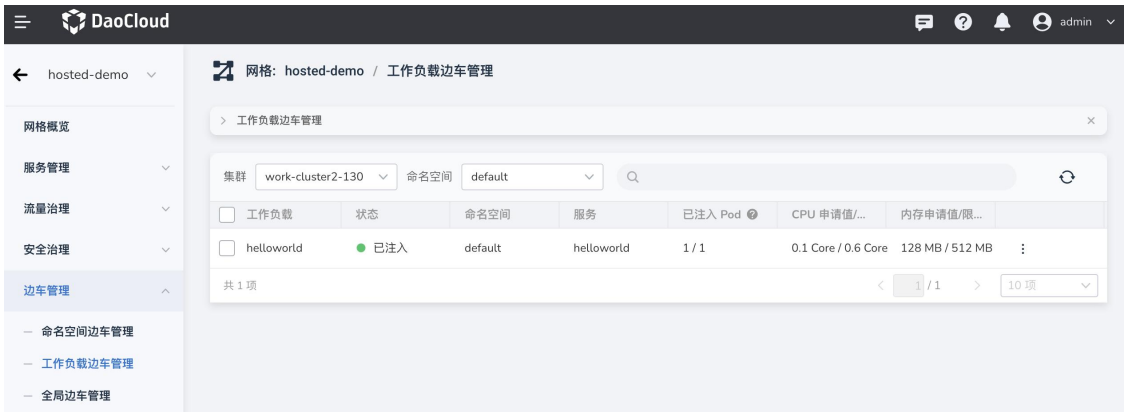
禁用流量透传

查验流量透传效果

在真实的网格集群中，查验流量透传前后的效果。

1. 准备工作

   - 准备一个网格集群，例如 10.64.30.130
   - 在命名空间中，配置工作负载 helloworld，并注入边车
   - 启用流量透传，然后比对该负载的流量路由变化



工作负载

2. 通过控制台或 ssh 登录到网格。

   ssh root@10.64.30.130

3. 查看命名空间中的 svc，获取 clusterIP 和 Port：

   ```
   $ kubectl get svc -n default
   NAME          TYPE          CLUSTER-IP       EXTERNAL-IP   PORT(S)
                 AGE
   helloworld    ClusterIP     10.211.201.221   <none>        5000/TCP
     39d
   kubernetes    ClusterIP     10.211.0.1       <none>        443/TCP
     62d
   test-cv       NodePort      10.211.72.8      <none>        2222:30186/TC
   P     62d
   ```

4. 执行 curl 命令查看 helloworld 的流量路由：

   ===“启用流量透传前”

   ```bash
   $ curl -sSI 10.211.201.221:5000/hello
   HTTP/1.1 200 OK
   content-type: text/html; charset=utf-8
   content-length: 65
   server: istio-envoy # (1)
   date: Tue, 07 Feb 2023 03:08:33 GMT
   x-envoy-upstream-service-time: 100
   x-envoy-decorator-operation: helloworld.default.svc.cluster.local:5000/*
   ```

   1. 流量经过 istio-envoy，即边车的代理

   ===“启用流量透传后”

```bash
$ curl -sSI 10.211.201.221:5000/hello
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 65
Server: Werkzeug/0.12.2 Python/2.7.13 # (1)
Date: Tue, 07 Feb 2023 03:08:10 GMT
```

1. 流量直接进入工作负载本身

虚拟服务参数配置

通过功能界面可以知道，服务网格的路由配置主要分为三个部分：HTTP 路由、TLS 路由和 TCP 路由。本页介绍创建和编辑 **虚拟服务** 时不同路由规则下的具体参数配置。

基本配置

UI 元素	YAML 字段	描述
名称	metadata.name	必填。虚拟服务名称。格式要求：小写字母、数字和中划线（-）组成，必须以小写字母开头，以小写字母或数字结尾，最长 63 个字符。
命名空间	metadata.namespace	必填。虚拟服务所属命名空间。 同一个命名空间内，请求身份认证不可重名。
应用范围	spec.gateways	必填。虚拟服务应用的范围，包含两类：- 指定网关规则（可添加多个），可用于对外暴露网格内部服务；- 对所有边车（-mesh）生效。
所属服务	spec.hosts	必填。应用虚拟服务的服务对象，可包含三类：- 来自 Kubernetes 注册中心的注册服务；- 来自服务条目的注册服务；- 服务域名。

路由配置

- 支持对不同路由类型添加多条，每个类型中执行顺序由上至下，排名在前的路由规则优先生效
- 支持快速拖动排序
- 支持路由名称收起，方便界面编辑时突出当前路由配置的具体信息

HTTP 路由

基本信息

UI 元素	YAML 字段	描述
路由名称	spec.http.–name	必填。http 路由名称。格式要求：小写字母、数字和中划线（–）组成，必须以小写字母开头，以小写字母或数字结尾，最长 63 个字符。

路由匹配规则

路由匹配规则： HTTP 路由用于将 HTTP 请求路由到不同的目标实例，您可以按照 URI、端口以及 Header 等条件进行匹配。例如，您可以将所有请求路径为 /reviews 的请求路由到 reviews 服务上，同时将所有请求路径为 /details 的请求路由到 details 服务上。

必填。YAML 字段为 spec.http.–name.match。

通过 URI 路径、端口等方式对请求做匹配，可添加多条规则，执行顺序由上至下，优先使用排名在前的匹配规则。

UI 元素	YAML 字段	描述
匹配 URI	spec.http.–name.match.uri	可选。对请求的 URI 路径做匹配，有三种匹配方式：– 精确（exact）：字段完全匹配– 前缀（prefix）：对字段前缀的匹配– 正则（regex）：基于 RE2 样式正则表达式的匹配
匹配端口	spec.http.–name.match.port	可选。对请求的端口做匹配。
匹配 header	spec.http.–name.match.header	必填。对请求的 http header 做匹配，同样支持三种匹配方式：– 精确（exact）：字段完全匹配– 前缀（prefix）：对字段前缀的匹配– 正则

UI 元素	YAML 字段	描述
		（regex）：基于 RE2 样式正则表达式的匹配

路由目标/重定向规则

路由目标：路由目标用于将请求路由到不同的目标实例，您可以按照版本、权重等条件进行匹配。例如，您可以将所有请求路径为 /reviews 的请求路由到 reviews 服务的 v1 版本上，同时将所有请求路径为 /reviews 的请求路由到 reviews 服务的 v2 版本上。

路由目标功能和重定向功能为互斥功能，一条 HTTP 路由规则内仅可以二选一。

!!! note

当用户开启`代理`开关后，该项及相关内容置灰。

路由规则

UI 元素	YAML 字段	描述
路由目标	spec.http.–name.route.–destination	可选。已匹配请求的路由目标，可添加多条，优先执行排名在前的路由目标。
服务名称	spec.http.–name.route.–destination.host	必填。路由目标服务的名称或 IP。
版本服	spec.http.–name.route.–destination.subset	可选。列表内容来自所选服务的可用目标规则。

UI 元素	YAML 字段	描述
权重	spec.http.–name.route.weight	可选。本条路由目标内各条所占流量的分配权重。所有路由目标的权重总和应为 100。
端口	spec.http.–name.route.–destination.port.number	可选。路由目标服务的端口。

重定向

UI 元素	YAML 字段	描述
重定向	spec.http.–name.redirect	可选。重定向用于将请求转发至其他路径。
重定向路径	spec.http.–name.redirect.uri	必填。新的访问地址路径（URI）。
authority	spec.http.–name.redirect.authority	可选。URI 路径中认证信息部分，通常 // 表示开始，/ 表示结束。
端口	spec.http.–name.redirect.port.number	可选。重定向服务的端口号。
响应码	spec.http.–name.redirect.redirectCode	可选。指定响应码，当返回指定错误码时，执行重定向操作，默认不填为 301。

可选设置

另外还提供了 6 个可选设置，您可以根据实际需求启用或禁用。

重写

重写功能允许您修改请求或响应的头、URI 或主体等部分。例如，您可以使用重写规则将所有请求中的 URI 路径“/foo”重写为“/bar”，或者将响应头中的 Cache-Control 标头添加到所有响应中。

UI 元素	YAML 字段	描述
重写	spec.http.-name.rewrite	可选。默认关闭 I 可以重现完整路径，也可以仅重写 http 前缀。
重写路径	spec.http.-name.rewrite.uri	必填。新的访问地址路径（URI）。
authority	spec.http.-name.redirect.authority	可选。URI 路径中认证信息部分，通常 // 表示开始，/ 表示结束。

超时

超时规则允许您设置请求的最长等待时间。如果请求在规定的时间内没有得到响应，则该请求将失败。这样可以帮助您避免因某些请求处理较慢而导致其他请求失败的情况。

UI 元素	YAML 字段	描述
超时时长	spec.http.-name.timeout	可选。,用于定义向目标服务发起请求后可容忍的超时时长。输入格式：数字 + 单位（s）。

重试

重试规则允许您指定在发生故障时应该如何重试请求。例如，您可以设置最大的重试次数和间隔时间，以及要重试的错误类型。这样可以帮助您确保请求能够成功处理，从而提高可靠性。

UI 元素	YAML 字段	描述
重试	spec.http.–name.retries	可选。默认关闭。重试功能用于定义当请求反馈为异常时再次发起请求的尝试次数。
重试次数	spec.http.–name.retries.attempts	可选。一个反馈异常的请求的可重试次数，重试间隔默认为 25ms。当重试和超时功能均开启时，重试次数和重试超时的乘积与超时时长取最短者生效，因此实际的重试次数可能会小于设置值。
重试超时	spec.http.–name.retries.perTryTimeout	可选。每次重试的超时时长，默认与超时功能的设置（http.route.timeout）相同。输入格式：数字 + 单位（s）。
重试条件	spec.http.–name.retries.retryOn	可选。允许重试的前提条件，该项下包含多个复选内容：– 5xx：当上游服务器返回 5xx 响应码或无响应（断开/重置/读取超时），envoy 将重试。– refused–stream：当上游服务器用错误码 REFUSED_STREAM 重置数据流，envoy 将重试。– gateway–error：仅针对 502、503、504 错误进行重试。– retriable–status–codes：当上游服务器的返回码与响应码或请求头中 x–envoy–retriable–status–codes 定义相同，则重试。– reset：当上游服务器无响应时（断开/

UI 元素	YAML 字段	描述
		重置/读取超时），将重试。– connect–failure：当与上游服务器的连接失败（连接超时等）导致请求失败，将重试。– retriable–headers：当上游服务器响应码与重试策略中定义匹配或与 x–ENVIGET–retriable–header–NAME 头匹配，将尝试重试。– envoy–ratelimited：当存在报头 x–ENVISENT–ratelimited 时，将重试。– retriable–4xx：当上游服务器返回 4xx 响应码时（目前仅有 409），envoy 将重试。

故障注入

故障注入规则可以模拟错误或延迟，以帮助测试和验证系统的健壮性。例如，您可以使用故障注入规则模拟后端服务返回的错误响应。

UI 元素	YAML 字段	描述
故障注入	spec.http.–name.fault	可选。默认关闭。故障注入功能用于在应用层向目标服务注入故障，并可提供“延迟”和“终止”两种故障类型；使用故障注入时，不能启用超时和重试功能。
延迟时长	spec.http.–name.fault.delay.fixedDelay	必填。请求响应可延迟的时长 输入格式：数字 + 单位（s）。
故障注入占比	spec.http.–name.fault.delay.percent	可选。所有请求中故障注入比率，默认 100%。

UI 元素	YAML 字段	描述
	age	
终止响应码	spec.http.–name.fault.abort.httpStatus	必填。用于终止当前请求的 http 返回码。
故障注入占比	spec.http.–name.fault.abort.percentage	可选。所有请求中故障注入比率，默认 100%。

代理虚拟服务

代理服务规则可用于将请求路由到网格中的代理服务，而非直接路由到后端服务。这样可以帮助您在不影响后端服务的情况下，在请求路径上添加处理逻辑。

YAML 字段为 spec.http.–name.delegate，默认关闭。

- 虚拟服务代理功能可以将路由配置拆分至主从两个虚拟服务中，由主虚拟服务完成基本设置和匹配规则，由代理虚拟服务完成具体路由规则。
- 开启代理功能后，仅主虚拟服务的路由匹配生效，代理虚拟服务的路由匹配规则不需设置。
- 代理虚拟服务的路由规则会和主虚拟服务路由规则合并执行。
- 代理功能不支持嵌套，仅主虚拟服务可以开启代理功能。
- 当虚拟服务有“路由目标/重定向”配置时，不可配置代理。

UI 元素	YAML 字段	描述
代理虚拟服务	spec.http.–name.delegate.name	必填。用于代理的从虚拟服务 注意：一个已配置代理项的虚拟服务不可做为代理，即不可嵌套代理；一个配置了 spec.hosts 字段的

UI 元素	YAML 字段	描述
		虚拟服务不可作为代理。
所属命名空间	spec.http.–name.delegate.namespace	可选。从虚拟服务所属命名空间，默认与主虚拟服务相同。

注意，若需要使用代理虚拟服务配置时，基础配置中的所属服务必须为空。

流量镜像

流量镜像规则可让您将流量复制到指定的目标服务中。这样可以帮助您在不影响主要请求路径的情况下，对流量进行抽样和分析，以便进行异常检测和故障排除。

UI 元素		YAML 字段	描述。
流量镜像	n	spec.http.–name.mirror	可选。默认关闭。用于将请求流量复制至其他目标服务。
镜像至服务	y	spec.http.–name.mirror.host	必填。流量镜像的传输目标服务。
流量镜像占比	n	spec.http.–name.mirror.mirrorPercentage:value	可选。复制的请求流量与原请求流量的比率，默认 100%。
服务版本	n	spec.http.–name.mirror.subset	可选。服务版本列表内容来自当前目标服务的可用“目标规则”。

TLS 路由

TLS 路由用于将加密的 TLS 流量路由到不同的目标实例。您可以根据 SNI 头部的值选择将流量路由到哪个目标实例。例如，您可以将所有 SNI 头部为

example.com 的请求路由到 example 服务上，同时将所有 SNI 头部为 blog.example.com 的请求路由到 blog 服务上。

- 可添加多条，执行顺序由上至下，排名在前的路由规则优先生效
- 多条 TLS 路由规则之间可以拖动排序
- 每条路由可收起，仅显示路由名称

路由匹配规则

YAML 字段为 spec.tls.–name.match。

通过端口（–port）和 SNI（–port.sniHosts）名称方式对请求做匹配，可添加多条规则，执行顺序由上至下，优先使用排名在前的匹配规则。

路由目标

UI 元素	YAML 字段	描述
添加路由目标	spec.tls.–name.route	必填。添加路由目标信息，可添加多条，执行顺序由上至下
服务名称	spec.tls.–name.route.–destination.host	必填。目标服务名称，下拉列表包含当前命名空间下启用 tls 协议的所有服务
端口	spec.tls.–name.route.–destination.port.number	可选。目标服务端口
服务版本	spec.tls.–name.route.–destination.subset	可选。服务版本列表内容来自当前目标服务的可用“目标规则”。
权重	spec.tls.–name.route.–destination.weight	可选。本条“tls 路由”规则内各个“路由目标”所占流量的分配权重，各条规则的权重总和应为 100。

TCP 路由

TCP 路由用于将 TCP 流量路由到不同的目标实例。您可以根据端口号进行匹配。例如，您可以将所有 端口号为:3306 流量路由到 db 服务上，同时将所有端口号为:22 流量路由到 ssh 服务上。

路由匹配规则

UI 元素	YAML 字段	描述
添加路由匹配规则	spec.tcp.–name.match	可选。通过端口（–port）方式对请求做匹配，可添加多条规则，执行顺序由上至下，优先使用排名在前的匹配规则。
端口	spec.tcp.–name.match.–port	必填。tcp 端口号。

路由目标

UI 元素	YAML 字段	描述
添加路由目标	spec.tcp.–name.route	必填。添加路由目标信息，可添加多条，执行顺序由上至下。
服务名称	spec.tcp.–name.route.–destination.host	必填。目标服务名称，下拉列表包含当前命名空间下可用 tcp 协议的所有服务。
端口	spec.tcp.–name.route.–destination.port.number	可选。目标服务端口。
服务版本	spec.tcp.–name.route.–destination.subset	可选。服务版本列表内容来自当前目标服务的可用“目标规则”。
权重	spec.tcp.–name.route.–destination.weight	可选。本条“TCP 路由”规则内各个“路由目标”所占流量的分配权重，各条规则的权重总和应为 100。

网关规则

网关规则（Gateway）用于将服务暴露于网格之外，相较于 Kubernetes 的 ingress 对象，istio-gateway 增加了更多的功能：

- L4-L6 负载均衡
- 对外 mTLS
- SNI 的支持
- 其他 Istio 中已经实现的内部网络功能：Fault Injection、Traffic Shifting、Circuit Breaking、Mirroring

概念介绍

对于 L7 的支持，网关规则通过与虚拟服务配合实现。几个重要主字段如下：

- Selector

  选择用于南北流量的 istio 网关，可以使用多个，也可以与其他规则共用一个。

- Servers

  对外暴露服务的相关信息，包括 hosts（服务名称）、监听端口、协议类型等。

- TLS

  提供对外的 mTLS 协议配置，用户可以启用三种 TLS 模式，并可以自定义 CA 证书等操作。

示例：

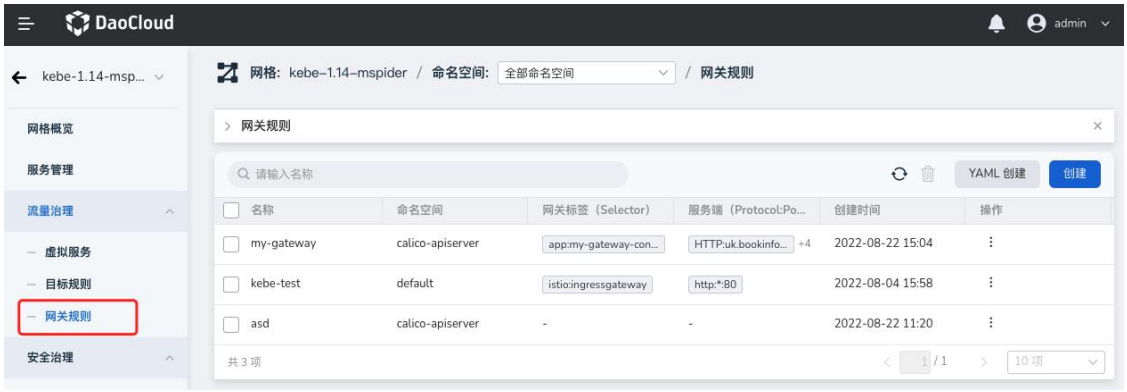
```
spec:
  selector:
```

```
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - istio-grafana.frognew.com
```

操作步骤

服务网格提供了两种创建方式：向导和 YAML。通过向导创建的具体操作步骤如下：

1. 在左侧导航栏点击流量治理 -> 网关规则，点击右上角的创建按钮。



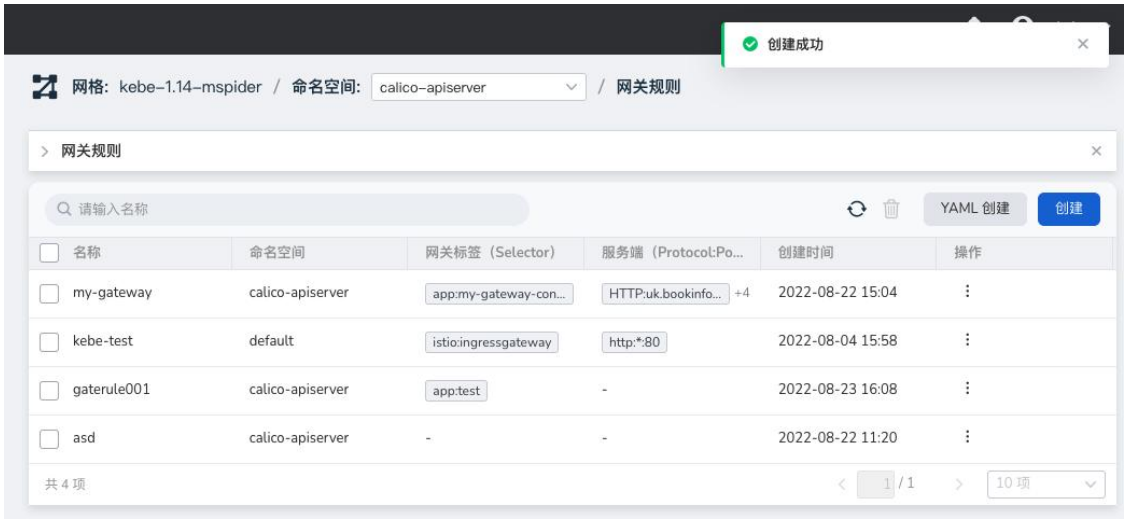
创建

2. 在创建网关规则界面中，配置基本信息，并根据需要添加服务端后点击确定。



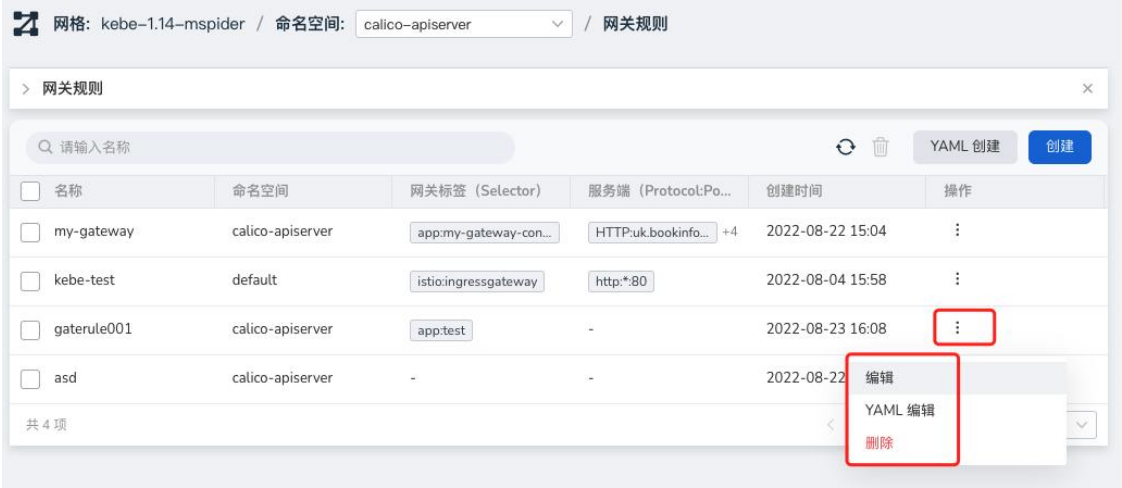
创建网关规则

3. 返回网关规则列表，屏幕提示创建成功。



创建成功

4. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。


网格: kebe-1.14-mspider / 命名空间: calico-apiserver / 网关规则
网关规则
请输入名称
YAML 创建
创建
名称
命名空间
网关标签（Selector）
服务端（Protocol:Po...
创建时间
操作
my-gateway
calico-apiserver
app:my-gateway-con...
HTTP:uk.bookinfo...
+4
2022-08-22 15:04
kebe-test
default
istio:ingressgateway
http:*:80
2022-08-04 15:58
gaterule001
calico-apiserver
app:test
-
2022-08-23 16:08
asd
calico-apiserver
-
-
2022-08-22
编辑
YAML 编辑
删除
共 4 项

更多操作

流量治理

流量治理为用户提供了三种资源配置，虚拟服务、目标规则、网关规则。 通过配置相应规则可以实现路由、重定向、离群检测、分流等多项流量治理功能。用户可以通过向导或 YAML 形式创建、编辑治理策略。

- 虚拟服务主要用于对请求流量的路由定制规则，并可以对数据流做出分流、重定向、超时返回等处理
- 目标规则则更关注流量本身的治理，为请求流量提供更强大的负载均衡、连接存活探寻、离群检测等功能
- 网关规则为 Istio 网关提供服务在网关的暴露方式

在实际运用中，需要三类策略配合使用：

- 由虚拟服务来定义路由规则和描述满足条件的请求去哪里
- 目标规则则是定义子集和策略，描述到达目标的请求该怎么处理
- 如果有外部服务通信需求，则需要配置网关规则中的端口映射等细节来实现需求

在服务网格中，为用户提供了向导和 YAML 两种创建/编辑形式，用户可以根据个人习惯自由选择。

- 向导创建方式为用户提供简单直观的互动方式，一定程度上降低了用户的学习成本
- YAML 创建形式更适合资深用户，用户可以直接编写 YAML 文件创建治理策略，并且创建窗口中也为用户提供了较为常用的治理策略模板，提高用户编写速度

虚拟服务

在虚拟服务（VirtualService）中，可以通过多种匹配方式（端口、host、header 等）实现对不同的地域、用户请求做路由转发，分发至特定的服务版本中，并按权重比划分负载。

虚拟服务提供了 HTTP、TCP、TLS 三种协议的路由支持。

虚拟服务列表

虚拟服务列表展示了网格下的虚拟服务 CRD 信息，用户可以按命名空间查看，也可以基于作用范围、规则标签做 CRD 筛选，规则标签如下：

- HTTP 路由
- TCP 路由
- TLS 路由
- 重写
- 重定向
- 重试
- 超时
- 故障注入
- 代理服务
- 流量镜像

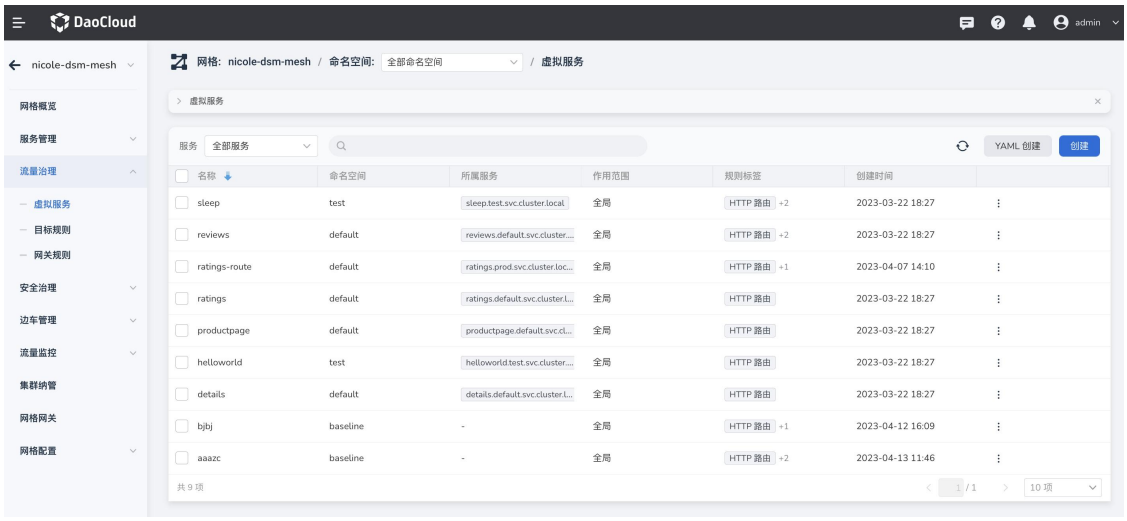
这些标签的字段配置，请参阅 Istio 虚拟服务参数配置。

虚拟服务提供了两种创建方式：图形向导创建和 YAML 创建。

图形向导创建步骤

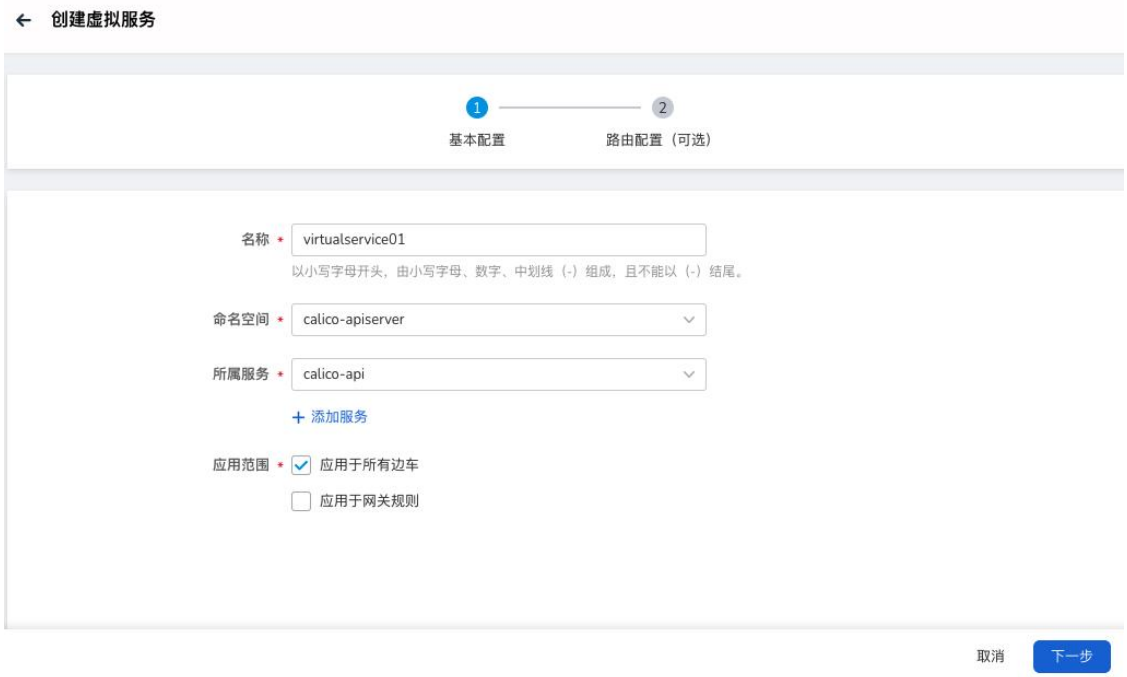
通过图形向导创建的具体操作步骤如下（参阅虚拟服务参数配置）：

1. 在左侧导航栏点击流量治理 -> 虚拟服务，点击右上角的创建按钮。



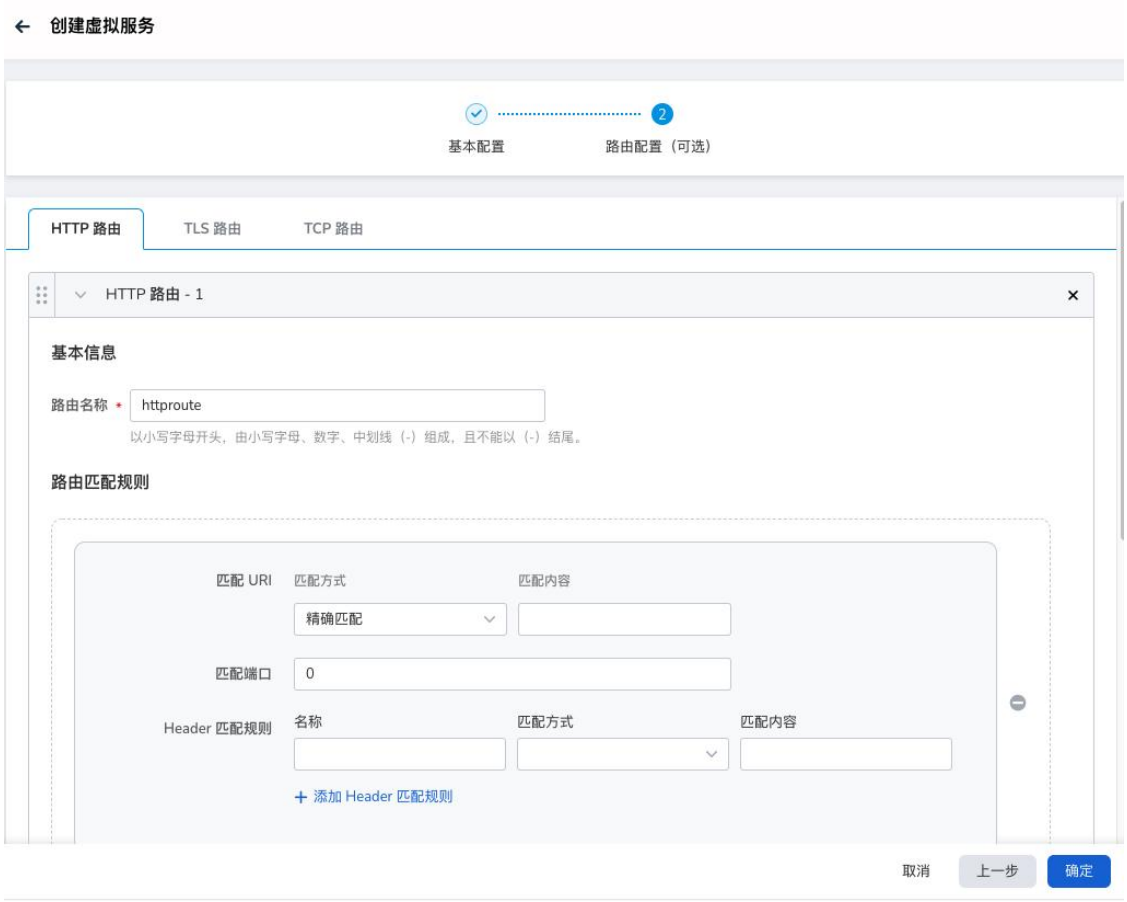
创建

2. 在创建虚拟服务界面中，先确认并选择需要将虚拟服务创建到的命名空间、所属服务和应用范围后，点击下一步。



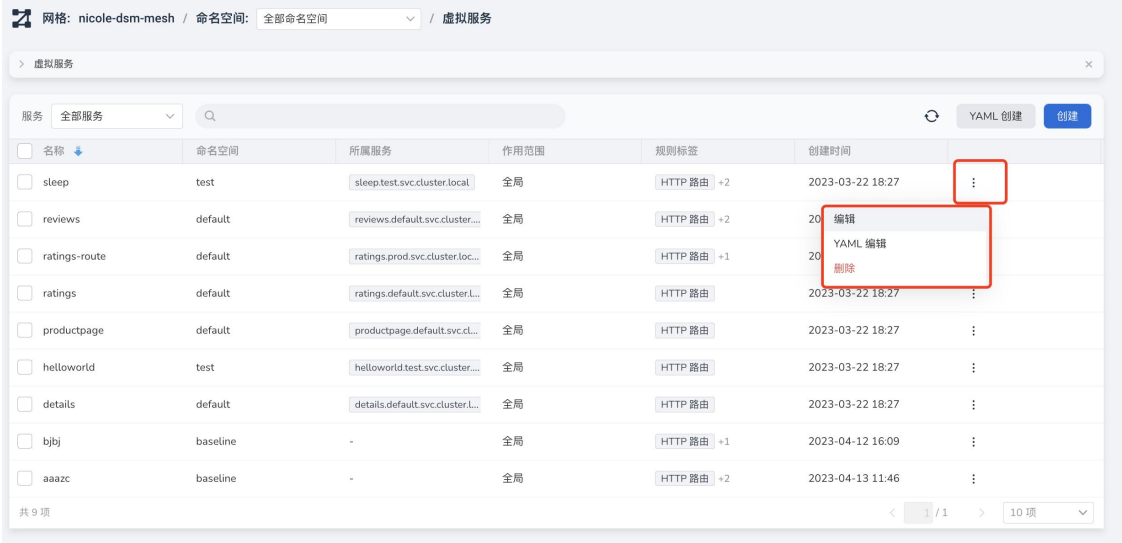
创建虚拟服务

3. 按屏幕提示分别配置 HTTP 路由、TLS 路由和 TCP 路由后，点击确定。



路由配置

4. 返回虚拟服务列表，屏幕提示创建成功。在虚拟服务列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

YAML 创建

通过 YAML 创建的操作相对简单，用户可以点击 YAML 创建按钮，进入创建页面直接编写 YAML，也可以使用页面内提供的模板简化编辑操作， 编辑窗口会提供基本语法检查功能，帮助用户完成编写工作。以下是一个 YAML 示例：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  annotations:
    ckube.daocloud.io/cluster: dywtest3
    ckube.daocloud.io/indexes: '{"activePolices":"HTTP_ROUTE,RETRIES,","cluster":"dywtest3","createdAt":"2023-08-07T09:27:48Z","gateway":"nginx-gw/nginx-gwrule","gateways":"[\"nginx-gw/nginx-gwrule\"]","hosts":"[\"www.nginx.app.com\"]","is_deleted":"false","labels":"","name":"nginx-vs","namespace":"nginx-gw"}'
  creationTimestamp: "2023-08-07T09:27:48Z"
  generation: 10
  managedFields:
    - apiVersion: networking.istio.io/v1beta1
      fieldsType: FieldsV1
```

```
      fieldsV1:
        f:metadata:
          f:annotations:
            .: {}
            f:ckube.daocloud.io/cluster: {}
            f:ckube.daocloud.io/indexes: {}
        f:spec:
          .: {}
          f:gateways: {}
          f:hosts: {}
          f:http: {}
      manager: cacheproxy
      operation: Update
      time: "2023-08-09T03:06:31Z"
  name: nginx-vs
  namespace: nginx-gw
  resourceVersion: "477662"
  uid: 446e8dcf-3c26-47ec-8754-997c21e4df17
spec:
  gateways:
    - nginx-gw/nginx-gwrule
  hosts:
    - www.nginx.app.com
  http:
    - match:
        - uri:
            prefix: /
      name: nginx-http
      retries:
        attempts: 2
        perTryTimeout: 5s
        retryOn: 5xx
```

```
    route:
      - destination:
          host: nginx.nginx-test.svc.cluster.local
          port:
            number: 80
status: {}
```

概念介绍

- hosts

  流量的目标主机。可以来自服务注册信息，服务条目（service entry），或用户自定义的服务域名。可以是带有通配符前缀的 DNS 名称，也可以是 IP 地址。 根据所在平台情况，还可能使用短名称来代替 FQDN。这种场景下，短名称到 FQDN 的具体转换过程是要靠下层平台完成的。

  一个主机名只能在一个 VirtualService 中定义。同一个 VirtualService 中可以用于控制多个 HTTP 和 TCP 端口的流量属性。

  需要注意的是，当使用服务的短名称时（例如使用 reviews，而不是 reviews.default.svc.cluster.local），服务网格会根据规则所在的命名空间来处理这一名称，而非服务所在的命名空间。 假设 default 命名空间的一条规则中包含了一个 reviews 的 host 引用，就会被视为 reviews.default.svc.cluster.local，而不会考虑 reviews 服务所在的命名空间。

  为了避免可能的错误配置，建议使用 FQDN 来进行服务引用。 hosts 字段对 HTTP 和 TCP 服务都是有效的。 网格中的服务也就是在服务注册表中注册的服务，必须使用他们的注册名进行引用；只有 Gateway 定义的服务才可以使用 IP 地址。

  示例：

```
spec:
  hosts:
  - ratings.prod.svc.cluster.local
```

- Gateways

  通过将 VirtualService 绑定到同一 Host 的网关规则，可向网格外部暴露这些 Host。

  网格使用默认保留字 mesh 指代网格中的所有 Sidecar。 当这一字段被省略时，就会使用默认值（mesh），也就是针对网格中的所有 Sidecar 生效。 如果为 gateways 字段设置了网关规则（可以有多个），就只会应用到声明的网关规则中。 如果想同时对网关规则和所有服务生效，需要显式的将 mesh 加入 gateways 列表。

  示例：

```
gateways:
- bookinfo-gateway
- mesh
```

- HTTP

  有序规则列表。该字段包含了针对 HTTP 协议的所有路由配置功能，对名称前缀为 http-、http2-、grpc- 的服务端口，或者协议为 HTTP、HTTP2、GRPC 以及终结的 TLS， 另外还有使用 HTTP、HTTP2 以及 GRPC 协议的 ServiceEntry 都是有效的。 流量会使用匹配到的第一条规则。

  HTTP 主要字段说明：

  - Match

匹配要激活的规则要满足的条件。单个匹配块内的所有条件都具有 AND 语义，而匹配块列表具有 OR 语义。 如果任何一个匹配块成功，则匹配该规则。

- Route

  HTTP 规则可以重定向或转发（默认）流量。

- Redirect

  HTTP 规则可以重定向或转发（默认）流量。 如果在规则中指定了流量通过选项，则将忽略路由/重定向。 重定向原语可用于将 HTTP 301 重定向发送到其他 URI 或 Authority。

- Rewrite

  重写 HTTP URI 和 Authority header，重写不能与重定向原语一起使用。

- Fault

  故障注入策略，适用于客户端的 HTTP 通信。 如果在客户端启用了故障注入策略，则不会启用超时或重试。

- Mirror/MirrorPercent

  将 HTTP 流量镜像到另一个目标，并可以设置镜像比例。

- TCP

  一个针对透传 TCP 流量的有序路由列表。 TCP 路由对所有 HTTP 和 TLS 之外的端口生效。 进入流量会使用匹配到的第一条规则。

- TLS

一个有序列表，对应的是透传 TLS 和 HTTPS 流量。路由过程通常利用 ClientHello 消息中的 SNI 来完成。 TLS 路由通常应用在 https-、tls- 前缀的平台服务端口，或者经 Gateway 透传的 HTTPS、TLS 协议端口，以及使用 HTTPS 或者 TLS 协议的 ServiceEntry 端口上。 注意：没有关联 VirtualService 的 https- 或者 tls- 端口流量会被视为透传 TCP 流量。

Tcp 协议和 tls 的子字段相对简单，仅包含 match 和 route 两部分，并且与 HTTP 相似，不再累述。

参考资料

- 什么是虚拟服务?

目标规则

目标规则（DestinationRule）同样是服务治理中重要的组成部分，目标规则通过端口、服务版本等方式对请求流量进行划分，并对各请求分流量订制 envoy 流量策略，应用到流量的策略不仅有负载均衡，还有最小连接数、离群检测等。

字段概念介绍

几个重要字段如下：

- Host

  使用 Kubernetes Service 的短名称。含义同 VirtualService 中 destination 的 host 字段一致。服务一定要存在于对应的服务注册中心中，否则会被忽略。

- LoadBalancer

  默认情况下，Istio 使用轮询的负载均衡策略，实例池中的每个实例依次获取请求。 Istio 同时支持如下的负载均衡模型，可以在 DestinationRule 中为流向某个特定服务或服务子集的流量指定这些模型。

  - 随机：请求以随机的方式转到池中的实例。
  - 权重：请求根据指定的百分比转到实例。
  - 最少请求：请求被转到最少被访问的实例。

- Subsets

  subsets 是服务端点的集合，可以用于 A/B 测试或者分版本路由等场景。可以将一个服务的流量切分成 N 份供客户端分场景使用。 name 字段定义后主要供 VirtualService 里 destination 使用。 每个子集都是在 host 对应服务的基础上基于一个或多个 labels 定义的，在 Kubernetes 中它是

附加到像 Pod 这种对象上的键/值对。 这些标签应用于 Kubernetes 服务的 Deployment 并作为元数据信息（Metadata）来识别不同的版本。

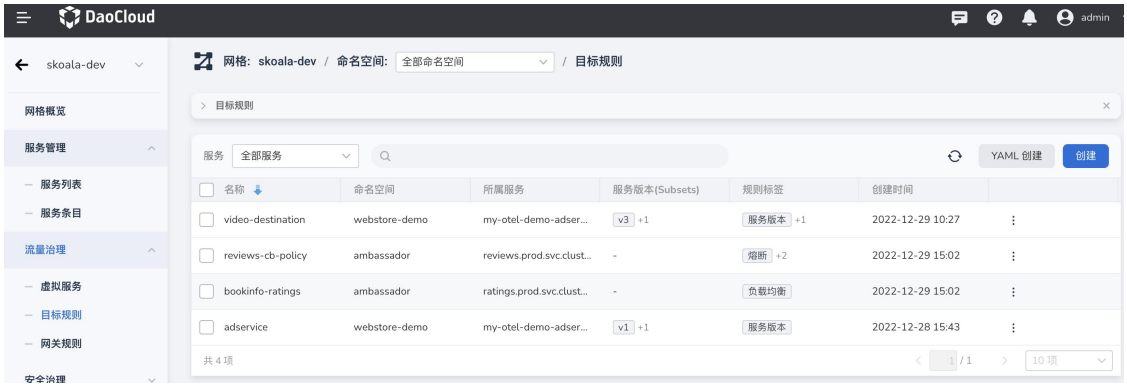
- OutlierDetection

  离群检测是减少服务异常和降低服务延迟的一种设计模式，主要是无感的处理服务异常并保证不会发生级联甚至雪崩。 如果在一定时间内服务累计发生错误的次数超过了预先定义的阈值，就会将该错误的服务从负载均衡池中移除，并持续关注服务的健康状态，当服务回复正常后，又会将服务再移回到负载均衡池。

目标规则列表介绍

目标规则列表展示了网格下的目标规则 CRD 信息，并提供了目标规则生命期管理能力。 用户可以基于规则名称、规则标签做 CRD 筛选，规则标签如下：

- 服务版本
- 负载均衡
- 地域负载均衡
- HTTP 连接池
- TCP 连接池
- 客户端 TLS
- 离群检测



目标规则列表

操作步骤

服务网格提供了两种创建方式：图形向导创建和 YAML 创建。通过图形向导创建的具体操作步骤如下：

1. 在左侧导航栏点击流量治理 -> 目标规则，点击右上角的创建按钮。



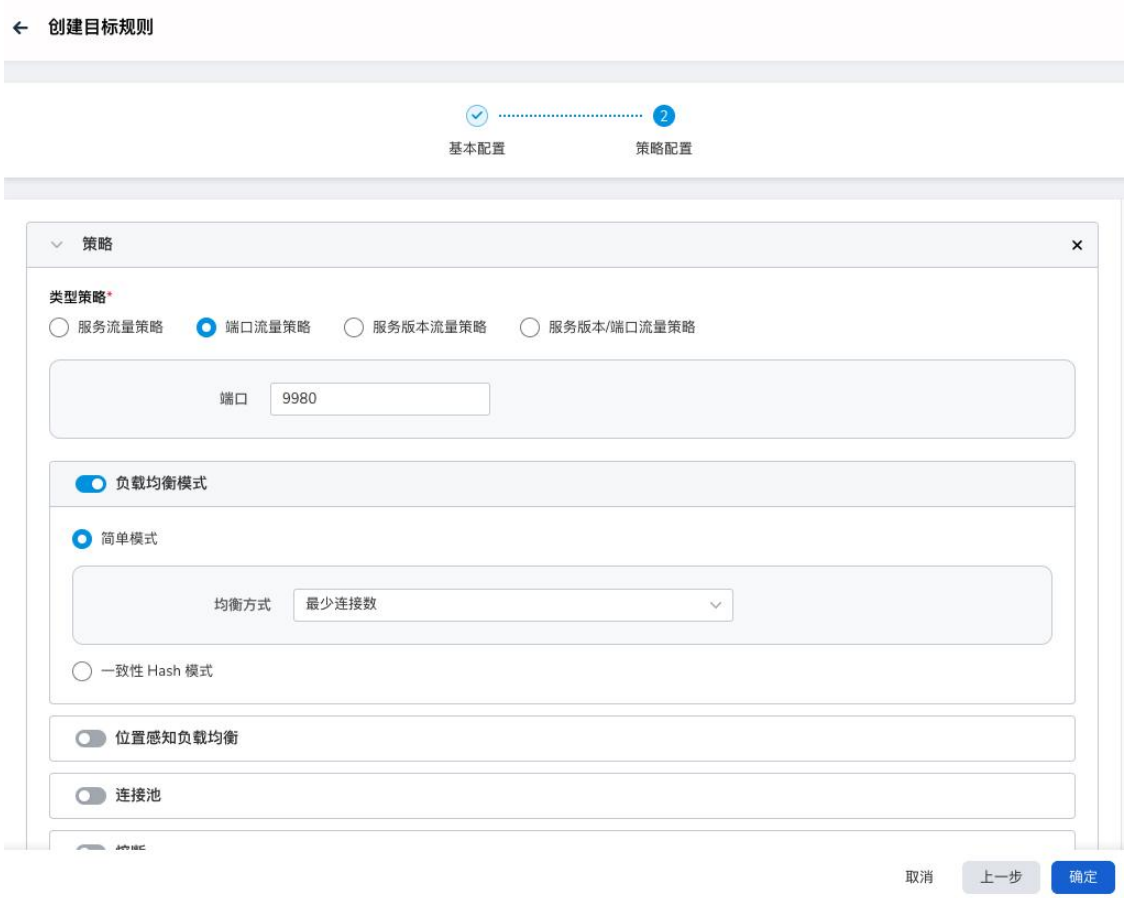
创建

2. 在创建目标界面中，先进行基本配置后点击下一步。



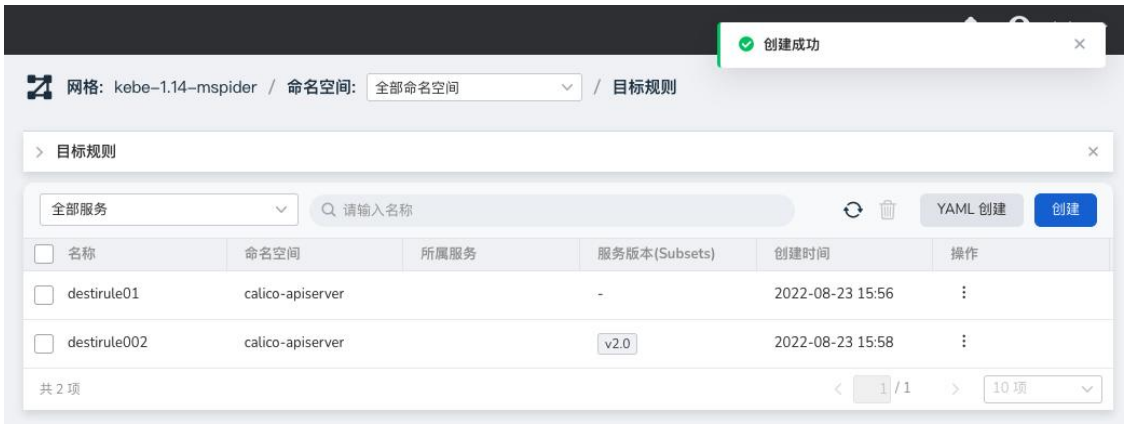
创建目标

3. 按屏幕提示选择策略类型，并配置对应的治理策略后，点击确定。



治理策略

4. 返回目标规则列表，屏幕提示创建成功。



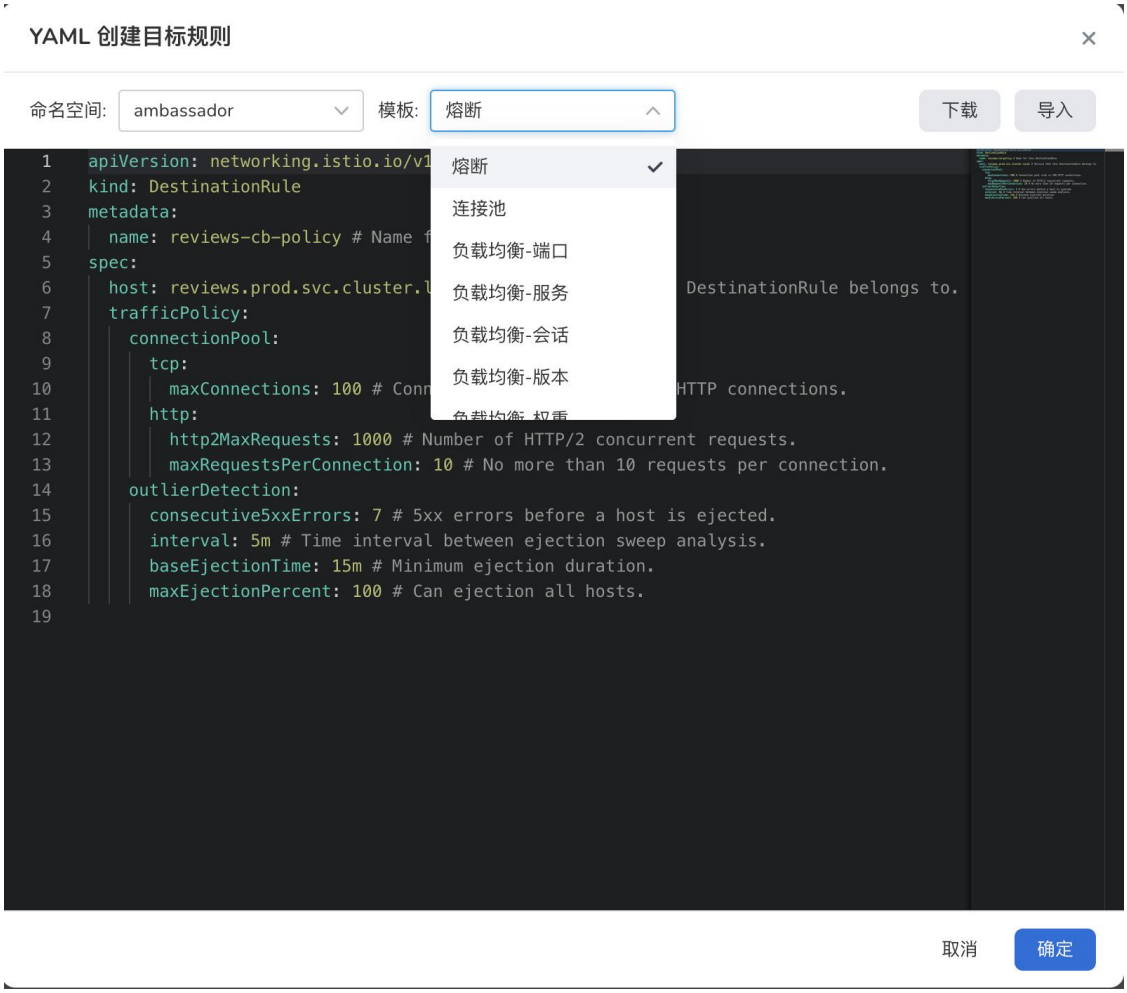
创建成功

5. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

YAML 创建方式与虚拟服务相似，您可以直接借助内置模板创建 YAML 文件，如下图所示。



YAML 创建

以下是一个 目标规则的 YAML 示例：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  annotations:
    ckube.daocloud.io/cluster: dywtest3
    ckube.daocloud.io/indexes: '{"activePolices":"","cluster":"dywtest3","createdAt":
"2023-08-10T02:18:04Z","host":"kubernetes","is_deleted":"false","labels":"","name":
"dr01","namespace":"default"}'
  creationTimestamp: "2023-08-10T02:18:04Z"
  generation: 1
```

```
  managedFields:
    - apiVersion: networking.istio.io/v1beta1
      fieldsType: FieldsV1
      fieldsV1:
        f:spec:
          .: {}
          f:host: {}
          f:trafficPolicy:
            .: {}
            f:portLevelSettings: {}
      manager: cacheproxy
      operation: Update
      time: "2023-08-10T02:18:04Z"
  name: dr01
  namespace: default
  resourceVersion: "708763"
  uid: ff95ba70-7b92-4998-b6ba-9348d355d44c
spec:
  host: kubernetes
  trafficPolicy:
    portLevelSettings:
      - port:
          number: 9980
status: {}
```

策略介绍

地域负载均衡

地域负载均衡是 Istio 具备的一个基于工作负载部署所在的 Kubernetes 集群工作节点上的地域标签，用作流量转发优化的策略。 配置方式主要有：流量分发规则（权重分布）和流量转移规则（故障转移）：

- 流量分发规则：主要配置源负载位置访问到目标负载位置在不同的区域之间的流量权重分配
- 流量转移规则：流量的故障转移一般需要配合离群检测功能使用，可以达到更及时检测工作负载故障进行流量转移

注意，地域标签是在网格成员集群的工作节点上的 label，注意检查节点的标签配置：

- 地区：topology.kubernetes.io/region
- 可用区域：topology.kubernetes.io/zone
- 分区：topology.istio.io/subzone 分区是 istio 特有的配置，以实现更细粒度划分

另外地域是根据分层顺序进行匹配排列的，不同 region 的 zone 是两个不同的可用区域。

详情请参阅 Istio 官方文档： https://istio.io/latest/zh/docs/tasks/traffic-management/locality-load-balancing/

对等身份认证

对等身份认证指的是在不对应用源码做侵入式修改的情况下，提供服务间的双向安全认证，同时密钥以及证书的创建、分发、轮转也都由系统自动完成，对用户透明，从而大大降低了安全配置管理的复杂度。

!!! note

启用对等身份认证后，相应的目标规则也需要开启 mTLS 模式，否则将无法正常访问。

一个对全网格生效的严格 mTLS 策略。生效后，网格内部服务间访问将必须启用 mTLS。

YAML 示例：

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "default"
  namespace: "istio-system"
spec:
  selector:
    matchLabels:
      app: reviews
  mtls:
    mode: STRICT
```

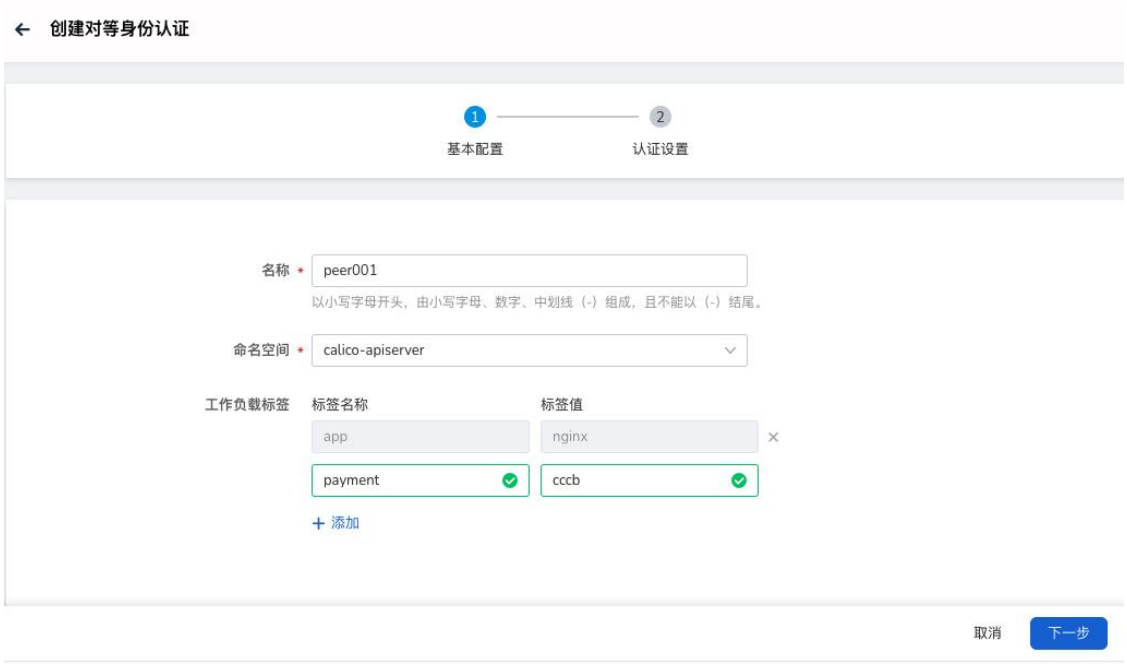
服务网格提供了两种创建方式：向导和 YAML。通过向导创建的具体操作步骤如下：

1. 在左侧导航栏点击安全治理 -> 对等身份认证，点击右上角的创建按钮。



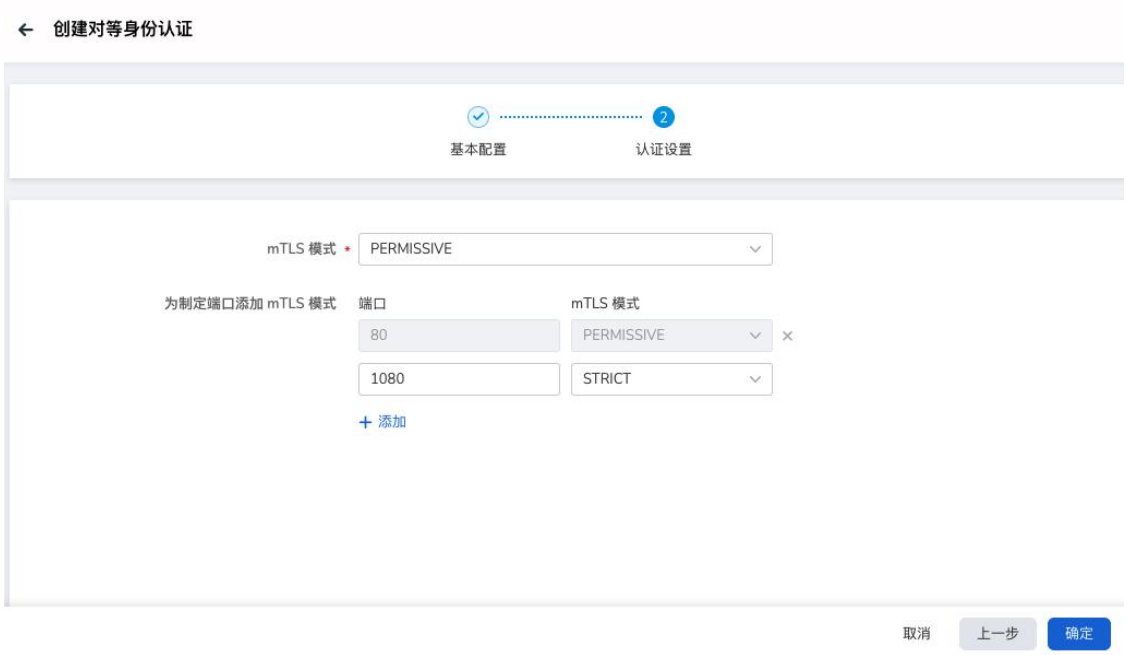
创建

2. 在创建对等身份认证界面中，先进行基本配置后点击下一步。



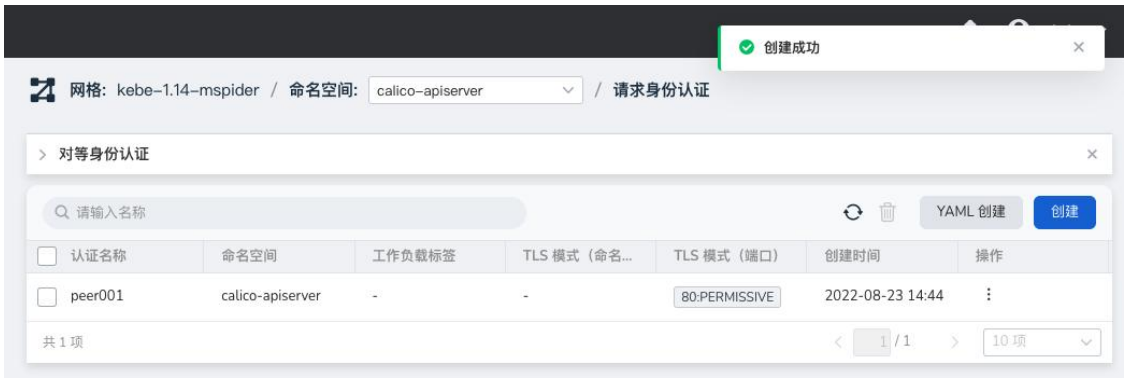
基本配置

3. 按屏幕提示进行认证设置后，点击确定。参阅 mTLS 模式参数配置。



认证设置

4. 屏幕提示创建成功。



成功

5. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

!!! note

- 具体参数的配置，请参阅[对等身份认证参数配置](./params.md#_2)
- 参阅[服务网格身份和认证](./mtls.md)。
- 更直观的操作演示，可参阅[视频教程](../../../videos/mspider.md#_5)

服务网格身份和认证

身份（Identity）是任何安全基础架构的基本概念。 在工作负载间开始通信时，通信双方必须交换包含身份信息的凭证以进行双向验证。 在客户端，根据安全命名信息检查服务器的标识，以查看它是否是工作负载授权的运行程序。 在服务器端，服务器可以根据授权策略确定客户端可以访问哪些信息， 审计谁在什么时间访问了什么，根据他们使用的工作负载向客户收费， 并拒绝任何未能支付账单的客户访问工作负载。

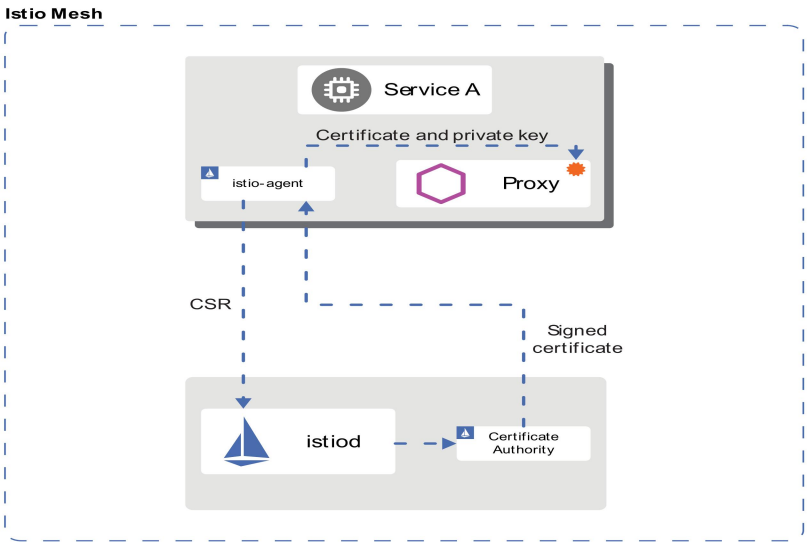
DCE 5.0 服务网格身份模型使用经典的 service identity（服务身份）来确定一个请求源端的身份。 这种模型有极好的灵活性和粒度，可以用服务身份来标识人类用户、单个工作负载或一组工作负载。 在没有服务身份的平台上，服务网格可以使用其它可以对服务实例进行分组的身份，例如服务名称。

下面的列表展示了在不同平台上可以使用的服务身份：

- Kubernetes：Kubernetes 服务帐户
- GKE/GCE：GCP 服务帐户
- 本地（非 Kubernetes）：用户帐户、自定义服务帐户、服务名称、 服务网格服务帐户或 GCP 服务帐户。自定义服务帐户引用现有服务帐户， 就像客户的身份目录管理的身份一样。

身份和证书管理

服务网格 PKI (Public Key Infrastructure，公钥基础设施）使用 X.509 证书为每个工作负载都提供强大的身份标识。 istio-agent 与每个 Envoy 代理一起运行，与 istiod 一起协作来自动化的进行大规模密钥和证书轮换。下图显示了这个机制的运行流程。



workflow

服务网格通过以下流程提供密钥和证书：

1. istiod 提供 gRPC 服务以接受证书签名请求（CSRs）。
2. istio-agent 在启动时创建私钥和 CSR，然后将 CSR 及其凭据发送到 istiod 进行签名。
3. istiod CA 验证 CSR 中携带的凭据，成功验证后签署 CSR 以生成证书。
4. 当工作负载启动时，Envoy 通过秘密发现服务（SDS）API 向同容器内的 istio-agent 发送证书和密钥请求。
5. istio-agent 通过 Envoy SDS API 将从 istiod 收到的证书和密钥发送给 Envoy。
6. istio-agent 监控工作负载证书的过期时间。上述过程会定期重复进行证书和密钥轮换。

认证

服务网格提供两种类型的认证：

- 对等身份认证：用于服务到服务的认证，以验证建立连接的客户端。 服务网格提供双向 TLS 作为传输认证的全栈解决方案，无需更改服务代码就可以启用它。这个解决方案：

    - 为每个服务提供强大的身份，表示其角色，以实现跨集群和云的互操作性。
    - 保护服务到服务的通信。
    - 提供密钥管理系统，以自动进行密钥和证书的生成、分发和轮换。
- 请求身份认证：用于终端用户认证，以验证附加到请求的凭据。 服务网格使用 JSON Web Token（JWT）验证启用请求级认证， 并使用自定义认证实现或任何 OpenID Connect 的认证实现（例如下面列举的）来简化的开发人员体验。
    - ORY Hydra
    - Keycloak
    - Auth0
    - Firebase Auth
    - Google Auth

在所有情况下，服务网格都通过自定义 Kubernetes API 将认证策略存储在 Istio config store。 Istiod 使每个代理保持最新状态， 并在适当时提供密钥。此外，服务网格的认证机制支持宽容模式（permissive mode）， 以帮助您在强制实施前了解策略更改将如何影响您的安全状况。

mTLS 认证

mTLS 全称为 Mutual Transport Layer Security，即双向传输层安全认证。mTLS 允许通信双方在 SSL/TLS 握手的初始连接期间进行相互认证。

DCE 5.0 服务网格通过客户端和服务器端 PEP(Policy Enforcement Policy）建立服务到服务的通信通道，PEP 被实现为 Envoy 代理。 当一个工作负载使用 mTLS 认证向另一个工作负载发送请求时， 该请求的处理方式如下：

1. 服务网格将出站流量从客户端重新路由到客户端的本地边车 Envoy。
2. 客户端 Envoy 与服务器端 Envoy 开始双向 TLS 握手。在握手期间， 客户端 Envoy 还做了安全命名检查， 以验证服务器证书中显示的服务帐户是否被授权运行目标服务。

3. 客户端 Envoy 和服务器端 Envoy 建立了一个双向的 TLS 连接， 服务网格将流量从客户端 Envoy 转发到服务器端 Envoy。
4. 服务器端 Envoy 授权请求。如果获得授权，它将流量转发到通过本地 TCP 连接的后端服务。

服务网格将 TLSv1_2 作为最低 TLS 版本为客户端和服务器配置了如下的加密套件：

- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- AES256-GCM-SHA384
- AES128-GCM-SHA256

宽容模式

服务网格双向 TLS 具有一个宽容模式（permissive mode）， 允许服务同时接受纯文本流量和双向 TLS 流量。这个功能极大地提升了双向 TLS 的入门体验。

在运维人员希望将服务移植到启用了双向 TLS 的 服务网格上时， 许多非 服务网格客户端与非 服务网格服务器端之间的通信会产生问题。 通常情况下，运维人员无法同时为所有客户端安装 服务网格边车， 甚至在某些客户端上没有这样做的权限。即使在服务器端上安装了 服务网格边车， 运维人员也无法在不中断现有连接的情况下启用双向 TLS。

启用宽容模式后，服务可以同时接受纯文本和双向 TLS 流量。 这个模式为入门提供了极大的灵活性。服务器中安装的服务网格边车 立即接受双向 TLS 流量而不会打断现有的纯文本流量。因此， 运维人员可以逐步安装和配置客户端 服务网格边车 发送双向 TLS 流量。 一旦客户端配置完成，运维人员便可以将服务器端配置为仅 TLS 模式。

安全命名

服务器身份（Server identity）被编码在证书里， 但服务名称（service name）通过服务发现或 DNS 被检索。 安全命名信息将服务器身份映射到服务名称。身份 A 到服务名称 B 的映射表示“授权 A 运行服务 B“。控制平面监视 apiserver， 生成安全命名映射，并将其安全地分发到 PEP。 以下示例说明了为什么安全命名对身份验证至关重要。

假设运行服务 datastore 的合法服务器仅使用 infra-team 身份。 恶意用户拥有 test-team 身份的证书和密钥。 恶意用户打算模拟合法服务以检查从客户端发送的数据。 恶意用户使用证书和 test-team 身份的密钥部署伪造服务器。 假设恶意用户成功劫持（通过 DNS 欺骗、BGP/路由劫持、ARP 欺骗等）发送到 datastore 的流量并将其重定向到伪造的服务器。

当客户端调用 datastore 服务时，它从服务器的证书中提取 test-team 身份， 并用安全命名信息检查 test-team 是否被允许运行 datastore。 客户端检测到 test-team 不允许运行 datastore 服务，认证失败。

请注意，对于非 HTTP/HTTPS 流量，安全命名不能保护其免于 DNS 欺骗， 如攻击者劫持了 DNS 并修改了目的地 IP 地址。这是因为 TCP 流量不包含主机名信息， Envoy 只能依靠目的地 IP 地址进行路由，因此 Envoy 有可能将流量路由到劫持 IP 地址所在的服务上。这种 DNS 欺骗甚至可以在客户端 Envoy 接收到流量之前发生。

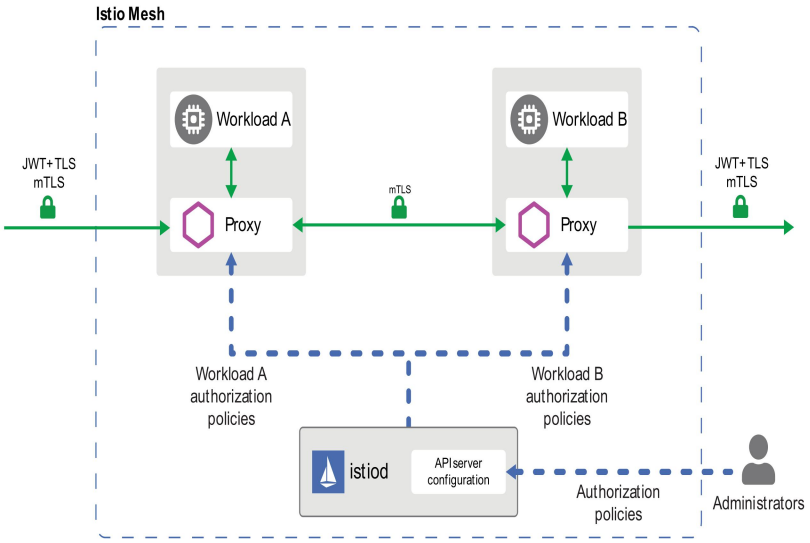
认证架构

您可以使用对等身份认证和请求身份认证策略为在服务网格中接收请求的工作负载指定认证要求。 网格运维人员使用 .yaml 文件来指定策略。部署后，策略将保存在服务网格配置存储中。 服务网格控制器监视配置存储。

在任何的策略变更时，新策略都会转换为适当的配置，告知 PEP 如何执行所需的认证机制。 控制平面可以获取公钥并将其附加到 JWT 验证的配置中。作为替代方案， Istiod 提供了 服务网格系统管理的密钥和证书的路径，并将它们安装到应用程序 Pod 用于双向 TLS。

服务网格异步发送配置到目标端点。代理收到配置后，新的认证要求会立即生效。

发送请求的客户端服务负责遵循必要的认证机制。对于请求身份认证， 应用程序负责获取 JWT 凭证并将其附加到请求。对于对等身份认证， 服务网格自动将两个 PEP 之间的所有流量升级为双向 TLS。如果认证策略禁用了双向 TLS 模式，则服务网格将继续在 PEP 之间使用纯文本。要覆盖此行为， 请使用目标规则显式禁用双向 TLS 模式。



服务网格将这两种认证类型以及凭证中的其他声明（如果适用）输出到下一层：授权。

认证策略

本节中提供了更多服务网格认证策略方面的细节。正如认证架构中所说的， 认证策略是对服务收到的请求生效的。要在双向 TLS 中指定客户端认证策略， 需要在 DetinationRule 中设置 TLSSettings。

和其他的 服务网格配置一样，可以用 .yaml 文件的形式来编写认证策略。部署策略使用 kubectl。 下面例子中的认证策略要求：与带有 app:reviews 标签的工作负载的传输层认证，必须使用双向 TLS：

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
```

```
  name: "example-peer-policy"
  namespace: "foo"
spec:
  selector:
    matchLabels:
      app: reviews
  mtls:
    mode: STRICT
```

策略存储

服务网格将网格范围的策略存储在根命名空间。这些策略使用一个空的 selector 应用到网格中的所有工作负载。具有命名空间范围的策略存储在相应的命名空间中。 它们仅适用于其命名空间内的工作负载。如果您配置了 selector 字段， 则认证策略仅适用于与您配置的条件匹配的工作负载。

Peer 和 请求身份认证策略用 kind 字段区分， 分别是 PeerAuthentication 和 RequestAuthentication。

Selector 字段

Peer 和 请求身份认证策略使用 selector 字段来指定该策略适用的工作负载的标签。 以下示例显示适用于带有 app：product-page 标签的工作负载的策略的 selector 字段：

```
selector:
  matchLabels:
    app: product-page
```

如果您没有为 selector 字段提供值， 则 服务网格会将策略与策略存储范围内的所有工作负载进行匹配。 因此，selector 字段可帮助您指定策略的范围：

- 网格范围策略：为根命名空间指定的策略，不带或带有空的 selector 字段。

- 命名空间范围的策略：为非 root 命名空间指定的策略，不带有或带有空的 selector 字段。
- 特定于工作负载的策略：在常规命名空间中定义的策略，带有非空 selector 字段。

Peer 和 请求身份认证策略对 selector 字段遵循相同的层次结构原则， 但是 服务网格以略微不同的方式组合和应用这些策略。

只能有一个网格范围的 对等身份认证策略， 每个命名空间也只能有一个命名空间范围的 对等身份认证策略。 当您为同一网格或命名空间配置多个网格范围或命名空间范围的 对等身份认证策略时， 服务网格会忽略较新的策略。当多个特定于工作负载的 对等身份认证策略匹配时， 服务网格将选择最旧的策略。

服务网格按照以下顺序为每个工作负载应用最窄的匹配策略：

1. 特定于工作负载的
2. 命名空间范围
3. 网格范围

服务网格可以将所有匹配的 请求身份认证策略组合起来， 就像它们来自单个 请求身份认证策略一样。因此， 您可以在网格或命名空间中配置多个网格范围或命名空间范围的策略。 但是，避免使用多个网格范围或命名空间范围的 请求身份认证策略仍然是一个好的实践。

对等身份认证

对等身份认证策略指定 服务网格对目标工作负载实施的双向 TLS 模式。支持以下模式：

- PERMISSIVE：工作负载接受双向 TLS 和纯文本流量。 此模式在迁移因为没有边车 而无法使用双向 TLS 的工作负载的过程中非常有用。 一旦工作负载完成边车 注入的迁移，应将模式切换为 STRICT。
- STRICT： 工作负载仅接收双向 TLS 流量。

- DISABLE：禁用双向 TLS。从安全角度来看，除非您提供自己的安全解决方案，否则请勿使用此模式。
- UNSET：继承父作用域的模式。UNSET 模式的网格范围对等身份认证策略默认使用 PERMISSIVE 模式。

下面的 对等身份认证策略要求命名空间 foo 中的所有工作负载都使用双向 TLS：

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "example-policy"
  namespace: "foo"
spec:
  mtls:
    mode: STRICT
```

对于特定于工作负载的 对等身份认证策略，可以为不同的端口指定不同的双向 TLS 模式。 您只能将端口范围的双向 TLS 配置在工作负载声明过的端口上。以下示例为 app:example-app 工作负载禁用了端口 80 上的双向 TLS， 并对所有其他端口使用命名空间范围的 对等身份认证策略的双向 TLS 设置：

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "example-workload-policy"
  namespace: "foo"
spec:
  selector:
    matchLabels:
      app: example-app
  portLevelMtls:
    80:
      mode: DISABLE
```

上面的 对等身份认证策略仅在有如下 Service 定义时工作， 将流向 example-service 服务的请求绑定到 example-app 工作负载的 80 端口

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: foo
spec:
  ports:
  - name: http
    port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: example-app
```

请求身份认证

请求身份认证策略指定验证 JSON Web Token（JWT）所需的值。这些值包括：

- token 在请求中的位置
- 请求的 issuer
- 公共 JSON Web Key Set（JWKS）

服务网格会根据 请求身份认证策略中的规则检查提供的令牌（如果已提供）， 并拒绝令牌无效的请求。当请求不带有令牌时，默认将接受这些请求。 要拒绝没有令牌的请求，请提供授权规则，该规则指定对特定操作（例如，路径或操作）的限制。

如果 请求身份认证策略使用唯一的位置，则可以在这些策略中指定多个 JWT。 当多个策略与一个工作负载匹配时，服务网格会将所有规则组合起来， 就好像这些规则被指定为单个策略一样。此行为对于开发接受来自不同 JWT 提供者的工

作负载时很有用。 但是，不支持具有多个有效 JWT 的请求，因为此类请求的输出主体未被定义。

Principal

使用 对等身份认证策略和双向 TLS 时，服务网格将身份从 对等身份认证提取到 source.principal 中。 同样，当您使用 请求身份认证策略时，服务网格会将 JWT 中的身份赋值给 request.auth.principal。 使用这些 principal 设置授权策略并作为遥测的输出。

更新认证策略

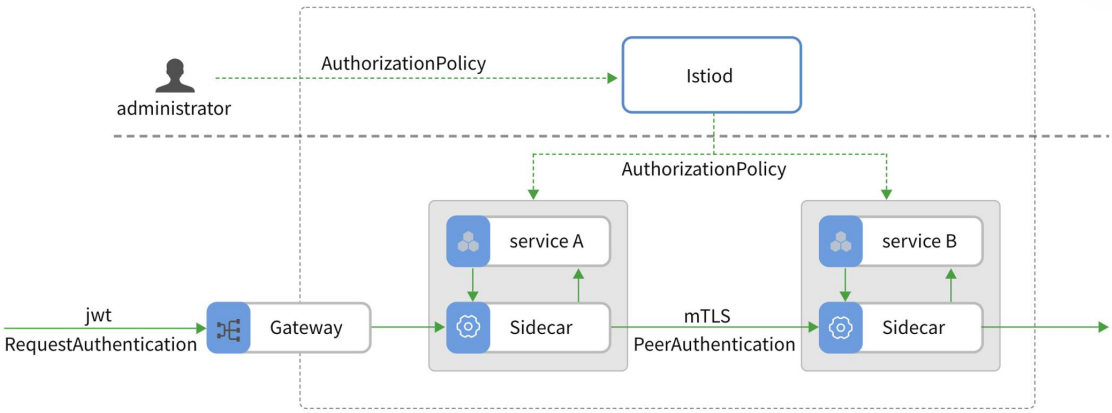
您可以随时更改认证策略，服务网格几乎实时将新策略推送到工作负载。 但是，服务网格无法保证所有工作负载都同时收到新政策。 以下建议有助于避免在更新认证策略时造成干扰：

- 将对等身份认证策略的模式从 DISABLE 更改为 STRICT 时， 请使用 PERMISSIVE 模式来过渡，反之亦然。当所有工作负载成功切换到所需模式时， 您可以将策略应用于最终模式。您可以使用 服务网格遥测技术来验证工作负载已成功切换。
- 将请求身份认证策略从一个 JWT 迁移到另一个 JWT 时， 将新 JWT 的规则添加到该策略中，而不删除旧规则。这样， 工作负载将接受两种类型的 JWT，当所有流量都切换到新的 JWT 时， 您可以删除旧规则。但是，每个 JWT 必须使用不同的位置。

下一步：设置对等身份认证和请求身份认证

安全治理

服务网格提供了一种授权机制和两种认证方式（请求身份认证和对等身份认证），用户可以在服务网格中通过向导和 YAML 编写两种方式创建、编辑资源文件，并可以针对网格全局、命名空间、工作负载三个层面创建规则。当资源创建成功后，Istiod 将转换为配置分发至边车代理执行。



安全治理

授权机制

服务网格提供的授权机制允许您控制对服务的访问。 使用授权机制，您可以定义规则，指定允许哪些服务相互通信。 这些规则可以基于各种因素，例如流量的源和目的地、使用的协议以及发出请求的客户端的身份。 通过使用 服务网格的授权机制，您可以确保只有经过授权的流量允许在您的服务网格中流动。

以下是如何使用 服务网格的授权机制限制两个服务之间流量的示例：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: my-auth-policy
spec:
  selector:
```

```
    matchLabels:
      app: my-service
  action: ALLOW
  rules:
  - from:
    - source:
        notNamespaces: ["my-namespace"]
    to:
    - operation:
        methods: ["GET"]
```

在此示例中，我们创建了一个 AuthorizationPolicy 资源，允许流量从除 my-namespace 以外的任何命名空间流向 my-service 服务，但仅限于 GET 请求。 另请参见图形界面的创建方式。

请求身份验证

请求身份验证用于验证向服务发出请求的客户端的身份。

以下是如何使用请求级身份验证验证客户端身份的示例：

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: my-authn-policy
spec:
  selector:
    matchLabels:
      app: my-service
  jwtRules:
  - issuer: "https://my-auth-server.com"
    jwksUri: "https://my-auth-server.com/.well-known/jwks.json"
```

在此示例中，我们创建了一个 RequestAuthentication 资源，用于验证向 my-service 服务发出请求的客户端的身份。 我们使用 JSON Web Token（JWT）

来验证客户端，指定了发行者和用于验证令牌的公钥的位置。 另请参见图形界面的创建方式。

对等身份验证

对等身份验证用于验证服务本身的身份。服务网格提供了几种对等身份验证选项，包括双向 TLS 身份验证和基于 JSON Web Token 的身份验证。

以下是如何使用双向 TLS 身份验证验证服务身份的示例：

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: my-peer-authn-policy
spec:
  selector:
    matchLabels:
      app: my-service
  mtls:
    mode: STRICT
```

在此示例中，我们创建了一个 PeerAuthentication 资源，要求 my-service 服务进行双向 TLS 身份验证。 我们使用 STRICT 模式，该模式要求客户端和服务器都提供有效的证书。 另请参见图形界面的创建方式。

更多说明，请参阅服务网格身份和认证。

安全治理参数配置

本页介绍有关对等身份认证、请求身份认证、授权策略相关的参数配置。

对等身份认证

采用图形向导模式时，对等身份认证分为基本配置和认证设置两步，各参数说明如下。

基本配置

UI 项	YAML 字段	描述
名称	metadata.name	必填。对等身份认证名称，同一个命名空间下不可重名。
命名空间	metadata.namespace	必选。对等身份认证所属的命名空间。当选择网格的根命名空间时，将创建全局策略。全局策略仅能创建一个，需在界面中做检查，避免用户重复创建。
工作负载标签	spec.selector	可选。应用对等身份认证策略的工作负载选择标签，可添加多个标签，无需排序。
标签名称	spec.selector.matchLabels	必填。由小写字母、数字、连字符（-）、下划线（_）及小数点（.）组成
标签值	spec.selector.matchLabels.{标签名称}	必填。由小写字母、数字、连字符（-）、下划线（_）及小数点（.）组成

认证设置 - mTLS 模式

UI 项	YAML 字段	描述
mTLS 模式	spec.mTLS.mode	必填。用于设定命名空间的 mTLS 模式：– UNSET：继承父级选项。否则视

UI 项	YAML 字段	描述
		为 PERMISSIVE– PERMISSIVE：明文和 mTLS 连接– STRICT：仅 mTLS 连接– DISABLE：仅明文连接
为指定端口添加 mTLS 模式	spec.portLevelMtls	可选。针对指定端口设置 mTLS 规则，可添加多条规则，无需排序。– UNSET：继承父级选项。否则视为 PERMISSIVE– PERMISSIVE：明文和 mTLS 连接– STRICT：仅 mTLS 连接– DISABLE：仅明文连接

请求身份认证

采用图形向导模式时，请求身份认证分为基本配置和认证设置两步，各参数说明如下。

基本配置

UI 项	YAML 字段	描述
名称	metadata.name	必填。请求身份认证名称，同一个命名空间下不可重名。
命名空间	metadata.namespace	必选。请求身份认证所属的命名空间。当选择网格的根命名空间时，将创建全局策略。全局策略仅能创建一个，需在界面中做检查，避免用户重复创建。同一个命名空间内，请求身份认证的名称不能重复。
工作负载标签	spec.selector	可选。应用请求身份认证策略的工作负载选择标签，可添加多条选择标签，无需排序。
标签名称	spec.selector.matchLabels	必填。由小写字母、数字、连字符（–）、下划线（_）及小数点（.）组成
标签值	spec.selector.matchLab	必填。由小写字母、数字、连字符（–）、

UI 项	YAML 字段	描述
	els.{标签名称}	下划线（_）及小数点（.）组成

认证设置

UI 项	YAML 字段	描述
添加 JWT 规则	spec.jwtRules	可选。用于用户请求认证的 JWT 规则，可添加多条规则。
Issuer	spec.jwtRules.issuers	必填。JSON Web Token (JWT) 签发人信息。
Audiences	spec.jwtRules.issuers.Audiences	可选。配置可访问的 audiences 列表，如果为空，将访问 service name。
jwksUri	spec.jwtRules.issuers.jwksUri	可选。JSON Web Key (JWK) 的 JSON 文件路径，与 jwks 互斥，二选一。例如 https://www.googleapis.com/oauth2/v1/certs
jwks	spec.jwtRules.issuers.jwks	可选。JSON Web Key Set (JWKS) 文件内容，与 jwksUri 互斥，二选一。

更多请参阅 OpenID Provider Metadata。

授权策略

采用图形向导模式时，授权策略的创建分为基本配置和策略设置两步，各参数说明如下。

基本配置

可配置项	YAML 字段	描述
名称	metadata.name	必填。授权的策略名称。
命名空间	metadata.namespace	必选。授权策略所属命名空间，当选择网格根命名空间时，将创建全局策略，

可配置项	YAML 字段	描述
		全局策略仅能创建一个，需在界面做检查，避免用户重复创建。同一个命名空间内，请求身份认证不可重名。
工作负载标签	spec.selector	可选。应用授权策略的工作负载选择标签，可添加多条选择标签，无需排序。
标签名称	spec.selector.matchLabels	可选。由小写字母、数字、连字符（-）、下划线（_）及小数点（.）组成。
标签值	spec.selector.matchLabels.{标签名称}	可选。由小写字母、数字、连字符（-）、下划线（_）及小数点（.）组成。

策略设置

可配置项	YAML 字段	描述
策略动作	spec.action	可选。包含：- 允许（allow）- 拒绝（deny）- 审计（audit）- 自定义（custom）选择自定义时，增加 provider 输入项。
Provider	spec.provider.name	必填。仅在策略动作选择为自定义时，才显示该输入框。
请求策略	spec.rules	可选。包含请求来源、请求操作、策略条件三部分，可添加多条，按顺序执行。
添加请求来源	spec.rules.-from	可选。请求来源可基于命名空间、IP 段等进行定义，可添加多条。各项参数参见下文 请求来源 Source。
添加请求操作	spec.rules.-to	可选。请求操作是对筛选出的请求执行的操作，例如发送至指定端口或主机，可添加多个操作。各项参数参见下文请

可配置项	YAML 字段	描述
		求操作 Operation。
添加策略条件	spec.rules.–when	必填。策略条件是一个可选设置，可以增加类似黑名单（values）、白名单的限制条件（notValues），可添加多个策略条件。各项参数参见下文策略条件 Condition。

请求来源 Source

您可以增加请求来源（Source）。Source 指定一个请求的来源身份，对请求来源中的字段执行逻辑与运算。

例如，如果 Source 是：

- 主体为“admin”或“dev”
- 命名空间为“prod”或“test”
- 且 ip 不是“1.2.3.4”.

匹配的 YAML 内容为：

```
principals: ["admin", "dev"]
namespaces: ["prod", "test"]
notIpBlocks: ["1.2.3.4"]
```

具体字段说明如下：

Key 字段	类型	描述
principals	string[]	可选。从对等证书衍生的对等身份列表。对等身份的格式为 "<TRUST_DOMAIN>/ns/<NAMESPACE>/sa/<SERVICE_ACCOUNT>"，例如 "cluster.local/ns/default/sa/productpage"。此字段要求启用 mTLS，且等同于 source.principal 属性。如

Key 字段	类型	描述
		果不设置，则允许所有主体。
notPrincipals	string[]	可选。对等身份的反向匹配列表。
requestPrincipals	string[]	可选。从 JWT 派生的请求身份列表。请求身份的格式为 "<ISS>/<SUB>"，例如 "example.com/sub-1"。此字段要求启用请求身份验证，且等同于 request.auth.principal 属性。如果不设置，则允许所有请求主体。
notRequestPrincipals	string[]	可选。请求身份的反向匹配列表。
namespaces	string[]	可选。从对等证书衍生的命名空间。此字段要求启用 mTLS，且等同于 source.namespace 属性。如果不设置，则允许所有命名空间。
notNamespaces	string[]	可选。命名空间的反向匹配列表。
ipBlocks	string[]	可选。根据 IP 数据包的来源地址进行填充的 IP 段列表。支持单个 IP（例如“1.2.3.4”）和 CIDR（例如“1.2.3.0/24”）。这等同于 source.ip 属性。如果不设置，则允许所有 IP。
notIpBlocks	string[]	可选。IP 段的反向匹配列表。
remoteIpBlocks	string[]	可选。根据 X-Forwarded-For 标头或代理协议进行填充的 IP 段列表。要使用此字段，您必须在安装 Istio 或在 ingress 网关使用注解时在 meshConfig 下配置 gatewayTopology 的 numTrustedProxies 字段。 支持单个 IP（例如“1.2.3.4”）和 CIDR（例如“1.2.3.0/24”）。这等同于 remote.ip 属性。如果不设置，则允许所有 IP。
notRemoteIpBlocks	string[]	可选。远程 IP 段的反向匹配列表。

请求操作 Operation

您可以增加请求操作（Operation）。Operation 指定请求的操作，对操作中的字段执行逻辑与操作。

例如，以下操作将匹配：

- 主机后缀为“.example.com”
- 方法为“GET”或“HEAD”
- 补丁没有前缀“/admin”

```
hosts: ["*.example.com"]
methods: ["GET", "HEAD"]
notPaths: ["/admin*"]
```

Key 字段	类型	描述
hosts	string[]	可选。在 HTTP 请求中指定的主机列表。不区分大小写。如果不设置，则允许所有主机。仅适用于 HTTP。
notHosts	string[]	可选。在 HTTP 请求中指定的主机反向匹配列表。不区分大小写。
ports	string[]	可选。连接中指定的端口列表。如果不设置，则允许所有端口。
notPorts	string[]	可选。连接中所指定端口的反向匹配列表。
methods	string[]	可选。HTTP 请求中指定的方法列表。对于 gRPC 服务，这将始终是“POST”。如果不设置，则允许所有方法。仅适用于 HTTP。
notMethods	string[]	可选。HTTP 请求中所指定方法的反向匹配列表。
paths	string[]	可选。HTTP 请求中指定的路径列表。对于 gRPC 服务，这将是“/package.service/method”格式的完全限定名称。如果不设置，则允许所有路径。仅适用于 HTTP。

Key 字段	类型	描述
notPaths	string[]	可选。路径的反向匹配列表。

策略条件 Condition

您还可以增加策略条件(Condition)。Condition 指定其他必需的属性。

Key 字段	描述	支持的协议	Value 示例
request.headers	HTTP 请求头，需要用 [] 括起来	HTTP only	["Mozilla/*"]
source.ip	源 IP 地址，支持单个 IP 或 CIDR	HTTP and TCP	["10.1.2.3"]
remote.ip	由 X-Forwarded-For 请求头或代理协议确定的原始客户端 IP 地址，支持单个 IP 或 CIDR	HTTP and TCP	["10.1.2.3", "10.2.0.0/16"]
source.namespace	源负载实例命名空间，需启用双向 TLS	HTTP and TCP	["default"]
source.principal	源负载的标识，需启用双向 TLS	HTTP and TCP	["cluster.local/ns/default/sa/productpage"]
request.auth.principal	已认证过 principal 的请求	HTTP only	["accounts.my-svc.com/104958560606"]
request.	此身份验证信息的目	HTTP only	["my-

Key 字段	描述	支持的协议	Value 示例
auth.audiences	标主体		svc.com"]
request.auth.presenter	证书的颁发者	HTTP only	["123456789012.my–svc.com"]
request.auth.claims	Claims 来源于 JWT。需要用 [] 括起来	HTTP only	["*@foo.com"]
destination.ip	目标 IP 地址，支持单个 IP 或 CIDR	HTTP and TCP	["10.1.2.3", "10.2.0.0/16"]
destination.port	目标 IP 地址上的端口，必须在 [0, 65535] 范围内	HTTP and TCP	["80", "443"]
connection.sni	服务器名称指示，需启用双向 TLS	HTTP and TCP	["www.example.com"]
experimental.envoy.filters.*	用于过滤器的实验性元数据匹配，包装的值 [] 作为列表匹配	HTTP and TCP	["[update]"]

!!! note

无法保证 `experimental.*` 密钥向后的兼容性，可以随时将它们删除，但须谨慎操作。

请求身份认证

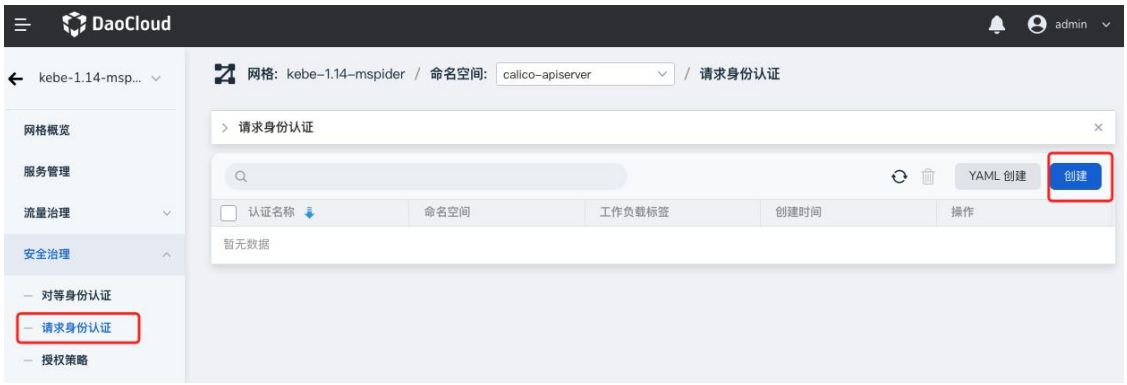
当一个外部用户对网格内部服务发起请求时，可以采用这种认证模式。在这种模式下，使用 JSON Web Token (JWT) 实现请求加密。 每个请求身份认证需要配置一个授权策略。

所有标签为 app: httpbin 的工作负载需要使用 JWT 认证。示例如下：

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: httpbin
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "issuer-foo"
    jwksUri: https://example.com/.well-known/jwks.json
```

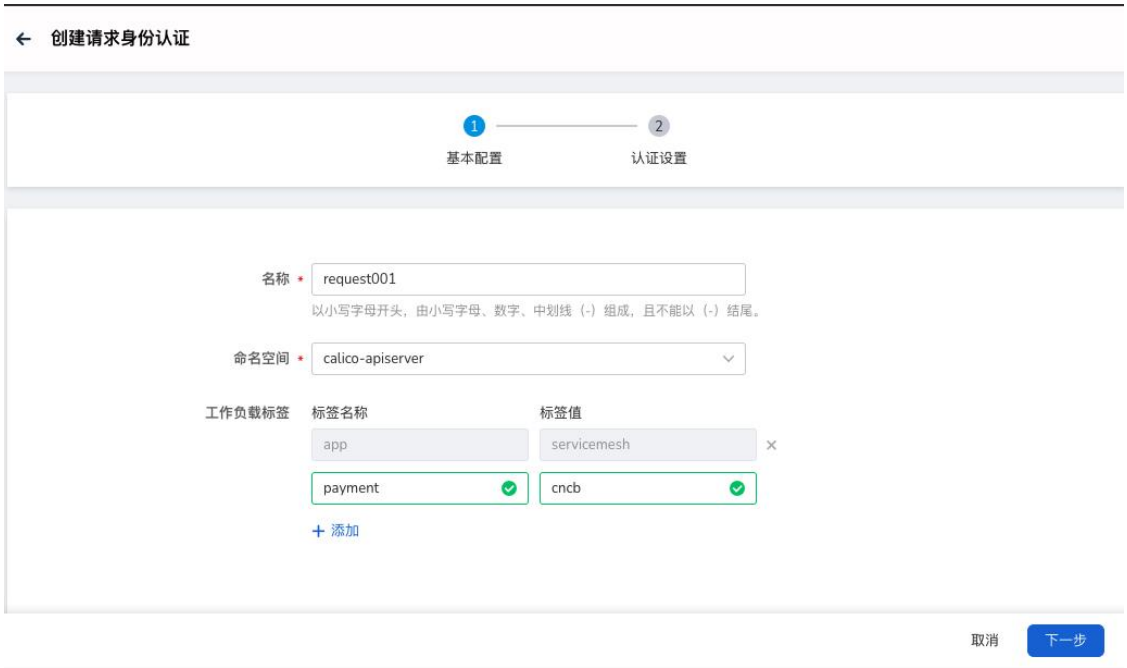
服务网格提供了两种创建方式：向导向导和 YAML。通过向导创建的具体操作步骤如下：

1. 在左侧导航栏点击安全治理 -> 请求身份认证，点击右上角的创建按钮。



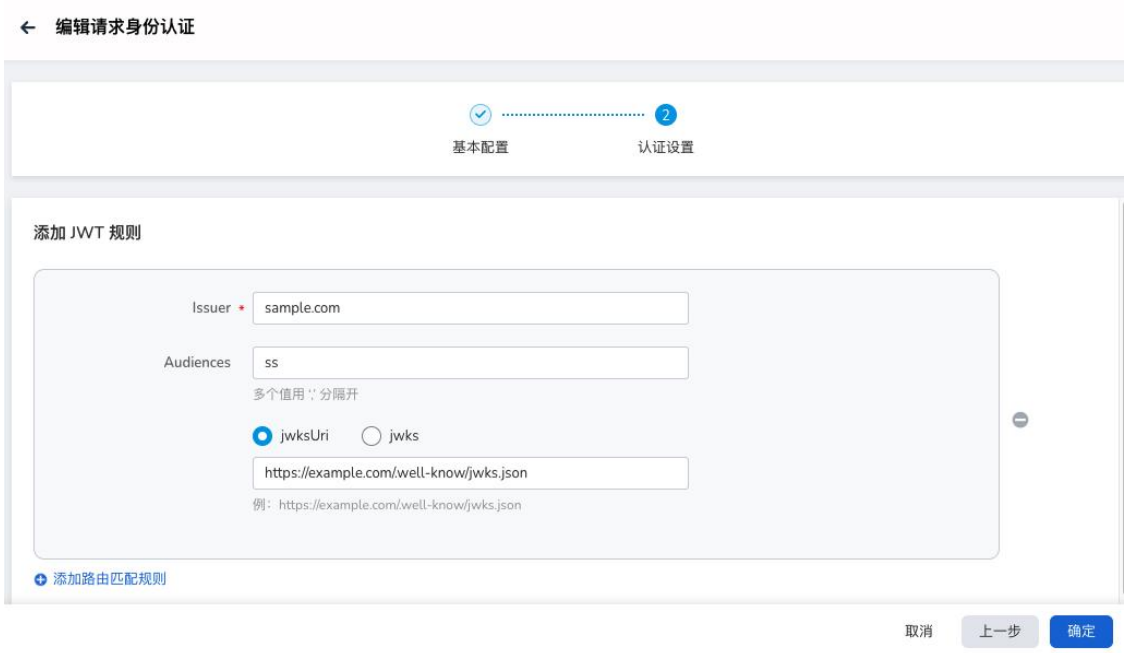
创建

2. 在创建请求身份认证界面中，先进行基本配置后点击下一步。



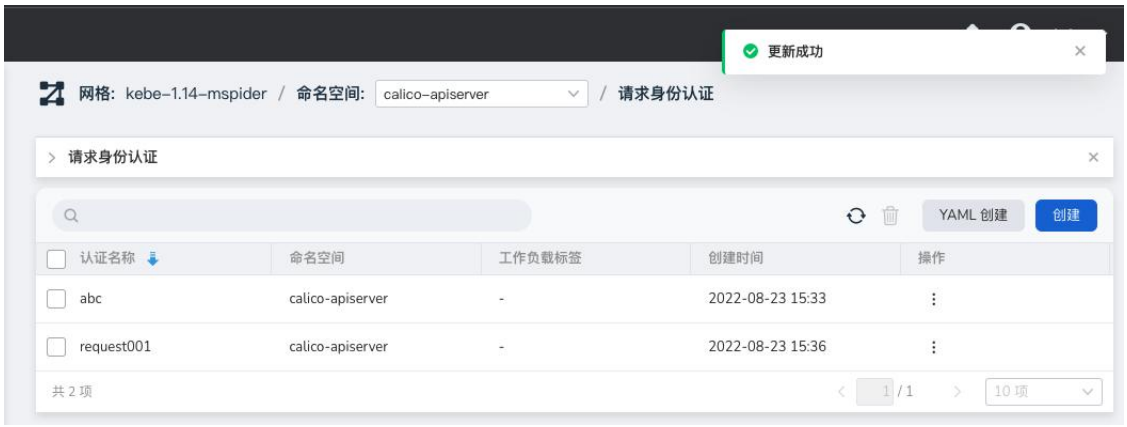
基本配置

3. 按屏幕提示进行认证设置后，点击确定，系统将验证所配置信息。参阅认证设置的参数配置。



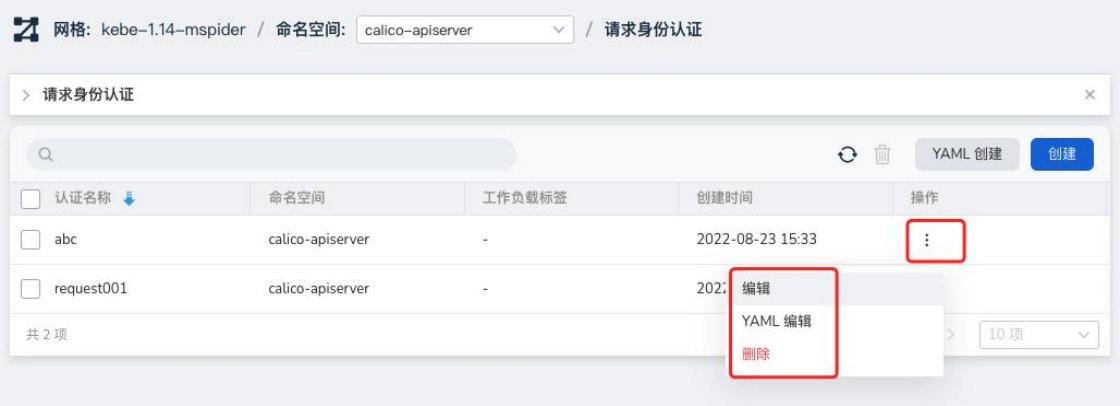
认证设置

4. 验证通过后，屏幕提示创建成功。



成功

5. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

!!! note

- 具体参数的配置，请参阅[请求身份认证参数配置](./params.md#_5)。
- 参阅[服务网格身份和认证](./mtls.md)。
- 更直观的操作演示，可参阅[视频教程](../../../videos/mspider.md#_5)。

授权策略

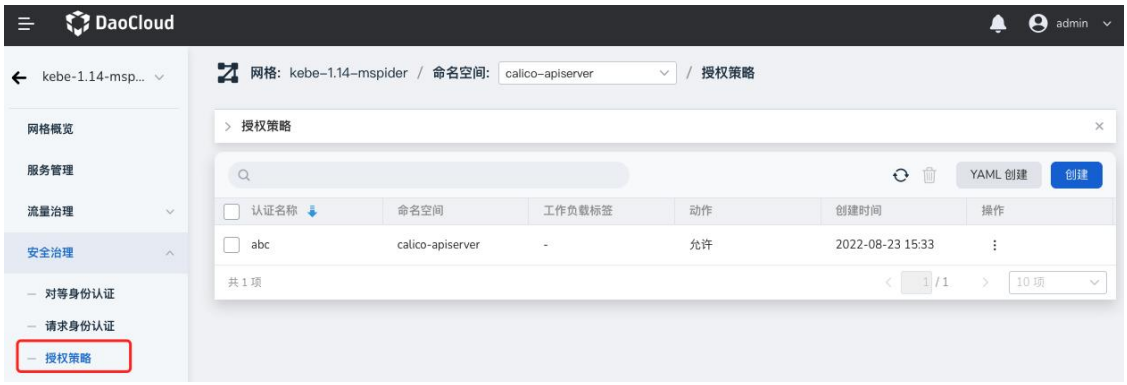
授权策略类似于一种四层到七层的“防火墙”，它会像传统防火墙一样，对数据流进行分析和匹配，然后执行相应的动作。 无论是来自内部还是外部的请求，都适用授权策略。

授权策略的参考 YAML 示例如下：

```yaml
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: "ratings-viewer"
  namespace: default
spec:
  selector:
    matchLabels:
      app: ratings
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/default/sa/bookinfo-reviews"]
    to:
    - operation:
        methods: ["GET"]
```

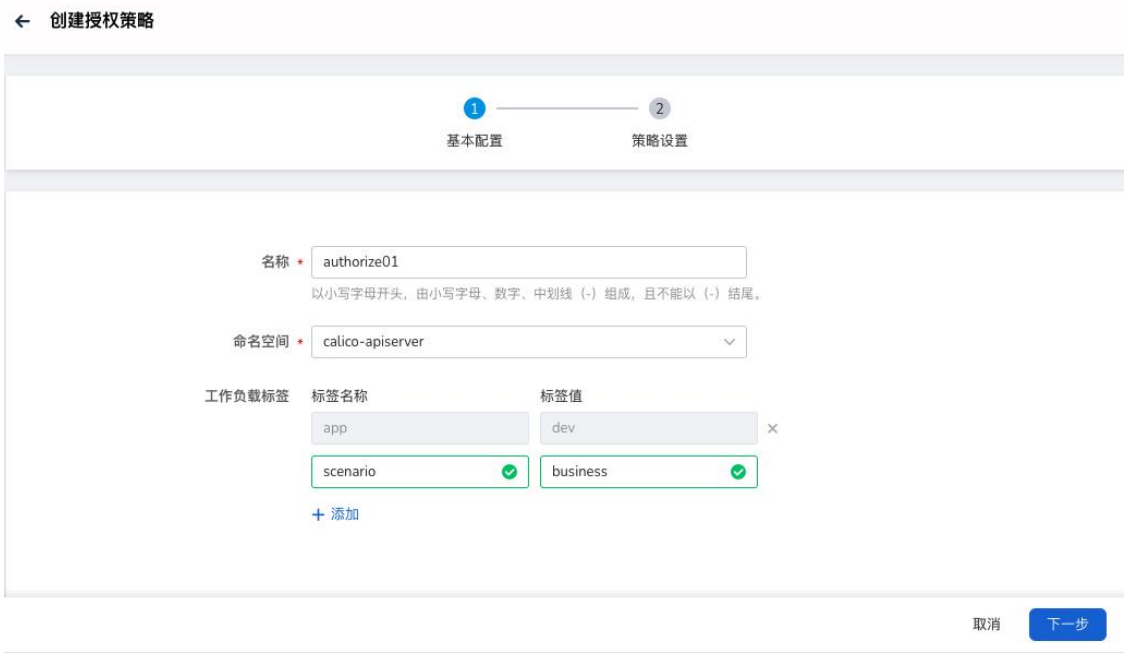
服务网格提供了两种创建方式：向导和 YAML。通过向导创建的具体操作步骤如下：

1. 在左侧导航栏点击安全治理 -> 授权策略，点击右上角的创建按钮。



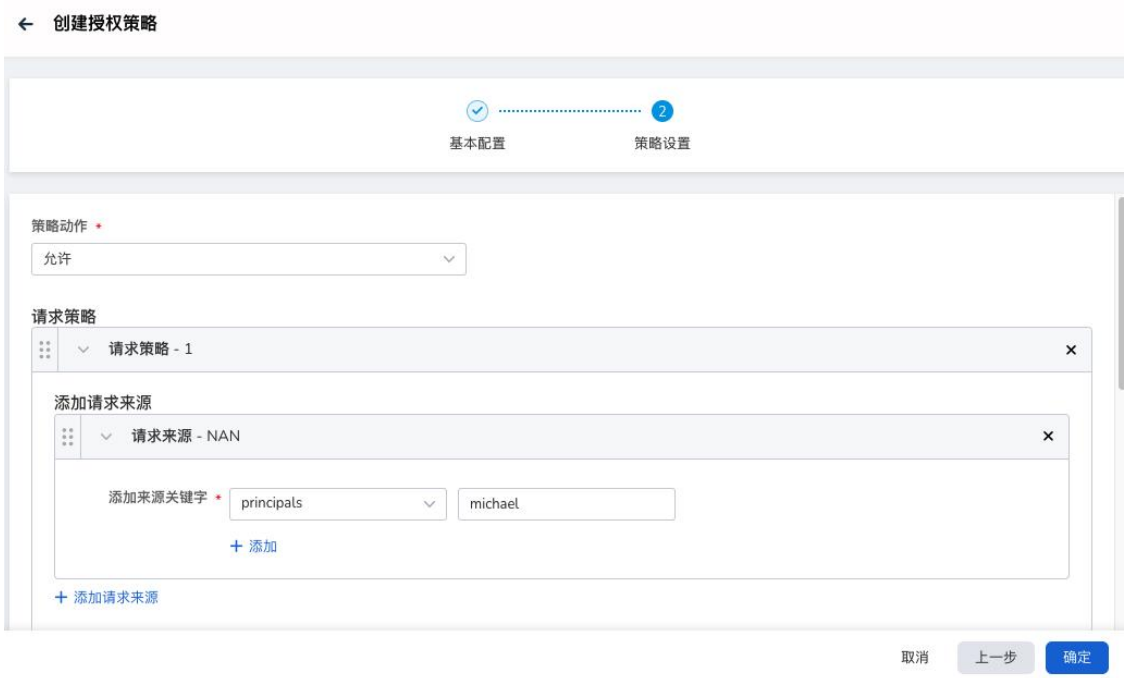
创建

2. 在创建授权策略界面中，先进行基本配置后点击下一步。



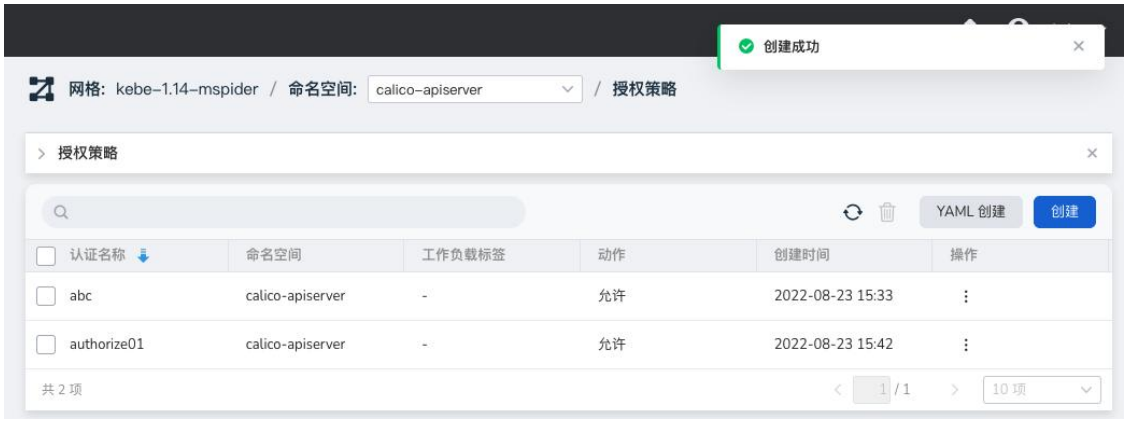
基本配置

3. 按屏幕提示进行策略设置后，点击确定。参阅策略设置参数说明。



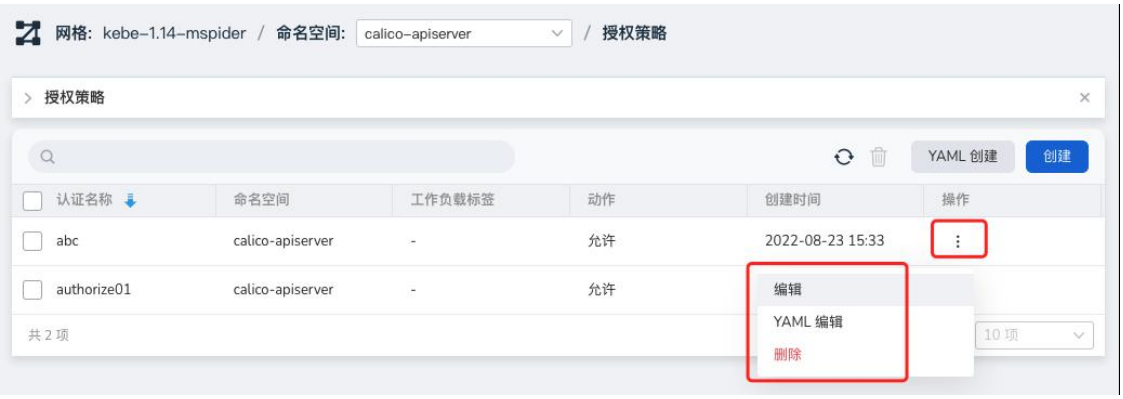
策略设置

4. 返回授权列表，屏幕提示创建成功。



创建成功

5. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

!!! note

- 具体参数的配置，请参阅[授权策略参数配置](./params.md#_8)。
- 更直观的操作演示，可参阅[视频教程](../../../videos/mspider.md)。

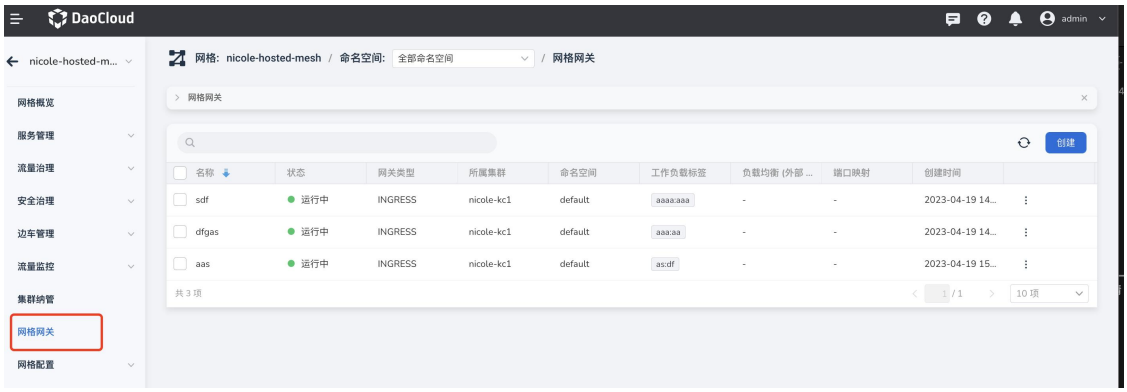
管理网格网关

网格网关功能提供了网格网关实例的创建、管理功能，用户可以在该页面管理当前网格内的网格网关实例。

网格网关分为 Ingress（入站）和 Egress（出站）两类。 Ingress 网关用于定义服务网格内应用的流量入口，所有进入服务网格内应用的流量应对经过 Ingress 网关。 Egress 网关用于定义了网格内应用的流量出口，可以让外部服务的流量统一经过 Egress 网关，从而实现更精确的流量控制。 网关实例运行的也是 Envoy，但与边车不同，网格以具体的实例单独运行。

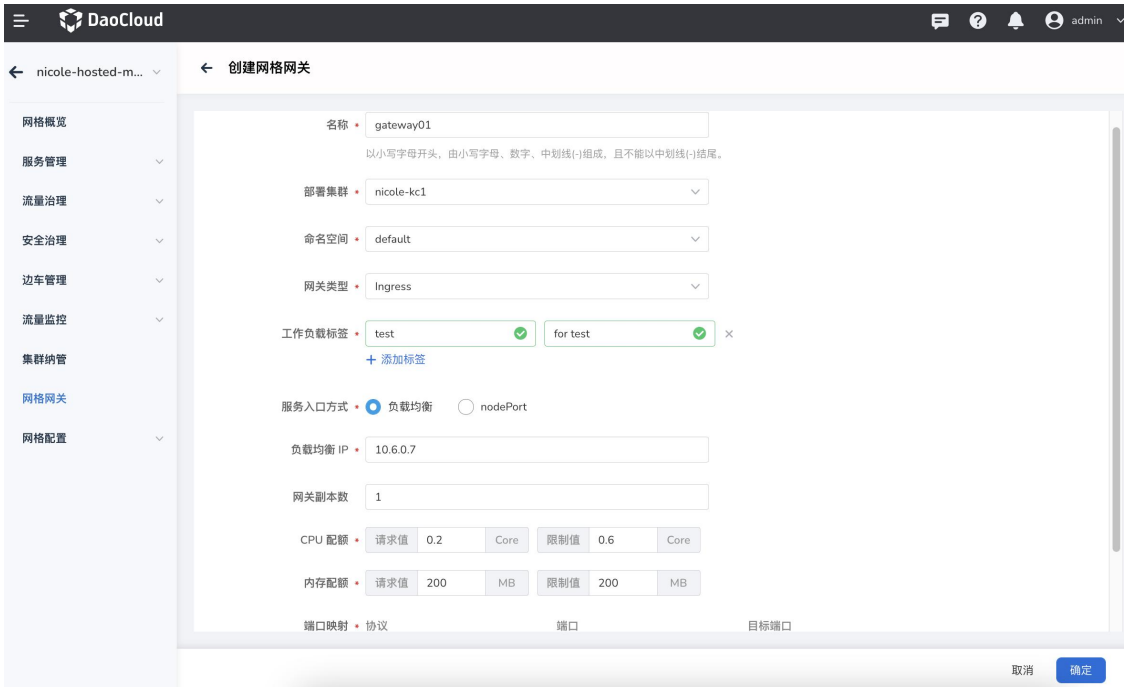
创建网格网关的操作步骤如下。

1. 在左侧导航栏中点击网格网关，进入网关列表，点击右上角的创建按钮。



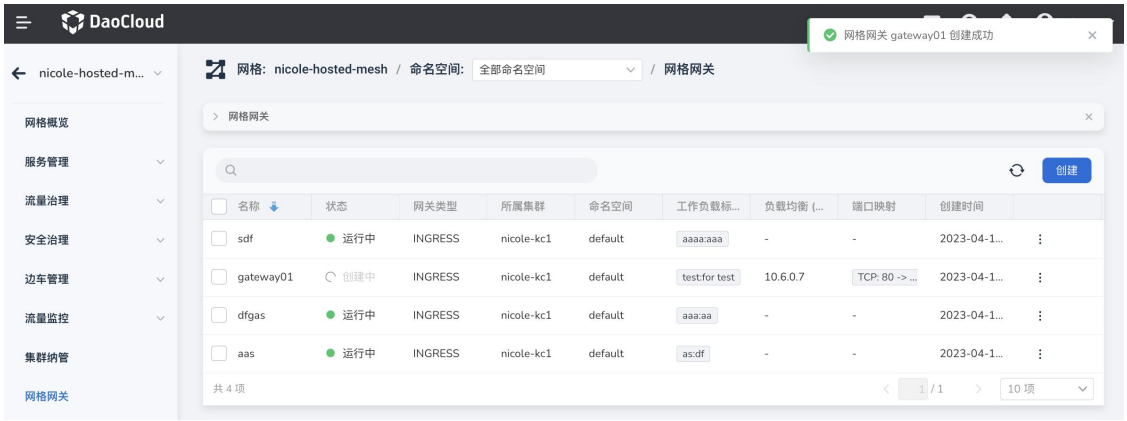
创建

2. 在创建网格网关窗口中，按提示配置各项参数后点击确定。



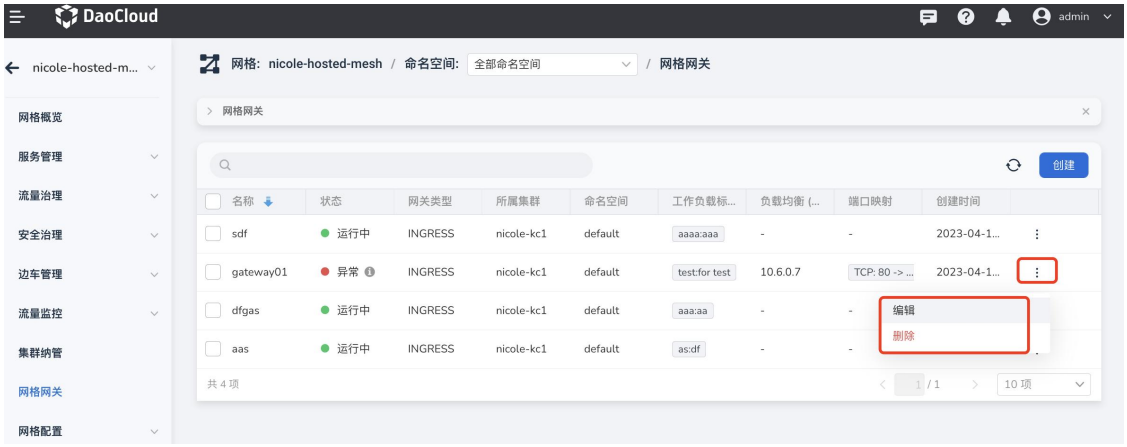
创建

3. 返回网格网关列表，屏幕右上角提示创建成功。刚创建的网格网关状态为创建中。



创建

4. 几秒后刷新页面，状态将变为运行中，表示网关成功配置。点击列表右侧的 ...，可以执行编辑和删除操作。



创建

!!! info

如有任何异常，请查看异常提示尝试解决问题。

更多参阅[视频教程](../../../videos/mspider.md)。

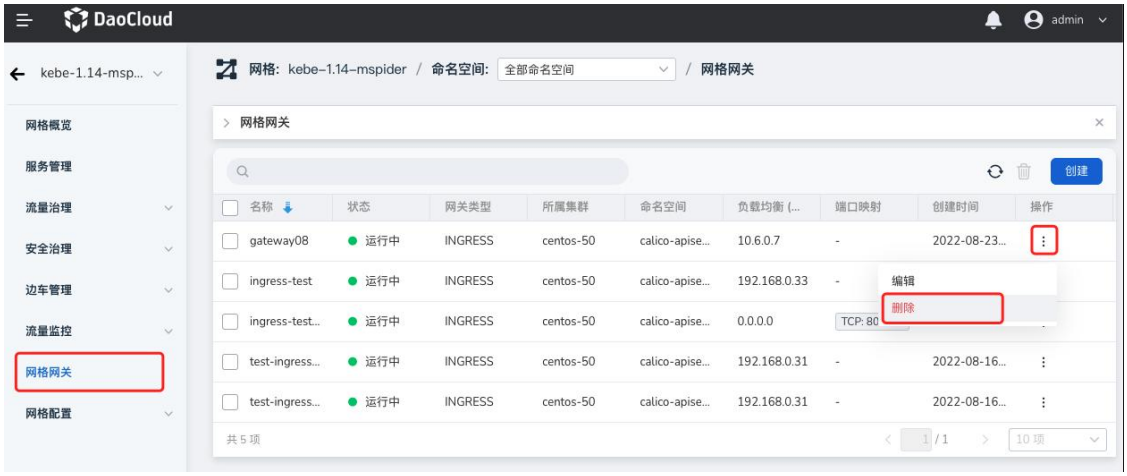
删除网格网关

如果想要从网关列表中删除一个集群时，可以按照以下步骤删除集群。

删除一个网关

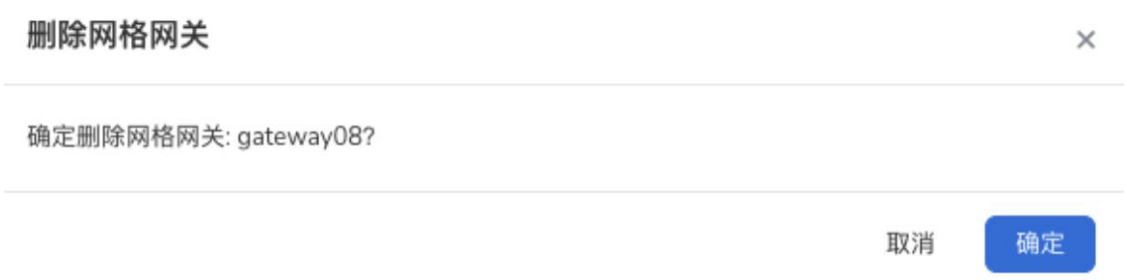
这是推荐操作，可以防止误删除。

1. 在左侧导航栏点击网格网关，在网格列表右侧点击 ... 按钮，在弹出菜单中选择删除。



删除网关

2. 在弹出窗口中，确认信息无误后，点击确定，该网关将被删除。



删除网关

!!! warning

删除网关后，该网关相关的信息将会丢失，请谨慎操作。

批量删除

如果确实需要一次性删除多个网关，可以采用此项操作。

1. 在网关列表中勾选多个网关后，点击右上角的垃圾箱图标。



删除网关

2. 在弹出窗口中，确认信息无误后，点击确定，选中的网关将全部被删除。

!!! warning

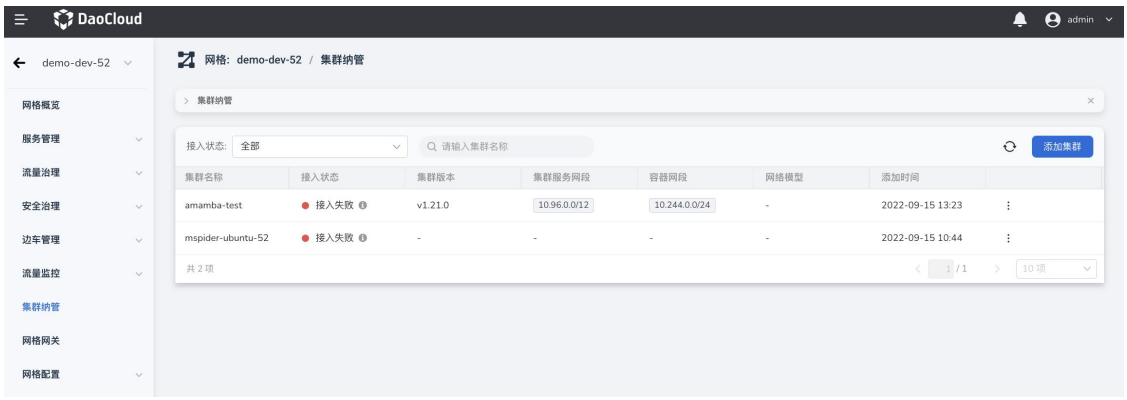
如非必要，请勿使用批量删除功能，删除的网关无法恢复，请谨慎操作。

添加集群

当用户创建一个托管网格后，还未接入任何托管的集群，此时网格处于未就绪状态，用户需要添加一个或多个集群。

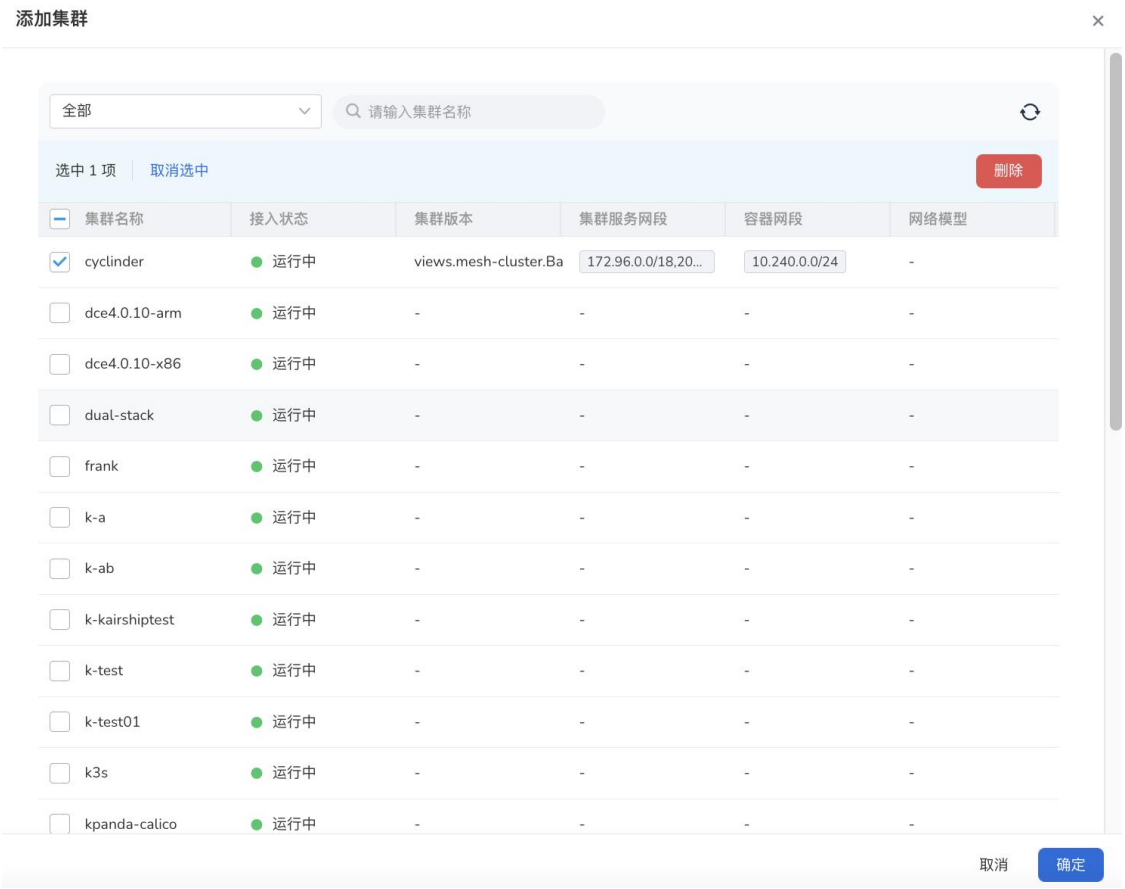
具体操作步骤如下：

1. 在左侧导航栏点击集群纳管，点击添加集群按钮。或者在网格列表中，点击最右侧的 ... 按钮，在弹出菜单中选择添加集群。



添加集群按钮

2. 在弹出的集群列表中，勾选待加入的集群后，点击确定按钮，完成集群的添加操作。



添加集群页面

!!! note

1. 仅 **托管网格** 可以添加集群，专有网格无法添加集群。

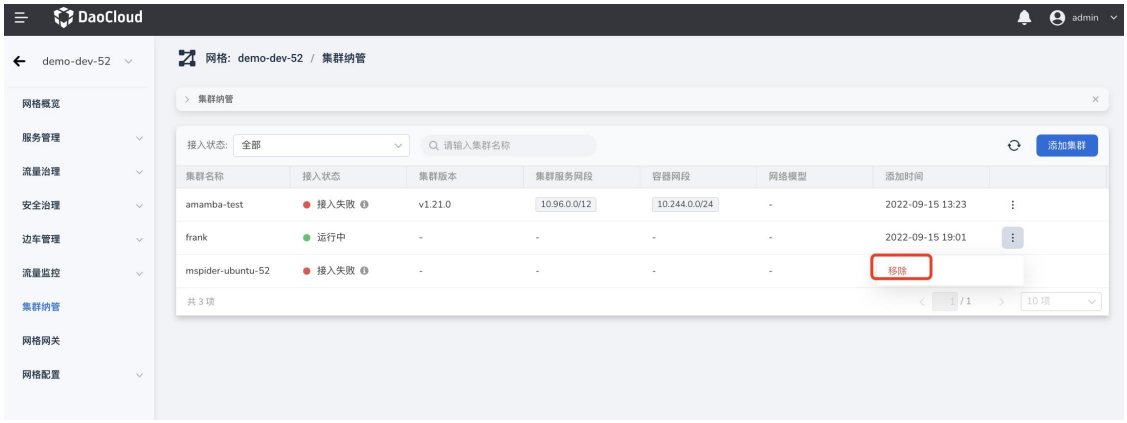
2. 仅可添加状态为`运行中`的集群至网格，其他状态的集群无法选中，也无法添加。

3. 集群添加的过程会持续几秒，请关注网格列表中的`集群统计`信息，了解正常集群的数量变化。

移除集群

如果想要从托管网格中移除一个集群，可以按照以下步骤移除集群。

1. 在左侧导航栏点击集群纳管，在集群列表右侧，点击移除按钮。或者在网格列表中，点击最右侧的 ... 按钮，在弹出菜单中选择移除。



移除集群

2. 在弹出窗口中，确认信息无误后，点击确定。



移除集群

移除集群，需要完成一些前置操作，例如：

- 卸载已注入的边车。
- 清除集群相关的网格网关。
- 其他前置操作，请按屏幕提示操作。

!!! warning

移除集群，将无法通过网格集中管理集群，另外集群日志等相关信息可能会丢失，请谨慎操作。

集群纳管

服务网格的集群纳管页面为用户提供了集群接入、移除功能。

集群接入是一个集群加入网格管理边界的第一步。

完成接入的集群将出现在边车管理的集群列表中，然后用户可以对该集群的命名空间、工作负载启用边车注入，实现对集群的聚合治理管控。

该功能仅在托管网格有效，在专有网格和外接网格模式下未开放该功能。具体参见服务网格分类。 托管网格采用单控制面多集群的方式，实现对多集群统一获取服务注册信息、下发治理策略，各托管集群的控制面组件仅负责接受转发来自唯一管理集群控制面的策略，不执行任何本地创建的策略。

服务条目

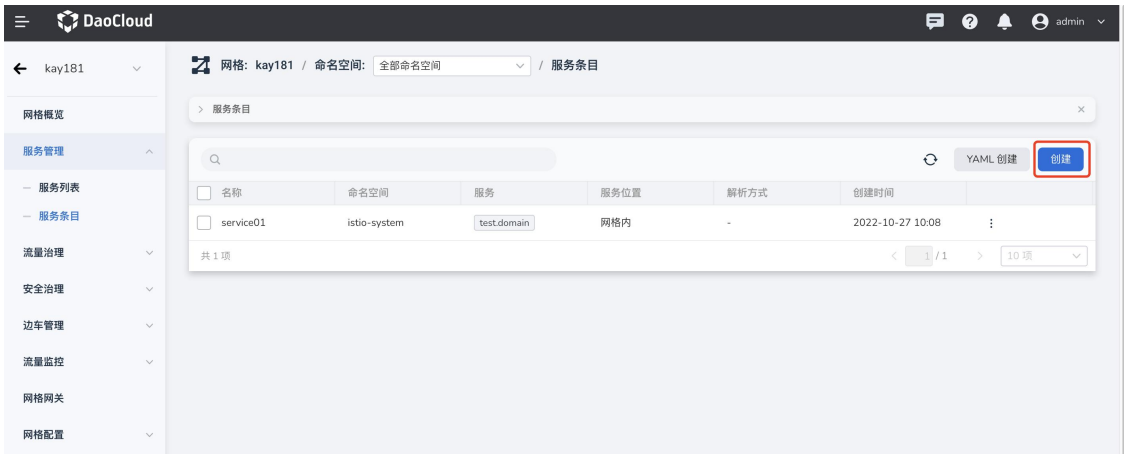
服务条目可以将外部服务、Web API 或虚拟机添加到服务网格的内部服务注册表中。 以一个外部服务为例，添加服务条目后，Envoy 代理可以将流量发送到该外部服务，服务网格也可以通过虚拟服务和目标规则对该外部服务进行流量治理，就好像该服务条目是网格中的正常服务一样。

服务网格提供了两种创建方式：向导创建和 YAML 创建。

图形向导创建

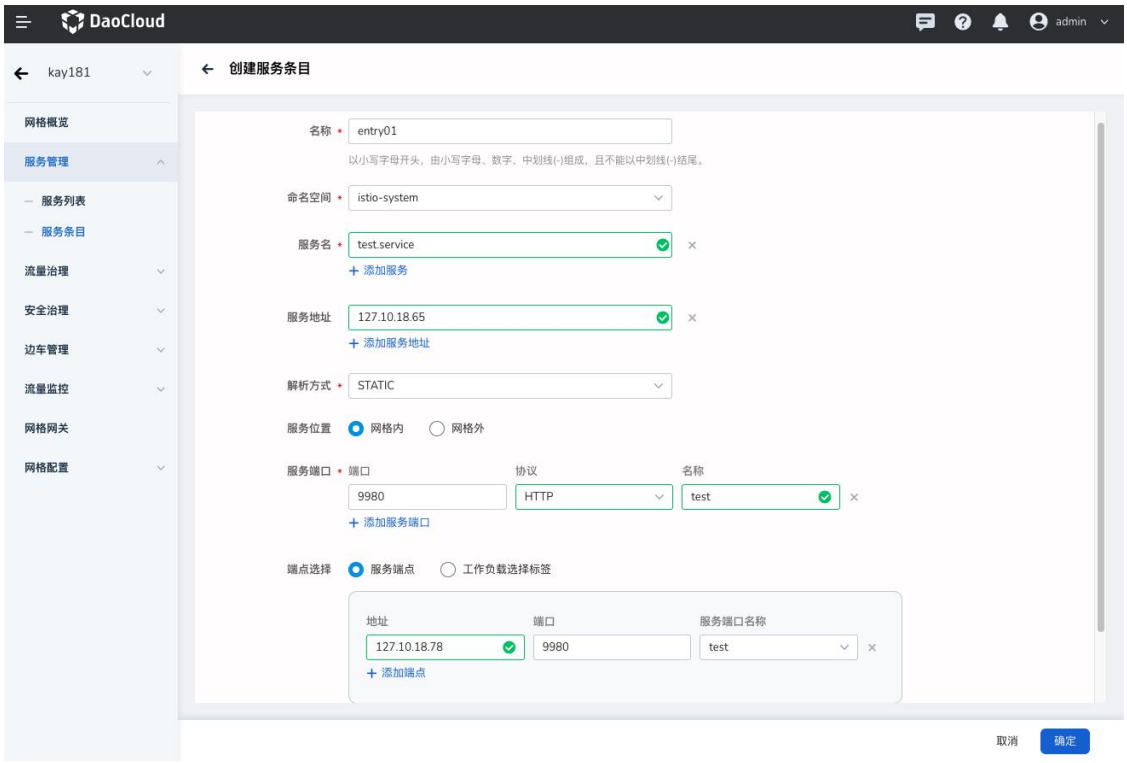
这种创建方式比较简单直观。

1. 进入所选网格后，在左侧导航栏点击流量治理 -> 服务条目，点击右上角的创建按钮。



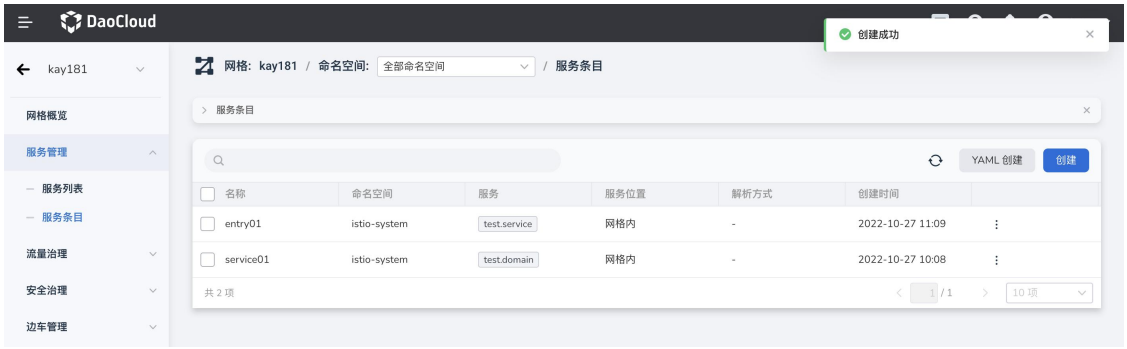
创建

2. 在创建服务条目页面中，配置各项参数后点击确定。具体参数的含义，请参见参数说明。



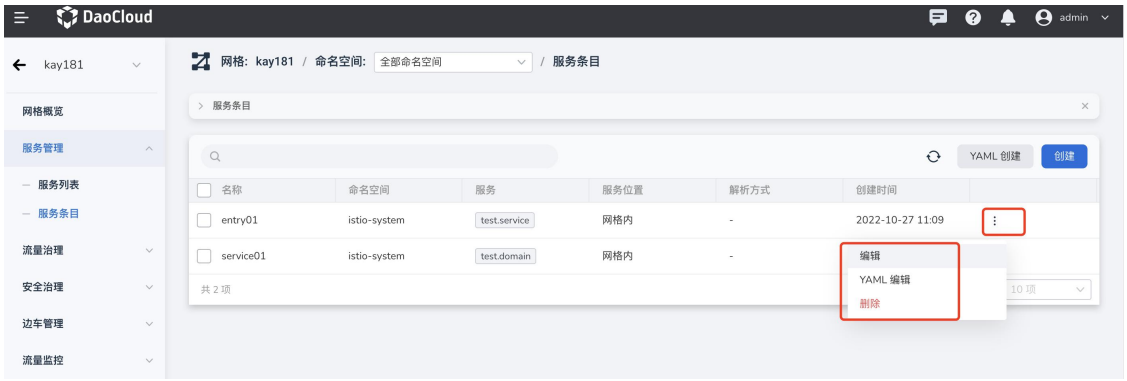
配置参数

3. 返回服务条目列表，屏幕提示创建成功。



创建成功

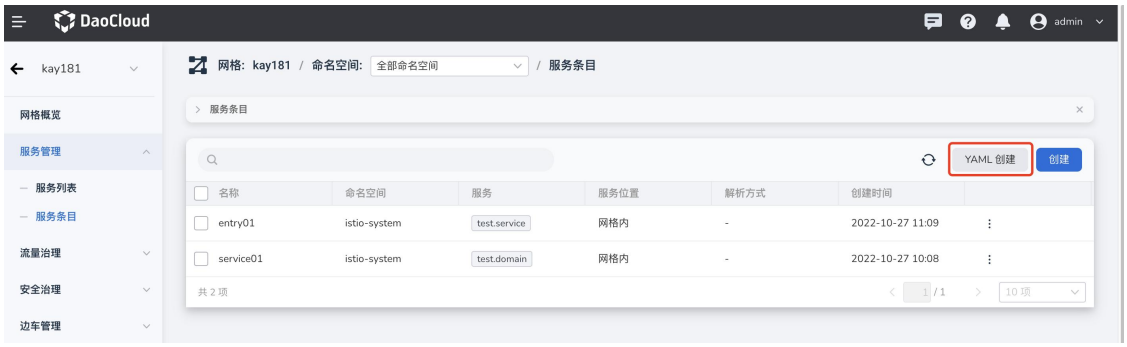
4. 在列表右侧，点击操作一列的 ⋮，可通过弹出菜单进行更多操作。



更多操作

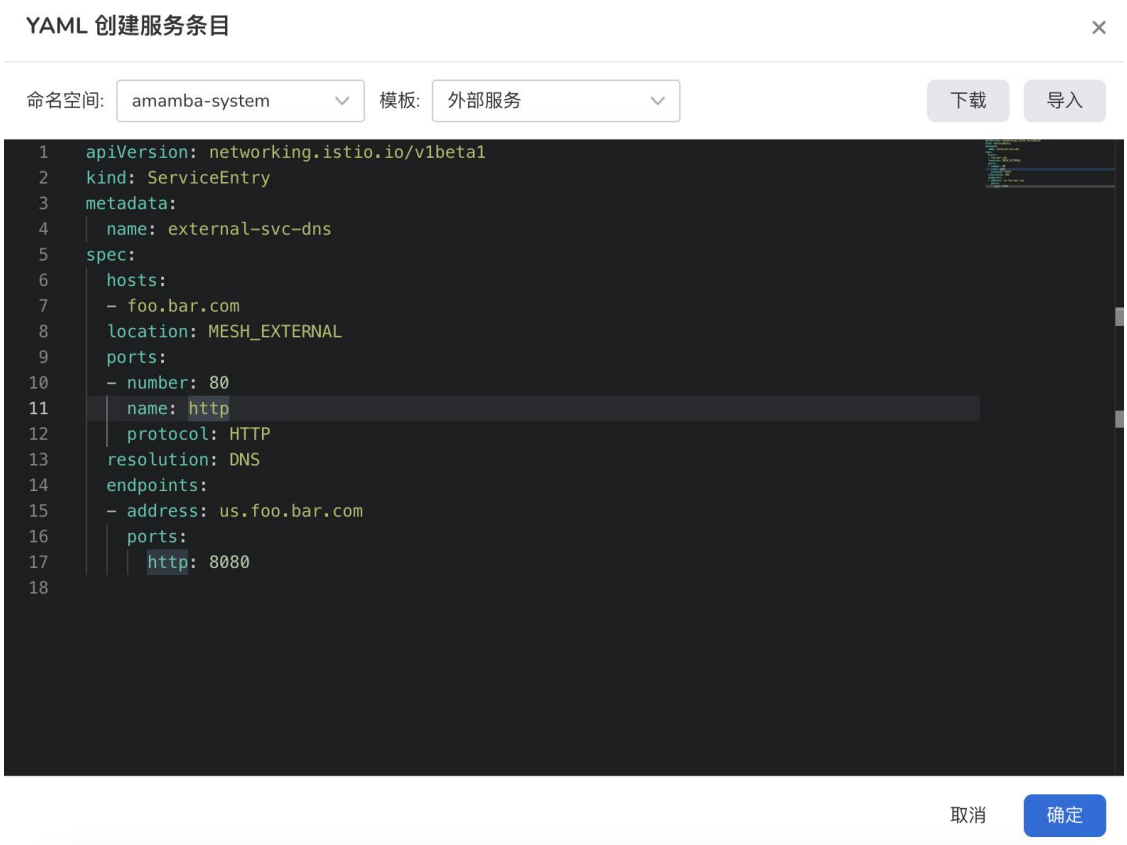
YAML 创建

1. 进入所选网格后，在左侧导航栏点击流量治理 –> 服务条目，点击右上角的 YAML 创建按钮。



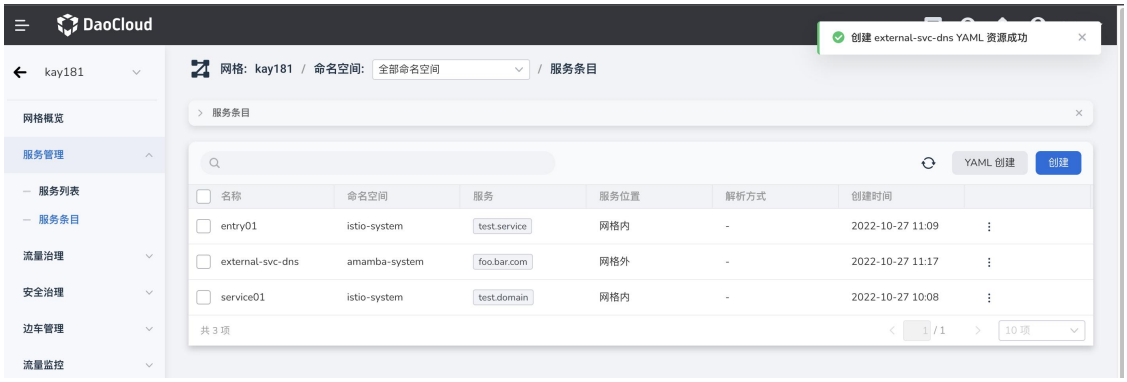
YAML 创建

2. 选择命名空间，选择模板，修改各个字段参数，或者直接导入现有的 YAML 文件，参数确认无误后点击确定。



YAML 参数配置

3. 返回服务条目列表，屏幕提示创建成功。



创建成功

以下是一个标准的服务条目 YAML 示例：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: entry01
  namespace: istio-system
spec:
  addresses:
  - 127.10.18.65
  endpoints:
  - address: 127.10.18.78
    ports:
      test: 9980
  exportTo:
  - istio-system
  hosts:
  - test.service
  location: MESH_INTERNAL
  ports:
  - name: test
    number: 9980
    protocol: HTTP
  workloadSelector: {}
status: {}
```

字段参数说明

上述 YAML 文件和创建向导中各项参数的含义简单说明如下。

- Hosts

  服务名。可用于流量治理策略（虚拟服务、目标规则等）中的 hosts 字段匹配。

  - 在 HTTP 流量中，服务名将是 HTTP 主机或 Authority 头

- 在具有 SNI 名称的 HTTP、TLS 流量中，服务名将是 SNI 名称

- Addresses

  服务地址。与服务关联的虚拟 IP 地址，也可以是 CIDR 前缀。

  - 如果设置了 Addresses 字段，将对请求 HTTP 流量的 服务名和 IP/CIDR 做匹配，确认是否属于该服务。
  - 如果 Addresses 字段为空，则仅根据目标端口识别流量。此时网格中的任何其他服务都不能共享使用该端口，边车将会转发所有该端口的传入流量至指定目标 IP/主机。

- Ports

  服务端口。与服务相关联的端口。如果端点是 Unix 域套接字地址，则必须有一个端口。

- Location

  服务地址。需要输入一个有效的 IP 地址。用于标明服务是否处于网格内部。

- Resolution

  解析方式。提供了对服务地址的多种解析方式：

  - NONE：直接将流量转发至服务地址或服务端点的地址（如果存在）。
  - STATIC：使用服务端点中的静态地址。
  - DNS：尝试通过异步查询环境 DNS 来解析 IP 地址。
  - 如果未设置服务端点，并且未使用通配符，将解析服务名字段中指定的 DNS 地址。
  - 如果指定了服务端点，将解析服务端点中指定的 DNS 地址。DNS 解析不能与 Unix 域套接字服务端点一起使用。

- DNS_ROUND_ROBIN：尝试通过异步查询环境 DNS 来解析 IP 地址。与 DNS 方式不同的是，DNS_ROUND_ROBIN 方式仅使用连接建立后返回的第一个 IP 地址，而不依赖于 DNS 解析的完整结果。

- Endpoint

  服务端点。与服务相关的端点信息，包含 IP 地址、端口和服务端口名称等。

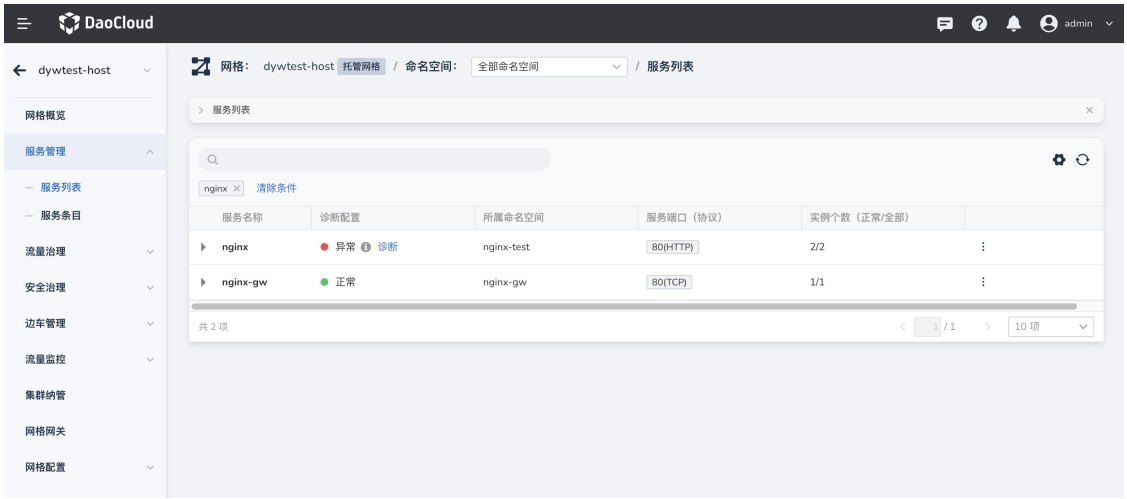
- WorkloadSelector

  工作负载选择标签。一个键值对，用于选择网格内部服务的工作负载，该项和服务端点为二选一。

一键诊断和修复

DCE 5.0 服务网格针对纳管的服务，内置了一键诊断和修复功能，可以通过图形化界面进行操作。

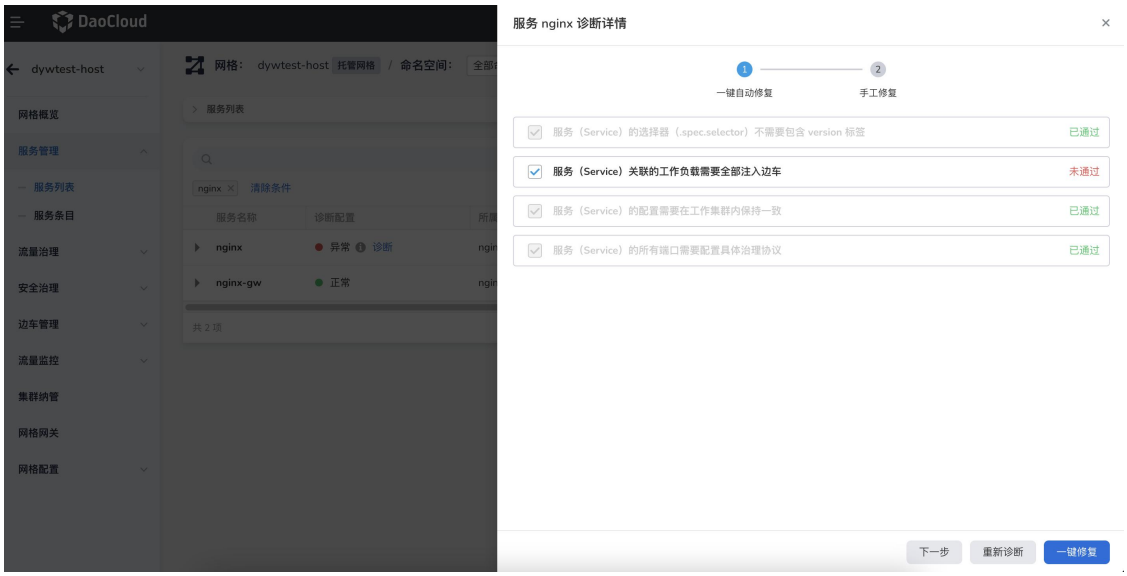
1. 进入某个服务网格后，点击服务管理 -> 服务列表。在诊断配置列中，状态异常的服务旁会出现诊断字样，点击诊断。



诊断

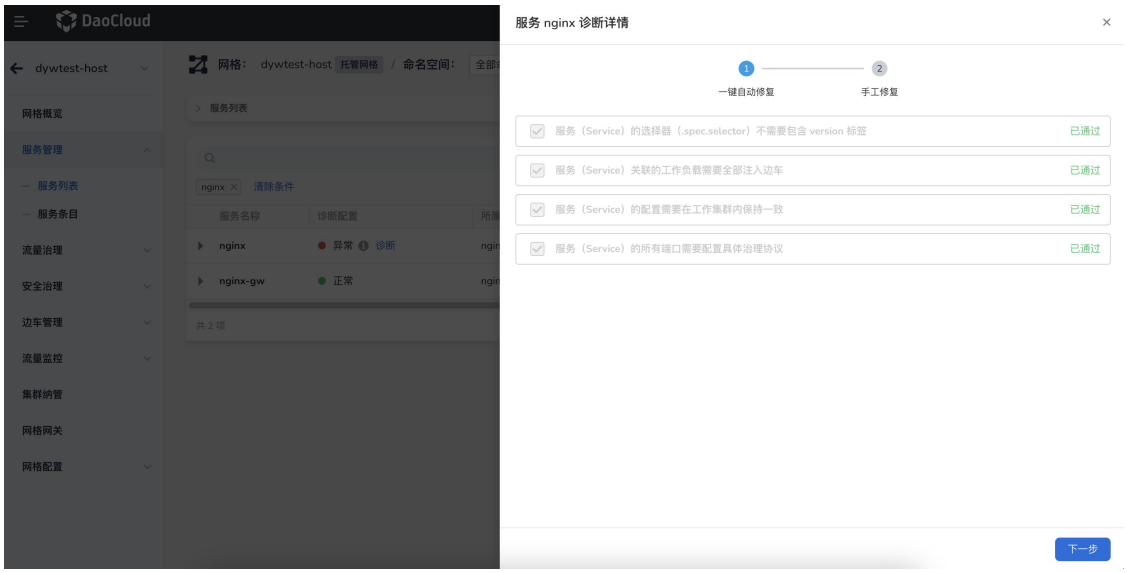
2. 从右侧滑入一个诊断的弹窗，按照内置的 checklist 进行检查。已通过表示正常，未通过表示需要修复。

   勾选未通过的条目，点击一键修复按钮。可以点击重新诊断刷新 checklist，通常会在几分钟内完成修复。



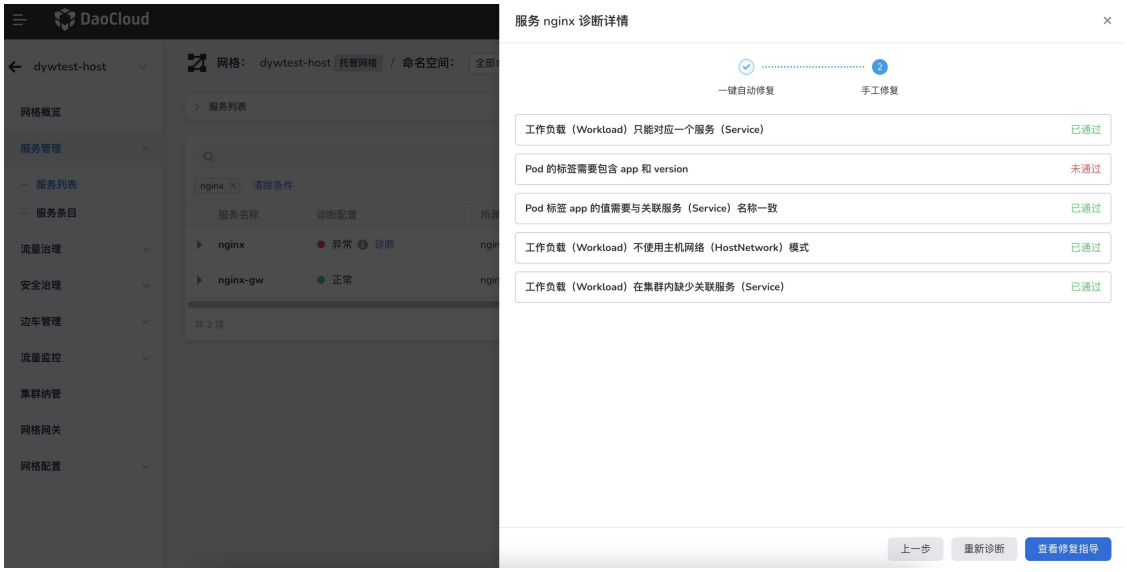
修复

3. 成功修复之后，checklist 各项将变灰且全部显示为已通过，点击下一步。



下一步

4. 列出了需要手工修复的检查项，您可以点击查看修复指导阅读对应的文档页面，手动修复检查项。



手工修复

!!! note

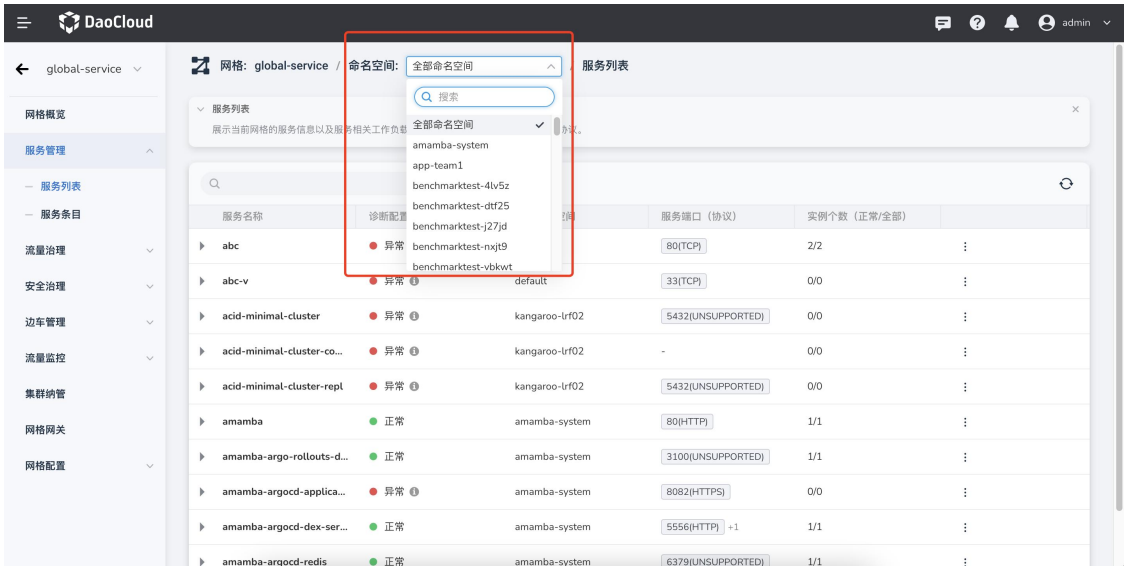
对于以下系统命名空间中的服务，不建议使用一键修复：

分类	命名空间	作用
Istio 系统命名空间	istio-system	承载 Istio 控制平面组件及其资源
	istio-operator	部署和管理 Istio Operator
K8s 系统命名空间	kube-system	控制平面组件
	kube-public	集群配置和证书等
	kube-node-lease	监测和维护节点的活动
DCE 5.0 系统命名空间	amamba-system	应用工作台
	ghippo-system	全局管理
	insight-system	可观测性
	ipavo-system	首页仪表盘
	kairship-system	多云编排
	kant-system	云边协同

分类	命名空间	作用
	kangaroo-system	镜像仓库
	kcoral-system	应用备份
	kubean-system	集群生命周期管理
	kpanda-system	容器管理
	local-path-storage	本地存储
	mcamel-system	中间件
	mspider-system	服务网格
	skoala-system	微服务引擎
	spidernet-system	网络模块

服务管理

服务管理列出了当前网格下集群中已注入边车的所有服务，您可以基于命名空间筛选服务。



服务列表

服务网格对各集群的服务做了聚合处理，同一个命名空间下的同名服务将聚合为一个服务，这样有利于对跨集群协同服务进行统一的流量治理。

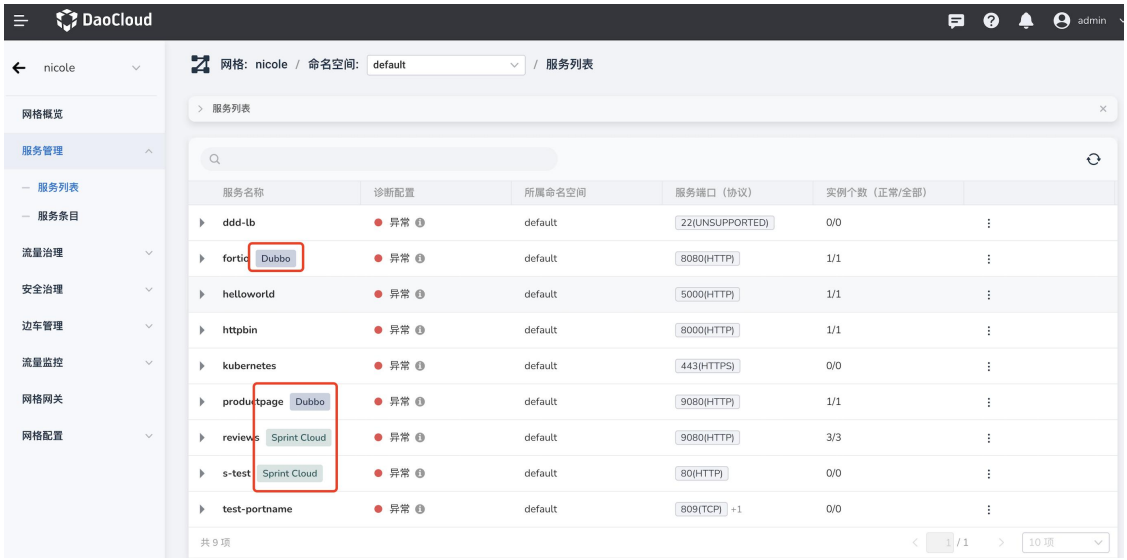
服务管理会对列表内服务状态做诊断配置检测，检测结果及含义如下：

- 正常：服务在各集群的 Pod 均已注入边车，并且各项设置完全相同
- 告警：该服务下工作负载的部分 Pod 没有注入边车
- 异常：当服务存在以下任一问题时，都会显示为异常状态
  - 所有 Pod 均未注入边车
  - 服务在各集群的端口设置不一致
  - 服务在各集群的工作负载选择器标签设置不一致
  - 服务在各集群的访问方式不一致

!!! note

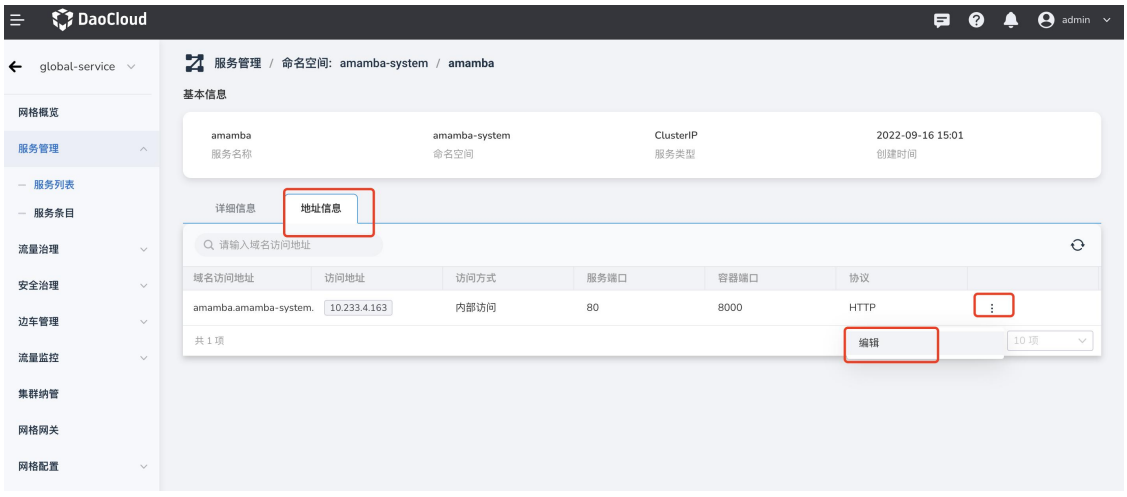
各集群内同名服务应尽量保证各项配置完全一致，否则可能导致部分工作负载无法正常工作。

服务列表中也标注了来自其他微服务注册中心的服务，如下图所示。



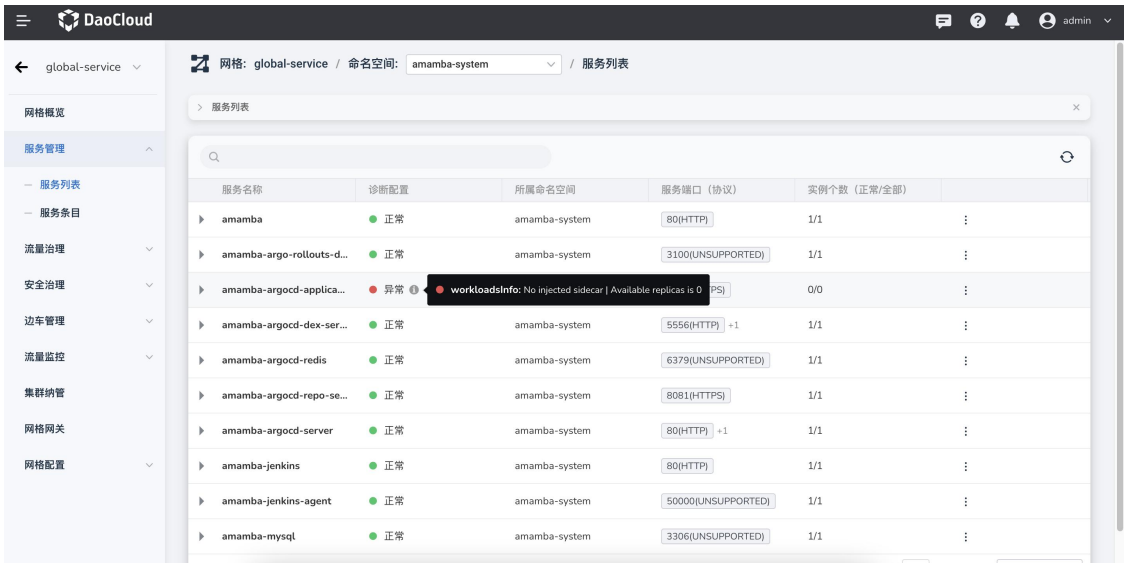
其他微服务注册中心的服务

您可以点击某个服务名称，进入详情页查看各集群的服务地址、端口等具体信息，还可以在地址信息页签中修改通信协议。



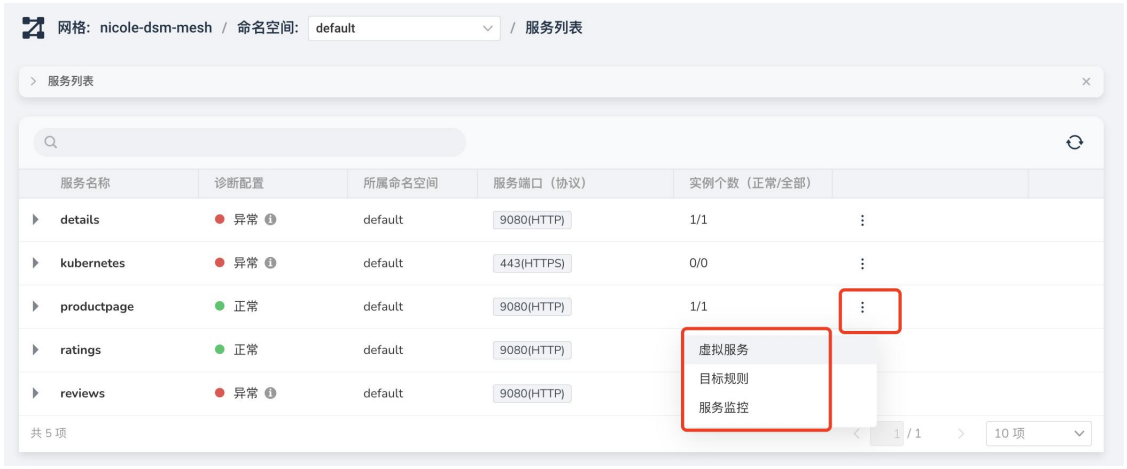
地址信息

请特别关注服务列表中诊断配置这一列。当诊断信息为异常时，光标悬浮在 ⓘ 上将显示异常原因。异常状态会影响下一阶段的服务流量治理等网格相关能力。



异常提示

在服务列表右侧，点击 ⋮ 选择相应的菜单项，可以跳转至流量治理和安全治理。



菜单项

关于如何创建和配置服务，请参阅创建服务。

外部应用对网格内服务访问

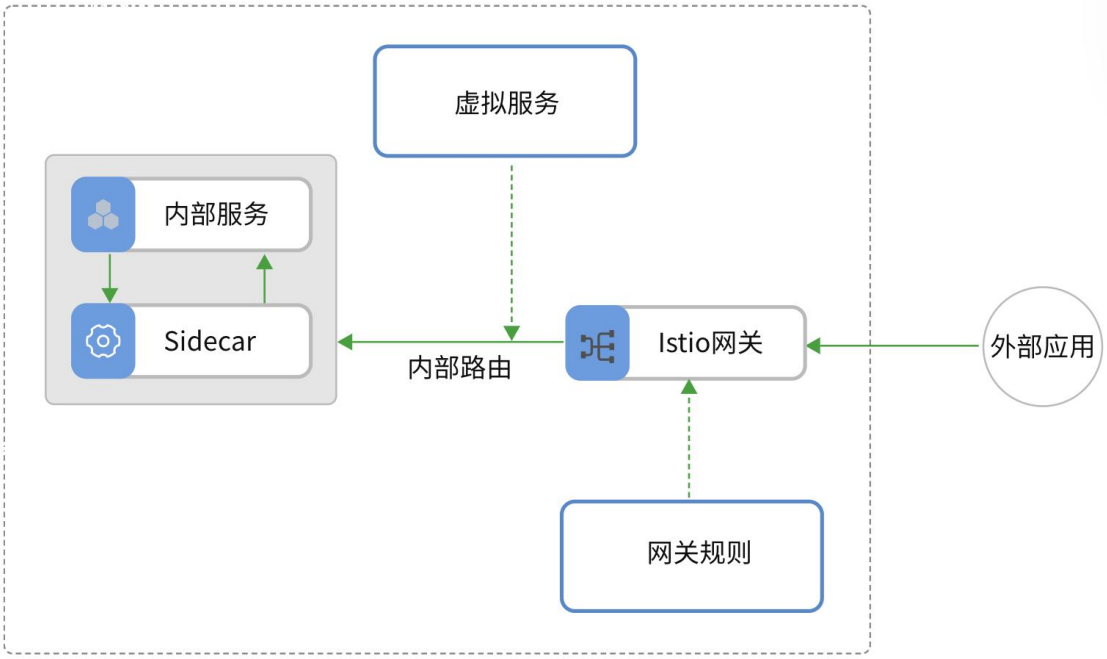
本页说明外部应用如何通过配置来访问网格内的服务。

前置条件：

- 服务 bookinfo.com 运行于网格 global-service 的 default 命名空间下
- 网格提供 ingressgateway 网关实例

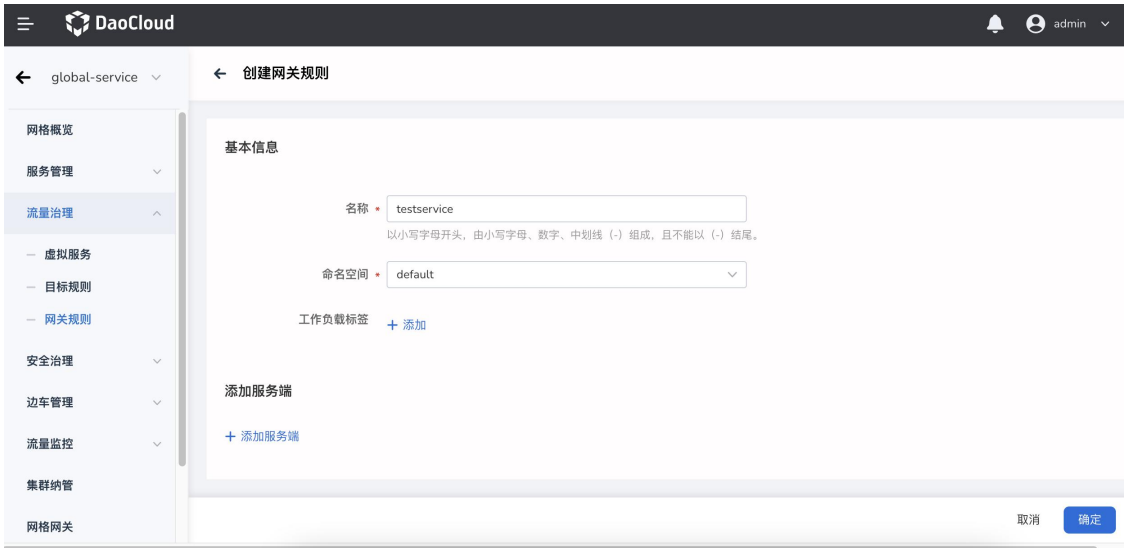
配置目标： 实现内部服务 bookinfo.com 对外暴露。

1. 通过 URI 匹配方式，实现外部应用对服务 bookinfo.com 的指定页面访问路由。



访问路由

2. 点击流量治理 -> 网关规则 -> 创建为 istio 网关创建网关规则，对外暴露服务及端口。



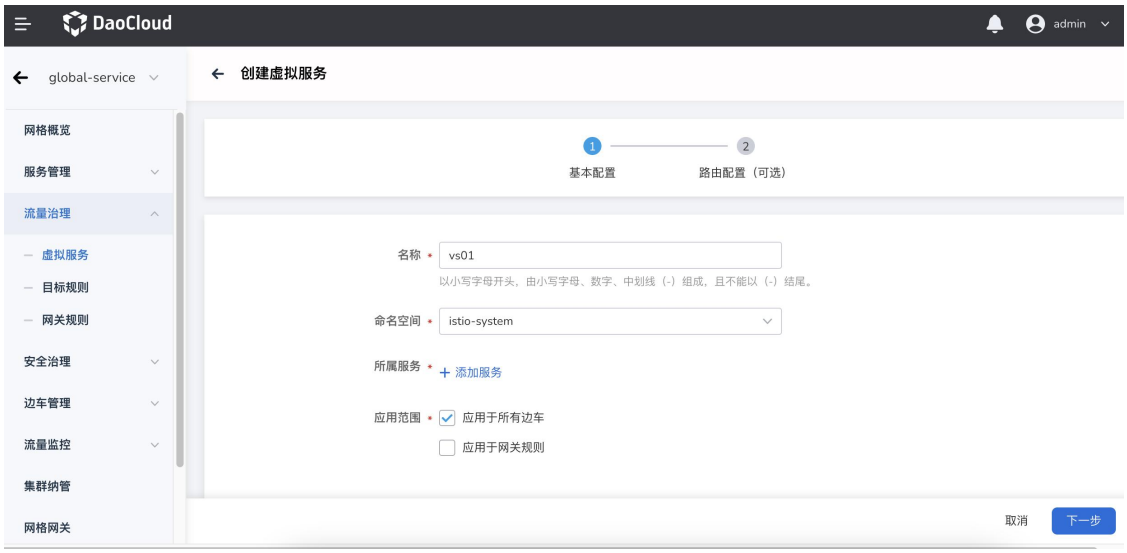
创建规则

完成配置后的 YAML 示例如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # 使用默认控制器
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - bookinfo.com
```

3. 点击确定回到网关规则列表，可见创建成功提示。

4. 点击流量治理 ->虚拟服务 -> 创建来创建路由规则，基于请求中的 URI 路由到指定页面。



创建路由规则

完成配置后的 YAML 示例如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - bookinfo.com
  gateways:
  - bookinfo-gateway
  http:
  - match:
      - uri:
          exact: /productpage
      - uri:
          exact: /login
      - uri:
          exact: /logout
      - uri:
```

```
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

5. 点击确定回到虚拟服务列表，可以看到创建成功的提示。

!!! info

更直观的操作演示，可参阅[视频教程](../../../videos/mspider.md)。

多云网络互联

多云网络互联是在多集群模式下，多集群之间网络互不相通（Pod 无法直接建立通讯）的情况下提供的一套解决方案，可以快速地将多集群的网络连通，实现跨集群的 Pod 互相访问。

使用场景

如果您的环境满足以下条件，可尝试请使用多云网络互联功能：

1. 同一个网格有多个集群
2. 多集群的 Pod 网络无法直接连通（包括 Pod CIDR 冲突等情况）

!!! note

多云网络互联仅适用于托管网格。

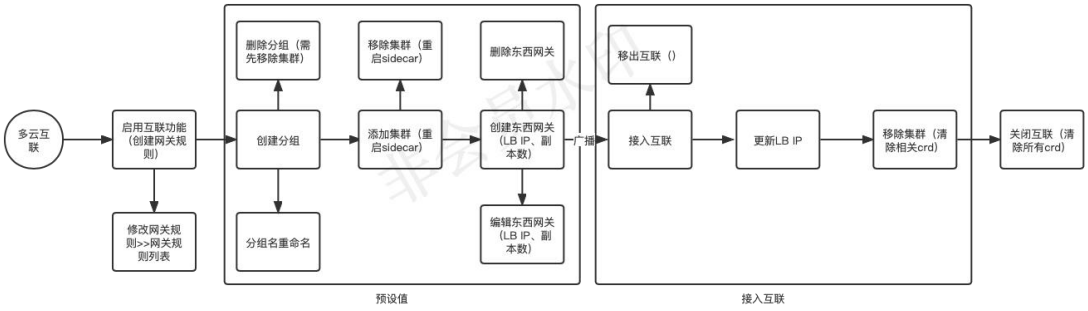
名词解释

- 网络分组：
  - 用于辨别网格内集群的网络拓扑关系。
  - 如果某些集群的网络能够直接互通（跨集群的 Pod IP 没有冲突且能够直接连通），那么这些集群应该被放在一个网络分组里，否则将属于不同的网络分组。
  - 网络分组和集群的网络 CNI 实现类型没有明确的关联关系。
- 东西网关：
  - 用于不同网络分组间通信互联，一个网络分组内可以创建多个。
  - 用户可以在网络分组下任意集群创建东西网关，当移动集群至其他网络分组时，东西网关也会随之迁移。
  - 东西网关所用负载均衡 IP 通常有集群所在的云平台自动分配，也可以由用户配置设定。

- 基本设置：创建用于东西网关的网关规则配置，该配置将开放 15443 端口用于网络分组间的通信，用户可以在网关规则列表中查看 CRD 文件详细内容，但不建议修改，可能会导致网络通信失败。
- 网络分组列表：
  - 展示当前创建的网络分组及分组下集群、东西网关信息；
  - 在该列表下用户可以对网络分组、分组下集群及东西网关执行增删操作，并可以在网络分组间迁移集群。
- 互联列表：
  - 包含至少一个东西网关的网络分组可以加入互联列表，加入互联的网络分组可以和其他网络分组通信。
  - 已加入互联的网络分组可以修改负载均衡 IP，但如需修改其他配置（例如增删集群、东西网关），需要先移除网络分组互联列表。
  - 加入互联和移除互联会引起网格控制面的重启，建议谨慎操作。

操作流程

建议操作流程如下图所示



操作流程

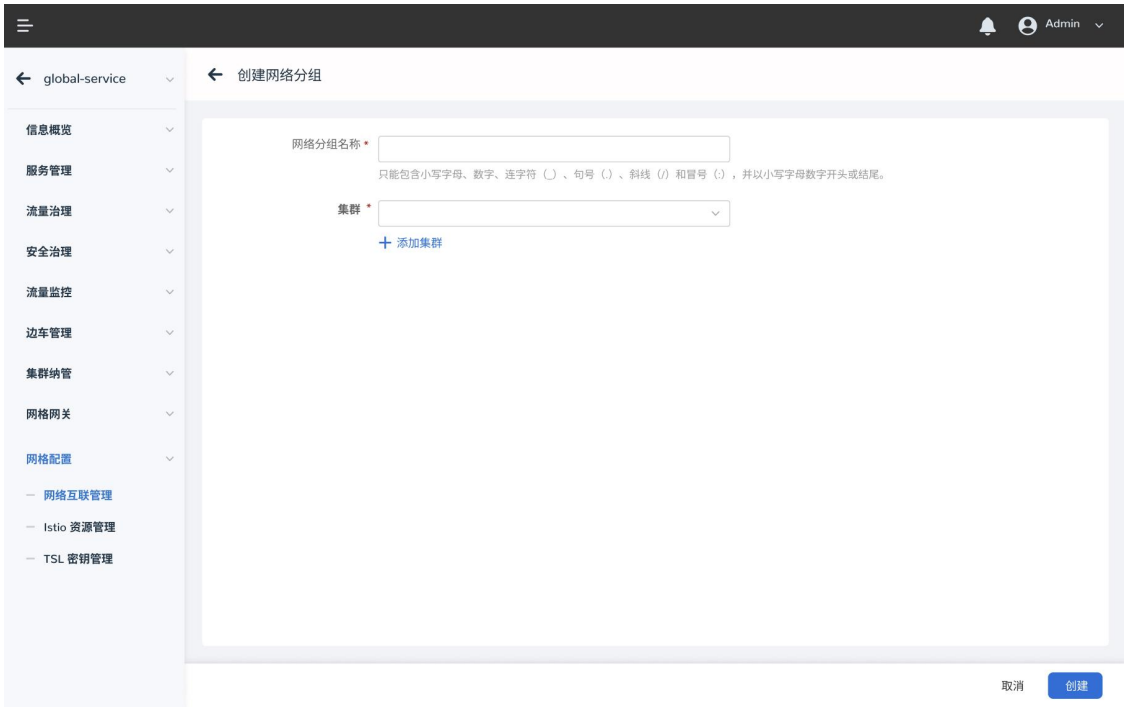
操作步骤

1. 进入某个网格，点击启用开关，将自动创建用于东西网关的网关规则。点击创建网络分组按钮，为网络分组添加至少一个集群。



创建

2. 为网络分组填入名称，并添加至少一个集群。



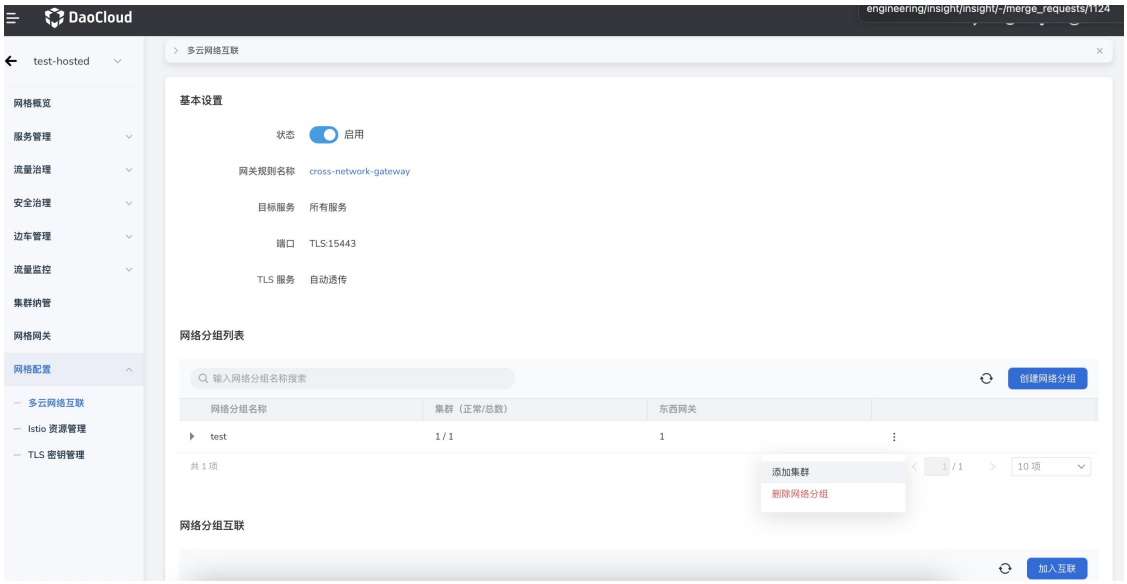
创建对话框

!!! note

网络分组名称创建后无法修改。

3. 增删集群。

创建网络分组后，即可为分组内添加更多的集群，同一分组的集群需确保处于同一网络类型并可以互通，否则可能造成互联失败。



添加集群

添加完成后即可在网络分组下看到多个集群。



集群列表

!!! note

网络分组中至少要保留一个集群，如需移除所有集群，请删除网络分组。

4. 创建/删除东西网关。

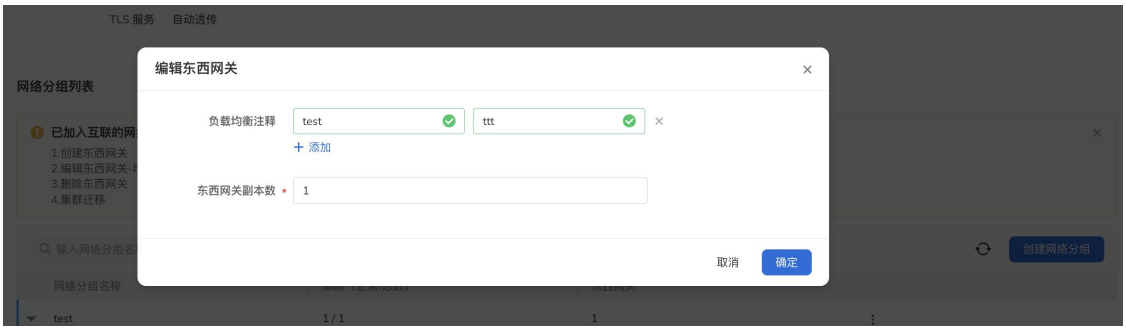
东西网关用于网络分组间通信，可以在分组内的任一集群上创建一个或多个网关。点击一个集群右侧的 ⋮，选择编辑东西网关：



编辑东西网关

创建配置项如下：

- 负载均衡注释为可选设置，部分云平台会以注释方式提供负载均衡 IP 分配，请参考云平台提供的技术文档；
- 东西网关副本数默认为 1，如果需要提高网关可用性，可创建多个副本；点击确定；



网关参数

选择一个东西网关，点击操作中的删除东西网关，可以删除该网关，如图所示：



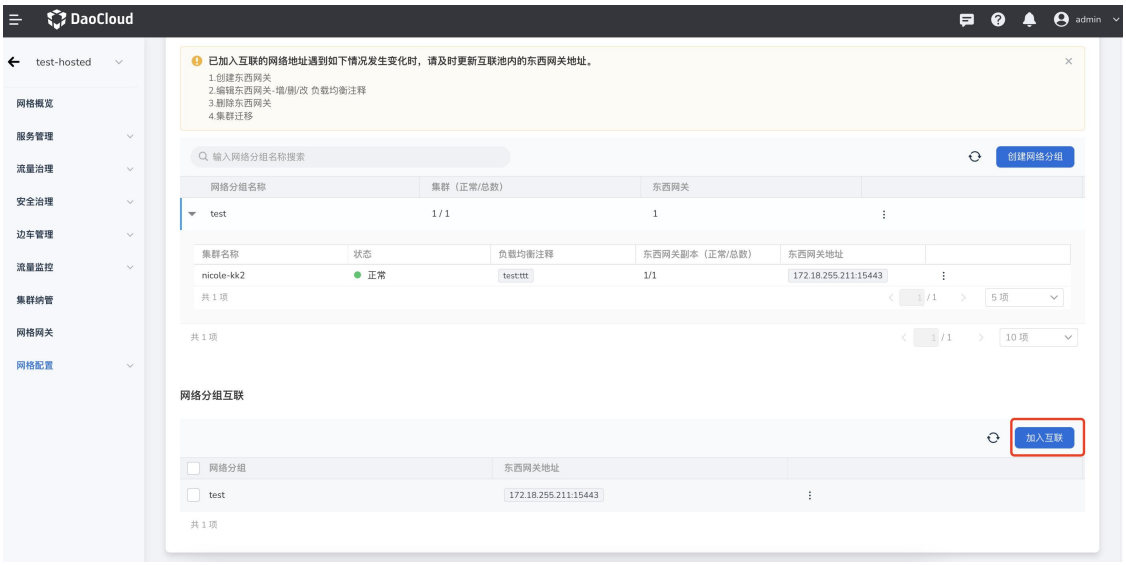
删除网关

!!! note

每个网络分组至少需要创建一个东西网关，否则无法加入互联。

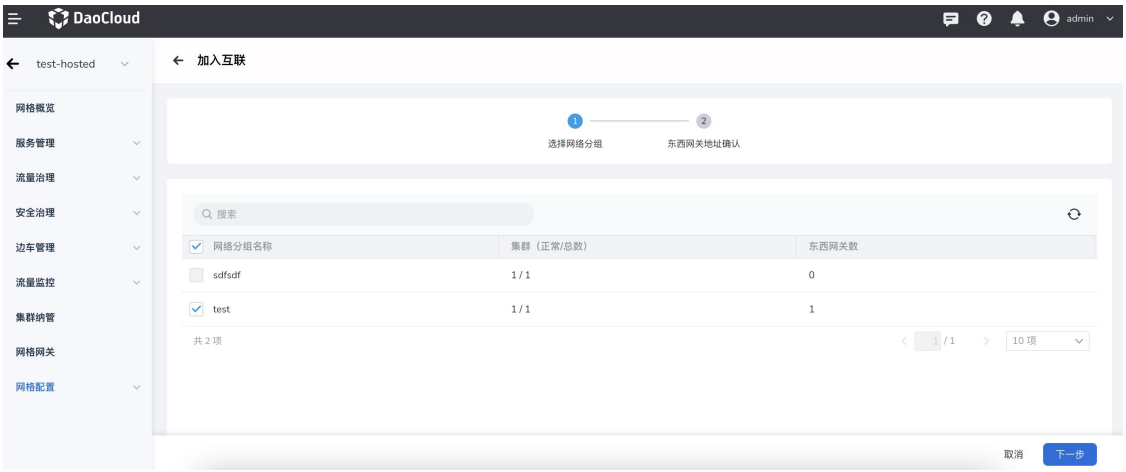
5. 网络分组互联

1. 如果要将一个网络分组加入互联，点击加入互联按钮。



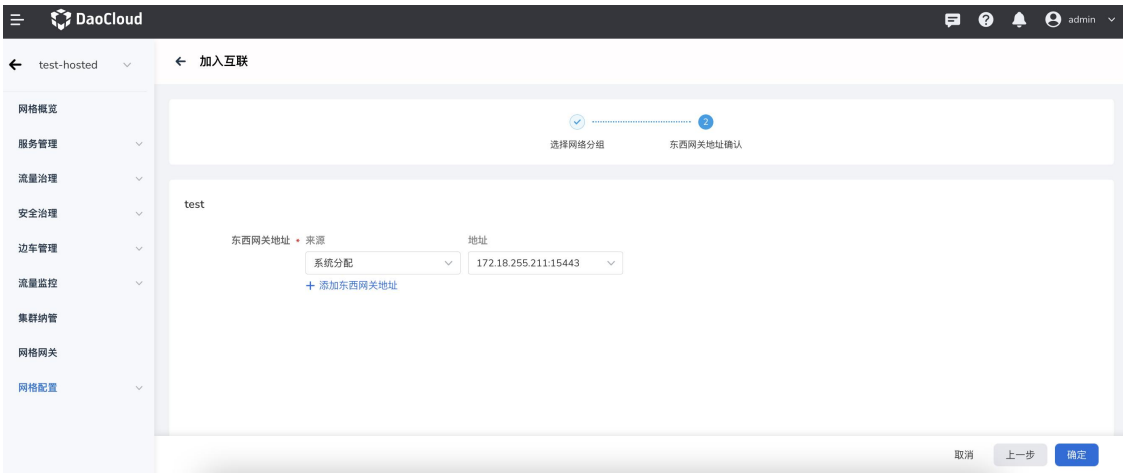
加入互联

2. 勾选一个网络分组，点击下一步。



选择集群

3. 选择可用的东西网关地址，点击确定。



选择东西网关地址

4. 在互联列表中可以看到新加入的网络分组，列表内的分组之间彼此建立了通信。

!!! note

– 没有东西网关的网络分组无法加入互联

– 加入分组互联操作会引起网格控制面重启，建议加入互联前确保网络分组配置无误

其他操作

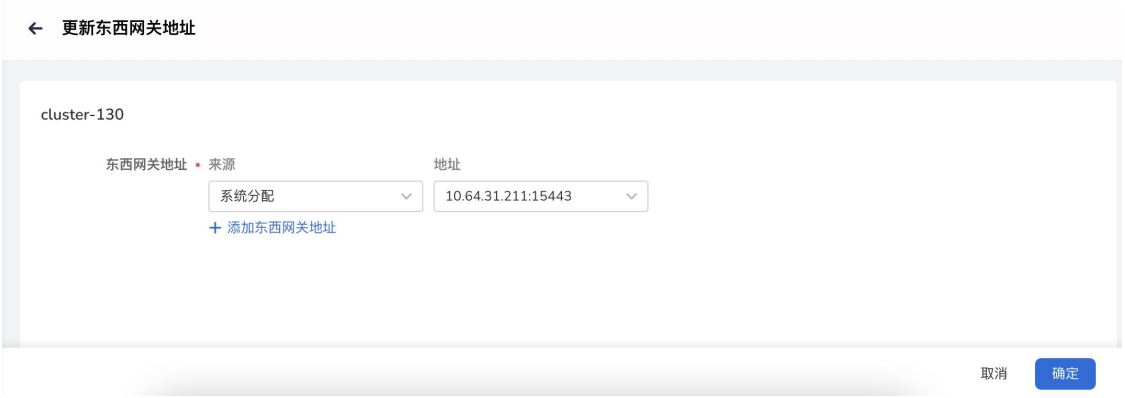
1. 更新东西网关地址。

   用户可以对处于互联状态的网络分组增删东西网关地址。在互联列表中勾选一个分组，点击更新东西网关地址。



   更新东西网关地址

   更新地址后点击确定。



   确定

2. 移除互联

   如果网络分组不再需要和其他网络分组建立互联，或需要调整分组内集群及东西网关设置，可以移除互联。 选择需要移除的网络分组，点击移除互联按钮，出现下图对话框。

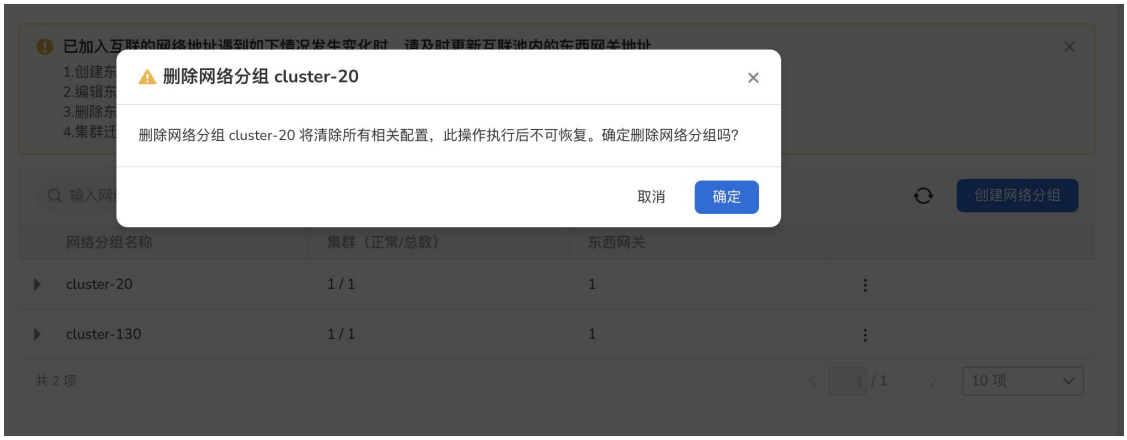
   移除互联

   移除互联

经过二次确认，网络分组即可移除互联列表。

3. 删除网络分组

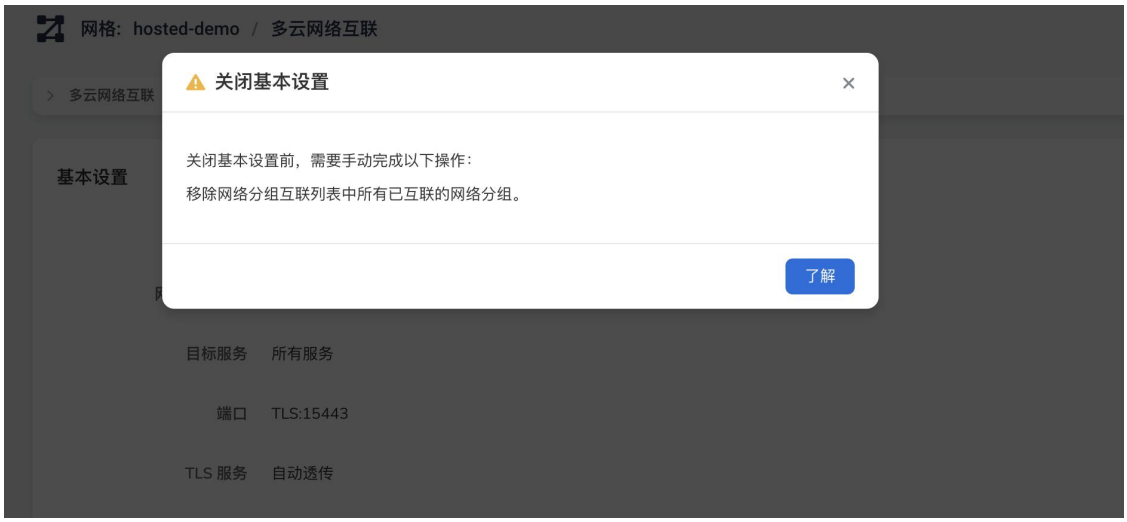
在需要删除的网络分组操作下拉框中点击删除网络分组，即可删除所选网络分组。



删除分组

4. 关闭多云网络互联功能

在顶部基本设置区域，点击启用状态的滑块，将弹出关闭多云互联的前提条件：



关闭互联

先移除网络分组：



创建

在文本框输入关闭基本设置进行确认后，即可关闭多云互联功能：



创建

多集群网络通讯原理与架构分析

近年来，多集群架构已经成为“老生常谈”。我们喜欢高可用，喜欢异地多可用区，而多集群架构天生就具备了这样的能力。 另一方面我们也希望通过多集群混合云来降低成本，利用到不同集群各自的优势和特性，以便使用不同集群的最新技术与支持现实中越来越复杂的云上业务。 最为本质的问题时单集群模式下存在承载能力上限，并且集群故障时，无法快速故障转移。

就是因为这些种种原因，多集群几乎成为了云计算的新潮流。 随着新架构的诞生，新的难题也接踵而来，尤其是在多集群场景下，其中业务的治理、监控、网络通讯也面临着严重的考验。 这时不得不提到服务网格，该技术从理论上来说对于多云多集群场景下，从设计基因中就自带了解决上面问题的能力。

为什么说服务网格能够支持复杂的多云场景呢？ 服务网格将自身能力下沉到基础设施层面，因此从理论上来说，在多云环境中，它天然能够感知自己云上业务的各种信息，比如说服务地址、服务之间链路关系等网络信息， 网格将感知的信息聚会成一个完整的多云网络拓扑，在一个已知且能控制的多云拓扑中，用户能够从高纬度的观测并且控制业务通讯。

上述都只理论分析，再多的理论分析也不如一个实践论证，因此我们将自身服务网格产品（Mspider）到底如何支持多云环境进行一个由浅入深的整体剖析。

多云服务网格的优势

首先，我们回到最初的起点，我们为什么要选择多云服务网格？ 所有的技术都是为了解决问题而存在，不具有优势与解决问题的技术，是虚假且没有意义的。 因此我们简单介绍一下相对于单集群场景下，多云场景下的服务网格能够给我们带来哪些优势?

故障隔离与故障转移

在单集群场景下，业务服务 A 出现问题时，我们通过将业务流量转移到同集群的另一个业务实例上。 但是如果集群出现故障时，这时候会变成多个业务同时故障，这时候如何将该集群的所有业务转移到另一个集群上，这是已经非常费时且复杂地。 但是实际地生产环境下，每分每秒是非常昂贵。 但是多云场景下，集群维度的故障转移是基本能力，用户能够快速地转移其所有业务的流量。 用户只需要定义集群之间故障转移的策略，例如 A 集群故障时，将业务流量转移到集群 B 中。

地域感知与故障转移

当多云环境下，对云上的地域进行定义；在业务健康时，可以让业务之间通讯以就近原则，发生故障时，可以快速转移至最近的地域环境中。

地域感知对于多云的重要性不言而喻。 因为现实世界、实际地域区域会影响网络通讯，中间有时间空间资源等多种因素。 例如当中国北京的集群都故障时，随机将业务转移到美国，但是实际环境中中国上海的集群空闲在那里，这肯定是不合理的。

多维度团队或项目隔离

虽然随着多云的体量扩大，但是团队的复杂度并没有增加，团队可以将多云划分成不同的区域、不同的集群、不同业务项目。 团队拥有多云能力的情况下，可以只需要管理自己相关的集群与业务。

多云维度的观测能力

单集群模式下，用户只能知道当前集群内的业务运行情况，每个集群之间的业务通讯时不透明且独立的。用户无法感知到集群外的业务通讯链路。

但是多云场景下，只要属于同一个网格，网格能将同一网络下不同集群之间的通讯情况进行聚合分析，将其绘制成一张完整的多云网络拓扑。 好比原先的单集群只是一张 2D 的照片，现在多云场景下，服务网格能够提供一张 3D 立体的拓扑网络。 这不仅仅是范围的扩大，而是维度层次上的增加。

多云下网络通讯原理

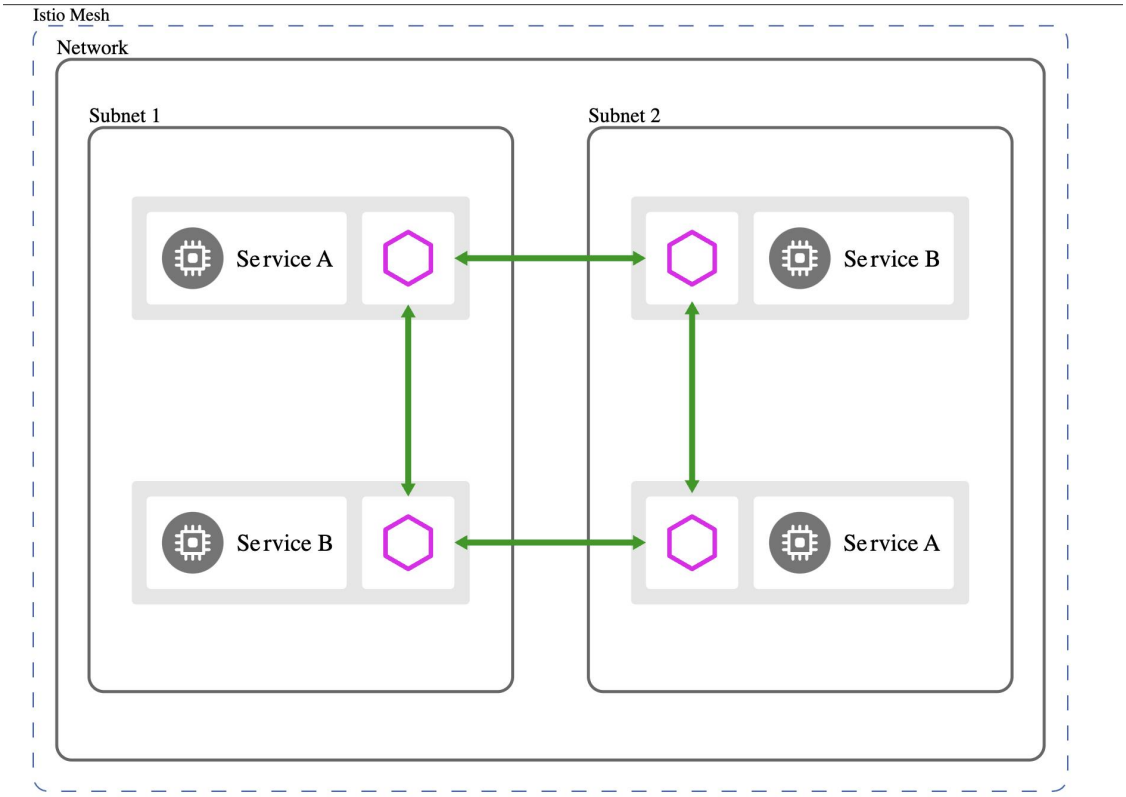
在了解了多云场景下巨大的优势，我们回归现实，多云场景有伴随着哪些问题？其中哪个问题最为根本？ 首先我们从字面上理解一下这个意思，多个云环境？多个云厂商？ 多个云集群？这里的多个，就意味着存在不同。 而且在多云世界里，网络通讯是一个非常硬核的问题，云厂商之间能够直接通讯吗？ 多集群之间如何通讯无阻？

在单集群的情况，所以的工作负载都在同一网络上（集群内部网络），但是多集群的场景下，每个集群可能存在多种因素导致网络无法互通，这里列举常见的几种情况：

- 地域问题导致集群实际网络之间存在网络隔离
- 初始化集群时，保持默认配置导致集群子网都一样，因此集群与集群之间网络会存在 IP 冲突。 以上几种情况下的多集群之间网络通讯是存在问题的，为了保持集群内部网络的独立性以及集群网络之间网络打通，Istio 提供了两种网络模型，用户可以根据集群状态与用户需求自由选择不同的网络模型。

单一网络模型

顾名思义，在单一网络的情况下，多集群之间的网络实例能给直接通讯，这时候不需要网格进行任何操作，服务之间网络通讯能够直达，其架构图如下：



架构图

单一网络模型的优势是非常明显的，用户通讯时，无需复杂的配置和中转，因此网络解析也更加直观，因此也能带来更高的网络性能。

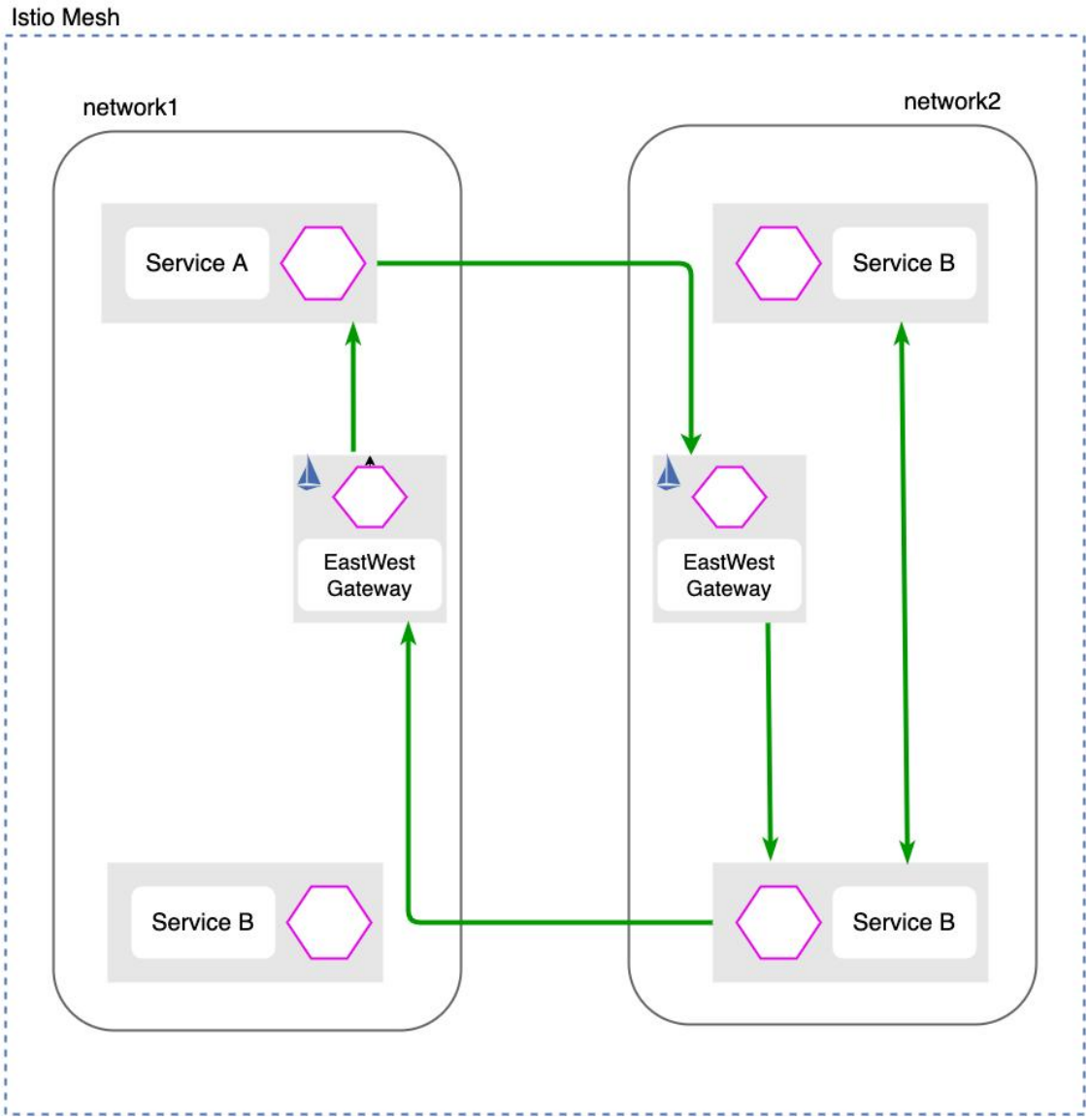
在这时不得不提的是，有的用户会问，如果拥有多个集群，并且当时多集群之间网络存在冲突，但是又想要多集群之间网络选择单一网络模型时如何做？在这里简单提供一些网络改造需要解决的问题：

- 子网冲突问题：需要解决集群实例网络与集群服务网络的子网冲突问题
- 网络隔离问题：集群之间网络可能由于地域、安全等多种因素造成网络隔离的现状，用户可以选择通过隧道或直接路由，解决跨多个集群的 Pod IP 路由。

多网络模型

在多网络模型中，不同网络中的工作负载实例不能够直接通讯，因此服务网格提供了一种解决方案，让业务实例通过一个或多个 Mesh 网关相互访问，从而解决工作实例网络不能互相通讯的问题。

在服务通讯时，网格会根据网络分区进行服务发现，当请求服务与目标服务不属于同一网络分区时，服务请求将会转发给目标分区的东西网关，通过东西网关路由到真实的目标服务。



服务请求分发

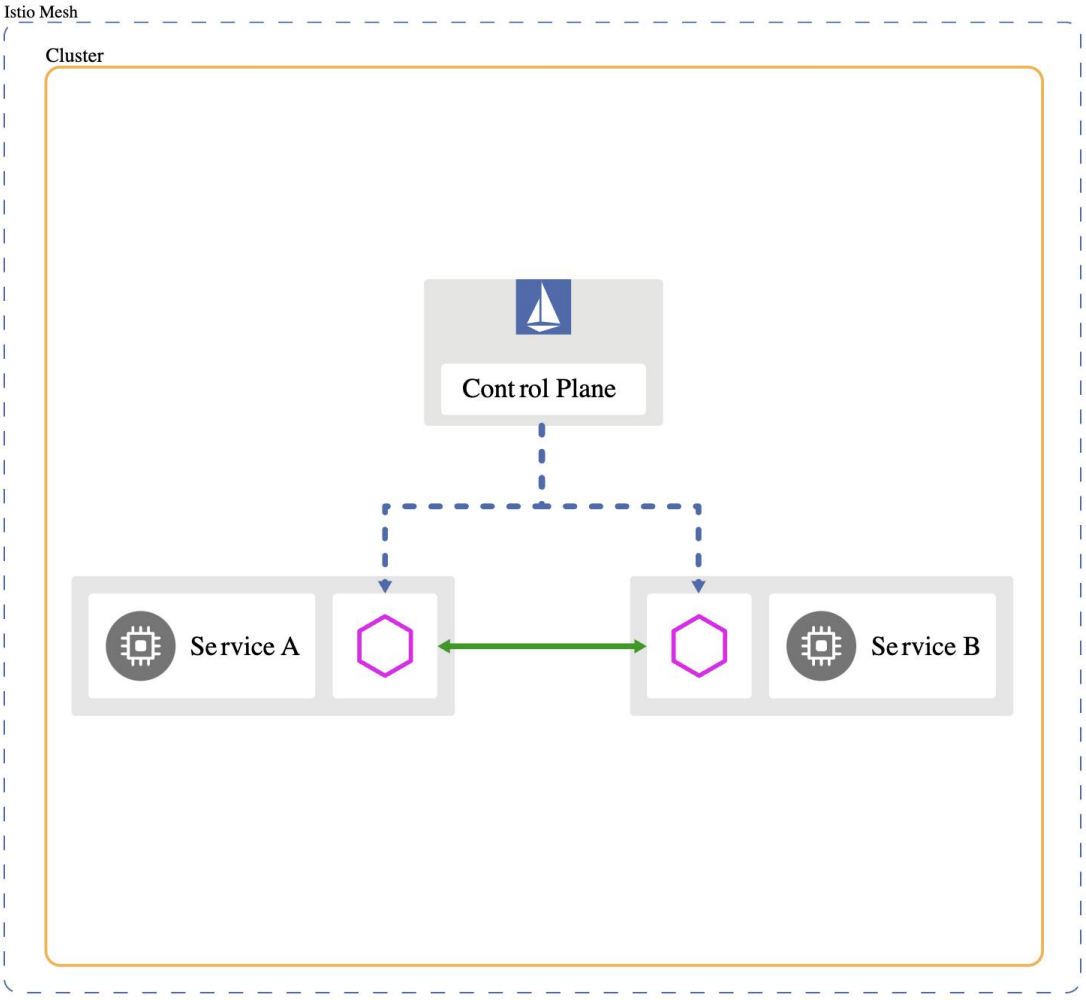
多云服务网格架构的抉择

在上述网络方面的思考以后，用户可能对服务网格自身架构存在好奇，例如服务网格控制面组件位置在哪里？ 如何运行多集群模式？ 这架构拥有哪些优势或者缺陷？ 让我们具体分析现有服务网格提供的架构，现在主要有如下三种架构：

- 单集群单控制面
- 多集群单控制面
- 多集群多控制面

单集群单控制面

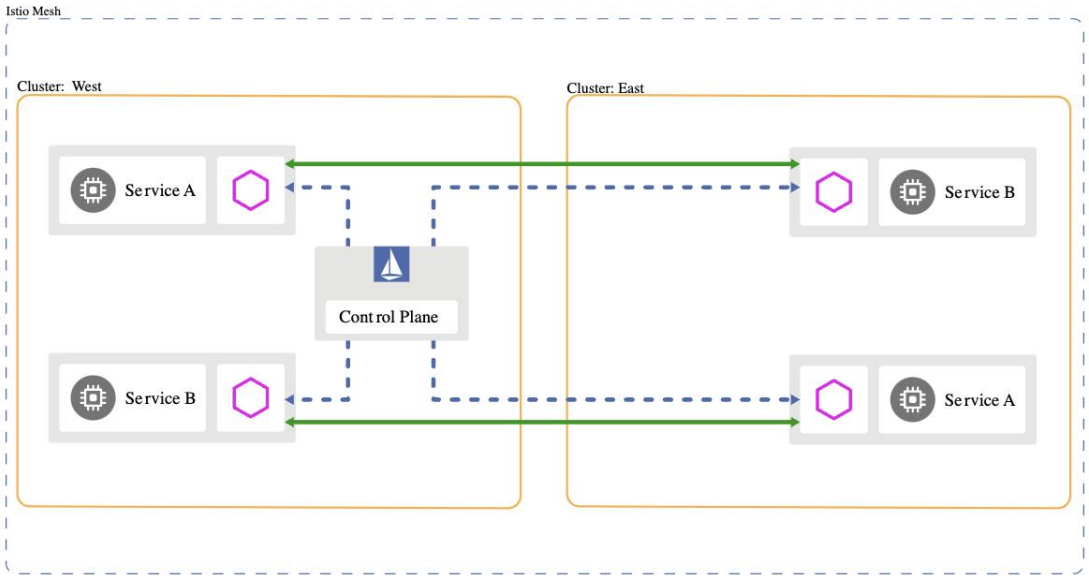
在最简单的情况下，可以在单一集群上使用控制平面运行网格。



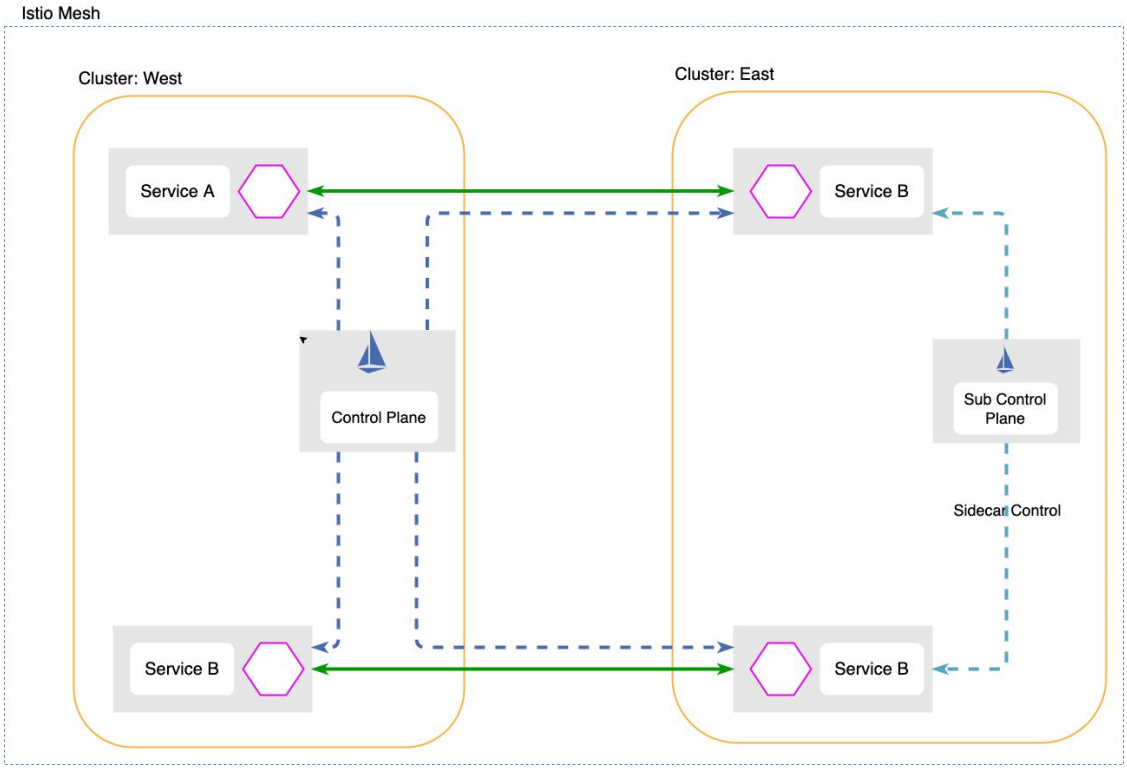
控制平面运行网格

多集群单控制面

多集群部署也可以共享控制平面实例。 在这种情况下，在核心控制集群部署核心的共享控制面实例，其网格核心的通讯策略都由 **共享控制面实例** 控制。 在从集群中，其实也存在一个从控制面实例，该实例其核心能力是 **控制本集群的边车生命周期管理** 。



共享控制面实例



控制本集群边车

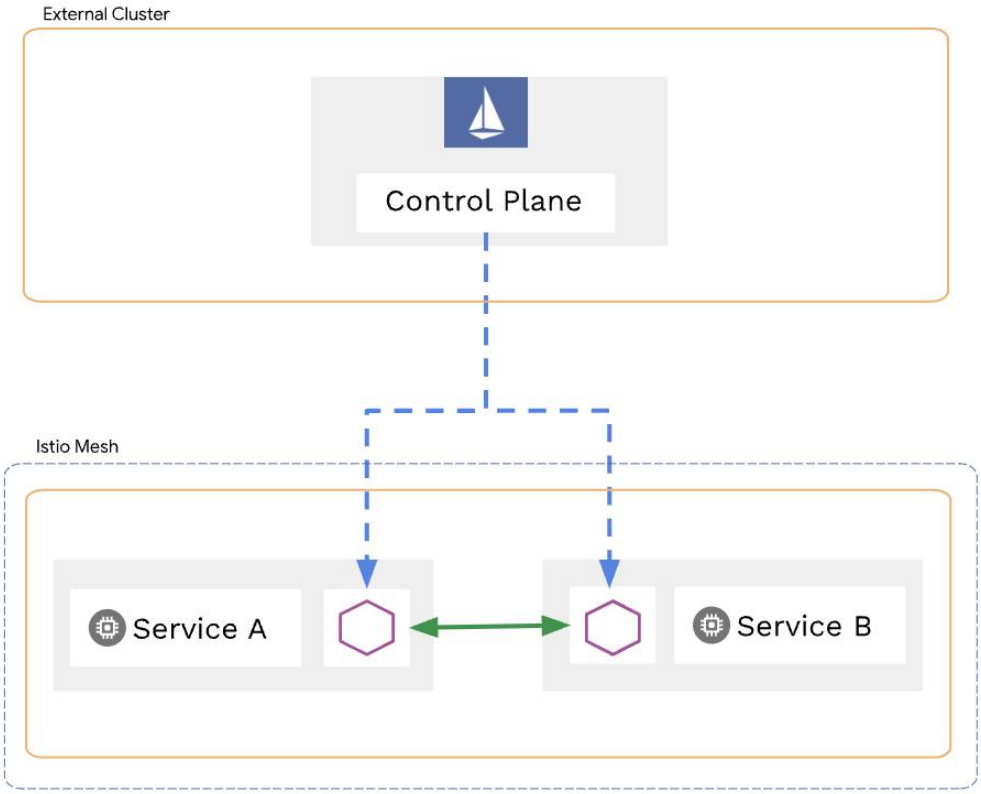
思考一个问题，从集群的边车如何与共享控制面建立连接？其回答时共享控制面需要暴露自身控制面服务。

在单一网络模型中主集群的共享控制平面可以通过稳定 IP(例如集群 IP)访问。

但是在多网络模型的多集群中，需要通过 Mesh 网关对外 **暴露** 共享控制平面。

在实际的生产环境，可以根据实际的云供应商或者内部网络的规划灵活选择； 例如内部负载平衡器，避免将控制面暴露到公共网络中，因为其存在潜在的安全隐患。

共享控制面在上面描述中都是属于某一个网格集群中，其实其远程共享控制面也可以部署在网格外的集群中， 这样能够让网格的控制面与数据面在物理上实现隔离，避免主集群的控制面与数据面同时出现问题，可以分散风险。



远程共享部署

多集群多控制面

为了获得高可用性，您可以在多个集群、区或地域之间部署多个控制平面实例。

在具有多个共享控制面的多集群场景下，每个控制面都属于某个集群， 共享控制面接受用户定义的配置（例如 Service、ServiceEntry、DestinationRule 等）来则自身集群的 Kubernetes API Server。 因此每个主控制面集群都拥有独立的配置来源。

因此这里存在一个问题，多个控制面集群如何同步配置？ 这是一个很复杂的问题，需要增加另外的同步操作。 在大型的生产环境可能需要配合 CI/CD 工具一起自动化该过程，实现配置同步。

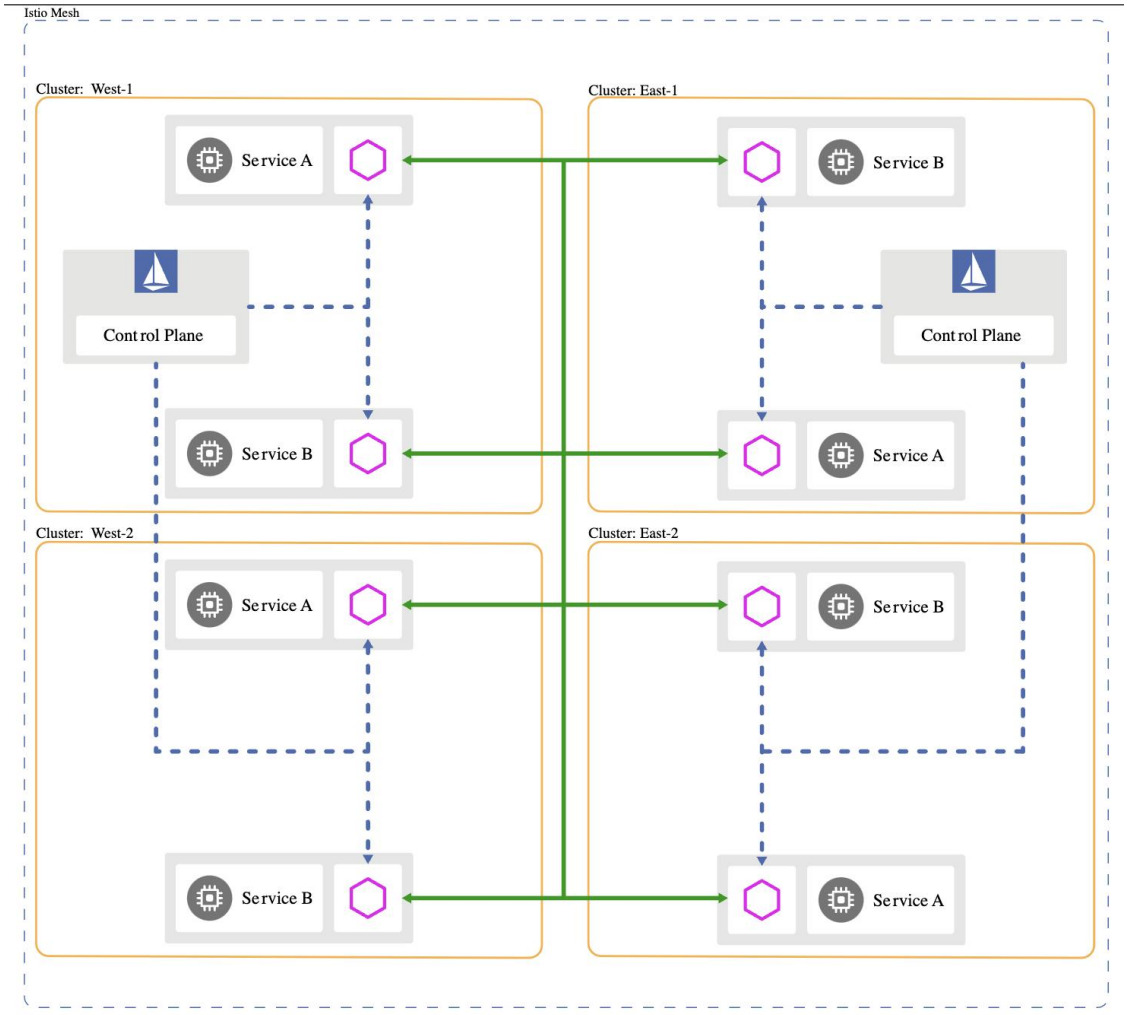


image-20221209083658951

在这种多控制面模型下的在部署难度与配置复杂度上需要付出很大代价。 但是也能收获更高的回报，其核心优势在于：

- **高可用** ： 如果控制平面不可用，故障范围仅限于由该控制平面管理的群集中的工作负载。
- **配置隔离** ： 您可以在一个群集，区域或地区中进行配置更改，而不会影响其他群集。
- **控制面分区域滚动发布** ： 能够在不同的控制面区域内，对网格控制面进行滚动发布，包括金丝雀发布网格控制面，其影响范围也只是该控制面所在的区域。

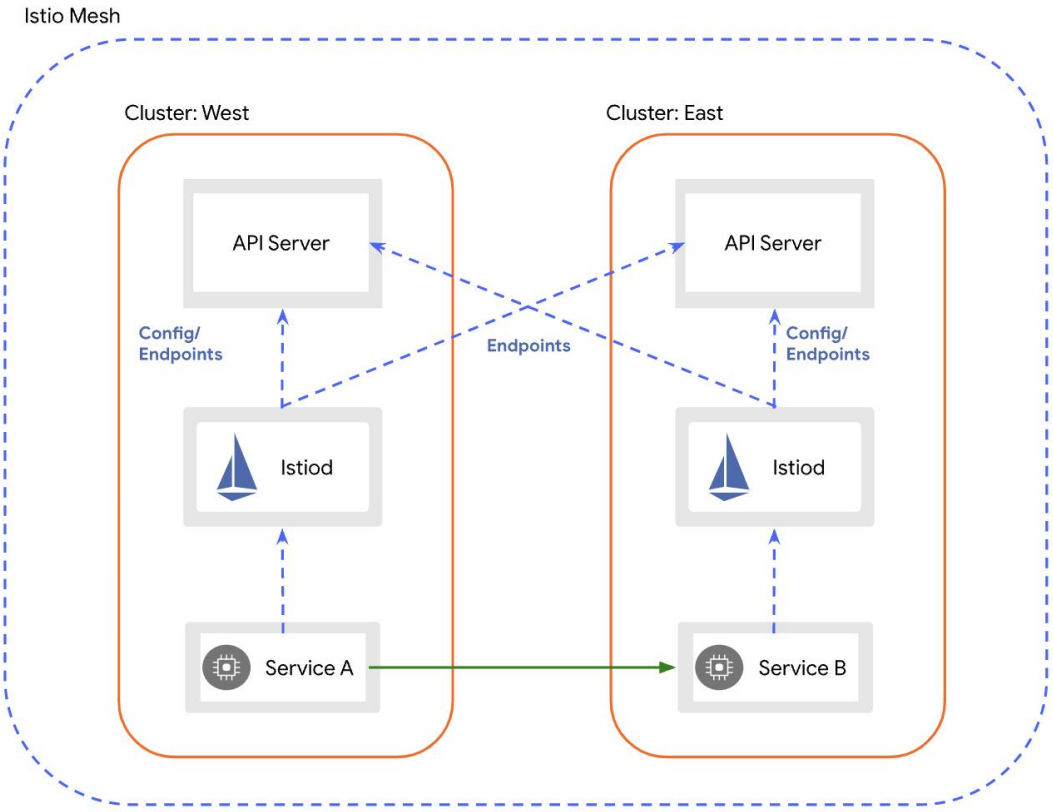
多云环境下的服务发现

了解了服务网格控制面架构以后，这时候我们不得不提到服务网格数据面的通讯与运行原理， 理解这些原理能提高我们在实际运行服务网格遇到问题的解决能力。其中服务网格数据面通讯里面最为核心的技术便是服务发现能力。

在单集群模式下，服务网格控制面会从集群注册中心获取所有注册在集群中的服务，然后将集群注册中心获取到的服务信息聚合成一个服务端点列表，向每个边车提供。

在多群集场景下的服务发现能力更加复杂，因为集群之间存在注册中心隔离，网格为了运行多集群采用的解决方案是服务网格控制平面会观察并且记录每个集群中的注册中心的服务， 然后将集群中的 **同命名空间下** 的 **同名服务** 进行聚合，最后梳理成一张完整的 **网格服务端点列表** ，然后下发给网格内的边车。因此服务才能具有发现其他集群服务端点的能力。

因此网格控制面为了能够访问其控制面的多集群的注册中心，在部署阶段需要从集群 **授权远程密钥** 给所属的控制面集群。 授权以后控制面才能对多集群进行服务发现，因此才具有后续的跨集群负载均衡能力。



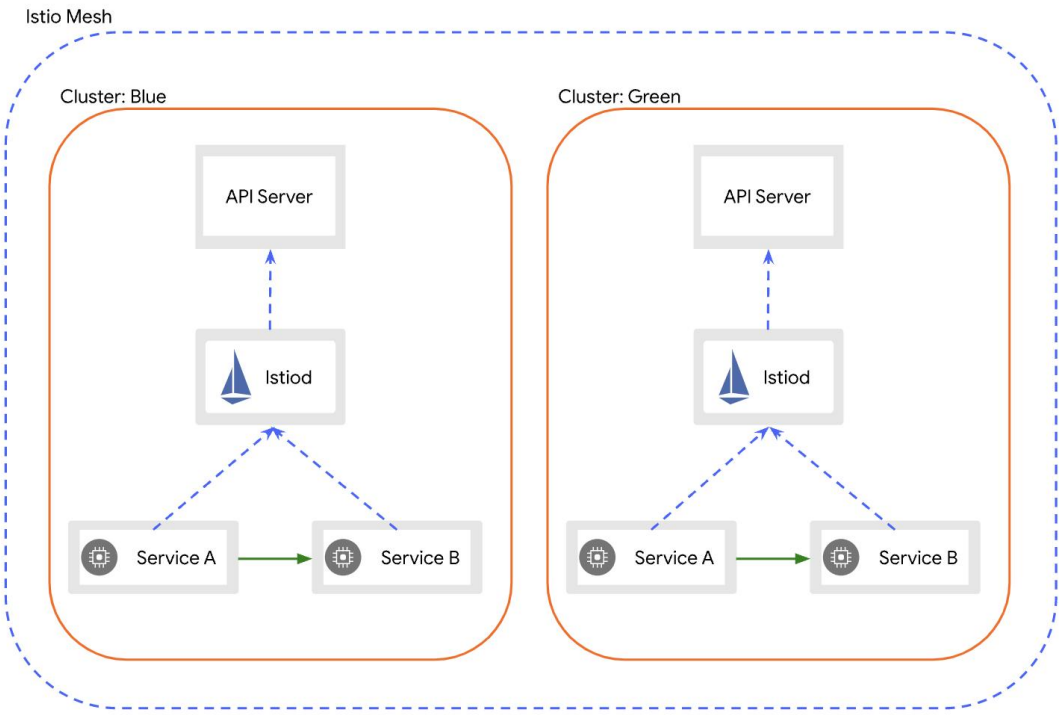
授权远程密钥

在多集群网格部署模型下，多集群服务默认的策略是：每个集群均衡负载。 但是在复杂且庞大的生产环境中，其实很多服务只需要在某些区域进行流量通讯，这时候就可以采用本地优先负载均衡策略（具体方式为参考 Istio 官方提到的 Locality Load Balancing）。

在某些场景下我们会发现，跨集群的流量负载能力并不是频繁操作，时时刻刻的跨集群负载也不是我们所期望的。 例如说我们只是需要进行蓝绿部署，其版本位于不同的集群中； 例如当某个集群的某个服务出现故障时，能够切换到备用集群的服务中。 这些场景都是偶发的。发生频率很低，但是又是刚需。

这时候我们怎么样才能通过网格实现呢？在 Istio 服务网格下有两种方式：

1. 我们可以不交换集群的 API Server 远程密钥，这样的话集群只能进行自身集群内的服务发现。 如果需要进行跨集群流量负载，可以通过 ServiceEntry 配合外部负载器的方式实现。



多副本恢复

2. 通过配置 VirtualService 与 DestinationRule 策略，禁用多集群之间的流量负载。

服务网格技术方案

上述几个章节我们简单围绕服务网格三个核心维度（网络模型、控制面架构、数据面服务发现）分析其相关原理与优势。 但是这时候我们会发现在网络模型和控制面架构部分介绍时都存在多种方案的情形，那么我们如何选择具体那种方案呢?

DaoCloud 深耕在服务网格产品多年的 ，拥有实际落地客户，以及从单集群演变到多集群客户解决方案的经验，我们产品内部也经过了多种方案的验证与实践，采

集多位客户的场景，最后将我们的多集群服务网格产品的架构选型方案如下：自由的网络模型 + 多集群单控制面。

网络模型

在选择网格模型落地时，其实并不是非 A 即 B。 比如说用户需要更高效果，其打通部分集群之间网络成本在用户容忍范围内，用户可以选择部分集群网络为单一网络模式，在无法打通集群网络的集群中选择多网络模式。

每种模型都有自己独特的优势与代价，那么我能不能同时拥有呢？ 这个答案时肯定的，我们都要。所以我们提出一个全能网络模型的概念，用户可以根据自己的需要选择网络模型：

- 单一网络
- 多网络
- 单一网络 + 多网络

这样丰富的网络模型方案，让我们能够更加贴合用户需求解决实际问题。

多集群单控制面

到这一部分，很多人会提问为什么选择多集群单控制面模式呢？从上述文章看起来多集群多控制面提供的能力不是更加丰富吗？

这里不得不提到实现多集群多控制面在落地的时候存在一个非常痛苦的点：多控制面从部署到运用到维护的成本非常高。 我们怎么得出这个结论的呢？在早期我们内部选择了多集群多控制面的验证，但是在这个过程中发现多控制面存在一些弊端：

- 策略复杂高隐患：
    - 配置复杂：不同的控制面需要不同的策略，虽然换个角度看是独立的控制面策略，但是在多集群场景下，集群肯定是简单的一个两个，应用的总数也会随着集群的扩大变得更加复杂，非常容易发生部分集群忘记配置，或者更改配置时需要重复配置多套相同的策略。

- 策略冲突：多套控制面的策略非常多，用户需要准确掌握每个策略，不然很容易造成策略冲突。

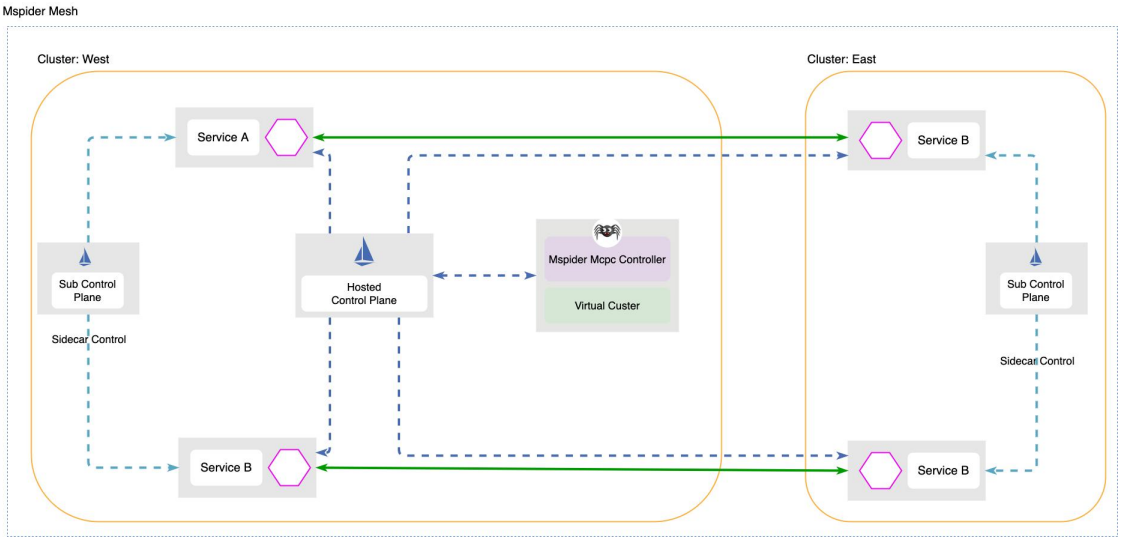
- 控制面部署与升级维护成本高：当存在多套控制面时，我们需要维护多套控制面的部署与升级。

- 污染多个控制面集群：由于网格需要聚会多集群命名空间与服务，会污染控制面集群。让我举例说明：当集群 A 拥有命名空间 N1、N2；集群 B 拥有命名空间 N3 时；且集群 A、B 都控制面集群时，集群 A 需要新增 N3，集群 B 需要新增 N1、N2。当多集群与集群中业务到达一定体量时，这是非常恐怖的。

但是在多集群单控制面模式下能够大部分避免上面存在的问题，尤其是其中治理策略复杂这块，所有的策略将会统一在一个集群中，因此其中配置难度与管理难度将大大降低。

服务网格架构

接下来让我们来了解一下服务网格的实际架构：



托管网格

在原生的多集群单控制面下其实并不能解决上文提到的污染多个控制面集群问题，但是我们仔细看上面的架构，大家会发现在这部分架构与社区提供的方案有一个很

大的不同，在控制面集群中多了一套服务网格组件，这一套组件的最大的特色是提供了一个虚拟集群，该集群与网格控制面连接，并且目的是将网格资源与控制面集群的资源进行隔离，避免产生脏资源，这时候就能完美解决治理资源污染集群的问题，并且避免用户误操作删除控制面策略资源。

总结一下服务网格技术选型方案的的优势：

- **更简单的管理** ： 将整个服务网格的管理集中在一个控制平面中，可以帮助组织更方便、更快速地管理整个服务网格。
- **低复杂度的配置与更好的性能** ： 通过将整个服务网格的所有数据都统一在一个控制平面中，可以更快速地处理服务的路由和治理策略，从而提高整个服务网格的性能。
- **更强的安全性** ：在单控制面多集群模式中，我们可以更好地控制对整个服务网格的访问，并使用更严格的安全策略来保护其数据。
- **支持跨集群负载与容灾能力** ：能够灵活的配置跨集群流量负载与容灾策略。
- **策略配置统一高效** ：不需要配置复杂的控制面配置同步与合并问题，尤其是控制面越多，其业务服务的体量越大，其同步和合并一定会是一个非常复杂且存在风险的存在。
- **灵活的控制面与数据面隔离** ：服务网格允许用户将网格控制面与数据面集群进行隔离，并且在用户资源有限的情况下也支持集群同时兼具控制面与数据面在同一集群下。
- **网络模式灵活切换** ：服务网格默认场景是多集群位于单网络，但是多网络模式也能够支持部署。
- **网格资源隔离** ：在开源模式下，控制面集群需要管理所有集群的网格策略，这时候会存在污染控制面集群的命名空间，或者部分网格资源会受到集群本身操作影响（例如某个命名空间被删除）。

参考

本文部分图片来自 Istio 官网与 DaoCloud 文档网站。

参阅 Istio 部署模型。

多集群灰度发布

多集群灰度发布的实现逻辑是，通过在不同集群部署不同的业务应用，然后通过服务网格配置相应的策略实现业务版本之间流量调整，然后根据运行情况下线版本。

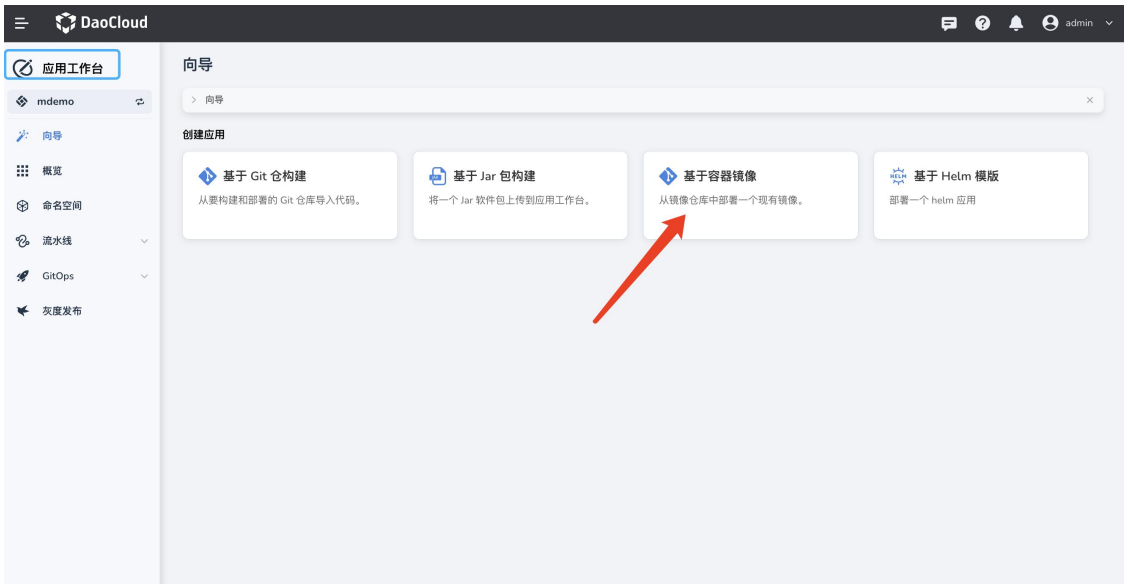
前置准备，参考网格多云部署文档，搭建网格基础设施。

创建 demo 应用并且验证流量

集群部署 v1 版本 helloworld

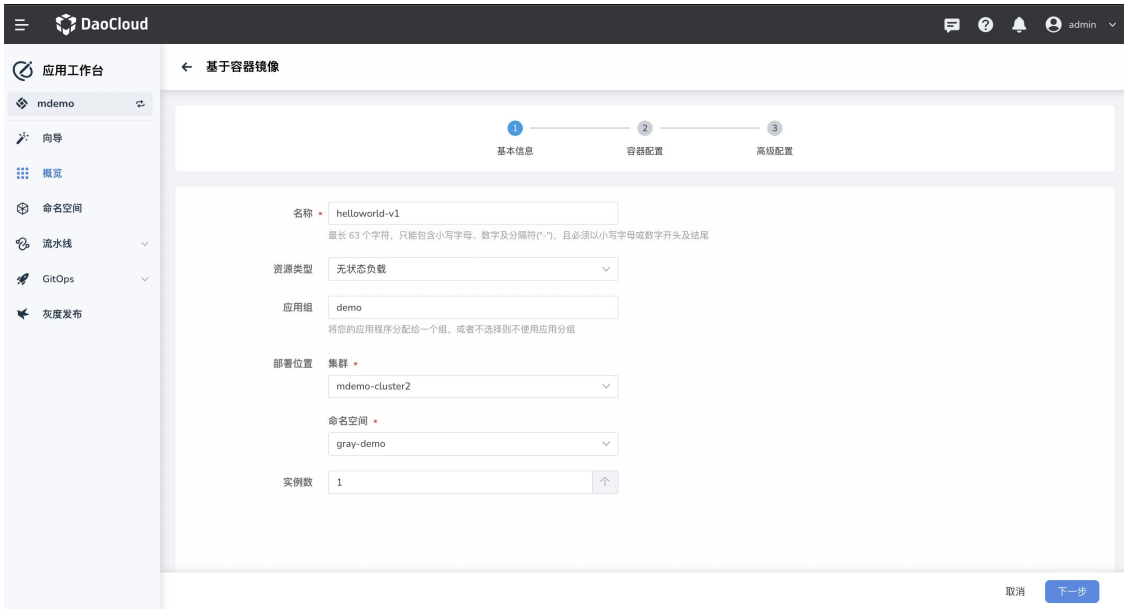
首先选择一个命名空间（gray-demo），并且在网格页面开启该命名空间边车注入。

在应用工作台部署应用，在这里我们以 istio 的 helloworld 为例。



部署应用

选择对应集群（mdemo-cluster2）与命名空间，并且配置工作负载基本信息。

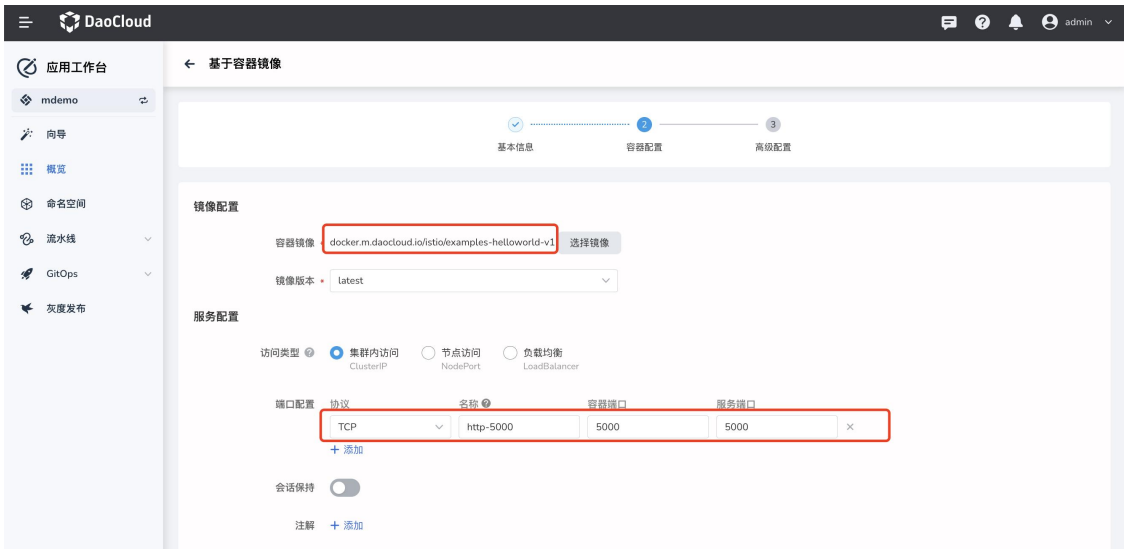


配置工作负载

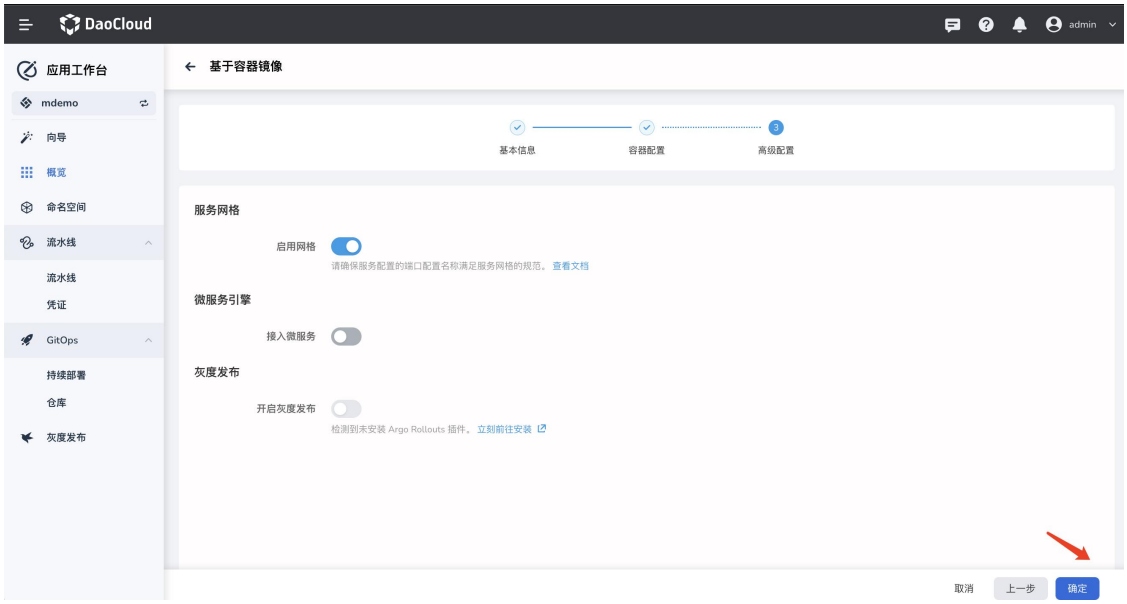
- 选择镜像：docker.m.daocloud.io/istio/examples-helloworld-v1
- 版本：latest

配置服务信息：

- 访问类型：集群内访问
- 端口配置
  - 协议：TCP
  - 名称：http-5000
  - 容器端口：5000
  - 服务端口：5000

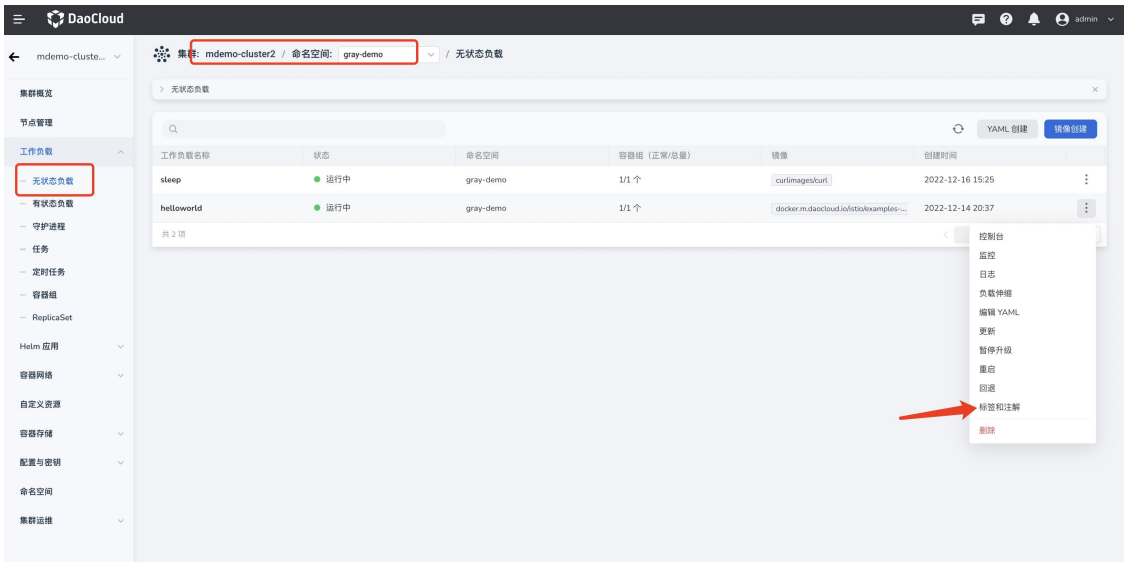


访问类型

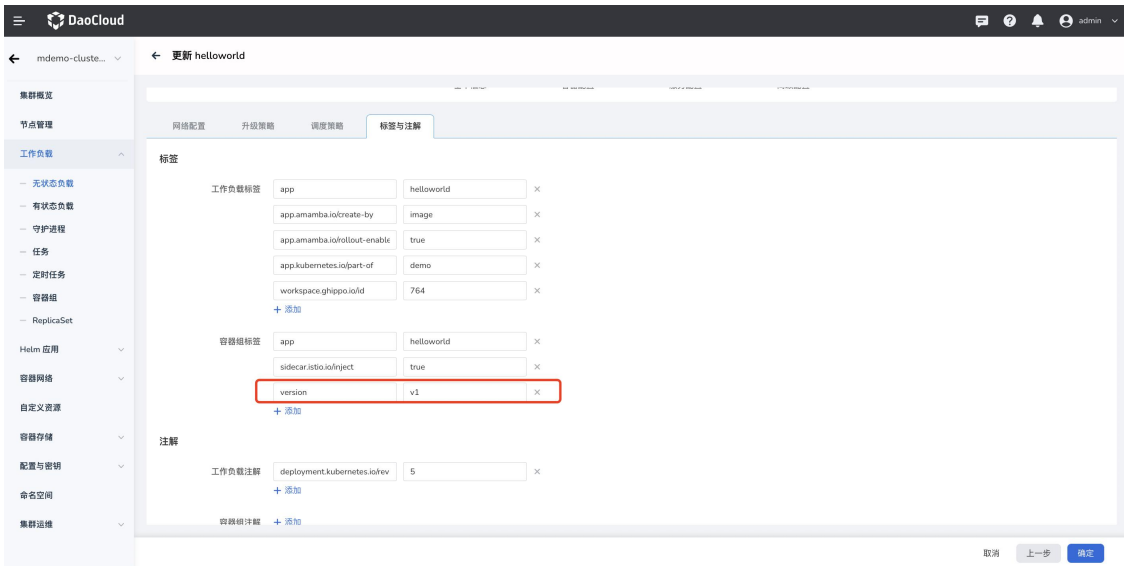


端口配置

为了区分不同版本的工作负载，需要在容器管理平台找到对应的工作负载，点击标签与注解，给容器组标签添加键值对： “version”：“v1”



标签与注解



添加键值对

集群部署 v2 版本 helloworld

注意服务命名与命名空间必须相同。

首先选择上面一致命名空间（gray-demo），并且在网格页面开启该命名空间边车注入。

部署流程与上面一样，其主要区别是：

- 镜像发生了变化：docker.m.daocloud.io/istio/examples-helloworld-v2
- 在容器平台给相应 **容器组标签** 加上 label “version”：“v2”

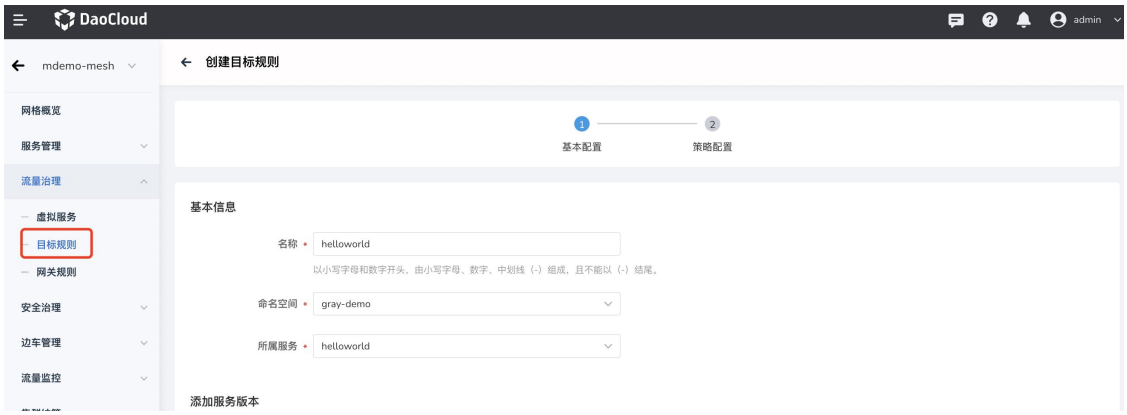
部署灰度应用策略

多集群目标规则

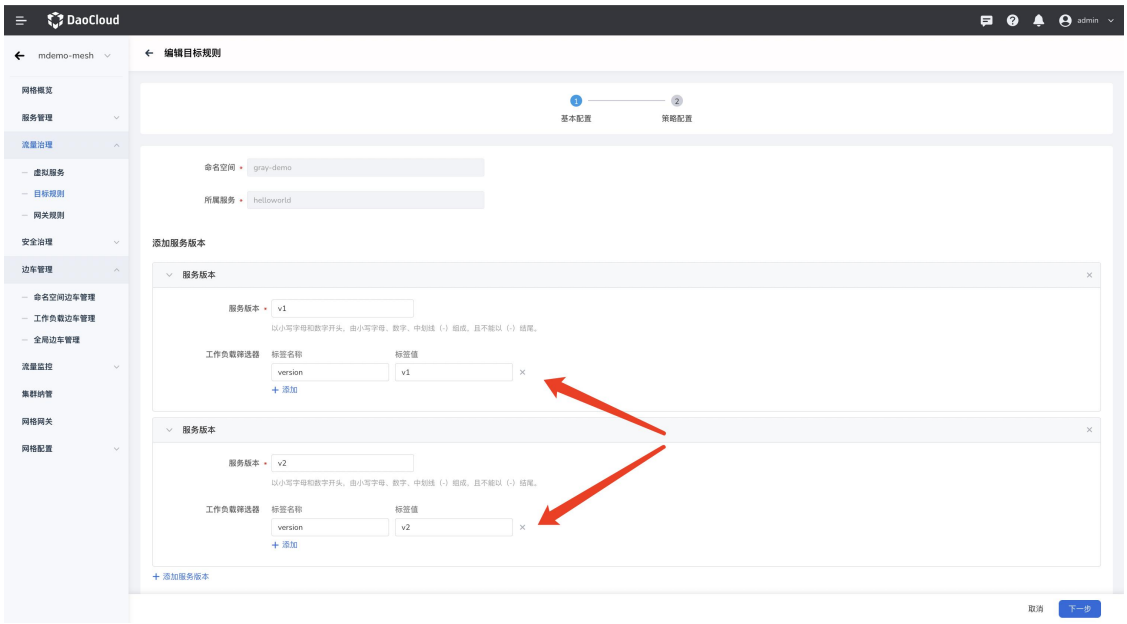
首先创建 DestinationRule，通过定义 SubSet 定义不同集群的业务版本。

其标签键值对为上文中添加的容器组标签：version: <VERSION>。

策略：必须开启 Istio 双向 TLS

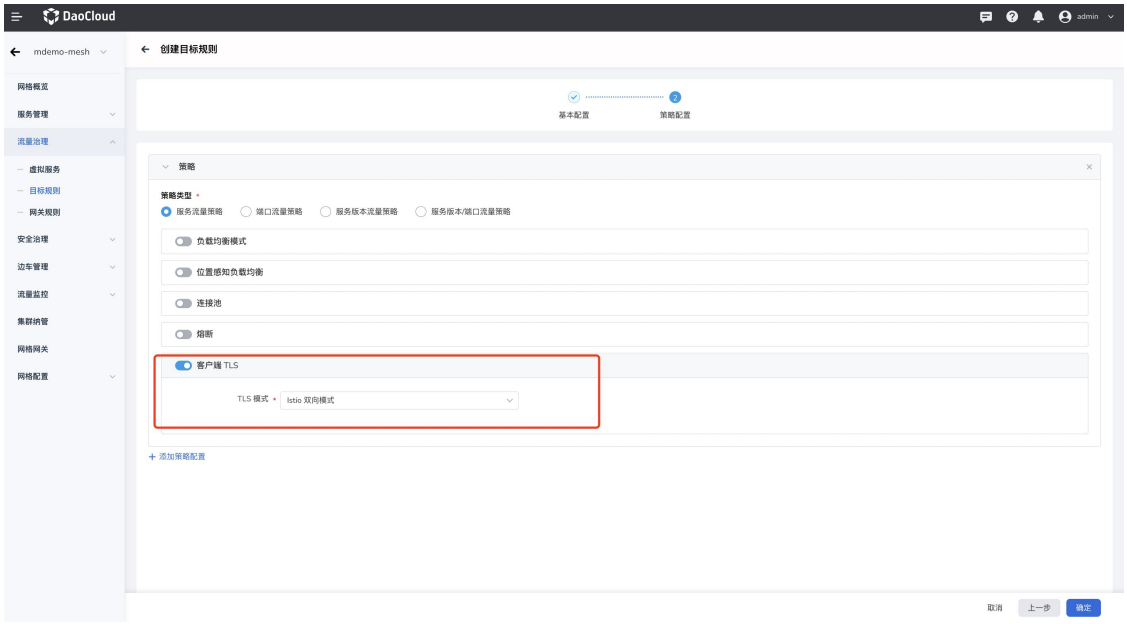


开启双向 TLS



开启双向 TLS

必须开启 **Istio 双向** TLS 模式



开启双向 TLS

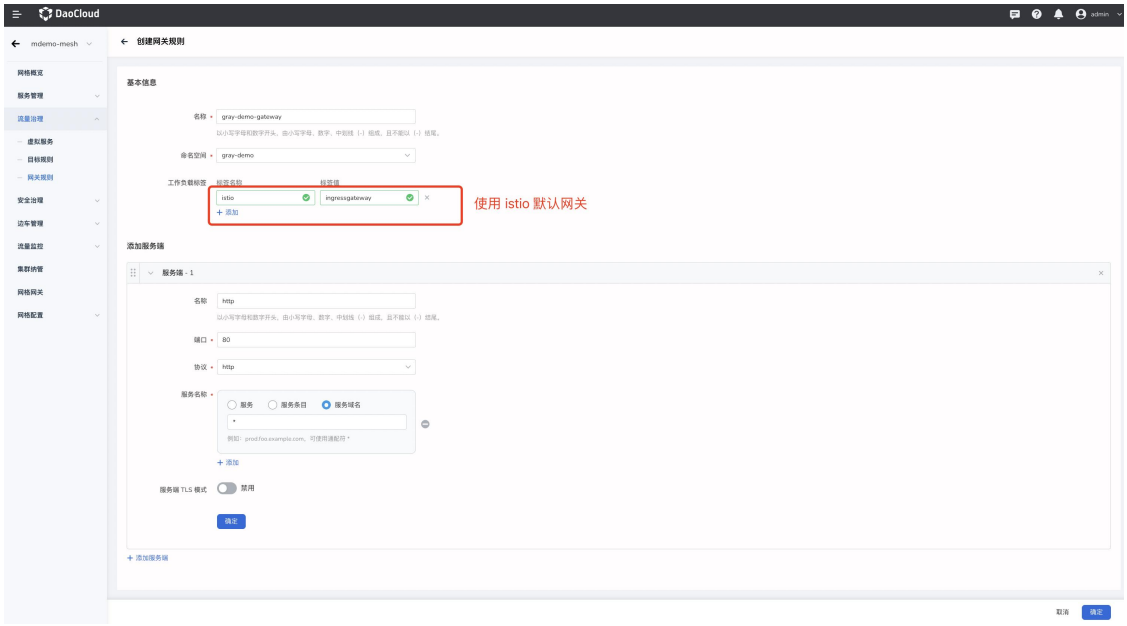
目标规则 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
```

```
metadata:
  name: helloworld
  namespace: gray-demo
spec:
  host: helloworld
  subsets:
    - labels:
        version: v1
      name: v1
      trafficPolicy:
        tls:
          mode: ISTIO_MUTUAL
    - labels:
        version: v2
      name: v2
      trafficPolicy:
        tls:
          mode: ISTIO_MUTUAL
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

通过 ingress 暴露服务

首先需要创建一条网关规则：

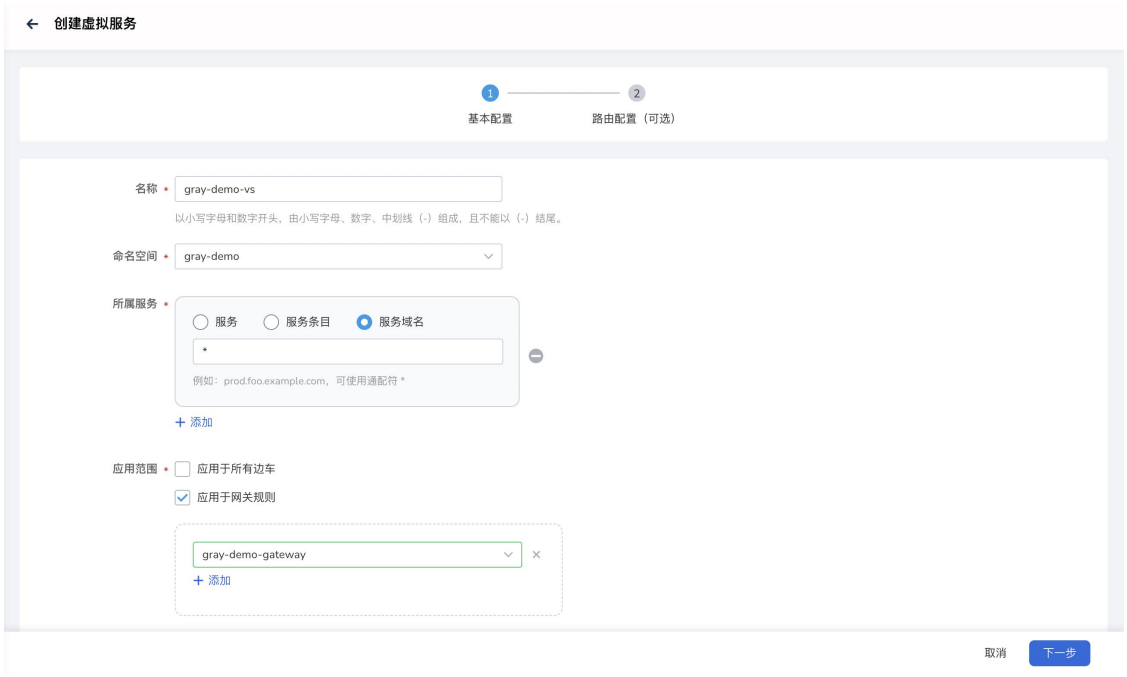


创建一条网关规则

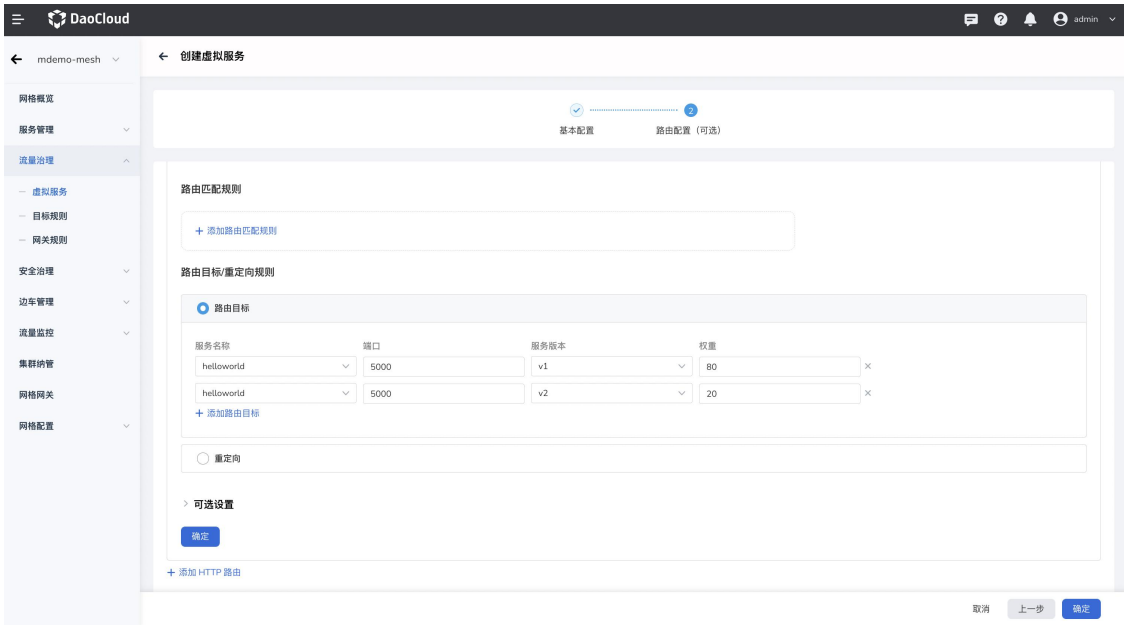
网关规则 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: gray-demo-gateway
  namespace: gray-demo
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - "*"
```

然后配置访问服务所需的虚拟服务规则。



配置虚拟服务规则



配置虚拟服务规则

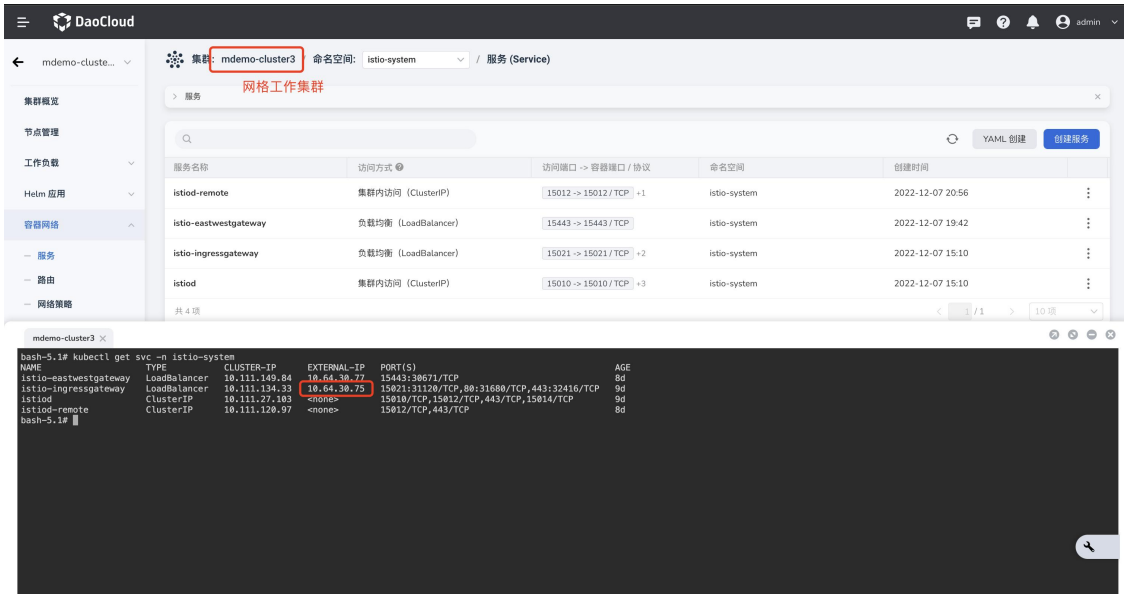
虚拟服务 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
```

```
metadata:
  name: gray-demo-vs
  namespace: gray-demo
spec:
  gateways:
    - gray-demo-gateway
  hosts:
    - "*"
  http:
    - name: http-5000
      route:
        - destination:
            host: helloworld.gray-demo.svc.cluster.local
            port:
              number: 5000
            subset: v1
          weight: 80
        - destination:
            host: helloworld.gray-demo.svc.cluster.local
            port:
              number: 5000
            subset: v2
          weight: 20
```

验证

INGRESS_LB_IP 是指 Ingress 网格负载均衡地址，可以在容器管理平台中查看，如果没有有效的负载均衡 IP 可以通过 NodePort 方式访问。



验证

（由于容器管理平台界面无法直接查看外部 IP，因此使用控制台的能力查看）

在浏览器中访问： http://${INGRESS_LB_IP}/hello

确认其 v1 与 v2 版本的访问比例是否与上面策略的比例为 8:2



确认

网格下多云网络流量分流配置

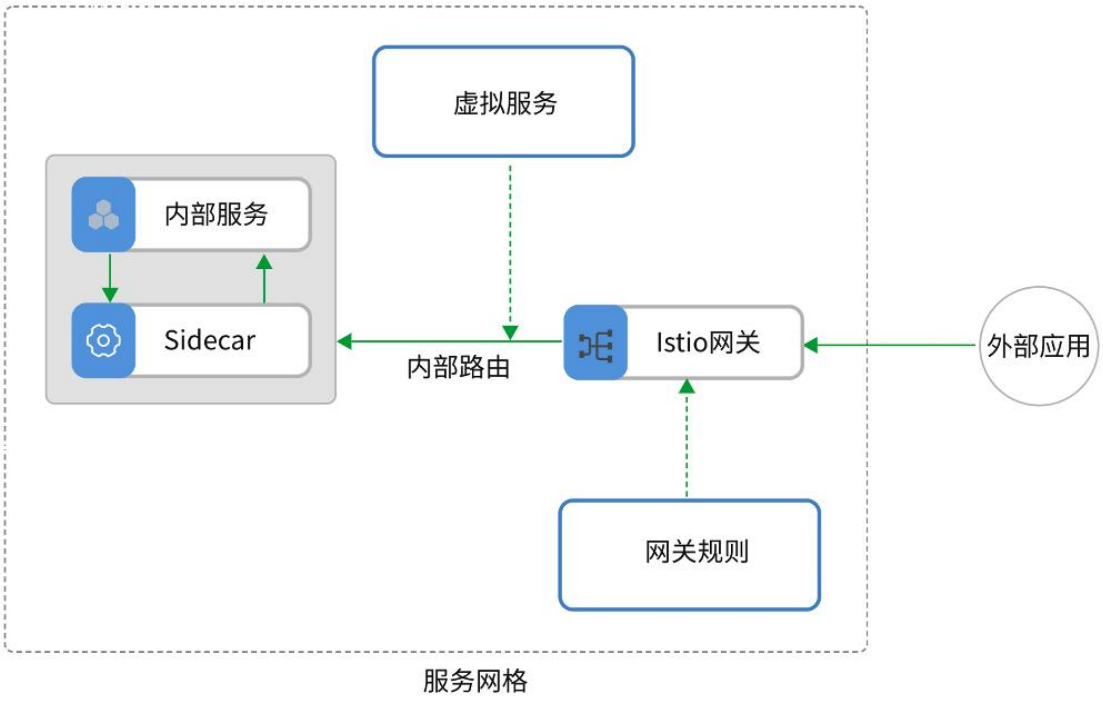
本文讲述如何在多云网络中为工作负载配置不同的流量。

前置条件：

- 服务 helloworld 运行于网格 hosted-mesh 的命名空间 helloworld 下
- 开启多云网络互联
- 网格提供 ingressgatway 网关实例

配置步骤：

1. 基于所属集群，对请求流量做权重分流；



流程

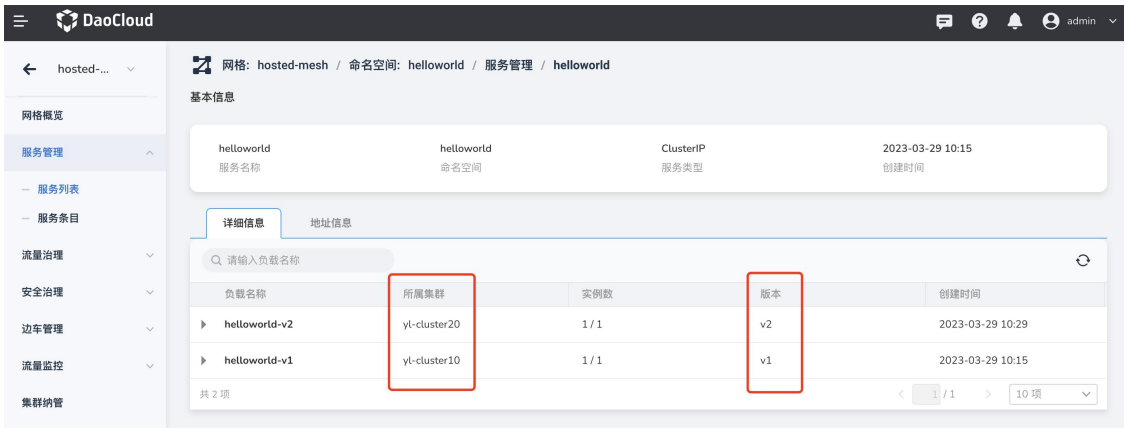
2. 为两个集群的 helloworld 工作负载分别添加标签：

   所属集群　　标签　　值

所属集群　　标签　　值

yl-cluster10　version　v1

yl-cluster20　version　v2



添加标签

3. 在左侧导航栏点击边车管理 -> 工作负载边车管理，为两个集群的 helloworld 工作负载注入边车。

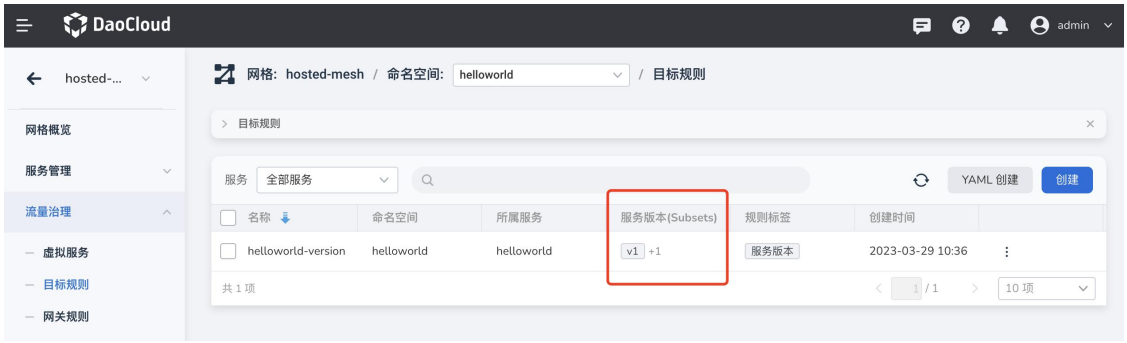


集群 01



集群 02

4. 在左侧导航栏点击流量治理 –> 目标规则 –> 创建，创建两个服务版本。



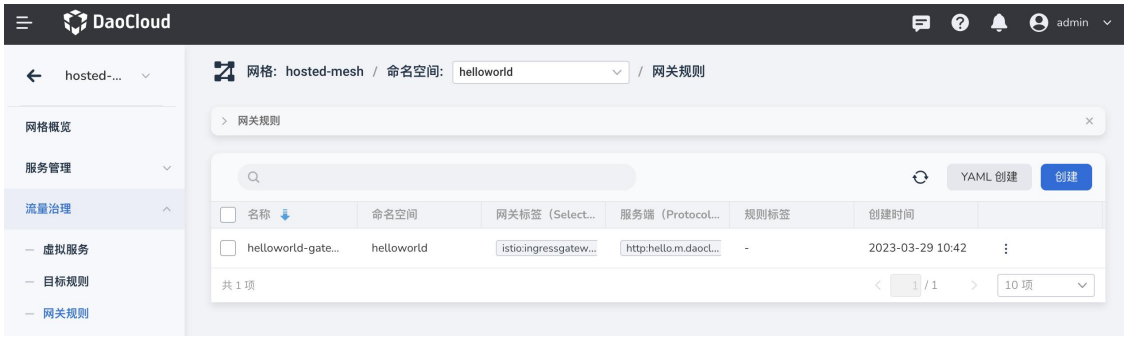
服务版本

对应的 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld-version
  namespace: helloworld
spec:
  host: helloworld
  subsets:
    - labels:
        version: v1
        name: v1
```

```
    - labels:
        version: v2
      name: v2
```

5. 在左侧导航栏点击流量治理 -> 网关规则 -> 创建，创建网关规则。

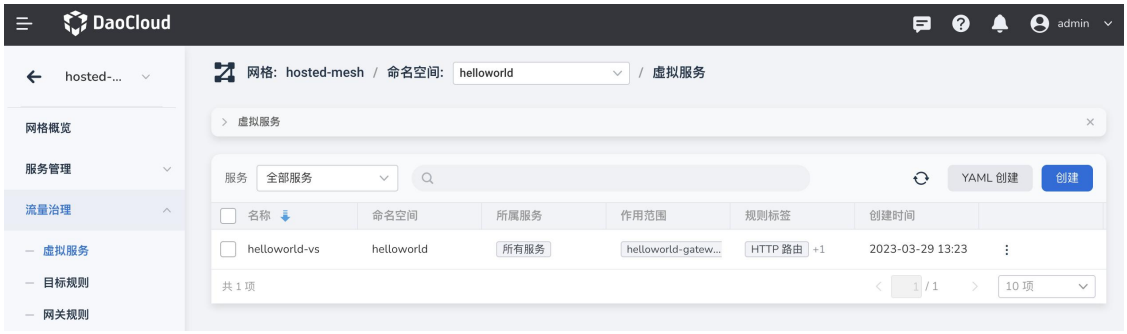


网关规则

对应的 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: helloworld-gateway
  namespace: helloworld
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - hello.m.daocloud
      port:
        name: http
        number: 80
        protocol: http
```

6. 在左侧导航栏点击流量治理 -> 虚拟服务 -> 创建，创建路由规则，基于权重比例把流量分流至两个集群：



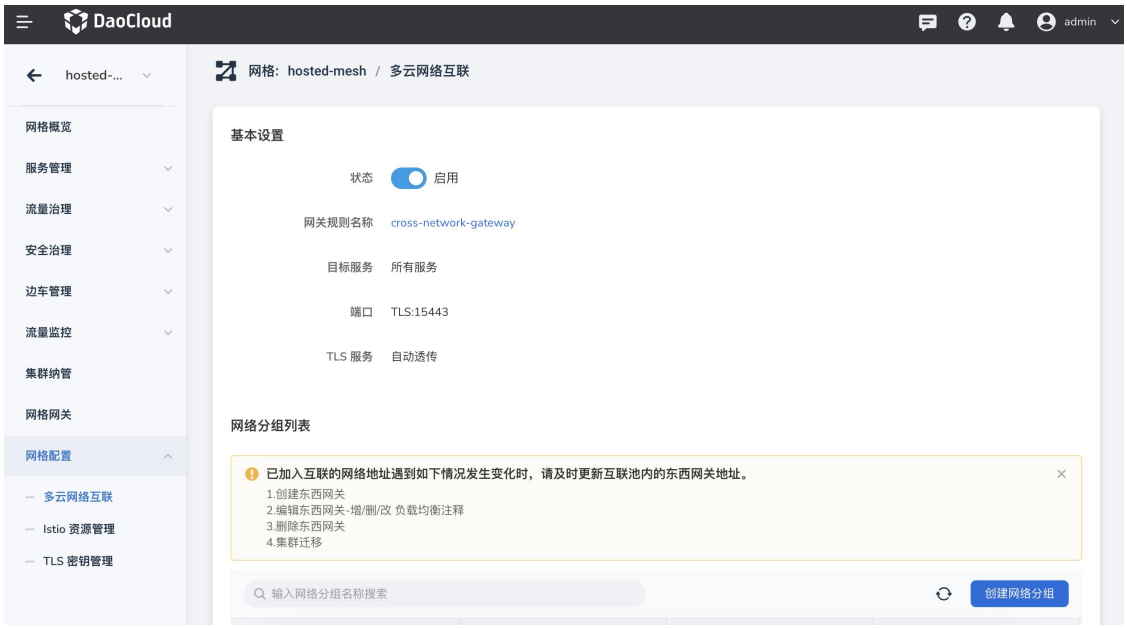
虚拟服务

对应的 YAML 如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: helloworld-version
  namespace: helloworld
spec:
  gateways:
    - helloworld-gateway
  hosts:
    - helloworld.helloworld.svc.cluster.local
  http:
    - match:
      - uri:
          exact: /hello
      name: helloworld-routes
      route:
        - destination:
            host: helloworld
            port:
```

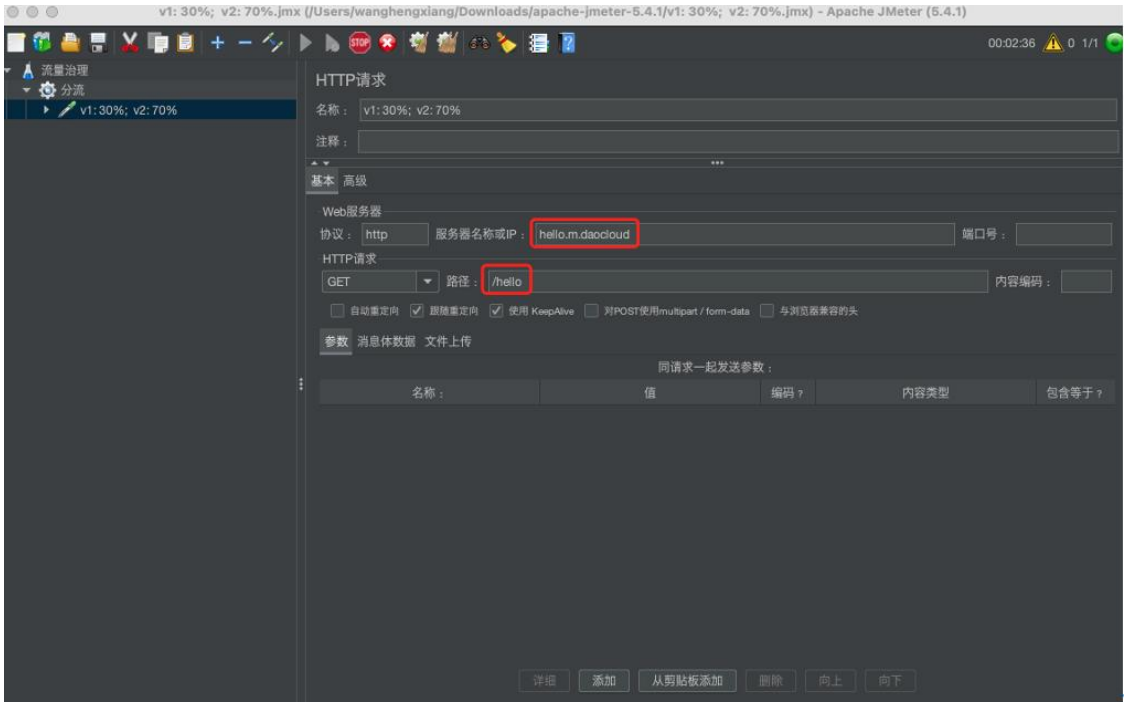
```
      number: 5000
    subset: v1
    weight: 30
  - destination:
    host: helloworld
    port:
      number: 5000
    subset: v2
    weight: 70
```

7. 在左侧导航栏点击网格配置 -> 多云网络互联，开启多云网络互联。

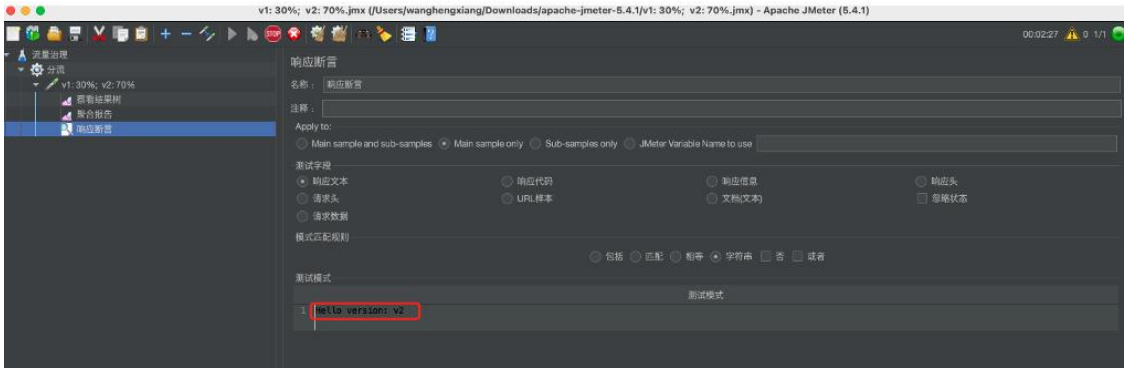


多云网络互联

8. 通过 JMeter 发起 1000 次 get 请求，设置断言

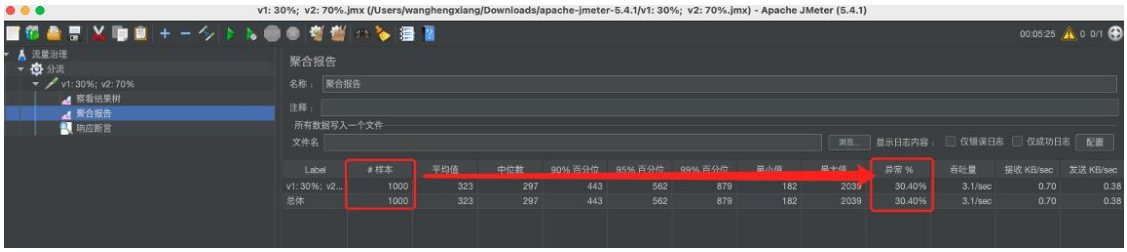


JMeter01



JMeter02

9. 查看请求聚合报告（设置断言 helloworld v2），成功率 70%，异常率比例接近 30%。


v1: 30%; v2: 70%.jmx (/Users/wanghengxiang/Downloads/apache-jmeter-5.4.1/v1: 30%; v2: 70%.jmx) - Apache JMeter (5.4.1)
00:05:25 0 0/1
流量治理
分流
v1: 30%; v2: 70%
察看结果树
聚合报告
响应断言
聚合报告
名称：
聚合报告
注释：
所有数据写入一个文件
文件名
浏览...
显示日志内容：
仅错误日志
仅成功日志
配置
Label
样本
平均值
中位数
90% 百分位
95% 百分位
99% 百分位
最小值
最大值
异常 %
吞吐量
接收 KB/sec
发送 KB/sec
v1: 30%; v2...
1000
323
297
443
562
879
182
2039
30.40%
3.1/sec
0.70
0.38
总体
1000
323
297
443
562
879
182
2039
30.40%
3.1/sec
0.70
0.38

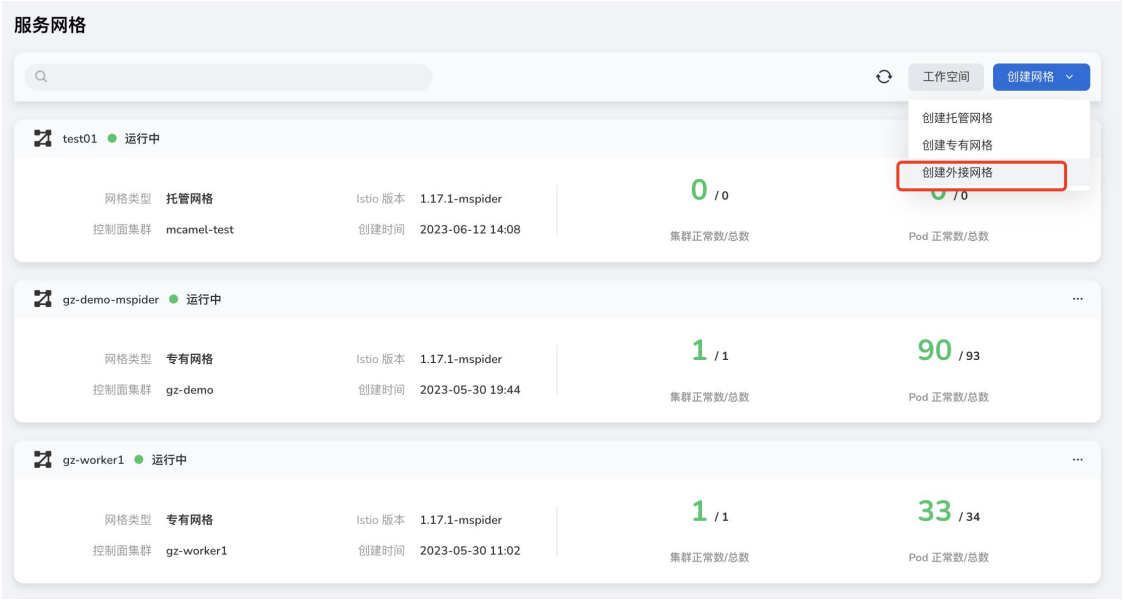
报告

创建外接网格

外接网格指的是可以将企业现成的网格接入到 DCE 5.0 服务网格中进行统一管理。

下文说明创建外接网格的步骤：

1. 在网格列表页面的右上角，点击创建网格按钮，在下拉列表中选择创建外接网格。

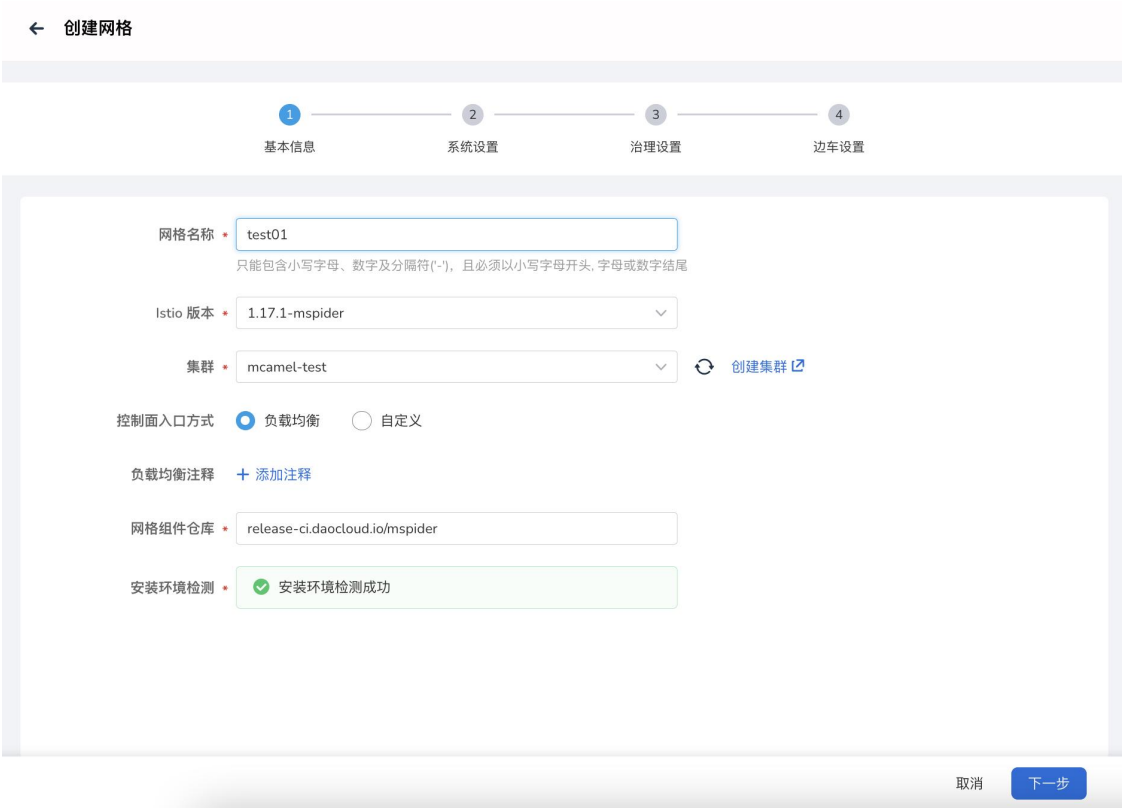


创建外接网格

2. 系统会自动检测安装环境，检测成功后填写以下基本信息，点击确定。

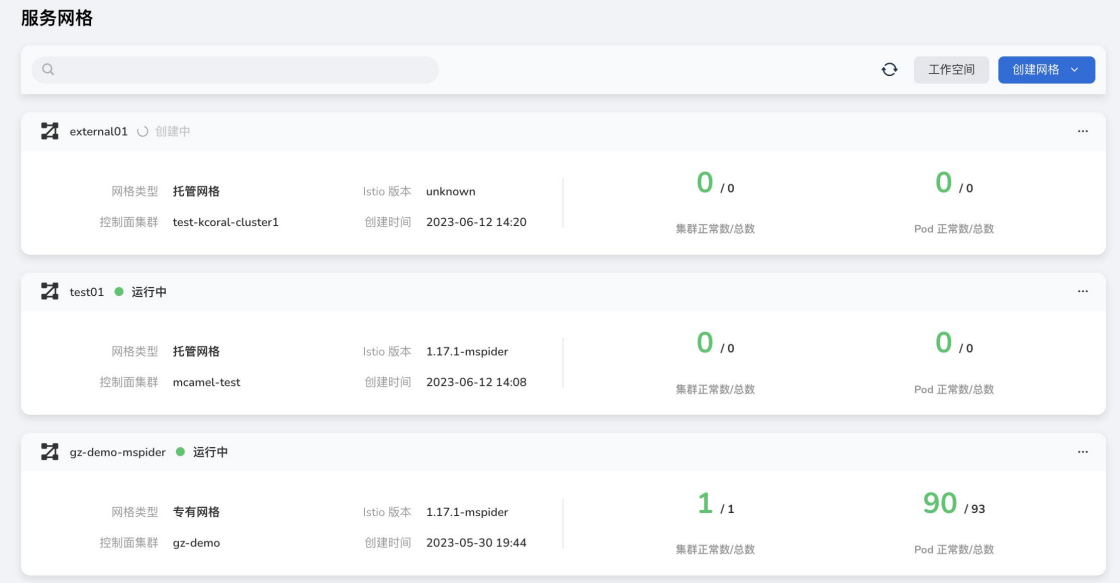
   - 网格名称：以小写字母开头，由小写字母、数字、中划线（-）组成，且不能以中划线（-）结尾
   - Istio 版本：所有被纳管集群的 Istio 都将使用此版本。
   - 集群：这是运行网格控制面的集群，集群下拉列表中展示了每个集群的版本及其健康状态。

- 控制面入口方式：支持负载均衡和自定义。
- 网格组件仓库：输入包含数据面组件镜像的镜像仓库地址，例如 release-ci.daocloud.io/mspider。



基本信息

3. 自动返回网格列表，新创建的网格默认位于第一个，一段时间后状态将从创建中变为运行中。 点击右侧的 ...，可以编辑基本信息、添加集群、进入控制台以及删除等操作。



网格列表

下一步：服务管理

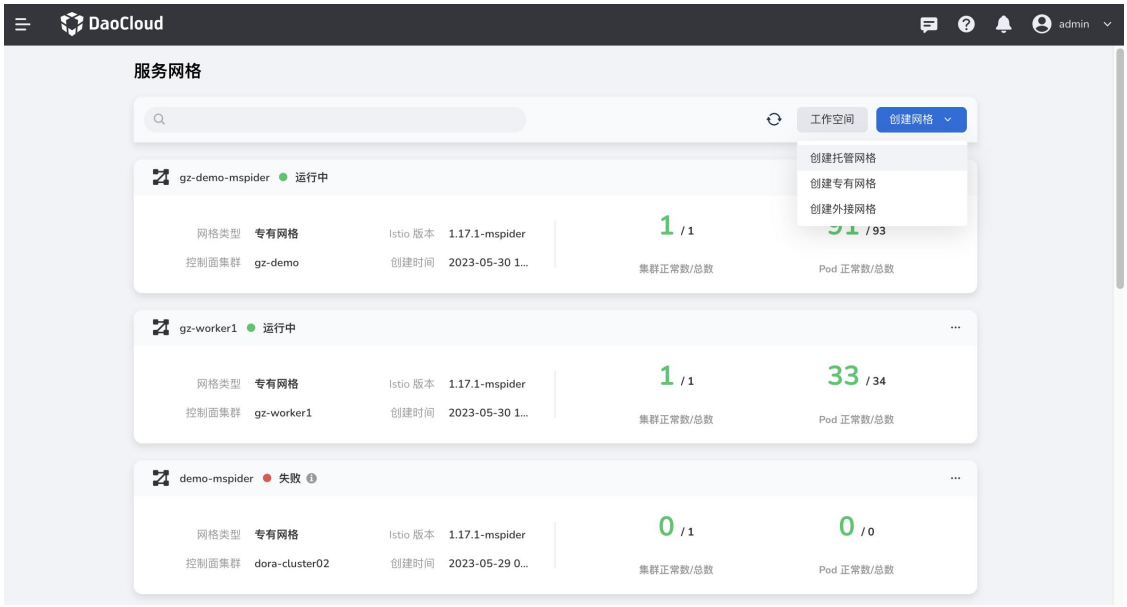
创建托管/专有网格

DCE 5.0 服务网格支持 3 种网格：

- **托管网格** 完全托管在 DCE 5.0 服务网格内。这种网格将核心控制面组件从工作集群中分离，控制面可以部署到一个独立的集群中，能够对同一网格中的多集群服务进行统一治理。
- **专有网格** 采用 Istio 传统结构，仅支持一个集群，集群内设有专门的控制面。
- **外接网格** 指的是可以将企业现成的网格接入到 DCE 5.0 服务网格中进行统一管理。参见创建外接网格。

下文说明创建托管网格/专有网格的步骤：

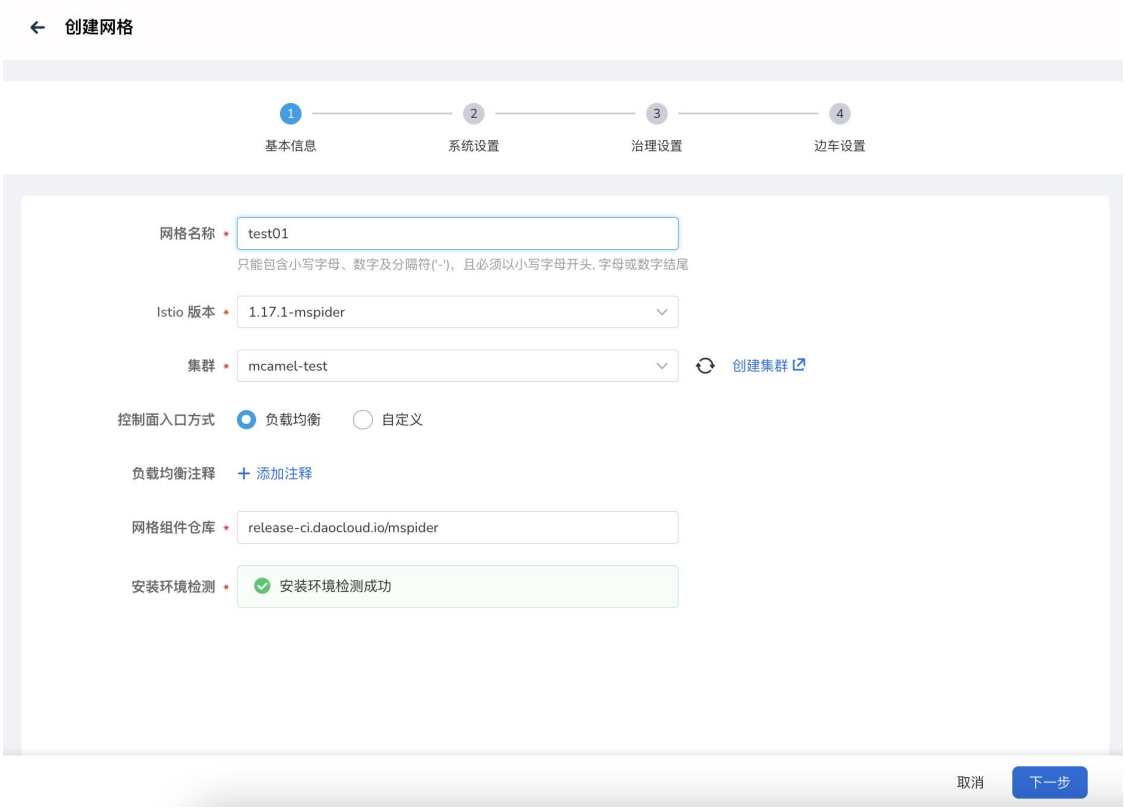
1. 在网格列表页面的右上角，点击创建网格按钮，在下拉列表中选择一种网格。



创建网格

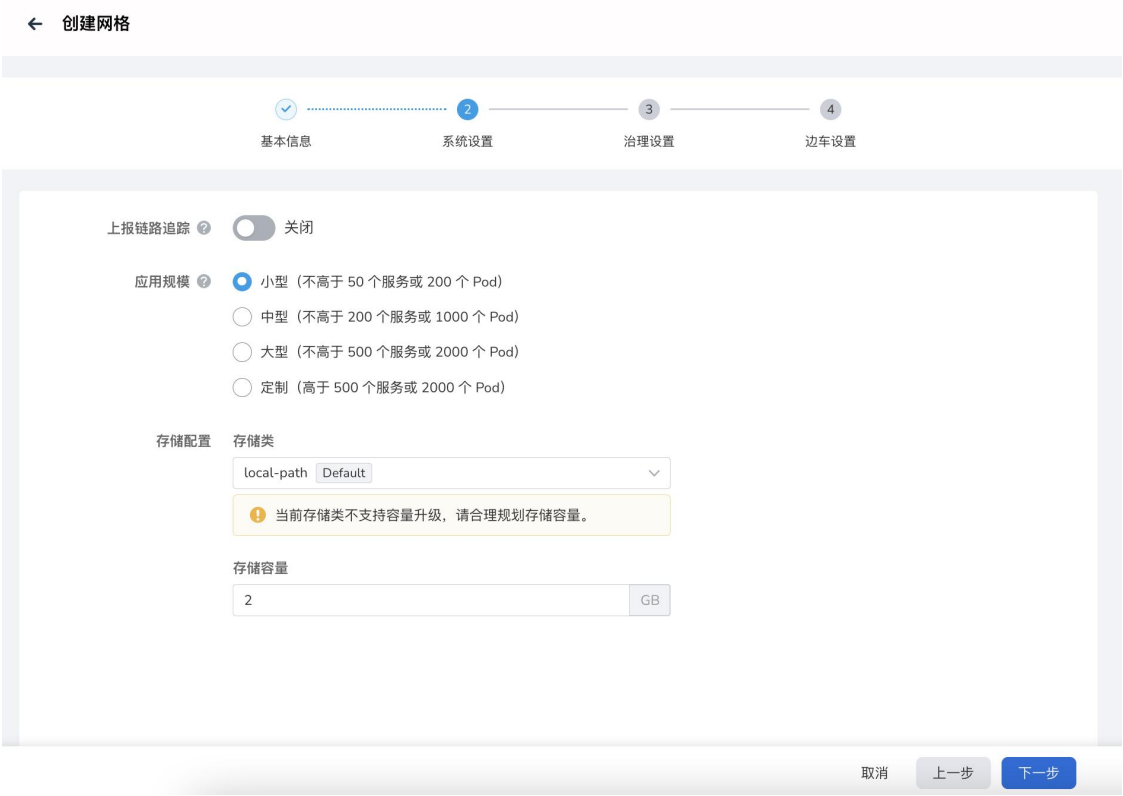
2. 系统会自动检测安装环境，检测成功后填写以下基本信息，点击下一步。

- 网格名称：以小写字母开头，由小写字母、数字、中划线（-）组成，且不能以中划线（-）结尾
- Istio 版本：如果是托管网格，所有被纳管集群的 Istio 都将使用此版本。
- 集群：这是运行网格控制面的集群，集群下拉列表中展示了每个集群的版本及其健康状态。
- 控制面入口方式：支持负载均衡和自定义。
- 网格组件仓库：输入包含数据面组件镜像的镜像仓库地址，例如 release-ci.daocloud.io/mspider。



基本信息

3. 系统设置。配置是否上报链路追踪，设置网格规模，选择存储配置后点击下一步。

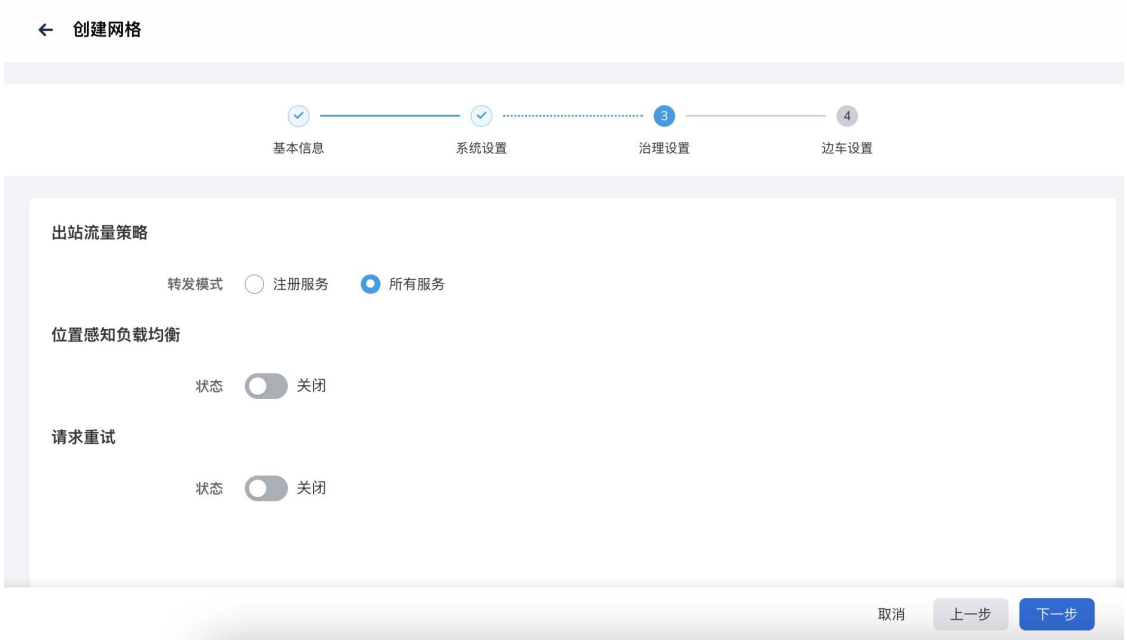


系统设置

!!! note

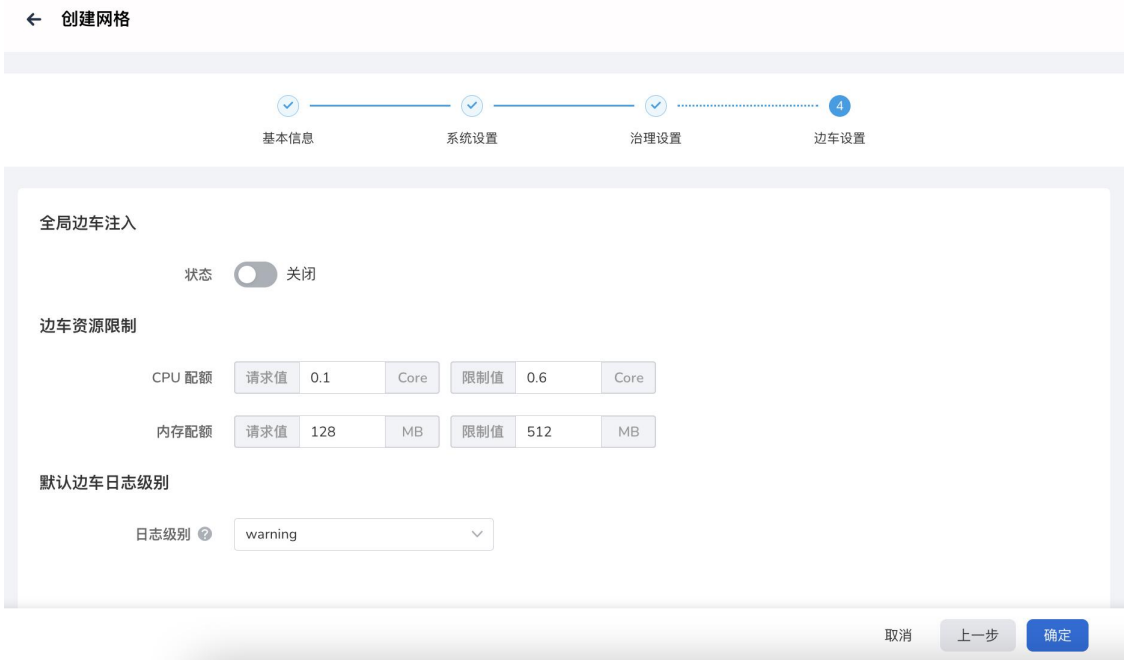
- 存储配置仅适用于托管网格，专有网格、外接网格不适用此项。
- 当控制面所在集群为 OCP 时，可选安装 OCP 组件

4. 治理设置。设置出站流量策略、位置感知负载均衡、请求重试等。参阅请求重试参数说明。



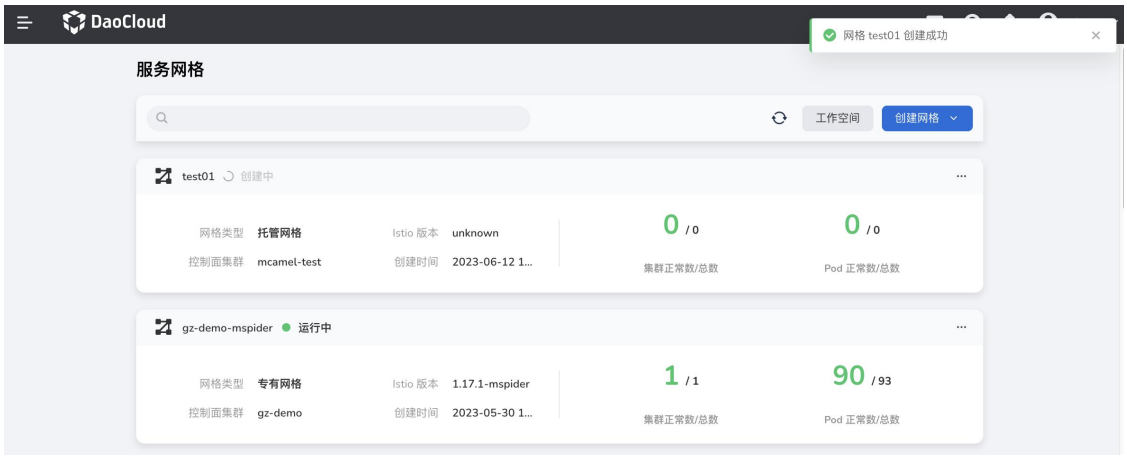
治理设置

5. 边车设置。设置全局边车、边车资源限制、默认边车日志级别点击确定。参阅日志级别说明。



边车设置

6. 自动返回网格列表，新创建的网格默认位于第一个，一段时间后状态将从创建中变为运行中。点击右侧的 ... 可以编辑网格基本信息、添加集群、进入控制台操作等。



网格列表

!!! info

创建托管网格后，还未接入任何托管的集群，此时网格处于`未就绪`状态。
用户可以[添加集群](../cluster-management/README.md)，等待集群接入完成后，选择需要服务治理的集群接入。

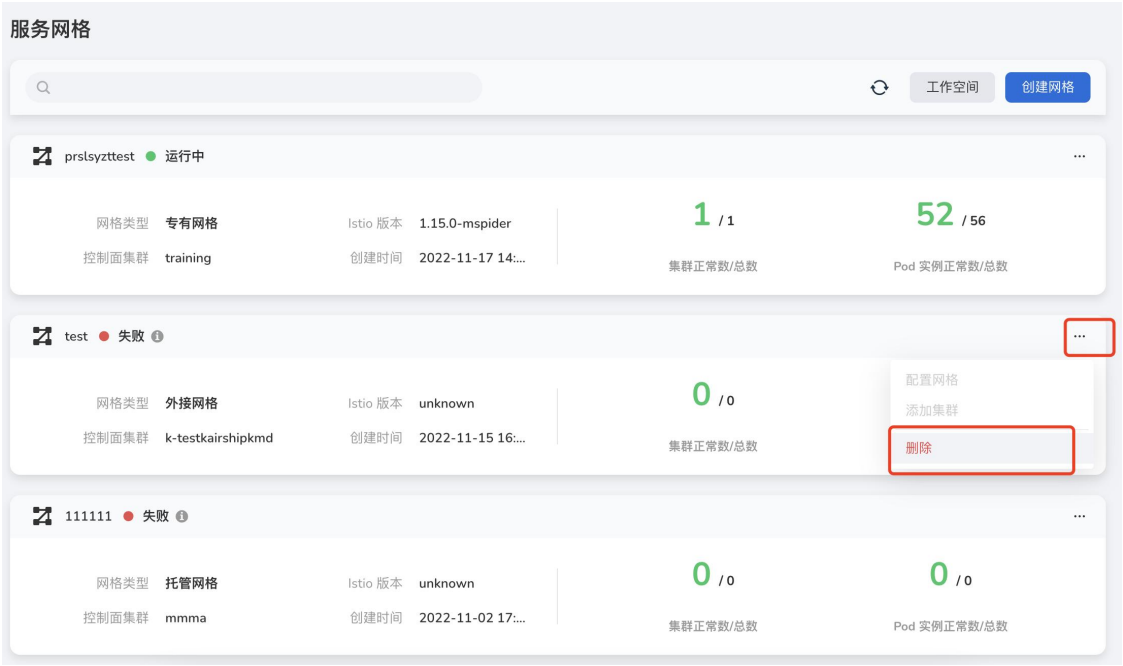
删除网格

当用户不再使用网格治理服务时可以使用删除操作，删除网格需用户完成一系列前置操作，才可以激活对话框的确定按钮。

!!! danger

执行删除操作后，网格不可恢复，如需再次纳管，需重新创建网格。

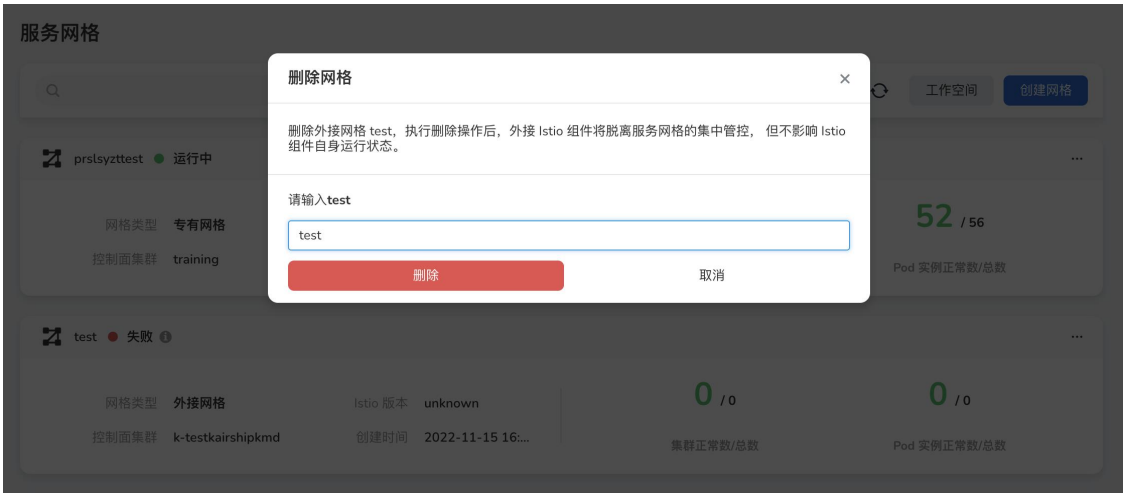
1. 在网格列表的右侧，点击 ... 按钮，在弹出菜单中选择删除。



删除网格

2. 按照网格类型，系统将自动检测是否满足删除条件。

    - 删除外接网格。只需输入网格名称确认，就能完成删除操作。



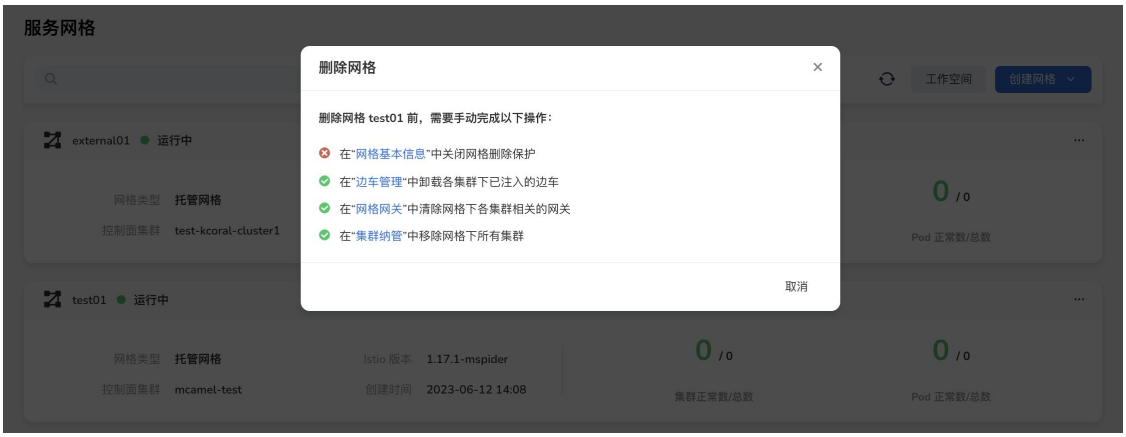
外接网格

- 删除专有网格。需要按提示关闭网格删除保护、移除边车、清除网关后，输入网格名称确认删除操作。



专有网格

- 删除托管网格。需要按提示关闭网格删除保护、移除已注入的边车、清除网关、移除网格下的集群后，输入网格名称确认删除操作。


服务网格
工作空间
创建网格
external01
运行中
网格类型
托管网格
控制面集群
test-kcoral-cluster1
0 / 0
Pod 正常数/总数
删除网格
删除网格 test01 前，需要手动完成以下操作：
在"网格基本信息"中关闭网格删除保护
在"边车管理"中卸载各集群下已注入的边车
在"网格网关"中清除网格下各集群相关的网关
在"集群纳管"中移除网格下所有集群
取消
test01
运行中
网格类型
托管网格
Istio 版本
1.17.1-mspider
控制面集群
mcamel-test
创建时间
2023-06-12 14:08
0 / 0
集群正常数/总数
0 / 0
Pod 正常数/总数

托管网格

创建网格的参数说明

最大重试次数 Max retries

服务网格中内嵌了 Envoy 组件，在创建网格时可以设置 Envoy 相关的一些参数。 更多细节请参阅 Envoy 文档。

HTTP

- 5xx

  如果上游服务器响应任何 5xx 响应代码，或者根本不响应（断开连接/重置/读取超时），将尝试重试 （包括 connect-failure 和 refused-stream）。

- connect-failure

  如果由于与上游服务器的连接失败（连接超时等）导致请求失败，将尝试重试（包含在 5xx 中）。

- envoy-ratelimited

  如果标头 x-envoy-ratelimited 存在，Envoy 将重试。

- gateway-error

  此策略类似于 5xx 策略，但只会重试导致 502、503 或 504 的请求。

- http3-post-connect-failure

  如果通过 HTTP/3 向上游服务器发送请求并在连接后失败，将尝试重试。

- refused-stream

  如果上游服务器使用 REFUSED_STREAM 错误代码重置流，将尝试重试。 此重置类型表示请求可以安全重试（包含在 5xx 中）。

- reset

  如果上游服务器根本没有响应（断开/重置/读取超时），将尝试重试。

- retriable-4xx

  如果上游服务器使用可重试的 4xx 响应代码进行响应，将尝试重试。 目前，该类别中唯一的响应代码是 409。

  注意：小心打开此重试类型。在某些情况下，409 可能表示需要更新乐观锁定修订版。 因此，调用者不应重试，需要读取然后再尝试写入。如果在这种情况下发生重试，它总是会失败并返回另一个 409。

- retriable-headers

  如果上游服务器响应包含任何与重试策略或 x-envoy-retriable-header-names 标头匹配的标头，将尝试重试。

- retriable-status-codes

  如果上游服务器响应任何与重试策略或 x-envoy-retriable-status-codes 标头中定义的响应代码匹配的响应代码，将尝试重试。

gRPC

- cancelled

  如果响应头中的 gRPC 状态码是“cancelled”，将尝试重试。

- deadline-exceeded

  如果响应头中的 gRPC 状态码是“deadline-exceeded”，将尝试重试。

- internal

  如果响应头中的 gRPC 状态码是“internal”，将尝试重试。

- resource-exhausted

如果响应头中的 gRPC 状态码是“resource-exhausted”，将尝试重试。

- unavailable

  如果响应头中的 gRPC 状态码是“unavailable”，将尝试重试。

边车日志级别

- critical

  任何强制关闭服务或应用程序以防止数据丢失（或进一步丢失数据）的错误。 您只为最令人发指的错误和保证数据损坏或丢失的情况保留这些。

- debug

  使用 debug，您将以详细的方式提供诊断信息。 它很冗长，包含的信息比您在使用该应用程序时所需的信息还要多。 debug 日志记录级别用于获取诊断、故障排除或测试应用程序所需的信息。 这确保了应用程序的平稳运行。

- error

  与 critical 日志记录级别不同，错误并不意味着您的应用程序正在中止。 相反，只是无法访问服务或文件。此 error 表明您的应用程序中的某些重要内容失败。 当严重问题导致应用程序内的功能无法有效运行时，将使用此日志级别。 大多数时候，应用程序会继续运行，但最终需要解决这个问题。

- info

  要记录的通用信息（服务启动/停止、配置假设等）。 您希望始终可用但在正常情况下通常不关心的信息。 这是开箱即用的配置级别。

- off

  此日志级别不记录任何内容。 此 off 级别用于关闭日志记录并且是最大可能级别。 使用此日志级别，根本不会记录任何内容。

- trace

  trace 日志级别捕获有关应用程序行为的所有详细信息。 它主要是诊断性的，比 debug 日志级别更细粒度。 此日志级别用于您需要查看应用程序中发生的事情或使用的第三方库中发生的事情的情况。 您可以使用 trace 日志级别来查询代码中的参数或解释算法的步骤。

- warning

  当您检测到意外的应用程序问题时，将使用 warning 日志级别。 这意味着您不太确定问题是否会再次发生或仍然存在。 此时您可能不会注意到对您的应用程序有任何损害。 此问题通常是停止特定进程运行的情况。但这并不意味着该应用程序已受到损害。 事实上，代码应该继续照常工作。您最终应该检查这些警告，以防问题再次发生。

网格规模对应资源用量说明

对于不同的服务网格规模，所需要的组件资源不尽相同，此处详细介绍在各个网格规模下，各个组件的资源用量（默认可以手动调整）。

小型网格

组件	内存请求	内存限制	CPU 请求	CPU 限制
mcpc-controller	50Mi	1600Mi	100m	300m
reg-proxy	50Mi	300Mi	100m	300m
ckube-remote	50Mi	500Mi	100m	500m
hosted-apiserver.etcd	100Mi	1000Mi	100m	500m
hosted-apiserver.apiserver	100Mi	1000Mi	100m	1000m
istiod	200Mi	1500Mi	200m	1500m
gateway	50Mi	900Mi	50m	1000m

中型网格

组件	内存请求	内存限制	CPU 请求	CPU 限制
mcpc-controller	200Mi	1600Mi	100m	600m
reg-proxy	100Mi	600Mi	100m	500m
ckube-remote	200Mi	1000Mi	200m	1000m
hosted-apiserver.etcd	200Mi	1000Mi	100m	500m
hosted-apiserver.apiserver	200Mi	1000Mi	100m	1000m
istiod	500Mi	2000Mi	400m	2000m
gateway	150Mi	1500Mi	300m	2000m

大型网格

组件	内存请求	内存限制	CPU 请求	CPU 限制
mcpc-controller	300Mi	2000Mi	300m	1000m
reg-proxy	200Mi	1000Mi	100m	1000m
ckube-remote	400Mi	2000Mi	200m	500m
hosted-apiserver.etcd	300Mi	1500Mi	100m	500m
hosted-apiserver.apiserver	300Mi	1500Mi	400m	2000m
istiod	800Mi	3000Mi	600m	3000m
gateway	500Mi	2000Mi	600m	3000m

TLS 密钥管理

密钥（Secret）是以键/值对形式保存的、包含证书、私钥、凭证等敏感信息的对象，用于服务间的链路 TLS 加密通信。 服务网格提供了界面化的 TLS 密钥管理功能。

准备凭证和私钥

以 helloworld 为例创建一个证书和一个私钥。

openssl req -out example_certs1/helloworld.example.com.csr -newkey rsa:2048 -nodes -keyout example_certs1/helloworld.example.com.key -subj "/CN=helloworld.example.com/O=helloworld organization"
openssl x509 -req -sha256 -days 365 -CA example_certs1/example.com.crt -CAkey example_certs1/example.com.key -set_serial 1 -in example_certs1/helloworld.example.com.csr -out example_certs1/helloworld.example.com.crt

生成的证书类似如下：

-----BEGIN CERTIFICATE REQUEST-----
MIICiDCCAXACAQAwQzEfMB0GA1UEAwwWaGVsbG93b3JsZC5leGFtcGxlLmNvbTEg
MB4GA1UECgwXaGVsbG93b3JsZCBvcmdhbml6YXRpb24wggEiMA0GCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQDHLmX2uLbgWyrGC1/FMVPCToOoFnM0kKCQyAEbYxqX
HJV4CQ3V7UlyEldj/w0QK+eY8dD8QVKKLiX4DCNYM7Rv/X2Jltw0GHG6788VstGy
1tvNv9u/wsgHV7J0ybxn+iEIgppLTKiLjuqZv/8HPNvE/CcvGmPbH2depd5nvYxq
kTNhYU1T8wPfSPSRoPZncwqjwnFy+lkjWzO/NBYYtVDU21VAuDJmqF8oO6cFSulm
OG0GDKaE0eXkYnRQSfssZHtGSEgb/R6feZNhIa4DajHYiGu37qqdS+gPodzGtRGl
ryc2gsXJNywnlSpw78ne3GCgvCVshHV1a9+XaWo0nkN9AgMBAAGgADANBgkqhkiG
9w0BAQsFAAOCAQEAZiDs4ICVbmZhscdlE/vJlNWLL77dAa2nstFQHrck0rlNdqDF

YYF84J8OjNpaHklzwQ+icKJ9j6FjdBkmCvfpd9nMnihe6XTbfNNEx4hyGCg1My5x
XmeKsd/4U5oJCIe3qb0U/KDslAbdv859DnSGHnf3rwx22FnaIlw2bnbylkRXqlPO
QjCoOu9Yor2sxfDw+gVZdUnsKVQYrII4RPJrBL+DWs9uUu/LrfZx8VHyGQSSNaMY
ivThfuaDOSlWR1Rqd6+z13/nmJrV0HJWwOk18Of5O/Cqb9sx5aduML9zmd2hpv5a
qCLeUusdlSW12hVnudaP7TM3RE7R6r1TwwXE2As==
-----END CERTIFICATE REQUEST-----

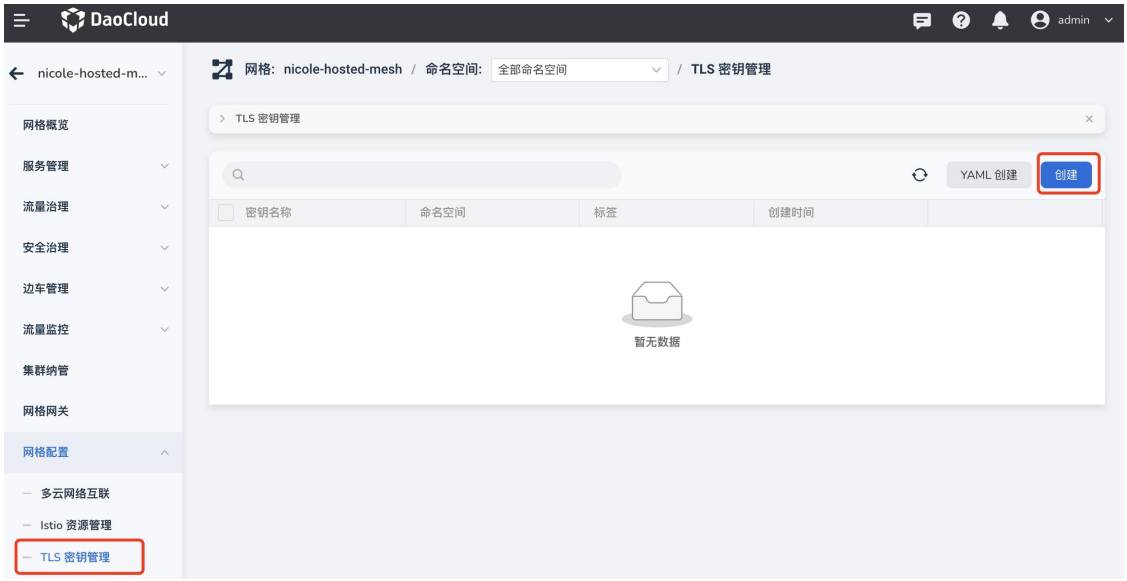
生成的私钥类似如下：

-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDHLmX2uLbgWyrG
C1/FMVPCToOoFnM0kKCQyAEbYxqXHJV4CQ3V7UlyEldj/w0QK+eY8dD8QVKKLiX4
DCNYM7Rv/X2Jltw0GHG6788VstGy1tvNv9u/wsgHV7J0ybxn+iElgppLTKiLjuqZ
v/8HPNvE/CcvGmPbH2depd5nvYxqkTNhYU1T8wPfSPSRoPZncwqjwnFy+lkjWzO/
NBYYtVDU21VAuDJmqF8oO6cFSulmOG0GDKaE0eXkYnRQSfssZHtGSEgb/R6feZNh
la4DajHYiGu37qqdS+gPodzGtRGlryc2gsXJNywnlSpw78ne3GCgvCVshHV1a9+X
aWo0nkN9AgMBAAECggEASMh/oeRljx5BRxUo7dnUjZA7M9TgTDj9M2qiXOud79lW
HELSt3V5/SBlGPk+QMaKXq0efM8x5iRu43EsCTL7oGC02u4xkSfNh03SEk/BqOQy
dcDCABShG23EXEDLhAM20Yoj75gXjseumng6LN7kP9sURnYSPyP+letulelByX6Y
is/bKKGcs4L5txH6KpMJL8hOW/6nK1AREeDVmd4Y98CoU+OC2R9ThR2FFRXgdYx
h
2xCZ68+XQ1bUf2+24Yl9/73iPvKRPe9zZKPhMak1kd7Lnuy3DEXAvjhpPs2eqiJ9
J7n6DkDn/fAMN6yvkm07vBbWS8AV421iKX+rf7b3AQKBgQDjm4Zg+/NXj+egrJio
Qt7UflklP1dGje60OfKSZj7F3pBh/R1+HHmXh0YHOvDy0OEBUY9h97XyYHlBvPMf
fnyr2S0czk/HpcS6TnFj9QaVjtQdZ8h7Sf+3uPWHgqvu+gBJarA5fQZSgNRezEBo
5mqh+wXvzQioTrm3E1nBYUxw8QKBgQDgBxoD0DrLgypliqR95i4lyf4Yl5MzgurO
F2MLFkQ6QWdMfzZnh7Cs/CdqUD3weZTqPFom3zDe1NFHG5xnGrdIoKzUMZ5JVG8
t
xznpckq7mPnm8ghGZ6R515SFl5l5iEyy15ct1v+nKcjRNHmYe0d5tlWRDrfGcXJu
DfC73CT7TQKBgGJjlqFSCl0vbeds9Mi3r2+XLgoS3o1nSWqrKgrTHAuY/Dz/l8lu
OPmhxknV1taAKOPTB/JHjGVr/5x0u3w/x1DaHsA3BxG8vN/0jNuyzdfU2CiI9mol
QN+AmtKrT/uMlpeaAPe47gS4lBWioa02/Z1rz9MrhLSM44caXFBV6R9hAoGAaUgN
GsOuDdw7b9HwEdat00aFKjT1xZx92pK1Eg3JzJLWB+Y03Byxk+oAX/8LzMpmiFoK

iAAVyHK9UyyPqQiuH+yarDlRUCblBN9+wM3cfyMaNkWCTAwDCNueSdX/41SBrv6Q
ZpOGm7mQTXjauCUfZvvGVXBUP2crPrtAahjALHUCgYEAxGvVH0I6Uphpf4gGqqcm
1ge0R6nGtPeKn6/Nh2Y3sHVxGvByFkuRXqZ2gSw6etleSJhQLLcvZm/K6FZu0sMQ
C4jSbYTJupuL8x4uSCEH5X/na/6AwUfmwA0P7jGuKKhHgsHpH8OHly6eHonM7pzn
WQuRxan+FuKE3RiKHFXkslks=
-----END PRIVATE KEY-----

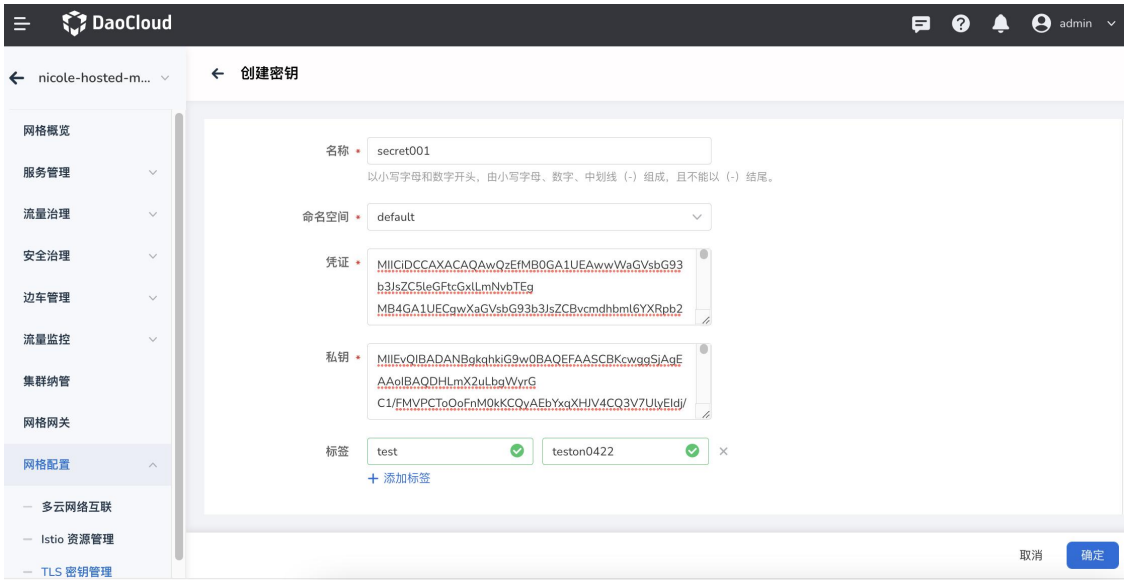
创建密钥

1. 进入某个网格后，在左侧导航栏点击网格配置 -> TLS 密钥管理，点击创建按钮。



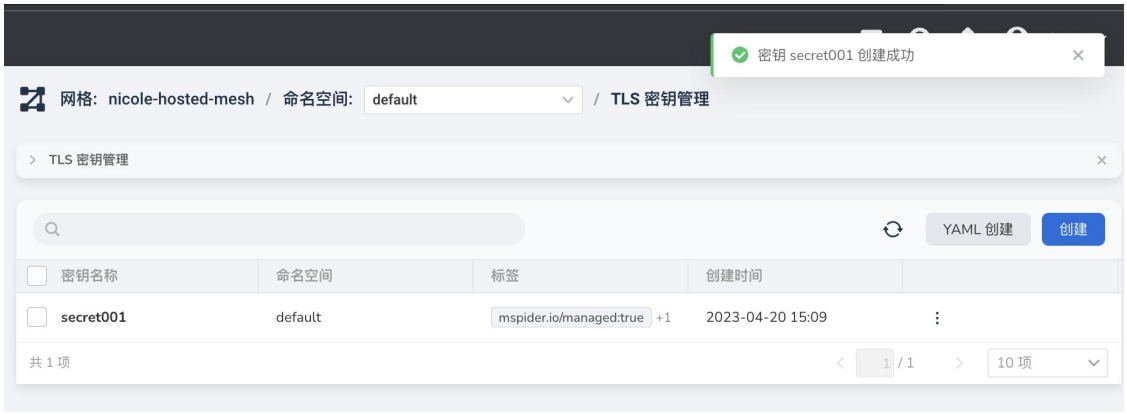
点击创建

2. 填写名称，选择命名空间，填入刚创建的凭证和私钥，根据情况添加标签后，点击确定。



填写

3. 屏幕提示创建成功，点击右侧的 ⋮ 按钮，可以执行编辑、YAML 编辑以及删除等操作。



填写

本例创建成功后的密钥 YAML 如下：

```yaml
metadata:
  name: secret001
  namespace: default
  uid: 1b43fc33-7898-40f9-9868-b65570f35a3d
  resourceVersion: '450493'
```

```
  creationTimestamp: '2023-04-20T07:09:33Z'
  labels:
    mspider.io/managed: 'true'
    test: teston0422
  annotations:
    ckube.daocloud.io/indexes: >-
      {"cluster":"nicole-hosted-mesh-hosted","createdAt":"2023-04-20T07:09:33
Z","is_deleted":"false","labels":"\"mspider.io/managed=true\",\"test=teston0422\",",
"name":"secret001","namespace":"default","type":"kubernetes.io/tls"}
    ckube.doacloud.io/cluster: nicole-hosted-mesh-hosted
  managedFields:
    - manager: cacheproxy
      operation: Update
      apiVersion: v1
      time: '2023-04-20T07:09:33Z'
      fieldsType: FieldsV1
      fieldsV1:
        f:data:
          .: {}
          f:tls.crt: {}
          f:tls.key: {}
        f:metadata:
          f:labels:
            .: {}
            f:mspider.io/managed: {}
            f:test: {}
        f:type: {}
data:
  tls.crt: >-
    TUlJQ2lEQ0NBWEFDQVFBd1F6RWZNQjBHQTFVRUF3d1dhR1ZzYkc5M2lzSnNa
QzVsZUdGdGNHeGxMbU52YlRFZwpNQjRHQTFVRUNnd1hhR1ZzYkc5M2lzSnNaQ0J
2Y21kaGJtbDZZWFJwYjI0d2dnRWlNQTBHQ1NxR1NJYjNEUUVCCkFRVUFBNElCRHd
```

Bd2dnRUtBb0lCQVFESExtWDJ1TGJnV3lyR0MxL0ZNVlBDVG9Pb0ZuTTBrS0NReUFF
Yll4cVgKSEpWNENRM1Y3VWx5RUlkai93MFFLK2VZOGREOFFWS0tMaVg0RENOW
U03UnYvWDJKbHR3MEdlRzY3ODhWc3RHeQoxdHZOdjl1L3dzZ0hWN0oweWJ4bitp
RWxncHBMVEtpTGp1cVp2LzhlUE52RS9DY3ZHbVBiSDJkZXBkNW52WXhxCmtUTmh
ZVTFUOHdQZlNQU1JvUFpuY3dxanduRnkrSWtqV3pPL05CWVl0VkRVMjFWQXVESm1
xRjhvTzZjRlN1bG0KT0cwR0RLYUUwZVhrWW5SUVNmc3NaSHRHU0VnYi9SNmZlWk
5oSWE0RGFqSFlpR3UzN3FxZFMrZ1BvZHpHdFJHSQpyeWMyZ3NYSk55d25JU3B3N
zhuZTNHQ2d2Q1ZzaEhWMWE5K1hhV28wbmtOOUFnTUJBQUdnQURBTkJna3Foa2l
HCjI3MEJBUXNGQUFPQ0FRRUFaaURzNElDVmJtWmhzY2RJRS92SklOV0xMNzdkQ
WEybnN0RlFIcmNrMHJJTmRxREYKWVlGODRKOE9qTnBhSGtJendRK2ljS0o5ajZGa
mRCa21DdmZwZDluTW5paGU2WFRiZk5ORXg0aHlHQ2cxTXk1eApYbWVLc2QvNFU1
b0pDbGUzcWIwVS9LRHNJQWJkdjg1OURuU0dlbmYzcnd4MjJGbmFJbHcyYm5ieUlrU
lhxSVBPClFqQ29PdTlZb3lyc3hmRHcrZ1ZaZFVuc0tWUVlySUk0UlBKckJMK0RXczl1V
XUvTHJmWng4Vkh5R1FTU05hTVkKaXZUaGZ1YURPU0lXUjFScWQ2K3oxMy9ubUpy
VjBlSld3T2sxOE9mNU8vQ3FiOXN4NWFkdU1MOXptZDJocHY1YQpxQ0xlVXVzZGxTV
zEyaFZudWRhUDdUTTNSRTdSNnlxVHd3WEUyQT09s

tls.key: >-

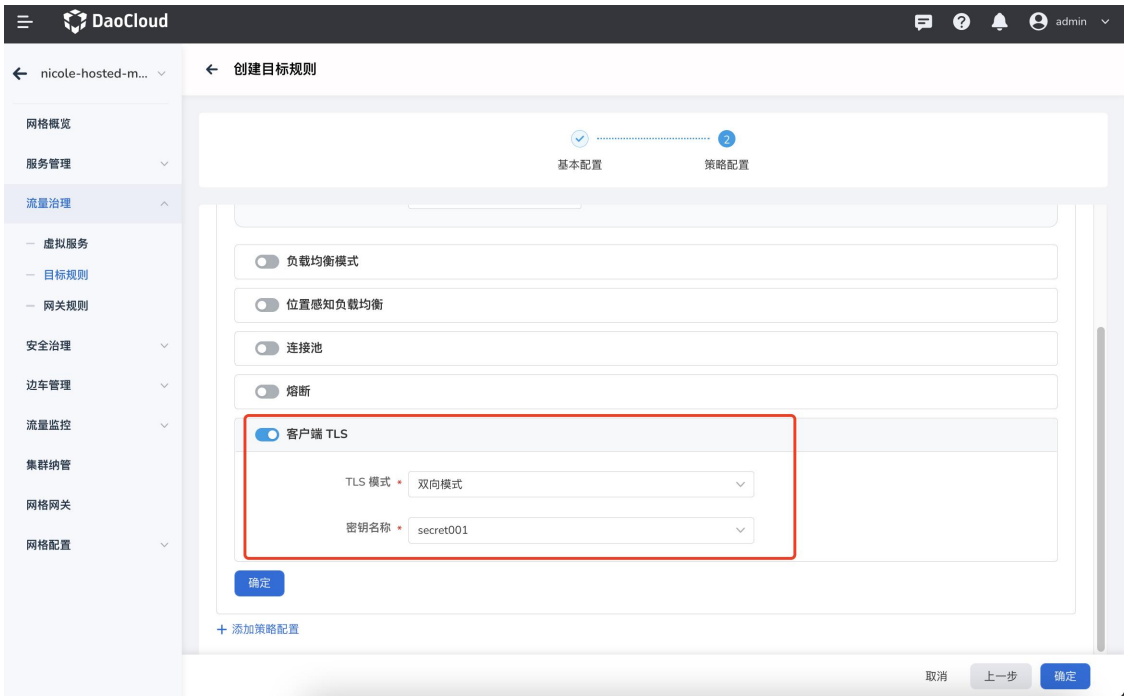
TUlJRXZRSUJBREFOQmdrcWhraUc5dzBCQVFFRkFBU0NCS2N3Z2dTakFnRUF
Bb0lCQVFESExtWDJ1TGJnV3lyRwpDMS9GTVZQQ1RvT29Gbk0wa0tDUXlBRWJZeHF
YSEpWNENRM1Y3VWx5RUlkai93MFFLK2VZOGREOFFWS0tMaVg0CkRDTllNN1J2L1
gySmx0dzBHSEc2Nzg4VnN0R3kxdHZOdjl1L3dzZ0hWN0oweWJ4bitpRWxncHBMVEt
pTGp1cVoKdi84SFBOdkUvQ2N2R21QYkgyZGVwZDVudll4cWtUTmhZVTFUOHdQZlN
QU1JvUFpuY3dxanduRnkrSWtqV3pPLwpOQllZdFZEVTIxVkF1REptcUY4b082Y0ZTd
WxtT0cwR0RLYUUwZVhrWW5SUVNmc3NaSHRHU0VnYi9SNmZlWk5oCklhNERhakh
ZaUd1MzdxcWRTK2dQb2R6R3RSR0lyeWMyZ3NYSk55d25JU3B3NzhuZTNHQ2d2Q1
ZzaEhWMWE5K1gKYVdvMG5rTjlBZ01CQUFFQ2dnRUFTTWgvb2VSbGp4NUJSeFVvN
2RuVWpaQTdNOVRnVERqOU0ycWlYT3VkNzlJVwplRUxTdDNWNS9TQmxHUGsrUU1
hS1hxMGVmTTh4NWlSdTQzRXNDVEw3b0dDMDJ1NHhrU2ZOaDAzU0VrL0JxT1F5C
mRjRENBQlNoRzlzRVhFRExoQU0yMFlvajc1Z1hqc2V1bW5nNkxON2tQOXNVUm5ZU1
B5UCtsZXR1bGVJQnlYNlkKaXMvYktLR2NzNEw1dHhlNktwTUpMOGhPVy82bksxQVJ
FZURWbWQ0WTk4Q29VK09DMll5VGhSMkZGUlhnZFl4aAoyeENaNjgrWFExYlVmMis
yNFlJOS83M2lQdktSUGU5elpLUGhNYWsxa2Q3TG51eTNERVhBdmpocFBzMmVxaU

o5Cko3bjZEa0RuL2ZBTU42eXZrbTA3dkJiV1M4QVY0MjFpS1grcmY3YjNBUUtCZ1FE
am00WmcrL05YaitlZ3JKaW8KUXQ3VWZJa2xwMWRHamU2ME9mS1NaajdGM3BCa
C9SMStlSG1YaDBZSE92RHkwT0VCVVk5aDk3WHlZSElCdlBNZgpmbnlyMlMwY3prL0
hwY1M2VG5GajlRYVZqdFFkWjhoN1NmKzN1UFdlZ3F2dStnQkphckE1ZlFaU2dOUmV6
RUJvCjVtcWgrd1h2elFpb1RybTNFMW5CWVV4dzhRS0JnUURnQnhvRDBEckxneXBJa
XFSOTVpNGx5ZjRZSTVNemd1ck8KRjJNTEZrUTZRV2RNZnpabmg3Q3MvQ2RxVUQz
d2VaVHFQRm9tM3pEZTFORkhHNXhuR3JkSW9LelVNWjVKVkc4dAp4em5wY2txN21
Qbm04Z2hHWjZSNTE1U0ZsNWw1aUV5eTE1Y3QxdituS2NqUk5lbVllMGQ1dElXUkRyZ
kdjWEp1CkRmQzczQ1Q3VFFLQmdHSmpscUZTQ0kwdmJlZHM5TWkzcjlrWExnb1Mzb
zFuU1dxcktnclRlQXVZL0R6L2w4SXUKT1BtaHhrblYxdGFBS09QVElvSkhqR1ZyLzV4M
HUzdy94MURhSHNBM0J4Rzh2Ti8wak51eXpkZlUyQ2lsOW1vbApRTitBbXRLclQvdU1J
cGVhQVBlNDdnUzRJQldpb2EwMi9aMXJ6OU1yaExTTTQ0Y2FYRkJWNll5aEFvR0FhV
WdOCkdzT3VEZHc3YjlId0VkYXQwMGFGS2pUMXhaeDkycEsxRWczSnpKTFdCK1kw
M0J5eGsrb0FYLzhMek1wbWlGb0sKaUFBVnllSzlVeXlQcVFpdUgreWFyRElSVUNibEJ
OOSt3TTNjZnlNYU5rV0NUQXdEQ051ZVNkWC80MVNCcnY2UQpacE9HbTdtUVRYa
mF1Q1VmWnZ2R1ZYQlVQMmNyUHJ0QWFoakFMSFVDZ1lFQXhHdlZlMEk2VXBocGY
0Z0dxcWNtCjFnZTBSNm5HdFBlS242L05oMlkzc0hWeEd2QnlGa3VSWHFaMmdTdzZ
ldGxlU0poUUxMY3ZabS9LNkZadTBzTVEKQzRqU2JZVEp1cHVMOHg0dVNDRUg1WC
9uYS82QXdVZm13QTBQN2pHdUtLaEhnc0hwSDhPSGx5NmVlb25NN3B6bgpXUXVS
eGFuK0Z1S0UzUmlLSEZYa3Nsaz0s=
type: kubernetes.io/tls

使用场景

创建好的 TLS 密钥可用于：

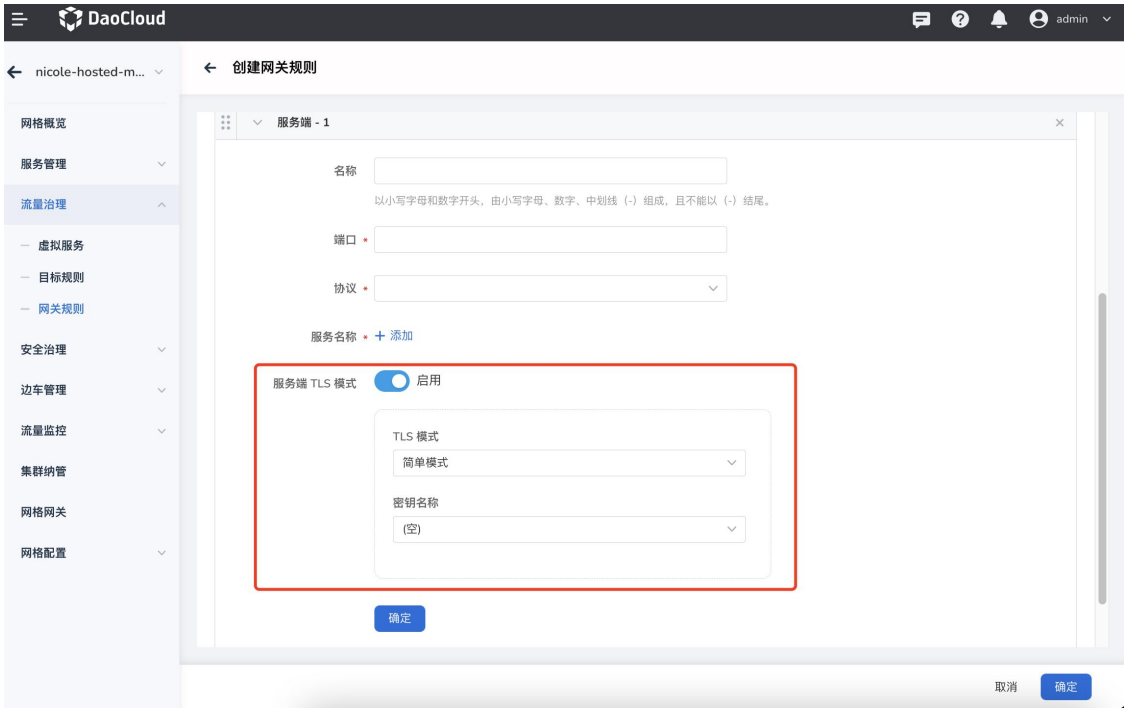
1. 目标规则。添加策略配置时，启用客户端 TLS 后，可以在以下 2 种模式种选择创建好的密钥：
   - 简单模式
   - 双向模式



目标规则

2. 网关规则。启用服务端 TLS 模式后，可以在以下 3 种模式种选择创建好的密钥：

   - 简单模式
   - 双向模式
   - Istio 双向模式


DaoCloud
admin
nicole-hosted-m...
网格概览
服务管理
流量治理
虚拟服务
目标规则
网关规则
安全治理
边车管理
流量监控
集群纳管
网格网关
网格配置
创建网关规则
服务端 - 1
名称
以小写字母和数字开头，由小写字母、数字、中划线（-）组成，且不能以（-）结尾。
端口 *
协议 *
服务名称 * + 添加
服务端 TLS 模式
启用
TLS 模式
简单模式
密钥名称
(空)
确定
取消
确定

网关规则

组件资源弹性伸缩

用户可以在容器管理对服务网格的控制面组件实现弹性伸缩策略，目前提供了三种弹性伸缩方式：指标收缩（HPA）、定时收缩（CronHPA）、垂直伸缩（VPA），用户可以根据需求选择合适的弹性伸缩策略。下面以指标收缩（HPA）为例，介绍创建弹性伸缩策略的方法。

前提条件

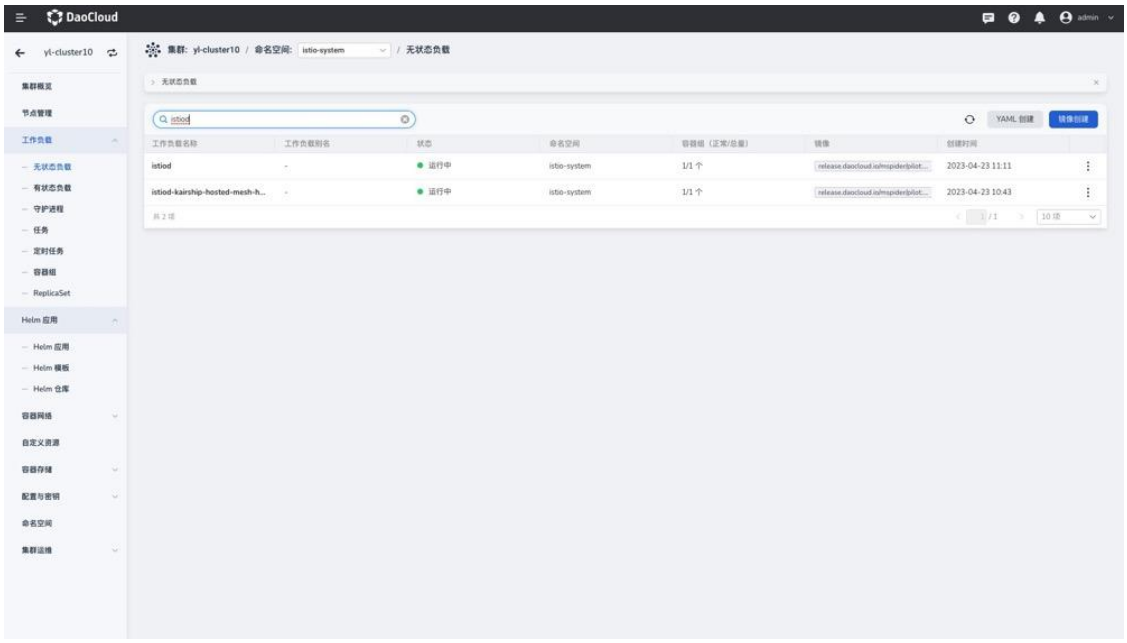
确保集群已安装 helm 应用 Metrics Server，可参考安装 metrics-server 插件

![环境依赖](https://docs.daocloud.io/daocloud-docs-images/docs/mspider/user-guide/mesh-config/images/meshrcfg07.png)

创建策略

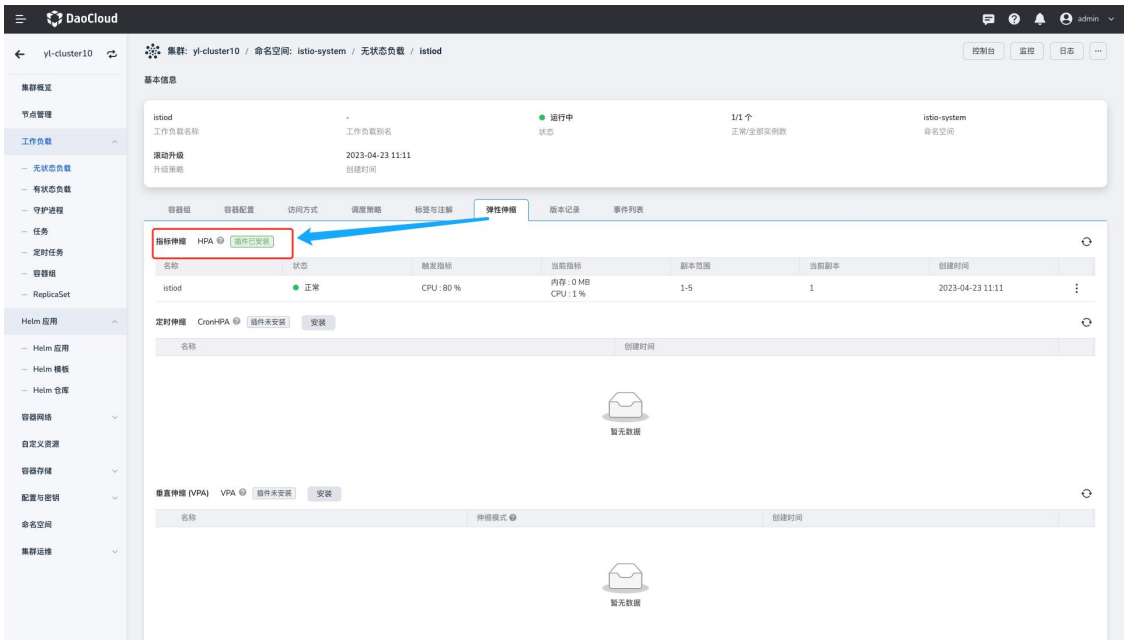
以专有集群的 istiod 为例，具体操作如下：

1. 在[容器管理]中选择对应集群，点击进入工作负载 -> 无状态负载页面查找 istiod；



查找 istiod

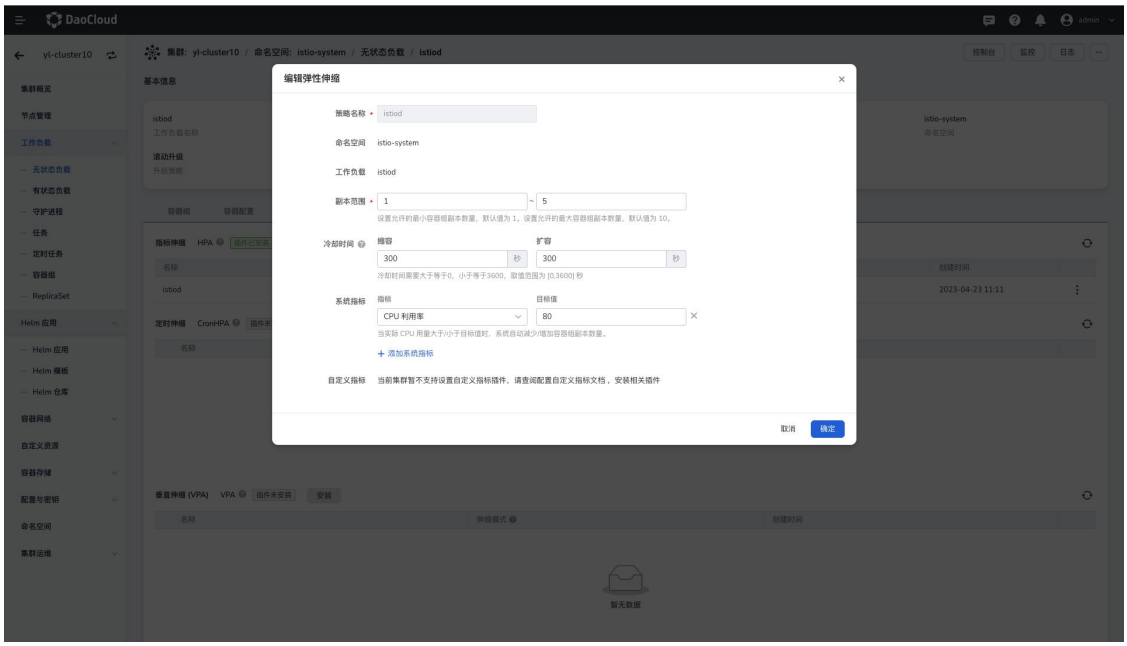
2. 点击工作负载名称进入弹性伸缩标签页；



标签页

3. 点击编辑按钮，配置弹性伸缩策略参数；

- 策略名称：输入弹性伸缩策略的名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 hpa-my-dep。
- 命名空间：负载所在的命名空间。
- 工作负载：执行弹性伸缩的工作负载对象。
- 目标 CPU 利用率：工作负载资源下 Pod 的 CPU 使用率。计算方式为：工作负载下所有的 Pod 资源/工作负载的请求（request`）值。当实际 CPU 用量大于/小于目标值时，系统自动减少/增加 Pod 副本数量。
- 目标内存用量：工作负载资源下的 Pod 的内存用量。当实际内存用量大于/小于目标值时，系统自动减少/增加 Pod 副本数量。
- 副本范围：Pod 副本数的弹性伸缩范围。默认区间为为 1 - 10。



编辑页

4. 点击确定完成编辑，此时新的策略已生效。

更多弹性伸缩配置

请参考：

- 创建 HPA 弹性伸缩策略
- 创建 VPA 弹性伸缩策略

服务网格组件资源自定义

本文介绍如何对通过容器管理配置网格的控制面组件资源。

前提条件

集群已被服务网格纳管，网格组件已正常安装； 登录账号具有全局管理集群及工作集群中命名空间 istio-system 的 admin 或 editor 权限；

自定义操作

以托管网格下工作集群上 istiod 为例，具体操作如下：

1. 服务网格下查看托管网格 nicole-dsm-mesh 的接入集群为 nicole-dsm-c2，如下图所示。



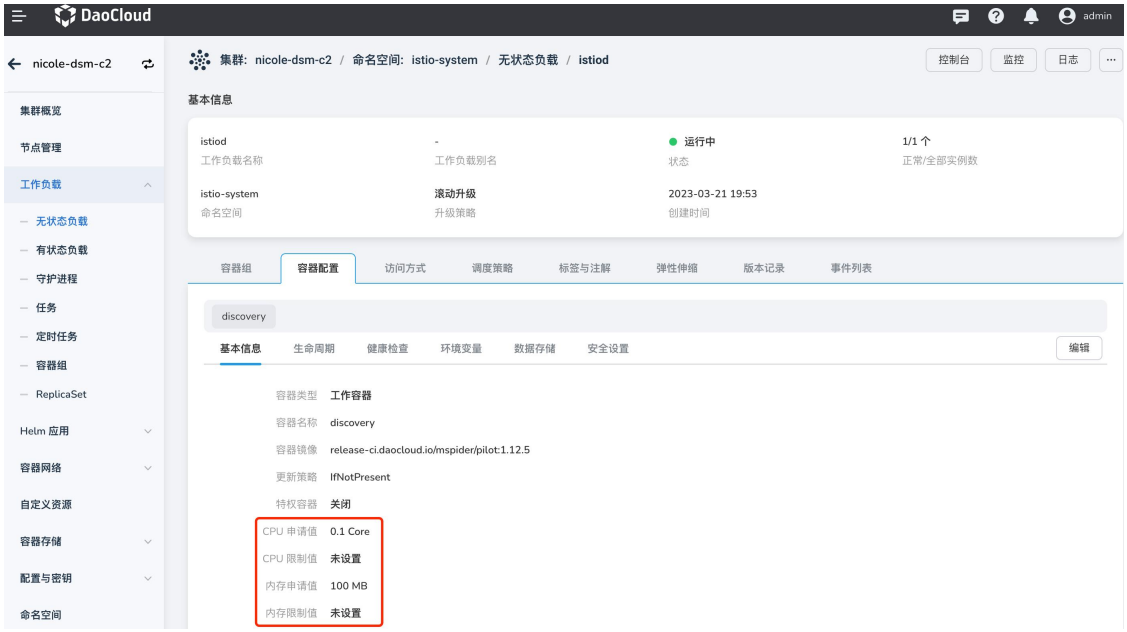
接入集群

2. 点击集群名称，跳转至容器管理模块中集群页面，点击进入工作负载 -> 无状态负载页面查找 istiod；



查找 istiod

3. 点击工作负载名称进入容器配置 –> 基本信息标签页；



查看配额

4. 点击编辑按钮，修改 CPU 和内存配额，点击下一步、确定。



修改配额

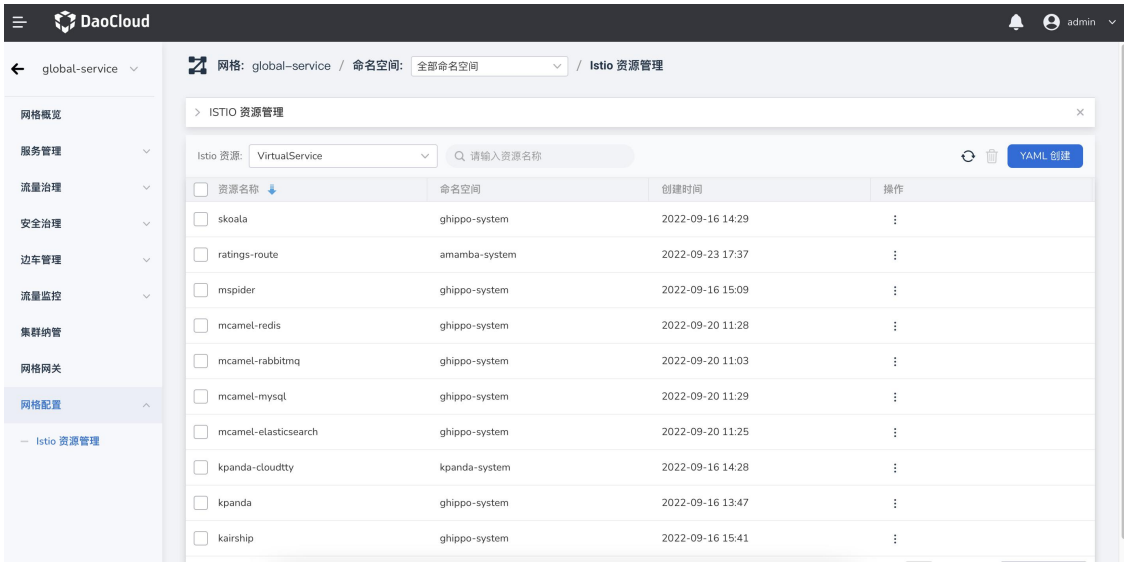
5. 查看该工作负载下的 Pod 资源信息，可见已变更.

```
bash-5.1# kubectl get po istiod-56f47d76c9-2xfrk -n istio-system -oyaml |grep request -C 5
      timeoutSeconds: 5
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 200Mi
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
```

确认配额

Istio 资源管理

Istio 资源管理页面按资源类型列出了 Istio 的所有资源，为用户提供了各类资源的展示、创建、编辑、删除等能力。

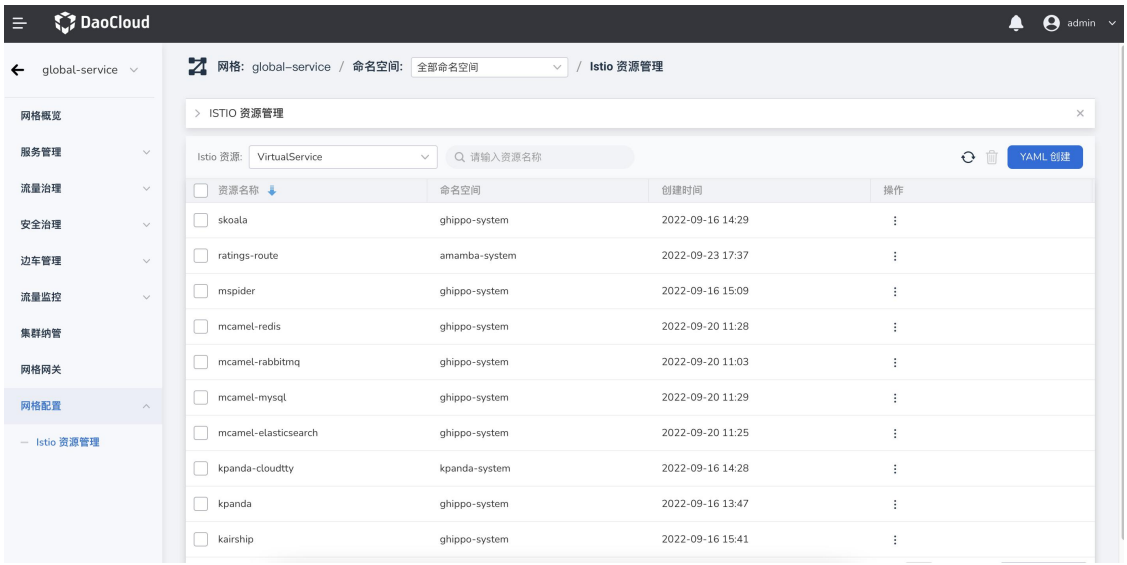


img

UI 操作示例

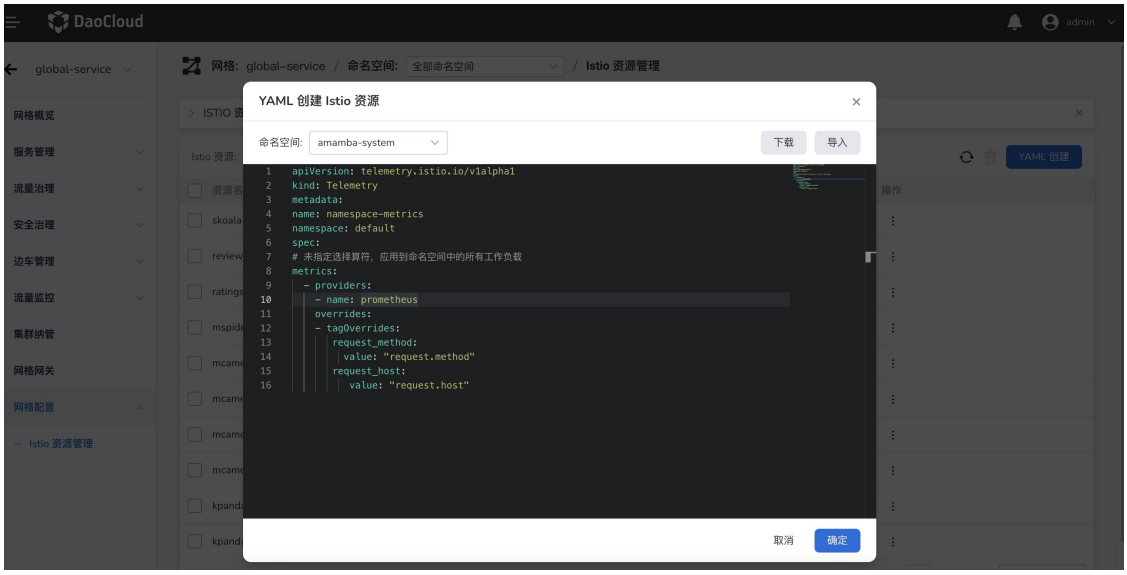
用户可以用 YAML 形式对资源进行增删改查操作。此处是一个 telementry 遥测资源的创建/删除示例。

1. 在左侧导航栏中点击网格配置 –> Istio 资源管理，点击右上角的 YAML 创建按钮。



img

2. 在 YAML 创建页面中，输入正确的 YAML 语句后点击确定。

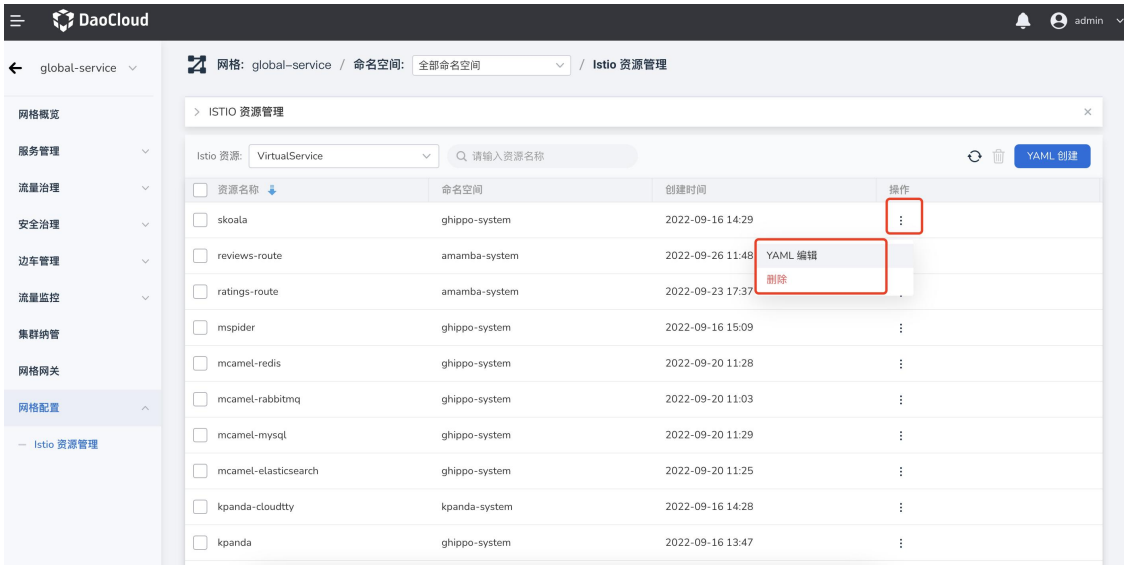


img

```yaml
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: namespace-metrics
  namespace: default
```

```yaml
spec:
  # 未指定选择算符，应用到命名空间中的所有工作负载
  metrics:
  - providers:
    - name: prometheus
    overrides:
    - tagOverrides:
          request_method:
            value: "request.method"
          request_host:
            value: "request.host"
```

3. 返回资源列表，点击操作一列的 ⋮ 按钮，可以从弹出菜单中选择编辑和删除等更多操作。



img

Istio 资源类型

Istio 常见的资源类型如下：

流量治理

资源类型	描述
DestinationRule（目标规则）	目标规则是服务治理中重要的组成部分，目标规则通过端口、服务版本等方式对请求流量进行划分，并对各请求分流量订制 Envoy 流量策略，应用到流量的策略不仅有负载均衡，还有最小连接数、离群检测等。
EnvoyFilter	该资源提供了对 Envoy 配置的能力，可以定义新的过滤器、监听器。使用该资源时需谨慎，错误配置可能会对整个网格环境的稳定性造成较大影响。注意： – EnvoyFilter 可以配置在 Istio 根目录（对所有工作负载生效）或某个工作负载（使用工作负载选择标签）； – 当多个 EnvoyFilter 作用于同一个工作负载时，会按创建顺序优先执行；
Gateway（网关规	网关规则用于定义网格边缘的负载均衡器，用于将服务暴露于网格之外，或提供内部服务的对外访问。相较于 k8s 的 ingress 对象，istio 的网关规则增加了更多的功能： l L4–L6 负载均衡 l 对外 mTLS l SNI 的支持 l 其他 istio 中已经实现的内部网络功能： Fault Injection，Traffic Shifting，Circuit Breaking， Mirroring 对于 L7 的支持，网关规则通过与虚拟服务配合实现。

资源类型	描述
则）	
ProxyConfig	用于暴露代理的配置选项，例如：代理的线程数。该资源为可选资源，如不创建，系统将使用内建默认值；注意： – ProxyConfig 中任何配置变更需要重启相关工作负载才会生效； – 作用于网格或命名空间的 ProxyConfig 不可以包含任何工作负载选择标签，否则将仅作用于选定的工作负载；
WorkloadEntry	该资源提供了对非 k8s 标准工作负载的描述，例如：运行于虚拟机的工作负载。WorkloadEntry 需要与服务条目配合使用，由服务条目通过筛选器建立服务与工作负载之间的对应关系；
ServiceEntry（服务条目）	服务条目允许向 Istio 的内部服务注册表中添加其他条目，以便网格中自动发现的服务可以访问/路由到这些手动指定的服务。服务条目描述服务的属性包括：DNS 名称、VIP、端口、协议、端点等。这些服务可以是网格外部的（例如：web API）或网格内部无法自动注册的服务（例如：一个数据库，几个 VM）。
SideCar	Sidecar 用于描述边车对工作负载实例之间的流量转发配置。默认情况下，Istio 将支持转发网格中所有工作负载实例之间的通信，并接受与工作负载相关的所有端口流量。 注意： – 根命名空间下的 SideCar 对所有没有配

资源类型	描述
	置 SideCar 的命名空间及工作负载生效； – 任意命名空间或工作负载如果存在多个 SideCar，将被定义为行为未定义（无生效资源）；
VirtualService（虚拟服务）	在虚拟服务中，提供了 HTTP、TCP、TLS 三种协议的路由支持，可以通过多种匹配方式（端口、host、header 等）实现对不同的地域、用户请求做路由转发，分发至特定的服务版本中，按权重比划分负载，并提供了故障注入、流量镜像等多种治理工具。
WorkloadGroup	WorkloadGroup 描述了工作负载实例的集合，定义了工作负载实例引导代理的规范细节，包括元数据和标识。WorkloadGroup 仅用于非 Kubernetes 工作负载，模拟 Kubernetes 中边车注入和部署方式，引导 Istio 代理，

安全治理

资源类型	描述
AuthorizationPolicy（授权策略）	授权策略类似于一种四层到七层的“防火墙”，它会像传统防火墙一样，对数据流进行分析和匹配，然后执行相应的动作。无论是来自外部的请求，或是网格内服务间的请求，都适用授权策略。
PeerAuthent	对等身份认证提供服务间的双向安全认证，同时密钥以及证书的创

资源类型	描述
ication（对等身份认证）	建、分发、轮转也都由系统自动完成，对用户透明，从而大大降低了安全配置管理的复杂度。
RequestAuthentication（请求身份认证）	请求身份认证用于外部用户对网格内部服务发起请求的认证，用户使用 jwt 实现请求加密；每个请求身份认证需要配置一个授权策略

代理扩展

资源类型	描述
WasmPlugin	WasmPlugin 通过 WebAssembly 过滤器为 Istio 代理提供扩展功能，其提供的过滤器能力可以和 Istio 内部过滤器形成复杂交互关系。

系统设置

资源类型	描述
Telemetry	该资源定义了如何为网格内的工作负载生成遥测，为 Istio 提供了指标、日志、全链路追踪三项可观测性工具的配置。 注意：创建在 Istio 根目录并且不包含工作负载选择器的遥测资源将对网格全局生效。

2023-07-23

使用网格完成定向服务访问限制

在业务场景中，我们需要限制只允许服务来访问某些特定的服务，这里可以组合 Istio 能力来进行统一的管理。

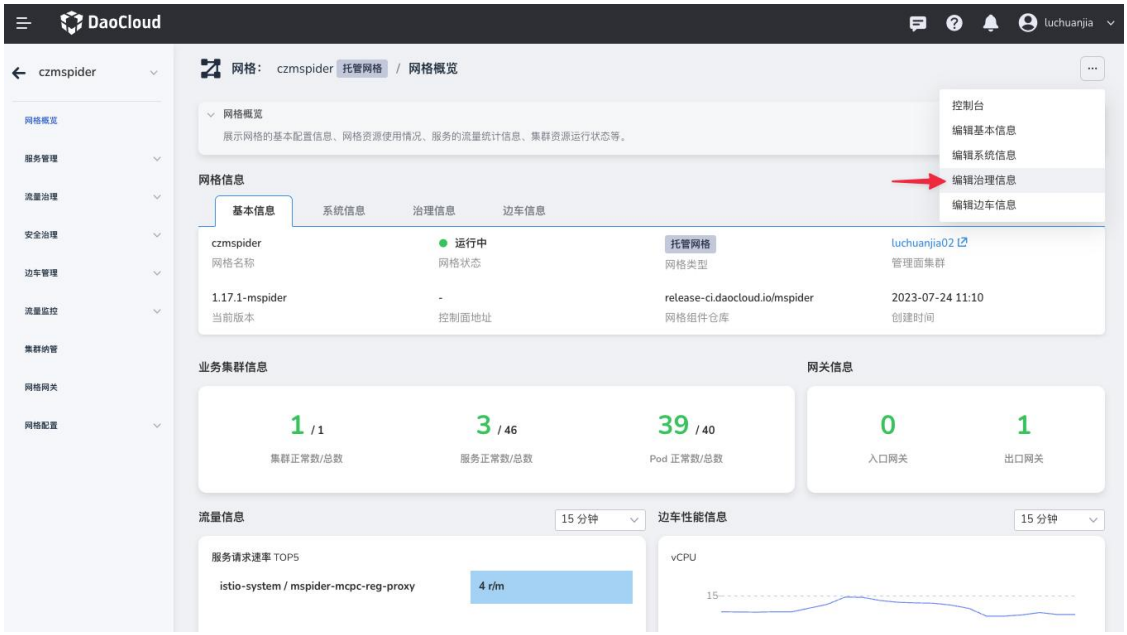
在 Istio 中，我们可以使用 Egress 来控制服务对外访问的流量，同时也可以使用 Service Entry 来控制网格外服务，结合授权策略（Authorized Policy）来控制服务对外访问的权限。本文将介绍如何使用 Egress 和授权策略来控制服务对外访问的流量和权限。

准备工作

首先，您需要确保你的网格属于正常的状态，如果您还没有安装 Istio，请参考 创建网格。

网格启用仅出口流量

配置网格启用仅出口流量，请修改网格的治理信息，查看下方的截图介绍，注意修改了之后，我们的服务对集群外的访问需要配合 Service Entry 来使用。

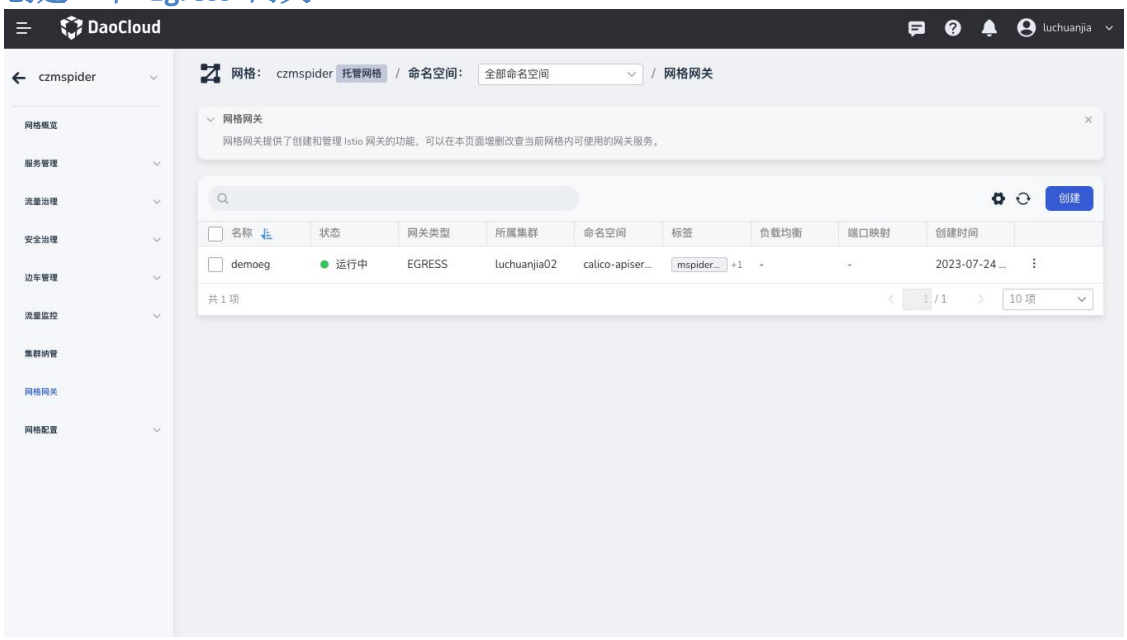

DaoCloud
luchuanjia
czmspider
网格概览
服务管理
流量治理
安全治理
边车管理
流量监控
集群纳管
网格网关
网格配置
网格： czmspider 托管网格 / 网格概览
网格概览
展示网格的基本配置信息、网格资源使用情况、服务的流量统计信息、集群资源运行状态等。
控制台
编辑基本信息
编辑系统信息
编辑治理信息
编辑边车信息
网格信息
基本信息
系统信息
治理信息
边车信息
czmspider
网格名称
运行中
网格状态
托管网格
网格类型
luchuanjia02
管理面集群
1.17.1-mspider
当前版本
-
控制面地址
release-ci.daocloud.io/mspider
网格组件仓库
2023-07-24 11:10
创建时间
业务集群信息
1 / 1
集群正常数/总数
3 / 46
服务正常数/总数
39 / 40
Pod 正常数/总数
网关信息
0
入口网关
1
出口网关
流量信息
15 分钟
服务请求速率 TOP5
istio-system / mspider-mcpc-reg-proxy
4 r/m
边车性能信息
15 分钟
vCPU
15

image

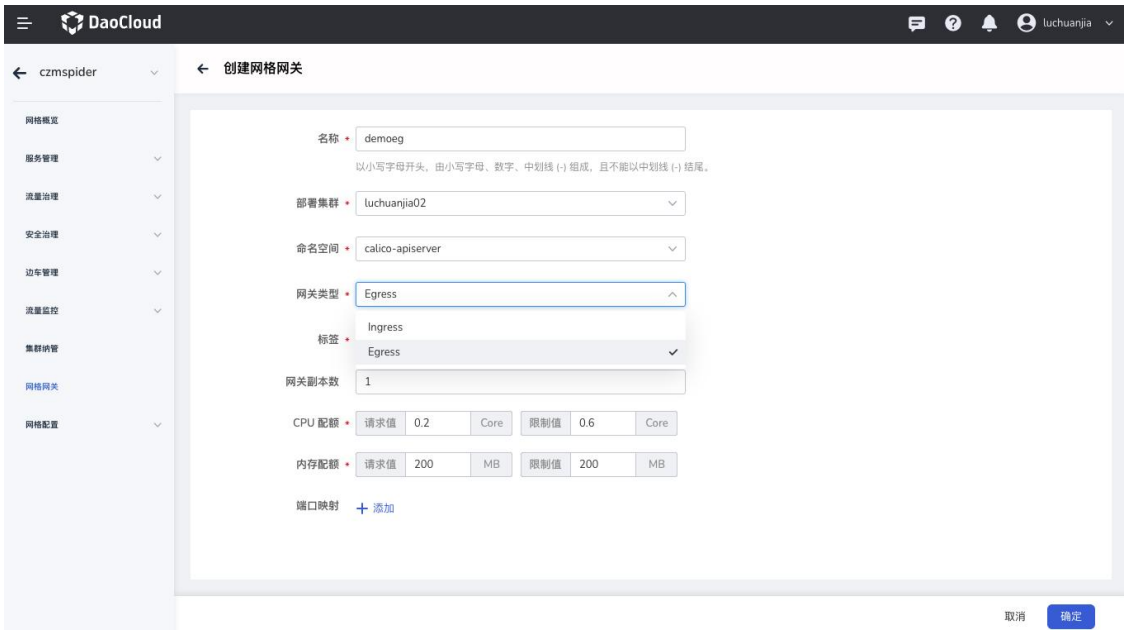

DaoCloud
luchuanjia
czmspider
网格概览
服务管理
流量治理
安全治理
边车管理
流量监控
集群纳管
网格网关
网格配置
编辑治理信息
出站流量策略
转发模式
注册服务
所有服务
位置感知负载均衡
状态
关闭
请求重试
状态
关闭
取消
确定

image

创建一个 Egress 网关



image



image

准备测试应用

可以使用任意应用进行测试，在后续步骤中，我们会通过 kubectl exec pod 进入 pod 内进行网络访问测试，建议至少保证应用内有 curl 命令即可。

这里是用了一个简单的 bookinfo 来做示例，您也可以使用其他应用。

另外，需要保证应用的 Pod 被成功注入了 sidecar，这个可以在网格的界面中查看对应服务的状态。



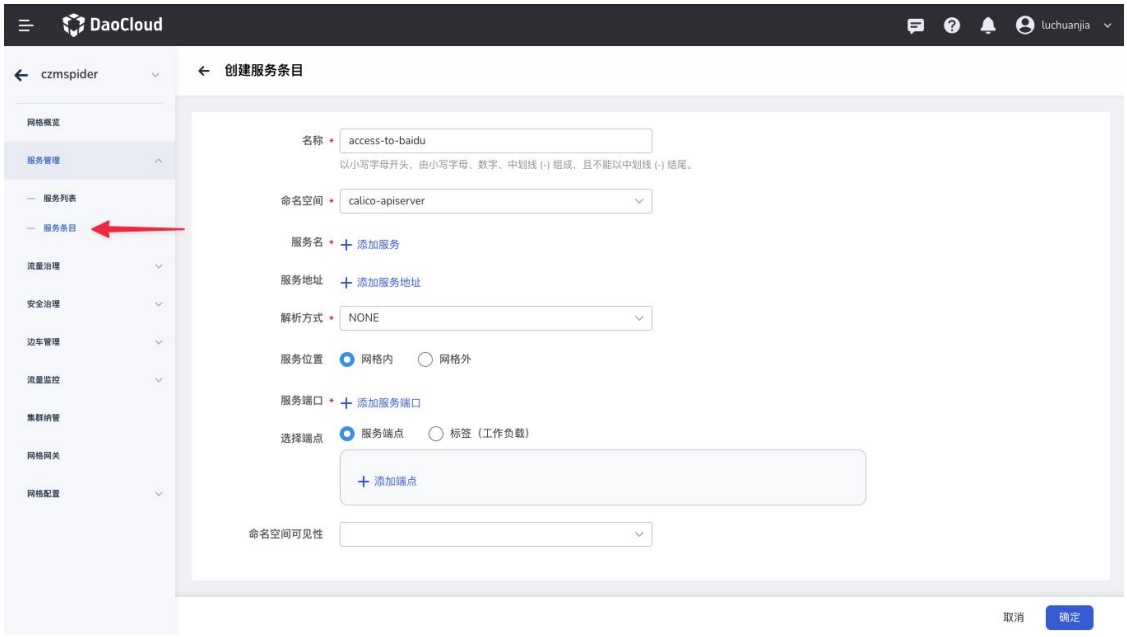
image

规则配置

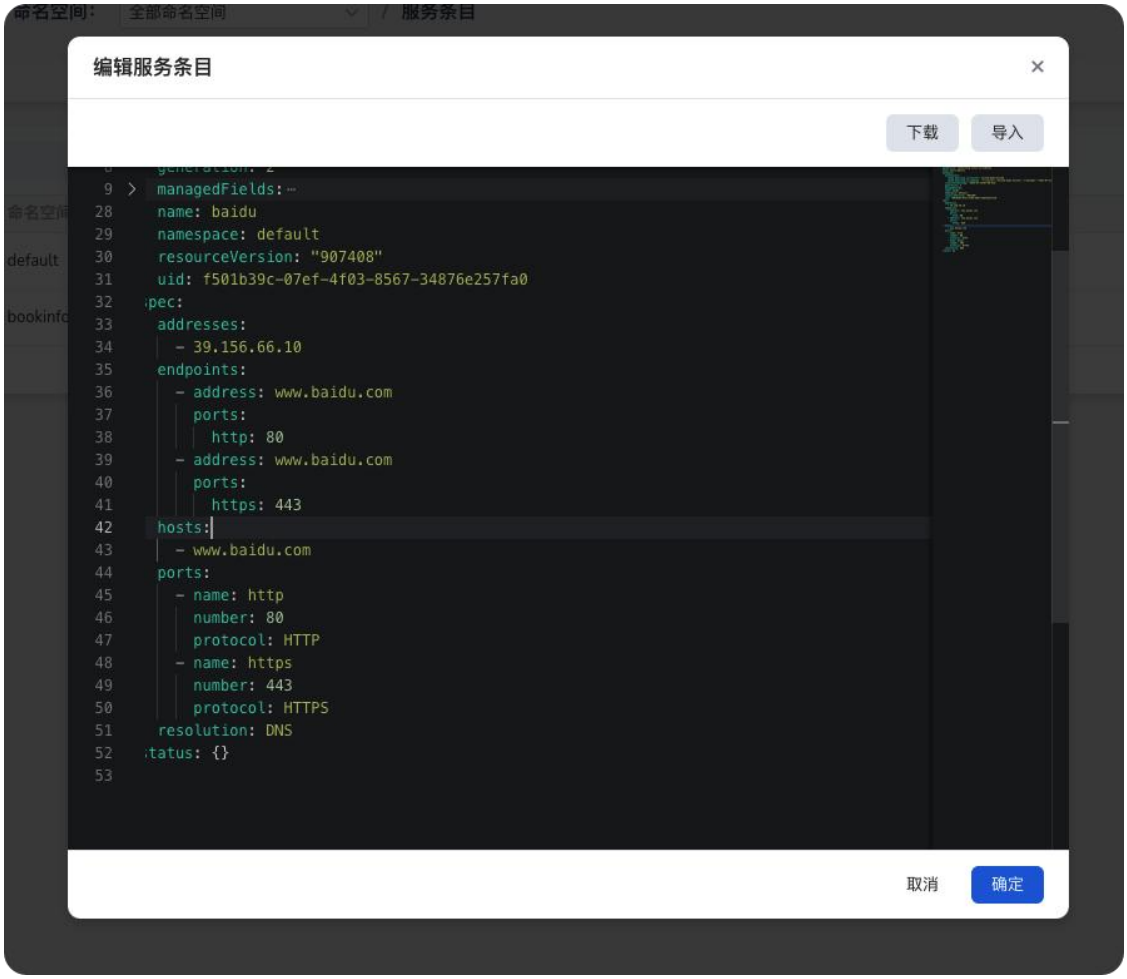
下方规则均给出示例展示，您可以在服务网格的界面中进行创建，用 YAML 的方式展示，方便于理解资源的定义。

创建 Service Entry

这里我们首先创建一个允许的出口访问地址，这里我们使用了 **baidu** 的地址，可以安装如图进行操作

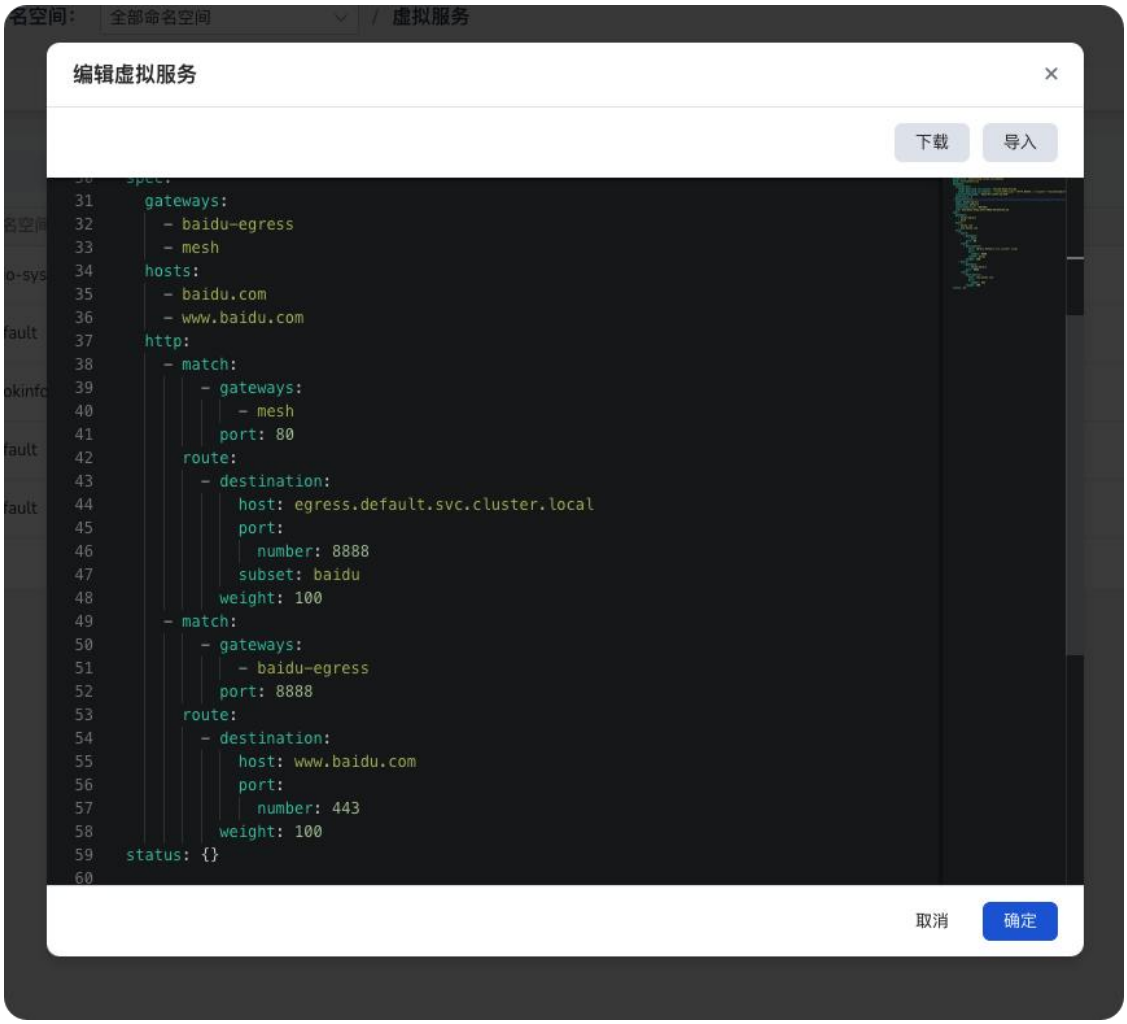

DaoCloud
luchuanjia
czmspider
网格概览
服务管理
服务列表
服务条目
流量治理
安全治理
边车管理
流量监控
集群纳管
网格网关
网格配置
创建服务条目
名称 * access-to-baidu
以小写字母开头，由小写字母、数字、中划线 (-) 组成，且不能以中划线 (-) 结尾。
命名空间 * calico-apiserver
服务名 * 添加服务
服务地址 添加服务地址
解析方式 * NONE
服务位置 网格内 网格外
服务端口 * 添加服务端口
选择端点 服务端点 标签（工作负载）
添加端点
命名空间可见性
取消
确定

image


编辑服务条目
下载
导入
managedFields: …
name: baidu
namespace: default
resourceVersion: "907408"
uid: f501b39c-07ef-4f03-8567-34876e257fa0
spec:
addresses:
- 39.156.66.10
endpoints:
- address: www.baidu.com
ports:
http: 80
- address: www.baidu.com
ports:
https: 443
hosts:
- www.baidu.com
ports:
- name: http
number: 80
protocol: HTTP
- name: https
number: 443
protocol: HTTPS
resolution: DNS
status: {}
取消
确定

image

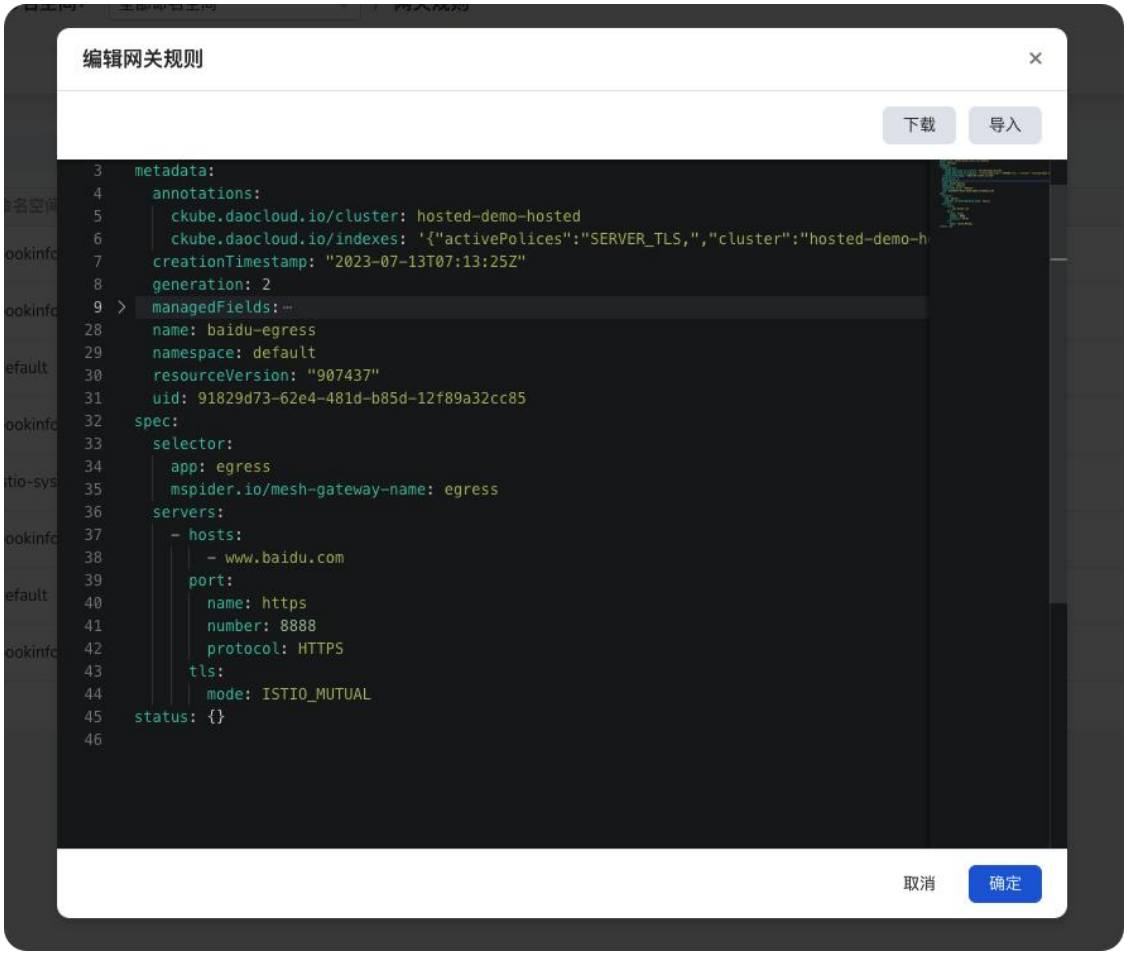
创建 Virtual Service



image

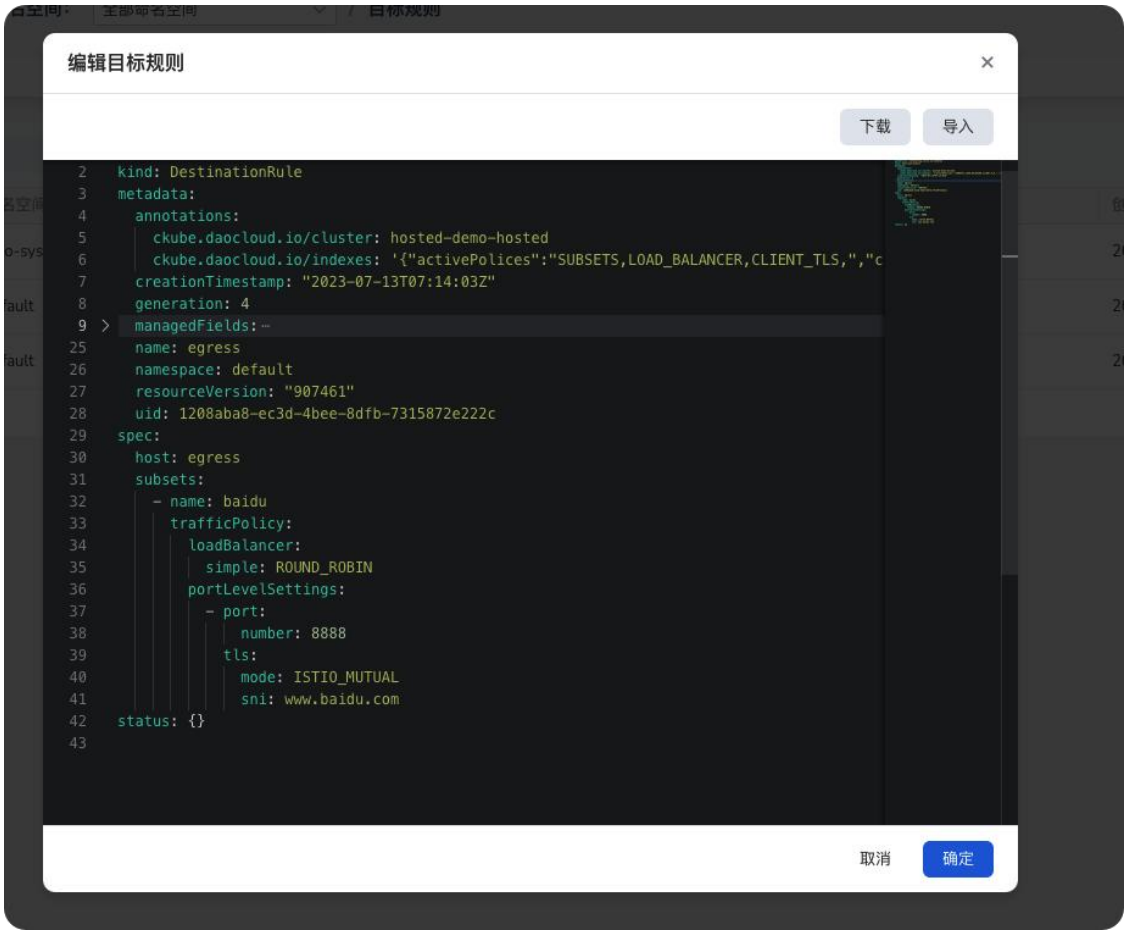
创建 网关规则

注意使用 ISTIO_MUTAL ， 这样才可以用授权策略


编辑网关规则
下载
导入
metadata:
annotations:
ckube.daocloud.io/cluster: hosted-demo-hosted
ckube.daocloud.io/indexes: '{"activePolices":"SERVER_TLS,","cluster":"hosted-demo-h
creationTimestamp: "2023-07-13T07:13:25Z"
generation: 2
managedFields: ⋯
name: baidu-egress
namespace: default
resourceVersion: "907437"
uid: 91829d73-62e4-481d-b85d-12f89a32cc85
spec:
selector:
app: egress
mspider.io/mesh-gateway-name: egress
servers:
- hosts:
- www.baidu.com
port:
name: https
number: 8888
protocol: HTTPS
tls:
mode: ISTIO_MUTUAL
status: {}
取消
确定

image

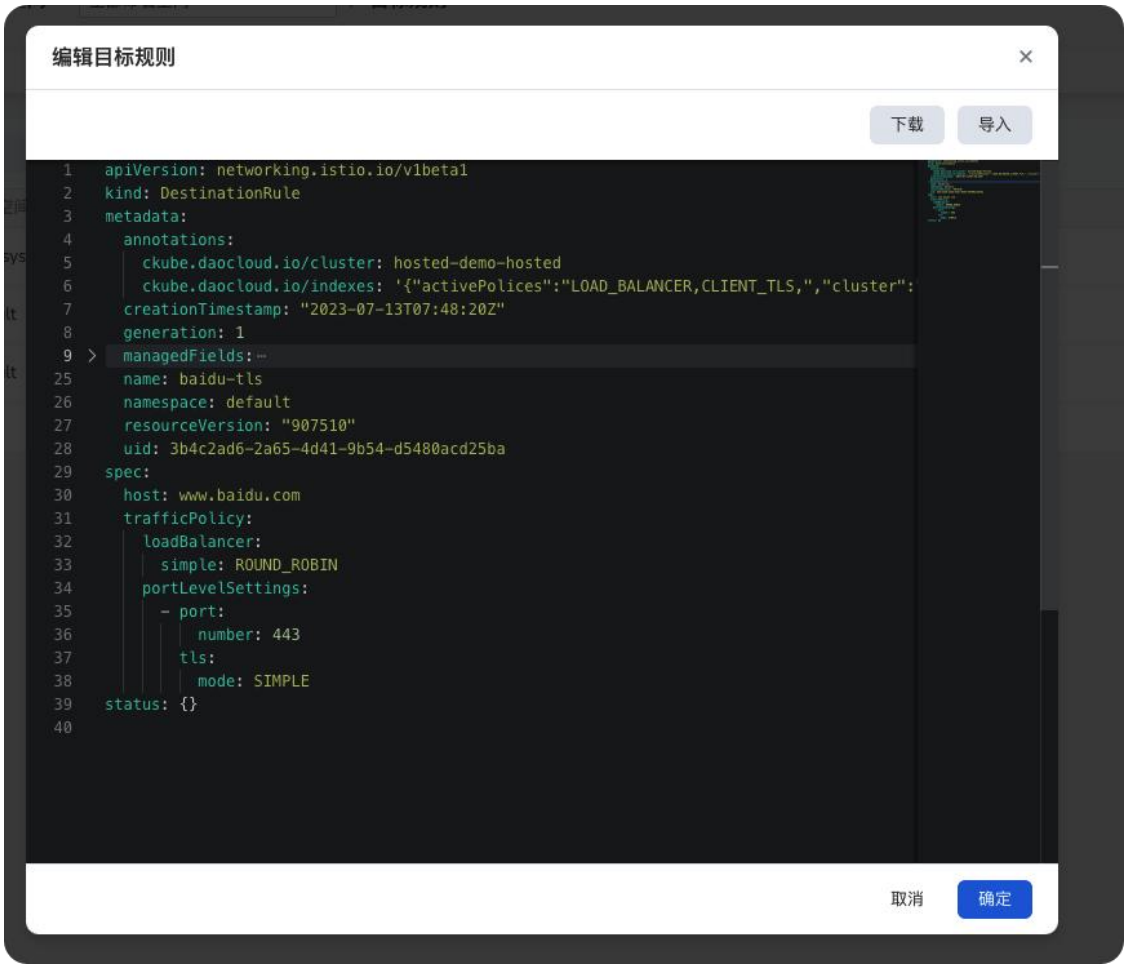
创建网关规则的 DR



image

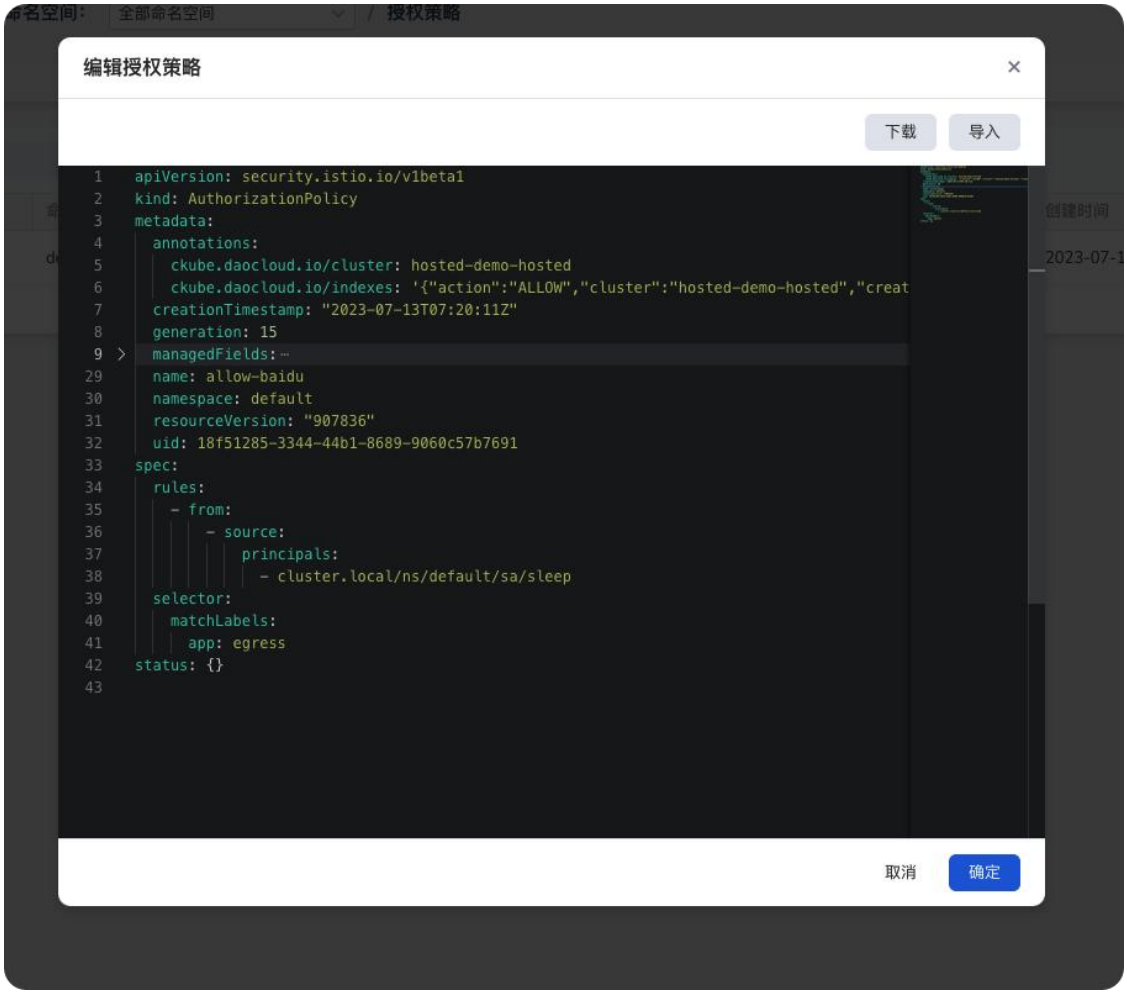
创建 baidu 的 DR

让所有流量走 HTTPS


编辑目标规则
下载
导入
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
annotations:
ckube.daocloud.io/cluster: hosted-demo-hosted
ckube.daocloud.io/indexes: '{"activePolices":"LOAD_BALANCER,CLIENT_TLS,","cluster":
creationTimestamp: "2023-07-13T07:48:20Z"
generation: 1
managedFields:
name: baidu-tls
namespace: default
resourceVersion: "907510"
uid: 3b4c2ad6-2a65-4d41-9b54-d5480acd25ba
spec:
host: www.baidu.com
trafficPolicy:
loadBalancer:
simple: ROUND_ROBIN
portLevelSettings:
- port:
number: 443
tls:
mode: SIMPLE
status: {}
取消
确定

image

启用授权策略



image

功能测试

从示例应用的 Pod 中访问 baidu

可以成功看到访问的结果是正常的，这是因为我们在网格中启用了出口流量，并且限制了来源使用的服务。

```
[root@work-cluster-1-master ~]# kubectl exec -it deploy/sleep -- curl www.baidu.com -v
*   Trying 39.156.66.10:80...
* Connected to www.baidu.com (39.156.66.10) port 80 (#0)
> GET / HTTP/1.1
> Host: www.baidu.com
> User-Agent: curl/7.87.0-DEV
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< accept-ranges: bytes
< cache-control: private, no-cache, no-store, proxy-revalidate, no-transform
< content-length: 2443
< content-type: text/html
< date: Thu, 13 Jul 2023 07:55:33 GMT
< etag: "588603eb-98b"
< last-modified: Mon, 23 Jan 2017 13:23:55 GMT
< pragma: no-cache
< server: envoy
```

image

从其他应用的 Pod 中访问 baidu

此时因为有限定来源服务，所以对于从其他服务发起的咨询，会被拒绝。

```
[root@work-cluster-1-master ~]# kubectl -n bookinfo exec -it ratings-v2-59b9459df8-9p9h7 -- cur
l www.baidu.com -v
* Rebuilt URL to: www.baidu.com/
*   Trying 39.156.66.10...
* TCP_NODELAY set
* Connected to www.baidu.com (39.156.66.10) port 80 (#0)
> GET / HTTP/1.1
> Host: www.baidu.com
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< content-length: 19
< content-type: text/plain
< date: Thu, 13 Jul 2023 07:55:51 GMT
< server: envoy
< x-envoy-upstream-service-time: 5
<
* Curl_http_done: called premature == 0
* Connection #0 to host www.baidu.com left intact
RBAC: access denied[root@work-cluster-1-master ~]#
```

image

2023-08-14

支持接入自定义工作负载类型

DCE 5.0 提供了增强的服务网格模块，能够部署在 Kubernetes 集群中，并且自动纳管集群内的服务，实现流量控制、服务发现、流量监控和故障熔断等功能。

DCE 5.0 服务网格默认支持注入 Deployment、DaemonSet 和 StatefulSet 类型的工作负载，可直接在工作负载页面执行 Sidecar 注入动作，将服务加入到网格中。

然而在实际的生产业务中，可能会因为集群发行版本的问题，导致出现特殊的工作负载类型，传统服务网格的能力对此类特殊工作负载类型无法支持治理能力。

DCE 5.0 服务网格提供的 Istio 版本中已对此能力进行了增强，用户仅需简单的配置即可完成对特殊工作负载类型的治理。

启用特定工作负载

标准方式，通过 helm upgrade 升级的方式，为网格模块增加对应的特殊工作负载类型。

备份参数

```
# 首先，备份现有的网格 Helm 参数配置
helm -n mspider-system get values mspider > mspider.yaml
```

更新配置

编辑配置文件，并追加自定义工作负载类型的配置，如果存在多个配置类型，可以增加多个：

```
global:
  # 以 DeploymentConfig 为例
  # --- add start ---
  custom_workloads:
```

```
    - localized_name: # 中英文必须要写
        en-US: DeploymentConfig
        zh-CN: 自定义工作负载类型
      name: deploymentconfigs
      path:
        pod_template: .spec.template # 定义工作负载 Pod 内容
        replicas: .spec.replicas # 定义工作负载副本数
        status_ready_replicas: .status.availableReplicas # 定义健康的副本数
      resource_schema:
        group: apps.openshift.io # 自定义工作负载的 CRD 的属组
        kind: DeploymentConfig
        resource: deploymentconfigs
        version: v1 # 自定义工作负载的 CRD 的版本
  # --- add end ---
  debug: true
  # ...
```

使用 Helm 更新 mspider ：

```
# 添加 repo，如果不存在的话
helm repo add mspider https://release.daocloud.io/chartrepo/mspider
helm repo update mspider

# 执行更新
export VERSION=$(helm list -n mspider-system | grep "mspider" | awk '{print $NF}')
helm upgrade --install --create-namespace \
-n mspider-system mspider mspider/mspider \
--cleanup-on-fail \
--version=$VERSION \
--set global.imageRegistry=release.daocloud.io/mspider \
-f mspider.yaml
```

示例应用

在 OCP 中，支持一个新的工作负载 DeploymentConfig，本文以此为例演示如何成功支持纳管此工作负载。

DeploymentConfig

```yaml
# filename dc-nginx.yaml
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: nginx-deployment-samzong
spec:
  replicas: 1
  selector:
    app: nginx-app-samzong
  template:
    metadata:
      labels:
        app: nginx-app-samzong
    spec:
      containers:
        - image: nginx:latest
          imagePullPolicy: Always
          name: nginx-samzong
          ports:
            - containerPort: 80
              protocol: TCP
```

使用上面的 yaml 创建一个名为 nginx-deployment-samzong 的应用，然后创建关联的 svc：

```yaml
# filename dc-nginx-svc.yaml
apiVersion: v1
kind: Service
```

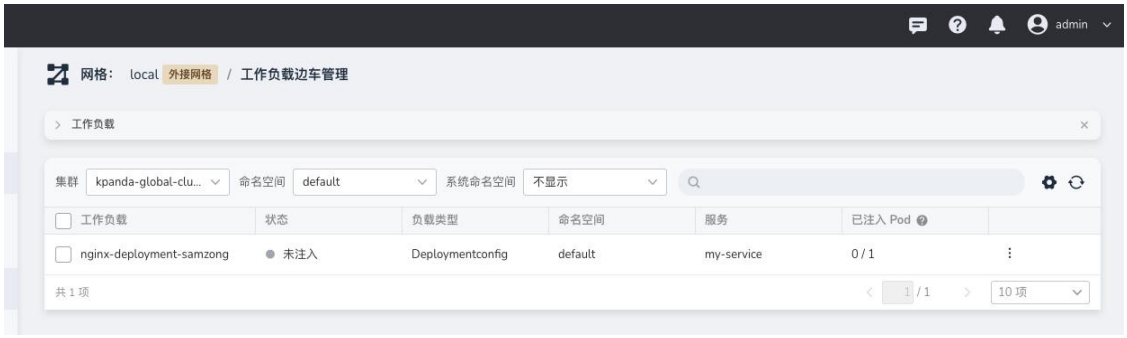
```
metadata:
  name: my-service
spec:
  selector:
    app: nginx-app-samzong
  ports:
    port: 80
    protocol: TCP
    targetPort: 80
```

这是一个标准的 Kubernetes 服务，我们通过 app: nginx-app-samzong 来绑定到预先创建的 DeploymentConfig。

```
kubectl -n NS_NAME apply -f dc-nginx.yaml dc-nginx-svc.yaml
```

效果

在工作负载成功启动之后，可以在边车管理中查看工作负载。默认为未注入，我们可以手工注入。



image

在服务列表可以看到对应的服务，此时服务的工作负载也是正常在运行的，我们可以增加对应的策略来提供对服务的访问。



image

结语

通过上面的操作，我们可以看到，通过简单的配置，就可以支持自定义的工作负载类型。 这样就可以支持更多的工作负载类型，让用户可以更加灵活地使用 DCE 5.0 服务网格。

Tags

Following is a list of relevant tags:

TAGS