d.run 让算力更自由

d.run (DaoCloud Runs Intelligence)，揭示一个新一代软件体系下的全新算力世界，让算力更自由。

DaoCloud

DaoCloud Inc.

1. d.run 文档

1.1 d.run 简介

1.1.1 d.run 介绍

d.run 是 DaoCloud 自研的 AIGC 综合性算力运维和运营平台，整合云原生调度能力，一站式管理算力集群资源， 内置算法开发、模型中心和模型微调，轻松创建智能问答提供生成式智能聊天服务，管理用户反馈。

「DaoCloud 道客」作为算力服务提供商和专业技术支持方，与各大政企、科研机构和运营商等智算中心建设方与运营方紧密合作， 在上海、合肥、天津、香港等多个地区建设算力枢纽中心，实现算力资源跨行业、跨地区、跨层级地自由流动，并通过构建 AI 开放生态平台， 集成丰富的模型套件工具，全面推动算力与应用的协同创新，为所有用户提供高效、可靠、易得的 AI 算力云服务，加速 AI 普惠。

点击注册 **d.run**

产品全景图

d.run 作为算力一体化解决方案，能够打造千卡、万卡等大规模高速互联的算力网络，通过性能优化的 DaoCloud Enterprise AI 调度异构算力， 基于多种模型套件加速 AI 应用创新，配合一系列运维和运营服务，全方位保障算力的稳定、可靠。

d.run 面向用户提供功能丰富、易用性好、性能高效的模型套件，集成数据准备、模型开发、模型训练、模型部署等各个环节所需的便捷实用的生态产品和开源工具， 支持高效的模型开发与训练，帮助用户轻松将 AI 技术应用到实际业务中，赋能产业升级。

- 全面的 **AI** 开发与生产工具 集成了从数据的获取、探索，到模型的开发、训练、部署、微调、管理等丰富的模型套件工具选项， 可低成本、高效率地应用 AI 技术，加速产品和业务的创新。
- 便捷的数据管理与准备工具 提供对接各类数据源、统一管理数据集、可视化数据探索、数据清洗、数据增强等工具， 最大程度提升数据科学家和算法工程师的工作效率，使其更专注于模型开发。
- 高效的模型开发与训练支持 能够提供易用的集成开发环境，支持交互式开发和拖拉拽的可视化建模， 通过分布式训练架构提升训练效率，降低算力成本，实现模型快速高效部署，打通开发训练和生产环境。

d.run 优势

- 灵活的资源调度 ：提供灵活的计算、网络、存储资源调度方案，有效提高资源利用率，用户可按需使用算力资源，显著降低算力使用成本。
- 优异的处理能力 ：提供业界领先的算力性能，软硬协同优化，通过 GPU 共享和加速显著缩短训练时间，提高算法的准确性
- 功能丰富的模型套件 ：通过 AI 生态开放平台提供各类便捷实用、性能高效的模型套件，为模型开发与训练提供支持，加速 AI 应用落地及创新。
- 全方位的运维和运营服务 ：提供优质的算力运维和运营服务，实现软硬件一站式的性能调优，配合高效、可视化的平台管理，为企业的算力使用保驾护航。

d.run 的这些优势来自于丰富的产品模块和功能特性：

d.run 新手尝鲜步骤

一名新用户注册 d.run 之后，首先需要购买算力，支持的范围囊括了单卡、双卡...千卡、万卡。 您可以随时检查自己购买的资源，查阅历史购买记录。

您的订单生效后，d.run 后台会自动创建算力集群。 您可以在全局管理 中设置用户、用户组、工作空间和平台个性化设置。

在模型中心，您可以一键部署高可用的大语言模型服务，通过统一的模型推理接口进行分布式推理。 与智能问答和流程编排模块紧密集成，模型中心支持行业内外流行的各种大模型。 您可以轻松接入各类在线模型 API Key，并根据需求灵活切换本地和在线模型服务。

接下来使用 DataTunerX 进行模型微调，尝试微调数据集、超参组，实现模型微调、模型评估和模型推理全生命周期的自动化处理。 通过高效利用算力集群中的底层分布式算力资源，DataTunerX 能够进行矩阵式的模型微调实验，从而推动大型模型的敏捷和自动化迭代。

微调好模型后，AI流程引擎可以通过智能化的流程创建文生文、文生图、图生图、文生视频等智能问答。 您可以随时调整智能问答所调用的语料库和占用的 GPU 等资源，还能随时了解用户对智能问答的反馈并迅速做出响应。

通过 AI 流程可以为各行业、各场景分门别类地生成不同智能问答， 一站式管理智能问答的上下层、前后端设施，把控用户需求，洞察前沿趋势。

参见演示视频：

行业背景

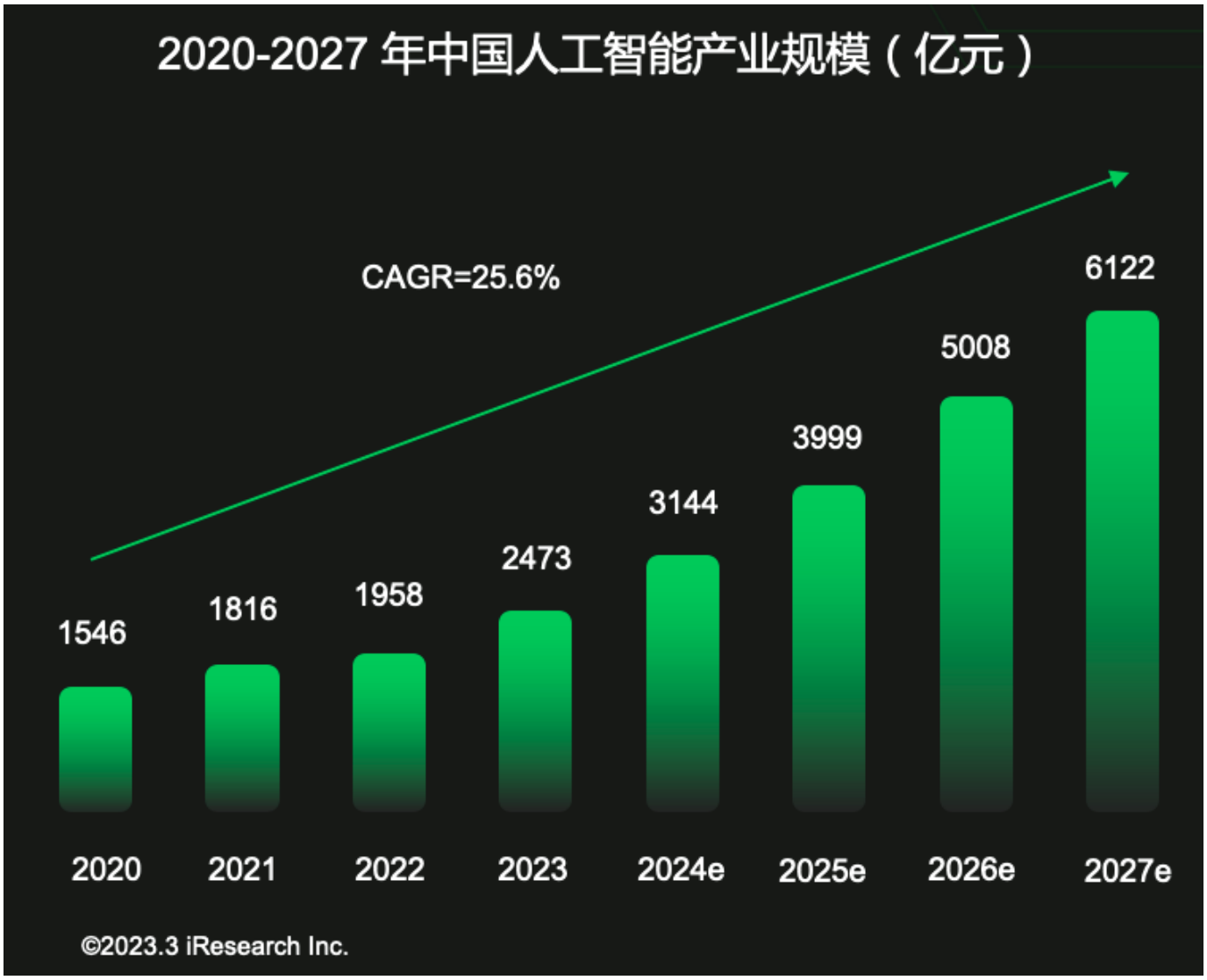
全球人工智能行业大爆发，对智能算力提出了指数级的增长要求。

> **SemiAnalysis**
>
> GPT-4 在大约 2.5 万个 A100 （GPU 利用率 32%-36%）上训练 90-100 天，单次训练成本 6300 万美元。

> 埃隆·马斯克
>
> GPT-5 的训练可能需要 3-5 万块 H100。

在国内，算力成为新质生产力基建热点，从中央到地方各层级政策密集出台。

- 2024.2 国务院国资委明确央企要把发展 AI 放在全局工作中统筹谋划，加快建设一批智能算力中心
- 人民银行、工信部等 6 部门联合发文：鼓励金融机构加大对绿色低碳算力基础设施的信贷支持力度
- 北京：“十四五 期间 形成规模化先进算力供给能力，支撑千亿级参数量的大型语言模型、大型视觉模型、多模态大模型、科学计算大模型...
- 上海：到 2024 年，本市数据中心算力供给呈现以智算算力等高性能算力为主的多元算力协同体系，总算力超 15EFLOPS，高性能算力占比达到 35%
- 广东：到 2025年，智能算力规模实现全国第一、全球领先，通用人工智能技术创新体系较为完备，人工智能高水平应用场景进一步拓展，核心产业规模突破3000 亿元，企业数量超 2000 家
- 江苏：到 2027 年底，全省 算力规模达到 70 万核 CPU 和 400PB 存储、新增高性能算力持续增长
- 浙江：到 2024 年形成 10 项以上人工智能重大科技成果，获得 1000 项以上核心发明专利；打造 3-4 个千亿级人工智能产业集群，产业营业收入年均增长 15% 以上
- 河南：到 2025 年 智算和超算算力规模超过 2000PFLOPS，高性能算力占比超过 30%
- 山东：到 2025 年，全省 数据中心 在用标准机架数达到 45 万个以上，平均利用率提升到 60% 以上
- 湖南：到 2027 年长沙人工智能创新中心、马栏山视频超算中心等加快建设，建成和在建规模以上数据中心 51 个，标准机架 17.2 万个
- 成都：到 2025 年，全市人工智能产业产值突破 1500 亿元，人工智能创新应用场景数量达到 100 个以上
- 贵州：2023-2025 年贵州省通用算力、智算算力、超算算力的总规模分别达 2Eflops、5Eflops 和 10Eflops



1.1.2 快速入门

本页演示注册 d.run 账号后，如何免费体验、用一元体验 d.run 部分功能。

注册账号

点击注册 **d.run**

您通过邮箱注册账号之后，d.run 系统会发送一封邮件，请点击邮件中的链接激活账号。 更多细节，请参阅以下演示视频。

您在成功激活账号后，d.run 系统会为您自动分配以下资源：

1. 一个默认的工作空间
2. 一个向量化的模型服务：bge-large-zh
3. 一个默认的大模型：chatglm3-6b
4. 一个模型训练队列

免费体验

您可以通过使用上述几个自动分配的资源来免费体验 d.run 的部分功能：

- 场景 1：使用本地模型服务创建 RAG 应用
- 场景 2：使用在线模型服务创建 RAG 应用
- 场景 3：使用流程引擎创建更灵活的应用

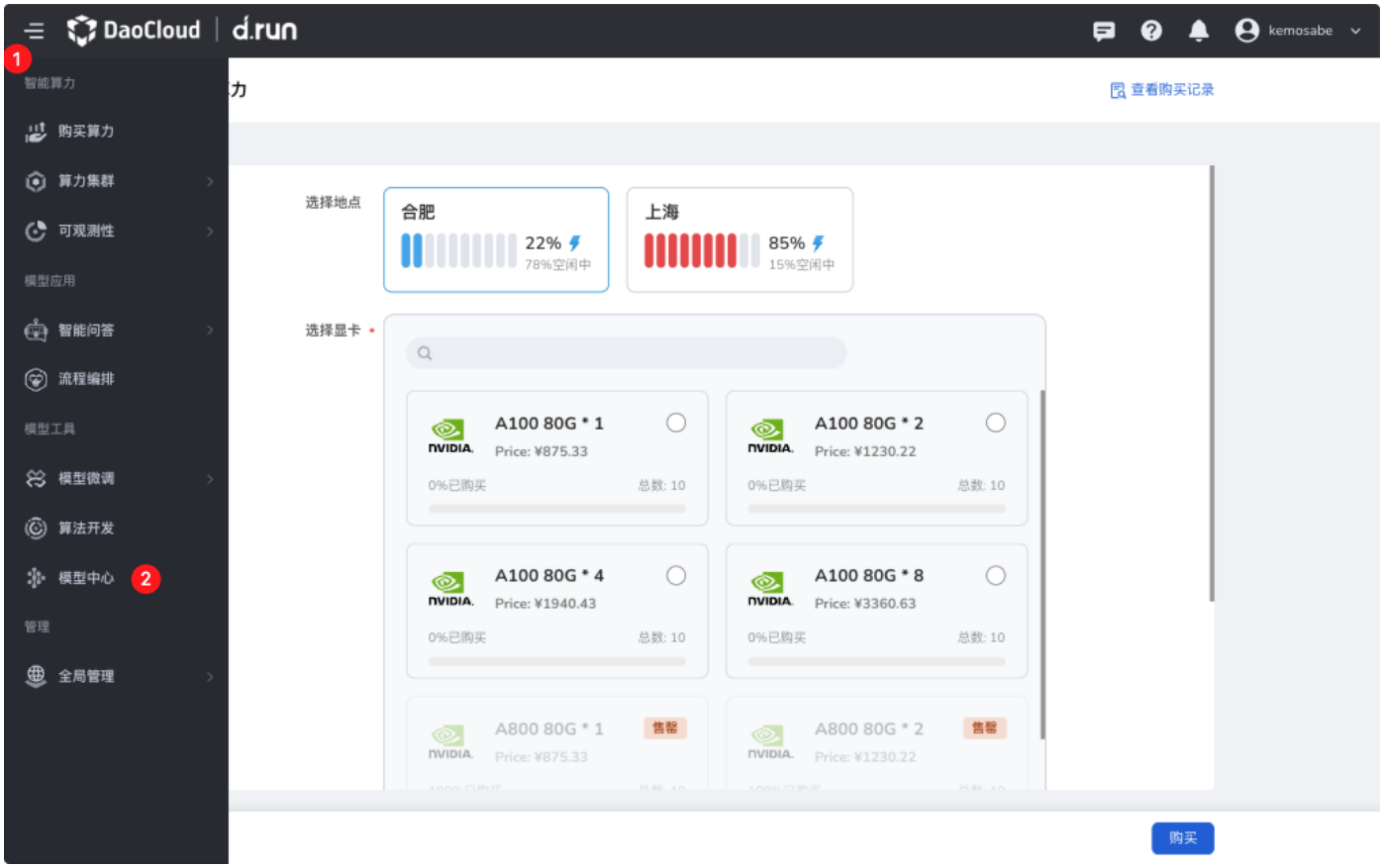
使用本地模型创建 RAG 应用

Tip

RAG（Retrieval Augmented Generation，检索增强生成）指的是通过检索获取相关的知识并将其融入提示词，让大模型能够参考相应的知识从而给出合理的回答。

d.run 在免费模式下就可以通过图形界面，使用本地模型来创建一个 RAG 聊天应用：

1. 点击左上角的 ≡ 打开导航栏，进入 模型中心



d.run 默认分配了 2 个模型：

- chatglm3-6b 是一个可以部署到消费级显卡上的本地模型，支持中英双语对话，基于通用语言模型 (GLM) 架构，具有 62 亿参数，最低只需 6GB 显存。
- bge-large-zh 是一个向量化模型，能够将文本转换为向量，适合推理和微调。

2. 准备语料

语料是模型训练的基础，我们先准备一个 CSV 文件。第一列是问题，第二列是答案：

```
#,问题,答案
1,什么是边缘原生应用准则白皮书？,边缘原生应用准则白皮书是物联网边缘工作组（IOT Edge Working Group）发布的一个文件，探索边缘原生的定义，以及“云原生”和“边缘原生”之间的异同。
2,DCE 5.0 社区版包含哪些开源项目？,"DCE 社区版包含的开源软件有：\n\ncloudtty：Kubernetes 网页版控制台，易于使用\nClusterpedia：Kubernetes 多集群资源百科全书，已入选 CNCF 沙箱孵化，中国移动等已部署至生产\nFerry：Kubernetes 多集群通信组件，消除多集群复杂度\nHwameiStor：高可用的本地存储方案，更快、更强、更可靠，已入选 CNCF 全景图，正在申请 Sandbox\nKLTS：对 k8s（最新版本 - 0.03）等 10 多个版本的持续维护\nKubean：容器化集群的全生命周期管理工具，正在申请 Landscape\nKWOK：模拟成千上万的 kubelet\nMerbridge：使用 eBPF 加速服务网格，已入选 CNCF 全景图和 Sandbox\nSpiderpool：云原生网络 IPAM 自动化管理软件，建议作为插件用于 Underlay CNI，正在申请 Landscape 和 Sandbox\n公开镜像加速：加速国外镜像的下载\n以上只是 ""DaoCloud 道客"" 技术的冰山一角，还有更多项目。"
3,KWOK 是什么？,KWOK 是 Kubernetes WithOut Kubelet 的缩写，帮助用户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。
4,Karmada 是用于什么场景的项目？,Karmada 是用于多云和混合云场景中的项目，可以实现应用跨数据中心、跨可用区和跨集群高可用。
```

Tip

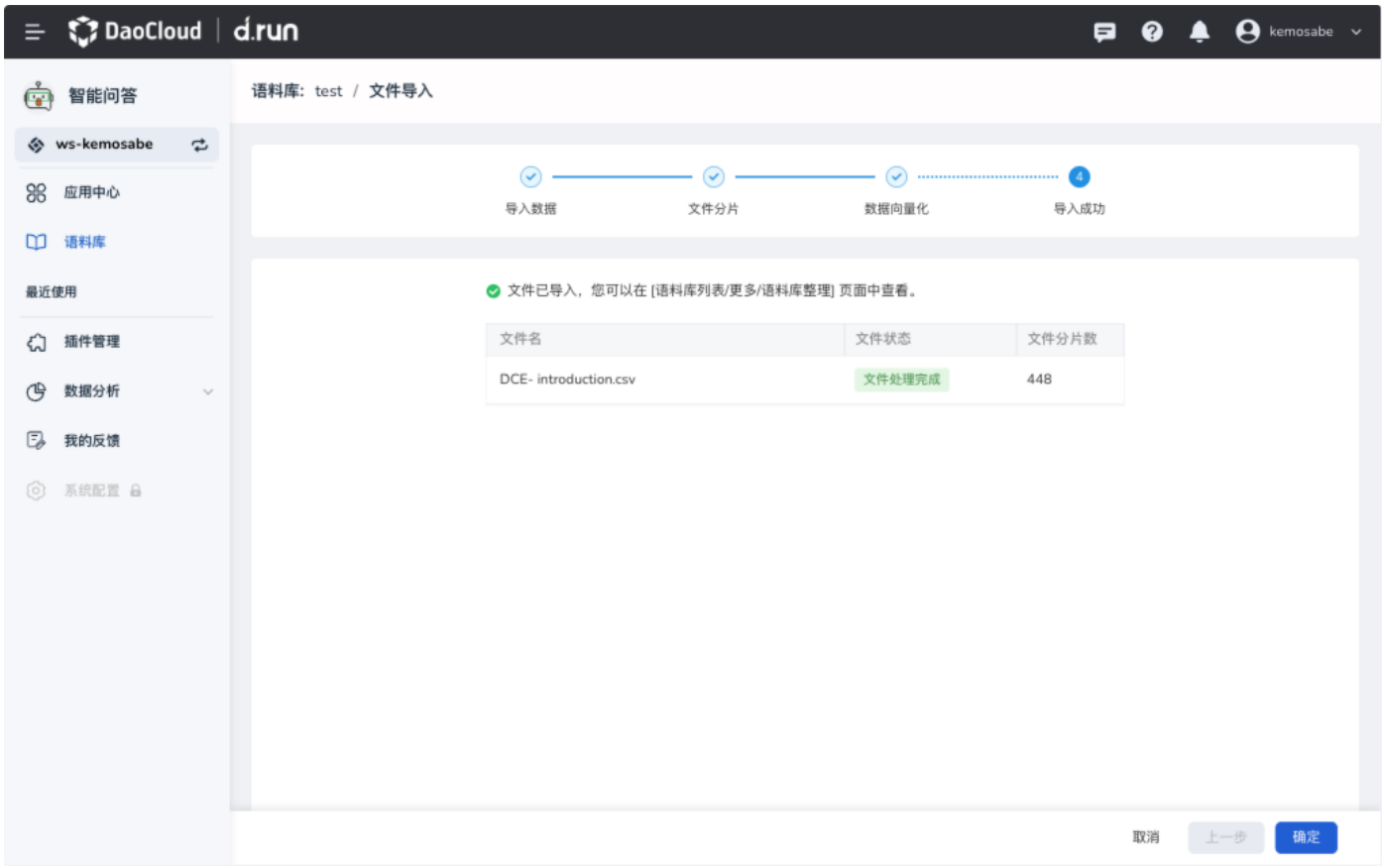
点击 DCE-introduction.csv 可下载这个 CSV 文件。

3. 创建和导入语料库

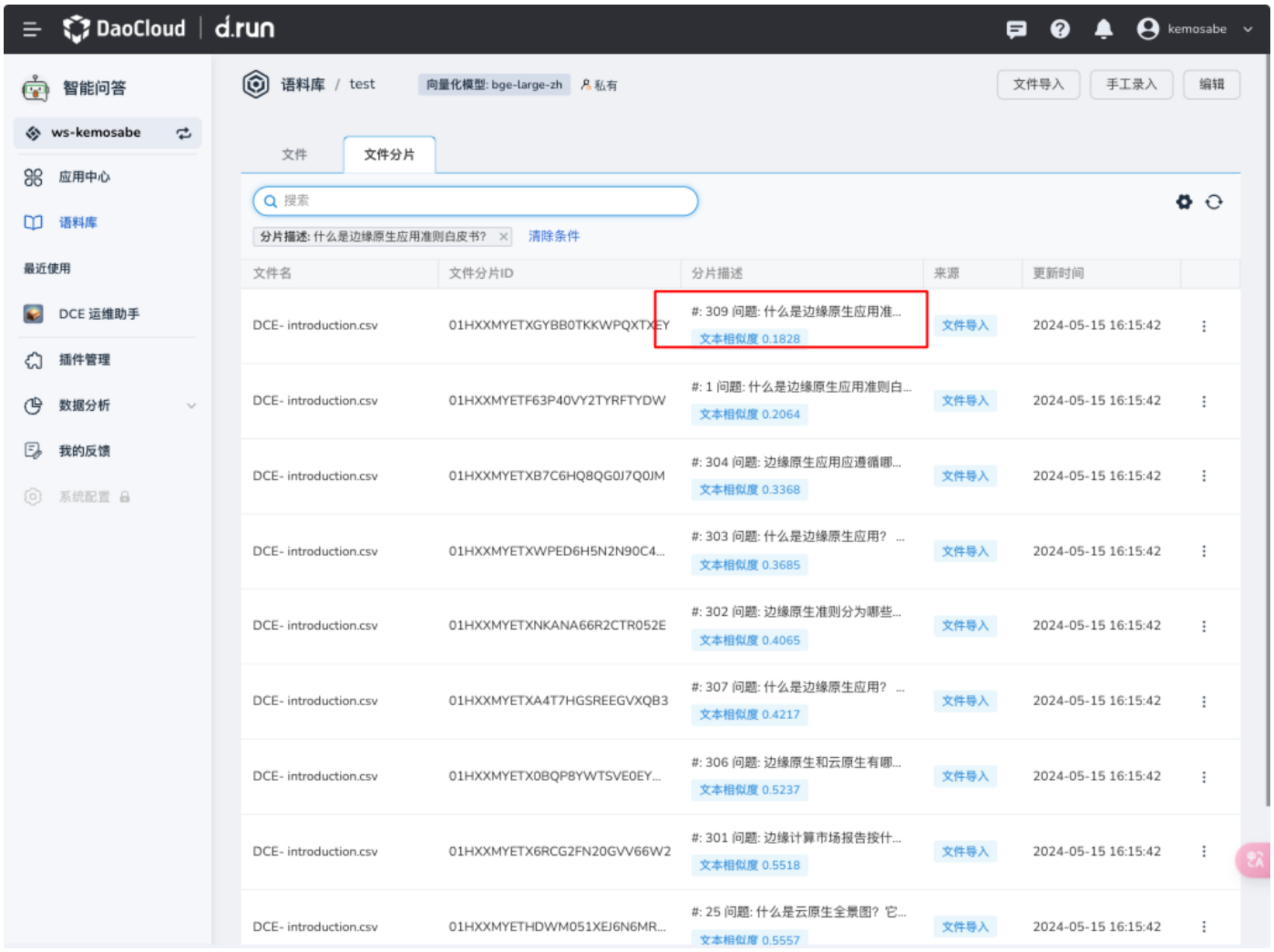
a. 点击左上角的 ≡ 打开导航栏，点击 智能问答 -> 语料库 -> 创建语料 ，填写各项参数后，点击 确定

b. 返回语料库列表，点击刚创建的语料库名称，进入语料库详情页，点击右上角的 文件导入

c. 选择准备好的 CSV 文件，按照向导页面，导入语料文件。



d. 返回语料库列表，点击语料名称进入详情页，选择 文件分片 页签，点击搜索栏，选择 分片描述， 输入一个问题进行搜索，例如 `什么是边缘原生应用准则白皮书?`， 可以看到搜索结果和对应的相似度。


DaoCloud | d.run
kemosabe
智能问答
ws-kemosabe
应用中心
语料库
最近使用
DCE 运维助手
插件管理
数据分析
我的反馈
系统配置
语料库 / test
向量化模型: bge-large-zh
私有
文件导入
手工录入
编辑
文件
文件分片
搜索
分片描述: 什么是边缘原生应用准则白皮书?
清除条件
文件名
文件分片ID
分片描述
来源
更新时间
DCE- introduction.csv
01HXXMYETXGYBB0TKKWPQXTXEY
#: 309 问题: 什么是边缘原生应用准...
文本相似度 0.1828
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETF63P40VY2TYRFTYDW
#: 1 问题: 什么是边缘原生应用准则白...
文本相似度 0.2064
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETXB7C6HQ8QG0J7Q0JM
#: 304 问题: 边缘原生应用应遵循哪...
文本相似度 0.3368
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETXWPED6H5N2N90C4...
#: 303 问题: 什么是边缘原生应用? ...
文本相似度 0.3685
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETXNKANA66R2CTR052E
#: 302 问题: 边缘原生准则分为哪些...
文本相似度 0.4065
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETXA4T7HGSREEGVXQB3
#: 307 问题: 什么是边缘原生应用? ...
文本相似度 0.4217
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETX0BQP8YWTSVE0EY...
#: 306 问题: 边缘原生和云原生有哪...
文本相似度 0.5237
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETX6RCG2FN20GVV66W2
#: 301 问题: 边缘计算市场报告按什...
文本相似度 0.5518
文件导入
2024-05-15 16:15:42
DCE- introduction.csv
01HXXMYETHDWM051XEJ6N6MR...
#: 25 问题: 什么是云原生全景图? 它...
文本相似度 0.5557
文件导入
2024-05-15 16:15:42

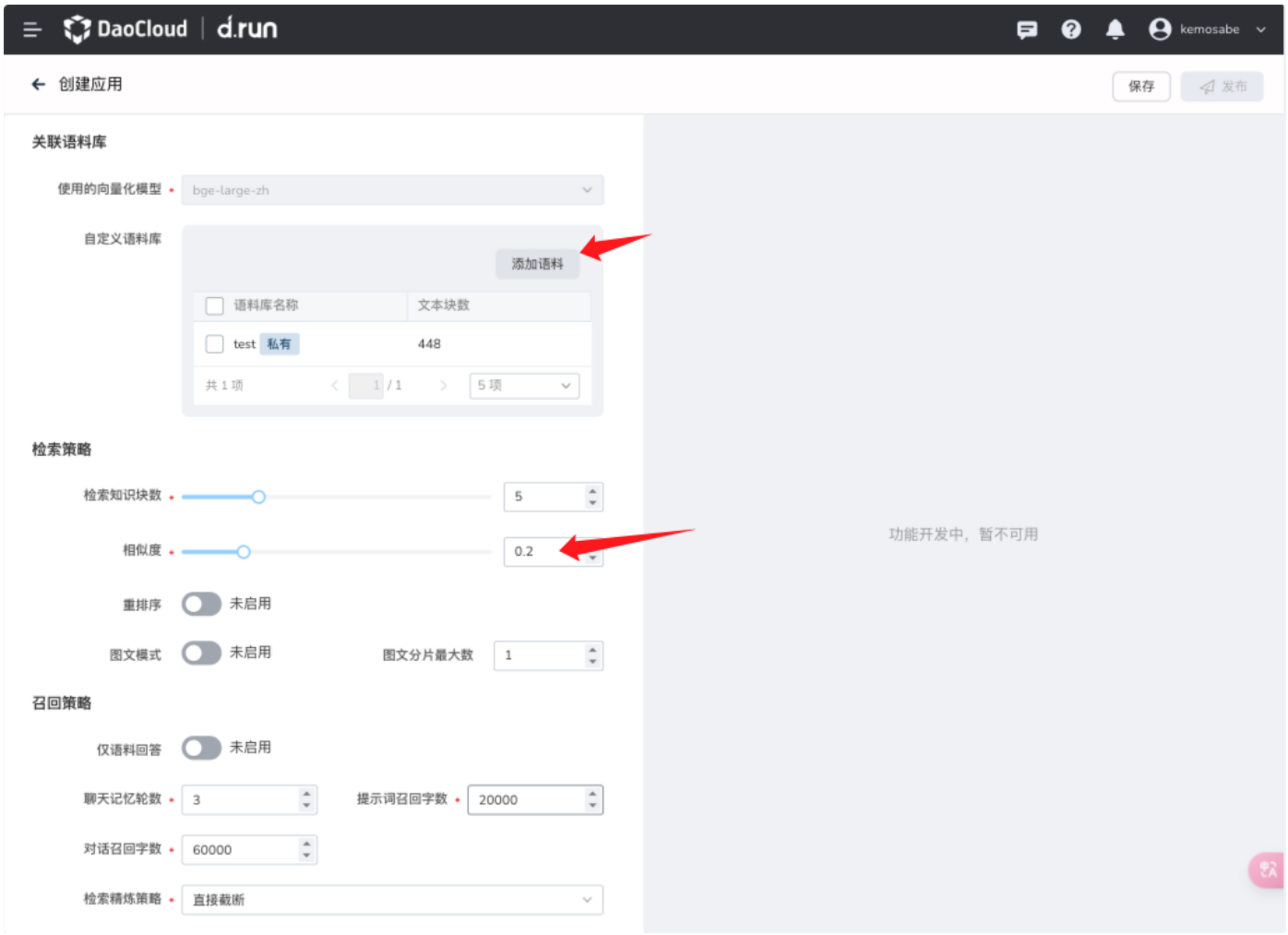
4. 创建聊天应用

a. 点击左上角的 ≡ 打开导航栏，进入 应用中心 ，点击右侧的 创建 按钮

b. 配置好表单的各项参数后，点击右上角的 保存 、 发布 按钮

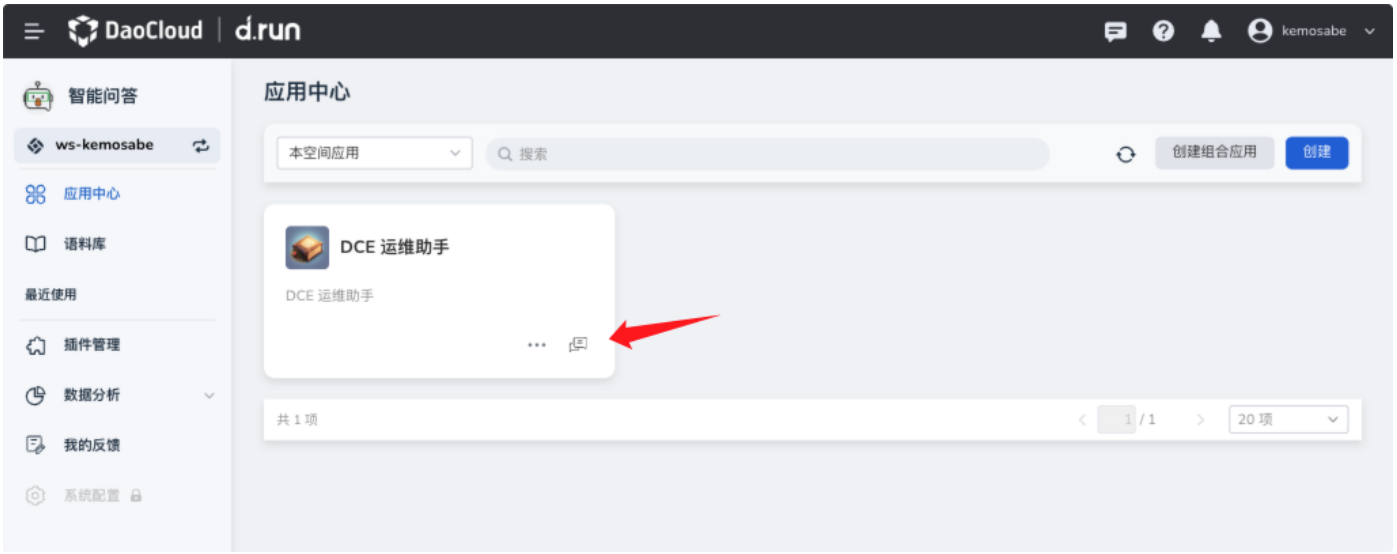
- 例如我们将此应用命名为 **DCE** 运维助手
- 选择本地模型服务 `chatglm3-6b`，选择向量化模型服务 `bge-large-zh`，
- 根据应用场景修改提示词，关联创建的语料库，修改相似度为 `0.2`


DaoCloud | d.run
kemosabe
创建应用
保存
发布
基础设置
名称 *
DCE 运维助手
介绍 *
DCE 运维助手
AI 配置
大语言模型服务 *
本地模型服务
在线模型服务
chatglm3-6b
随机度 *
0.01
严谨
发散
向量化模型 *
bge-large-z...
bge-large-zh
语料库提示词模版
1.回答内容必须在"{corpus_search_content}"中。
2.如果问题在所提供的资料信息内无法找到，你会回答:"抱歉,资料库已知的信息中，没有找到您需要的结果,请尝试修改引用的资料库或使用模型自有能力回答。"
{user_inputs_content}
嵌入提示词模版
{user_inputs_content}
功能开发中，暂不可用

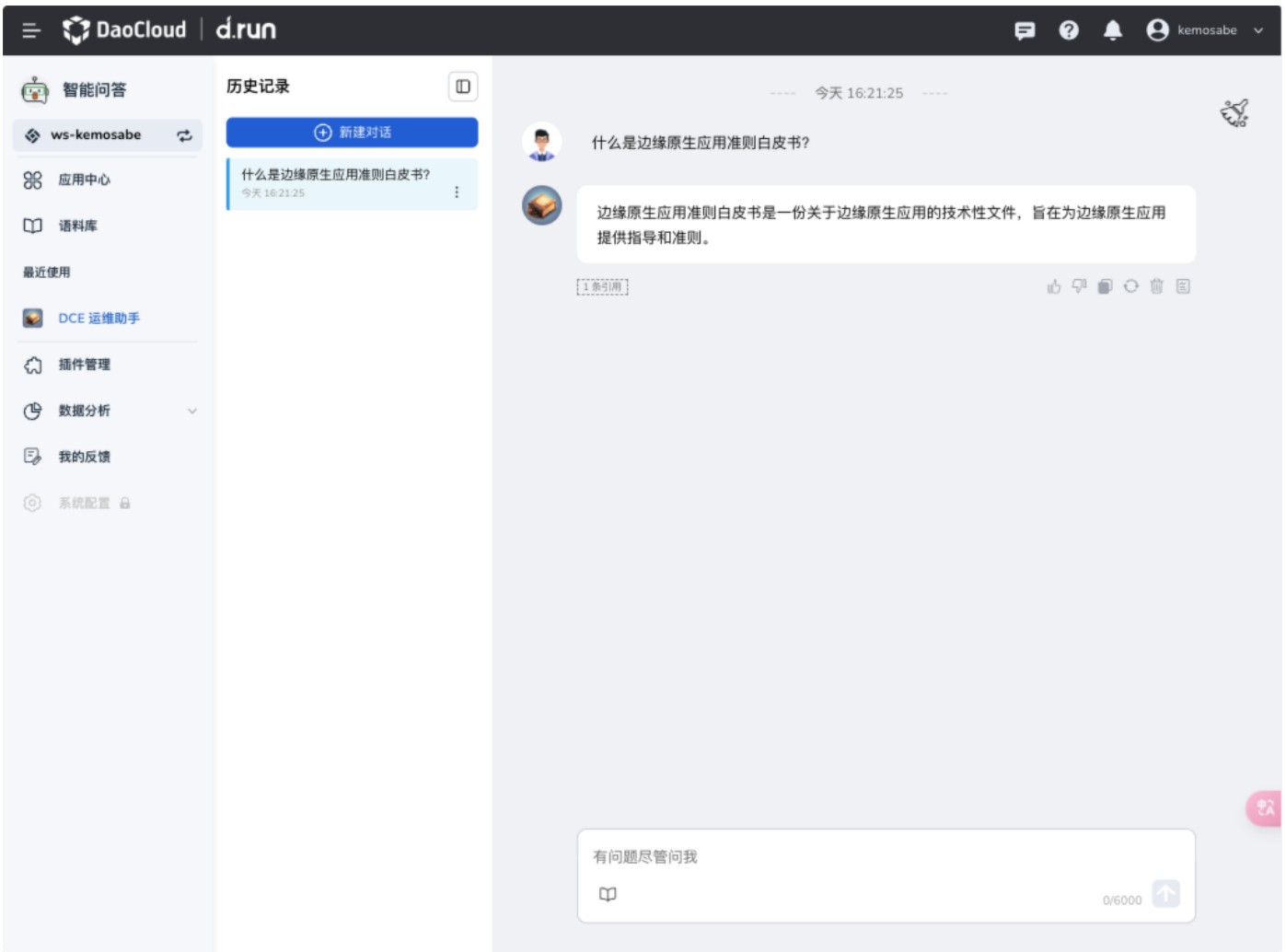

DaoCloud | d.run
kemosabe
创建应用
保存
发布
关联语料库
使用的向量化模型 *
bge-large-zh
自定义语料库
添加语料
语料库名称
文本块数
test
私有
448
共 1 项
1 / 1
5 项
检索策略
检索知识块数 *
5
相似度 *
0.2
重排序
未启用
图文模式
未启用
图文分片最大数
1
召回策略
仅语料回答
未启用
聊天记忆轮数 *
3
提示词召回字数 *
20000
对话召回字数 *
60000
检索精炼策略 *
直接截断
功能开发中，暂不可用

5. 开始对话

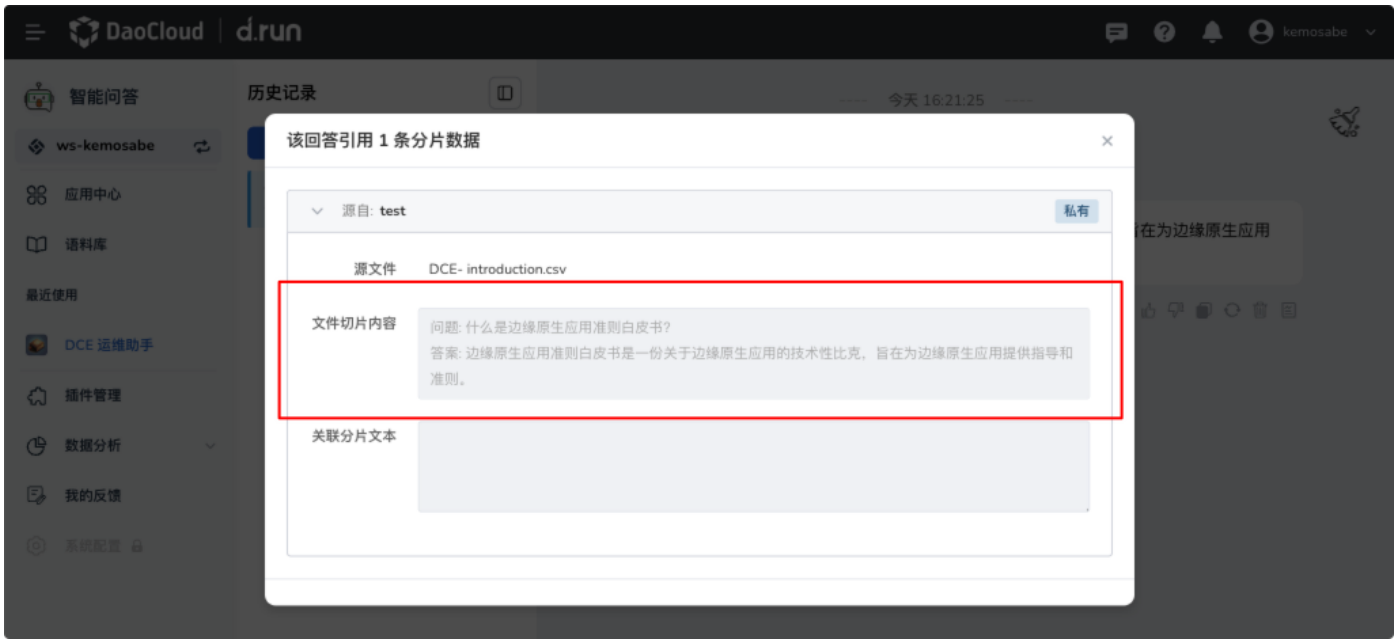
a. 在应用中心，点击应用卡片右下角的 对话 图标



b. 在输入框中输入问题，按下回车键，就会根据刚导入的语料库给出答案



c. 如果你是管理员，在每条回答下方，还可以看到 `xxx 条引用` 的提示。 点击该文字提示，可以看到引用了语料库中的哪些数据。 如果发现引用的数据有问题，可以在 语料库 -> 语料库详情 -> 文件切片 中找到并修改对应的语料。

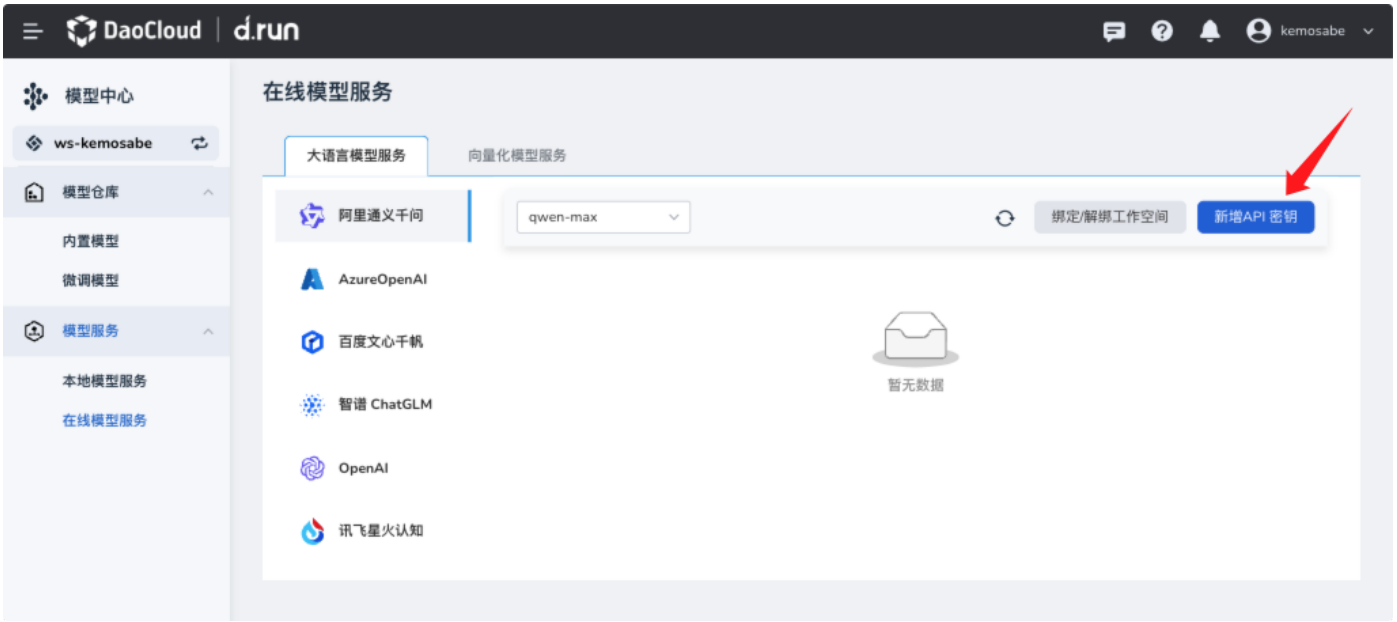


自此，这个名为 **DCE** 运维助手 的 RAG 聊天应用已被成功创建。

使用在线模型创建 RAG 应用

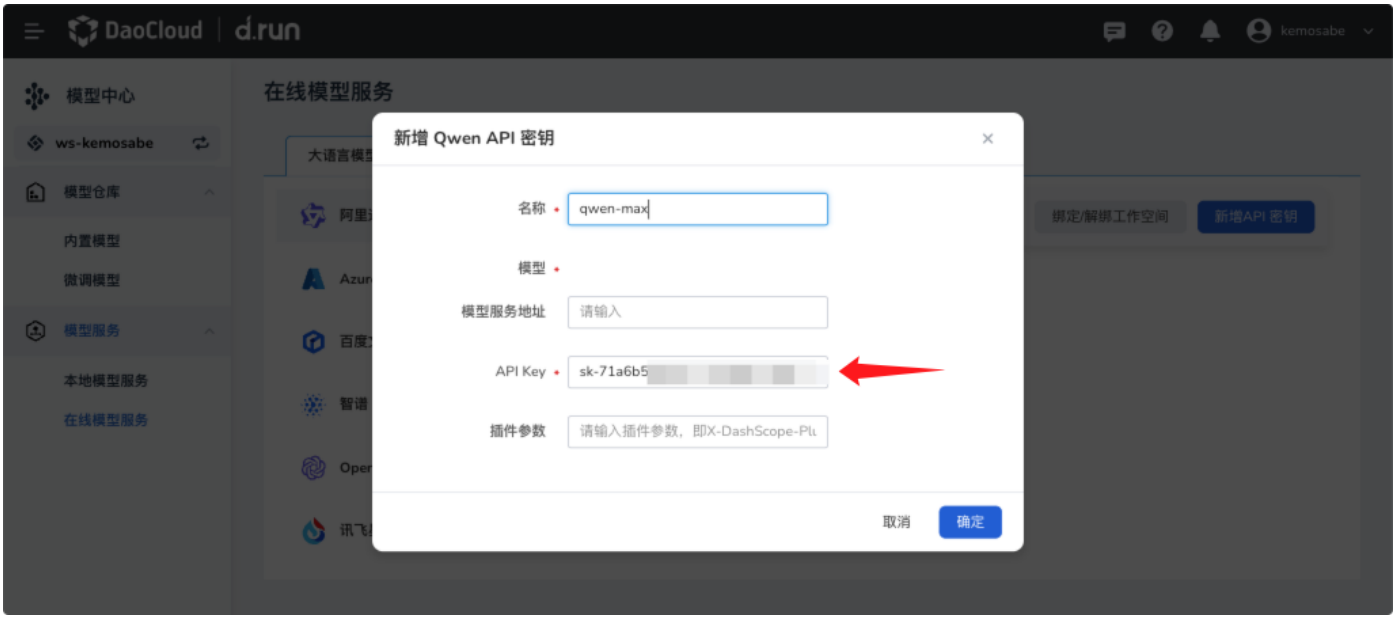
除了场景 1 所述的本地模型外，d.run 默认还提供了在线模型服务。

1. 点击左上角的 ≡ 打开导航栏，进入 模型中心 ，依次选择 模型服务 -> 在线模型服务 -> 新增 **API** 密钥 按钮



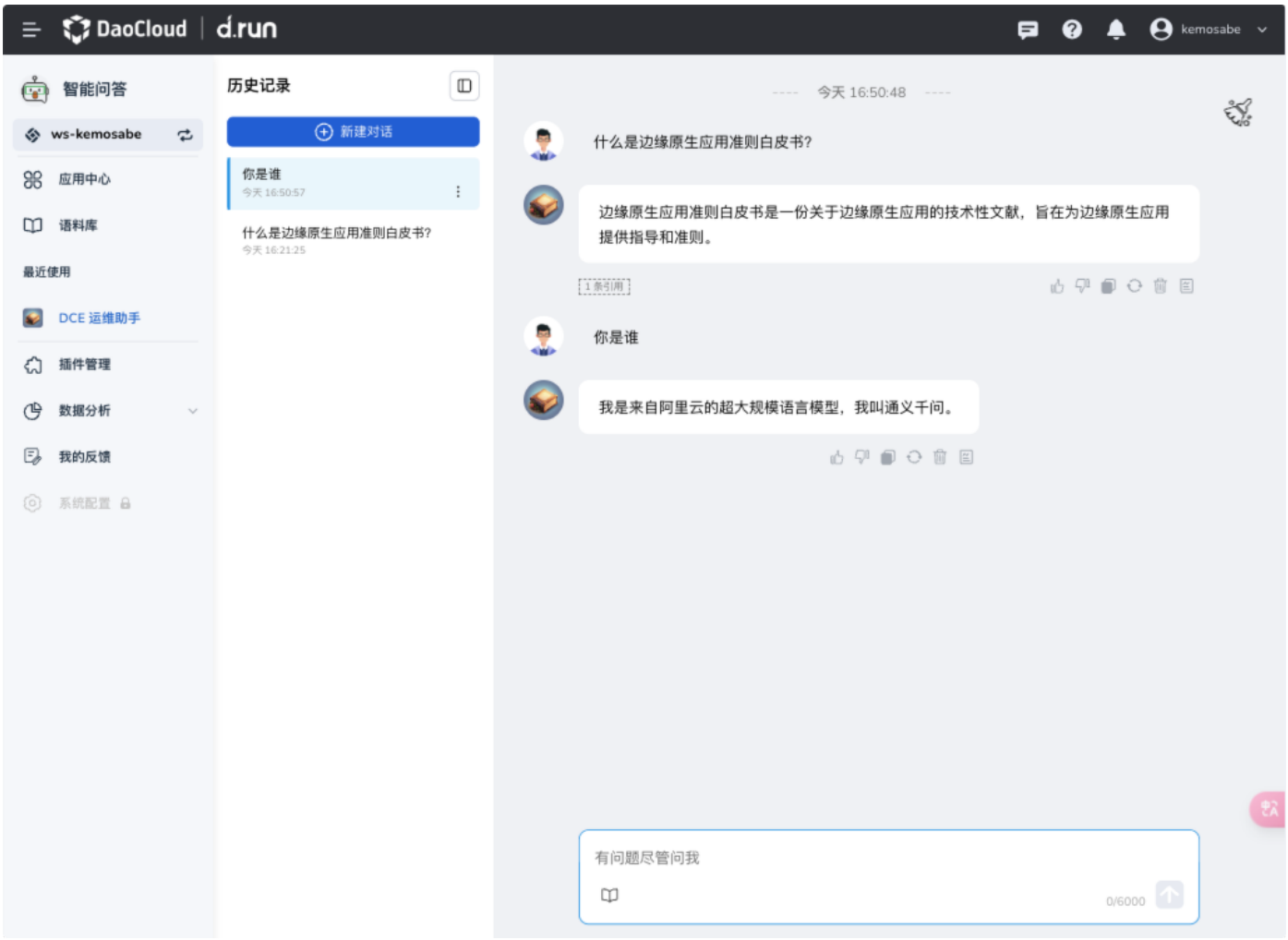
2. 填写表单

其中 API Key 可以去大模型官网去申请，例如 OpenAI API Key、阿里模型广场 等等



3. 返回 智能问答 -> 应用中心 ，找到刚创建的 **DCE** 运维助手 ，点击卡片下方的 **...** -> 下架 -> 编辑， 更改为 在线模型服务 。选择上一步通过 API Key 添加的在线模型服务，确认无误后点击右上角的 保存 -> 发布 。

4. 开始对话，做一些简单的提问。可以看到，已切换到了在线模型服务：通义千问。



5. 以此类推，你可以通过 API Key，添加试用主流的各种在线大模型。

使用流程引擎创建更灵活的应用

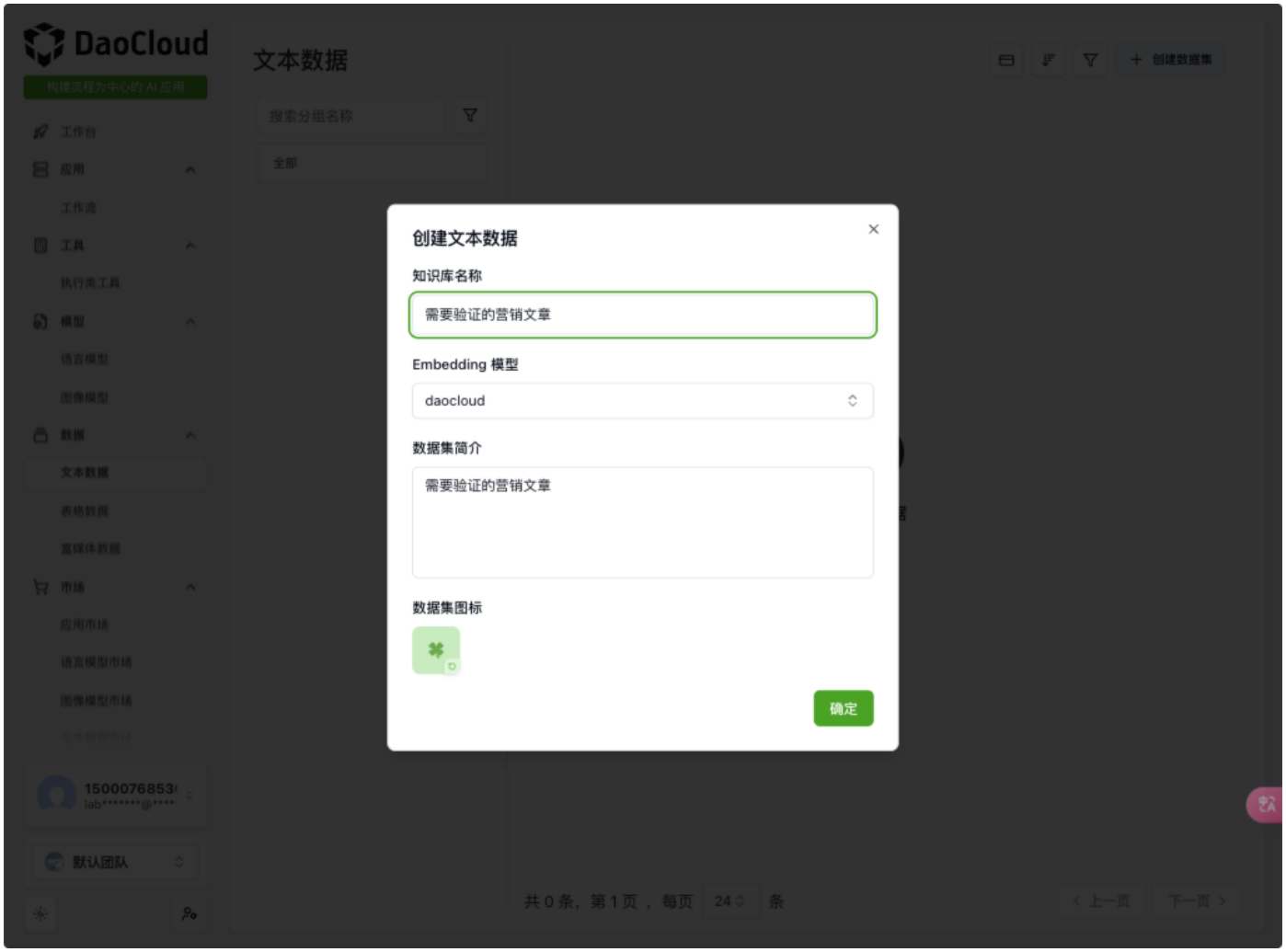
> **Tip**
>
> 本节还在建设中，将陆续补充完善。

如果您觉得使用 智能问答 创建的应用，还无法解决实际问题，可以使用 流程引擎 ，通过工具自定义构建流程，创建更灵活的应用。

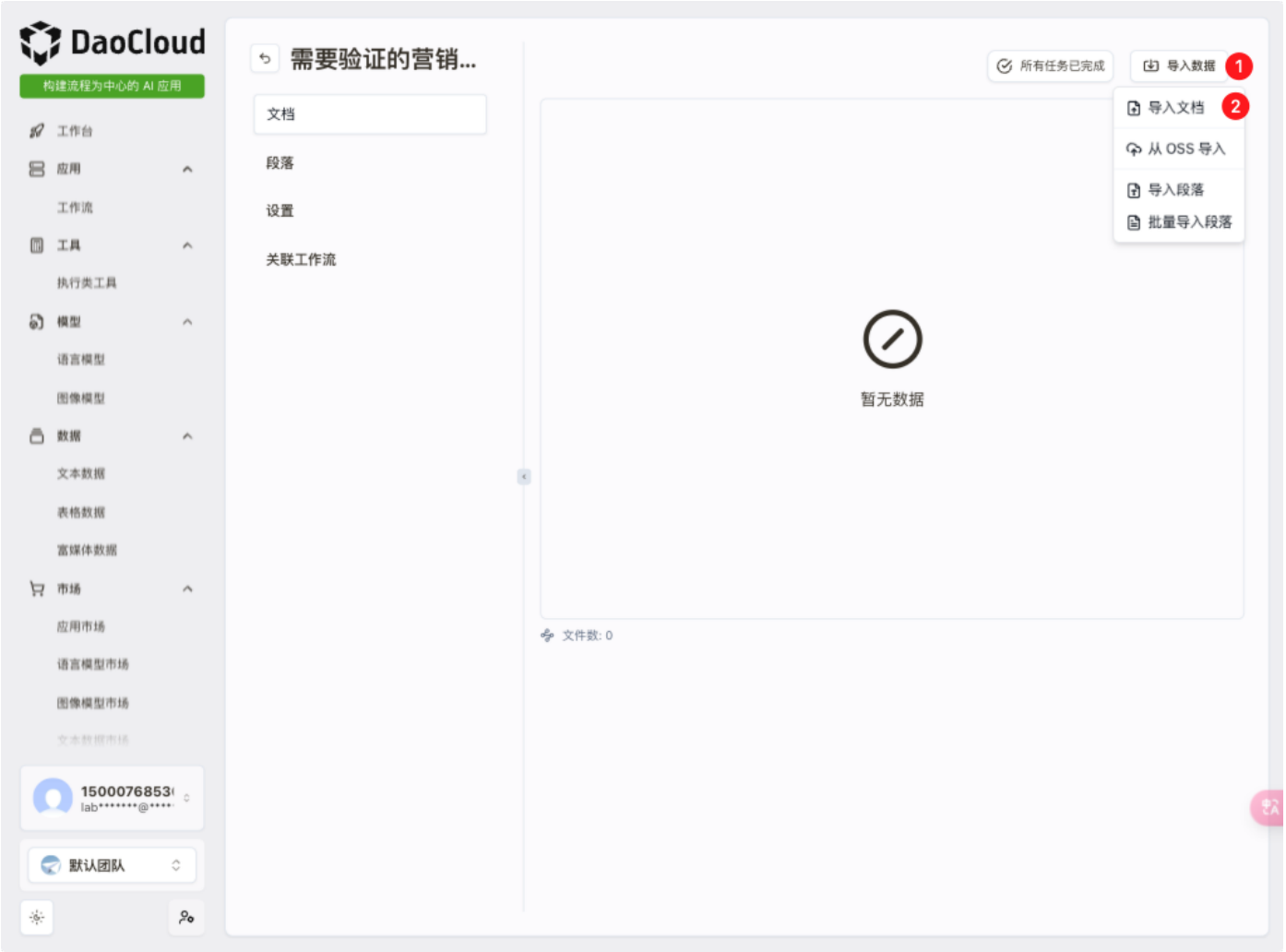
下面我们创建一个应用，来处理 文案合规 的场景。

1. 准备一篇营销文章，包含 绝对 、 万能 等词条，点击查阅营销文章 Demo 文本文件。

2. 点击左上角的 ≡ 打开导航栏，进入 流程编排 ，创建文本数据



3. 进入 需要验证的营销文章 ，点击 导入数据 -> 导入文档



4. 上传准备好的营销文章，点击 上传 -> 提交

5. 使用以下工具构建流程：
   - 循环
   - 文本向量搜索
   - 大语言模型多轮对话
6. 测试流程

一元体验

本节介绍以 1 元购买算力后可以体验的功能。

> **Tip**
>
> 本节还在建设中，将陆续补充完善。

算法开发

在线 Notebook、分布式模型训练

模型中心

体验部署本地模型，接入 HuggingFace 模型

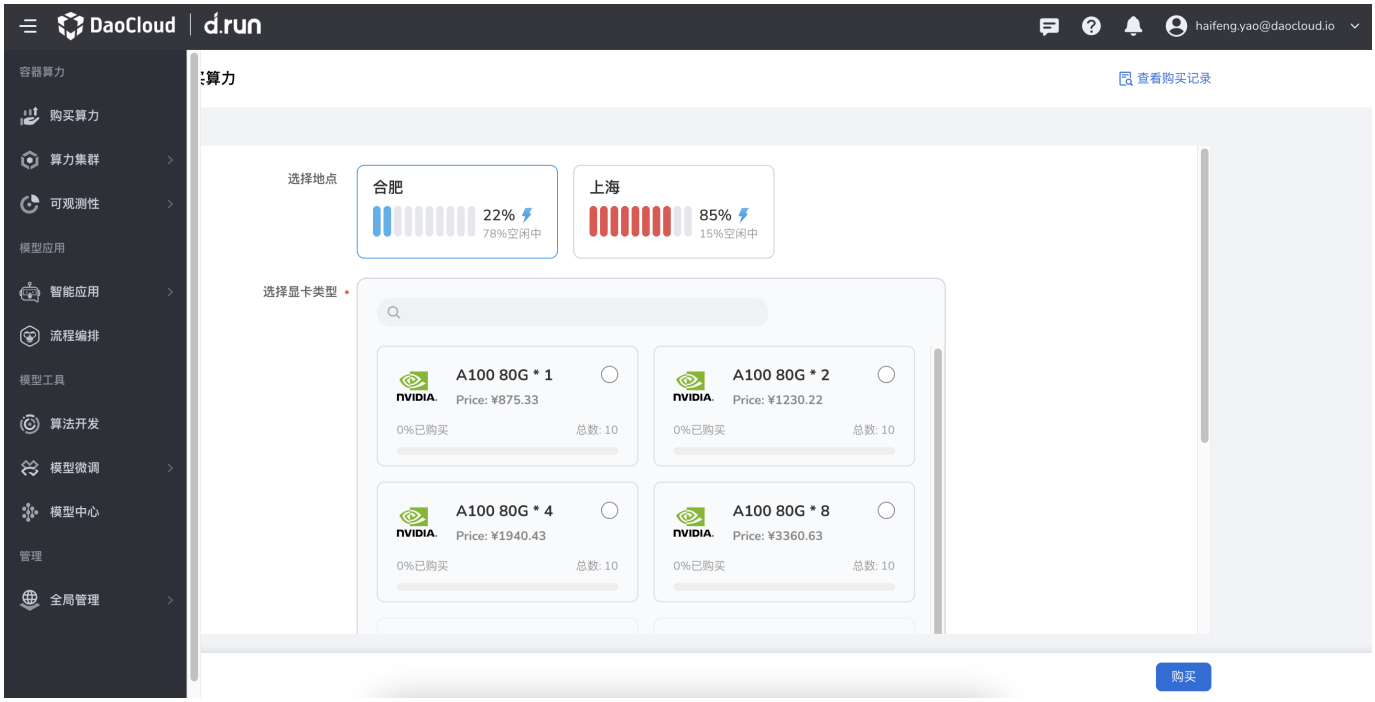
模型微调

一站式模型微调

1.2 智能算力

1.2.1 购买算力

用户使用 d.run 各项服务之前，需要先购买可用的算力，助力企业将算力变为算利。 用户也可以很方便的查看购买记录细目，做好成本预算。



Tip

A100 80G * 1 指的是 1 张 A100 显卡，显存共 80 GB。

勾选最后一个选项 **CPU 1G * 1** ，可以用一元体验完整功能。

注册并体验 **d.run**

1.2.2 算力集群

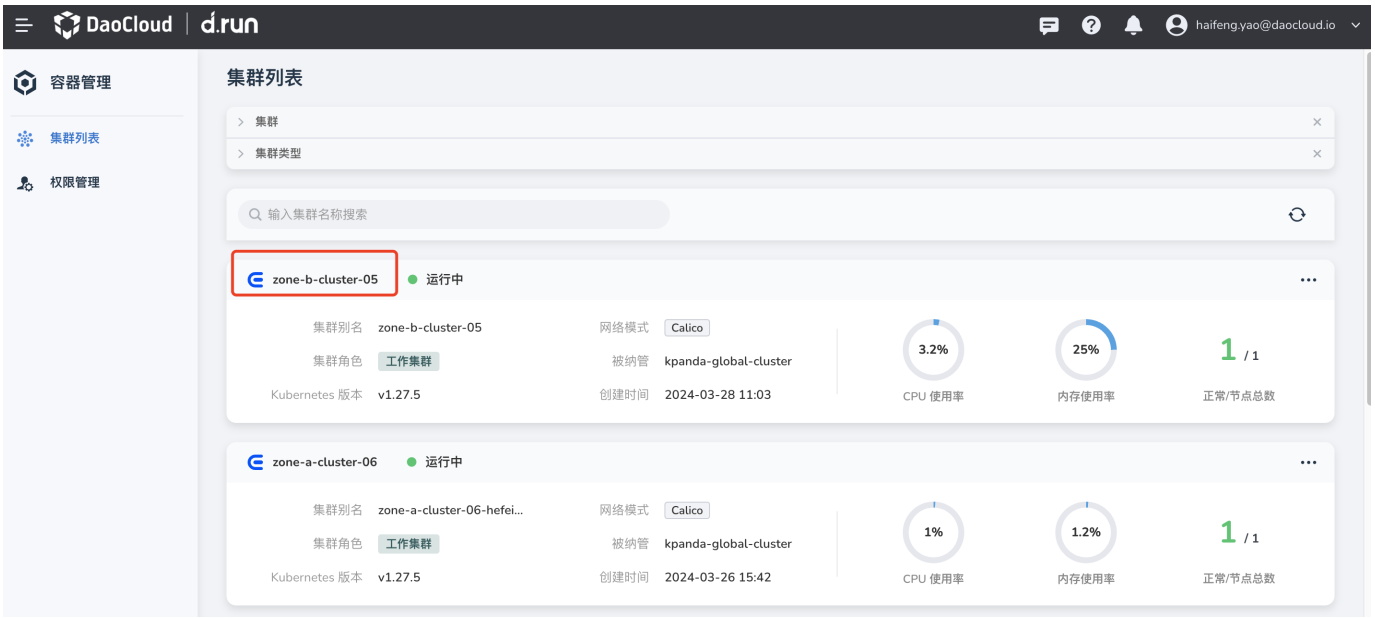
介绍

算力集群

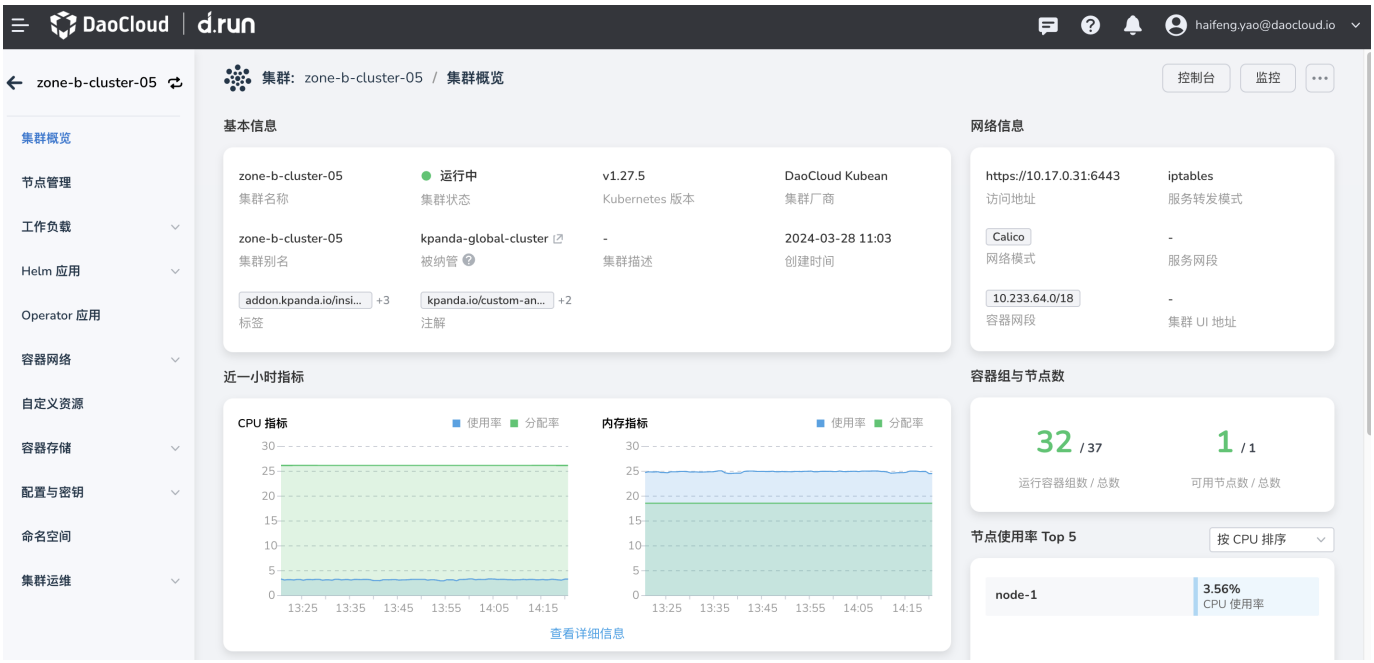
购买算力之后，可用的集群将显示在集群列表中。

> **Tip**
>
> 集群（Cluster）指容器运行所需要的云资源组合，关联了若干服务器节点，在这些节点上运行容器化应用，每个集群至少有一个工作节点。 容器平台可自建集群并进行集群全生命周期管理，以及纳管各云原生厂商任意符合一定版本要求的 Kubernetes 集群。

集群列表显示了集群的一些基本信息，点击集群名称。



进入 集群概览 页面，您可以通过左侧导航栏管理节点、工作负载、Helm 应用等等。



容器管理是基于 Kubernetes 构建的面向云原生应用的容器化管理模块。 它基于原生多集群架构，解耦底层基础设施，实现多云与多集群统一化管理，大幅简化企业的应用上云流程， 有效降低运维管理和人力成本。通过容器管理，您可以便捷创建 Kubernetes 集群，快速搭建企业级的容器云管理平台。

!!! tip

容器化是 AI 数据中心发展的不可或缺的趋势，它是应用封装、部署和托管的自然延伸。

!!! success

通过容器化，您可以更快速、简单地开发和部署 AI 应用程序，比构建虚拟设备更加高效。 容器化架构带来了令人瞩目的运维和经济效益，包括更低的许可成本、更高的物理资源利用率、更好的可扩展性以及更高的服务可靠性。

展望未来，容器虚拟化将帮助企业更好地利用混合云和多云环境，实现更优化的资源管理和应用部署。

注册并体验 **d.run**

功能特性

算力集群提供基础设施能力，支持超大规模算力集群、异构 GPU 等一站式托管，并提供一系列如 vGPU 等软硬一体加速方案。

主要功能	细分项
异构加速	支持 Nvidia、天数、昇腾等异构硬件加速
	支持物理 GPU 单卡资源算力、显存、切分功能
	多个服务容器可共享单张GPU卡，并支持限制和隔离每个服务容器所占用的GPU算力、显存额度。保障服务间互不干扰，保障服务性能，提升资源利用效率
	支持按照项目对 GPU 资源进行配额管理
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
	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
	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
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

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

Pod 也称为容器组，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 ：，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”

```
[root@a-master1 ~]# kubectl get pod -n skoala-system -w
NAME                                                READY   STATUS      RESTARTS   AGE
helm-operation-install-skoala-init-mgzpx-2kchw      0/1     OOMKilled   0          9m
helm-operation-install-skoala-init-mgzpx-5ttl5      0/1     OOMKilled   0          8m21s
helm-operation-install-skoala-init-mgzpx-ghzhl      0/1     OOMKilled   0          11m
helm-operation-install-skoala-init-mgzpx-hr8tr      0/1     OOMKilled   0          7m21s
helm-operation-install-skoala-init-mgzpx-j9vwm      0/1     OOMKilled   0          9m43s
helm-operation-install-skoala-init-mgzpx-pw22c      0/1     OOMKilled   0          12m
helm-operation-install-skoala-init-mgzpx-rvzrq      0/1     OOMKilled   0          13m
hive-7876fcd8d6-mnzrq                               2/2     Running     0          21h
sesame-5d776f5cc9-npvnv                             2/2     Running     0          21h
ui-b6f74f78-fd4sr                                   2/2     Running     0          21h
```

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。

点击查看如何调整脚本

- 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

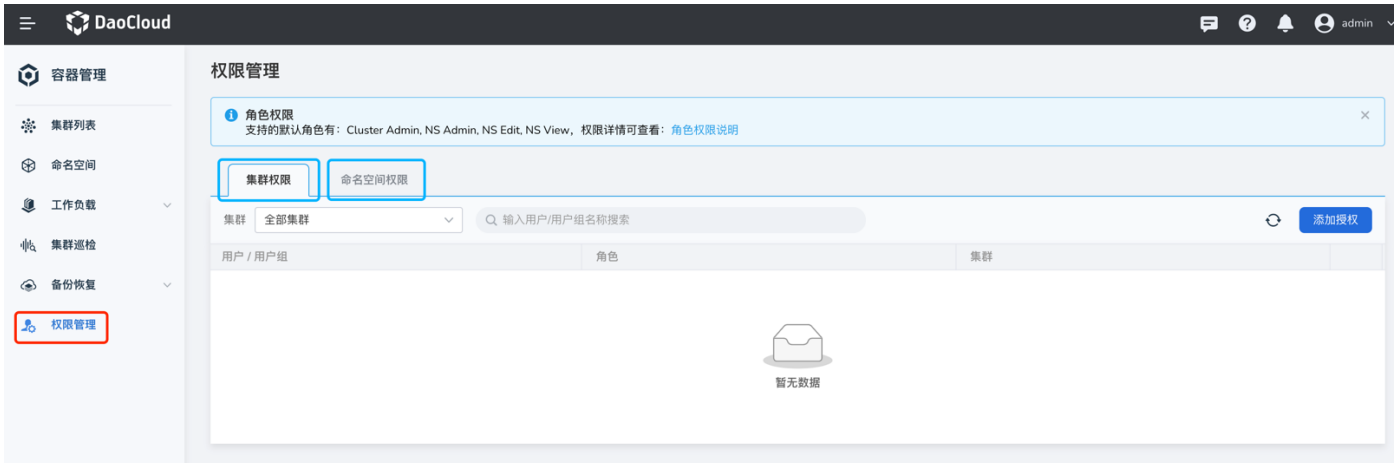
```
kubectl get cm -n kpanda-system skoala-dev-setting -o yaml
apiVersion: v1
data:
clusterSetting: '{"plugins":[{"name":1,"intelligent_detection":true},{"name":2,"enabled":true,"intelligent_detection":true},{"name":3},{"name":
6,"intelligent_detection":true},{"name":7,"intelligent_detection":true},{"name":8,"intelligent_detection":true},{"name":9,"intelligent_detection":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

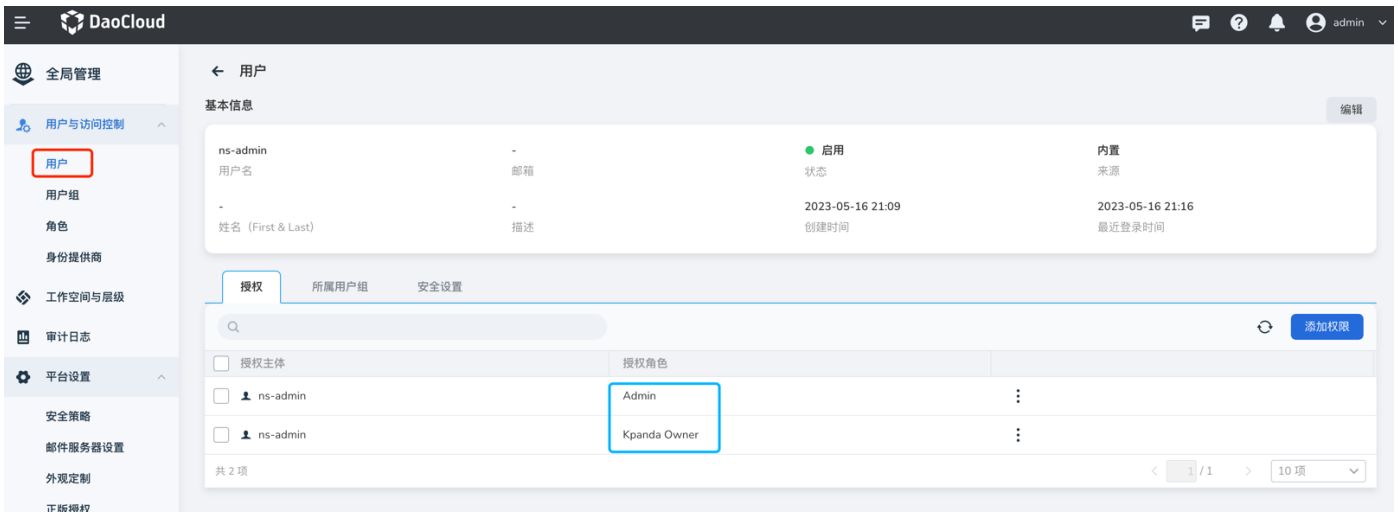
2. 容器管理模块和全局管理模块的权限问题

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。



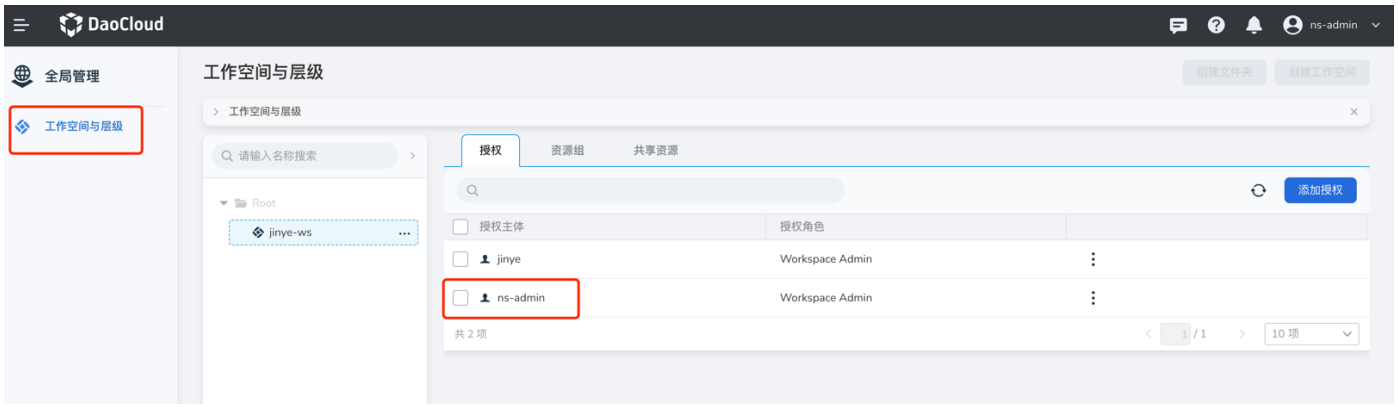
- 全局管理模块中用户的授权：使用 admin 账号，进入 全局管理 -> 用户与访问控制 -> 用户 菜单，找到对应用户。在 授权所属用户组 标签页，如果有类似 Admin、Kpanda Owner 等拥有容器管理权限的角色，那即使在容器管理没有绑定集群权限或命名空间权限，也可以看到全部的集群，可以参考用户授权文档说明



- 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

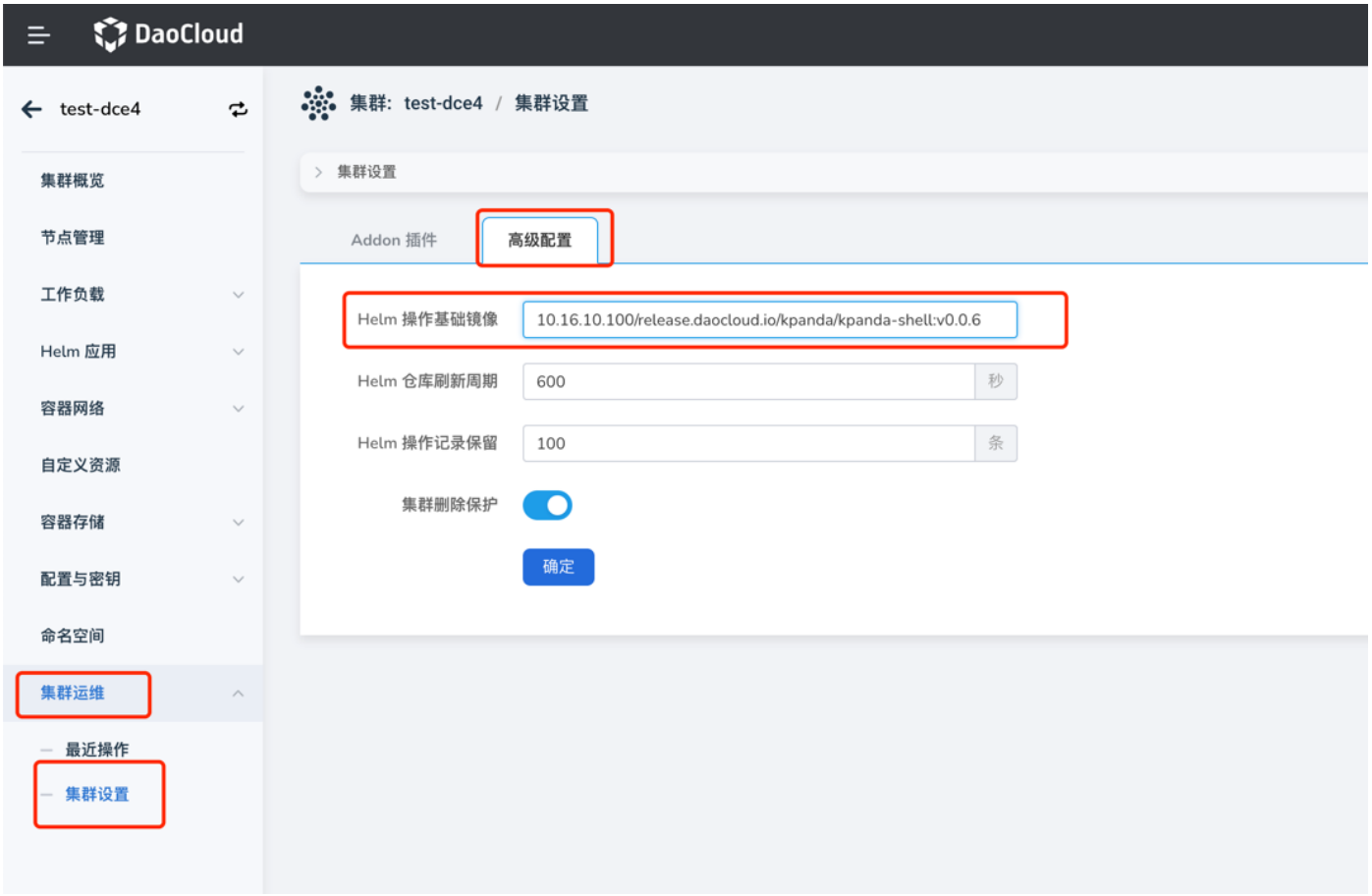


3. Helm 安装应用时，无法拉取 kpanda-shell 镜像

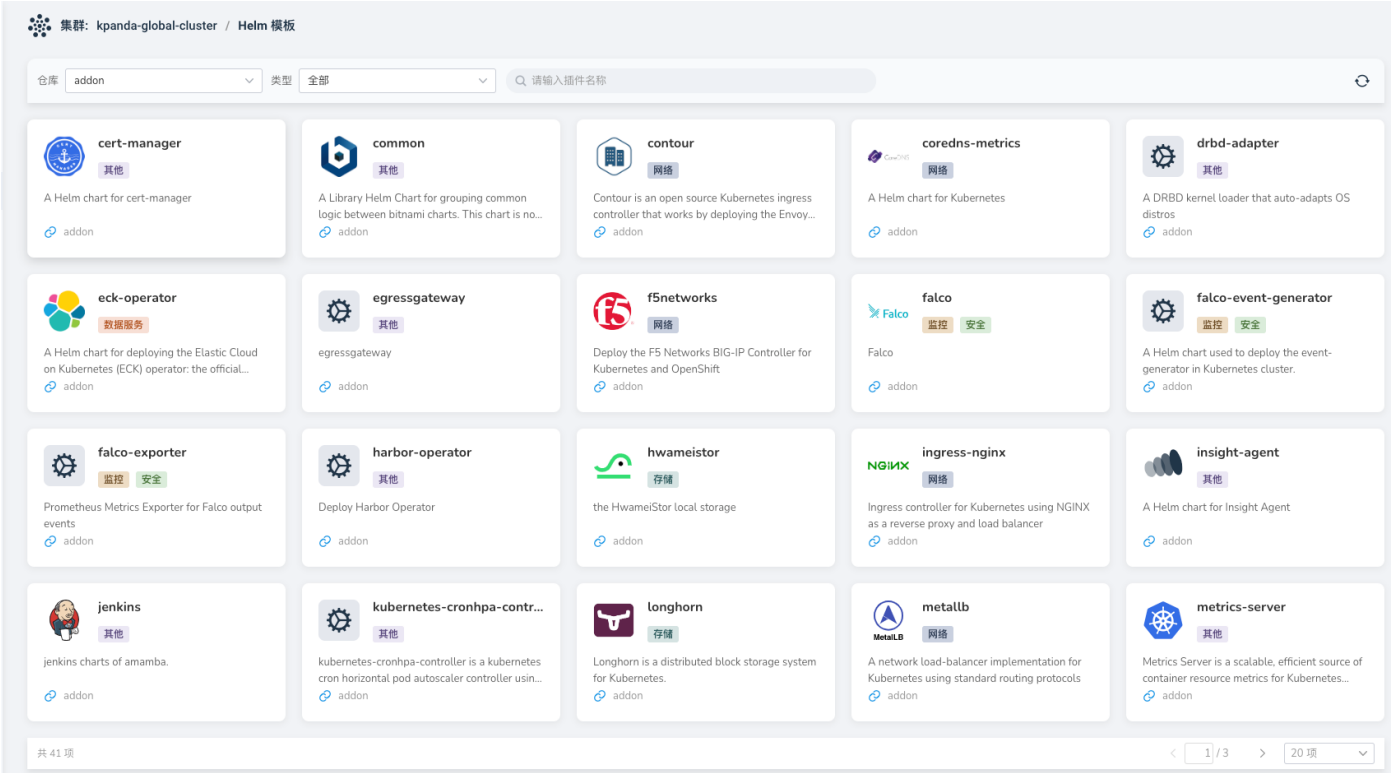
使用离线安装后，接入的集群安装helm应用经常会遇到拉取 kpanda-shell 镜像失败，如图：

```
Events:
  Type     Reason          Age                  From               Message
  ----     ------          ----                 ----               -------
  Normal   Scheduled       28m                  default-scheduler  Successfully assigned test-xjw/helm-operation-install-ingress-nginx-9s6hv-mv5m6 to worker-node-1
  Normal   SandboxChanged  28m                  kubelet            Pod sandbox changed, it will be killed and re-created.
  Warning  Failed          27m (x3 over 28m)    kubelet            Failed to pull image "10.16.10.100/release.daocloud.io/kpanda/kpanda-shell:v0.0.6": rpc error: cod
e = Unknown desc = Error response from daemon: Get "https://10.16.10.100/v2/": x509: cannot validate certificate for 10.16.10.100 because it doesn't contain any IP SAN
s
  Warning  Failed          27m (x3 over 28m)    kubelet            Error: ErrImagePull
  Warning  Failed          26m (x7 over 28m)    kubelet            Error: ImagePullBackOff
  Normal   Pulling         26m (x4 over 28m)    kubelet            Pulling image "10.16.10.100/release.daocloud.io/kpanda/kpanda-shell:v0.0.6"
  Normal   BackOff         3m10s (x111 over 28m)  kubelet          Back-off pulling image "10.16.10.100/release.daocloud.io/kpanda/kpanda-shell:v0.0.6"
[root@controller-node-1 ~]#
```

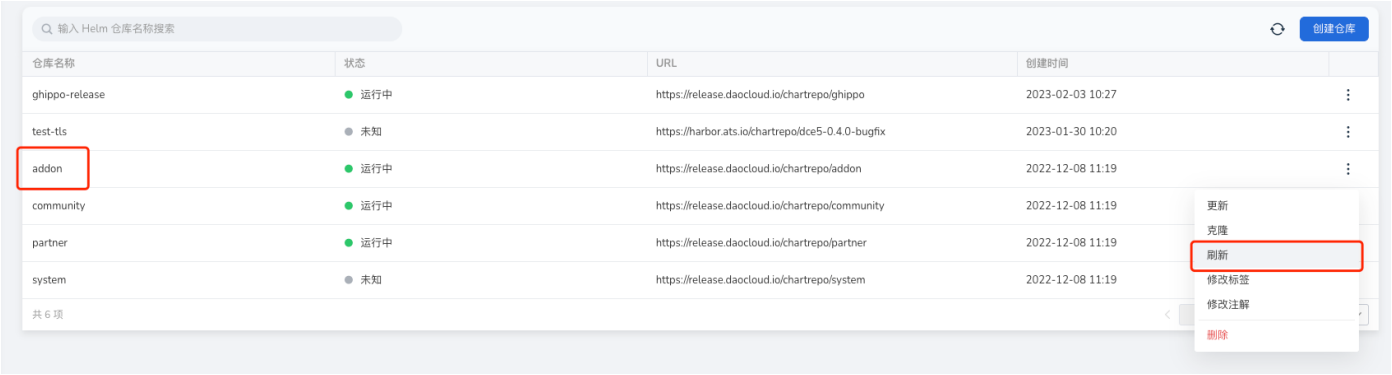
此时，只需要去集群运维-集群设置页面，高级配置标签页，修改 Helm 操作基础镜像为一个可以被该集群正常拉取到的 kpanda-shell 的镜像即可。



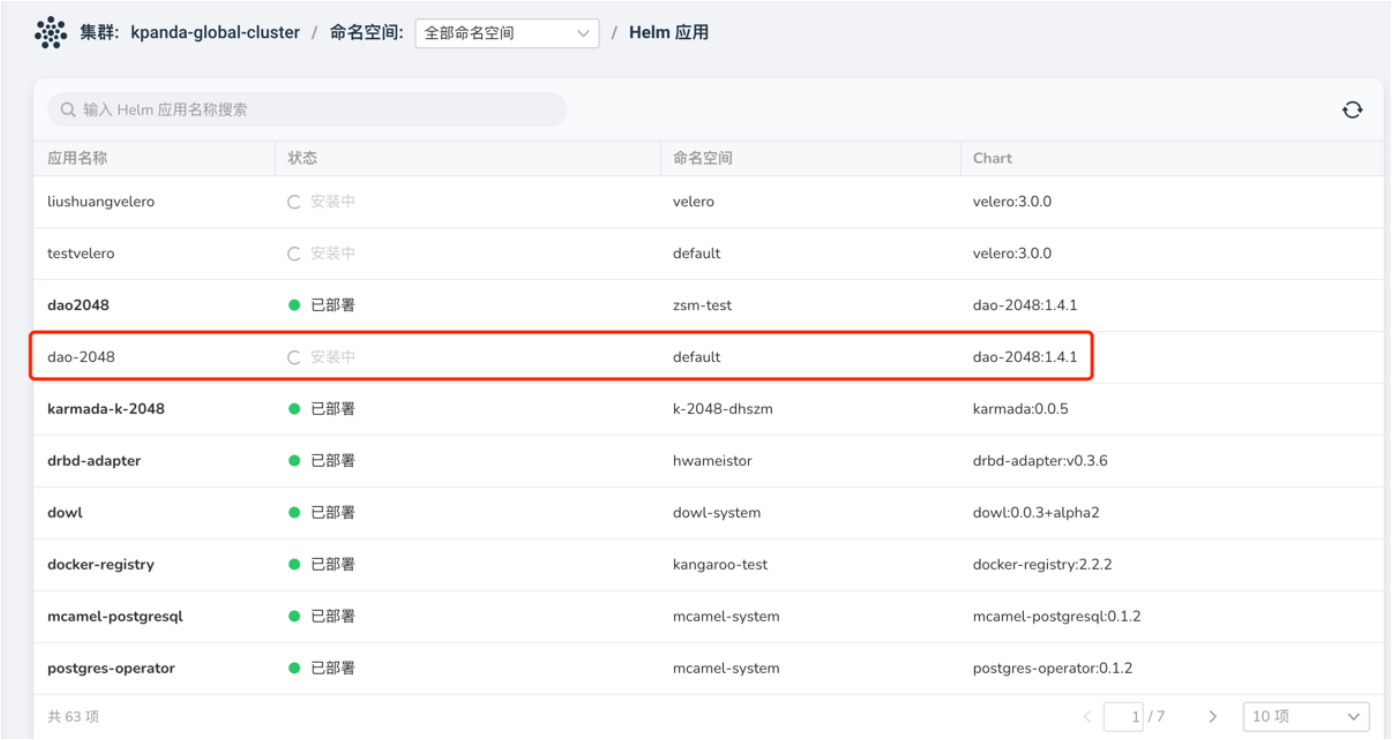
4. Helm Chart 界面未显示最新上传到对应 Helm Repo 的 Chart，如图：



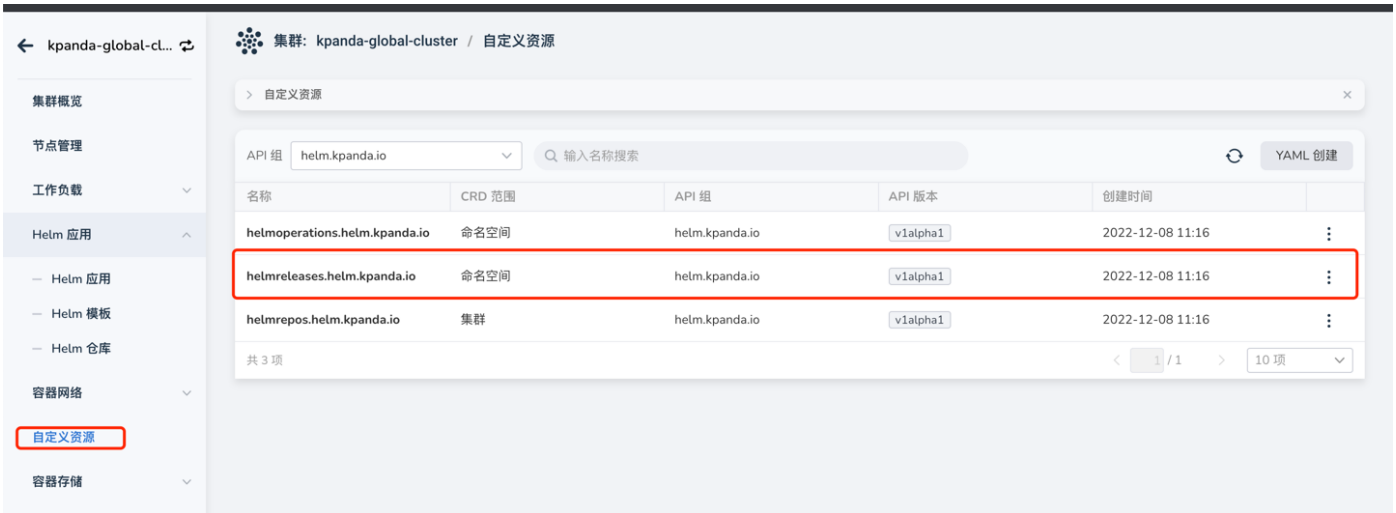
此时，只需要去 Helm 仓库刷新对应的 Helm 仓库即可。

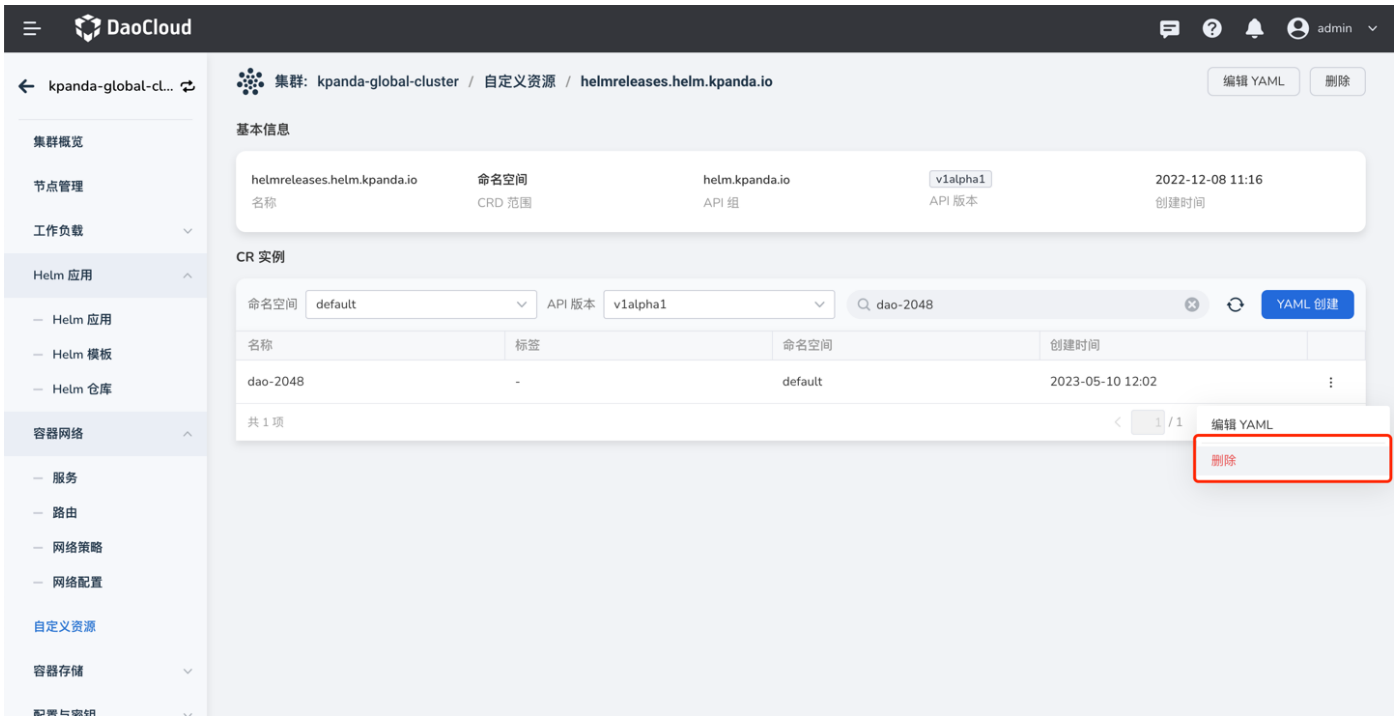


5. Helm 安装应用失败时卡在安装中无法删除应用重新安装，如图：



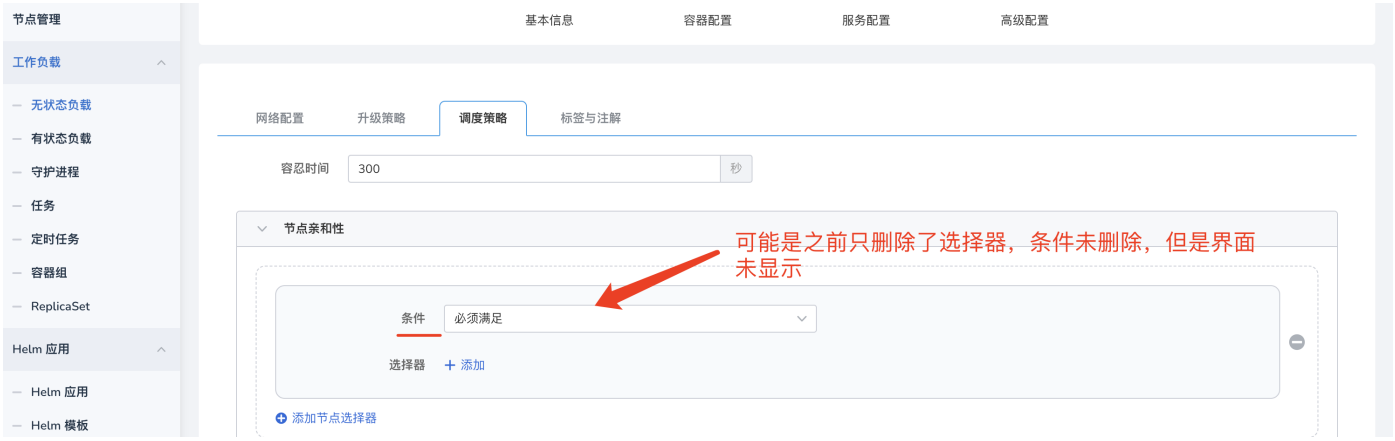
此时，只需要去自定义资源页面，找到 helmreleases.helm.kpanda.io CRD，然后找到对应的 helmreleases CR 删除即可。

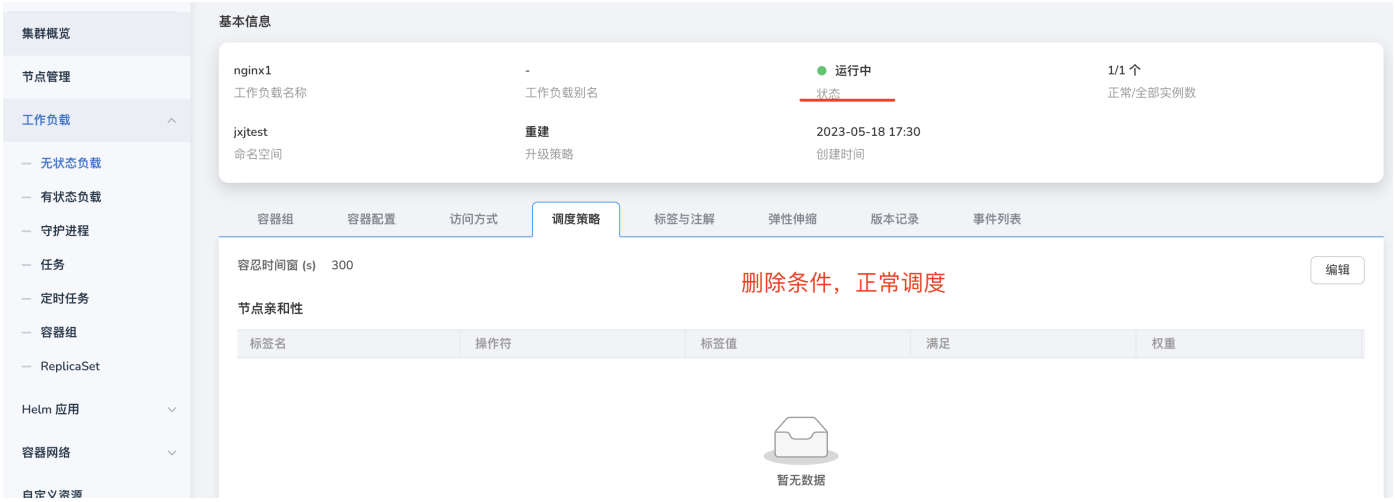




6. 工作负载 -> 删除节点亲和性 等调度策略后，调度异常，如图：

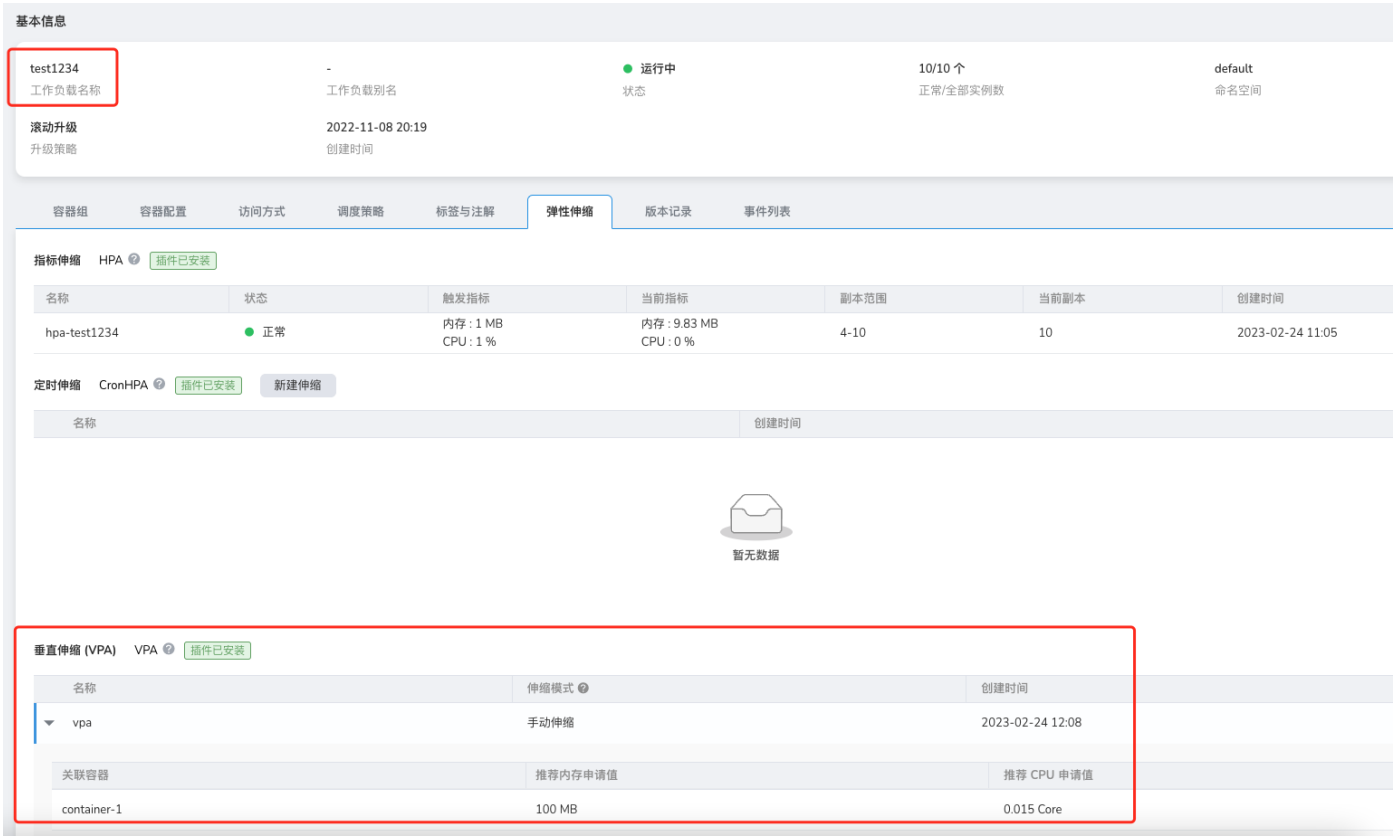
此时，可能是因为策略没有删除干净，点击编辑，删除所有策略。





7. 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:



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。

8. 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。

   参考申请证书流程。

   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

集群管理

访问集群

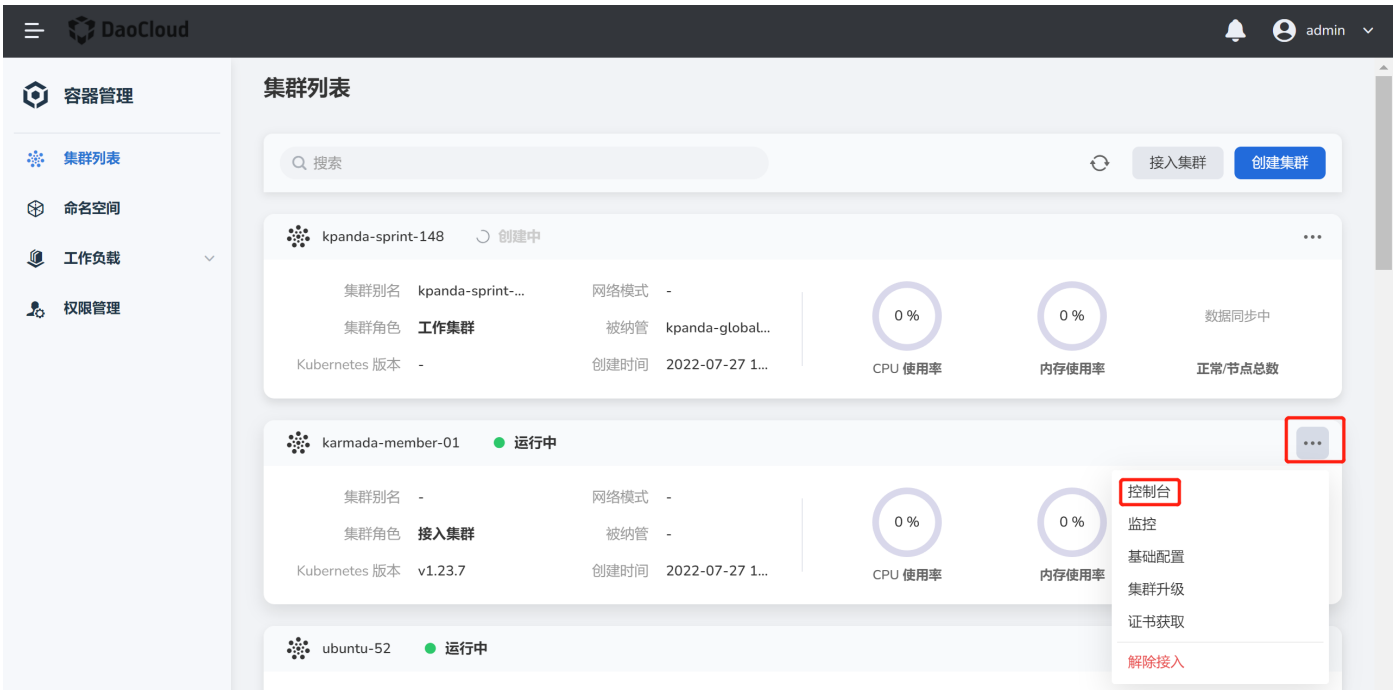
对于算力集群，不仅可以通过 UI 界面直接访问，也可以通过其他两种方式进行访问控制：

- 通过 CloudShell（控制台）在线访问
- 下载集群 kubeconfig 证书后通过 kubectl 进行访问

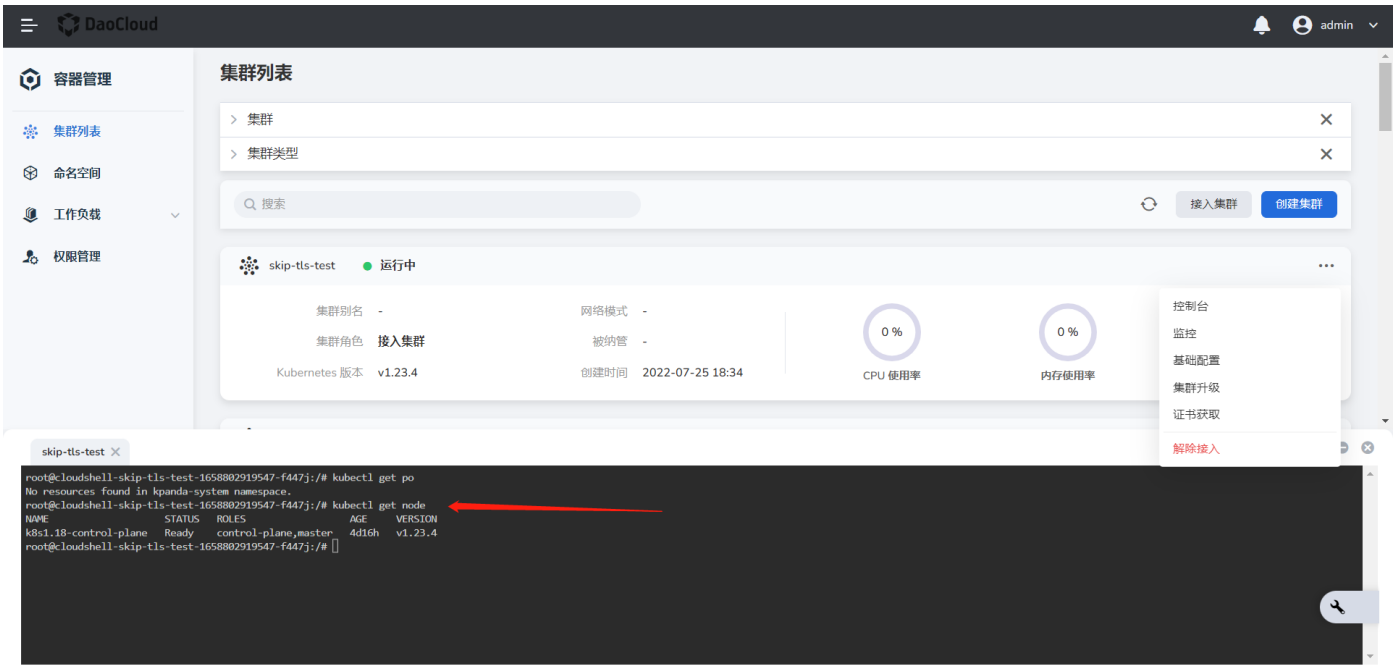
> **Note**
>
> 访问集群时，用户应具有 Cluster Admin 权限或更高权限。

通过控制台访问

1. 在 集群列表 页选择需要通过控制台访问的集群，点击右侧的 **...** 操作图标并在下拉列表中点击 控制台 。



2. 在控制台执行 **kubectl get node** 命令，验证控制台与集群的连通性。如图，控制台将返回集群下的节点信息。



现在，您可以通过控制台来访问并管理该集群了。

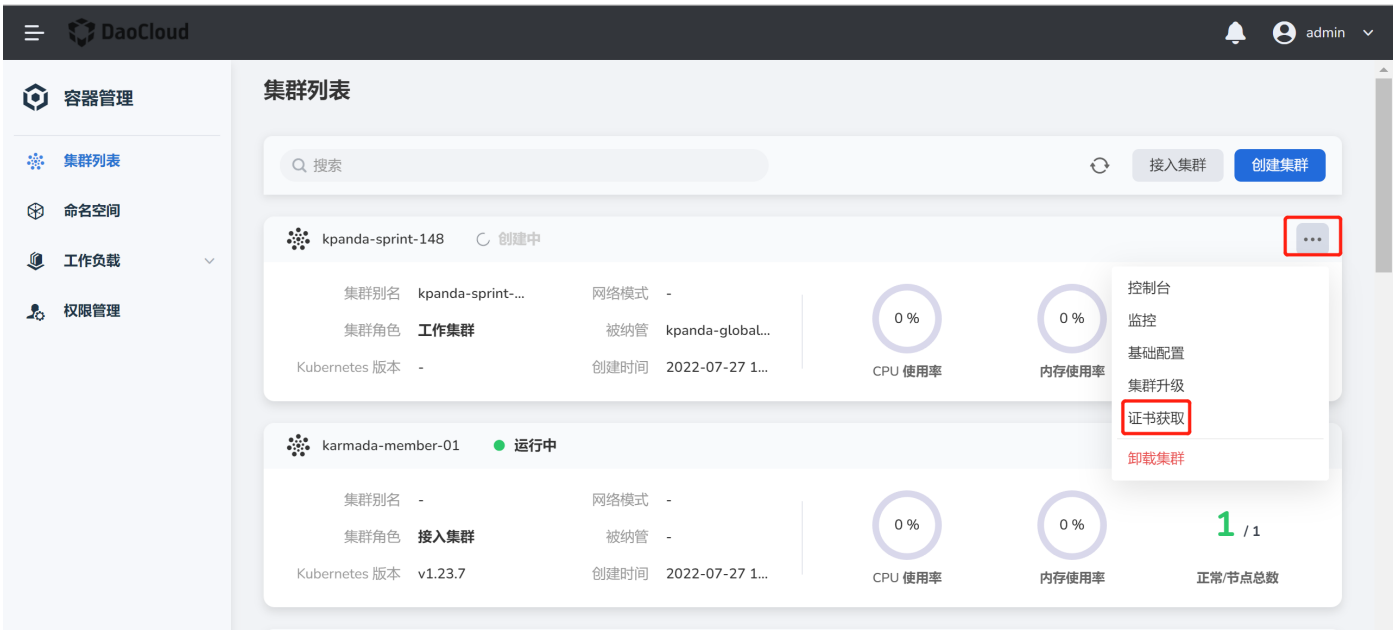
通过 kubectl 访问

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，请参阅如何安装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 下载 **kubeconfig** 。



2. 选择证书有效期并点击 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。

卸载/解除接入集群

通过 d.run 容器管理平台 创建的集群 支持 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。d.run 不支持删除接入型的集群。

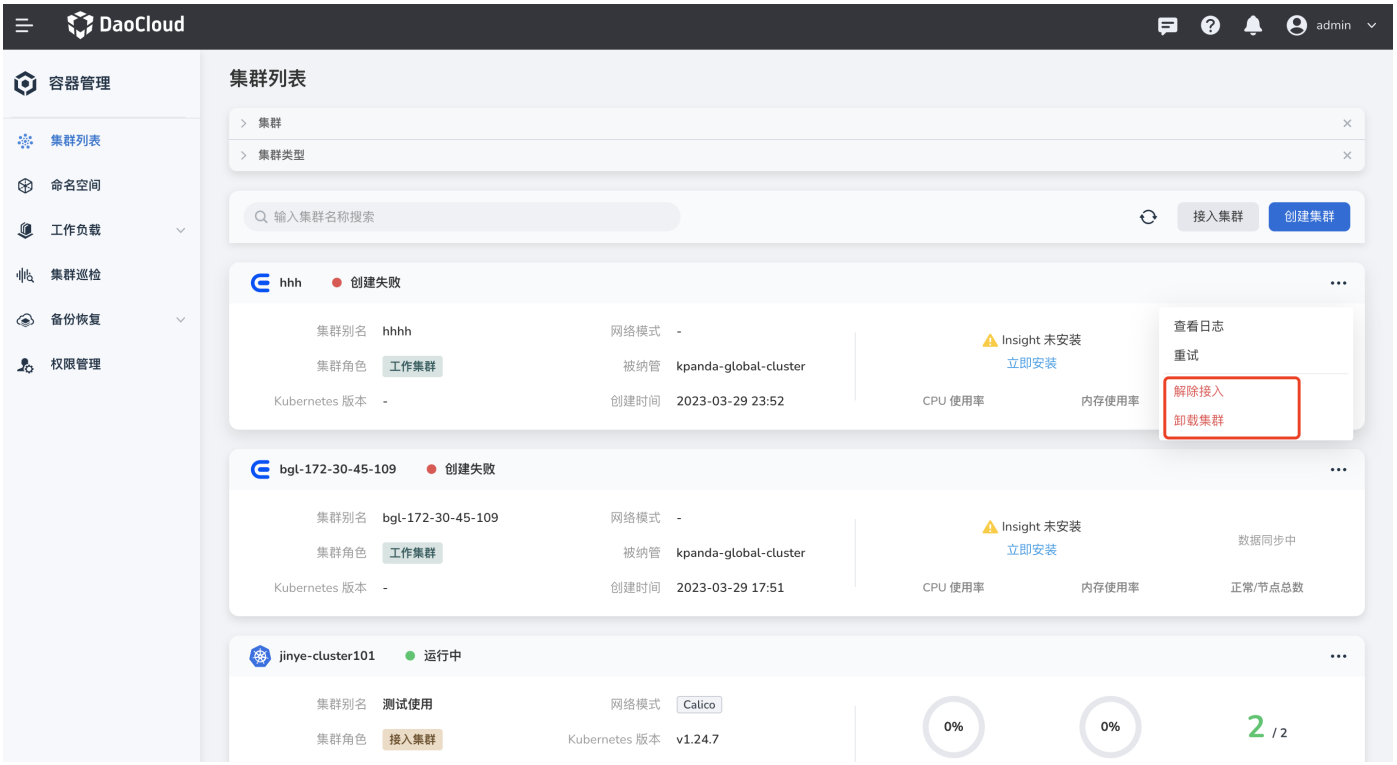
在 d.run 平台中， 卸载集群 和 解除接入 的区别在于：

- 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。

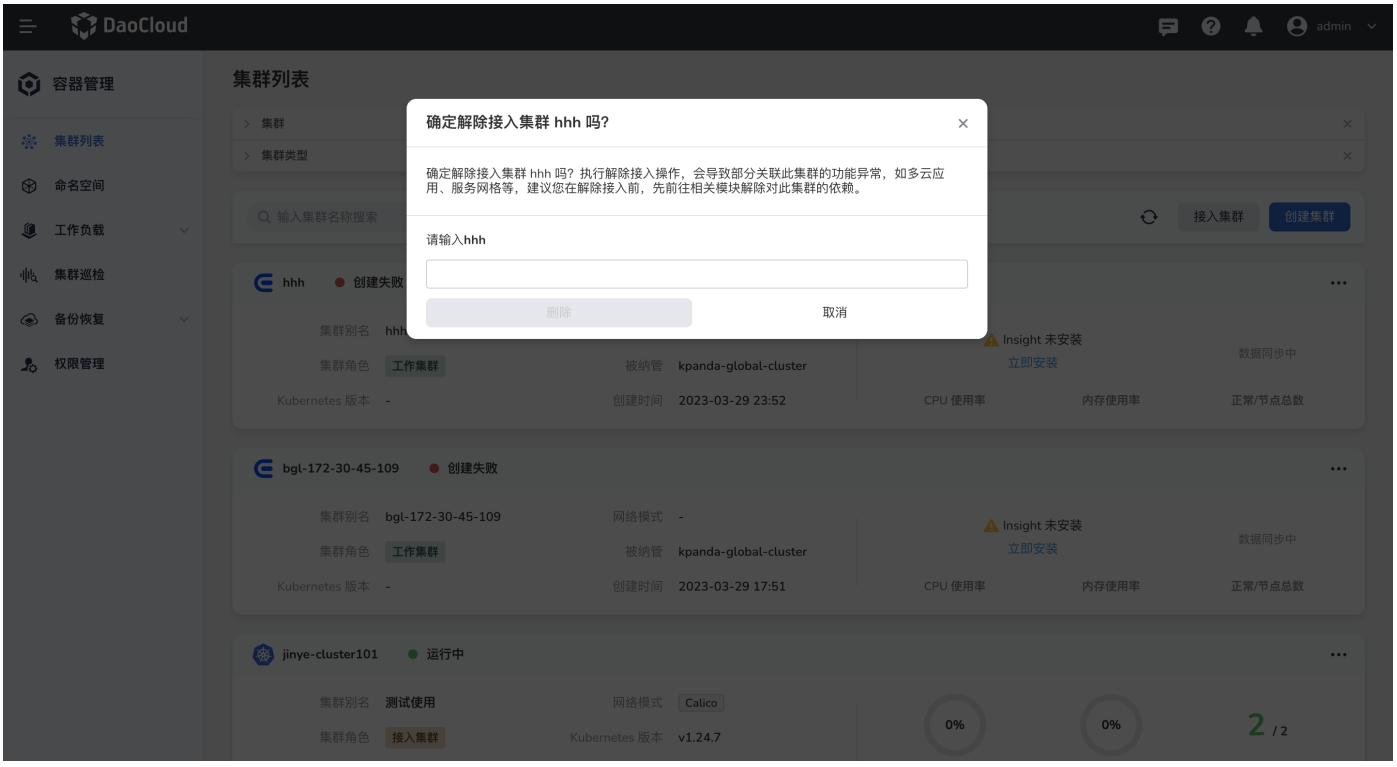
卸载集群

> **Note**
>
> • 当前操作用户应具备 Admin 或 Kpanda Owner 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。

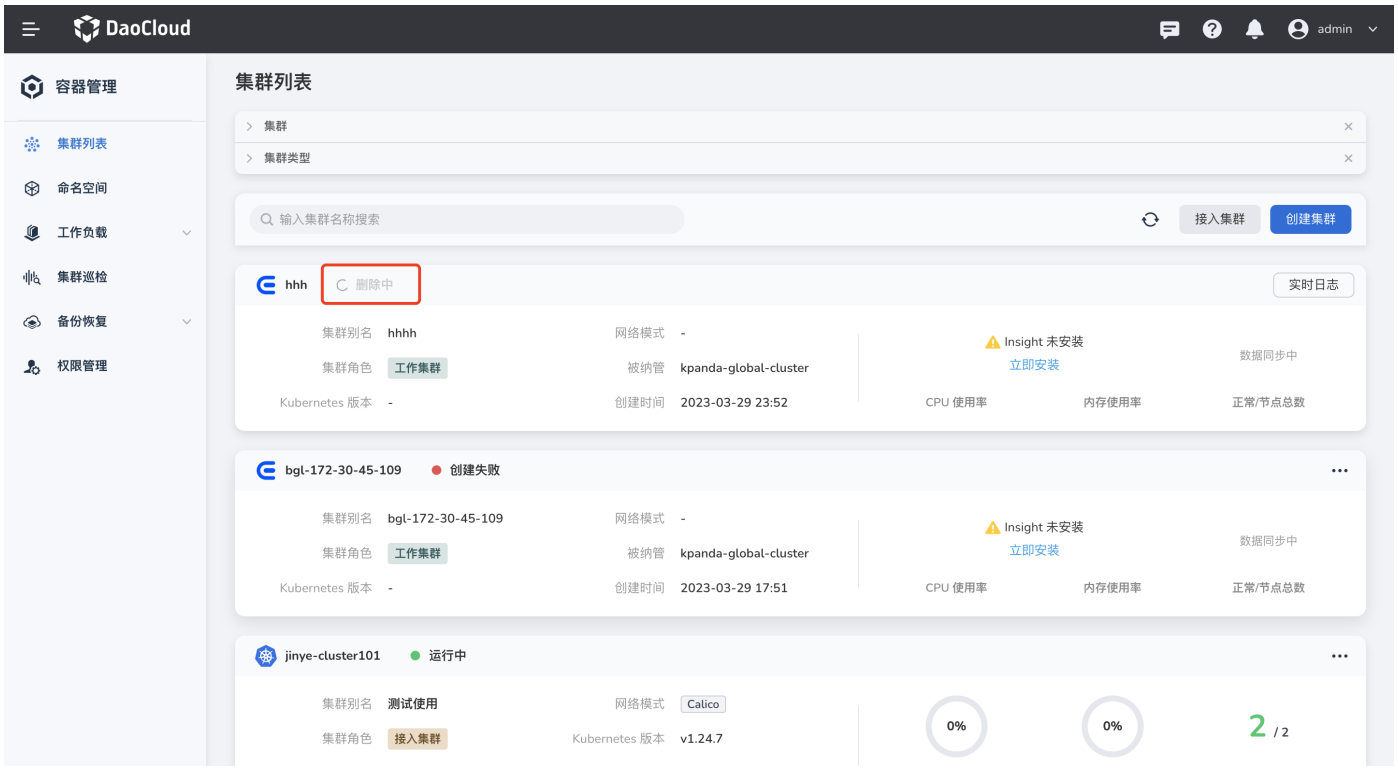
1. 在 集群列表 页找到需要卸载集群，点击右侧的 ... 并在下拉列表中点击 卸载集群 。



2. 输入集群名称进行确认，然后点击 删除 。

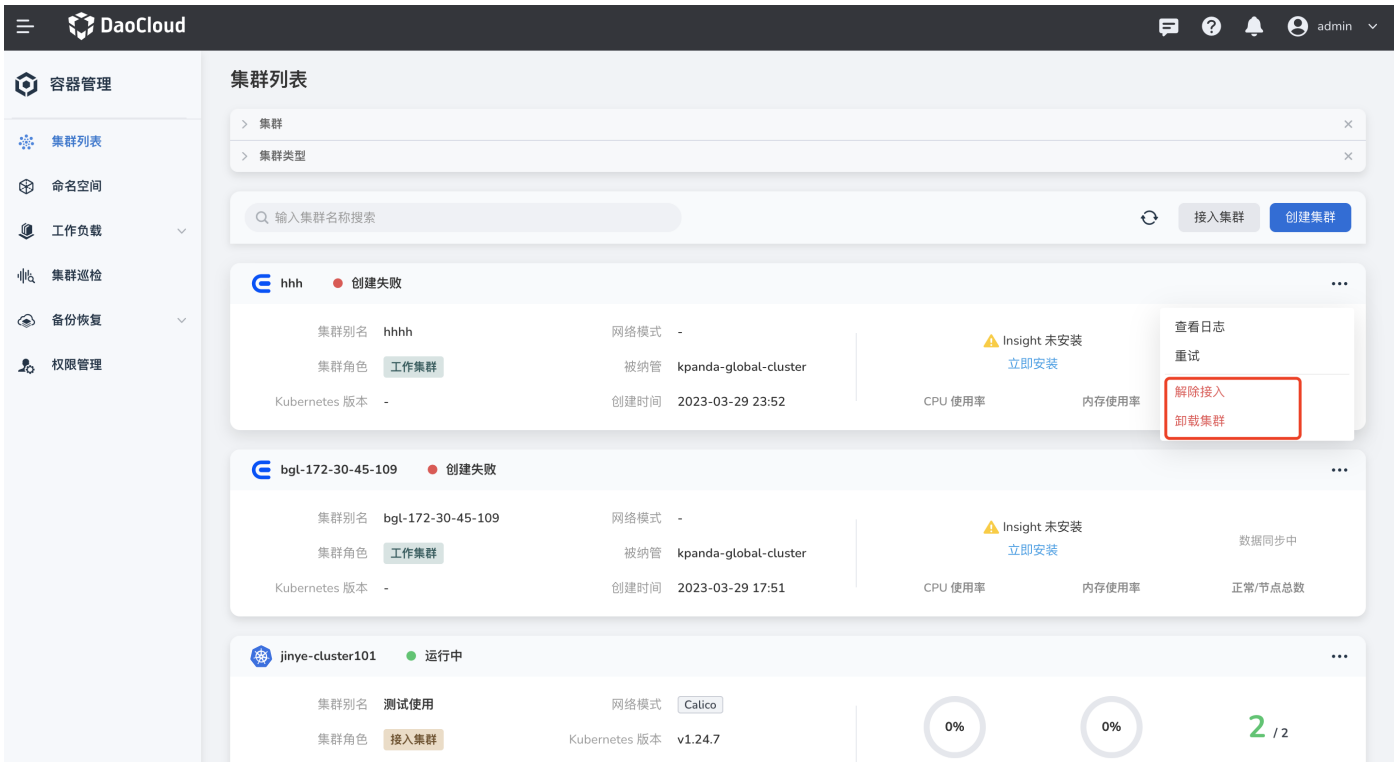


3. 返回 集群列表 页可以看到该集群的状态已经变成 删除中 。卸载集群可能需要一段时间，请您耐心等候。

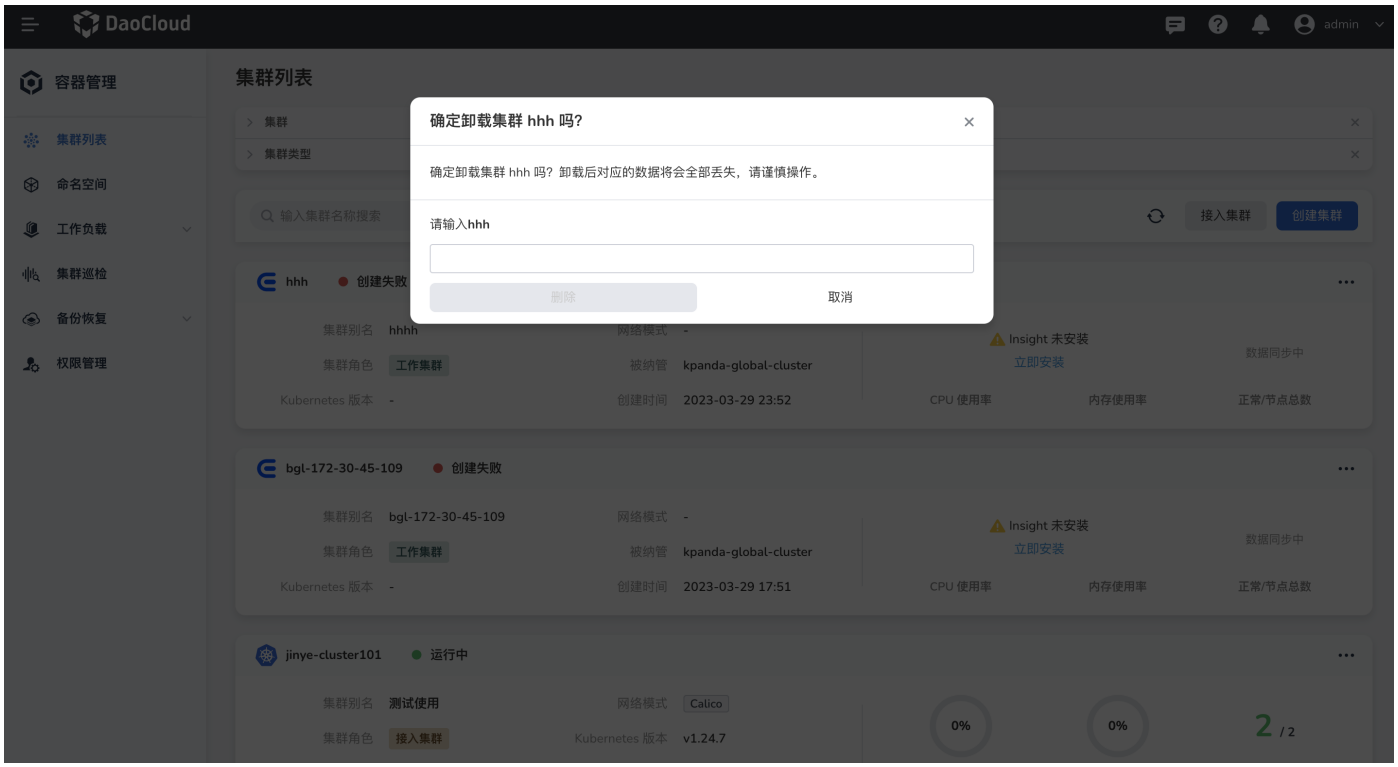


解除接入集群

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 。



2. 输入集群名称进行确认，然后点击 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：

删除 kpanda-system、insight-system 命名空间：

```
kubectl delete ns kpanda-system insight-system
```

集群角色

d.run 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

此集群用于运行容器管理、全局管理、可观测性等系统组件。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。

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

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。

支持项	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

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。

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因

集群版本支持范围

在 d.run 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 1.26、1.27、1.28。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 1.29 版本已经发布，此时社区支持的版本范围是 1.27、1.28、1.29。

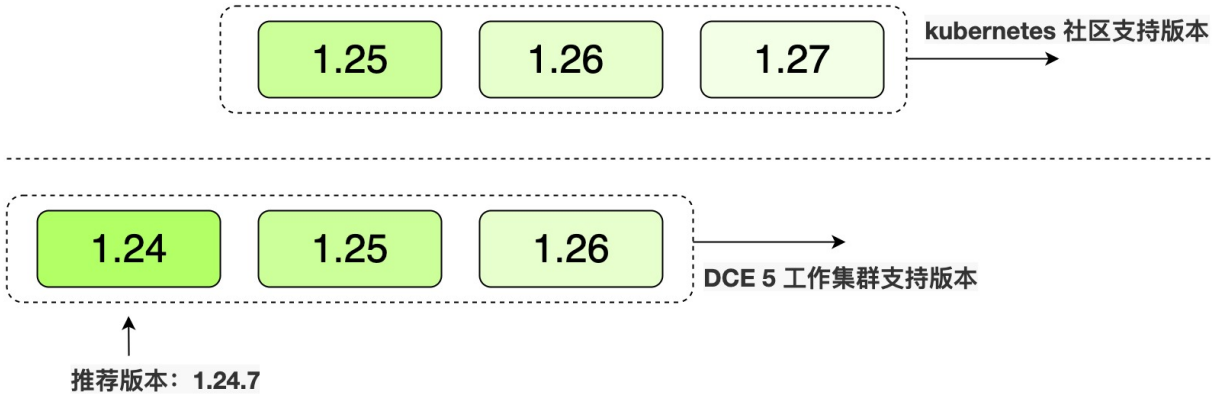
为了保障集群的安全和稳定性，在 d.run 中使用界面创建集群时支持的版本范围将始终比社区版本 低一个版本。

例如，社区支持的版本范围是 1.25、1.26、1.27，则在 d.run 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，d.run 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，d.run 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 版本支持范围

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	**d.run** 安装器	发布时间
• 1.26 • 1.27 • 1.28	• 1.26 • 1.27 • 1.28	**1.27.5**	v0.13.0	2023.11.30



权限管理

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：

- 可管理、编辑、查看对应集群
- 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
- 可授权用户为集群内角色 (Cluster Admin、NS Admin、NS Editor、NS Viewer)

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 Editor、NS Viewer。

NS Admin

NS Admin 具有以下权限：

- 可查看对应命名空间
- 管理、编辑、查看 命名空间下的所有工作负载，及自定义资源
- 可授权用户为对应命名空间角色 (NS Editor、NS Viewer)

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-ns-admin
  resourceVersion: "15173"
  uid: 69f64c7e-70e7-4c7c-a3e0-053f507f2bc3
rules:
```

```
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

NS Editor

NS Editor 具有以下权限：

- 可查看对应有权限的命名空间
- 管理、编辑、查看 命名空间下的所有工作负载

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 Viewer

NS Viewer 具有以下权限：

- 可查看对应命名空间
- 可查看对应命名空间下的所有工作负载，及自定义资源

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。

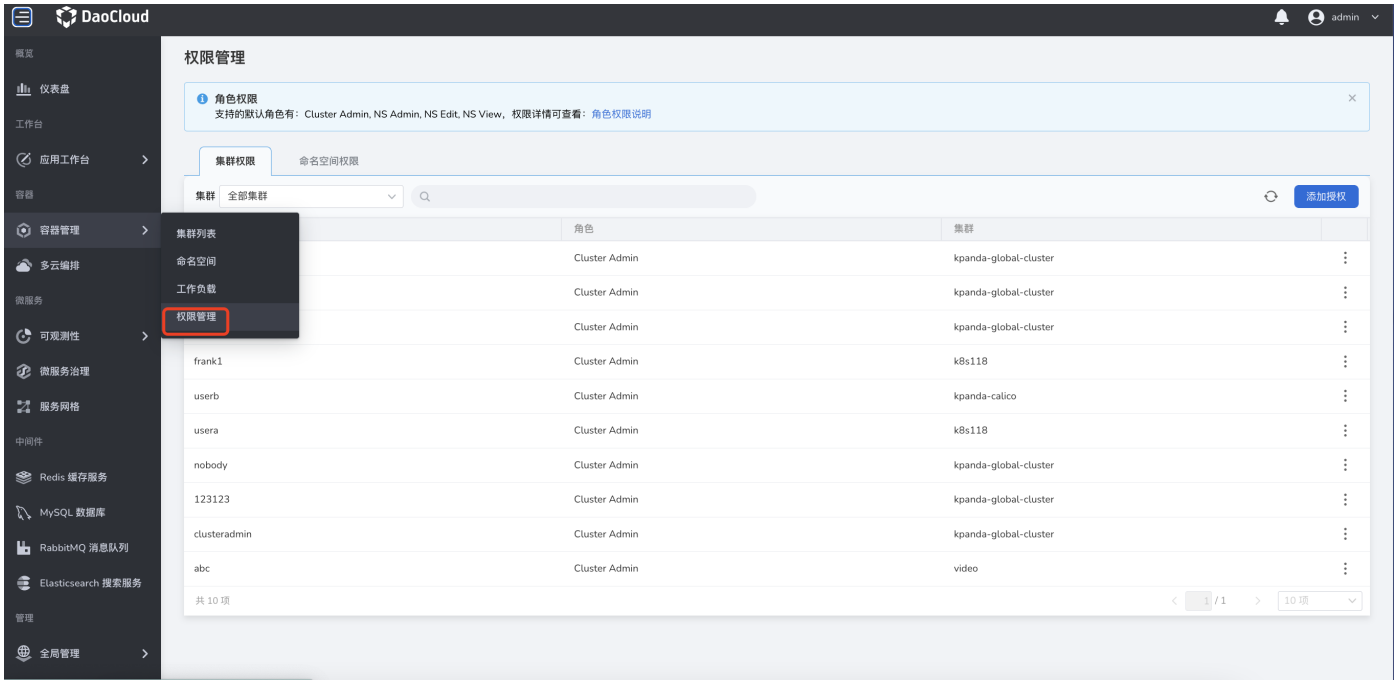
前提条件

给用户/用户组授权之前，请完成如下准备：

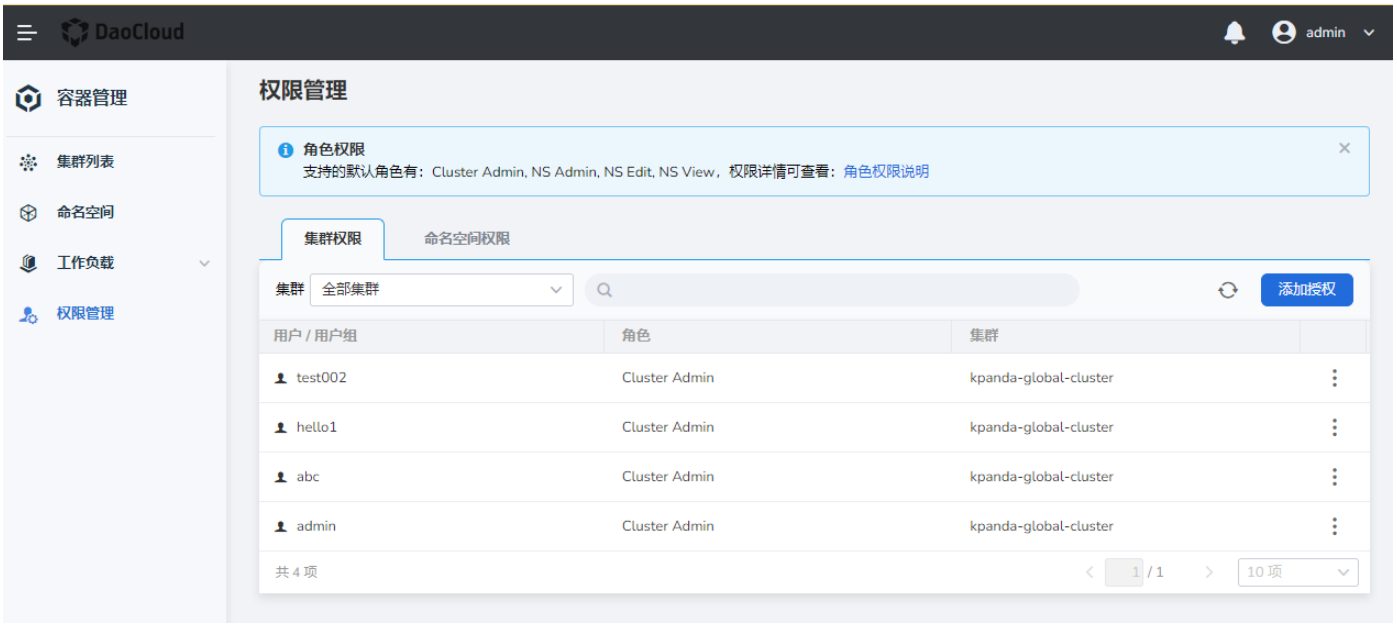
- 已在全局管理中创建了待授权的用户/用户组，请参考用户。
- 仅 Kpanda Owner 及当前集群的 Cluster Admin 具备集群授权能力。详情可参考权限说明。
- 仅 Kpanda Owner、当前集群的 Cluster Admin，当前命名空间的 NS Admin 具备命名空间授权能力。

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。



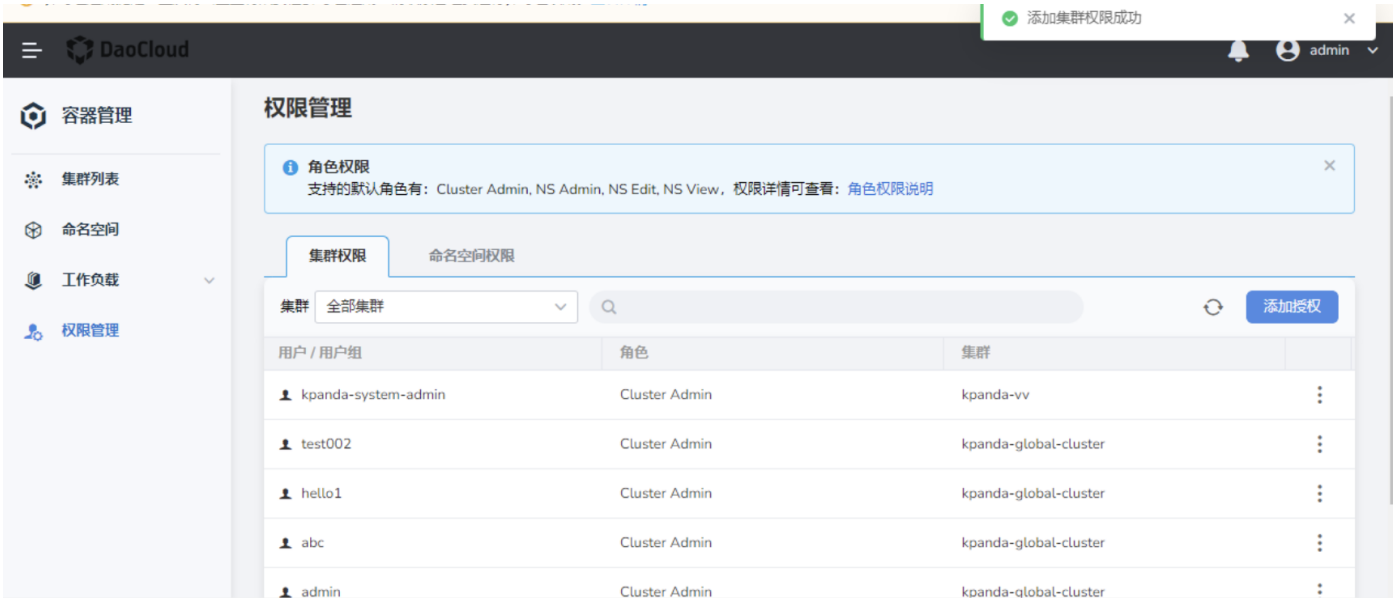
2. 点击 添加授权 按钮。



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 。

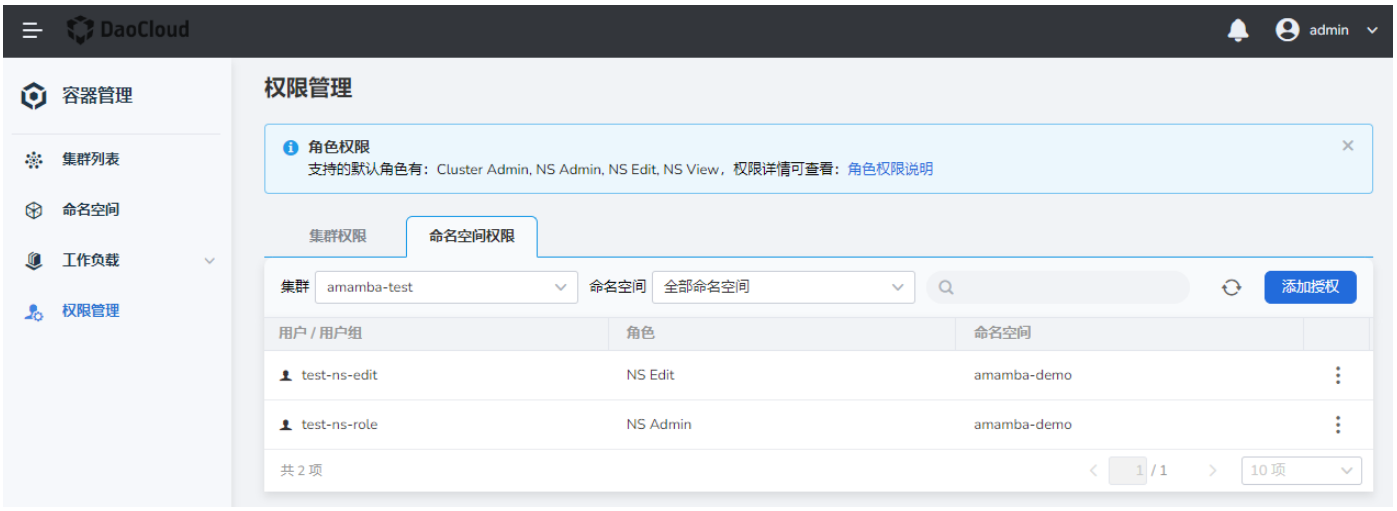
   目前仅支持的集群角色为 **Cluster Admin** ，详情权限可参考权限说明。如需要给多个用户/用户组同时进行授权， 可点击 添加用户权限 进行多次添加。

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 。



命名空间授权

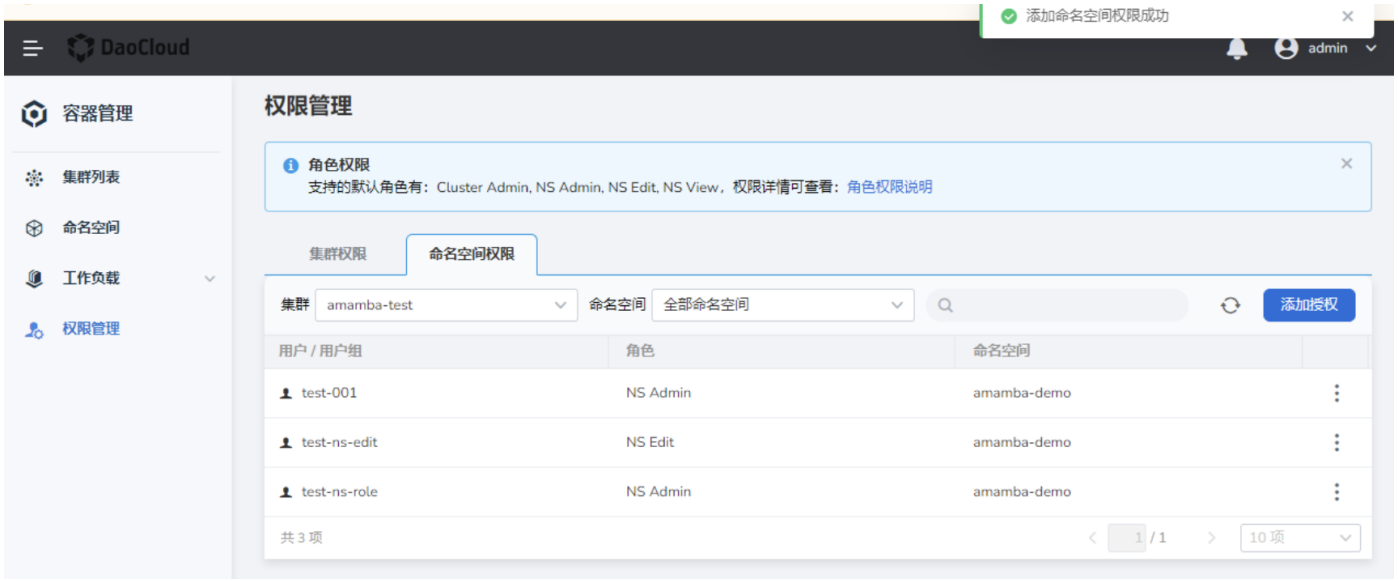
1. 用户登录平台后，点击左侧菜单栏 容器管理 下的 权限管理 ，点击 命名空间权限 页签。



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 。

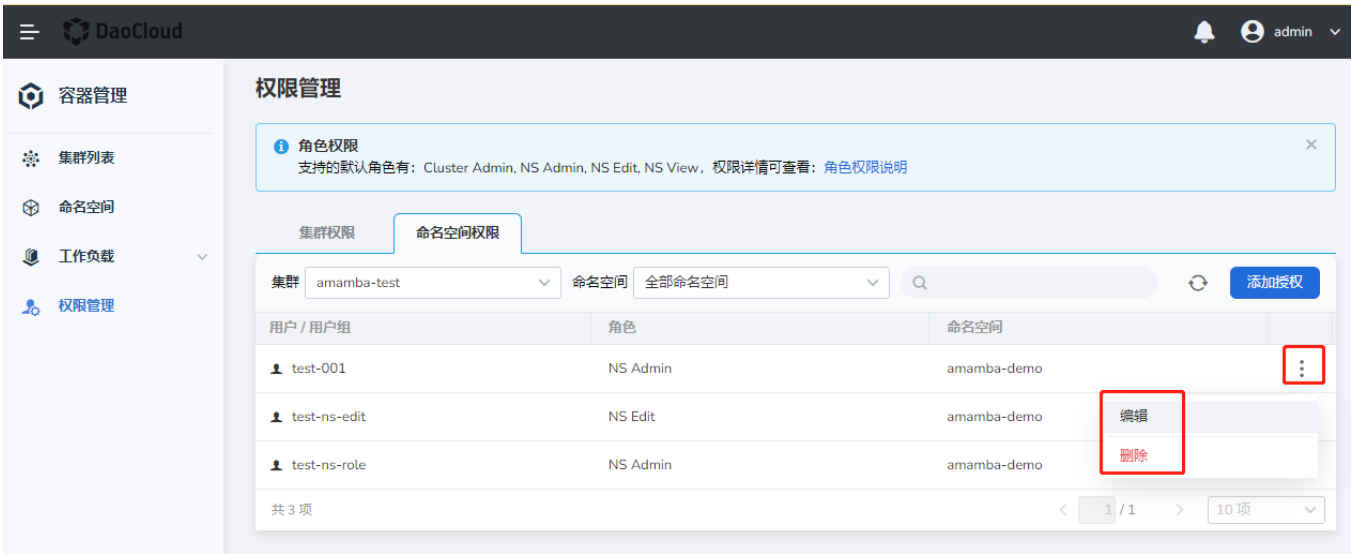
   目前支持的命名空间角色为 NS Admin、NS Edit、NS View，详情权限可参考权限说明。如需给多个用户/用户组同时进行授权，可点击 添加用户权限 进行多次添加。点击 确定 完成权限授权。

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 。



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 。



节点管理

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	推荐
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	
x86	UOS Linux	
x86	openEuler	

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 **。

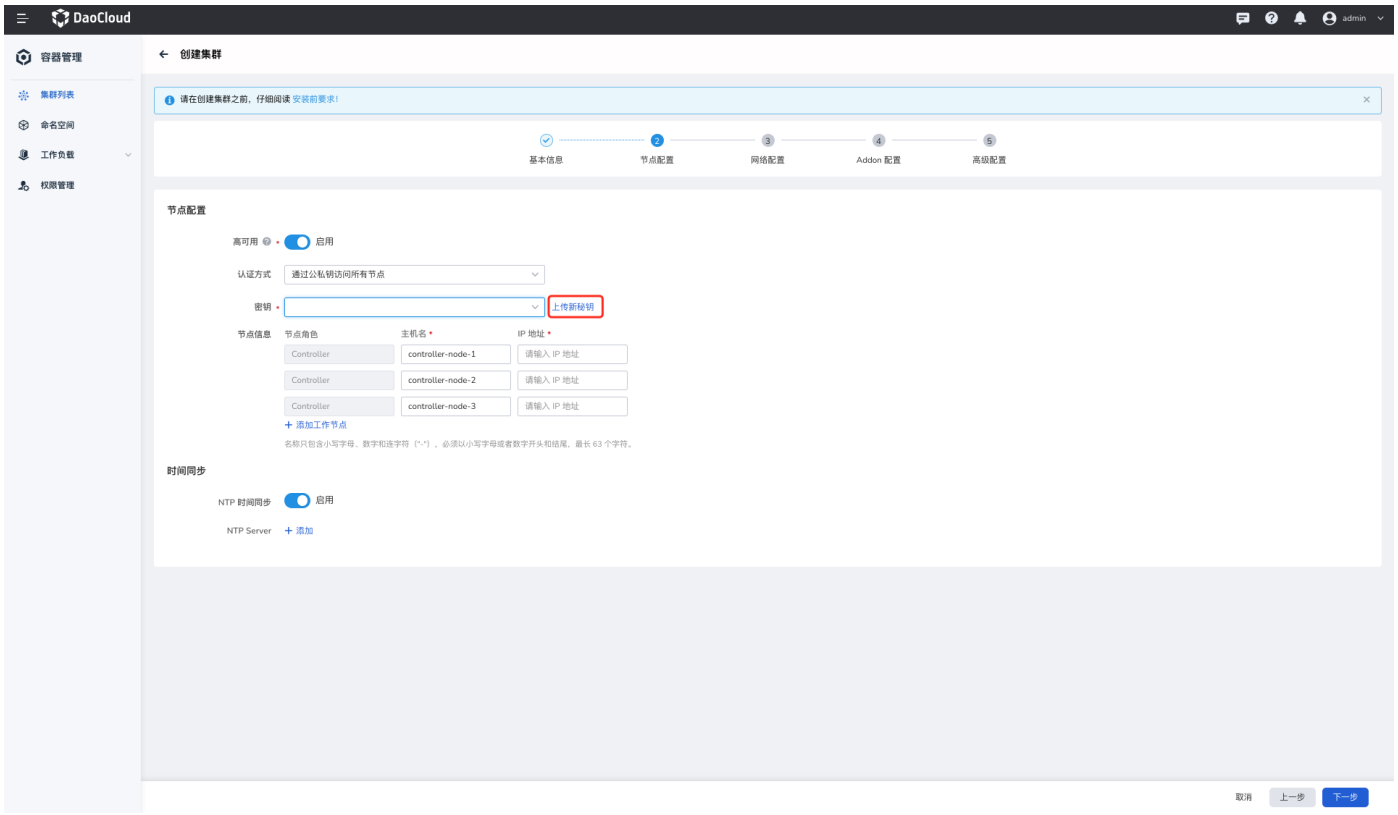
4. 执行如下命令，查看步骤 1 所创建的私钥文件 **id_rsa** 。

```
cat /root/.ssh/id_rsa
```

输出如下内容：

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEA3UvyKINzY5BFuemQ+uJ6q+GqgfvnWwNC8HzZhpcMSjJy26MM
UtBEBJxy8fMi57XcjYxPibXW/wnd+32ICCycqCwByUmuXeCC1cjlCQDqjcAvXae7
Y54IXGF7wm2IsMNwf0kjFEXjuS48FLDA0mGRaN3BG+Up5geXcHckg3K5LD8kXFFx
dEmSIjdyw55NaUitmEdHzN7cIdfi6Z56jcV8dcFBgWKUx+ebiyPmZBkXToz6GnMF
rswzzZCl+G6Jb2xTGy7g7ozb4BoZd1IpSD5EhDanRrESVE0C5YuJ5zUAC0CvVd1l
v67AK8Ko6MXToHp01/bcsvlM6cqgwUFXZKVeOwIDAQABAoIBAQCO36GQlo3BEjxy
M2HvGJmqrx+unDxafliRe4nVY2AD515Qf4xNSzke4QM1QoyenMOwf446krQkJPK0
k+9nl6Xszby5gGCbK4BNFk8I6RaGPjZWeRx6zGUJf8avWJiPxx6yjz2esSC9RiR0
F0nmiiefVMyAfgv2/5++dK2WUFNNRKLgSRRpP5bRaD5wMzzxtSSXrUon6217HO8p
3RoWsI51MbVzhdVgpHUNABcoa0rpr9svT6XLKZxY8mxpKFYjM0Wv2JIDABg3kBvh
QbJ7kStCO3naZjKMU9UuSqVJs06cflGYw7Or8/tABR3LErNQKPjkhAQqt0DXw7Iw
3tKdTAJBAoGBAP687U7JAOqQkcphek2E/A/sbO/d37ix7Z3vNOy065STrA+ZWMZn
pZ6Ui1B/oJpoZssnfvIoz9sn559X0j67TljFALFd2ZGS0Fqh9KVCqDvfk+Vst1dq
+3r/yZdTOyswoccxkJiC/GDwZGK0amJWqvob39JCZhDAKIGLbGMmjdAHAoGBAN5k
m1WGnni1nZ+3dryIwgB6z1hWcnLTamzSET6KhSuo946ET0IRG9xtlheCx6dqICbr
Vk1Y4NtRZjK/p/YGx59rDWf7E3I8ZMgR7mjieOcUZ4lUlA4l7ZIlW/2WZHW+nUXO
Ti20fqJ8qSp4BUvOvuth1pz2GLUHe2/Fxjf7HIstAoGBAPHpPr9r+TfIlPsJeRj2
6lzA3G8qWFRQfGRYjv0fjv0pA+RIb1rzgP/I90g5+63G6Z+R4WdcxI/OJJNY1iuG
uw9n/pFxm7U4JC990BPE6nj5iLz+clpNGYckNDBF9VG9vFSrSDLdaYkxoVNvG/xJ
a9Na90H4lm7f3VewrPy310KvAoGAZr+mwNoEh5Kpc6xo8Gxi7aPP/mlaUVD6X7Ki
gvmu02AqmC7rC4QqEiqTaONkaSXwGusqIWxJ3yp5hELmUBYLzszAEeV/s4zRp1oZ
g133LBRSTbHFAdBmNdqK6Nu+KGRb92980UMOKvZbliKDl+W6cbfvVu+gtKrzTc3b
aevb4TUCgYEAnJAxyVYDP1nJf7bjBSHXQu1E/DMwbtrqw7dylRJ8cAzI7IxfSCez
7BYWq41PqVd9/zrb3Pbh2phiVzKe783igAIMqummcjo/kZyCwFsYBzK77max1jF5
aPQsLbRS2aDz8kIH6jHPZ/R+15EROmdtLmA7vIJZGerWWQR0dUU+XXA=
```

将私钥内容复制后填至界面密钥输入框。


DaoCloud
admin
容器管理
集群列表
命名空间
工作负载
权限管理
创建集群
请在创建集群之前，仔细阅读 安装前要求！
基本信息
节点配置
网络配置
Addon 配置
高级配置
节点配置
高可用
启用
认证方式
通过公私钥访问所有节点
密钥
上传新秘钥
节点信息
节点角色
主机名 *
IP 地址 *
Controller
controller-node-1
controller-node-2
controller-node-3
请输入 IP 地址
添加工作节点
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。
时间同步
NTP 时间同步
启用
NTP Server
添加
取消
上一步
下一步

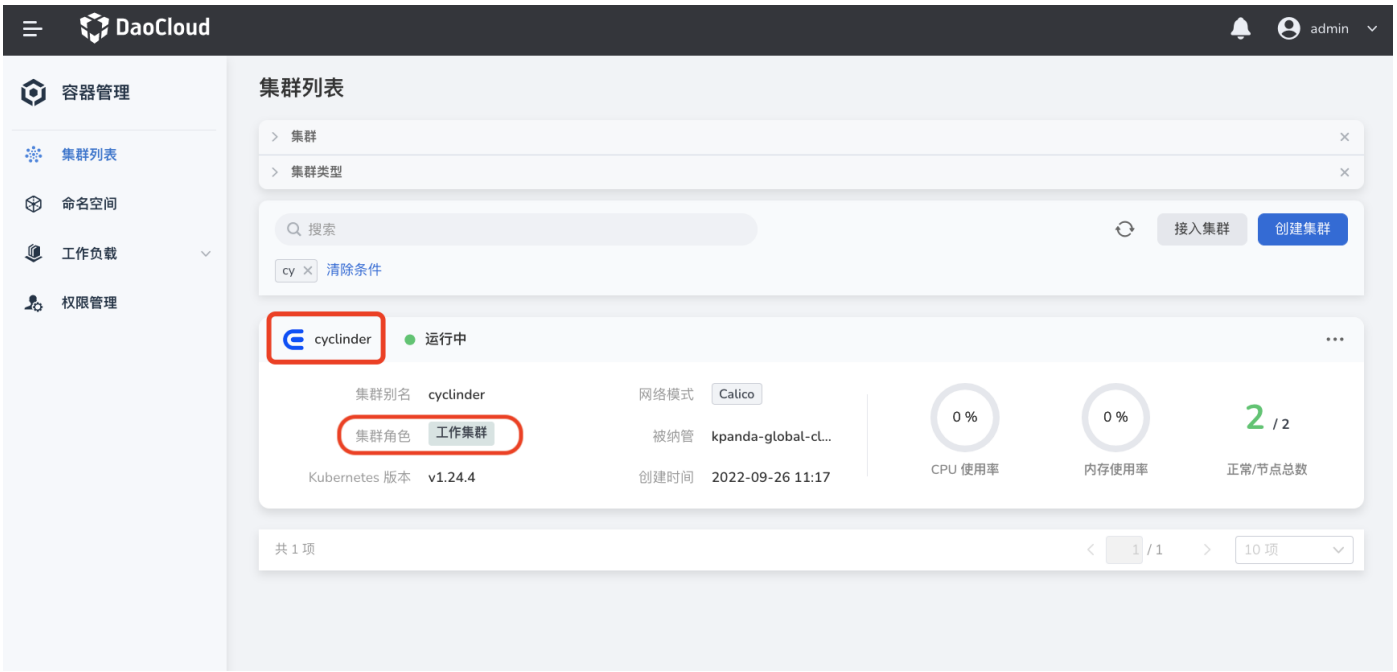
集群节点扩容

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。

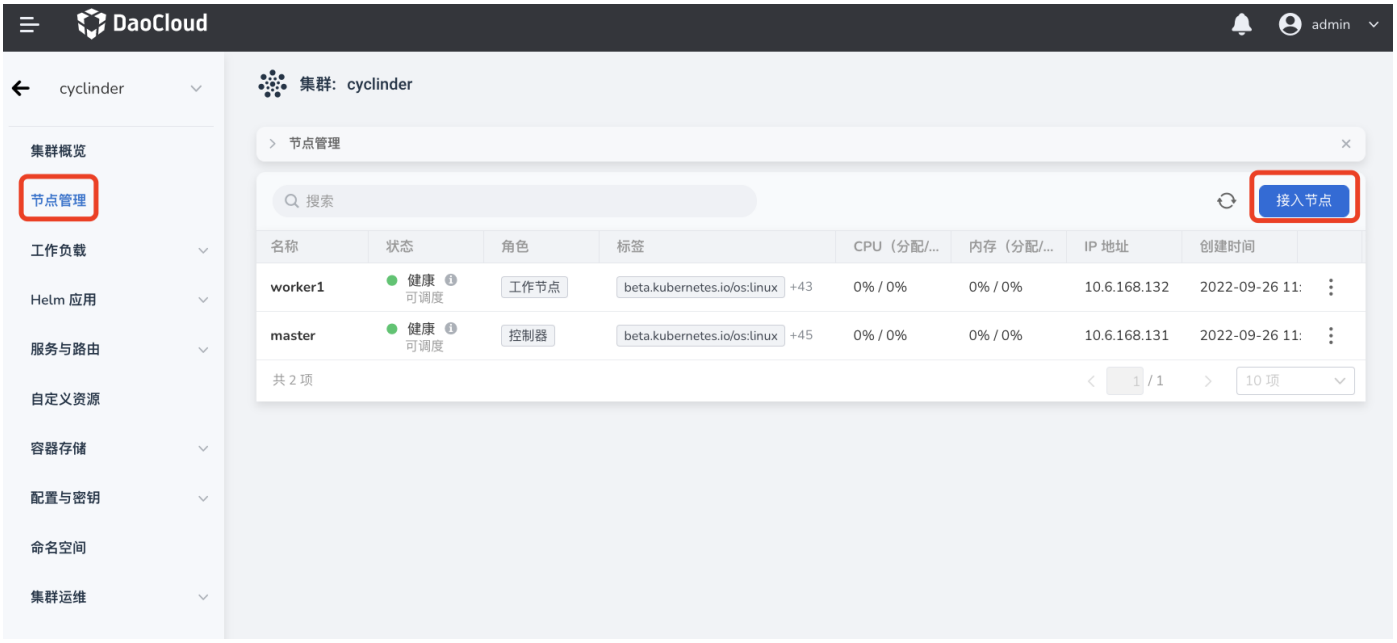
只有通过容器管理模块创建的集群才支持节点扩缩容，从外部接入的集群不支持此操作。 本文主要介绍同种架构下工作集群的 工作节点 扩容，如需为集群增加控制节点或异构工作节点， 请参阅对工作集群的控制节点扩容、为工作集群添加异构节点、为全局服务集群的工作节点扩容。

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。

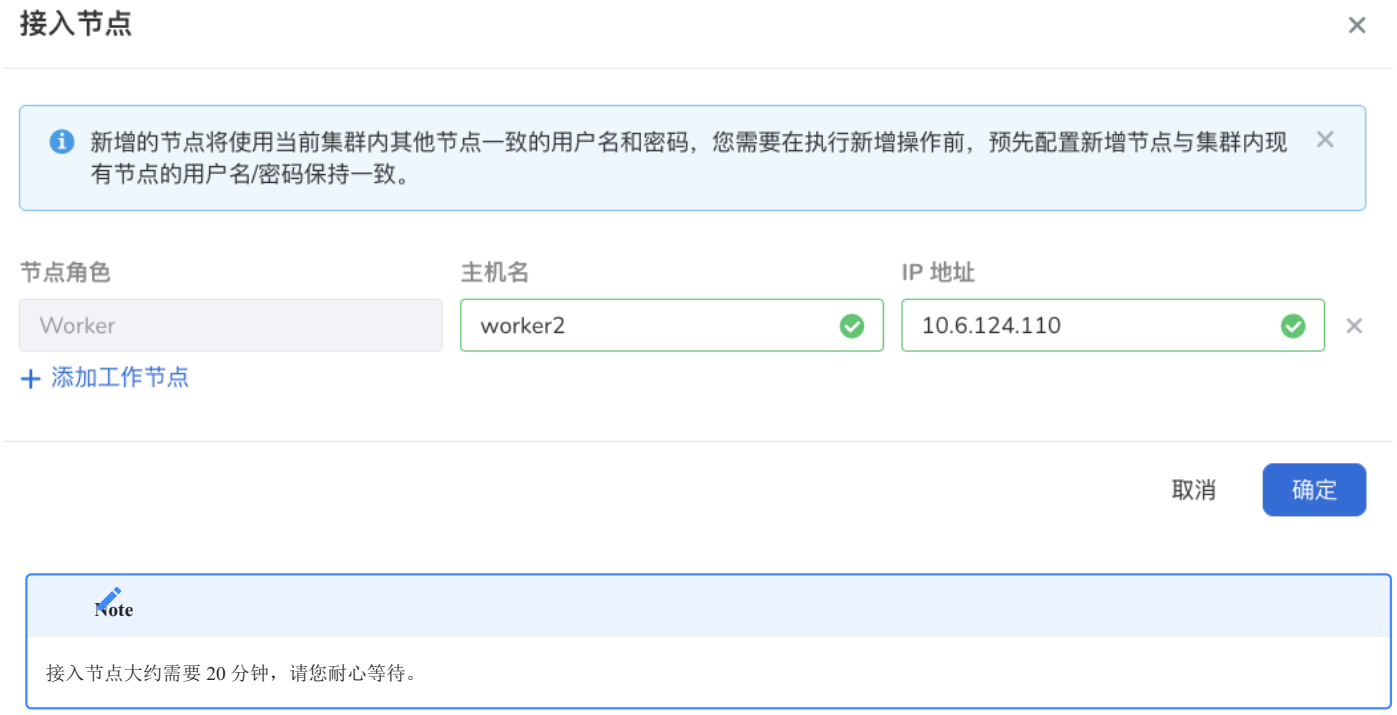


2. 在左侧导航栏点击 节点管理 ，然后在页面右上角点击 接入节点 。



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。



Note

接入节点大约需要 20 分钟，请您耐心等待。

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。

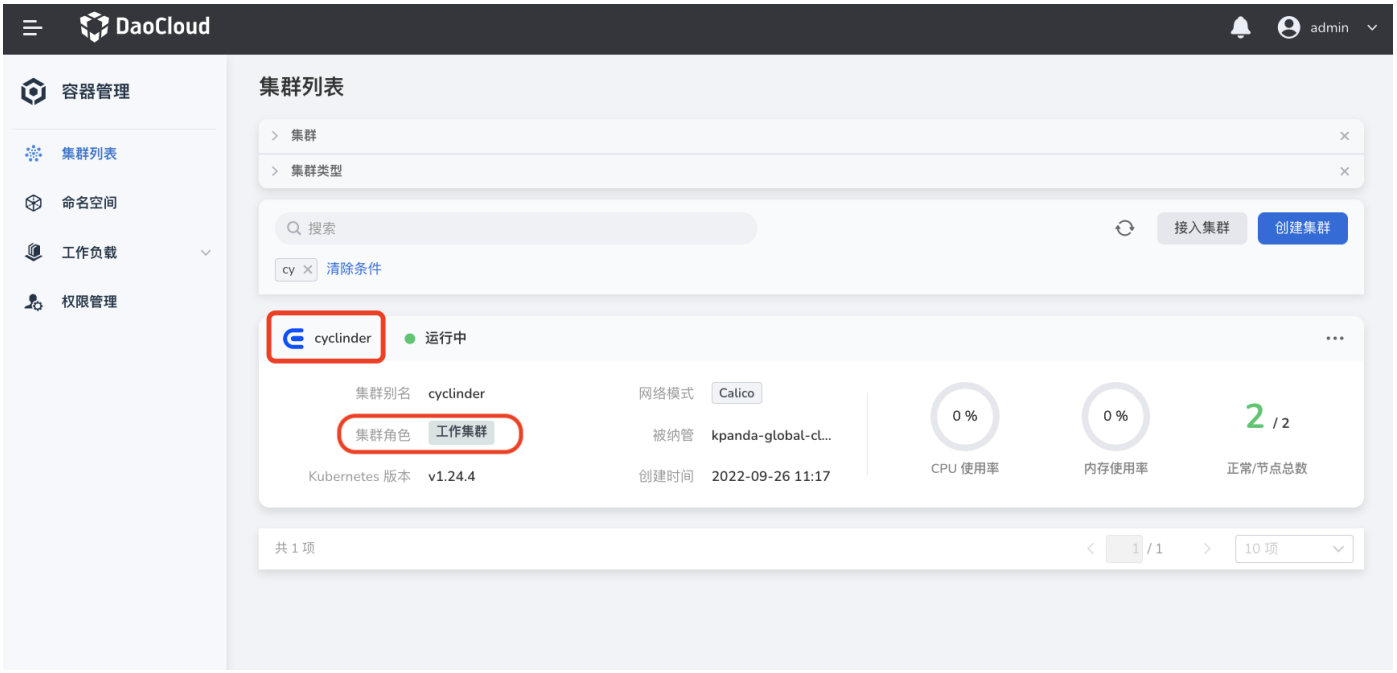
注意事项

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。

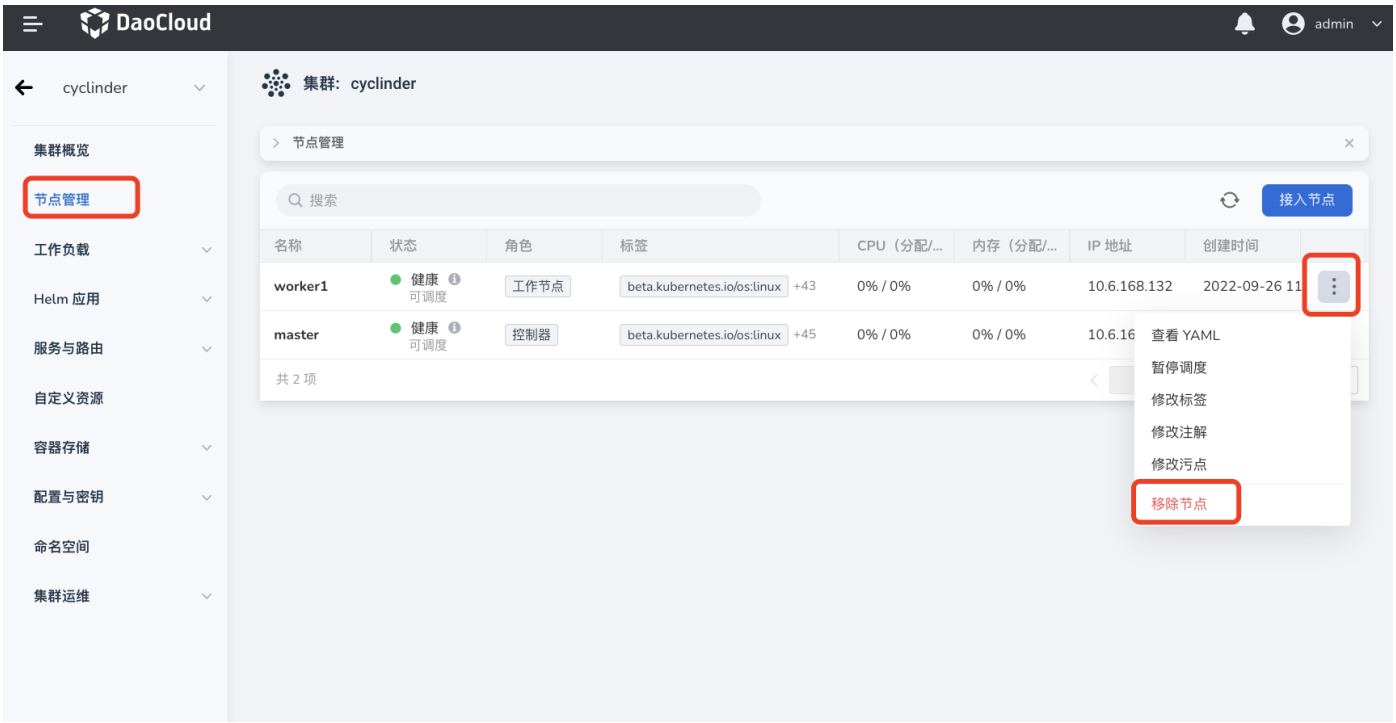
操作步骤

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。



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 。



3. 输入节点名称,并点击 删除 进行确认。

节点污点管理

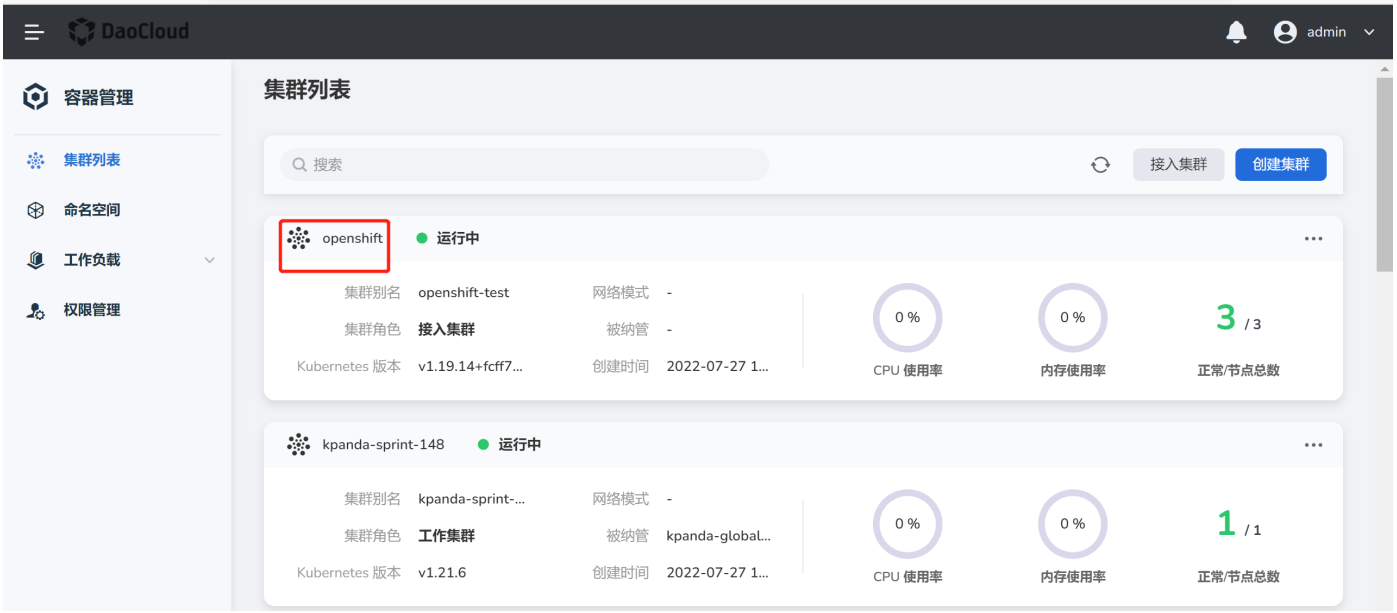
污点 (Taint) 能够使节点排斥某一类 Pod，避免 Pod 被调度到该节点上。 每个节点上可以应用一个或多个污点，不能容忍这些污点的 Pod 则不会被调度该节点上。 有关污点的更多详情，可参考 Kubernetes 官方文档污点和容忍度。

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。



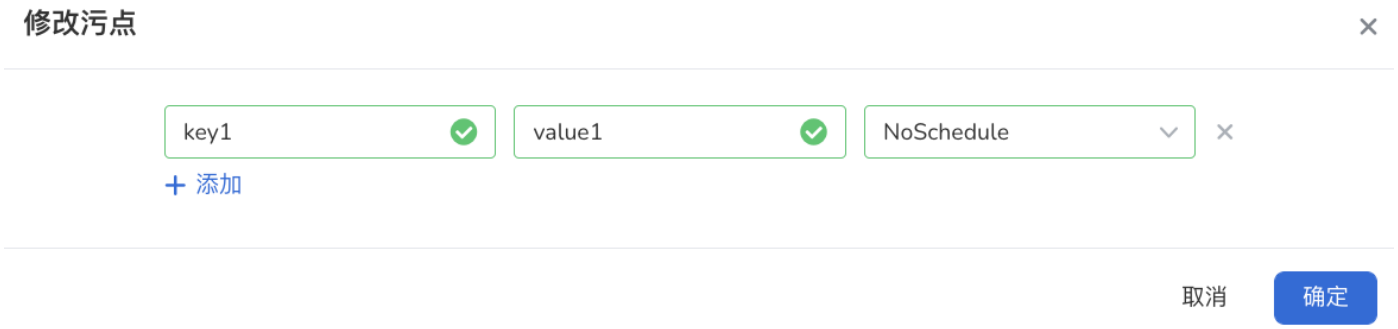
2. 在左侧导航栏，点击 节点管理 ，找到需要修改污点的节点，点击右侧的 ⋮ 操作图标并点击 修改污点 按钮。

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：

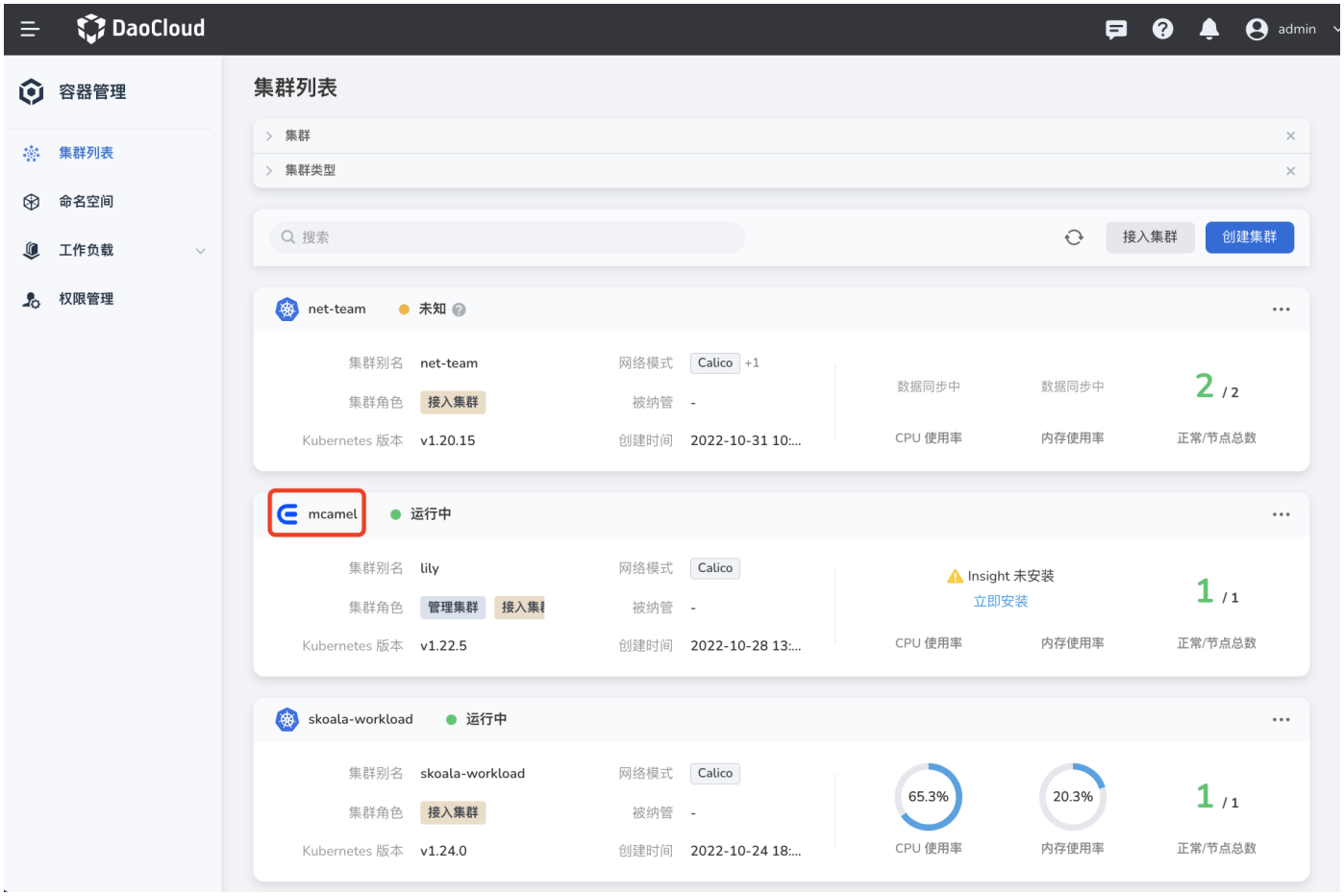
- NoSchedule：不能容忍某个污点的 Pod 不会被调度到存在该污点的节点上。
- PreferNoSchedule：尽量避免 将不能容忍某个污点的 Pod 不会被调度到存在该污点的节点上。
- NoExecute：保持现状。如果不能容忍某个污点的 Pod 在节点设置污点之前，已经运行在该节点上，也不会驱逐该 Pod。如果不能容忍某个污点的 Pod 在节点设置污点之前，还未运行在该节点上，则不会被调度到该节点。


修改污点
key1
value1
NoSchedule
添加
取消
确定

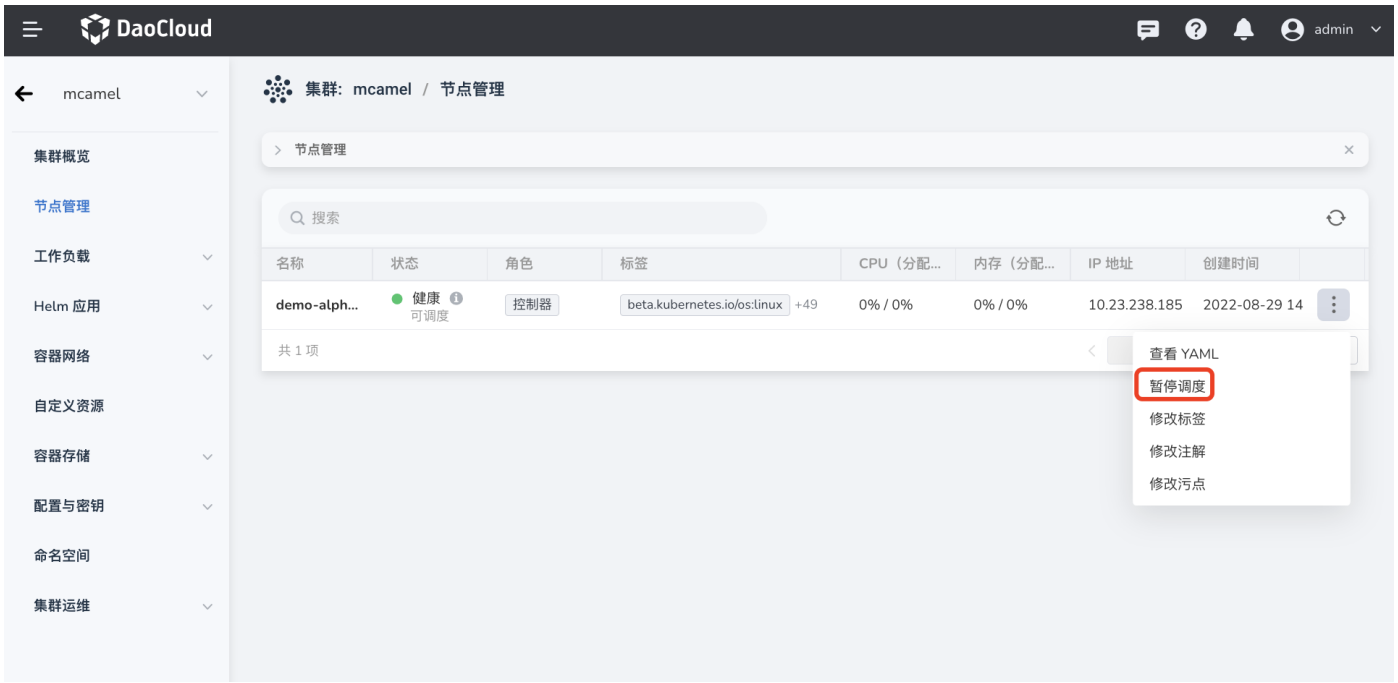
节点调度

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。

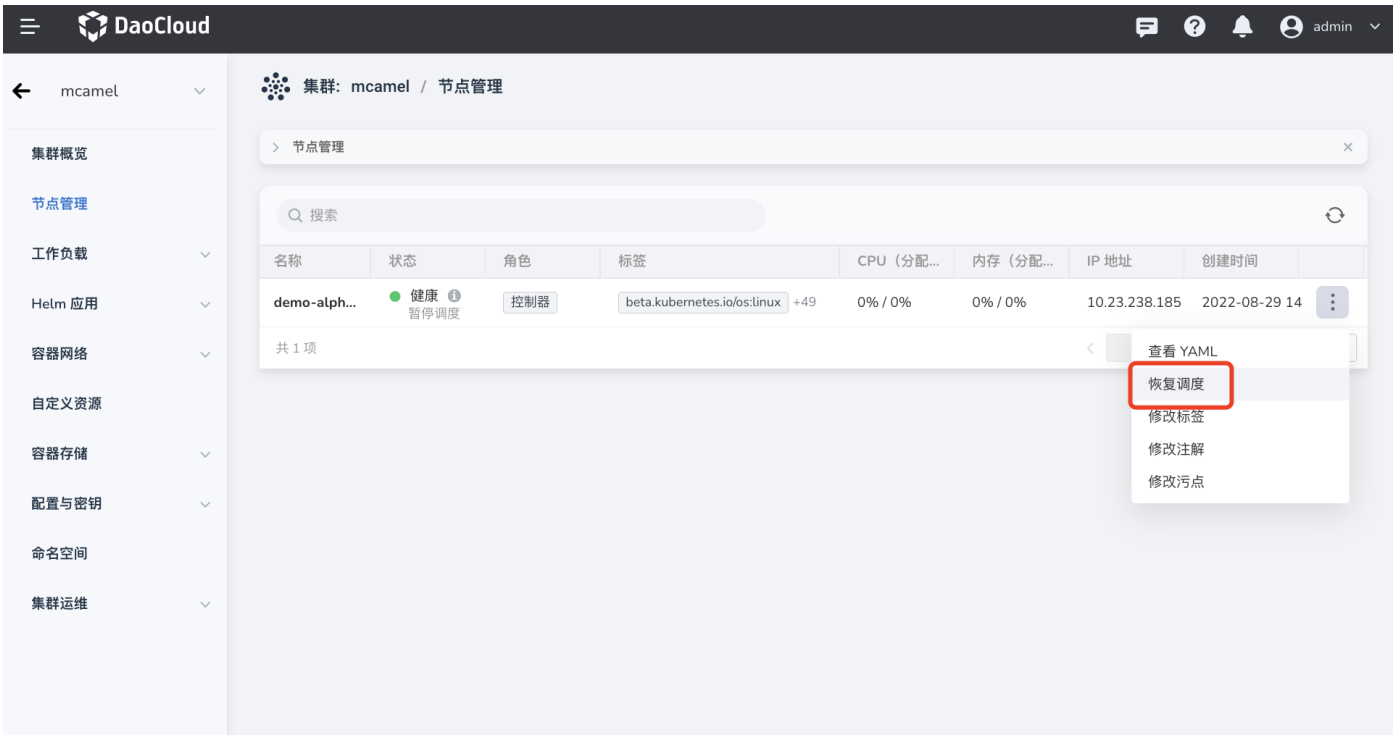
1. 在 集群列表 页面点击目标集群的名称。



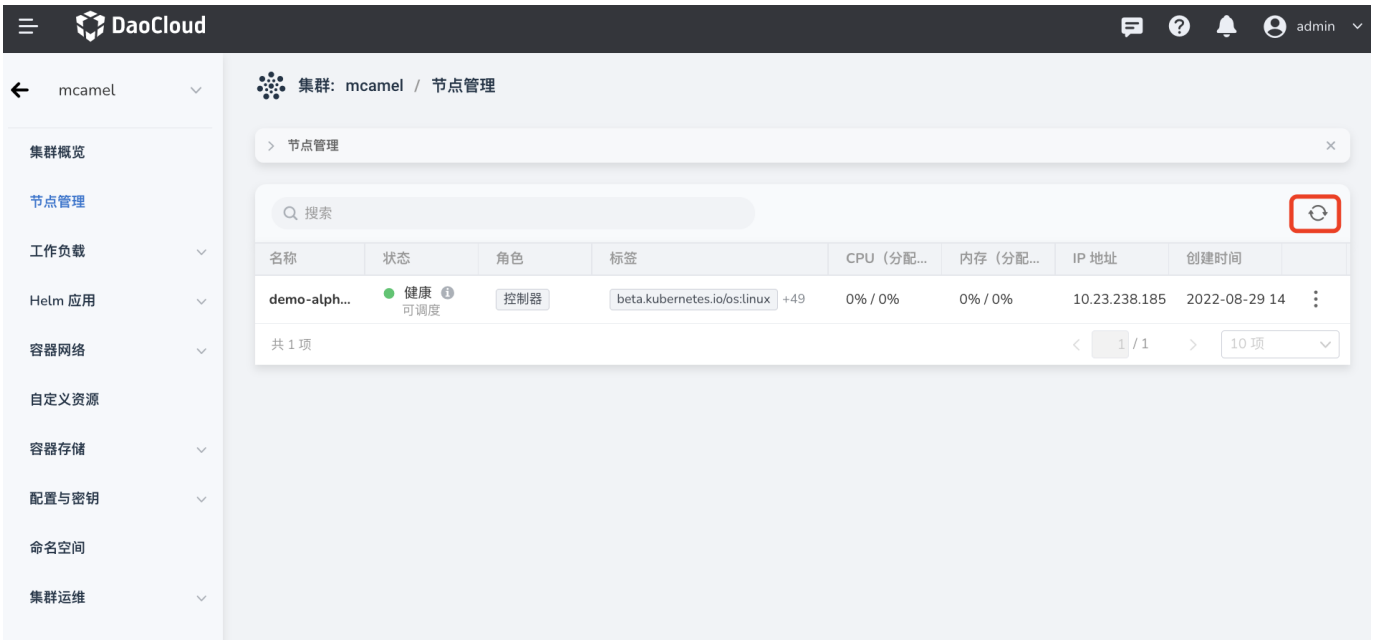
2. 在左侧导航栏点击 节点管理 ，在节点右侧点击 ⋮ 操作图标，点击 暂停调度 按钮即可暂停调度该节点。



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。



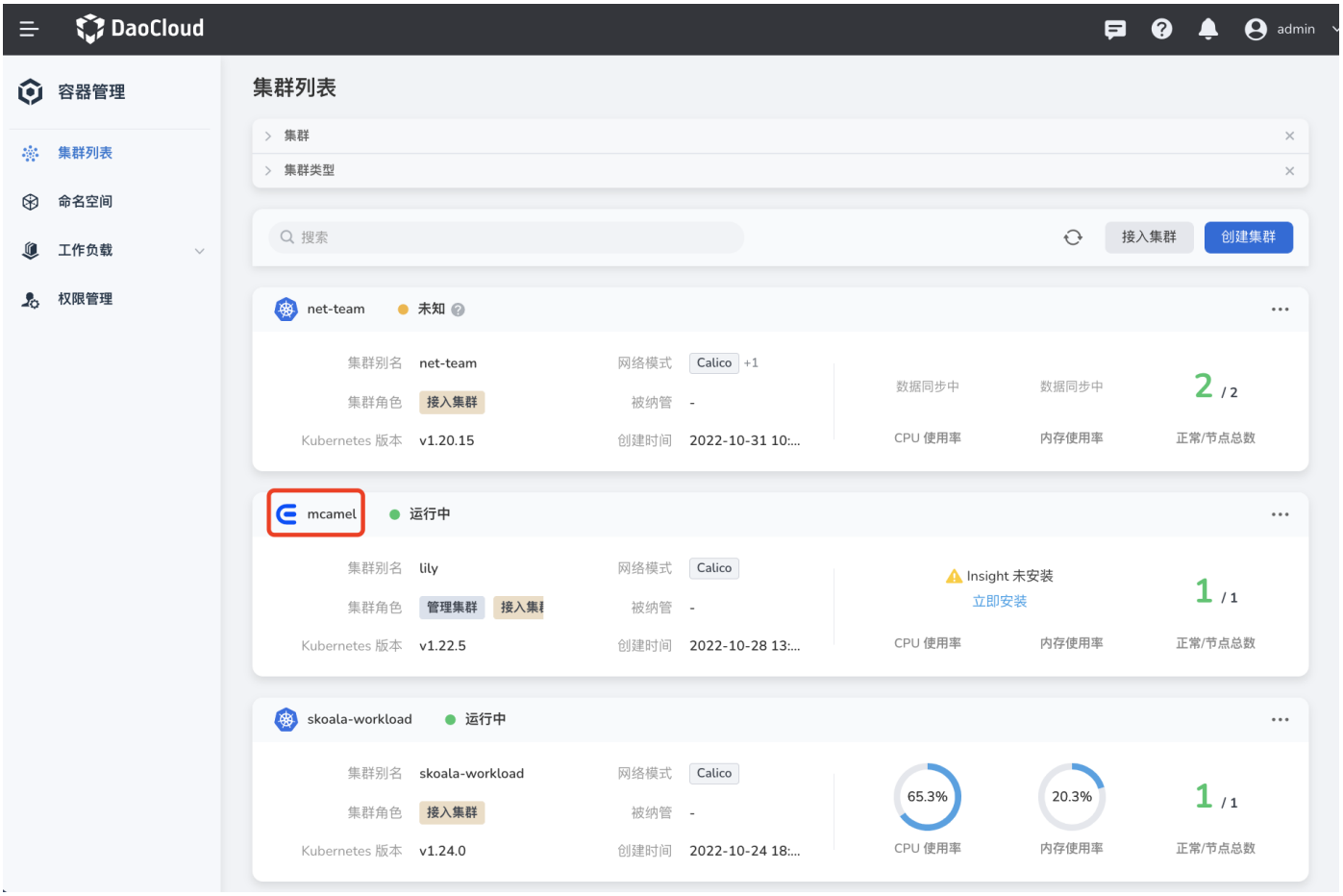
节点调度状态可能因网络情况有所延迟，点击搜索框右侧的刷新图标可以刷新节点调度状态。



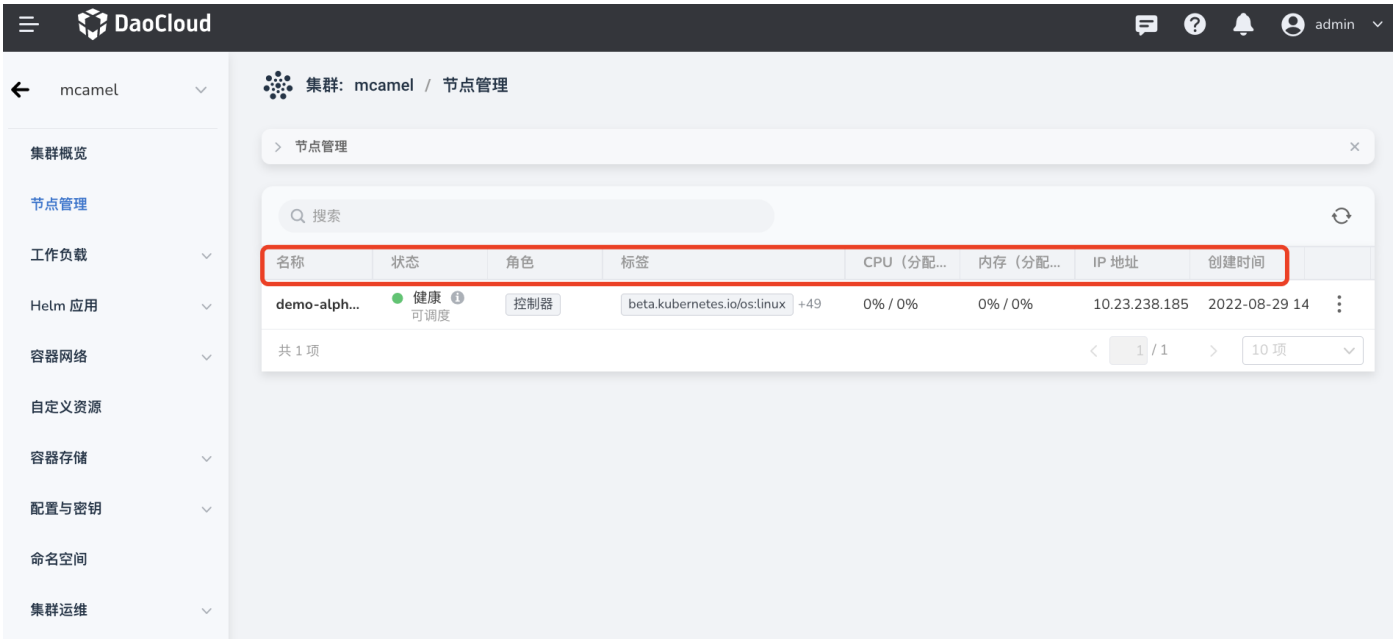
节点详情

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。

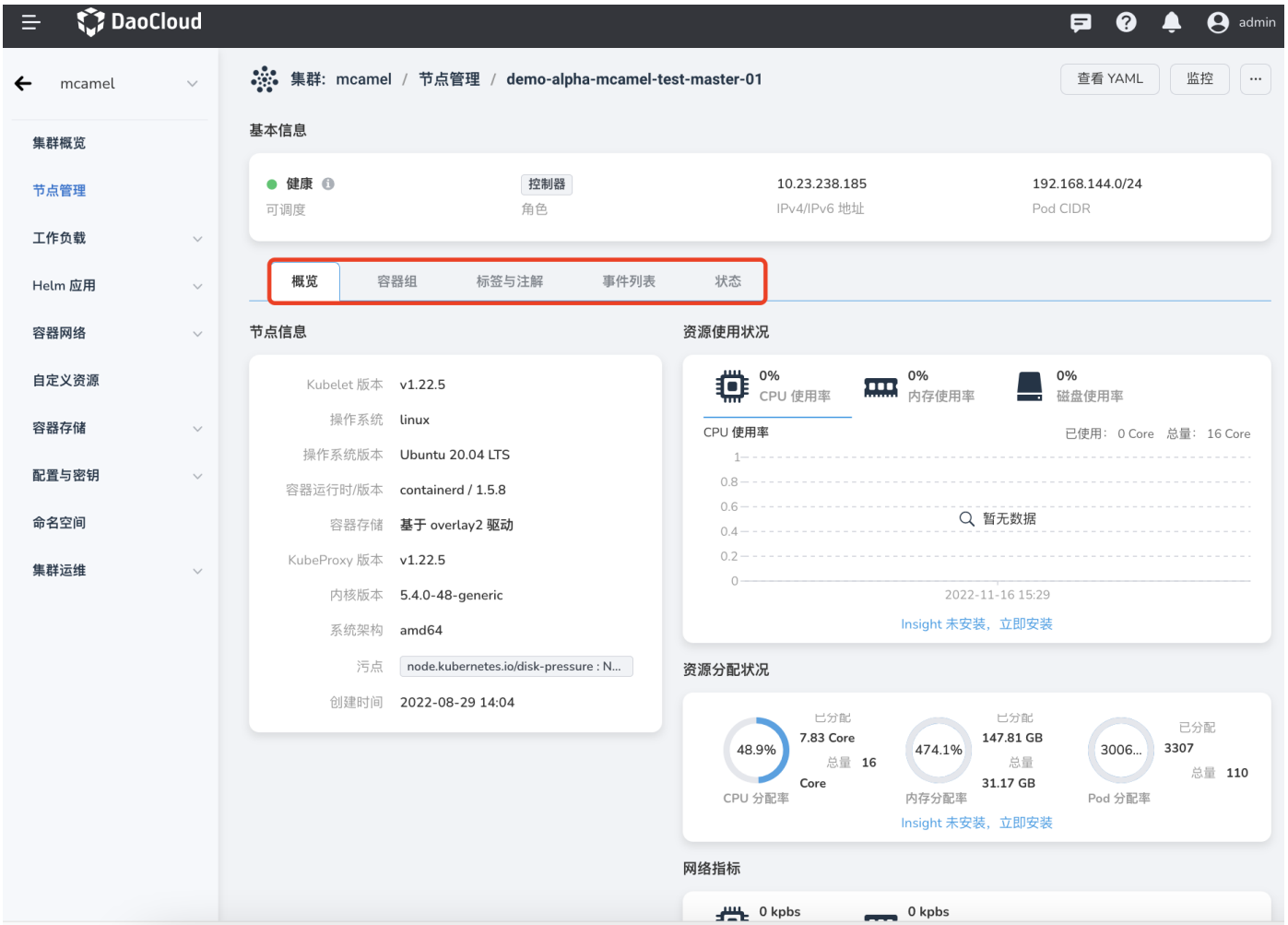
1. 在 集群列表 页面点击目标集群的名称。



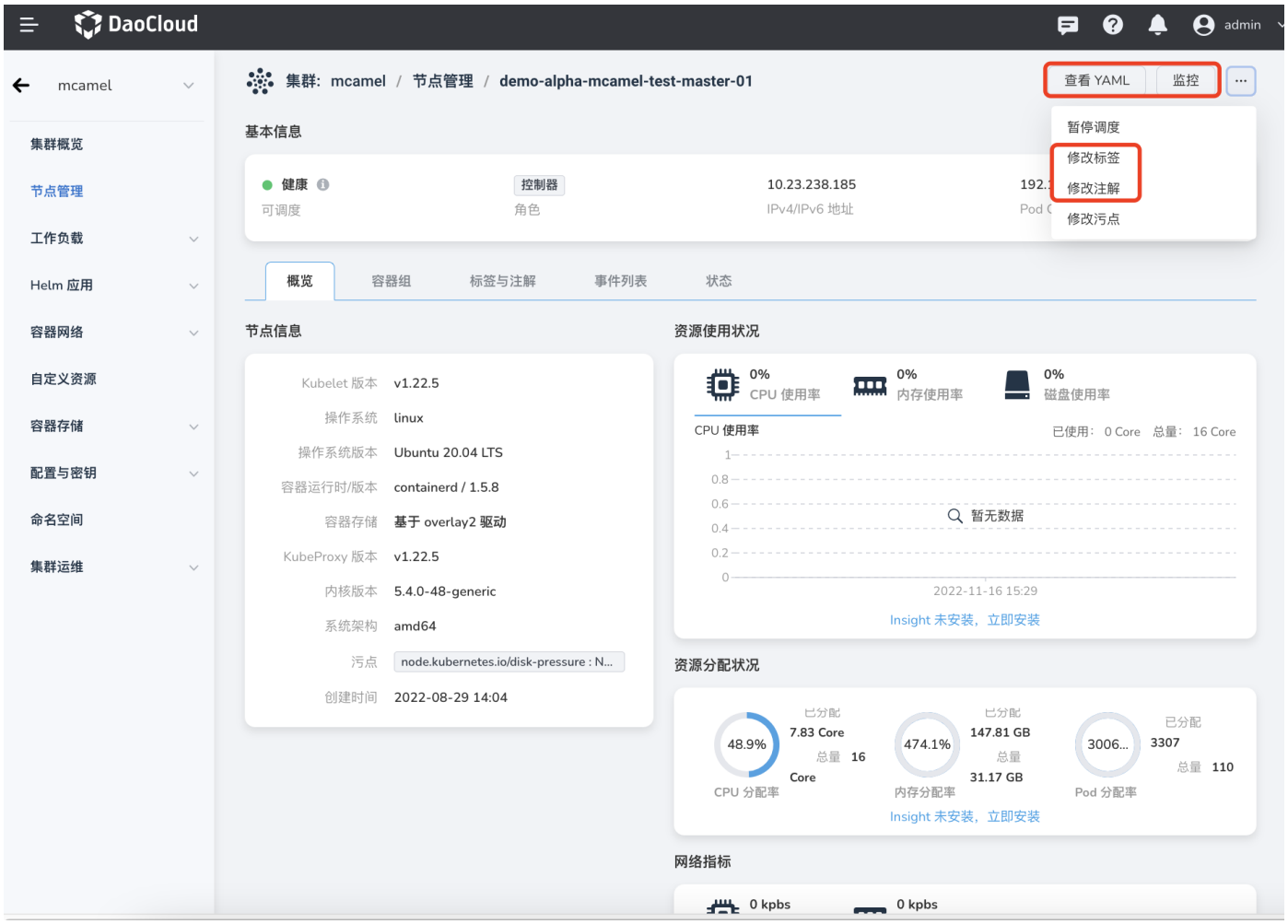
2. 在左侧导航栏点击 节点管理 ，可以查看节点状态、角色、标签、CPU/内存使用情况、IP 地址、创建时间。



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。



此外，还可以查看节点的 YAML 文件、监控信息、标签和注解等。


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 / demo-alpha-mcamel-test-master-01
查看 YAML
监控
暂停调度
修改标签
修改注解
修改污点
基本信息
健康
可调度
控制器
角色
10.23.238.185
IPv4/IPv6 地址
概览
容器组
标签与注解
事件列表
状态
节点信息
Kubelet 版本 v1.22.5
操作系统 linux
操作系统版本 Ubuntu 20.04 LTS
容器运行时/版本 containerd / 1.5.8
容器存储 基于 overlay2 驱动
KubeProxy 版本 v1.22.5
内核版本 5.4.0-48-generic
系统架构 amd64
污点 node.kubernetes.io/disk-pressure : N...
创建时间 2022-08-29 14:04
资源使用状况
0% CPU 使用率
0% 内存使用率
0% 磁盘使用率
CPU 使用率
已使用: 0 Core 总量: 16 Core
暂无数据
2022-11-16 15:29
Insight 未安装，立即安装
资源分配状况
48.9%
已分配 7.83 Core
总量 16 Core
CPU 分配率
474.1%
已分配 147.81 GB
总量 31.17 GB
内存分配率
3006...
已分配 3307
总量 110
Pod 分配率
Insight 未安装，立即安装
网络指标
0 kpbs
0 kpbs

标签与注解

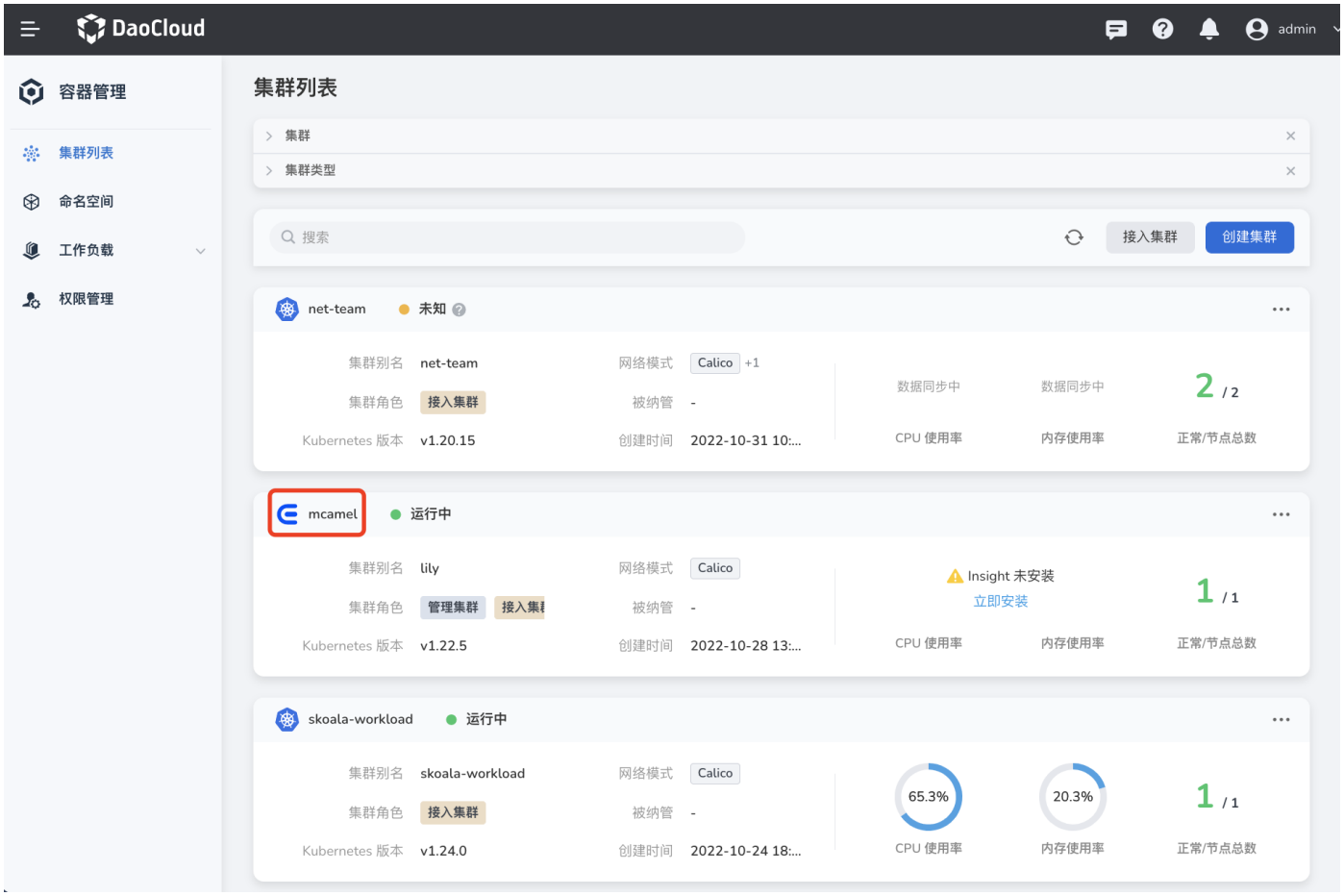
标签（Labels）是为 Pod、节点、集群等 Kubernetes 对象添加的标识性键值对，可结合标签选择器查找并筛选满足某些条件的 Kubernetes 对象。每个键对于给定对象必须是唯一的。

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。

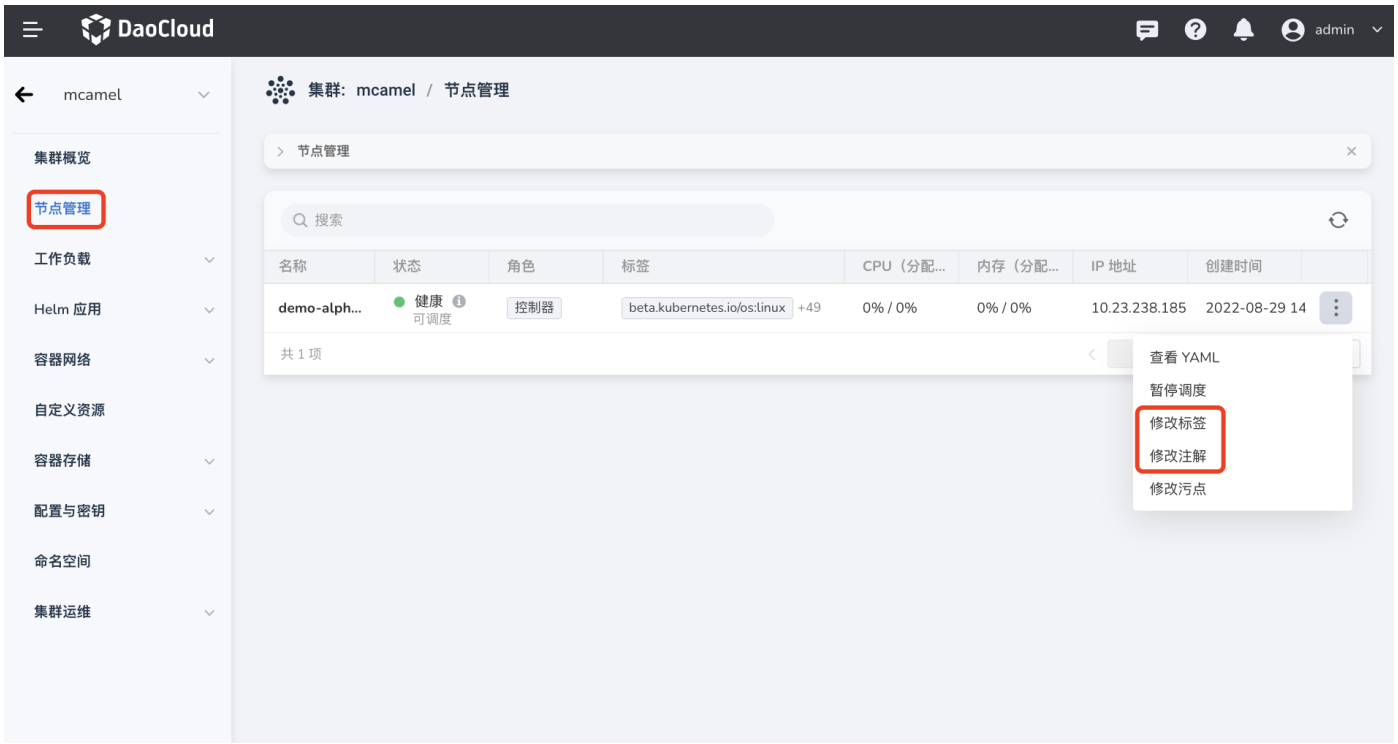
有关标签和注解的更多信息，可参考 Kubernetes 的官方文档标签和选择算符 或注解。

添加/删除标签与注解的步骤如下：

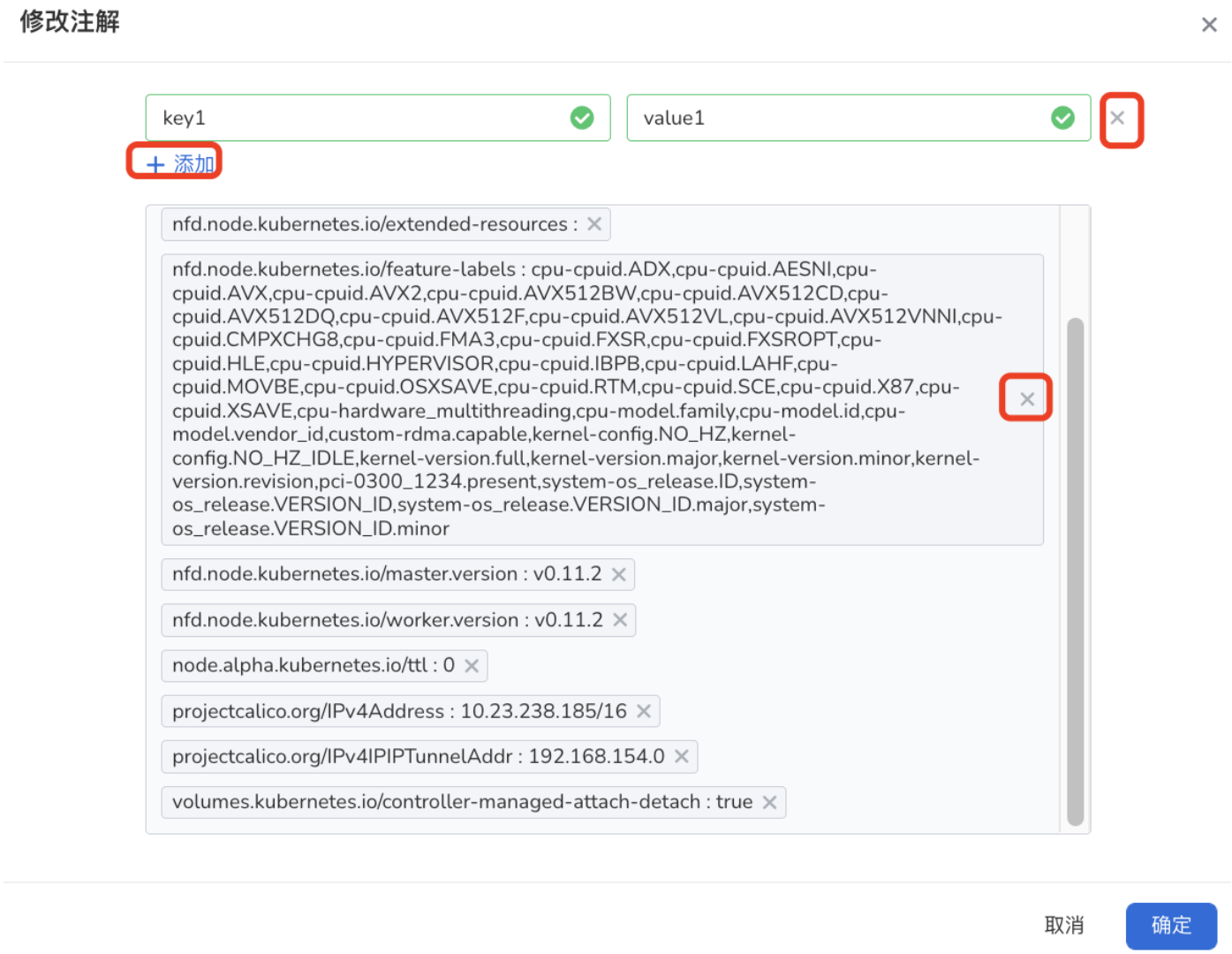
1. 在 集群列表 页面点击目标集群的名称。



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 。



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 。


修改注解
key1
value1
添加
nfd.node.kubernetes.io/extended-resources :
nfd.node.kubernetes.io/feature-labels : cpu-cpuid.ADX,cpu-cpuid.AESNI,cpu-cpuid.AVX,cpu-cpuid.AVX2,cpu-cpuid.AVX512BW,cpu-cpuid.AVX512CD,cpu-cpuid.AVX512DQ,cpu-cpuid.AVX512F,cpu-cpuid.AVX512VL,cpu-cpuid.AVX512VNNI,cpu-cpuid.CMPXCHG8,cpu-cpuid.FMA3,cpu-cpuid.FXSR,cpu-cpuid.FXSROPT,cpu-cpuid.HLE,cpu-cpuid.HYPERVISOR,cpu-cpuid.IBPB,cpu-cpuid.LAHF,cpu-cpuid.MOVBE,cpu-cpuid.OSXSAVE,cpu-cpuid.RTM,cpu-cpuid.SCE,cpu-cpuid.X87,cpu-cpuid.XSAVE,cpu-hardware_multithreading,cpu-model.family,cpu-model.id,cpu-model.vendor_id,custom-rdma.capable,kernel-config.NO_HZ,kernel-config.NO_HZ_IDLE,kernel-version.full,kernel-version.major,kernel-version.minor,kernel-version.revision,pci-0300_1234.present,system-os_release.ID,system-os_release.VERSION_ID,system-os_release.VERSION_ID.major,system-os_release.VERSION_ID.minor
nfd.node.kubernetes.io/master.version : v0.11.2
nfd.node.kubernetes.io/worker.version : v0.11.2
node.alpha.kubernetes.io/ttl : 0
projectcalico.org/IPv4Address : 10.23.238.185/16
projectcalico.org/IPv4IPIPTunnelAddr : 192.168.154.0
volumes.kubernetes.io/controller-managed-attach-detach : true
取消
确定

工作负载

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 d.run 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

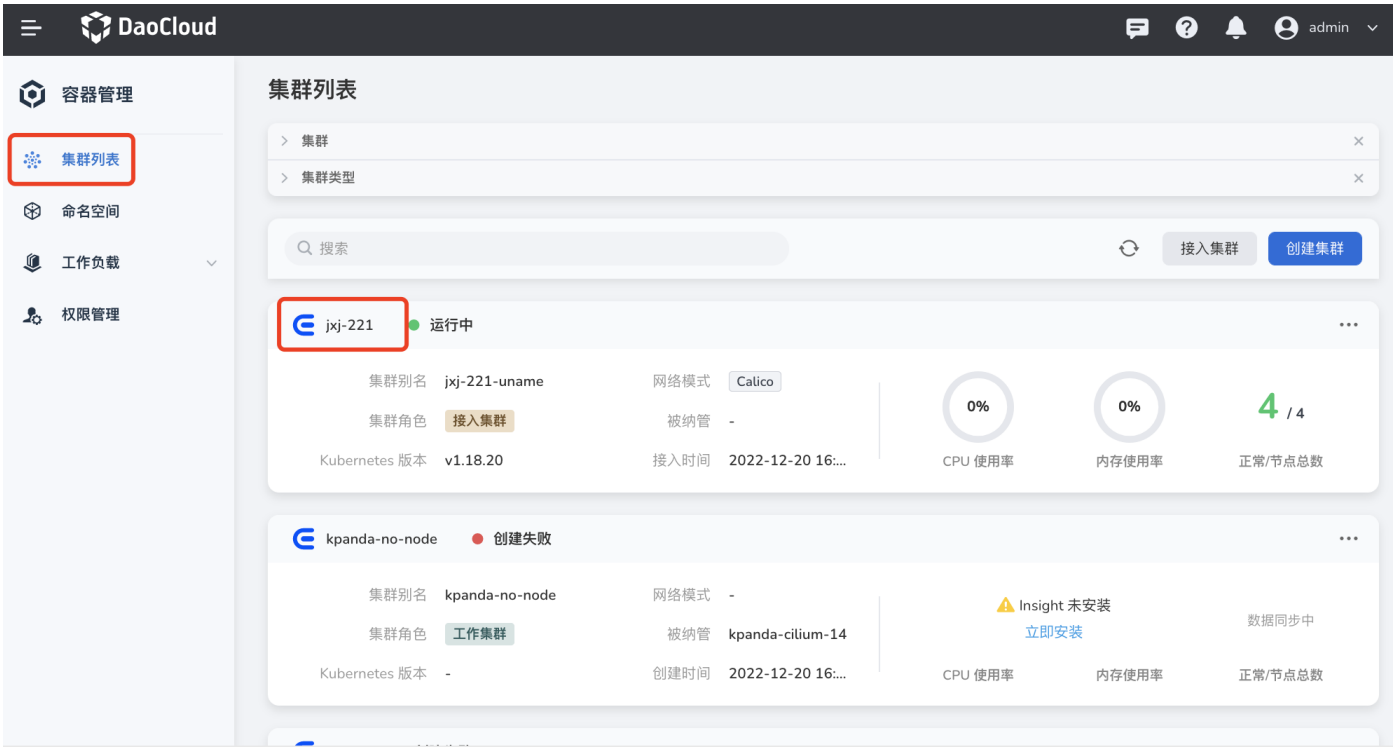
在使用镜像创建无状态负载之前，需要满足以下前提条件：

- 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。

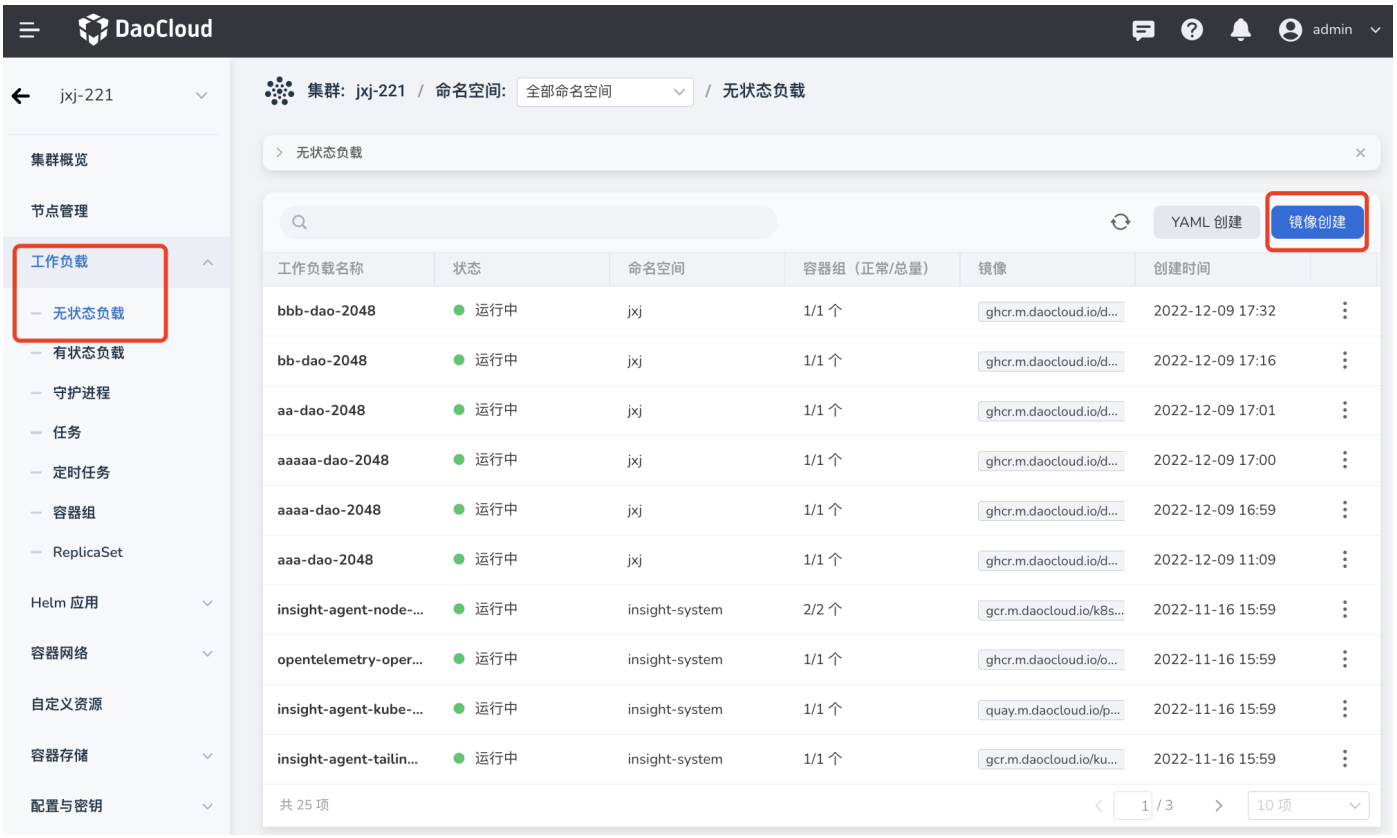
镜像创建

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。

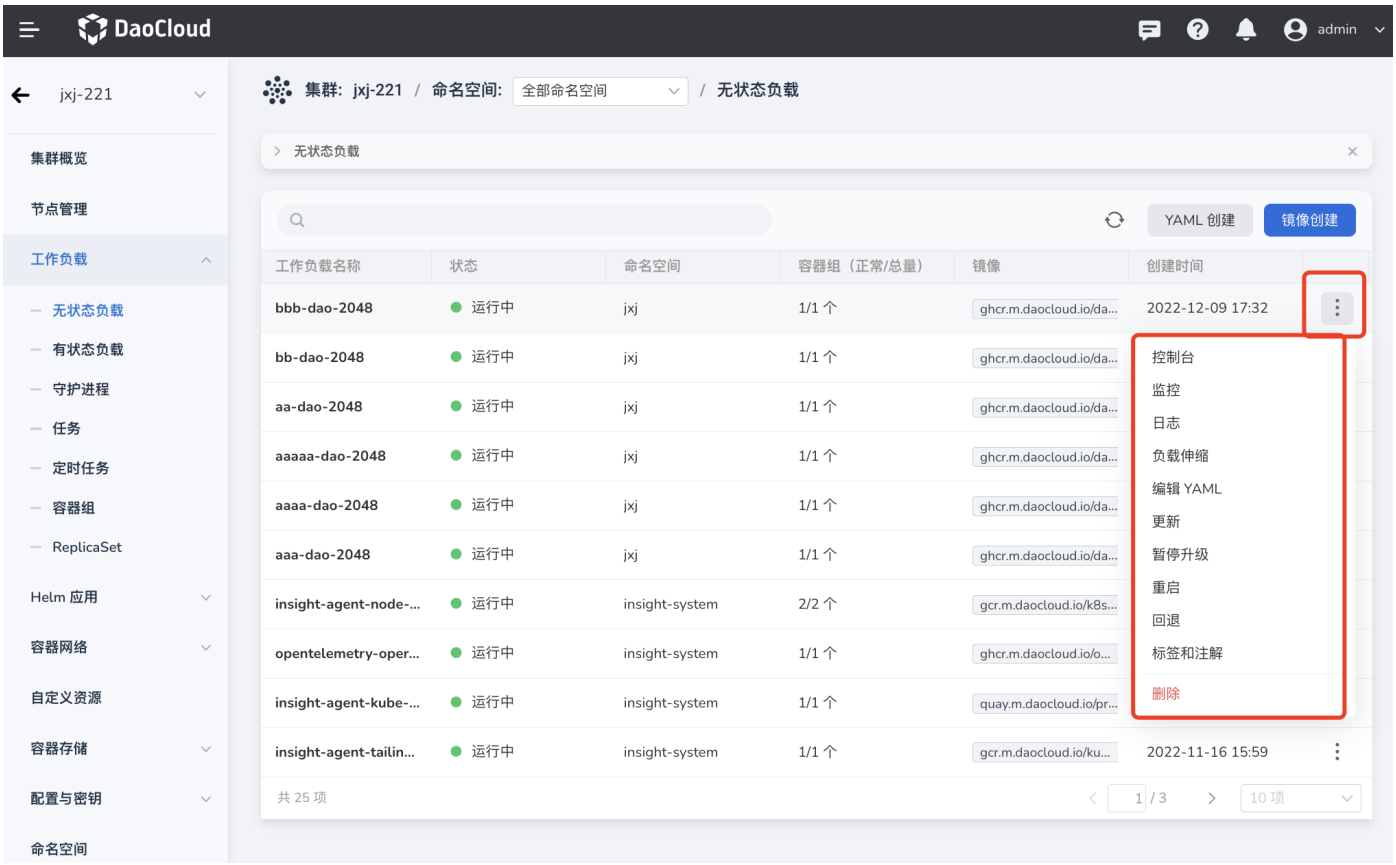


2. 在集群详情页面，点击左侧导航栏的 工作负载 -> 无状态负载，然后点击页面右上角的 镜像创建 按钮。



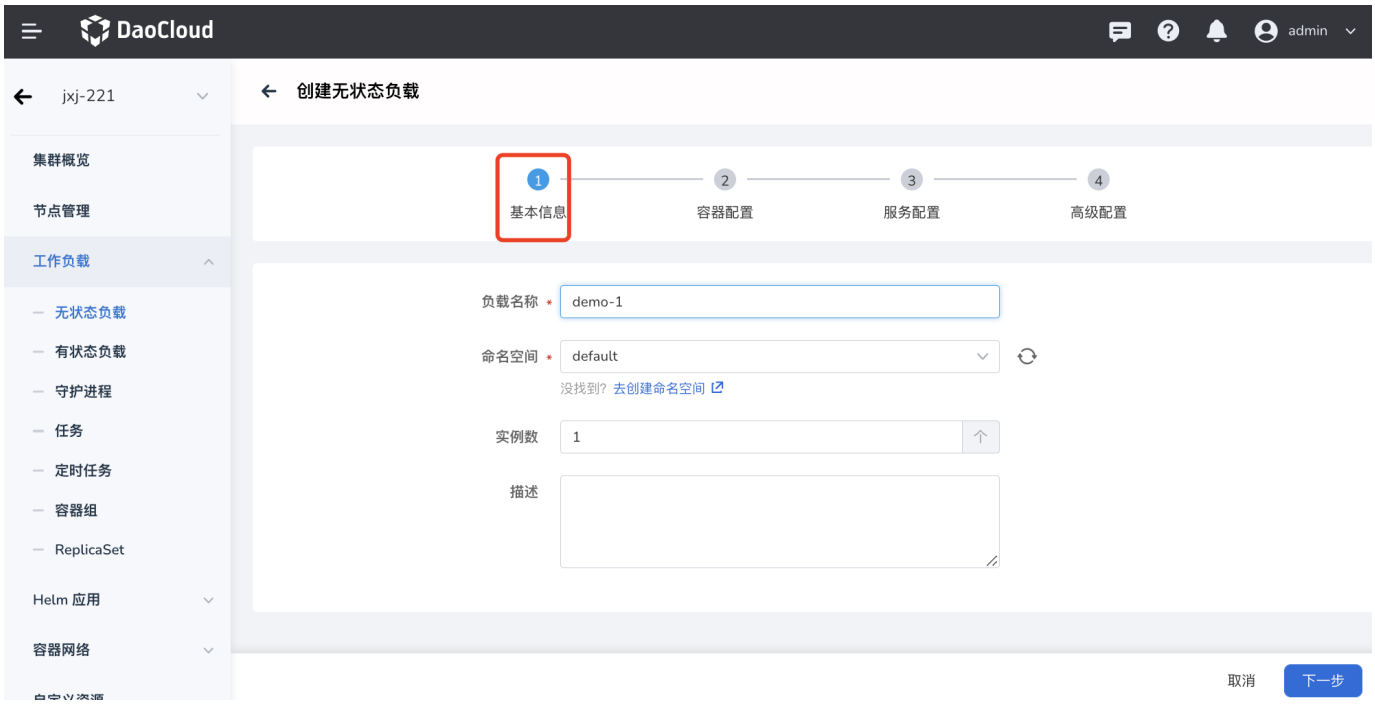
3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击 确定 完成创建。

   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。



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。



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 d.run 的[镜像仓库](../../../kangaroo/intro/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。

![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）"

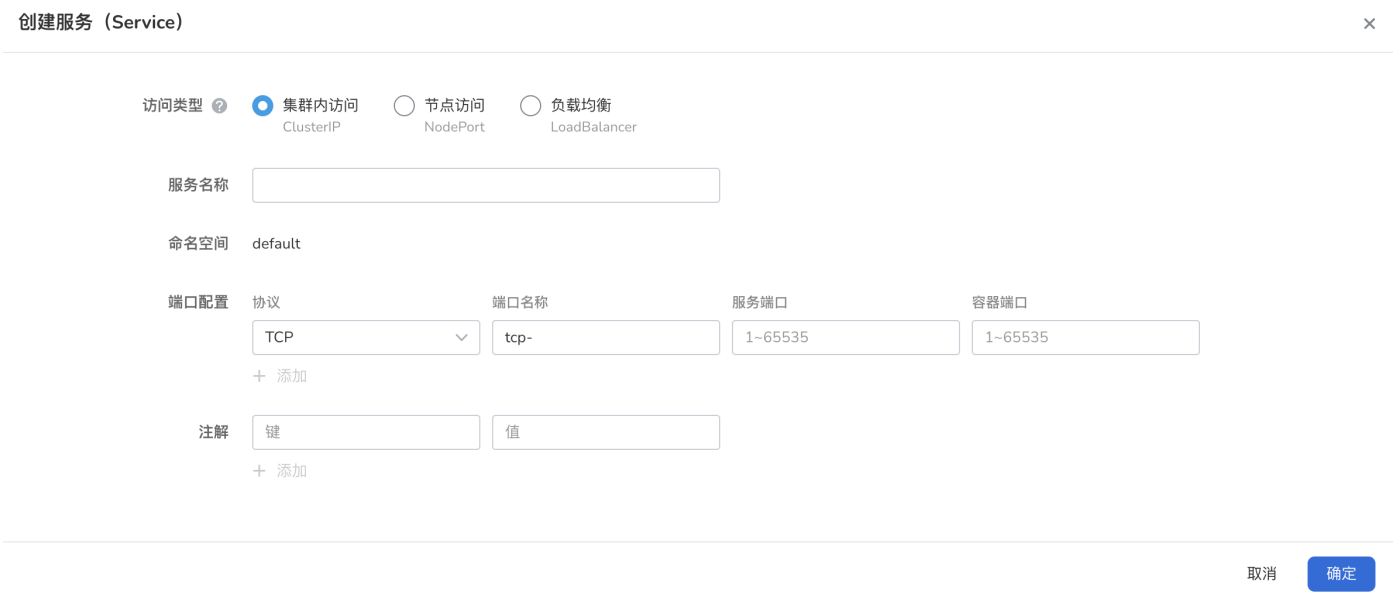
```
通过 Linux 内置的账号权限隔离机制来对容器进行安全隔离。您可以通过使用不同权限的账号 UID（数字身份标记）来限制容器的权限。例如，输入 __0__ 表示使用 root 账号的权限。

![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。

2. 参考创建服务，配置服务参数。



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/config/use-ippool/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"

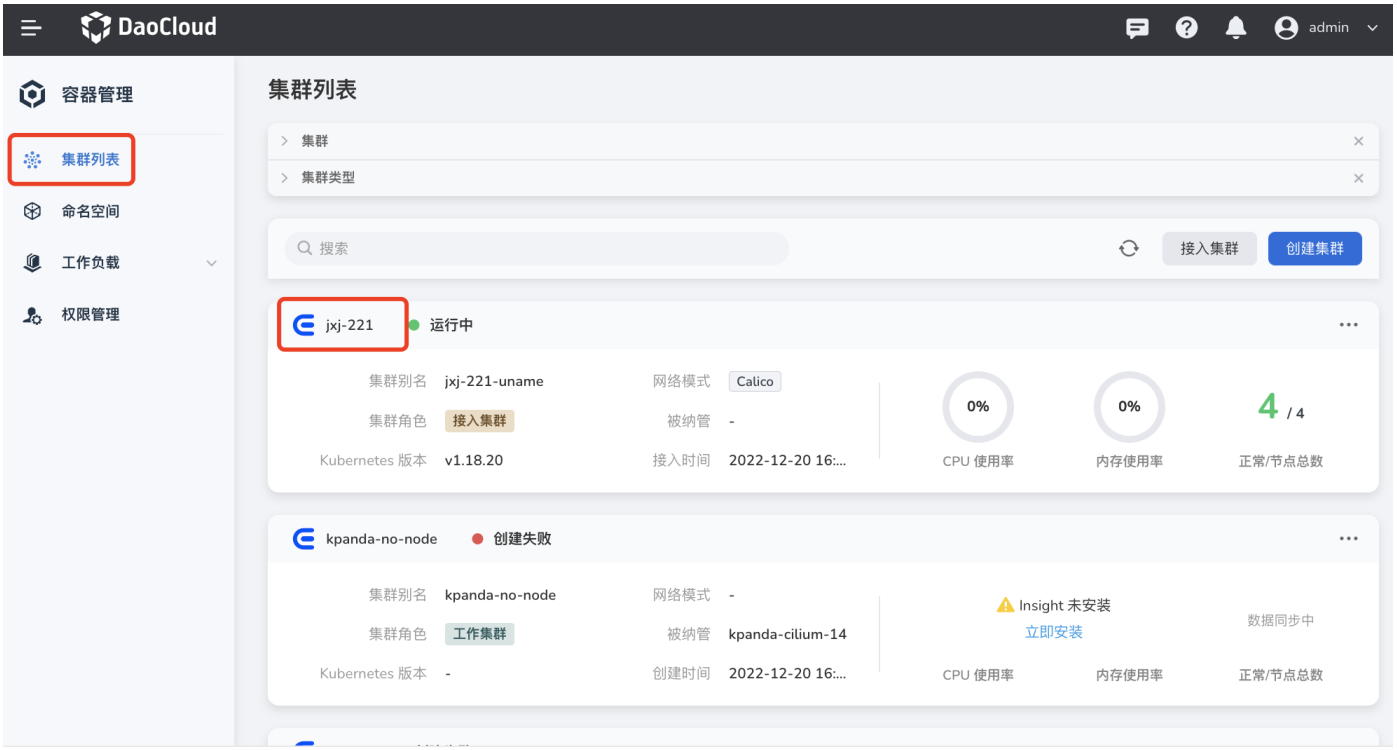
可以点击 __添加__ 按钮为工作负载和容器组添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy16.png)

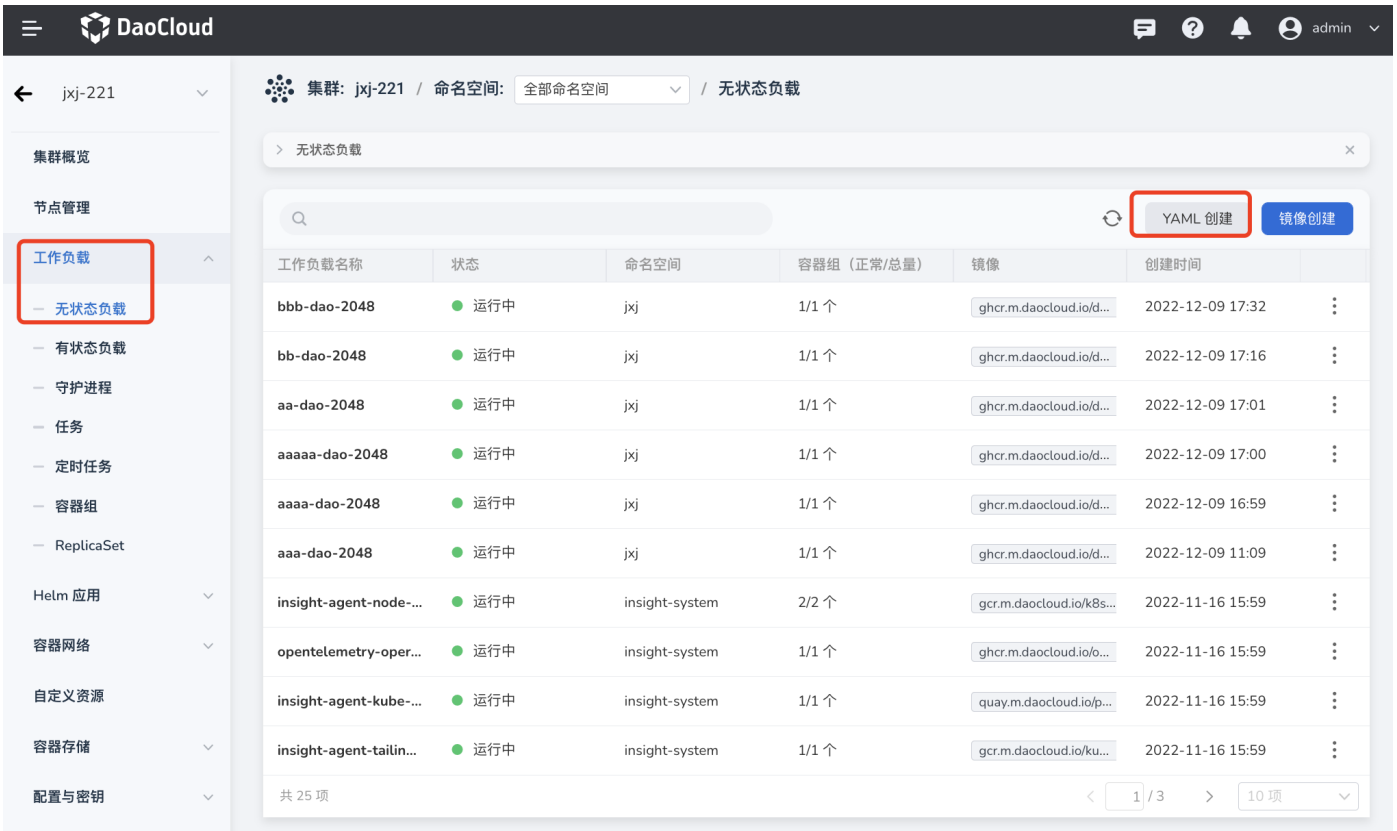
YAML 创建

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。



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。



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。

YAML 创建

```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
```

取消 确定

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 d.run 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

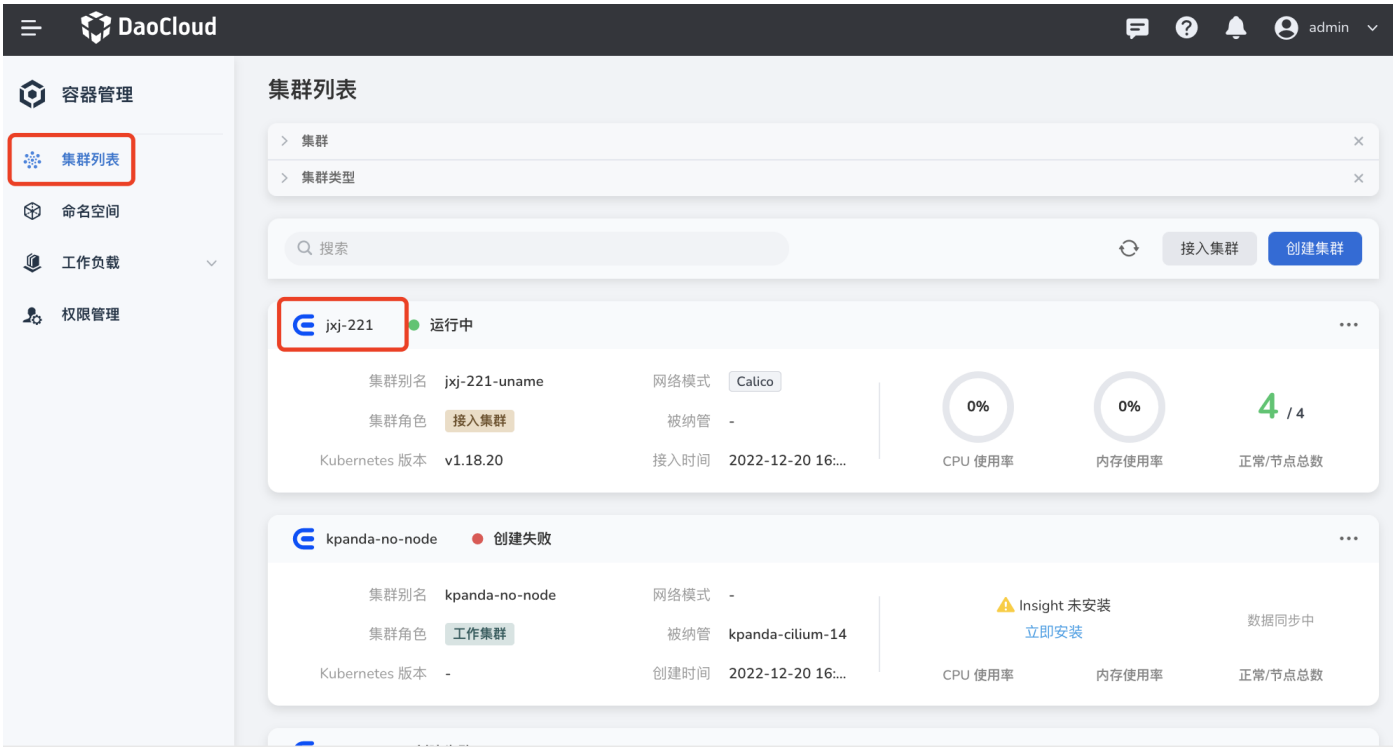
在使用镜像创建有状态负载之前，需要满足以下前提条件：

- 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。

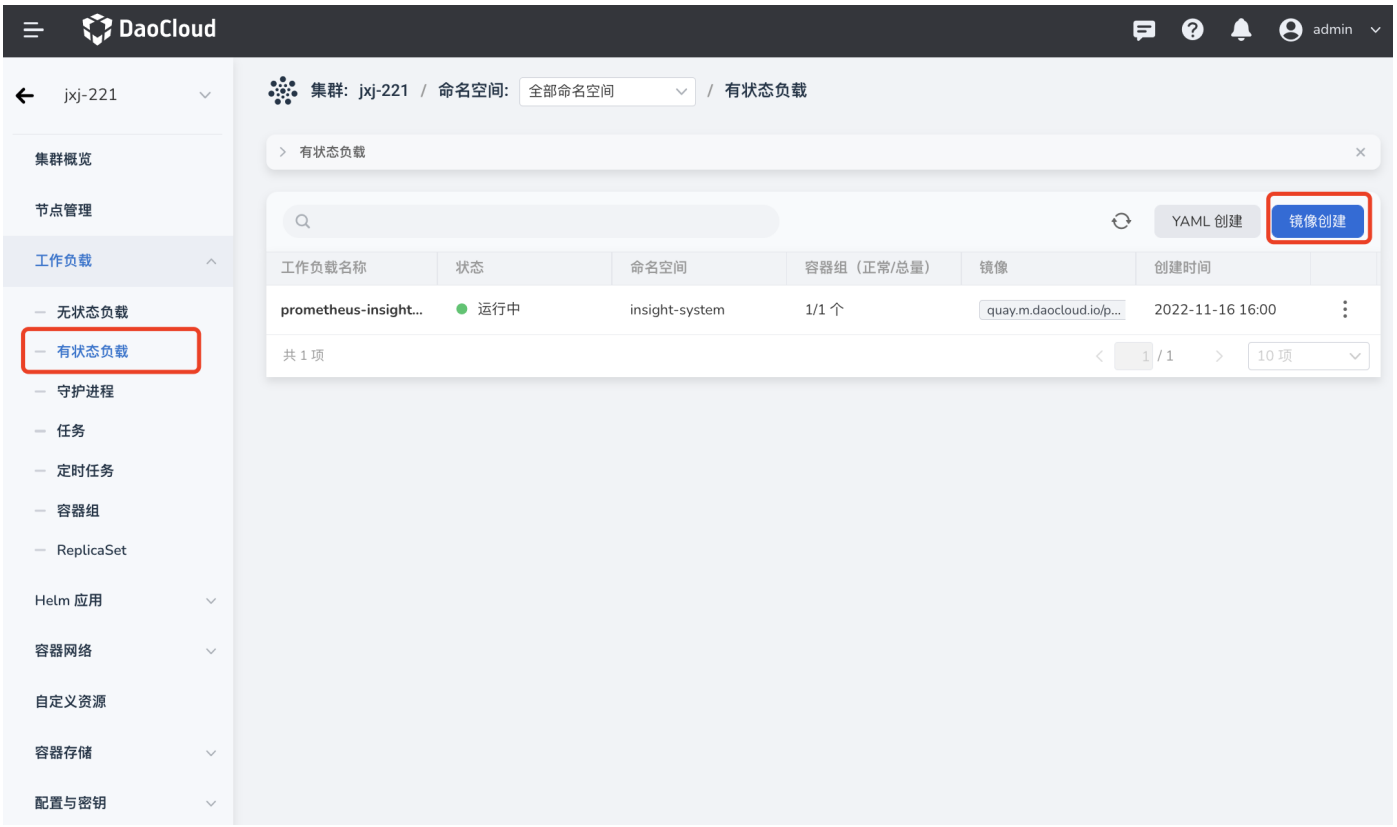
镜像创建

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。



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。



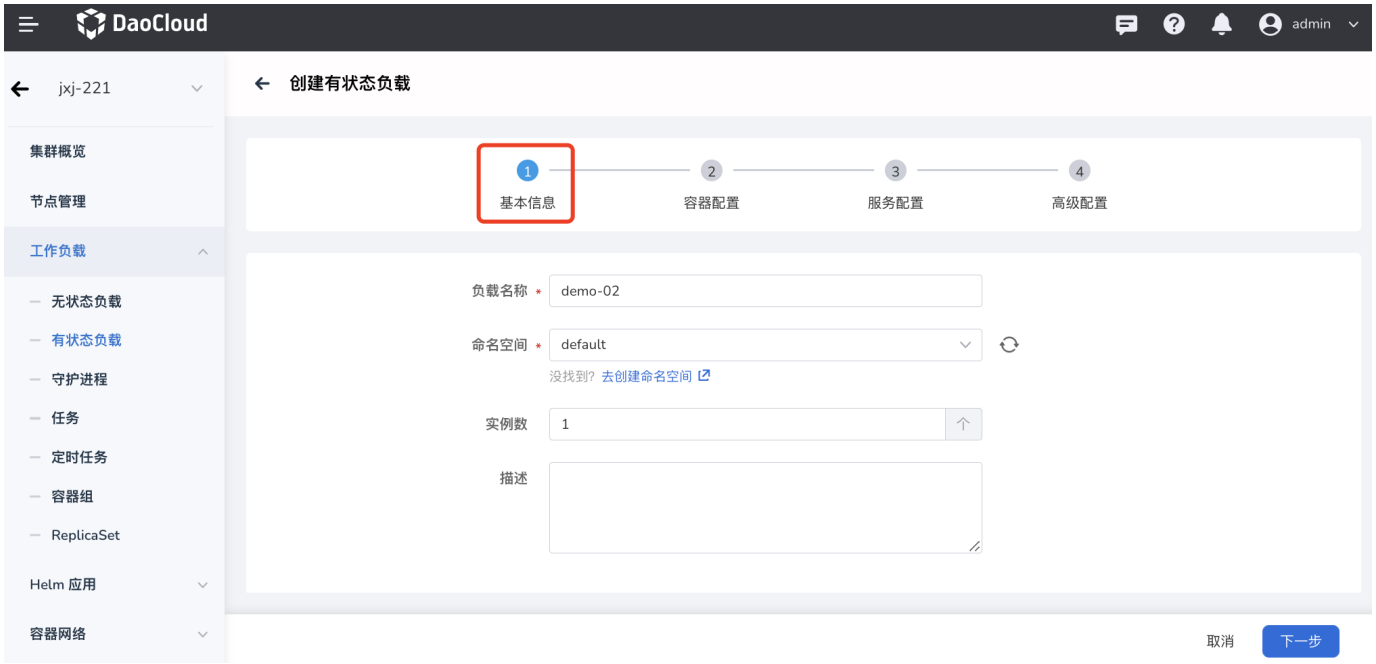
3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击 确定 完成创建。

   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。

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。



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 d.run 的[镜像仓库](../../../kangaroo/intro/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)。

"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)。

"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)。

"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)。

"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。

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。

2. 参考创建服务，配置服务参数。

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/config/use-ippool/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。
- 最大保留版本数：设置版本回滚时保留的旧版本数量。默认 10。
- 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"

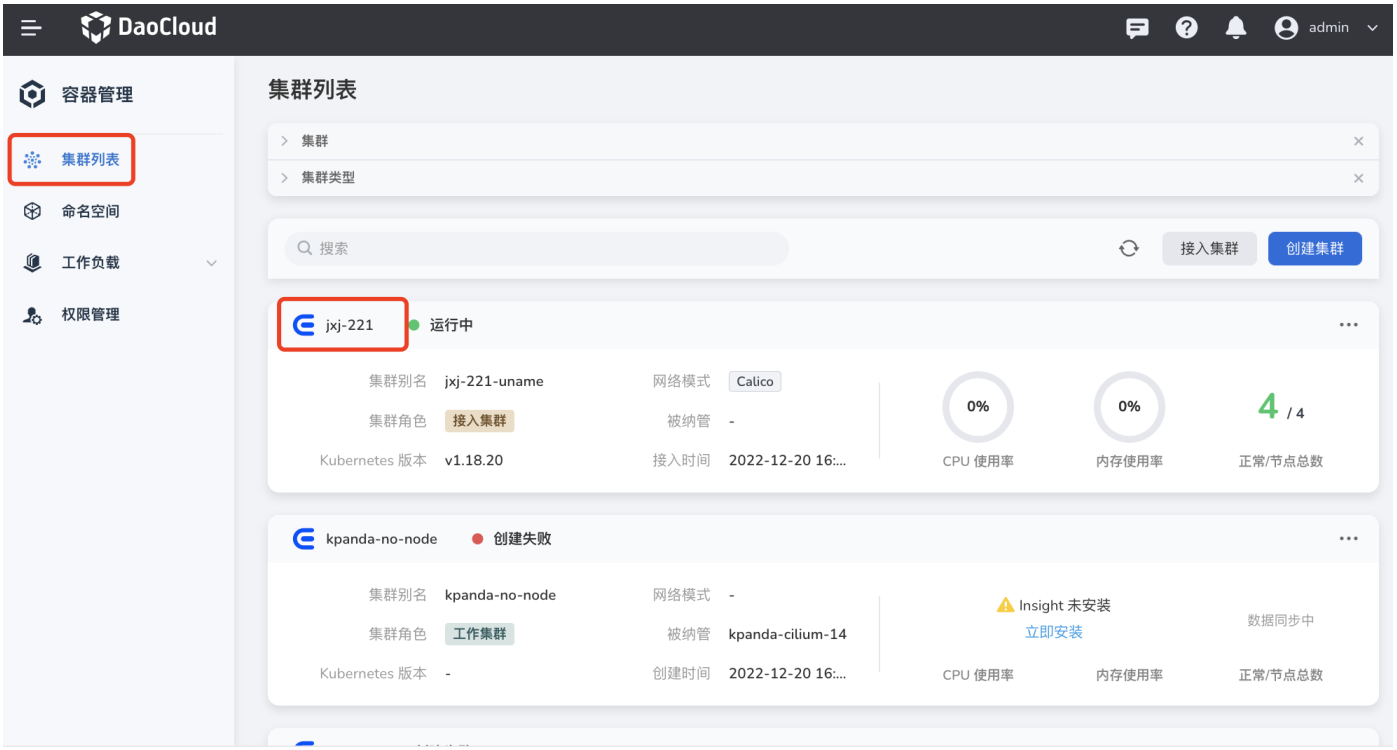
可以点击 __添加__ 按钮为工作负载和容器组添加标签和注解。

![标签与注解](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy16.png)

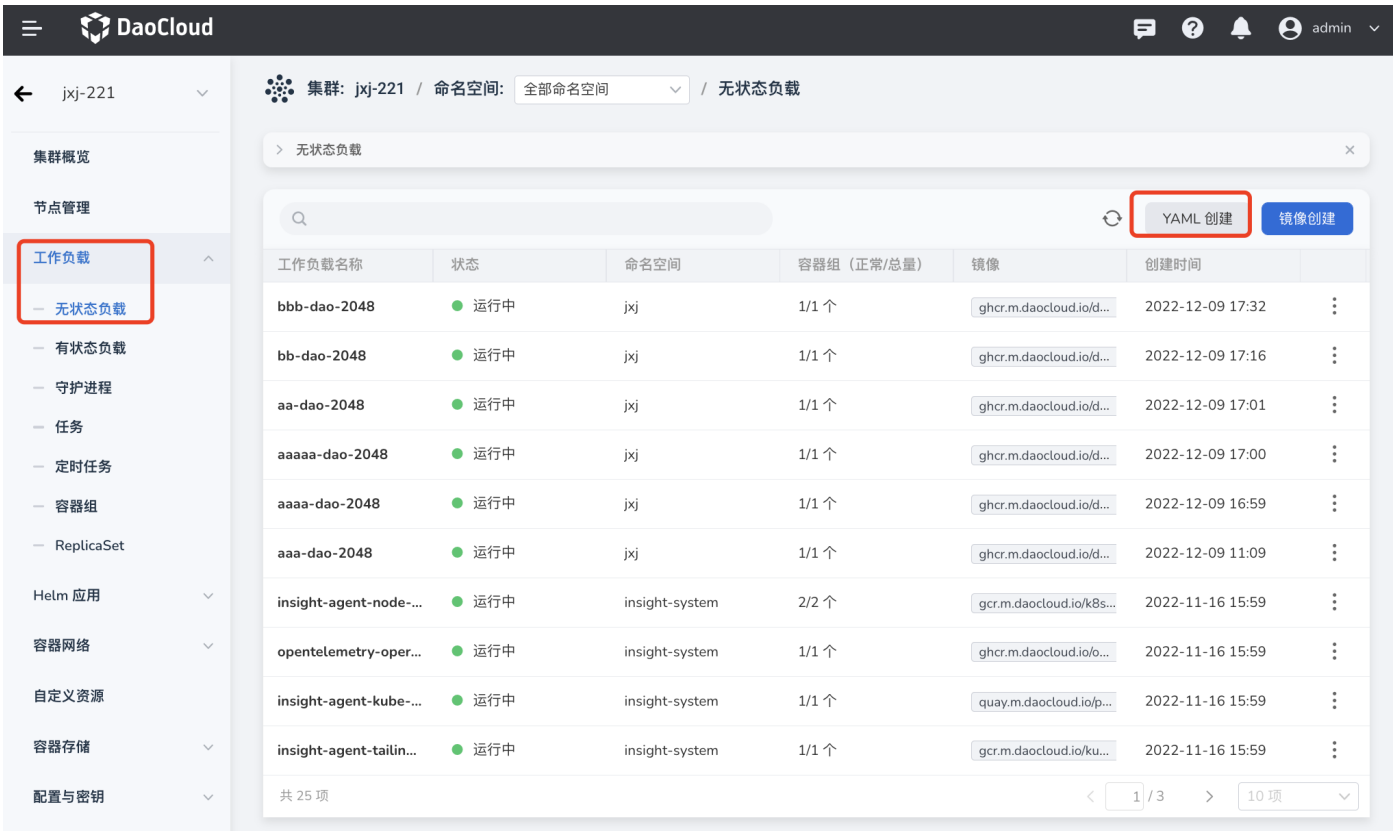
YAML 创建

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。

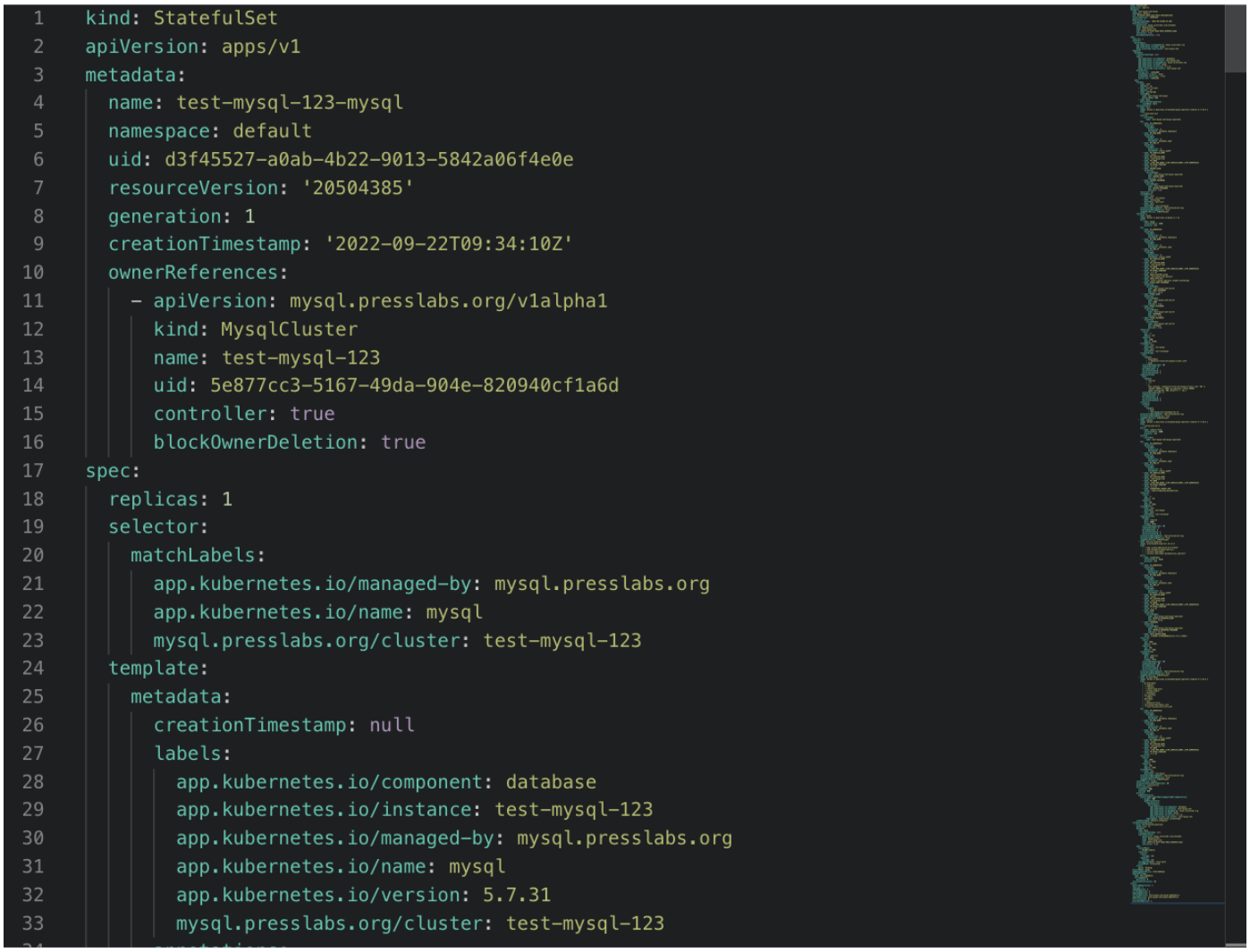


2. 在集群详情页面，点击左侧导航栏的 工作负载 -> 有状态负载，然后点击页面右上角的 **YAML** 创建 按钮。



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。

YAML 创建



```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
```

取消 确定

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_NAMESPACE)
            - 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_NAMESPACE)
          - 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_NAMESPACE)
      - 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_NAMESPACE)
      - 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_NAMESPACE)
      - 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

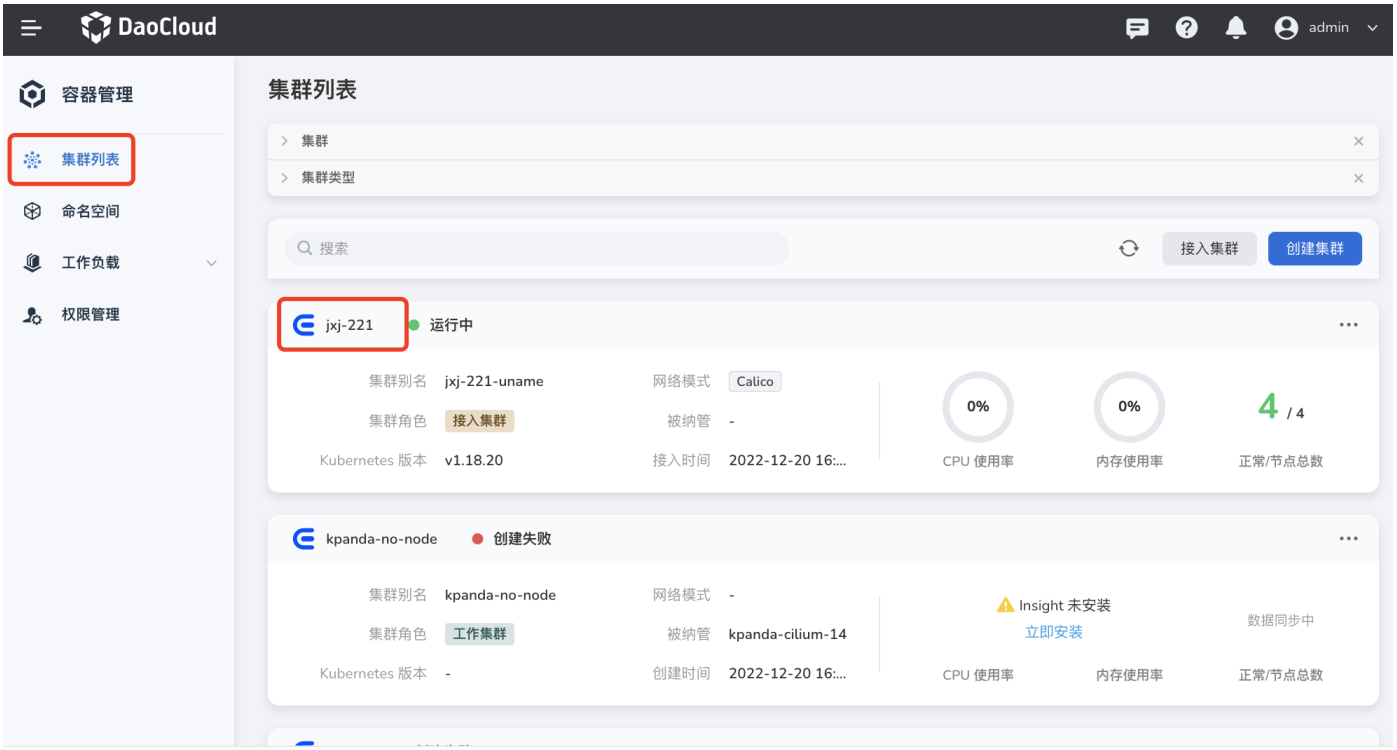
创建 DaemonSet 之前，需要满足以下前提条件：

- 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。

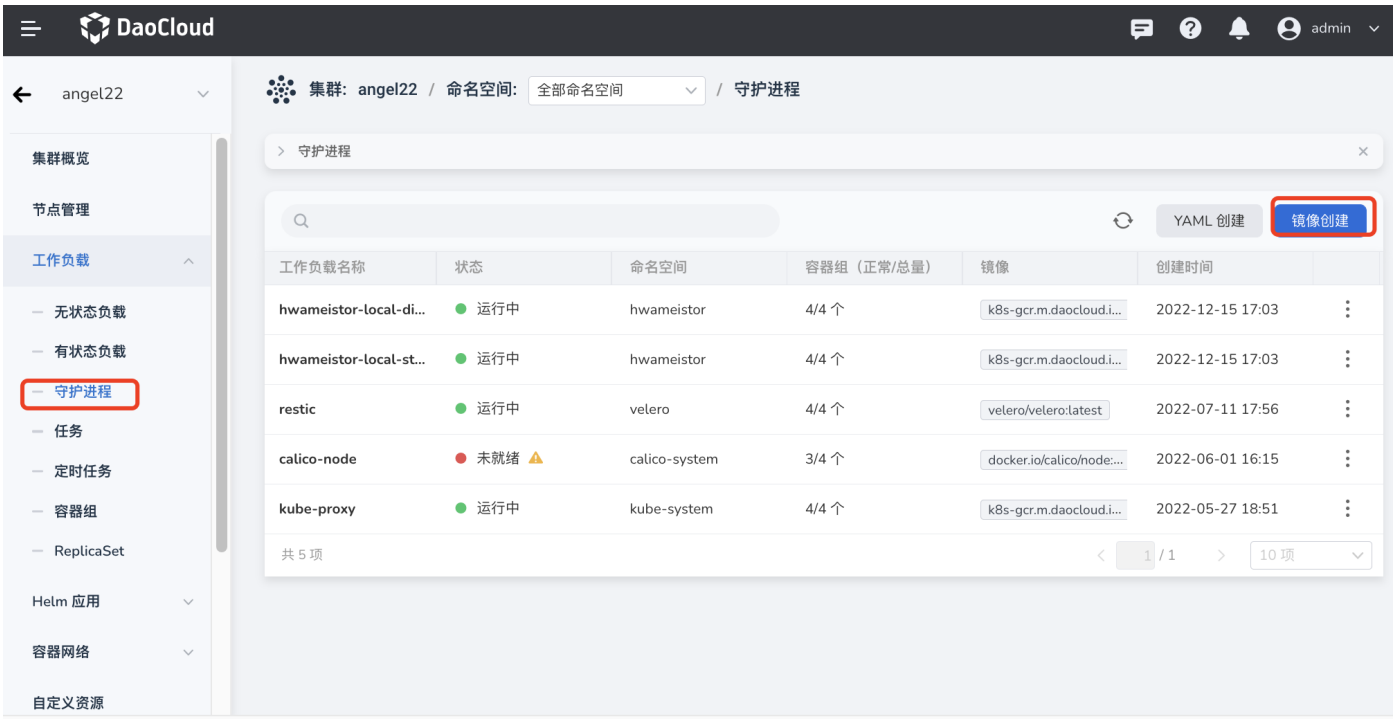
镜像创建

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。

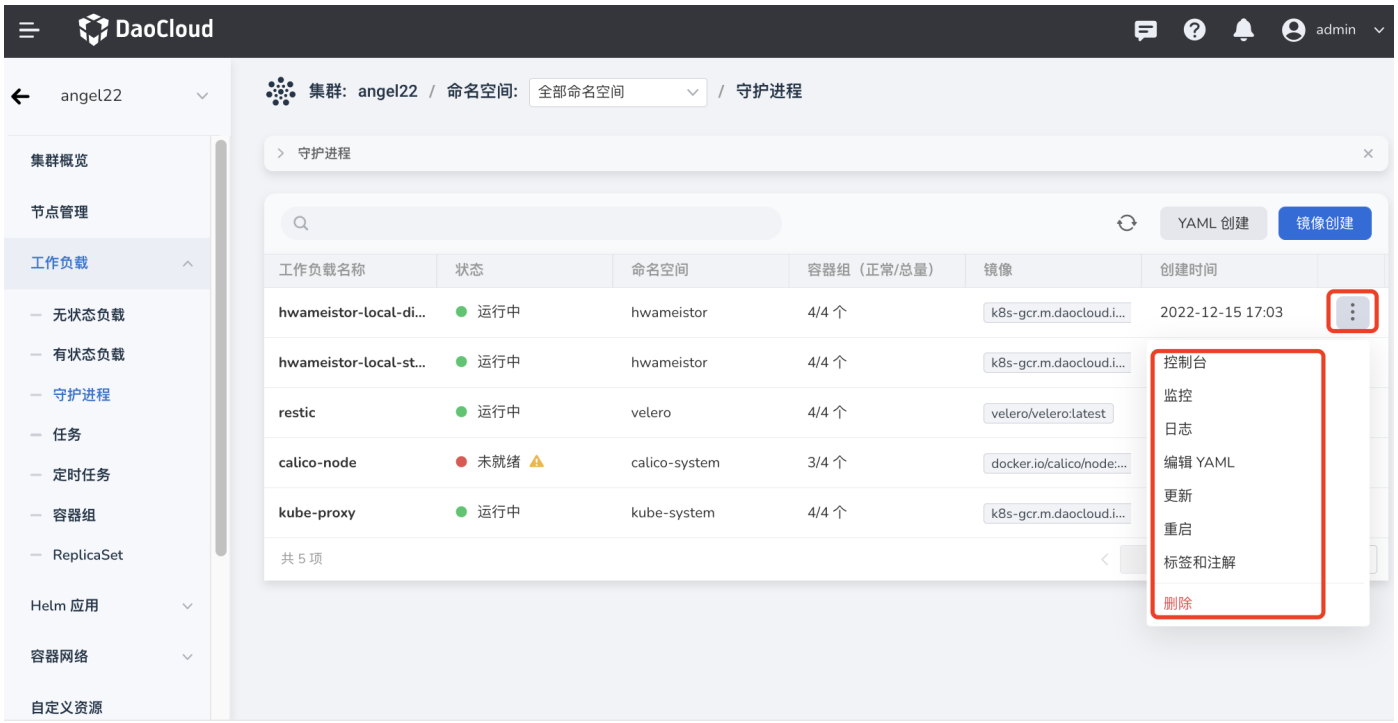


2. 在集群详情页面，点击左侧导航栏的 工作负载 -> 守护进程，然后点击页面右上角的 镜像创建 按钮。



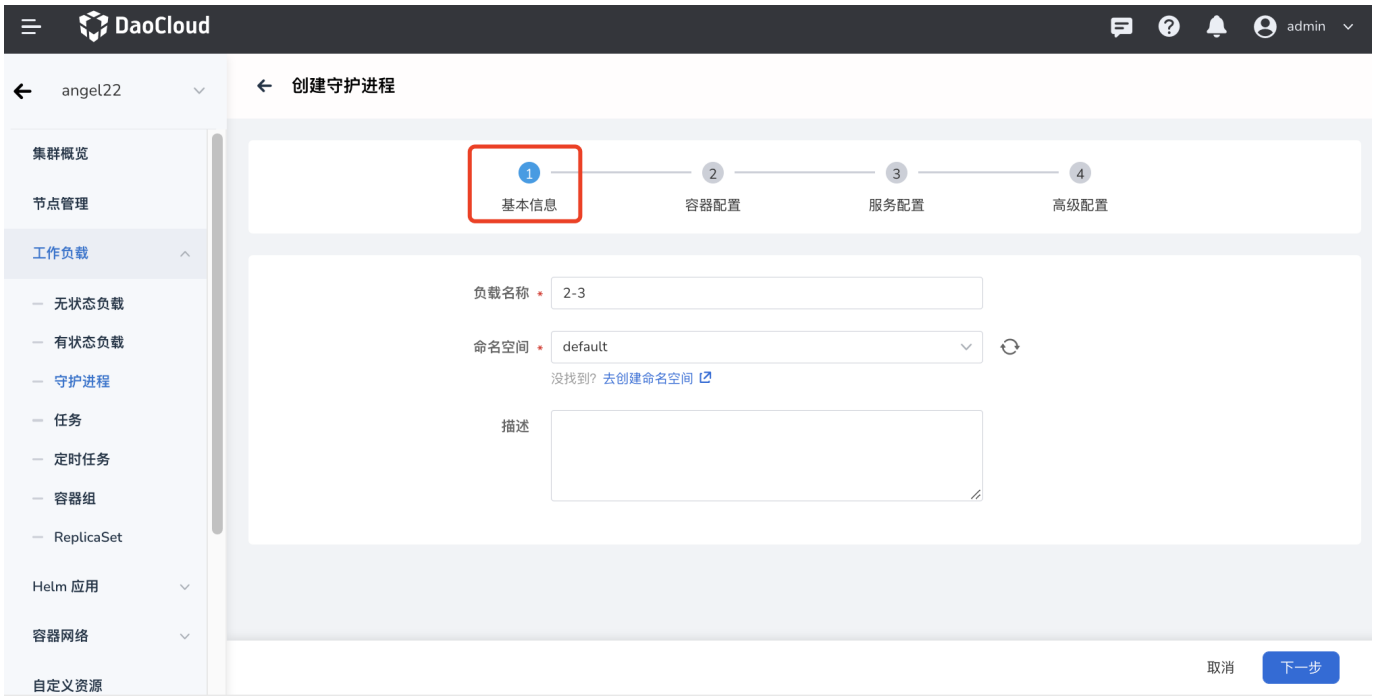
3. 依次填写基本信息、容器配置、服务配置、高级配置后，在页面右下角点击 确定 完成创建。

   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。



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。



- 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 d.run 的[镜像仓库](../../../kangaroo/intro/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)
```

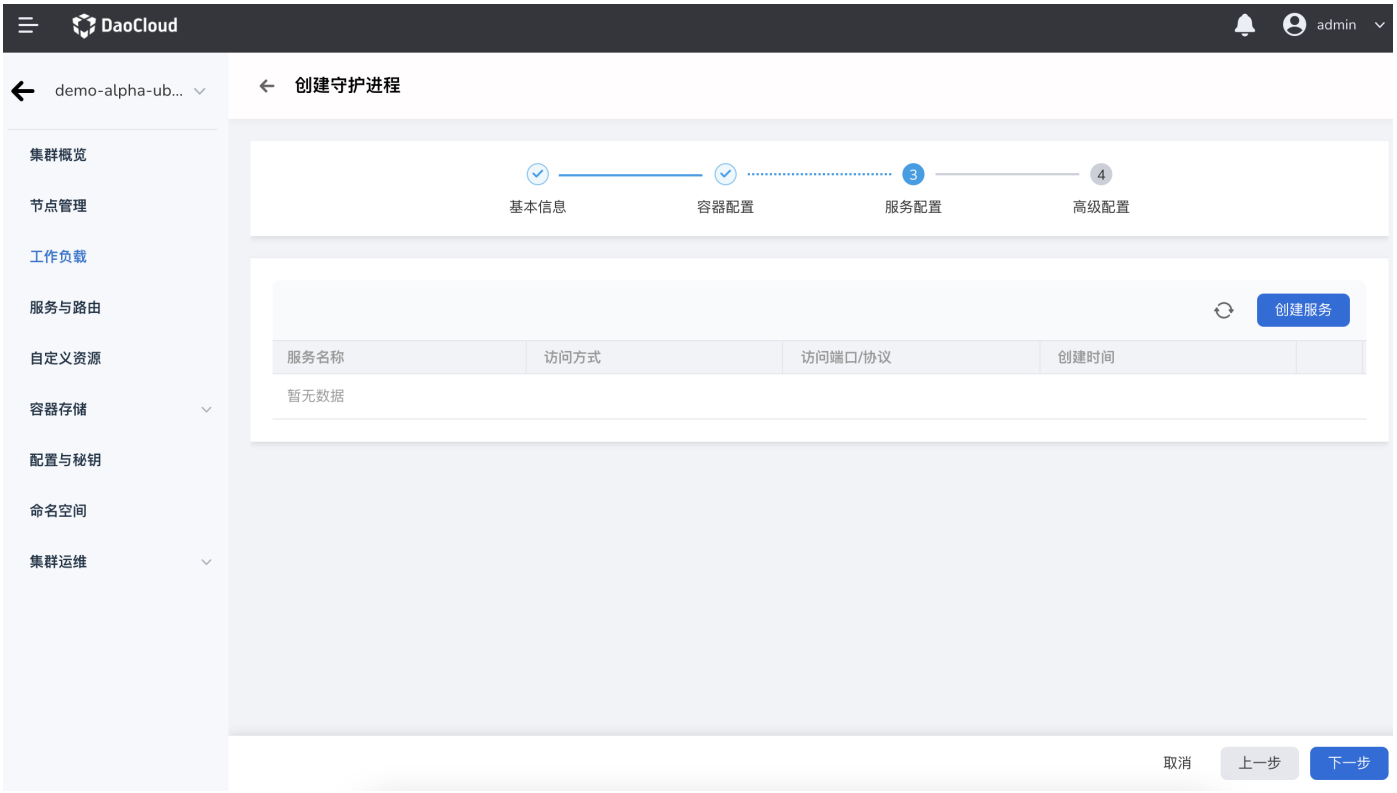
"安全设置（选填）"

```
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。



2. 配置服务参数，详情请参考创建服务。

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 配置](https://docs.daocloud.io/daocloud-docs-images/docs/kpanda/images/deploy17.png)

- 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)。
```

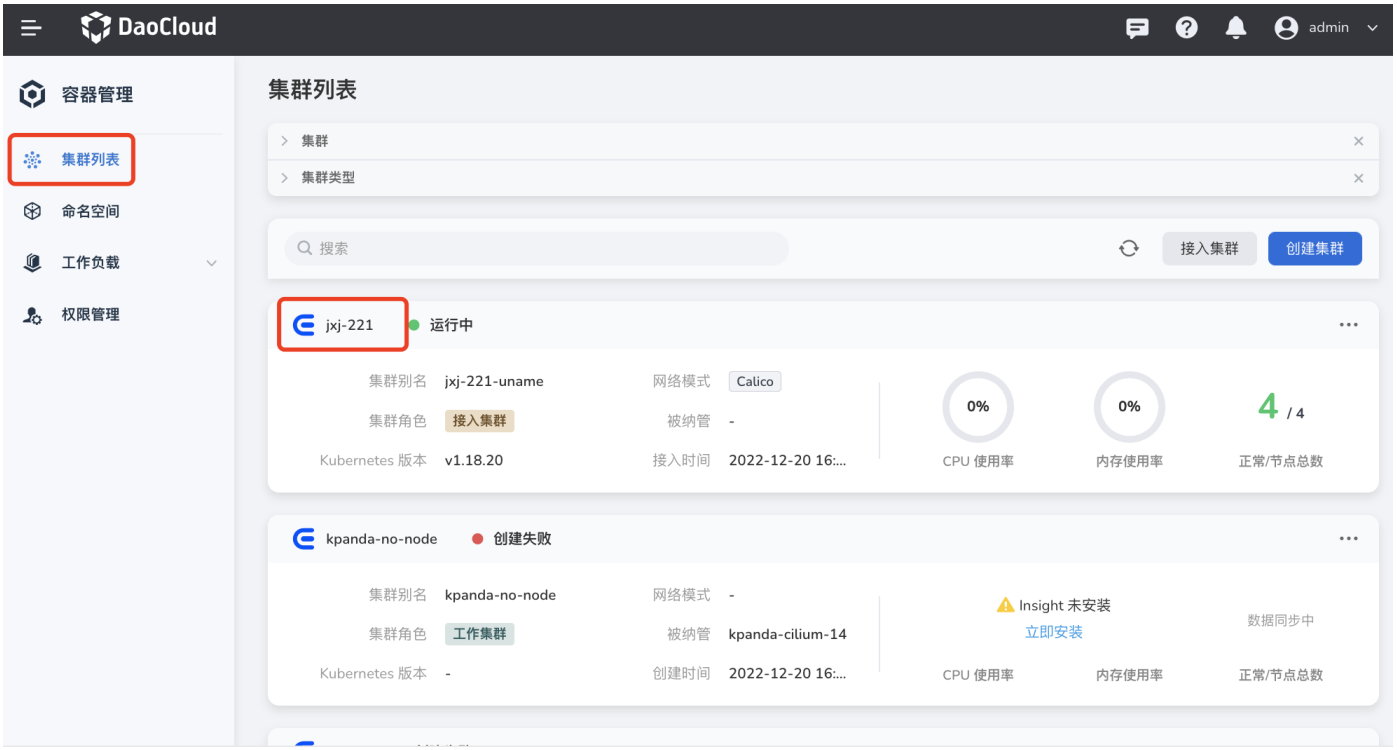
"标签与注解"

```
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)
```

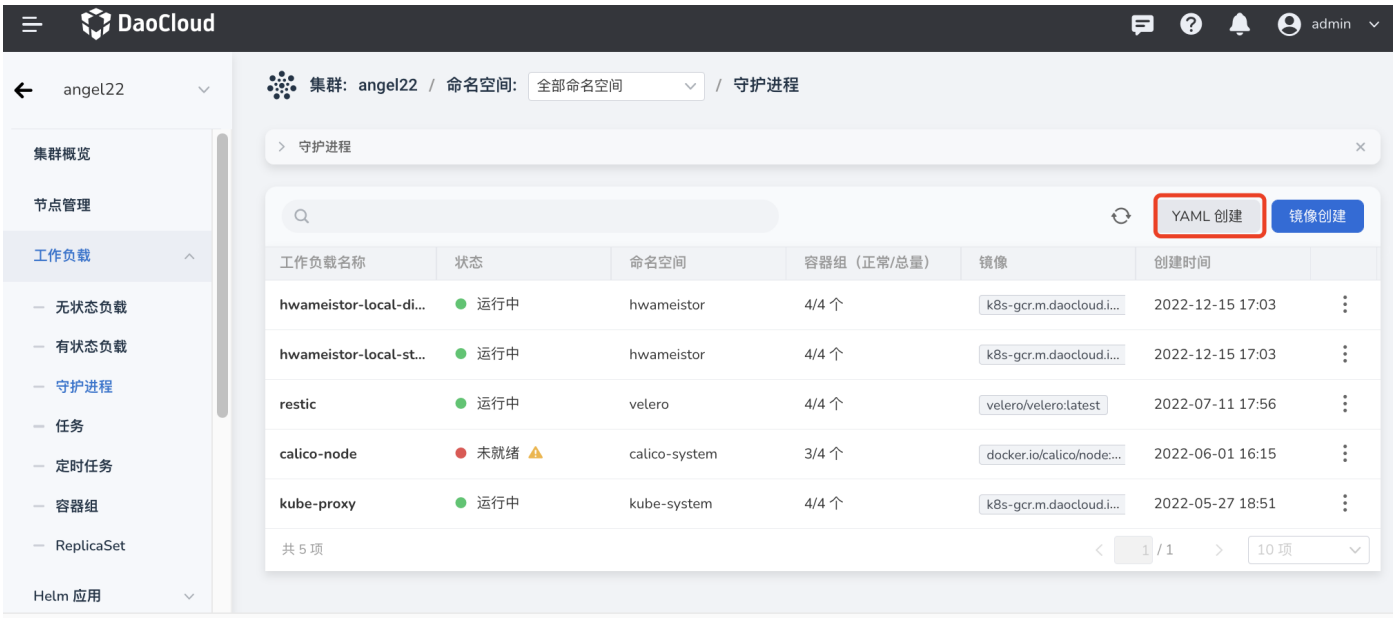
YAML 创建

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。

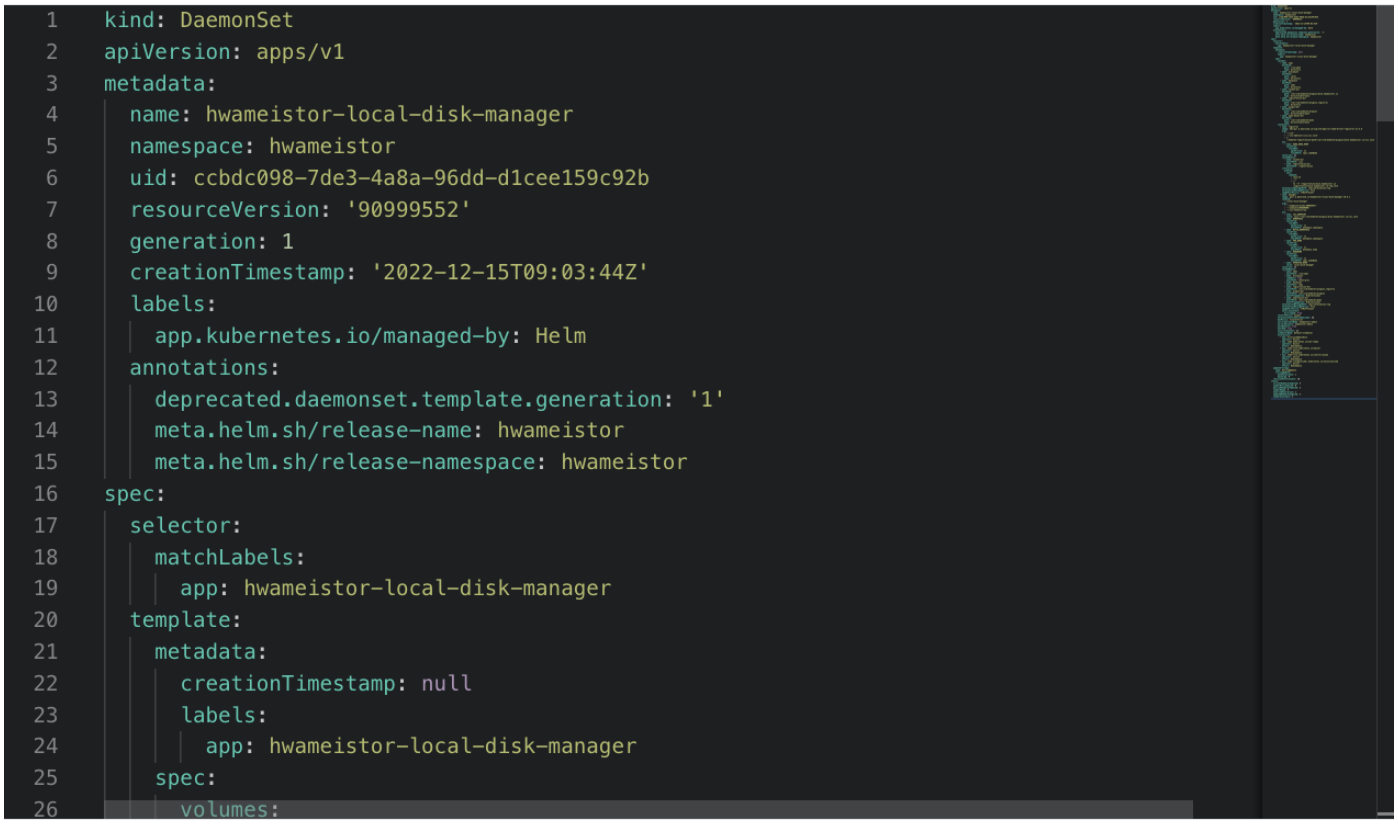


2. 在集群详情页面，点击左侧导航栏的 工作负载 -> 守护进程，然后点击页面右上角的 **YAML** 创建 按钮。



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。

YAML 创建



```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:
```

取消 确定

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-d1ceel59c92b
  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.hwameistor.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

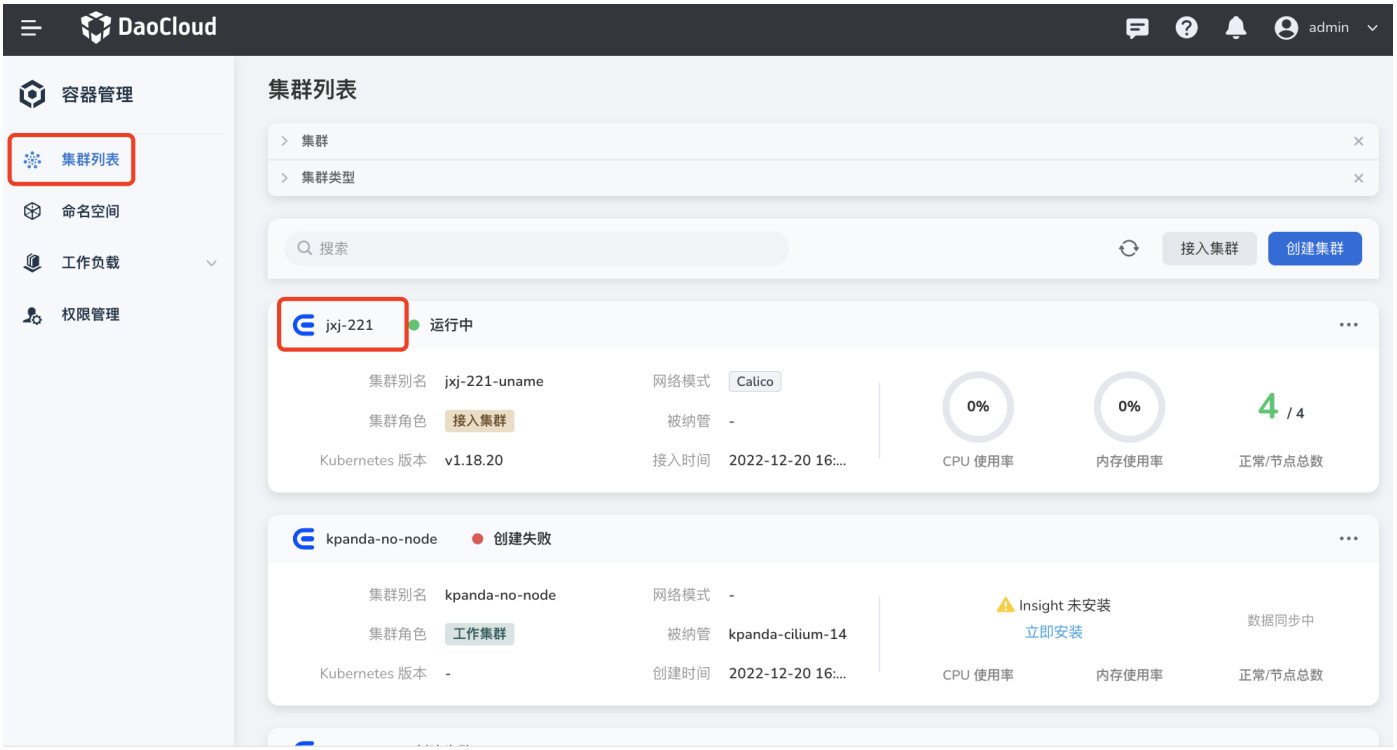
创建定时任务（CronJob）之前，需要满足以下前提条件：

- 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。

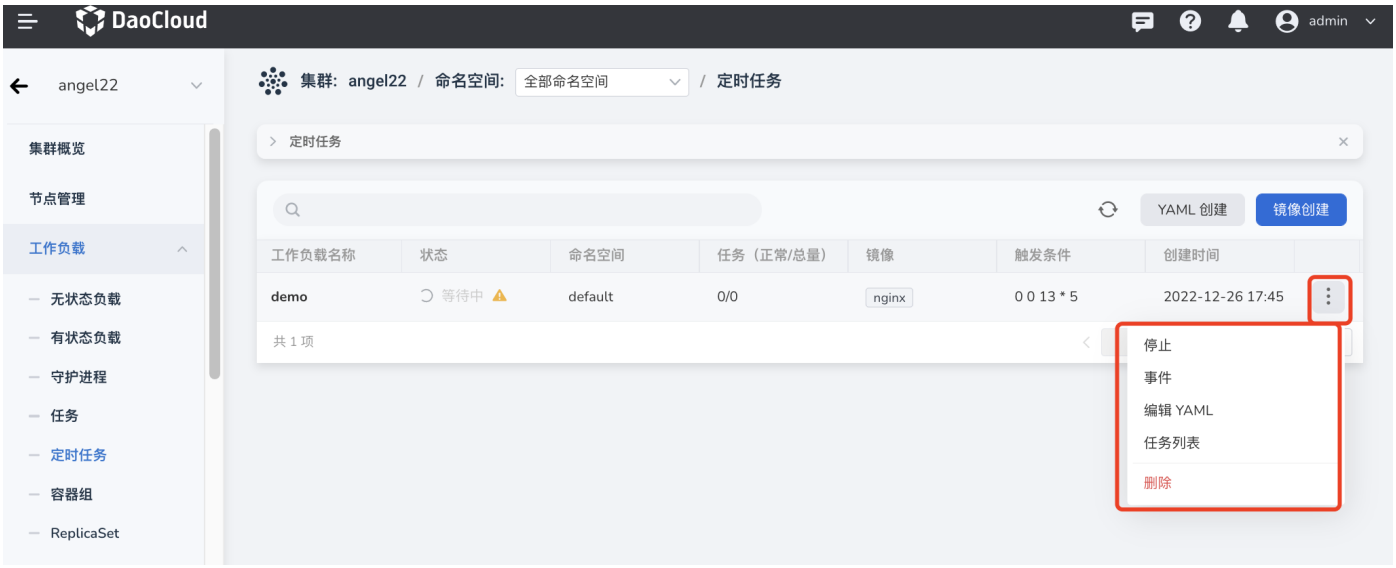
1. 点击左侧导航栏上的 集群列表，然后点击目标集群的名称，进入 集群详情 页面。



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。

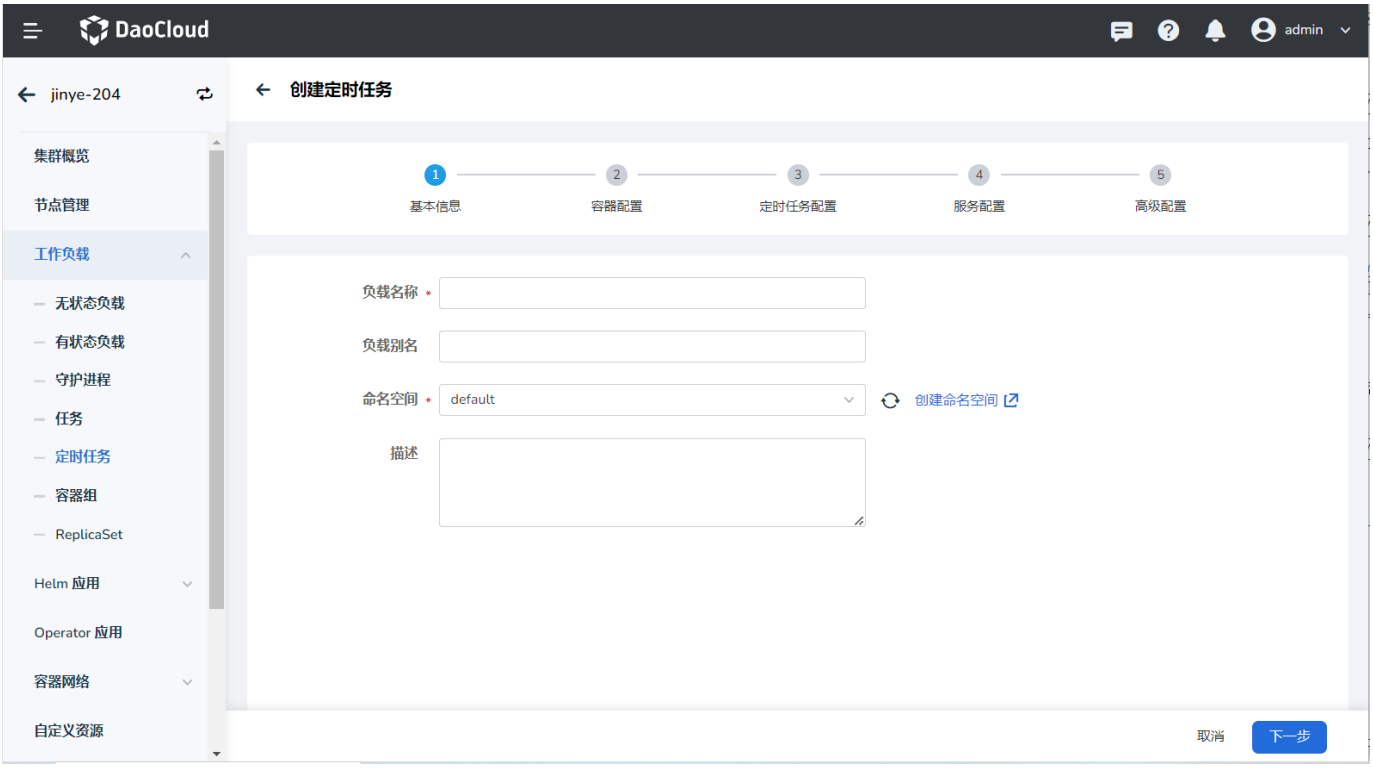
3. 依次填写基本信息、容器配置、定时任务配置、高级配置后，在页面右下角点击 确定 完成创建。

   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。



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。



- 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](../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 d.run 的[镜像仓库](../../../kangaroo/intro/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)
```

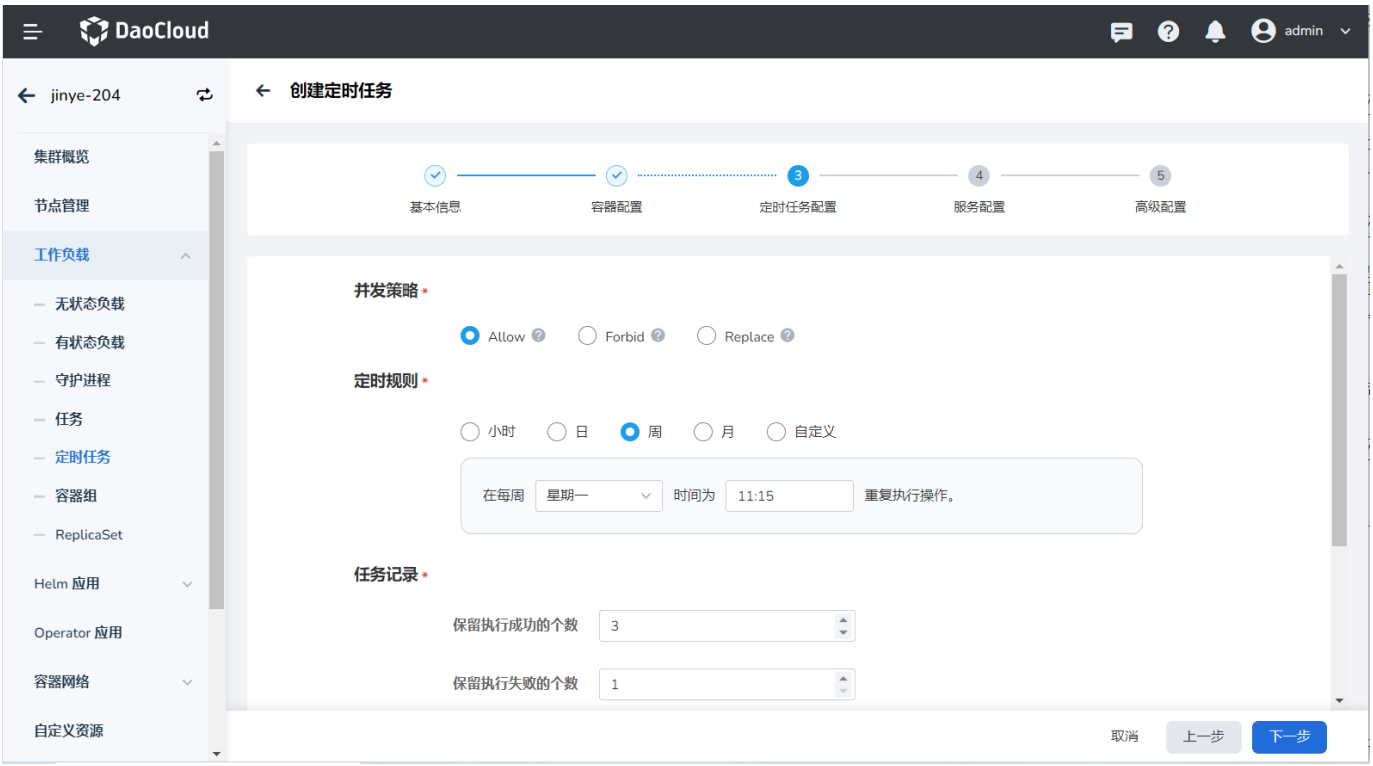
"数据存储（选填）"

```
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



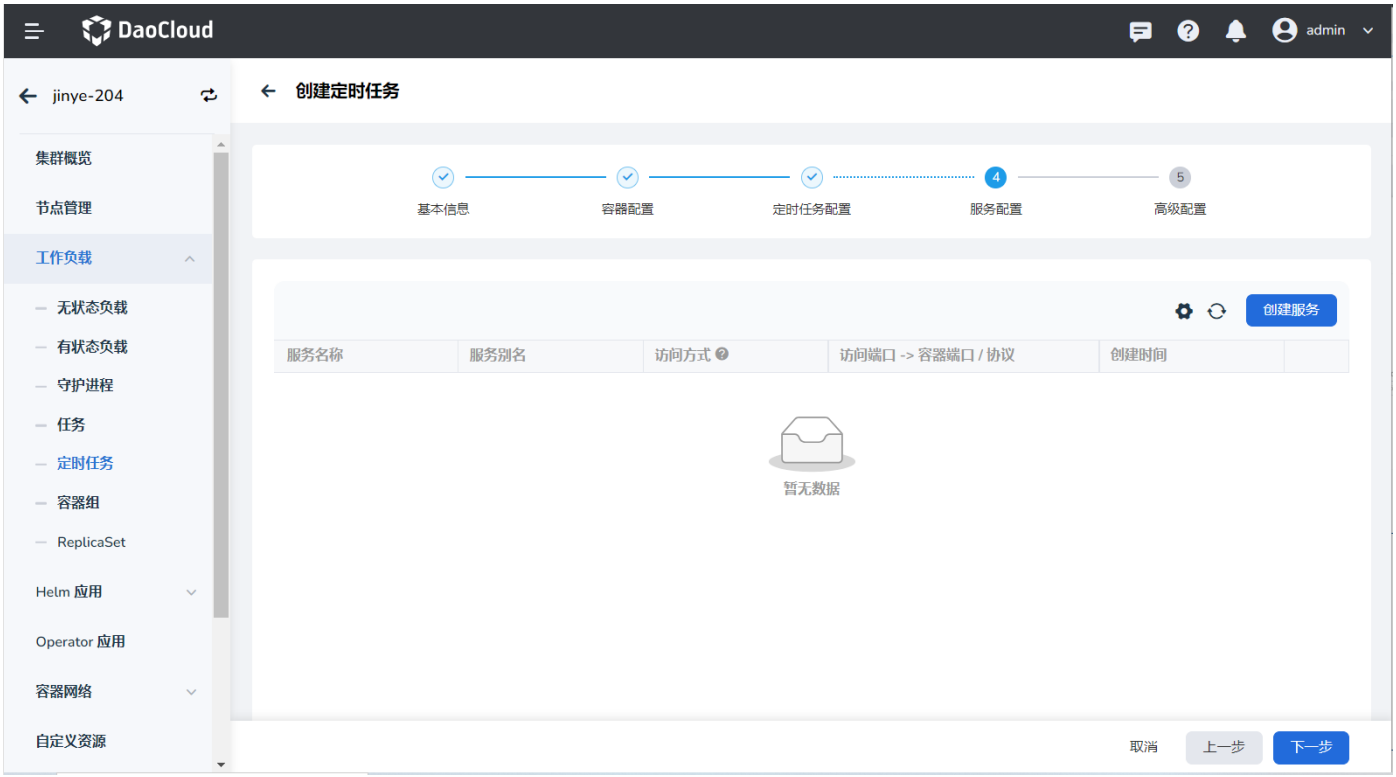
- 并发策略：是否允许多个 Job 任务并行执行。
- **Allow**：可以在前一个任务未完成时就创建新的定时任务，而且多个任务可以并行。任务太多可能抢占集群资源。
- **Forbid**：在前一个任务完成之前，不能创建新任务，如果新任务的执行时间到了而之前的任务仍未执行完，CronJob 会忽略新任务的执行。
- **Replace**：如果新任务的执行时间到了，但前一个任务还未完成，新的任务会取代前一个任务。

  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。

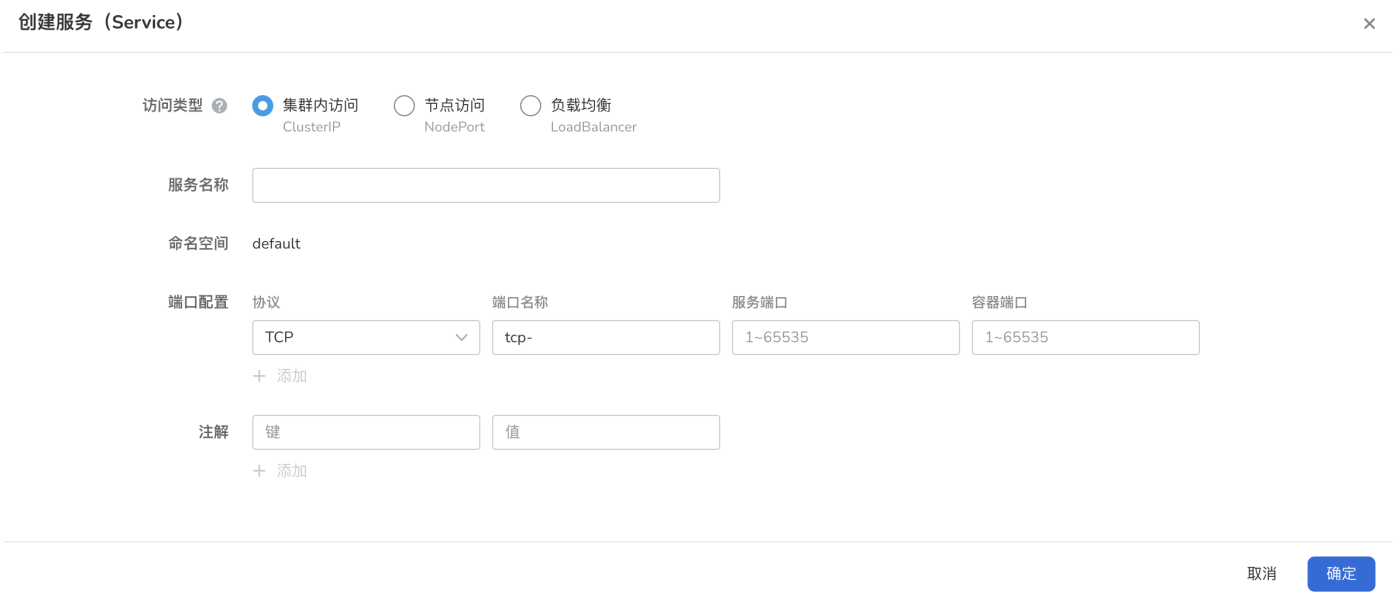
服务配置

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。



2. 参考创建服务，配置服务参数。

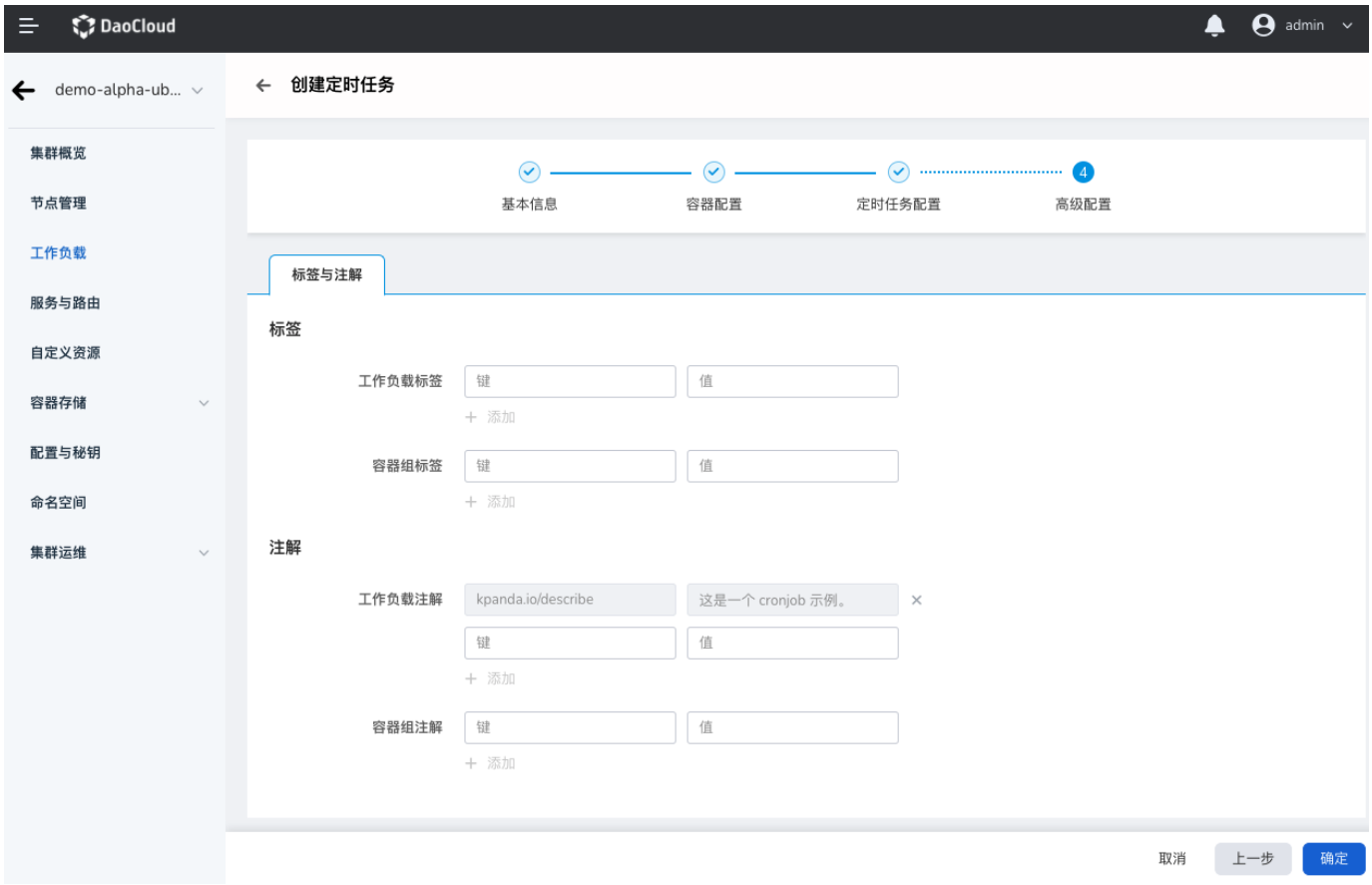


3. 点击 确定，点击 下一步。

高级配置

定时任务的高级配置主要涉及标签与注解。

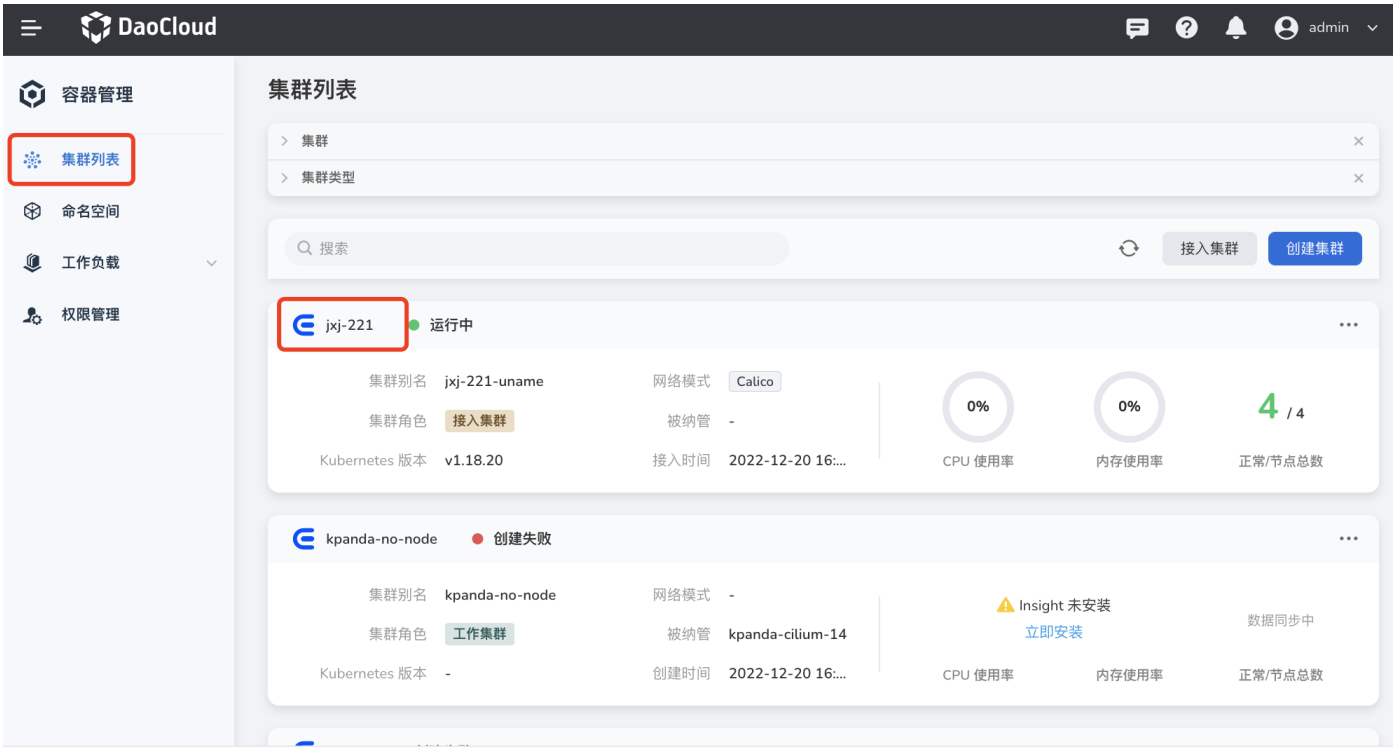
可以点击 添加 按钮为工作负载实例 Pod 添加标签和注解。


DaoCloud
admin
demo-alpha-ub...
集群概览
节点管理
工作负载
服务与路由
自定义资源
容器存储
配置与秘钥
命名空间
集群运维
创建定时任务
基本信息
容器配置
定时任务配置
4
高级配置
标签与注解
标签
工作负载标签
键
值
添加
容器组标签
键
值
添加
注解
工作负载注解
kpanda.io/describe
这是一个 cronjob 示例。
键
值
添加
容器组注解
键
值
添加
取消
上一步
确定

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。

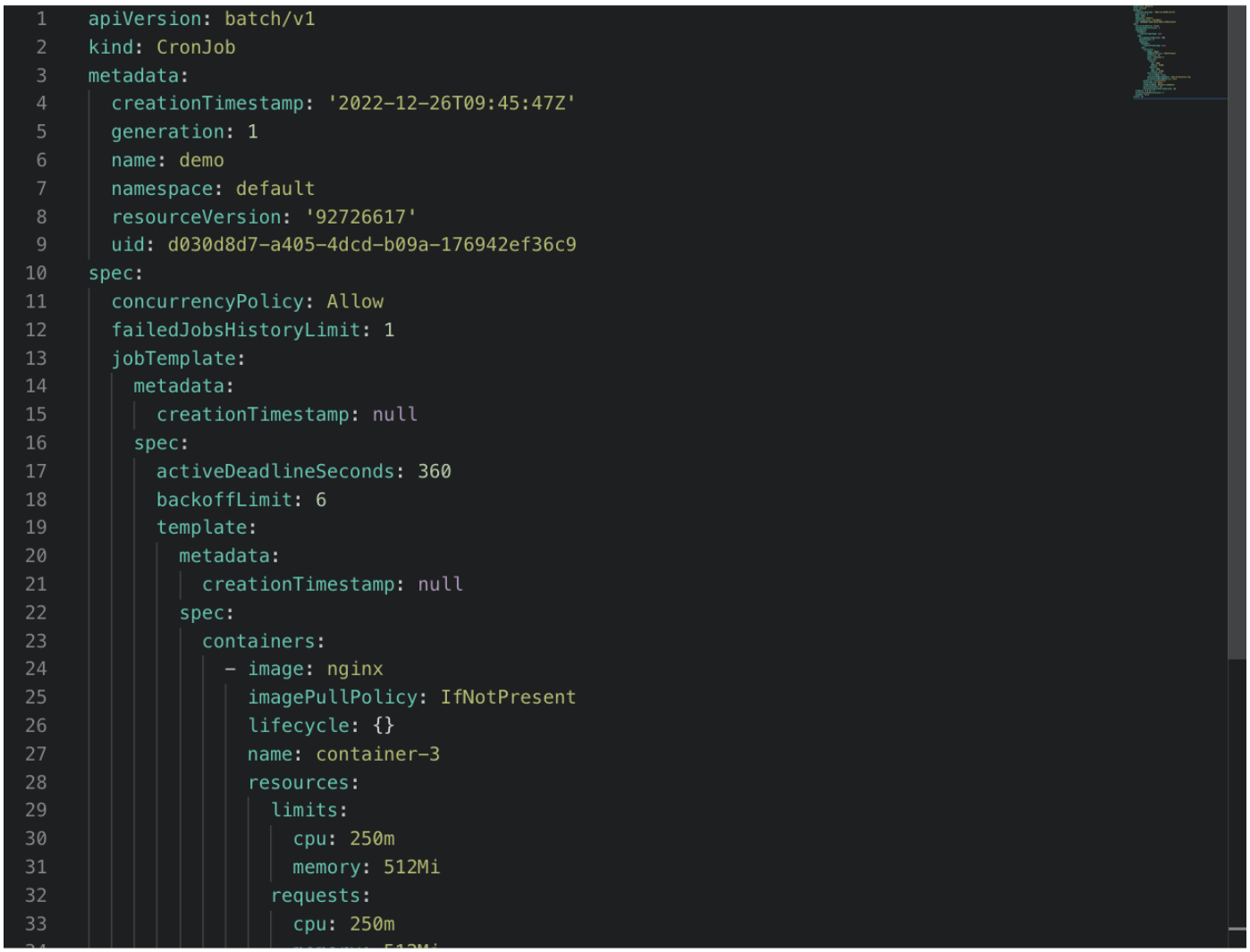
1. 点击左侧导航栏上的 集群列表，然后点击目标集群的名称，进入 集群详情 页面。



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。

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。

YAML 创建



```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
```

取消 确定

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: {}
```

工作负载参数配置

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 d.run 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

容器管理依据 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。
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。

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
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
```

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
运行参数	【类型】选填 【含义】输入控制容器运行命令参数。	port=8080

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=8080

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=8080
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。	--

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

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
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:
  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：

```
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。

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。

弹性伸缩

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

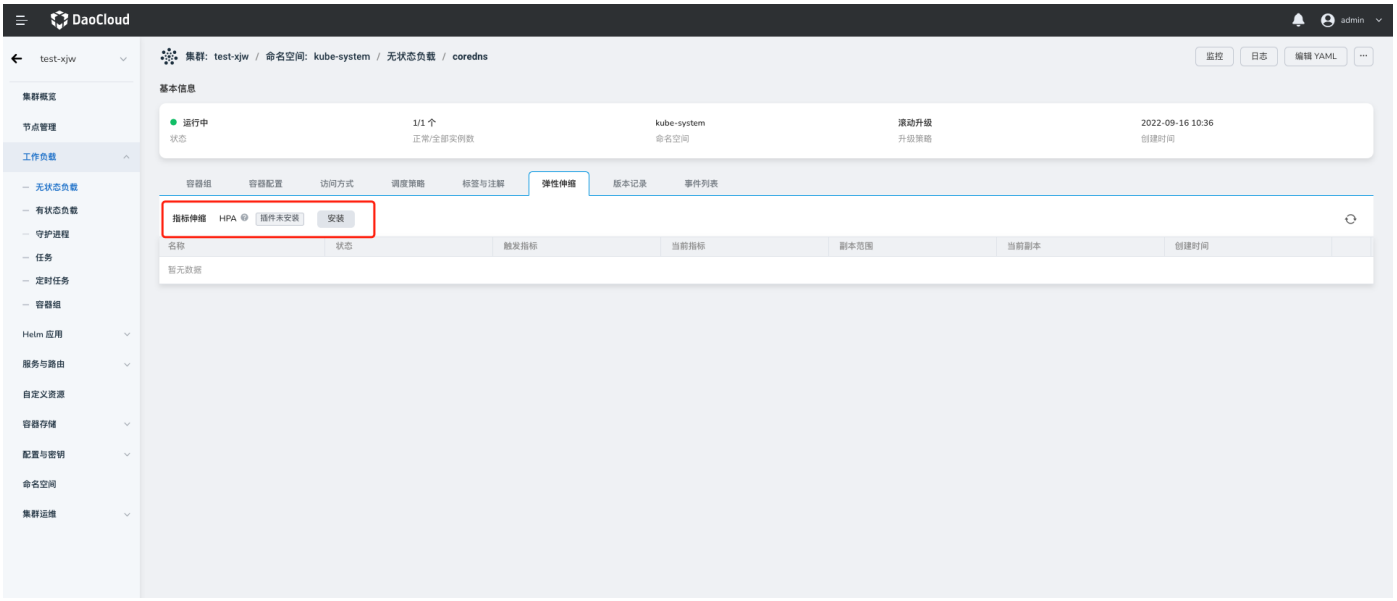
安装 **metrics-server** 插件前，需要满足以下前提条件：

- 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。

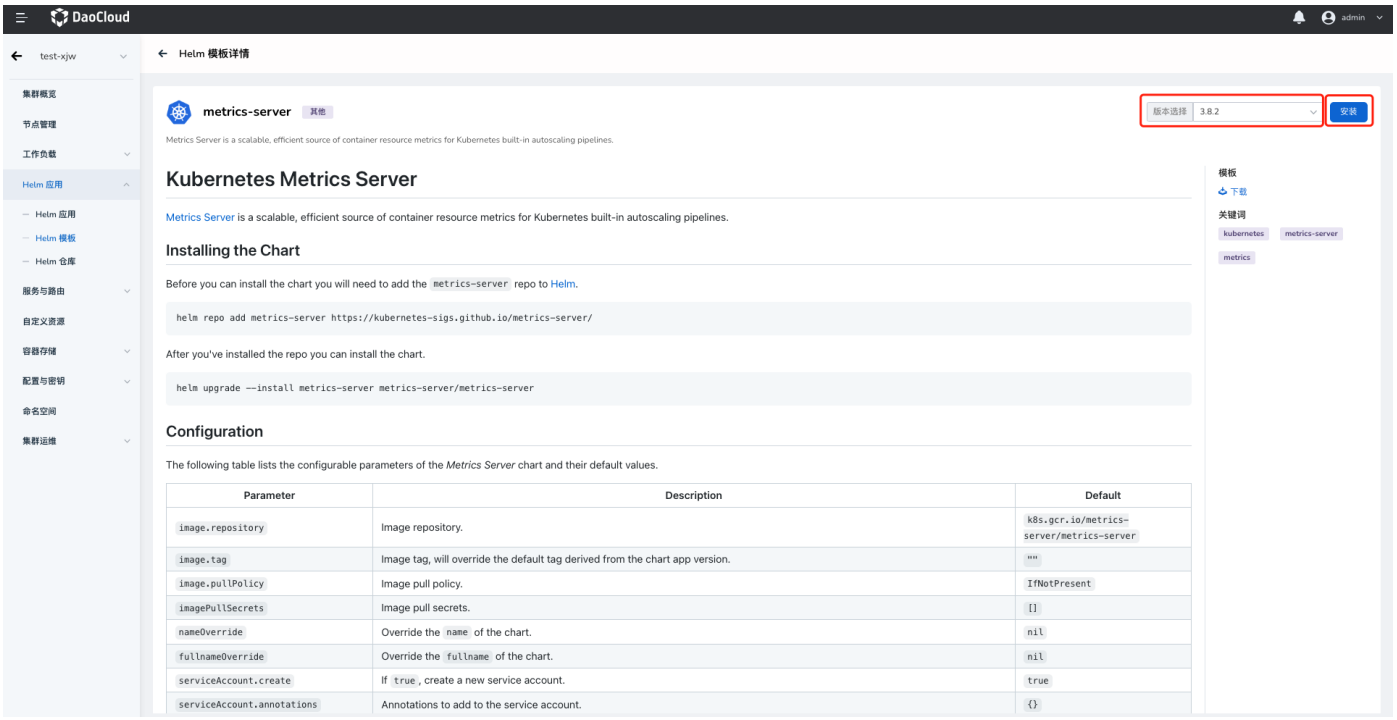
操作步骤

请执行如下步骤为集群安装 **metrics-server** 插件。

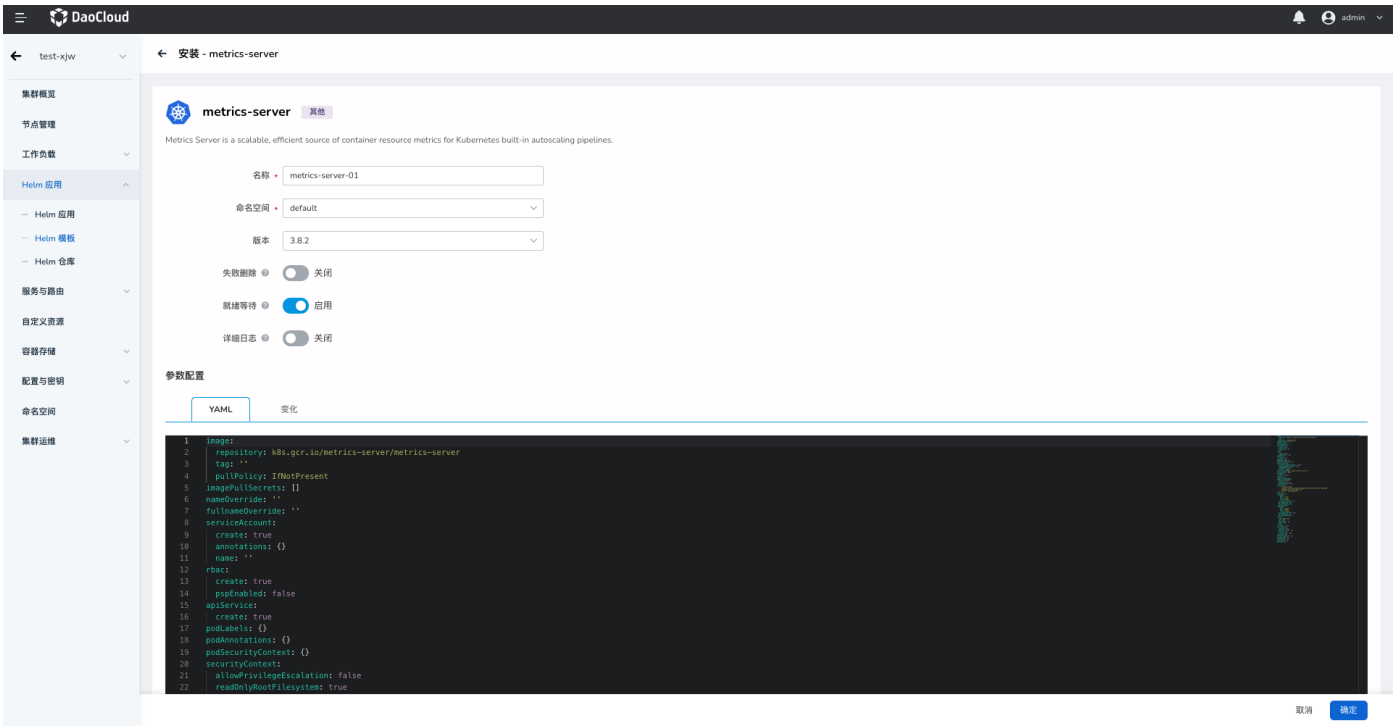
1. 在工作负载详情下的弹性伸缩页面，点击 安装 按钮，进入 **metrics-server** 插件安装界面。



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。



3. 在安装配置界面配置基本参数。



- 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。

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 KUBERNETES-CRONHPA-CONTROLLER 插件

容器副本定时水平扩缩容策略（CronHPA）能够为周期性高并发应用提供稳定的计算资源保障， **kubernetes-cronhpa-controller** 则是实现 CronHPA 的关键组件。

本节介绍如何安装 **kubernetes-cronhpa-controller** 插件。

> **Note**
>
> 为了使用 CornHPA，不仅需要安装 **kubernetes-cronhpa-controller** 插件，还要安装 **metrics-server** 插件。

前提条件

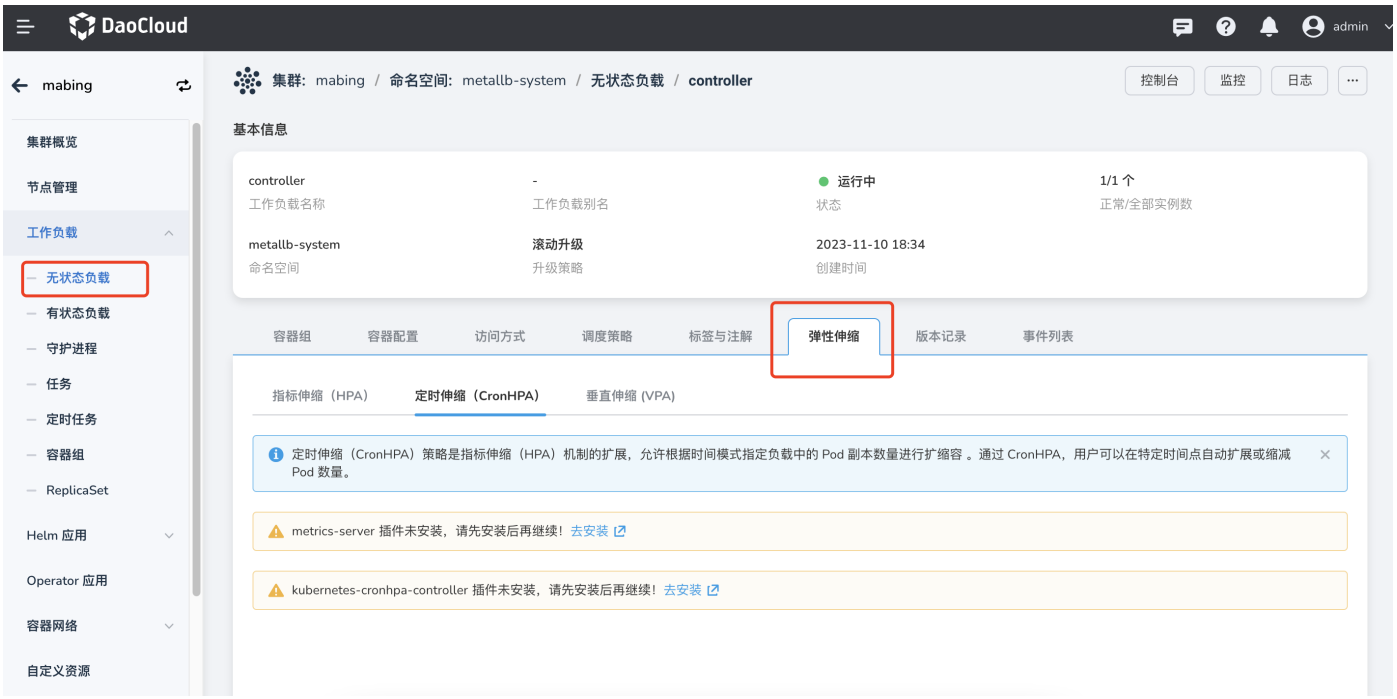
安装 **kubernetes-cronhpa-controller** 插件之前，需要满足以下前提条件：

- 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。

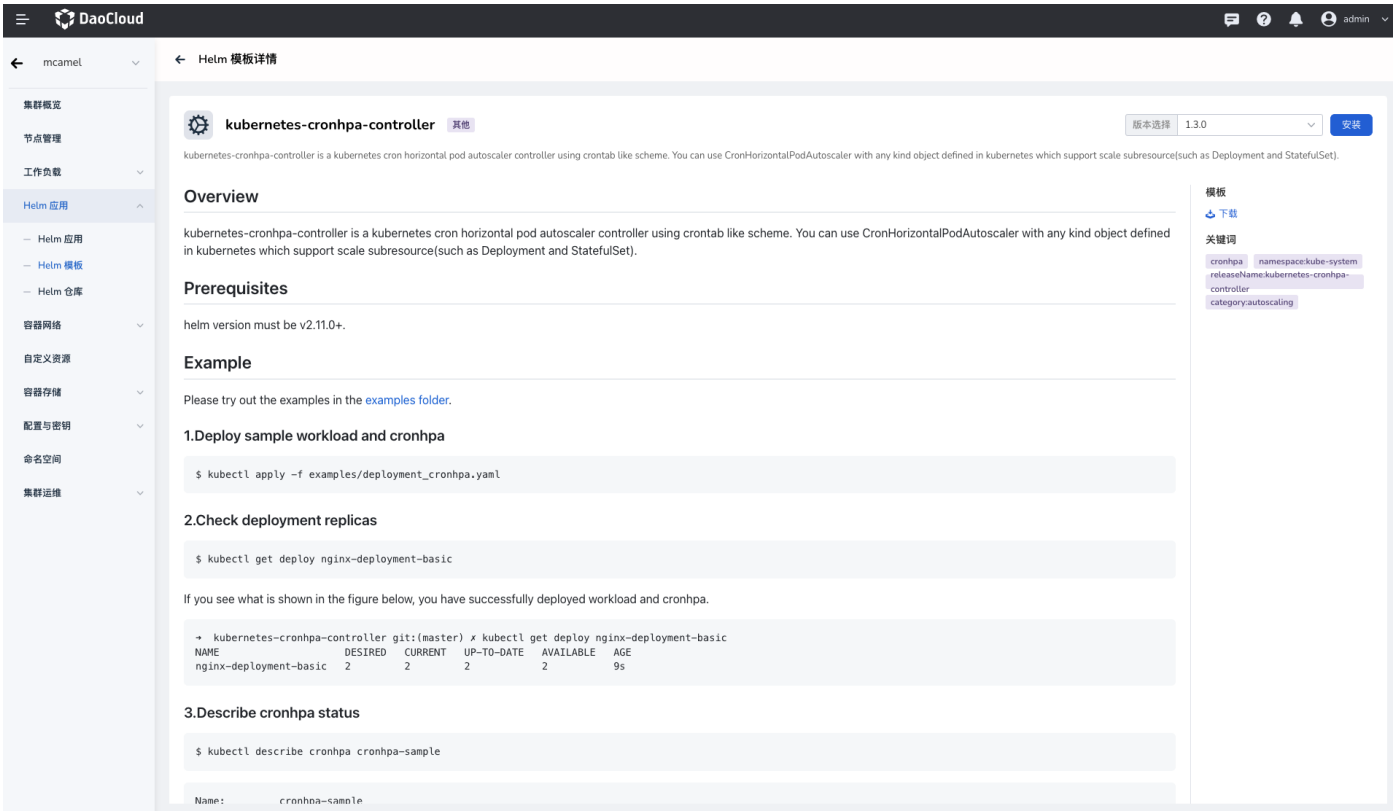
操作步骤

参考如下步骤为集群安装 **kubernetes-cronhpa-controller** 插件。

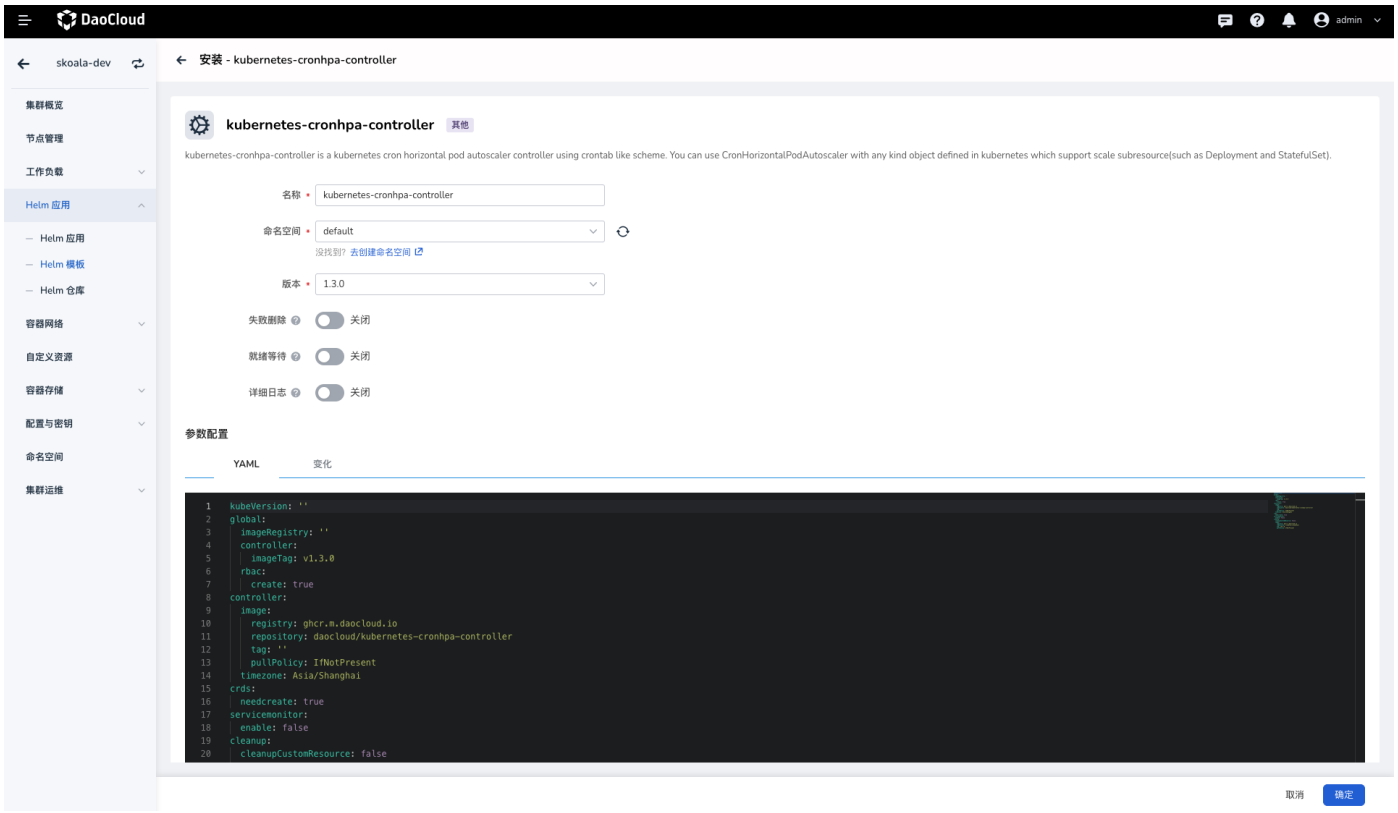
1. 在 集群列表 页面找到需要安装此插件的目标集群，点击该集群的名称，然后在左侧点击 工作负载 -> 无状态工作负载 ，点击目标工作负载的名称。
2. 在工作负载详情页面，点击 弹性伸缩 页签，在 **CronHPA** 右侧点击 安装 。



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 **1.3.0** 或更高版本。



4. 参考以下说明配置参数。



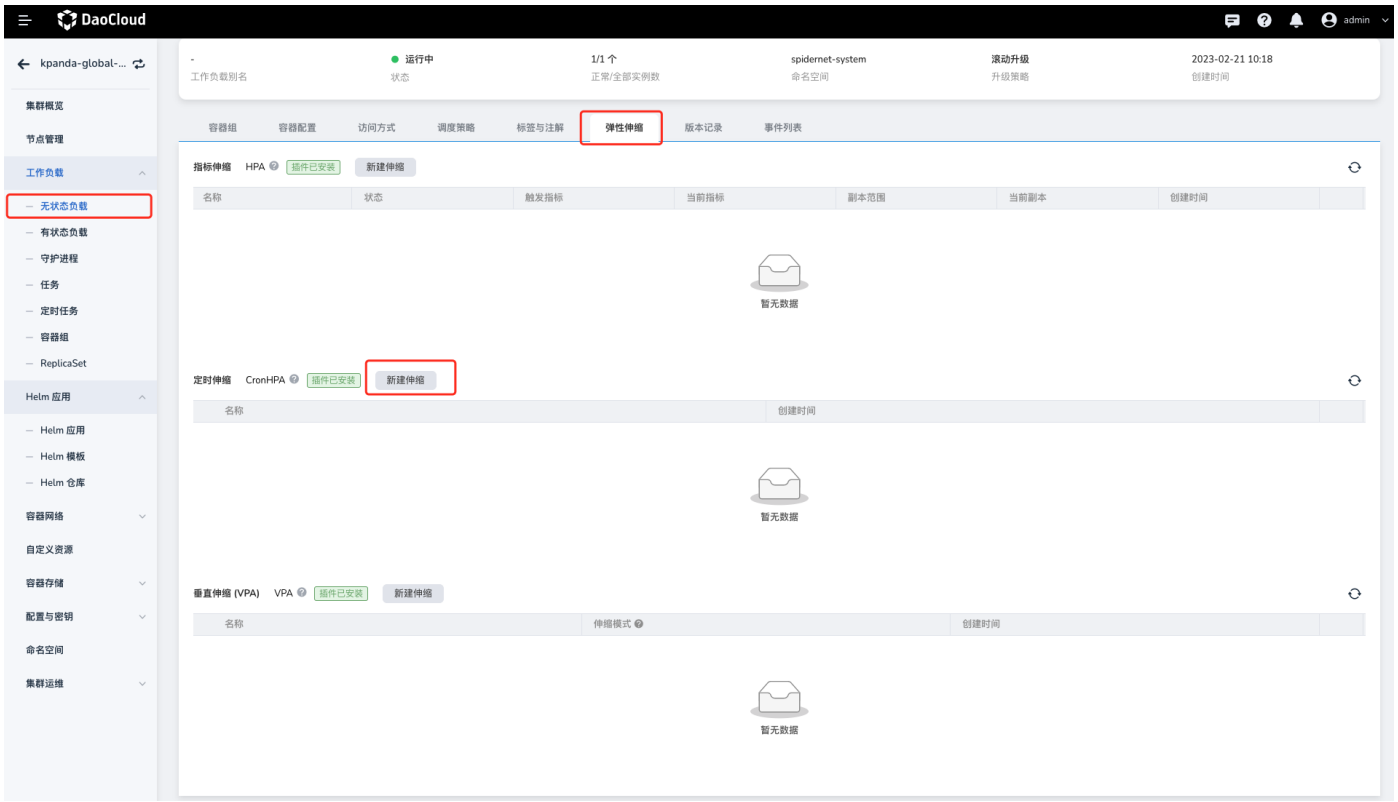
- 名称：输入插件名称，请注意名称最长 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾，例如 kubernetes-cronhpa-controller。
- 命名空间：选择将插件安装在哪个命名空间，此处以 **default** 为例。
- 版本：插件的版本，此处以 **1.3.0** 版本为例。
- 就绪等待：启用后，将等待应用下的所有关联资源都处于就绪状态，才会标记应用安装成功。
- 失败删除：如果插件安装失败，则删除已经安装的关联资源。开启后，将默认同步开启 就绪等待 。
- 详情日志：开启后，将记录安装过程的详细日志。

> **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 **kubernetes-cronhpa-controller** 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 CronHPA 策略了。


DaoCloud
admin
kpanda-global-...
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
Helm 应用
Helm 模板
Helm 仓库
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
-
工作负载别名
运行中
状态
1/1 个
正常/全部实例数
spidernet-system
命名空间
滚动升级
升级策略
2023-02-21 10:18
创建时间
容器组
容器配置
访问方式
调度策略
标签与注解
弹性伸缩
版本记录
事件列表
指标伸缩 HPA 插件已安装 新建伸缩
名称
状态
触发指标
当前指标
副本范围
当前副本
创建时间
暂无数据
定时伸缩 CronHPA 插件已安装 新建伸缩
名称
创建时间
暂无数据
垂直伸缩 (VPA) VPA 插件已安装 新建伸缩
名称
伸缩模式
创建时间
暂无数据

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。

为了使用 VPA 策略，不仅需要安装 **vpa** 插件，还要安装 **metrics-server** 插件。

前提条件

安装 **vpa** 插件之前，需要满足以下前提条件：

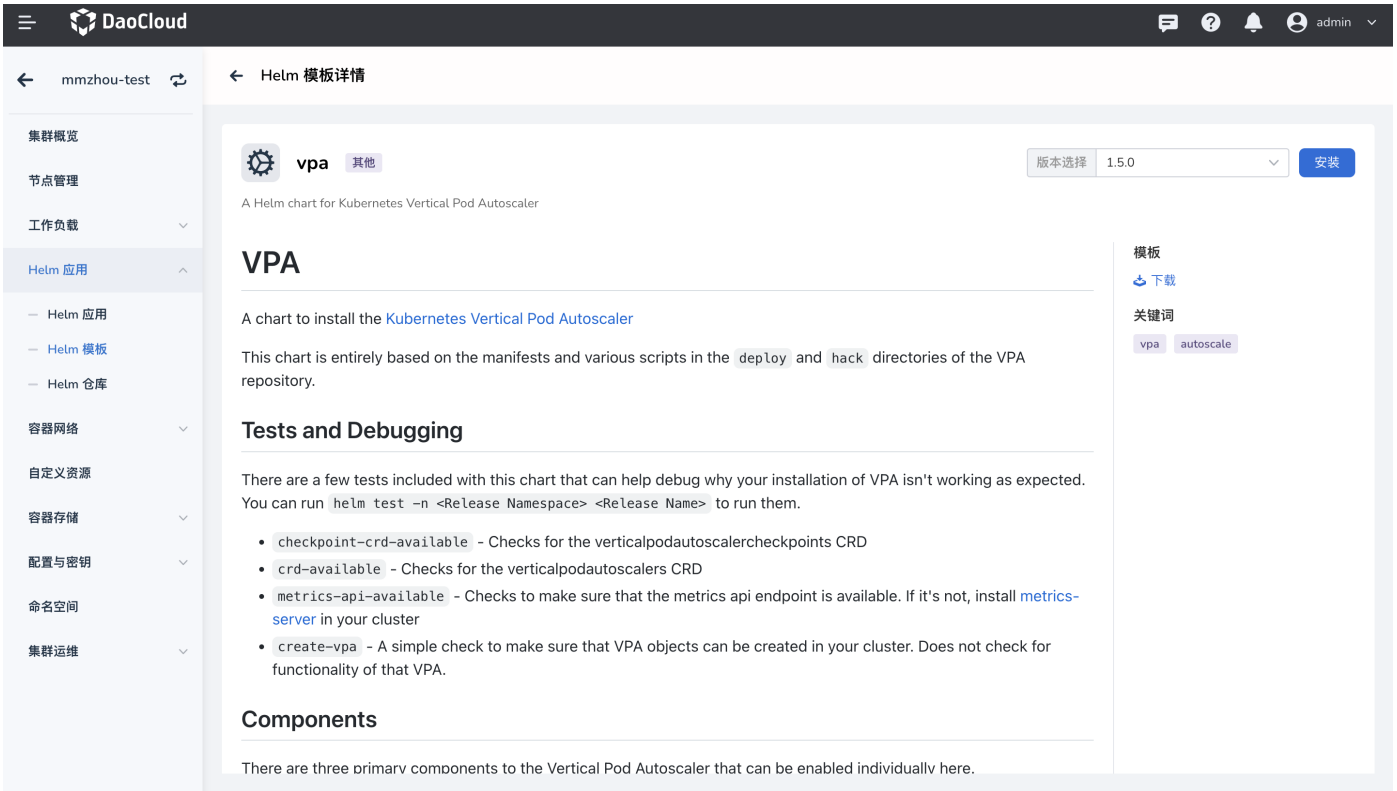
- 创建一个命名空间。
- 当前操作用户应具有 NS Edit 或更高权限，详情可参考命名空间授权。

操作步骤

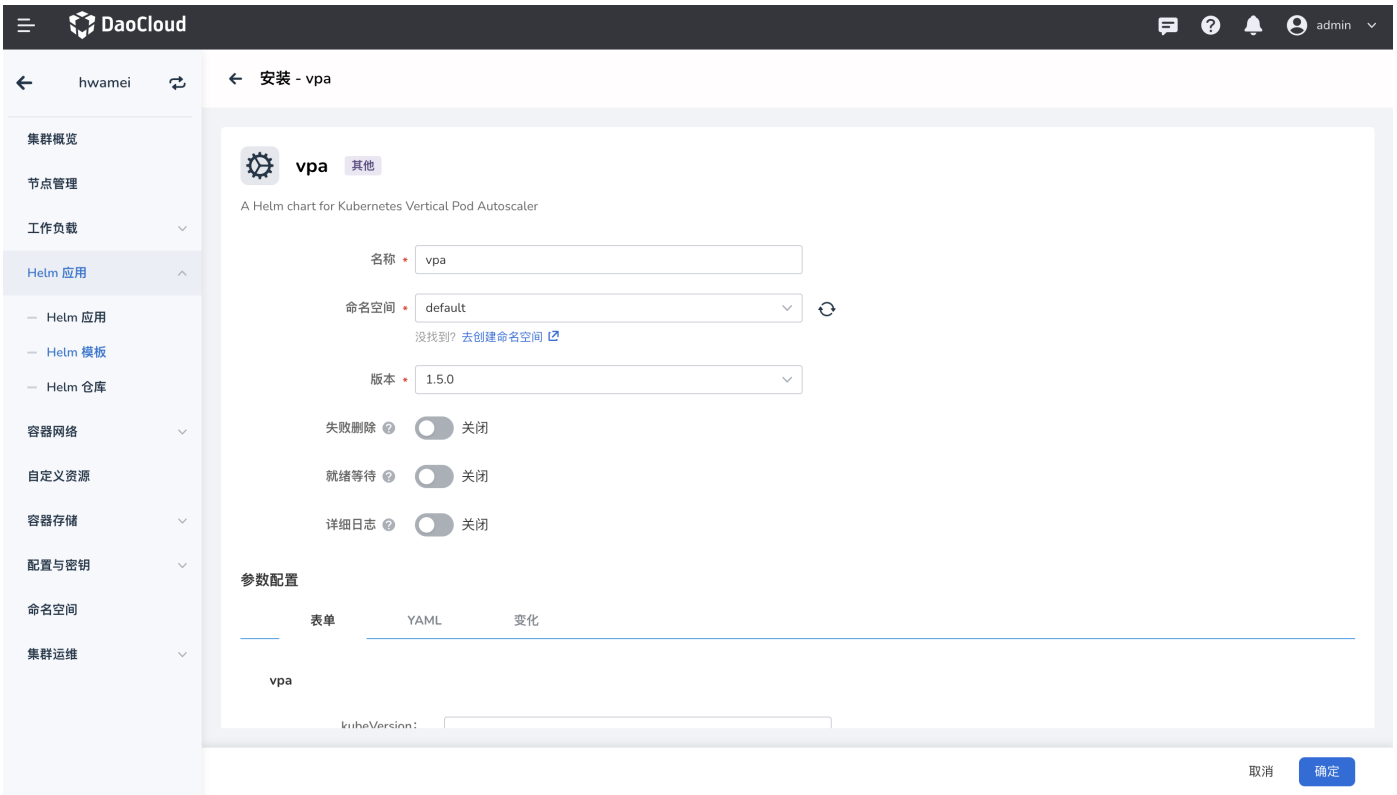
参考如下步骤为集群安装 **vpa** 插件。

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 。

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。



4. 查看以下说明配置参数。



- 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。


垂直伸缩 (VPA)
VPA
插件已安装
新建伸缩
名称
伸缩模式
创建时间
暂无数据

创建 HPA

d.run 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/低于您设置的指标阀值，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

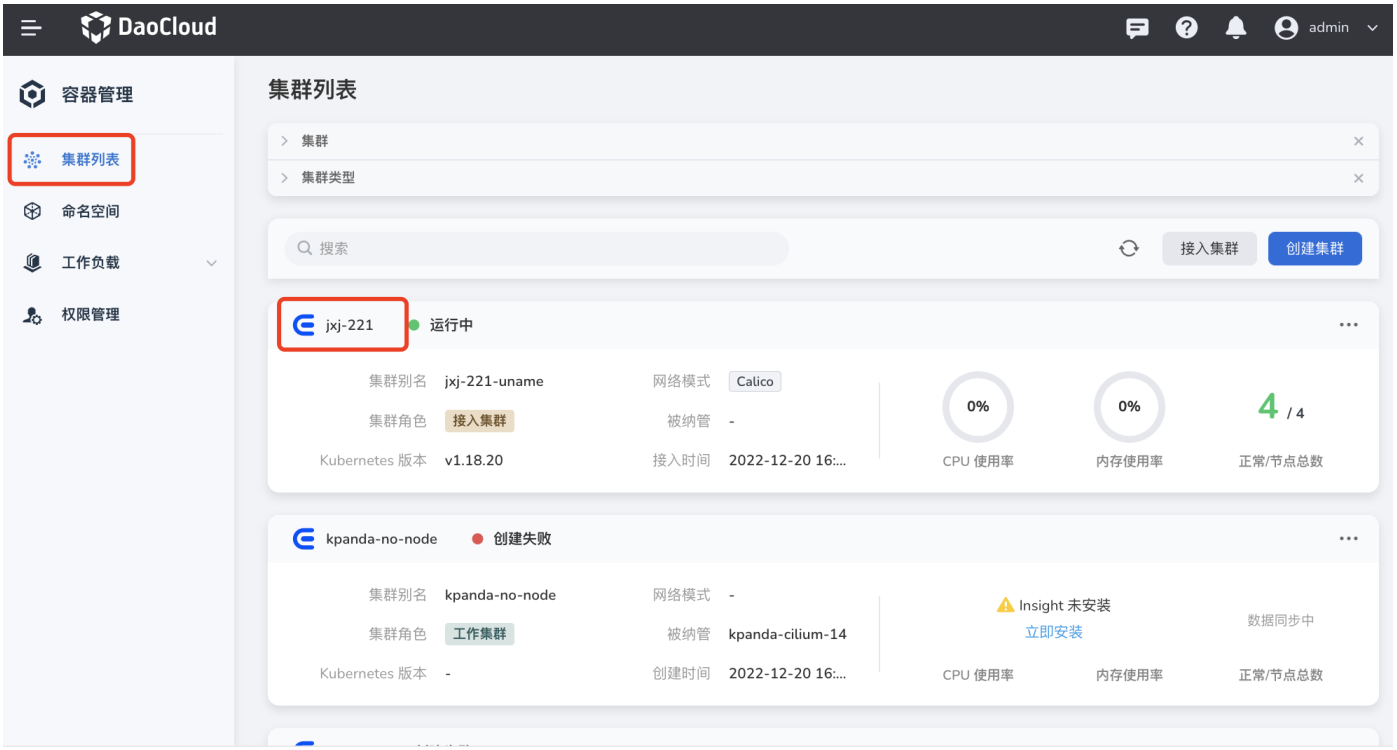
在为工作负载配置内置指标弹性伸缩策略之前，需要满足以下前提条件：

- 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。

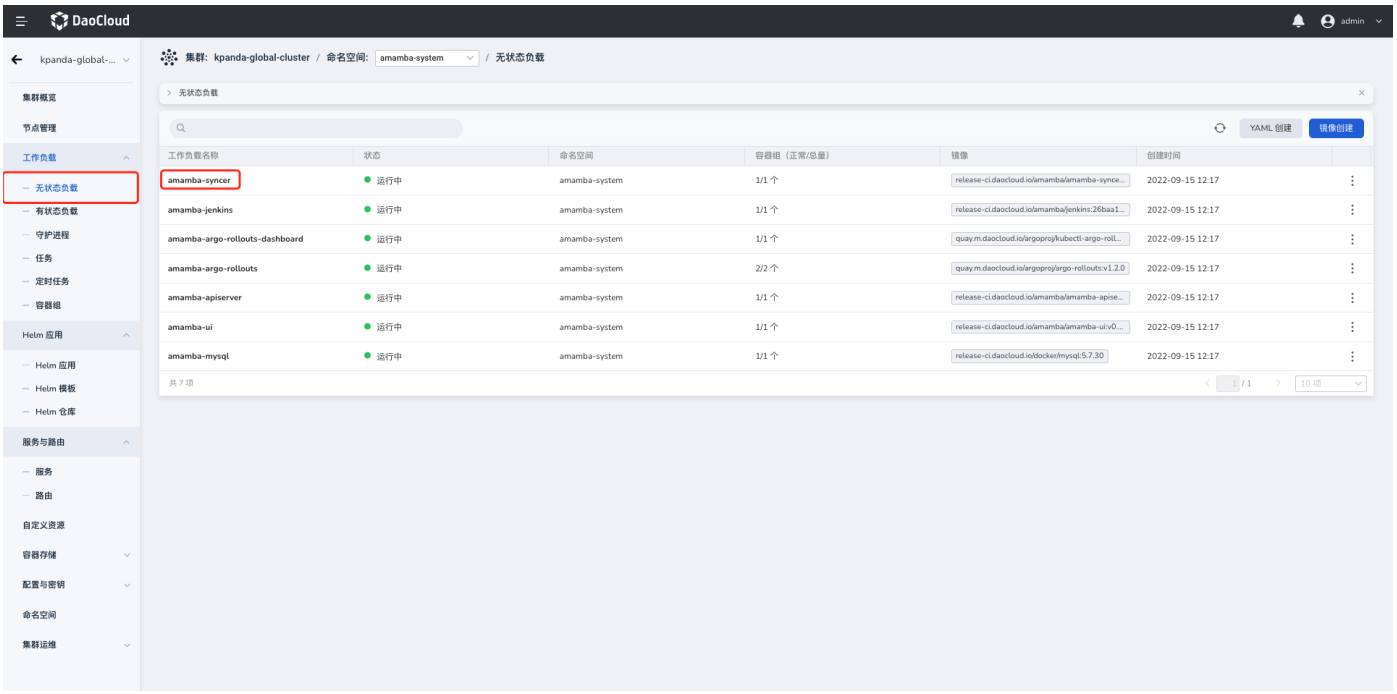
操作步骤

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。

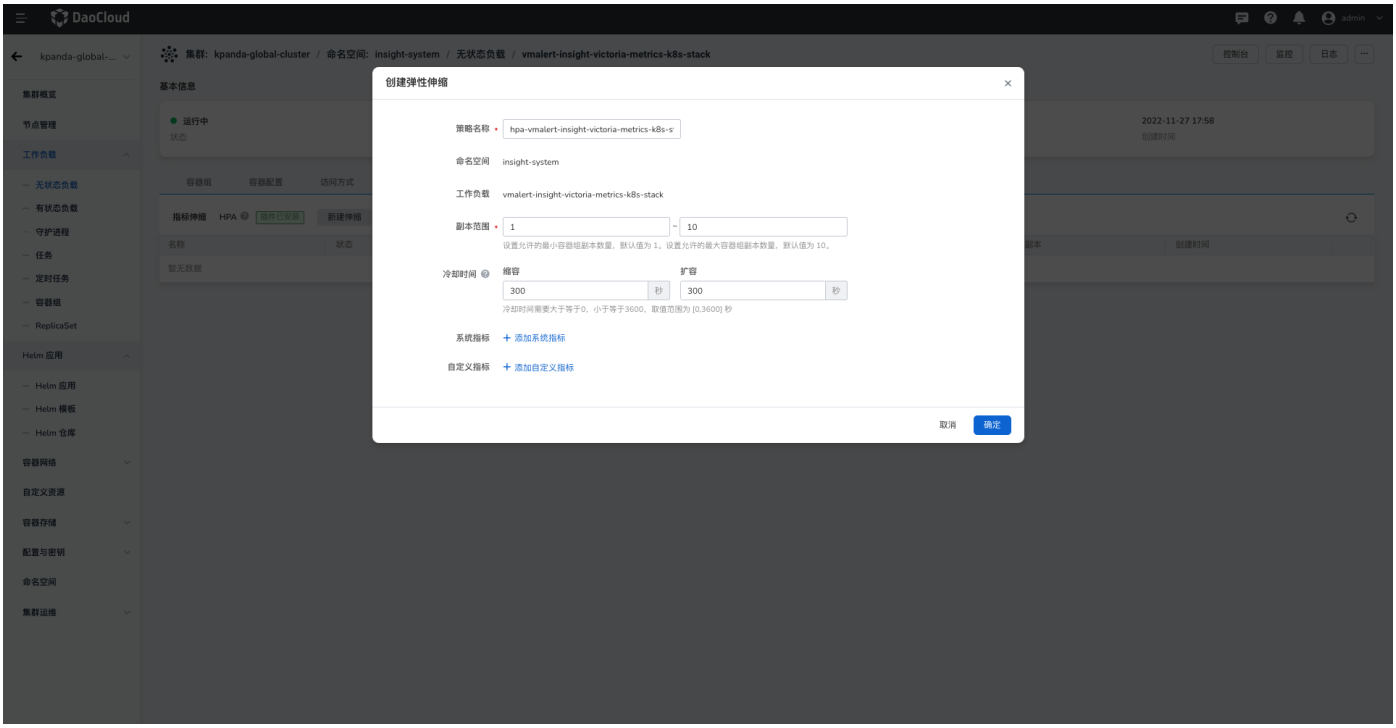


2. 在集群详情页面，点击左侧导航栏的 工作负载 进入工作负载列表后，点击一个负载名称，进入 工作负载详情 页面。

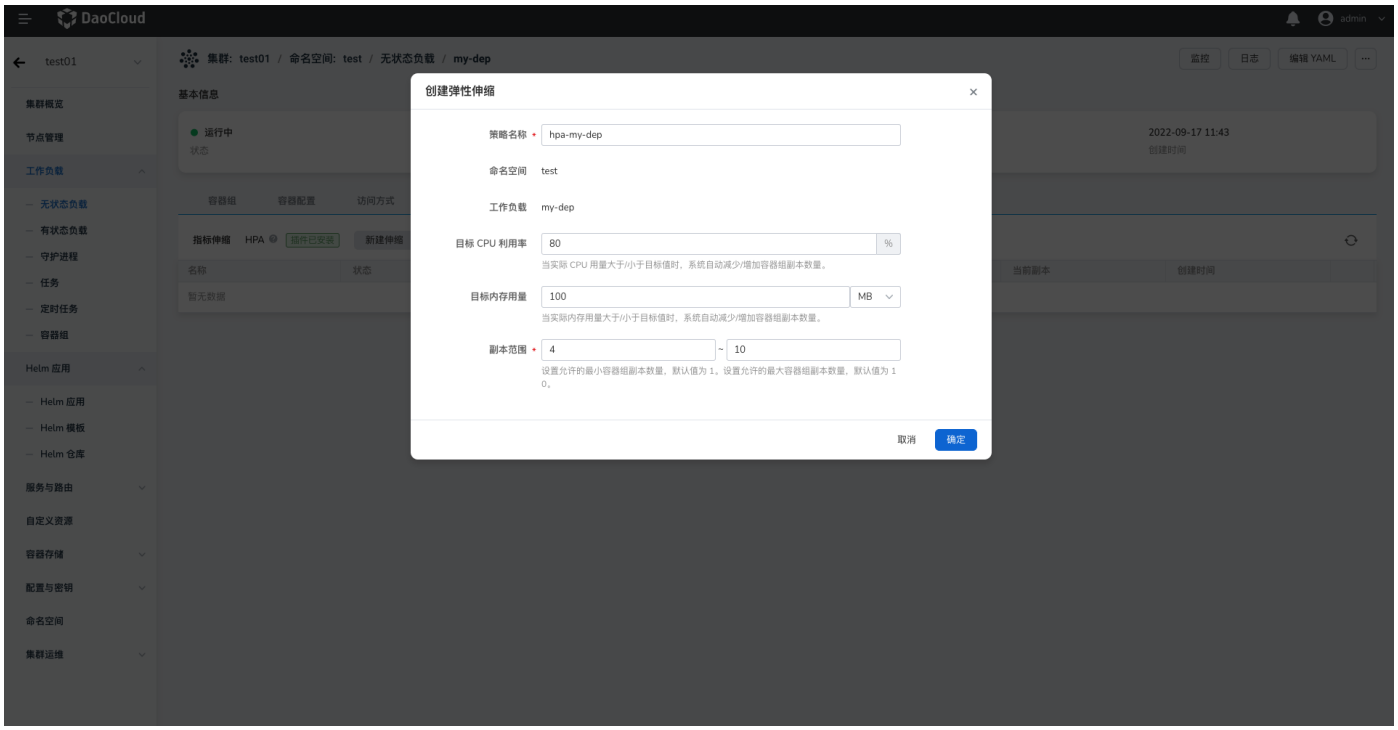


3. 点击 弹性伸缩 页签，查看当前集群的弹性伸缩配置情况。

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。



5. 创建自定义指标弹性伸缩策略参数。



- 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。

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。

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

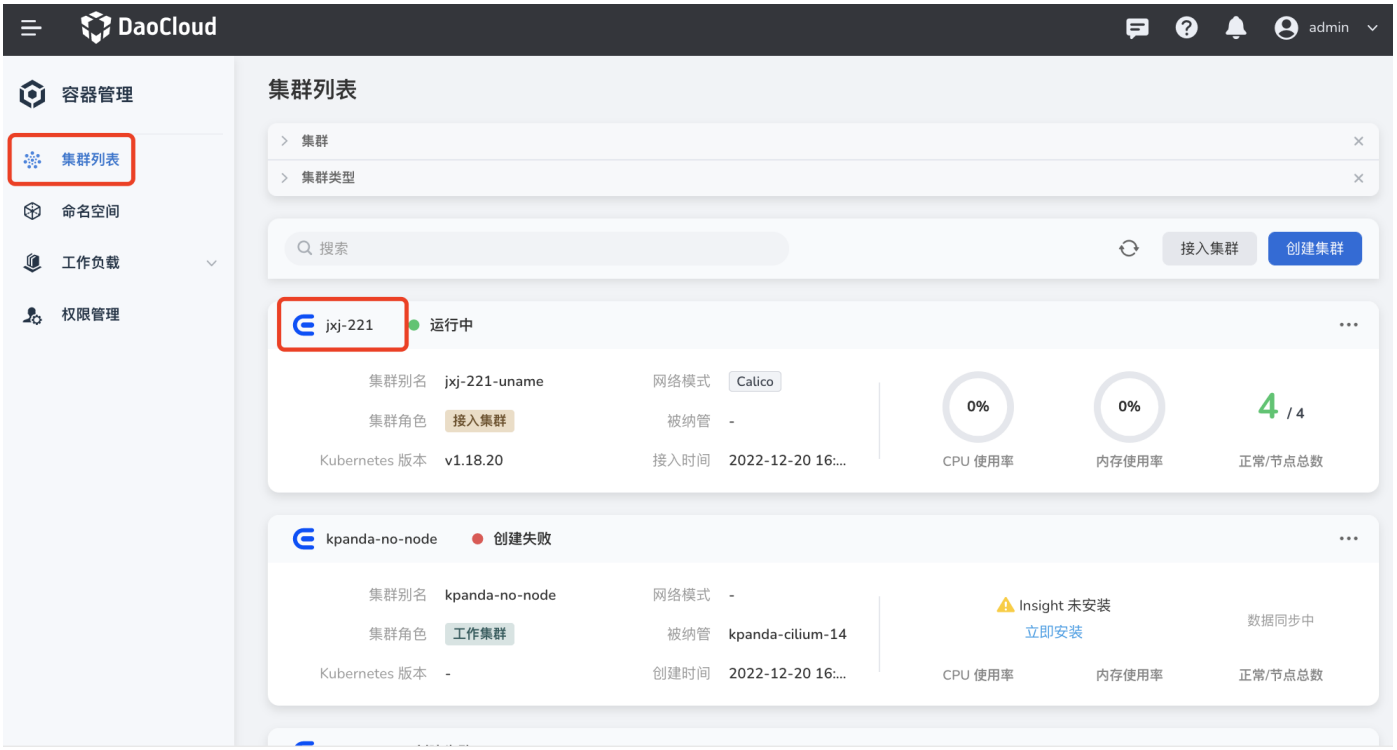
在为工作负载配置自定义指标弹性伸缩策略之前，需要满足以下前提条件：

- 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。

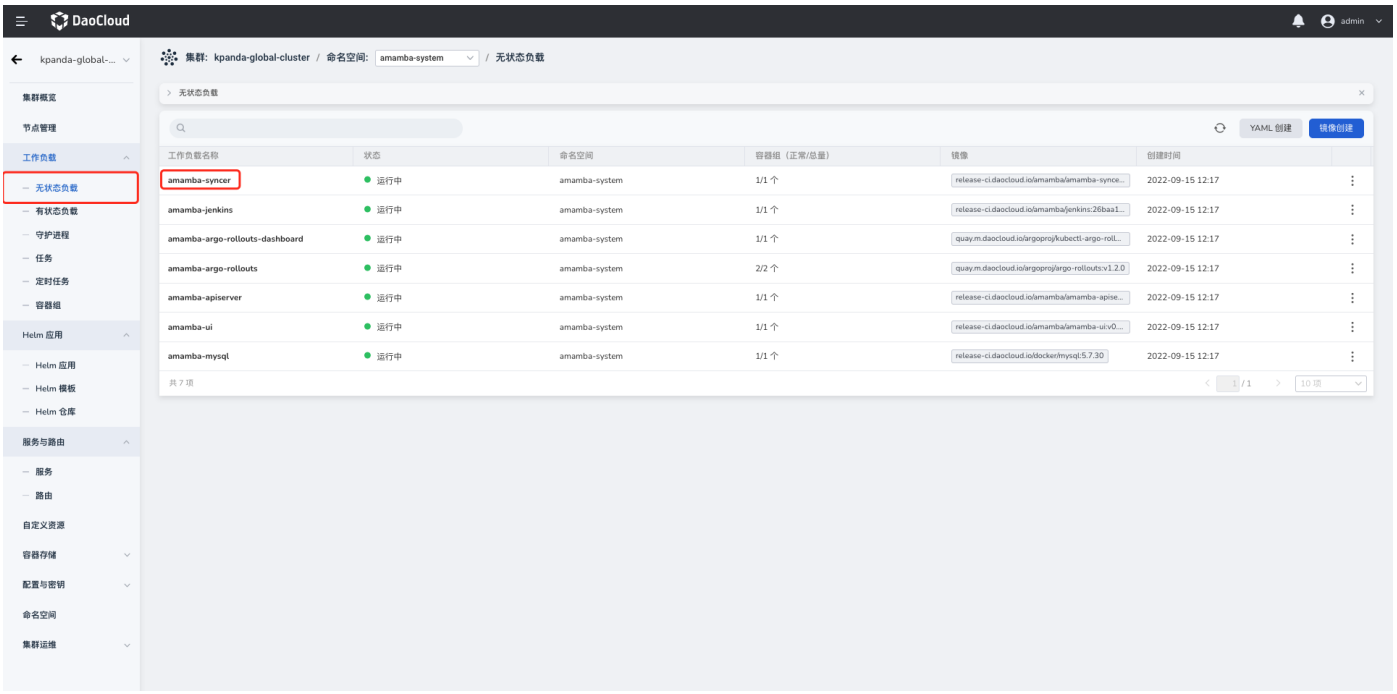
操作步骤

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。



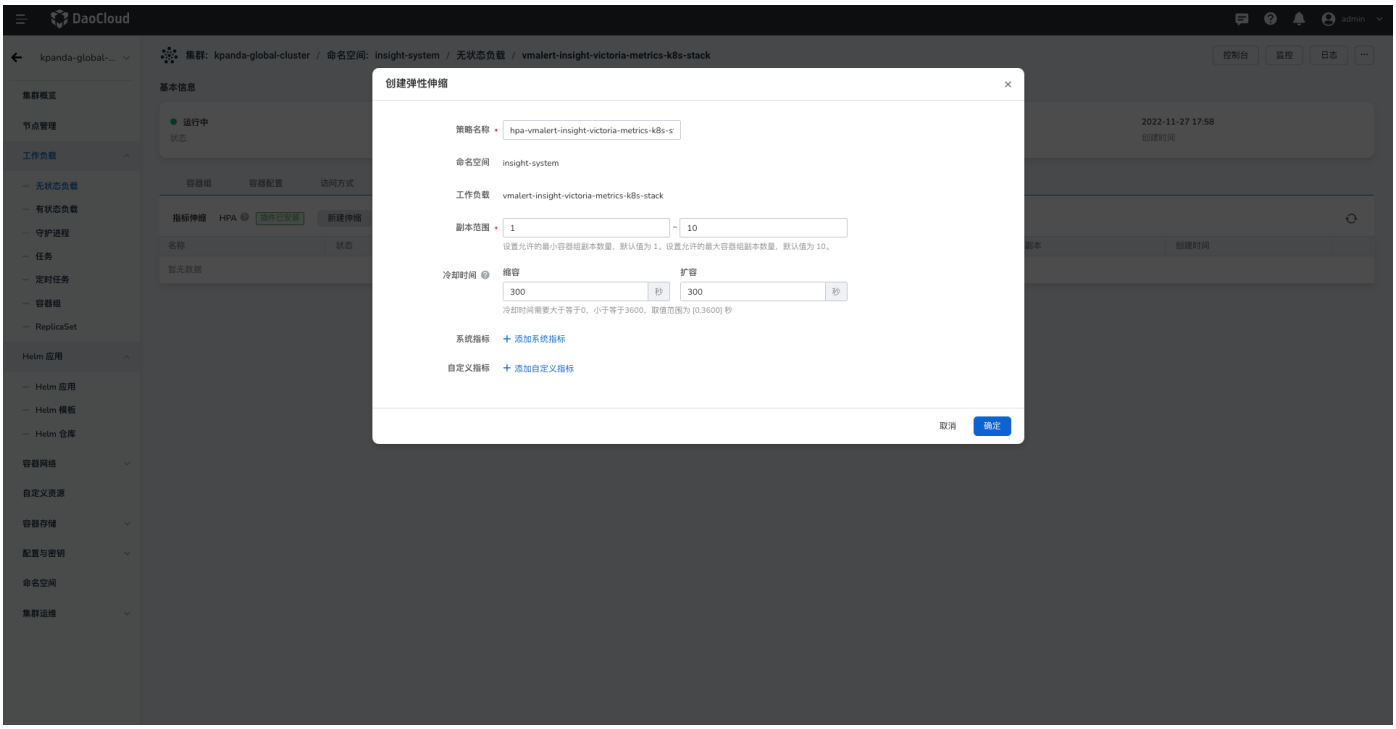
2. 在集群详情页面，点击左侧导航栏的 工作负载 进入工作负载列表后，点击一个负载名称，进入 工作负载详情 页面。



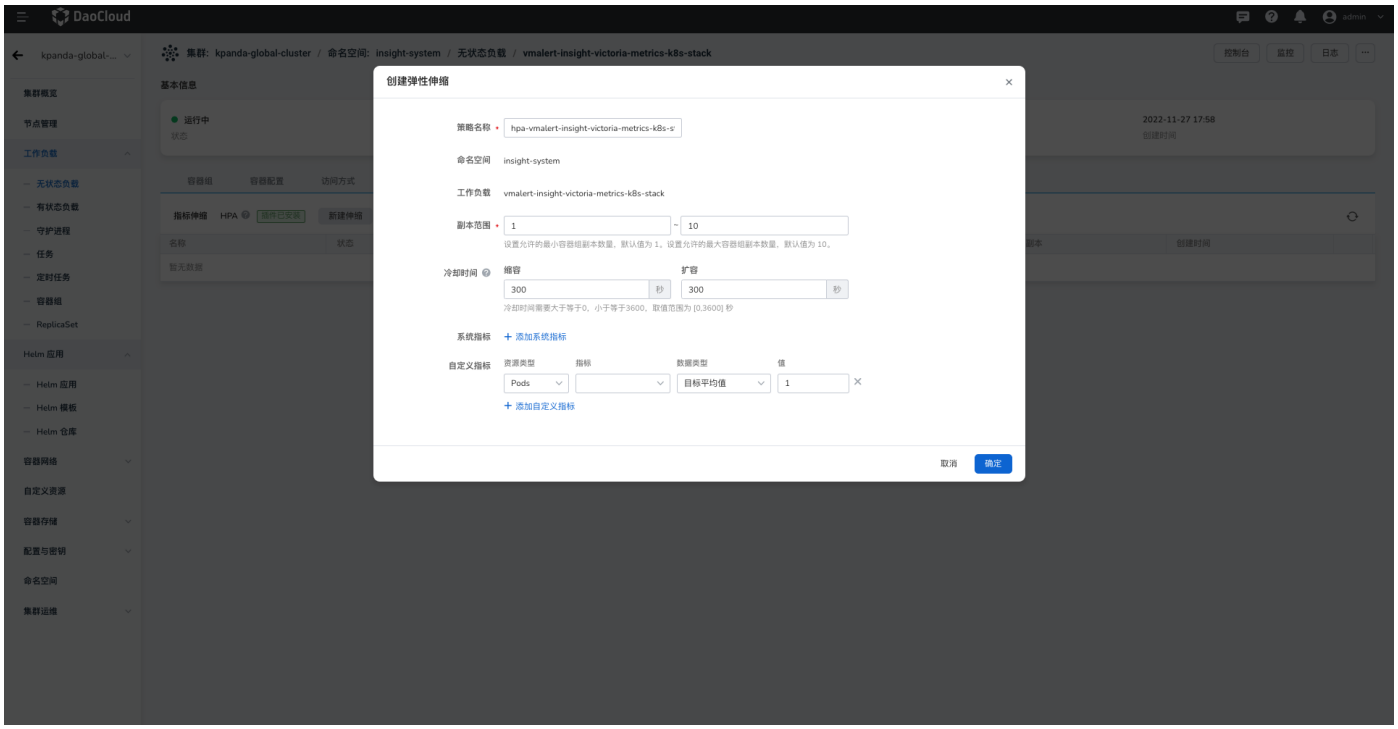
3. 点击 弹性伸缩 页签，查看当前集群的弹性伸缩配置情况。

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。



5. 创建自定义指标弹性伸缩策略参数。



- 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。
- 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。

d.run 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。 d.run 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

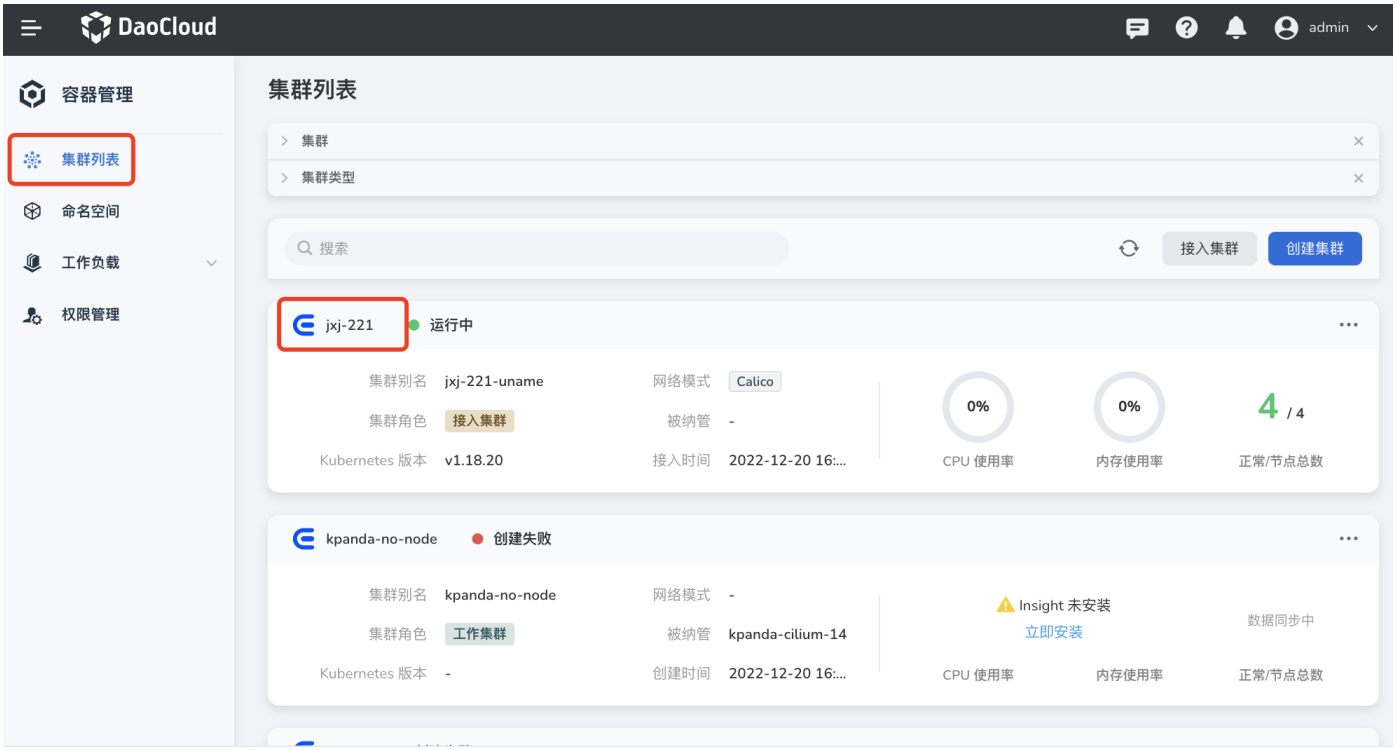
为工作负载配置垂直伸缩策略之前，需要满足以下前提条件：

- 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。

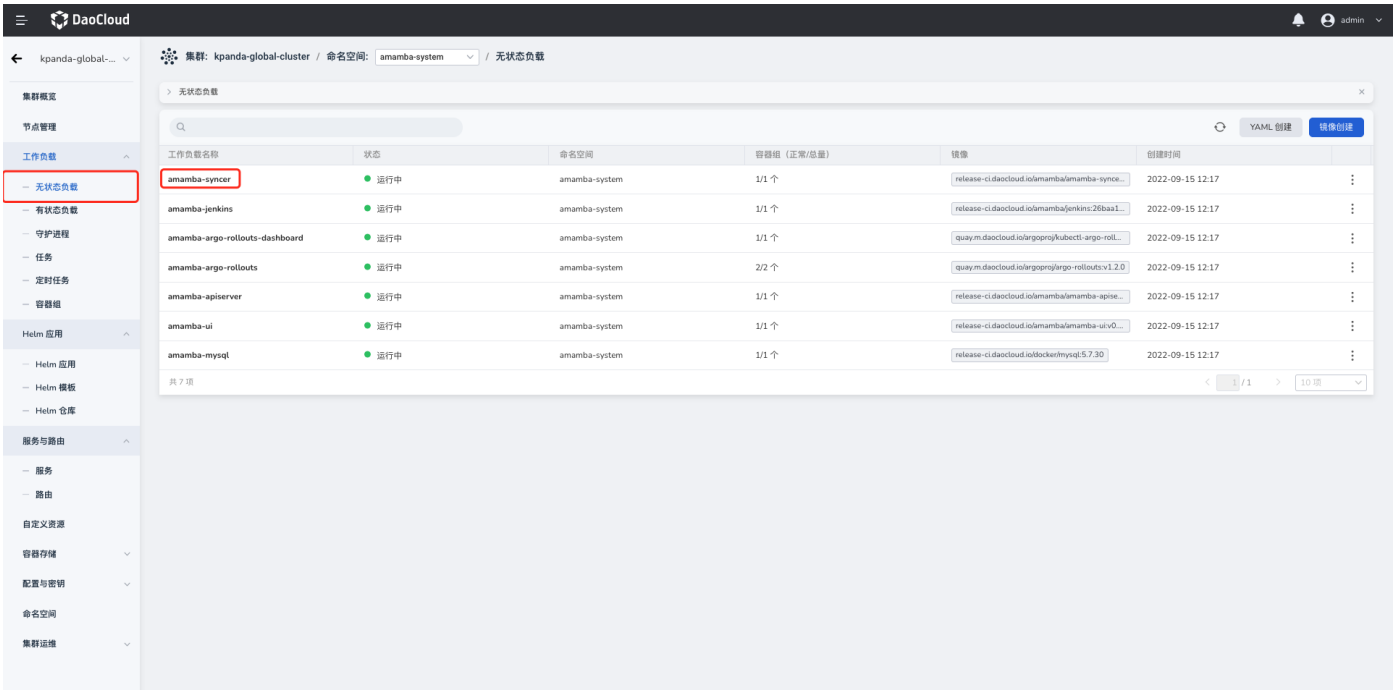
操作步骤

参考以下步骤，为工作负载配置内置指标弹性伸缩策略。

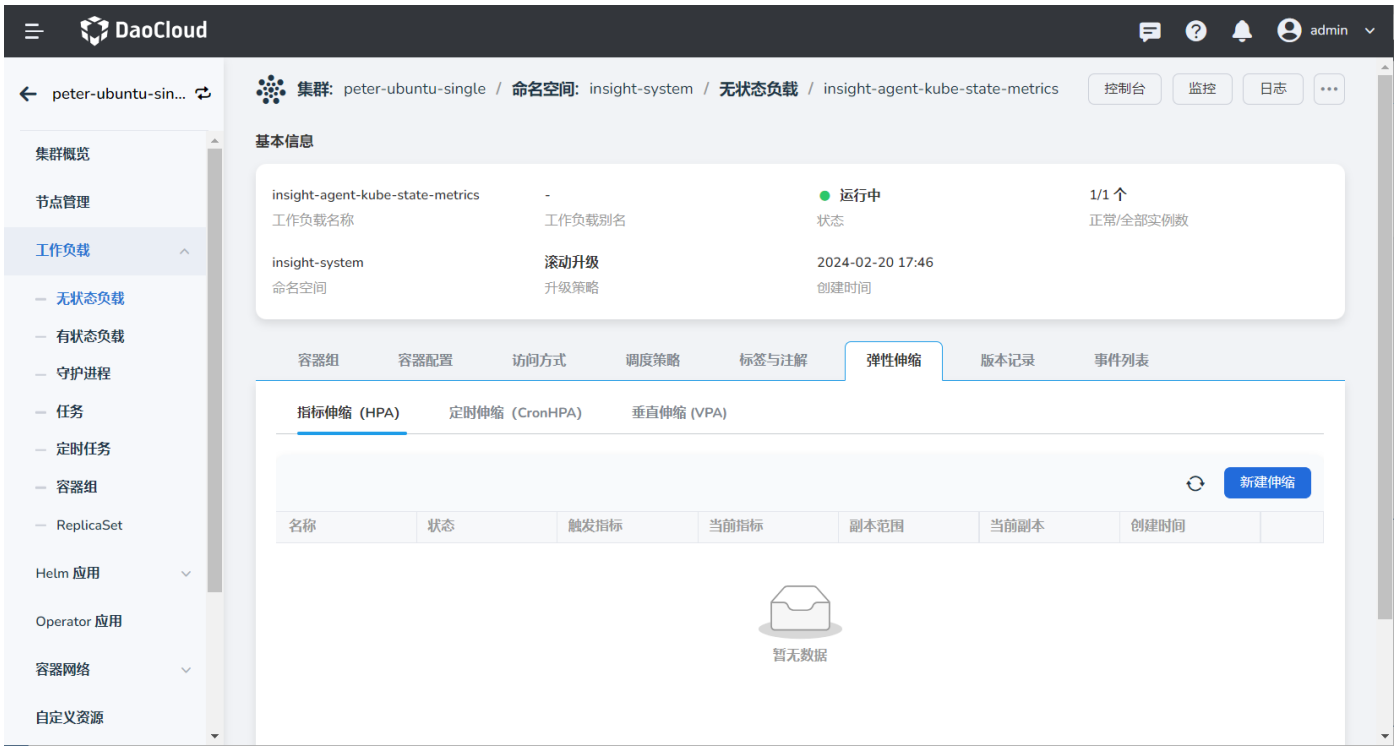
1. 在 集群列表 中找到目前集群，点击目标集群的名称。



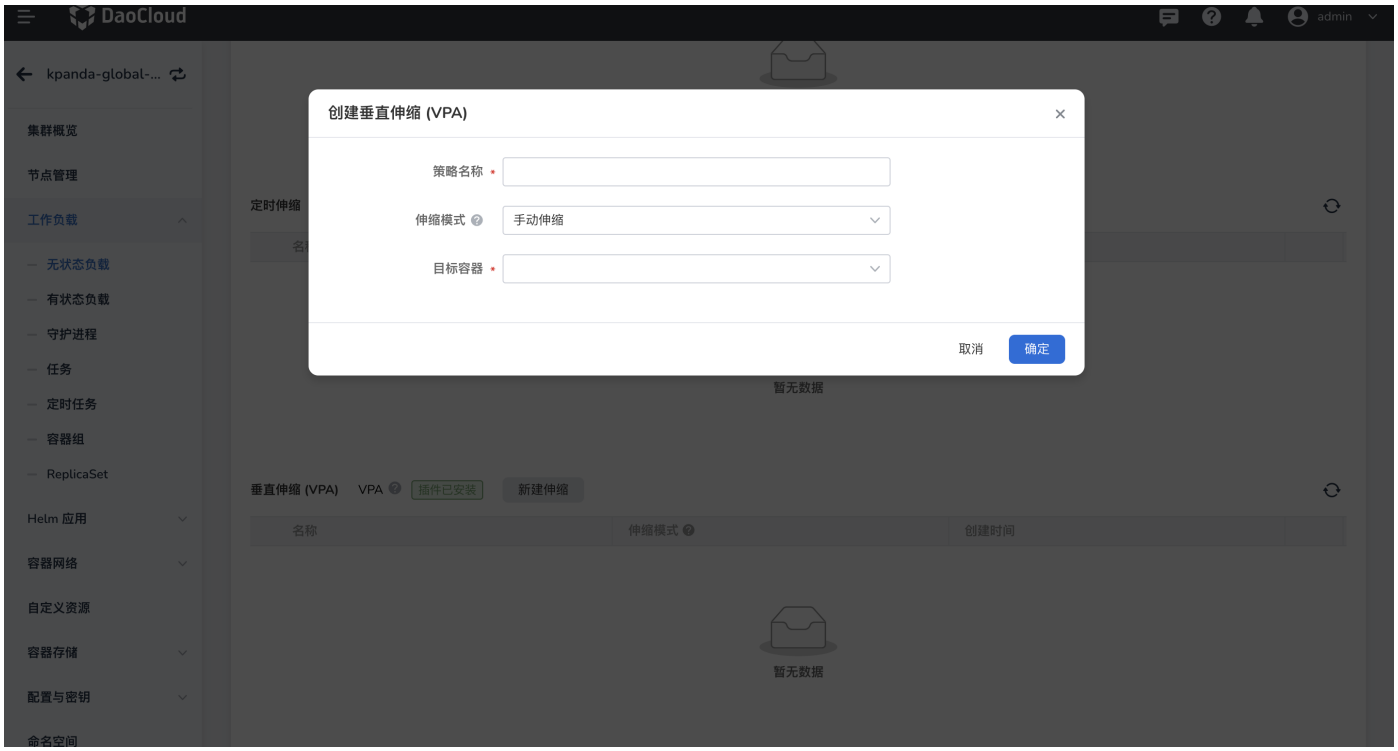
2. 在左侧导航栏点击 工作负载 ，找到需要创建 VPA 的负载，点击该负载的名称。



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。

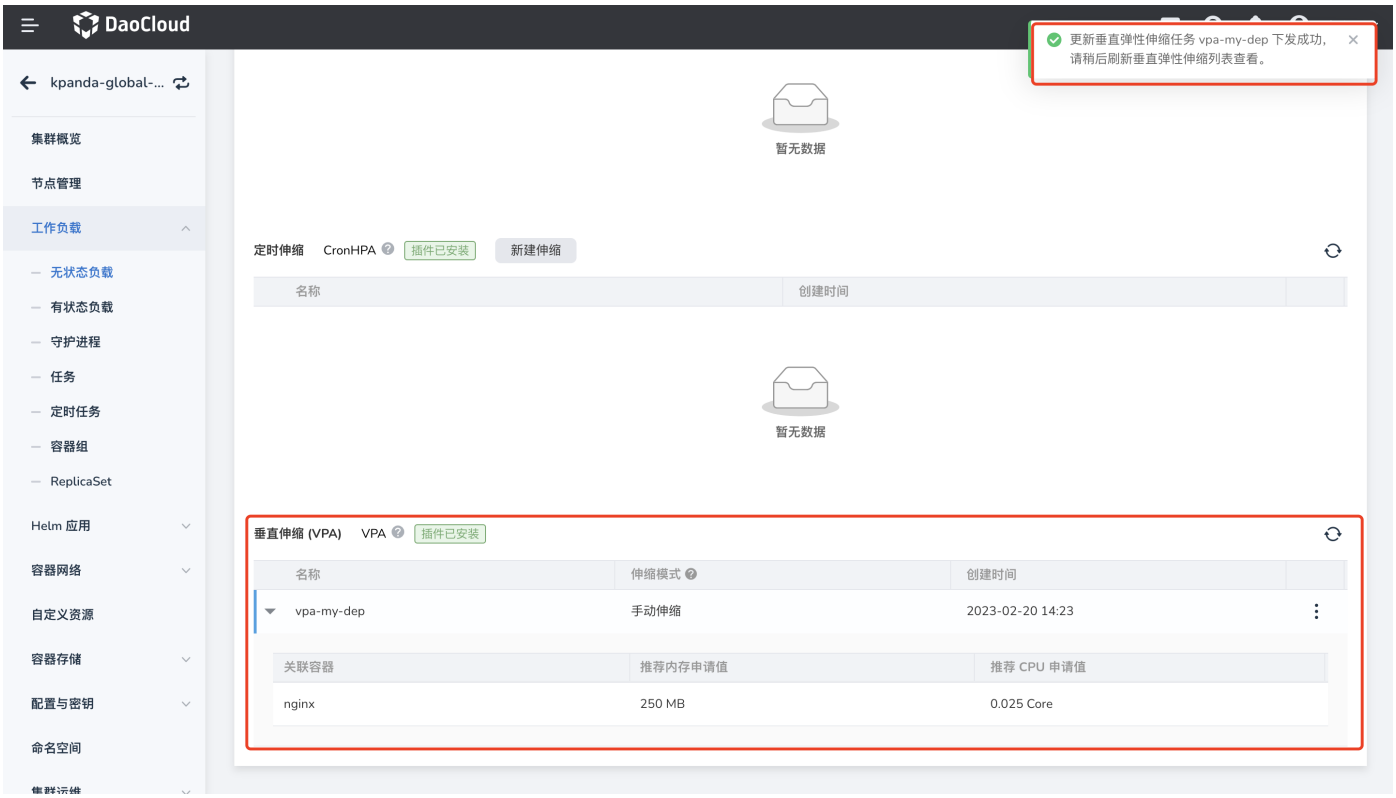


4. 点击 新建伸缩 按钮，并配置 VPA 垂直伸缩策略参数。



- 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。


DaoCloud
更新垂直弹性伸缩任务 vpa-my-dep 下发成功，
请稍后刷新垂直弹性伸缩列表查看。
kpanda-global-...
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
暂无数据
定时伸缩 CronHPA 插件已安装 新建伸缩
名称
创建时间
暂无数据
垂直伸缩 (VPA) VPA 插件已安装
名称
伸缩模式
创建时间
vpa-my-dep
手动伸缩
2023-02-20 14:23
关联容器
推荐内存申请值
推荐 CPU 申请值
nginx
250 MB
0.025 Core

HPA 和 CRONHPA 兼容规则

CronHPA 和 HPA 兼容冲突

定时伸缩 CronHPA 通过设置定时的方式触发容器的水平副本伸缩。为了防止突发的流量冲击等状况，您可能已经配置 HPA 保障应用的正常运行。 如果同时检测到了 HPA 和 CronHPA 的存在，由于 CronHPA 和 HPA 相互独立无法感知，就会出现两个控制器各自工作，后执行的操作会覆盖先执行的操作。

对比 CronHPA 和 HPA 的定义模板，可以观察到以下几点：

- CronHPA 和 HPA 都是通过 scaleTargetRef 字段来获取伸缩对象。
- CronHPA 通过 jobs 的 crontab 规则定时伸缩副本数。
- HPA 通过资源利用率判断伸缩情况。

说明：如果同时设置 CronHPA 和 HPA，会出现 CronHPA 和 HPA 同时操作一个 scaleTargetRef 的场景。

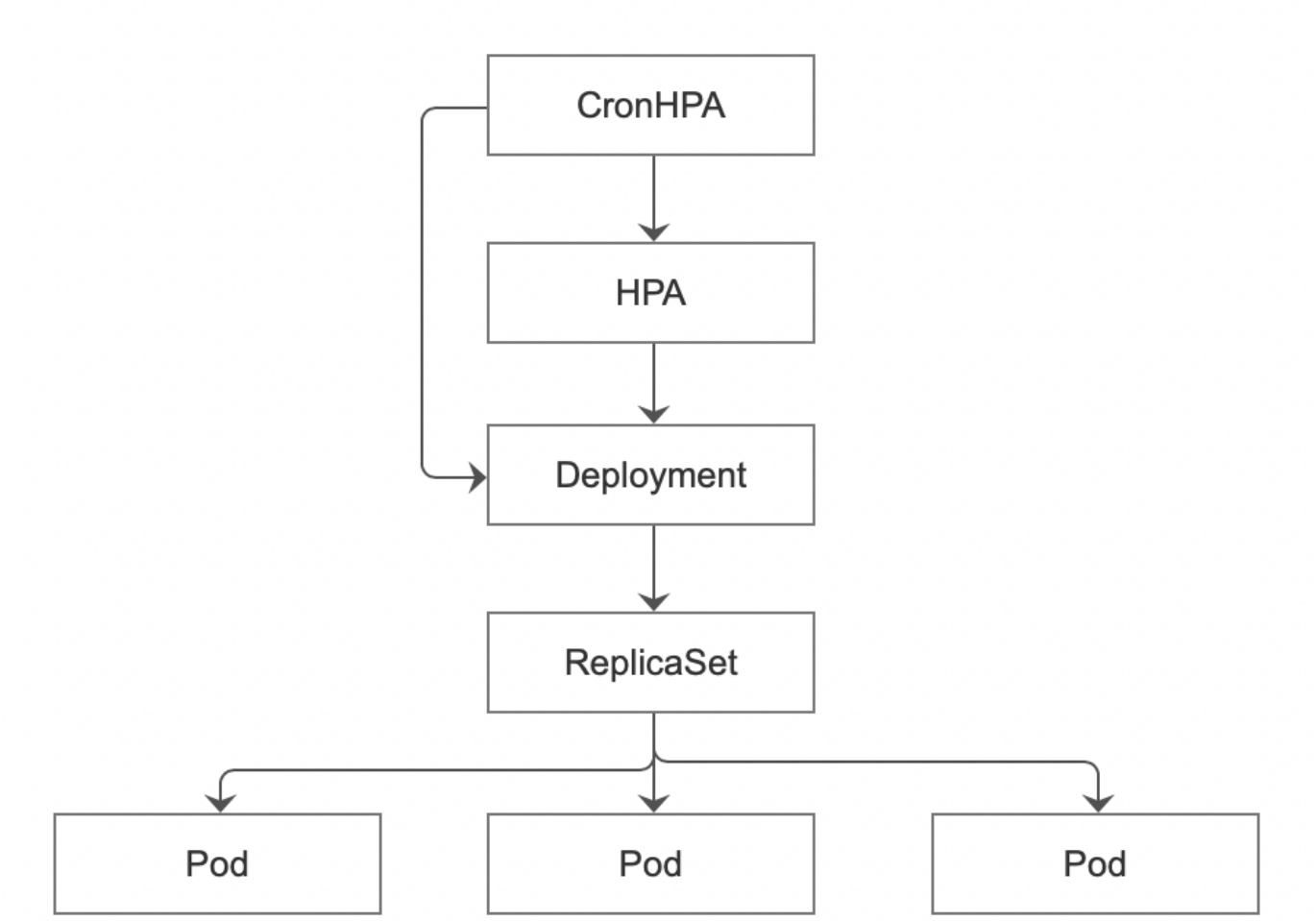
CronHPA 和 HPA 兼容方案

从上文可知，CronHPA 和 HPA 同时使用会导致后执行的操作覆盖先执行操作的本质原因是两个控制器无法相互感知，那么只需要让 CronHPA感知 HPA 的当前状态就能解决冲突问题。

系统会将 HPA 作为定时伸缩 CronHPA 的扩缩容对象，从而实现对该 HPA 定义的 Deployment 对象的定时扩缩容。

HPA 的定义将 Deployment 配置在 scaleTargetRef 字段下，然后 Deployment 通过自身定义查找 ReplicaSet，最后通过 ReplicaSet 调整真实的副本数目。

DCE5.0 将 CronHPA 中的 scaleTargetRef 设置为 HPA 对象，然后通过 HPA 对象来寻找真实的 scaleTargetRef，从而让 CronHPA 感知 HPA 的当前状态。

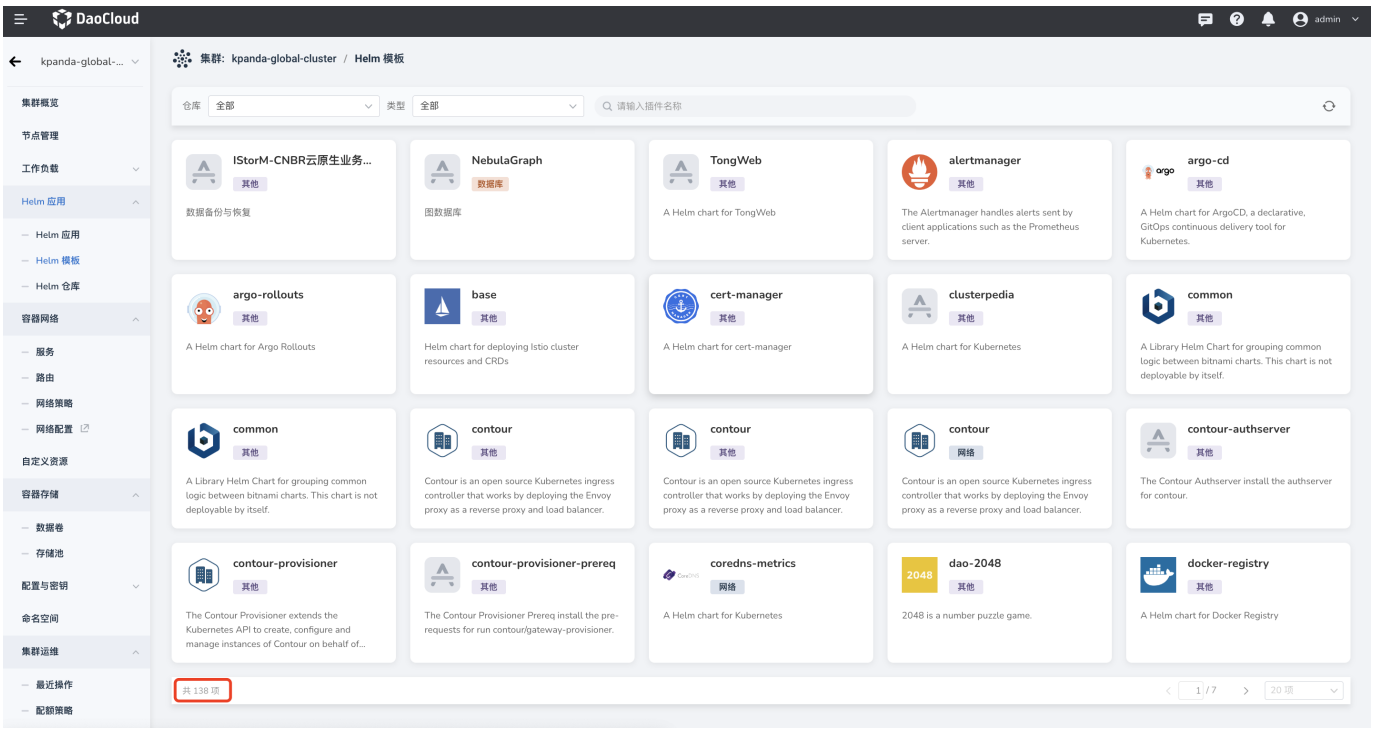


CronHPA 会通过调整 HPA 的方式感知 HPA。CronHPA 通过识别要达到的副本数与当前副本数两者间的较大值，判断是否需要扩缩容及修改 HPA 的上限； CronHPA 通过识别 CronHPA 要达到的副本数与 HPA 的配置间的较小值，判断是否需要修改 HPA 的下限。

Helm 应用

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。 容器管理提供了上百个 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。



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 d.run 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 d.run 中称为 **Helm** 应用 。
- Repository：用于发布和存储 Chart 的存储库。Repository 在 d.run 中称为 **Helm** 仓库 。

更多详细信息，请前往 Helm 官网查看。

相关操作：

- 上传 Helm 模板，介绍上传 Helm 模板操作。
- 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。

上传 HELM 模板

本文介绍如何上传 Helm 模板，操作步骤见下文。

1. 引入 Helm 仓库，操作步骤参考引入第三方 Helm 仓库。
2. 上传 Helm Chart 到 Helm 仓库。

客户端上传 **页面上传**

> **Note**
>
> 此方式适用于 Harbor、ChartMuseum、JFrog 类型仓库。

a. 登录一个可以访问到 Helm 仓库的节点，将 Helm 二进制文件上传到节点，并安装 cm-push 插件。

安装插件流程参考安装 cm-push 插件。

b. 推送 Helm Chart 到 Helm 仓库，执行如下命令；

```
helm cm-push ${charts-dir} ${HELM_REPO_URL} --username ${username} --password ${password}
```

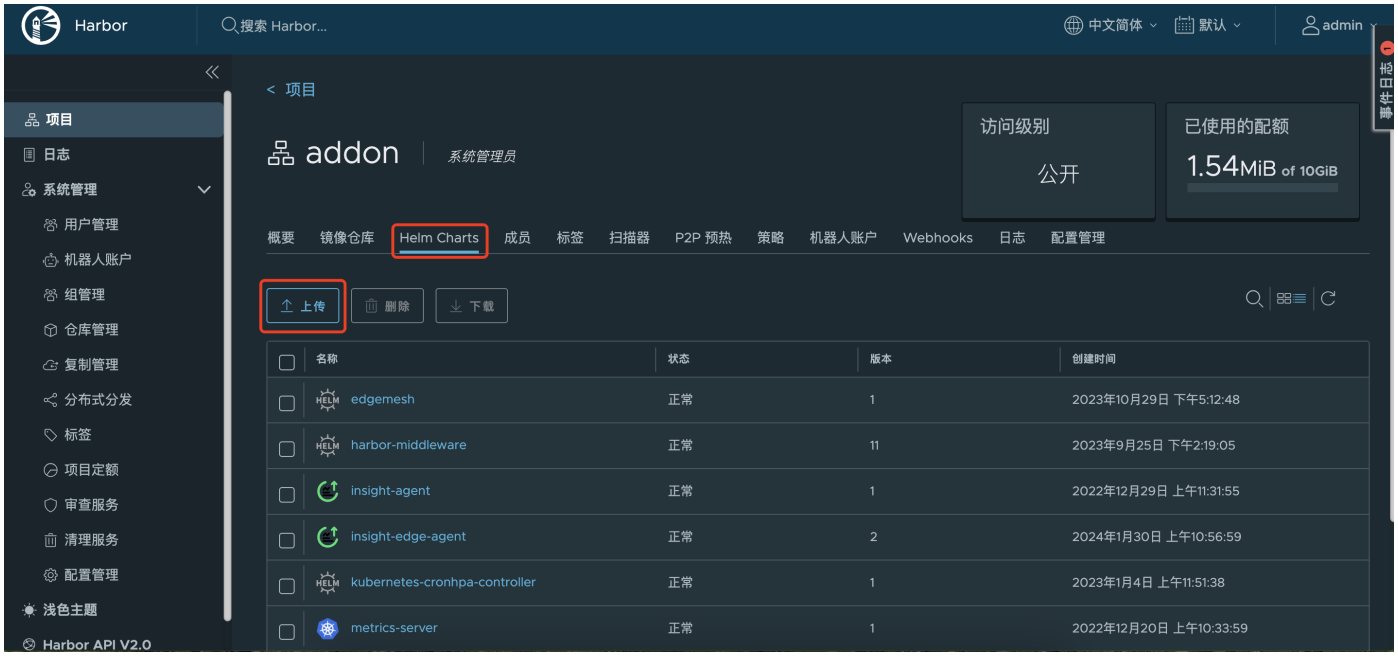
字段说明：

- charts-dir：Helm Chart 的目录，这里也可以直接推送打包好的 Chart（即 .tgz 文件）。
- HELM_REPO_URL：Helm 仓库的 URL。
- username/password：有推送权限的 Helm 仓库用户名和密码。

> **Note**
>
> 此方式仅适用于 Harbor 类型仓库。

a. 登录网页 Harbor 仓库，请确保登录用户有推送权限；

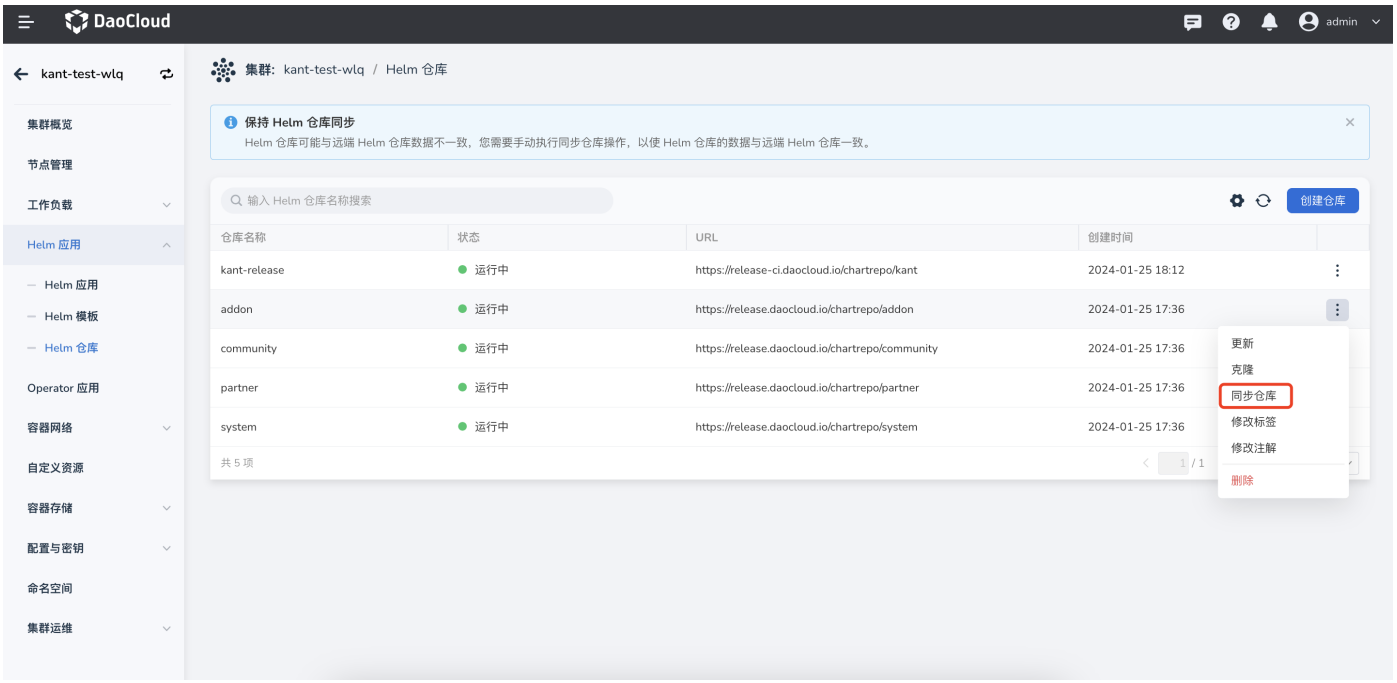
b. 进入到对应项目，选择 **Helm Charts** tab，点击页面 上传 按钮，完成 Helm Chart 上传。



3. 同步远端仓库数据

如果集群设置未开启 **Helm** 仓库自动刷新 设置，需要执行手动同步操作，大致步骤为：

进入 **Helm** 应用 -> **Helm** 仓库 ，点击对应仓库列表右侧操作按钮 同步仓库 ，完成仓库数据同步。


DaoCloud
admin
kant-test-wlq
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
Operator 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: kant-test-wlq / Helm 仓库
保持 Helm 仓库同步
Helm 仓库可能与远端 Helm 仓库数据不一致，您需要手动执行同步仓库操作，以使 Helm 仓库的数据与远端 Helm 仓库一致。
输入 Helm 仓库名称搜索
创建仓库
仓库名称
状态
URL
创建时间
kant-release
运行中
https://release-ci.daocloud.io/chartrepo/kant
2024-01-25 18:12
addon
运行中
https://release.daocloud.io/chartrepo/addon
2024-01-25 17:36
community
运行中
https://release.daocloud.io/chartrepo/community
2024-01-25 17:36
partner
运行中
https://release.daocloud.io/chartrepo/partner
2024-01-25 17:36
system
运行中
https://release.daocloud.io/chartrepo/system
2024-01-25 17:36
共 5 项
1 / 1
更新
克隆
同步仓库
修改标签
修改注解
删除

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。

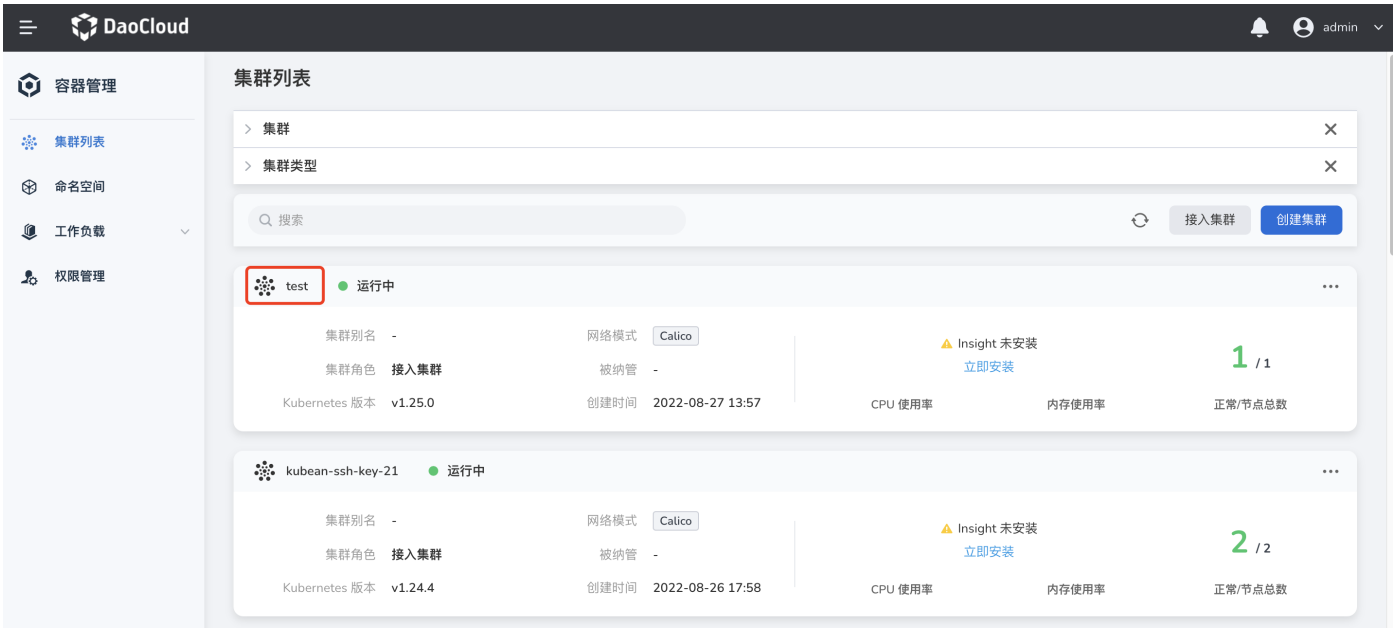
前提条件

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

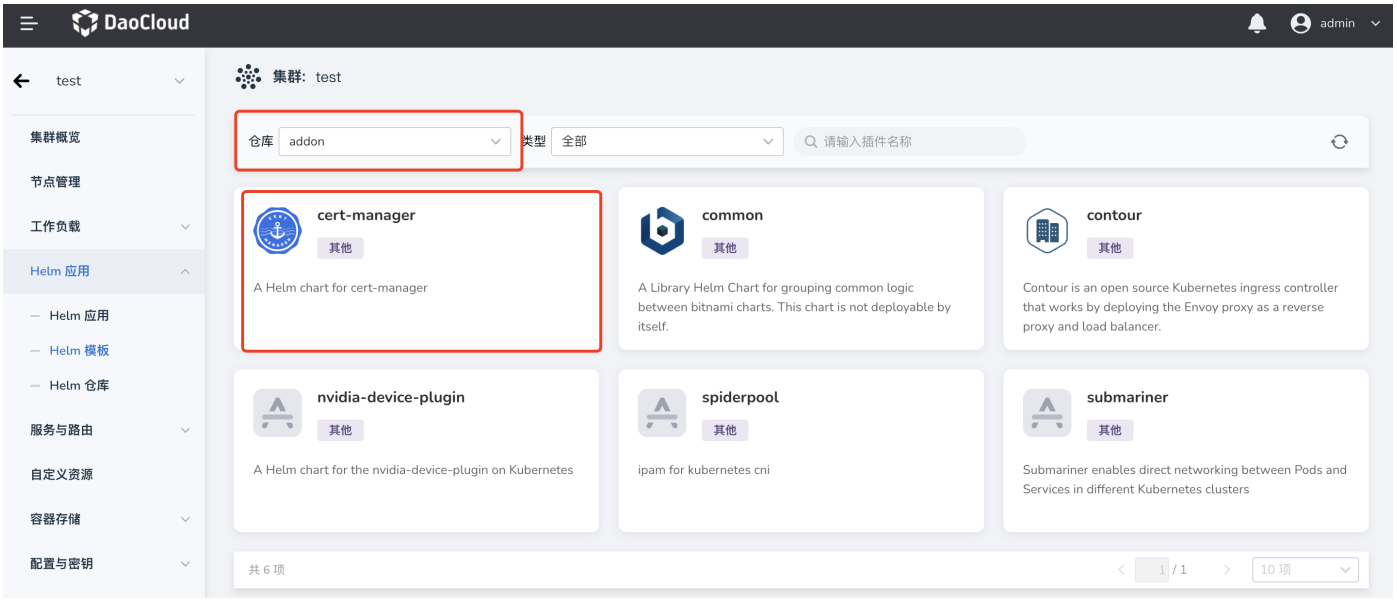
参照以下步骤安装 Helm 应用。

1. 点击一个集群名称，进入 集群详情 。

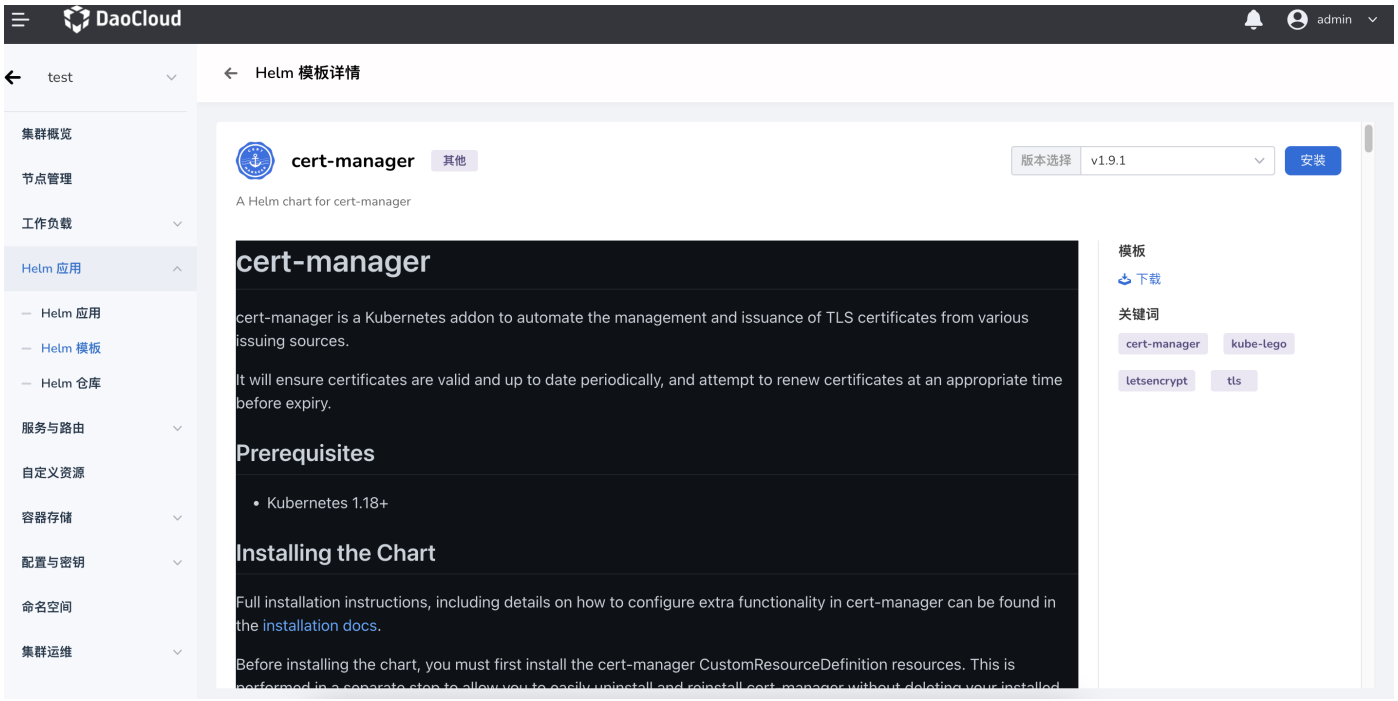


2. 在左侧导航栏，依次点击 **Helm** 应用 -> **Helm** 模板 ，进入 Helm 模板页面。

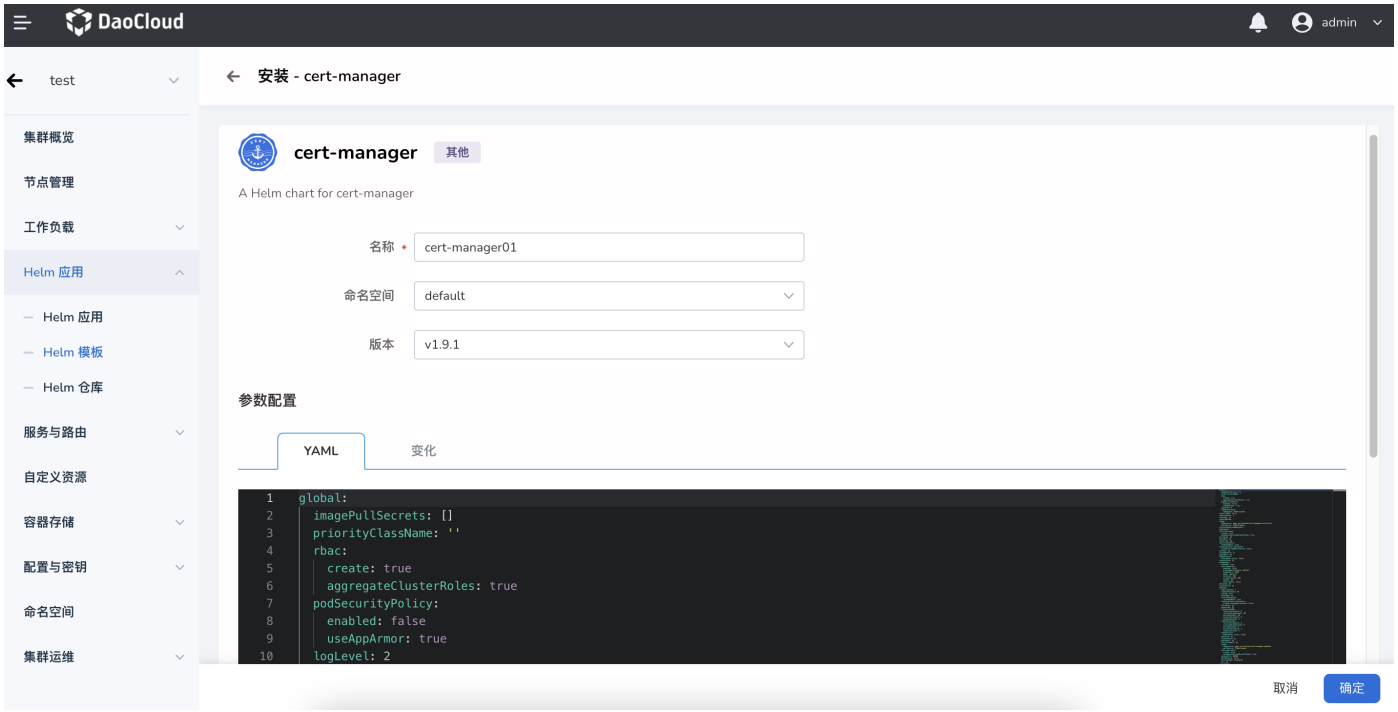
   在 Helm 模板页面选择名为 **addon** 的 Helm 仓库，此时界面上将呈现 **addon** 仓库下所有的 Helm chart 模板。 点击名称为 **cert-manager** 的 Chart。



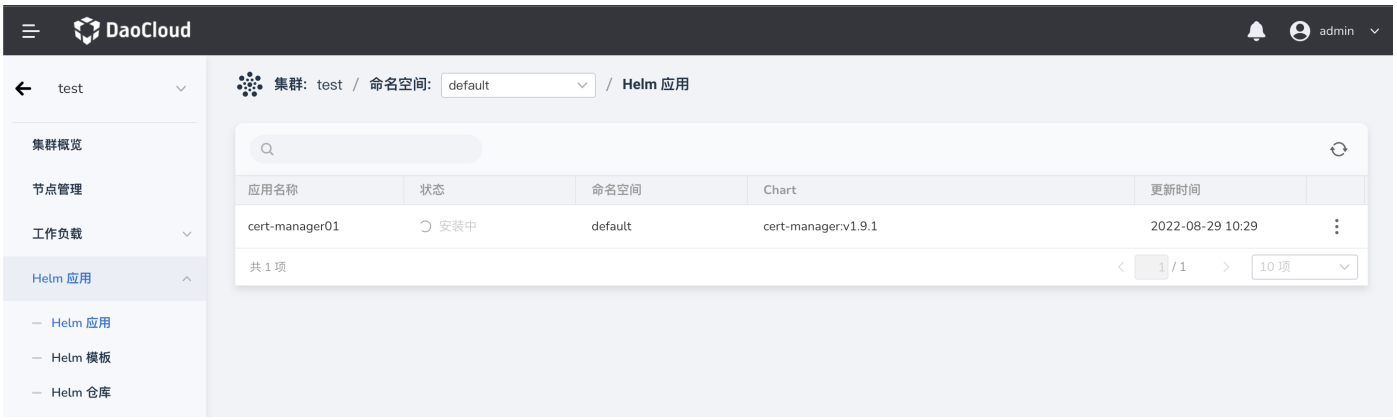
3. 在安装页面，能够看到 Chart 的相关详细信息，在界面右上角选择需要安装的版本，点击 安装 按钮。此处选择 v1.9.1 版本进行安装。



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 。



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 。

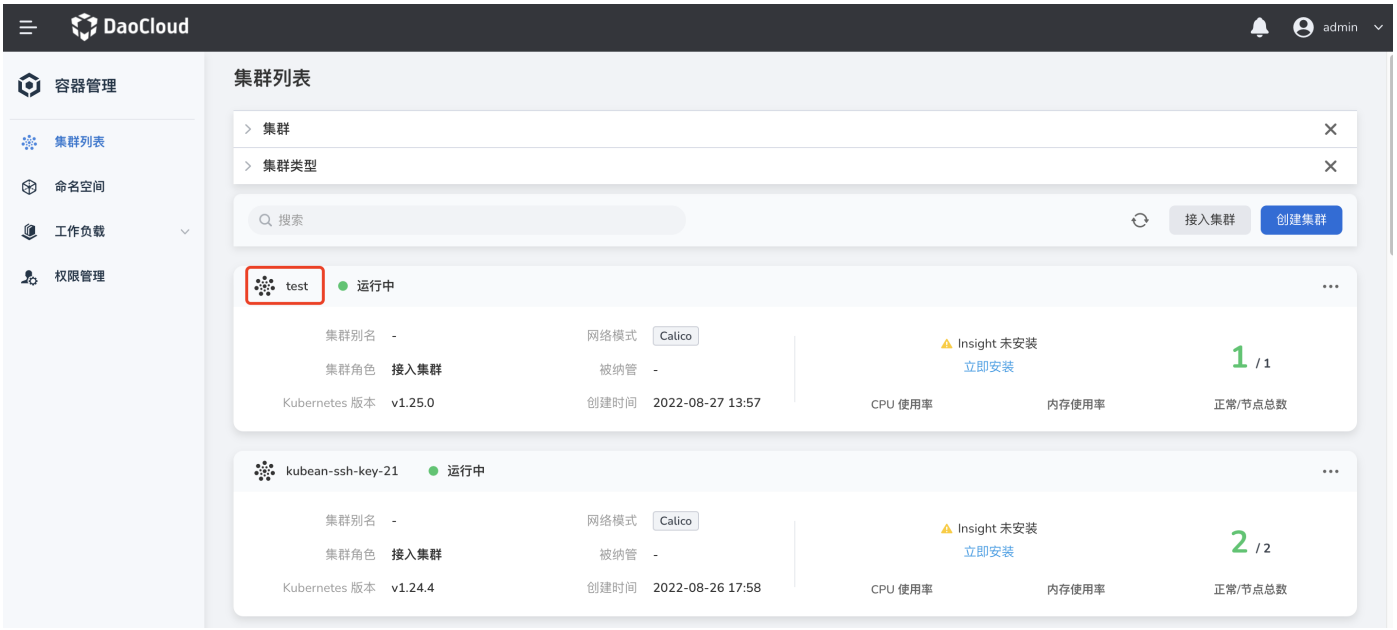


更新 Helm 应用

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。

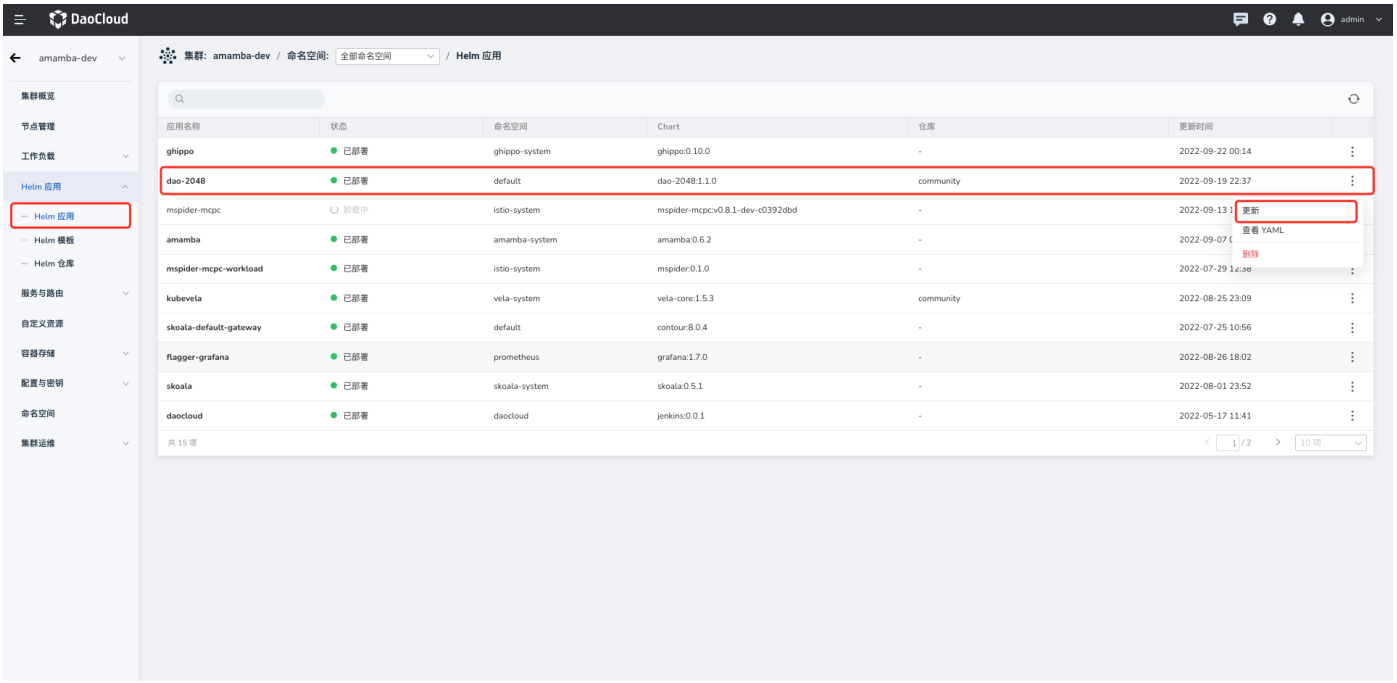
参照以下步骤更新 Helm 应用。

1. 点击一个集群名称，进入 集群详情 。

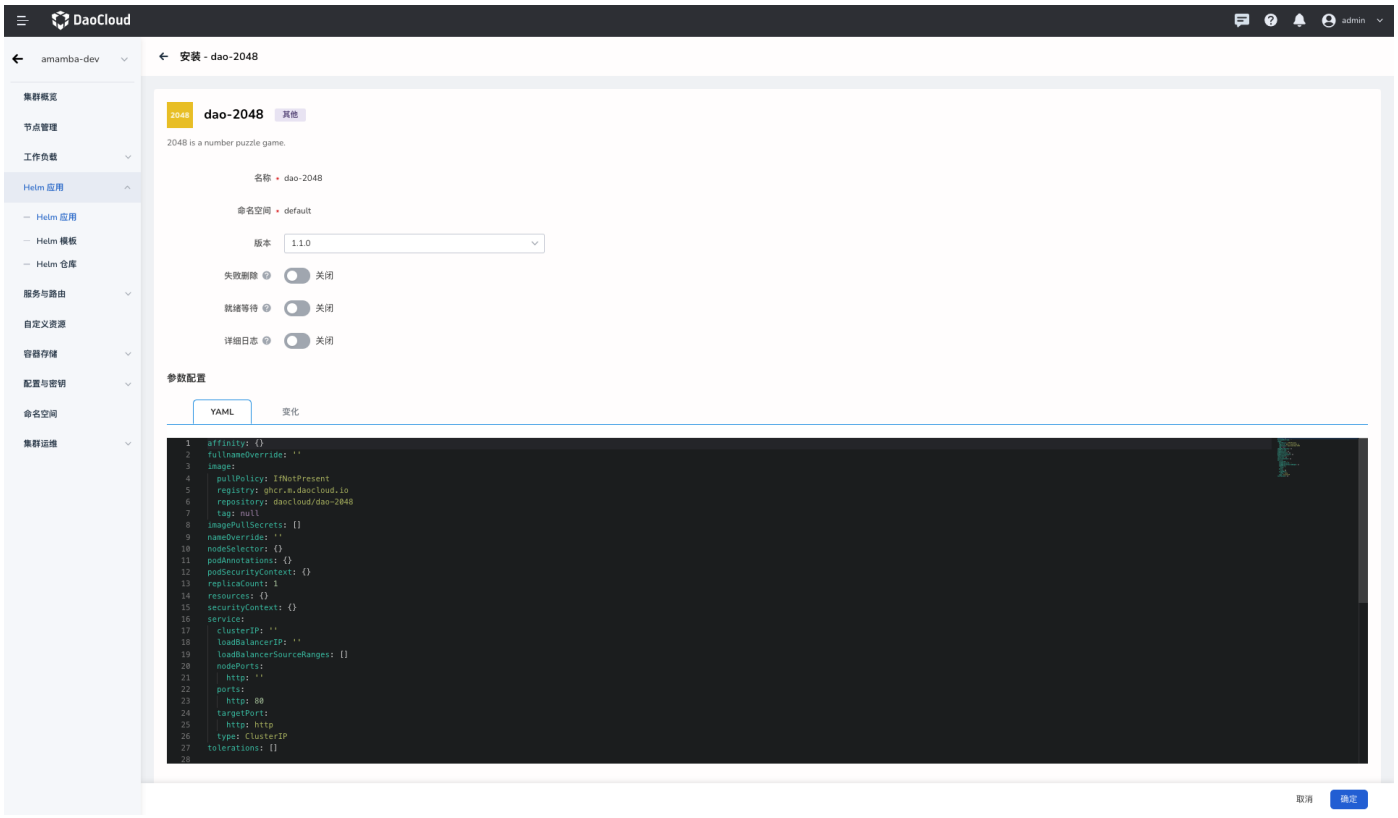


2. 在左侧导航栏，点击 **Helm** 应用 ，进入 Helm 应用列表页面。

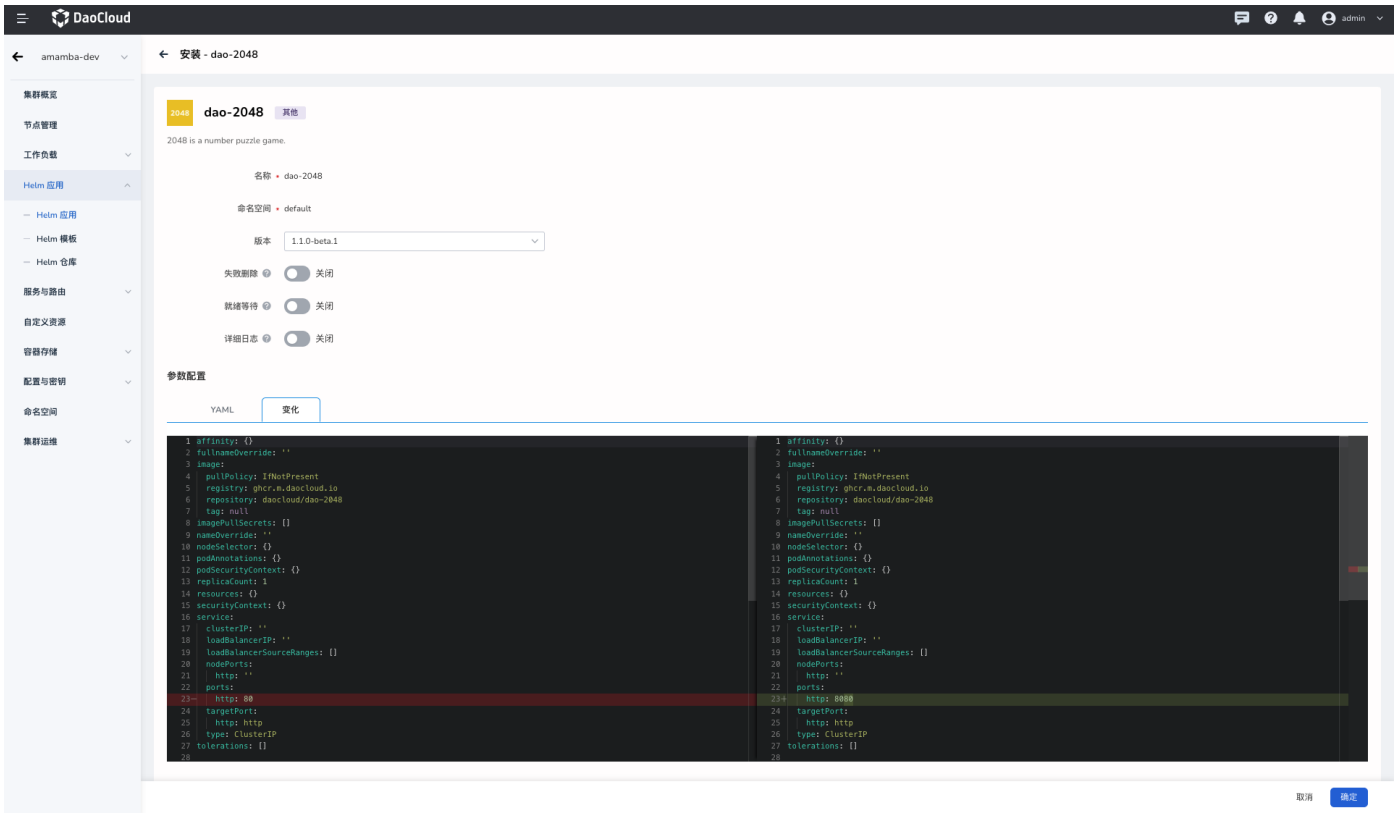
   在 Helm 应用列表页选择需要更新的 Helm 应用，点击列表右侧的 ┆ 操作按钮，在下拉选择中选择 更新 操作。



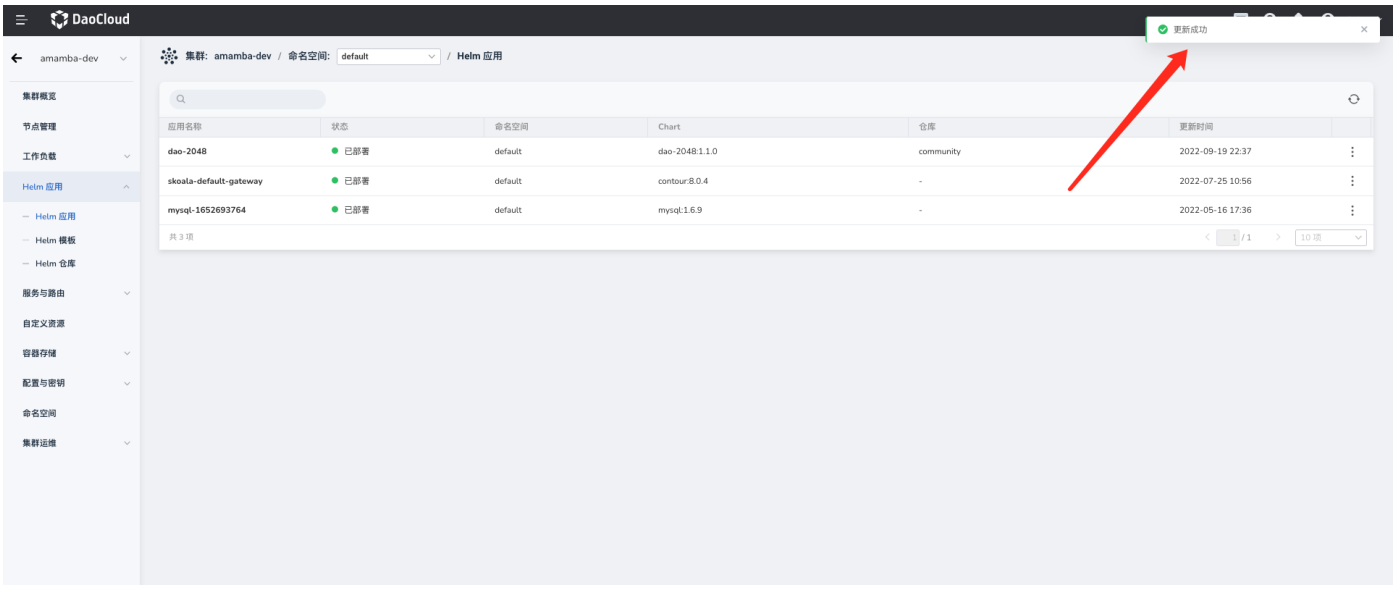
3. 点击 更新 按钮后，系统将跳转至更新界面，您可以根据需要对 Helm 应用进行更新，此处我们以更新 **dao-2048** 这个应用的 http 端口为例。



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。



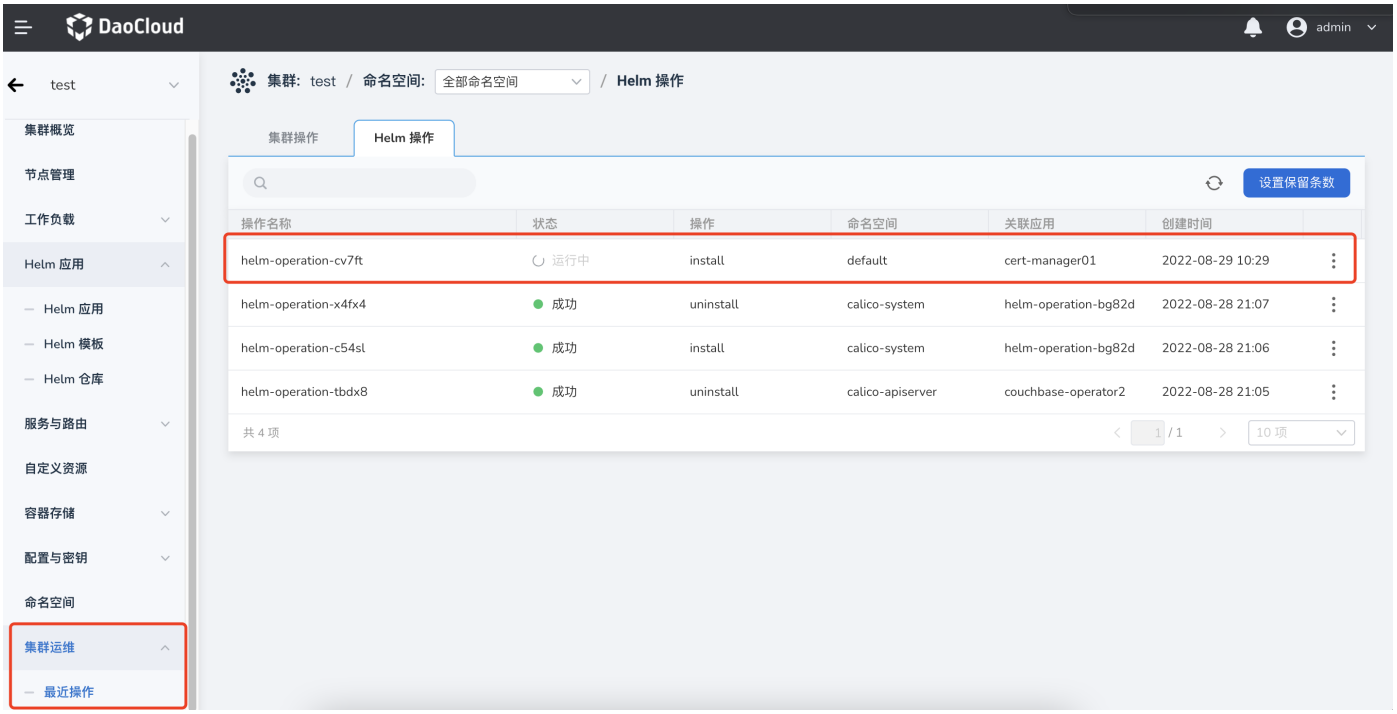
5. 系统将自动返回 Helm 应用列表，右上角弹窗提示 更新成功 。


DaoCloud
更新成功
amamba-dev
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
服务与路由
自定义资源
容器存储
配置与密钥
命名空间
集群运维
集群: amamba-dev / 命名空间: default / Helm 应用
应用名称 状态 命名空间 Chart 仓库 更新时间
dao-2048 已部署 default dao-2048:1.1.0 community 2022-09-19 22:37
skoala-default-gateway 已部署 default contour:8.0.4 - 2022-07-25 10:56
mysql-1652693764 已部署 default mysql:1.6.9 - 2022-05-16 17:36
共 3 项
1 / 1
10 项

查看 Helm 操作记录

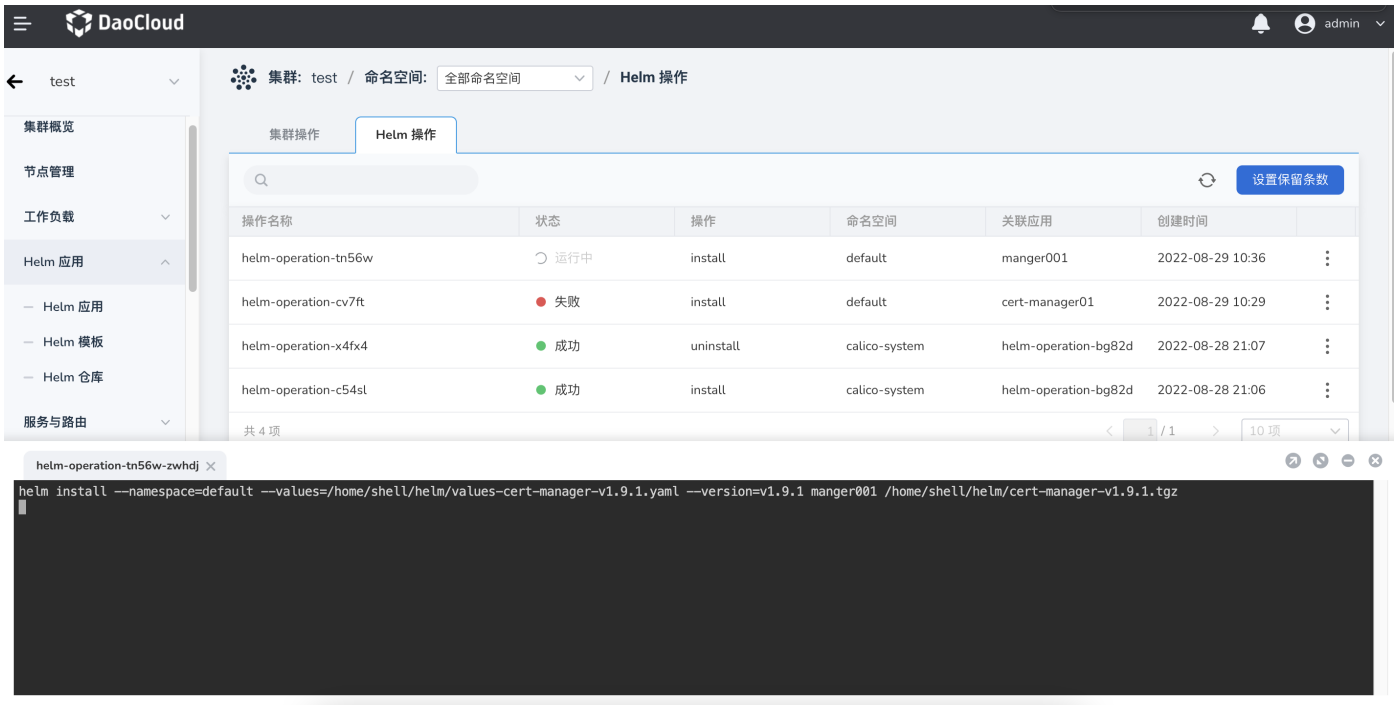
Helm 应用的每次安装、更新、删除都有详细的操作记录和日志可供查看。

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。



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 。

3. 此时页面下方将以控制台的形式展示详细的运行日志。

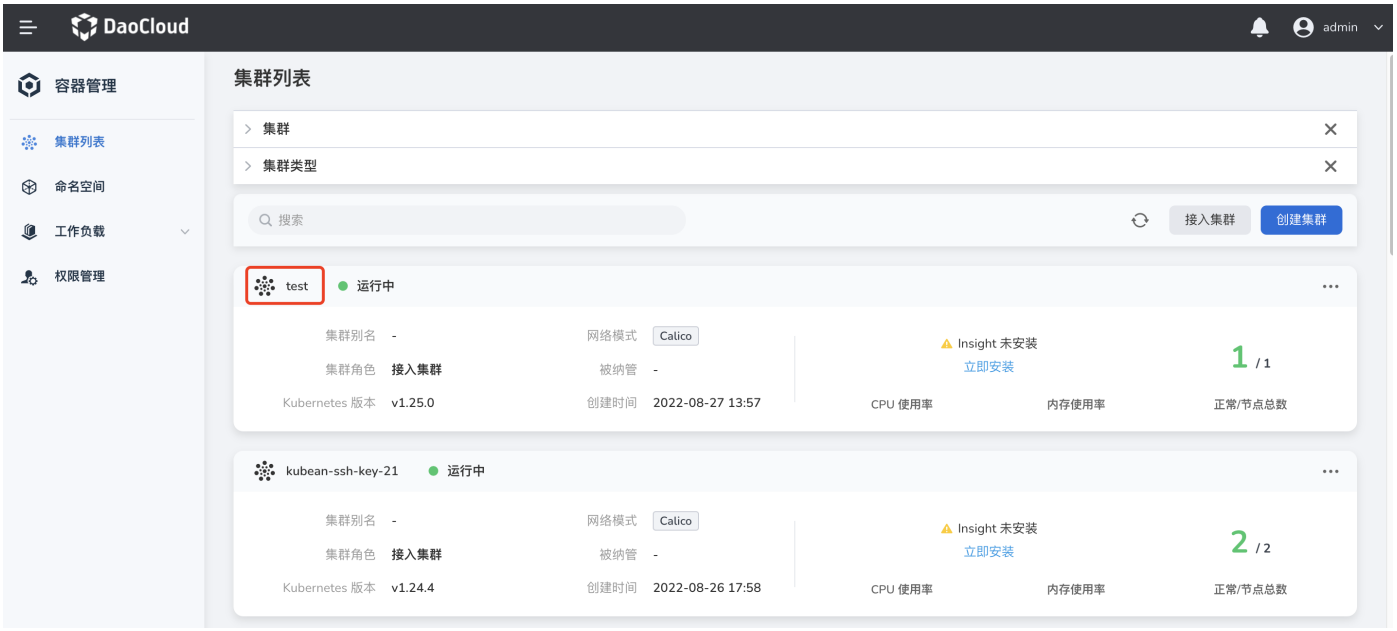

DaoCloud
admin
test
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
服务与路由
集群: test / 命名空间: 全部命名空间 / Helm 操作
集群操作
Helm 操作
设置保留条数
操作名称 状态 操作 命名空间 关联应用 创建时间
helm-operation-tn56w 运行中 install default manger001 2022-08-29 10:36
helm-operation-cv7ft 失败 install default cert-manager01 2022-08-29 10:29
helm-operation-x4fx4 成功 uninstall calico-system helm-operation-bg82d 2022-08-28 21:07
helm-operation-c54sl 成功 install calico-system helm-operation-bg82d 2022-08-28 21:06
共 4 项
1 / 1
10 项

helm-operation-tn56w-zwhdj
helm install --namespace=default --values=/home/shell/helm/values-cert-manager-v1.9.1.yaml --version=v1.9.1 manger001 /home/shell/helm/cert-manager-v1.9.1.tgz

删除 Helm 应用

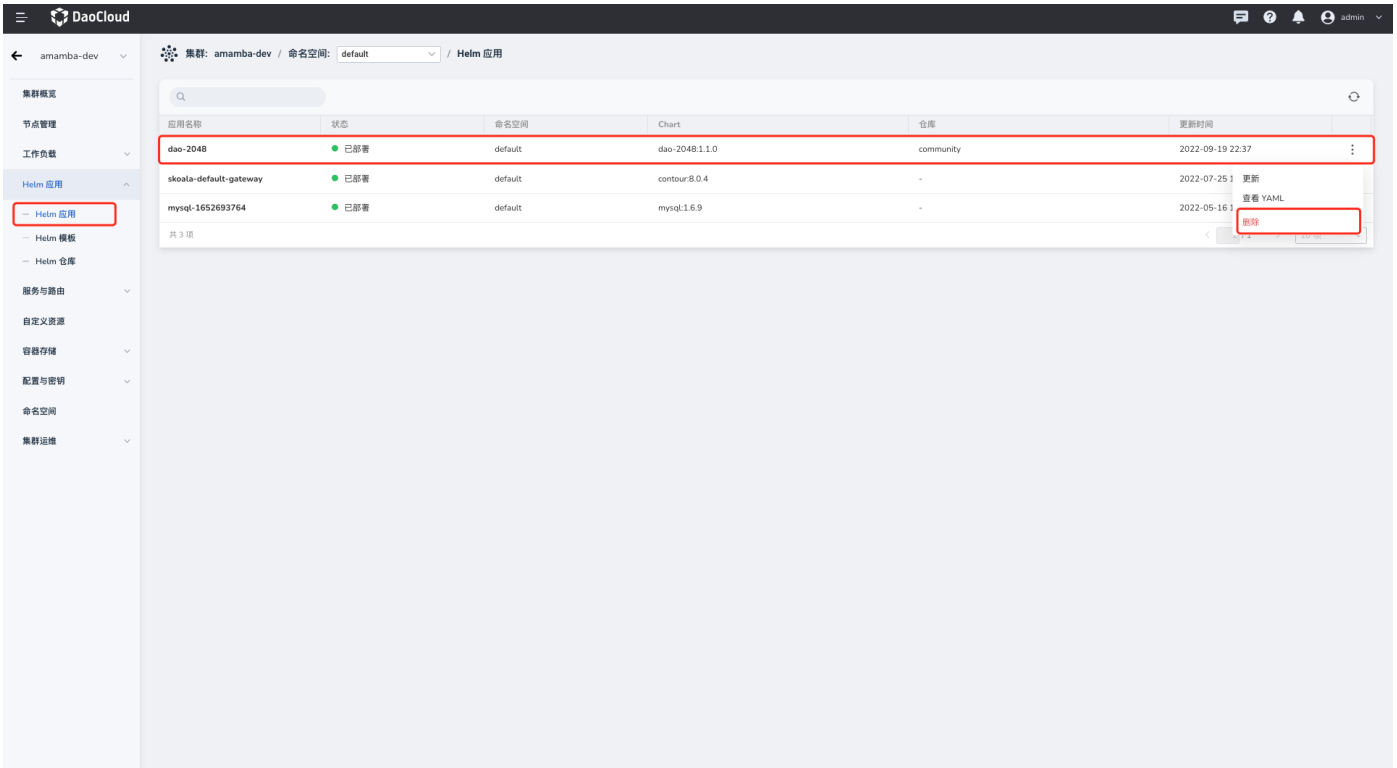
参照以下步骤删除 Helm 应用。

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 。

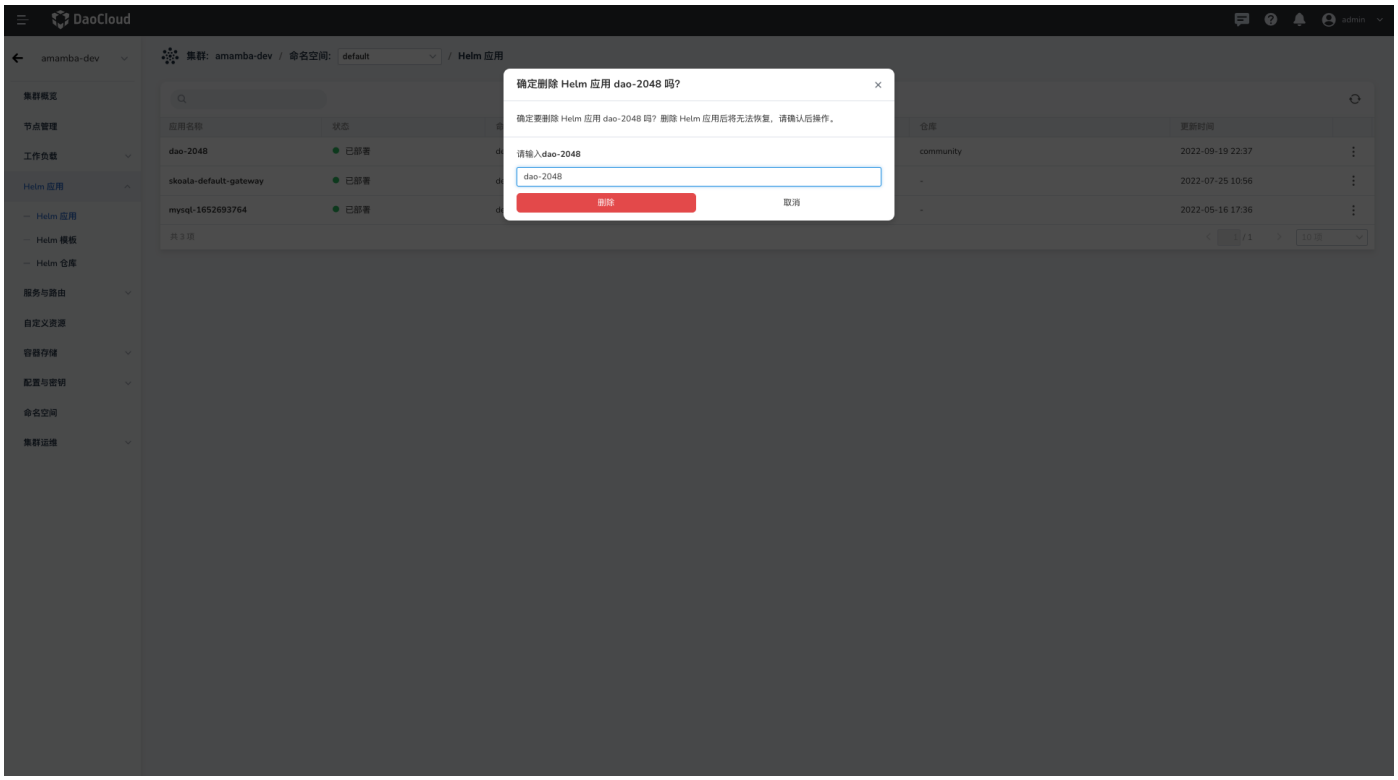


2. 在左侧导航栏，点击 **Helm** 应用 ，进入 Helm 应用列表页面。

   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 。



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。


DaoCloud
admin
amamba-dev
集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
服务与路由
自定义资源
容器存储
配置与密钥
命名空间
集群运维
集群: amamba-dev / 命名空间: default / Helm 应用
应用名称
状态
仓库
更新时间
dao-2048
已部署
community
2022-09-19 22:37
skoala-default-gateway
已部署
2022-07-25 10:56
mysql-1652693764
已部署
2022-05-16 17:36
共 3 项
10 项
确定删除 Helm 应用 dao-2048 吗?
确定要删除 Helm 应用 dao-2048 吗? 删除 Helm 应用后将无法恢复，请确认后操作。
请输入dao-2048
dao-2048
删除
取消

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。

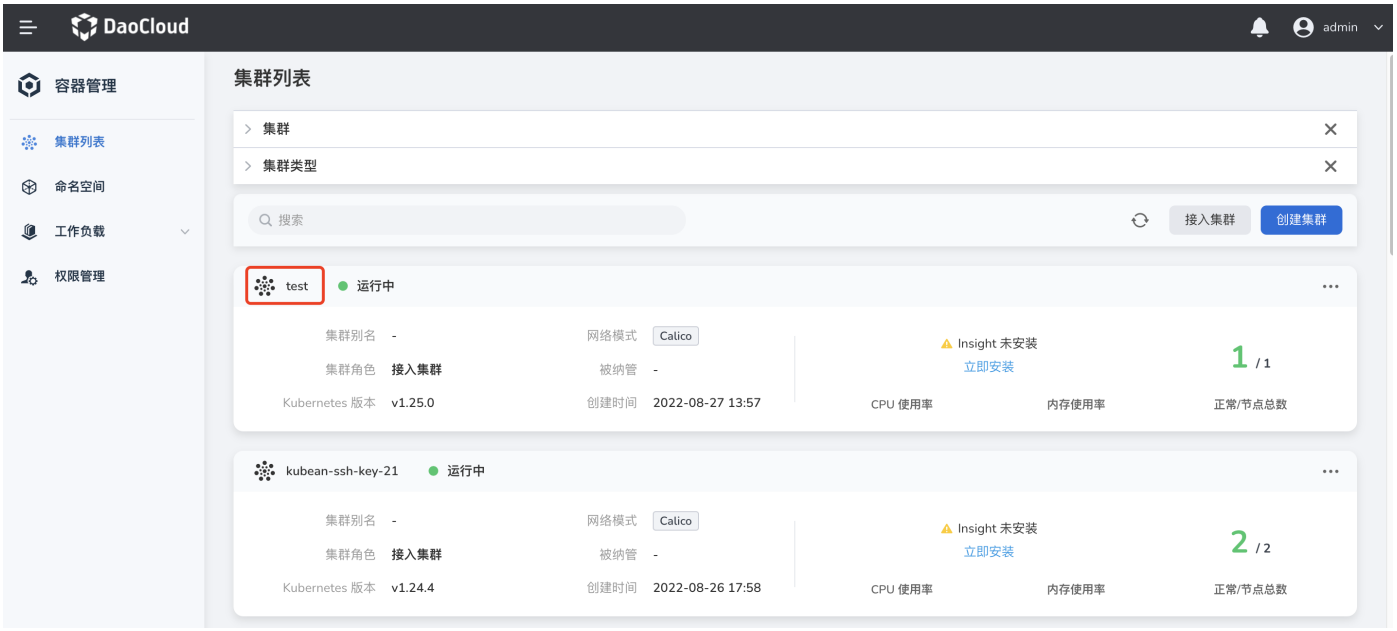
前提条件

- 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。

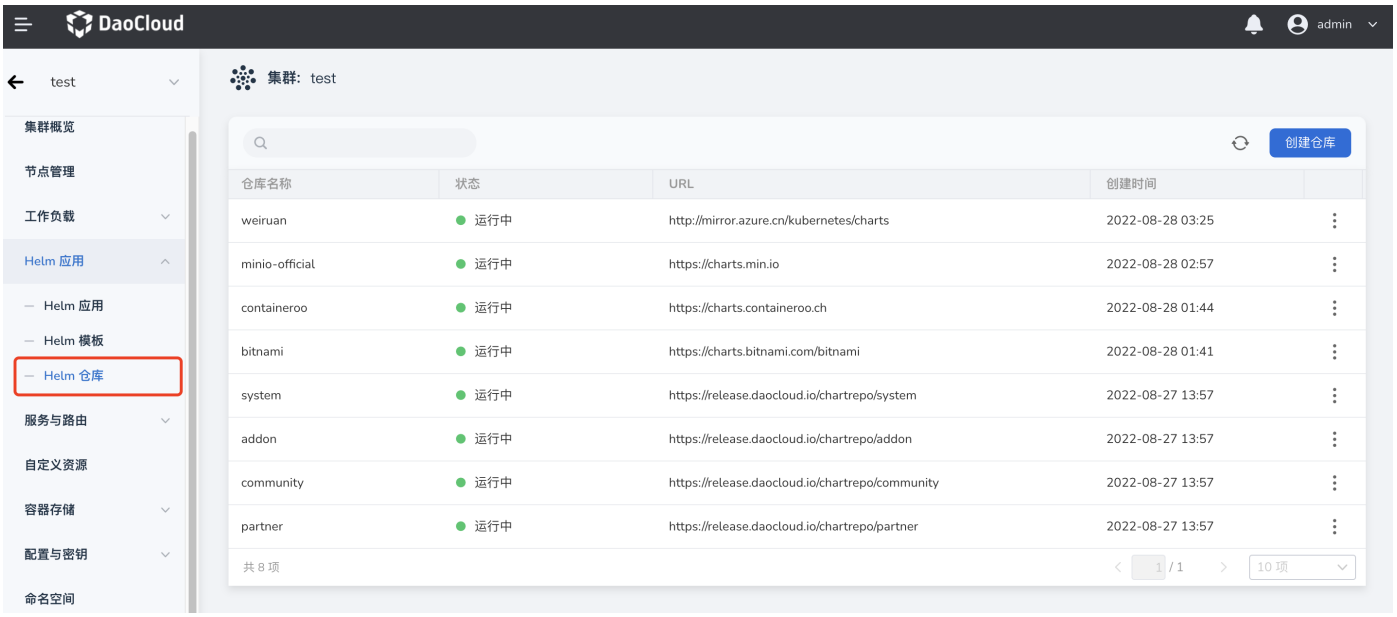
引入第三方 Helm 仓库

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 。

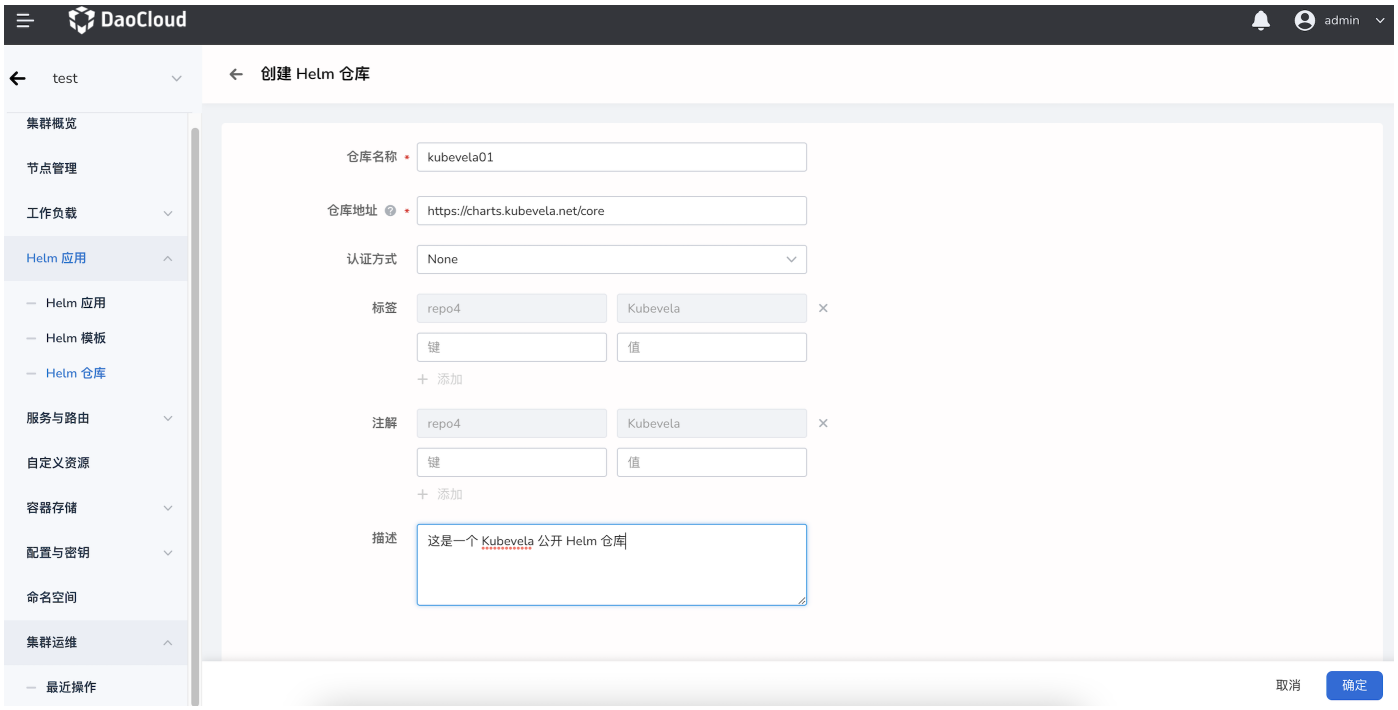


2. 在左侧导航栏，依次点击 **Helm** 应用 -> **Helm** 仓库 ，进入 Helm 仓库页面。

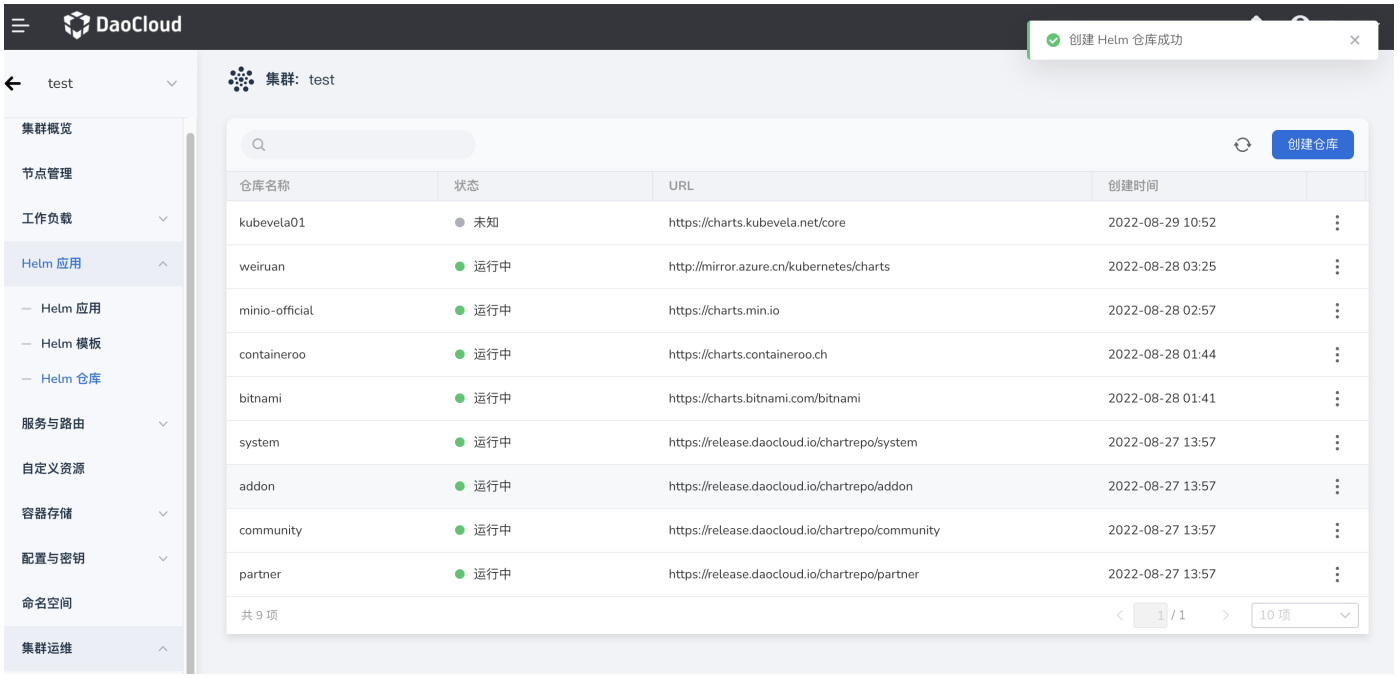


3. 在 Helm 仓库页面点击 创建仓库 按钮，进入创建仓库页面，按照下表配置相关参数。

- 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



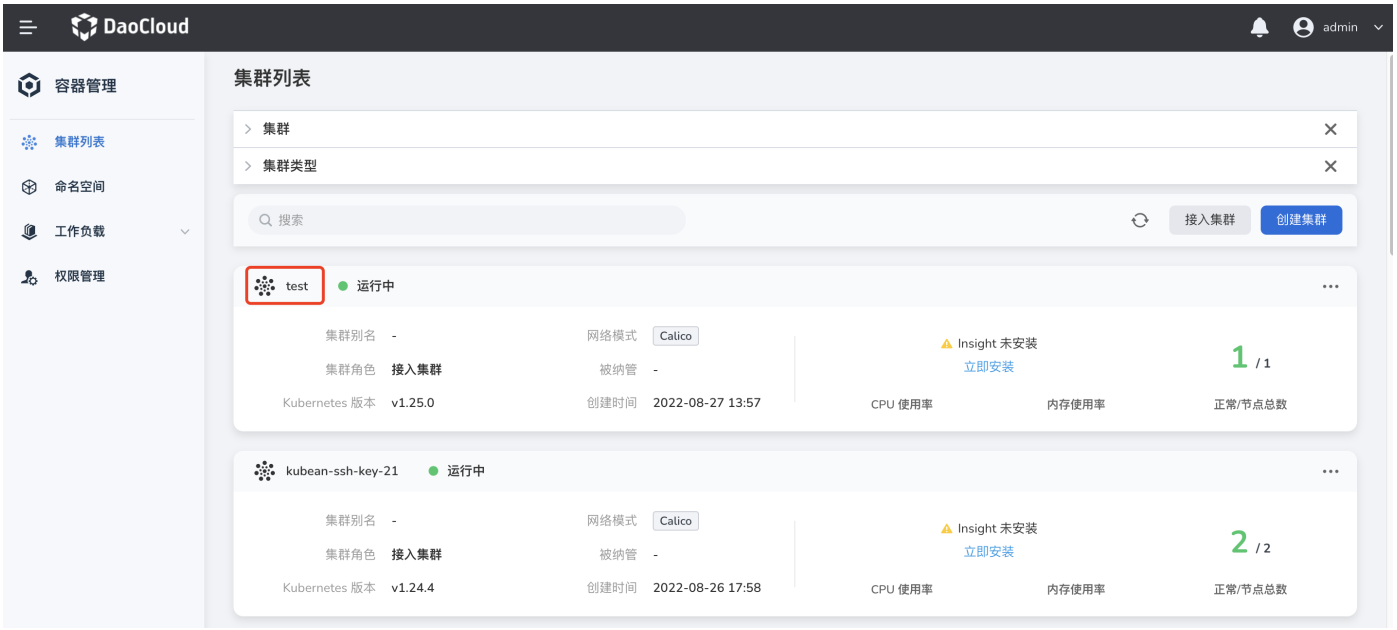
4. 点击 确定 ，完成 Helm 仓库的创建。页面会自动跳转至 Helm 仓库列表。



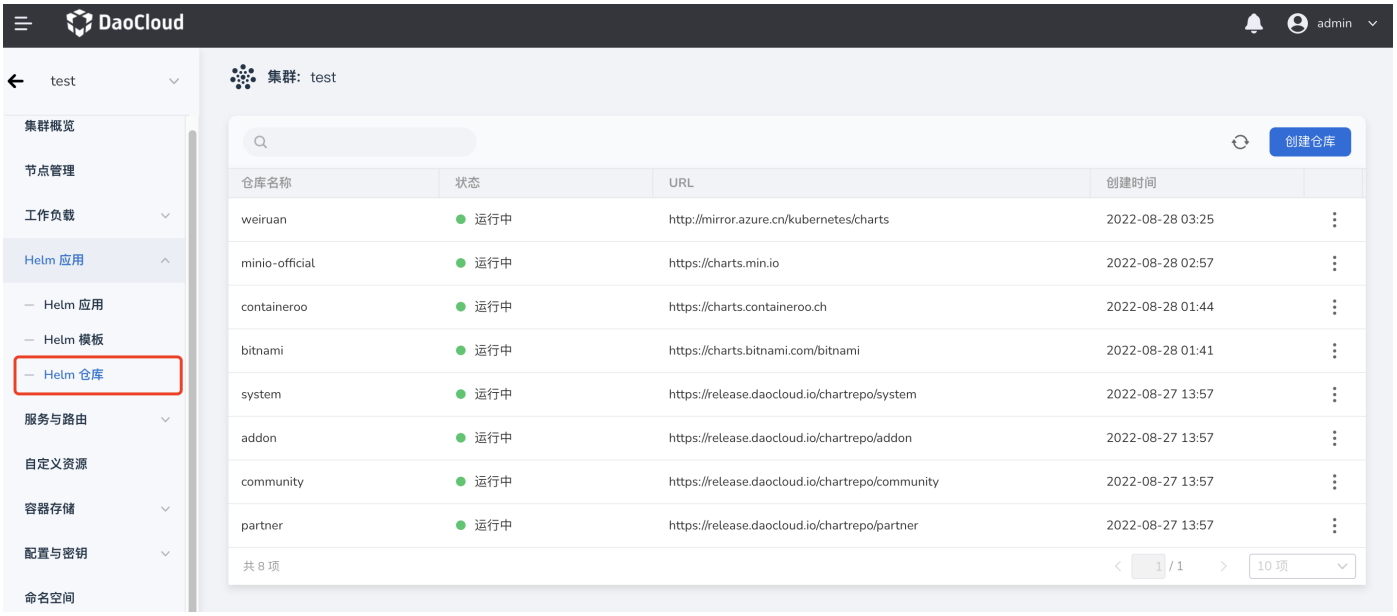
更新 Helm 仓库

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。

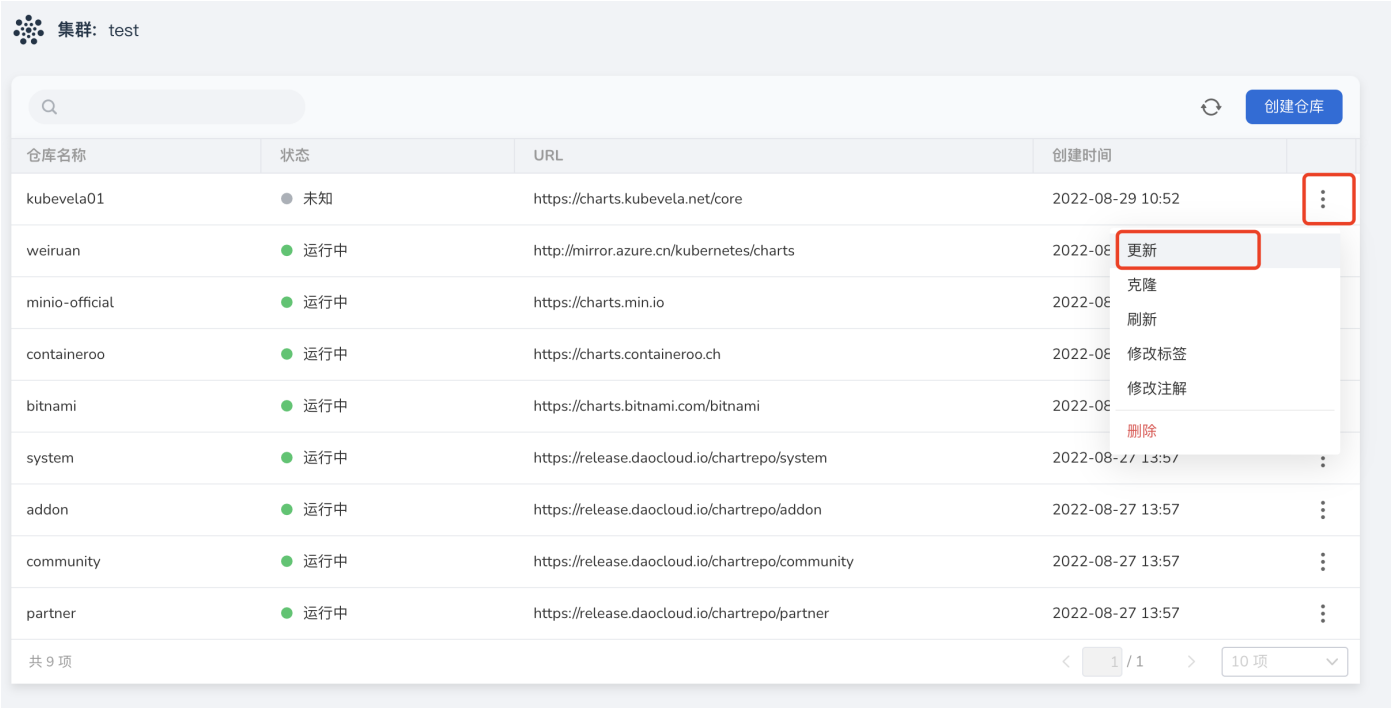
1. 找到待更新仓库所在的集群，点击集群名称，进入 集群详情 。



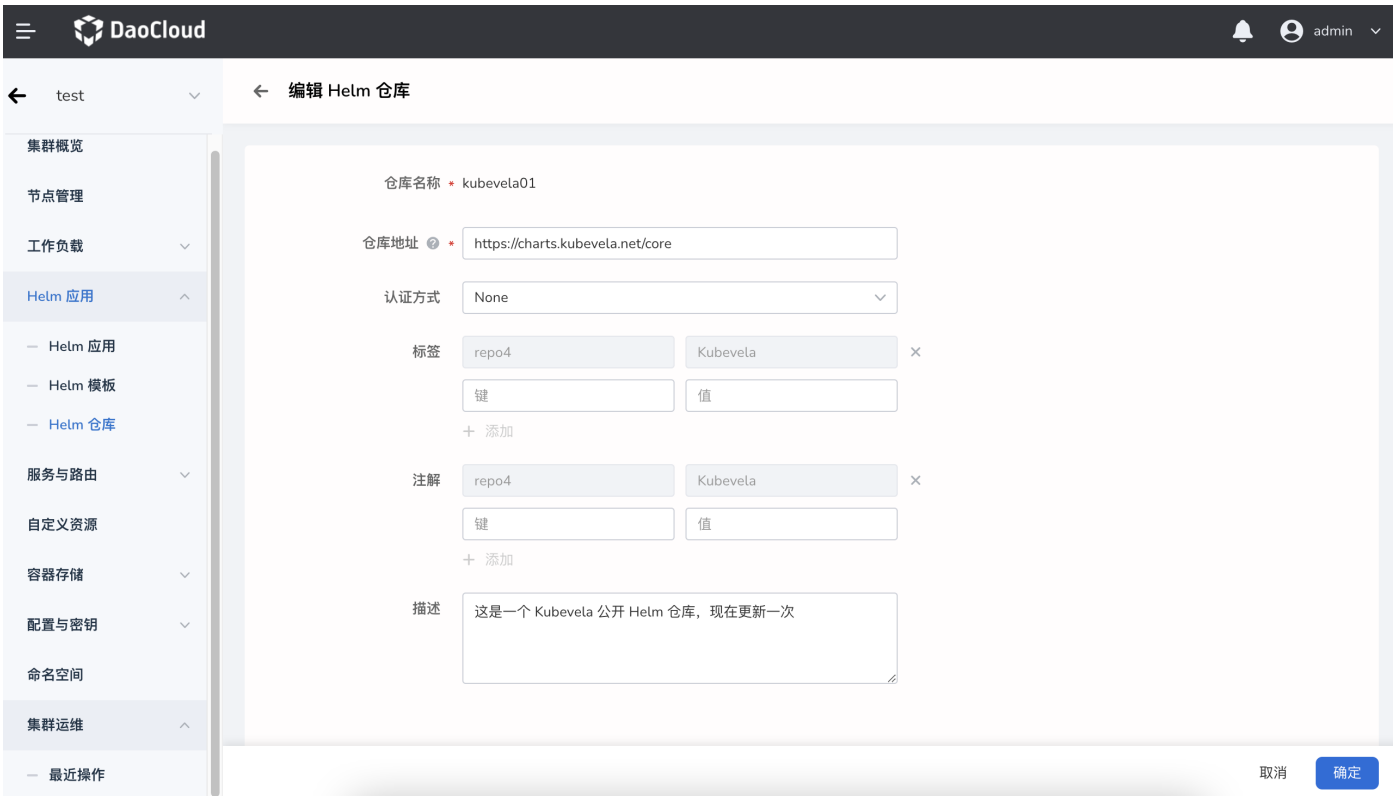
2. 在左侧导航栏，依次点击 **Helm** 应用 -> **Helm** 仓库 ，进入 Helm 仓库列表页面。



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 。



4. 在 编辑 **Helm** 仓库 页面进行更新，完成后点击 确定 。

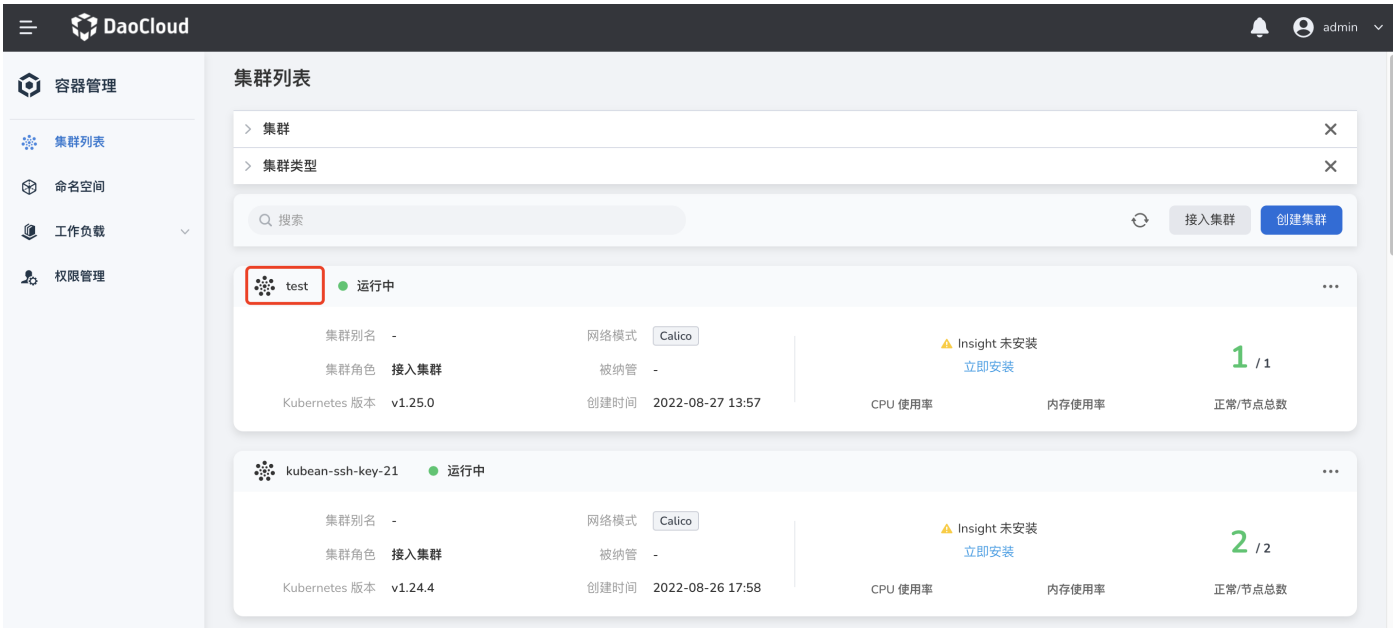


5. 返回 Helm 仓库列表，屏幕提示更新成功。

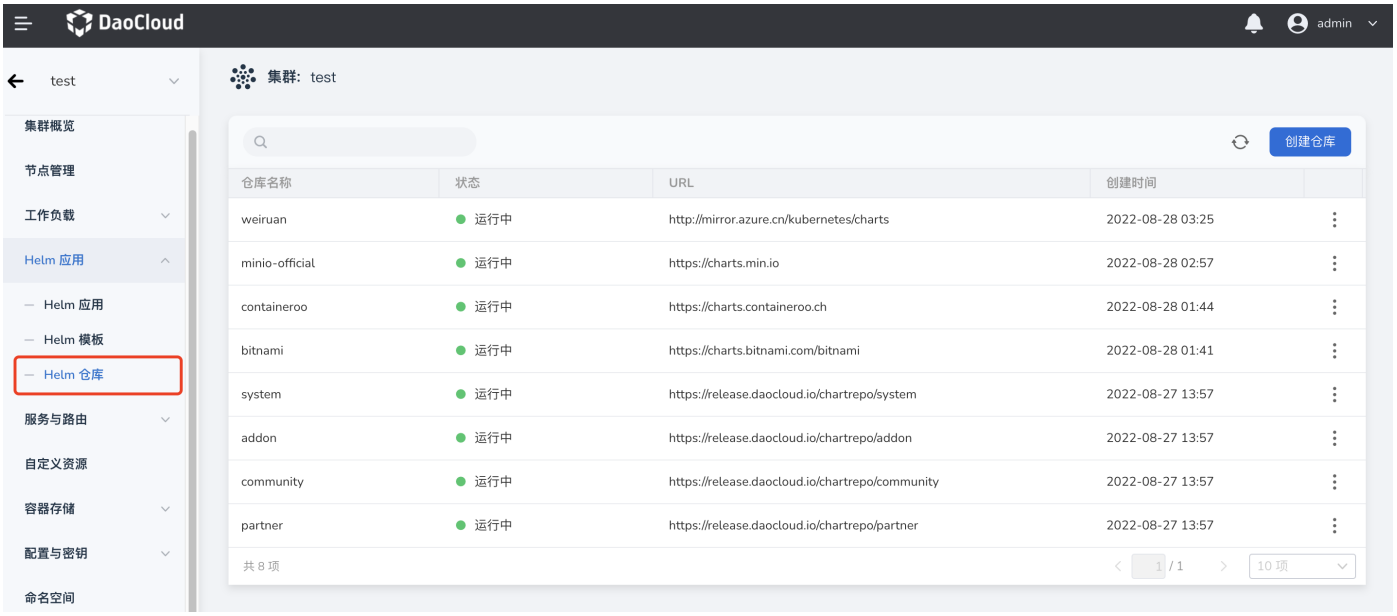
删除 Helm 仓库

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。

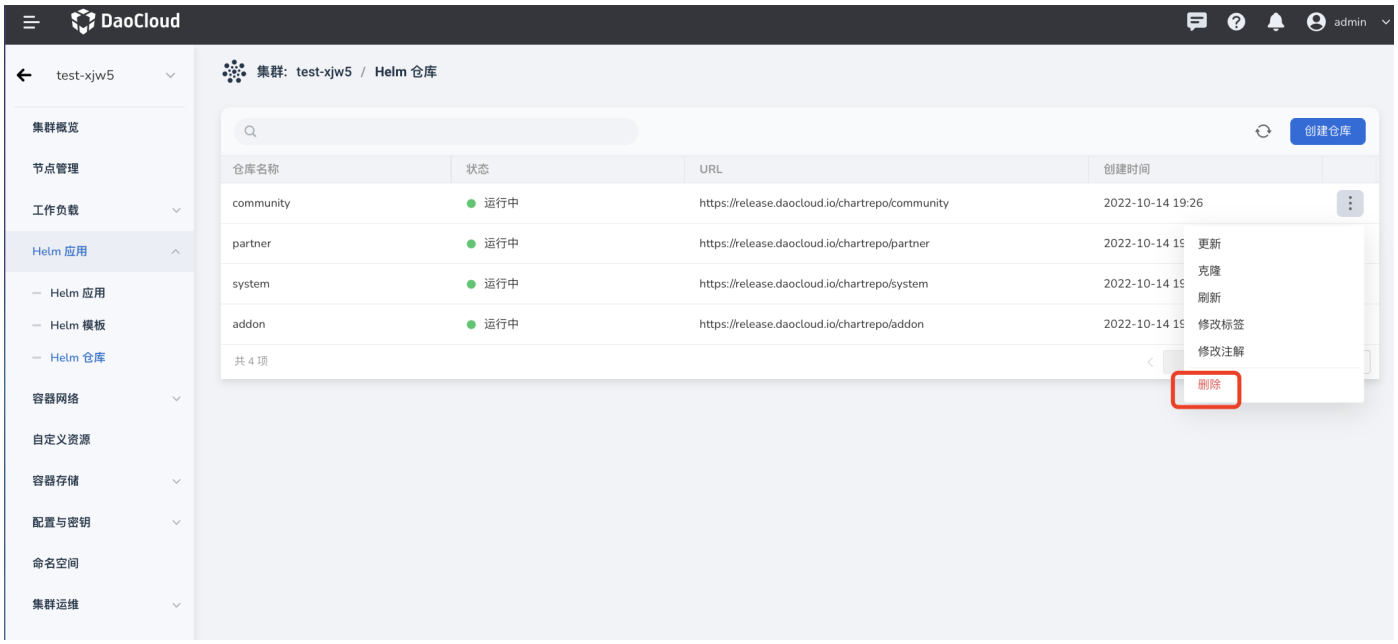
1. 找到待删除仓库所在的集群，点击集群名称，进入 集群详情 。



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。



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 。



4. 输入仓库名称进行确认，点击 删除 。

5. 返回 Helm 仓库列表，屏幕提示删除成功。

HELM 应用多架构和升级导入步骤

通常在多架构集群中，也会使用多架构的 Helm 包来部署应用，以解决架构差异带来的部署问题。 本文将介绍如何将单架构 Helm 应用融合为多架构，以及多架构与多架构 Helm 应用的相互融合。

导入

单架构导入

准备好待导入的离线包 `addon-offline-full-package-${version}-${arch}.tar.gz` ，可从 下载中心下载。 把路径填写至 **clusterConfig.yml** 配置文件，例如：

```
addonPackage:
  path: "/home/addon-offline-full-package-v0.9.0-amd64.tar.gz"
```

然后执行导入命令：

```
~/dce5-installer cluster-create -c /home/dce5/sample/clusterConfig.yaml -m /home/dce5/sample/manifest.yaml -d -j13
```

多架构融合

准备好待融合的离线包 `addon-offline-full-package-${version}-${arch}.tar.gz` ，可从 下载中心下载。

以 addon-offline-full-package-v0.9.0-arm64.tar.gz 为例，执行导入命令：

```
~/dce5-installer import-addon -c /home/dce5/sample/clusterConfig.yaml --addon-path=/home/addon-offline-full-package-v0.9.0-arm64.tar.gz
```

升级

单架构升级

准备好待导入的离线包 `addon-offline-full-package-${version}-${arch}.tar.gz` ，可从下载中心下载。

把路径填写至 clusterConfig.yml 配置文件，例如：

```
addonPackage:
  path: "/home/addon-offline-full-package-v0.11.0-amd64.tar.gz"
```

然后执行导入命令：

```
~/dce5-installer cluster-create -c /home/dce5/sample/clusterConfig.yaml -m /home/dce5/sample/manifest.yaml -d -j13
```

多架构融合

准备好待融合的离线包 `addon-offline-full-package-${version}-${arch}.tar.gz` ，可从下载中心下载。

以 addon-offline-full-package-v0.11.0-arm64.tar.gz 为例，执行导入命令：

```
~/dce5-installer import-addon -c /home/dce5/sample/clusterConfig.yaml --addon-path=/home/addon-offline-full-package-v0.11.0-arm64.tar.gz
```

注意事项

磁盘空间

离线包比较大，且过程中需要解压和 load 镜像，需要预留充足的空间，否则可能在过程中报 “no space left” 而中断。

失败后重试

如果在多架构融合步骤执行失败，重试前需要清理一下残留：

```
rm -rf addon-offline-target-package
```

镜像空间

如果融合的离线包中包含了与导入的离线包不一致的镜像空间，可能会在融合过程中因为镜像空间不存在而报错：

```
11:02PM: 【INFO】  import image from localhost/fluxcd/image-automation-controller:v0.14.0 to 10.5.14.100/localhost/fluxcd/image-automation-controller:v0.14.0-arm64
11:02PM: + podman push --tls-verify=false 10.5.14.100/localhost/fluxcd/image-automation-controller:v0.14.0-arm64
11:02PM: Getting image source signatures
11:02PM: Copying blob sha256:d35ce67929544871d4a3c69a5ac9ba46cf85ba43d06c55f03ed53df7fce8ffc5
11:02PM: Copying blob sha256:8deca1efb6b6188c28ca2b290ed564a423eae4d2acc703ee5a6ac3159e92e46f
11:02PM: Copying blob sha256:eafeb89251f15b6843277898a3b88ea6381a0efbceca744c29c5e22be3921a74
11:02PM: Copying blob sha256:03c9884b9f6a3a1fd7f5241dc33d9b6ed3ee3bdfd9fe1761fa6961e86f11ebdf
11:02PM: Copying blob sha256:a2192106d0ba7e89b5b66b3899b3b9063a5d4b0f9fc989307f39a086efe7f828
11:02PM: Error: trying to reuse blob sha256:a2192106d0ba7e89b5b66b3899b3b9063a5d4b0f9fc989307f39a086efe7f828 at destination: checking whether a blob
sha256:a2192106d0ba7e89b5b66b3899b3b9063a5d4b0f9fc989307f39a086efe7f828 exists in 10.5.14.100/localhost/fluxcd/image-automation-controller: authentication
required
11:02PM: + cleanup
11:02PM: + echo cleanup
11:02PM: + set +ex
11:02PM: cleanup
11:02PM: 【sched-info】 Sched shutdown...
11:02PM: 【DEBUG】 kind kubeconfig file locates in /var/lib/dce5/kind-my-cluster-installer.kube-conf
11:02PM: Generating the kube.conf for my-cluster ...
11:02PM: Inspect Cluster with new Context...(KUBECONFIG=/tmp/tmp.QGAXw5MLSH/my-cluster-kube.conf)
11:02PM: ----------------------
11:02PM: NAME           STATUS   ROLES           AGE     VERSION
11:02PM: prod-master1   Ready    control-plane   5h50m   v1.27.5
11:02PM: prod-master2   Ready    control-plane   5h49m   v1.27.5
11:02PM: prod-master3   Ready    control-plane   5h48m   v1.27.5
11:02PM: prod-worker1   Ready    <none>          5h46m   v1.27.5
11:02PM: prod-worker2   Ready    <none>          5h46m   v1.27.5
11:02PM: prod-worker3   Ready    <none>          5h46m   v1.27.5
11:02PM: ----------------------
11:02PM: ---------------------------------------
11:02PM: Unfortunately, the dce5 installation failed at step [16] ....
11:02PM:
11:02PM: You can add -j16+ to your installation command to retry from last step.
11:02PM: ---------------------------------------
[Error]:Failed to Install DCE5, check above error messages. shell execution err is exit status 125

[root@localhost sample]#
```

解决办法：只需要在融合之前创建好该镜像空间即可，例如上图报错可通过创建镜像空间 localhost 提前避免。

架构冲突

升级至低于 0.12.0 版本的 addon 时，由于目标离线包里的 charts-syncer 没有检查镜像存在则不推送功能，因此会在升级的过程中会重新把多架构冲成单架构。 例如：在 v0.10 版本将 addon 实现为多架构，此时若升级为 v0.11 版本，则多架构 addon 会被覆盖为单架构；若升级为 0.12.0 及以上版本则仍能够保持多架构。

将自定义 HELM 应用导入系统内置 ADDON

本文从离线和在线两种环境说明如何将 Helm 应用导入到系统内置的 Addon 中。

离线环境

离线环境指的是无法连通互联网或封闭的私有网络环境。

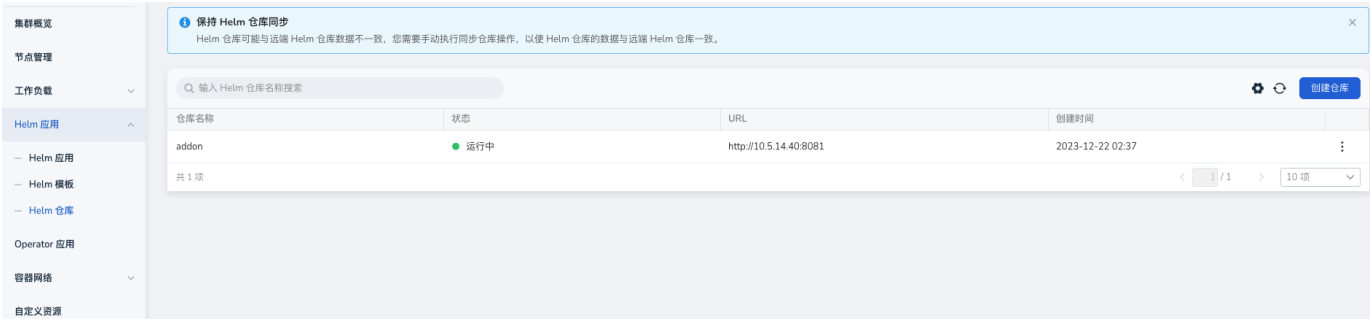
前提条件

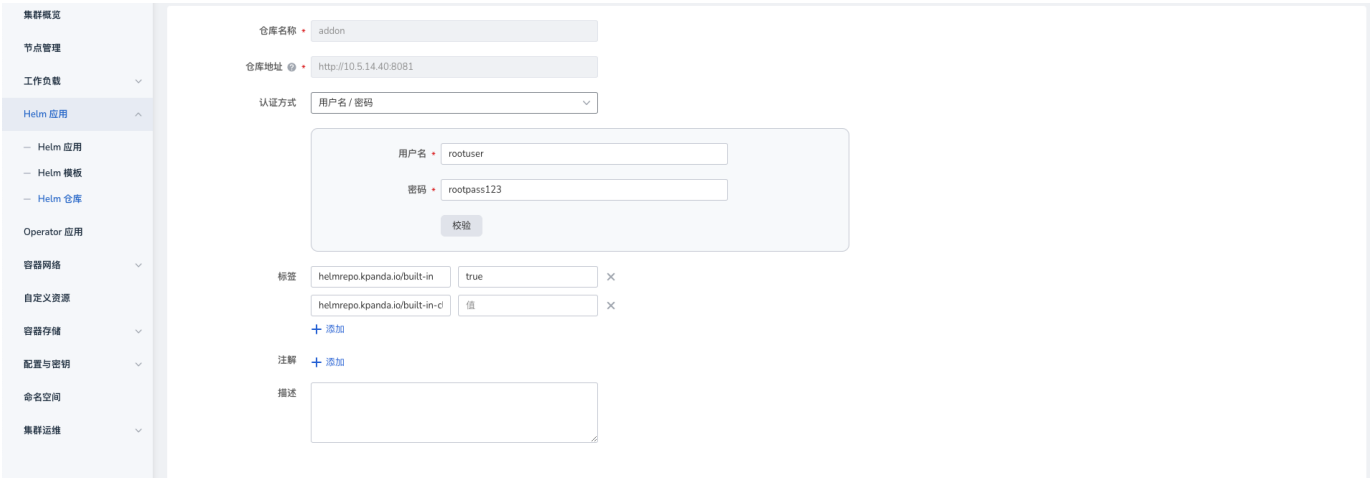
- 存在可以运行的 charts-syncer。 若没有，可点击下载。
- Helm Chart 已经完成适配 charts-syncer。 即在 Helm Chart 内添加了 `.relok8s-images.yaml` 文件。该文件需要包含 Chart 中所有使用到镜像， 也可以包含 Chart 中未直接使用的镜像，类似 Operator 中使用的镜像。

> **Note**
>
> - 如何编写 Chart 可参考 image-hints-file。 要求镜像的 registry 和 repository 必须分开，因为 load 镜像时需替换或修改 registry/repository。
> - 安装器所在的火种集群已安装 charts-syncer。 若将自定义 Helm 应用导入安装器所在火种集群，可跳过下载直接适配； 若未安装 charts-syncer二进制文件， 可立即下载。

同步 Helm Chart

1. 进入 `容器管理` -> `Helm 应用` -> `Helm 仓库`，搜索 addon，获取内置仓库地址和用户名/密码（系统内置仓库默认用户名/密码为 rootuser/rootpass123）。




集群概览
节点管理
工作负载
Helm 应用
Helm 应用
Helm 模板
Helm 仓库
Operator 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
仓库名称 *
addon
仓库地址 *
http://10.5.14.40:8081
认证方式
用户名 / 密码
用户名 *
rootuser
密码 *
rootpass123
校验
标签
helmrepo.kpanda.io/built-in
true
helmrepo.kpanda.io/built-in-cl
值
添加
注解
添加
描述

1. 同步 Helm Chart 到容器管理内置仓库 Addon

• 编写如下配置文件，可以根据具体配置修改，并保存为 `sync-dao-2048.yaml`。

```
source:  # helm charts 源信息
  repo:
    kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别，比如 CHARTMUSEUM
    url: https://release-ci.daocloud.io/chartrepo/community #  需更改为 chart repo url
    #auth: # 用户名/密码,若没有设置密码可以不填写
      #username: "admin"
      #password: "Harbor12345"
charts:  # 需要同步
  - name: dao-2048 # helm charts 信息，若不填写则同步源 helm repo 内所有 charts
    versions:
      - 1.4.1
target:  # helm charts 目标信息
  containerRegistry: 10.5.14.40 # 镜像仓库 url
  repo:
    kind: CHARTMUSEUM # 也可以是任何其他支持的 Helm Chart 仓库类别，比如 HARBOR
    url: http://10.5.14.40:8081 #  需更改为正确 chart repo url，可以通过 helm repo add $HELM-REPO 验证地址是否正确
    auth: # 用户名/密码，若没有设置密码可以不填写
      username: "rootuser"
      password: "rootpass123"
  containers:
    # kind: HARBOR # 若镜像仓库为 HARBOR 且希望 charts-syncer 自动创建镜像 Repository 则填写该字段
    # auth: # 用户名/密码，若没有设置密码可以不填写
      # username: "admin"
      # password: "Harbor12345"

# leverage .relok8s-images.yaml file inside the Charts to move the container images too
relocateContainerImages: true
```

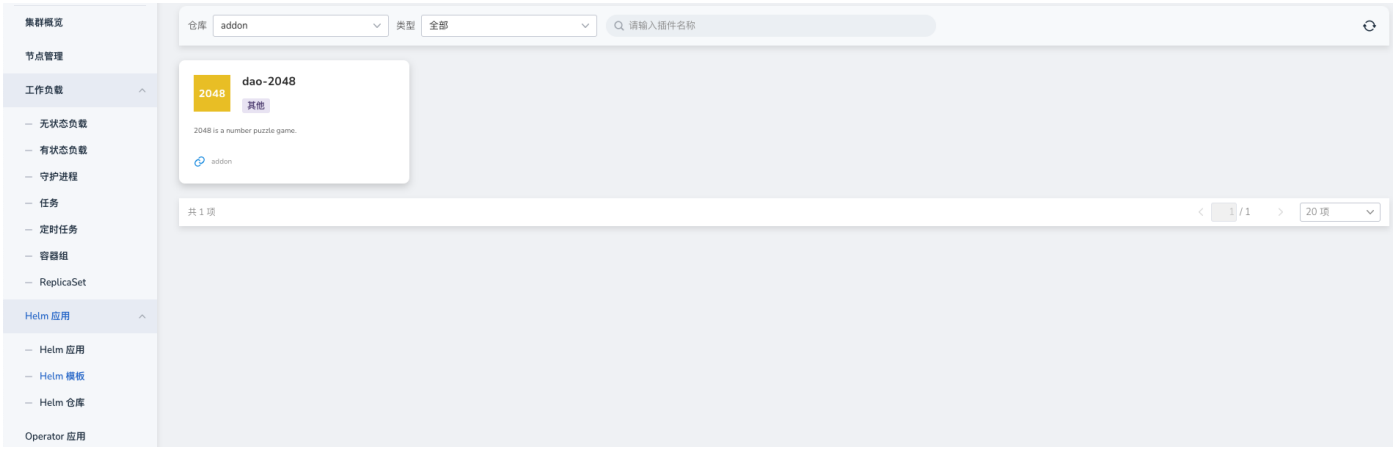
• 执行 charts-syncer 命令同步 Chart 及其包含的镜像

```
charts-syncer sync --config sync-dao-2048.yaml --insecure --auto-create-repository
```

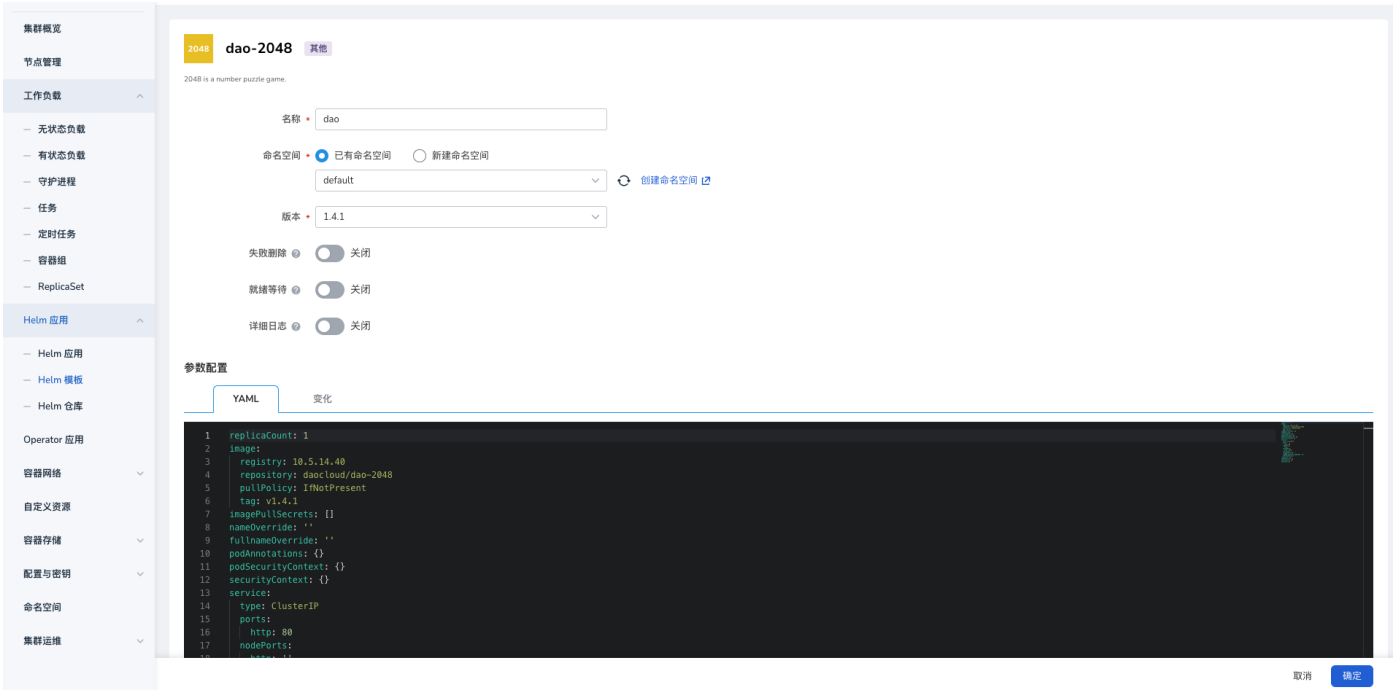
预期输出为：

```
I1222 15:01:47.119777    8743 sync.go:45] Using config file: "examples/sync-dao-2048.yaml"
W1222 15:01:47.234238    8743 syncer.go:263] Ignoring skipDependencies option as dependency sync is not supported if container image relocation is true or
syncing from/to intermediate directory
I1222 15:01:47.234685    8743 sync.go:58] There is 1 chart out of sync!
I1222 15:01:47.234706    8743 sync.go:66] Syncing "dao-2048_1.4.1" chart...
.relok8s-images.yaml hints file found
Computing relocation...

Relocating dao-2048@1.4.1...
Pushing 10.5.14.40/daocloud/dao-2048:v1.4.1...
Done
Done moving /var/folders/vm/08vw0t3j68z9z_4lcqyhg8nm0000gn/T/charts-syncer869598676/dao-2048-1.4.1.tgz
```

2. 待上一步执行完成后，进入 `容器管理` -> `Helm 应用` -> `Helm 仓库`，找到对应 Addon， 在操作栏点击 `同步仓库`，回到 Helm 模板就可以看到上传的 Helm 应用



3. 后续可正常进行安装、升级、卸载



在线环境

在线环境的 Helm Repo 地址为 `release.daocloud.io`。 如果用户无权限添加 Helm Repo，则无法将自定义 Helm 应用导入系统内置 Addon。 您可以添加自己搭建的 Helm 仓库，然后按照离线环境中同步 Helm Chart 的步骤将您的 Helm 仓库集成到平台使用。

容器网络

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。

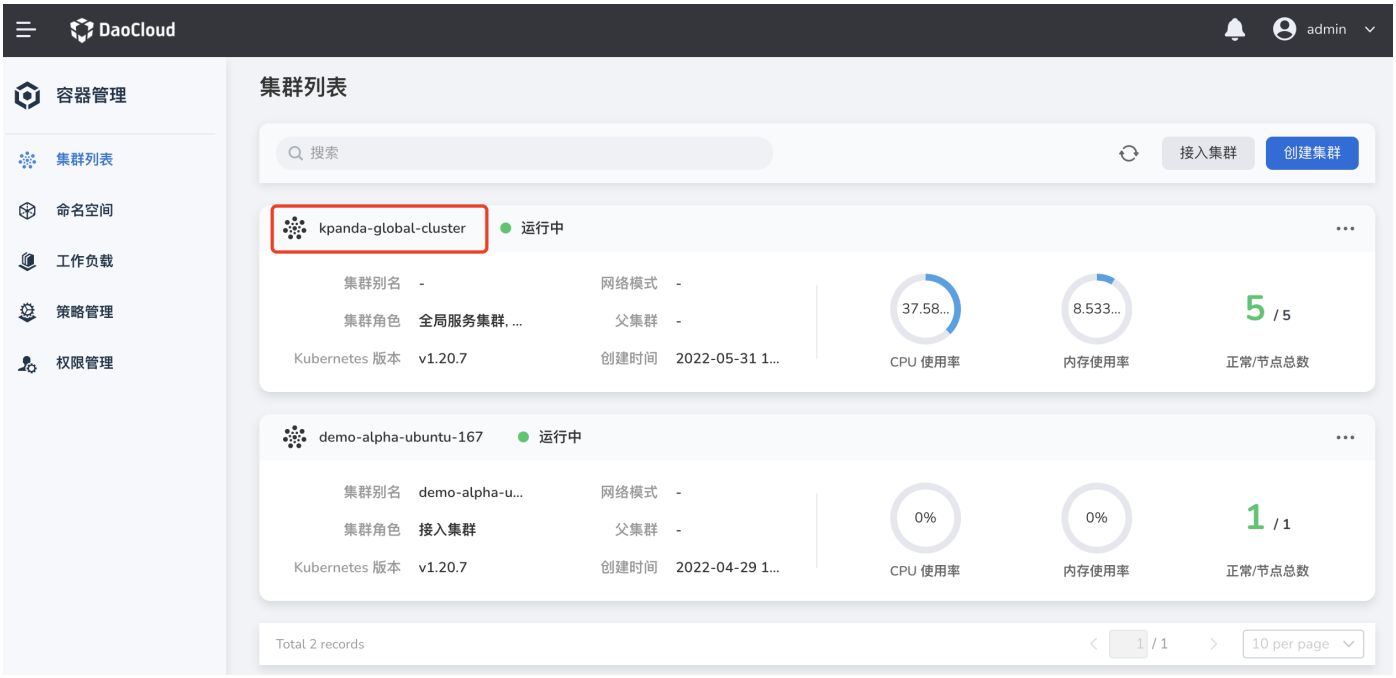
这就需要创建服务，通过服务您会获得一个固定的 IP 地址，从而实现工作负载前端和后端的解耦，让外部用户能够访问服务。同时，服务还提供了负载均衡（LoadBalancer）功能，使用户能从公网访问到工作负载。

前提条件

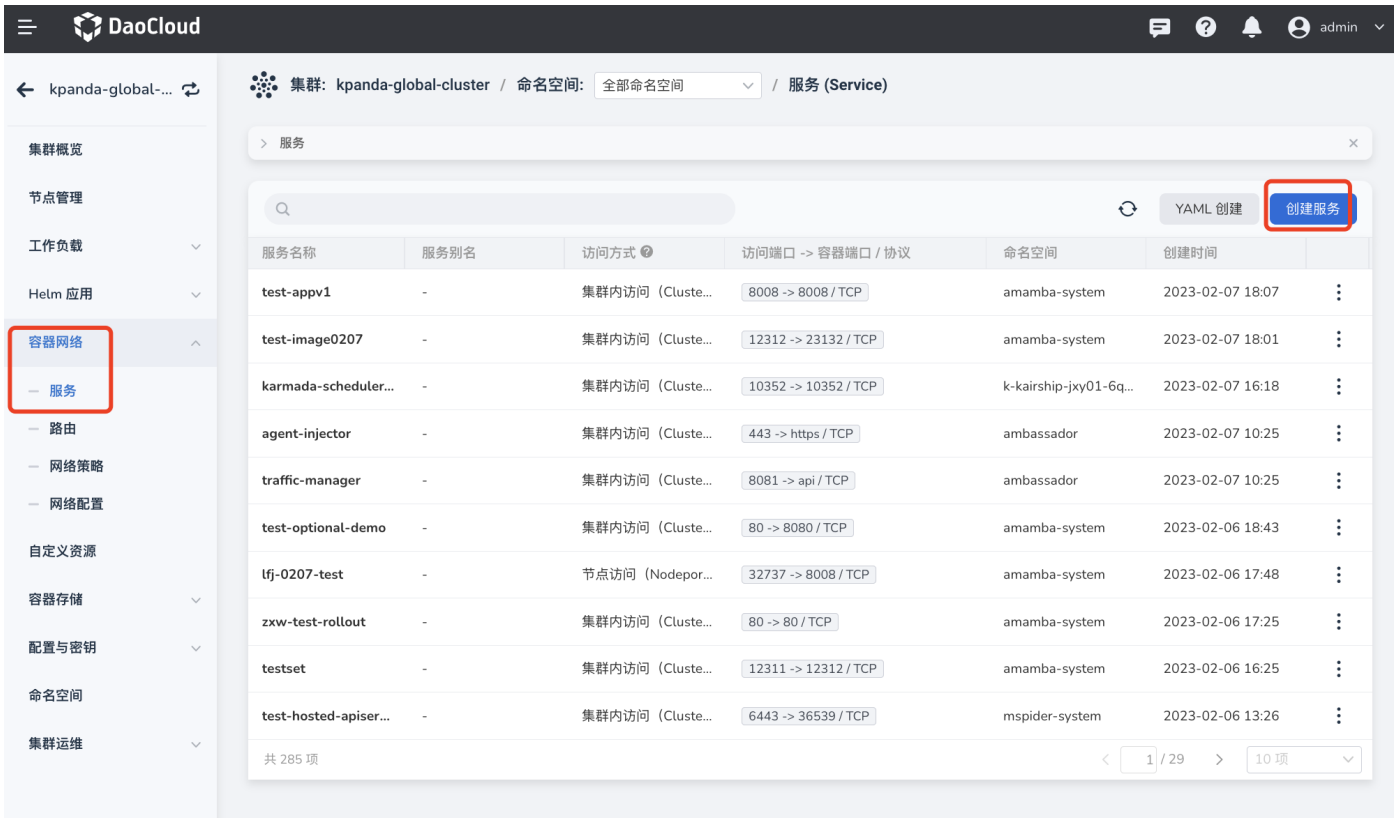
- 已完成一个命名空间的创建、用户的创建， 并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

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。



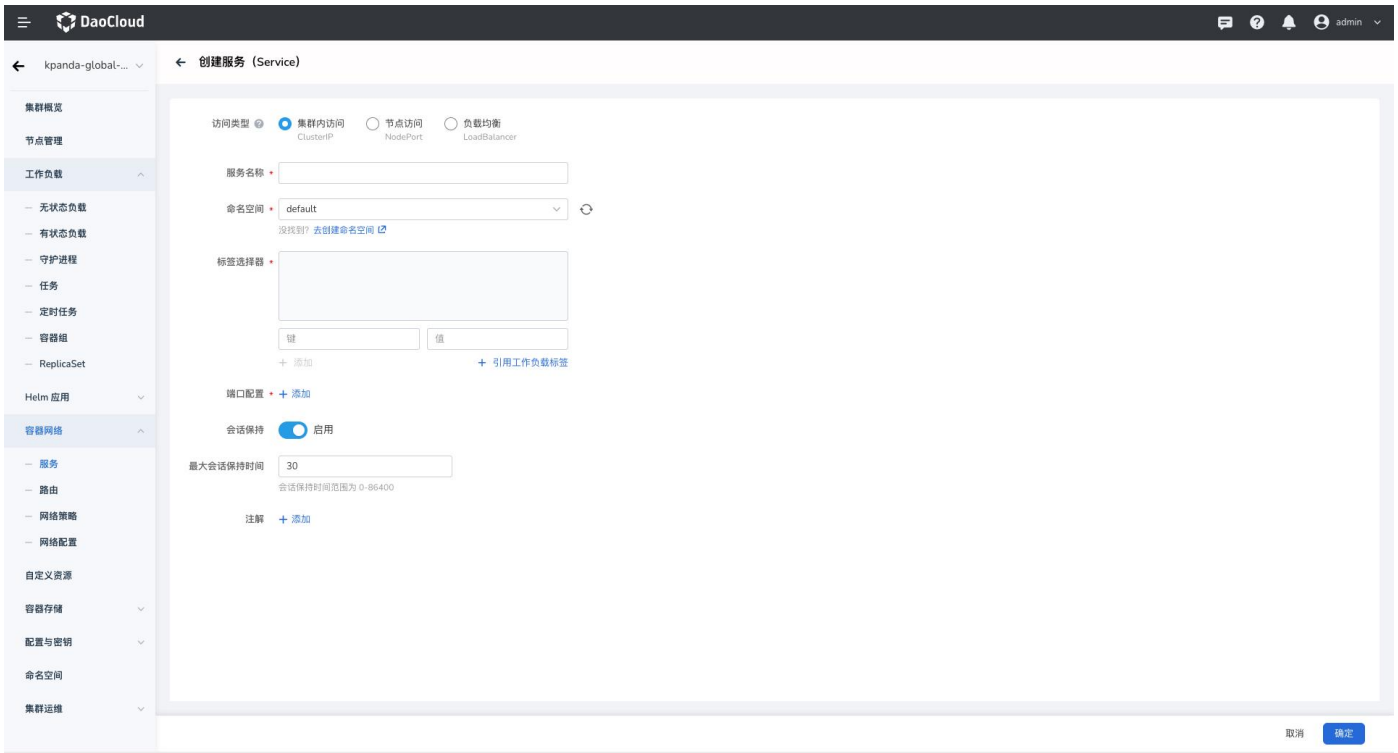
2. 在左侧导航栏中，点击 容器网络 -> 服务 进入服务列表，点击右上角 创建服务 按钮。



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 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。	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。 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
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 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 秒 .spec.sessionAffinityConfig.clientIP.timeoutSeconds 默认设置为 30 秒	30 秒
注解	【类型】选填 【含义】为服务添加注解	

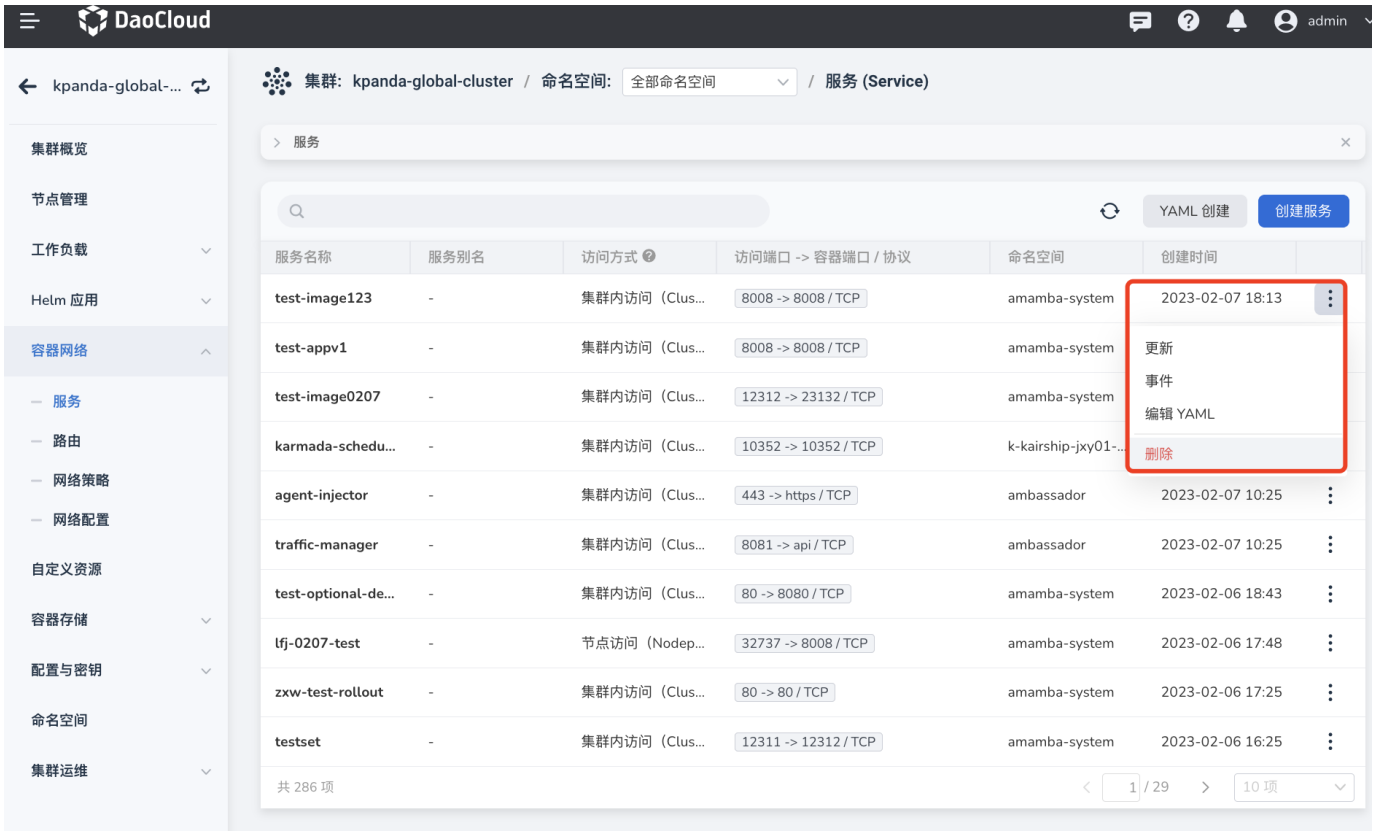
创建 LoadBalancer 服务

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。	
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。 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 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	

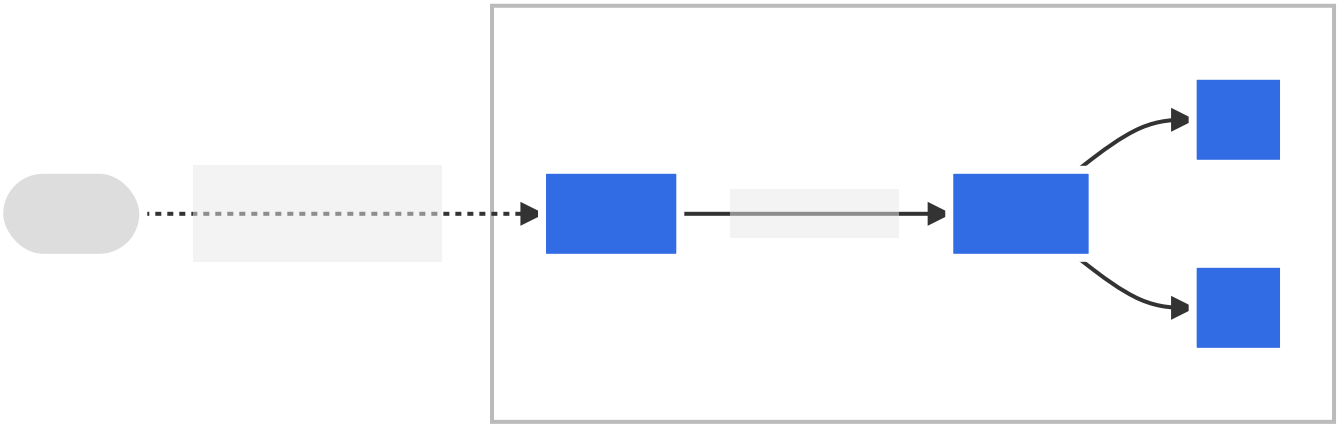
完成服务创建

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。



创建路由（INGRESS）

在 Kubernetes 集群中，Ingress 公开从集群外部到集群内服务的 HTTP 和 HTTPS 路由。流量路由由 Ingress 资源上定义的规则控制。下面是一个将所有流量都发送到同一 Service 的简单 Ingress 示例：



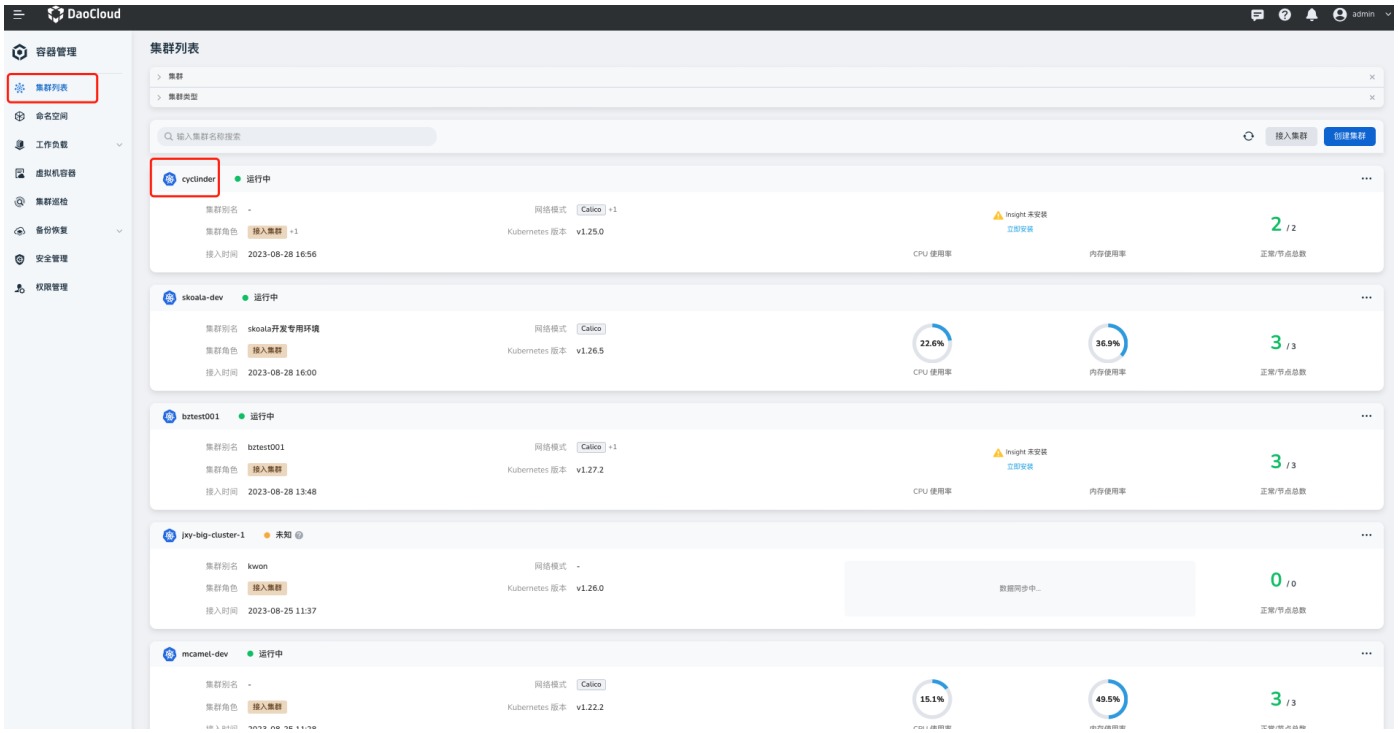
Ingress 是对集群中服务的外部访问进行管理的 API 对象，典型的访问方式是 HTTP。Ingress 可以提供负载均衡、SSL 终结和基于名称的虚拟托管。

前提条件

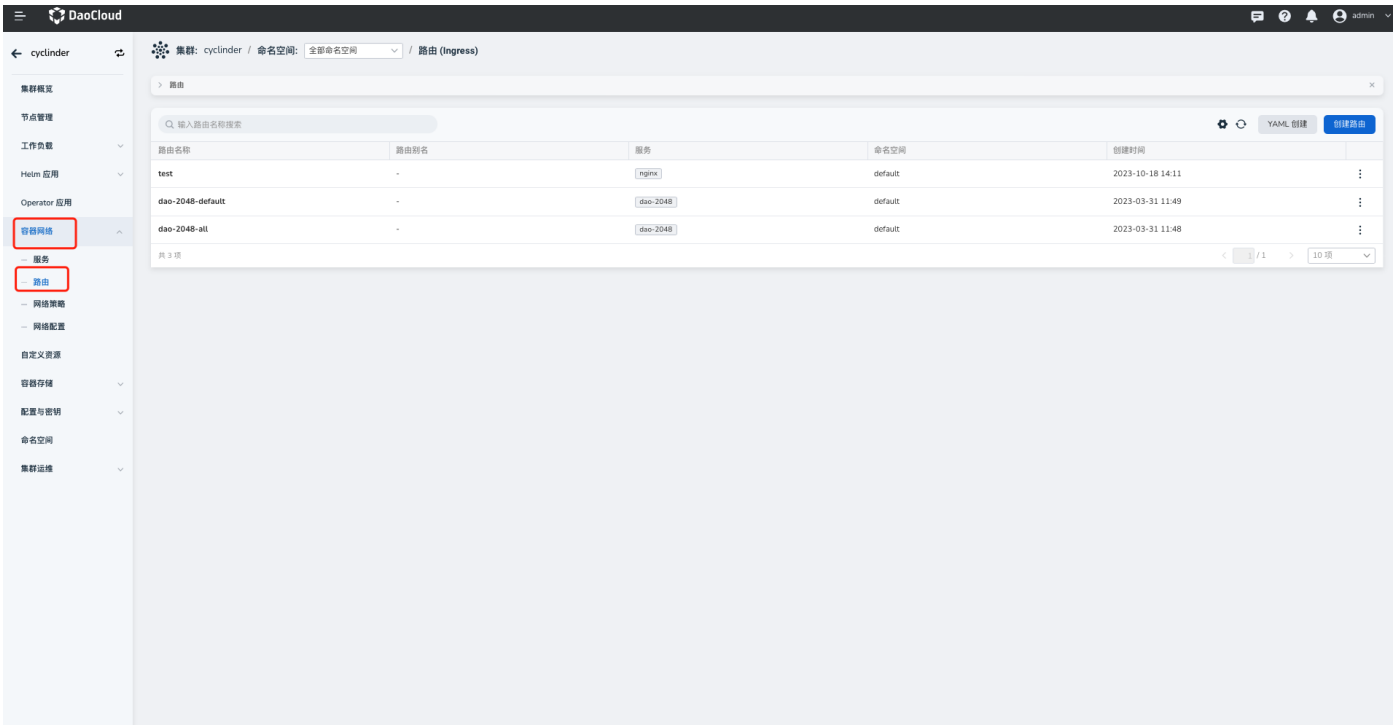
- 已完成一个命名空间的创建、用户的创建， 并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。
- 已经完成 Ingress 实例的创建，已部署应用工作负载，并且已创建对应 Service
- 单个实例中有多个容器时，请确保容器使用的端口不冲突，否则部署会失效。

创建路由

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。



2. 在左侧导航栏中，点击 容器网络 -> 路由 进入服务列表，点击右上角 创建路由 按钮。

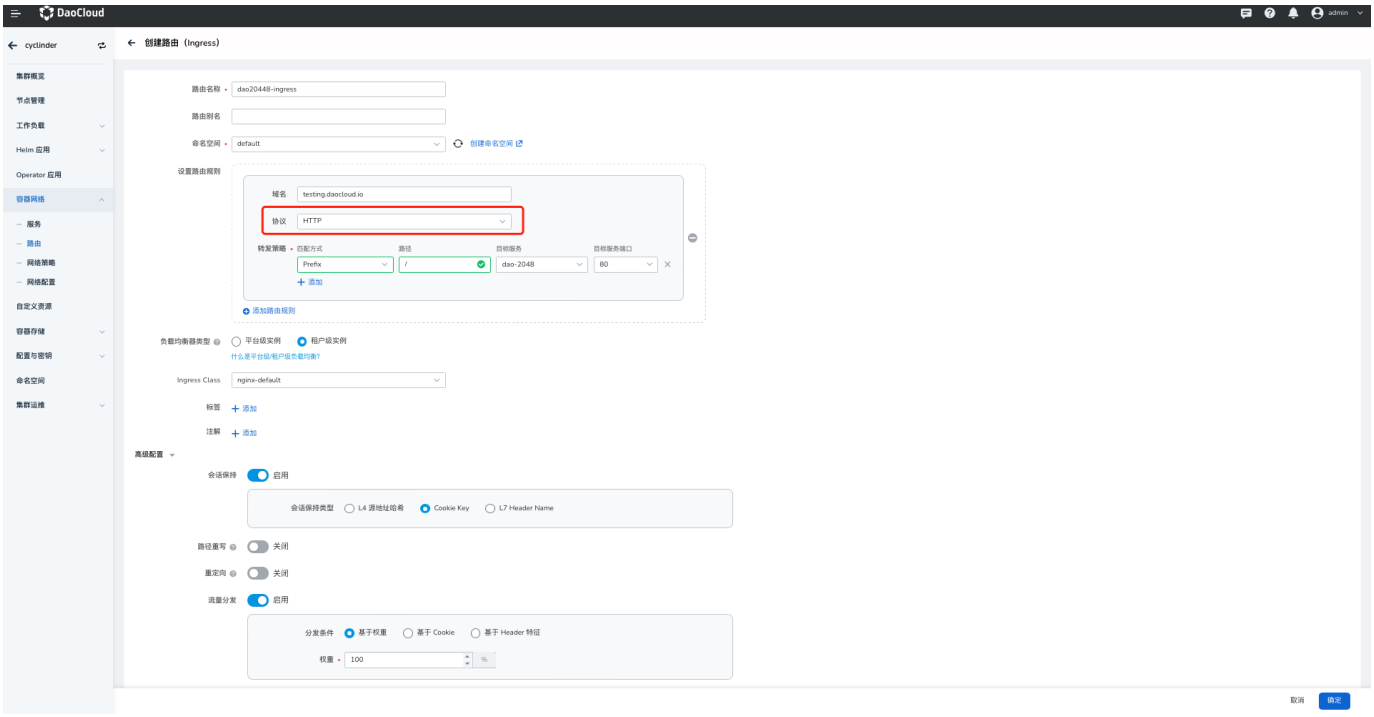


> **Note**
>
> 也可以通过 **YAML** 创建 一个路由。

3. 打开 创建路由 页面，进行配置。可选择两种协议类型，参考以下两个参数表进行配置。

   创建 HTTP 协议路由

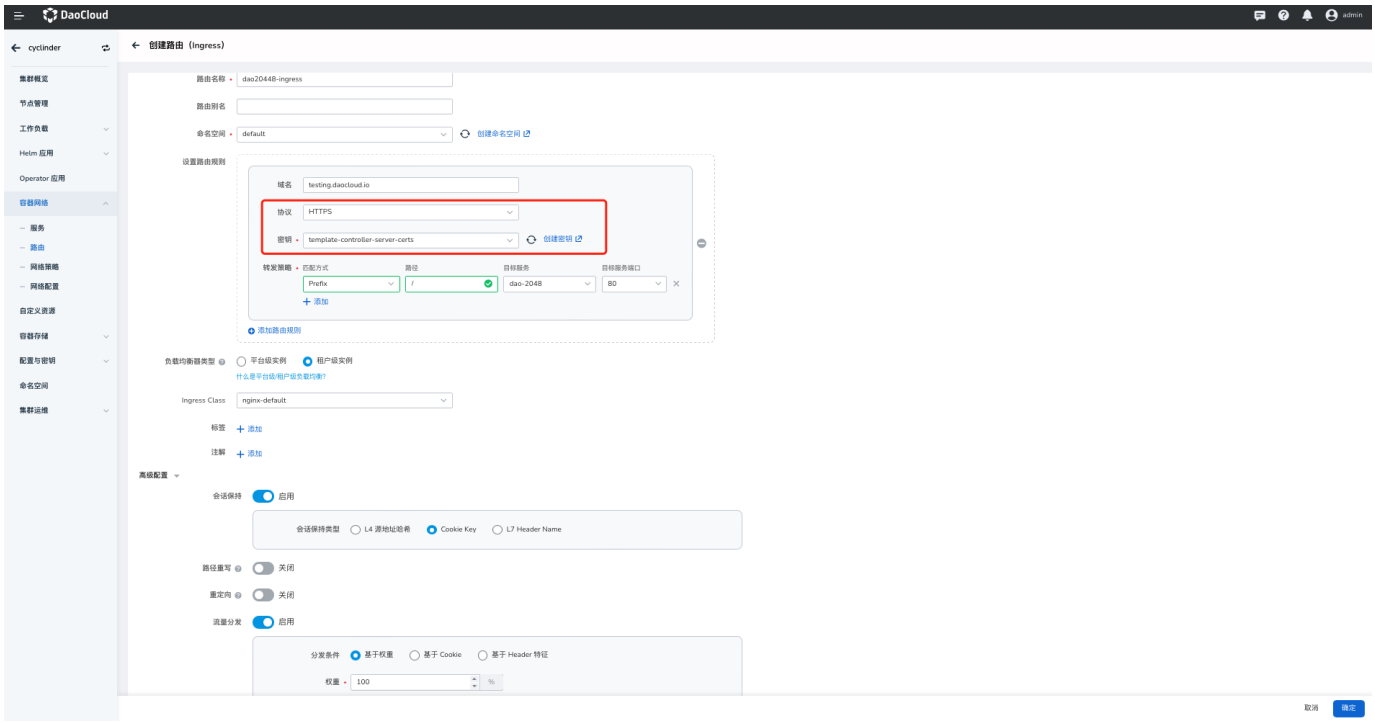
   输入如下参数：


DaoCloud
admin
cyclinder
集群概览
节点管理
工作负载
Helm 应用
Operator 应用
容器网络
服务
路由
网络策略
网络配置
自定义资源
容器存储
配置与密钥
命名空间
集群运维
创建路由（Ingress）
路由名称
dao20448-ingress
路由别名
命名空间
default
创建命名空间
设置路由规则
域名
testing.daocloud.io
协议
HTTP
转发策略
匹配方式
路径
目标服务
目标服务端口
Prefix
/
dao-2048
80
添加
添加路由规则
负载均衡器类型
平台级实例
租户级实例
什么是平台级/租户级负载均衡?
Ingress Class
nginx-default
标签
添加
注解
添加
高级配置
会话保持
启用
会话保持类型
L4 源地址哈希
Cookie Key
L7 Header Name
路径重写
关闭
重定向
关闭
流量分发
启用
分发条件
基于权重
基于 Cookie
基于 Header 特征
权重
100
%
取消
确定

- 路由名称 :必填，输入新建路由的名称。
- 命名空间 ： 必填，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。
- 设置路由规则 ：
- 域名 ： 必填，使用域名对外提供访问服务。默认为集群的域名。
- 协议 ： 必填，指授权入站到达集群服务的协议，支持 HTTP （不需要身份认证）或 HTTPS（需需要配置身份认证） 协议。这里选择 HTTP 协议的路由。
- 转发策略 :选填，指定 Ingress 的访问策略。 路径 ： 指定服务访问的URL路径，默认为根路径； 目标服务 ： 进行路由的服务名称； 目标服务端口 ： 服务对外暴露的端口。
- 负载均衡器类型 :必填，Ingress 实例的使用范围
- 平台级负载均衡器 ： 同一个集群内，共享同一个 Ingress 实例，其中 Pod 都可以接收到由该负载均衡分发的请求。
- 租户级负载均衡器 ： 租户负载均衡器，Ingress 实例独属于当前命名空，或者独属于某一工作空间，并且设置的工作空间中包含当前命名空间，其中 Pod 都可以接收到由该负载均衡分发的请求。
- **Ingress Class** :选填，选择对应的 Ingress 实例，选择后将流量导入到指定的 Ingress 实例。
- 为 None 时使用默认的 DefaultClass，请在创建 Ingress 实例时设置 DefaultClass，更多信息，请参考 Ingress Class。
- 若选择其他实例（如 **ngnix** ），则会出现高级配置，可设置 会话保持 ， 路径重写 ， 重定向 ，和 流量分发 。
- 会话保持 ： 选填，会话保持分为 三种类型： **L4** 源地址哈希 ， **Cookie Key** ， **L7 Header Name** ,开启后根据对应规则进行会话保持。
- **L4** 源地址哈希 : :开启后默认在 Annotation 中加入如下标签：nginx.ingress.kubernetes.io/upstream-hash-by: "$binary_remote_addr"
- **Cookie Key** ： 开启后来自特定客户端的连接将传递至相同 Pod，开启后 默认在 Annotation 中增加如下参数：nginx.ingress.kubernetes.io/affinity: "cookie"。nginx.ingress.kubernetes.io/affinity-mode: persistent
- **L7 Header Name** :开启后默认在 Annotation 中加入如下标签：nginx.ingress.kubernetes.io/upstream-hash-by: "$http_x_forwarded_for"
- 路径重写 :选填， **rewrite-target** ，某些场景中后端服务暴露的URL与Ingress规则中指定的路径不同，如果不进行URL重写配置，访问会出现错误。
- 重定向 ： 选填， **permanent-redirect** ，永久重定向，输入重写路径后，访问路径重定向至设置的地址。
- 流量分发 ： 选填，开启后并设置后，根据设定条件进行流量分发。
- 基于权重 ： 设定权重后，在创建的 Ingress 添加如下：Annotation: **nginx.ingress.kubernetes.io/canary-weight: "10"** 。
- 基于 **Cookie** ： 设定 Cookie 规则后，流量根据设定的 Cookie 条件进行流量分发。
- 基于 **Header** ： 设定 Header 规则后，流量根据设定的 Header 条件进行流量分发。
- 标签 ： 选填，为路由添加标签。
- 注解 ： 选填，为路由添加注解。

创建 HTTPS 协议路由

输入如下参数：



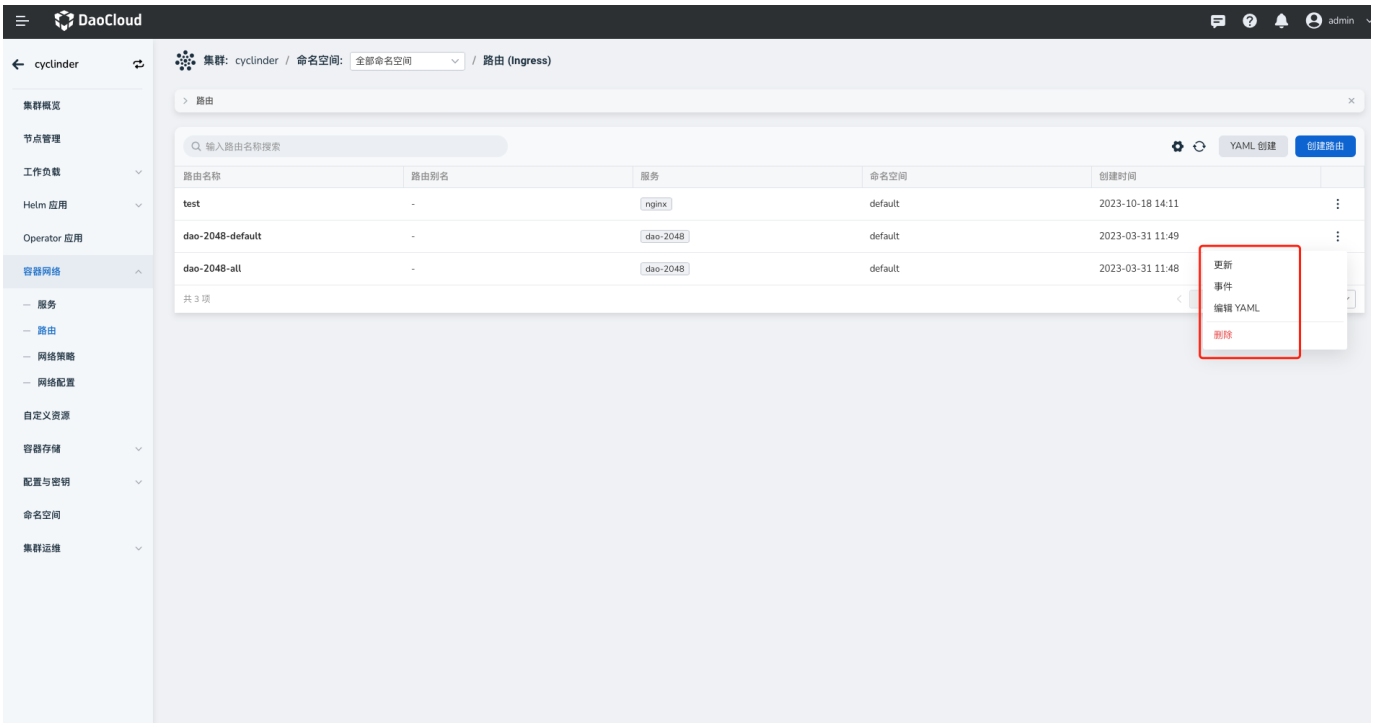
Note

注意：与 HTTP 协议 设置路由规则 不同，增加密钥选择证书，其他基本一致。

- 协议 :必填指授权入站到达集群服务的协议，支持 HTTP （不需要身份认证）或 HTTPS（需需要配置身份认证） 协议。这里选择 HTTPS 协议的路由。
- 密钥 ： 必填，Https TLS 证书，创建秘钥。

完成路由创建

配置完所有参数后，点击 确定 按钮，自动返回路由列表。在列表右侧，点击 ⋮ ，可以修改或删除所选路由。


DaoCloud
admin
cyclinder
集群概览
节点管理
工作负载
Helm 应用
Operator 应用
容器网络
服务
路由
网络策略
网络配置
自定义资源
容器存储
配置与密钥
命名空间
集群运维
集群: cyclinder / 命名空间: 全部命名空间 / 路由 (Ingress)
路由
输入路由名称搜索
YAML 创建
创建路由
路由名称
路由别名
服务
命名空间
创建时间
test
nginx
default
2023-10-18 14:11
dao-2048-default
dao-2048
default
2023-03-31 11:49
dao-2048-all
dao-2048
default
2023-03-31 11:48
共 3 项
更新
事件
编辑 YAML
删除

网络策略

网络策略（NetworkPolicy）可以在 IP 地址或端口层面（OSI 第 3 层或第 4 层）控制网络流量。 容器管理模块目前支持创建基于 Pod 或命名空间的网络策略，支持通过标签选择器来设定哪些流量可以进入或离开带有特定标签的 Pod。

有关网络策略的更多详情，可参考 Kubernetes 官方文档网络策略。

创建网络策略

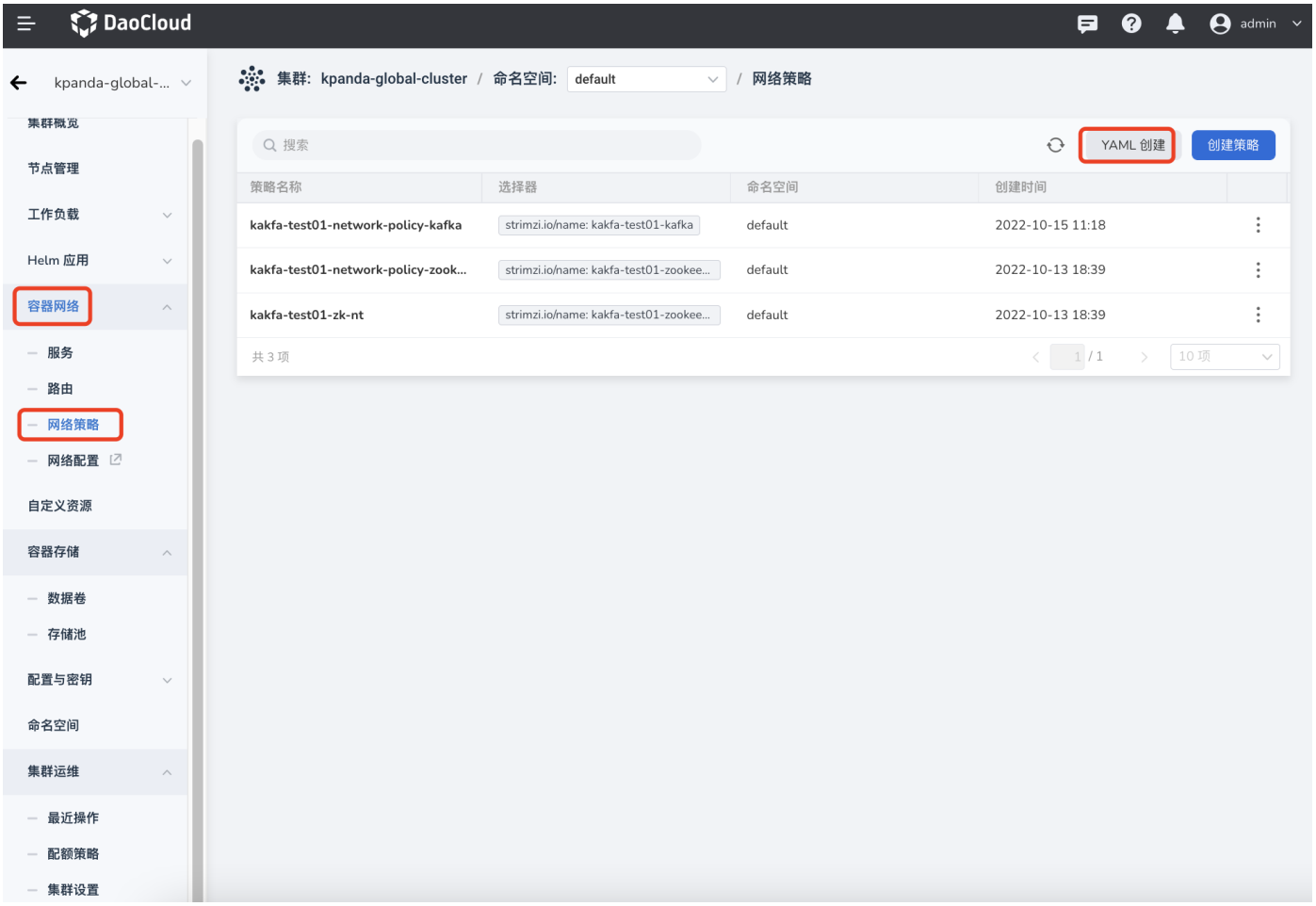
目前支持通过 YAML 和表单两种方式创建网络策略，这两种方式各有优劣，可以满足不同用户的使用需求。

通过 YAML 创建步骤更少、更高效，但门槛要求较高，需要熟悉网络策略的 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。

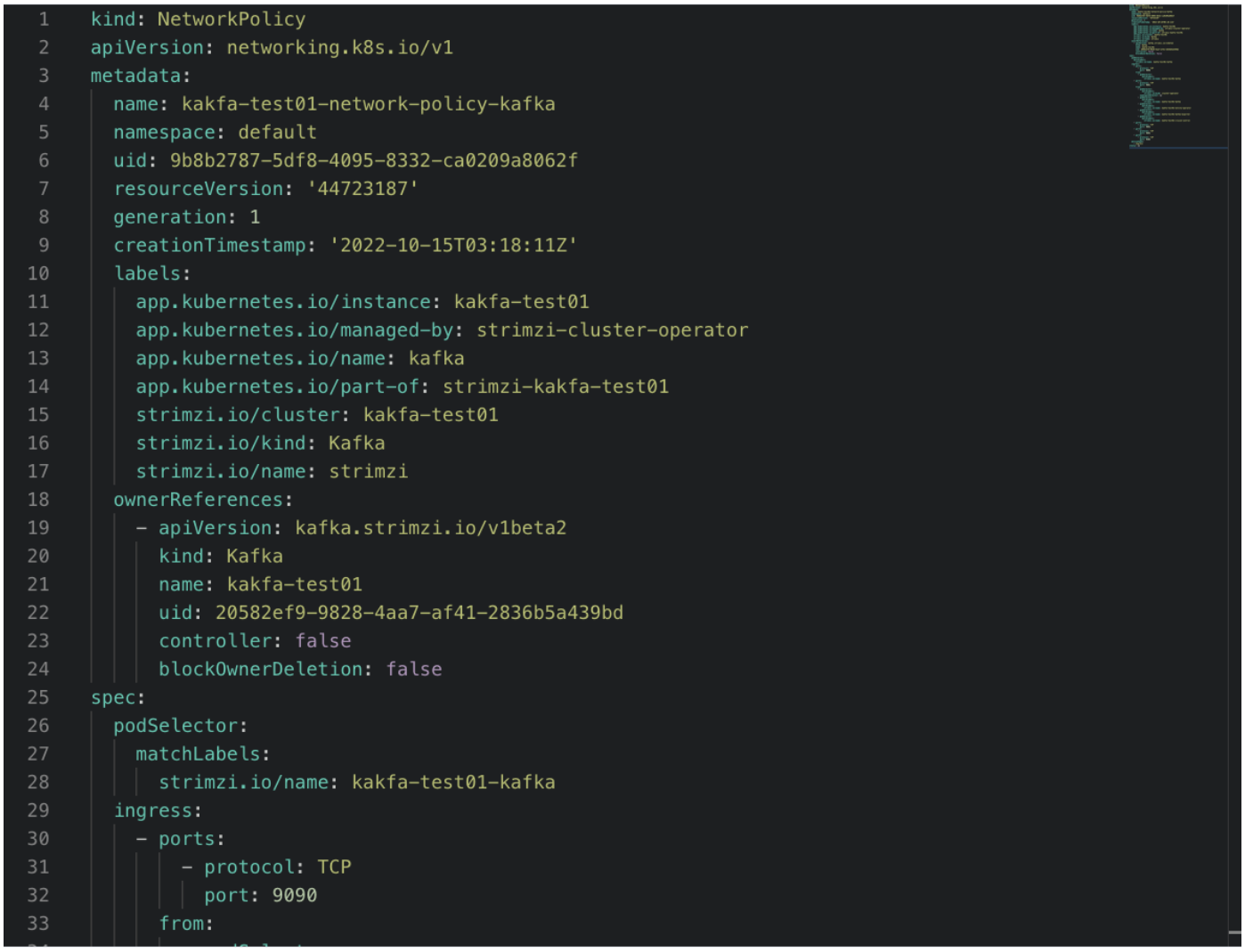
YAML 创建

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 容器网络 -> 网络策略 -> **YAML** 创建 。



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 。

YAML 创建

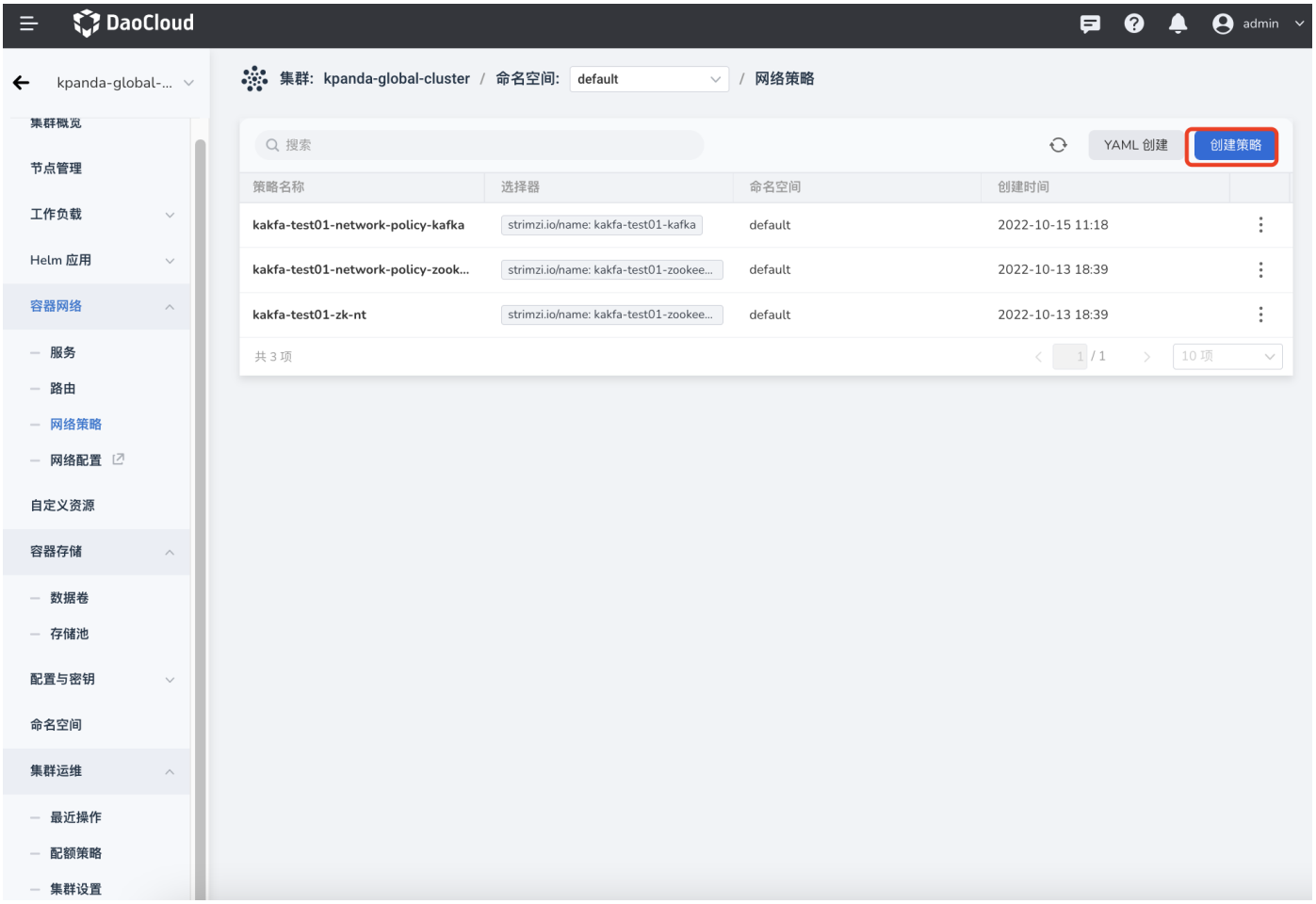


```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: kakfa-test01-network-policy-kafka
  namespace: default
  uid: 9b8b2787-5df8-4095-8332-ca0209a8062f
  resourceVersion: '44723187'
  generation: 1
  creationTimestamp: '2022-10-15T03:18:11Z'
  labels:
    app.kubernetes.io/instance: kakfa-test01
    app.kubernetes.io/managed-by: strimzi-cluster-operator
    app.kubernetes.io/name: kafka
    app.kubernetes.io/part-of: strimzi-kakfa-test01
    strimzi.io/cluster: kakfa-test01
    strimzi.io/kind: Kafka
    strimzi.io/name: strimzi
  ownerReferences:
    - apiVersion: kafka.strimzi.io/v1beta2
      kind: Kafka
      name: kakfa-test01
      uid: 20582ef9-9828-4aa7-af41-2836b5a439bd
      controller: false
      blockOwnerDeletion: false
spec:
  podSelector:
    matchLabels:
      strimzi.io/name: kakfa-test01-kafka
  ingress:
    - ports:
        - protocol: TCP
          port: 9090
      from:
```

取消 确定

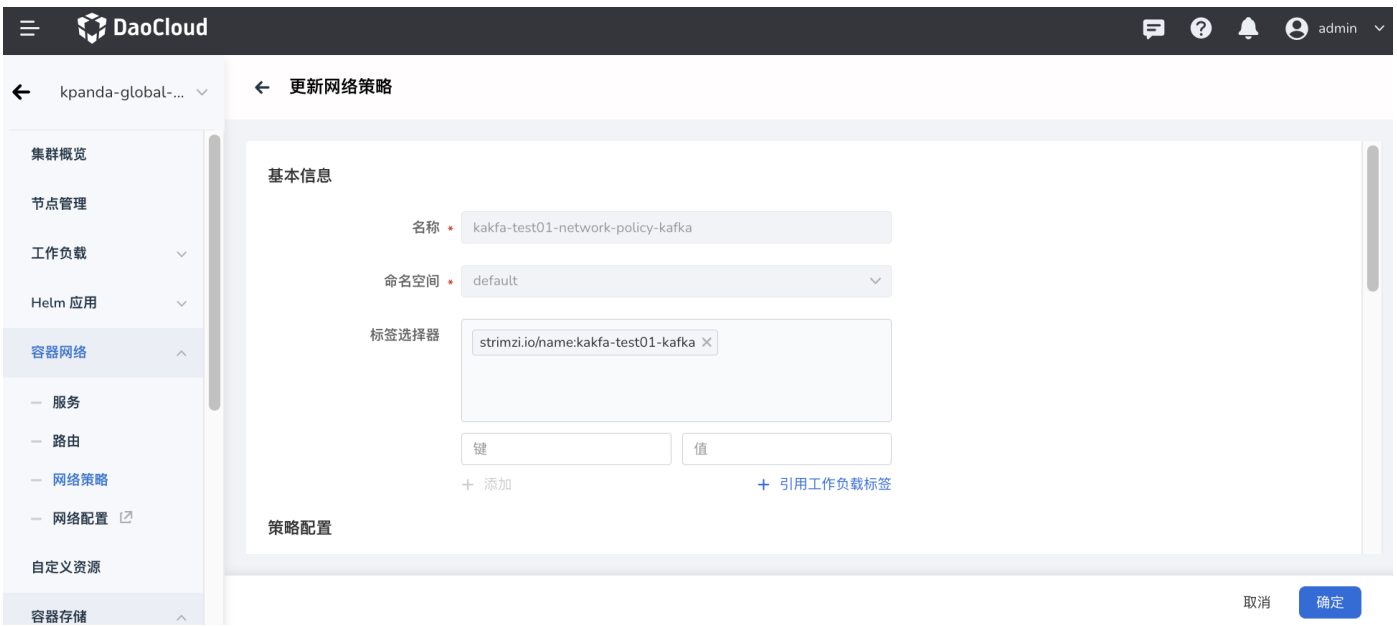
表单创建

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 容器网络 -> 网络策略 -> 创建策略 。



2. 填写基本信息。

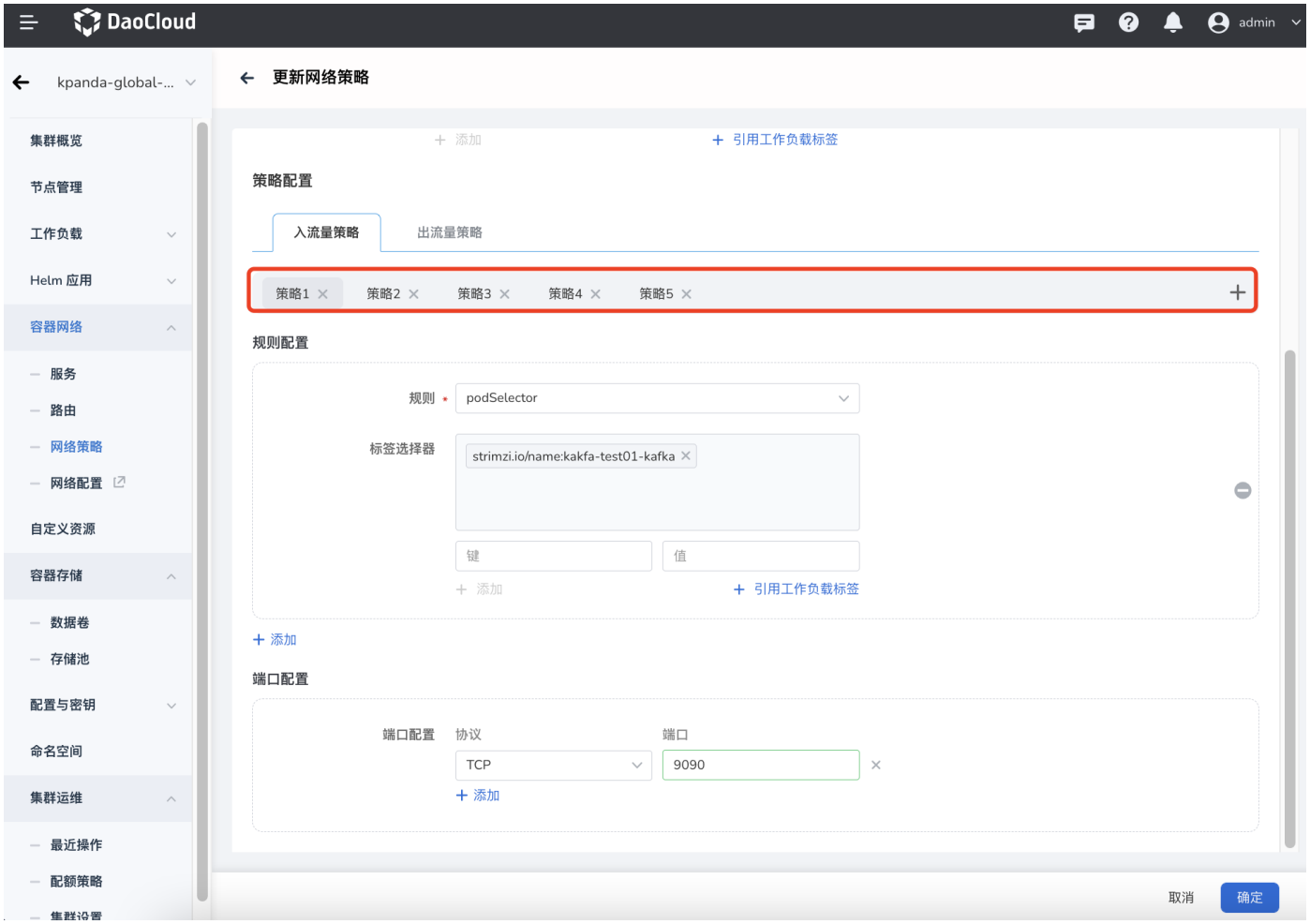
   名称和命名空间在创建之后不可更改。



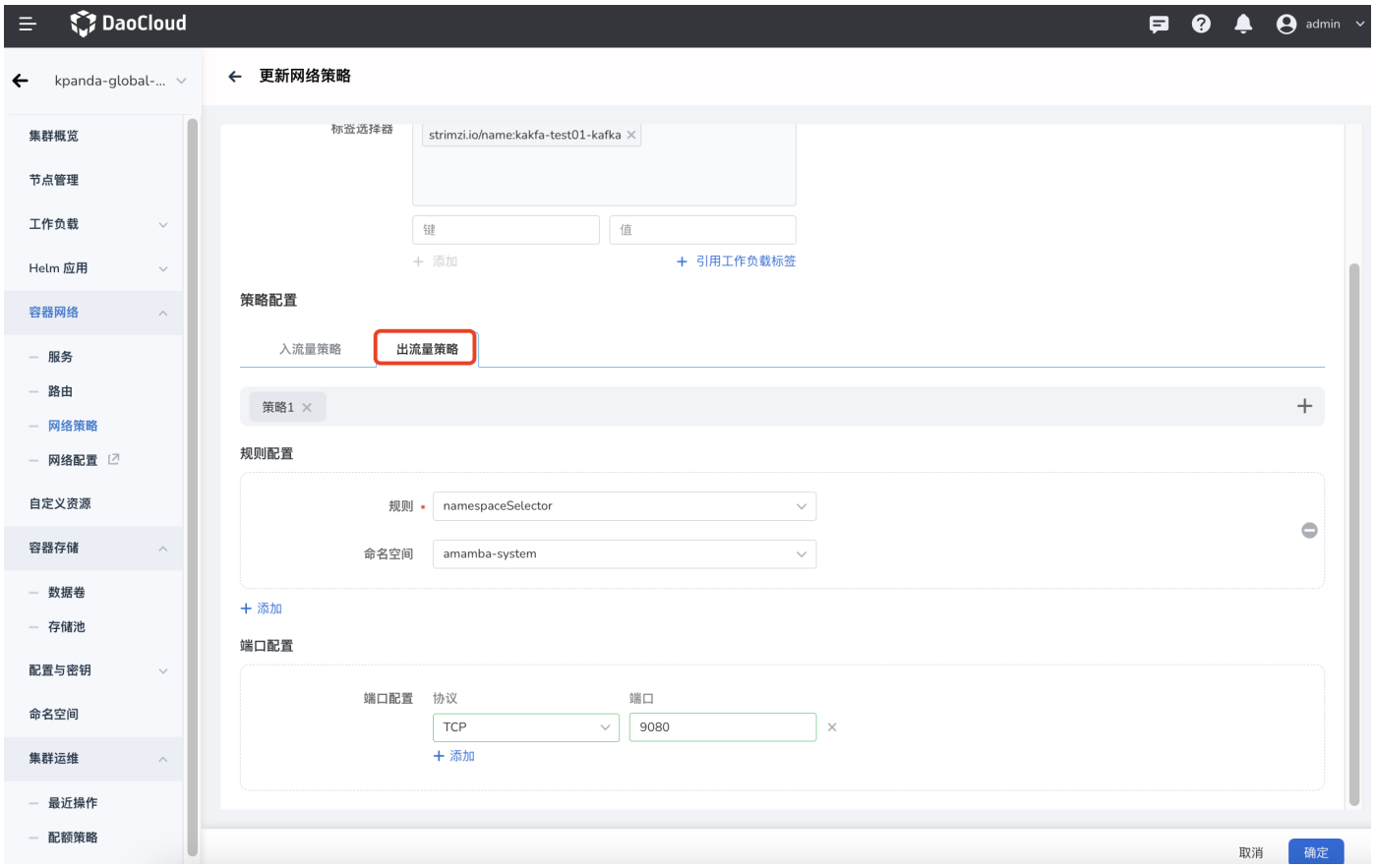
3. 填写策略配置。

策略配置分为入流量策略和出流量策略。如果源 Pod 想要成功连接到目标 Pod，源 Pod 的出流量策略和目标 Pod 的入流量策略都需要允许连接。如果任何一方不允许连接，都会导致连接失败。

• 入流量策略：点击 ＋ 开始配置策略，支持配置多条策略。多条网络策略的效果相互叠加，只有同时满足所有网络策略，才能成功建立连接。

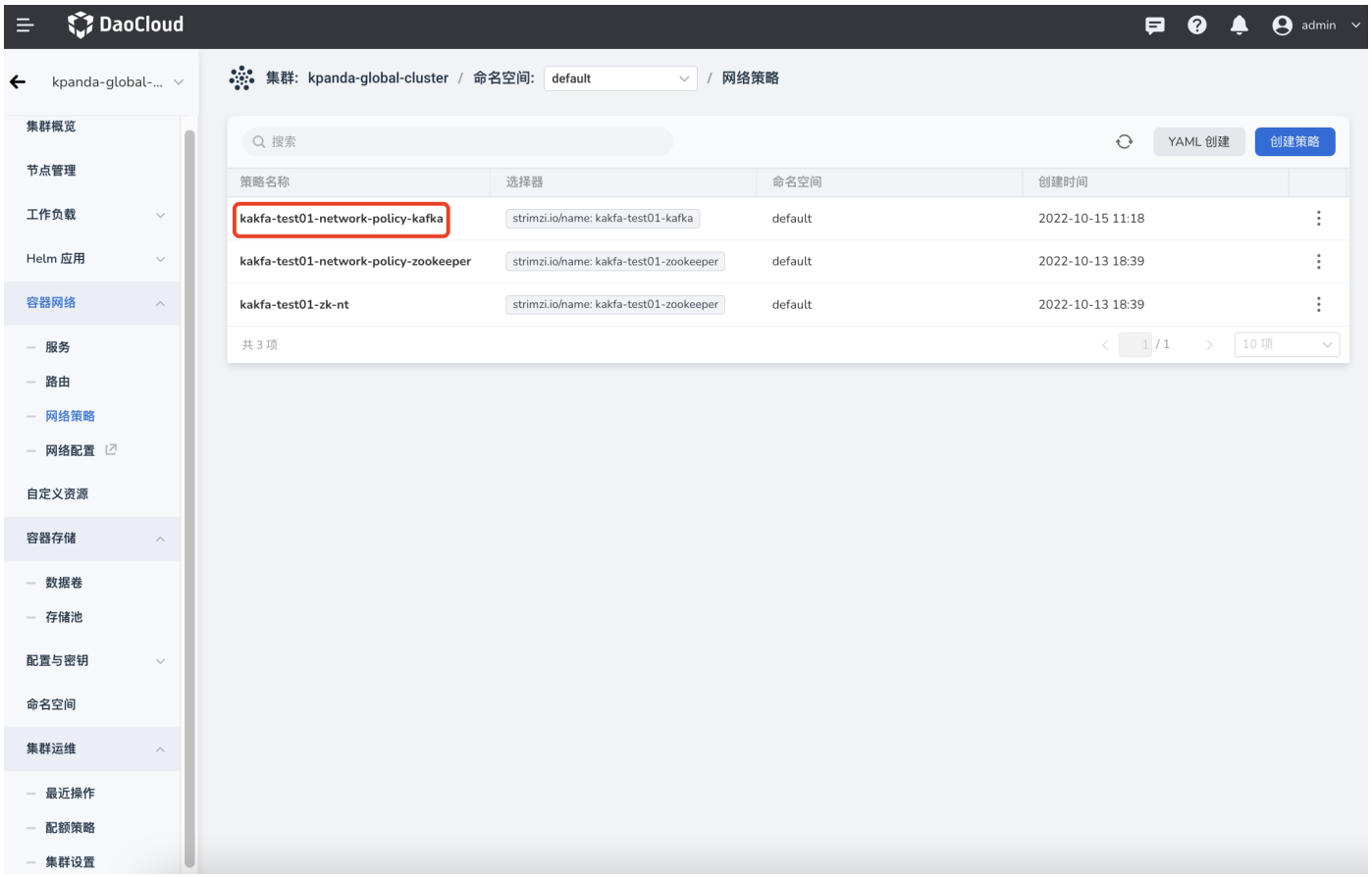


• 出流量策略

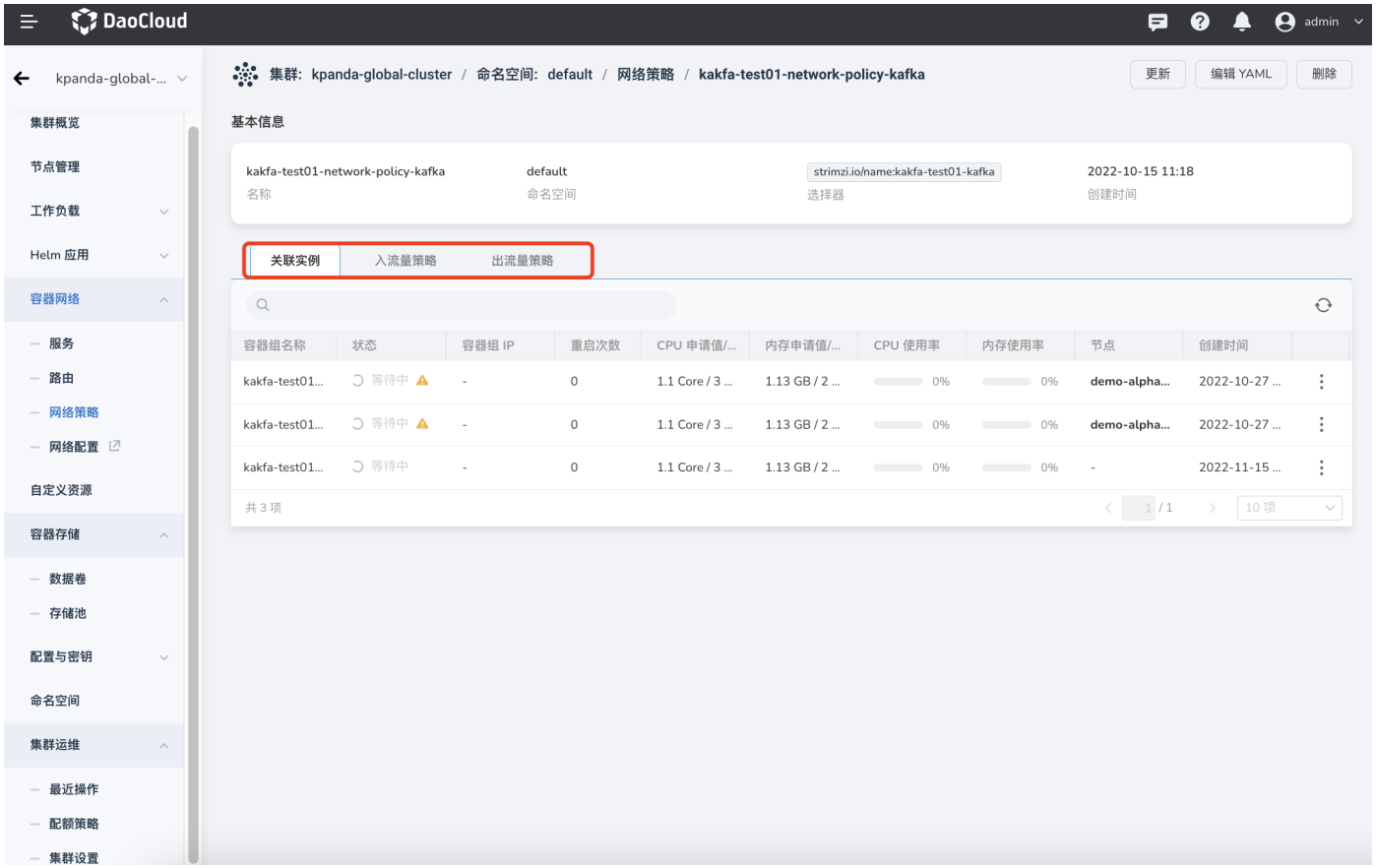


查看网络策略

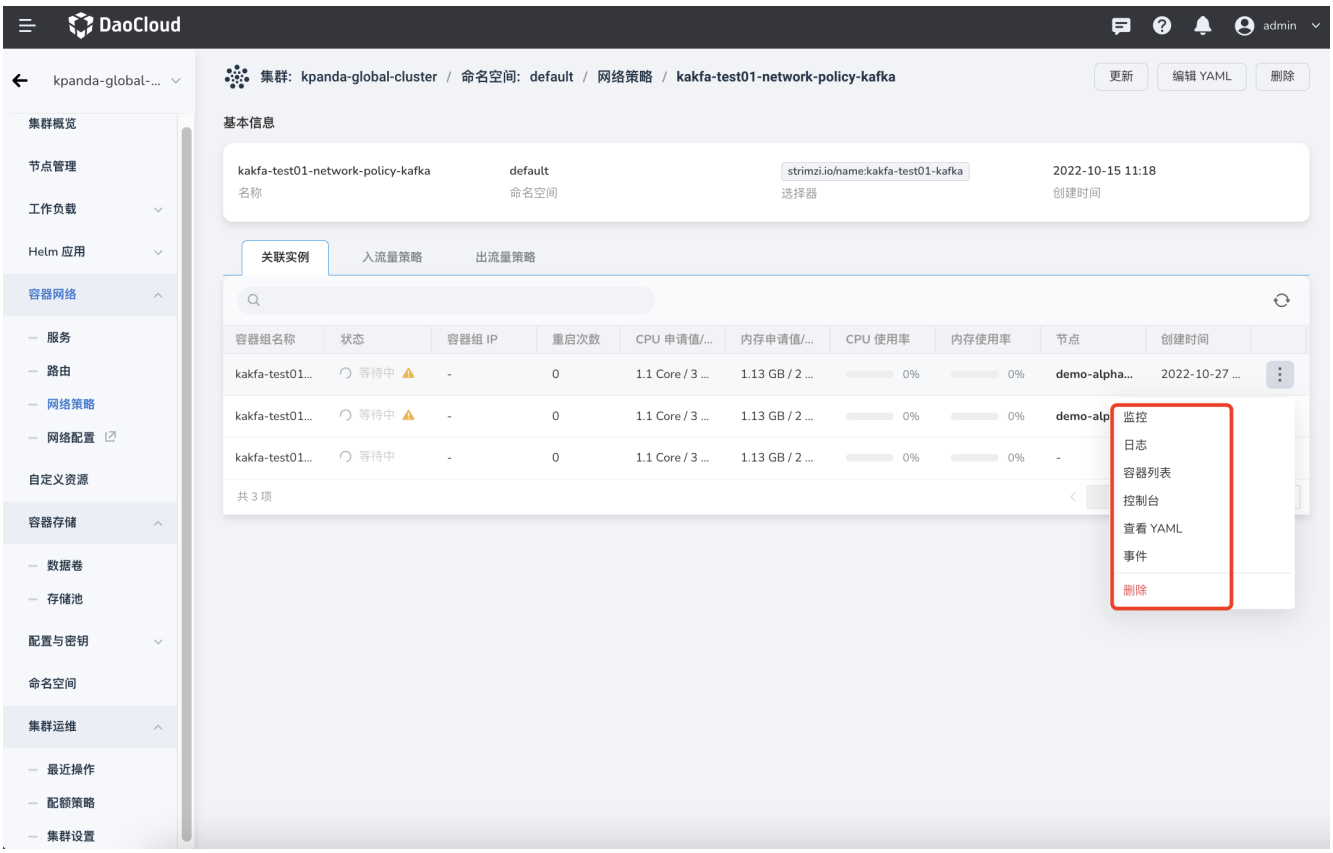
1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击 容器网络 -> 网络策略 ，点击网络策略的名称。



2. 查看该策略的基本配置、关联实例信息、入流量策略、出流量策略。



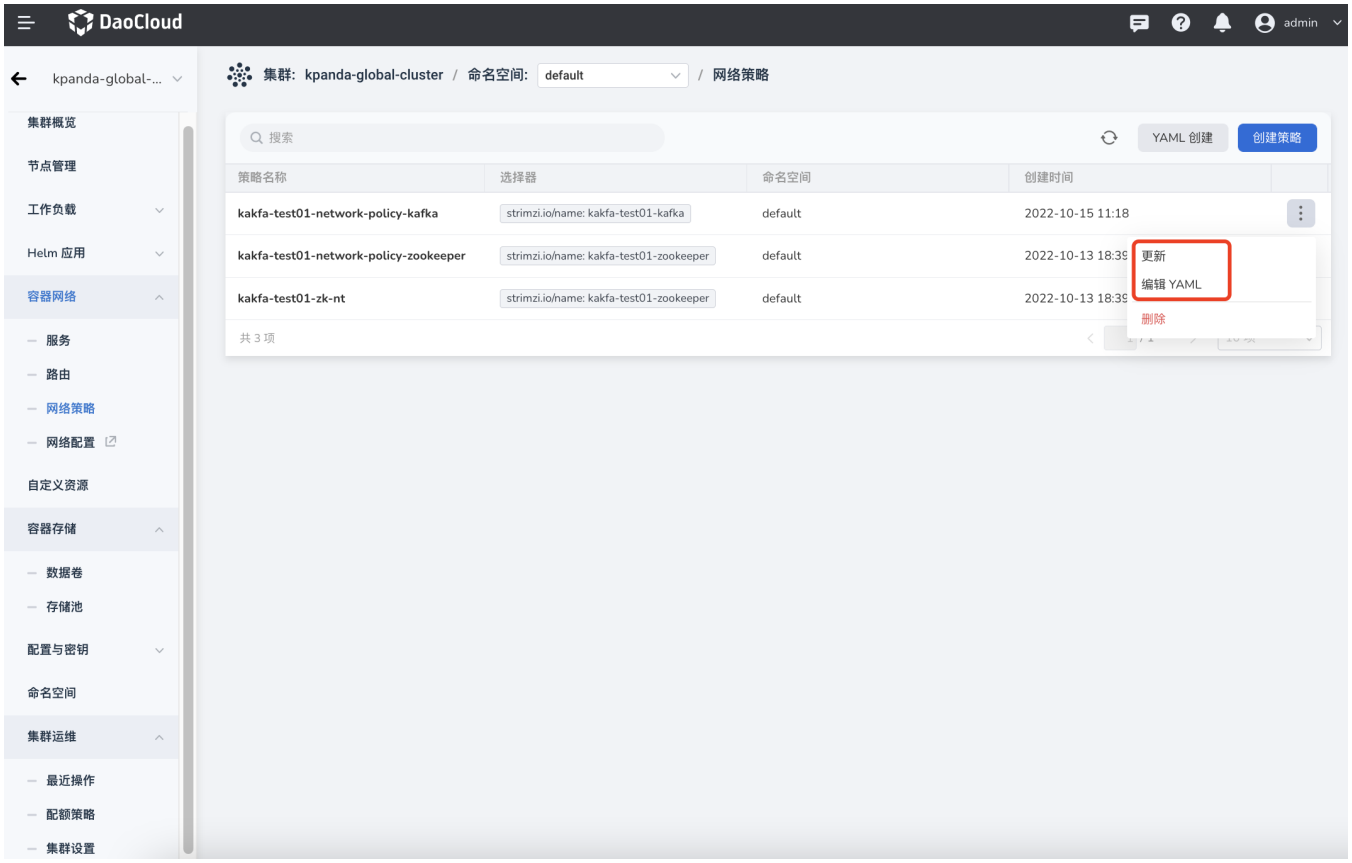
在关联实例页签下，支持查看实例监控、日志、容器列表、YAML 文件、事件等。



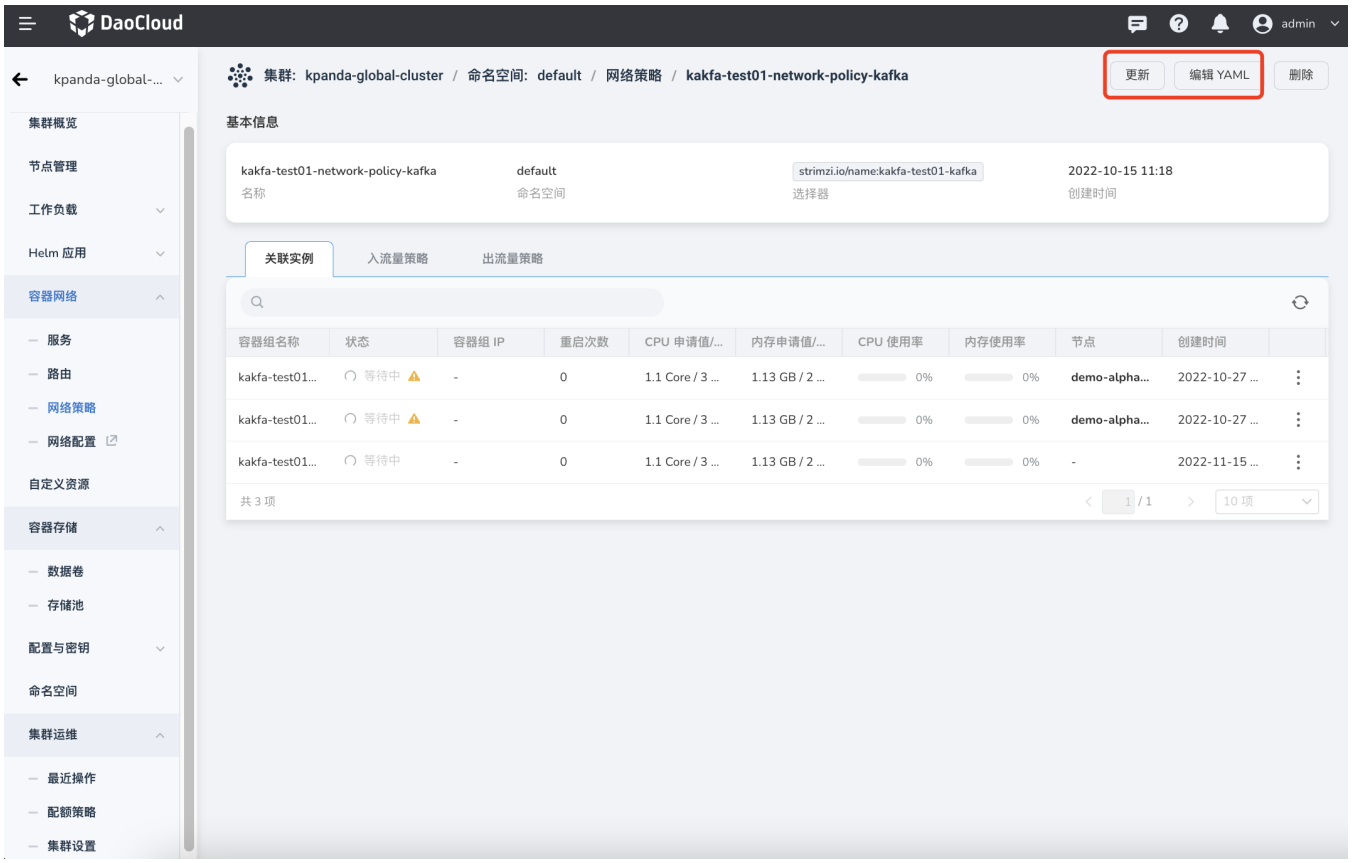
更新网络策略

有两种途径可以更新网络策略。支持通过表单或 YAML 文件更新网络策略。

• 在网络策略列表页面，找到需要更新的策略，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。



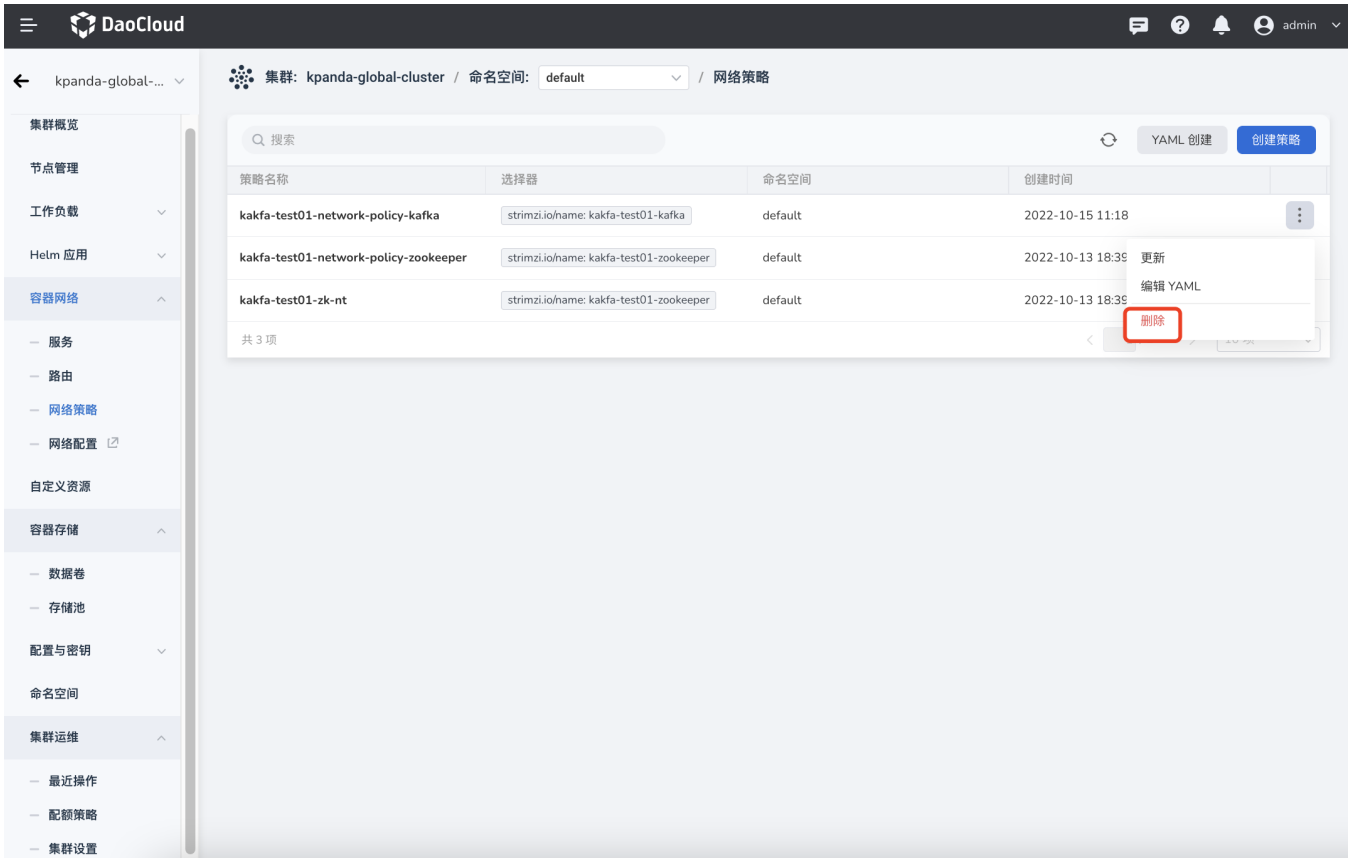
• 点击网络策略的名称，进入网络策略的详情页面后，在页面右上角选择 更新 即可通过表单更新，选择 编辑 **YAML** 即可通过 YAML 更新。



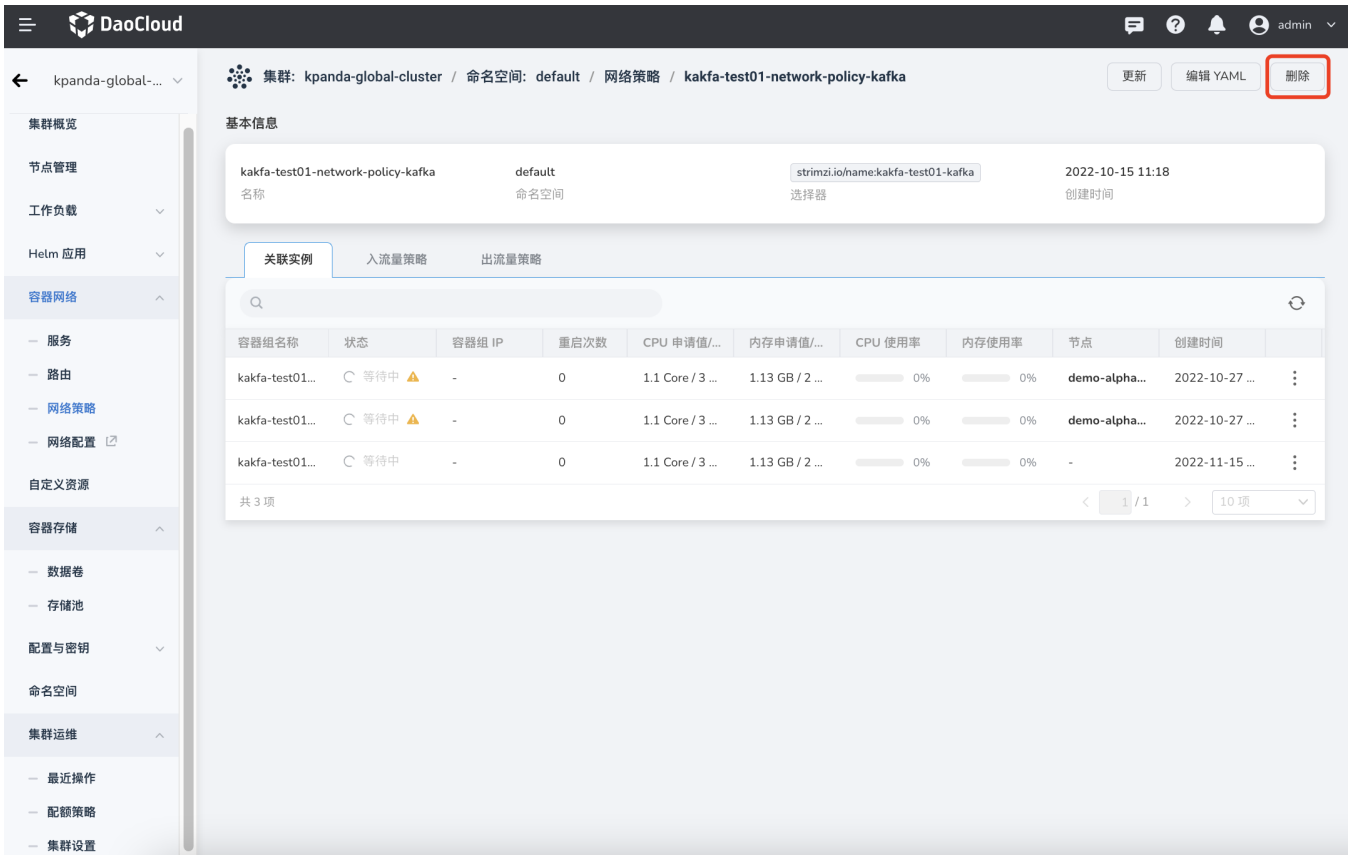
删除网络策略

有两种途径可以删除网络策略。支持通过表单或 YAML 文件更新网络策略。

• 在网络策略列表页面，找到需要更新的策略，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 删除。



• 点击网络策略的名称，进入网络策略的详情页面后，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 删除。



创建自定义资源 (CRD)

在 Kubernetes 中一切对象都被抽象为资源，如 Pod、Deployment、Service、Volume 等是 Kubernetes 提供的默认资源， 这为我们的日常运维和管理工作提供了重要支撑，但是在一些特殊的场景中，现有的预置资源并不能满足业务的需要， 因此我们希望去扩展 Kubernetes API 的能力，自定义资源（CustomResourceDefinition, CRD）正是基于这样的需求应运而生。

容器管理模块支持对自定义资源的界面化管理，主要功能如下：

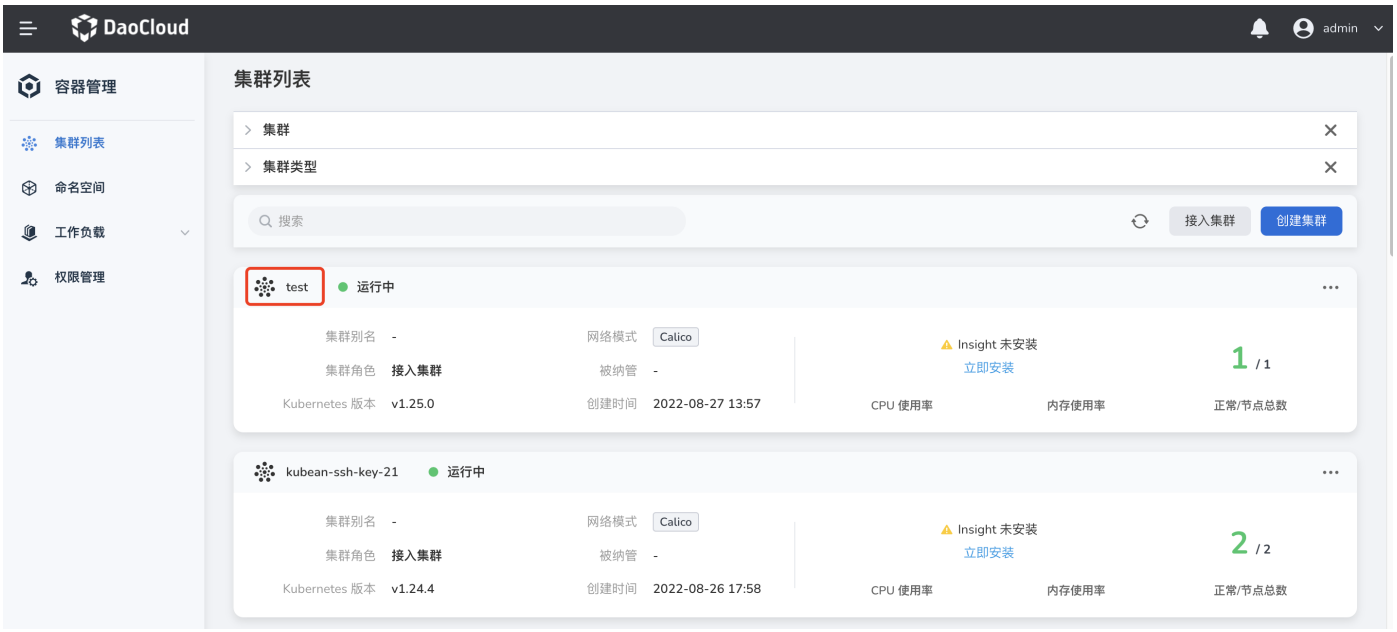
- 获取集群下自定义资源列表和详细信息
- 基于 YAML 创建自定资源
- 基于 YAML 创建自定义资源示例 CR（Custom Resource）
- 删除自定义资源

前提条件

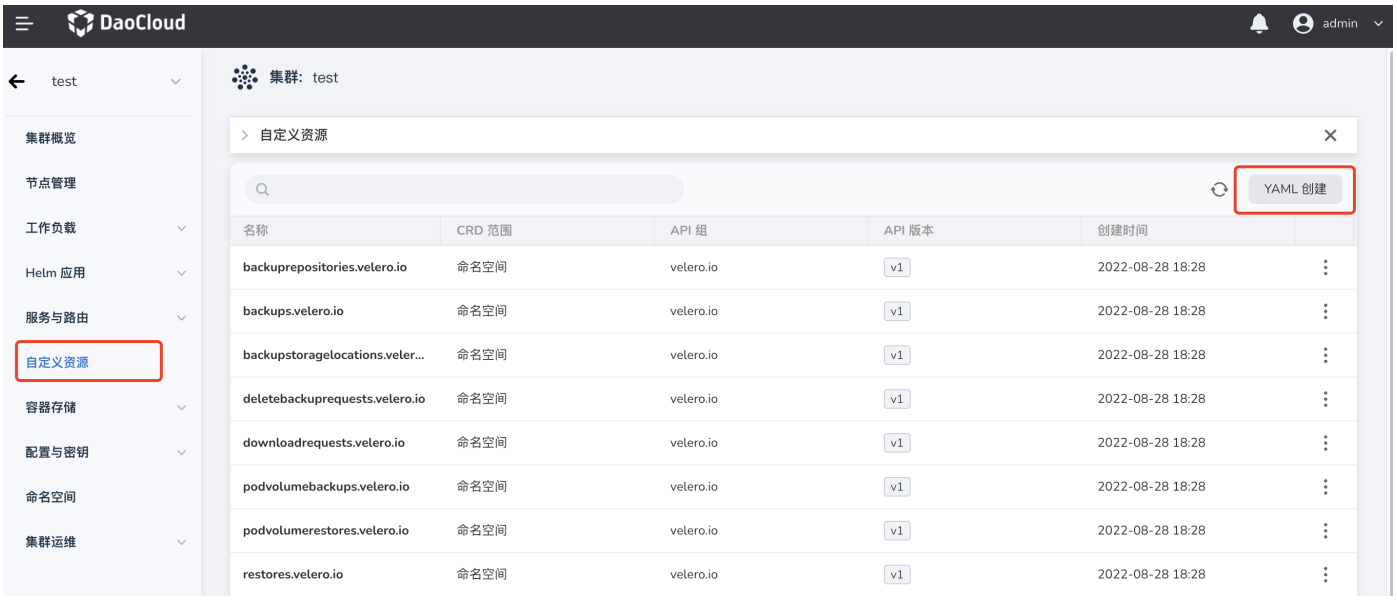
已完成一个命名空间的创建、用户的创建， 并将用户授权为 Cluster Admin 角色 ，详情可参考命名空间授权

通过 YAML 创建自定义资源

1. 点击一个集群名称，进入 集群详情 。

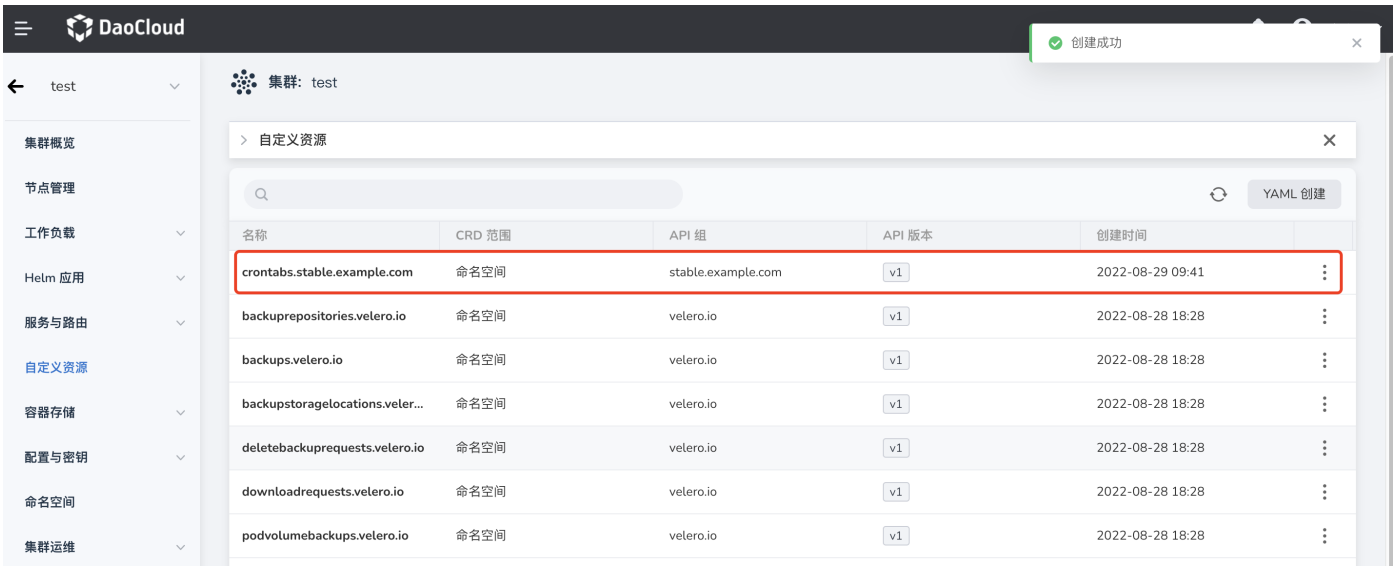


2. 在左侧导航栏，点击 自定义资源 ，点击右上角 **YAML** 创建 按钮。



3. 在 **YAML** 创建 页面中，填写 YAML 语句后，点击 确定 。

4. 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。



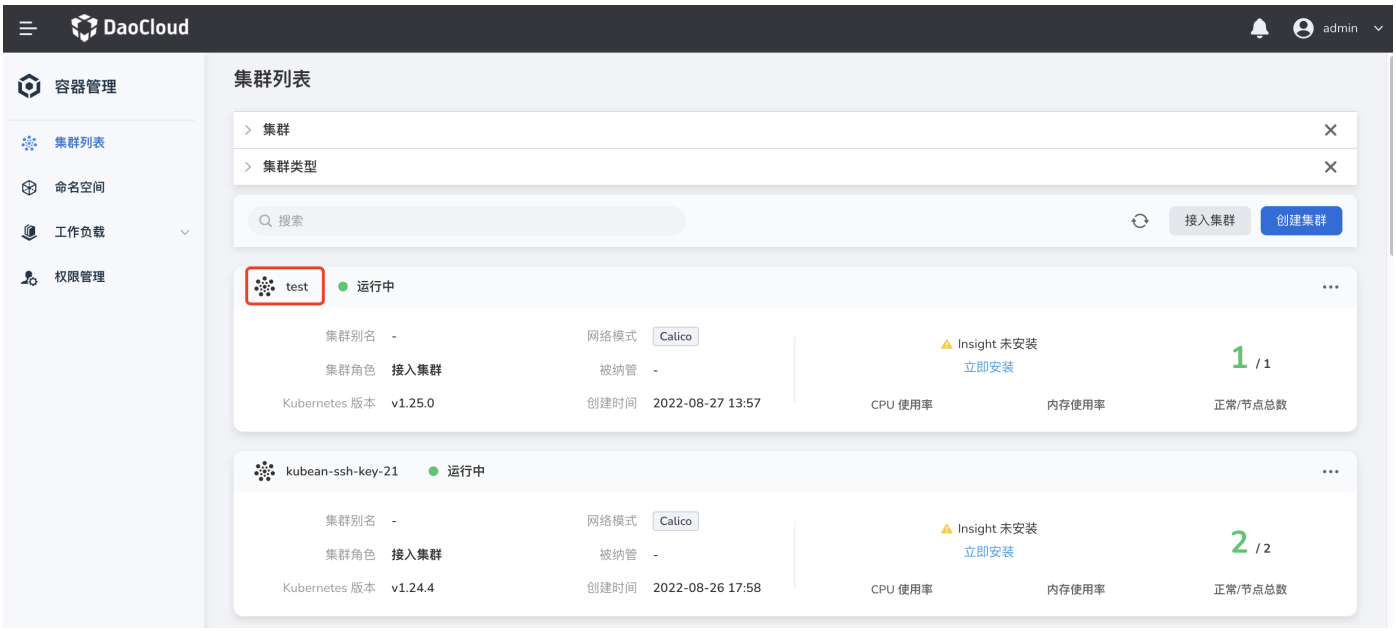
自定义资源示例：

CRD example

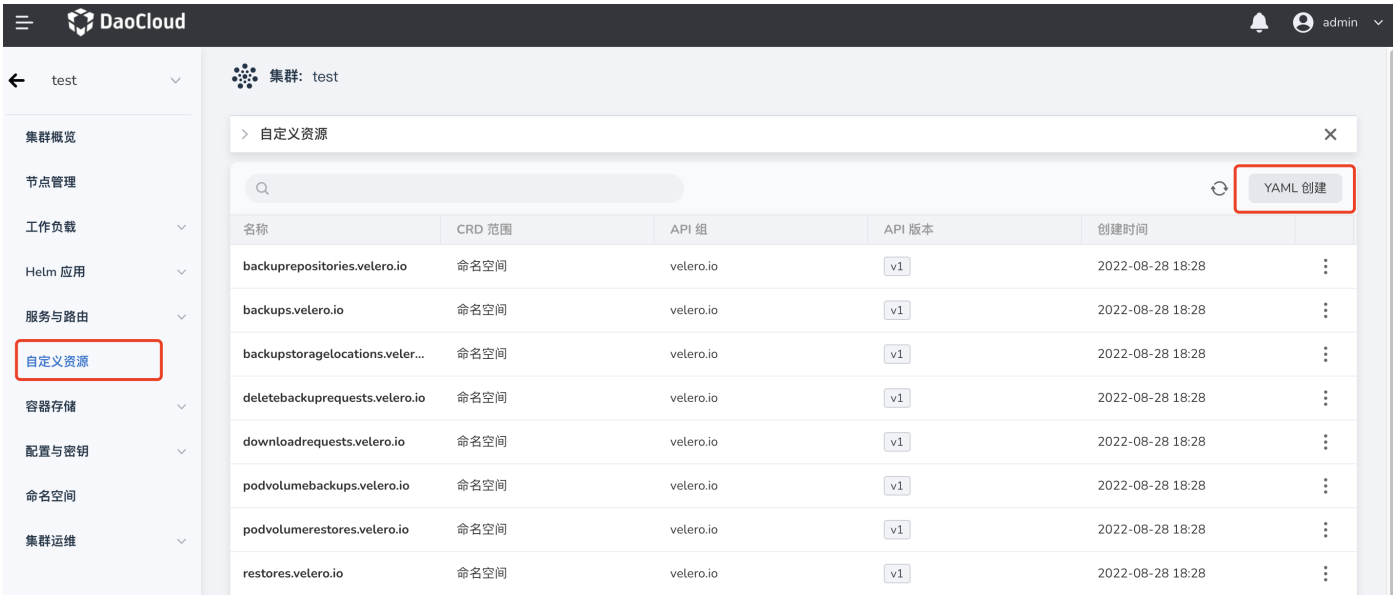
```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            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 创建自定义资源示例

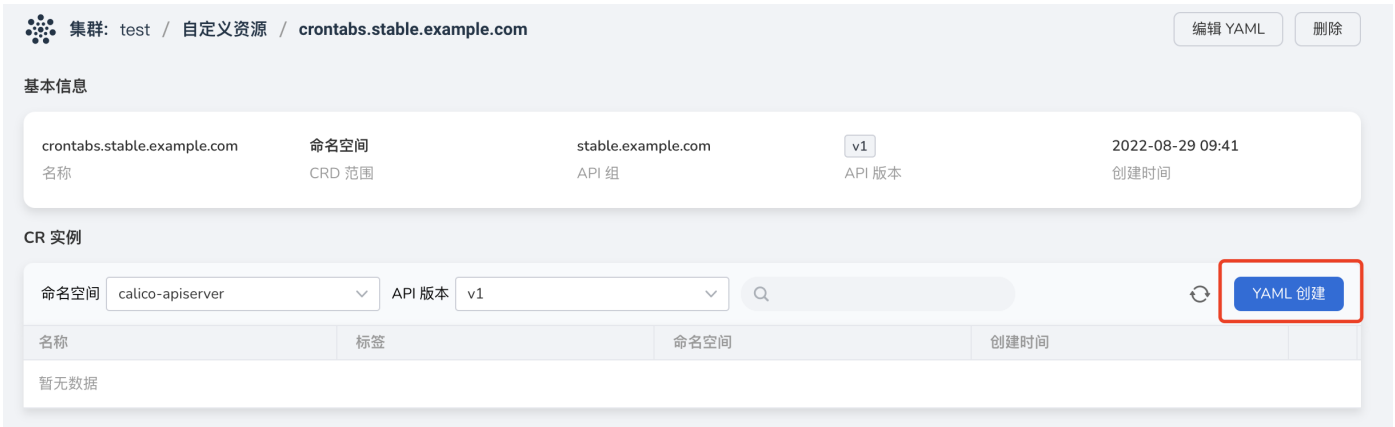
1. 点击一个集群名称，进入 集群详情 。



2. 在左侧导航栏，点击 自定义资源 ，点击右上角 **YAML 创建** 按钮。



3. 点击名为 `crontabs.stable.example.com` 的自定义资源，进入详情，点击右上角 **YAML 创建** 按钮。



4. 在 **YAML 创建** 页面中，填写 YAML 语句后，点击 确定 。

5. 返回 `crontabs.stable.example.com` 的详情页面，即可查看刚刚创建的名为 **my-new-cron-object** 的自定义资源。

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
```

容器存储

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。

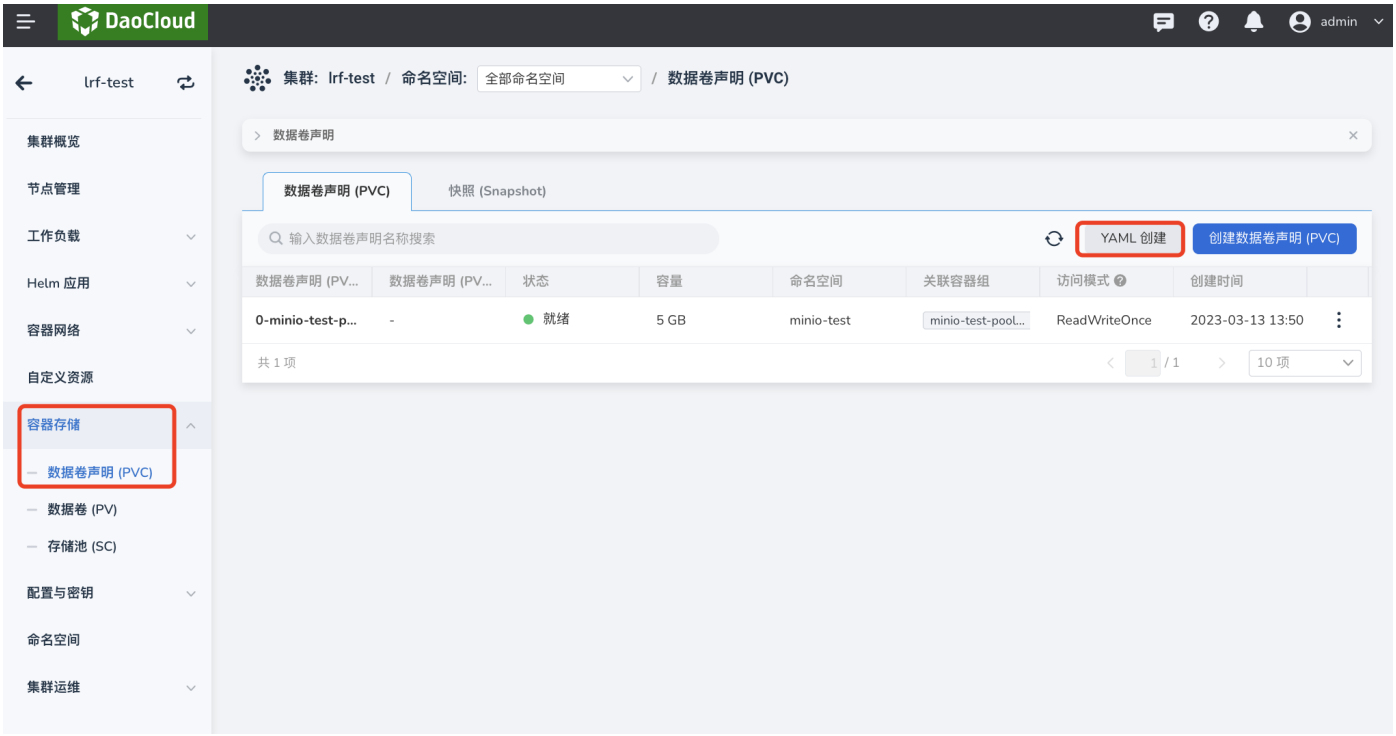
创建数据卷声明

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。

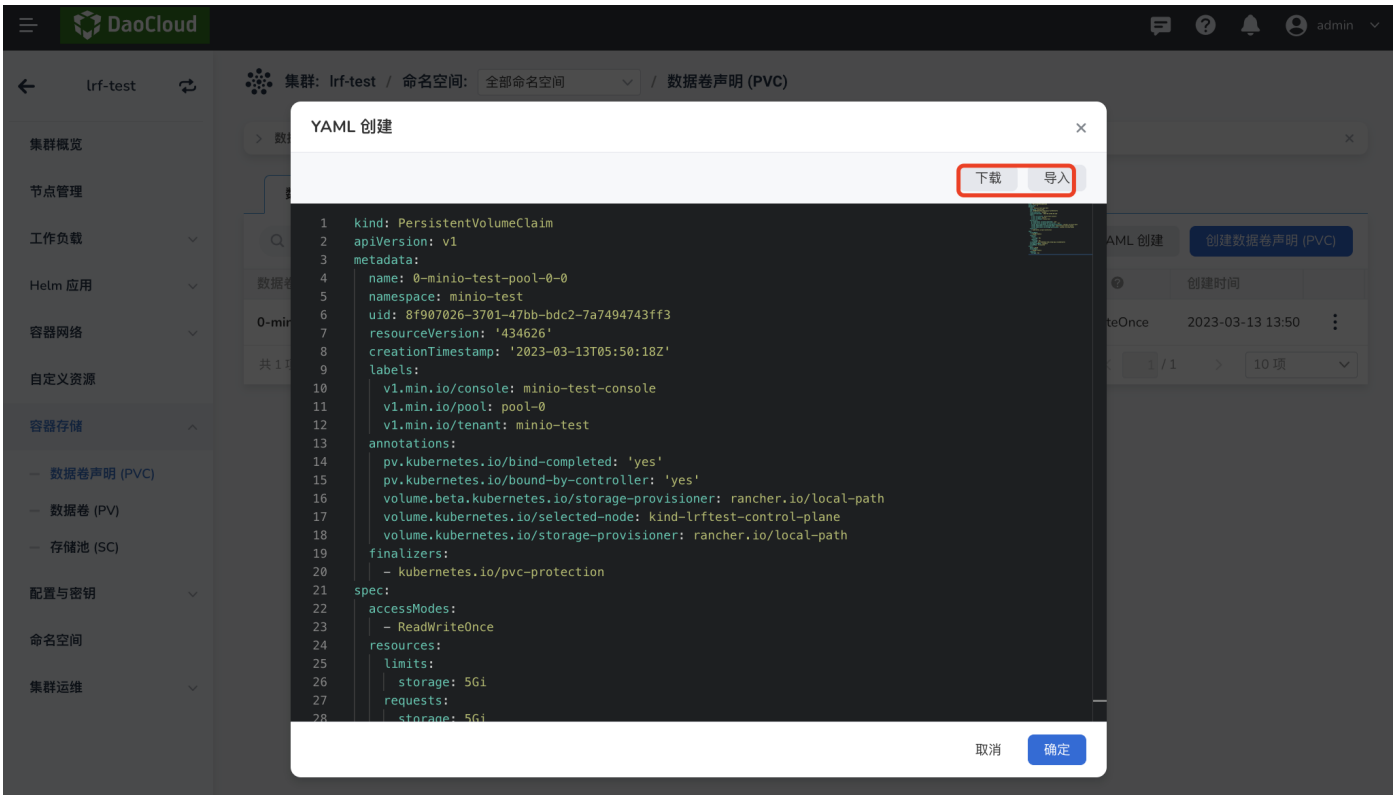
YAML 创建

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 。



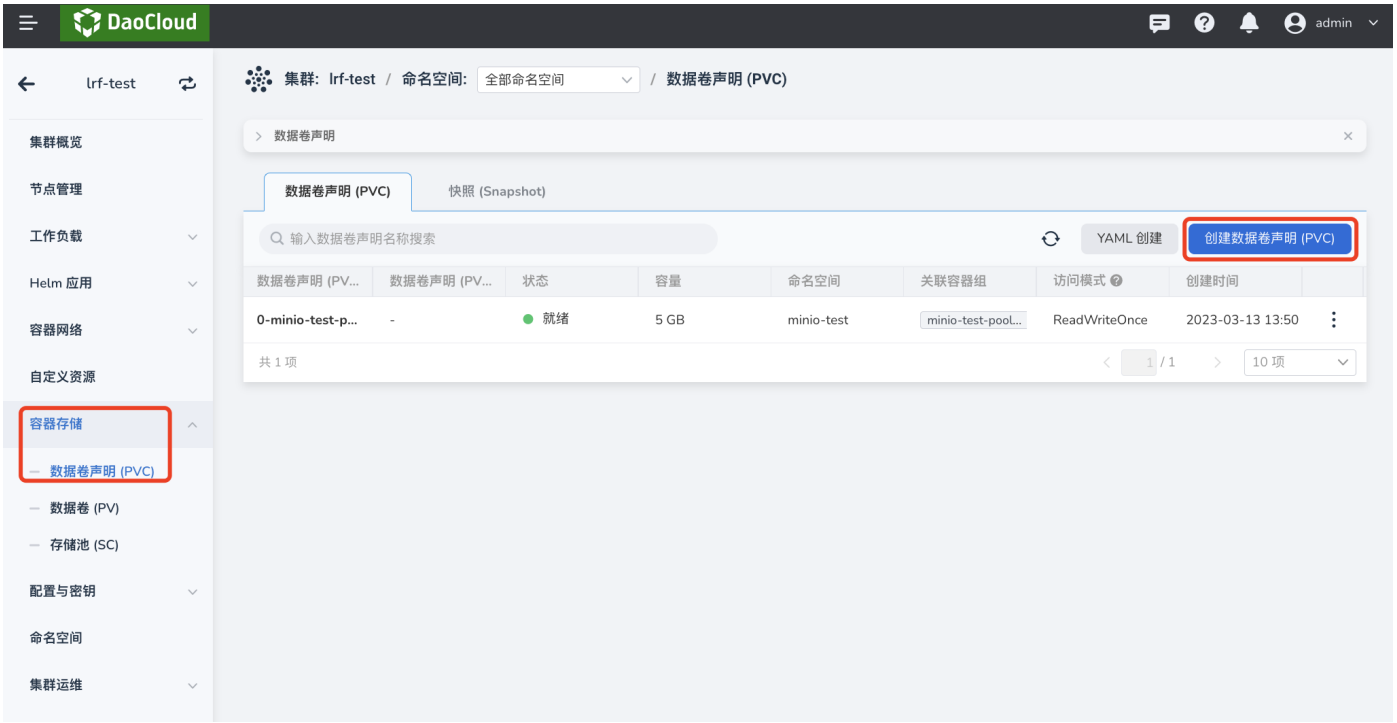
2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击 确定 。

   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。



表单创建

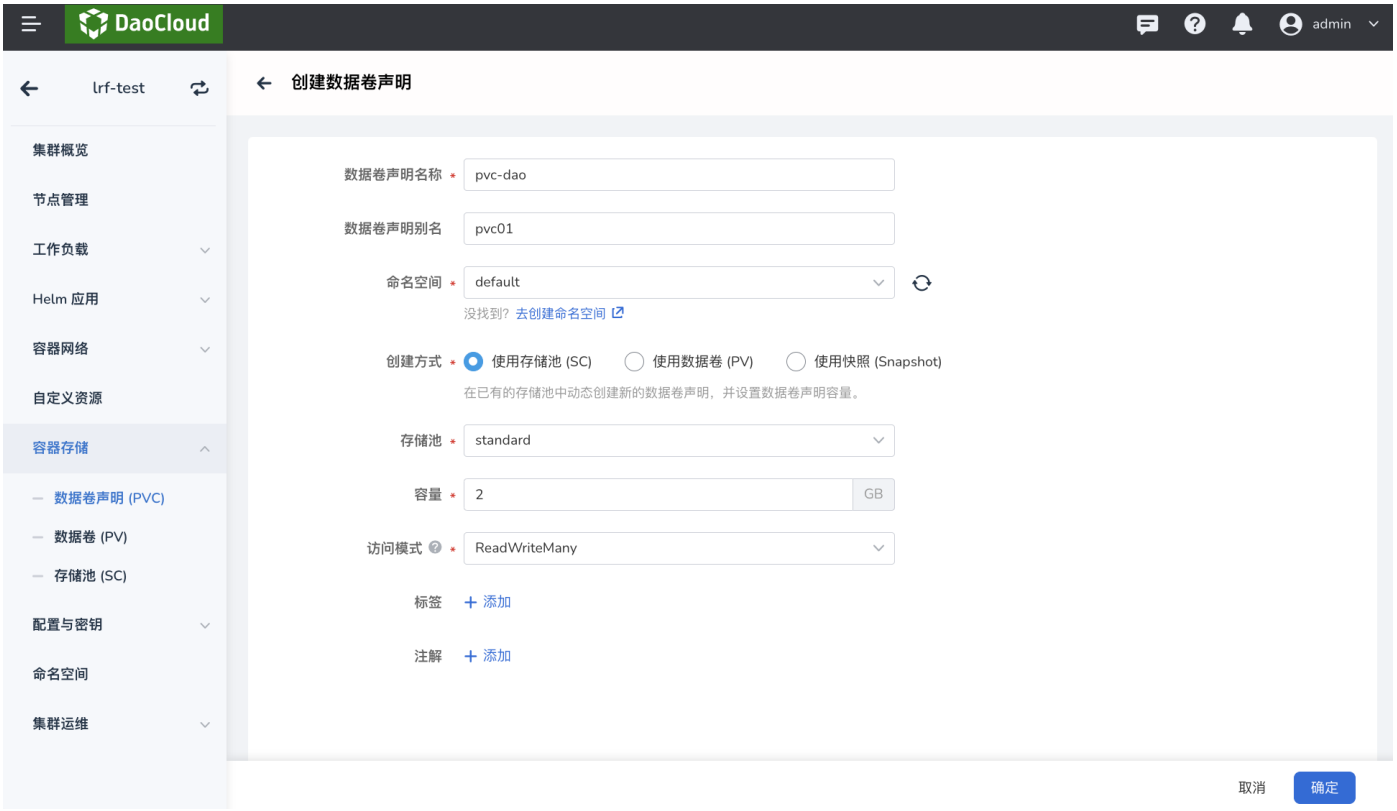
1. 在集群列表中点击目标集群的名称，然后在左侧导航栏点击 容器存储 -> 数据卷声明 **(PVC)** -> 创建数据卷声明 **(PVC)** 。



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。

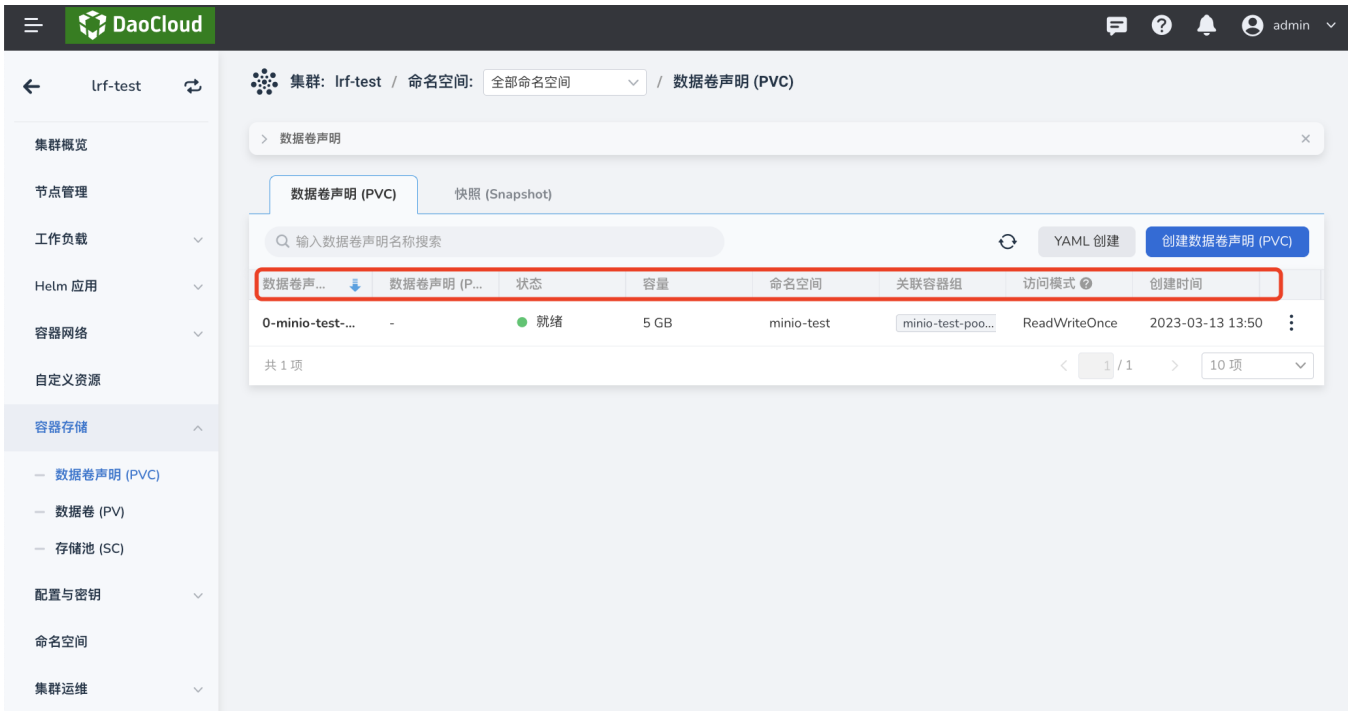
  基于快照创建时无法修改数据卷声明的容量，可以在创建完成后再进行修改。
- 选择创建方式之后，在下拉列表中选择想要使用的存储池/数据卷/快照。
- 访问模式：
- ReadWriteOnce，数据卷声明可以被一个节点以读写方式挂载。
- ReadWriteMany，数据卷声明可以被多个节点以读写方式挂载。
- ReadOnlyMany，数据卷声明可以被多个节点以只读方式挂载。
- ReadWriteOncePod，数据卷声明可以被单个 Pod 以读写方式挂载。



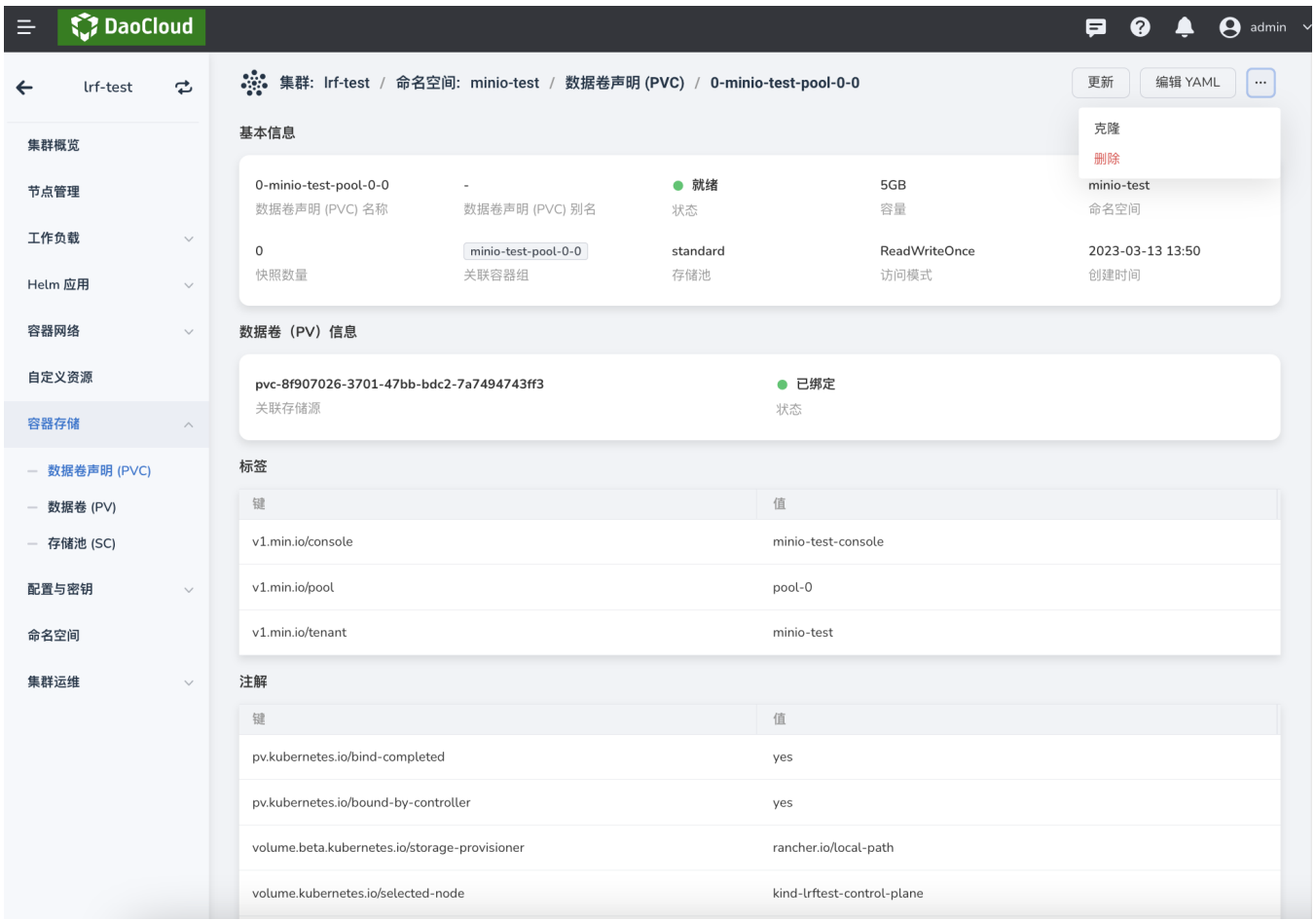
查看数据卷声明

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。

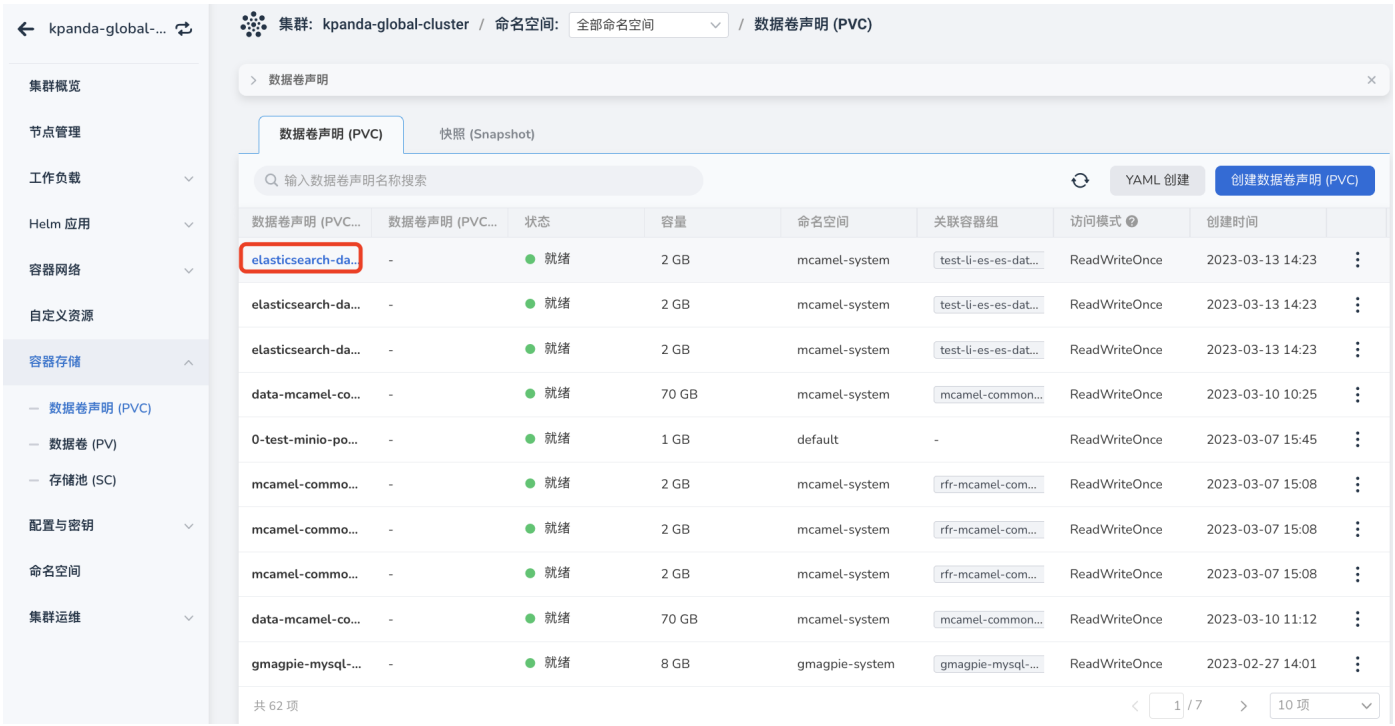


- 点击数据卷声明的名称，可以查看该数据卷声明的基本配置、存储池信息、标签、注解等信息。

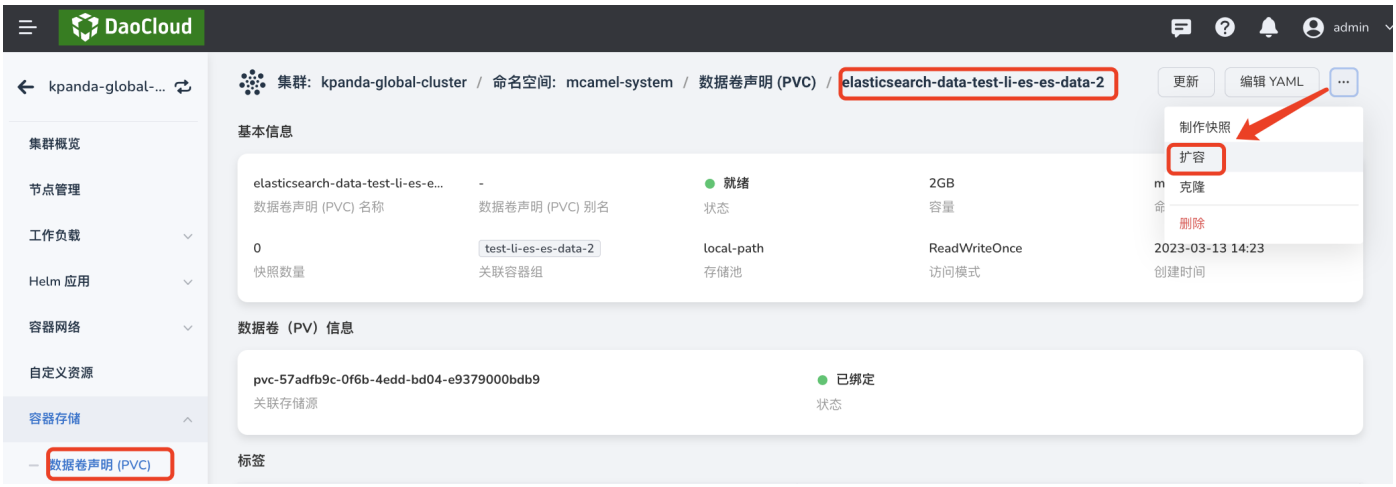


扩容数据卷声明

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。



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 。



3. 输入目标容量，然后点击 确定 。

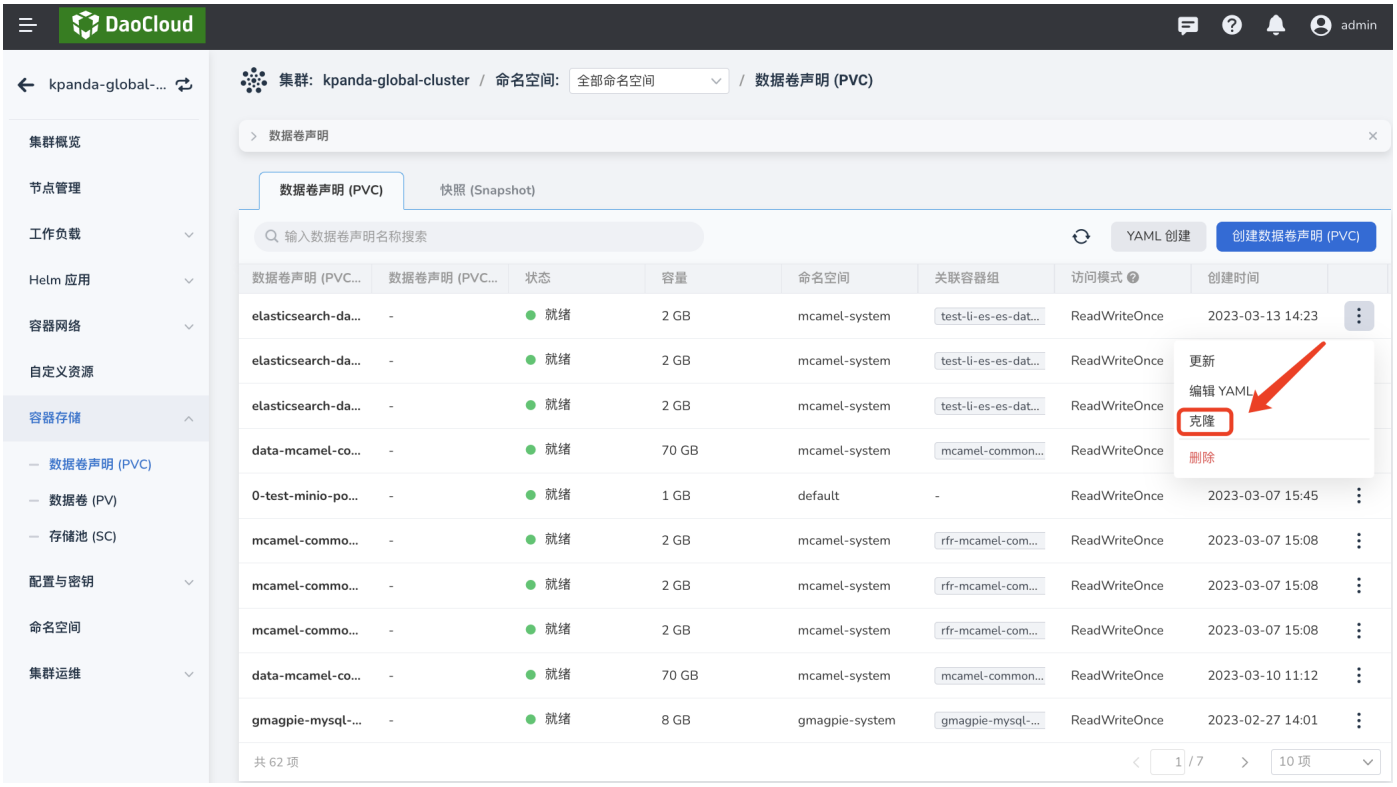
克隆数据卷声明

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。

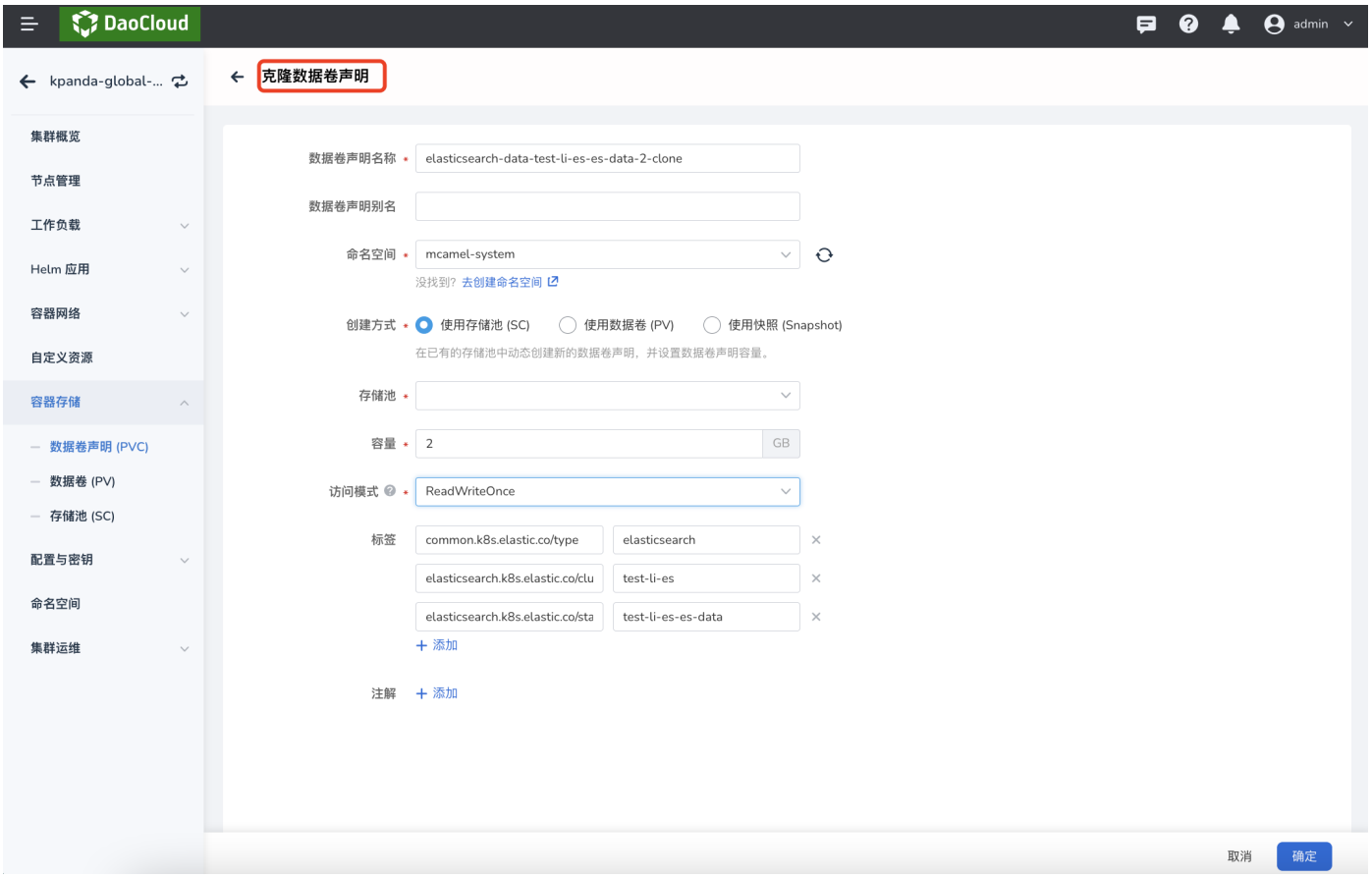
1. 进入克隆页面

- 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 。



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 。

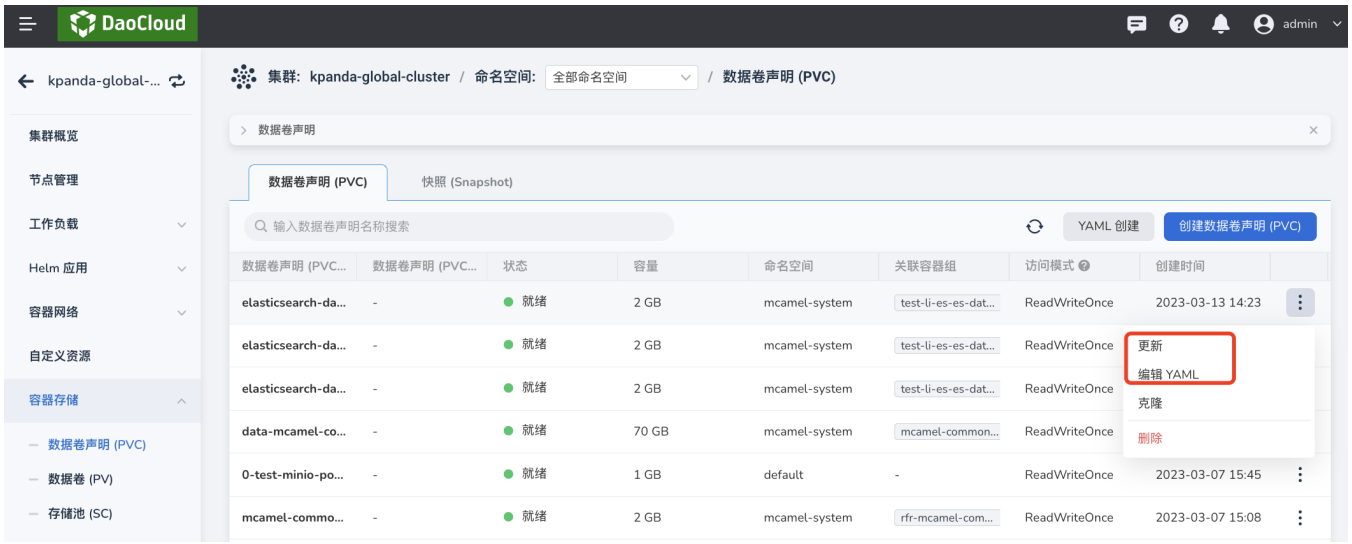


更新数据卷声明

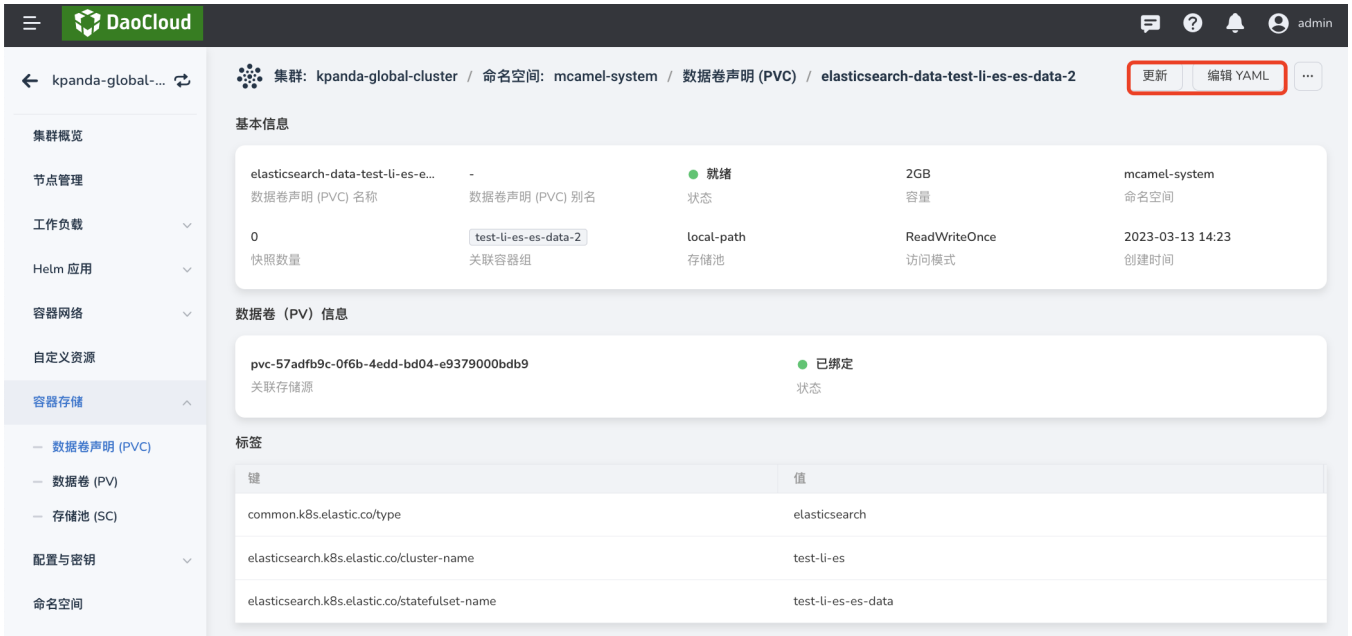
有两种途径可以更新数据卷声明。支持通过表单或 YAML 文件更新数据卷声明。

> **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。



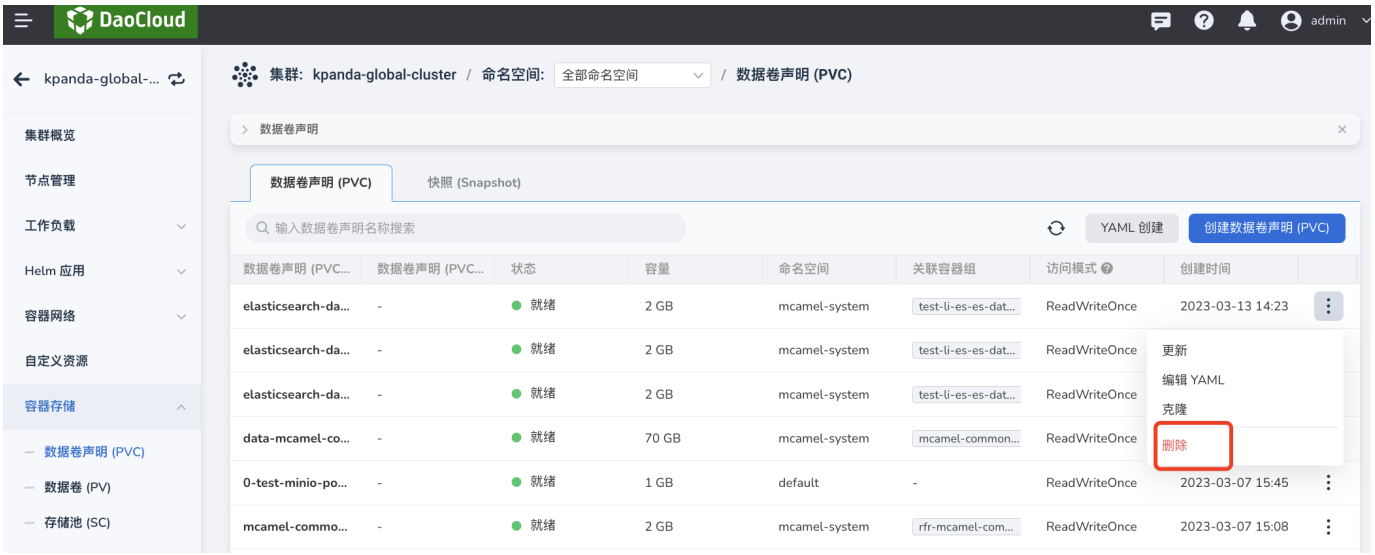
- 点击数据卷声明的名称，进入数据卷声明的详情页面后，在页面右上角选择 更新 即可通过表单更新，选择 编辑 **YAML** 即可通过 YAML 更新。



删除数据卷声明

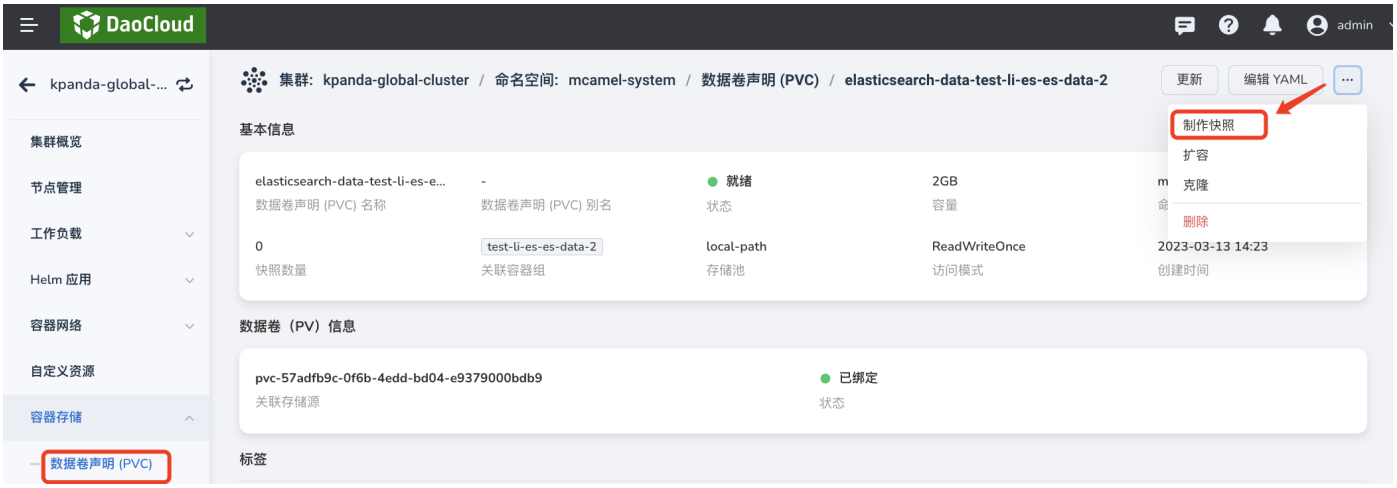
在数据卷声明列表页面，找到需要删除的数据，在右侧的操作栏下选择 删除 。

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 。

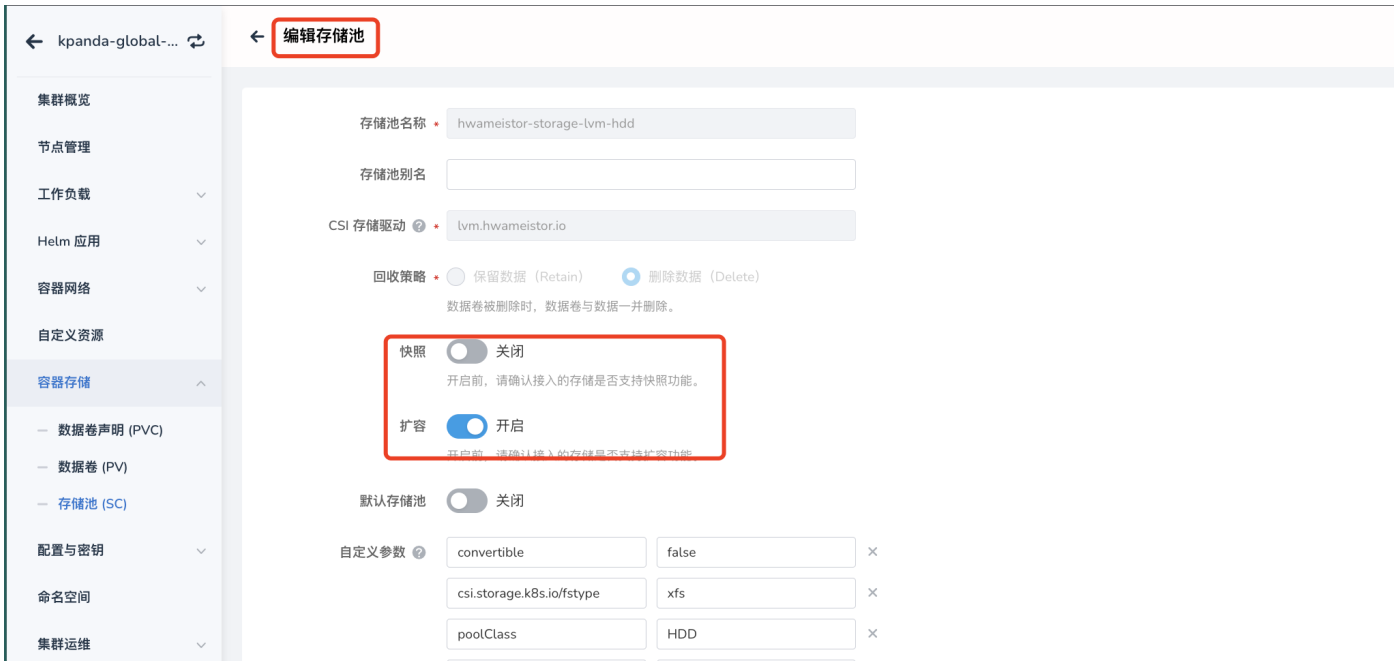


常见问题

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。



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

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。

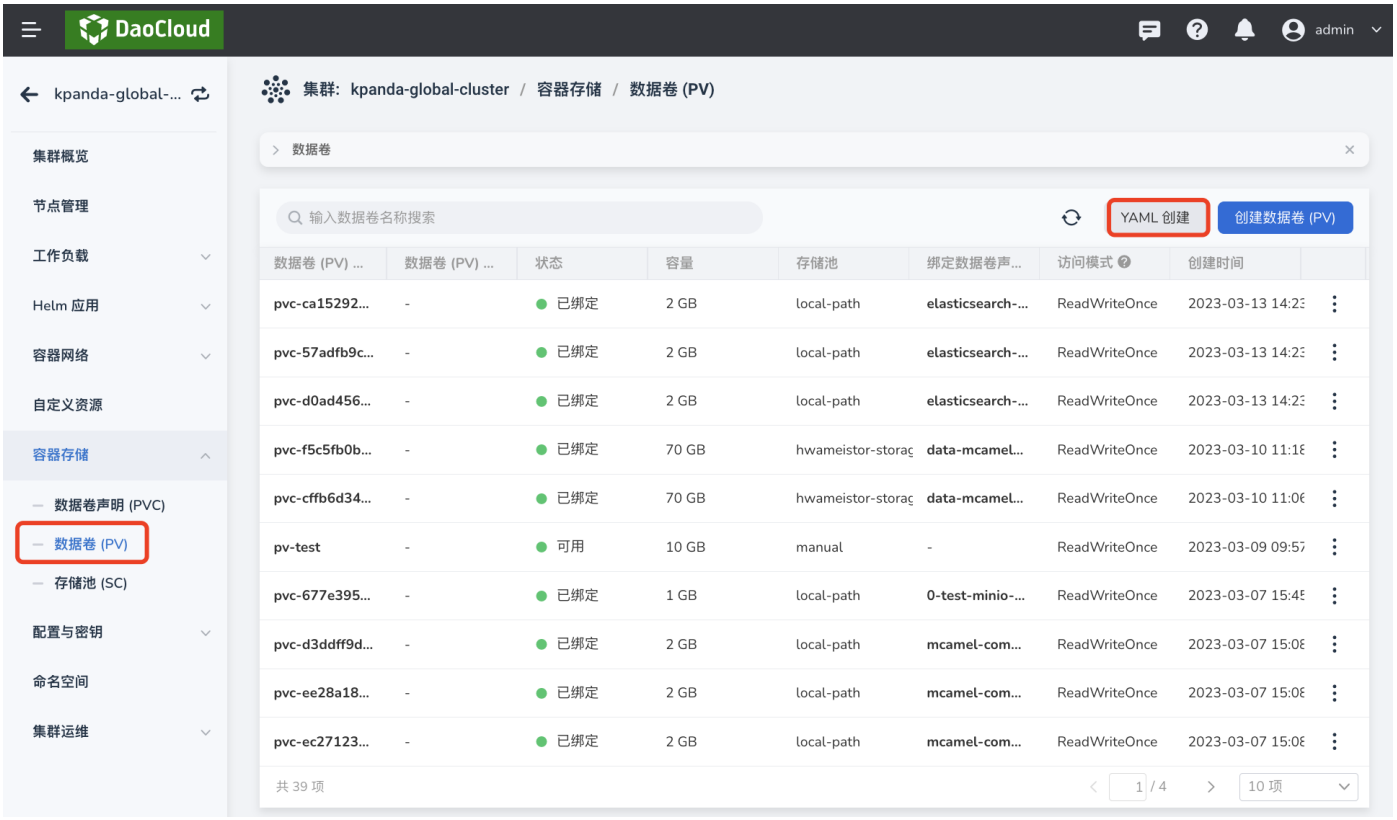
创建数据卷

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。

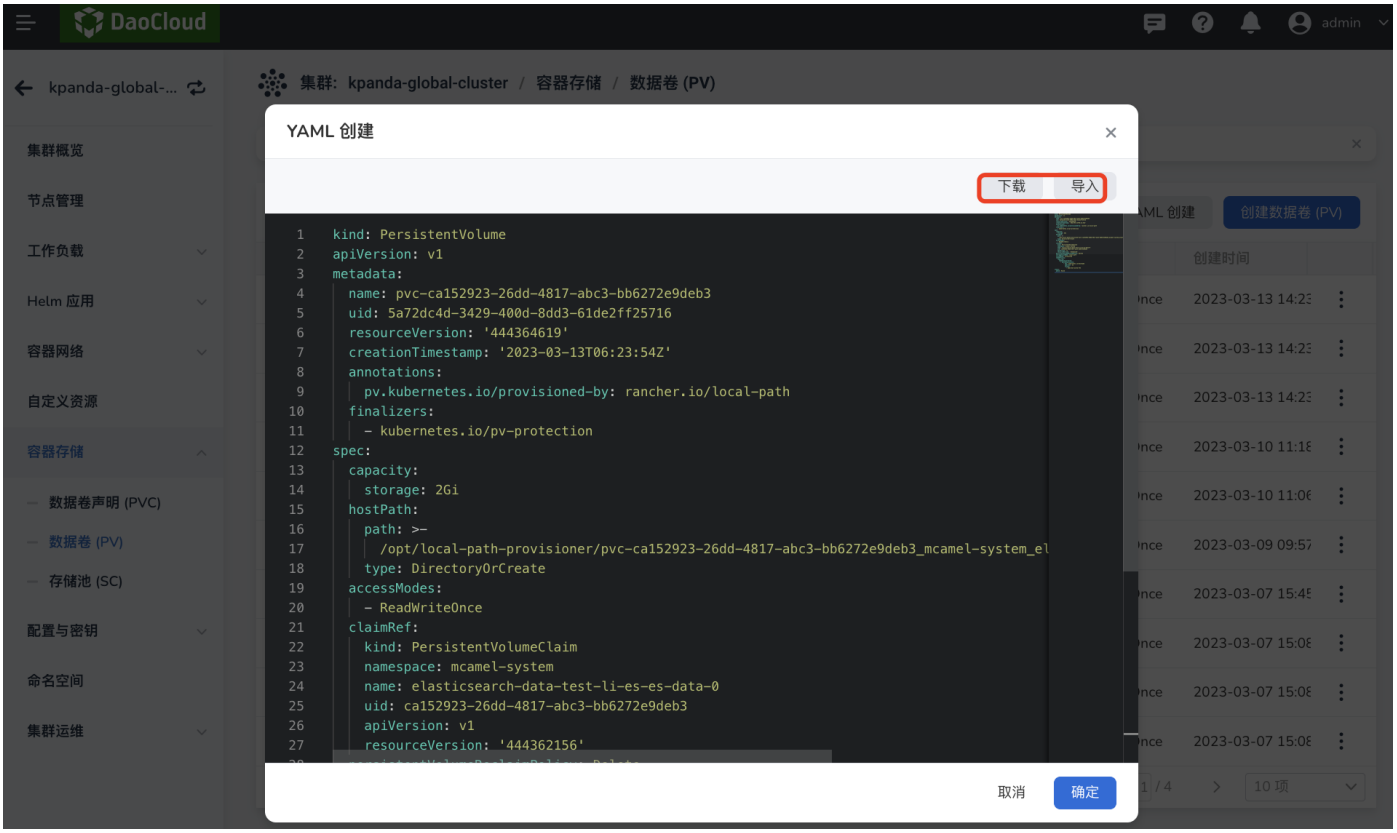
YAML 创建

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 。



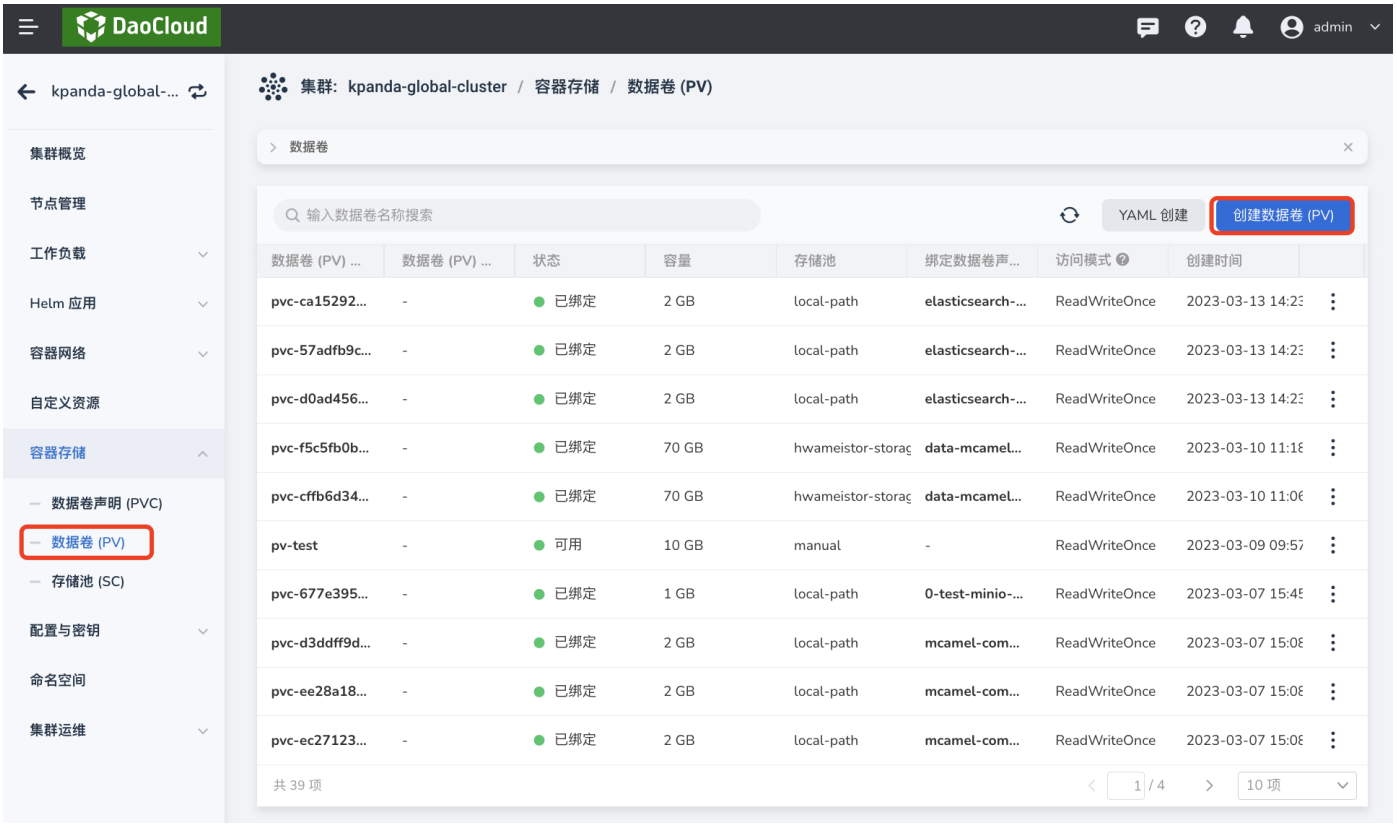
2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击 确定 。

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。



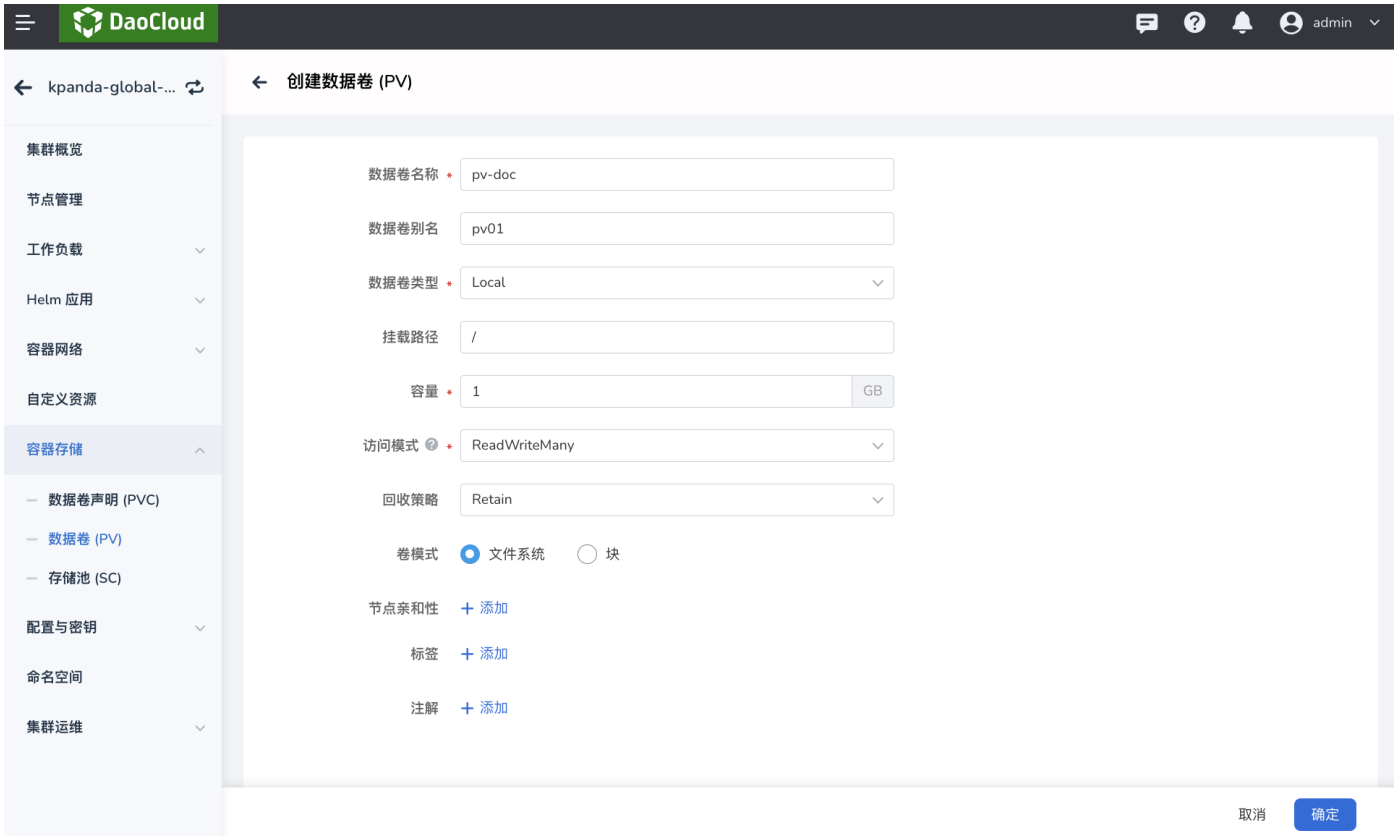
表单创建

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)** 。



2. 填写基本信息。

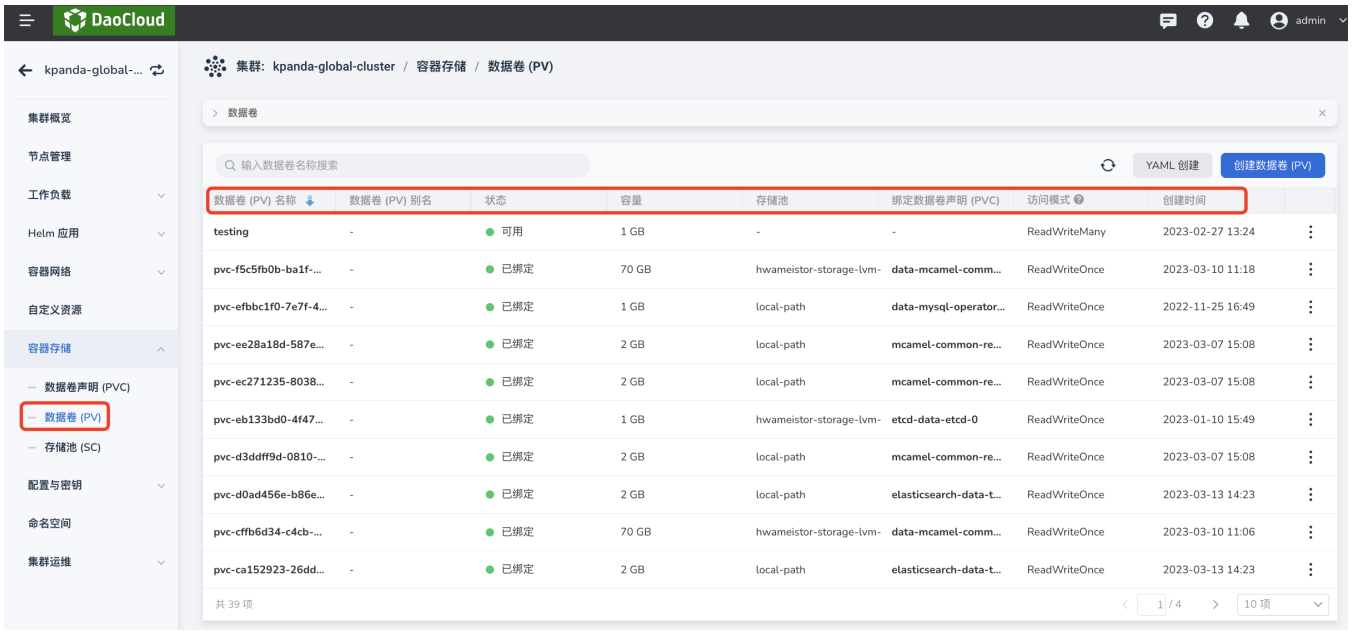
- 数据卷名称、数据卷类型、挂载路径、卷模式、节点亲和性在创建之后不可更改。
- 数据卷类型：有关卷类型的详细介绍，可参考 Kubernetes 官方文档卷。
- Local：将 Node 节点的本地存储包装成 PVC 接口，容器直接使用 PVC 而无需关注底层的存储类型。Local 卷不支持动态配置数据卷，但支持配置节点亲和性，可以限制能从哪些节点上访问该数据卷。
- HostPath：使用 Node 节点的文件系统上的文件或目录作为数据卷，不支持基于节点亲和性的 Pod 调度。
- 挂载路径：将数据卷挂载到容器中的某个具体目录下。
- 访问模式：
- ReadWriteOnce：数据卷可以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：



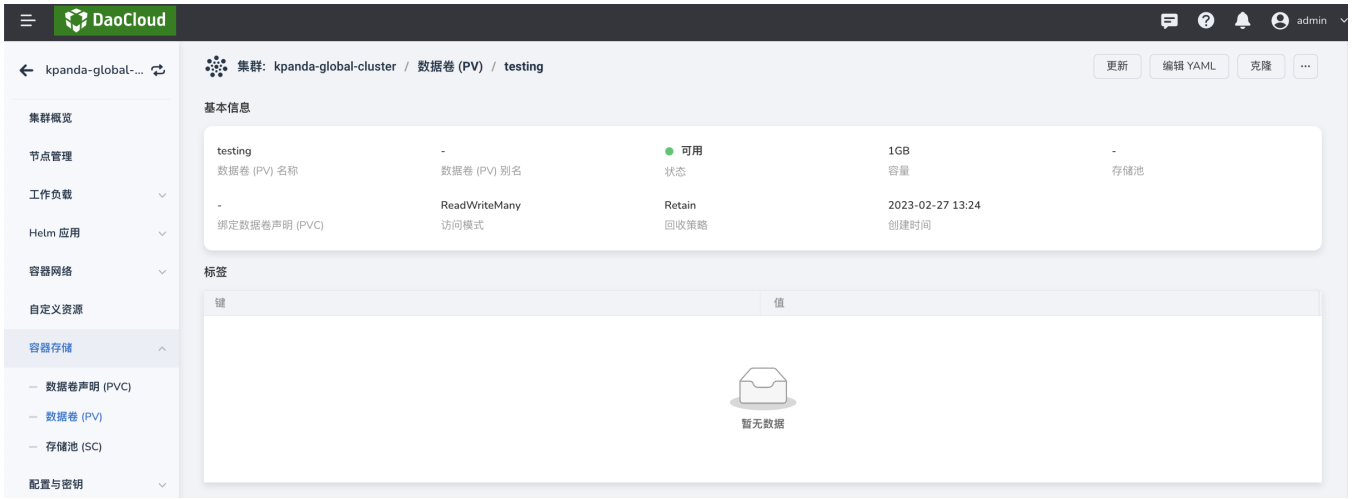
查看数据卷

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。



- 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。



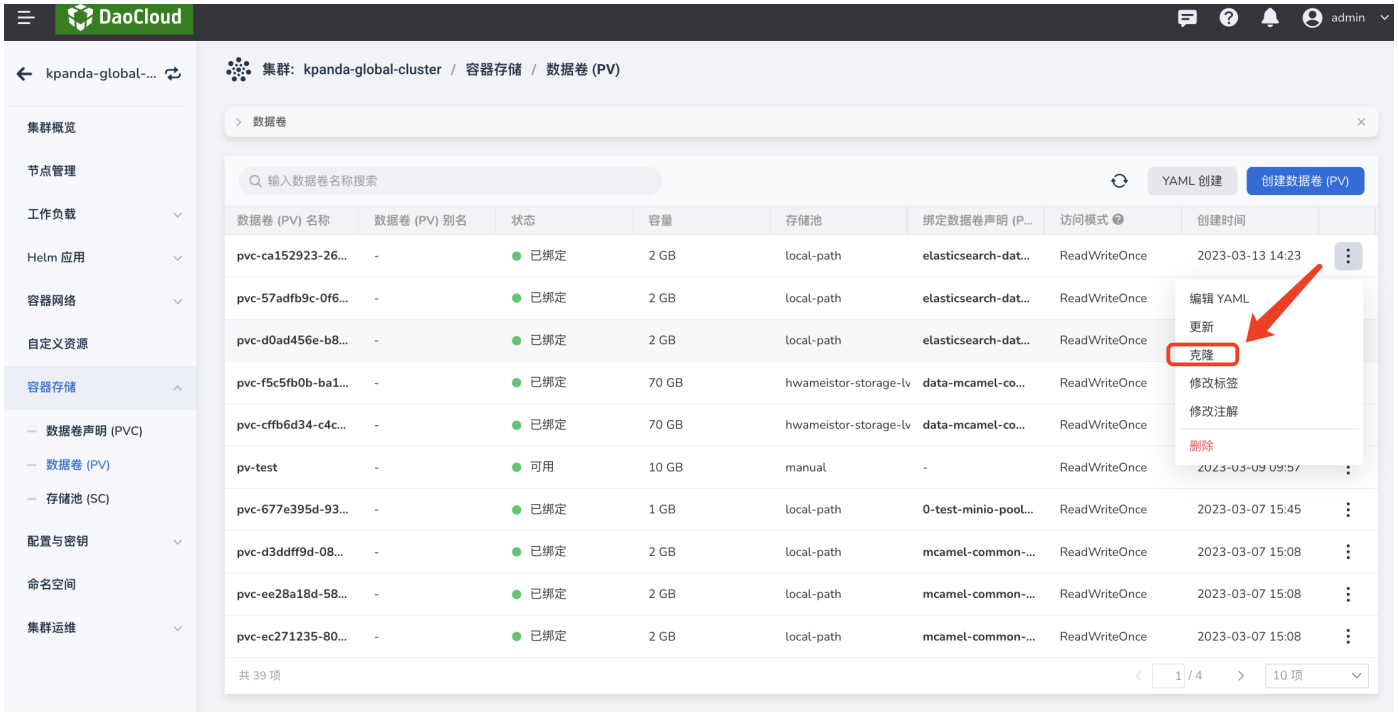
克隆数据卷

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

   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 。



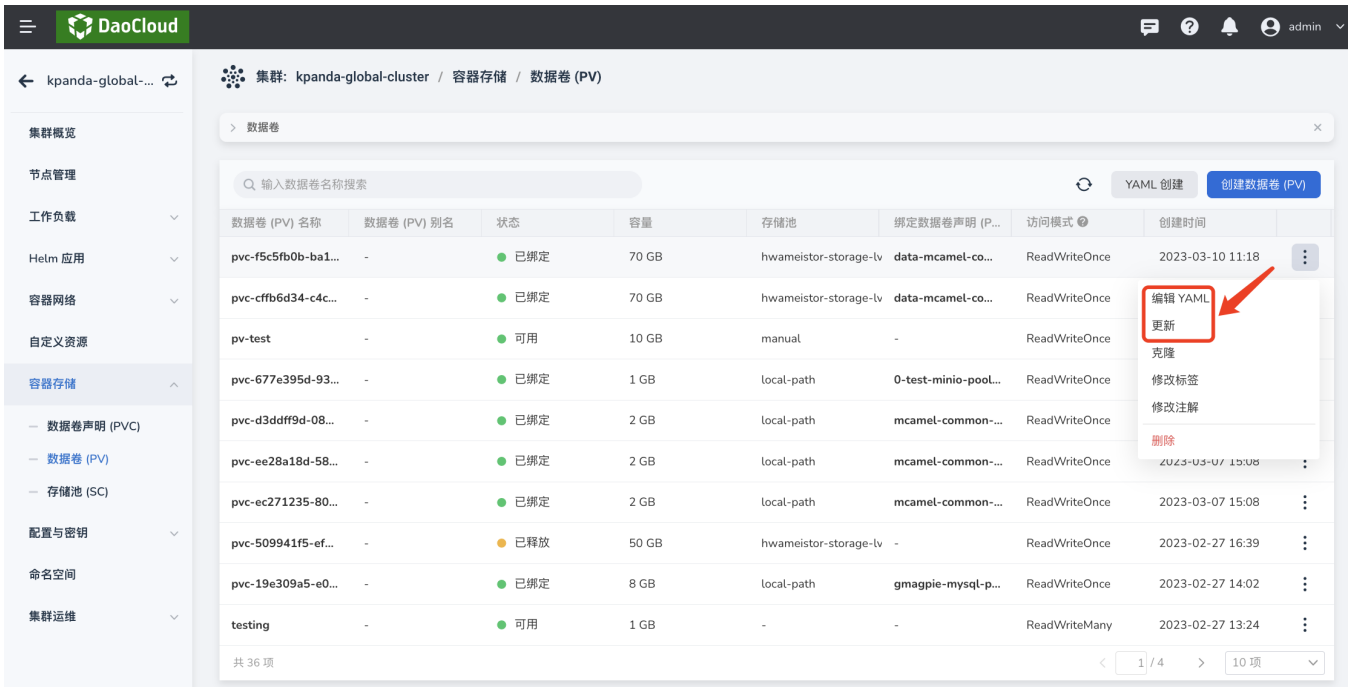
2. 直接使用原配置，或者按需进行修改，然后在页面底部点击 确定 。

更新数据卷

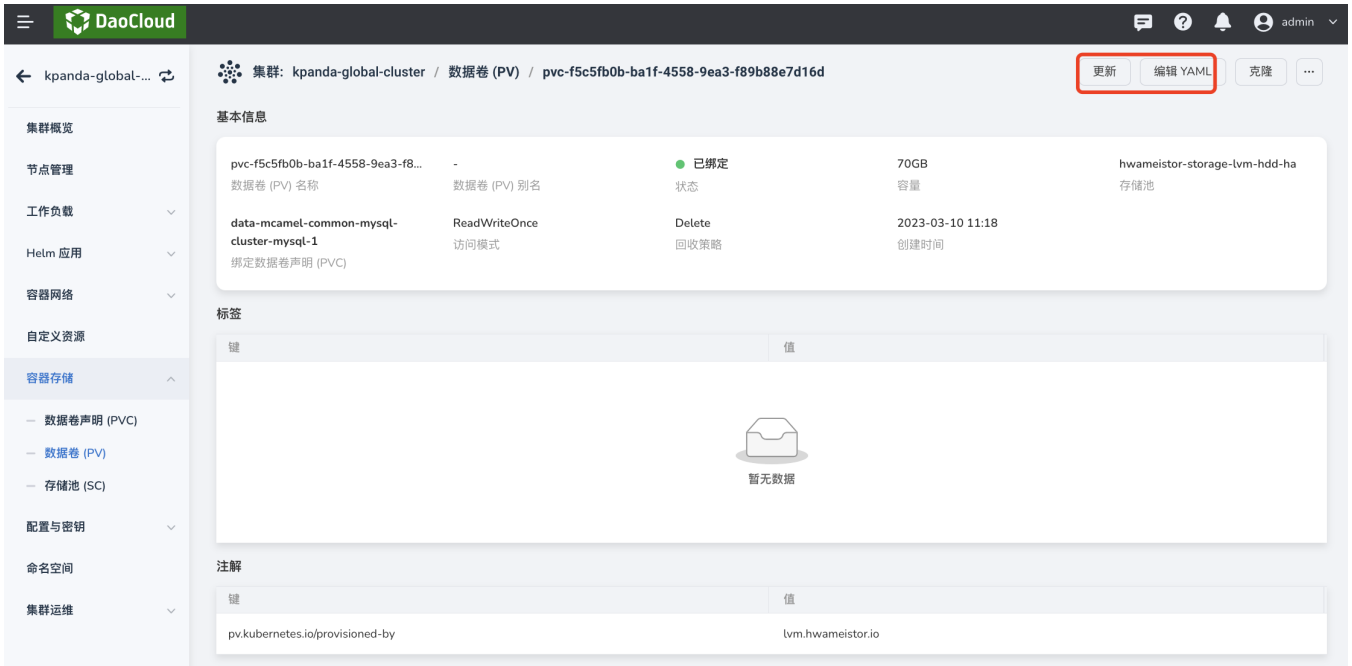
有两种途径可以更新数据卷。支持通过表单或 YAML 文件更新数据卷。

> **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。



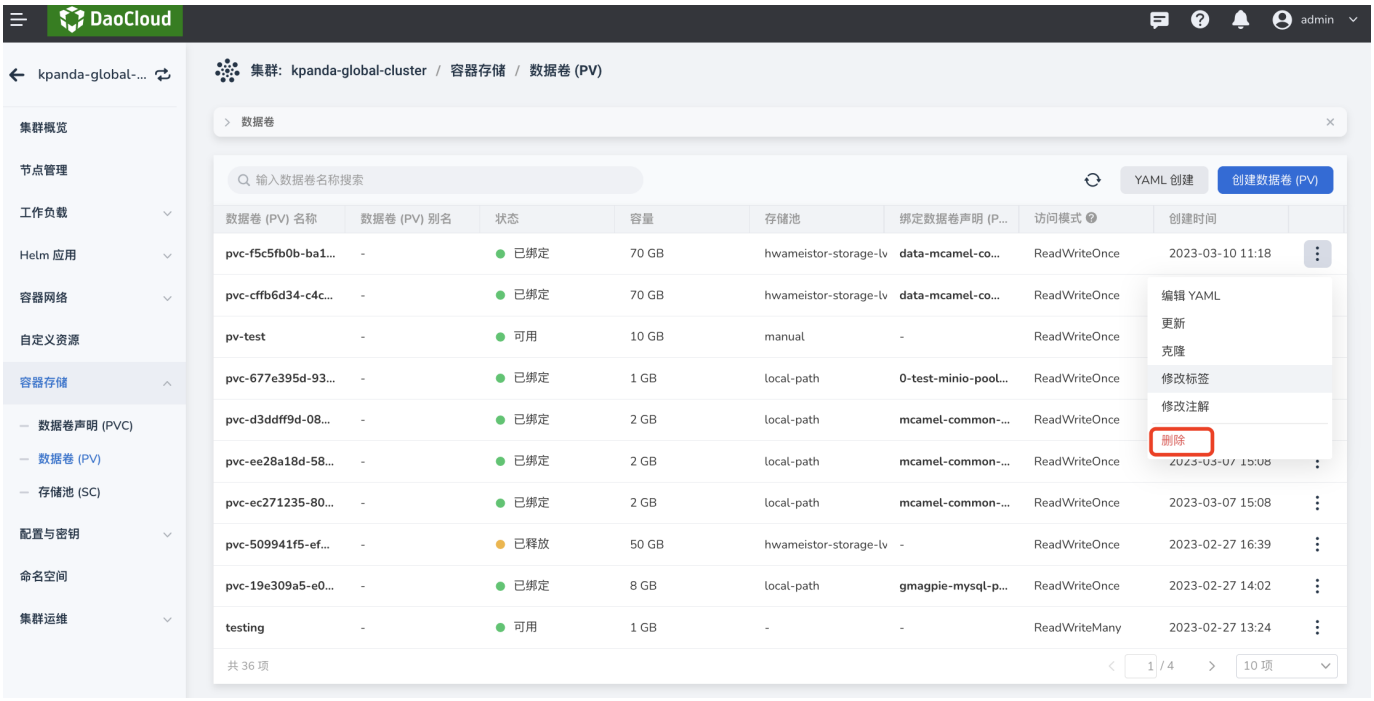
- 点击数据卷的名称，进入数据卷的详情页面后，在页面右上角选择 更新 即可通过表单更新，选择 编辑 **YAML** 即可通过 YAML 更新。



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

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。

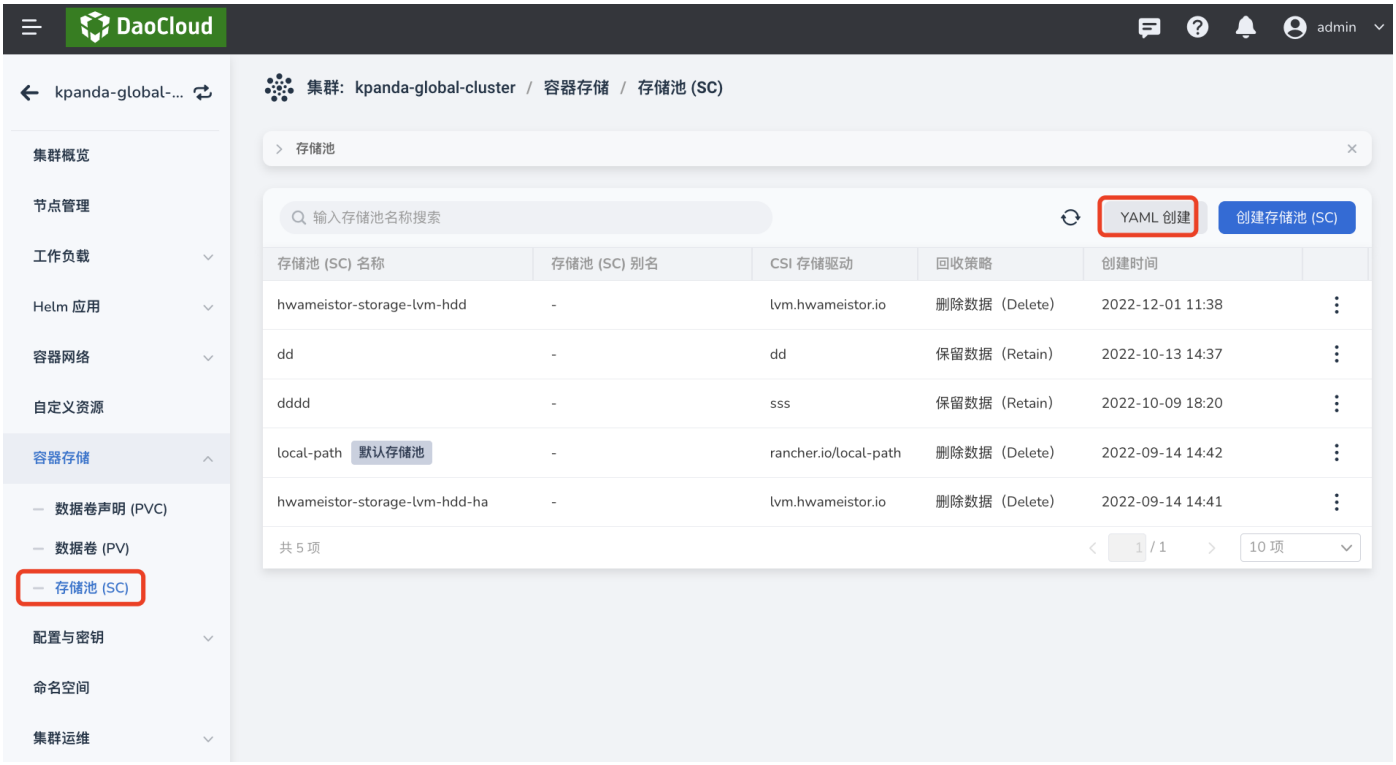
创建存储池(SC)

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。

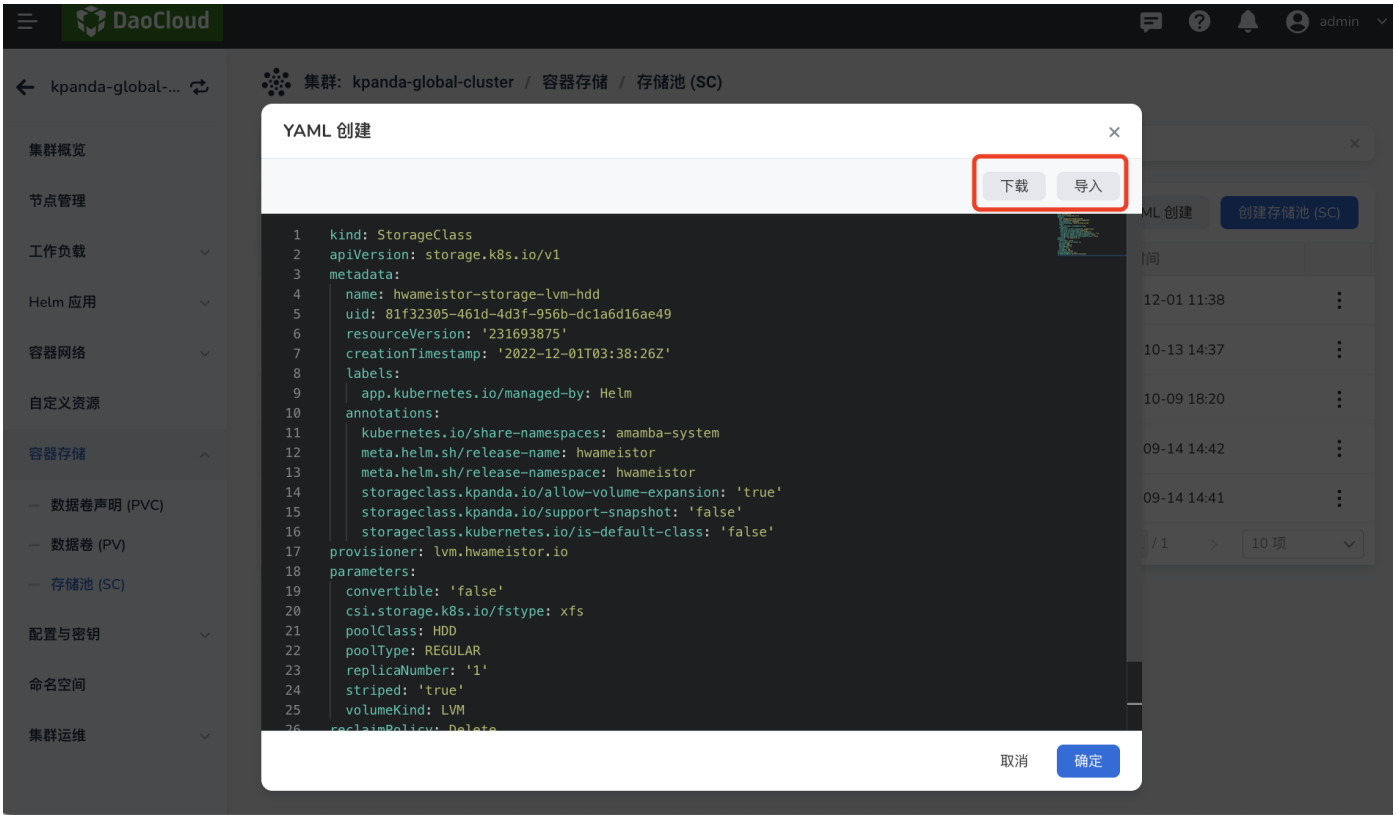
YAML 创建

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 。



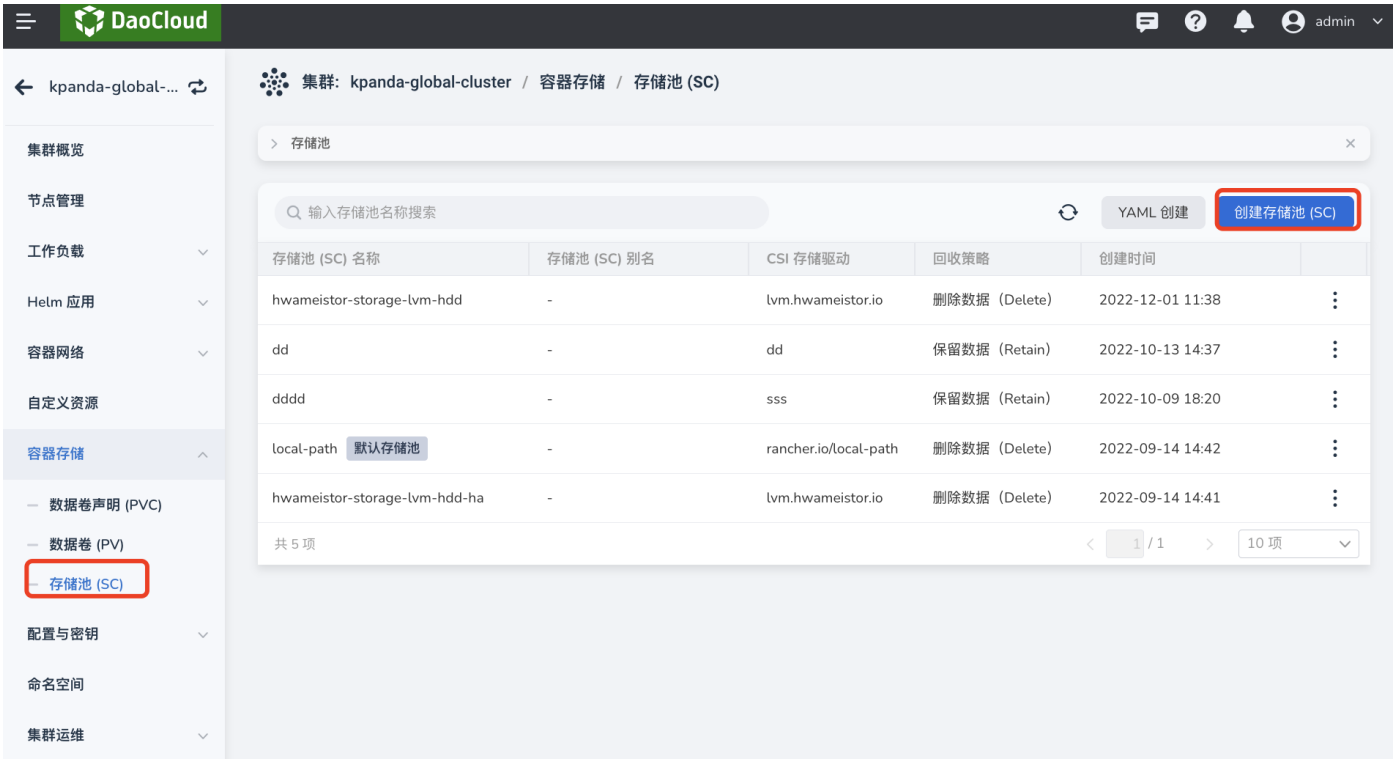
2. 在弹框中输入或粘贴事先准备好的 YAML 文件，然后在弹框底部点击 确定 。

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。



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)** 。



2. 填写基本信息，然后在底部点击 确定 。

自定义存储系统

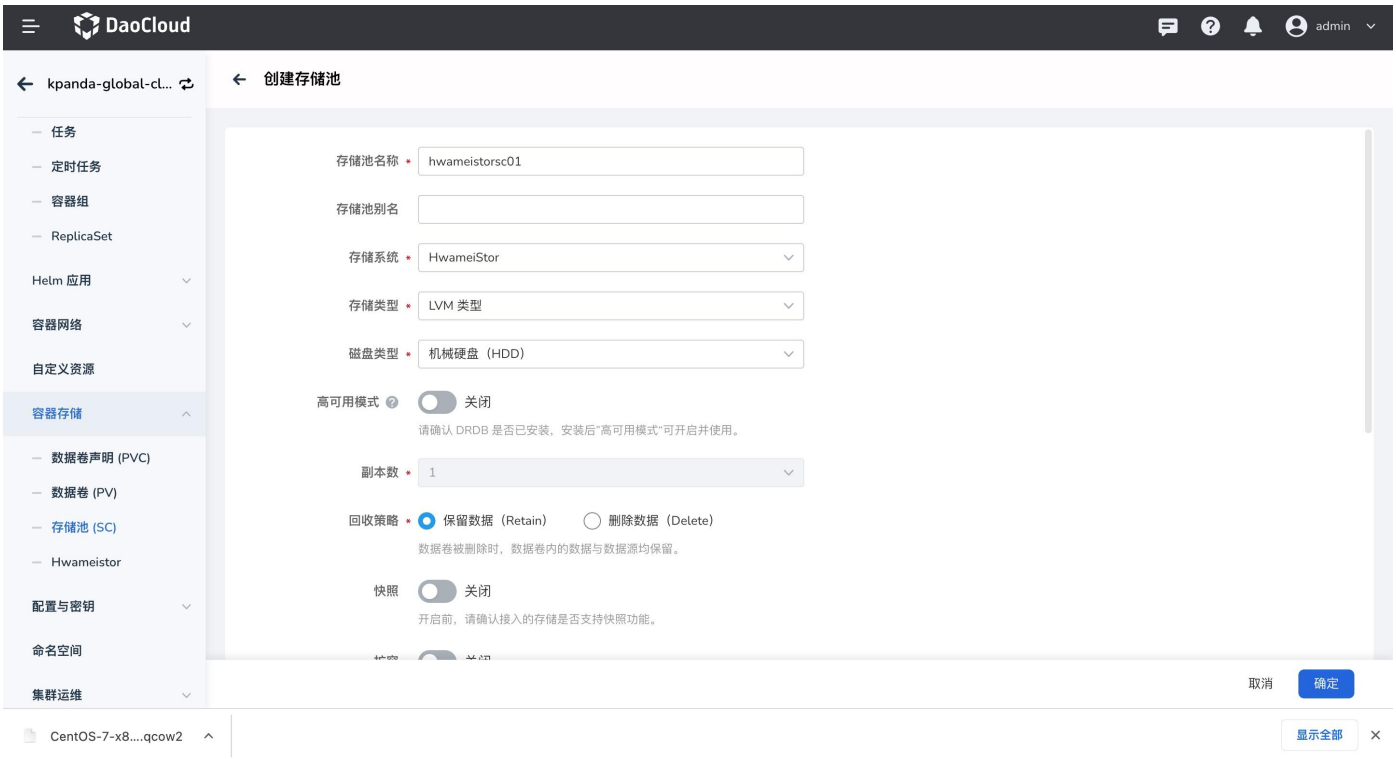
- 存储池名称、驱动、回收策略在创建后不可修改。
- CSI 存储驱动：基于标准 Kubernetes 的容器存储接口插件，需遵守存储厂商规定的格式，例如 **rancher.io/local-path** 。

  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

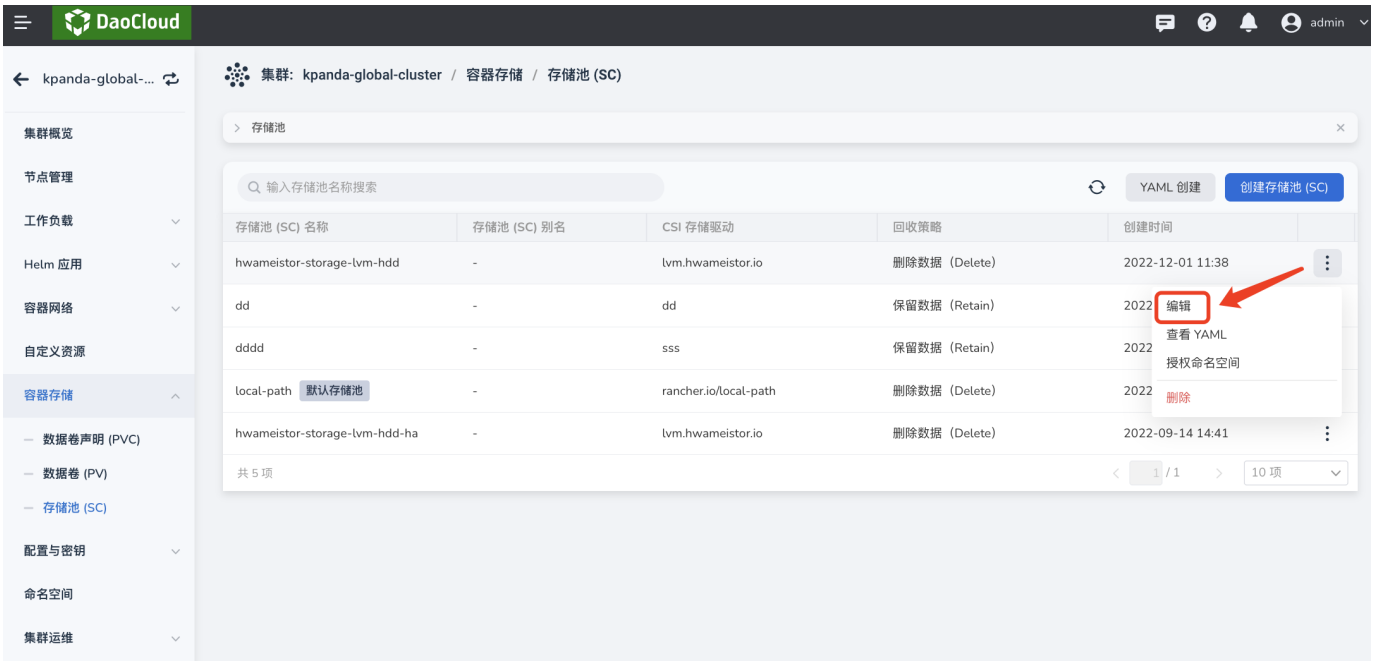
- 存储池名称、驱动、回收策略在创建后不可修改。
- 存储系统：Hwameistor 存储系统。
- 存储类型：支持 LVM，裸磁盘类型
- **LVM** 类型 ：Hwameistor 推荐使用方式，可使用高可用数据卷，对应的的 CSI 存储驱动为： **lvm.hwameistor.io** 。
- 裸磁盘数据卷 ： 适用于高可用场景，无高可用能力，对应的 CSI 驱动为： **hdd.hwameistor.io**
- 高可用模式：使用高可用能力之前请确认 **DRDB** 组件 已安装。开启高可用模式后，可将数据卷副本数设置为 1 和 2。 如需要可将数据卷副本从 1 Convert 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。

> **Note**
>
> 目前 Hwameistor xfs、ext4 两种文件系统，其中默认使用的是 xfs 文件系统，如果想要替换为 ext4，可以在自定义参数添加 **csi.storage.k8s.io/fstype: ext4**



更新存储池(SC)

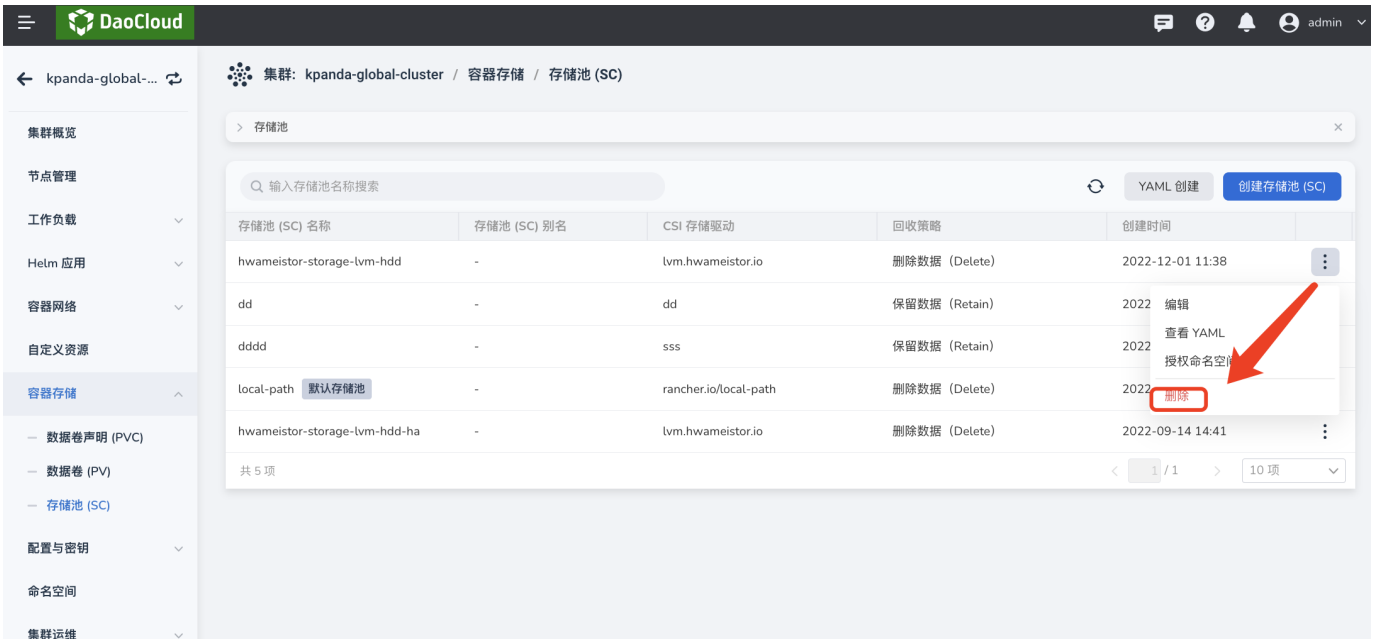
在存储池列表页面，找到需要更新的存储池，在右侧的操作栏下选择 编辑 即可通过更新存储池。



> **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)

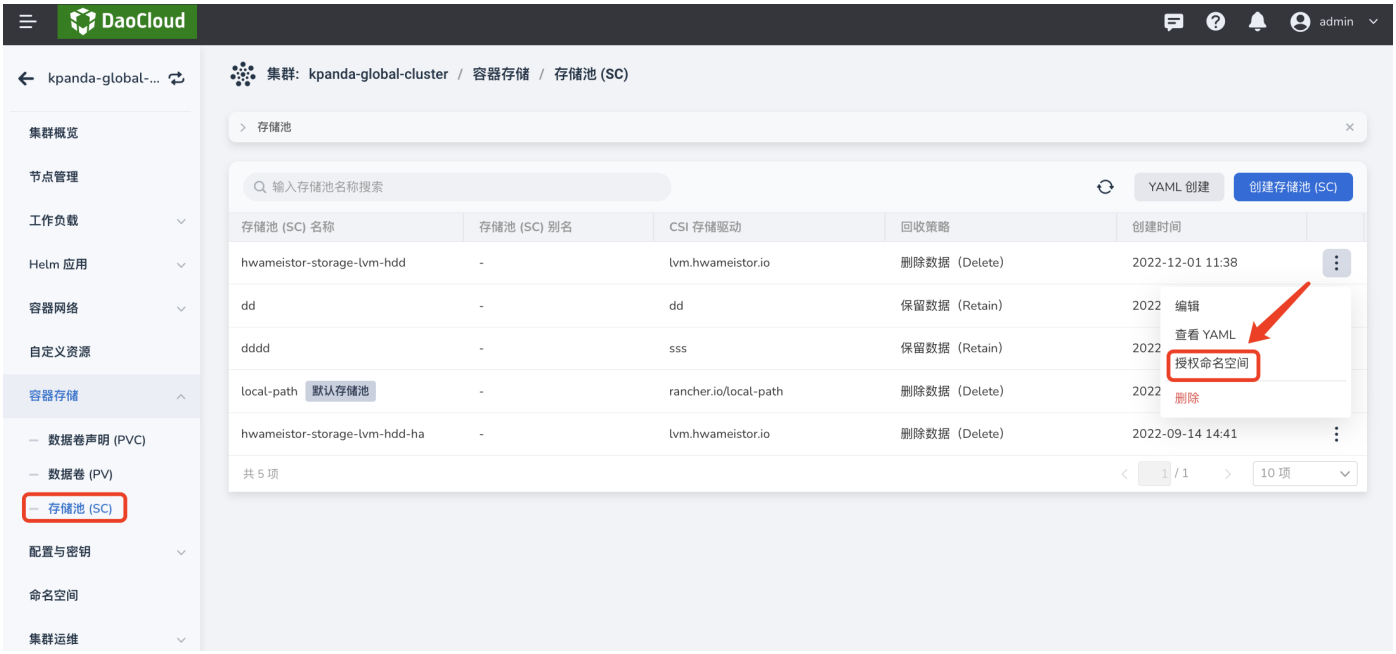
在存储池列表页面，找到需要删除的存储池，在右侧的操作栏下选择 删除 。


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 / 存储池 (SC)
存储池
输入存储池名称搜索
YAML 创建
创建存储池 (SC)
存储池 (SC) 名称
存储池 (SC) 别名
CSI 存储驱动
回收策略
创建时间
hwameistor-storage-lvm-hdd - lvm.hwameistor.io 删除数据（Delete） 2022-12-01 11:38
dd - dd 保留数据（Retain） 2022
dddd - sss 保留数据（Retain） 2022
local-path 默认存储池 - rancher.io/local-path 删除数据（Delete） 2022
hwameistor-storage-lvm-hdd-ha - lvm.hwameistor.io 删除数据（Delete） 2022-09-14 14:41
编辑
查看 YAML
授权命名空间
删除
共 5 项
1 / 1
10 项

共享存储池

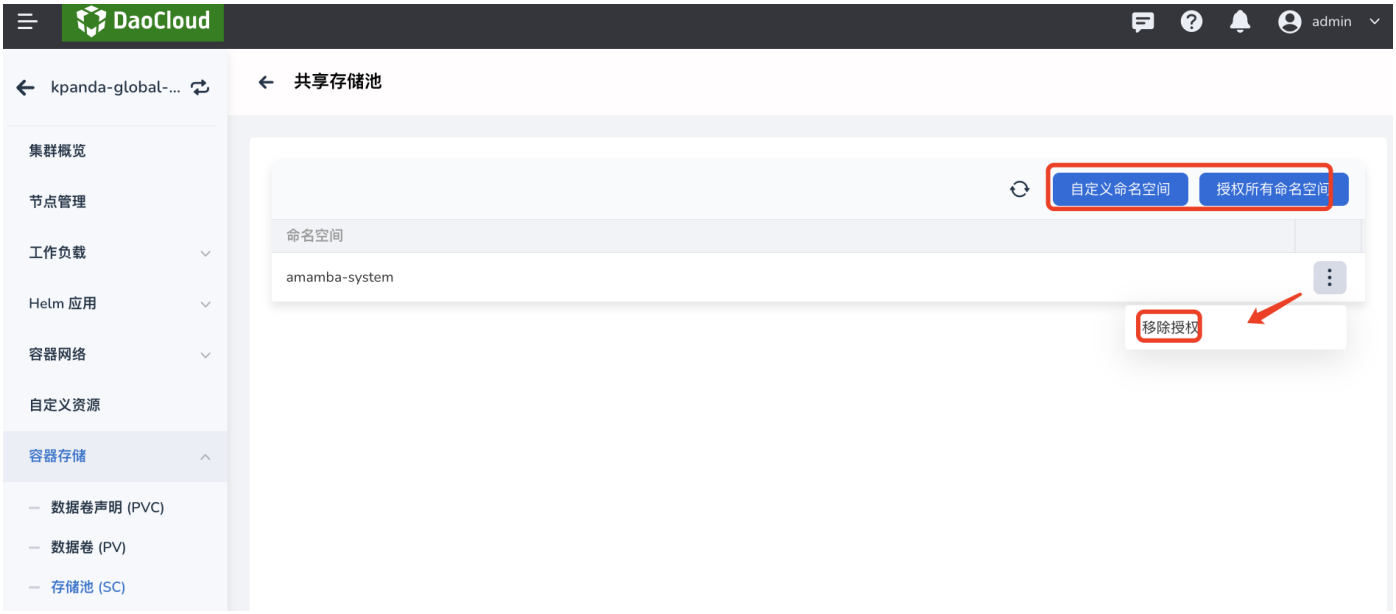
d.run 容器管理模块支持将一个存储池共享给多个命名空间使用，以便提高资源利用效率。

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 。



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。



配置与密钥

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，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：

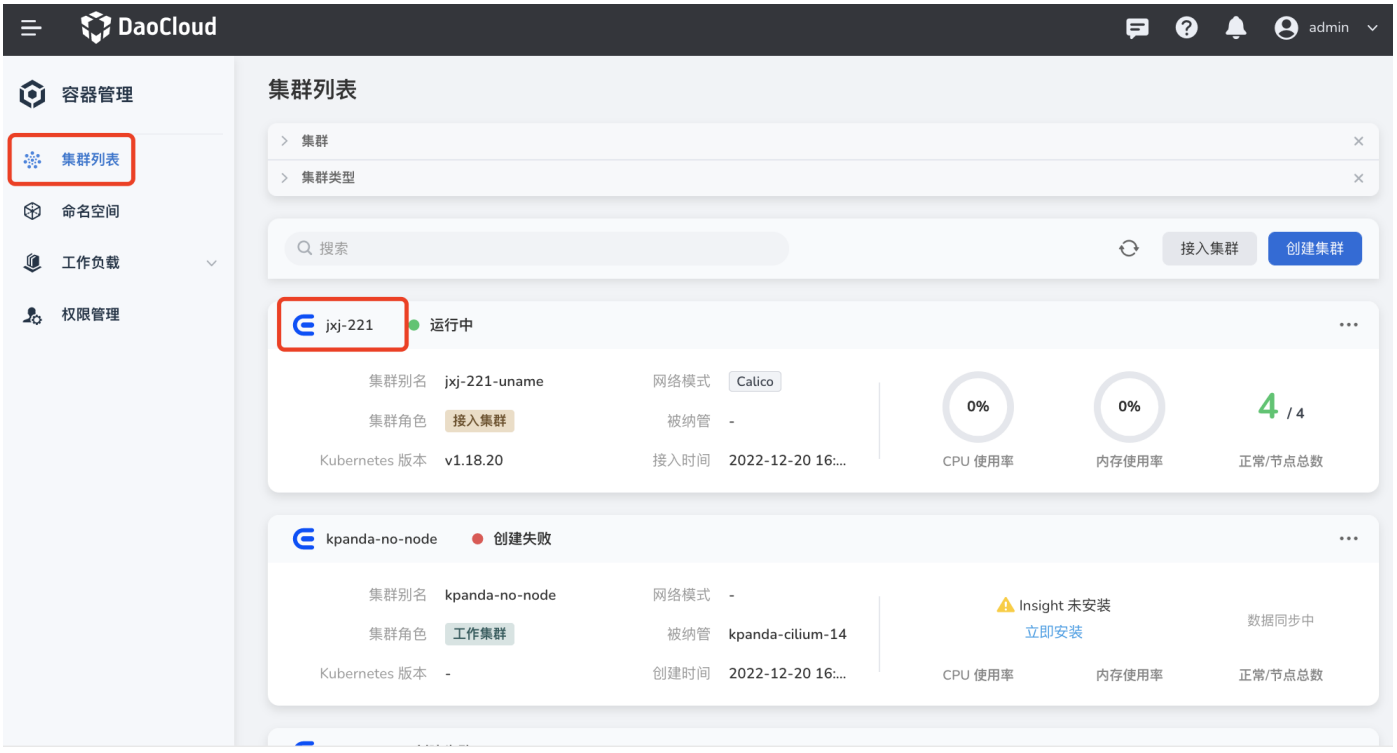
- 图形化表单创建
- YAML 创建

前提条件

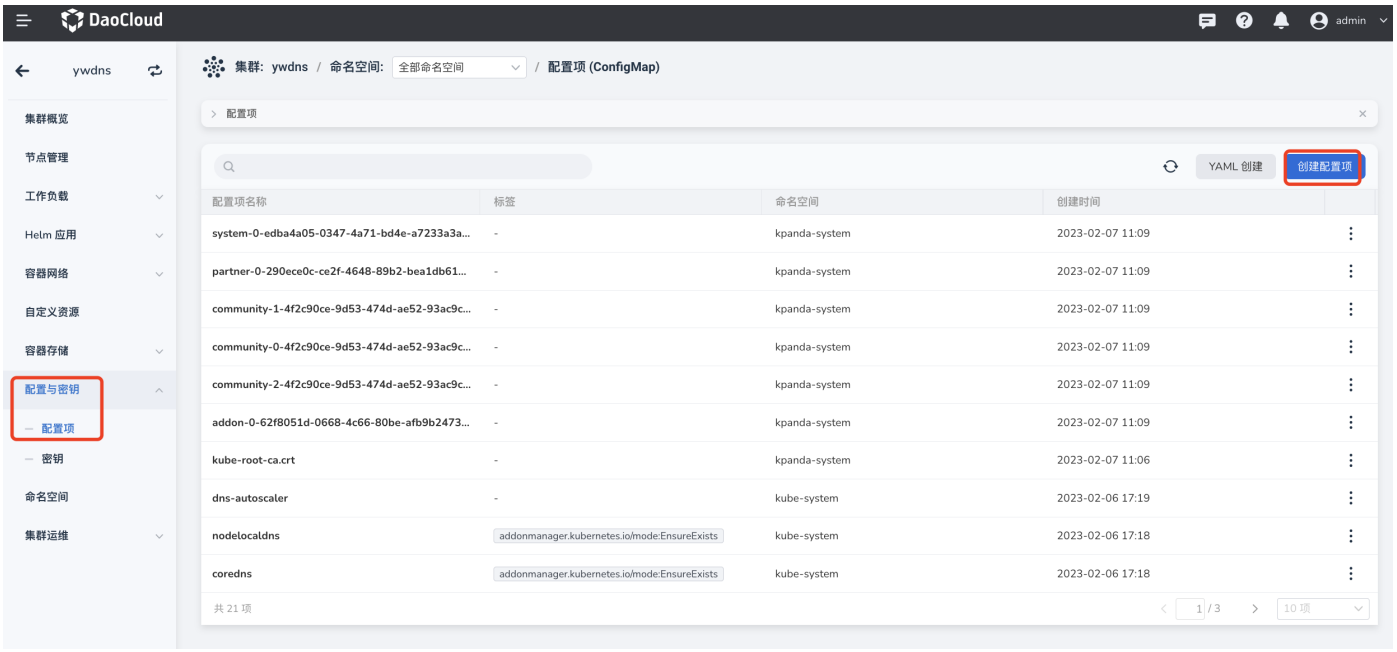
已完成一个命名空间的创建、用户的创建， 并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。

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 。

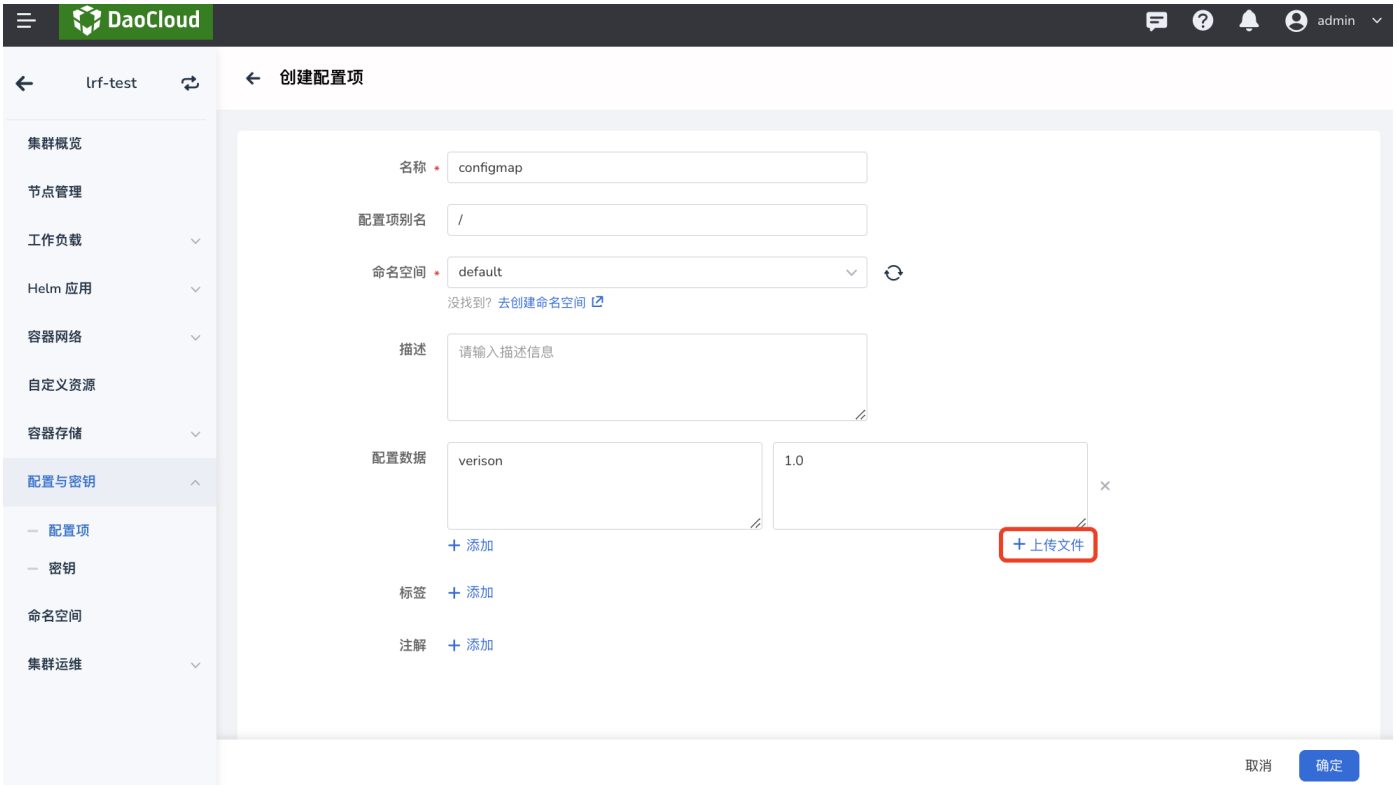


2. 在左侧导航栏，点击 配置与密钥 -> 配置项 ，点击右上角 创建配置项 按钮。

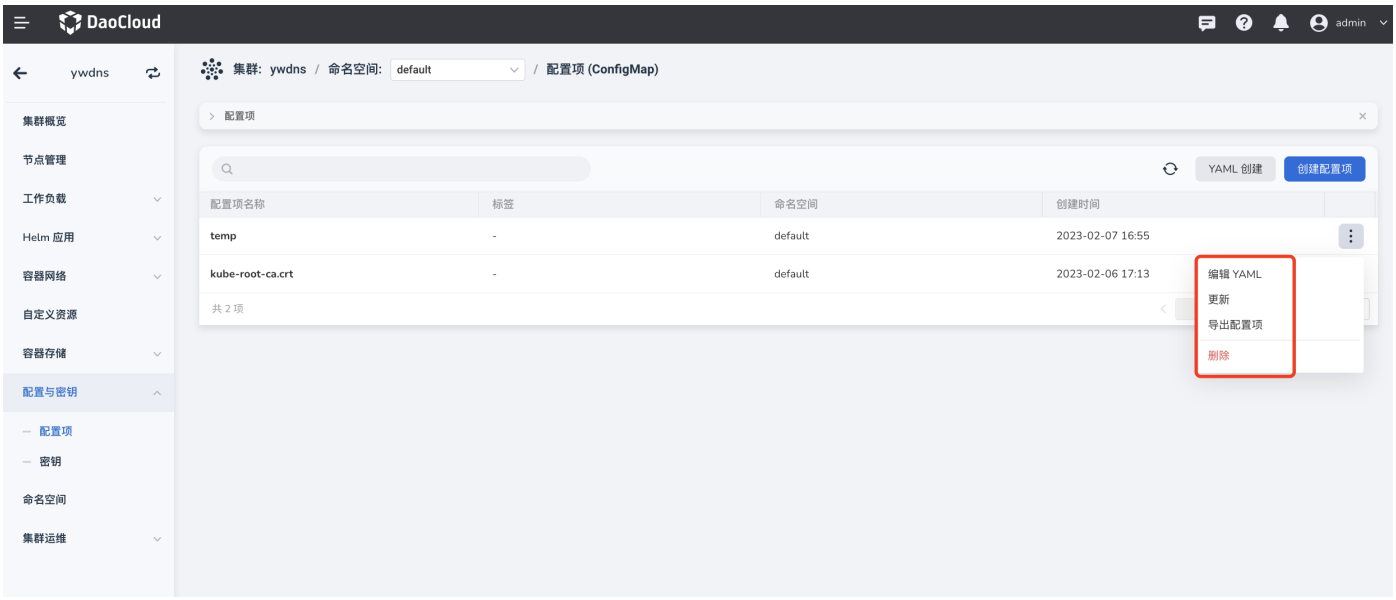


3. 在 创建配置项 页面中填写配置信息，点击 确定 。

> **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。

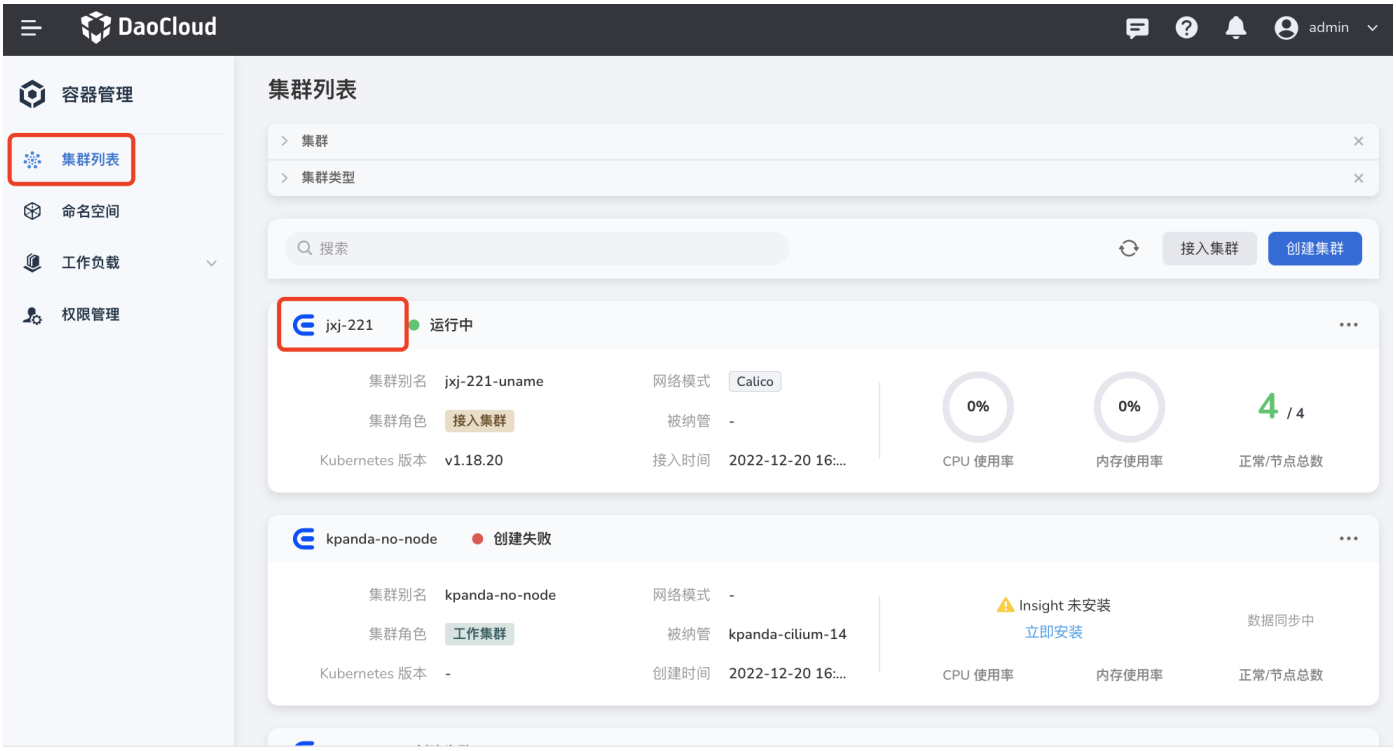


4. 创建完成后在配置项右侧点击更多可以，可以编辑 YAML、更新、导出、删除等操作。

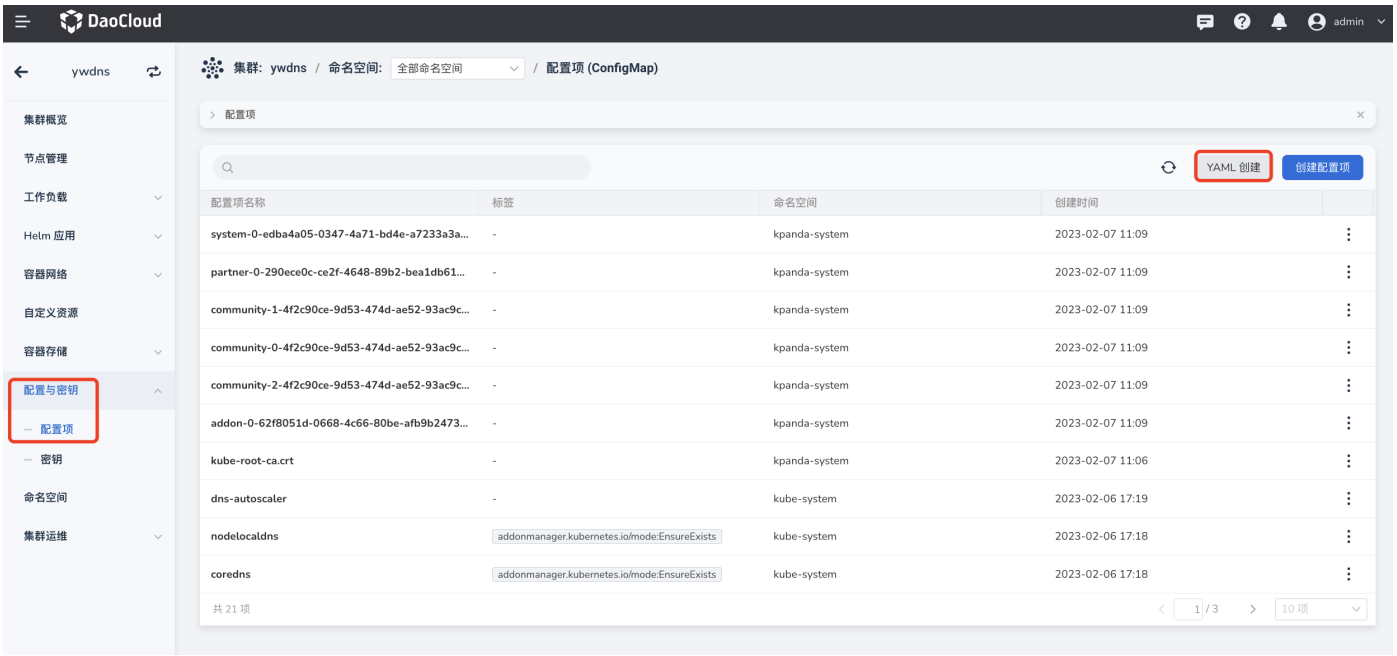


YAML 创建

1. 在 集群列表 页面点击某个集群的名称，进入 集群详情 。



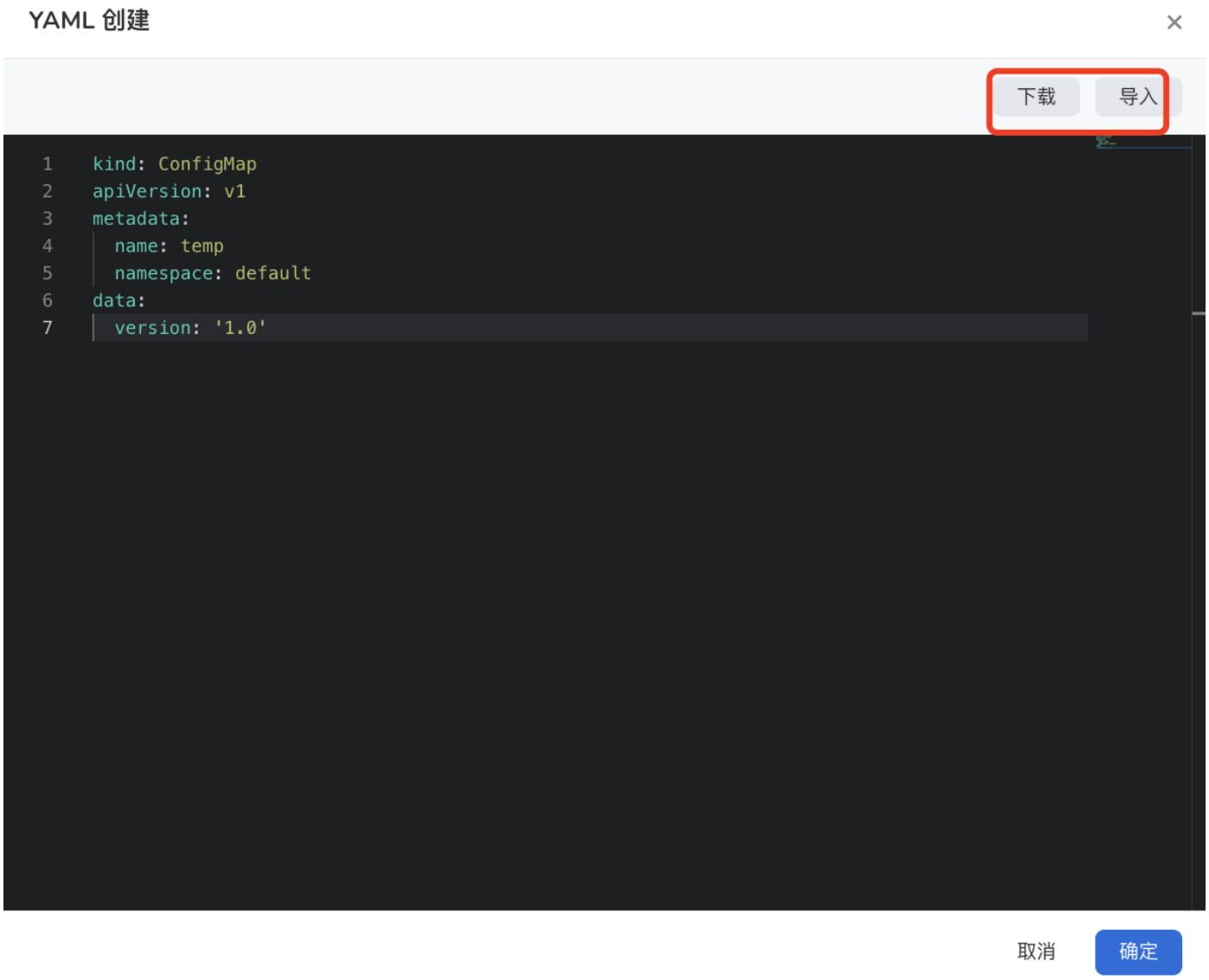
2. 在左侧导航栏，点击 配置与密钥 -> 配置项 ，点击右上角 **YAML** 创建 按钮。



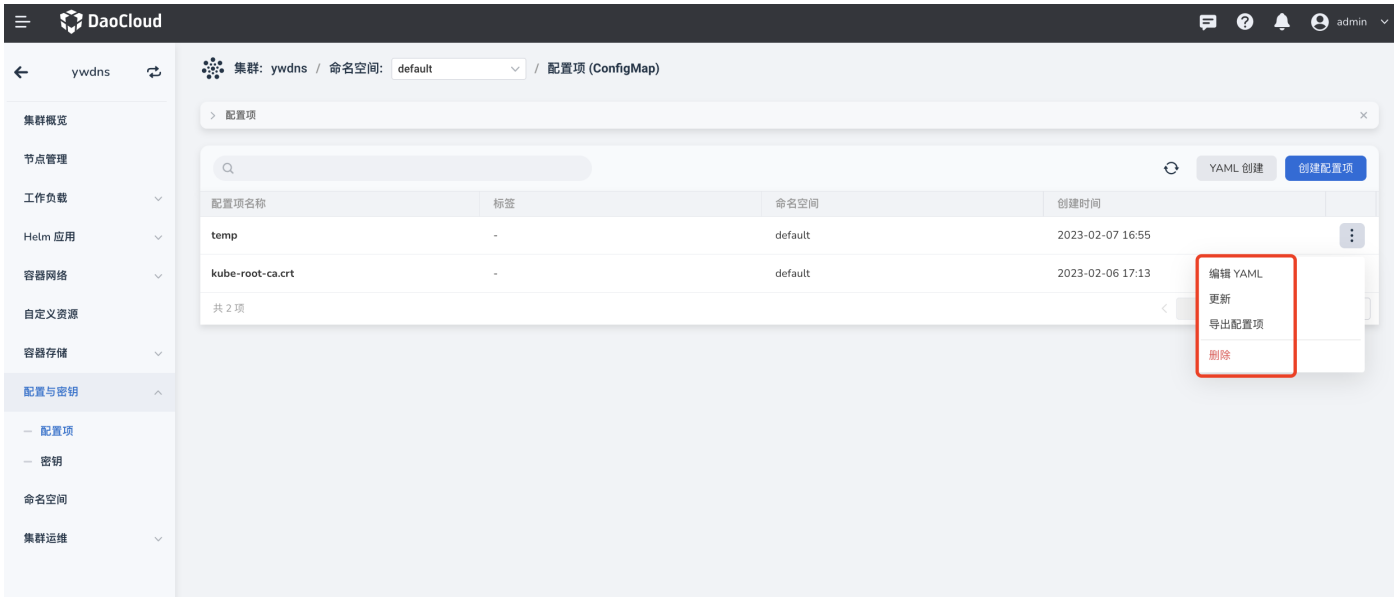
3. 填写或粘贴事先准备好的配置文件，然后在弹框右下角点击 确定 。

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。



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。



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，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

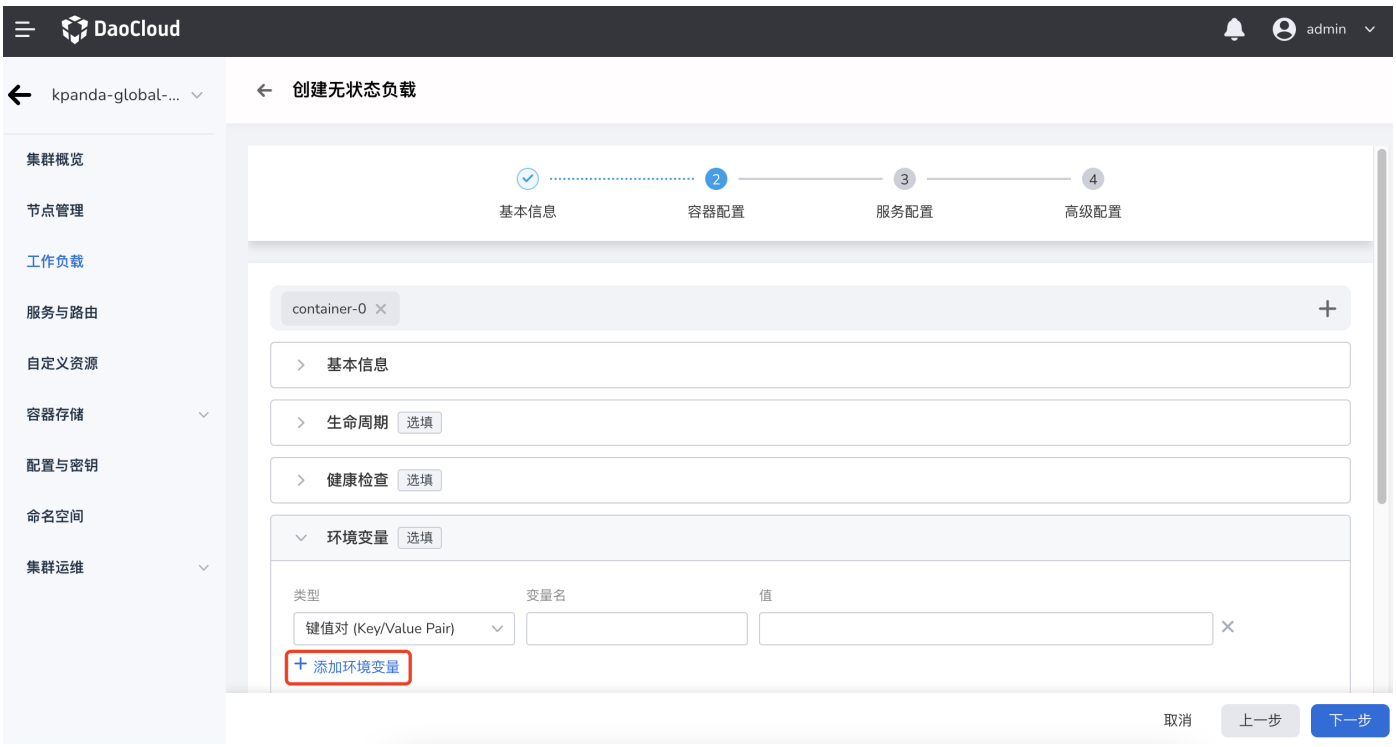
您可以通过图形化界面或者终端命令行来使用配置项作为容器的环境变量。

> **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。



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_TYPE_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
```

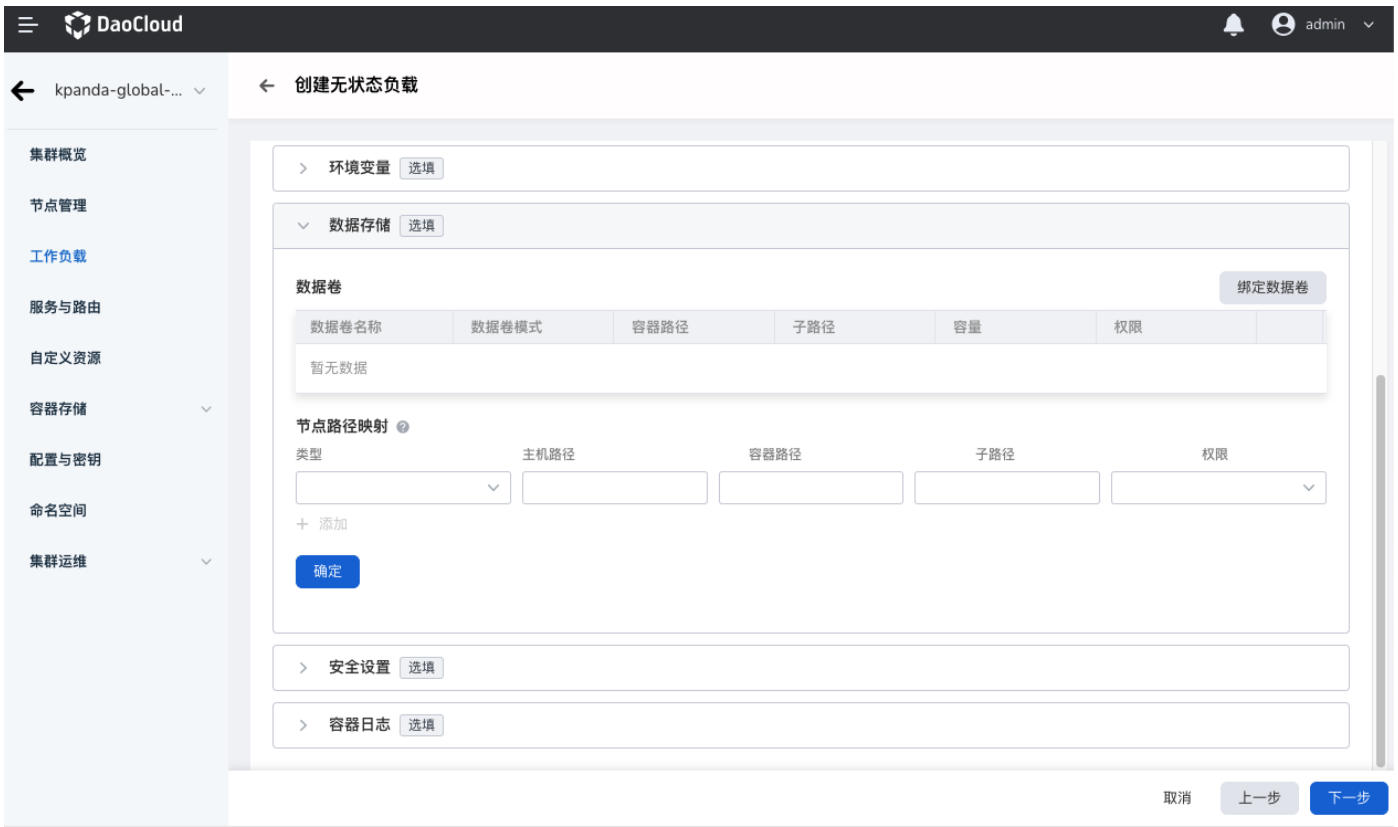
用作容器数据卷

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。



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：

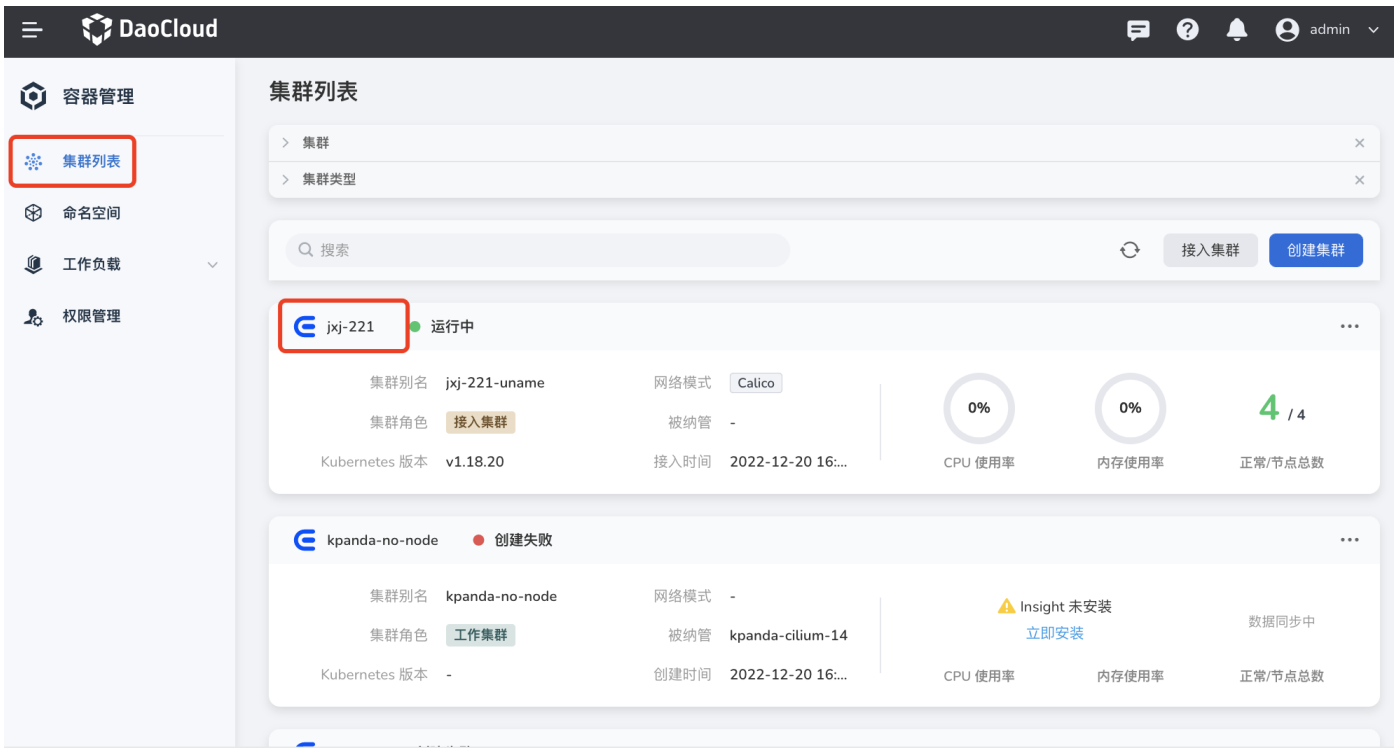
- 图形化表单创建
- YAML 创建

前提条件

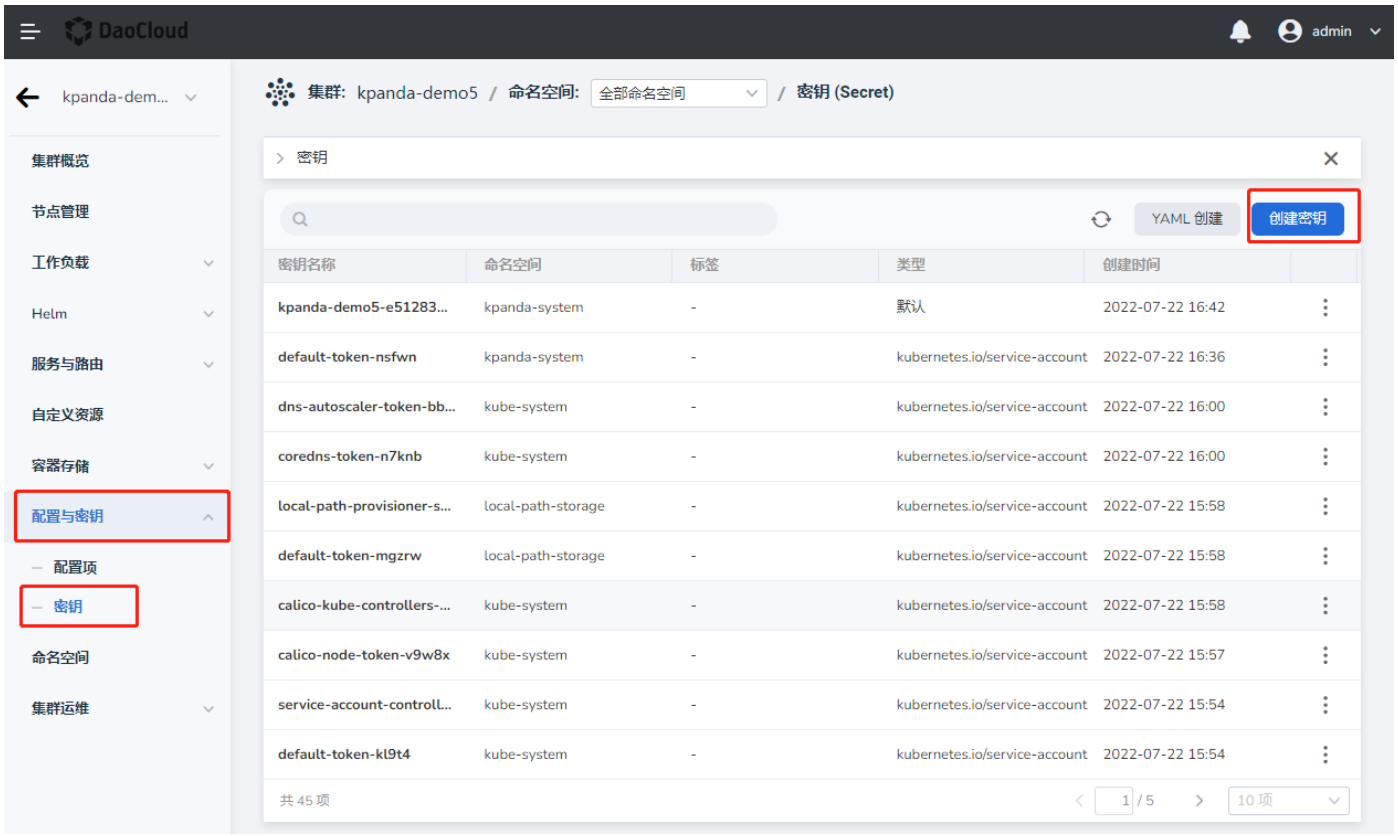
已完成一个命名空间的创建、用户的创建， 并将用户授权为 NS Edit 角色 ，详情可参考命名空间授权。

图形化表单创建

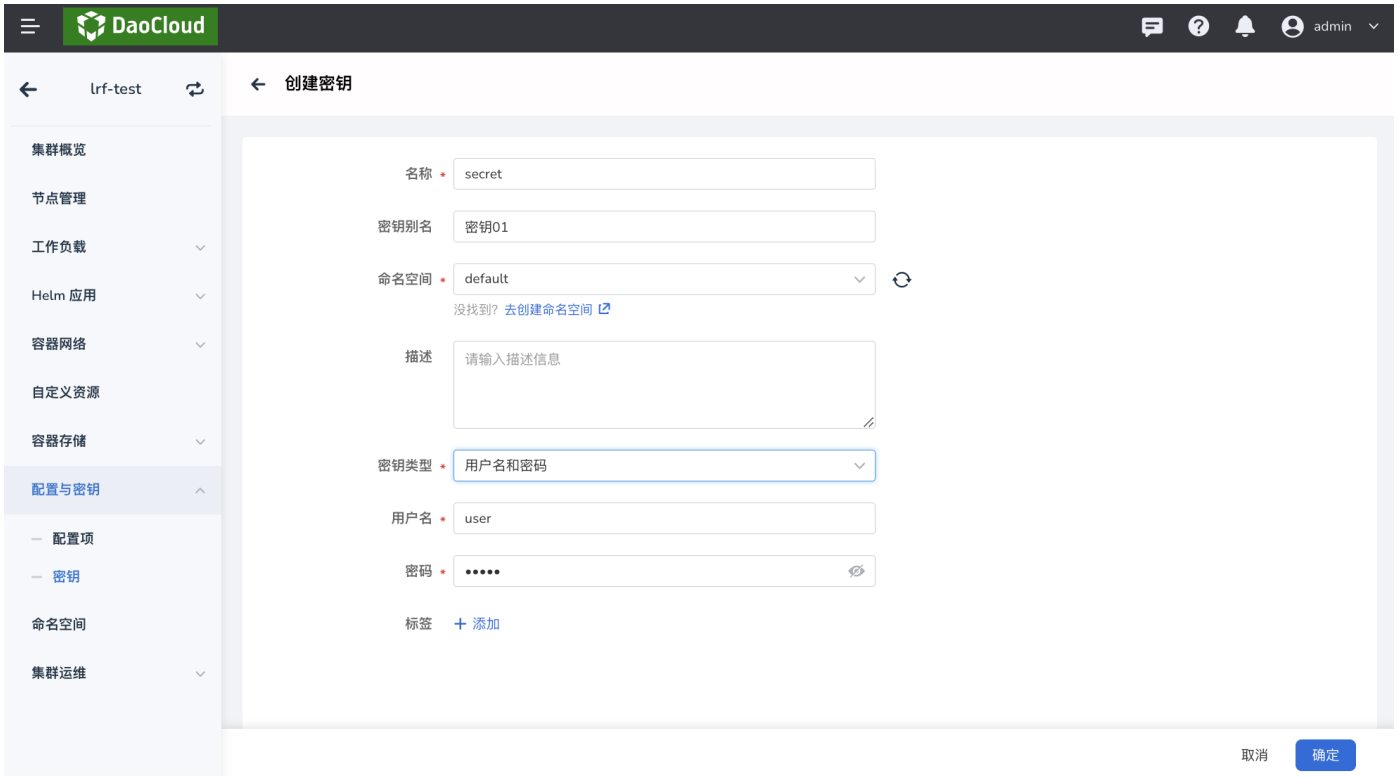
1. 在 集群列表 页面点击某个集群的名称，进入 集群详情 。



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。



3. 在 创建密钥 页面中填写配置信息，点击 确定 。

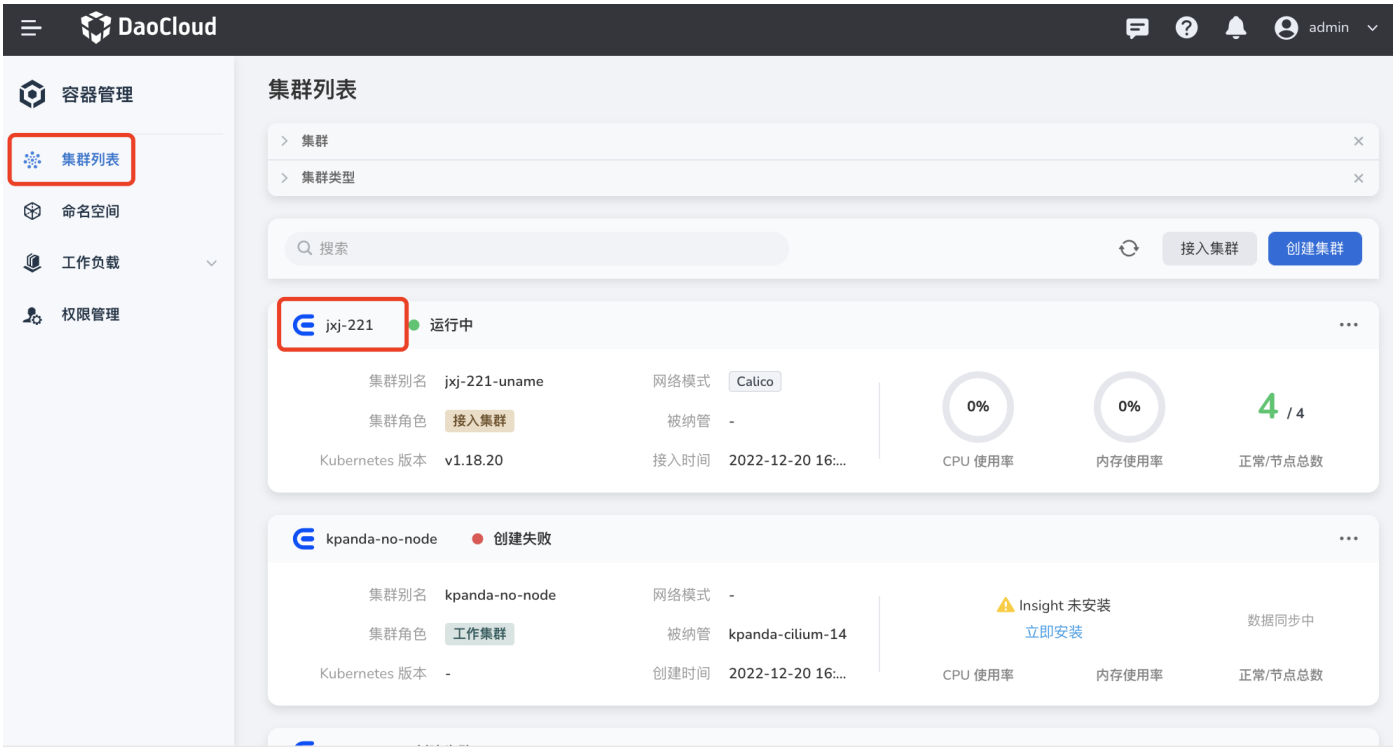


填写配置时需要注意：

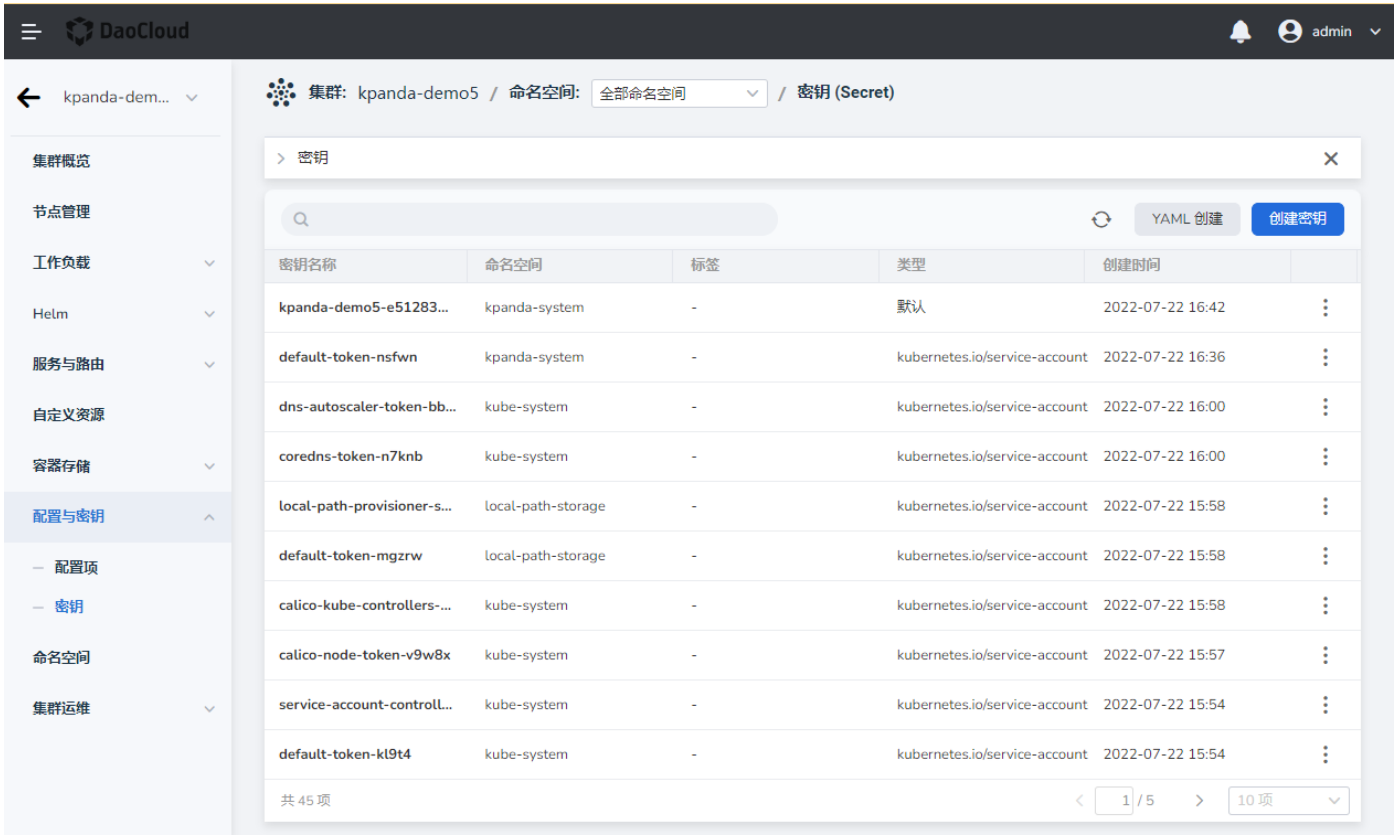
- 密钥的名称在同一个命名空间中必须具有唯一性
- 密钥类型：
- 默认（Opaque）：Kubernetes 默认的密钥类型，支持用户定义的任意数据。
- TLS (kubernetes.io/tls)：用于 TLS 客户端或者服务器端数据访问的凭证。
- 镜像仓库信息 (kubernetes.io/dockerconfigjson)：用于镜像仓库访问的凭证。
- 用户名和密码（kubernetes.io/basic-auth）：用于基本身份认证的凭证。
- 自定义：用户根据业务需要自定义的类型。
- 密钥数据：密钥所存储的数据，不同数据需要填写的参数有所不同
- 当密钥类型为默认（Opaque）/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账号和密码。
- 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 。

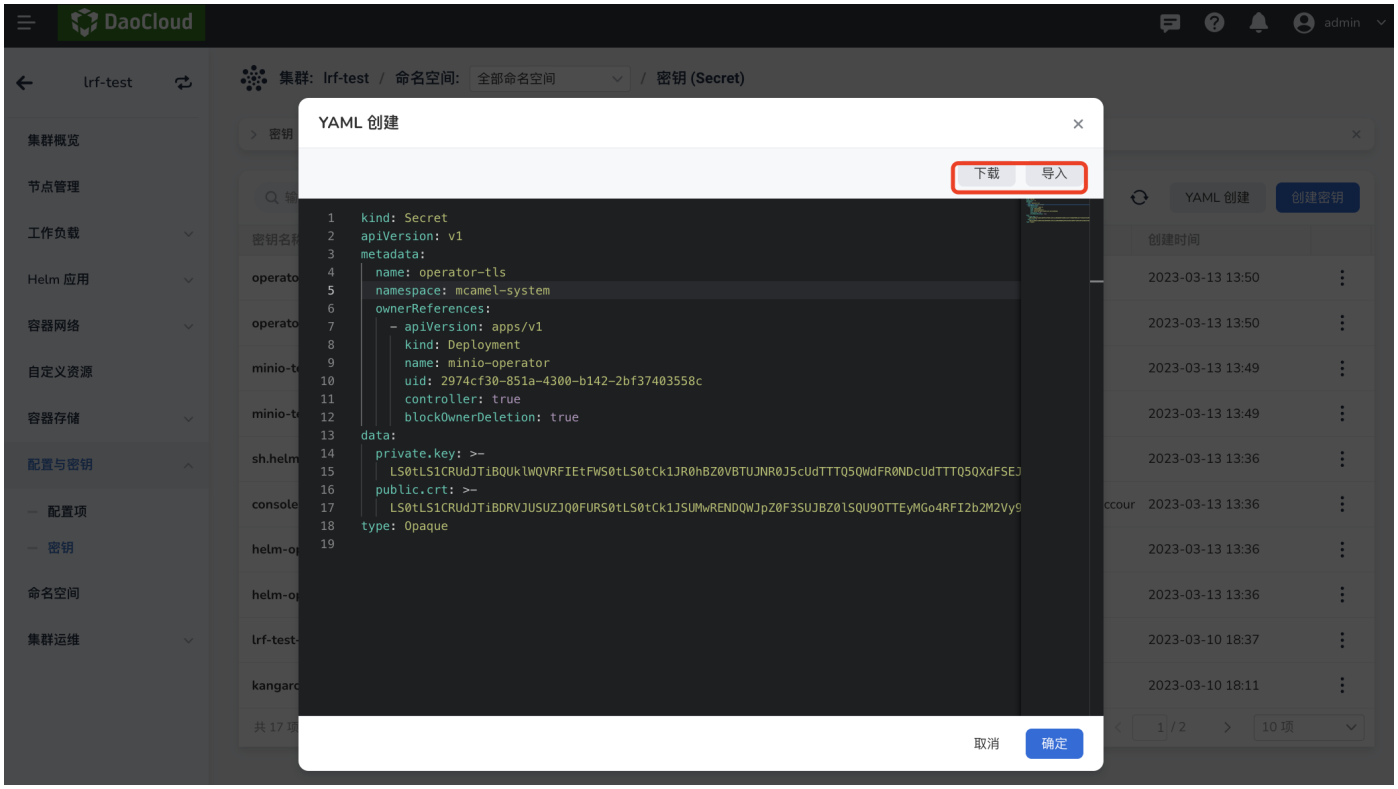


2. 在左侧导航栏，点击 配置与密钥 -> 密钥 ，点击右上角 **YAML** 创建 按钮。



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。



密钥 YAML 示例

```
```yaml
apiVersion: v1
```
```

```
kind: Secret
metadata:
  name: secretdemo
type: Opaque
data:
  username: ******
  password: ******
```

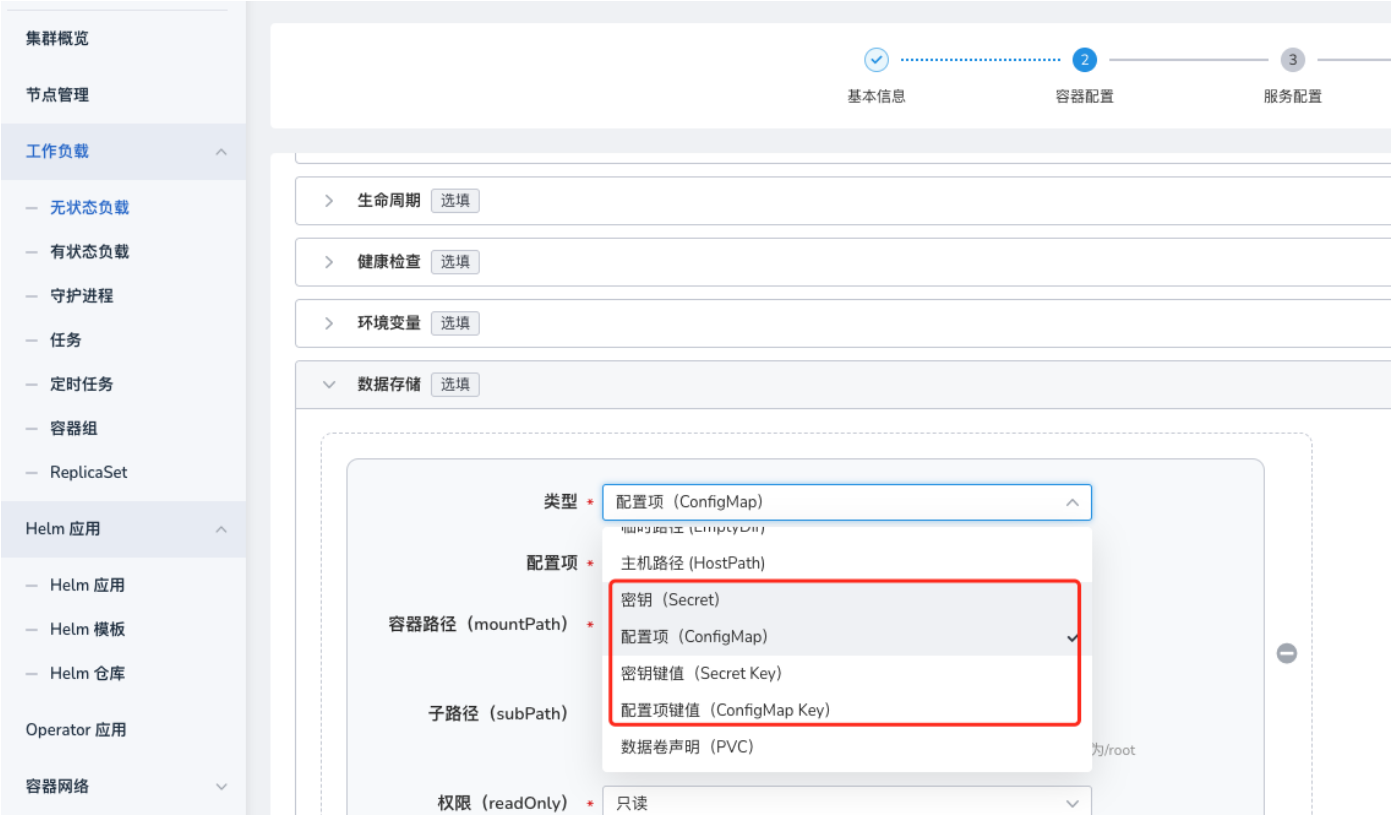
下一步：使用密钥

CONFIGMAP/SECRET 热加载

ConfigMap/Secret 热加载是指将 ConfigMap/Secret 作为数据卷挂载在容器中挂载时，当配置发生改变时，容器将自动读取 ConfigMap/Secret 更新后的配置，而无需重启 Pod。

操作步骤

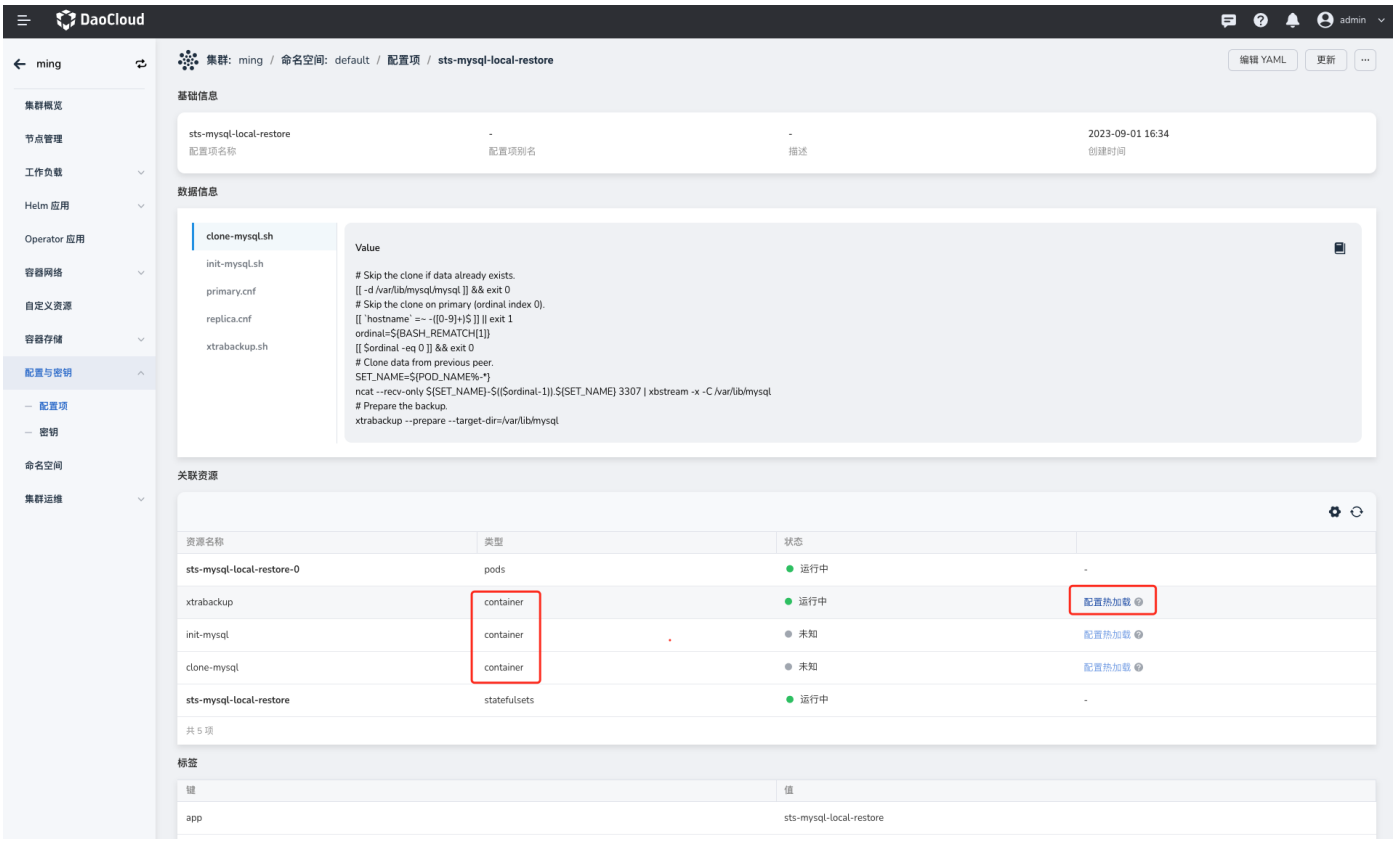
1. 参考创建工作负载 - 容器配置，配置容器数据存储，选择 **ConfigMap** 、 **ConfigMap Key** 、 **Secret** 、 **Secret Key** 作为数据卷挂载至容器。



Note

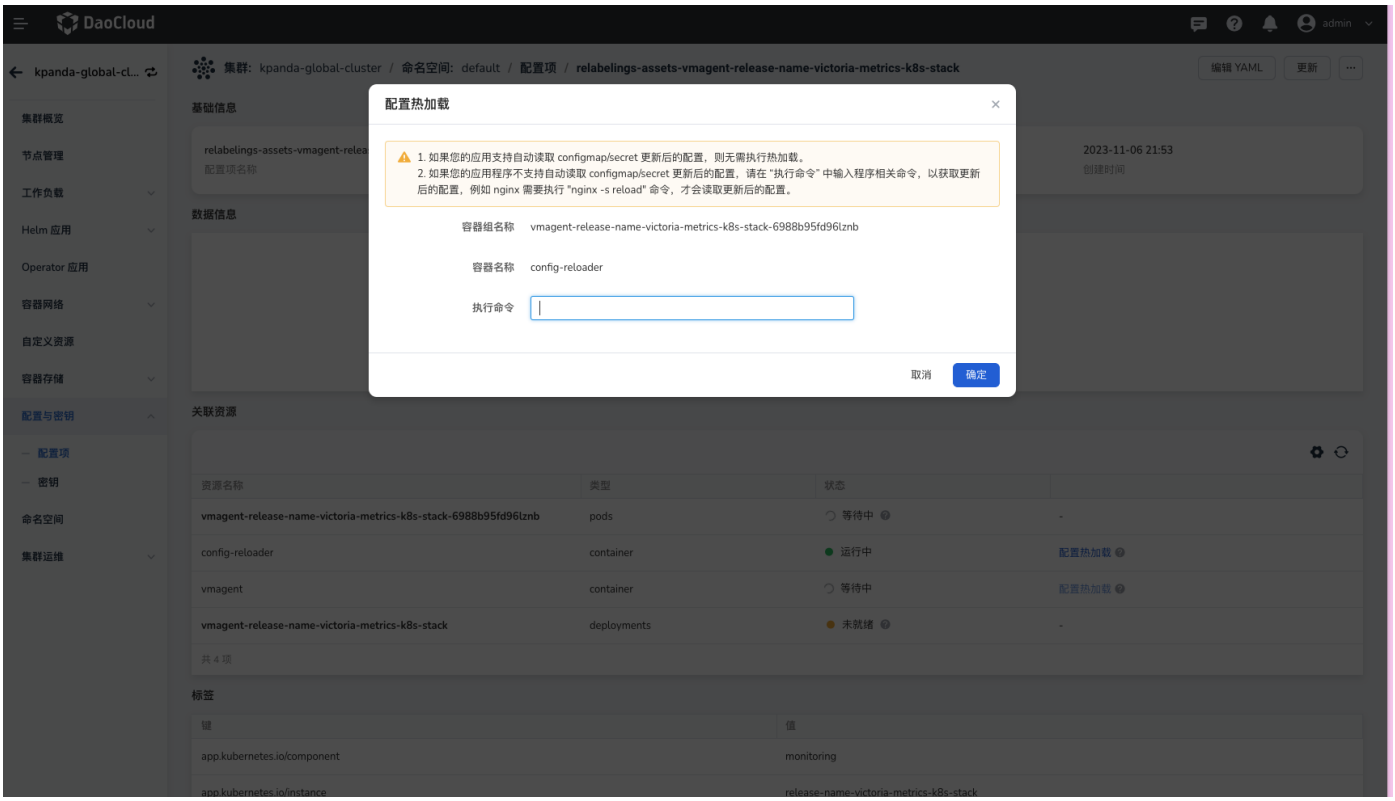
使用子路径（SubPath）方式挂载的配置文件不支持热加载。

2. 进入 配置与密钥 页面，进入配置项详情页面，在 关联资源 中找到对应的 **container** 资源，点击 立即加载 按钮，进入配置热加载页面。



> **Note**
>
> 如果您的应用支持自动读取 CConfigMap/Secret 更新后的配置，则无需手动执行热加载操作。

3. 在热加载配置弹窗中，输入进入容器内的 执行命令 并点击 确定 按钮，以重载配置。例如，在 nginx 容器中，以 root 用户权限，执行 **nginx -s reload** 命令来重载配置。



4. 在界面弹出的 web 终端中查看应用重载情况。

命名空间

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。

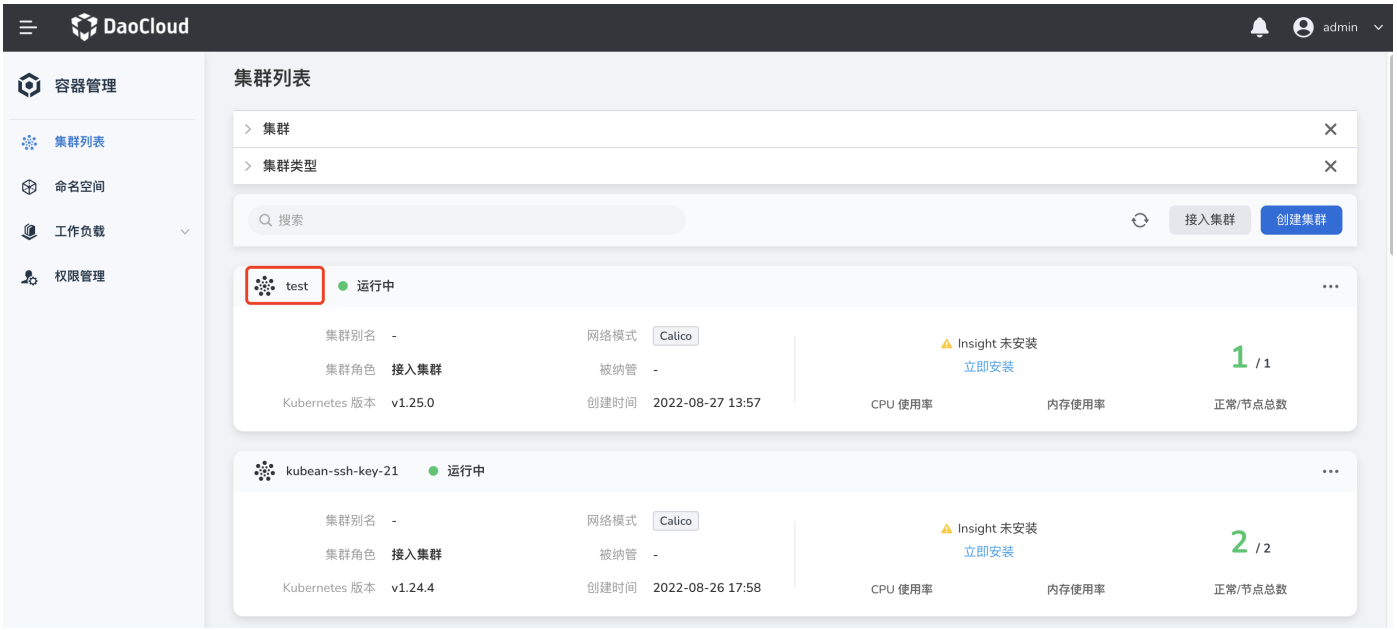
创建命名空间

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。

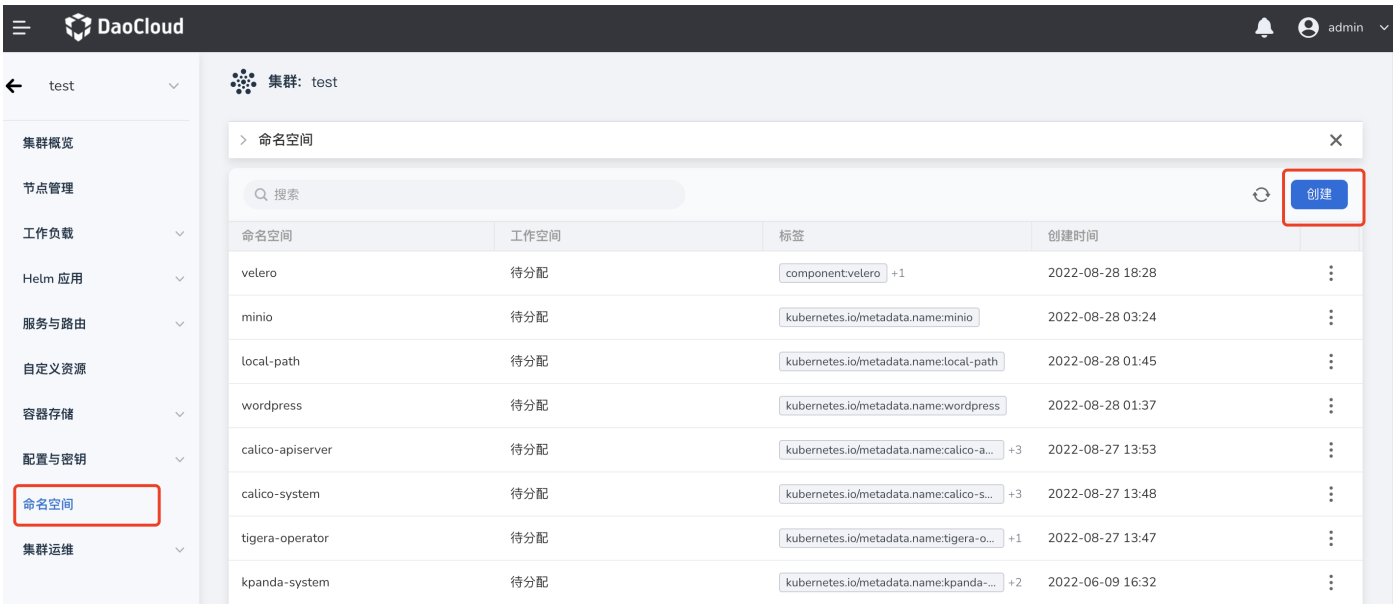
> **Note**
>
> 集群初始化后通常会自动生成默认的命名空间 **default** 。但对于生产集群而言，为便于管理，建议创建其他的命名空间，而非直接使用 **default** 命名空间。

表单创建

1. 在 集群列表 页面点击目标集群的名称。



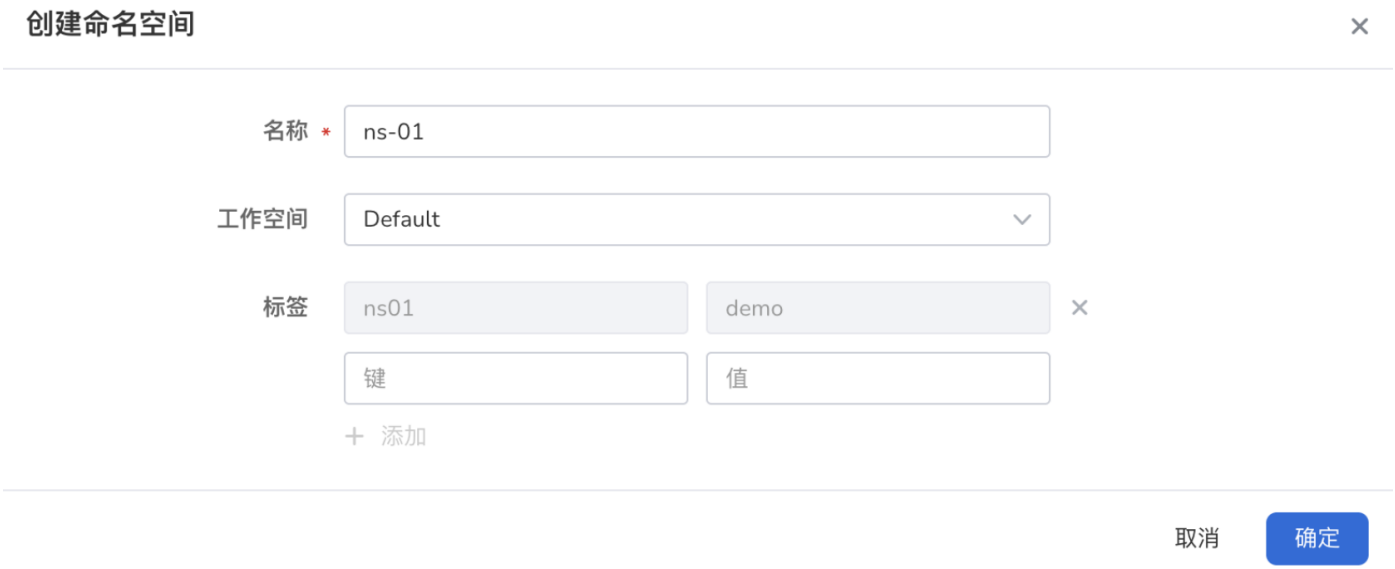
2. 在左侧导航栏点击 命名空间 ，然后点击页面右侧的 创建 按钮。



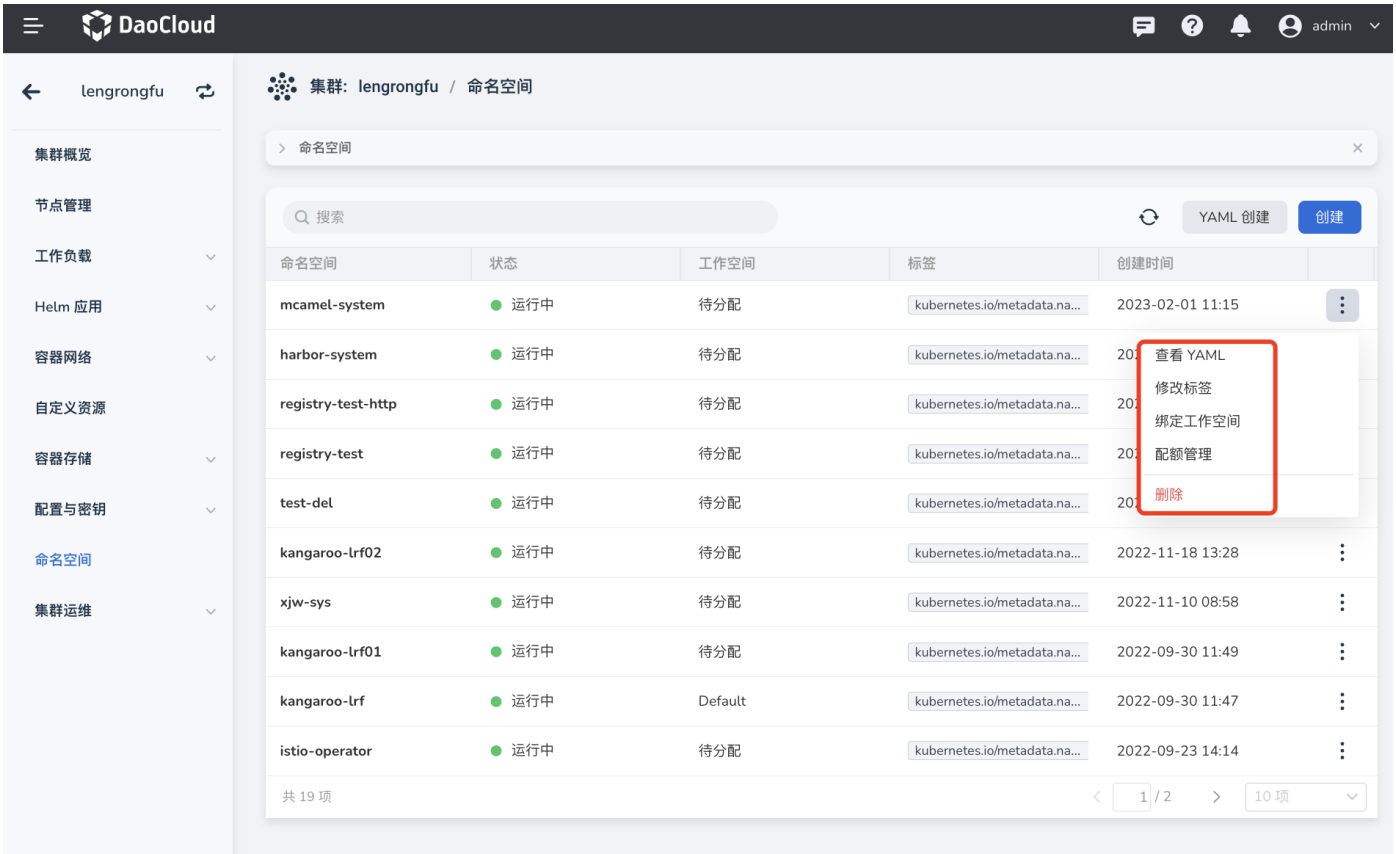
3. 填写命名空间的名称，配置工作空间和标签（可选设置），然后点击 确定 。

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。
- 命名空间创建完成后，仍然可以绑定/解绑工作空间。

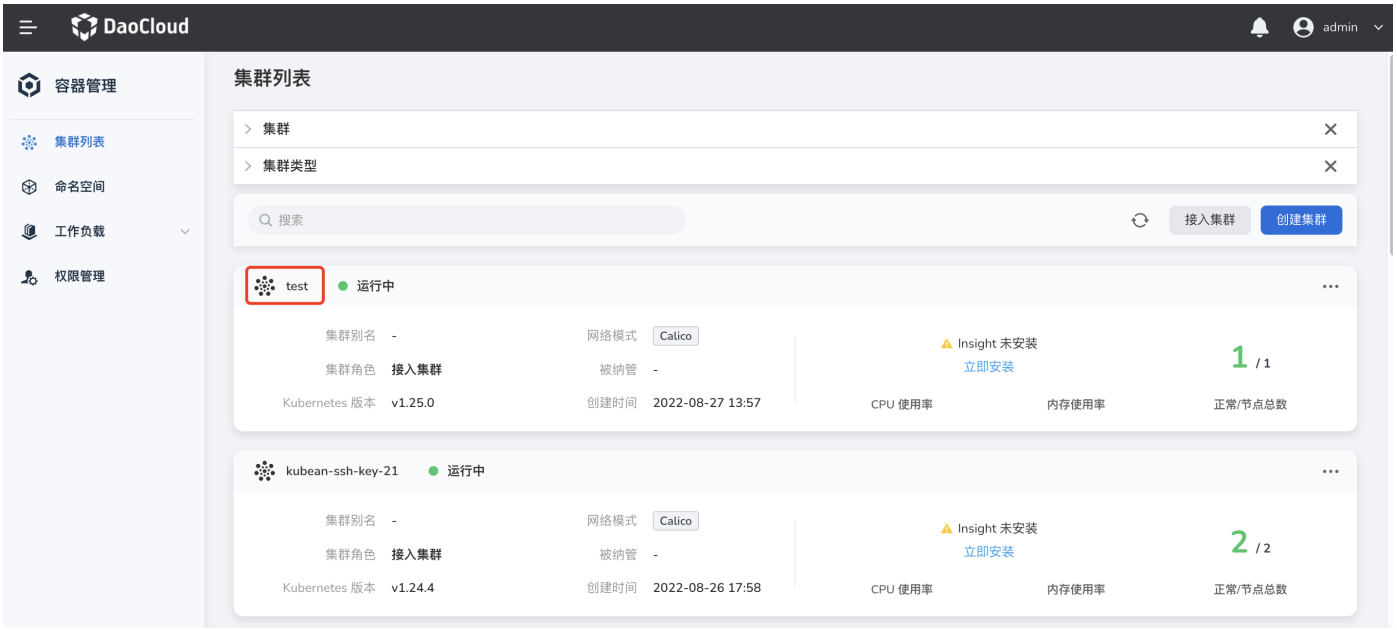


4. 点击 确定 ，完成命名空间的创建。在命名空间列表右侧，点击 ⋮ ，可以从弹出菜单中选择更新、绑定/解绑工作空间、配额管理、删除等更多操作。

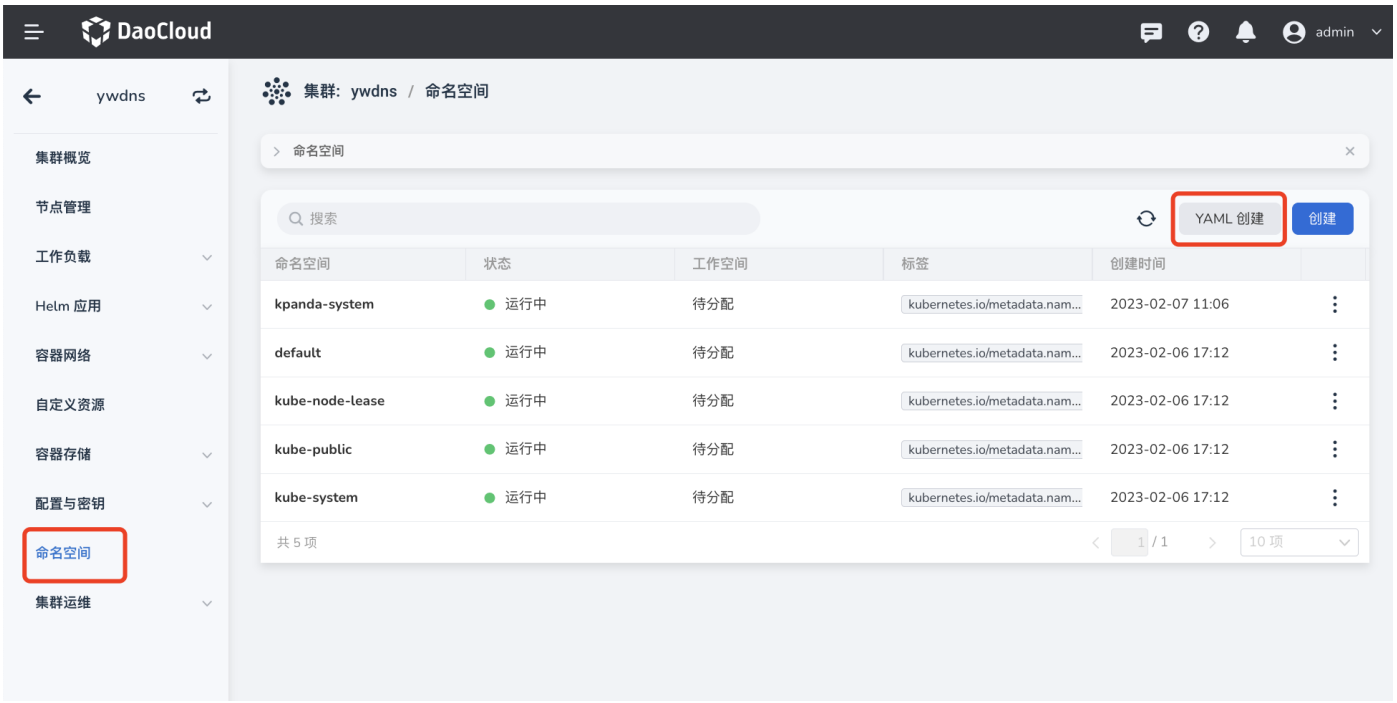


YAML 创建

1. 在 集群列表 页面点击目标集群的名称。

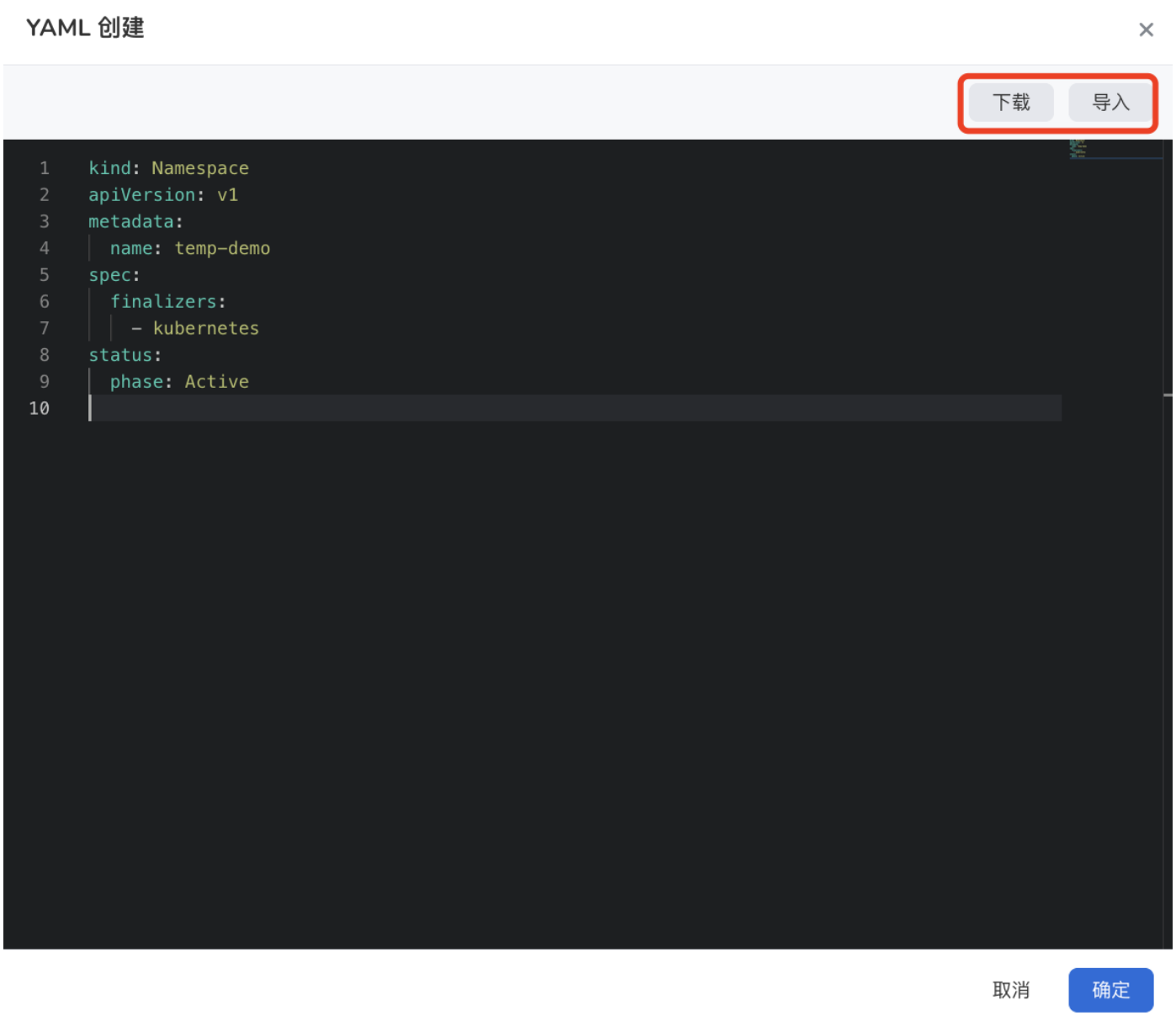


2. 在左侧导航栏点击 命名空间 ，然后点击页面右侧的 **YAML** 创建 按钮。



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。



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 d.run 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
metallb-system	metallb
cert-manager-system	cert-manager
contour-system	contour
kubean-system	kubean
ghippo-system	ghippo
kcoral-system	kcoral
kcollie-system	kcollie
insight-system	insight、insight-agent:
ipavo-system	ipavo
spidernet-system	spidernet
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"}]'` 。

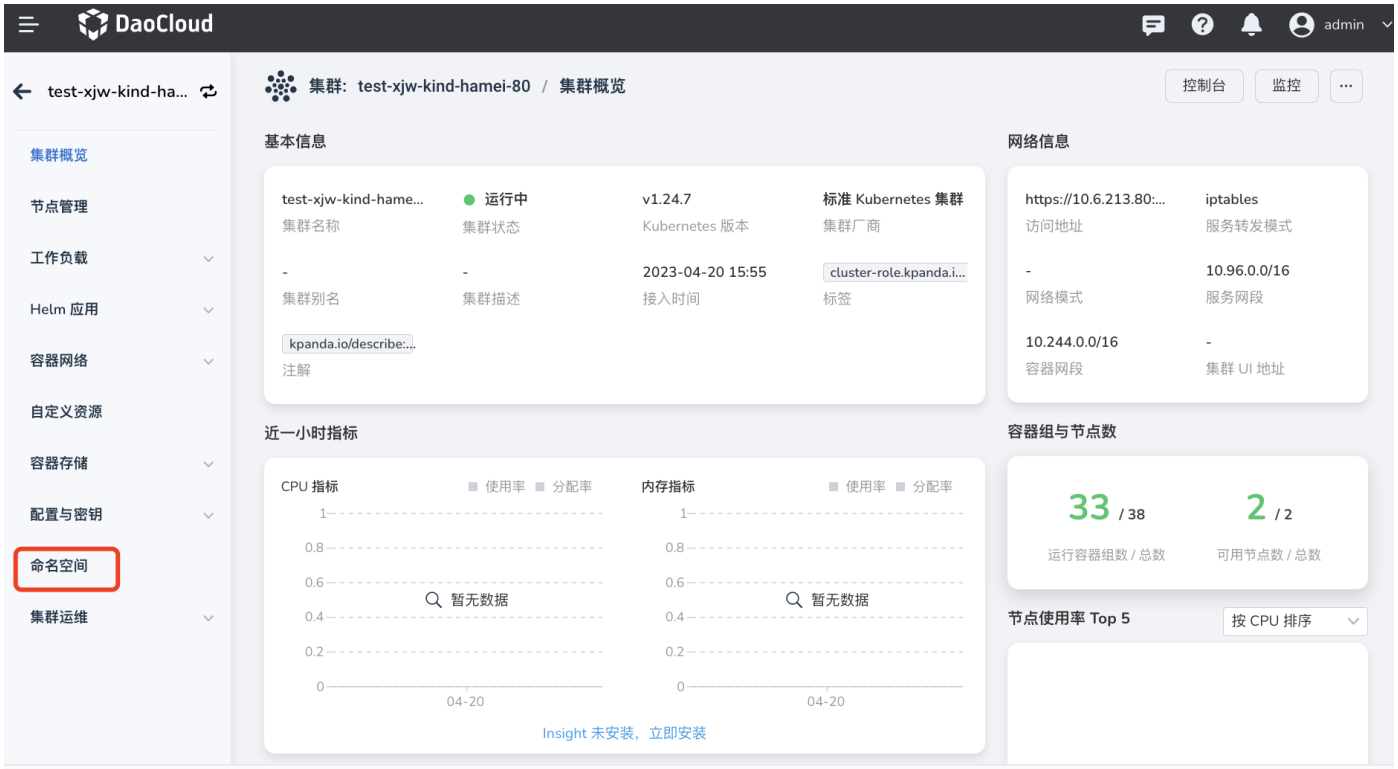
```
kubectl annotate ns <namespace-name> scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "effect":
"NoSchedule", "key": "ExclusiveNamespace"}]'
```

请确保将 `<namespace-name>` 替换为要添加注解的平台命名空间名称。

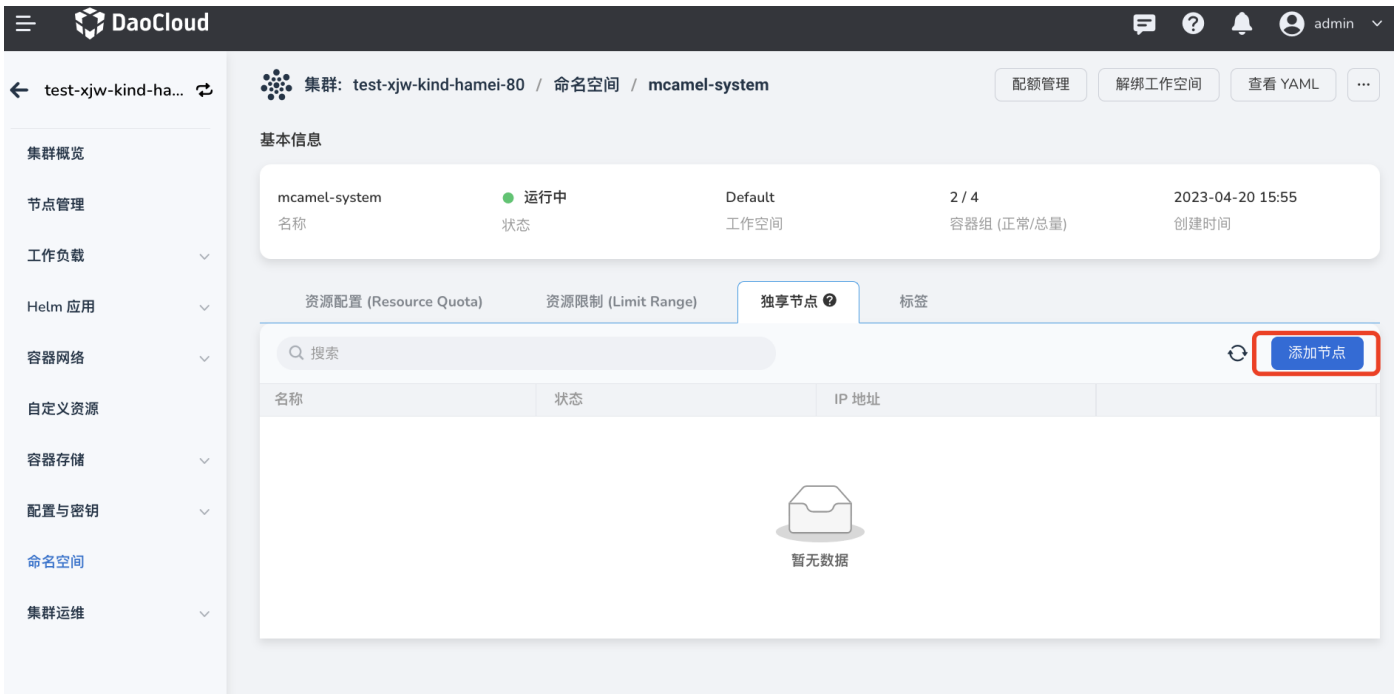
3. 使用界面为命名空间设置独享节点

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 d.run 的 UI 管理界面为命名空间设置独享节点了。

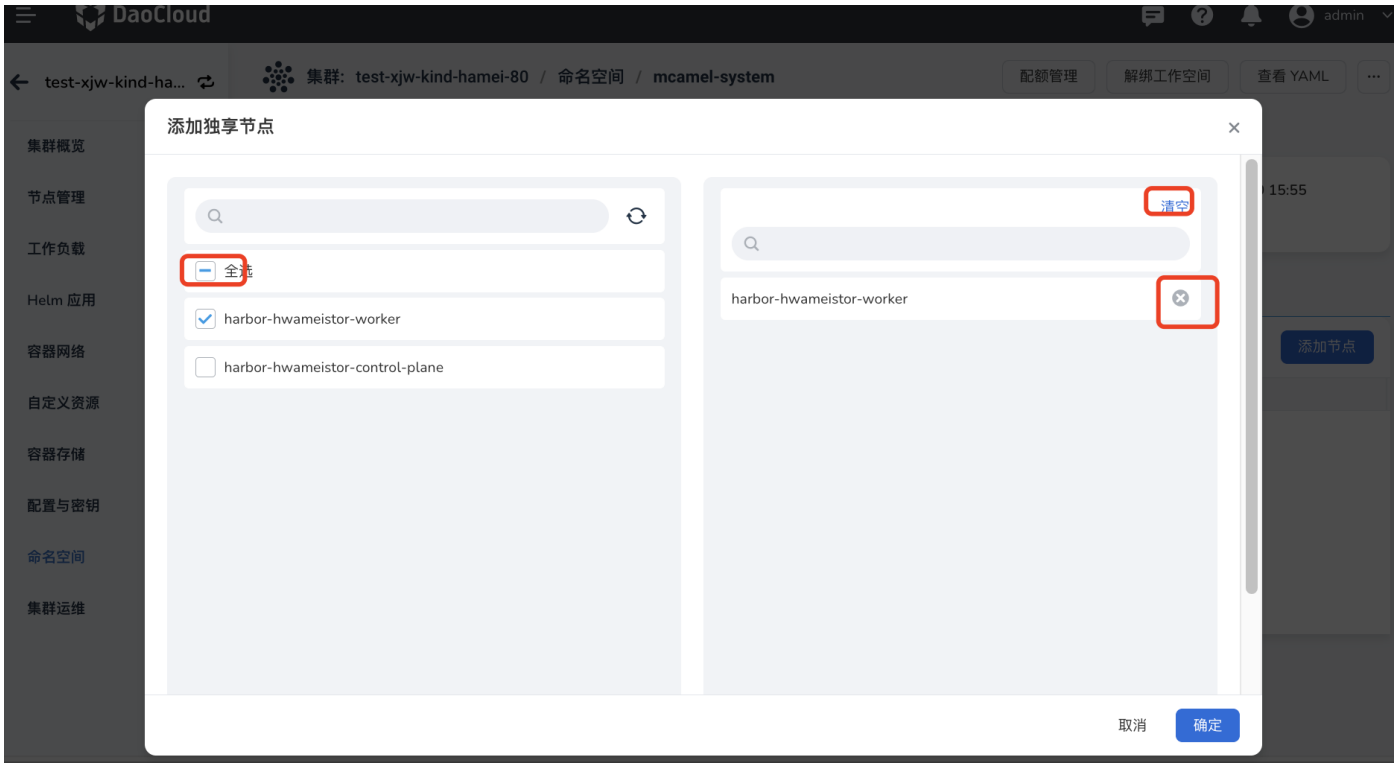
a. 在集群列表页面点击集群名称，然后在左侧导航栏点击 命名空间 。



b. 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 。

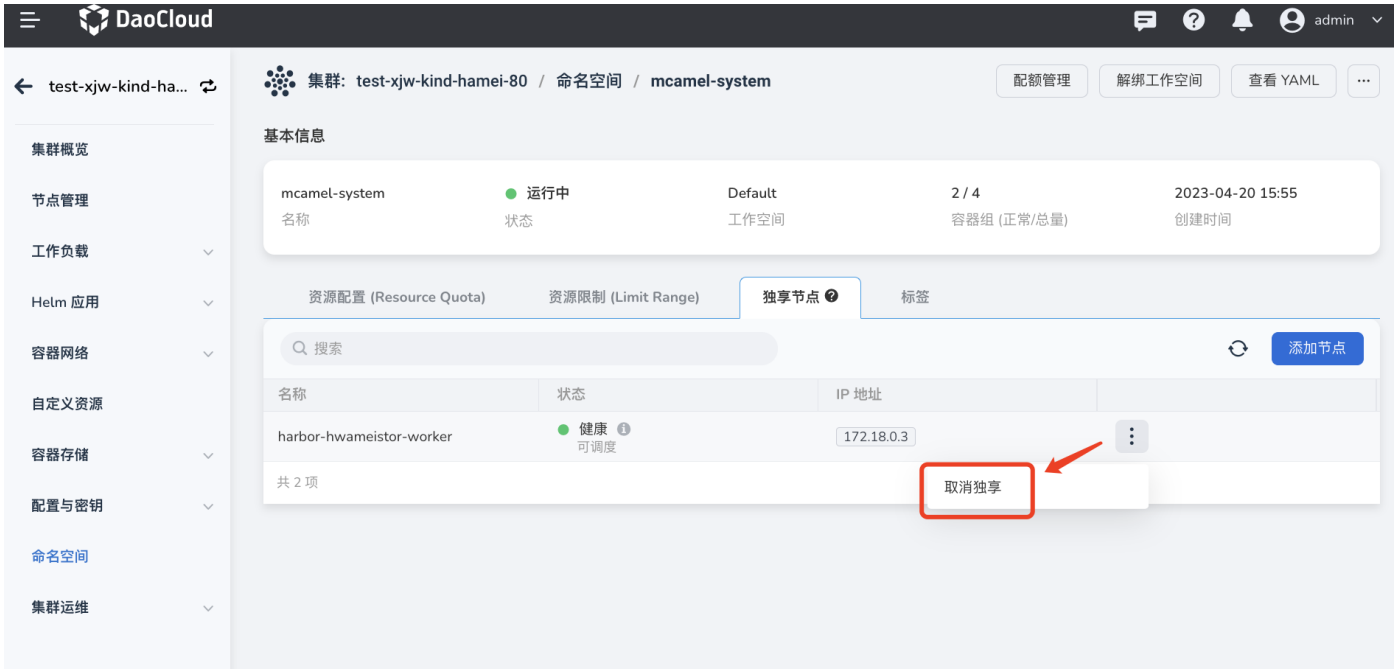


c. 在页面左侧选择让该命名空间独享哪些节点，在右侧可以清空或删除某个已选节点，最后在底部点击 确定 。



d. 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。



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
    ......
```

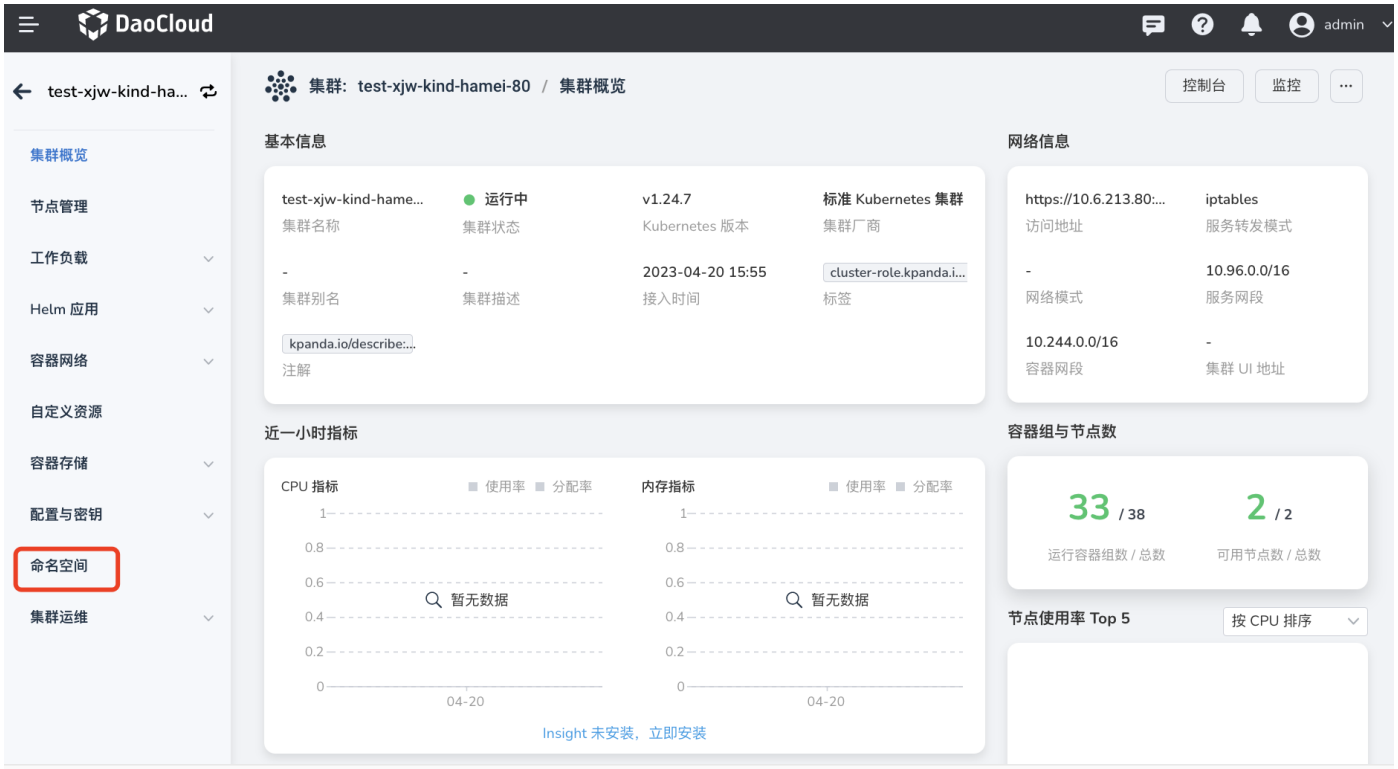
找到 **--enable-admission-plugins** 参数，加入（以英文逗号分隔的） **PodNodeSelector** 和 **PodTolerationRestriction** 准入控制器。参考如下：

```
# 加入 __ ,PodNodeSelector,PodTolerationRestriction__
- --enable-admission-plugins=NodeRestriction,PodNodeSelector,PodTolerationRestriction
```

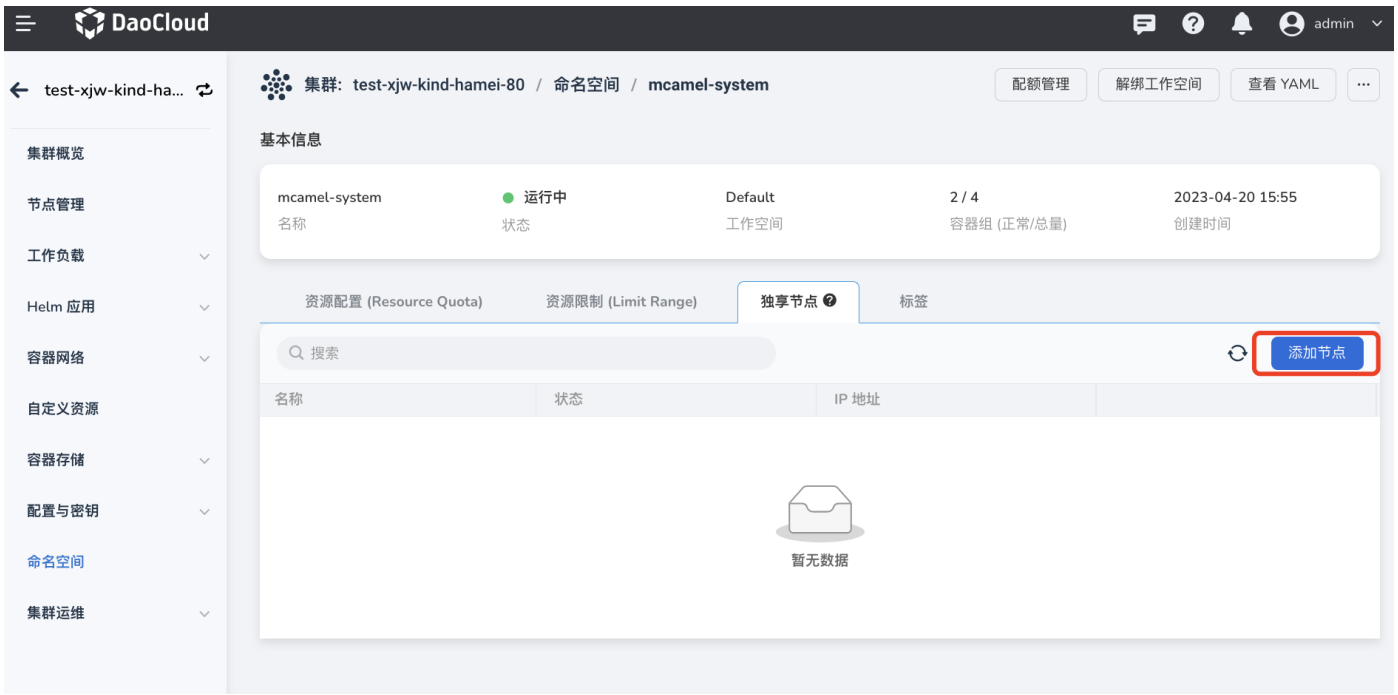
2. 使用界面为命名空间设置独享节点

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 d.run 的 UI 管理界面为命名空间设置独享节点了。

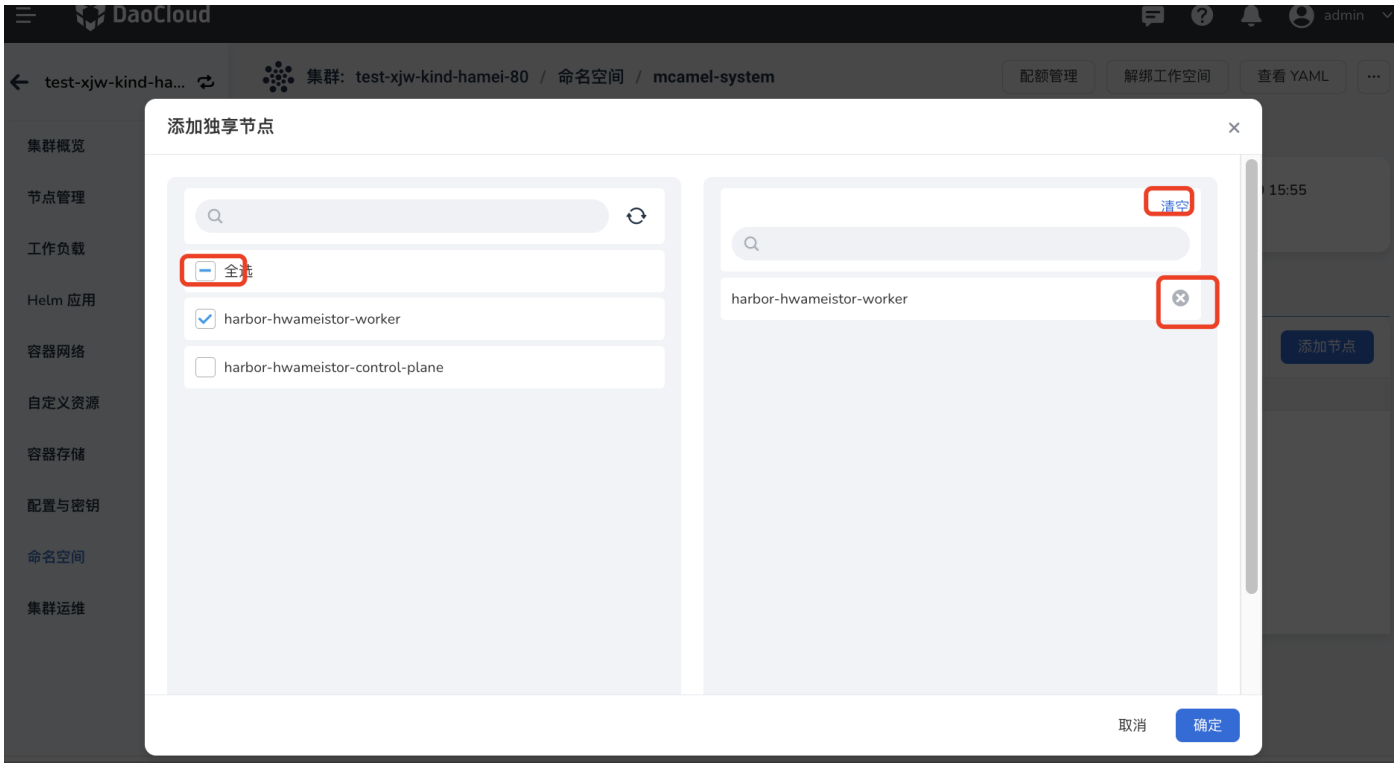
a. 在集群列表页面点击集群名称，然后在左侧导航栏点击 命名空间 。



b. 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 。

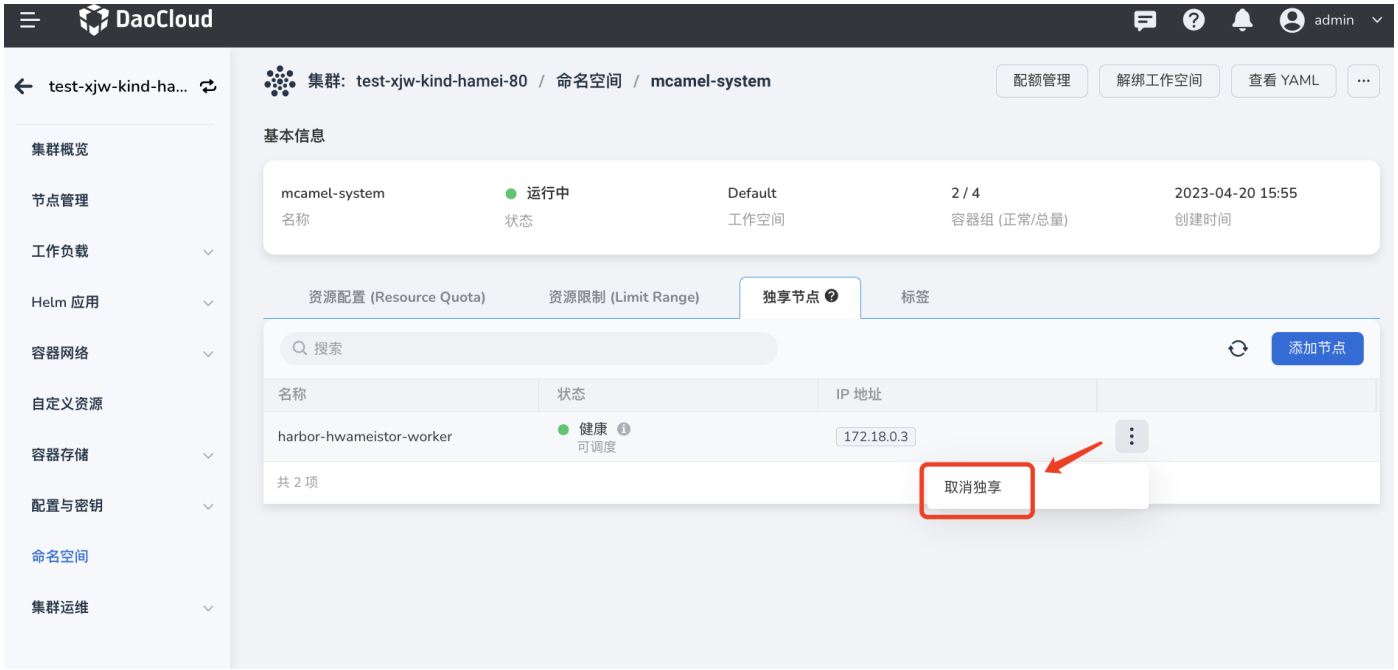


c. 在页面左侧选择让该命名空间独享哪些节点，在右侧可以清空或删除某个已选节点，最后在底部点击 确定 。



d. 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。



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。

POD 安全策略

Pod 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。

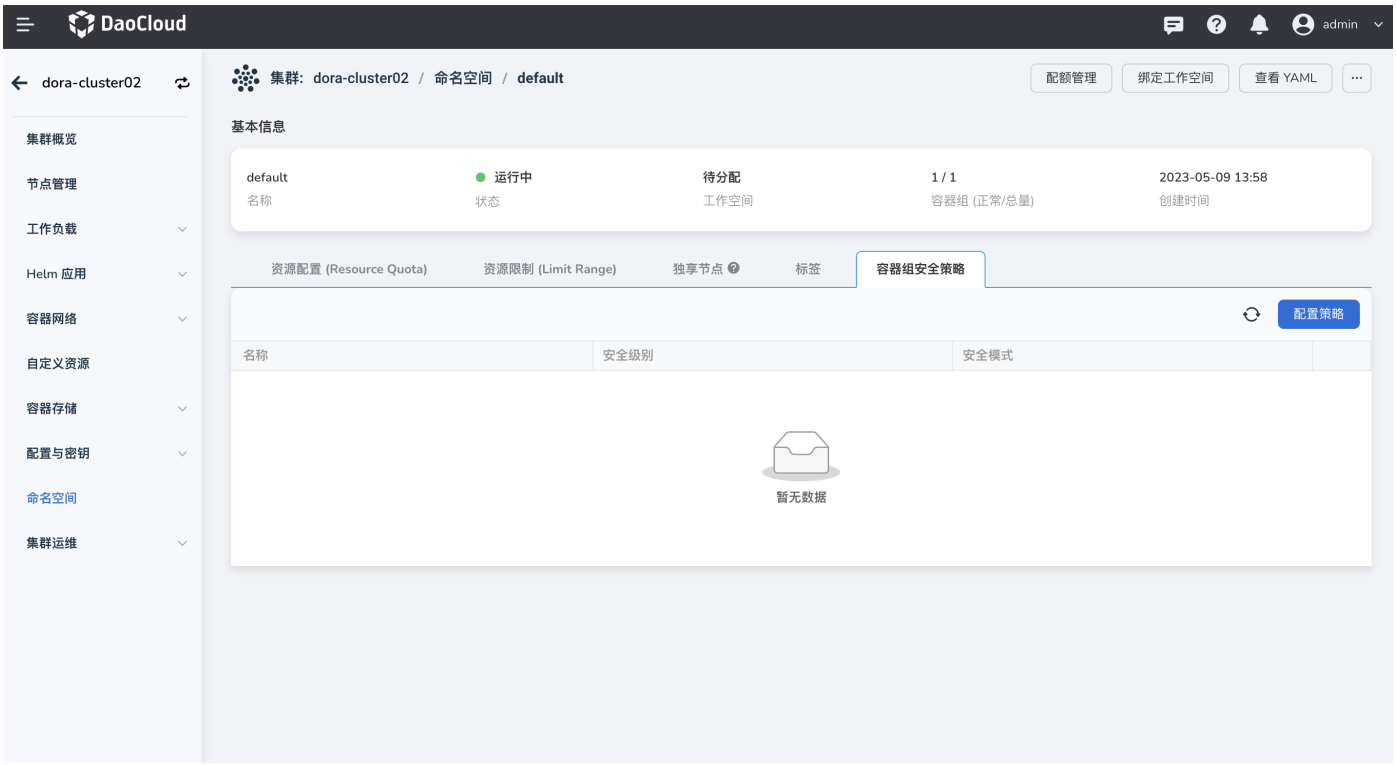
本节将介绍如何通过容器管理界面为命名空间配置 Pod 安全策略。

前提条件

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 Pod 安全策略

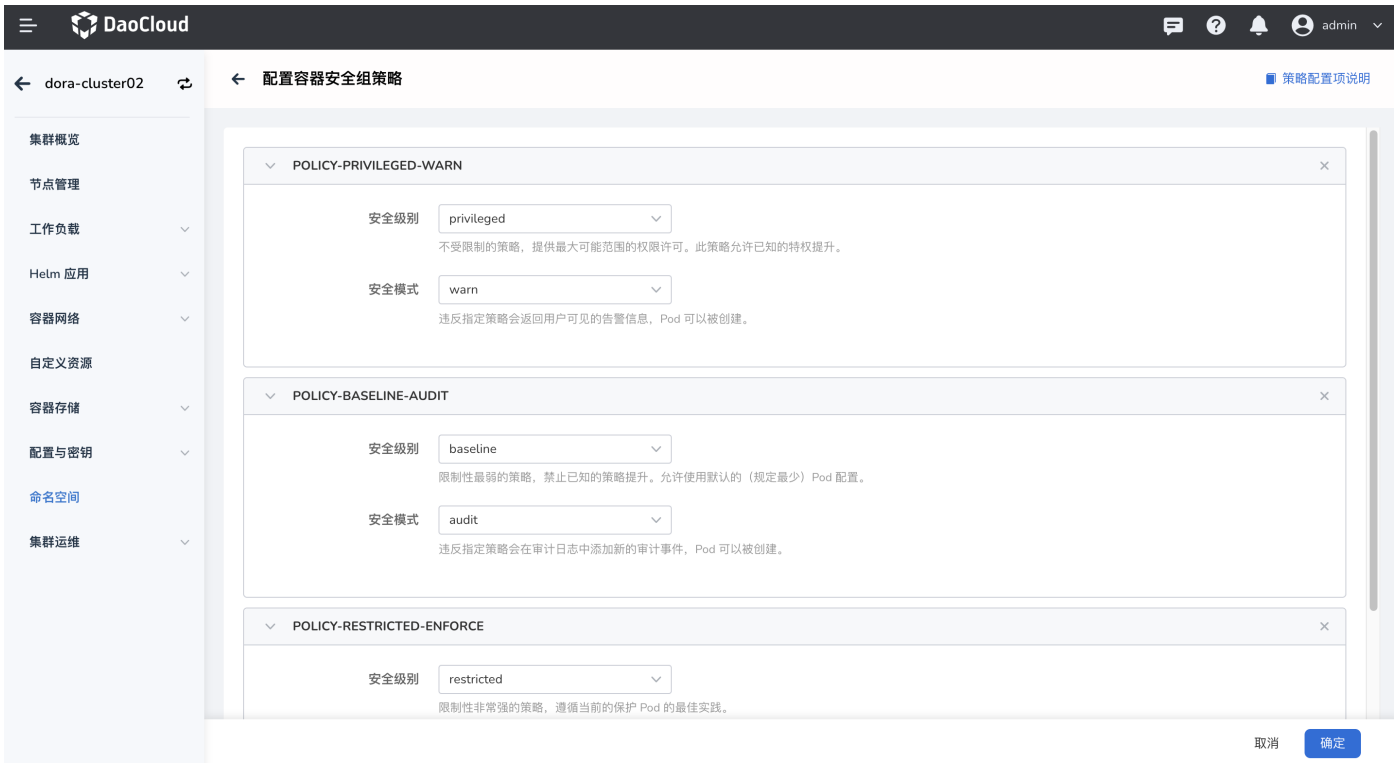
1. 选择需要配置 Pod 安全策略的命名空间，进入详情页。在 **Pod** 安全策略 页面点击 配置策略 ，进入配置页。



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。

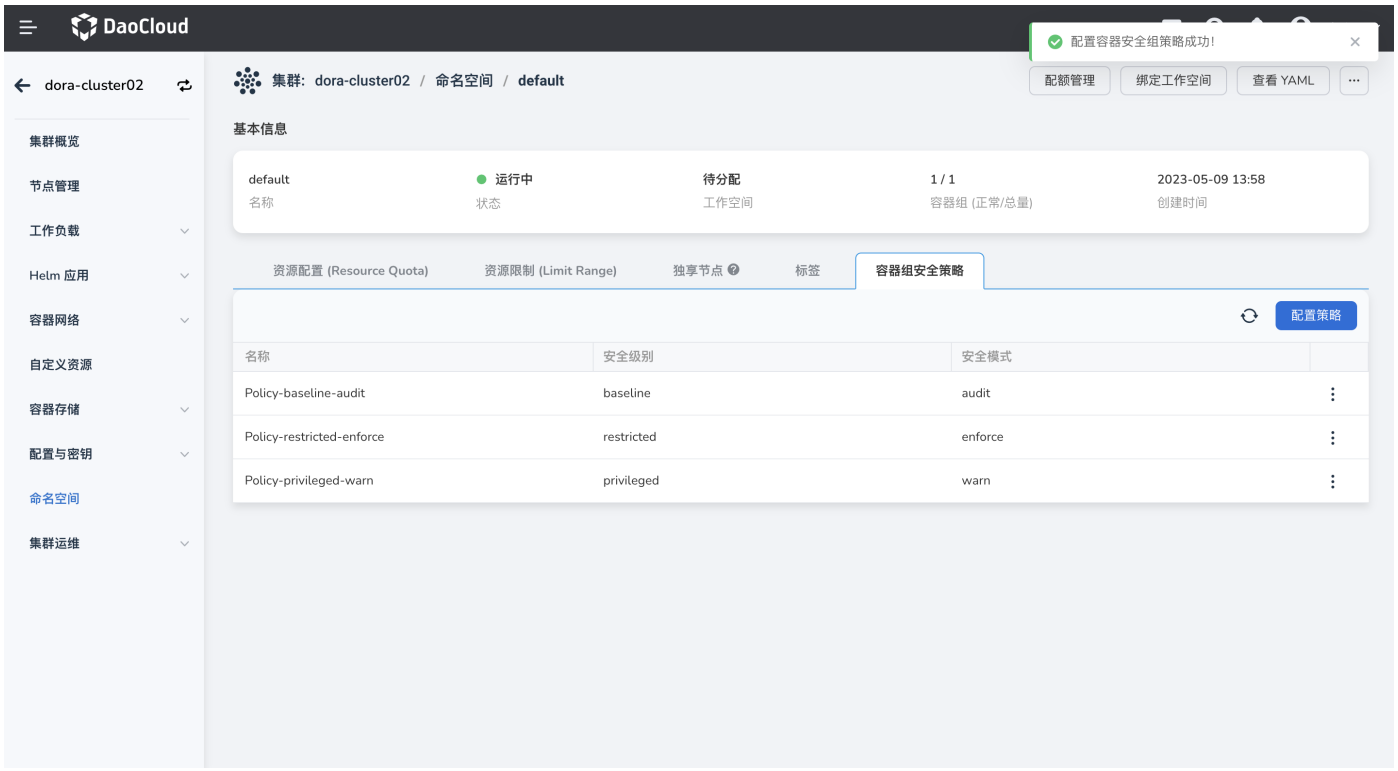
安全级别	描述
Privileged	不受限制的策略，提供最大可能范围的权限许可。此策略允许已知的特权提升。
Baseline	限制性最弱的策略，禁止已知的策略提升。允许使用默认的（规定最少）Pod 配置。
Restricted	限制性非常强的策略，遵循当前的保护 Pod 的最佳实践。

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。
Enforce	违反指定策略会导致 Pod 无法创建。

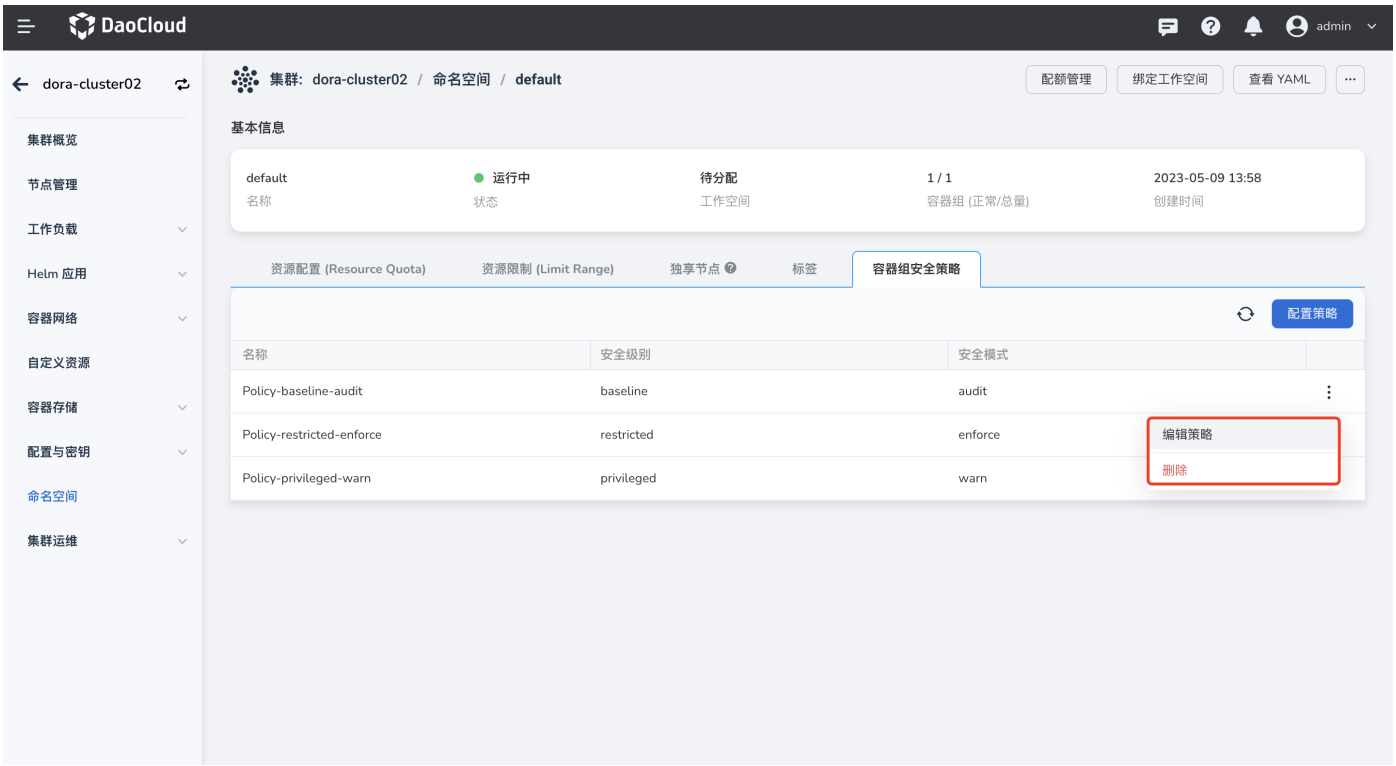


3. 不同的安全级别对应不同的检查项，若您不知道该如何为您的命名空间配置，可以点击页面右上角的 策略配置项说明 查看详细信息。

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。



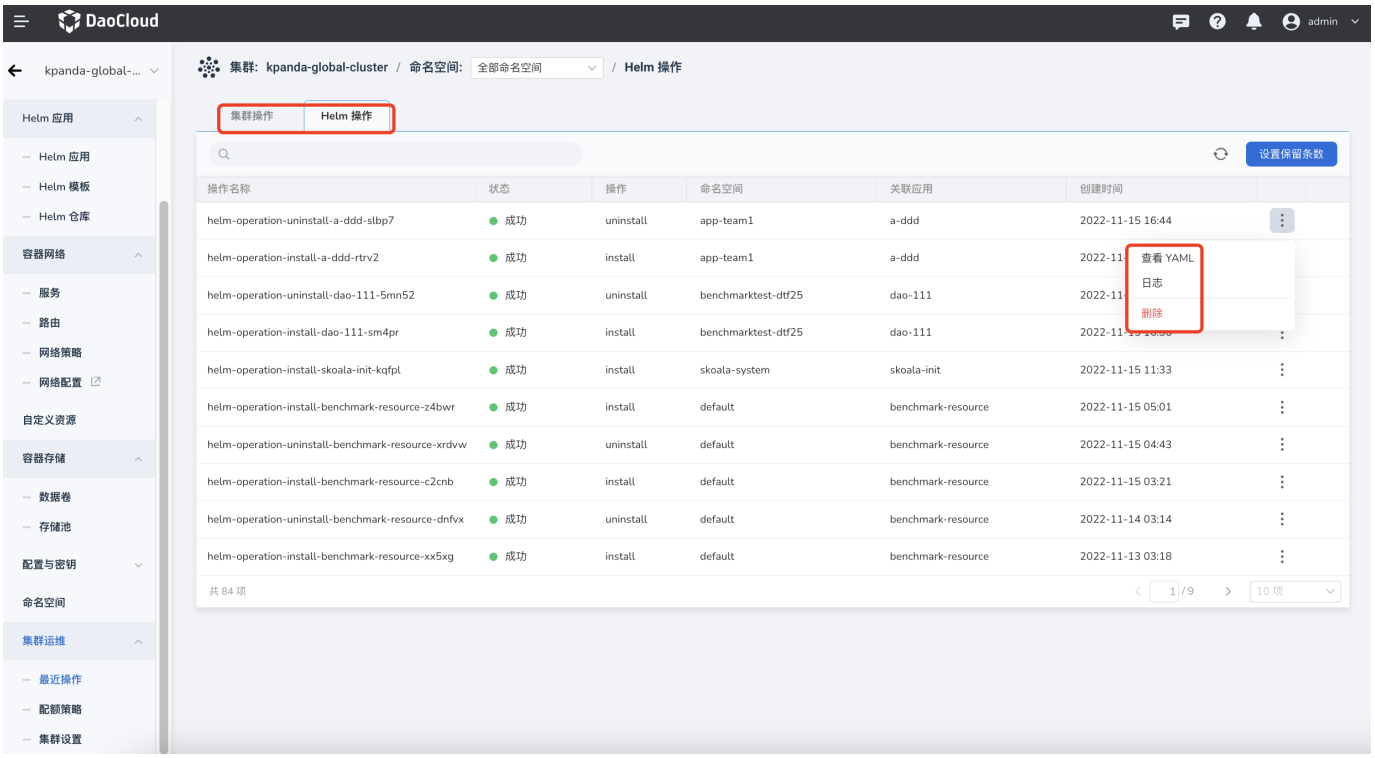
5. 点击 ⋮ 还可以编辑或者删除您配置的安全策略。


DaoCloud
admin
dora-cluster02
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: dora-cluster02 / 命名空间 / default
配额管理
绑定工作空间
查看 YAML
基本信息
default
名称
运行中
状态
待分配
工作空间
1 / 1
容器组 (正常/总量)
2023-05-09 13:58
创建时间
资源配置 (Resource Quota)
资源限制 (Limit Range)
独享节点
标签
容器组安全策略
配置策略
名称
安全级别
安全模式
Policy-baseline-audit
baseline
audit
Policy-restricted-enforce
restricted
enforce
Policy-privileged-warn
privileged
warn
编辑策略
删除

集群运维

最近操作

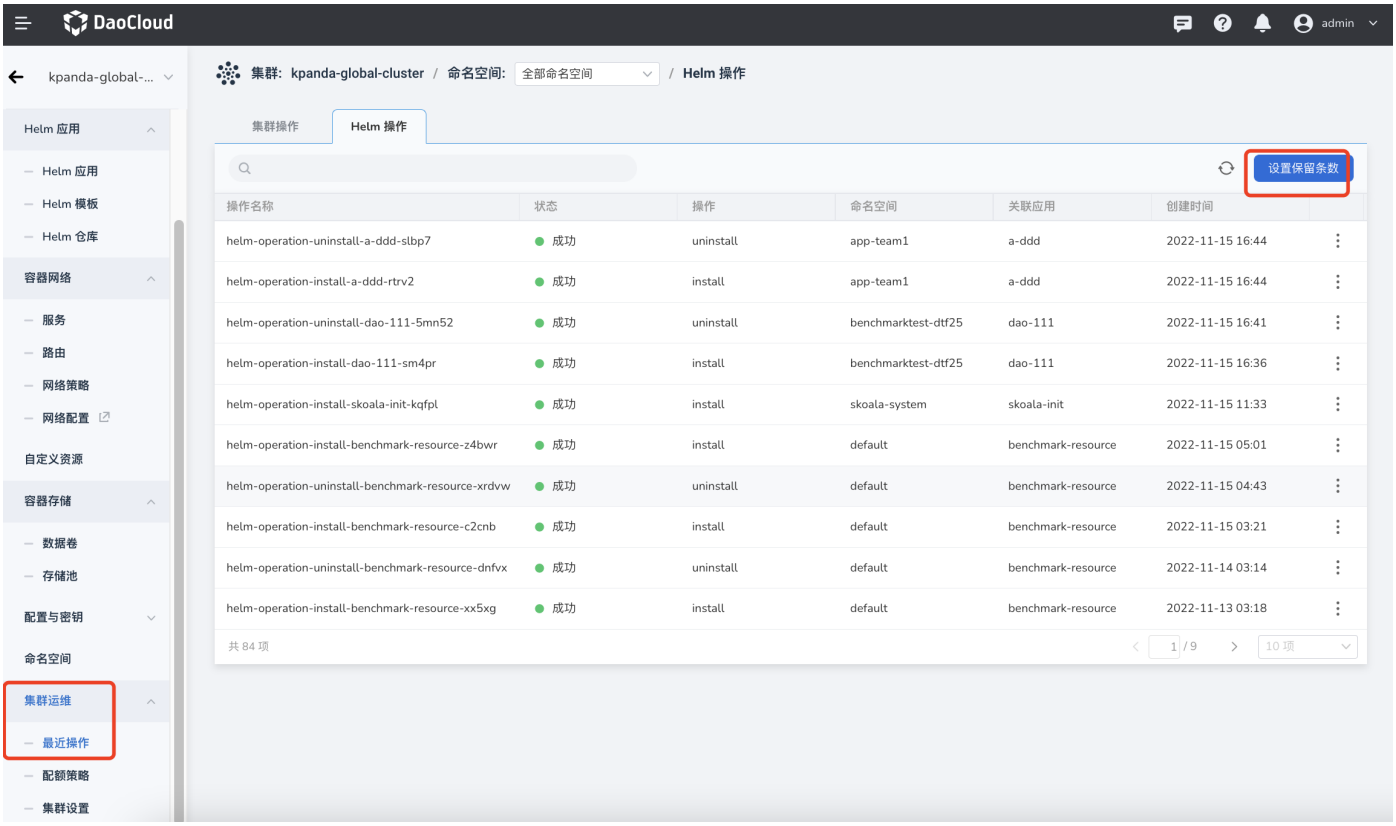
在该页面可以查看最近的集群操作记录和 Helm 操作记录，以及各项操作的 YAML 文件和日志，也可以删除某一条记录。



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 。



2. 设置需要保留多少条 Helm 操作记录，并点击 确定 。

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。

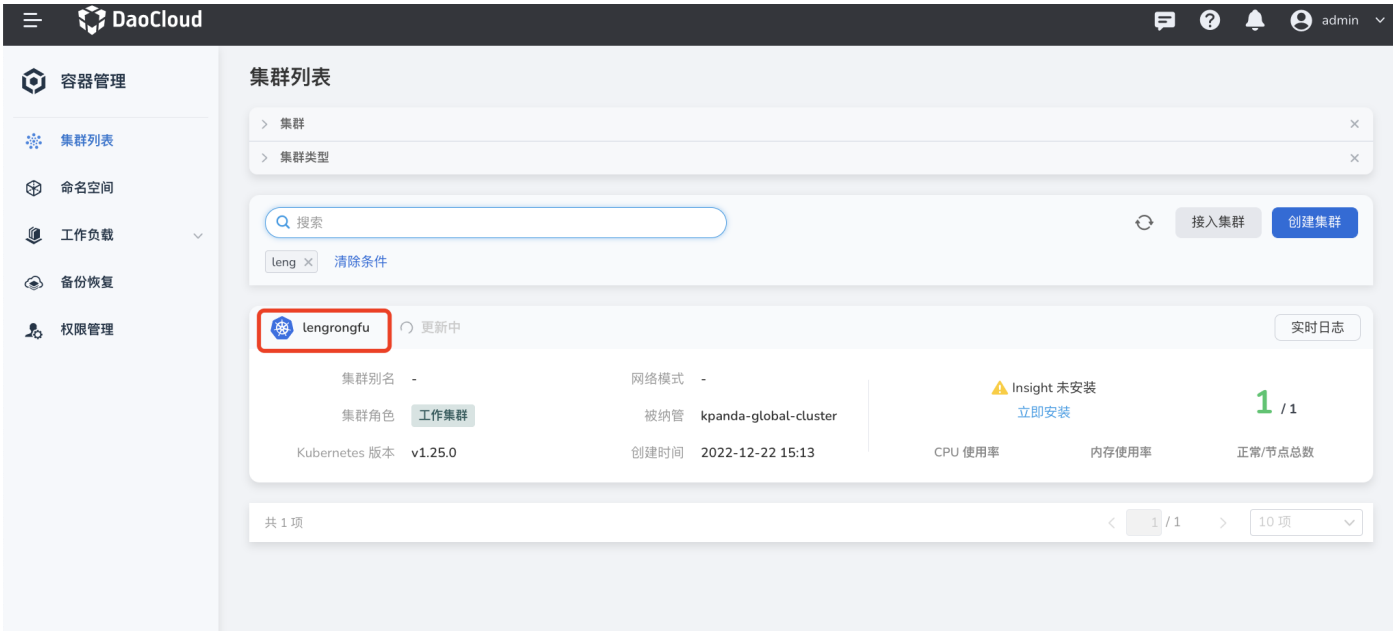
Danger

版本升级后将无法回退到之前的版本，请谨慎操作。

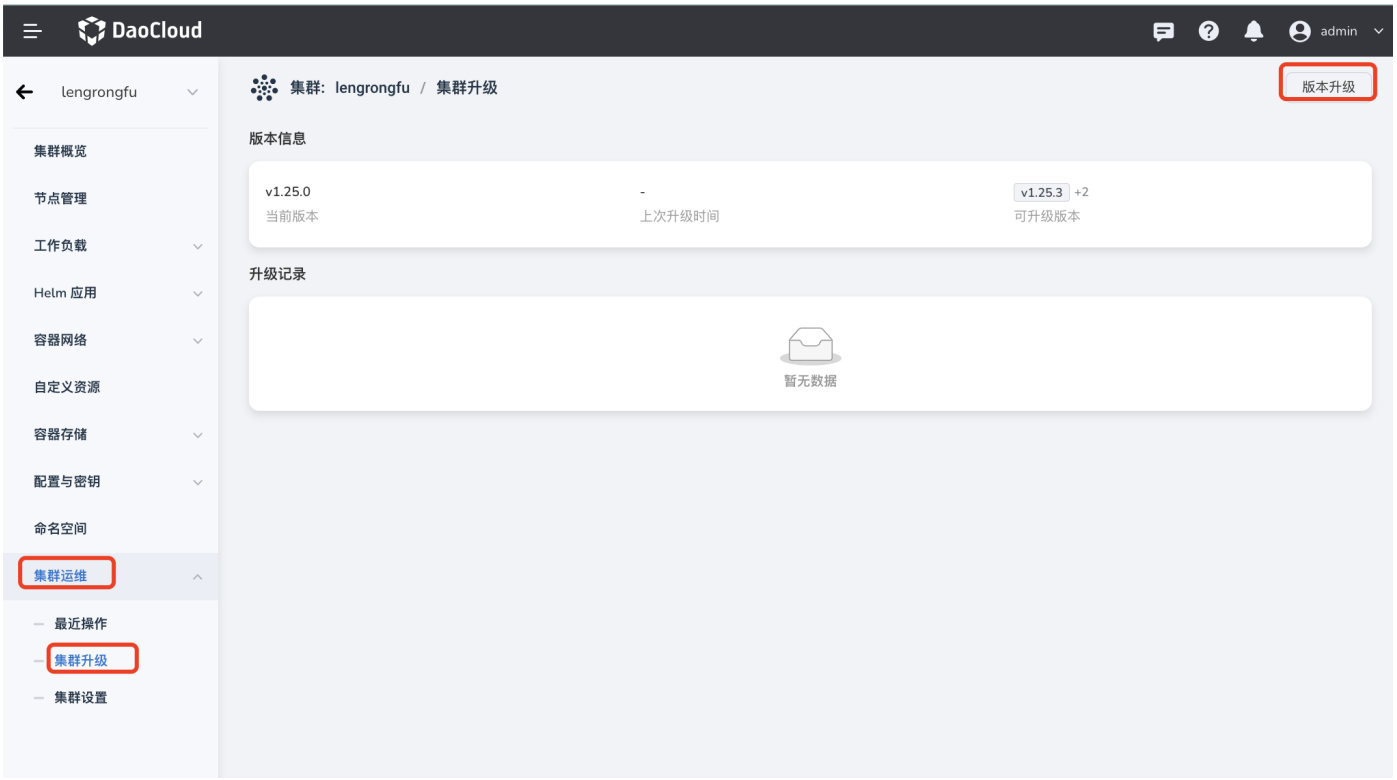
Note

- 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。如果左侧导航栏没有 集群升级 ，请检查该集群是否为 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应该已经接入容器管理模块，并且处于正常运行中。
- 如果需要修改集群参数，可以通过升级相同版本的方式实现，具体操作参考下文。

1. 在集群列表中点击目标集群的名称。



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 。



3. 选择可升级的版本，输入集群名称进行确认。

Note

如果您是想通过升级方式来修改集群参数，请参考以下步骤：

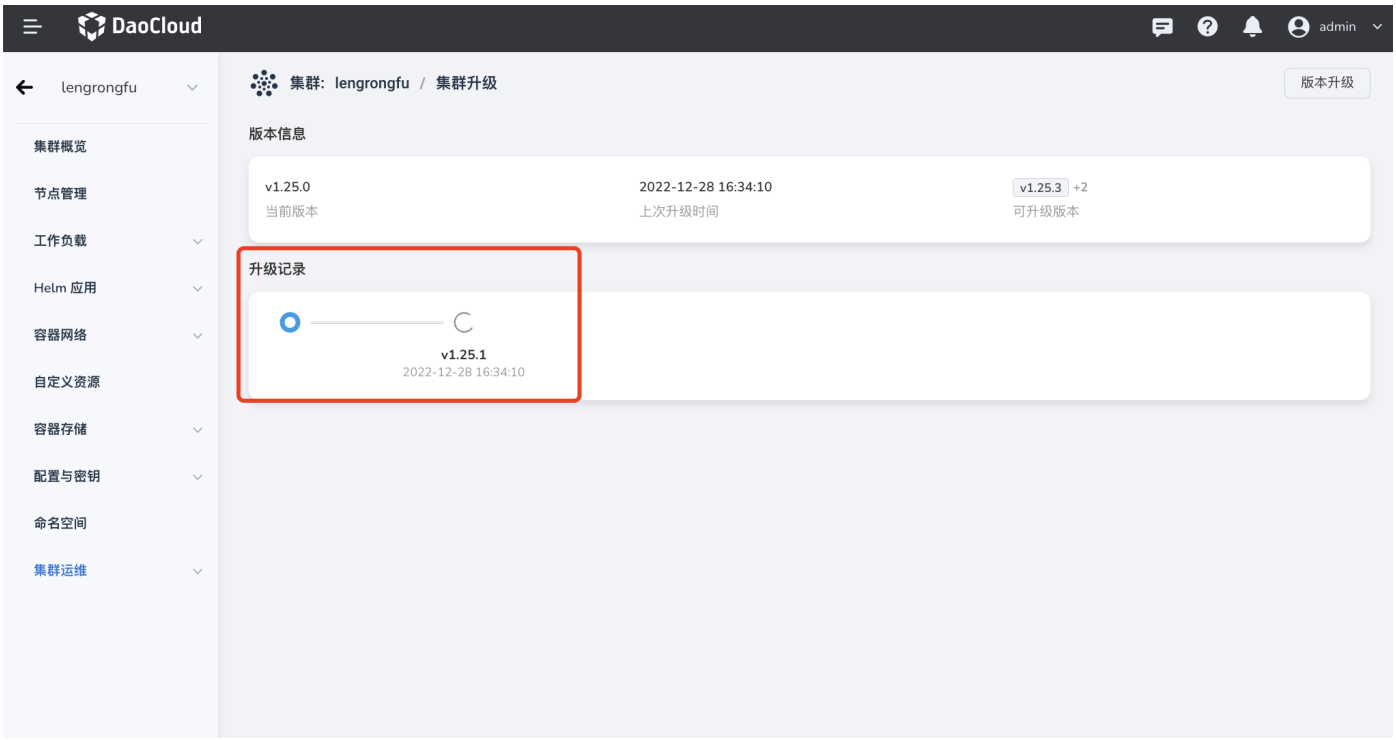
a. 找到集群对应的 ConfigMap，您可以登录控制节点执行如下命令，找到 varsConfRef 中的 ConfigMap 名称。

```
kubectl get cluster.kubean.io <clustername> -o yaml
```

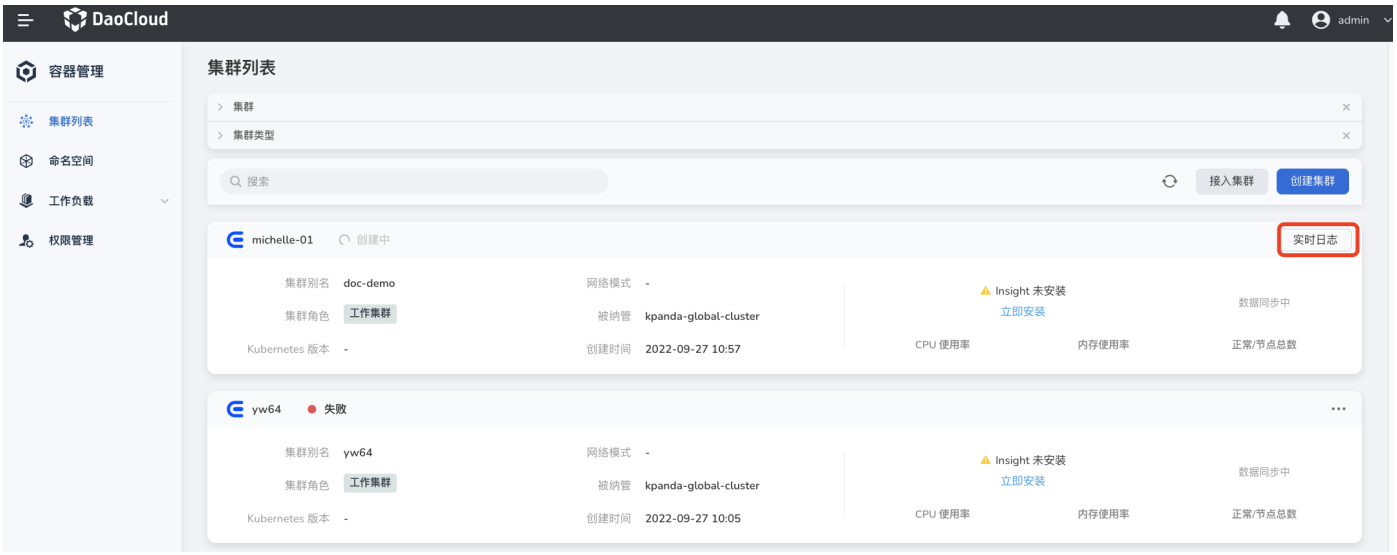
b. 根据需要，修改 ConfigMap 中的参数信息。

c. 在此处选择相同版本进行升级操作，升级完成即可成功更新对应的集群参数。

4. 点击 确定 后，可以看到集群的升级进度。



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。

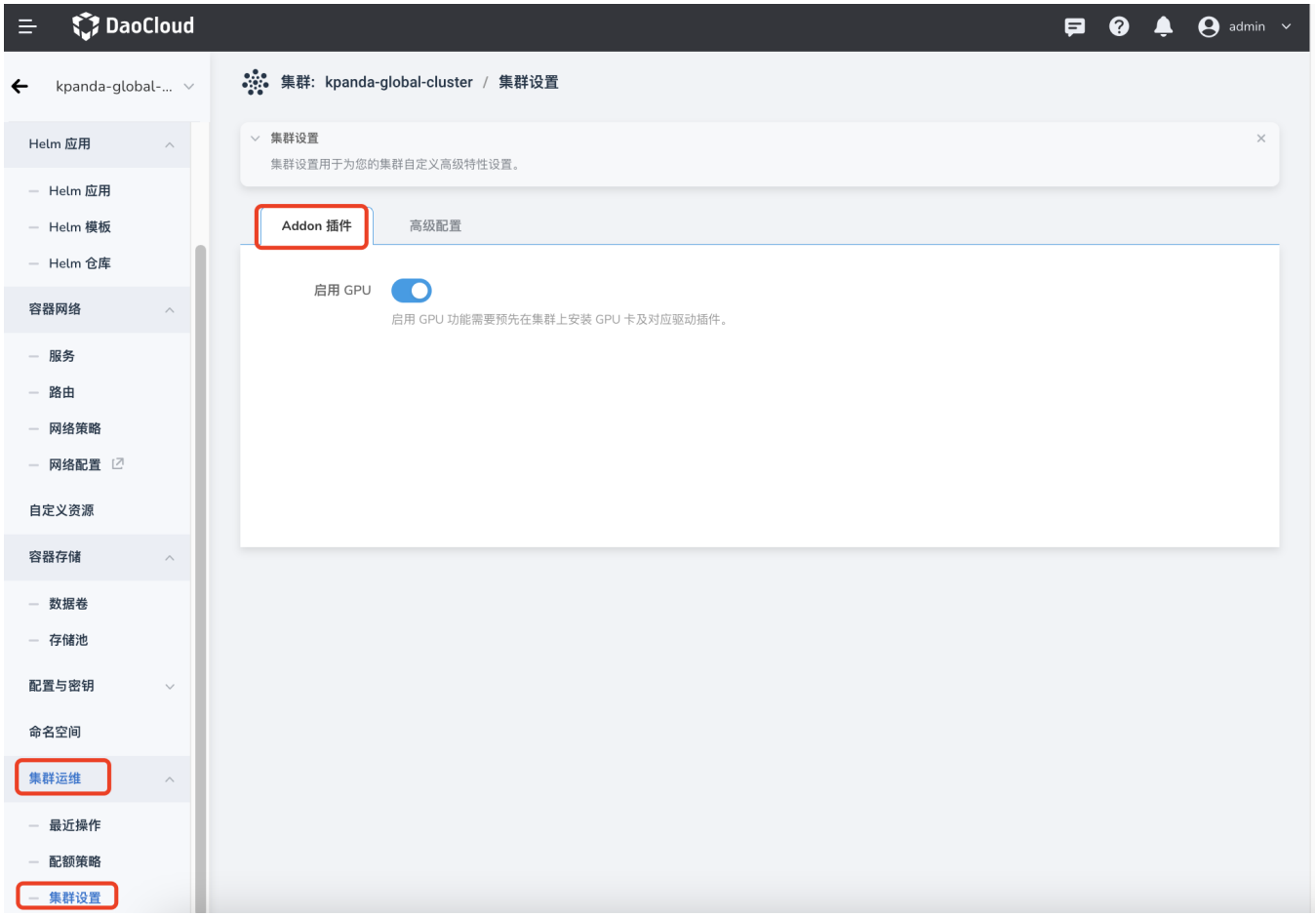

DaoCloud
admin
容器管理
集群列表
命名空间
工作负载
权限管理
集群列表
集群
集群类型
搜索
接入集群
创建集群
michelle-01
创建中
实时日志
集群别名 doc-demo
网络模式 -
集群角色 工作集群
被纳管 kpanda-global-cluster
Kubernetes 版本 -
创建时间 2022-09-27 10:57
Insight 未安装
立即安装
数据同步中
CPU 使用率
内存使用率
正常/节点总数
yw64
失败
集群别名 yw64
网络模式 -
集群角色 工作集群
被纳管 kpanda-global-cluster
Kubernetes 版本 -
创建时间 2022-09-27 10:05
Insight 未安装
立即安装
数据同步中
CPU 使用率
内存使用率
正常/节点总数

集群设置

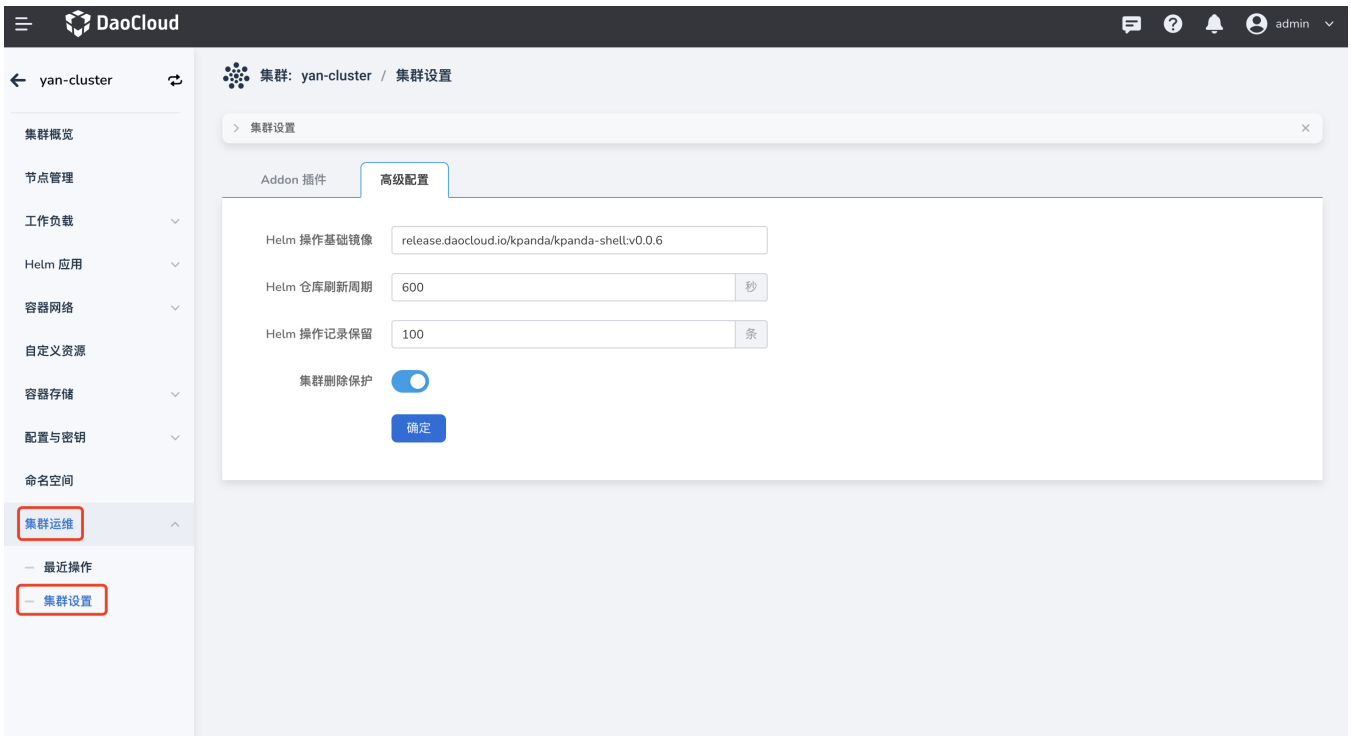
集群设置用于为您的集群自定义高级特性设置，包括是否启用 GPU、Helm 仓库刷新周期、Helm 操作记录保留等。

- 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 集群运维 -> 最近操作 -> 集群操作 。



- 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）



GPU 管理

GPU 管理概述

本文介绍 Daocloud 容器管理平台对 GPU为代表的异构资源统一运维管理能力。

背景

随着 AI 应用、大模型、人工智能、自动驾驶等新兴技术的快速发展，企业面临着越来越多的计算密集型任务和数据处理需求。 以 CPU 为代表的传统计算架构已无法满足企业日益增长的计算需求。此时，以 GPU 为代表的异构计算因在处理大规模数据、进行复杂计算和实时图形渲染方面具有独特的优势被广泛应用。

与此同时，由于缺乏异构资源调度管理等方面的经验和专业的解决方案，导致了 GPU 设备的资源利用率极低，给企业带来了高昂的 AI 生产成本。 如何降本增效，提高 GPU 等异构资源的利用效率，成为了当前众多企业亟需跨越的一道难题。

GPU 能力介绍

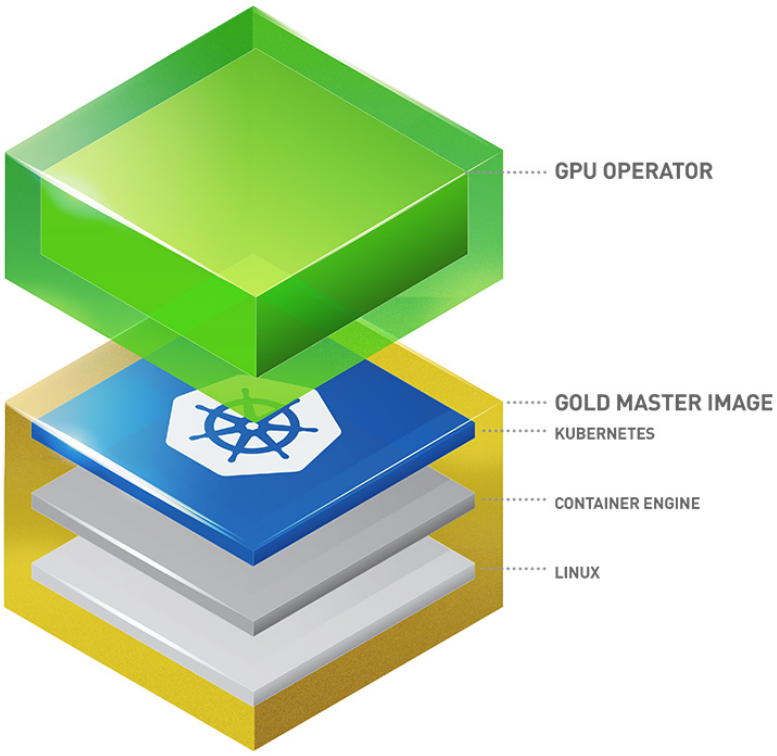
Daocloud 容器管理平台支持对 GPU、NPU 等异构资源进行统一调度和运维管理，充分释放 GPU 资源算力，加速企业 AI 等新兴应用发展。Daocloud GPU 管理能力如下：

- 支持统一纳管 NVIDIA、华为昇腾、天数等国内外厂商的异构计算资源。
- 支持同一集群多卡异构调度，并支持集群 GPU 卡自动识别。
- 支持 NVIDIA GPU、vGPU、MIG 等 GPU 原生管理方案，并提供云原生能力。
- 支持单块物理卡切分给不同的租户使用，并支持对租户和容器使用 GPU 资源按照算力、显存进行 GPU 资源配额。
- 支持集群、节点、应用等多维度 GPU 资源监控，帮助运维人员管理 GPU 资源。
- 兼容 TensorFlow、pytorch 等多种训练框架。

GPU Operator 介绍

同普通计算机硬件一样，NVIDIA GPU 卡作为物理硬件，必须安装 NVIDIA GPU 驱动后才能使用。 为了降低用户在 kuberneets 上使用 GPU 的成本，NVIDIA 官方提供了 NVIDIA GPU Operator 组件来管理使用 NVIDIA GPU 所依赖的各种组件。 这些组件包括 NVIDIA 驱动程序（用于启用 CUDA）、NVIDIA 容器运行时、GPU 节点标记、基于 DCGM 的监控等。 理论上来说用户只需要将 GPU 卡插在已经被 kubernetes 所纳管的计算设备上，然后通过 GPU Operator 就能使用 NVIDIA GPU 的所有能力了。 了解更多 NVIDIA GPU Operator 相关信息，请参考 NVIDIA 官方文档。 如何部署请参考 GPU Operator 离线安装

NVIDIA GPU Operator 架构图：


GPU OPERATOR
GOLD MASTER IMAGE
KUBERNETES
CONTAINER ENGINE
LINUX

GPU 支持矩阵

本页说明 d.run 支持的 GPU 及操作系统所对应的矩阵。

NVIDIA GPU

GPU 厂商及类型	支持 **GPU** 型号	适配的操作系统 **(**在线**)**	推荐内核	推荐的操作系统及内核	安装文档
NVIDIA GPU（整卡/**vGPU**）	• NVIDIA Fermi (2.1)架构： • NVIDIA GeForce 400 系列 • NVIDIA Quadro 4000 系列 • NVIDIA Tesla 20 系列 • NVIDIA Ampere 架构系列 (A100;A800;H100)	CentOS 7	• Kernel 3.10.0-123 ~ 3.10.0-1160 • 内核参考文档 • 建议使用操作系统对应 **Kernel** 版本	操作系统：CentOS 7.9；内核版本：**3.10.0-1160**	GPU Operator 离线安装
		CentOS 8	Kernel 4.18.0-80 ~ 4.18.0-348		
		Ubuntu 20.04	Kernel 5.4		
		Ubuntu 22.04	Kernel 5.19		
		RHEL 7	Kernel 3.10.0-123 ~ 3.10.0-1160		
		RHEL 8	Kernel 4.18.0-80 ~ 4.18.0-348		
NVIDIA MIG	• Ampere 架构系列： • A100 • A800 • H100	CentOS 7	Kernel 3.10.0-123 ~ 3.10.0-1160	操作系统：CentOS 7.9；内核版本：**3.10.0-1160**	GPU Operator 离线安装
		CentOS 8	Kernel 4.18.0-80 ~ 4.18.0-348		
		Ubuntu 20.04	Kernel 5.4		
		Ubuntu 22.04	Kernel 5.19		
		RHEL 7	Kernel 3.10.0-123 ~ 3.10.0-1160		
		RHEL 8	Kernel 4.18.0-80 ~ 4.18.0-348		

昇腾（Ascend）NPU

GPU 厂商及类型	支持 **NPU** 型号	适配的操作系统 **(**在线**)**	推荐内核	推荐的操作系统及内核	安装文档
昇腾**(Ascend 310)**	• Ascend 310; • Ascend 310P；	Ubuntu20.04	详情参考：内核版本要求	操作系统：CentOS 7.9；内核版本：**3.10.0-1160**	300 和 310P 驱动文档
		CentOS 7.6			
		CentOS 8.2			
		KylinV10SP1 操作系统			
		openEuler 操作系统			
昇腾**(Ascend 910P)**	Ascend 910	Ubuntu20.04	详情参考内核版本要求	操作系统：CentOS 7.9；内核版本：**3.10.0-1160**	910 驱动文档
		CentOS 7.6			
		CentOS 8.2			
		KylinV10SP1 操作系统			
		openEuler 操作系统			

天数智芯（Iluvatar） GPU

GPU 厂商及类型	支持 的 **GPU** 型号	适配的操作系统 **(**在线**)**	推荐内核	推荐的操作系统及内核	安装文档
天数智芯 **(Iluvatar vGPU)**	• BI100; • MR100；	CentOS 7	• Kernel 3.10.0-957.el7.x86_64 ~ 3.10.0-1160.42.2.el7.x86_64	操作系统：CentOS 7.9；内核版本：**3.10.0-1160**	补充中
		CentOS 8	• Kernel 4.18.0-80.el8.x86_64 ~ 4.18.0-305.19.1.el8_4.x86_64		
		Ubuntu 20.04	• Kernel 4.15.0-20-generic ~ 4.15.0-160-generic • Kernel 5.4.0-26-generic ~ 5.4.0-89-generic • Kernel 5.8.0-23-generic ~ 5.8.0-63-generic		
		Ubuntu 21.04	• Kernel 4.15.0-20-generic ~ 4.15.0-160-generic • Kernel 5.4.0-26-generic ~ 5.4.0-89-generic • Kernel 5.8.0-23-generic ~ 5.8.0-63-generic		
		openEuler 22.03 LTS	• Kernel 版本大于等于 5.1 且小于等于 5.10		

NVIDIA GPU 管理

NVIDIA GPU 卡使用模式

NVIDIA 作为业内知名的图形计算供应商，为算力的提升提供了诸多软硬件解决方案，其中 NVIDIA 在 GPU 的使用方式上提供了如下三种解决方案：

整卡（Full GPU）

整卡是指将整个 NVIDIA GPU 分配给单个用户或应用程序。在这种配置下，应用可以完全占用 GPU 的所有资源， 并获得最大的计算性能。整卡适用于需要大量计算资源和内存的工作负载，如深度学习训练、科学计算等。

vGPU（Virtual GPU）

vGPU 是一种虚拟化技术，允许将一个物理 GPU 划分为多个虚拟 GPU，每个虚拟 GPU 分配给不同的虚拟机或用户。 vGPU 使多个用户可以共享同一台物理 GPU，并在各自的虚拟环境中独立使用 GPU 资源。 每个虚拟 GPU 可以获得一定的计算能力和显存容量。vGPU 适用于虚拟化环境和云计算场景，可以提供更高的资源利用率和灵活性。

MIG（Multi-Instance GPU）

MIG 是 NVIDIA Ampere 架构引入的一项功能，它允许将一个物理 GPU 划分为多个物理 GPU 实例，每个实例可以独立分配给不同的用户或工作负载。 每个 MIG 实例具有自己的计算资源、显存和 PCIe 带宽，就像一个独立的虚拟 GPU。 MIG 提供了更细粒度的 GPU 资源分配和管理，可以根据需求动态调整实例的数量和大小。 MIG 适用于多租户环境、容器化应用程序和批处理作业等场景。

无论是在虚拟化环境中使用 vGPU，还是在物理 GPU 上使用 MIG，NVIDIA 为用户提供了更多的选择和优化 GPU 资源的方式。 Daocloud 容器管理平台全面兼容了上述 NVIDIA 的能力特性，用户只需通过简单的界面操作，就能够获得全部 NVIDIA GPU 的计算能力，从而提高资源利用率并降低成本。

- **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

开启配置详情参考 GPU Operator 离线安装。

如何使用

您可以参考以下链接，快速使用 Daocloud 关于 NVIDIA GPU 卡的管理能力。

- **NVIDIA GPU** 整卡使用
- **NVIDIA vGPU** 使用
- **NVIDIA MIG** 使用

GPU Operator 离线安装

GPU Operator 离线安装

d.run 预置了 CentOS 7.9，内核为 3.10.0-1160 的 GPU Operator 离线包。 本文介绍如何离线部署 GPU Operator，覆盖 NVIDIA GPU 以下几种使用模式的参数配置：

- 整卡模式
- vGPU 模式
- MIG 模式

> **Note**
>
> 安装后不支持从 MIG 模式切换为整卡模式或 vGPU 模式，仅支持整卡模式与 vGPU 模式的一键切换，请提前规划好您的使用模式。

详情请参考：NVIDIA GPU 卡使用模式，本文使用的 AMD 架构的 Centos 7.9 （3.10.0-1160）进行演示。如需使用 redhat8.4 部署， 请参考向火种节点仓库上传 Red Hat GPU Opreator 离线镜像和构建 Red Hat 8.4 离线 yum 源。

前提条件

1. 用户已经在平台上安装了 v0.12.0 及以上版本的 addon 离线包。
2. 待部署 GPU Operator 的集群节点内核版本必须完全一致。节点 发行版和 GPU 卡型号在 GPU 支持矩阵 范围内。

操作步骤

参考如下步骤为集群安装 GPU Operator 插件。

1. 登录平台，进入 容器管理 -> 待安装 **GPU Operator** 的集群 -> 进入集群详情。
2. 在 **Helm** 模板 页面，选择 全部仓库 ，搜索 **gpu-operator** 。
3. 选择 **gpu-operator** ， 点击 安装 。
4. 参考下文参数配置，配置 **gpu-operator** 安装参数，完成 **gpu-operator** 的安装。

参数配置 基本参数配置

- 名称 ： 输入插件名称。
- 命名空间 ： 选择将插件安装的命名空间。
- 版本 ： 插件的版本，此处以 **23.6.10** 版本为例。
- 就绪等待 ： 启用后，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 ： 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。

高级参数配置 Operator 参数配置

1. **InitContainer.image** ： 配置 CUDA 镜像，推荐默认镜像： **nvidia/cuda**
2. **InitContainer.repository** ： CUDA 镜像所在的镜像仓库，默认为 **nvcr.m.daocloud.io** 仓库
3. **InitContainer.version** : CUDA 镜像的版本，请使用默认参数

Driver 参数配置

1. **Driver.enable** ： 配置是否在节点上部署 NVIDIA 驱动，默认开启， 如果您在使用 GPU Operator 部署前，已经在节点上部署了 NVIDIA 驱动程序，请关闭。
2. **Driver.image** ： 配置 GPU 驱动镜像，推荐默认镜像： **nvidia/driver** 。
3. **Driver.repository** ： GPU 驱动镜像所在的镜像仓库，默认为 nvidia 的 **nvcr.io** 仓库。
4. **Driver.version** ： GPU 驱动镜像的版本，离线部署请使用默认参数，仅在线安装时需配置， 不同类型操作系统的 Driver 镜像的版本存在如下差异， 详情可参考：Nvidia GPU Driver 版本， 如下不同操作系统的 `Driver Version` 示例：

> **Note**
>
> 系统默认提供 525.147.05-centos7 的镜像，其他镜像需要参考向火种节点仓库上传镜像 。 注意版本号后无需填写 ubantu、centos、Red Hat等操作系统名称，若官方镜像含有操作系统后缀，请手动移除
>
> • Red Hat 系统，例如 `525.105.17`
>
> • Ubuntu 系统，例如 `535-5.15.0-1043-nvidia`
>
> • CentOS 系统，例如 `525.147.05`

5. **Driver.RepoConfig.ConfigMapName** ： 用来记录 GPU Operator 的离线 yum 源配置文件名称， 当使用预置的离线包时，global 集群可直接执行如下命令，工作集群 参考 **Global** 集群任意节点的 **yum** 源配置 。

• global 集群配置

```
kubectl create configmap local-repo-config -n gpu-operator --from-file=CentOS-Base.repo=/etc/yum.repos.d/extension.repo
```

• 工作集群配置

> 参考 **Global** 集群任意节点的 **yum** 源配置
>
> a. 使用 ssh 或其它方式进入 Global 集群的任意节点，获取平台离线源配置文件 **extension.repo** ：
>
> ```
> cat /etc/yum.repos.d/extension.repo #查看 extension.repo 中的内容。
> ```
>
> 预期输出如下：
>
> ```
> [extension-0]
> async = 1
> baseurl = http://x.x.x.x:9000/kubean/centos/$releasever/os/$basearch
> gpgcheck = 0
> name = kubean extension 0
>
> [extension-1]
> async = 1
> baseurl = http://x.x.x.x:9000/kubean/centos-iso/$releasever/os/$basearch
> gpgcheck = 0
> name = kubean extension 1
> ```
>
> b. 复制上述 **extension.repo** 文件中的内容，在待部署 GPU Operator 的集群的 **gpu-operator** 命名空间下， 新建名为 **local-repo-config** 的配置文件，可参考创建配置项进行创建。
>
> > **Note**
> >
> > 配置 **key** 值必须为 **CentOS-Base.repo** , **value** 值点离线源配置文件 **extension.repo** 中的内容。

其它操作系统或内核可参考如下链接创建 yum 源文件：

• 构建 CentOS 7.9 离线 yum 源

• 构建 Red Hat 8.4 离线 yum 源

Toolkit 配置参数

1. **Toolkit.enable** ：默认开启，该组件让 conatainerd/docker 支持运行需要gpu的容器。
2. **Toolkit.image** ：配置 Toolkit 镜像，推荐默认镜像： **nvidia/k8s/container-toolkit** 。
3. **Toolkit.repository** ：所在的镜像仓库，默认为 **nvcr.m.daocloud.io** 仓库。
4. **Toolkit.version** ：Toolkit 镜像的版本，版本号与官网保持一致即可。默认使用 centos 的镜像，如果使用ubuntu，需要手动修改 addon 的 yaml， 将 centos 改成 ubuntu。具体型号参考 NVIDIA Container Toolkit。

MIG 配置参数

详细配置方式请参考开启 MIG 功能

1. **MigManager.enabled** ：是否启用 MIG 能力特性。
2. **MigManager.Config.name**: MIG 的切分配置文件名，用于定义 MIG 的（GI, CI）切分策略。 默认为 **default-mig-parted-config** 。自定义参数参考开启 MIG 功能。
3. **Mig.strategy** ：节点上 GPU 卡的 MIG 设备的公开策略。NVIDIA 提供了两种公开 MIG 设备的策略： **single** 、 **mixed** 策略，详情参考 NVIDIA GPU 卡模式说明。

Node-Feature-Discovery 配置参数

Node-Feature-Discovery.enableNodeFeatureAPI ：启用或禁用节点特性 API（Node Feature Discovery API）。

- 当设置为 **true** 时，启用了节点特性 API。
- 当设置为 **false** 或 未设置 时，禁用节点特性 API。

下一步操作

完成上述相关参数配置和创建后：

1. 如果使用 整卡模式，应用创建时可使用 GPU 资源
2. 如果使用 **vGPU** 模式 ，完成上述相关参数配置和创建后，下一步请完成 vGPU Addon 安装
3. 如果使用 **MIG** 模式，并且需要给个别 GPU 节点按照某种切分规格进行使用，否则按照 MigManager.Config 中的 **default** 值进行切分。

- **single** 模式请给对应节点打上如下 Label：

```
kubectl label nodes {node} nvidia.com/mig.config="all-1g.10gb" --overwrite
```

- **mixed** 模式请给对应节点打上如下 Label：

```
kubectl label nodes {node} nvidia.com/mig.config="custom-config" --overwrite
```

切分后，应用可使用 MIG GPU 资源。

构建 CentOS 7.9 离线 yum 源 使用场景介绍

当工作节点的内核版本与 Global 集群的控制节点内核版本或 OS 类型不一致时，需要用户手动构建离线 yum 源。

本文介绍如何构建离线 yum 源， 并在安装 Gpu Operator 时，通过 `RepoConfig.ConfigMapName` 参数来使用。

前提条件

1. 用户已经在平台上安装了 v0.12.0 及以上版本的 addon 离线包。
2. 准备一个能够和待部署 GPU Operator 的集群网络能够联通的文件服务器，如 nginx 或 minio。
3. 准备一个能够访问互联网、待部署 GPU Operator 的集群和文件服务器的节点， 且节点上已经完成 Docker 的安装。

操作步骤

本文以内核版本为 `3.10.0-1160.95.1.el7.x86_64` 的 CentOS 7.9 节点为例，介绍如何构建 GPU operator 离线包的 yum 源。

检查集群节点的 OS 和内核版本

分别在 Global 集群的控制节点和待部署 GPU Operator 的节点执行如下命令，若两个节点的 OS 和内核版本一致则无需构建 yum 源， 可参考离线安装 GPU Operator 文档直接安装；若两个节点的 OS 或内核版本不一致，请执行下一步。

1. 执行如下命令，查看集群下待部署 GPU Operator 节点的发行版名称和版本号。

```
cat /etc/redhat-release
```

预期输出如下：

```
CentOS Linux release 7.9 (Core)
```

输出结果为当前节点内核版本 `CentOS 7.9` 。

2. 执行如下命令，查看集群下待部署 GPU Operator 节点的内核版本。

```
uname -a
```

预期输出如下：

```
Linux localhost.localdomain 3.10.0-1160.95.1.el7.x86_64 #1 SMP Mon Oct 19 16:18:59 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

输出结果为当前节点内核版本 `3.10.0-1160.el7.x86_64`。

制作离线 yum 源

在一个能够访问互联网和文件服务器的节点上进行操作。

1. 在一个能够访问互联网和文件服务器的节点上执行如下命令新建一个名为 **yum.sh** 的脚本文件。

```
vi yum.sh
```

然后按下 **i** 键进入插入模式，输入以下内容：

```
export TARGET_KERNEL_VERSION=$1

cat >> run.sh << \EOF
#! /bin/bash
echo "start install kernel repo"
echo ${KERNEL_VERSION}
mkdir centos-base

if [ "$OS" -eq 7 ]; then
    yum install --downloadonly --downloaddir=./centos-base perl
    yum install --downloadonly --downloaddir=./centos-base elfutils-libelf.x86_64
    yum install --downloadonly --downloaddir=./redhat-base elfutils-libelf-devel.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-headers-${KERNEL_VERSION}.el7.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-devel-${KERNEL_VERSION}.el7.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-${KERNEL_VERSION}.el7.x86_64
    yum install  -y --downloadonly --downloaddir=./centos-base groff-base
elif [ "$OS" -eq 8 ]; then
    yum install --downloadonly --downloaddir=./centos-base perl
    yum install --downloadonly --downloaddir=./centos-base elfutils-libelf.x86_64
    yum install --downloadonly --downloaddir=./redhat-base elfutils-libelf-devel.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-headers-${KERNEL_VERSION}.el8.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-devel-${KERNEL_VERSION}.el8.x86_64
    yum install --downloadonly --downloaddir=./centos-base kernel-${KERNEL_VERSION}.el8.x86_64
    yum install  -y --downloadonly --downloaddir=./centos-base groff-base
else
    echo "Error os version"
fi

createrepo centos-base/
ls -lh centos-base/
tar -zcf centos-base.tar.gz centos-base/
echo "end install kernel repo"
EOF

cat >> Dockerfile << EOF
FROM centos:7
ENV KERNEL_VERSION=""
ENV OS=7
RUN yum install -y createrepo
COPY run.sh .
ENTRYPOINT ["/bin/bash","run.sh"]
EOF

docker build -t test:v1 -f Dockerfile .
docker run -e KERNEL_VERSION=$TARGET_KERNEL_VERSION --name centos7.9 test:v1
docker cp centos7.9:/centos-base.tar.gz .
tar -xzf centos-base.tar.gz
```

按下 **esc** 键退出插入模式，然后输入 __ :wq__ 保存并退出。

2. 运行 **yum.sh** 文件：

```
bash -x yum.sh TARGET_KERNEL_VERSION
```

`TARGET_KERNEL_VERSION` 参数用于指定集群节点的内核版本，注意：发行版标识符（如 __ .el7.x86_64 __ ）无需输入。 例如：

```
bash -x yum.sh 3.10.0-1160.95.1
```

至此，您已经生成了内核为 **3.10.0-1160.95.1.el7.x86_64** 的离线的 yum 源： **centos-base** 。

上传离线 yum 源到文件服务器

在一个能够访问互联网和文件服务器的节点上进行操作。主要用于将上一步中生成的 yum 源上传到可以被待部署 GPU Operator 的集群进行访问的文件服务器中。 文件服务器可以为 Nginx 、 Minio 或其它支持 Http 协议的文件服务器。

本操作示例采用的是 DCE5 火种节点内置的 Minio 作为文件服务器，Minio 相关信息如下：

- 访问地址： `http://10.5.14.200:9000（一般为{火种节点 IP} + {9000 端口}）`
- 登录用户名：rootuser
- 登录密码：rootpass123
- 在节点当前路径下，执行如下命令将节点本地 mc 命令行工具和 minio 服务器建立链接。

```
mc config host add minio http://10.5.14.200:9000 rootuser rootpass123
```

预期输出如下：

```
Added `minio` successfully.
```

mc 命令行工具是 Minio 文件服务器提供的客户端命令行工具，详情请参考： MinIO Client。

- 在节点当前路径下，新建一个名为 **centos-base** 的存储桶（bucket）。

```
mc mb -p minio/centos-base
```

预期输出如下：

```
Bucket created successfully __minio/centos-base__ .
```

- 将存储桶 **centos-base** 的访问策略设置为允许公开下载。以便在后期安装 GPU-operator 时能够被访问。

```
mc anonymous set download minio/centos-base
```

预期输出如下：

```
Access permission for `minio/centos-base` is set to `download`
```

- 在节点当前路径下，将步骤二生成的离线 yum 源文件 **centos-base** 复制到 minio 服务器的 **minio/centos-base** 存储桶中。

```
mc cp centos-base minio/centos-base --recursive
```

在集群创建配置项用来保存 yum 源信息

在待部署 GPU Operator 集群的控制节点上进行操作。

1. 执行如下命令创建名为 **CentOS-Base.repo** 的文件，用来指定 yum 源存储的配置信息。

```
# 文件名称必须为 CentOS-Base.repo，否则安装 gpu-operator 时无法被识别
cat > CentOS-Base.repo << EOF
[extension-0]
baseurl = http://10.5.14.200:9000/centos-base/centos-base #步骤三中，放置 yum 源的文件服务器地址
gpgcheck = 0
name = kubean extension 0

[extension-1]
baseurl = http://10.5.14.200:9000/centos-base/centos-base #步骤三中，放置 yum 源的文件服务器地址
gpgcheck = 0
name = kubean extension 1
EOF
```

2. 基于创建的 **CentOS-Base.repo** 文件，在 gpu-operator 命名空间下，创建名为 **local-repo-config** 的配置文件：

```
kubectl create configmap local-repo-config  -n gpu-operator --from-file=CentOS-Base.repo=/etc/yum.repos.d/extension.repo
```

预期输出如下：

```
configmap/local-repo-config created
```

local-repo-config 配置文件用于在安装 gpu-operator 时，提供 `RepoConfig.ConfigMapName` 参数的值，配置文件名称用户可自定义。

3. 查看 **local-repo-config** 的配置文件的内容：

```
kubectl get configmap local-repo-config  -n gpu-operator -oyaml
```

预期输出如下：

```
apiVersion: v1
data:
CentOS-Base.repo: "[extension-0]\nbaseurl = http://10.6.232.5:32618/centos-base#步骤二中，放置 yum 源的文件服务器路径\ngpgcheck = 0\nname = kubean extension 0\n
\n[extension-1]\nbaseurl
    = http://10.6.232.5:32618/centos-base #步骤二中，放置 yum 源的文件服务器路径\ngpgcheck = 0\nname
    = kubean extension 1\n"
kind: ConfigMap
metadata:
creationTimestamp: "2023-10-18T01:59:02Z"
name: local-repo-config
namespace: gpu-operator
resourceVersion: "59445080"
uid: c5f0ebab-046f-442c-b932-f9003e014387
```

至此，您已成功为待部署 GPU Operator 的集群创建了离线 yum 源配置文件。 通过在离线安装 GPU Operator 时通过 `RepoConfig.ConfigMapName` 参数来使用。

向火种节点仓库上传 Red Hat GPU Opreator 离线镜像

本文以 Red Hat 8.4 的 `nvcr.io/nvidia/driver:525.105.17-rhel8.4` 离线驱动镜像为例，介绍如何向火种节点仓库上传离线镜像。

前提条件

1. 火种节点及其组件状态运行正常。
2. 准备一个能够访问互联网和火种节点的节点，且节点上已经完成 Docker 的安装。

操作步骤 在联网节点获取离线镜像

以下操作在联网节点上进行。

1. 在联网机器上拉取 `nvcr.io/nvidia/driver:525.105.17-rhel8.4` 离线驱动镜像：

```
docker pull nvcr.io/nvidia/driver:525.105.17-rhel8.4
```

2. 镜像拉取完成后，打包镜像为 `nvidia-driver.tar` 压缩包：

```
docker save nvcr.io/nvidia/driver:525.105.17-rhel8.4 > nvidia-driver.tar
```

3. 拷贝 `nvidia-driver.tar` 镜像压缩包到火种节点：

```
scp  nvidia-driver.tar user@ip:/root
```

例如：

```
scp  nvidia-driver.tar root@10.6.175.10:/root
```

推送镜像到火种节点仓库

以下操作在火种节点上进行。

1. 登录火种节点，将联网节点拷贝的镜像压缩包 `nvidia-driver.tar` 导入本地：

```
docker load -i nvidia-driver.tar
```

2. 查看刚刚导入的镜像：

```
docker images -a |grep nvidia
```

预期输出：

```
nvcr.io/nvidia/driver               e3ed7dee73e9   1 days ago   1.02GB
```

3. 重新标记镜像，使其与远程 Registry 仓库中的目标仓库对应：

```
docker tag <image-name> <registry-url>/<repository-name>:<tag>
```

- `<image-name>` 是上一步 nvidia 镜像的名称，
- `<registry-url>` 是火种节点上 Registry 服务的地址，
- `<repository-name>` 是您要推送到的仓库名称，
- `<tag>` 是您为镜像指定的标签。

例如：

```
registry: docker tag nvcr.io/nvidia/driver 10.6.10.5/nvcr.io/nvidia/driver:525.105.17-rhel8.4
```

4. 将镜像推送到火种节点镜像仓库：

```
docker push {ip}/nvcr.io/nvidia/driver:525.105.17-rhel8.4
```

接下来

参考构建 Red Hat 8.4 离线 yum 源和 GPU Operator 离线安装来为集群部署 GPU Operator。

构建 Red Hat 8.4 离线 yum 源 使用场景介绍

DCE 5 预置了 CentOS 7.9，内核为 3.10.0-1160 的 GPU operator 离线包。其它 OS 类型的节点或内核需要用户手动构建离线 yum 源。

本文介绍如何基于 Global 集群任意节点构建 Red Hat 8.4 离线 yum 源包，并在安装 Gpu Operator 时，通过 `RepoConfig.ConfigMapName` 参数来使用。

前提条件

1. 用户已经在平台上安装了 v0.12.0 及以上版本的 addon 离线包。
2. 待部署 GPU Operator 的集群节点 OS 必须为 Red Hat 8.4，且内核版本完全一致。
3. 准备一个能够和待部署 GPU Operator 的集群网络能够联通的文件服务器，如 nginx 或 minio。
4. 准备一个能够访问互联网、待部署 GPU Operator 的集群和文件服务器的节点，且节点上已经完成 Docker 的安装。
5. Global 集群的节点必须为 Red Hat 8.4 4.18.0-305.el8.x86_64。

操作步骤

本文以 Red Hat 8.4 4.18.0-305.el8.x86_64 节点为例，介绍如何基于 Global 集群任意节点构建 Red Hat 8.4 离线 yum 源包， 并在安装 Gpu Operator 时，通过 `RepoConfig.ConfigMapName` 参数来使用。

下载火种节点中的 yum 源

以下操作在 global 集群的 master 节点上执行。

1. 使用 ssh 或其它方式进入 global 集群内任一节点执行如下命令：

```
cat /etc/yum.repos.d/extension.repo #查看 extension.repo 中的内容
```

预期输出如下：

```
[extension-0]
baseurl = http://10.5.14.200:9000/kubean/redhat/$releasever/os/$basearch
gpgcheck = 0
name = kubean extension 0

[extension-1]
baseurl = http://10.5.14.200:9000/kubean/redhat-iso/$releasever/os/$basearch/AppStream
gpgcheck = 0
name = kubean extension 1

[extension-2]
baseurl = http://10.5.14.200:9000/kubean/redhat-iso/$releasever/os/$basearch/BaseOS
gpgcheck = 0
name = kubean extension 2
```

2. 在 **root** 路径下新建一个名为 **redhat-base-repo** 的文件夹

```
mkdir redhat-base-repo
```

3. 下载 yum 源中的 rpm 包到本地：

```
yum install yum-utils
```

4. 下载 extension-1 中的 rpm 包：

```
reposync  -p redhat-base-repo  -n --repoid=extension-1
```

5. 下载 extension-2 中的 rpm 包：

```
reposync  -p redhat-base-repo  -n --repoid=extension-2
```

下载 elfutils-libelf-devel-0.187-4.el8.x86_64.rpm 包

以下操作在联网节点执行操作，在操作前，您需要保证联网节点和 Global 集群 master 节点间的网络联通性。

1. 在联网节点执行如下命令，下载 **elfutils-libelf-devel-0.187-4.el8.x86_64.rpm** 包：

```
wget https://rpmfind.net/linux/centos/8-stream/BaseOS/x86_64/os/Packages/elfutils-libelf-devel-0.187-4.el8.x86_64.rpm
```

2. 在当前目录下将 **elfutils-libelf-devel-0.187-4.el8.x86_64.rpm** 包传输至步骤一中的节点上：

```
scp  elfutils-libelf-devel-0.187-4.el8.x86_64.rpm user@ip:~/redhat-base-repo/extension-2/Packages/
```

例如：

```
scp  elfutils-libelf-devel-0.187-4.el8.x86_64.rpm root@10.6.175.10:~/redhat-base-repo/extension-2/Packages/
```

生成本地 yum repo

以下操作在步骤一中 global 集群的 master 节点上执行。

1. 进入 yum repo 目录：

```
cd ~/redhat-base-repo/extension-1/Packages
cd ~/redhat-base-repo/extension-2/Packages
```

2. 生成目录 repo 索引：

```
yum install createrepo -y  # 若已安装 createrepo 可省略此步骤
createrepo_c ./
```

至此，您已经生成了内核为 `4.18.0-305.el8.x86_64` 的离线的 yum 源： **redhat-base-repo** 。

将本地生成的 yum repo 上传至文件服务器

本操作示例采用的是 DCE5 火种节点内置的 Minio 作为文件服务器，用户可基于自身情况选择文件服务器。Minio 相关信息如下：

- 访问地址： `http://10.5.14.200:9000（一般为{火种节点 IP} + {9000 端口}）`
- 登录用户名：rootuser
- 登录密码：rootpass123
- 在节点当前路径下，执行如下命令将节点本地 mc 命令行工具和 minio 服务器建立链接。

```
mc config host add minio 文件服务器访问地址 用户名 密码
```

例如：

```
mc config host add minio http://10.5.14.200:9000 rootuser rootpass123
```

预期输出如下：

```
Added `minio` successfully.
```

mc 命令行工具是 Minio 文件服务器提供的客户端命令行工具，详情请参考： MinIO Client。

- 在节点当前路径下，新建一个名为 **redhat-base** 的存储桶(bucket)。

```
mc mb -p minio/redhat-base
```

预期输出如下：

```
Bucket created successfully `minio/redhat-base`.
```

- 将存储桶 **redhat-base** 的访问策略设置为允许公开下载。以便在后期安装 GPU-operator 时能够被访问。

```
mc anonymous set download minio/redhat-base
```

预期输出如下：

```
Access permission for `minio/redhat-base` is set to `download`
```

- 在节点当前路径下，将步骤二生成的离线 yum 源文件 **redhat-base-repo** 复制到 minio 服务器的 **minio/redhat-base** 存储桶中。

```
mc cp redhat-base-repo minio/redhat-base --recursive
```

在集群创建配置项用来保存 yum 源信息

本步骤在待部署 GPU Operator 集群的控制节点上进行操作。

1. 执行如下命令创建名为 **redhat.repo** 的文件，用来指定 yum 源存储的配置信息。

```
# 文件名称必须为 redhat.repo，否则安装 gpu-operator 时无法被识别
cat > redhat.repo << EOF
[extension-0]
baseurl = http://10.5.14.200:9000/redhat-base/redhat-base-repo/Packages #步骤一中，放置 yum 源的文件服务器地址
gpgcheck = 0
name = kubean extension 0

[extension-1]
baseurl = http://10.5.14.200:9000/redhat-base/redhat-base-repo/Packages #步骤一中，放置 yum 源的文件服务器地址
gpgcheck = 0
name = kubean extension 1
EOF
```

2. 基于创建的 **redhat.repo** 文件，在 gpu-operator 命名空间下，创建名为 **local-repo-config** 的配置文件：

```
kubectl create configmap local-repo-config  -n gpu-operator --from-file=./redhat.repo
```

预期输出如下：

```
configmap/local-repo-config created
```

local-repo-config 配置文件用于在安装 gpu-operator 时，提供 `RepoConfig.ConfigMapName` 参数的值，配置文件名称用户可自定义。

3. 查看 **local-repo-config** 的配置文件的内容：

```
kubectl get configmap local-repo-config  -n gpu-operator -oyaml
```

至此，您已成功为待部署 GPU Operator 的集群创建了离线 yum 源配置文件。 通过在离线安装 GPU Operator 时通过 `RepoConfig.ConfigMapName` 参数来使用。

构建 Red Hat 7.9 离线 yum 源 使用场景介绍

d.run 预置了 CentOS 7.9，内核为 3.10.0-1160 的 GPU Operator 离线包。其它 OS 类型的节点或内核需要用户手动构建离线 yum 源。

本文介绍如何基于 Global 集群任意节点构建 Red Hat 7.9 离线 yum 源包，并在安装 Gpu Operator 时使用 `RepoConfig.ConfigMapName` 参数。

前提条件

1. 用户已经在平台上安装了 v0.12.0 及以上版本的 addon 离线包
2. 待部署 GPU Operator 的集群节点 OS 必须为 Red Hat 7.9，且内核版本完全一致
3. 准备一个能够与待部署 GPU Operator 的集群网络联通的文件服务器，如 nginx 或 minio
4. 准备一个能够访问互联网、待部署 GPU Operator 的集群和文件服务器的节点， 且节点上已经完成 Docker 的安装
5. Global 集群的节点必须为 Red Hat 7.9

操作步骤 1. 构建相关内核版本的离线 Yum 源

1. 下载 rhel7.9 ISO

Red Hat Enterprise Linux 8.4.0

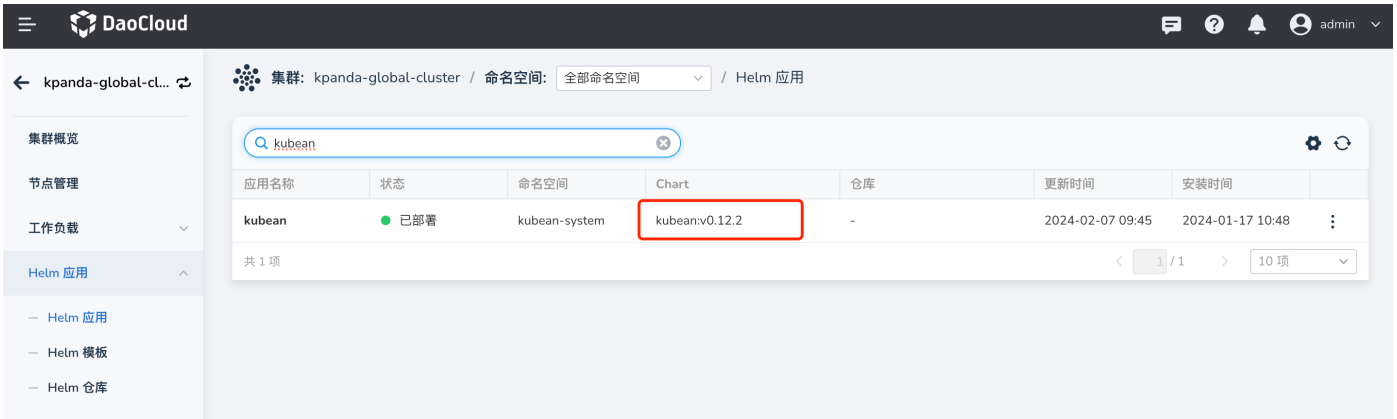
x86_64	DVD iso	Release date May 18, 2021	Download (9.43 GB)
x86_64	Boot iso	Release date May 18, 2021	Download (721 MB)
aarch64	DVD iso	Release date May 18, 2021	Download (6.86 GB)
aarch64	Boot iso	Release date May 18, 2021	Download (646.29 MB)

Red Hat Enterprise Linux 7.9.0

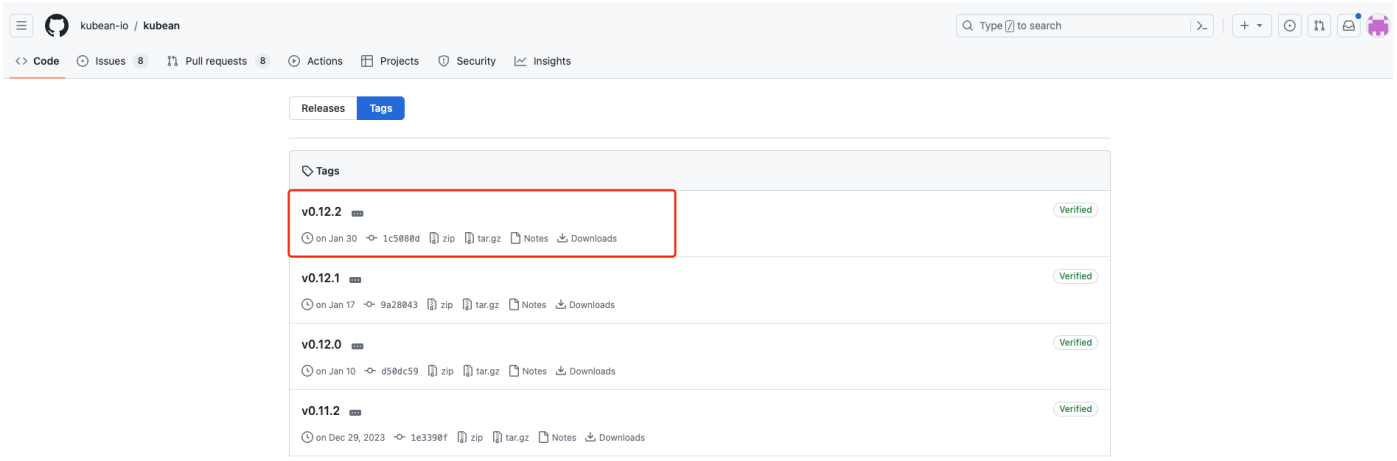
Boot ISO	RHEL x86_64	Release date August 11, 2020	Download (608 MB)
DVD ISO	RHEL x86_64	Release date August 11, 2020	Download (4.22 GB)

2. 下载与 Kubean 版本对应的的 rhel7.9 ospackage

在 容器管理 的 Global 集群中找到 **Helm** 应用 ，搜索 kubean，可查看 kubean 的版本号。



在 kubean的代码仓库 中下载该版本的 rhel7.9 ospackage。

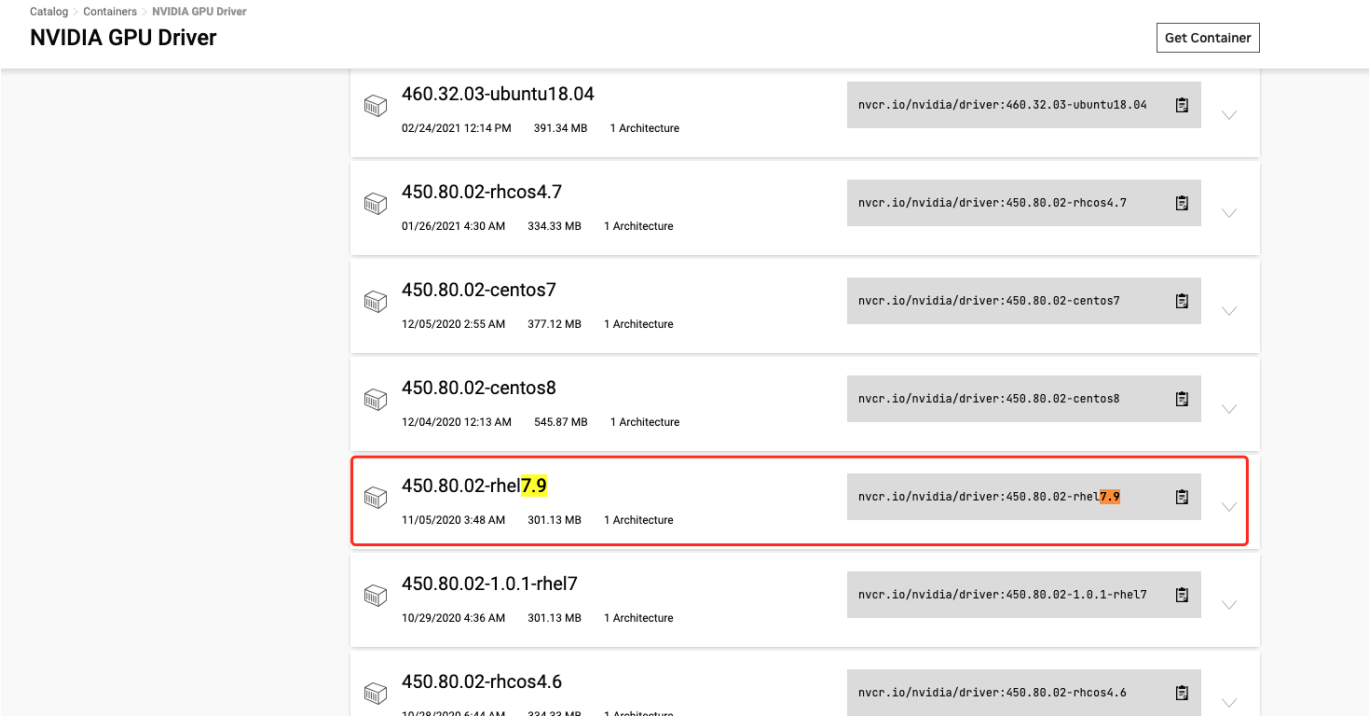


3. 通过安装器导入离线资源

   参考导入离线资源文档。

2. 下载 Red Hat 7.9 OS 的离线驱动镜像

   点击查看下载地址。



3. 向火种节点仓库上传 Red Hat GPU Opreator 离线镜像

   参考向火种节点仓库上传 Red Hat GPU Opreator 离线镜像。 注意：此参考以 rhel8.4 为例，请注意修改成 rhel7.9。

4. 在集群创建配置项用来保存 Yum 源信息

在待部署 GPU Operator 集群的控制节点上运行以下命令。

1. 执行如下命令创建名为 **CentOS-Base.repo** 的文件，用来指定 yum 源存储的配置信息。

```
# 文件名称必须为 CentOS-Base.repo，否则安装 gpu-operator 时无法被识别
cat > CentOS-Base.repo <<  EOF
[extension-0]
baseurl = http://10.5.14.200:9000/centos-base/centos-base #火种节点的的文件服务器地址，一般为{火种节点 IP} + {9000 端口}
gpgcheck = 0
name = kubean extension 0

[extension-1]
baseurl = http://10.5.14.200:9000/centos-base/centos-base #火种节点的的文件服务器地址，一般为{火种节点 IP} + {9000 端口}
gpgcheck = 0
name = kubean extension 1
EOF
```

2. 基于创建的 **CentOS-Base.repo** 文件，在 gpu-operator 命名空间下，创建名为 **local-repo-config** 的配置文件：

```
kubectl create configmap local-repo-config -n gpu-operator --from-file=CentOS-Base.repo=/etc/yum.repos.d/extension.repo
```

预期输出如下：

```
configmap/local-repo-config created
```

local-repo-config 配置文件用于在安装 gpu-operator 时，提供 `RepoConfig.ConfigMapName` 参数的值，配置文件名称用户可自定义。

3. 查看 **local-repo-config** 的配置文件的内容：

```
kubectl get configmap local-repo-config -n gpu-operator -oyaml
```

预期输出如下：

```
apiVersion: v1
data:
  CentOS-Base.repo: "[extension-0]\nbaseurl = http://10.6.232.5:32618/centos-base # 步骤 2 中，放置 yum 源的文件服务器路径 \ngpgcheck = 0\nname = kubean extension
0\n  \n[extension-1]\nbaseurl
  = http://10.6.232.5:32618/centos-base # 步骤 2 中，放置 yum 源的文件服务器路径 \ngpgcheck = 0\nname
  = kubean extension 1\n"
kind: ConfigMap
metadata:
  creationTimestamp: "2023-10-18T01:59:02Z"
  name: local-repo-config
  namespace: gpu-operator
  resourceVersion: "59445080"
  uid: c5f0ebab-046f-442c-b932-f9003e014387
```

至此，您已成功为待部署 GPU Operator 的集群创建了离线 yum 源配置文件。 其中在离线安装 GPU Operator 时使用了 `RepoConfig.ConfigMapName` 参数。

应用使用 GPU 整卡

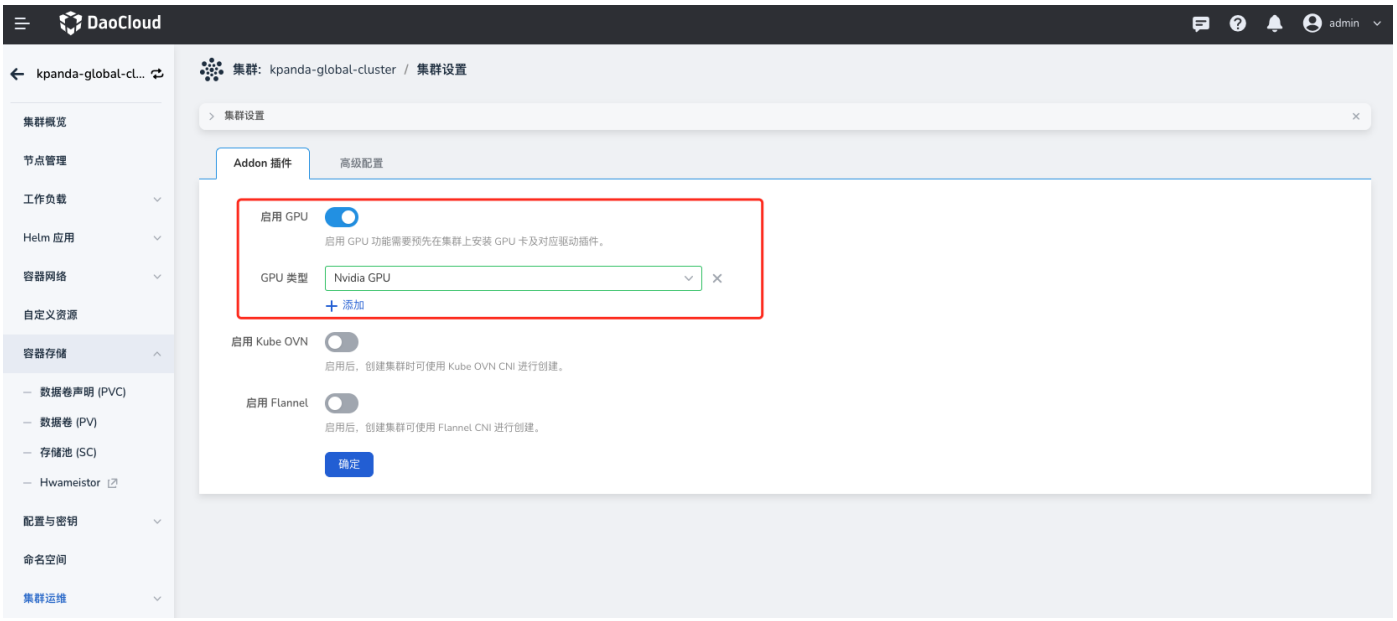
本节介绍如何在 d.run 平台将整个 NVIDIA GPU 卡分配给单个应用。

前提条件

- 当前集群已离线安装 GPU Operator 并已启用 NVIDIA DevicePlugin，可参考 GPU Operator 离线安装。
- 当前集群内 GPU 卡未进行任何虚拟化操作或被其它应用占用。

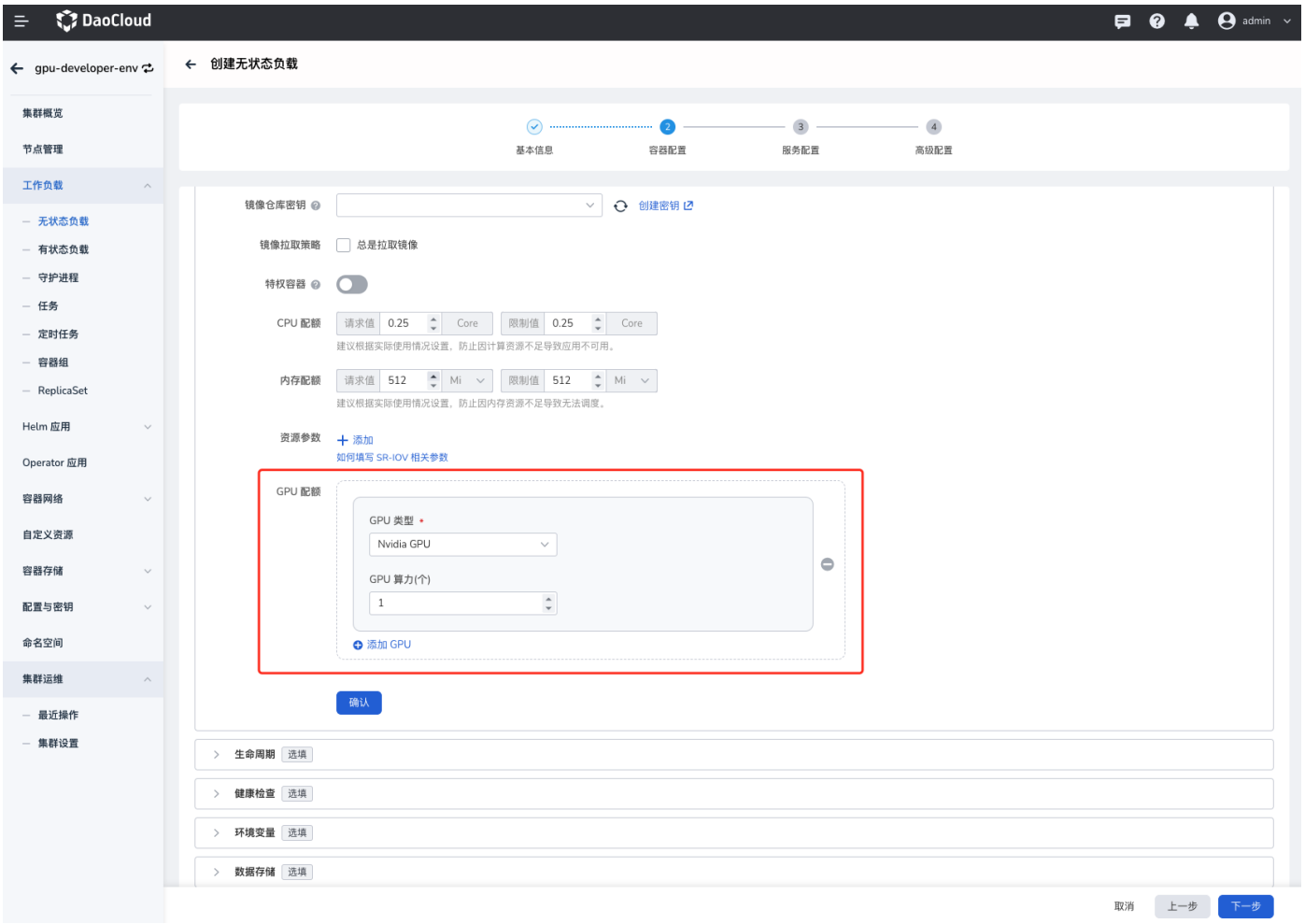
操作步骤 使用 UI 界面配置

1. 确认集群是否已检测 GPU 卡。点击对应 集群 -> 集群设置 -> **Addon** 插件 ， 查看是否已自动启用并自动检测对应 GPU 类型。 目前集群会自动启用 **GPU** ， 并且设置 **GPU** 类型为 **Nvidia GPU** 。



2. 部署工作负载，点击对应 集群 -> 工作负载 ，通过镜像方式部署工作负载，选择类型（Nvidia GPU）之后，需要配置应用使用的物理卡数量：

   物理卡数量（**nvidia.com/gpu**） ：表示当前 Pod 需要挂载几张物理卡，输入值必须为整数且 小于等于 宿主机上的卡数量。



   如果上述值配置的有问题则会出现调度失败，资源分配不了的情况。

使用 YAML 配置

创建工作负载申请 GPU 资源，在资源申请和限制配置中增加 **nvidia.com/gpu: 1** 参数配置应用使用物理卡的数量。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: full-gpu-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: full-gpu-demo
  template:
    metadata:
      labels:
        app: full-gpu-demo
    spec:
      containers:
      - image: chrstnhntschl/gpu_burn
        name: container-0
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 1   # 申请 GPU 的数量
          limits:
            cpu: 250m
            memory: 512Mi
            nvidia.com/gpu: 1   # GPU 数量的使用上限
      imagePullSecrets:
      - name: default-secret
```

Note

使用 **nvidia.com/gpu** 参数指定 GPU 数量时，requests 和 limits 值需要保持一致。

NVIDIA vGPU 模式

安装 NVIDIA vGPU Addon

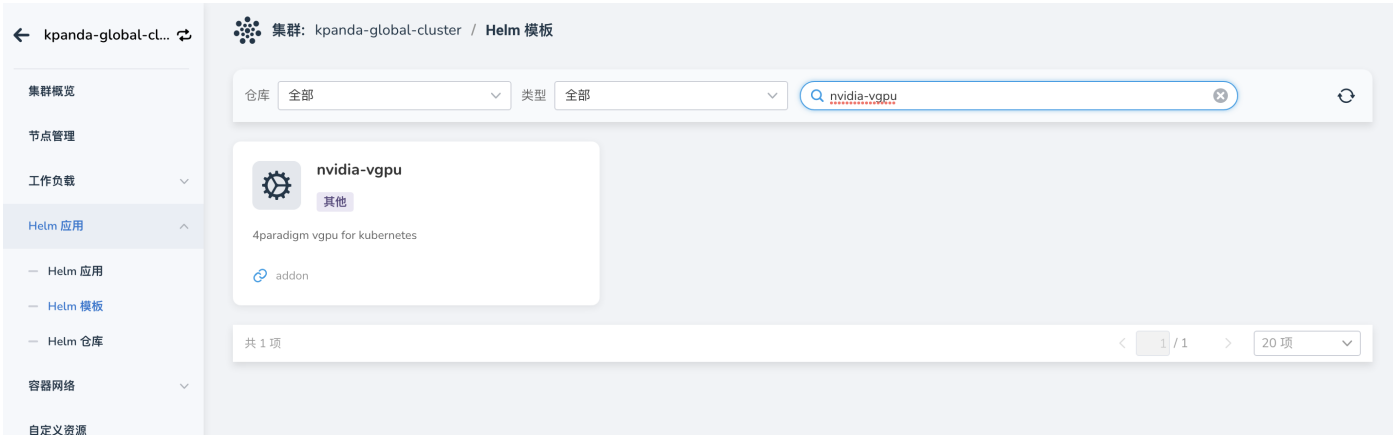
如需将一张 NVIDIA 虚拟化成多个虚拟 GPU，并将其分配给不同的虚拟机或用户，您可以使用 NVIDIA 的 vGPU 能力。 本节介绍如何在 d.run 平台中安装 vGPU 插件，这是使用 NVIDIA vGPU 能力的前提。

前提条件

- 参考 GPU 支持矩阵 确认集群节点上具有对应型号的 GPU 卡。
- 当前集群已通过 Operator 部署 NVIDIA 驱动，具体参考 GPU Operator 离线安装。

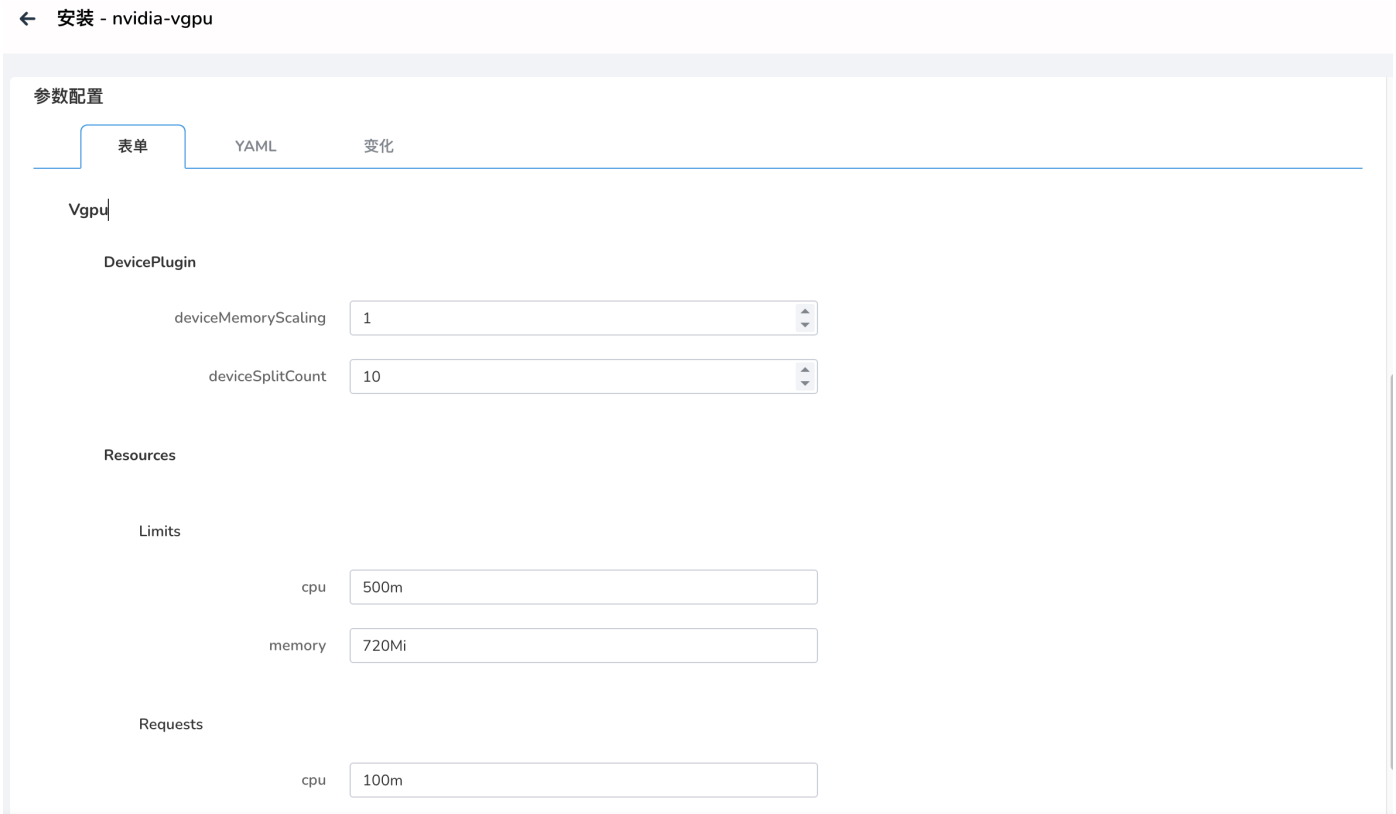
操作步骤

1. 功能模块路径： 容器管理 -> 集群管理 ，点击目标集群的名称，从左侧导航栏点击 **Helm** 应用 -> **Helm** 模板 -> 搜索 **nvidia-vgpu** 。



2. 在安装 vGPU 的过程中提供了几个基本修改的参数，如果需要修改高级参数点击 YAML 列进行修改：

- **deviceMemoryScaling** ： NVIDIA 装置显存使用比例，输入内容必须为整数，预设值是 1。可以大于 1（启用虚拟显存，实验功能）。 对于有 M 显存大小的 NVIDIA GPU，如果我们配置 **devicePlugin.deviceMemoryScaling** 参数为 S， 在部署了我们装置插件的 Kubernetes 集群中，这张 GPU 分出的 vGPU 将总共包含 **S * M** 显存。
- **deviceSplitCount** ： 整数类型，预设值是 10。GPU 的分割数，每一张 GPU 都不能分配超过其配置数目的任务。 若其配置为 N 的话，每个 GPU 上最多可以同时存在 N 个任务。
- **Resources** ： 就是对应 vgpu-device-plugin 和 vgpu-schedule pod 的资源使用量。



3. 安装成功之后会在指定 **Namespace** 下出现如下两个类型的 Pod，即表示 NVIDIA vGPU 插件已安装成功：

安装成功后，部署应用可使用 vGPU 资源。

Note

NVIDIA vGPU Addon 不支持从老版本 v2.0.0 直接升级为最新版 v2.0.0+1； 如需升级，请卸载老版本后重新安装。

应用使用 Nvidia vGPU

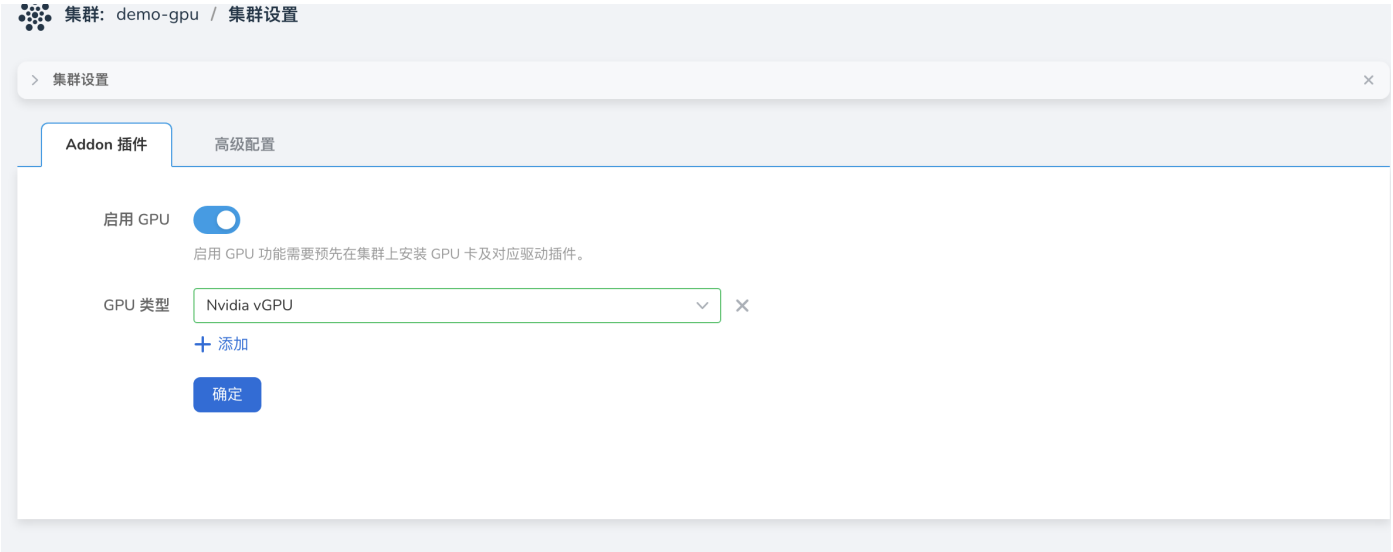
本节介绍如何在 d.run 平台使用 vGPU 能力。

前提条件

- 集群节点上具有对应型号的 GPU 卡
- 已成功安装 vGPU Addon，详情参考 GPU Addon 安装
- 已安装 GPU Operator，并已 关闭 **Nvidia.DevicePlugin** 能力，可参考 GPU Operator 离线安装

操作步骤 界面使用 vGPU

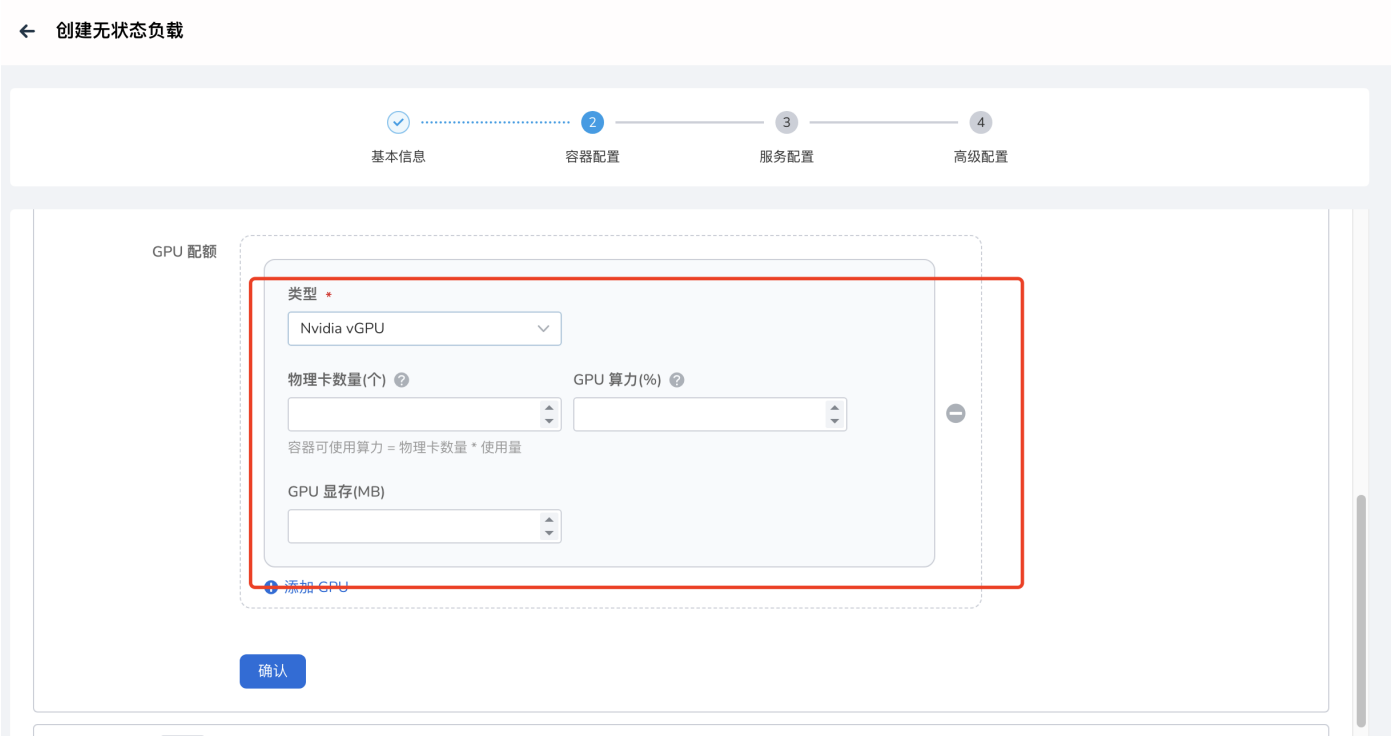
1. 确认集群是否已检测 GPU 卡。点击对应 集群 -> 集群设置 -> **Addon** 插件 ， 查看是否已自动启用并自动检测对应 GPU 类型。 目前集群会自动启用 **GPU** ， 并且设置 **GPU** 类型为 **Nvidia vGPU** 。



2. 部署工作负载，点击对应 集群 -> 工作负载 ，通过镜像方式部署工作负载，选择类型（Nvidia vGPU）之后，会自动出现如下几个参数需要填写：

- 物理卡数量（**nvidia.com/vgpu**）：表示当前 Pod 需要挂载几张物理卡，输入值必须为整数且 小于等于 宿主机上的卡数量。
- **GPU** 算力（**nvidia.com/gpucores**）：表示每张卡占用的 GPU 算力，值范围为 0-100； 如果配置为 0， 则认为不强制隔离；配置为100，则认为独占整张卡。
- **GPU** 显存（**nvidia.com/gpumem**）：表示每张卡占用的 GPU 显存，值单位为 MB，最小值为 1，最大值为整卡的显存值。

如果上述值配置的有问题则会出现调度失败，资源分配不了的情况。



YAML 配置使用 vGPU

参考如下工作负载配置，在资源申请和限制配置中增加 **nvidia.com/vgpu: '1'** 参数来配置应用使用物理卡的数量。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: full-vgpu-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: full-vgpu-demo
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: full-vgpu-demo
    spec:
      containers:
        - name: full-vgpu-demo1
          image: chrstnhntschl/gpu_burn
          resources:
            limits:
              nvidia.com/gpucores: '20'   # 申请每张卡占用 20% 的 GPU 算力
              nvidia.com/gpumem: '200'   # 申请每张卡占用 200MB 的显存
              nvidia.com/vgpu: '1'   # 申请GPU的数量
          imagePullPolicy: Always
      restartPolicy: Always
```

NVIDIA MIG 模式

NVIDIA 多实例 GPU(MIG) 概述 MIG 场景

- 多租户云环境

  MIG 允许云服务提供商将一块物理 GPU 划分为多个独立的 GPU 实例，每个实例可以独立分配给不同的租户。这样可以实现资源的隔离和独立性，满足多个租户对 GPU 计算能力的需求。

- 容器化应用程序

  MIG 可以在容器化环境中实现更细粒度的 GPU 资源管理。通过将物理 GPU 划分为多个 MIG 实例，可以为每个容器分配独立的 GPU 计算资源，提供更好的性能隔离和资源利用。

- 批处理作业

  对于需要大规模并行计算的批处理作业，MIG 可以提供更高的计算性能和更大的显存容量。每个 MIG 实例可以利用物理 GPU 的一部分计算资源，从而加速大规模计算任务的处理。

- **AI**/机器学习训练

  MIG 可以在训练大规模深度学习模型时提供更大的计算能力和显存容量。将物理 GPU 划分为多个 MIG 实例，每个实例可以独立进行模型训练，提高训练效率和吞吐量。

总体而言，NVIDIA MIG 适用于需要更细粒度的GPU资源分配和管理的场景，可以实现资源的隔离、提高性能利用率，并且满足多个用户或应用程序对 GPU 计算能力的需求。

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。

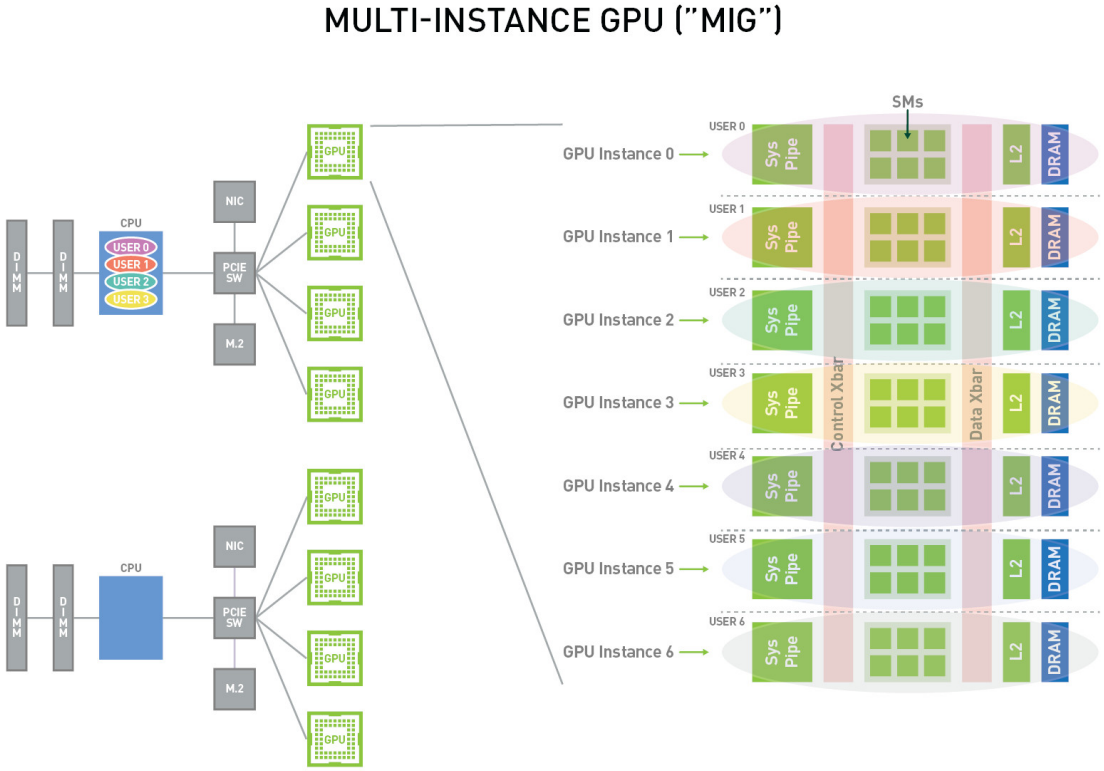
通过 MIG，每个 GPU 实例的处理器在整个内存系统中具有独立且隔离的路径——芯片上的交叉开关端口、L2 高速缓存组、内存控制器和 DRAM 地址总线都唯一分配给单个实例。

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 架构

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。



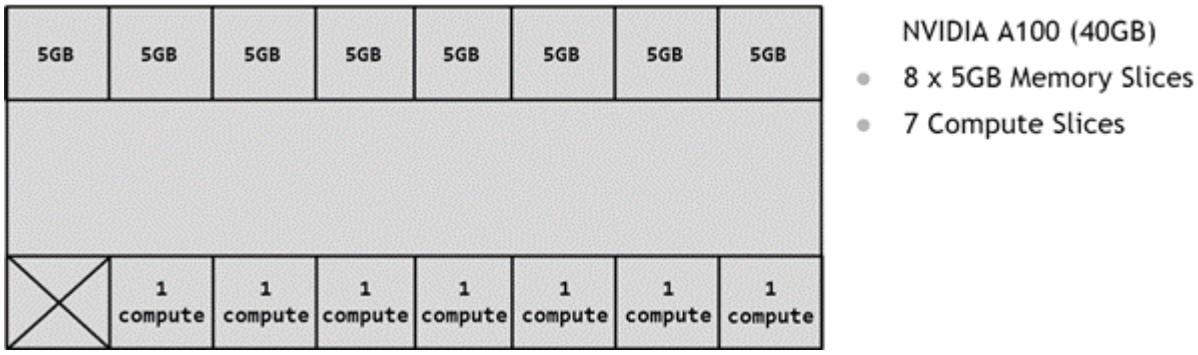
重要概念

- **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 内存资源的八分之一，每个 GPU SM 切片获得 SM 总数的七分之一。
- **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。

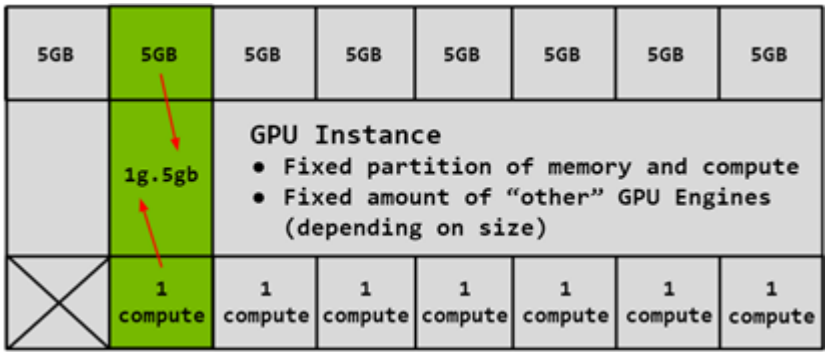
GPU 实例（GI）

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。

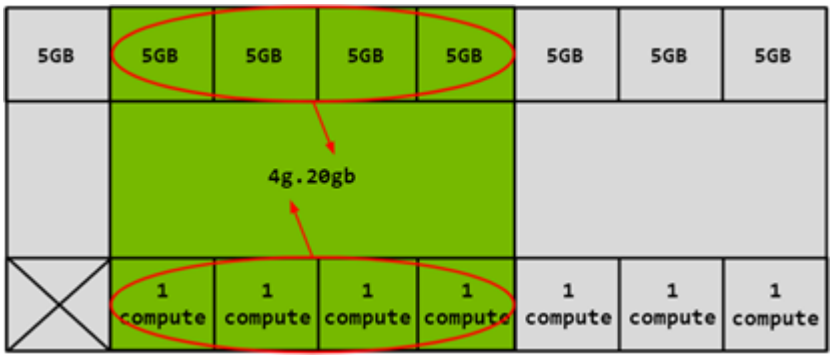
GPU 的分区是使用内存切片进行的，因此可以认为 A100-40GB GPU 具有 8x5GB 内存切片和 7 个 GPU SM 切片，如下图所示，展示了 A100 上可用的内存切片。



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：

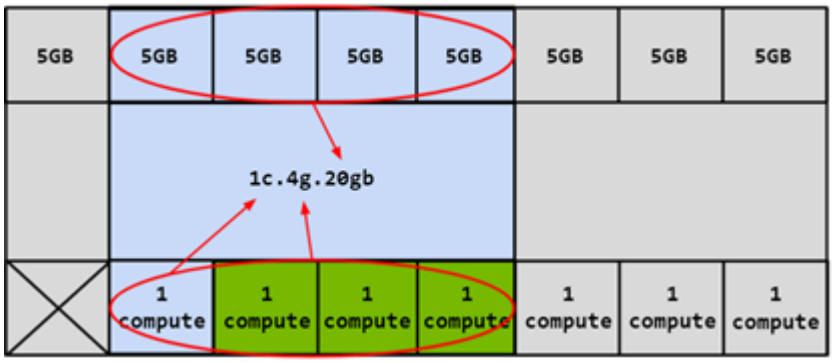


同样，4x5GB 内存切片可以与 4x1 计算切片结合使用以创建 **4g.20gb** 的 GI 配置文件：

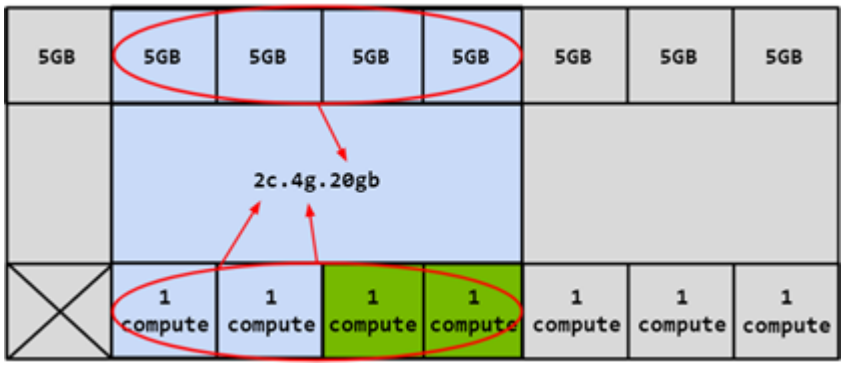


计算实例（CI）

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：



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）：



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** 。

使用 MIG GPU 资源

本节介绍应用如何使用 MIG GPU 资源。

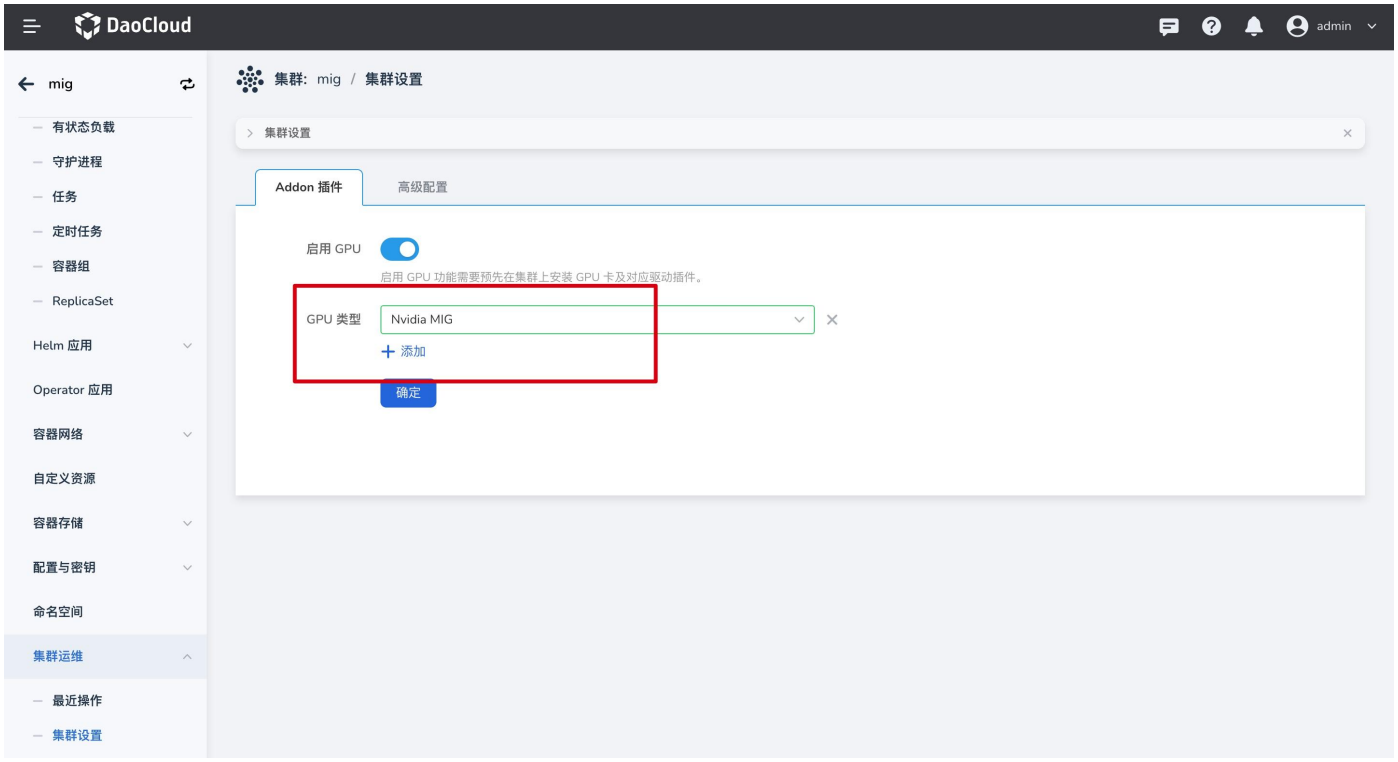
前提条件

- 已启用 NVIDIA DevicePlugin 和 MIG 能力，可参考 GPU Operator 离线安装。
- 集群节点上具有对应型号的 GPU 卡

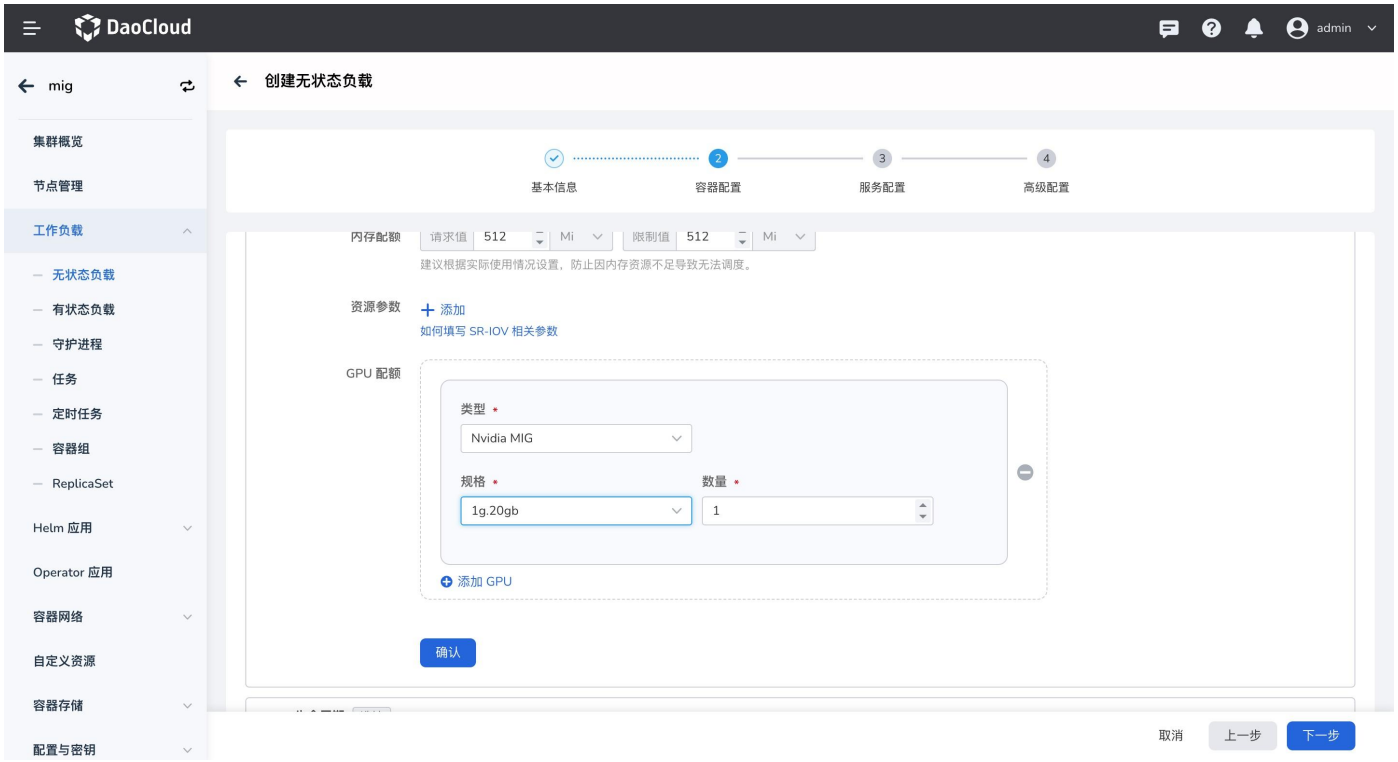
界面使用 MIG GPU

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 。



2. 通过镜像部署应用可选择并使用 NVIDIA MIG 资源。



YAML 配置使用 MIG

**** MIG Single 模式：****

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mig-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mig-demo
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mig-demo
    spec:
      containers:
        - name: mig-demo1
          image: chrstnhntschl/gpu_burn
          resources:
            limits:
              nvidia.com/gpu: 2 # 申请 MIG GPU 的数量
          imagePullPolicy: Always
      restartPolicy: Always
```

**** MIG Mixed 模式：****

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mig-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mig-demo
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mig-demo
    spec:
      containers:
        - name: mig-demo1
          image: chrstnhntschl/gpu_burn
          resources:
            limits:
              nvidia.com/mig-4g.20gb: 1 # 通过 nvidia.com/mig-g.gb 的资源类型公开各个 MIG 设备
          imagePullPolicy: Always
      restartPolicy: Always
```

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      |
|=============================================================================|
```

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	删除指定的 GI 实例
nvidia-smi mig -lgip	查看 GI 的 **profile**
nvidia-smi mig -cgi	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	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

昇腾 GPU 管理

昇腾 NPU 组件安装

本章节提供 昇腾 NPU 驱动和 Device Plugin 的安装指导。

前提条件

1. 安装前请确认支持的 NPU 型号，详情请参考昇腾 NPU 矩阵
2. 请确认 对应 NPU 型号所要求的内核版本是否匹配，详情请参考昇腾 NPU 矩阵
3. 准备 Kubernetes 基础环境

安装步骤

使用 NPU 资源之前，需要完成固件安装、NPU 驱动安装、 Docker Runtime 安装、用户创建、日志目录创建以及 NPU Device Plugin 安装，详情参考如下步骤。

安装固件

1. 安装请请确认内核版本在“二进制安装”安装方式对应的版本范围内，则可以直接安装NPU驱动固件。
2. 固件与驱动下载请参考固件下载地址
3. 固件安装请参考安装 NPU 驱动固件

安装 NPU 驱动

1. 如驱动未安装，请参考昇腾官方文档进行安装：例如 Ascend910，参考 910 驱动安装文档。
2. 运行 **npu-smi info** 命令，并且能够正常返回 NPU 信息，表示 NPU 驱动与固件已就绪。

```
root@atlas9000 ~]# npu-smi info
+-------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc2                 Version: 23.0.rc2                                        |
+---------------------------+---------------+-----------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+===============================================+
| 4     910A                | OK            | 69.4        37                0    / 0            |
| 0                         | 0000:04:00.0  | 0           2303 / 15039      0    / 32768         |
+===========================+===============+===============================================+
+---------------------------+---------------+-----------------------------------------------+
| NPU     Chip              | Process id    | Process name              | Process memory(MB)      |
+===========================+===============+===============================================+
| No running processes found in NPU 4                                                       |
+===========================+===============+===============================================+
```

安装 Docker Runtime

1. 下载 Ascend Docker Runtime

   社区版下载地址：https://www.hiascend.com/zh/software/mindx-dl/community

   ```
   wget -c https://mindx.obs.cn-south-1.myhuaweicloud.com/OpenSource/MindX/MindX%205.0.RC2/MindX%20DL%205.0.RC2/Ascend-docker-runtime_5.0.RC2_linux-x86_64.run
   ```

   安装到指定路径下，依次执行以下两条命令，参数为指定的安装路径:

   ```
   chmod u+x Ascend-docker-runtime_5.0.RC2_linux-x86_64.run
   ./Ascend-docker-runtime_{version}_linux-{arch}.run --install --install-path=<path>
   ```

1. 修改 containerd 配置文件

containerd 无默认配置文件时，依次执行以下3条命令，创建配置文件：

```
mkdir /etc/containerd
containerd config default > /etc/containerd/config.toml
vim /etc/containerd/config.toml
```

containerd 有配置文件时：

```
vim /etc/containerd/config.toml
```

根据实际情况修改 runtime 的安装路径，主要修改 runtime 字段：

```
...
[plugins."io.containerd.monitor.v1.cgroups"]
    no_prometheus = false
[plugins."io.containerd.runtime.v1.linux"]
    shim = "containerd-shim"
    runtime = "/usr/local/Ascend/Ascend-Docker-Runtime/ascend-docker-runtime"
    runtime_root = ""
    no_shim = false
    shim_debug = false
  [plugins."io.containerd.runtime.v2.task"]
    platforms = ["linux/amd64"]
 ...
```

执行以下命令，重启 containerd：

```
systemctl restart containerd
```

用户创建

在对应组件安装的节点上执行以下命令创建用户。

```
# Ubuntu 操作系统
useradd -d /home/hwMindX -u 9000 -m -s /usr/sbin/nologin hwMindX
usermod -a -G HwHiAiUser hwMindX
# Centos 操作系统
useradd -d /home/hwMindX -u 9000 -m -s /sbin/nologin hwMindX
usermod -a -G HwHiAiUser hwMindX
```

日志目录创建

在对应节点创建组件日志父目录和各组件的日志目录，并设置目录对应属主和权限。执行下述命令，创建组件日志父目录。

```
mkdir -m 755 /var/log/mindx-dl
chown root:root /var/log/mindx-dl
```

执行下述命令，创建 Device Plugin 组件日志目录。

```
mkdir -m 750 /var/log/mindx-dl/devicePlugin
chown root:root /var/log/mindx-dl/devicePlugin
```

Note

请分别为所需组件创建对应的日志目录，当前案例中只需要 Device Plugin 组件。 如果有其他组件需求请参考官方文档

安装 Device Plugin

1. 如驱动与 Device Plugin 未安装，请参考昇腾 NPU Device Plugin。
2. 镜像拉取可参考镜像拉取地址：harbor.daocloud.cn/library/ascend-k8sdeviceplugin:v5.0.RC2

!!! note

```
    昇腾镜像仓库中拉取的MindX DL镜像与组件启动yaml中的名字不一致，需要重命名拉取的镜像后才能启动。
    根据以下步骤将2中获取的镜像重新命名，同时建议删除原始名字的镜像。具体操作如下。
```

```
ctr -n k8s.io i tag harbor.daocloud.cn/library/ascend-k8sdeviceplugin:v5.0.RC2 ascend-k8sdeviceplugin:v5.0.RC2
```

1. 获取 **device-plugin-910-v5.0.RC2.yaml** 文件，请参考下载地址
2. 执行 Kube Apply：

```
# 根据环境实际情况选择使用的 yaml 文件，这里环境中使用的是 910 芯片。
# 需要给 node 打上 accelerator=huawei-Ascend910 的label，才能被调度启动 pod。
kubectl label nodes {node-name} accelerator=huawei-Ascend910
# 提交 device-plugin yaml
kubectl apply -f device-plugin-910-v5.0.RC2.yaml
```

注意： **device-plugin-910-v5.0.RC2.yaml** 中的镜像地址是 **ascend-k8sdeviceplugin:v5.0.RC2**

构建 **ascend-k8sdeviceplugin** 镜像：从下载的代码包中有 **Dockerfile** 文件 （详情参考：软件包说明），执行构建命令：

```
# 910 卡构建使用Dockerfile
docker build --no-cache -t  ascend-k8sdeviceplugin:v5.0.RC2 .
```

Dockerfile	**Ascend Device Plugin** 镜像构建文本文件
Dockerfile-310P-1usoc	Atlas 200I Soc A1 核心版上Ascend Device Plugin镜像构建文本文件。

1. NPU Device Plugin 默认安装在 **kube-system** 命名空间下。这是一个 DaemonSet 类型的工作负载， 可以通过 **kubectl get pod -n kube-system | grep ascend** 命令查看，输出如下：

```
[root@atlas9000 ~]# kubectl get pod -n kube-system | grep ascend
ascend-device-plugin-daemonset-6cg8n   1/1     Running   0              2d2h
[root@atlas9000 ~]#
```

应用使用昇腾（Ascend）NPU

本节介绍如何在 d.run 平台使用昇腾 GPU。

前提条件

- 当前集群已安装昇腾（Ascend） NPU 驱动。
- 当前集群内 NPU 卡未进行任何虚拟化操作或被其它应用占用。

快速使用

本文使用昇腾示例库中的 AscentCL 图片分类应用示例。

1. 下载昇腾代码库

   运行以下命令下载昇腾 Demo 示例代码库，并且请记住代码存放的位置，后续需要使用。

```
git clone https://gitee.com/ascend/samples.git
```

2. 准备基础镜像

   此例使用 Ascent-pytorch 基础镜像，可访问昇腾镜像仓库获取。

3. 准备 YAML

ascend-demo.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: resnetinfer1-1-1usoc
spec:
  template:
    spec:
      containers:
        - image: ascendhub.huawei.com/public-ascendhub/ascend-pytorch:23.0.RC2-ubuntu18.04 # Inference image name
          imagePullPolicy: IfNotPresent
          name: resnet50infer
          securityContext:
            runAsUser: 0
          command:
            - "/bin/bash"
            - "-c"
            - |
              source /usr/local/Ascend/ascend-toolkit/set_env.sh &&
              TEMP_DIR=/root/samples_copy_$(date '+%Y%m%d_%H%M%S_%N') &&
              cp -r /root/samples "$TEMP_DIR" &&
              cd "$TEMP_DIR"/inference/modelInference/sampleResnetQuickStart/python/model &&
              wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/003_Atc_Models/resnet50/resnet50.onnx &&
              atc --model=resnet50.onnx --framework=5 --output=resnet50 --input_shape="actual_input_1:1,3,224,224"  --soc_version=Ascend910 &&
              cd ../data &&
              wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog1_1024_683.jpg &&
              cd ../scripts &&
              bash sample_run.sh
          resources:
            requests:
              huawei.com/Ascend910: 1 # Number of the Ascend 910 Processors.
            limits:
              huawei.com/Ascend910: 1 # The value should be the same as that of requests .
          volumeMounts:
            - name: hiai-driver
              mountPath: /usr/local/Ascend/driver
              readOnly: true
            - name: slog
              mountPath: /var/log/npu/conf/slog/slog.conf
            - name: localtime # The container time must be the same as the host time.
              mountPath: /etc/localtime
            - name: dmp
              mountPath: /var/dmp_daemon
            - name: slogd
              mountPath: /var/slogd
            - name: hbasic
              mountPath: /etc/hdcBasic.cfg
            - name: sys-version
              mountPath: /etc/sys_version.conf
            - name: aicpu
              mountPath: /usr/lib64/aicpu_kernels
            - name: tfso
              mountPath: /usr/lib64/libtensorflow.so
            - name: sample-path
              mountPath: /root/samples
      volumes:
        - name: hiai-driver
          hostPath:
            path: /usr/local/Ascend/driver
        - name: slog
          hostPath:
            path: /var/log/npu/conf/slog/slog.conf
        - name: localtime
          hostPath:
            path: /etc/localtime
        - name: dmp
          hostPath:
            path: /var/dmp_daemon
        - name: slogd
          hostPath:
            path: /var/slogd
        - name: hbasic
          hostPath:
            path: /etc/hdcBasic.cfg
        - name: sys-version
          hostPath:
            path: /etc/sys_version.conf
```

```
        - name: aicpu
          hostPath:
            path: /usr/lib64/aicpu_kernels
        - name: tfso
          hostPath:
            path: /usr/lib64/libtensorflow.so
        - name: sample-path
          hostPath:
            path: /root/samples
      restartPolicy: OnFailure
```

以上 YAML 中有一些字段需要根据实际情况进行修改：

a. **atc ... --soc_version=Ascend910** 使用的是 **Ascend910** ，请以实际情况为主 您可以使用 **npu-smi info** 命令查看显卡型号然后加上 Ascend 前缀即可

b. **samples-path** 以实际情况为准

c. **resources** 以实际情况为准

4. 部署 Job 并查看结果

使用如下命令创建 Job：

```
kubectl apply -f ascend-demo.yaml
```

查看 Pod 运行状态：

```
[[root@atlas9000 ~]# kubectl get po
NAME                            READY   STATUS      RESTARTS   AGE
resnetinfer1-1-1usoc-8w76m      0/1     Completed   0          17m
[root@atlas9000 ~]#
```

Pod 成功运行后，查看日志结果。在屏幕上的关键提示信息示例如下图，提示信息中的 Label 表示类别标识，Conf 表示该分类的最大置信度，Class 表示所属类别。这些值可能会根据版本、环境有所不同，请以实际情况为准：

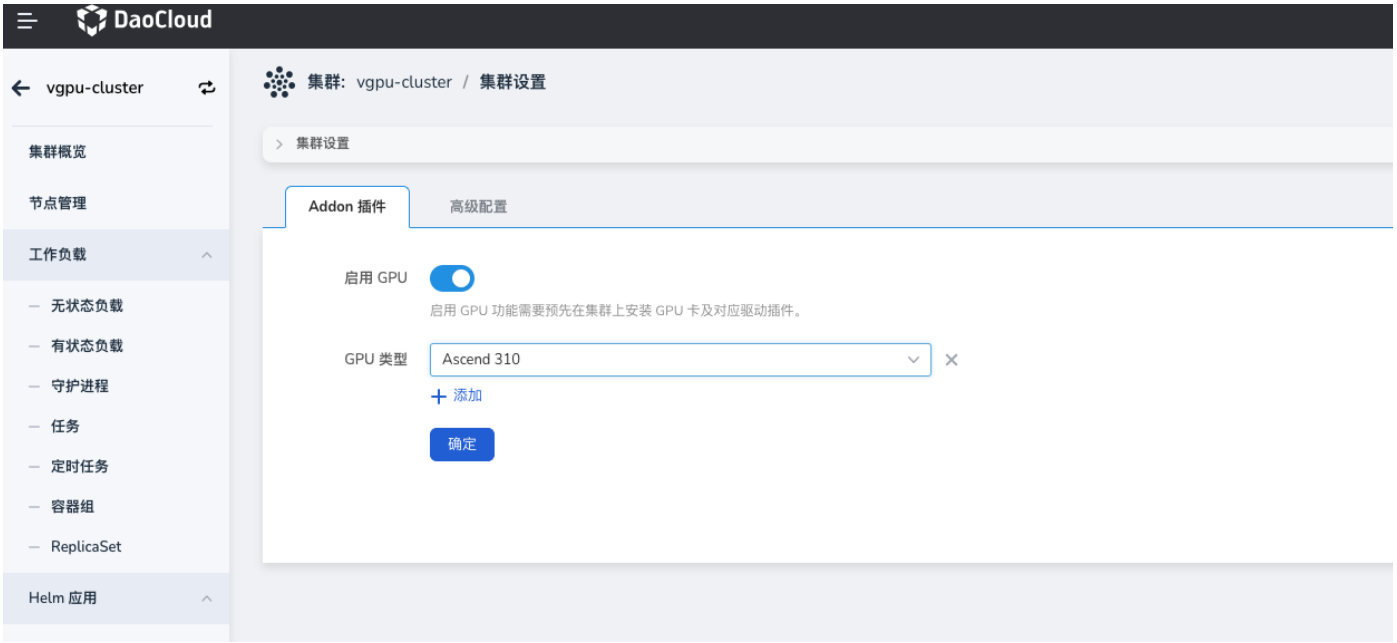
```
 99950K .......... .......... .......... .......... .......... 99%  285M 0s
100000K .......... .......... .......... .......... .......... 99% 85.6M 0s
100050K .......... .........                                    100%  270M=1.4s

2023-11-10 18:15:16 (70.9 MB/s) - 'resnet50.onnx' saved [102470393/102470393]

ATC start working now, please wait for a moment.
...
ATC run success, welcome to the next use.

--2023-11-10 18:15:42--  https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog1_1024_683.jpg
Resolving obs-9be7.obs.cn-east-2.myhuaweicloud.com (obs-9be7.obs.cn-east-2.myhuaweicloud.com)... 122.9.88.37
Connecting to obs-9be7.obs.cn-east-2.myhuaweicloud.com (obs-9be7.obs.cn-east-2.myhuaweicloud.com)|122.9.88.37|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35635 (35K) [image/jpeg]
Saving to: 'dog1_1024_683.jpg'

     0K .......... .......... .......... ....                 100% 1.14M=0.03s

2023-11-10 18:15:42 (1.14 MB/s) - 'dog1_1024_683.jpg' saved [35635/35635]

[INFO] The sample starts to run
out_dog1_1024_683.jpg
label:162  conf:0.902203  class:beagle
******run finish******
[root@atlas9000 ~]#
```

结果图片展示：


label:162 conf:0.902209 class:beagle

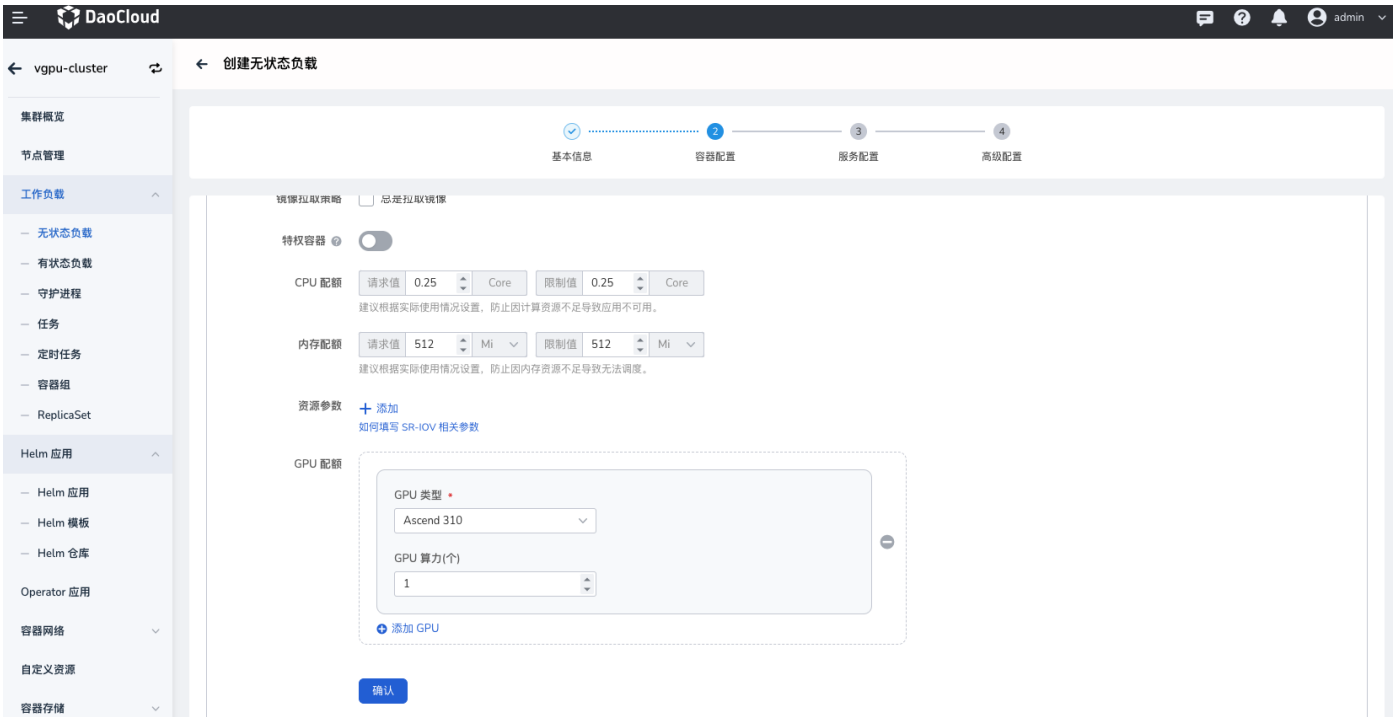
界面使用

1. 确认集群是否已检测 GPU 卡。点击对应 集群 -> 集群设置 -> **Addon** 插件 ， 查看是否已自动启用并自动检测对应 GPU 类型。 目前集群会自动启用 **GPU** ， 并且设置 **GPU** 类型为 **Ascend** 。



2. 部署工作负载，点击对应 集群 -> 工作负载 ， 通过镜像方式部署工作负载，选择类型（Ascend）之后，需要配置应用使用的物理卡数量：

   物理卡数量（**huawei.com/Ascend910**） ：表示当前 Pod 需要挂载几张物理卡，输入值必须为整数且**小于等于**宿主机上的卡数量。



如果上述值配置的有问题则会出现调度失败，资源分配不了的情况。

APP 使用天数智芯（ILUVATAR）GPU

本节介绍如何在 d.run 平台使用天数智芯虚拟 GPU。

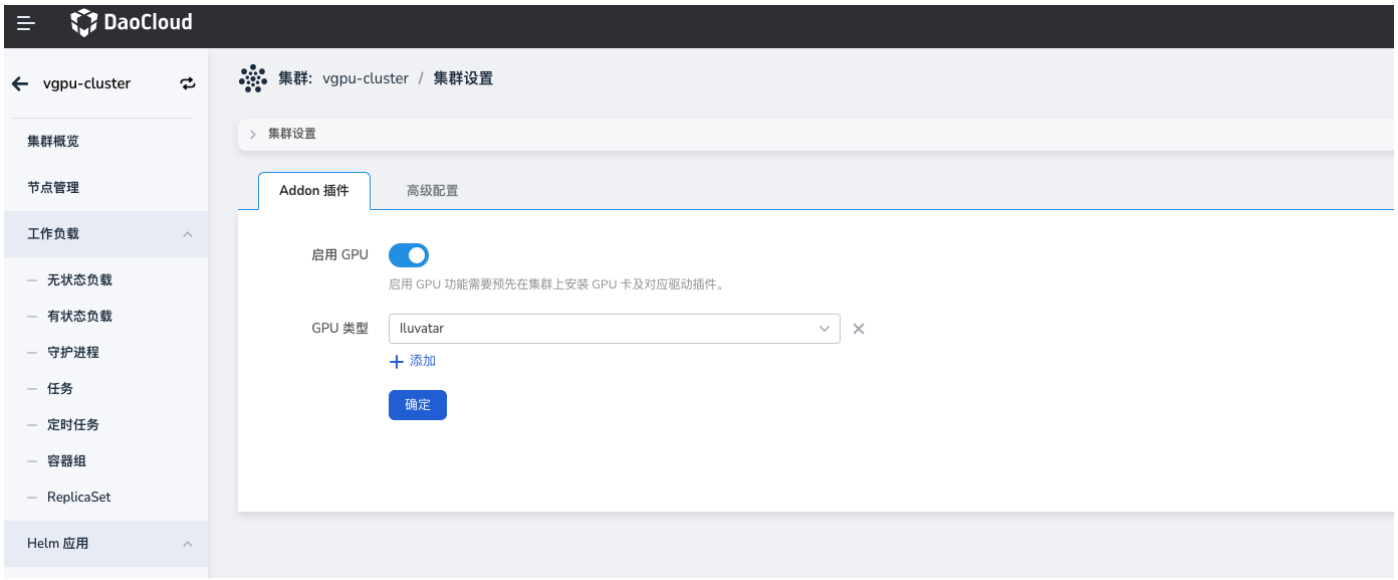
前提条件

- 当前集群已安装天数智芯 GPU 驱动，驱动安装请参考天数智芯官方文档，或联系道客生态团队获取企业级支持：peg-pem@daocloud.io。
- 当前集群内 GPU 卡未进行任何虚拟化操作且未被其它 App 占用。

操作步骤

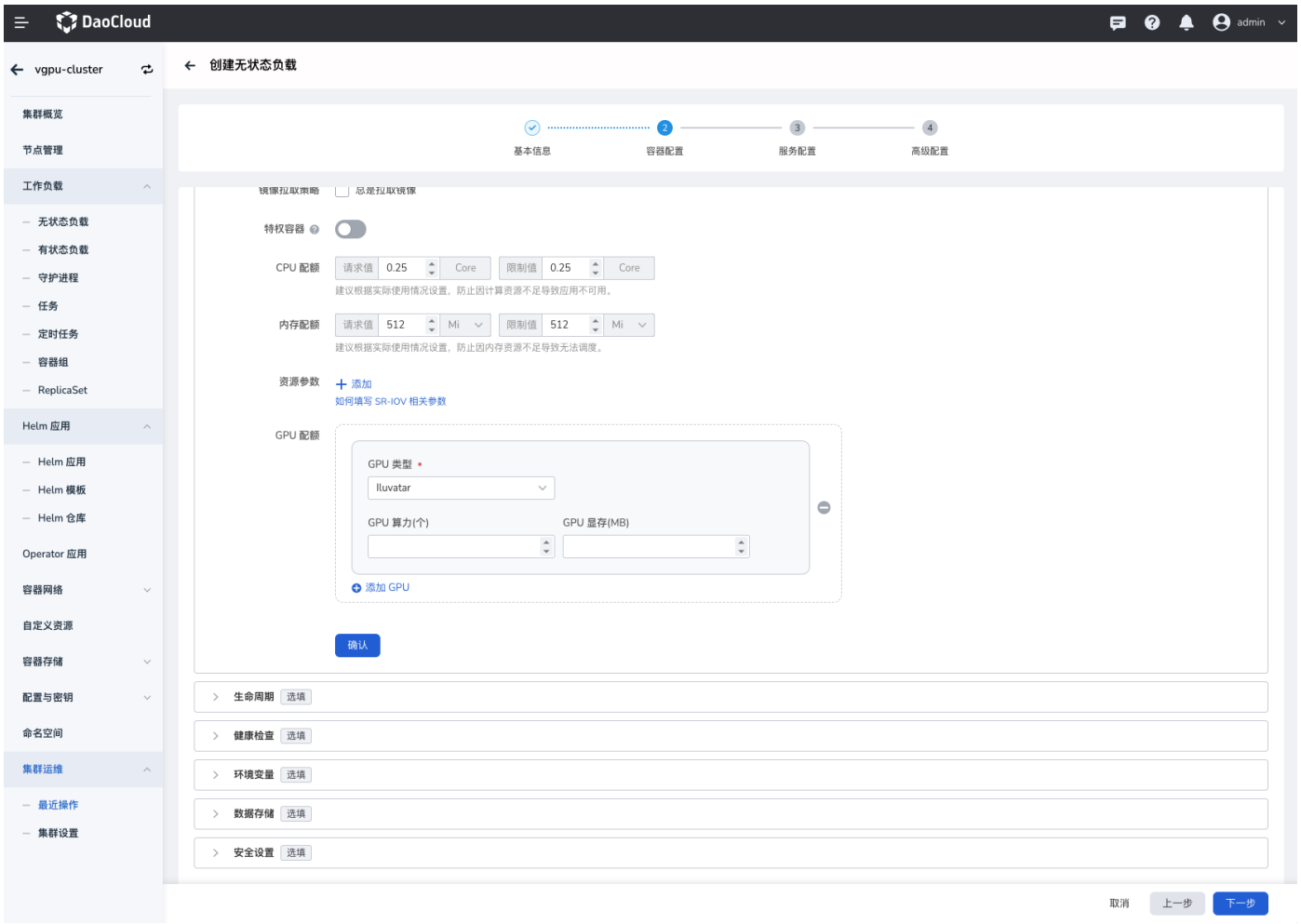
使用界面配置

1. 确认集群是否已检测 GPU 卡。点击对应 集群 -> 集群设置 -> **Addon** 插件 ， 查看是否已自动启用并自动检测对应 GPU 类型。 目前集群会自动启用 **GPU** ， 并且设置 **GPU** 类型为 **Iluvatar** 。



2. 部署工作负载。点击对应 集群 -> 工作负载 ， 通过镜像方式部署工作负载，选择类型（Iluvatar）之后，需要配置 App 使用的 GPU 资源：

- 物理卡数量（iluvatar.ai/vcuda-core）：表示当前 Pod 需要挂载几张物理卡，输入值必须为整数且 小于等于 宿主机上的卡数量。
- 显存使用数量（iluvatar.ai/vcuda-memory）：表示每张卡占用的 GPU 显存，值单位为 MB，最小值为 1，最大值为整卡的显存值。



如果上述值配置的有问题则会出现调度失败，资源分配不了的情况。

使用 YAML 配置

创建工作负载申请 GPU 资源，在资源申请和限制配置中增加 `iluvatar.ai/vcuda-core: 1`、`iluvatar.ai/vcuda-memory: 200` 参数，配置 App 使用物理卡的资源。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: full-iluvatar-gpu-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: full-iluvatar-gpu-demo
  template:
    metadata:
      labels:
        app: full-iluvatar-gpu-demo
    spec:
      containers:
      - image: nginx:perl
        name: container-0
        resources:
          limits:
            cpu: 250m
            iluvatar.ai/vcuda-core: '1'
            iluvatar.ai/vcuda-memory: '200'
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 512Mi
      imagePullSecrets:
      - name: default-secret
```

GPU 配额管理

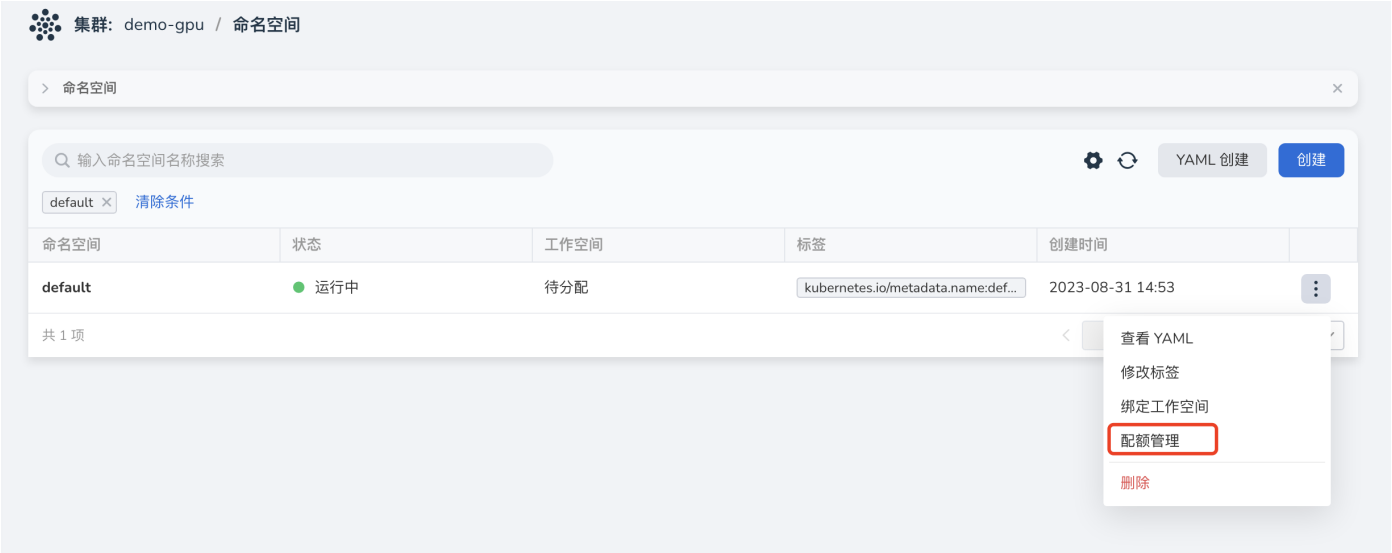
本节介绍如何在 d.run 平台使用 vGPU 能力。

前提条件

当前集群已通过 Operator 或手动方式部署对应类型 GPU 驱动（NVIDIA GPU、NVIDIA MIG、天数、昇腾）

操作步骤

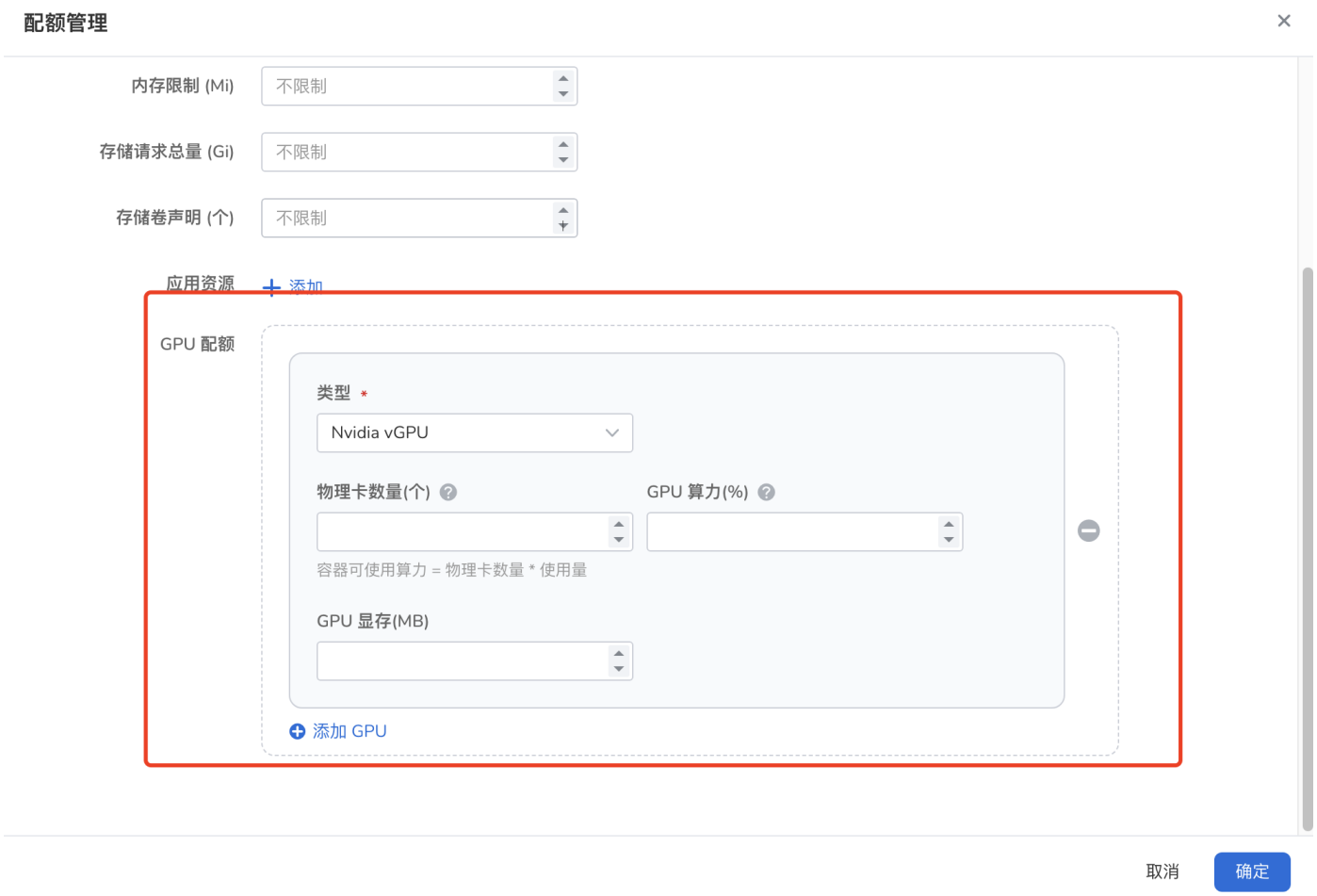
1. 进入 Namespaces 中，点击 配额管理 可以配置当前 Namespace 可以使用的 GPU 资源。



2. 当前命名空间配额管理覆盖的卡类型为：NVIDIA vGPU、NVIDIA MIG、天数、昇腾。

   NVIDIA vGPU 配额管理 ：配置具体可以使用的配额，会创建 ResourcesQuota CR：

- 物理卡数量（nvidia.com/vgpu）：表示当前 POD 需要挂载几张物理卡，并且要 小于等于 宿主机上的卡数量。
- GPU 算力（nvidia.com/gpucores）：表示每张卡占用的 GPU 算力，值范围为 0-100；如果配置为 0，则认为不强制隔离；配置为 100，则认为独占整张卡。
- GPU 显存（nvidia.com/gpumem）：表示每张卡占用的 GPU 显存，值单位为 MB，最小值为 1，最大值为整卡的显存值。


配额管理
内存限制 (Mi)
不限制
存储请求总量 (Gi)
不限制
存储卷声明 (个)
不限制
应用资源
添加
GPU 配额
类型 *
Nvidia vGPU
物理卡数量(个)
GPU 算力(%)
容器可使用算力 = 物理卡数量 * 使用量
GPU 显存(MB)
添加 GPU
取消
确定

GPU 监控指标

本页列出一些常用的 GPU 监控指标。

集群维度

指标名称	英文	描述	指标	表
GPU 卡数	Total GPU	集群下所有的 GPU 卡数量，切分的 MIG 实例也将会被统计成的单张的物理卡	`count(DCGM_FI_DEV_COUNT{cluster="$cluster",node=~"${node}"})`	数
GPU 平均使用率（整卡）	GPU Avg Utilization	集群下所有 GPU 卡的平均使用率	`avg(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s]))`	数
GPU 平均使用率（MIG）	GPU Avg Utilization（Only MIG Enably）	当启用 MIG 特性后，集群下所有 GPU 卡的平均使用率	`avg(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s]))`	数
GPU 卡平均显存使用率	GPU Avg Memory Utilization	集群下所有 GPU 卡的平均显存使用率	`sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s])) / sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} + DCGM_FI_DEV_FB_FREE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} [29s])) * 100`	数
GPU 卡功率	GPU Power Usage	集群下所有 GPU 卡的功率	`DCGM_FI_DEV_POWER_USAGE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`	柱
GPU 卡温度	GPU Temperature	集群下所有 GPU 卡的温度	`DCGM_FI_DEV_GPU_TEMP{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`	柱
GPU 使用率细节（整卡）	GPU Utilization Details	24 小时内，集群下所有 GPU 卡的使用率细节（包含 max、avg、current）	`DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`	折
GPU使用率细节（mig）	GPU Utilization Details（Only MIG Enably）	24 小时内，当启用 MIG 特性后，集群下所有 GPU 卡的使用率细节。（包含 max、avg、current）	`DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} * 100`	折
GPU 显存使用量	GPU Memory Used Details	24 小时内，集群下所有 GPU 卡的显存使用量细节（包含min 、max、avg、current）	`DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`	折
GPU 显存复制使用率	GPU Memory Copy Utilization	集群下所有 GPU 卡的显存复制使用率	`DCGM_FI_DEV_MEM_COPY_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`	折

节点维度

指标名称	英文	描述	指标
GPU 卡数	Total GPU	节点上所有的 GPU 卡数量，切分的 MIG 实例也将会被统计成的单张的物理卡	`count(DCGM_FI_DEV_COUNT{cluster="$cluster",node=~"${node}"})`
GPU 模式	GPU Mode	节点上 GPU 卡的模式使用模式，包含 整卡模式、MIG 模式、vGPU 模式	`topk(1,DCGM_FI_DEV_MIG_MODE{cluster="$cluster",node=~"$node"})`
GPU 平均使用率（整卡）	GPU Avg Utilization	节点上所有 GPU 卡的平均使用率	`avg(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s]))`
GPU 平均使用率（MIG）	GPU Avg Utilization（Only MIG Enably）	当启用 MIG 特性后，节点上所有 GPU 卡的平均使用率	`avg(max_over_time(DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s] * 100))`
GPU 卡平均显存使用率	GPU Avg Memory Utilization	节点上所有 GPU 卡的平均显存使用率	`sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}[29s])) / sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} + DCGM_FI_DEV_FB_FREE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} [29s])) * 100`
GPU 驱动版本	GPU Driver Version	节点上 GPU 卡驱动的版本信息	`DCGM_FI_DEV_MIG_MODE{cluster="$cluster",node=~"$node",gpu=~"$gpu"} {{modelName}}`
GPU 卡型号/规格	GPU Specifications	节点上 GPU 卡规格信息	`DCGM_FI_DEV_MIG_MODE{cluster="$cluster",node=~"$node",gpu=~"$gpu"}`
GPU 使用率细节（整卡）	GPU Utilization Details	24 小时内，节点上所有 GPU 卡的使用率细节（包含 max、avg、current）	`DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`
GPU 使用率细节（MIG）	GPU Utilization Details（Only MIG Enably）	24 小时内，当启用 MIG 特性后，节点上所有 GPU 卡的使用率细节。（包含 max、avg、current）	`DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"} * 100`
GPU 显存使用量	GPU Memory Used Details	24 小时内，节点上所有 GPU 卡的显存使用量细节（包含min 、max、avg、current）	`DCGM_FI_DEV_FB_USED{cluster="$cluster",node=~"${node}", gpu=~"${gpu}"}`

工作负载维度

维度	指标名称	英文	描述	指标
应用概览	Pod GPU 使用率（整卡）	Pod GPU Utilization	当前 Pod 所使用到的 GPU 卡的比率	`DCGM_FI_DEV_GPU_UTIL{cluster="$cluster",exported_nar`
应用概览	Pod GPU 使用率（MIG）	Pod GPU Utilization（Only MIG Enably）	当启用 MIG 特性后，当前 Pod 所使用到的 GPU 卡的比率	`DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster",exp * 100`
应用概览	Pod GPU 使用率（vGPU）	Pod GPU Utilization（vGPU）	当启用 vGPU 特性后，当前 Pod 所使用到的 GPU 卡的比率	`vGPUCorePercentage{cluster="$cluster",exported_names`
应用概览	Pod GPU 显存使用率	Pod GPU Memory Utilization	当前 Pod 所使用到的 GPU 卡的显存比率	`DCGM_FI_DEV_FB_USED{cluster="$cluster",exported_name`
应用概览	Pod GPU 显存使用率（vGPU）	Pod GPU Memory Utilization（vGPU）	当前 Pod 所使用到的 GPU 卡的显存比率(vGPU 模式)	`vGPUMemoryPercentage{cluster="$cluster",exported_nar`
应用概览	Pod 显存使用量	Pod 显存使用量	当前 Pod 所使用到的 GPU 卡的显存量	`DCGM_FI_DEV_FB_USED{cluster="$cluster",exported_name`
应用概览	Pod 显存使用量（vGPU）	Pod 显存使用量（vGPU）	当前 Pod 所使用到的 GPU 卡的显存比率(vGPU 模式)	`sum(GPUDeviceMemoryLimit{cluster="$cluster"}) * vGPUMemoryPercentage{cluster="$cluster",exported_nar`
应用概览	Pod GPU 显存复制使用率	Pod GPU Memory Copy Utilization	当前 Pod 所使用到的 GPU 卡的显存显存复制比率	`DCGM_FI_DEV_MEM_COPY_UTIL{cluster="$cluster",exporte`
应用概览	Pod 解码使用率	Pod Decode Utilization	当前 Pod 所使用到的 GPU 卡解码引擎比率	`DCGM_FI_DEV_DEC_UTIL{cluster="$cluster",exported_nar`
应用概览	Pod 编码使用率	Pod Encode Utilization	当前 Pod 所使用到的 GPU 卡编码引擎比率	`DCGM_FI_DEV_ENC_UTIL{cluster="$cluster",exported_nar`
GPU 卡-算力&显存	GPU 使用率细节（整卡）	GPU Utilization Details	24 小时内，Pod 关联的 GPU 卡的使用率细节（包含 max、avg、current）	`DCGM_FI_DEV_GPU_UTIL{cluster="$cluster", UUID="${gpu`
GPU 卡-算力&显存	GPU使用率细节（MIG）	GPU Utilization Details（Only MIG Enably）	24 小时内，当启用 MIG 特性后，Pod 关联的 GPU 卡的使用率细节。（包含 max、avg、current）	`DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster", UU`
GPU 卡-算力&显存	GPU 显存使用量	GPU Memory Used Details	24 小时内，Pod 关联的 GPU 卡的	`DCGM_FI_DEV_FB_USED{cluster="$cluster", UUID="${gpu`

维度	指标名称	英文	描述	指标
GPU 卡-算力&显存	GPU 显存复制使用率	GPU Memory Copy Utilization	Pod 关联的 GPU 卡的显存复制使用率	`DCGM_FI_DEV_MEM_COPY_UTIL{cluster="$cluster", UUID="`
GPU 卡-引擎概览	GPU 图形引擎活动百分比	GPU Graphics Engine Active	表示在一个监控周期内，Graphics 或 Compute 引擎处于 Active 的时间占总的时间的比例。	`DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster="$cluster", UU`
GPU 卡-引擎概览	GPU DRAM 活动百分比	GPU DRAM Active	表示内存带宽利用率（Memory BW Utilization）	`DCGM_FI_PROF_DRAM_ACTIVE{cluster="$cluster", UUID="$`
GPU 卡-引擎概览	Tensor 核心引擎使用率	GPU Tensor Core Engine Active	表示在一个监控周期内，Tensor Core管道（Pipe）处于Active时间占总时间的比例。	`DCGM_FI_PROF_PIPE_TENSOR_ACTIVE{cluster="$cluster",`
GPU 卡-引擎概览	FP16 引擎使用率	GPU FP16 Engine Active	表示在一个监控周期内，FP16 管道处于 Active 的时间占总的时间的比例。	`DCGM_FI_PROF_PIPE_FP16_ACTIVE{cluster="$cluster", UU`
GPU 卡-引擎概览	FP32 引擎使用率	GPU FP32 Engine Active	表示在一个监控周期内，FP32 管道处于 Active 的时间占总的时间的比例。	`DCGM_FI_PROF_PIPE_FP32_ACTIVE{cluster="$cluster", UU`
GPU 卡-引擎概览	FP32 引擎使用率	GPU FP64 Engine Active	表示在一个监控周期内，FP64 管道处于 Active 的时间占总的时间的比例。	`DCGM_FI_PROF_PIPE_FP64_ACTIVE{cluster="$cluster", UU`
GPU 卡-引擎概览	GPU 解码使用率	GPU Decode Utilization	GPU 卡解码引擎比率	`DCGM_FI_DEV_DEC_UTIL{cluster="$cluster", UUID="${gpu`
GPU 卡-引擎概览	GPU 编码使用率	GPU Encode Utilization	GPU 卡编码引擎比率	`DCGM_FI_DEV_ENC_UTIL{cluster="$cluster", UUID="${gpu`
GPU 卡-温度&功耗	GPU 卡温度	GPU Temperature	集群下所有 GPU 卡的温度	`DCGM_FI_DEV_GPU_TEMP{cluster="$cluster", UUID="${gpu`
GPU 卡-温度&功耗	GPU 卡功率	GPU Power Usage	集群下所有 GPU 卡的功率	`DCGM_FI_DEV_POWER_USAGE{cluster="$cluster", UUID="$`
GPU 卡-温度&功耗	GPU 卡-总耗能	GPU Total Energy Consumption	GPU 卡总共消耗的能量	`sum(DCGM_FI_DEV_POWER_USAGE{cluster="$cluster", UUI`
GPU 卡-温度&功耗	GPU 卡内存频率	GPU Memory Clock	内存频率	`DCGM_FI_DEV_MEM_CLOCK{cluster="$cluster",UUID="${gpu`

维度	指标名称	英文	描述	指标
GPU 卡-温度&功耗	GPU 卡应用SM 时钟频率	GPU APP SM Clock	应用的SM 时钟频率	`DCGM_FI_DEV_APP_SM_CLOCK{cluster="$cluster",UUID="$`
GPU 卡-温度&功耗	GPU 卡应用内存频率	GPU APP Memory Clock	应用内存频率	`DCGM_FI_DEV_APP_MEM_CLOCK{cluster="$cluster",UUID="$`
GPU 卡-温度&功耗	GPU 卡视频引擎频率	GPU Video Clock	视频引擎频率。	`DCGM_FI_DEV_VIDEO_CLOCK{cluster="$cluster",UUID="${`
GPU 卡-温度&功耗	GPU 卡降频原因	GPU-Clock Throttle Reasons	降频原因	`DCGM_FI_DEV_CLOCK_THROTTLE_REASONS{cluster="$cluste`
GPU 卡-Other details	PCLe 传输速率	PCIE TX BYTES	节点 GPU 卡通过 PCIe 总线传输的数据速率。	`rate(DCGM_FI_PROF_PCIE_RX_BYTES{cluster="$cluster",U`
GPU 卡-Other details	PCLe 接收速率	PCIE RX BYTES	节点 GPU 卡通过 PCIe 总线接收的数据速率。	`rate(DCGM_FI_PROF_PCIE_TX_BYTES{cluster="$cluster",U`

VOLCANO 安装和使用

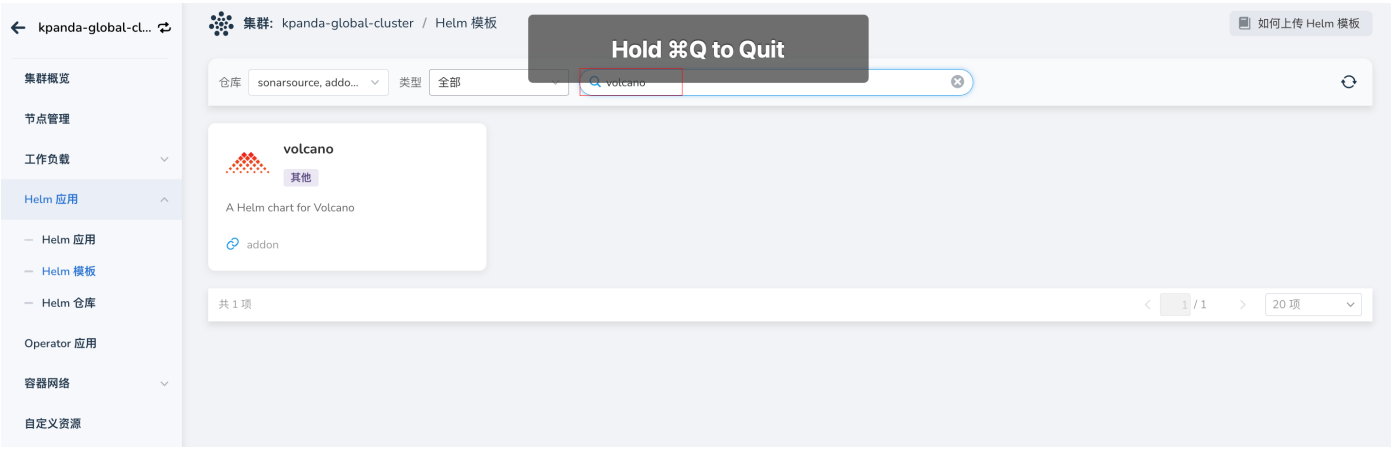
Volcano 是 CNCF 下首个基于 Kubernetes 的容器批处理计算平台，专注于高性能计算场景。 它填补了 Kubernetes 在机器学习、大数据、科学计算等领域缺失的功能，为这些高性能工作负载提供了必要的支持。

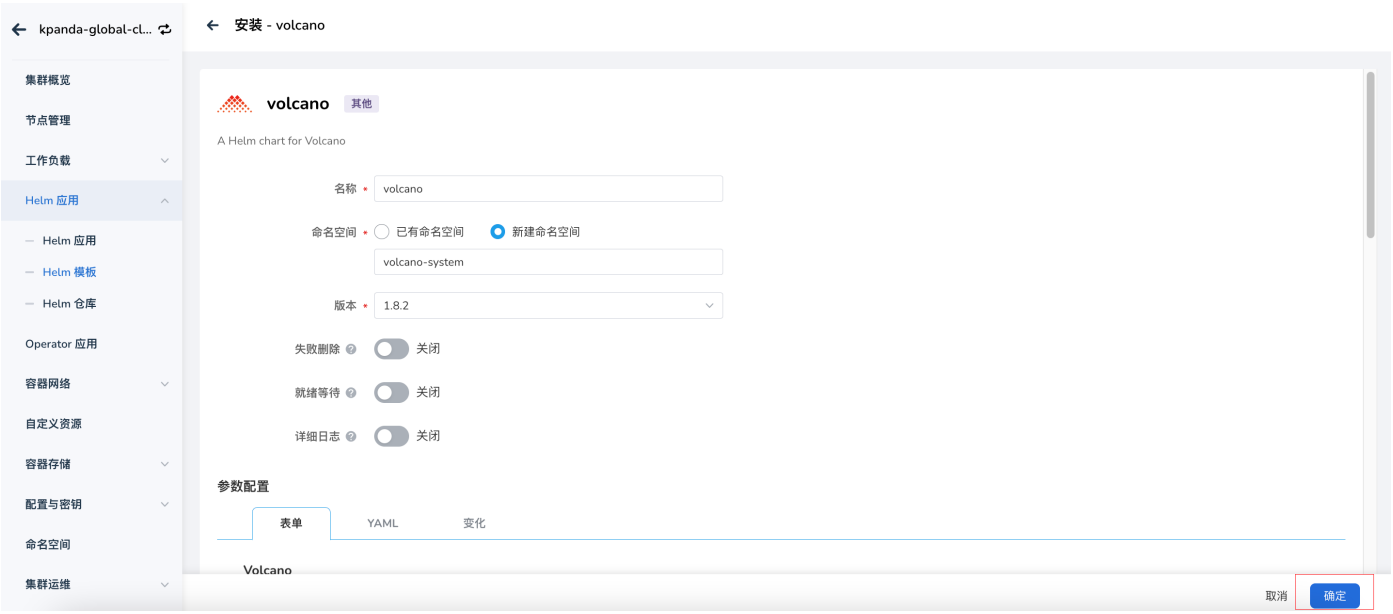
Volcano 与主流计算框架无缝对接，如 Spark、TensorFlow、PyTorch 等，并支持异构设备的混合调度，包括 CPU 和 GPU。

本文介绍如何安装和使用 volcano。

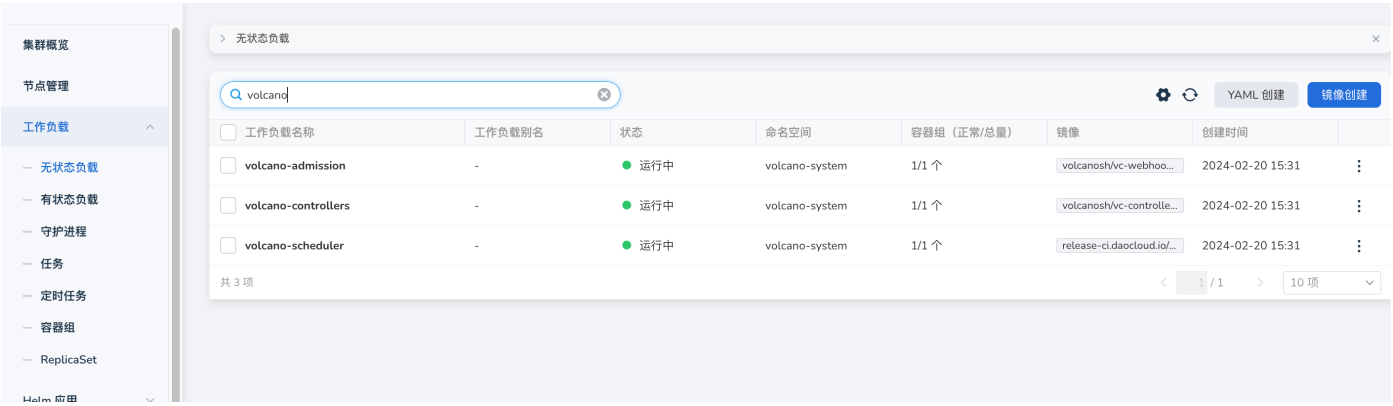
安装 Volcano

1. 在 集群详情 -> **Helm** 应用 -> **Helm** 模板 中找到 Volcano 并安装。





2. 检查并确认 Volcano 是否安装完成，即 volcano-admission、volcano-controllers、volcano-scheduler 组件是否正常运行。



Volcano 使用场景

使用 Volcano 调度 Job 资源

- Volcano 是单独的调度器，在创建工作负载的时候指定调度器的名称（schedulerName: volcano）。
- volcanoJob 资源是 Volcano 对 Job 的扩展，它将 job 拆分成更小的工作单位 task，task 之间可以相互作用。

使用示例：

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: nginx-job
spec:
  minAvailable: 2
  schedulerName: volcano
  tasks:
    - replicas: 1
      name: master
      template:
        spec:
          containers:
            - image: docker.m.daocloud.io/library/nginx:latest
              name: mpimaster
    - replicas: 2
      name: worker
      template:
        spec:
          containers:
            - image: docker.m.daocloud.io/library/nginx:latest
              name: mpiworker
```

并行计算 mpi

使用示例：

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: lm-mpi-job
  labels:
    "volcano.sh/job-type": "MPI" # volcano 原生支持 MPI 的调度作业
spec:
  minAvailable: 4
  schedulerName: volcano
  plugins:
    ssh: [] # volcano 插件，master 和 worker 之间可以免密登录
    svc: [] # master 和 worker 之间通过网络访问，自动创建 headless svc 资源
  policies:
    - event: PodEvicted
      action: RestartJob
  tasks:
    - replicas: 1
      name: mpimaster
      policies:
        - event: TaskCompleted
          action: CompleteJob
      template:
        spec:
          containers:
            - command:
                - /bin/sh
                - -c
                - |
                  MPI_HOST=`cat /etc/volcano/mpiworker.host | tr "\n" ","`;
```

```
                mkdir -p /var/run/sshd; /usr/sbin/sshd;
                mpiexec --allow-run-as-root --host ${MPI_HOST} -np 3 mpi_hello_world;
            image: docker.m.daocloud.io/volcanosh/example-mpi:0.0.1
            name: mpimaster
            ports:
              - containerPort: 22
                name: mpijob-port
            workingDir: /home
            resources:
              requests:
                cpu: "500m"
              limits:
                cpu: "500m"
        restartPolicy: OnFailure
        imagePullSecrets:
          - name: default-secret
    - replicas: 3
      name: mpiworker
      template:
        spec:
          containers:
            - command:
                - /bin/sh
                - -c
                - |
                mkdir -p /var/run/sshd; /usr/sbin/sshd -D;
            image: docker.m.daocloud.io/volcanosh/example-mpi:0.0.1
            name: mpiworker
            ports:
              - containerPort: 22
                name: mpijob-port
            workingDir: /home
            resources:
              requests:
                cpu: "1000m"
              limits:
                cpu: "1000m"
        restartPolicy: OnFailure
        imagePullSecrets:
          - name: default-secret
```

Volcano 支持 TensorFlow

使用示例：

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: tensorflow-benchmark
  labels:
    "volcano.sh/job-type": "Tensorflow" # volcano 原生支持 tensorflow 的调度作业
spec:
  minAvailable: 3
  schedulerName: volcano
  plugins:
    env: []
    svc: []
  policies:
    - event: PodEvicted
      action: RestartJob
  tasks:
    - replicas: 1
      name: ps
      template:
        spec:
          imagePullSecrets:
            - name: default-secret
          containers:
            - command:
                - sh
                - -c
                - |
                PS_HOST=`cat /etc/volcano/ps.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                WORKER_HOST=`cat /etc/volcano/worker.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                python tf_cnn_benchmarks.py --batch_size=32 --model=resnet50 --variable_update=parameter_server --flush_stdout=true --num_gpus=1 --
local_parameter_device=cpu --device=cpu --data_format=NHWC --job_name=ps --task_index=${VK_TASK_INDEX} --ps_hosts=${PS_HOST} --worker_hosts=${WORKER_HOST}
            image: docker.m.daocloud.io/volcanosh/example-tf:0.0.1
            name: tensorflow
            ports:
              - containerPort: 2222
                name: tfjob-port
            resources:
              requests:
                cpu: "1000m"
                memory: "2048Mi"
              limits:
                cpu: "1000m"
                memory: "2048Mi"
            workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
        restartPolicy: OnFailure
    - replicas: 2
      name: worker
```

```
    policies:
      - event: TaskCompleted
        action: CompleteJob
    template:
      spec:
        imagePullSecrets:
          - name: default-secret
        containers:
          - command:
              - sh
              - -c
              - |
                PS_HOST=`cat /etc/volcano/ps.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                WORKER_HOST=`cat /etc/volcano/worker.host | sed 's/$/&:2222/g' | tr "\n" ","`;
                python tf_cnn_benchmarks.py --batch_size=32 --model=resnet50 --variable_update=parameter_server --flush_stdout=true --num_gpus=1 --
local_parameter_device=cpu --device=cpu --data_format=NHWC --job_name=worker --task_index=${VK_TASK_INDEX} --ps_hosts=${PS_HOST} --worker_hosts=${WORKER_HOST}
            image: docker.m.daocloud.io/volcanosh/example-tf:0.0.1
            name: tensorflow
            ports:
              - containerPort: 2222
                name: tfjob-port
            resources:
              requests:
                cpu: "2000m"
                memory: "2048Mi"
              limits:
                cpu: "2000m"
                memory: "4096Mi"
            workingDir: /opt/tf-benchmarks/scripts/tf_cnn_benchmarks
        restartPolicy: OnFailure
```

如果您想了解 volcano 更多功能特性和使用场景，可以参考 Volcano 介绍。

1.2.3 可观测性

介绍

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 d.run 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

应用场景

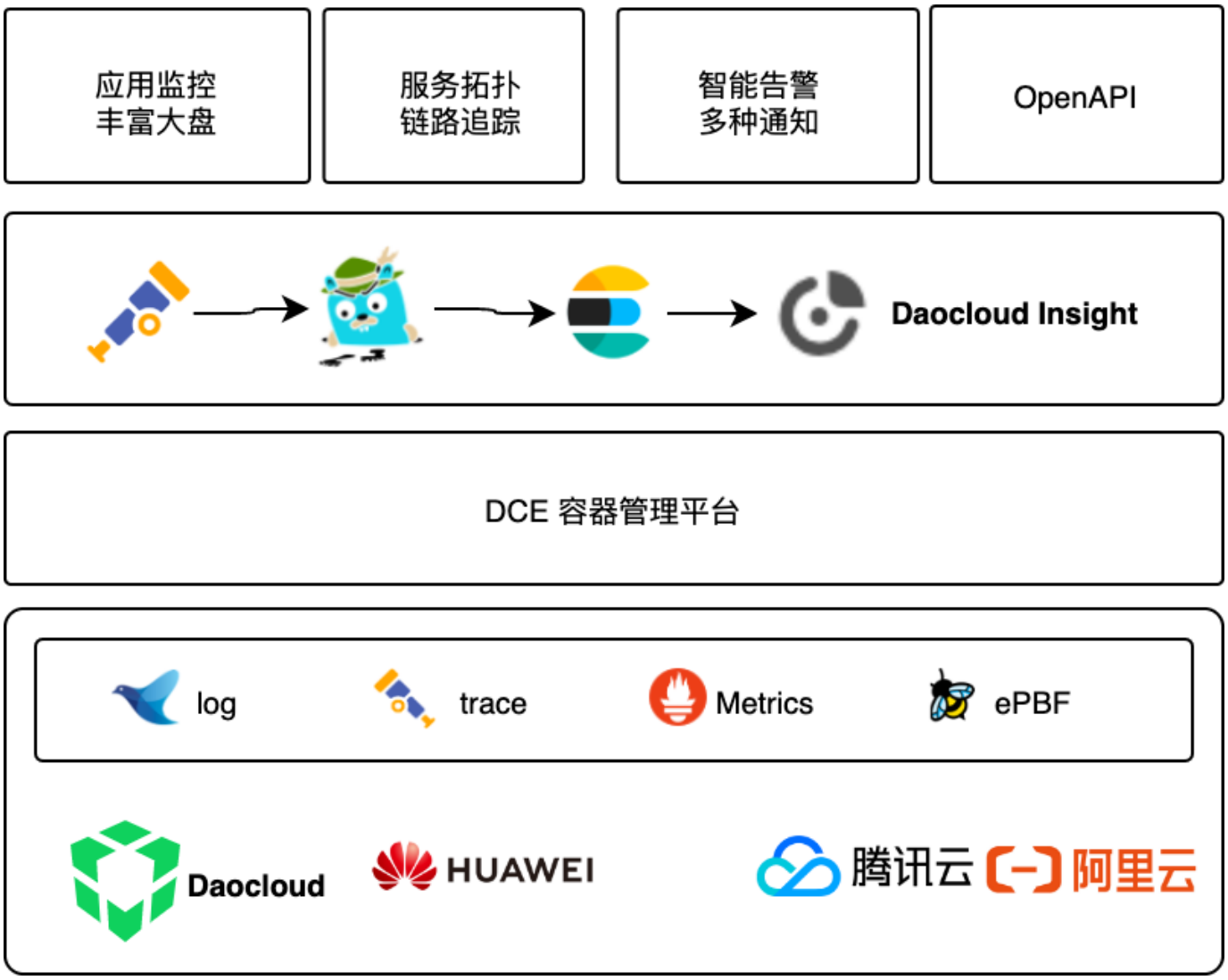
多集群数据统一采集及观测

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。



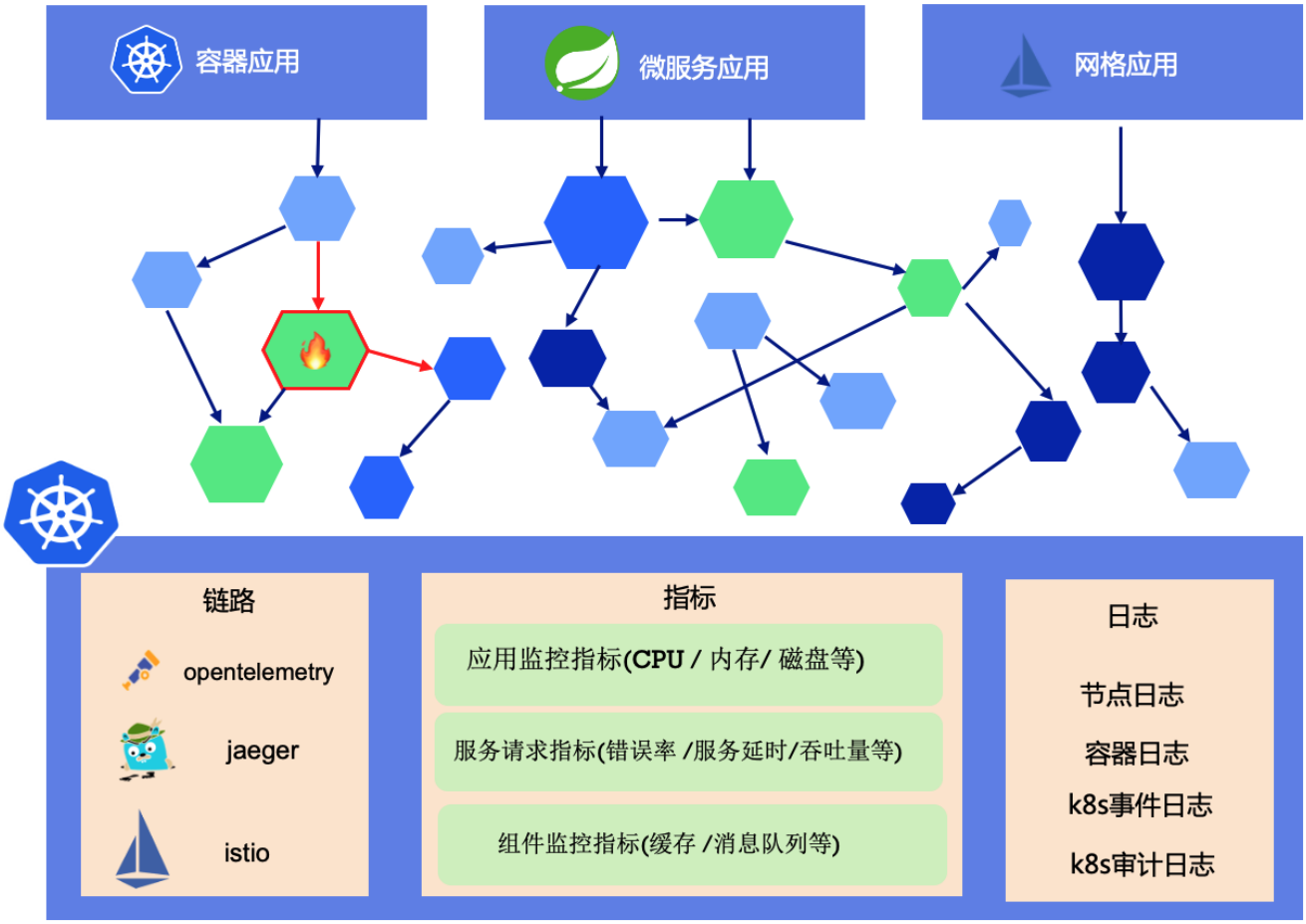
快速故障定位及排查

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。



功能列表

本页列出了可观测性 Insight 的功能特性。

类别	子类	描述
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。
基础设施	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
	命名空间监控	支持查看命名空间中运行的资源数量统计，以及命名空间中容器组使用的 CPU、内存量的总和。
	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
	事件	支持查看集群中产生的 Kubernetes 事件记录集合，并支持按照事件类型、对象、原因等进行查询。
	拨测	基于黑盒监控定期通过 HTTP、TCP 等方式对目标进行连通性测试，快速发现正在发生的故障。
指标	普通查询	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
	高级查询	支持通过原生 PromQL 语句，查询指标图表及数据详情
日志	普通查询	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
	高级查询	支持原生 lucene 语法，快速查询目标日志
	日志上下文	点击单行日志右侧的图标可查看该行日志的上下文信息。
	日志下载	支持下载一段时间内的日志 支持导出单条日志上下文的内容 支持自定义日志下载的字段
链路追踪	服务拓扑	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
	服务	可查看当前接入链路数据的服务列表，以及服务最近 15 分钟的吞吐率、错误率、请求延时 点击服务可查看所选服务最近 15 分钟的流量趋势以及该服务操作的聚合指标
	调用链	默认查询所选服务最近 15 分钟中的所有请求以及请求状态、延时、Span 数等 点击列表后侧的图标，可查询该链路的相关容器日志和链路日志。
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	

类别	子类	描述
		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
	告警静默	通过配置静默规则，可以在指定时间段内不再接收告警通知。
	告警抑制	通过配置抑制规则，可以抑制或阻止与某些特定告警相关的其他告警通知。
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
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

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	✓	✗	✗	✗
基础设施	查看集群监控	✓	✓	✗	✗
	查看节点监控	✓	✓	✗	✗
	查看命名空间监控	✓	✓	✓	✓
	查看工作负载监控	✓	✓	✓	✓
	查看事件	✓	✓	✓	✓
	查看拨测任务	✓	✓	✓	✓
	创建拨测任务	✓	✓	✓	✗
	编辑拨测任务	✓	✓	✓	✗
	删除拨测任务	✓	✓	✓	✗
指标	查询节点指标	✓	✓	✗	✗
	查询工作负载指标	✓	✓	✓	✓
	高级查询	✓	✓	✓	✓
日志	查询节点日志	✓	✓	✗	✗
	查询容器日志	✓	✓	✓	✓
	Lucene 语法查询节点日志	✓	✓	✗	✗
	Lucene 语法查询容器日志	✓	✓	✓	✓
链路追踪	查看服务拓扑	✓	✓	✓	✓
	查看服务列表	✓	✓	✓	✓
	查询调用链	✓	✓	✓	✓
	TraceID 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	✓	✓	✗	✗
		✓	✓	✓	✓

菜单	操作	Admin / Kpanda Owner	Cluster Admin	NS Admin / NS Edit	NS View
	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	✓	✓	✓	✗
	创建日志规则	✓	✓	✓	✗
	创建事件规则	✓	✓	✓	✗
	删除自定义告警规则	✓	✓	✓	✗
	查看内置告警规则	✓	✗	✗	✗
	修改内置告警规则	✓	✗	✗	✗
	YAML 导入告警规则	✓	✓	✓	✗
告警模板	查看告警模板列表	✓	✓	✓	✓
	创建告警模板	✓	✗	✗	✗
	编辑告警模板	✓	✗	✗	✗
	删除告警模板	✓	✗	✗	✗
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	✓	✗	✗	✗
告警静默	查看静默规则列表	✓	✓	✓	✓
	创建静默规则	✓	✓	✓	✗
	编辑静默规则	✓	✓	✓	✗
	删除静默规则	✓	✓	✓	✗
告警抑制	查看抑制规则列表	✓	✓	✓	✓

菜单	操作	Admin / Kpanda Owner	Cluster Admin	NS Admin / NS Edit	NS View
	创建抑制规则	✓	✓	✓	✗
	编辑抑制规则	✓	✓	✓	✗
	删除抑制规则	✓	✓	✓	✗
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	✓	✗	✗	✗
	修改系统配置	✓	✗	✗	✗

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。

仪表盘

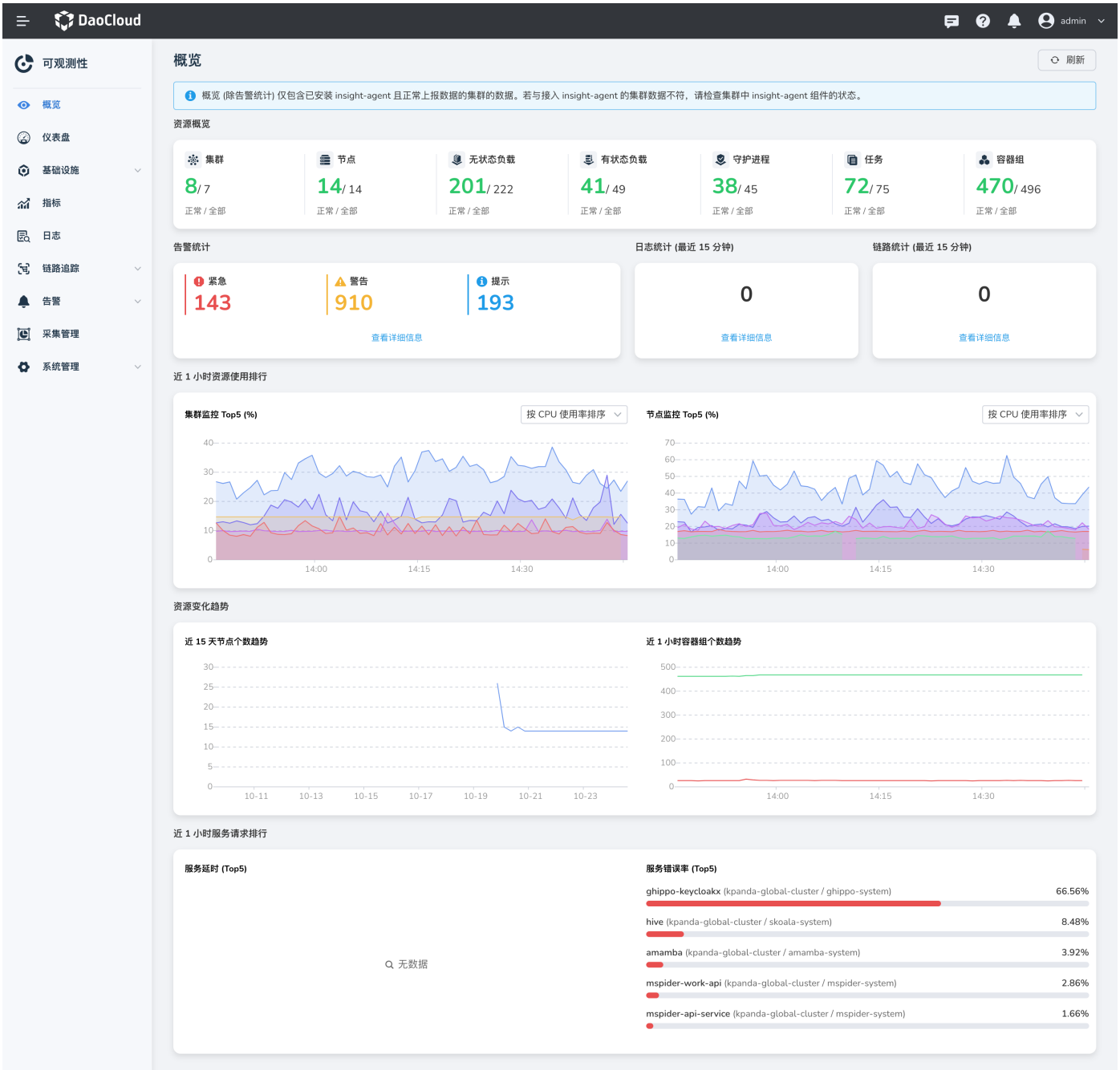
概览

概率 仅统计已安装 **insight-agent** 且其运行状态为正常的集群数据。可在概览中多集群的资源概况：

- 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。

操作步骤

1. 在左边导航栏选择 概览 。



仪表盘

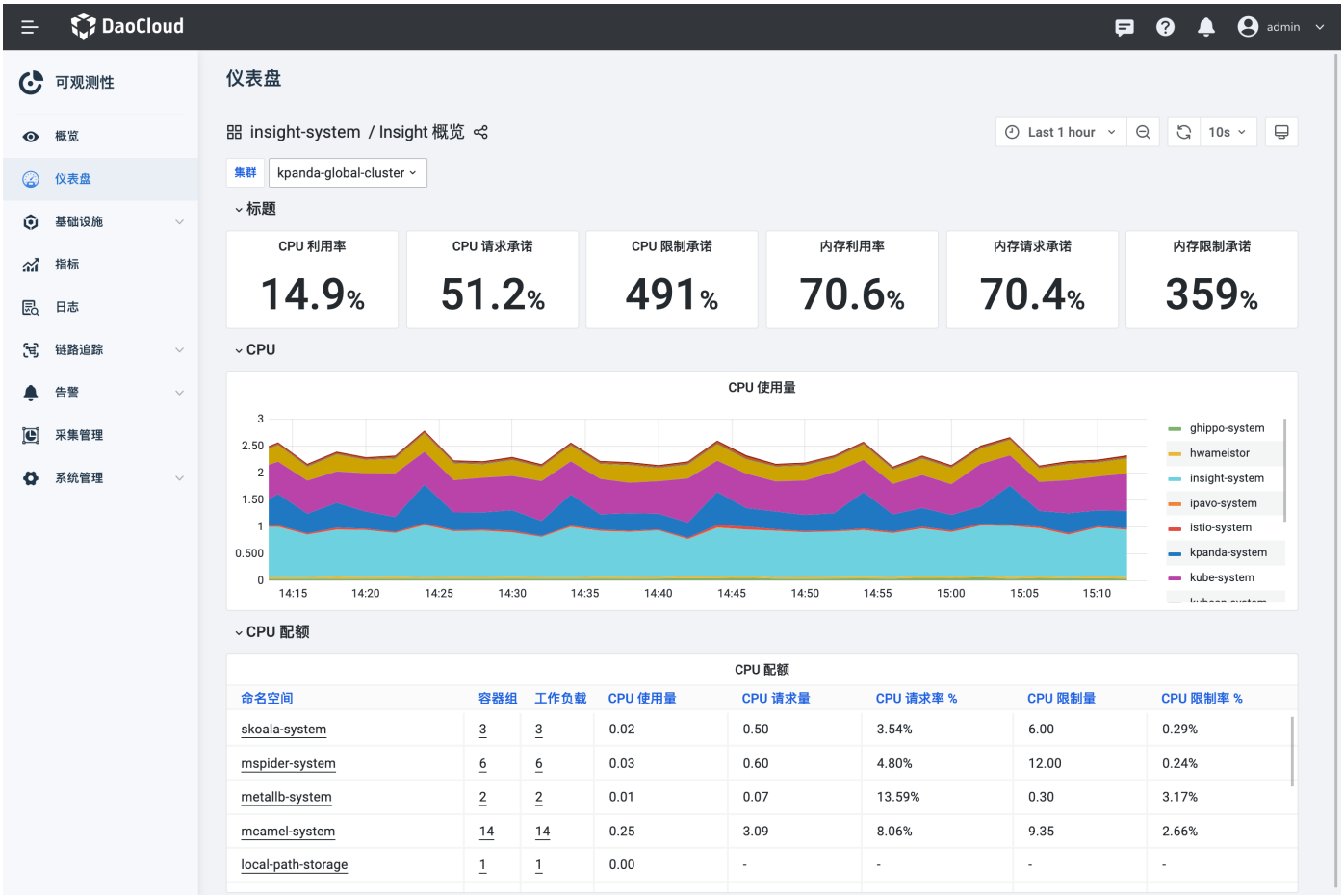
Grafana 是一种开源的数据可视化和监控平台，它提供了丰富的图表和面板，用于实时监控、分析和可视化各种数据源的指标和日志。可观测性 Insight 使用开源 Grafana 提供监控服务，支持从集群、节点、命名空间等多维度查看资源消耗情况，

关于开源 Grafana 的详细信息，请参见 Grafana 官方文档。

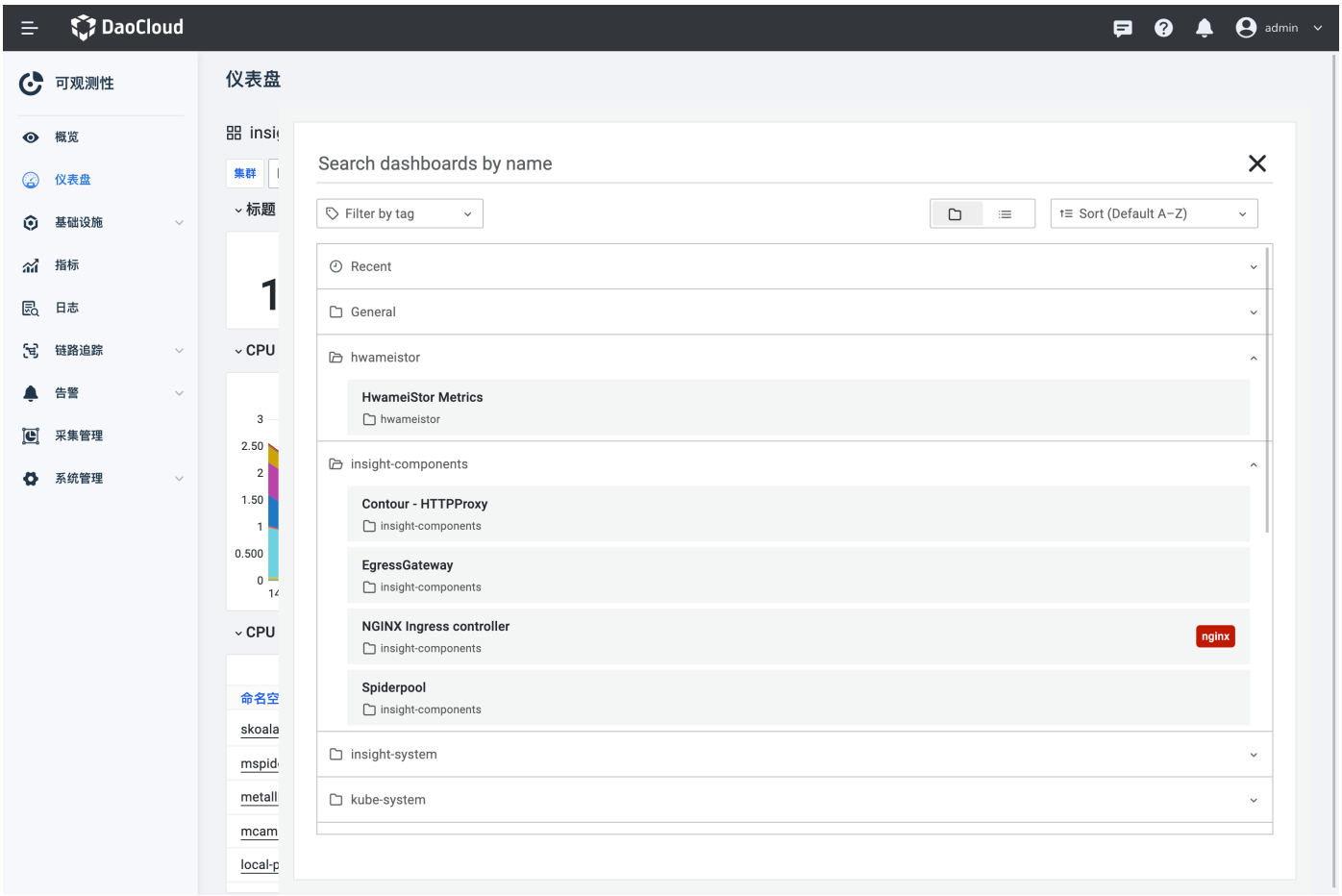
操作步骤

1. 在左侧导航栏选择 仪表盘 。

- 在 **Insight** /概览 仪表盘中，可查看多选集群的资源使用情况，并以命名空间、容器组等多个维度分析了资源使用、网络、存储等情况。
- 点击仪表盘左上侧的下拉框可切换集群。
- 点击仪表盘右下侧可切换查询的时间范围。



2. Insight 精选多个社区推荐仪表盘，可从节点、命名空间、工作负载等多个维度进行监控。点击 **insight-system / Insight** /概览 区域切换仪表盘。



Note

1. 访问 Grafana UI 请参考以管理员身份登录 Grafana。
2. 导入自定义仪表盘请参考导入自定义仪表盘。

访问原生 GRAFANA

Insight 借助 Grafana 提供了丰富的可视化能力，同时保留了访问原生 Grafana 的入口。

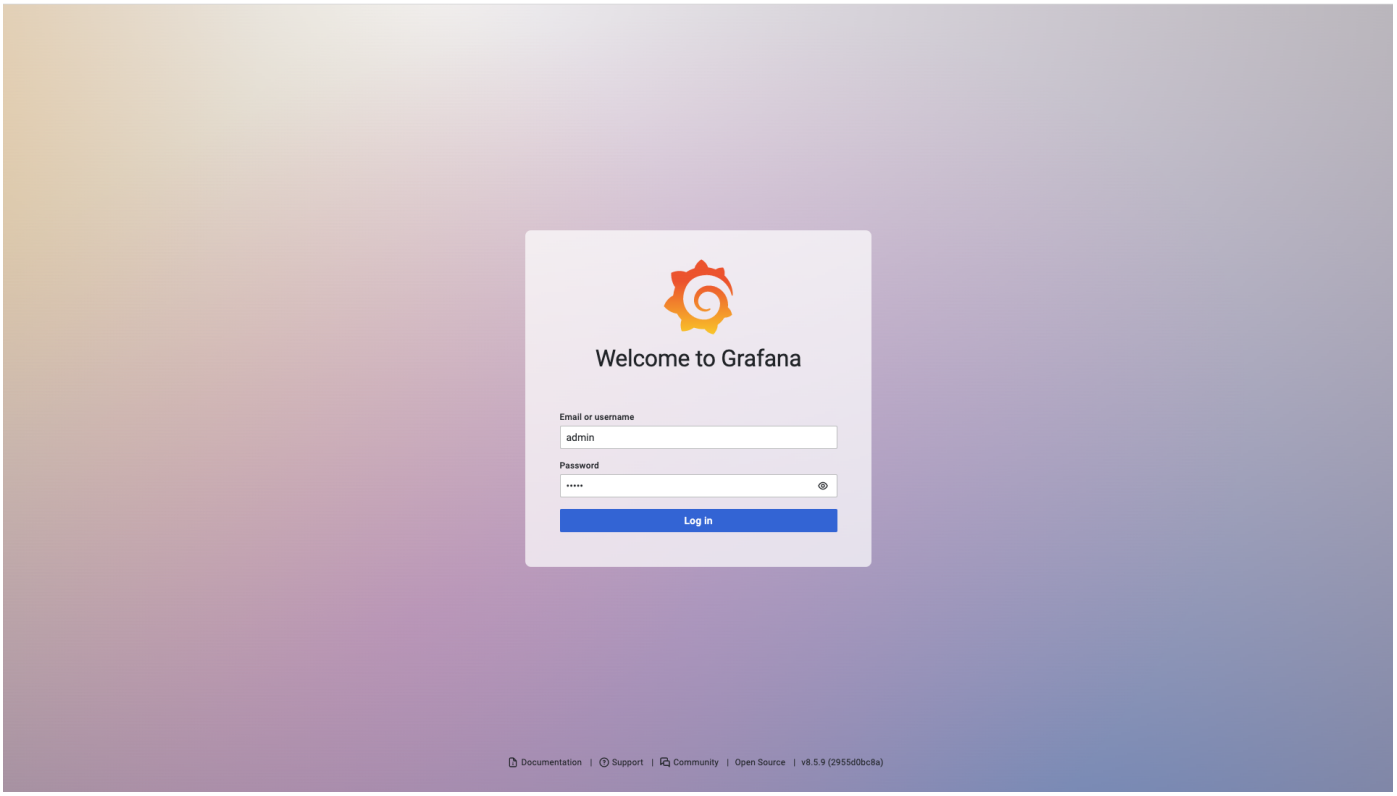
操作步骤

1. 登录浏览器，在浏览器中输入 Grafana 地址。

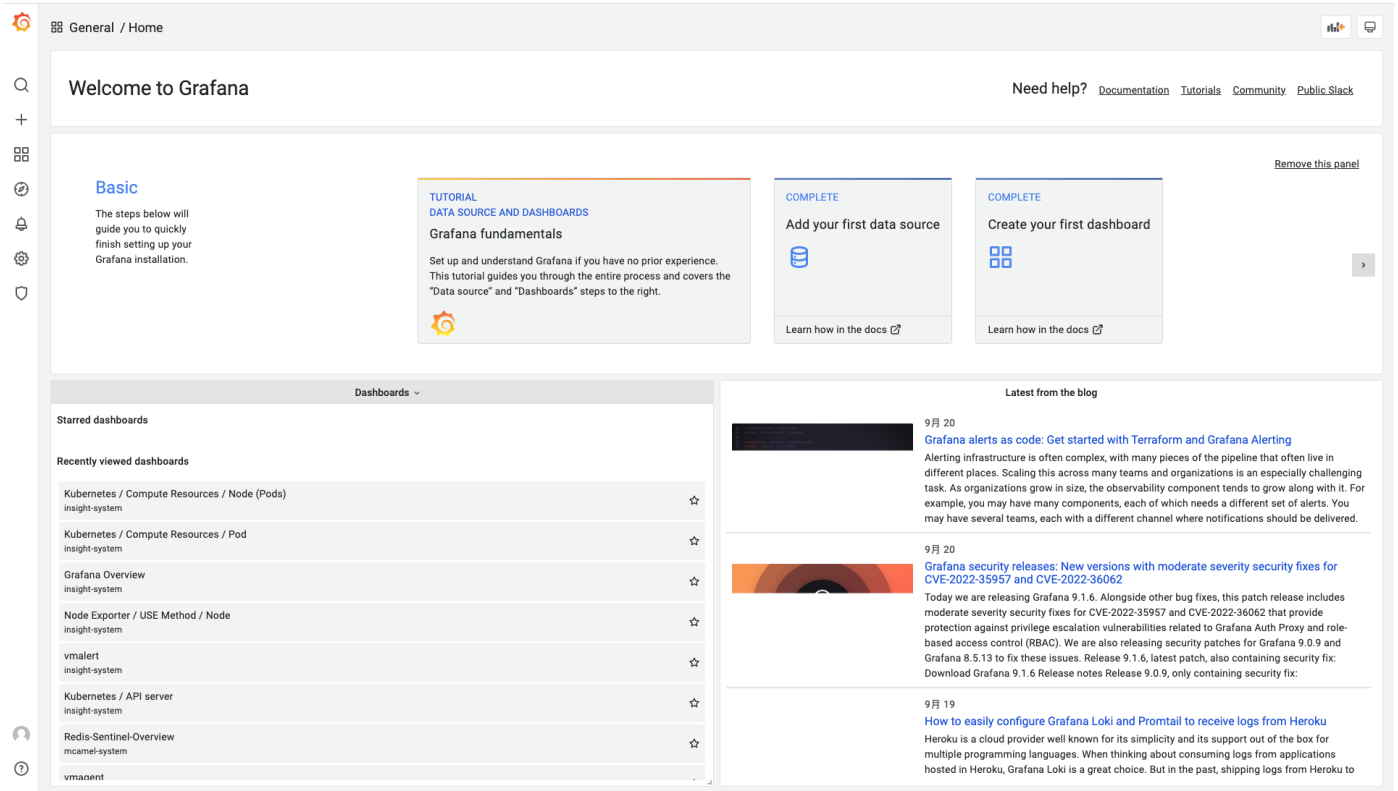
   访问地址： `http://ip:访问端口/ui/insight-grafana/login`

   例如： `http://10.6.10.233:30209/ui/insight-grafana/login`

2. 点击右下角的登录，使用默认用户名、密码（admin/admin）进行登录。



3. 点击 **Log in** 完成登录。

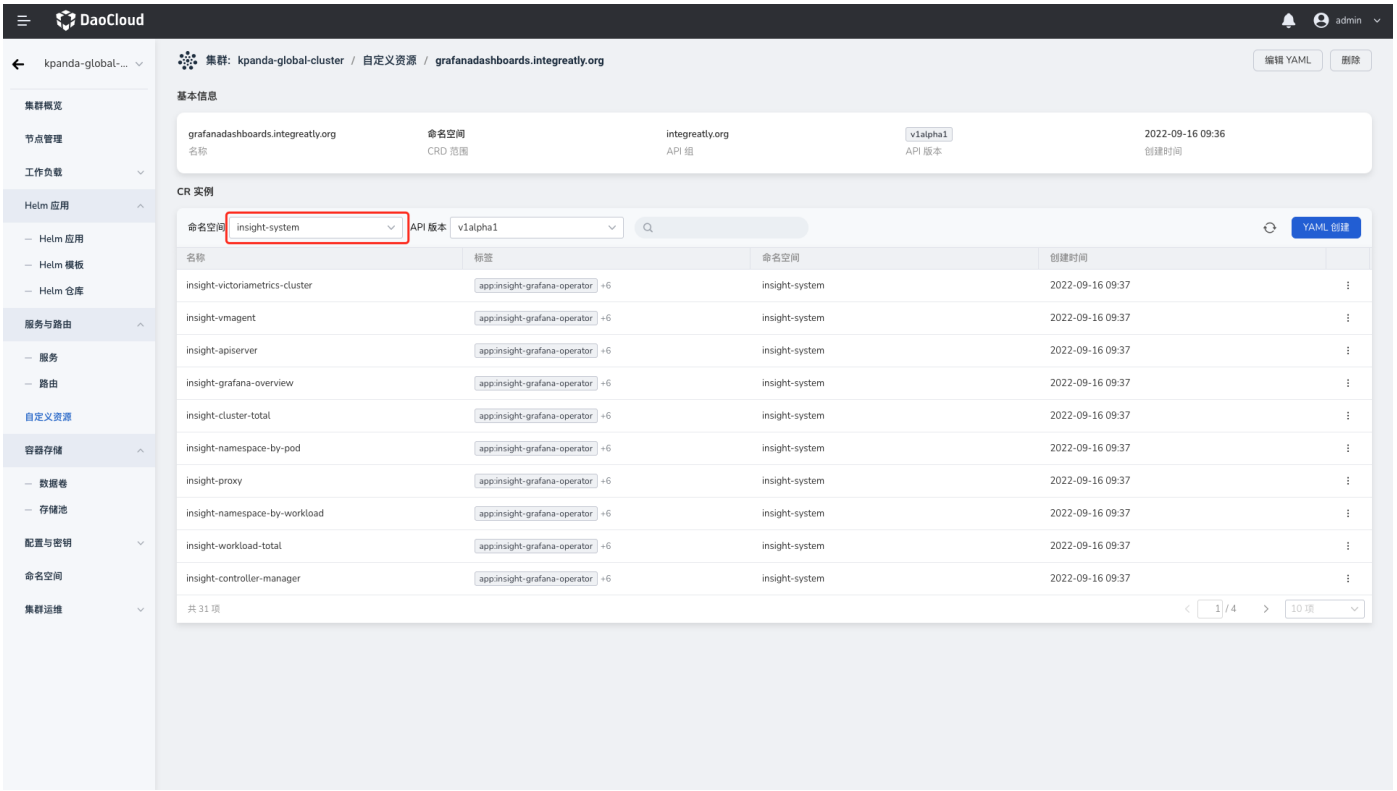

General / Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
COMPLETE
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Kubernetes / Compute Resources / Node (Pods)
insight-system
Kubernetes / Compute Resources / Pod
insight-system
Grafana Overview
insight-system
Node Exporter / USE Method / Node
insight-system
vmalert
insight-system
Kubernetes / API server
insight-system
Redis-Sentinel-Overview
mcamel-system
vmagent
Latest from the blog
9月 20
Grafana alerts as code: Get started with Terraform and Grafana Alerting
Alerting infrastructure is often complex, with many pieces of the pipeline that often live in different places. Scaling this across many teams and organizations is an especially challenging task. As organizations grow in size, the observability component tends to grow along with it. For example, you may have many components, each of which needs a different set of alerts. You may have several teams, each with a different channel where notifications should be delivered.
9月 20
Grafana security releases: New versions with moderate severity security fixes for CVE-2022-35957 and CVE-2022-36062
Today we are releasing Grafana 9.1.6. Alongside other bug fixes, this patch release includes moderate severity security fixes for CVE-2022-35957 and CVE-2022-36062 that provide protection against privilege escalation vulnerabilities related to Grafana Auth Proxy and role-based access control (RBAC). We are also releasing security patches for Grafana 9.0.9 and Grafana 8.5.13 to fix these issues. Release 9.1.6, latest patch, also containing security fix: Download Grafana 9.1.6 Release notes Release 9.0.9, only containing security fix:
9月 19
How to easily configure Grafana Loki and Promtail to receive logs from Heroku
Heroku is a cloud provider well known for its simplicity and its support out of the box for multiple programming languages. When thinking about consuming logs from applications hosted in Heroku, Grafana Loki is a great choice. But in the past, shipping logs from Heroku to

导入自定义仪表盘

通过使用 Grafana CRD，可以将仪表板的管理和部署纳入到 Kubernetes 的生命周期管理中，实现仪表板的版本控制、自动化部署和集群级的管理。本页介绍如何通过 CRD 和 UI 界面导入自定义的仪表盘。

操作步骤

1. 登录 d.run 平台，进入 容器管理 ，在集群列表中选择 **kpanda-global-cluster** 。
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。



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。
- **label** ： 必填， **operator.insight.io/managed-by: insight** 。

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
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。

> 自定义设计仪表盘，请参考添加仪表盘面板。

基础设施

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。

操作步骤

1. 进入 可观测性 产品模块。
2. 在左边导航栏选择 基础设施 > 集群 。在该页面可查看以下信息：

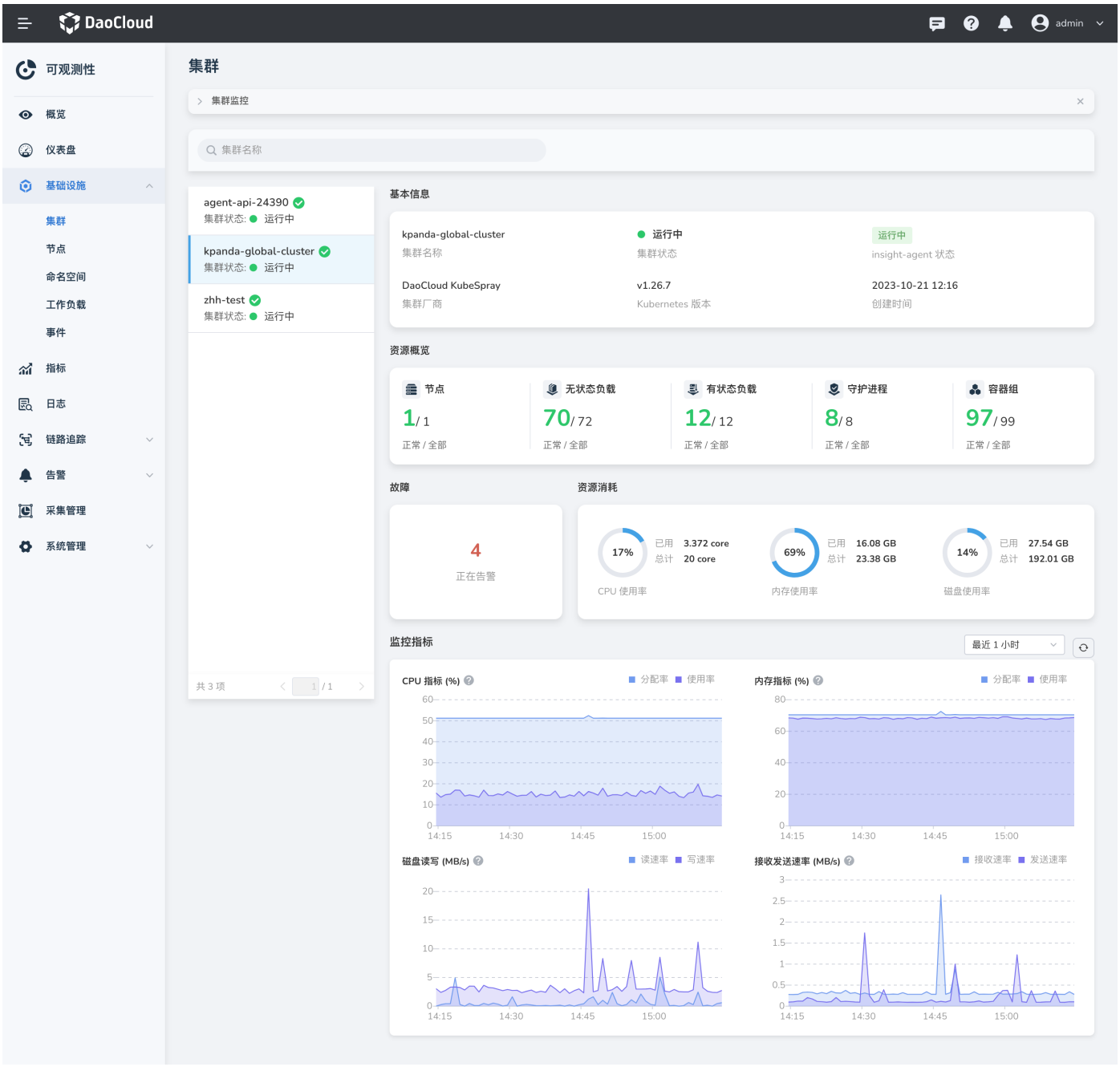
a. 集群列表 ：已安装 insight-agent 的集群列表，可点击目标集查看对应详细信息；

b. 资源概览 ：多选集群中的节点、工作负载的正常和全部的数量统计；

c. 故障 ：统计当前集群产生的告警数量；

d. 资源消耗 ：所选集群的 CPU、内存、磁盘的实际使用量和总量；

e. 指标说明 ：所选集群的 CPU、内存、磁盘读写、网络接收发送的变化趋势。



指标说明

指标名	说明
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。

操作步骤

1. 进入 可观测性 产品模块。
2. 在左边导航栏选择 基础设施 > 节点 。在该页面可查看以下信息：

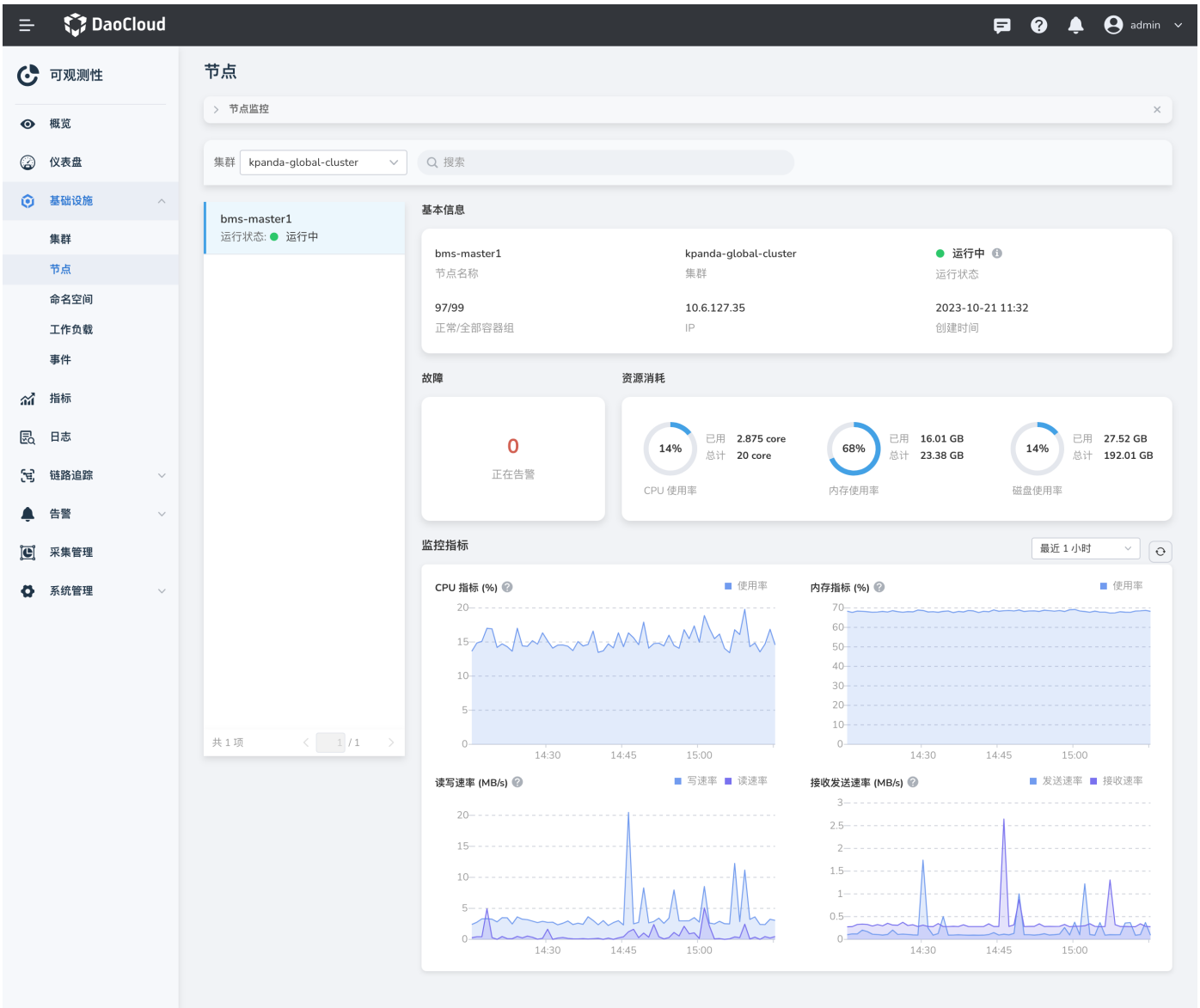
a. 集群切换 ：切换顶部的下拉框可切换集群；

b. 节点列表 ：所选集群中的节点列表，单击切换节点。

c. 故障 ：统计当前集群产生的告警数量；

d. 资源消耗 ：所选节点的 CPU、内存、磁盘的实际使用量和总量；

e. 指标说明 ：所选节点的 CPU、内存、磁盘读写、网络接收发送的变化趋势。



命名空间监控

以命名空间为维度，快速查询命名空间内的资源消耗和变化趋势。

前提条件

集群中已安装 insight-agent 且应用处于 运行中 状态。

操作步骤

1. 进入 可观测性 产品模块。
2. 在左边导航栏选择 基础设施 > 命名空间 。在该页面可查看以下信息：

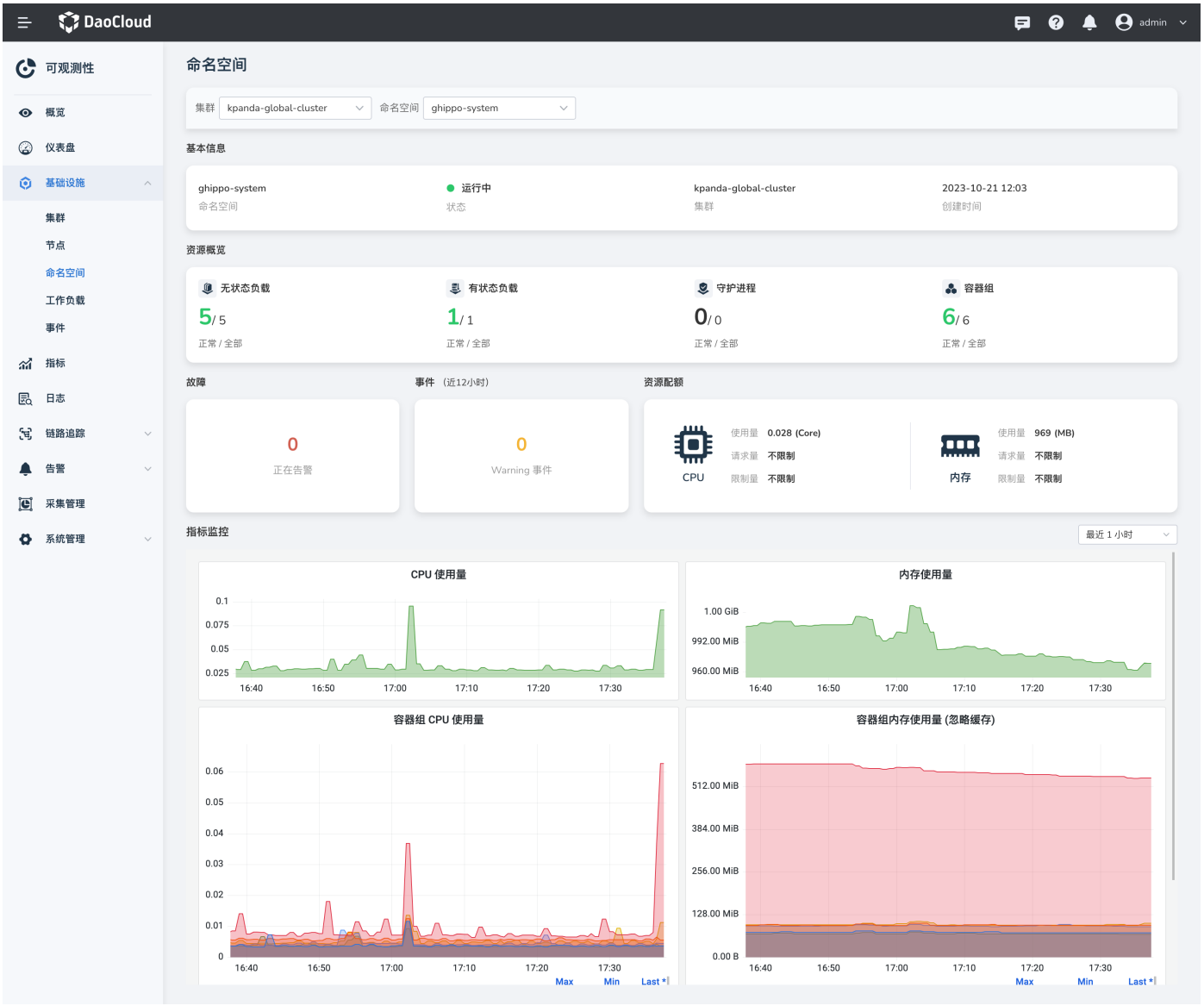
a. 切换命名空间 ：在顶部切换集群或命名空间；

b. 资源概览 ：统计所选命名空间下的正常和全部工作负载的数量；

c. 故障 ：统计所选命名空间下产生的告警数量；

d. 事件 ：统计所选命名空间下 24 小时内 Warning 级别的事件数量；

e. 资源消耗 ：统计所选命名空间下容器组的 CPU、内存使用量之和 及 CPU、内存配额情况。



指标说明

指标名	说明
CPU 使用量	所选命名空间中容器组的 CPU 使用量之和
内存使用量	所选命名空间中容器组的内存使用量之和
容器组 CPU 使用量	命名空间中各容器组的 CPU 使用量
容器组内存使用量	命名空间中各容器组的内存使用量

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。

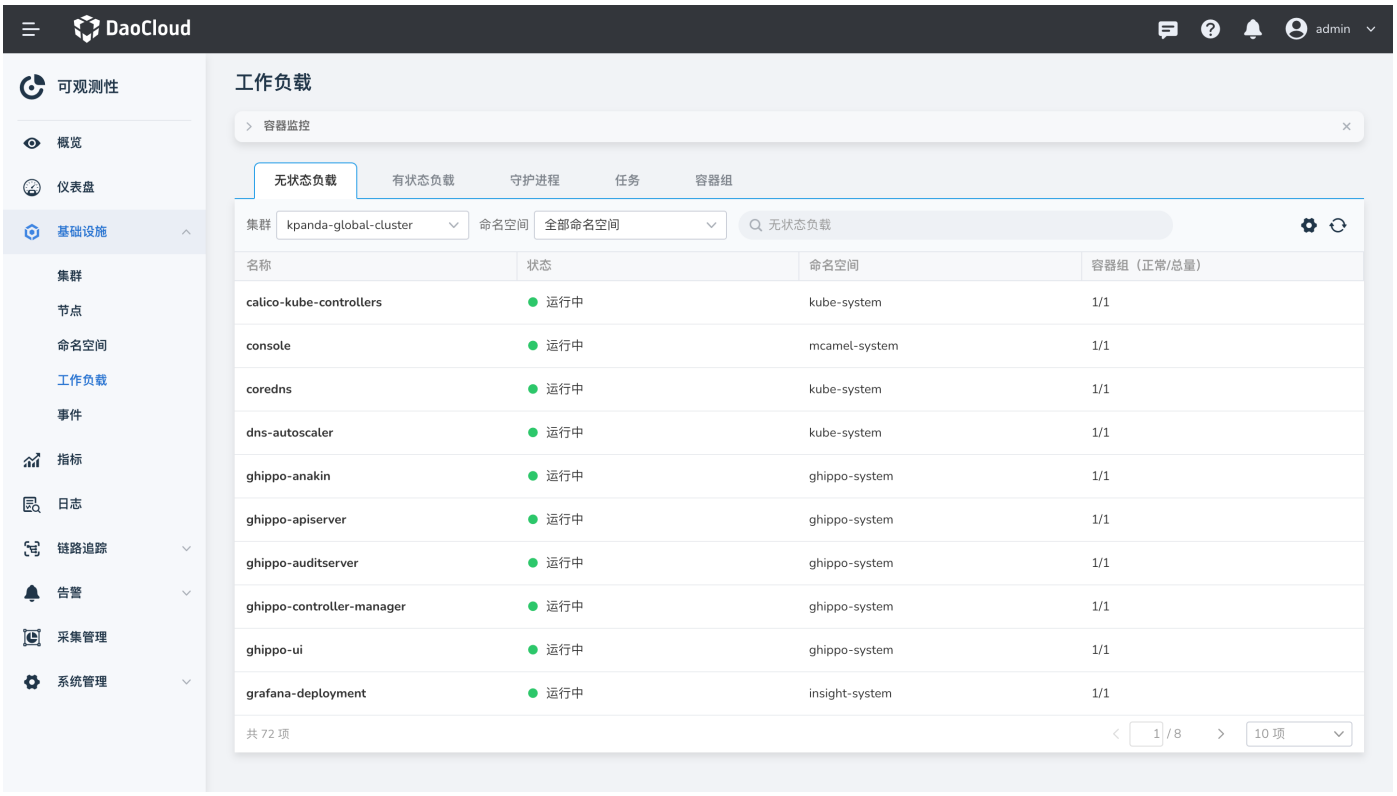
操作步骤

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 基础设施 -> 容器 。

3. 切换顶部 Tab，查看不同类型工作负载的数据。

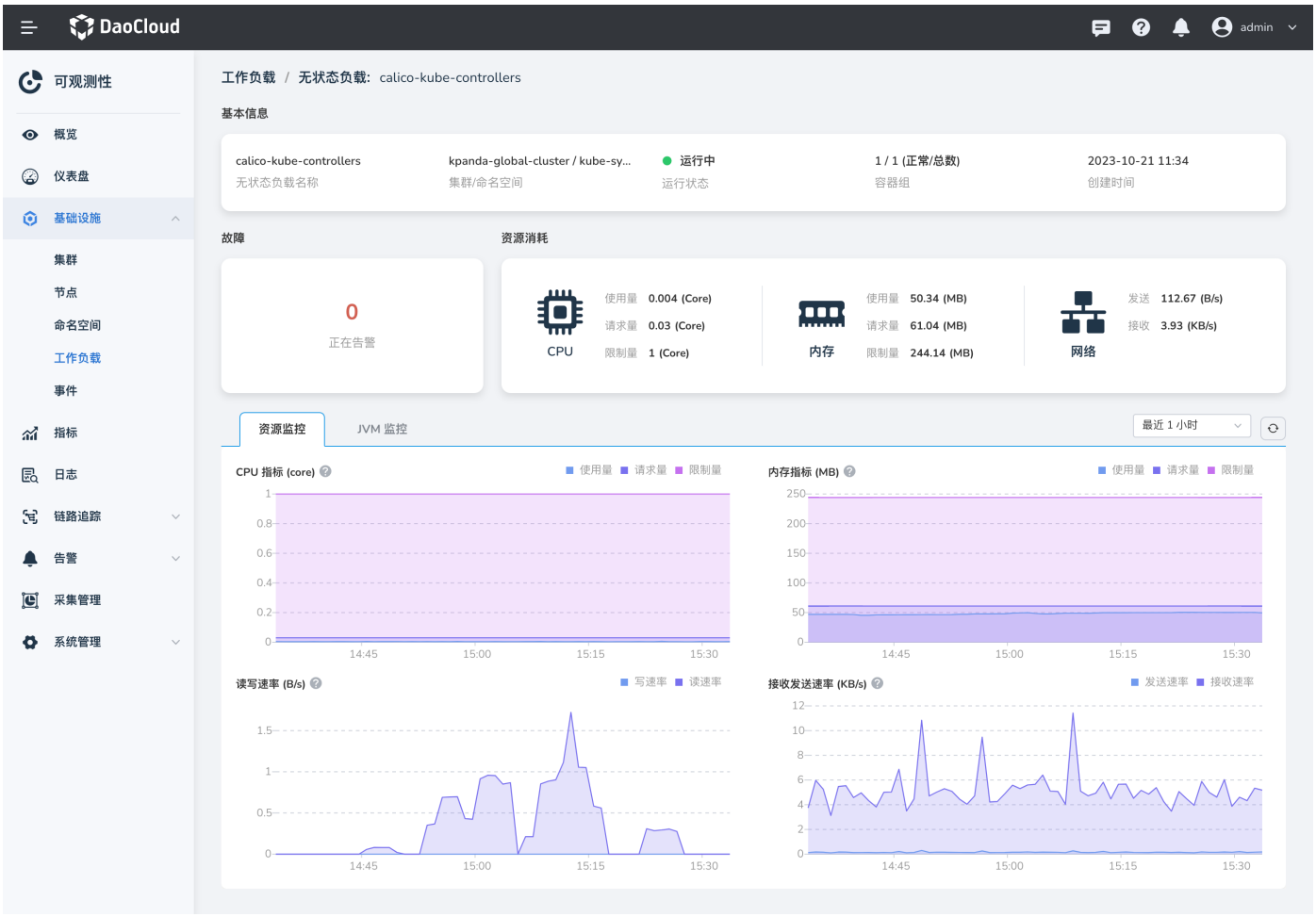


4. 点击目标工作负载名称查看详情。

a. 故障：在故障卡片中统计该工作负载当前正在告警的总数。

b. 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。

c. 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。



指标说明

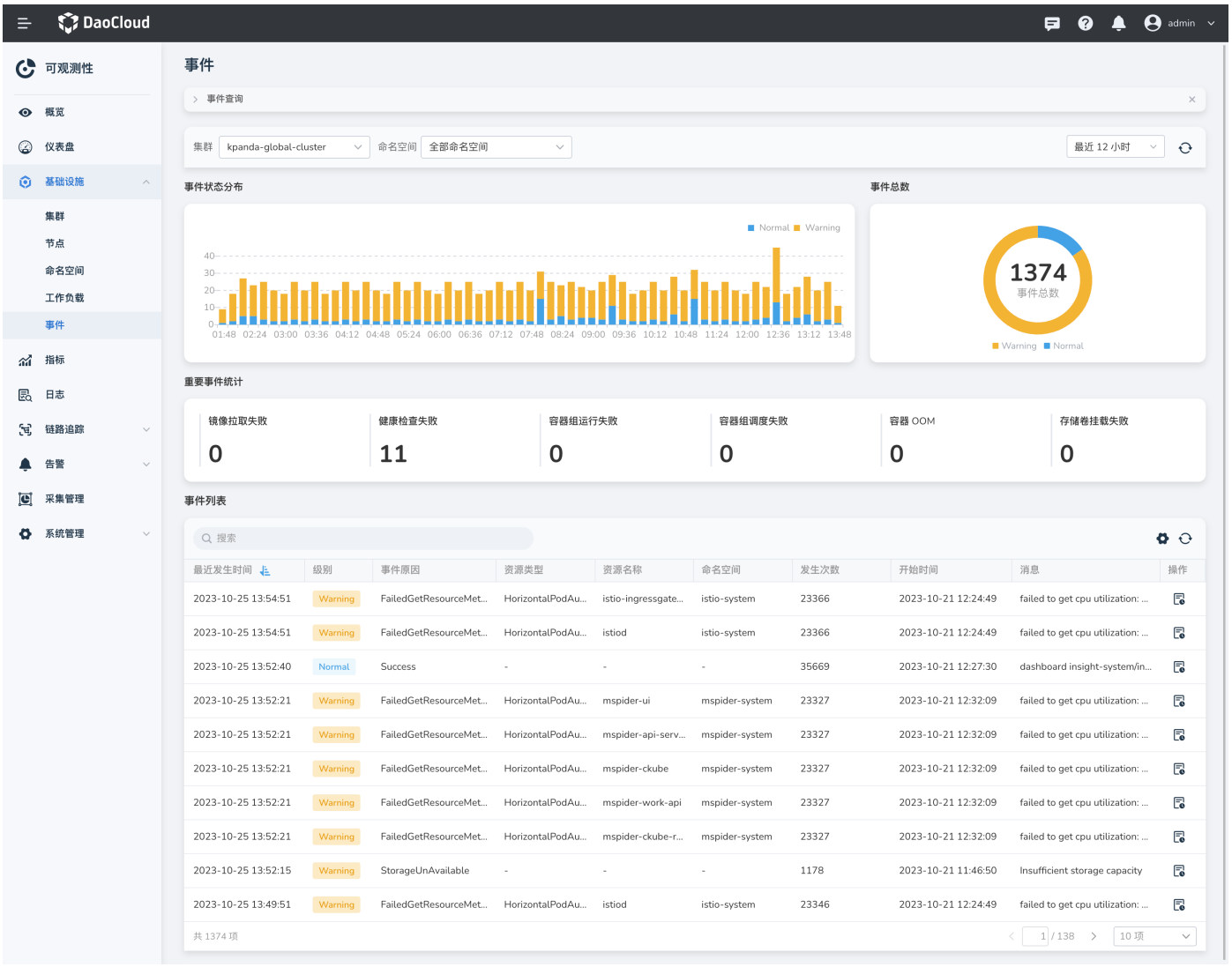
指标名称	说明
CPU 使用量	工作负载下容器组的 CPU 使用量之和。
CPU 请求量	工作负载下容器组的 CPU 请求量之和。
CPU 限制量	工作负载下容器组的 CPU 限制量之和。
内存使用量	工作负载下容器组的内存使用量之和。
内存请求量	工作负载下容器组的内存使用量之和。
内存限制量	工作负载下容器组的内存限制量之和。
磁盘读写速率	指定时间范围内磁盘每秒连续读取和写入的总和，表示磁盘每秒读取和写入操作数的性能度量。
网络发送接收速率	指定时间范围内，按工作负载统计的网络流量的流入、流出速率。

事件查询

d.run Insight 支持按集群、命名空间查询事件，并提供了事件状态分布图，对重要事件进行统计。

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 基础设置 > 事件 。



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

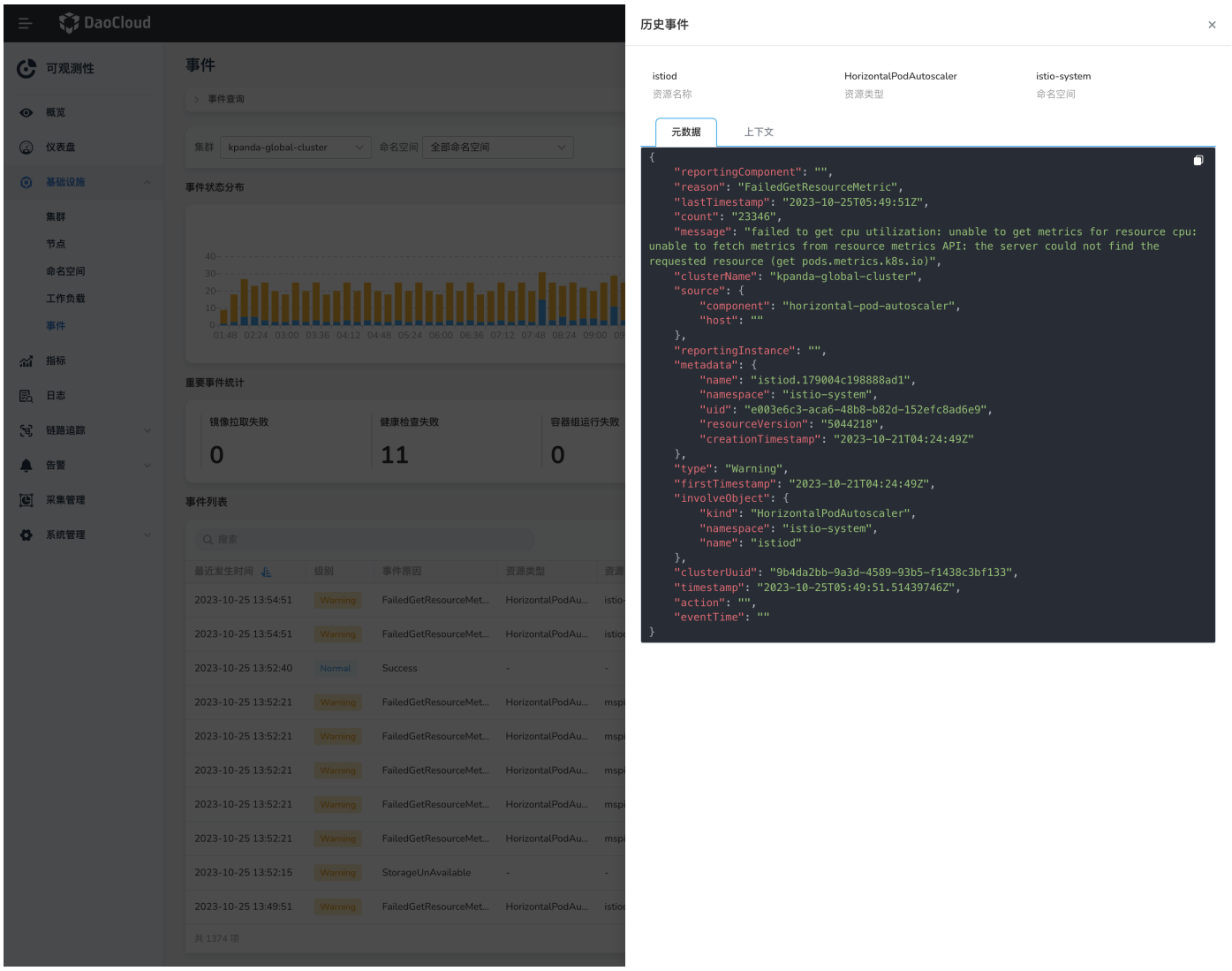
事件列表以时间为轴，以流水的形式展示发生的事件。您可以根据「最近发生时间」和「级别」进行排序。

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。

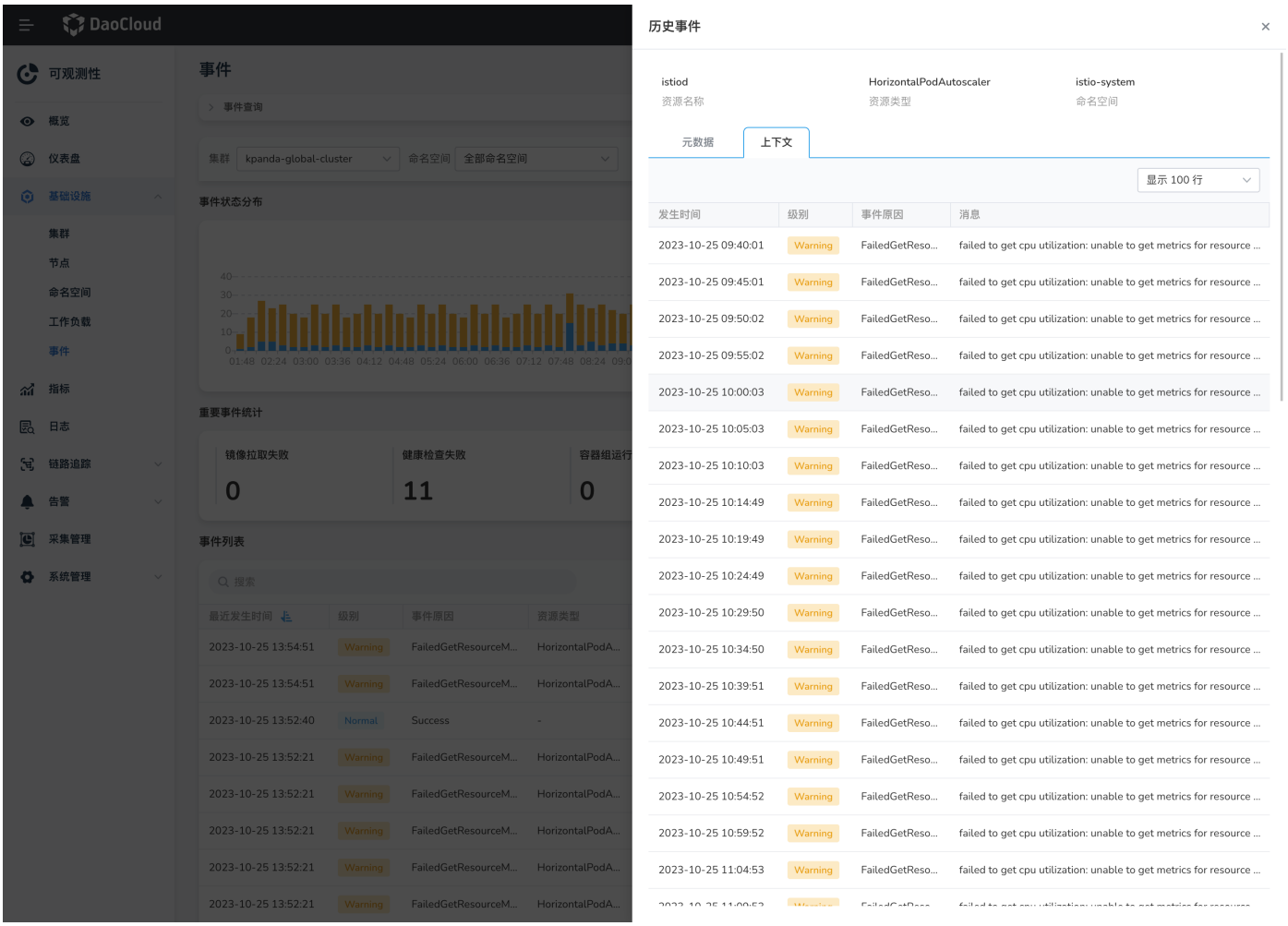
在需要的时候，您还可以点击刷新图标来更新当前的事件列表。

其他操作

1. 在事件列表中操作列的图标，可查看某一事件的元数据信息。



2. 点击顶部页签的 上下文 可查看该事件对应资源的历史事件记录。



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。

拨测

拨测（Probe）指的是基于黑盒监控，定期通过 HTTP、TCP 等方式对目标进行连通性测试，快速发现正在发生的故障。

Insight 基于 Prometheus Blackbox Exporter 工具通过 HTTP、HTTPS、DNS、TCP 和 ICMP 等协议，对网络进行探测并返回探测结果以便了解网络状态。

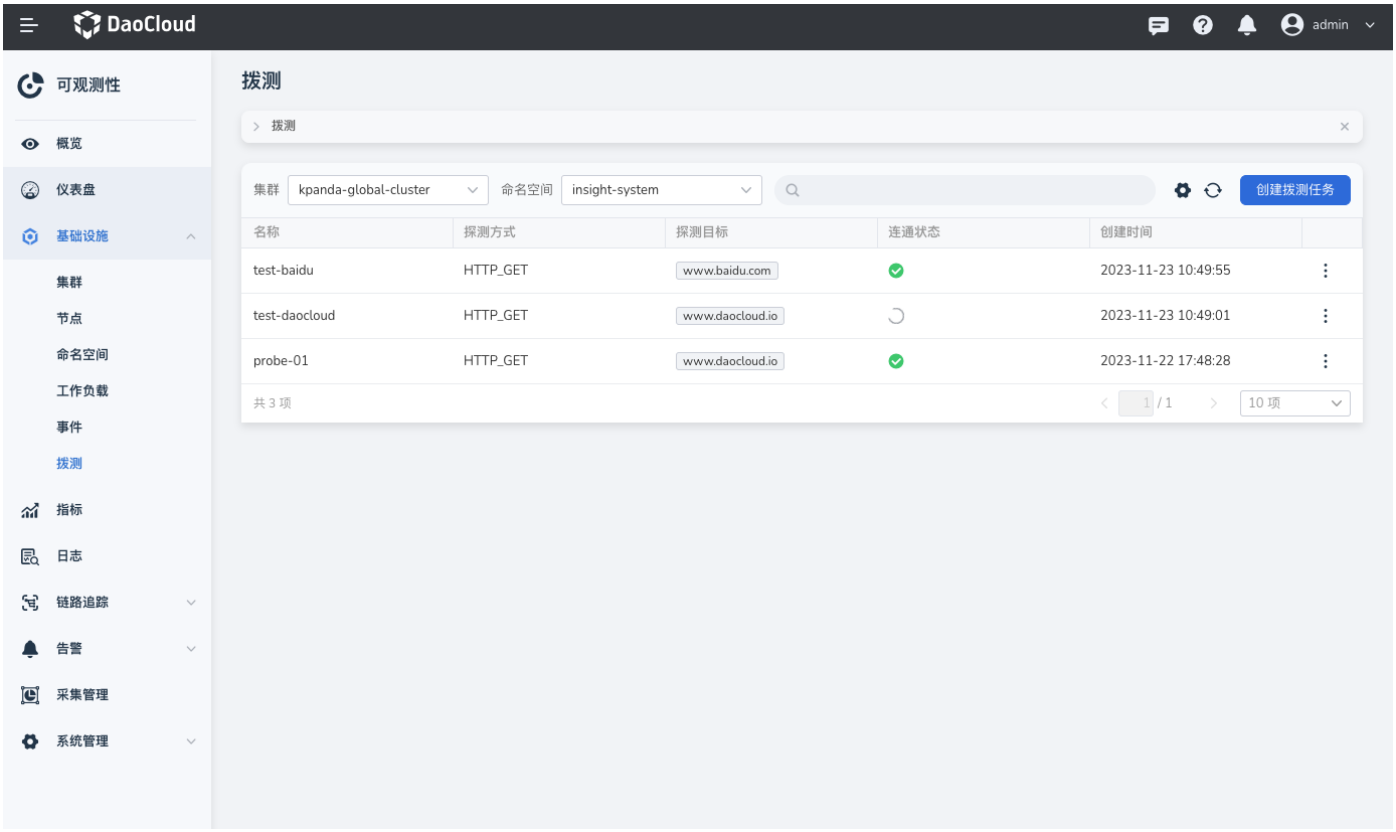
前提条件

集群中已安装 insight-agent 且应用处于 运行中 状态。

查看拨测任务

1. 进入 可观测性 产品模块；
2. 在左边导航栏选择 基础设施 -> 拨测 。

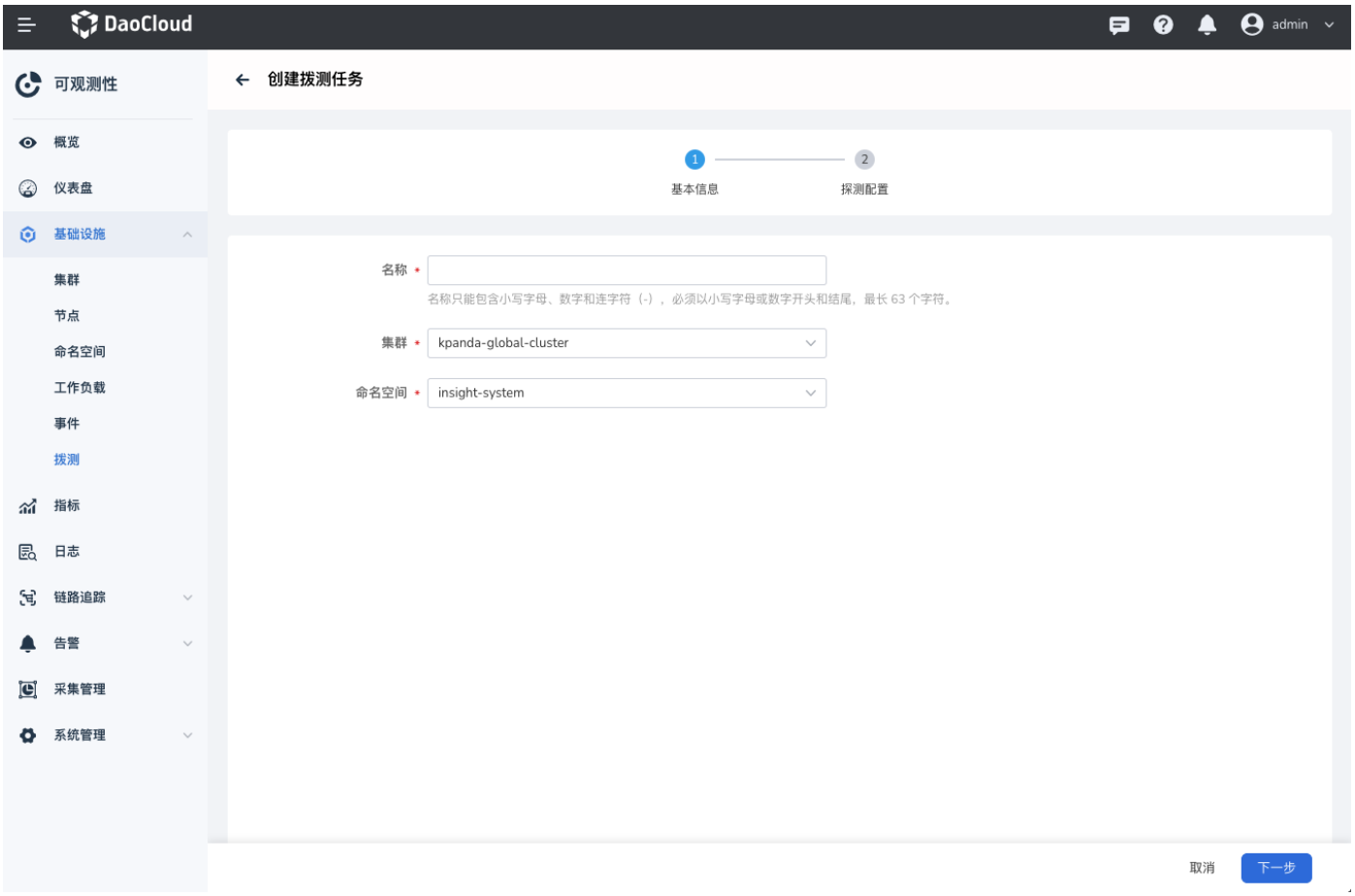
- 点击表格中的集群或命名空间下拉框，可切换集群和命名空间
- 你可以点击右侧的 ⚙ 修改显示的列，默认为拨测名称、探测方式、探测目标、连通状态、创建时间
- 连通状态有 3 种：
- 正常：Probe 成功连接到了目标，目标返回了预期的响应
- 异常：Probe 无法连接到目标，或目标没有返回预期的响应
- Pending：Probe 正在尝试连接目标
- 你可以在 🔍 搜索框中键入名称，模糊搜索某些拨测任务



创建拨测任务

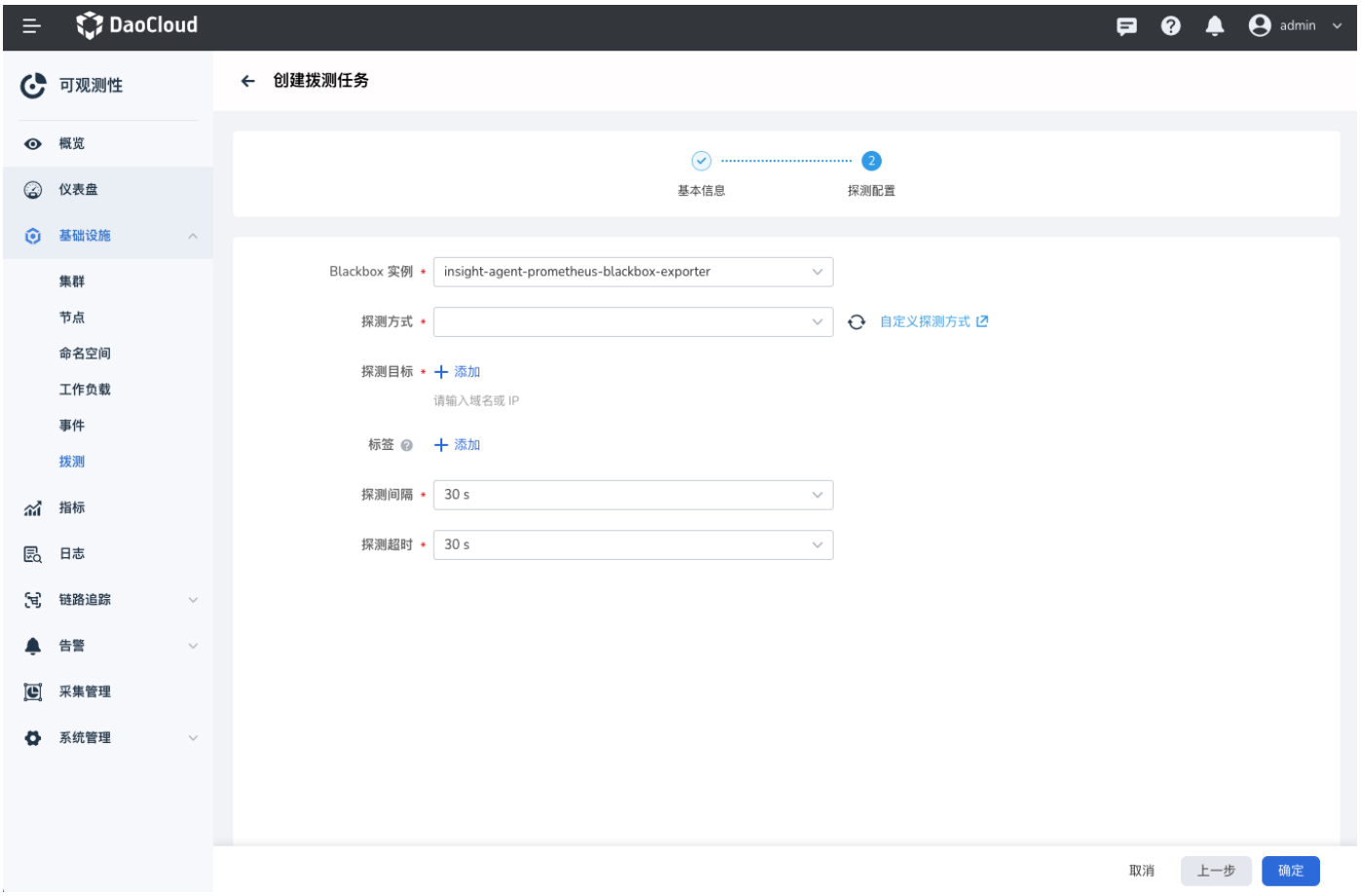
1. 点击 创建拨测任务 。
2. 填写基本信息后点击 下一步

- 集群：选择需要拨测的集群
- 命名空间：拨测所在的命名空间



3. 配置探测参数。

- Blackbox 实例：选择负责探测的 blackbox 实例
- 探测方式：
- HTTP：通过发送 HTTP 或 HTTPS 请求到目标 URL，检测其连通性和响应时间，这可以用于监测网站或 Web 应用的可用性和性能
- TCP：通过建立到目标主机和端口的 TCP 连接，检测其连通性和响应时间。这可以用于监测基于 TCP 的服务，如 Web 服务器、数据库服务器等
- 其他：支持通过配置 ConfigMap 自定义探测方式，可参考自定义拨测方式
- 探测目标：探测的目标地址，支持域名或 IP 地址等
- 标签：自定义标签，该标签会自动添加到 Prometheus 的 Label 中
- 探测间隔：探测间隔时间
- 探测超时：探测目标时的最长等待时间

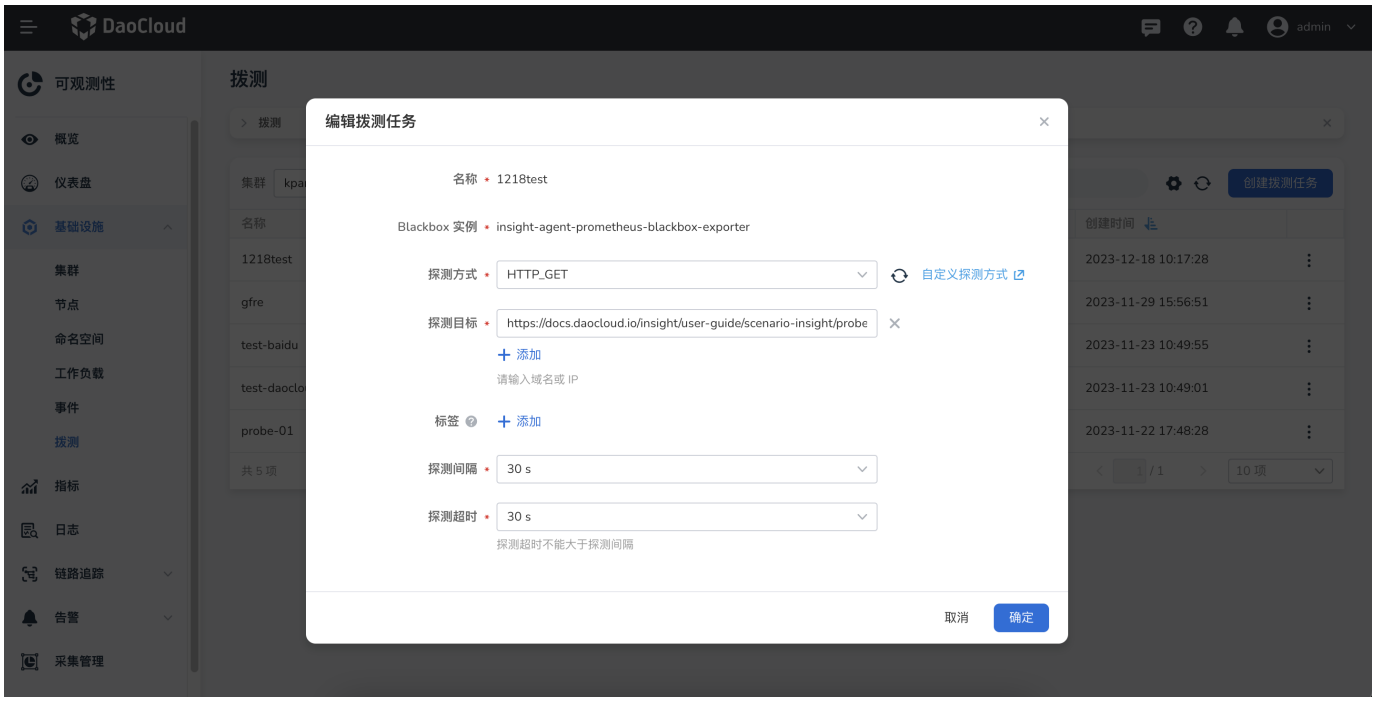


4. 配置完成后，点击 确定 即可完成创建。

> **Warning**
>
> 拨测任务创建完成后，需要大概 3 分钟的时间来同步配置。在此期间，不会进行探测，无法查看探测结果。

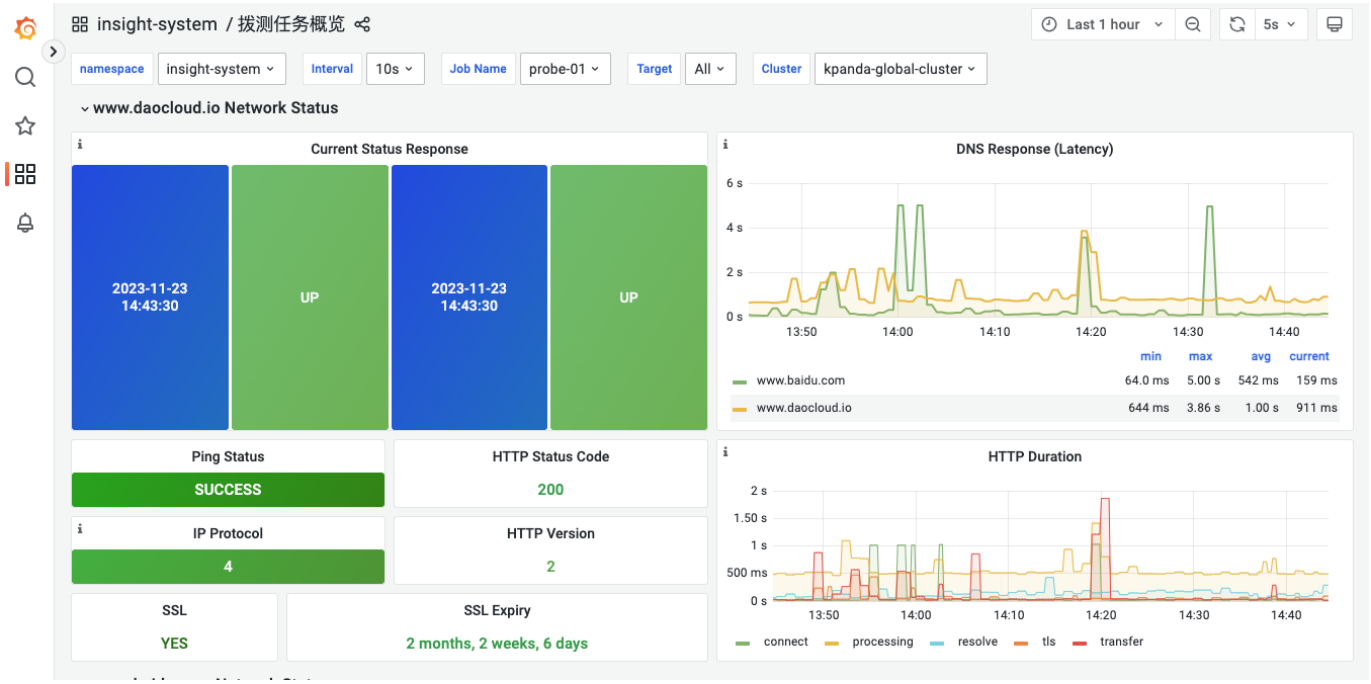
编辑拨测任务

点击列表右侧的 ┆ -> 编辑 ，完成编辑后点击 确定 。



查看监控面板

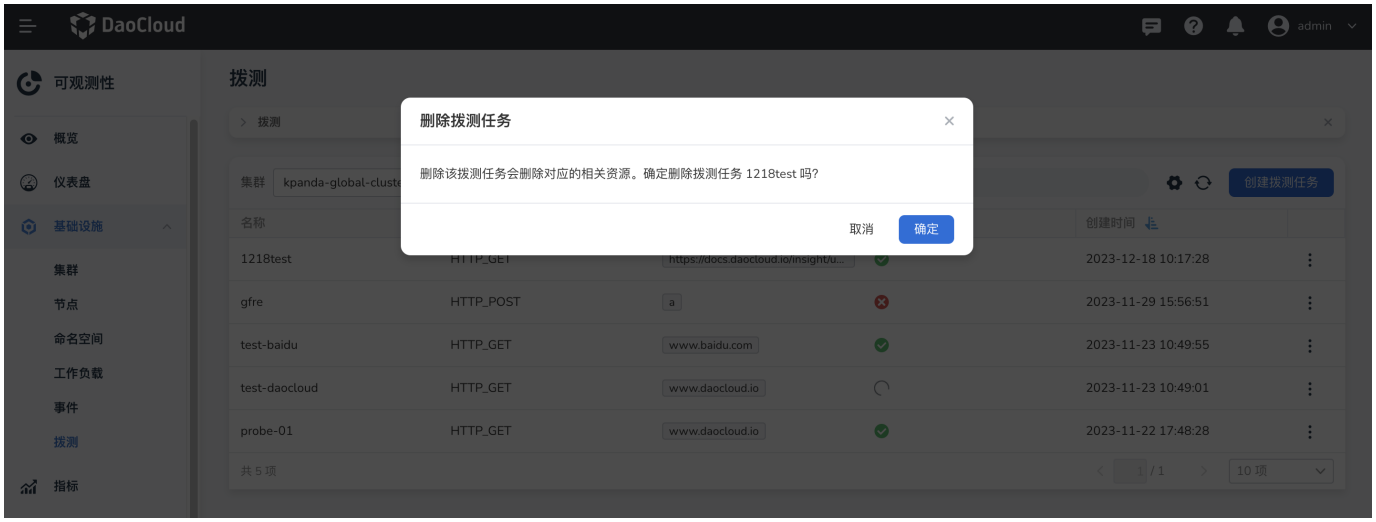
点击列表右侧的 ┇ -> 查看监控面板 ，跳转到 Grafana 拨测任务概览页面，以图表方式显示针对网络状况的探测结果。



指标名称	描述
Current Status Response	表示 HTTP 探测请求的响应状态码。
Ping Status	表示探测请求是否成功。1 表示探测请求成功，0 表示探测请求失败。
IP Protocol	表示探测请求使用的 IP 协议版本。
SSL Expiry	表示 SSL/TLS 证书的最早到期时间。
DNS Response (Latency)	表示整个探测过程的持续时间，单位是秒。
HTTP Duration	表示从发送请求到接收到完整响应的整个过程的时间。

删除拨测任务

点击列表右侧的 ┆ -> 删除 ，确认无误后点击 确定 。



> **Caution**
>
> 删除操作不可恢复，请谨慎操作。

指标查询

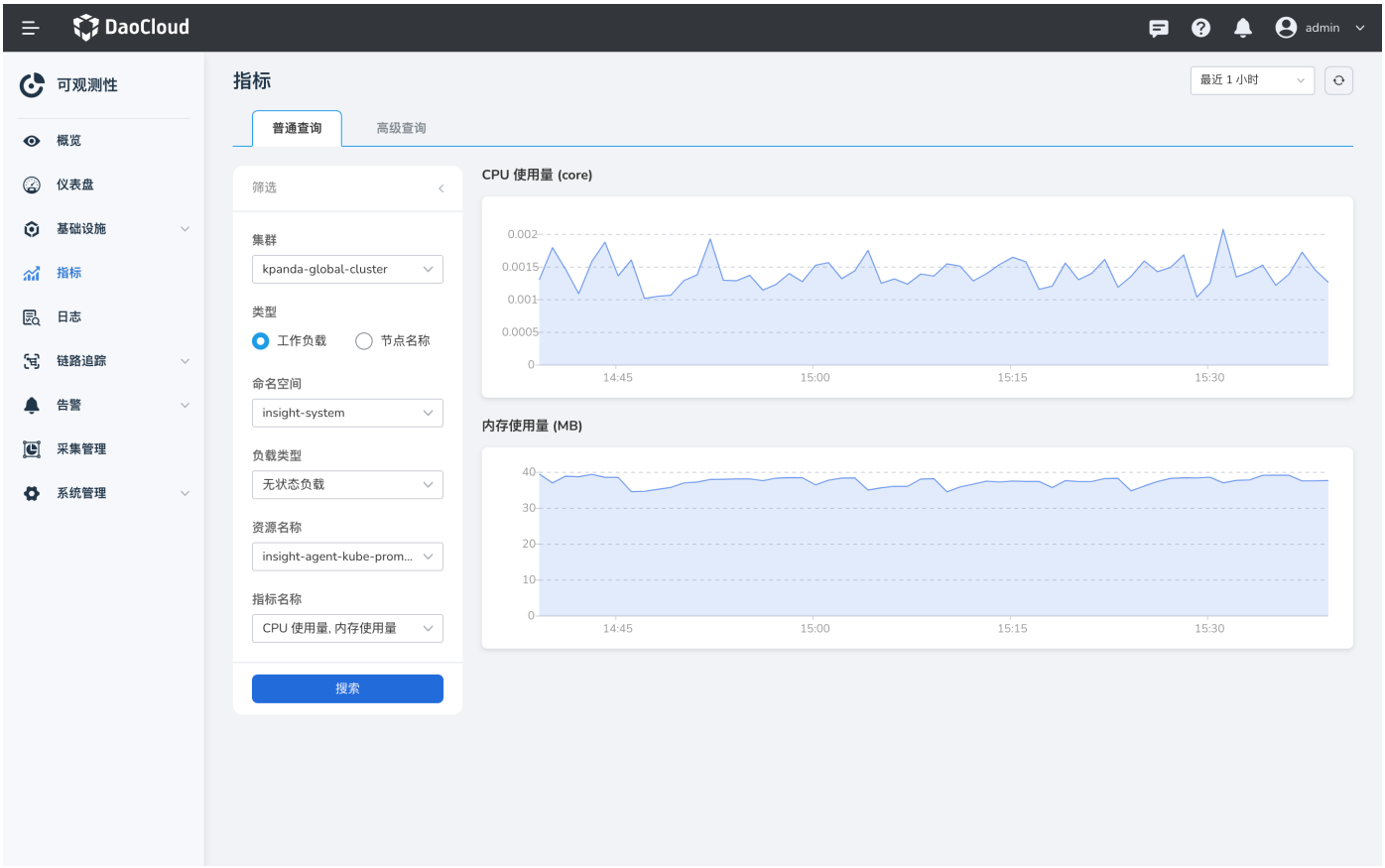
指标查询支持查询容器各资源的指标数据，可查看监控指标的趋势变化。同时，高级查询支持原生 PromQL 语句进行指标查询。

前提条件

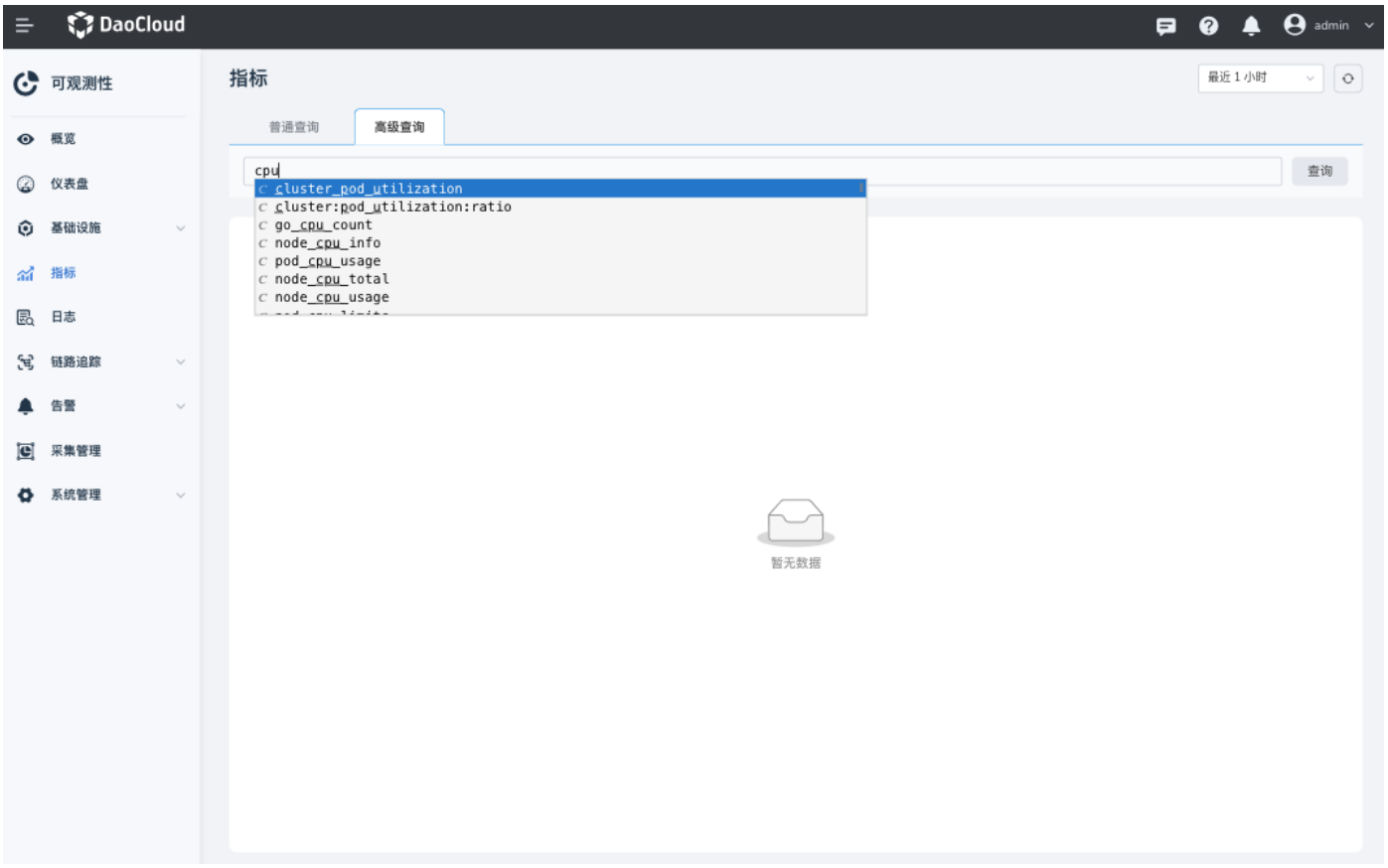
- 集群中已安装 insight-agent 且应用处于 运行中 状态。

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 指标 。
3. 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。
4. 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。



5. 点击 高级查询 页签通过原生的 PromQL 查询。



Note

参阅 PromQL 语法。

日志查询

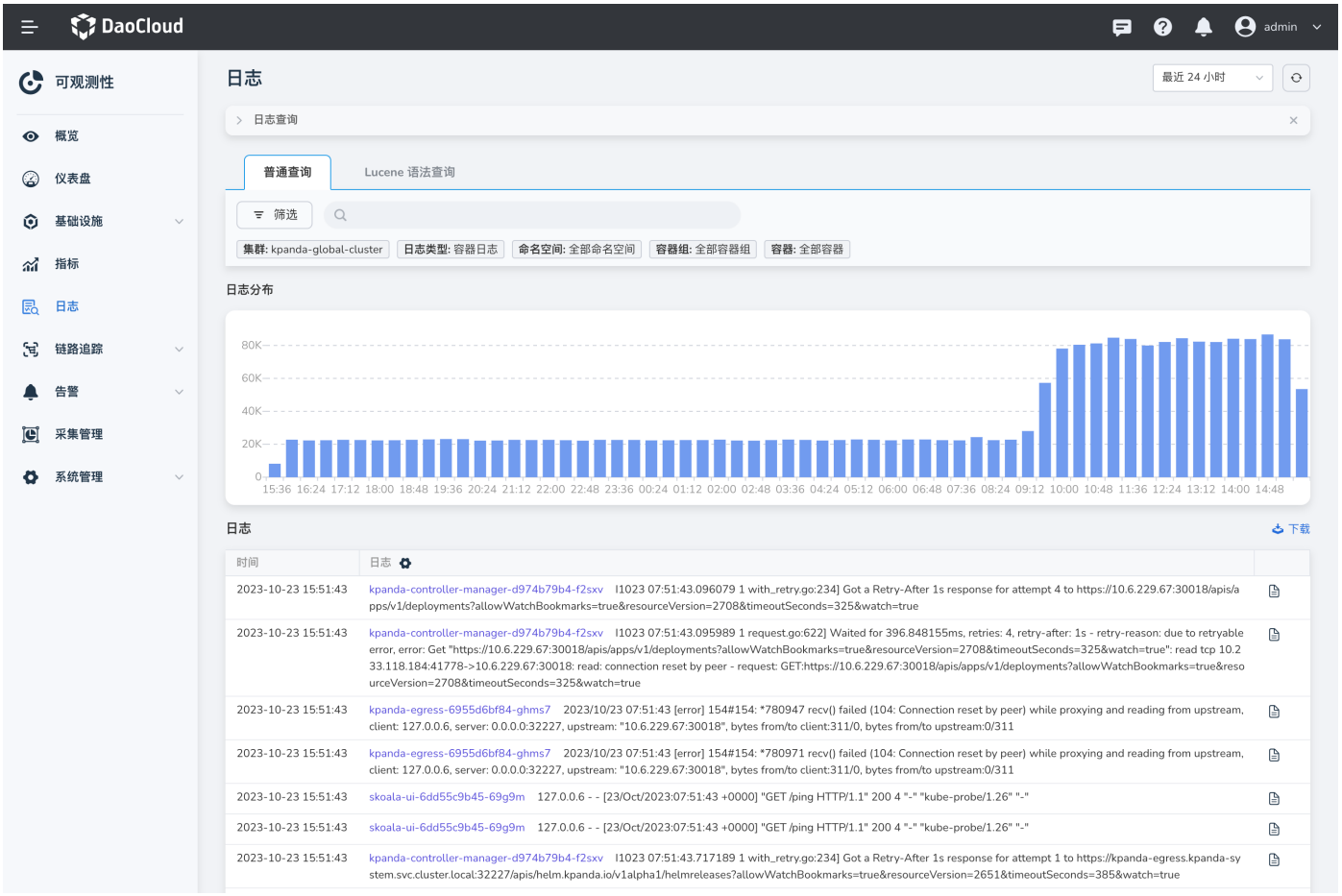
Insight 默认采集节点日志、容器日志以及 kubernetes 审计日志。在日志查询页面中，可查询登录账号权限内的标准输出 (stdout) 日志，包括节点日志、产品日志、Kubenetes 审计日志等，快速在大量日志中查询到所需的日志，同时结合日志的来源信息和上下文原始数据辅助定位问题。

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 日志 。

• 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；



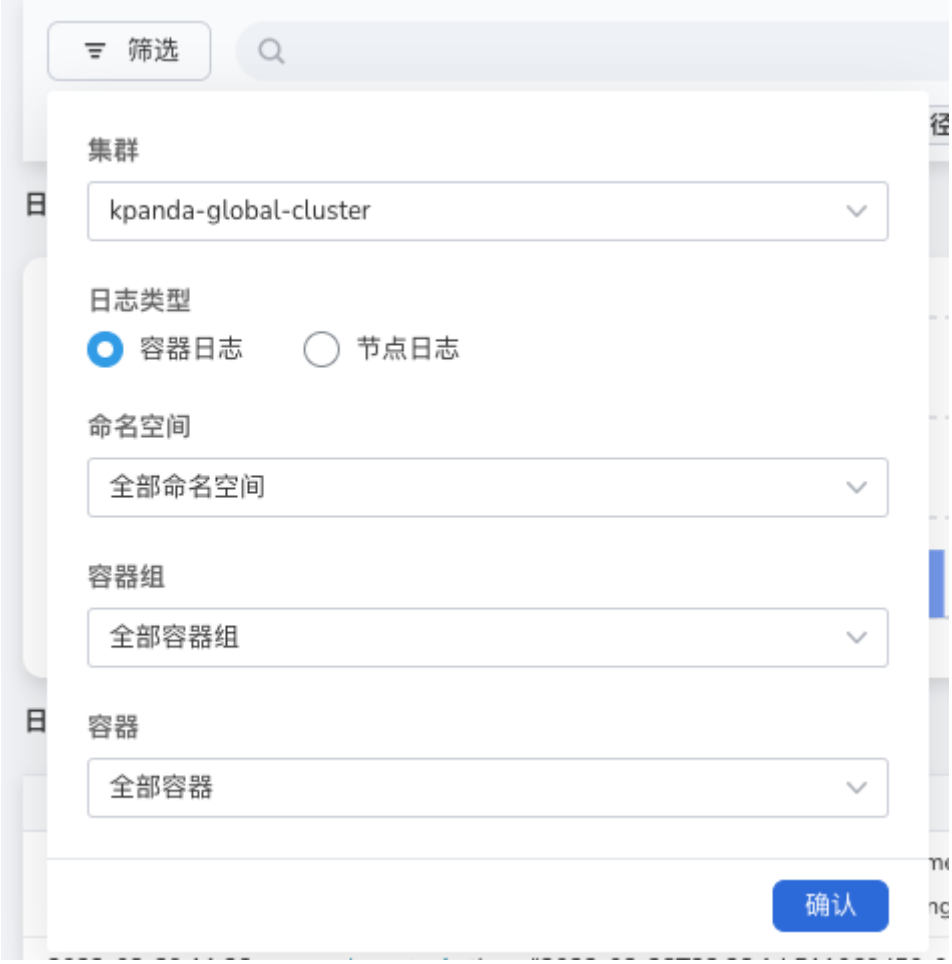
3. 顶部 Tab 默认进入 普通查询 。

a. 点击 筛选 展开过滤面板，可切换日志搜索条件和类型。

b. 日志类型：

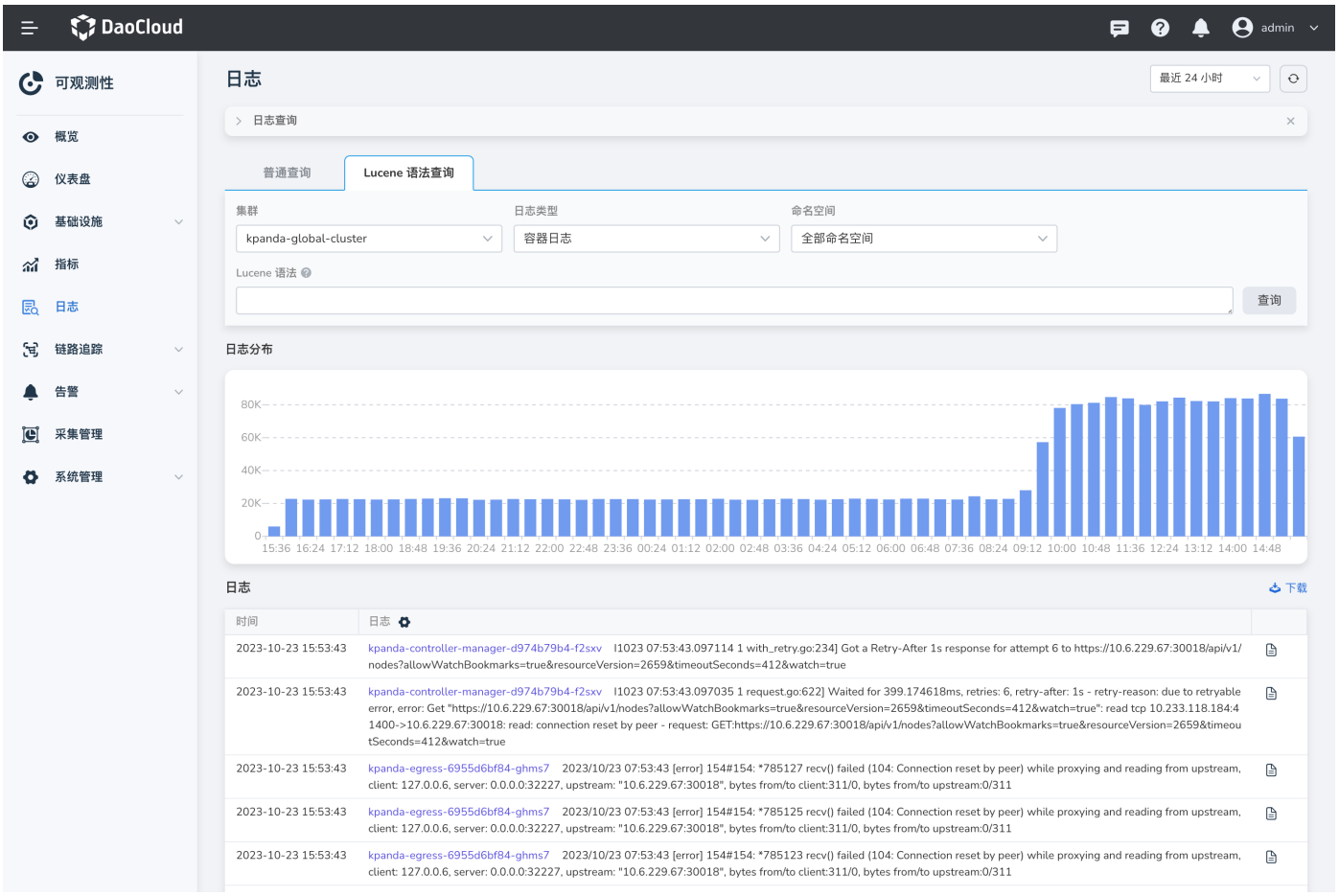
- 容器日志 ：记录集群中容器内部的活动和事件，包括应用程序的输出、错误消息、警告和调试信息等。支持通过集群、命名空间、容器组、容器过滤日志。
- 节点日志 ：记录集群中每个节点的系统级别日志。这些日志包含节点的操作系统、内核、服务和组件的相关信息。支持通过集群、节点、文件路径过滤日志。

c. 支持对单个关键字进行模糊搜索。



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。



Lucene 语法说明：

a. 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。

b. 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~。

c. 使用通配符 (*、?) 用作单字符通配符，表示匹配任意一个字符。

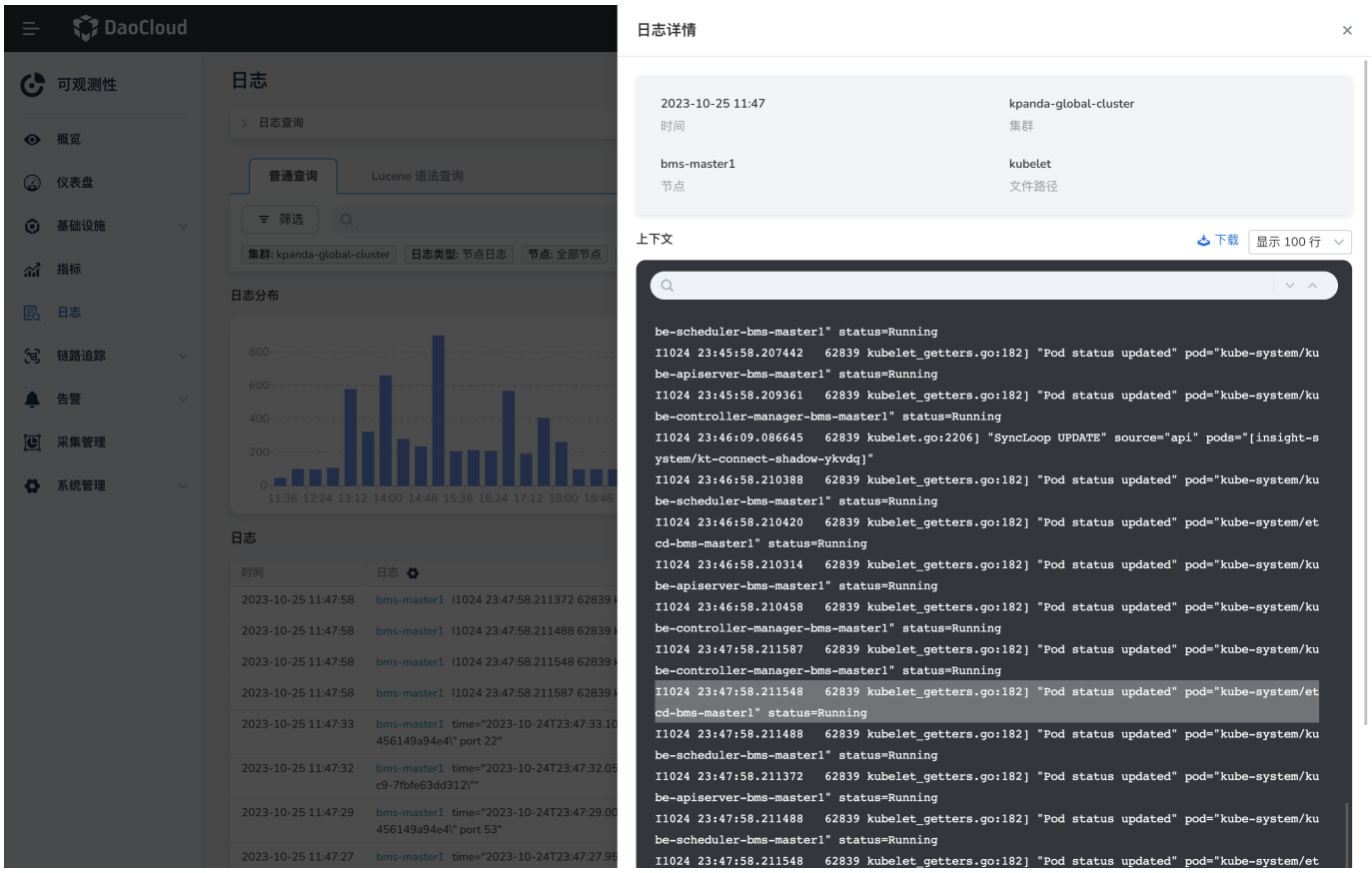
d. 使用方括号 [ ] 或花括号 { } 来查询范围，方括号 [ ] 表示闭区间，包含边界值。花括号 { } 表示开区间，排除边界值。范围查询只适用于能够进行排序的字段类型，如数字、日期等。例如：timestamp:[2022-01-01 TO 2022-01-31]。

e. 更多用法请查看：Lucene 语法说明。

其他操作

查看日志上下文

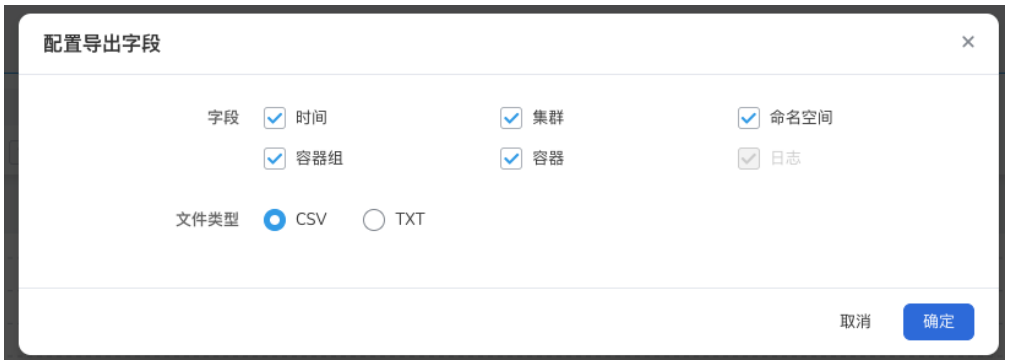
点击日志后的按钮，在右侧划出面板中可查看该条日志的默认 100 条上下文。可切换 显示行数 查看更多上下文内容。



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。



链路追踪

服务拓扑

服务拓扑图是对服务之间连接、通信和依赖关系的可视化表示。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。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。

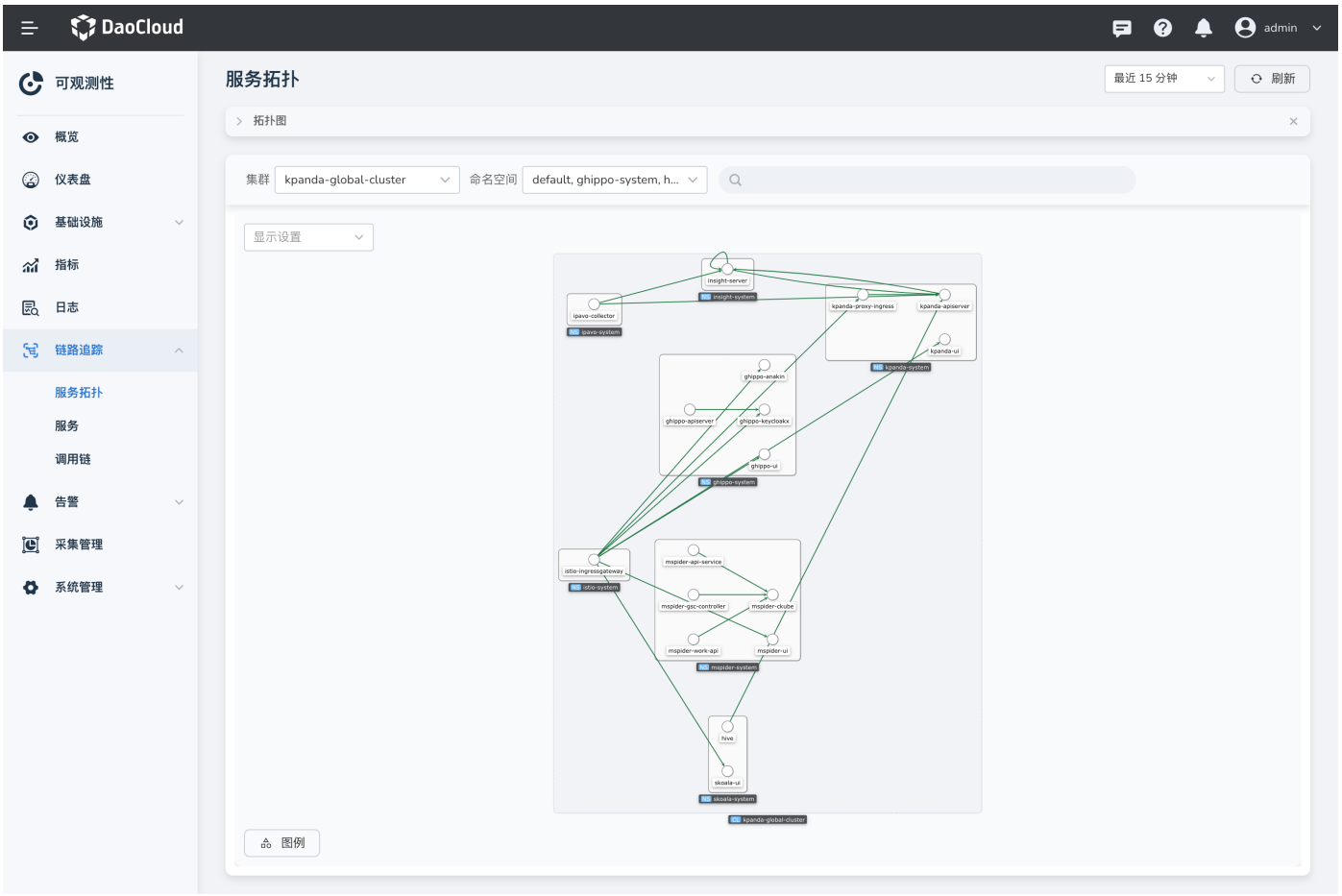
前提条件

1. 集群中已安装 insight-agent 且应用处于 运行中 状态。
2. 服务已通过 Operator 或 Opentelemetry SDK 的方式接入链路。

操作步骤

1. 进入 可观测性 产品模块，
2. 在左边导航栏选择 链路追踪 -> 服务拓扑 。
3. 在拓扑图中，您可按需执行以下操作：

- 单击 节点 ，从右侧划出服务的详情，可查看服务的请求延时、吞吐率、错误率的指标。点击服务名称可跳转至对应服务的详情页。
- 鼠标悬浮在连线上时，可查看两个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。



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）

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。

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。

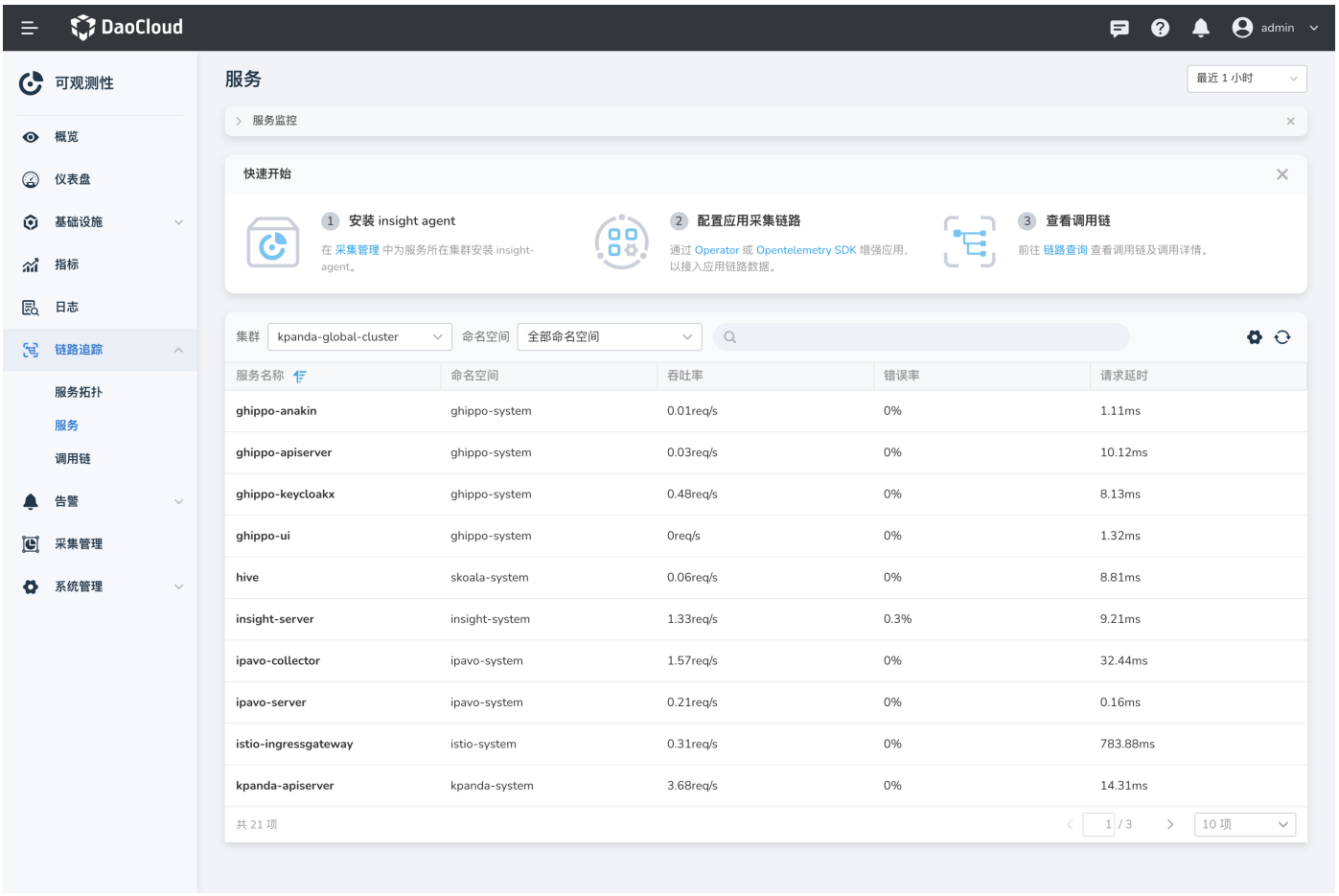
操作步骤

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 模块。

2. 在左边导航栏选择 链路追踪 -> 服务 。



Attention

a. 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。

b. 若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。

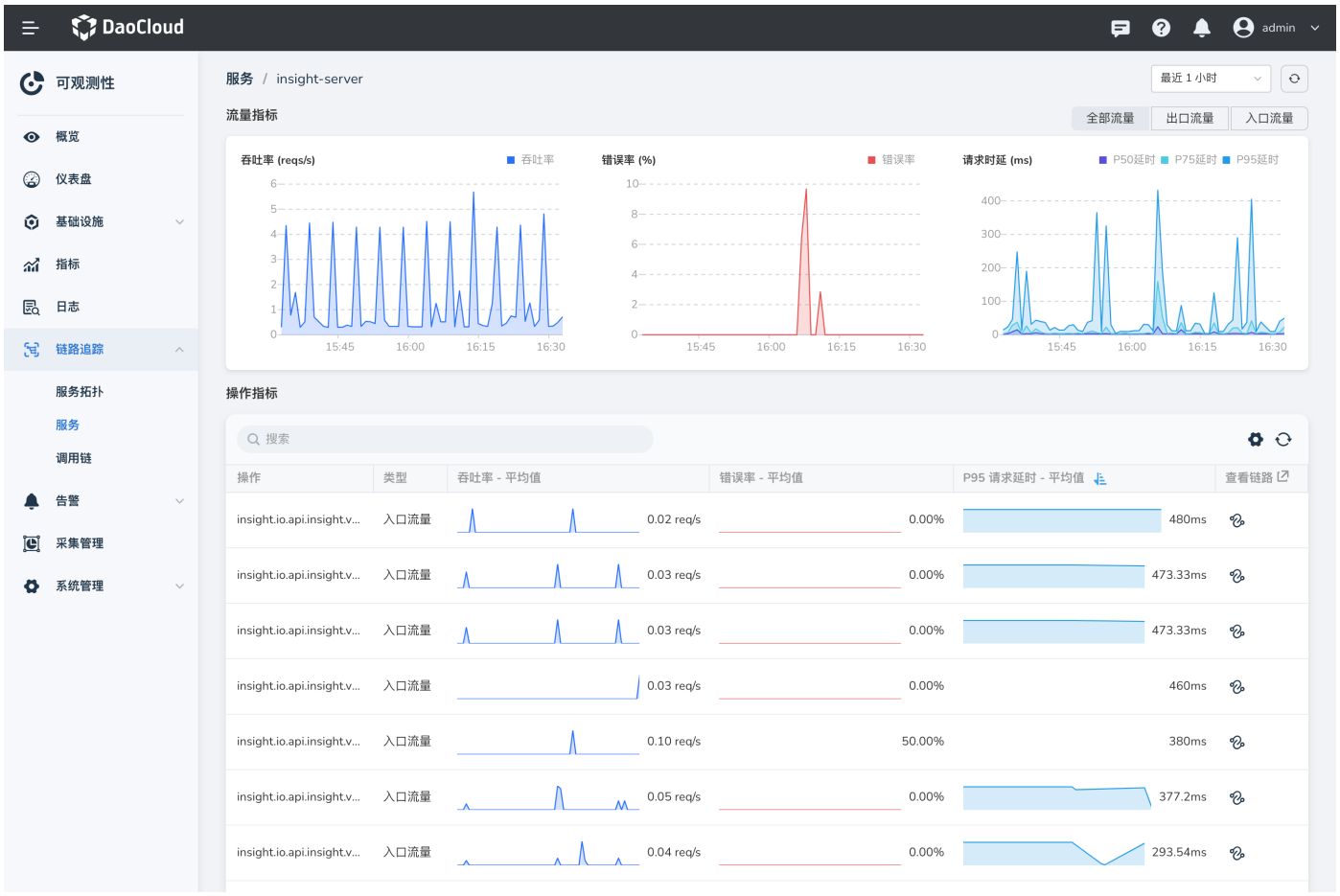
3. 点击服务名 (以 ghippo-keycloakx 为例)，点击进入服务详情页，查看服务的详细指标和该服务的操作指标。

a. 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。

b. 支持通过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。

c. 支持对操作指标中的吞吐率、错误率、请求延时等指标进行排序。

d. 点击单个操作后的图标，可跳转至 调用链 快速查询相关链路。



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。

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。

名词解释

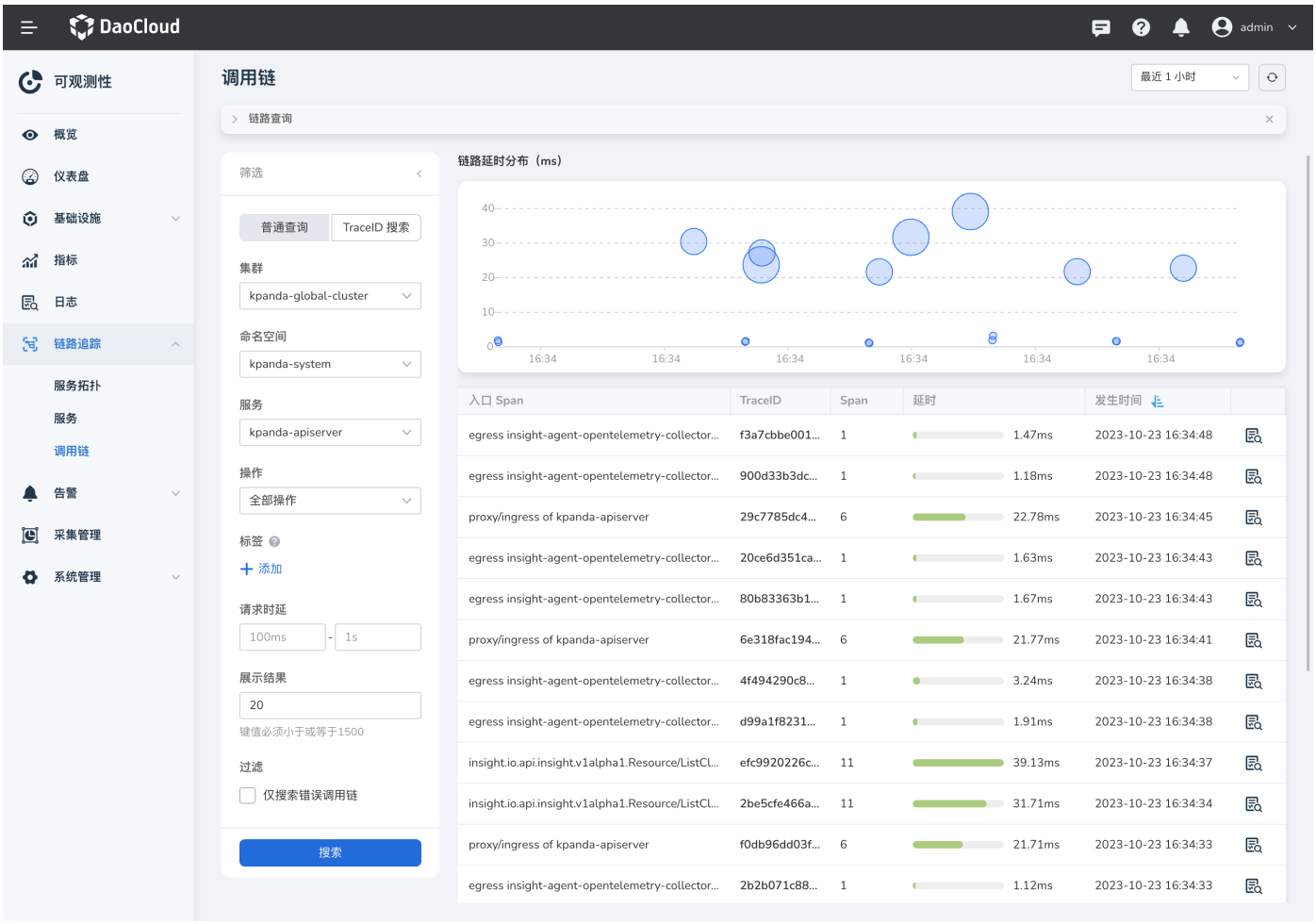
- TraceID：用于标识一个完整的请求调用链路。
- 操作：描述 Span 所代表的具体操作或事件。
- 入口 Span：入口 Span 代表了整个请求的第一个请求。
- 延时：整个调用链从开始接收请求到完成响应的持续时间。
- Span：整个链路中包含的 Span 个数。
- 发生时间：当前链路开始的时间。
- Tag：一组键值对构成的 Span 标签集合，Tag 是用来对 Span 进行简单的注解和补充，每个 Span 可以有多个简直对形式的 Tag。

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，

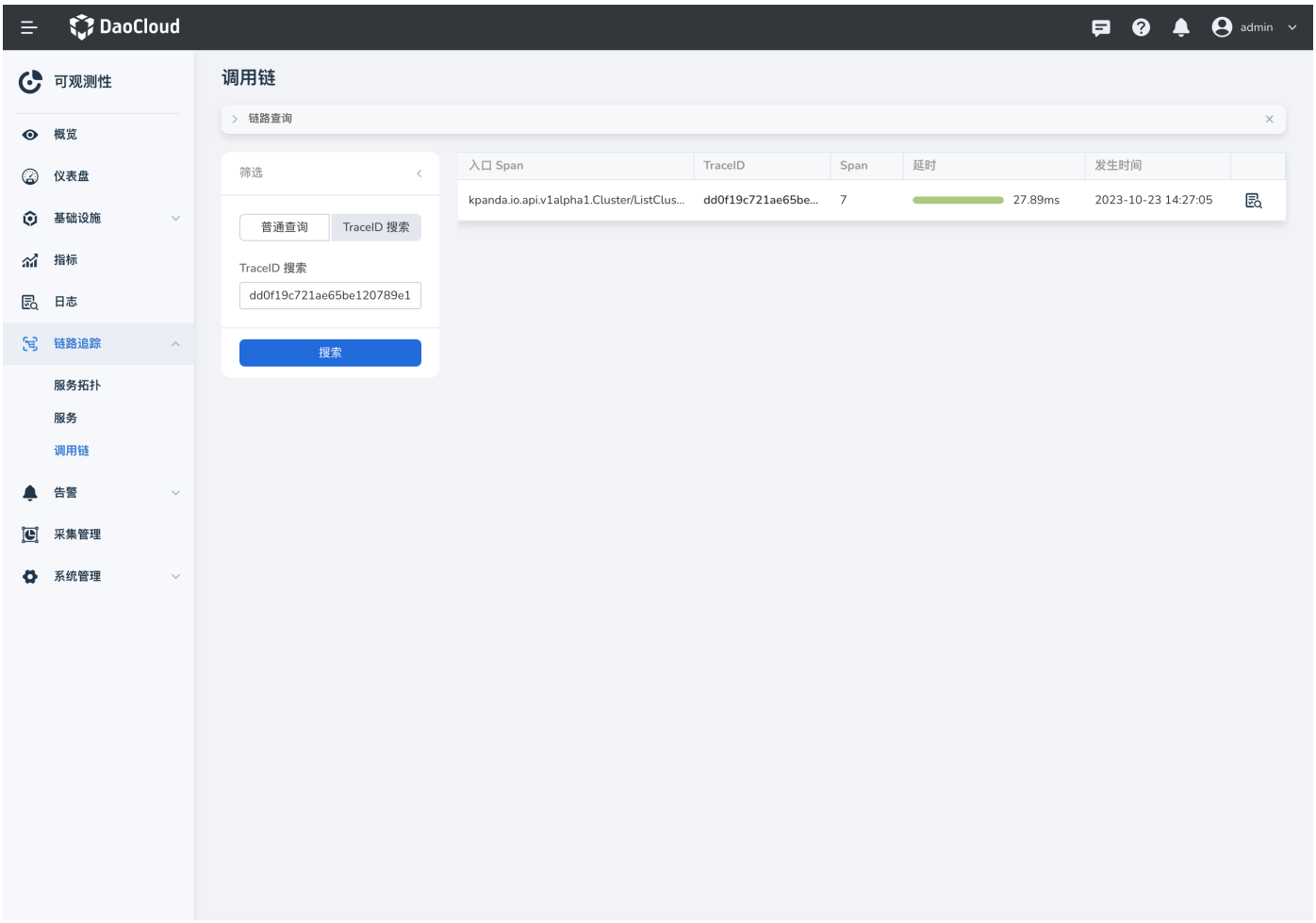
2. 在左边导航栏选择 链路追踪 -> 调用链 。



> **Note**
>
> 列表中支持对 Span 数、延时、发生时间进行排序。

3. 点击筛选栏中的 **Trace ID** 搜索 切换使用 TraceID 搜索链路。

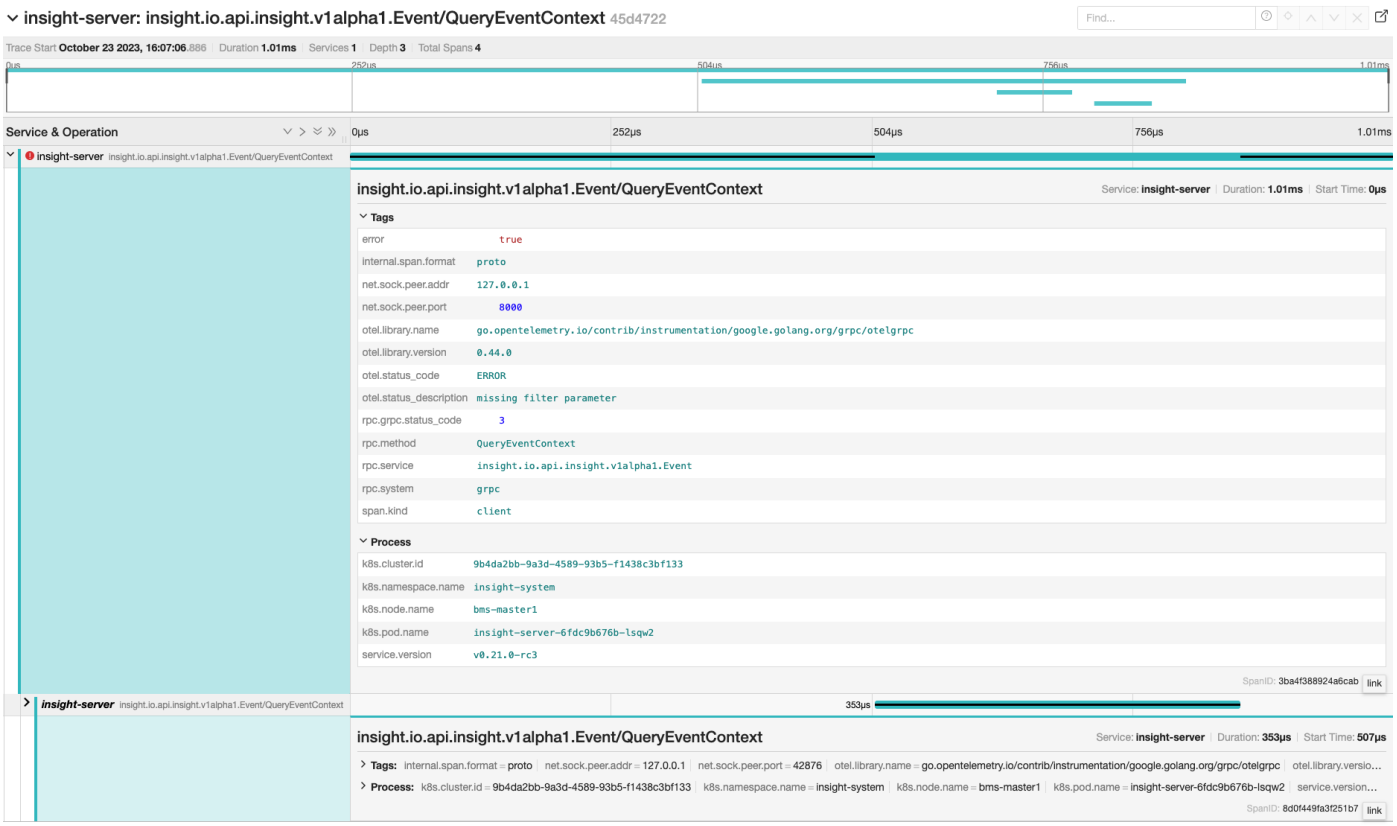
4. 使用 TraceID 搜索请输入完整的 TraceID。


DaoCloud
admin
可观测性
概览
仪表盘
基础设施
指标
日志
链路追踪
服务拓扑
服务
调用链
告警
采集管理
系统管理
调用链
链路查询
筛选
普通查询
TraceID 搜索
TraceID 搜索
dd0f19c721ae65be120789e1
搜索
入口 Span
TraceID
Span
延时
发生时间
kpanda.io.api.v1alpha1.Cluster/ListClus...
dd0f19c721ae65be...
7
27.89ms
2023-10-23 14:27:05

其他操作

查看链路详情

1. 点击链路列表中的某一链路的 TraceID，可查看该链路的详情调用情况。



查看关联日志

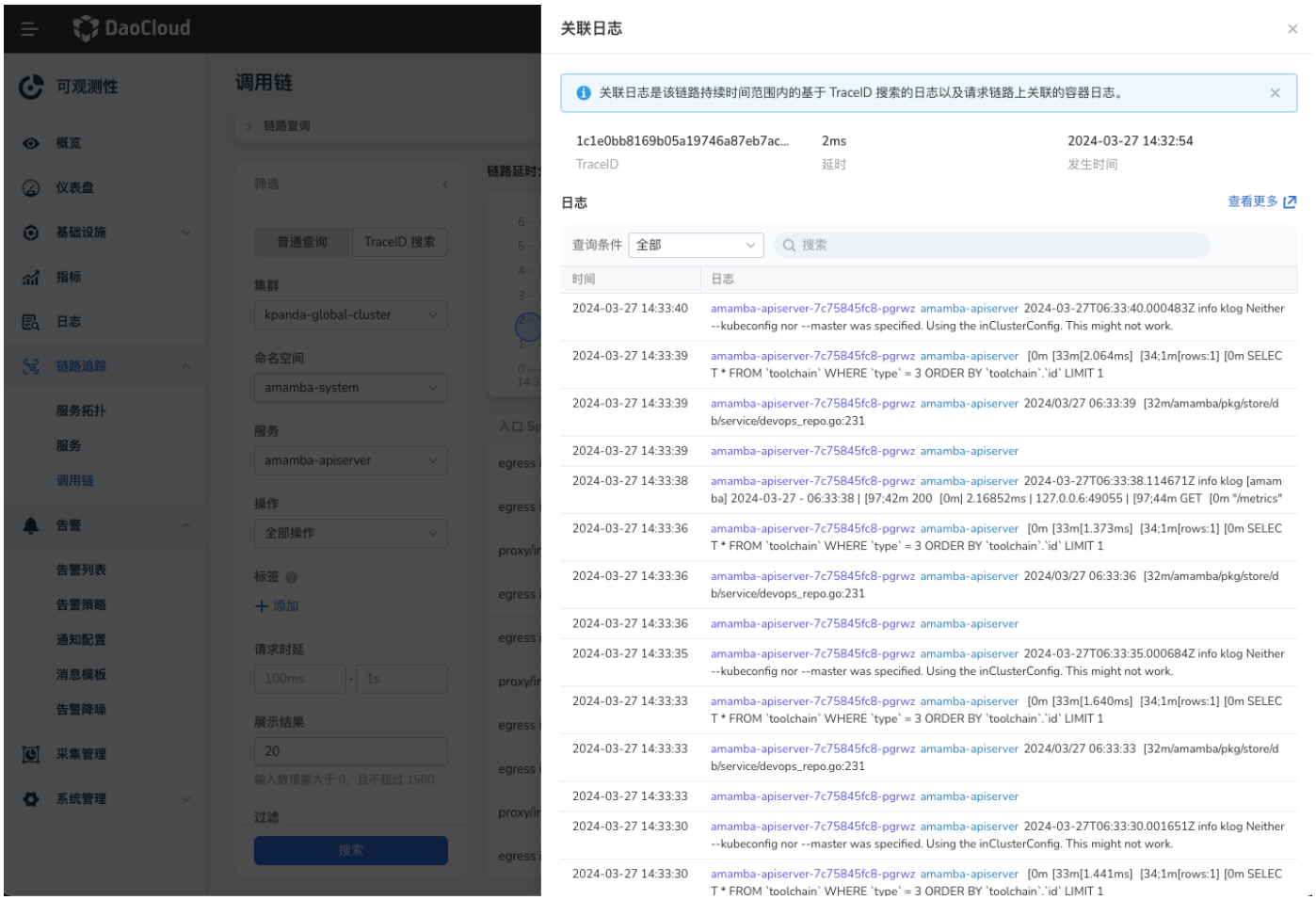
1. 点击链路数据右侧的图标，可查询该链路的关联日志。

• 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. 默认搜索全部日志，但可下拉根据链路的 TraceID 或链路调用过程中相关的容器日志进行过滤。



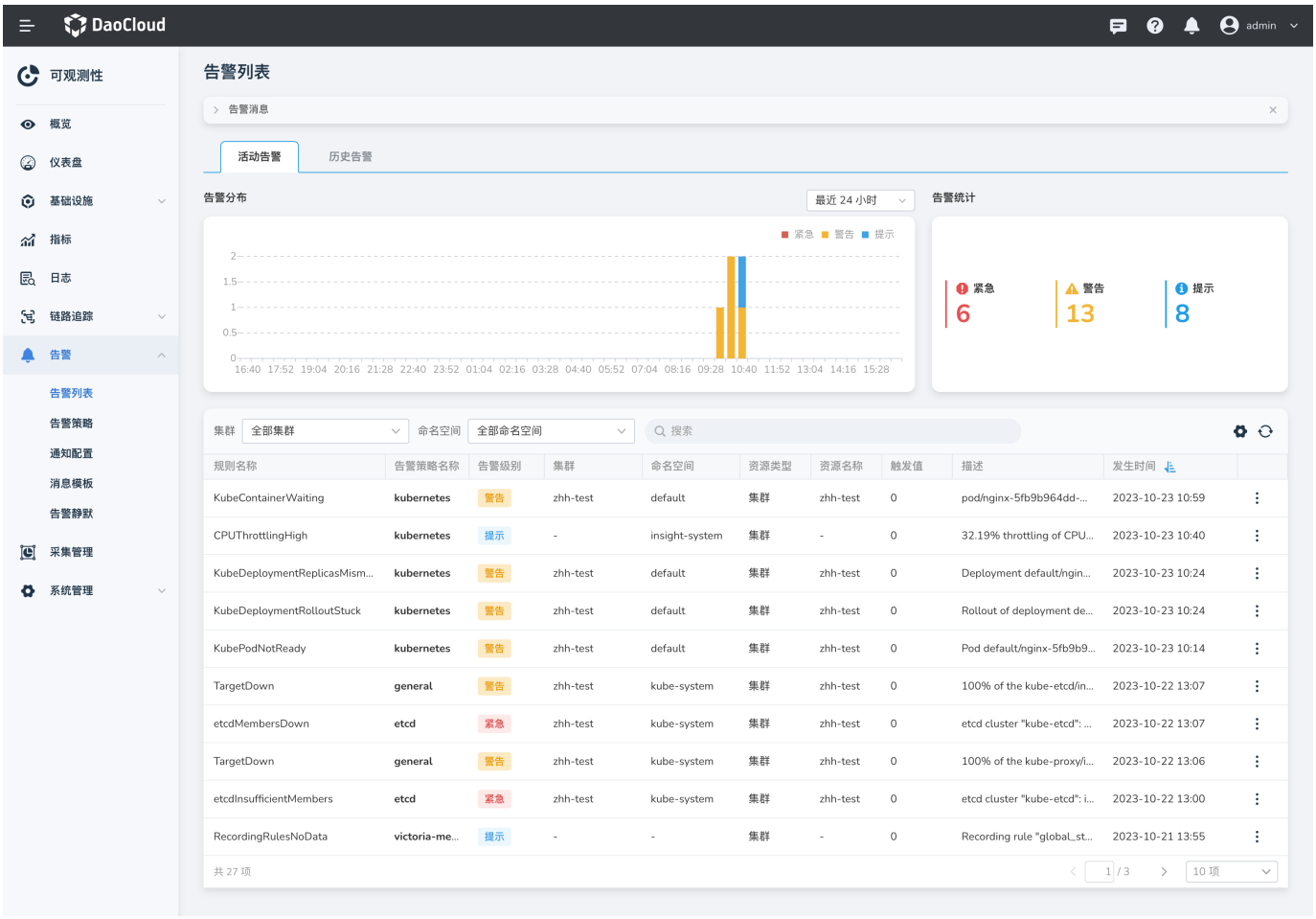
Note

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。

告警

告警中心

告警中心是 d.run 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。



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 d.run 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。

d.run 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。

告警策略

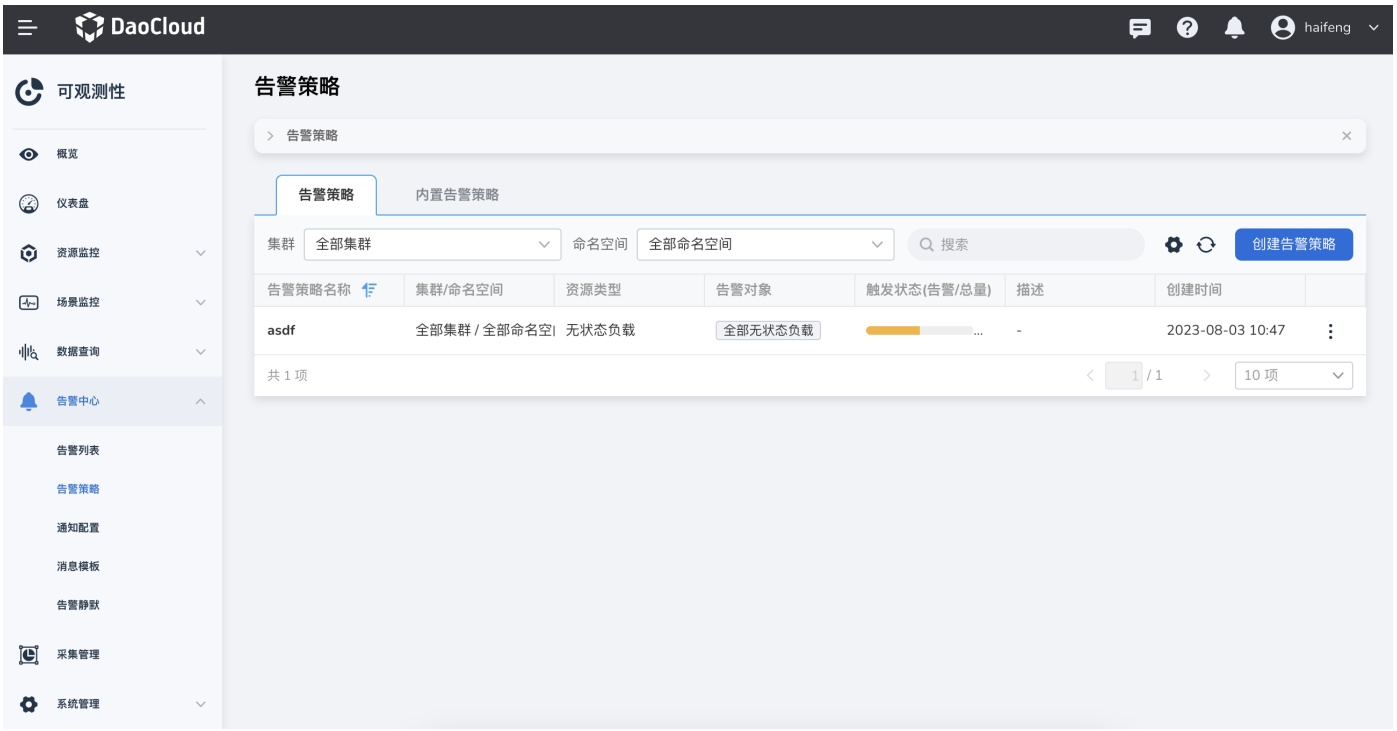
告警策略是在可观测性系统中定义的一组规则和条件，用于检测和触发警报，以便在系统出现异常或达到预定的阈值时及时通知相关人员或系统。

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、日志、事件设置告警规则。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。

查看告警策略

1. 点击一级导航栏进入 可观测性 。
2. 左侧导航栏中，选择 告警中心 -> 告警策略 。

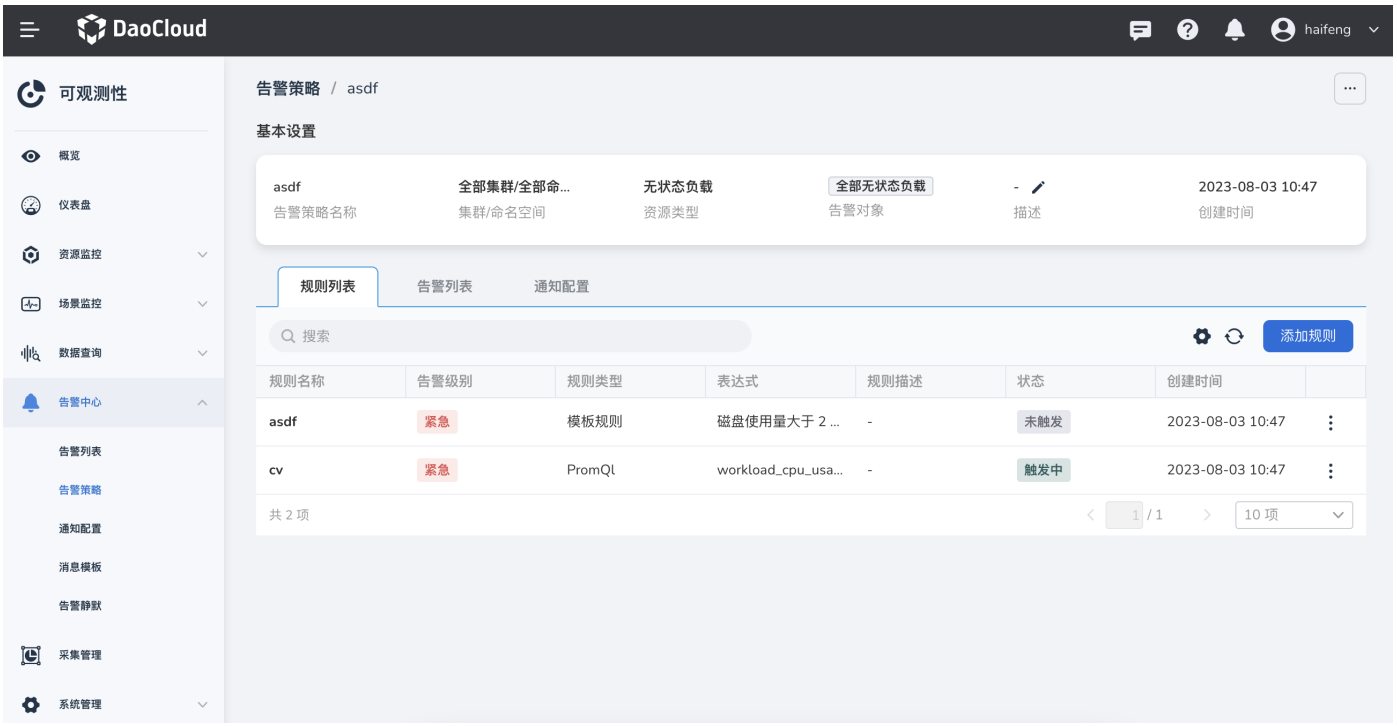
- 集群：单击集群下拉框可切换集群；
- 命名空间：单击命名空间切换下拉框。



3. 点击告警策略名称可查看策略的基本信息、规则以及通知配置。

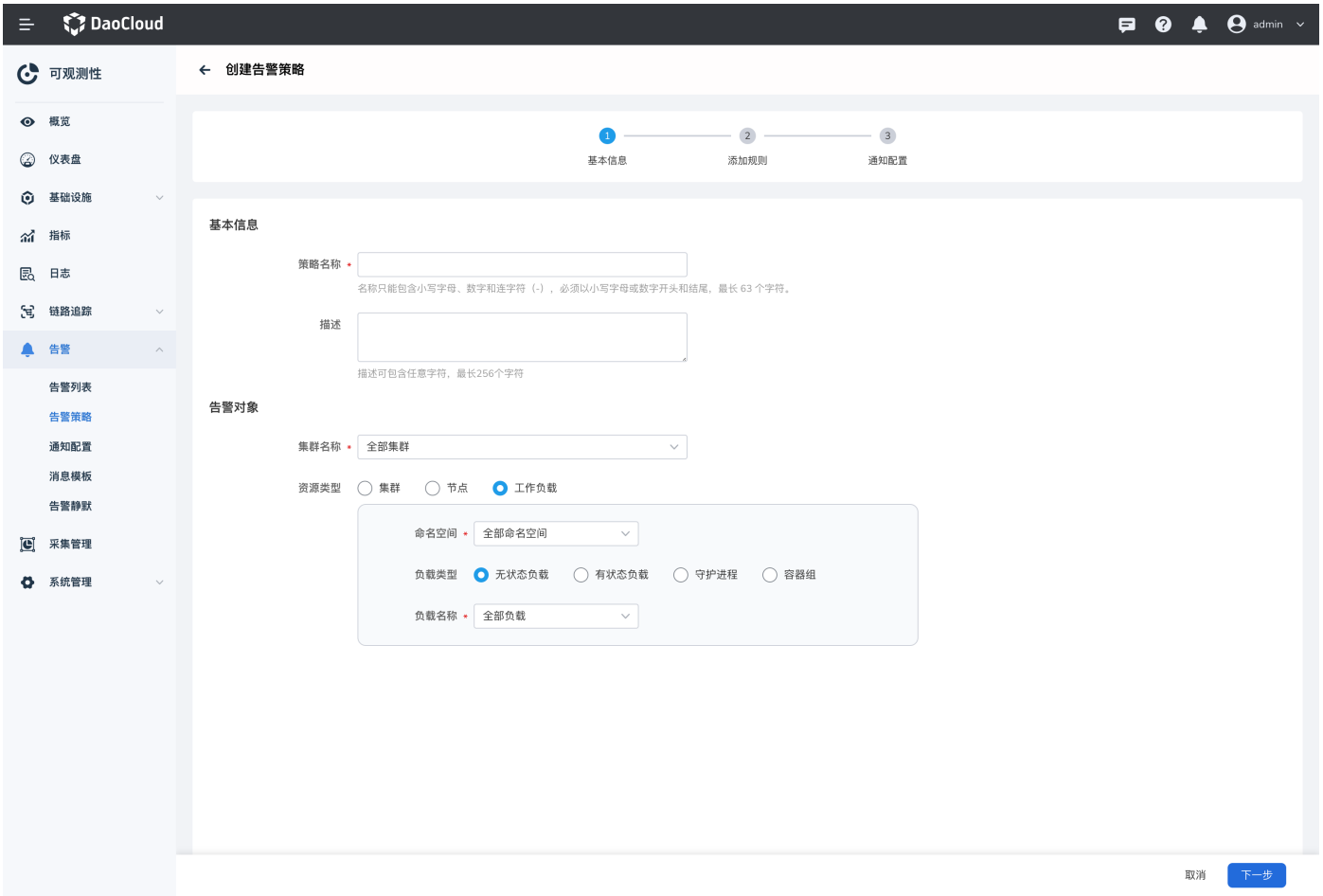
a. 在规则列表中可查看规则类型、规则的表达式、级别、状态等信息。

b. 进入策略详情，可以添加、编辑、删除其下的告警规则。



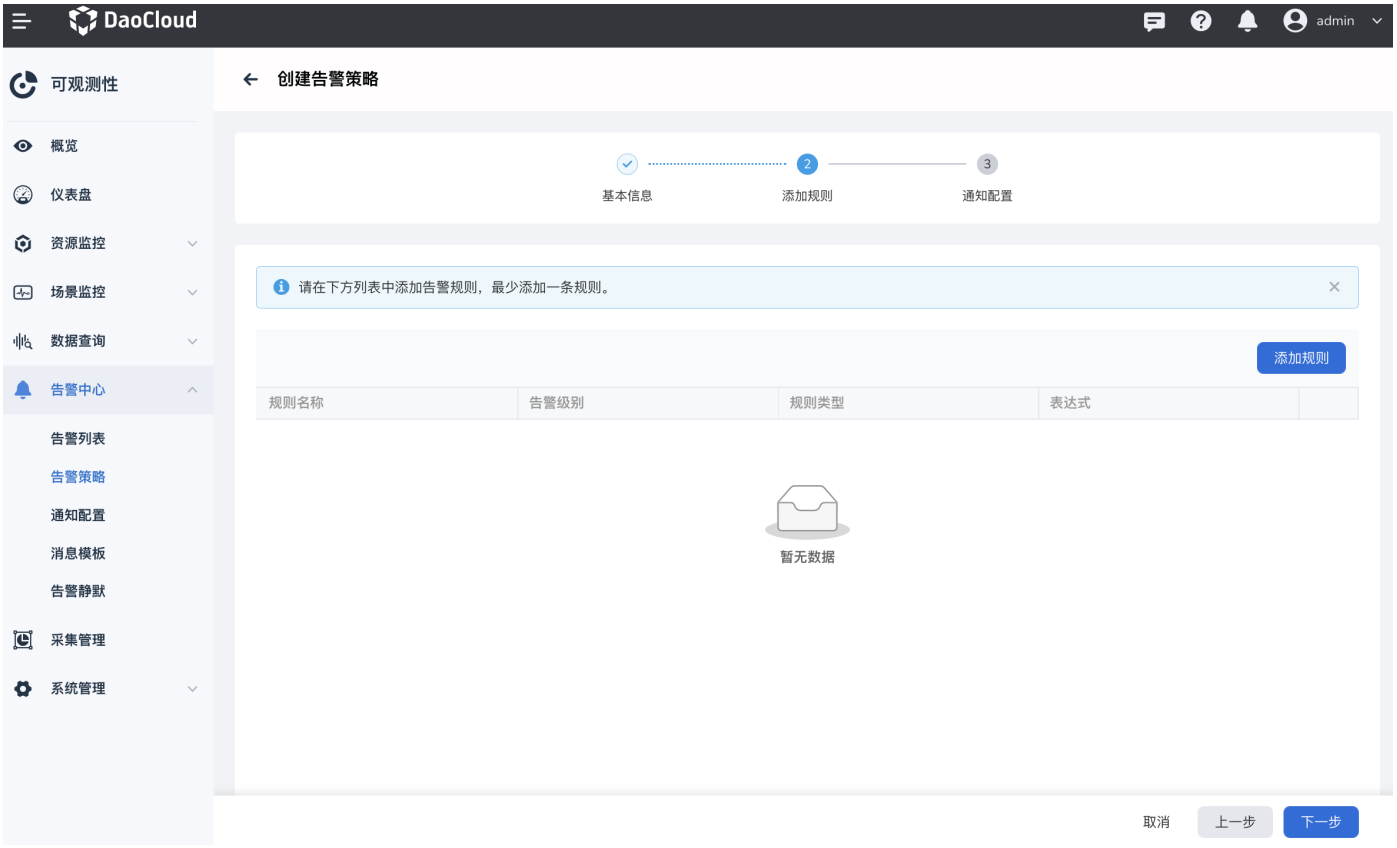
创建告警策略

1. 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 。



2. 列表需至少有一条规则。如果列表为空，可观测性提供了两种方式添加告警规则，

- 方式一：第一种为 表单创建 ，用户可自行定义规则以及阈值等信息。

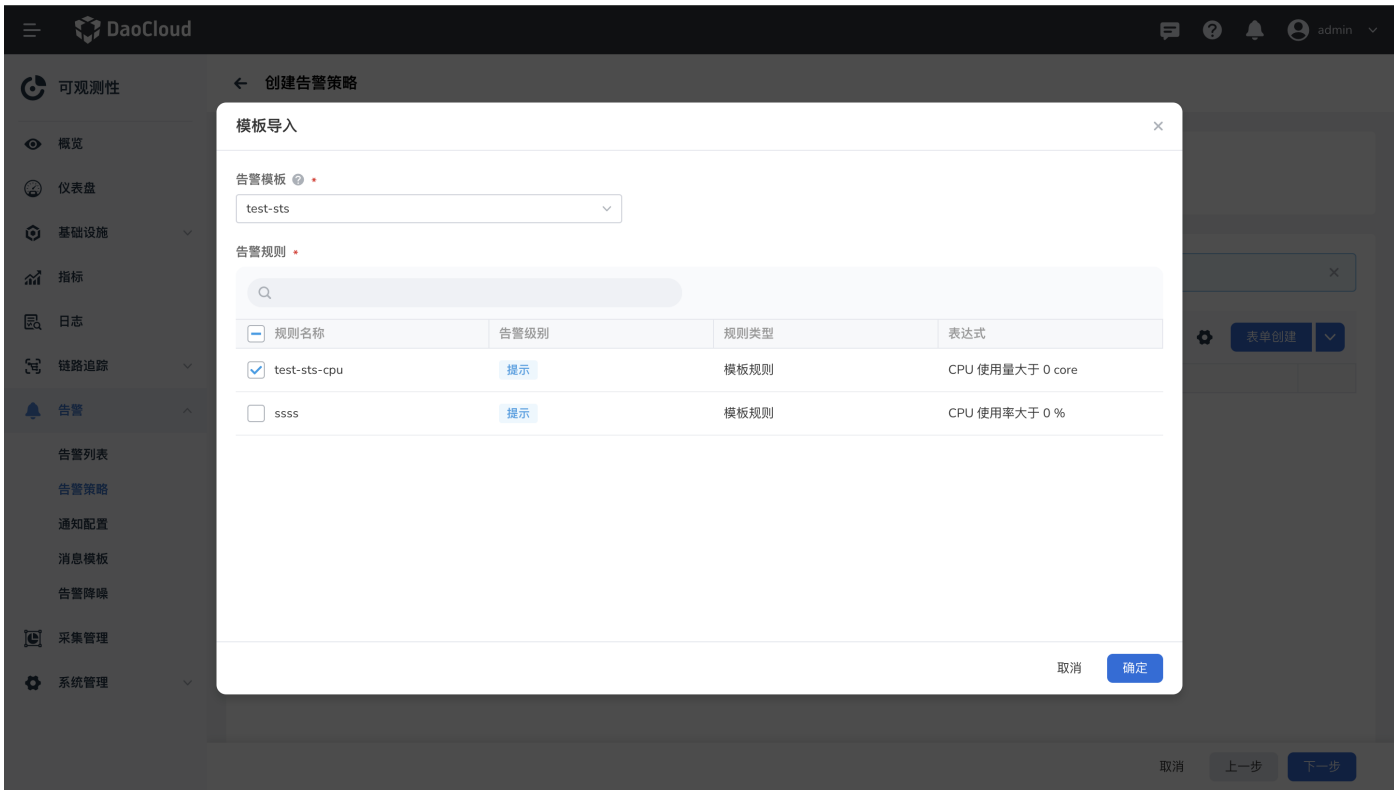


在弹窗中创建告警规则，填写各项参数后点击 确定 。

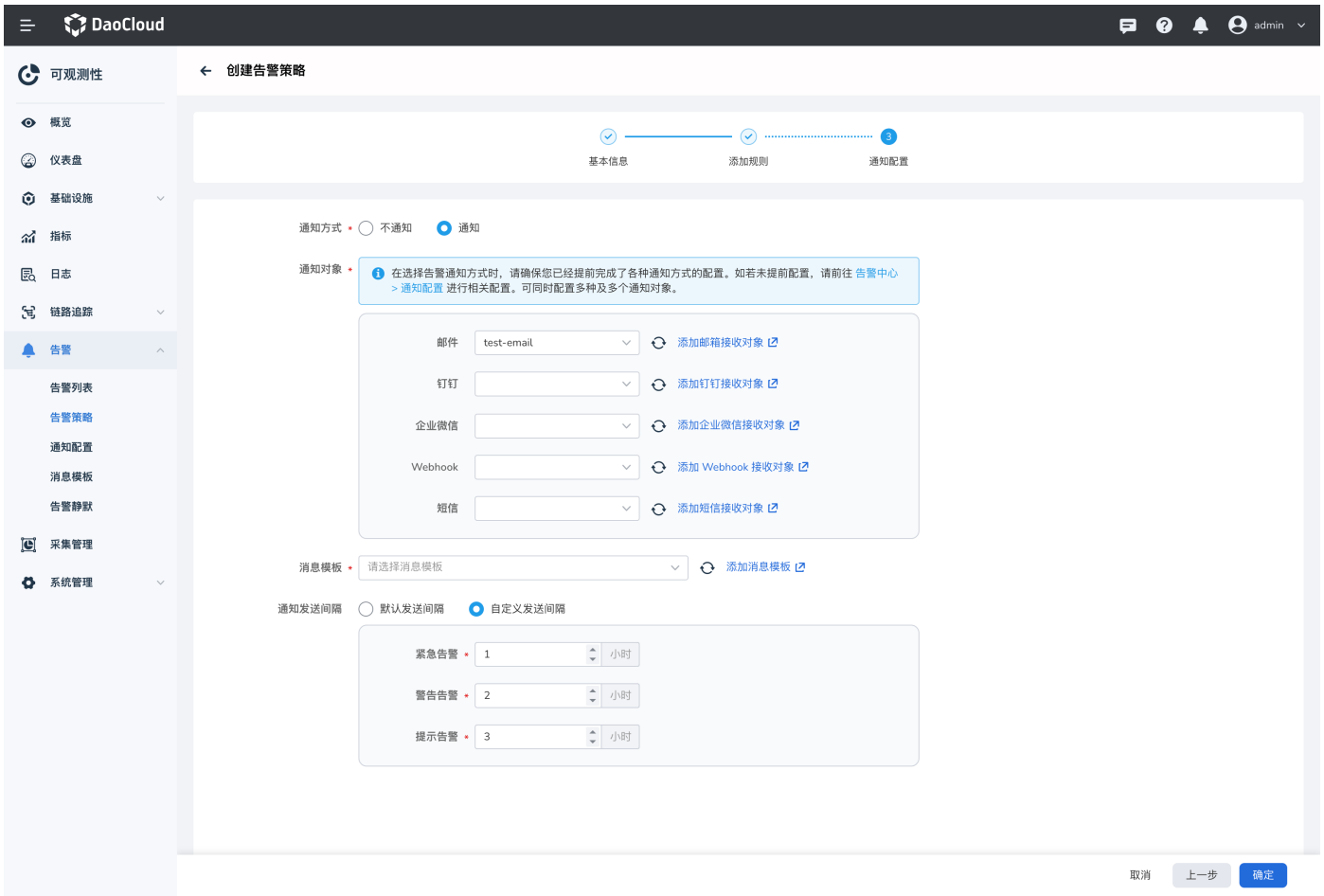
- 模板规则：预定义了基础指标，可以按 CPU、内存、磁盘、网络设定要监控的指标。
- PromQL 规则：输入一个 PromQL 表达式，具体请查询 Prometheus 表达式。
- 持续时长：告警被触发且持续时间达到该设定值后，告警策略将变为触发中状态。
- 告警级别：包含紧急、警告、信息三种级别。
- 高级设置：可以自定义标签和注解。

> **Info**
>
> 系统定义了内置标签，若自定义标签与内置标签的 `键` 值相同，则自定义标签不生效。 内置标签有：`severity`、`rule_id`，`source`、`cluster_name`、`group_id`、 `target_type` 和 `target`。

- 方式二：可点击 模板导入 ，选择平台管理员已创建好的告警模板批量导入告警规则。



3. 点击 下一步 后配置通知。



4. 配置完成后，点击 确定 按钮，返回告警策略列表。

Tip

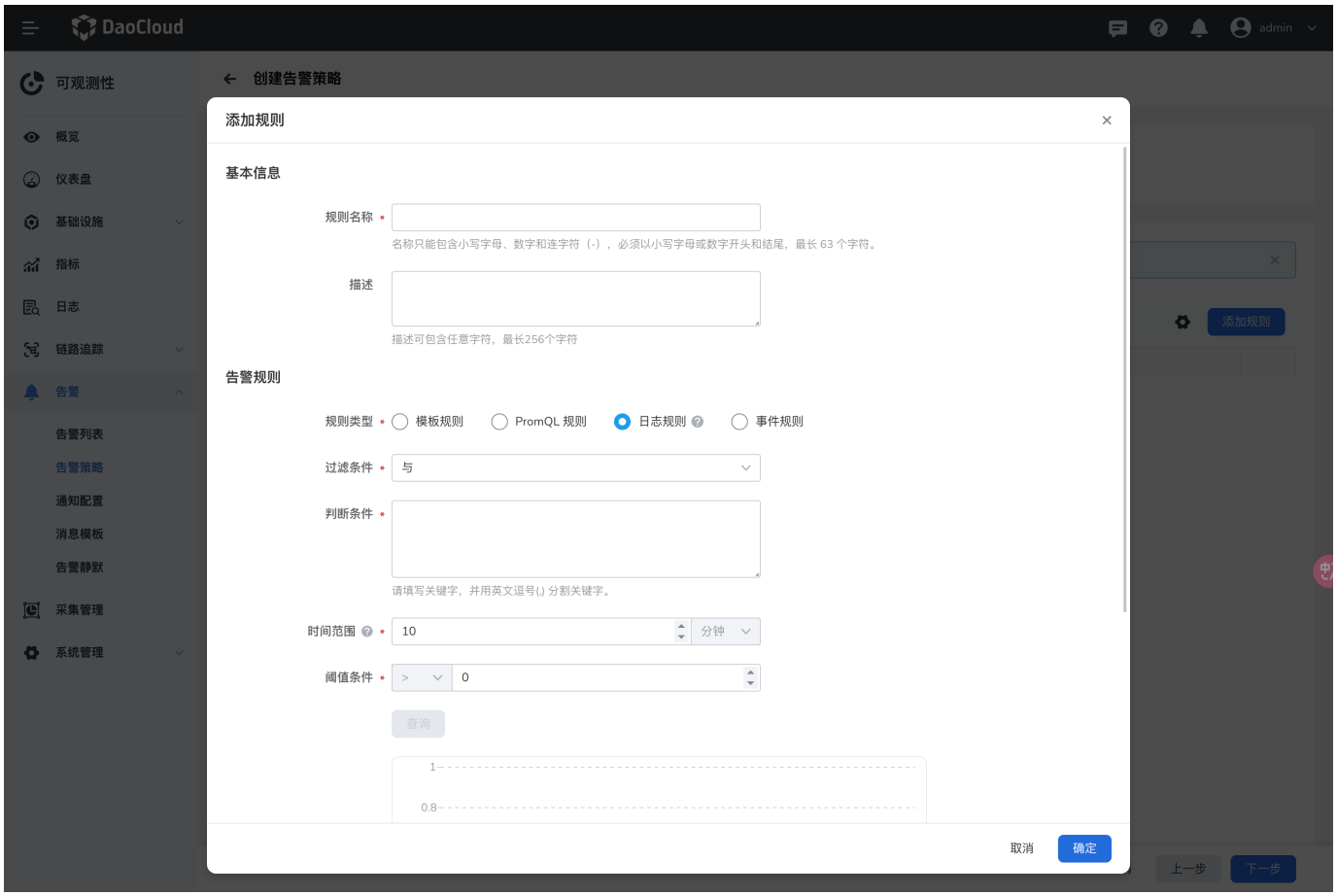
新建的告警策略为 未触发 状态。一旦满足规则中的阈值条件和持续时间后，将变为 触发中 状态。

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 。

Note

仅当资源对象选择节点或工作负载时，支持创建日志规则。



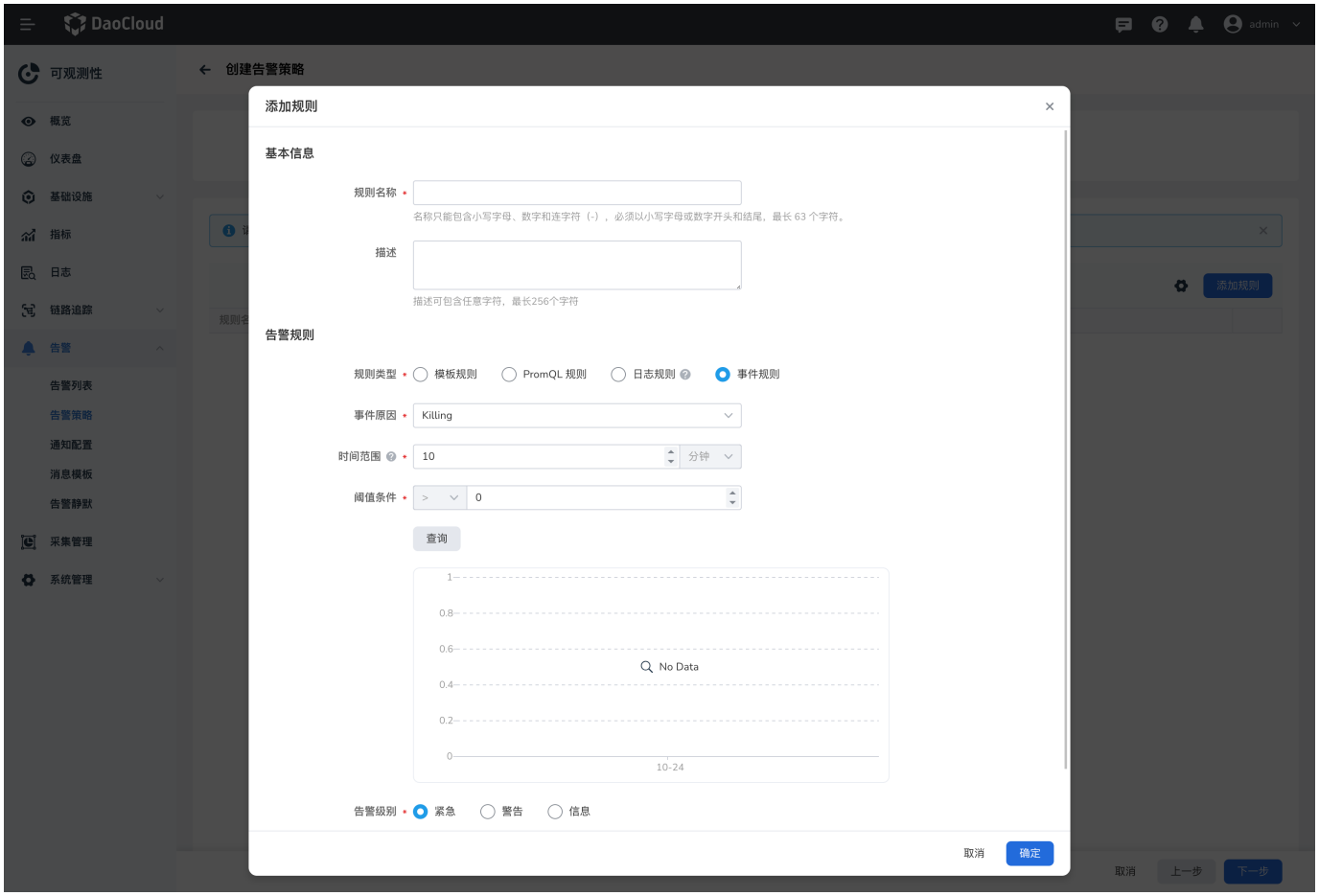
字段说明：

- 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 。

> **Note**
>
> 仅当资源对象选择工作负载时，支持创建事件规则。



字段说明：

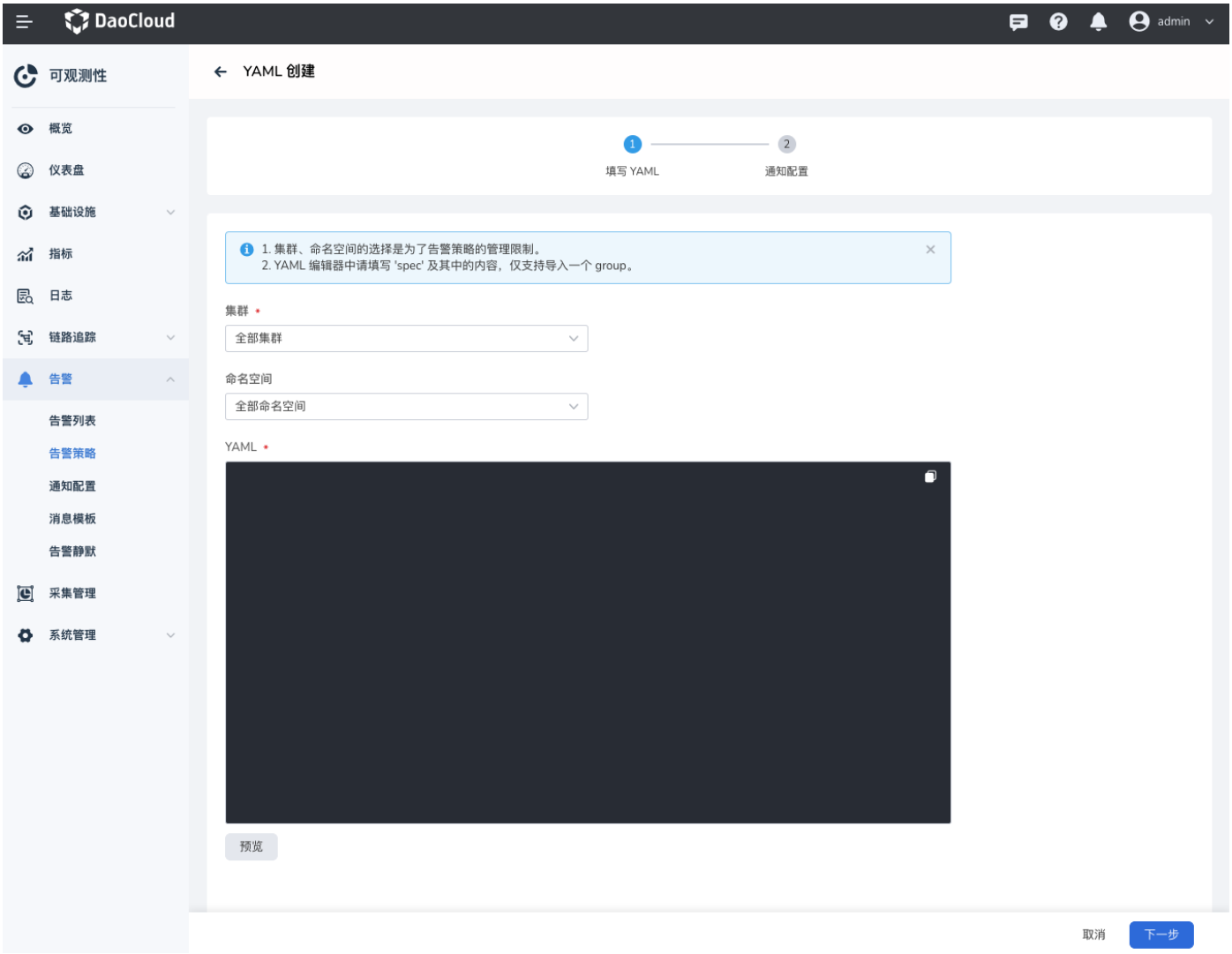
- 事件规则 ：仅支持资源对象选择工作负载
- 事件原因 ：不同的工作负载类型的事件原因不同，事件原因之间是“和”的关系。
- 时间范围 ：检测该时间范围内产生数据，若达到设置的阈值条件，则触发告警事件。
- 阈值条件 ：当产生的事件达到设置的阈值时，则触发告警事件。
- 趋势图 ：默认查询 10 分钟内的事件变化趋势，每个点的数值统计的是当前时间点到之前的某段时间（时间范围）内发生的总次数。

> **Warning**
>
> 删除后的告警策略将完全消失，请谨慎操作。

通过 YAML 导入告警策略

1. 进入告警策略列表，点击 **YAML** 创建 。
2. 集群、命名空间的选择是为了告警策略的管理权限。
3. YAML 编辑器中请填写 **spec** 及其中的内容，仅支持导入一个 group。
4. 告警规则名称 需要符合规范：名称只能包含大小写字母、数字、下划线（_）和连字符（-），必须以字母开头，最长 63 个字符。
5. 必填 **severity** 且符合规范：critical、warning、info。

6. 必填表达式 **expr** 。



7. 导入 YAML 文件后，点击 预览 ，可以对导入的 YAML 格式进行验证，并快速确认导入的告警规则。

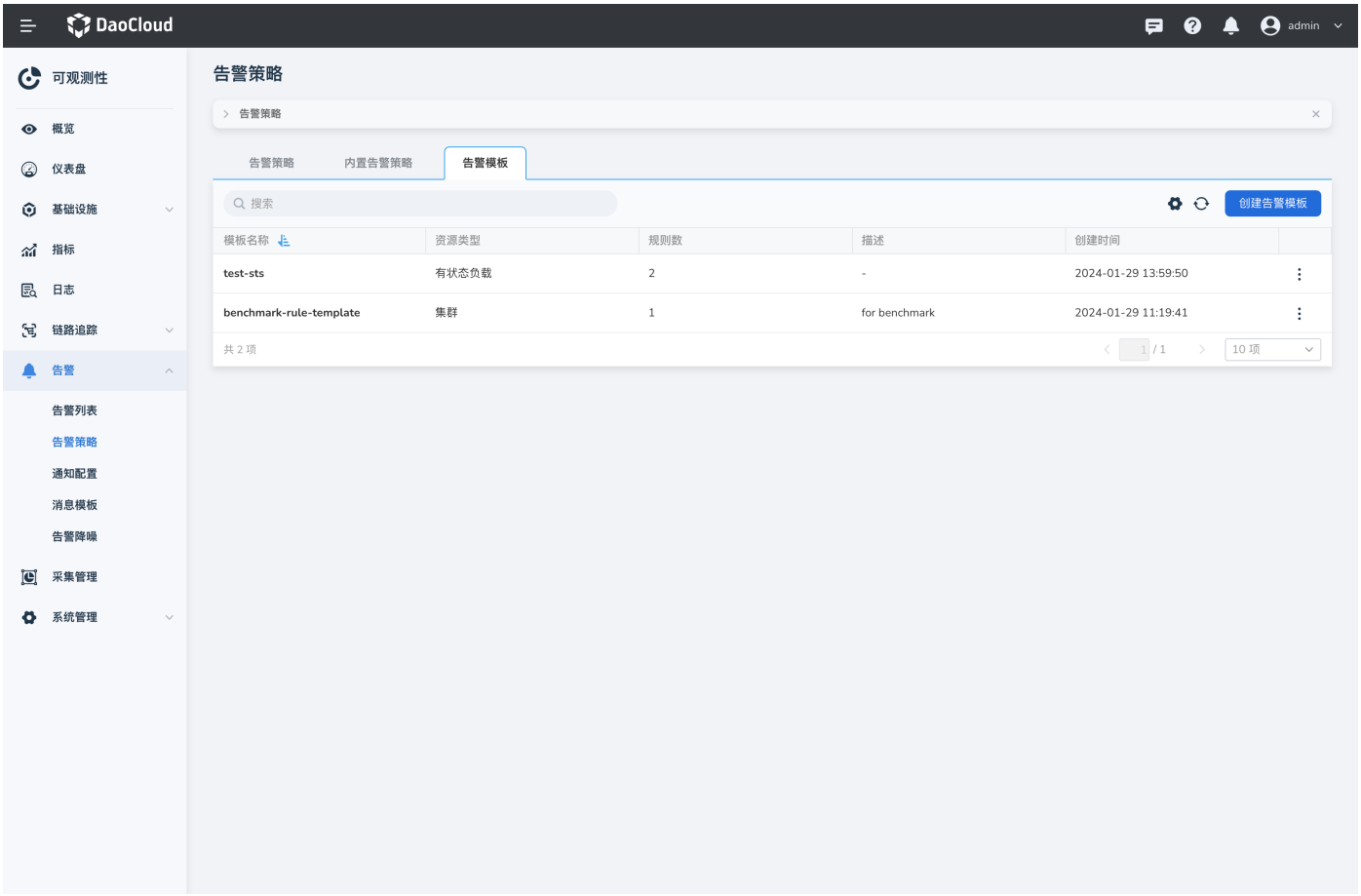

DaoCloud
admin
可观测性
YAML 创建
概览
仪表盘
基础设施
指标
日志
链路追踪
告警
告警列表
告警策略
通知配置
消息模板
告警静默
采集管理
系统管理
预览规则
基本信息
策略名称 prometheusoperator
集群 全部集群
命名空间 全部命名空间
告警规则
规则名称 告警级别 规则类型 表达式 规则描述 标签 注解
prometheusoper... 提示 PromQl (sum by (cluster,... Errors while perf... - summary: Error...
prometheusoper... 警告 PromQl (sum by (cluster,... Errors while perf... - summary: Error...
共 2 项
1 / 1
10 项
关闭
runbook_url: >-
./ui/insight-runbook/runbooks/prometheus-operator/prometheusoperatorlisterrors
summary: Errors while performing list operations in controller.
expr: >-
(sum by (cluster,controller,namespace)
(rate(prometheus_operator_list_operations_failed_total{job="insight-agent-kube-prometh-operator",namespace="insight-system"}[10m]))
预览

告警模板

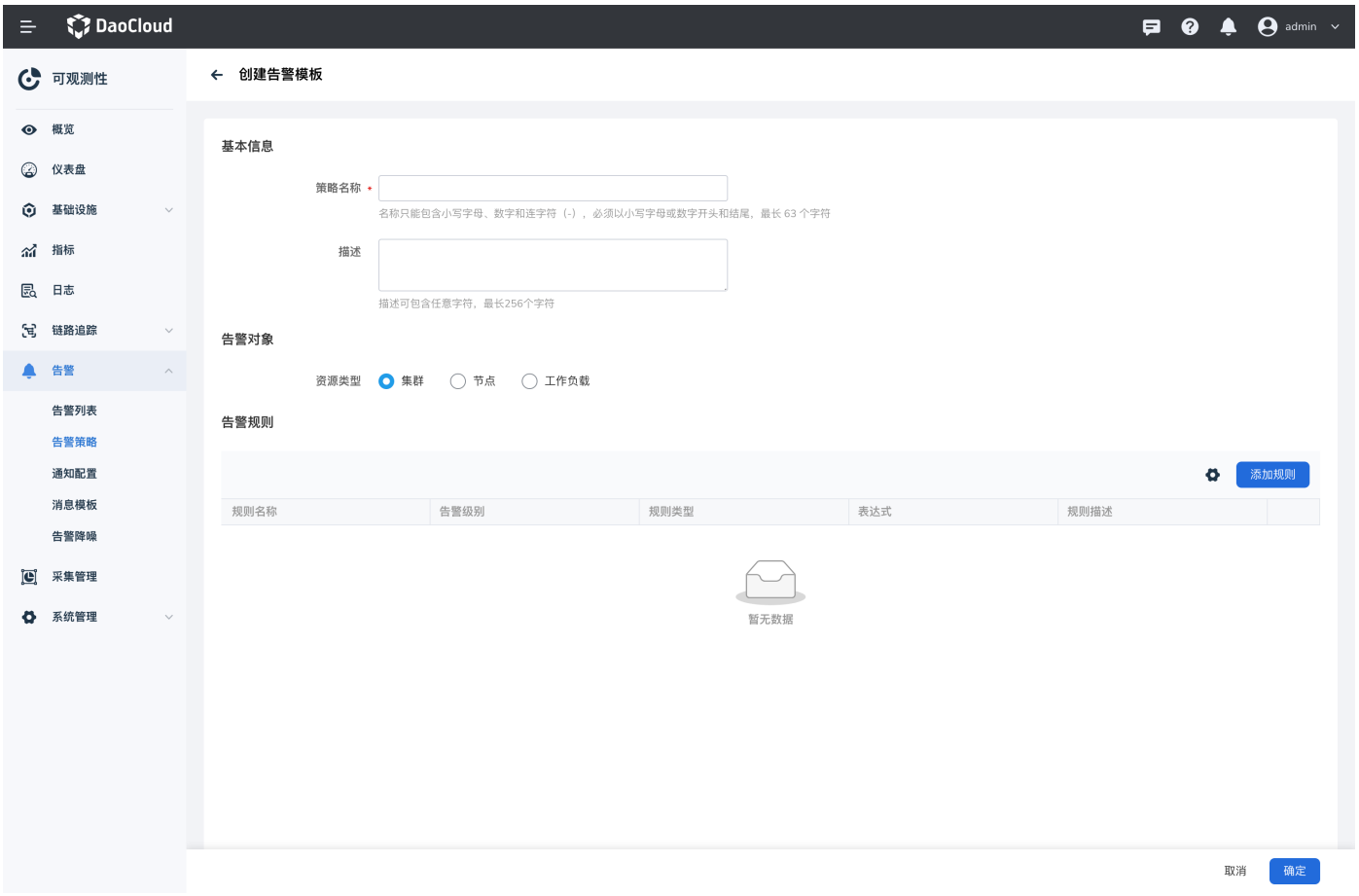
告警模板可支持平台管理员创建告警模板及规则，业务侧可以直接使用告警模板创建告警策略。 这个功能可以减少业务人员对告警规则的管理，且可以根据环境实际情况自行修改告警阈值。

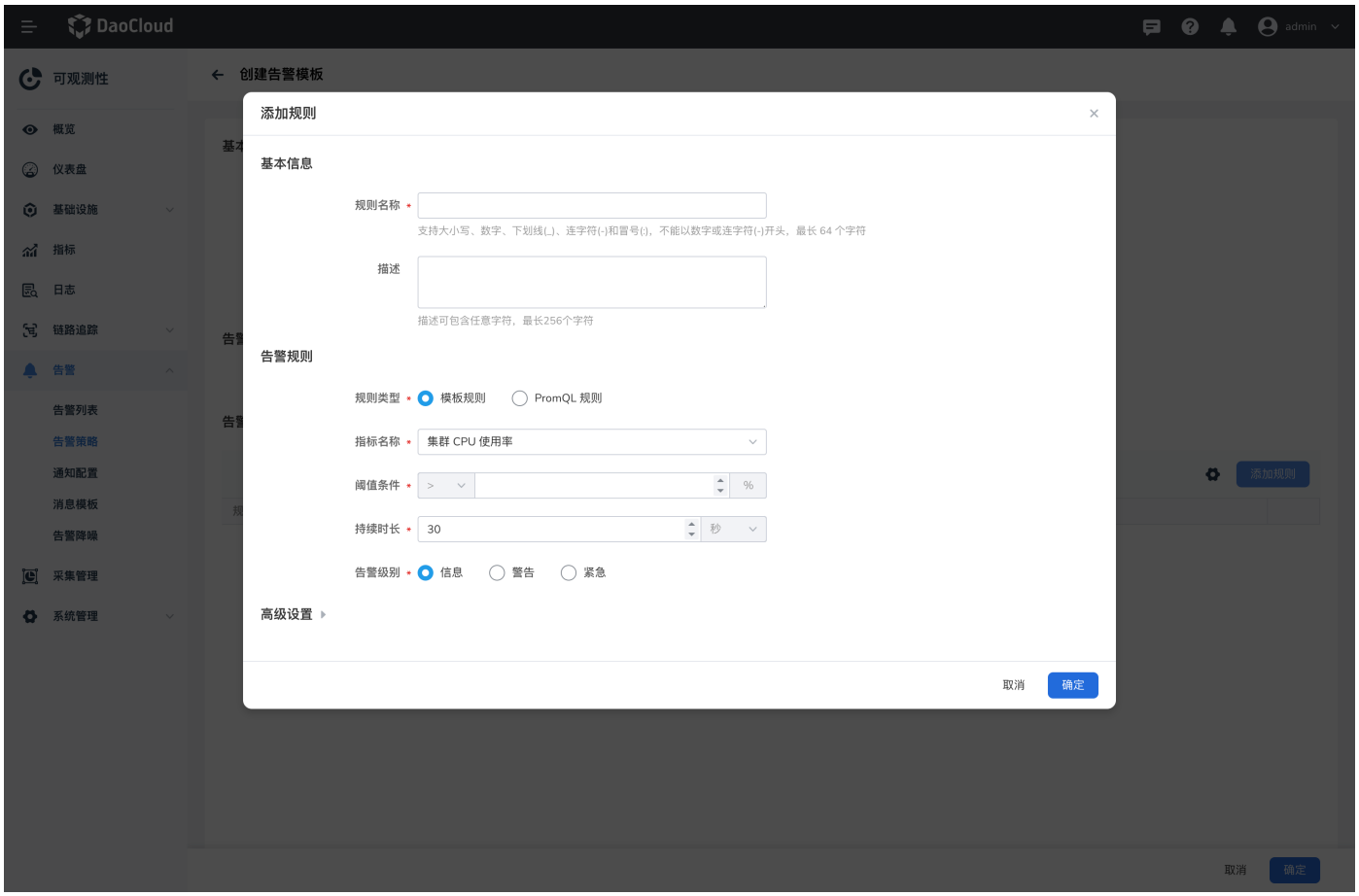
创建告警模板

1. 左侧导航栏中，选择 告警中心 -> 告警策略 ，单击顶部的 告警模板 。



2. 点击 创建告警模板 ，设置告警模板的名称、描述等信息。


DaoCloud
admin
可观测性
概览
仪表盘
基础设施
指标
日志
链路追踪
告警
告警列表
告警策略
通知配置
消息模板
告警降噪
采集管理
系统管理
创建告警模板
基本信息
策略名称 *
名称只能包含小写字母、数字和连字符（-），必须以小写字母或数字开头和结尾，最长 63 个字符
描述
描述可包含任意字符，最长256个字符
告警对象
资源类型
集群
节点
工作负载
告警规则
添加规则
规则名称
告警级别
规则类型
表达式
规则描述
暂无数据
取消
确定

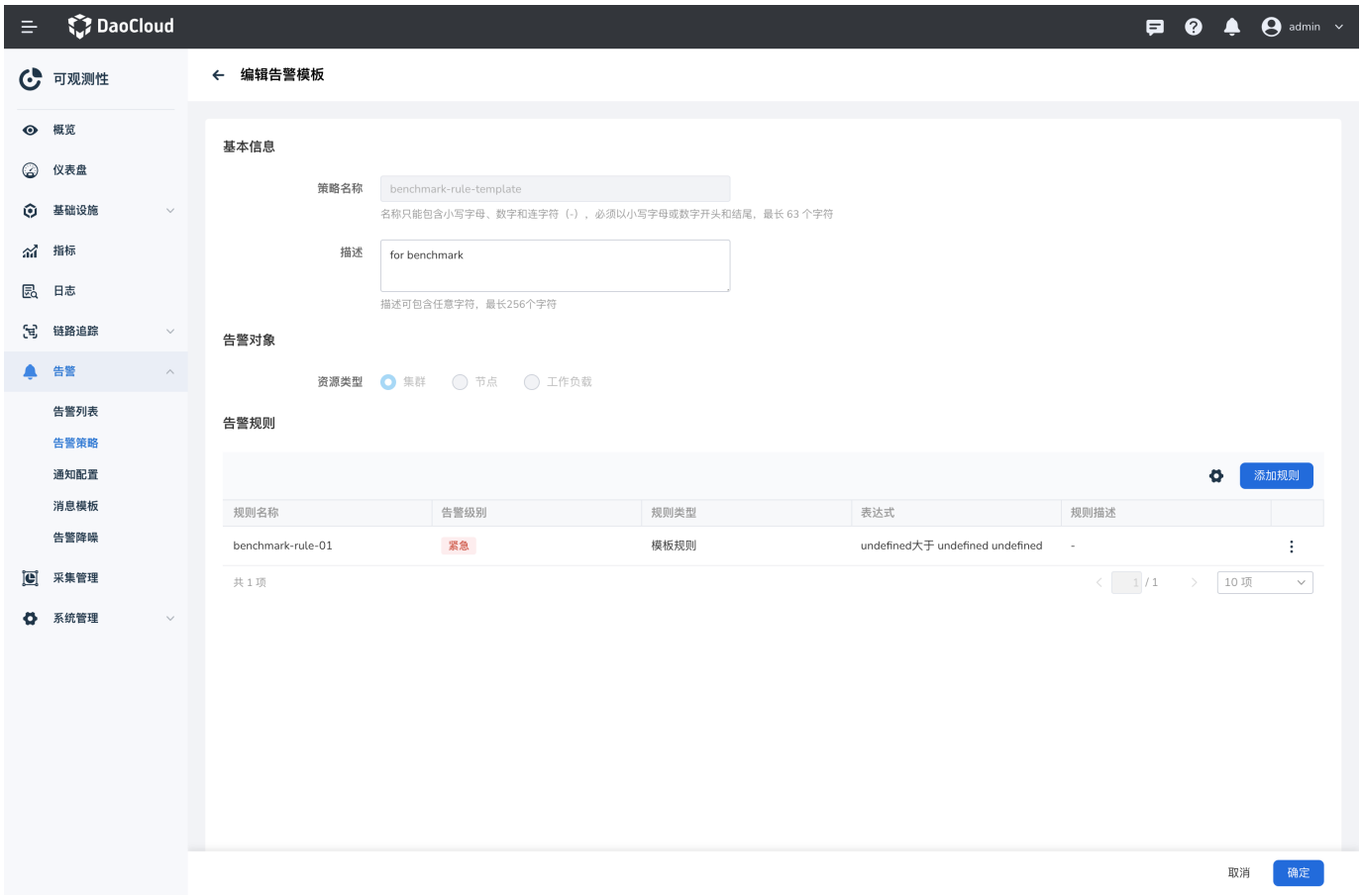


参数	说明
模板名称	名称只能包含小写字母、数字和连字符（-），必须以小写字母或数字开头和结尾，最长 63 个字符。
描述	描述可包含任意字符，最长 256 个字符。
资源类型	用于指定告警模板的匹配类型。
告警规则	支持预定义多个告警规则，可添加模板规则、PromQL 规则。

3. 点击 确定 完成创建后返回告警模板列表，点击模板名称后可查看模板详情。

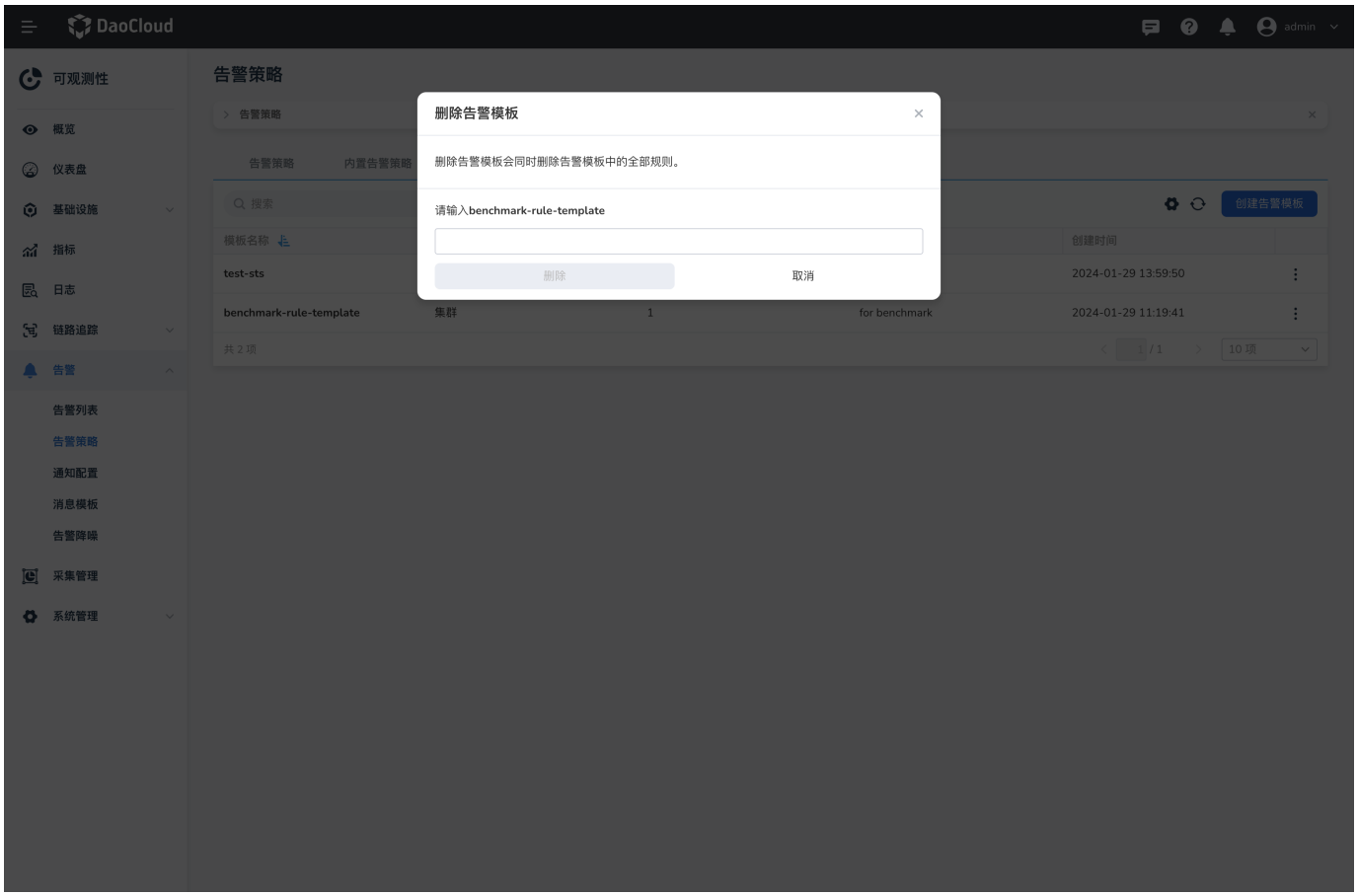
编辑告警模板

点击目标规则后的 ⋮ ，点击 编辑 ，进入抑制规则的编辑页。



删除告警模板

点击目标模板后侧的 ⋮ ，点击 删除 ，在输入框中输入告警模板的名称即可删除。

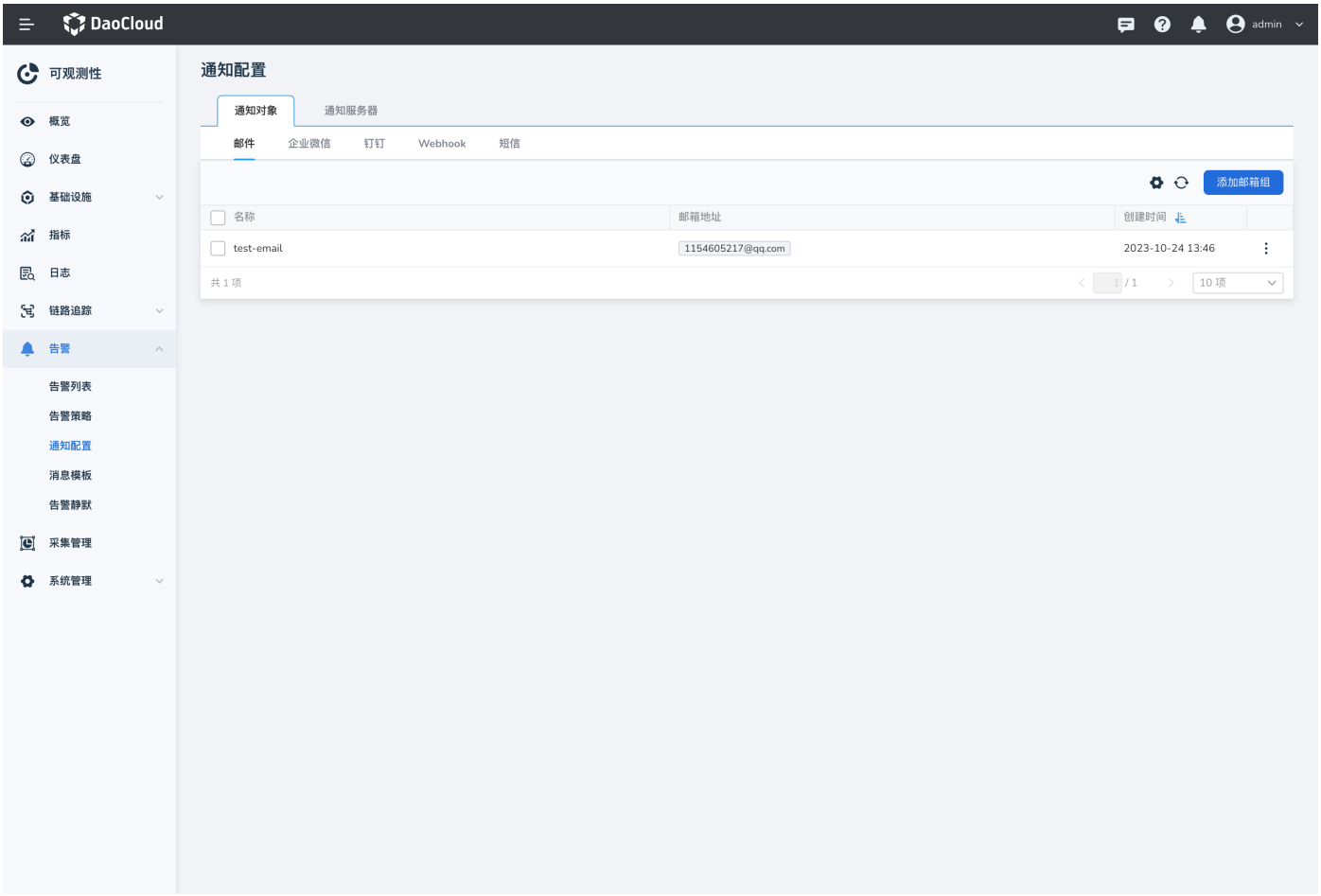

DaoCloud
admin
可观测性
概览
仪表盘
基础设施
指标
日志
链路追踪
告警
告警列表
告警策略
通知配置
消息模板
告警降噪
采集管理
系统管理
告警策略
告警策略
告警策略
内置告警策略
搜索
创建告警模板
模板名称
创建时间
test-sts
2024-01-29 13:59:50
benchmark-rule-template
集群
1
for benchmark
2024-01-29 11:19:41
共 2 项
1 / 1
10 项
删除告警模板
删除告警模板会同时删除告警模板中的全部规则。
请输入benchmark-rule-template
删除
取消

通知配置

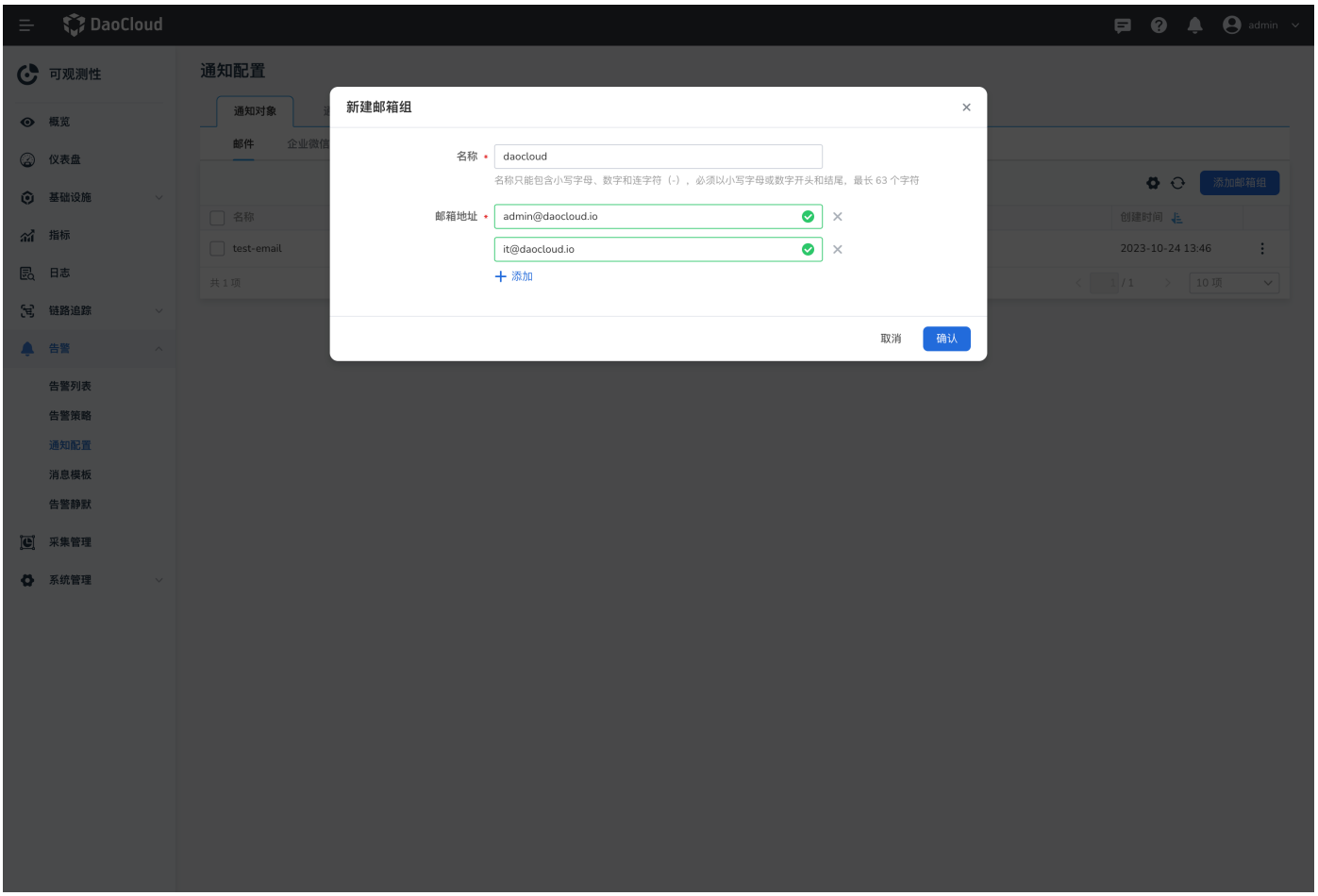
在 通知配置 页面，可以配置通过邮件、企业微信、钉钉、Webhook 和短信等方式向用户发送消息。

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。



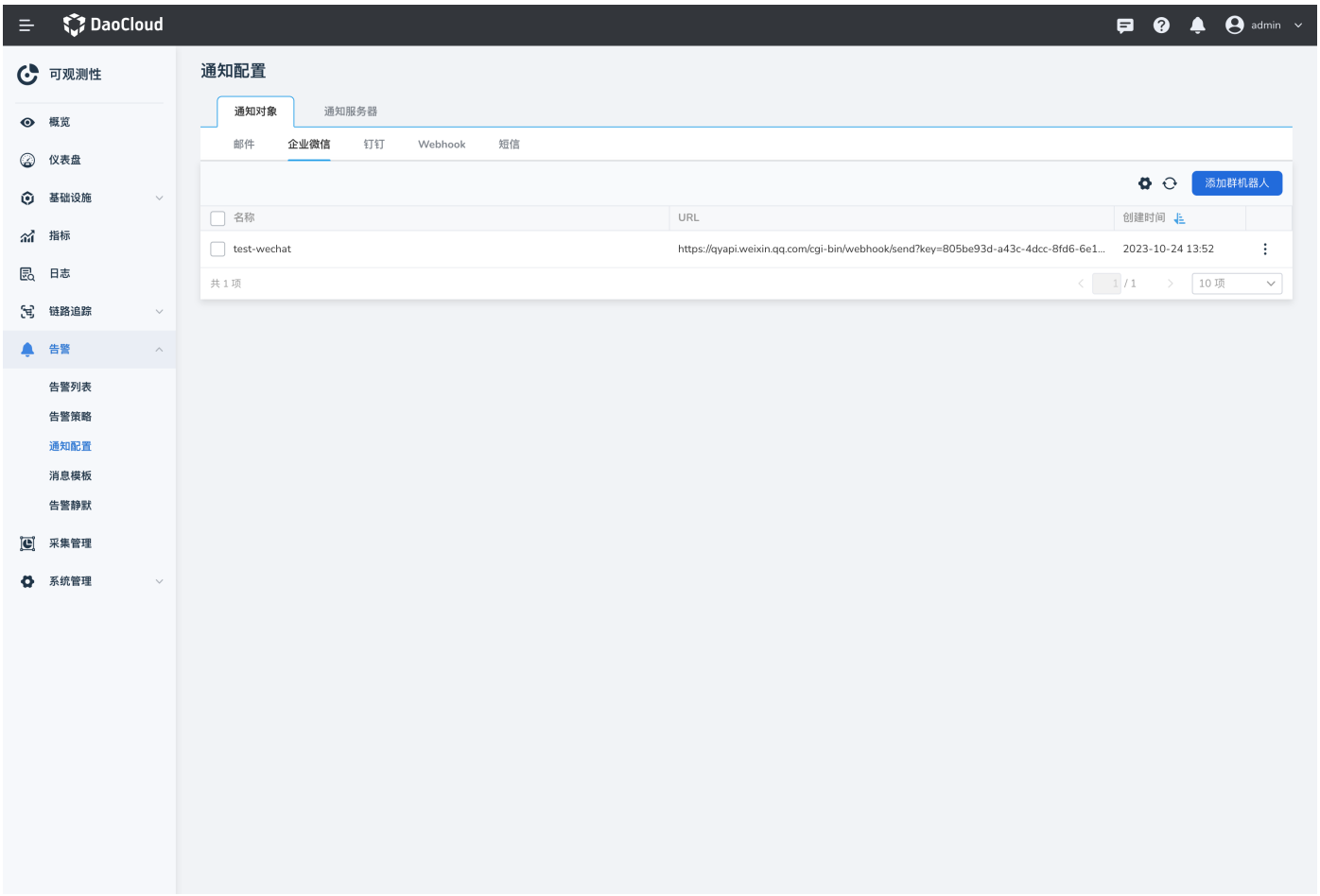
2. 点击 添加邮箱组 ，添加一个或多个邮件地址。



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。

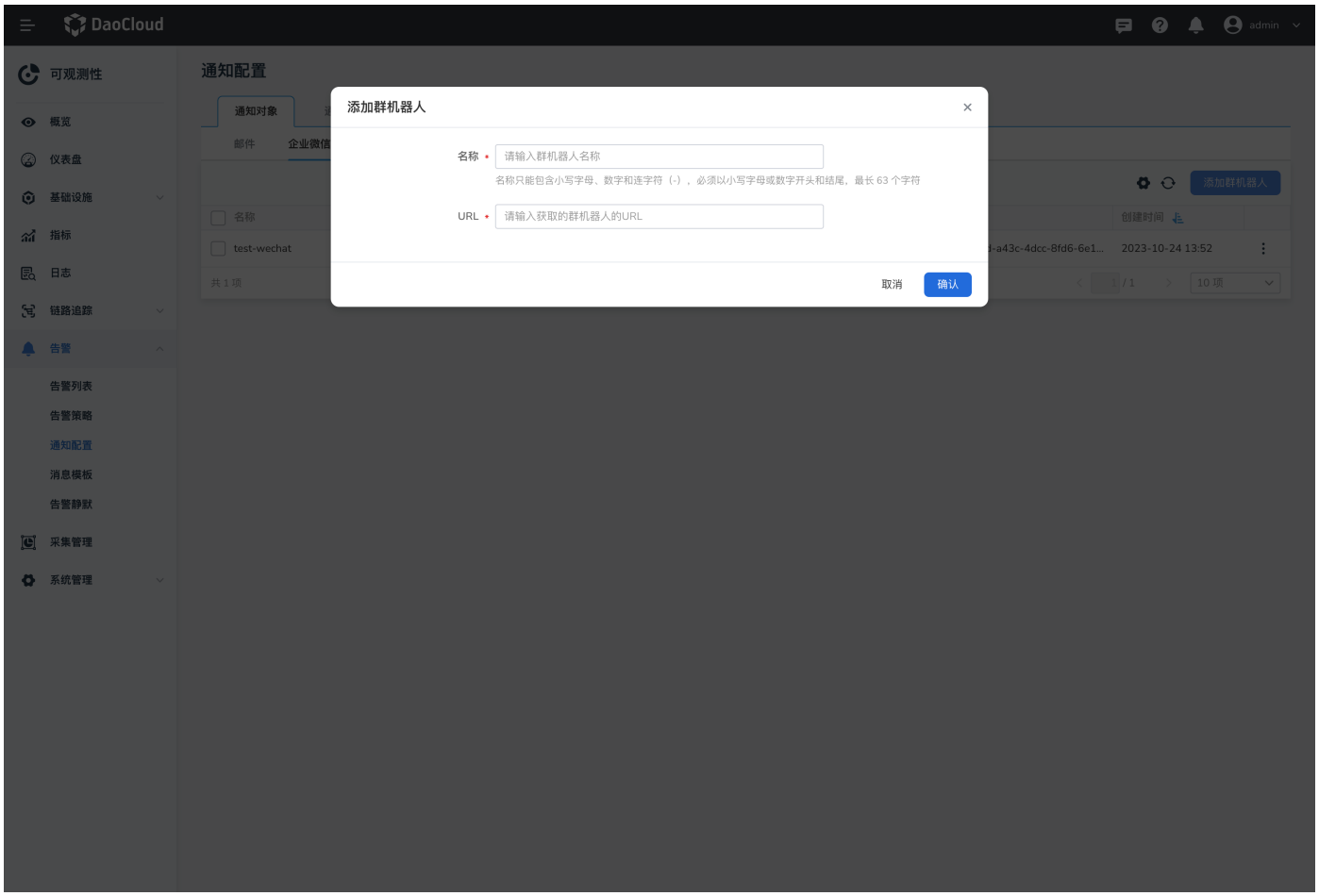
企业微信

1. 在左侧导航栏中点击 告警中心 -> 通知配置 -> 企业微信 。



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。

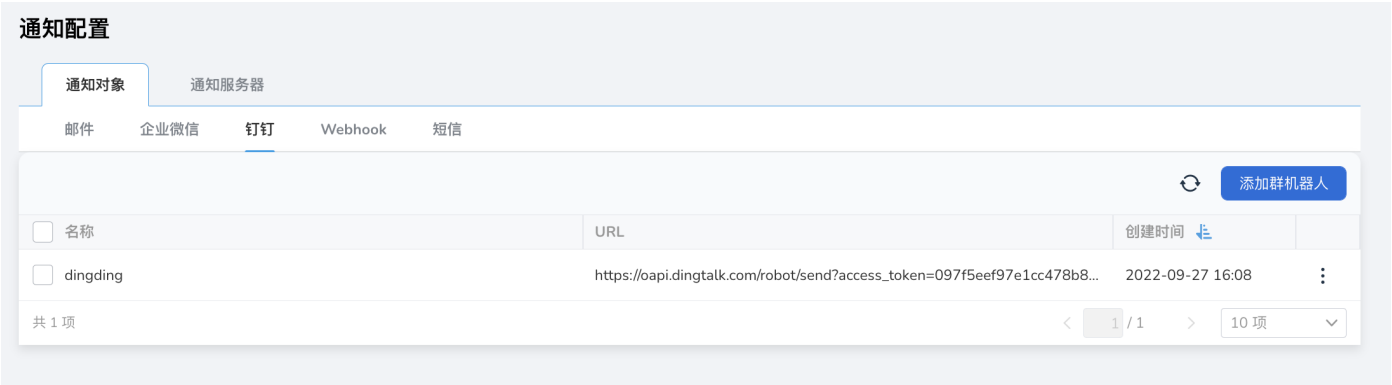
2. 点击 添加群机器人 ，添加一个或多个群机器人。



3. 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。

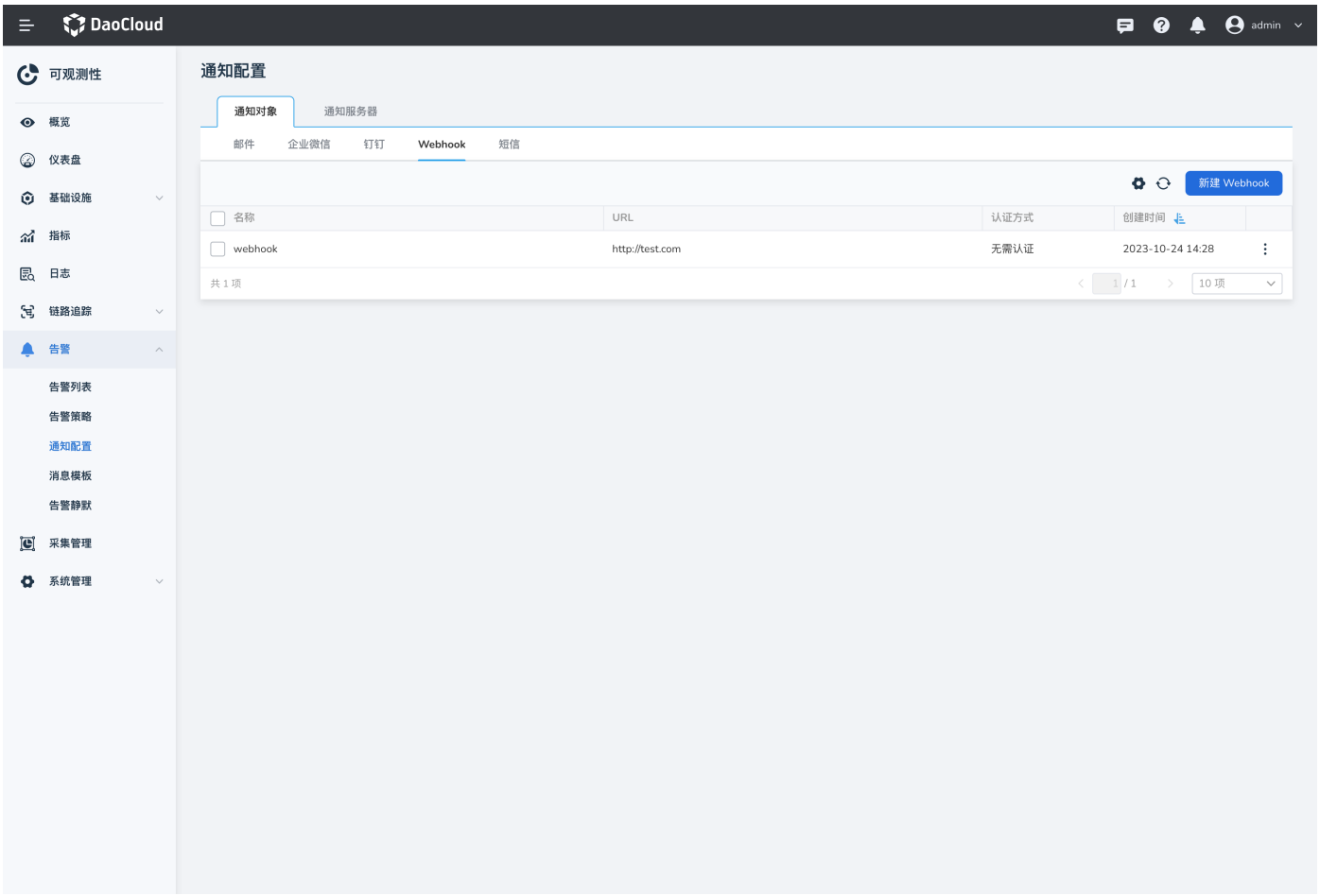


有关钉钉群机器人的 URL，请参阅钉钉官方文档：自定义机器人接入。

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。

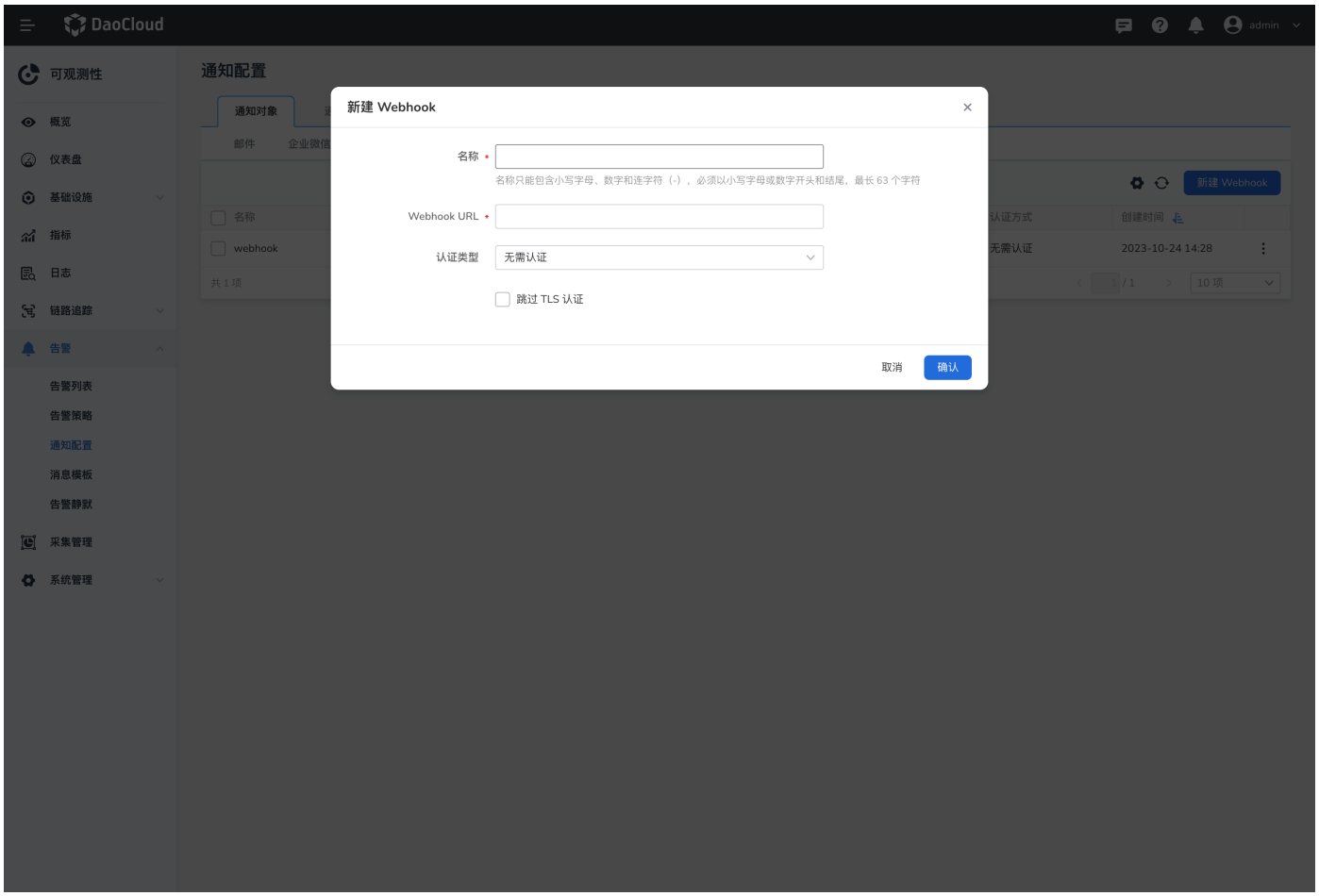
Webhook

1. 在左侧导航栏中点击 告警中心 -> 通知配置 -> **Webhook** 。



有关 Webhook URL 及更多配置方式，请参阅 webhook 文档。

2. 点击 新建 **Webhook** ， 添加一个或多个 Webhook。

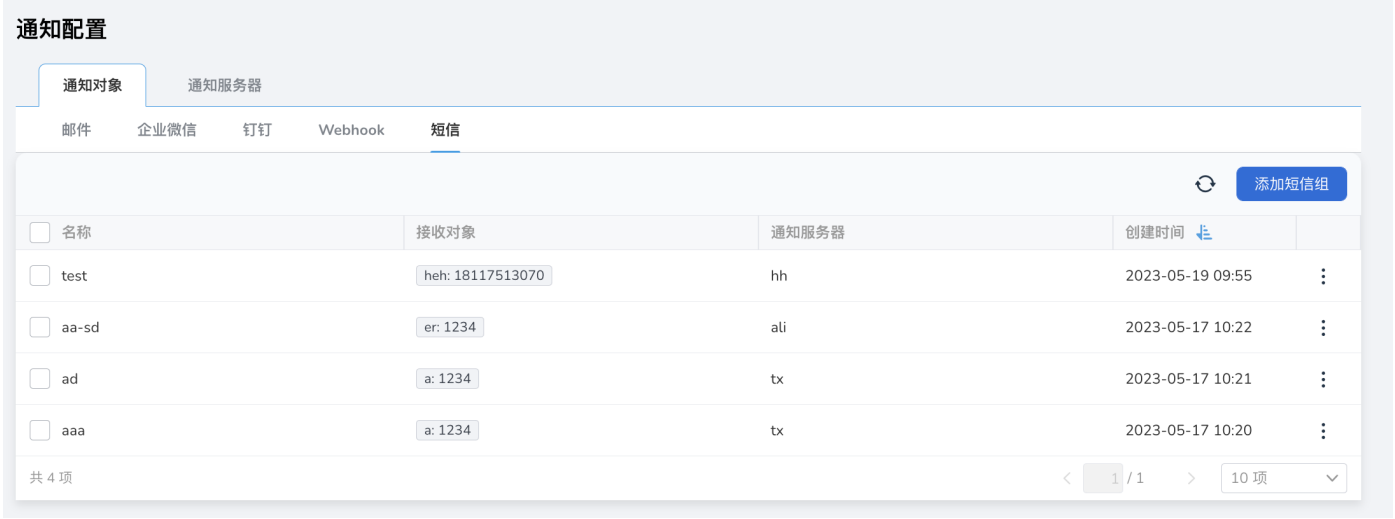


有关 Webhook URL 及更多配置方式，请参阅 webhook 文档。

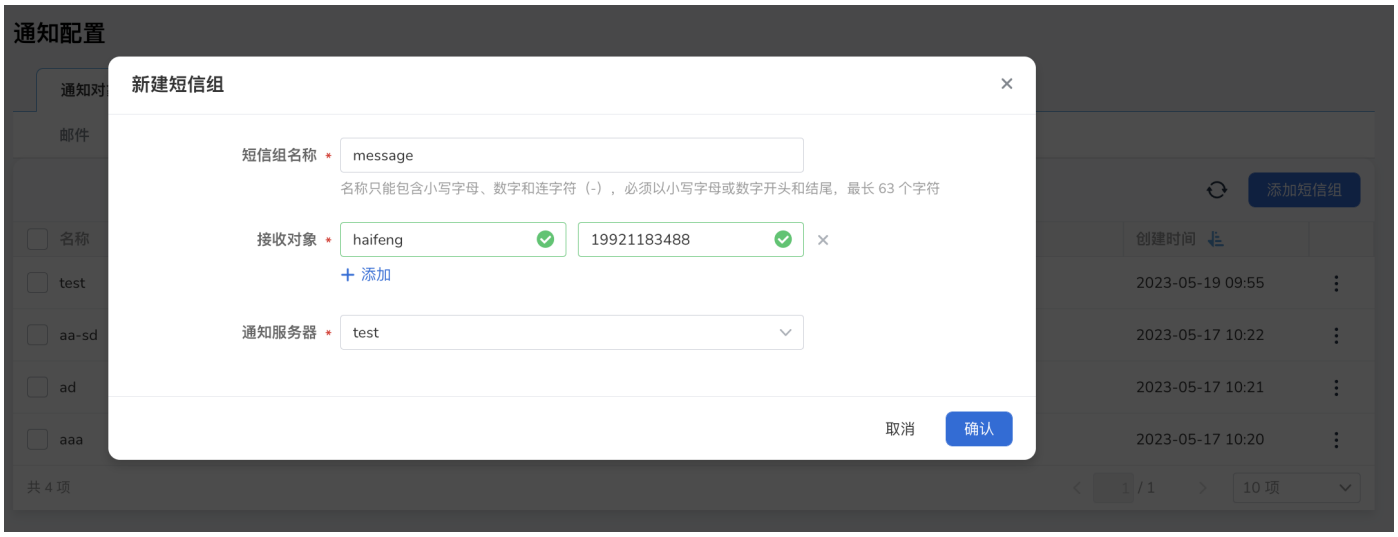
3. 配置完成后自动返回通知列表，点击列表右侧的 ⋮ ，选择 发送测试信息 ，还可以编辑或删除 Webhook。

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。



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。



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。

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。

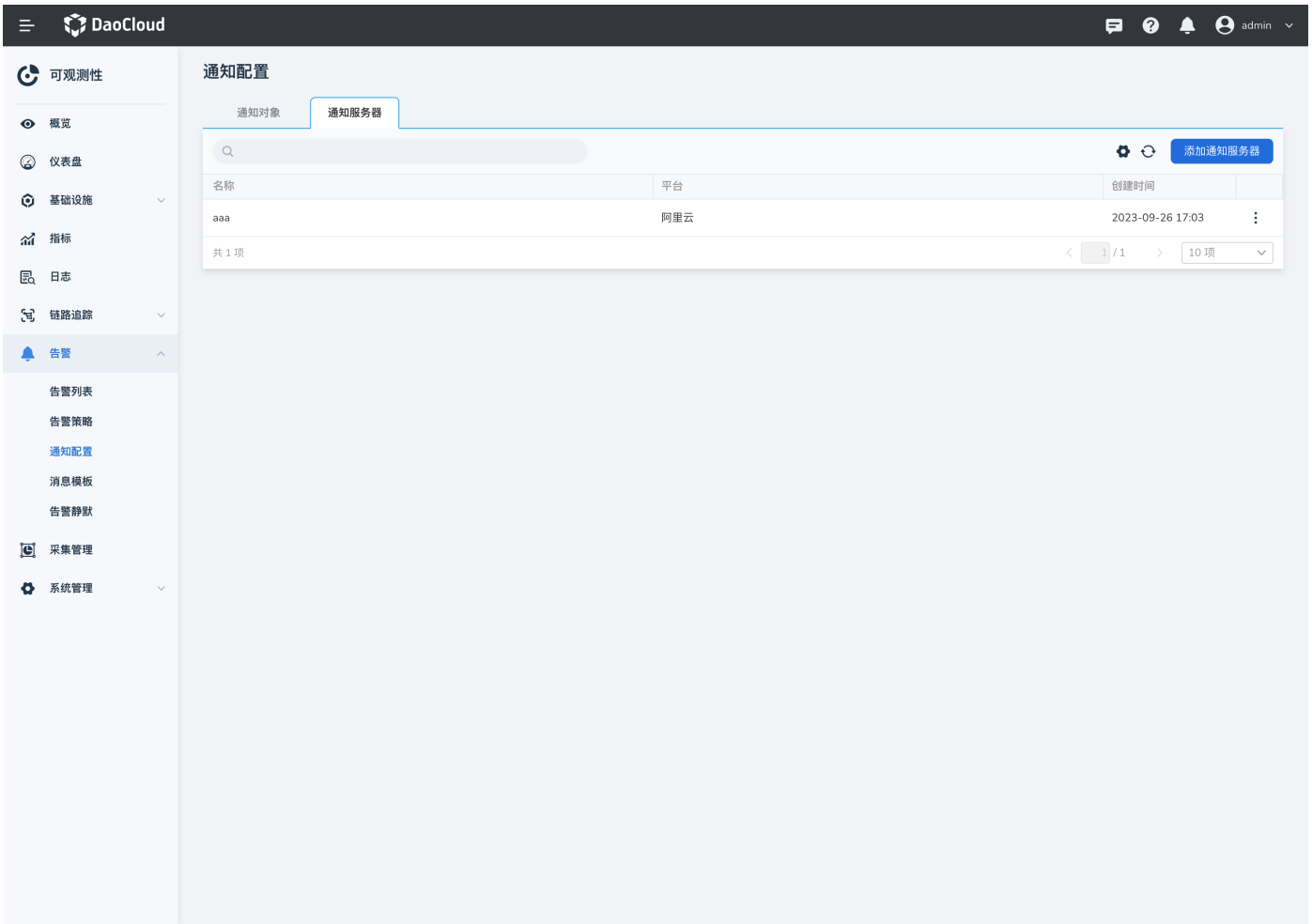
配置通知服务器

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 。



2. 点击 添加通知服务器 。

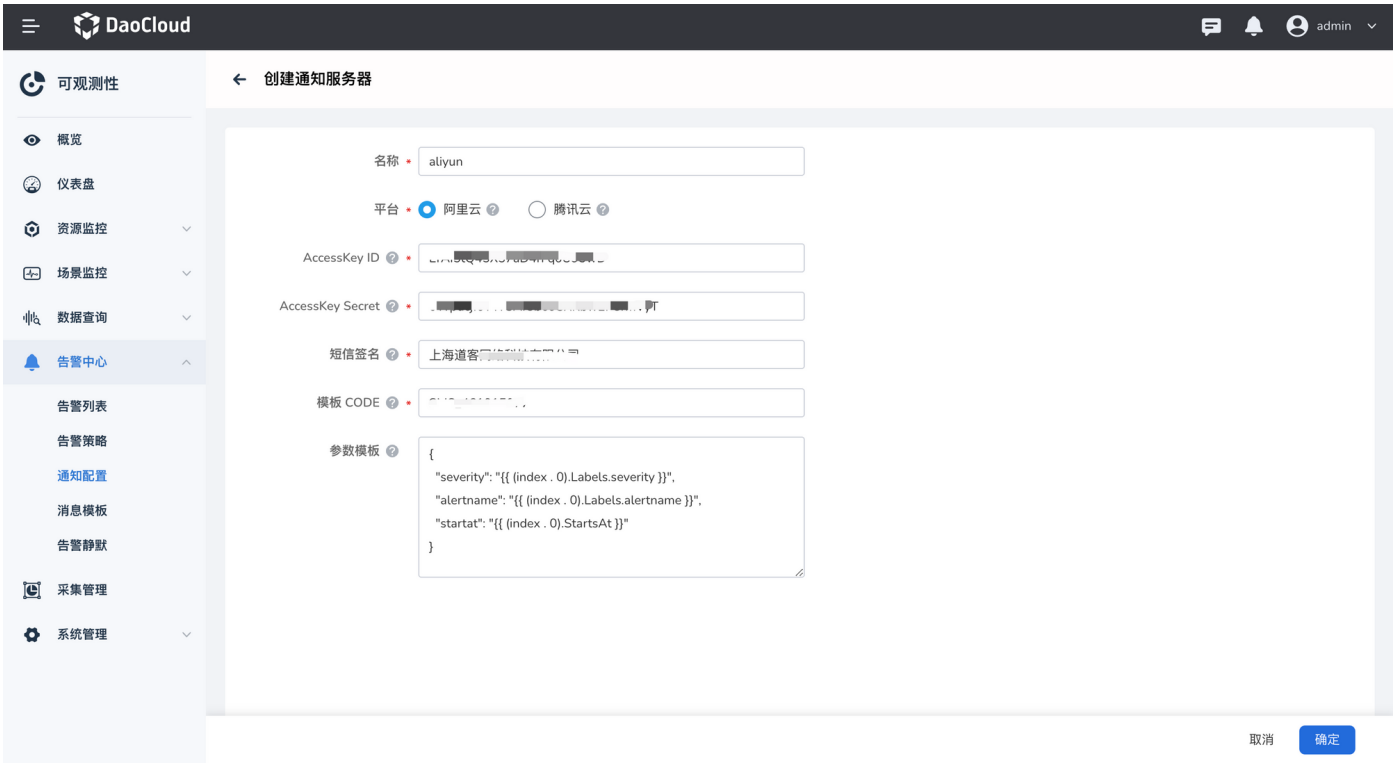
a. 配置阿里云服务器。

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。



Note

举例：在阿里云定义的模板内容为：\({severity}：\) 被触发。参数模板中的配置参考上图。} 在 ${startat

b. 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。

Note

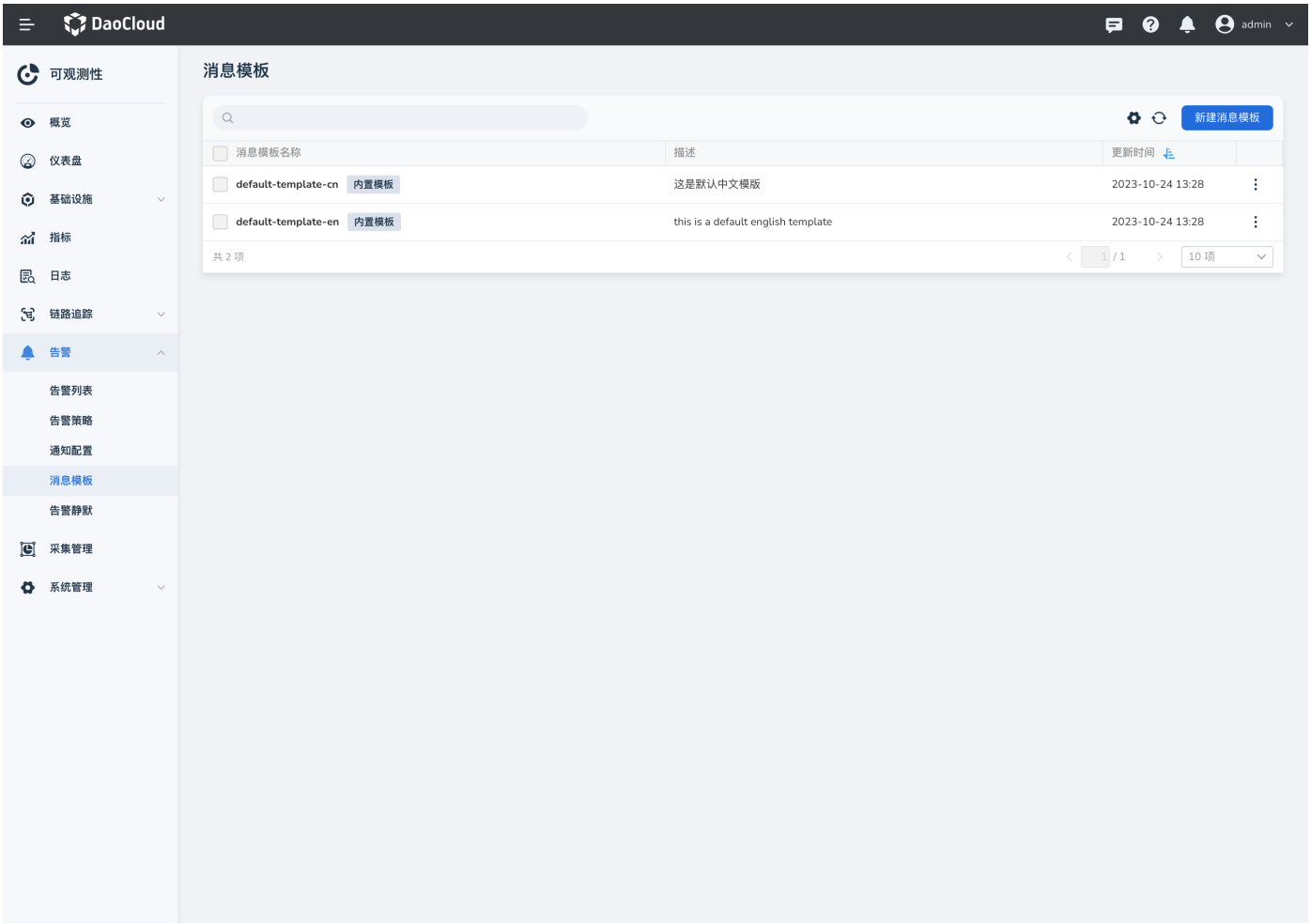
举例：在腾讯云定义的模板内容为：{1}：{2} 在 {3} 被触发。参数模板中的配置参考上图。

消息模板

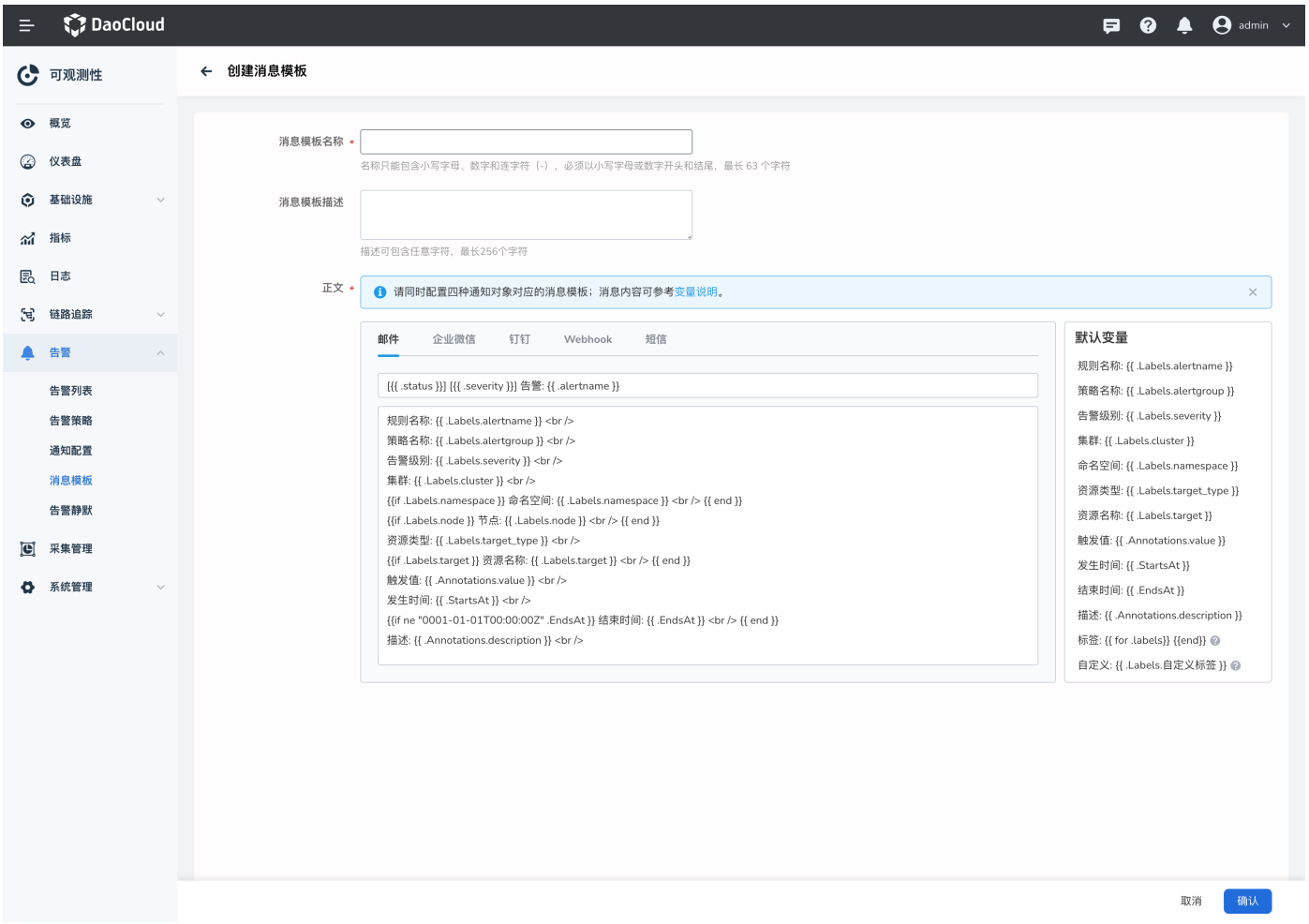
可观测性提供自定义消息模板内容的能力，支持邮件、企业微信、钉钉、Webhook 等不同的通知对象定义不同的消息通知内容。

创建消息模板

1. 在左侧导航栏中，选择 告警中心 -> 消息模板 。
2. Insight 默认内置中英文两个模板，以便用户使用。



3. 点击 新建消息模板 按钮，填写模板内容。



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

参数说明

参数	变量	描述
规则名称	{{ .Labels.alertname }}	触发告警的规则名称
策略名称	{{ .Labels.alertgroup }}	触发告警规则所属的告警策略名称
告警级别	{{ .Labels.severity }}	触发告警的级别
集群	{{ .Labels.cluster }}	触发告警的资源所在的集群
命名空间	{{ .Labels.namespace }}	触发告警的资源所在的命名空间
节点	{{ .Labels.node }}	触发告警的资源所在的节点
资源类型	{{ .Labels.target_type }}	告警对象的资源类型
资源名称	{{ .Labels.target }}	触发告警的对象名称
触发值	{{ .Annotations.value }}	触发告警通知时的指标值
发生时间	{{ .StartsAt }}	告警开始发生的时间
结束时间	{{ .EndsAT }}	告警结束的时间
描述	{{ .Annotations.description }}	告警的详细描述
标签	{{ for .labels}} {{end}}	告警的所有标签，使用 for 函数遍历 labels 列表，获取告警的所有标签内容。

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。

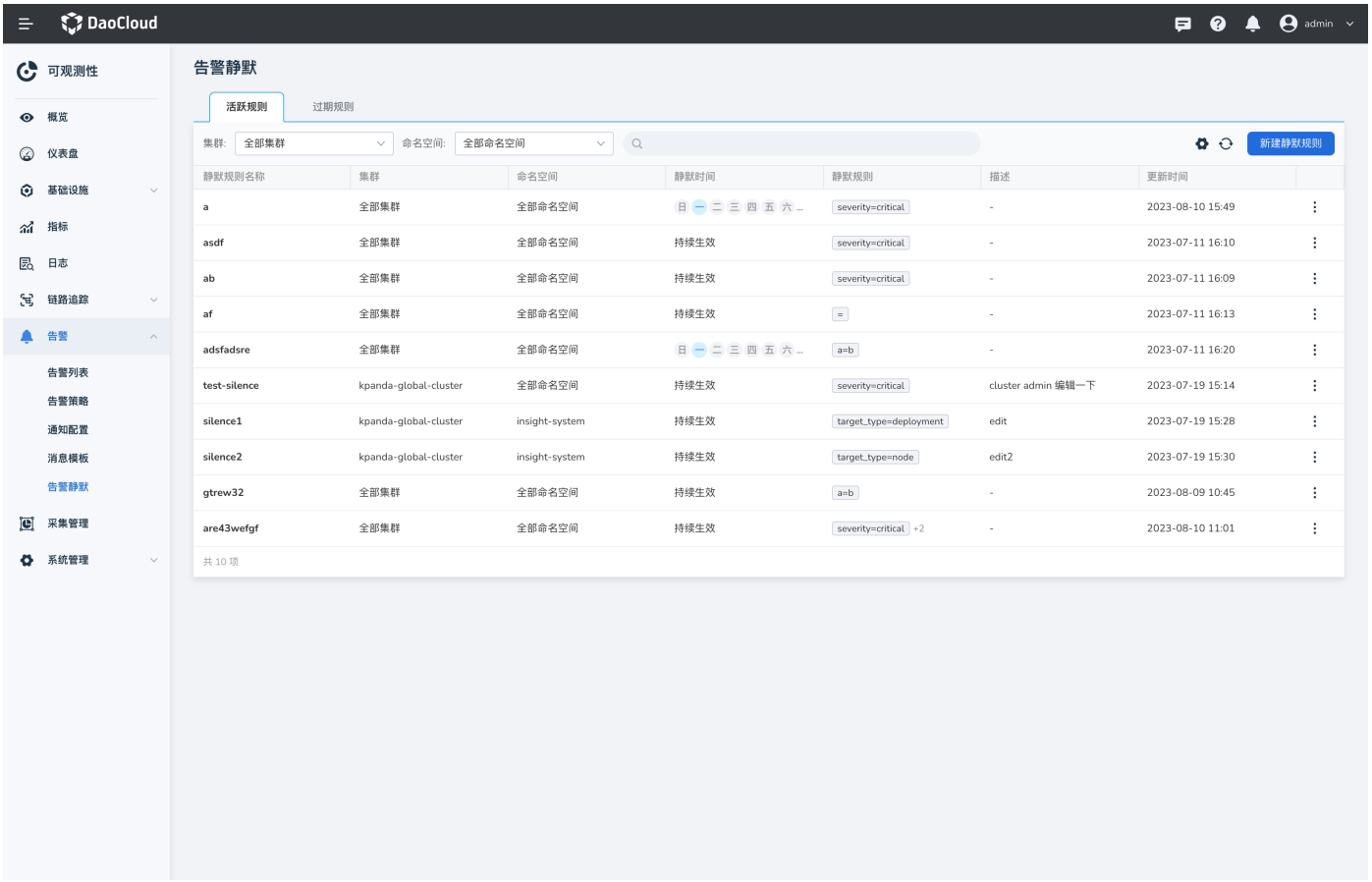
告警静默

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。该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。

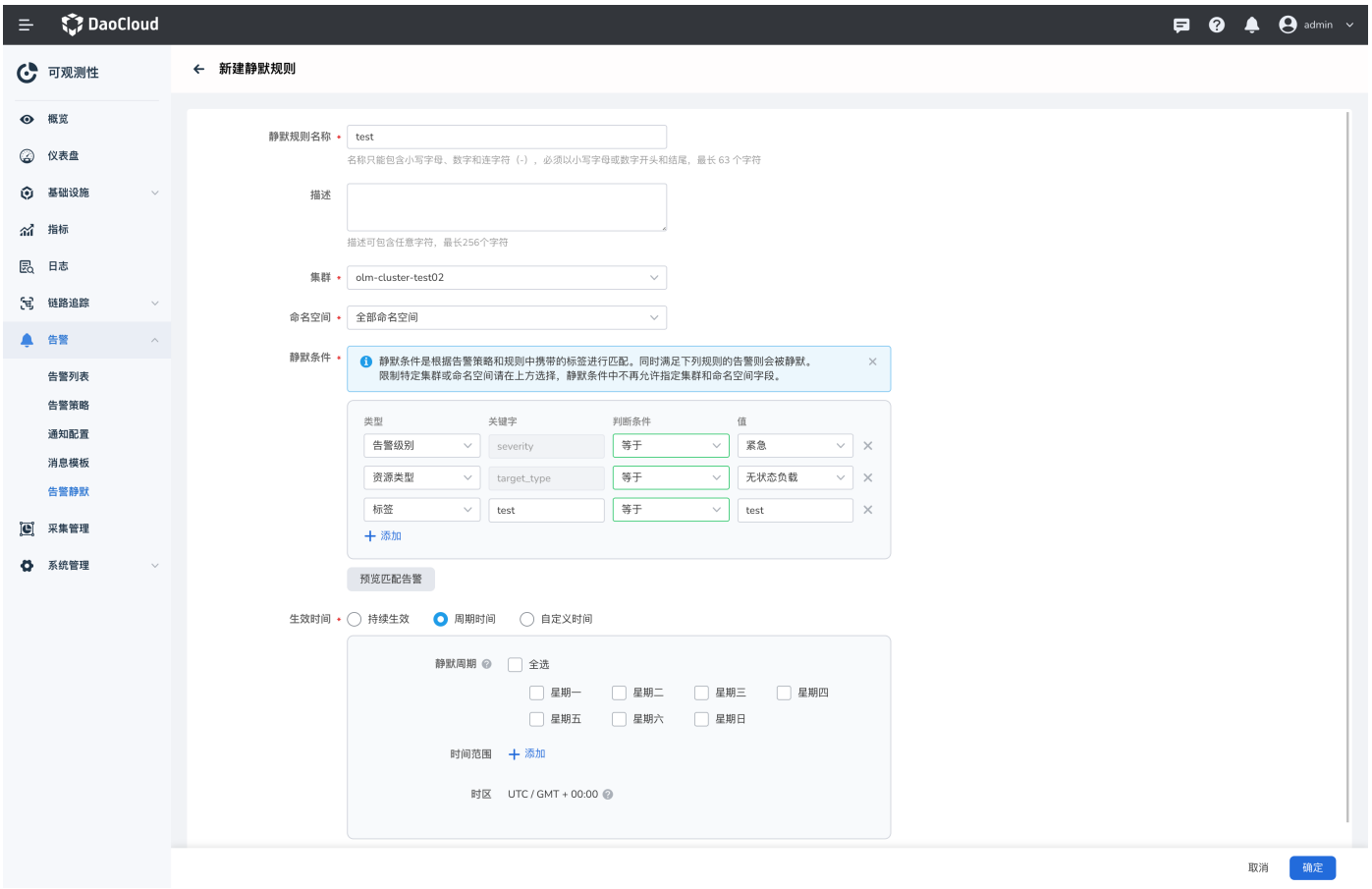
在告警静默页面上，用户可以看到两个页签：活跃规则和过期规则。 其中，活跃规则表示目前正在生效的规则，而过期规则则是以前定义过但已经过期（或者用户主动删除）的规则。

操作步骤

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。



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。



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。

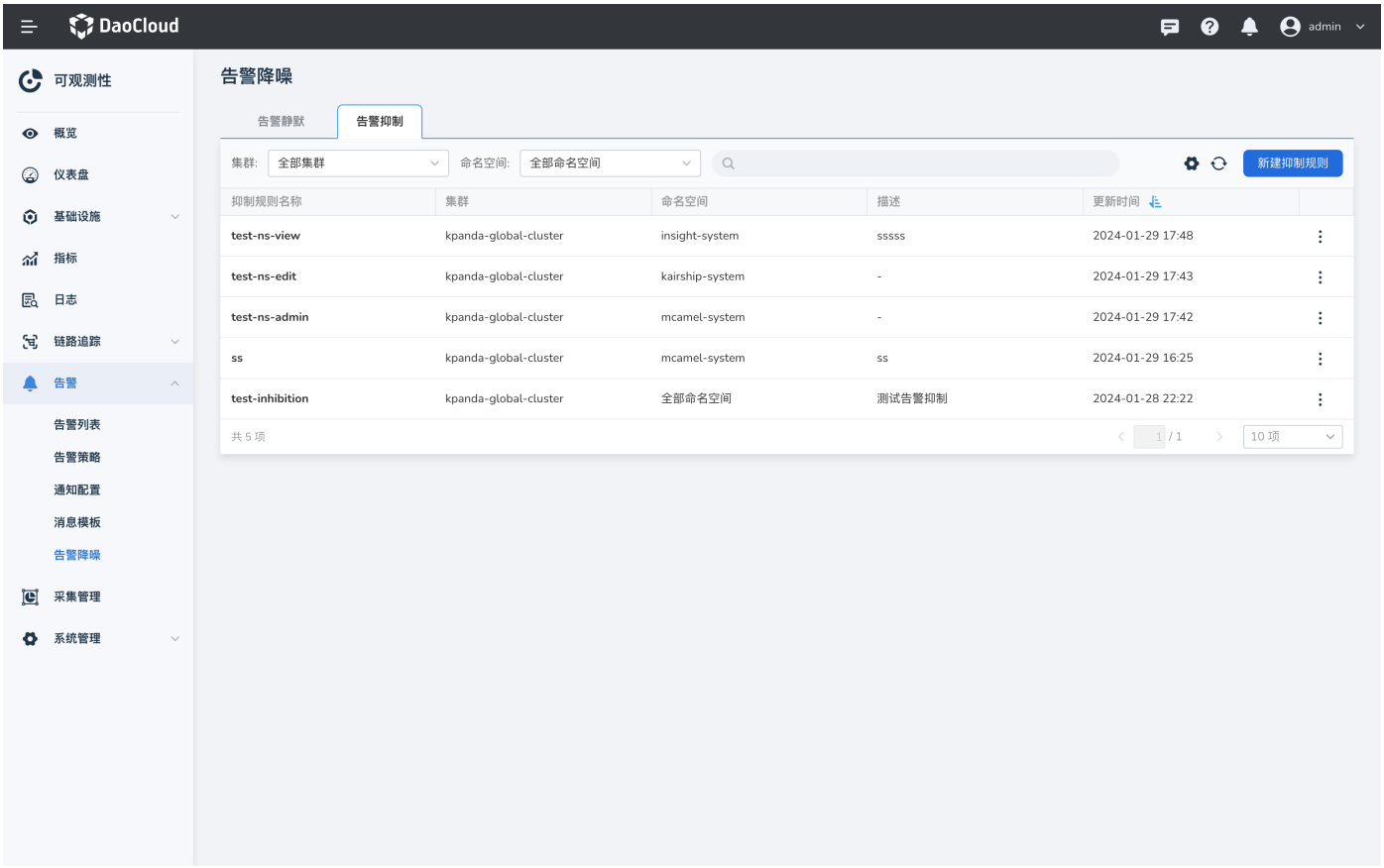
通过告警静默功能，您可以灵活地控制哪些告警需要被忽略，在什么时间段内生效，从而提高运维效率，减少误报的可能性。

告警抑制

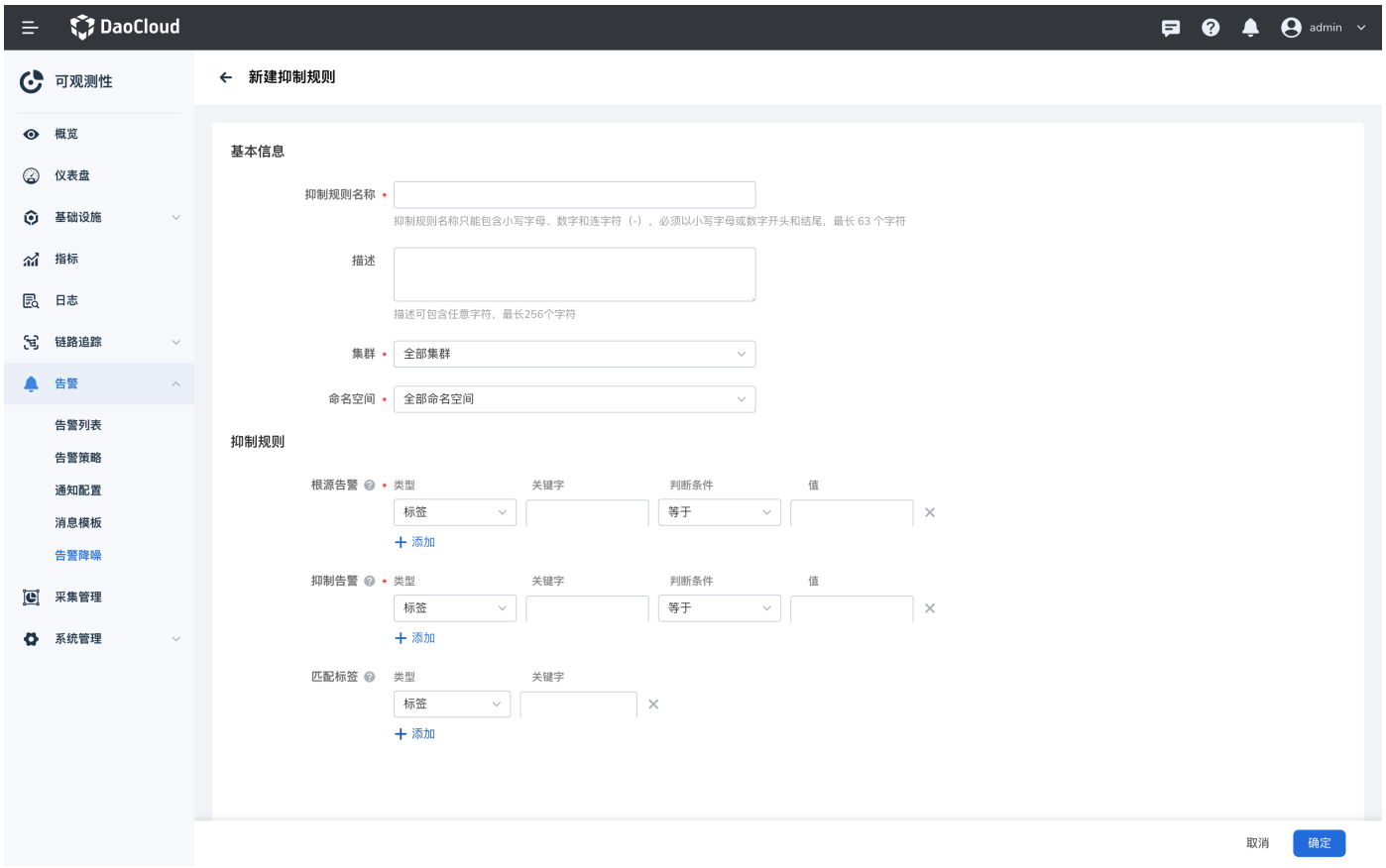
告警抑制主要是对于某些不需要立即关注的告警进行临时隐藏或者降低其优先级的一种机制。这个功能的目的是为了减少不必要的告警信息对运维人员的干扰，使他们能够集中精力处理更重要的问题。

创建抑制规则

1. 左侧导航栏中，选择 告警中心 -> 告警降噪 ，单击顶部的 告警抑制 。



2. 点击 新建抑制规则 ，设置抑制规则的名称、规则等。

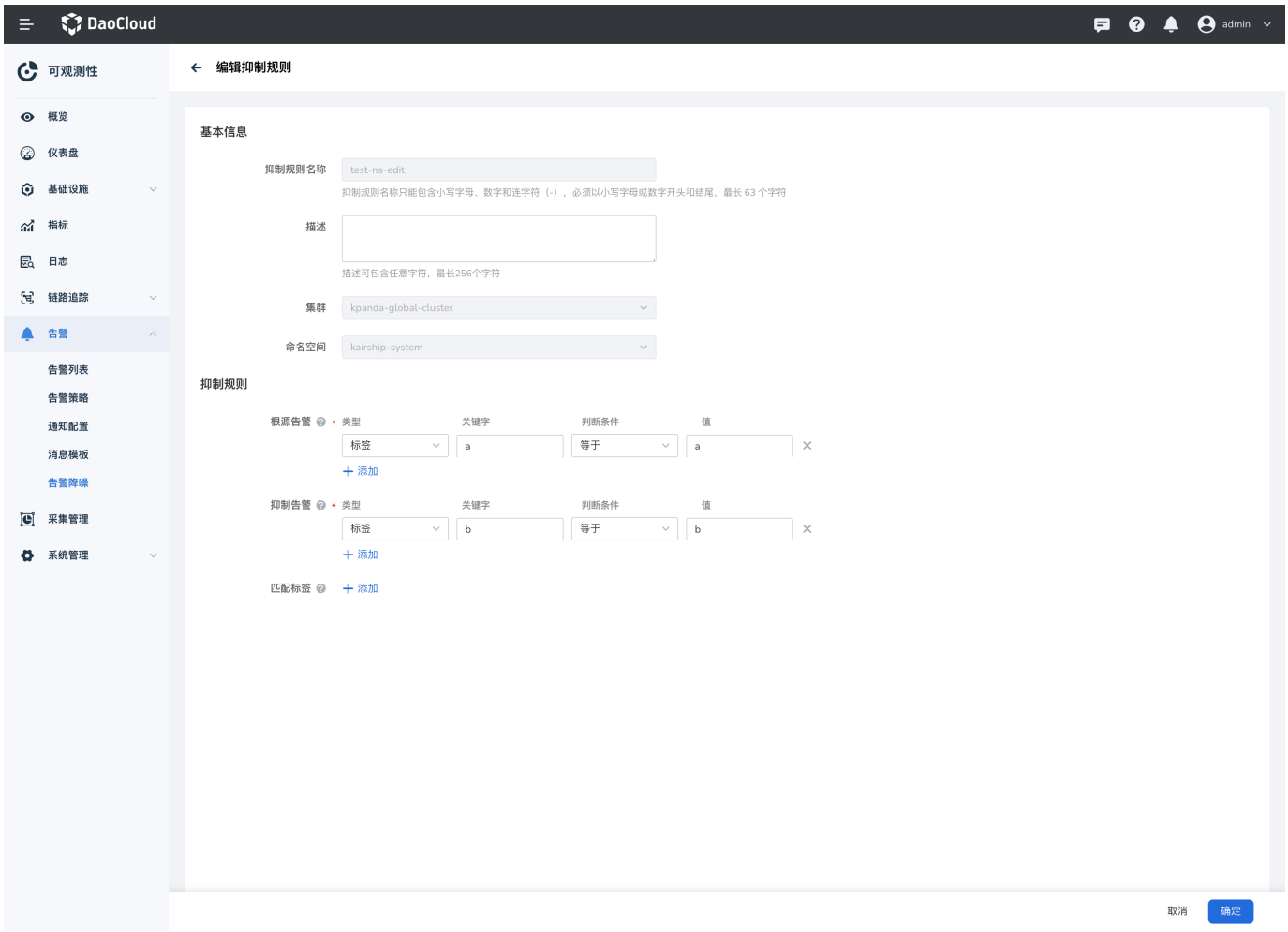


参数时间	说明
抑制规则名称	抑制规则名称只能包含小写字母、数字和连字符（-），必须以小写字母或数字开头和结尾，最长 63 个字符。
描述	描述可包含任意字符，最长 256 个字符。
集群	该抑制规则作用的集群。
命名空间	该抑制规则作用的命名空间。
根源告警	用于指定源警报（触发抑制的警报）的匹配条件。 取值范围说明： - 告警级别：指标或事件告警的级别，可以设置为：紧急、重要、提示。 - 资源类型：告警对象所对应的资源类型，可以设置为：集群、节点、无状态负载、有状容负载、守护进程、容器组。 - 标签：告警标识属性，由标签名和标签值构成，支持用户自定义。
抑制告警	用于指定目标警报（将被抑制的警报）的匹配条件。
匹配标签	用于指定应该比较的标签列表，以确定源警报和目标警报是否匹配。只有在 equal 中指定的标签在源和目标警报中的值完全相同的情况下，才会触发抑制。equal 字段是可选的。如果省略 equal 字段，则会将所有标签用于匹配

3. 点击 确定 完成创建后返回告警抑制列表，点击告警抑制名称后可查看抑制规则详情。

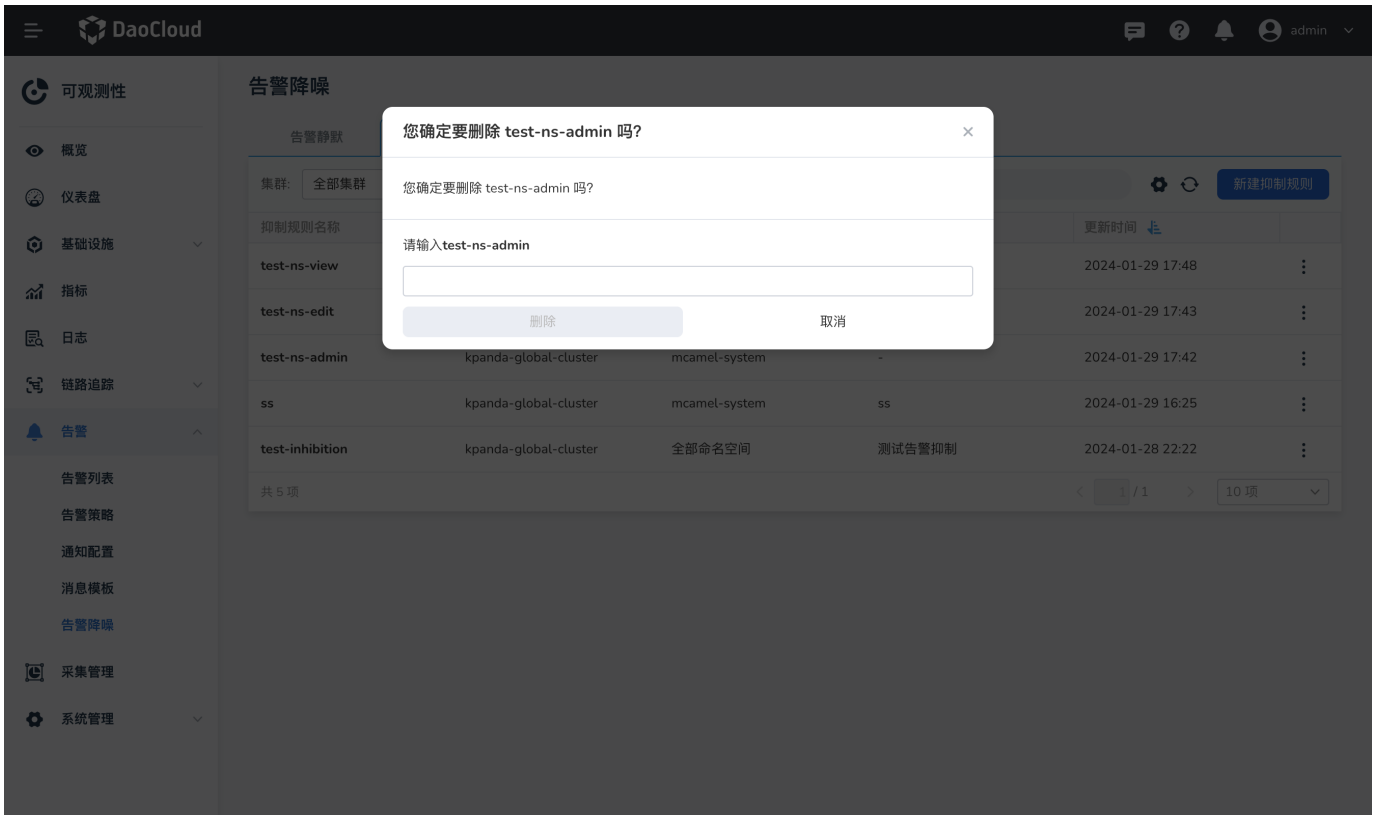
编辑抑制规则

点击目标规则后侧的 ⋮ ，点击 编辑 ，进入抑制规则的编辑页。



删除抑制规则

点击目标规则后侧的 ┆ ，点击 删除 ，在输入框中输入抑制规则的名称即可删除。


DaoCloud
admin
可观测性
概览
仪表盘
基础设施
指标
日志
链路追踪
告警
告警列表
告警策略
通知配置
消息模板
告警降噪
采集管理
系统管理
告警降噪
告警静默
集群:
全部集群
新建抑制规则
抑制规则名称
更新时间
test-ns-view
2024-01-29 17:48
test-ns-edit
2024-01-29 17:43
test-ns-admin
kpanda-global-cluster
mcamel-system
2024-01-29 17:42
ss
kpanda-global-cluster
mcamel-system
ss
2024-01-29 16:25
test-inhibition
kpanda-global-cluster
全部命名空间
测试告警抑制
2024-01-28 22:22
共 5 项
1 / 1
10 项
您确定要删除 test-ns-admin 吗?
您确定要删除 test-ns-admin 吗?
请输入test-ns-admin
删除
取消

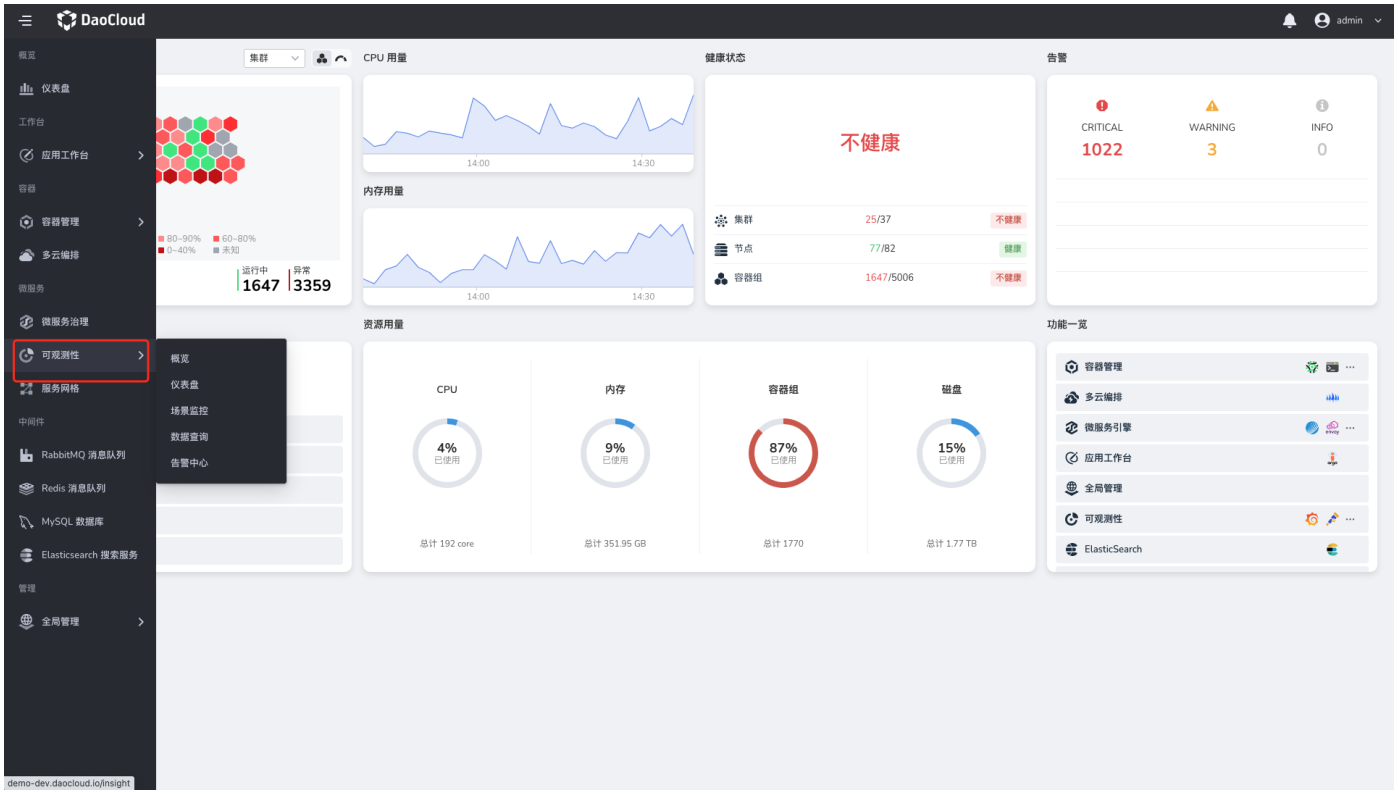
采集管理

采集管理

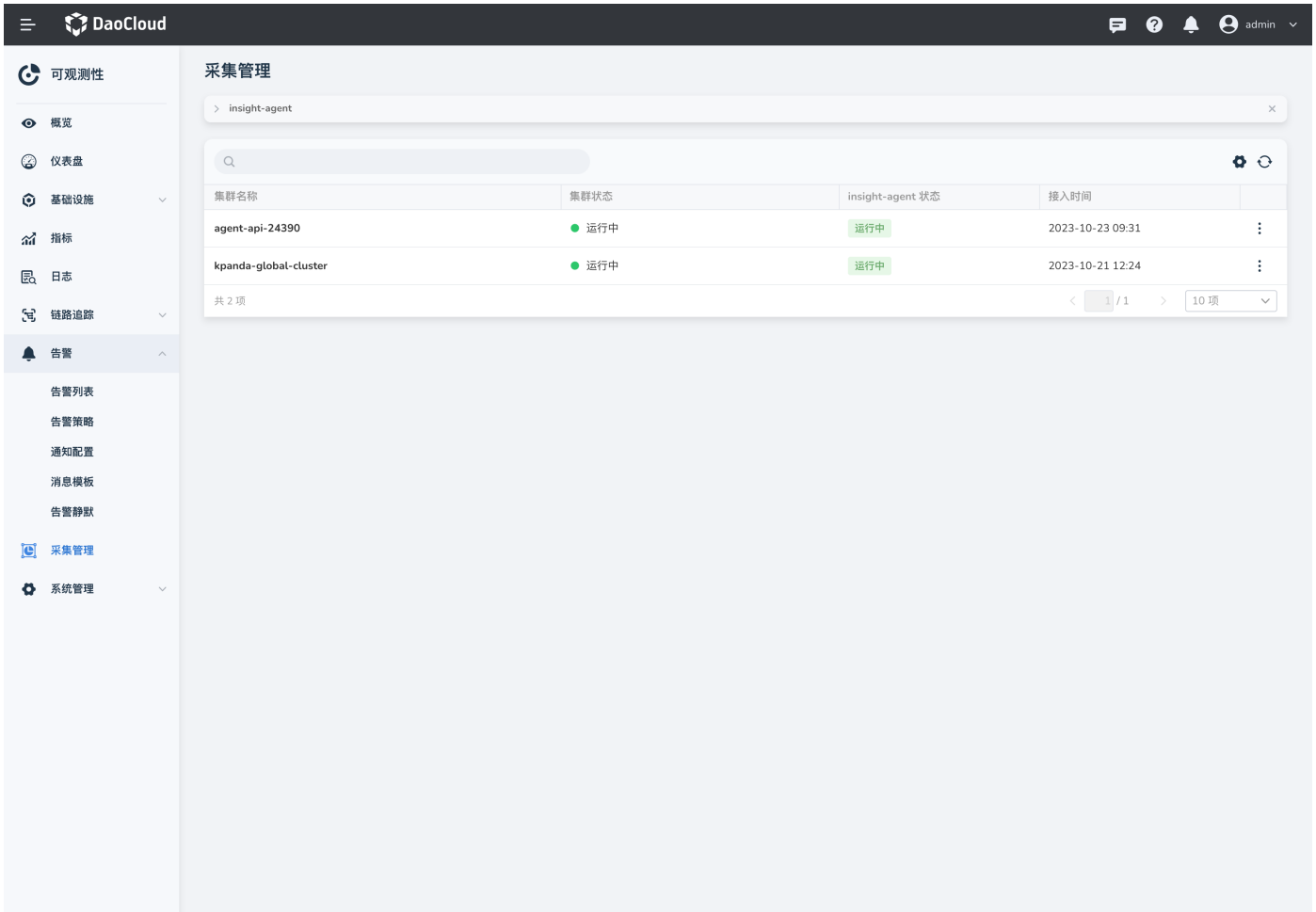
采集管理 主要是集中管理、展示集群安装采集插件 **insight-agent** 的入口，帮助用户快速的查看集群采集插件的健康状态，并提供了快捷入口配置采集规则。

具体操作步骤如下：

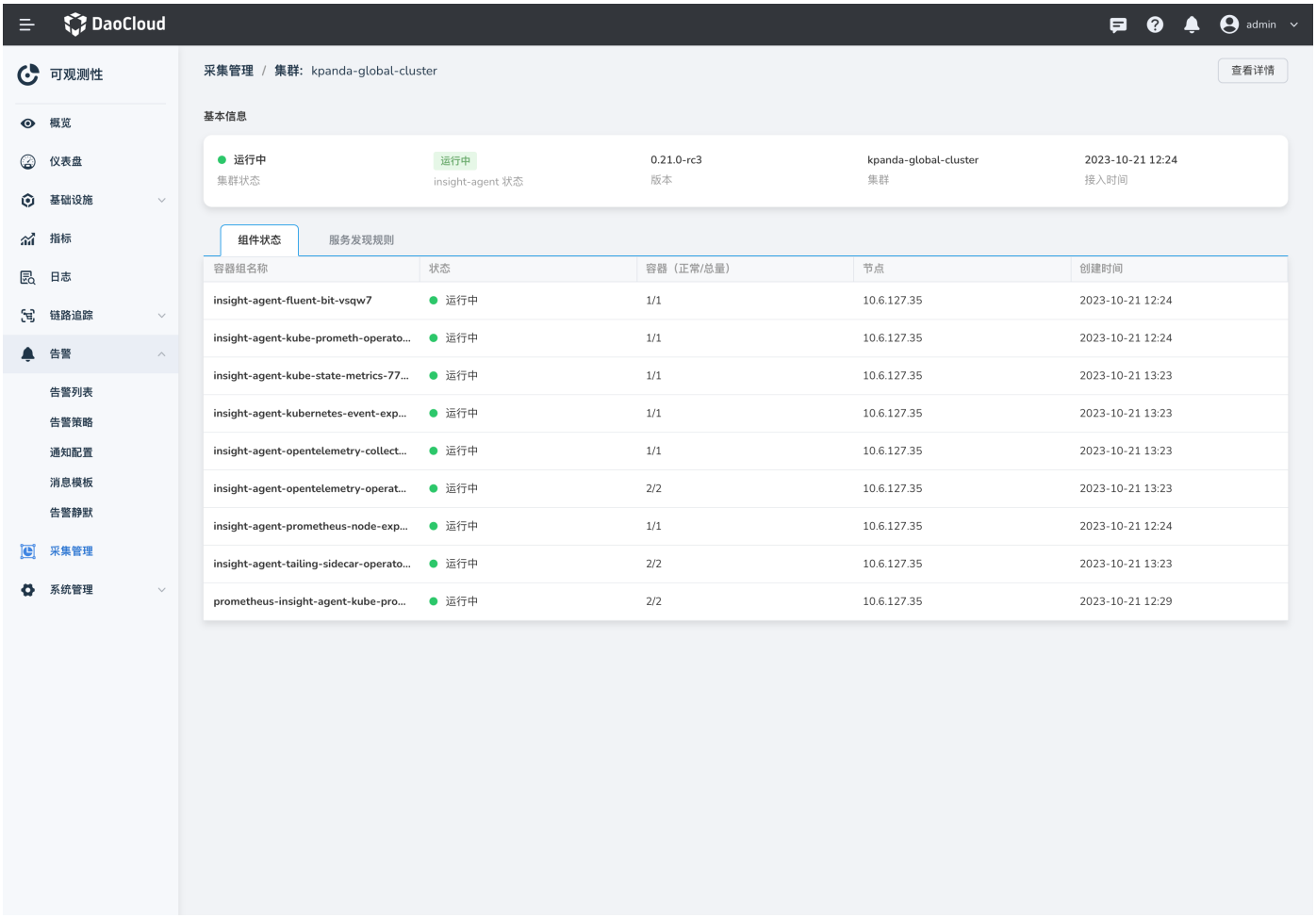
1. 点击左上角的，选择 可观测性 。



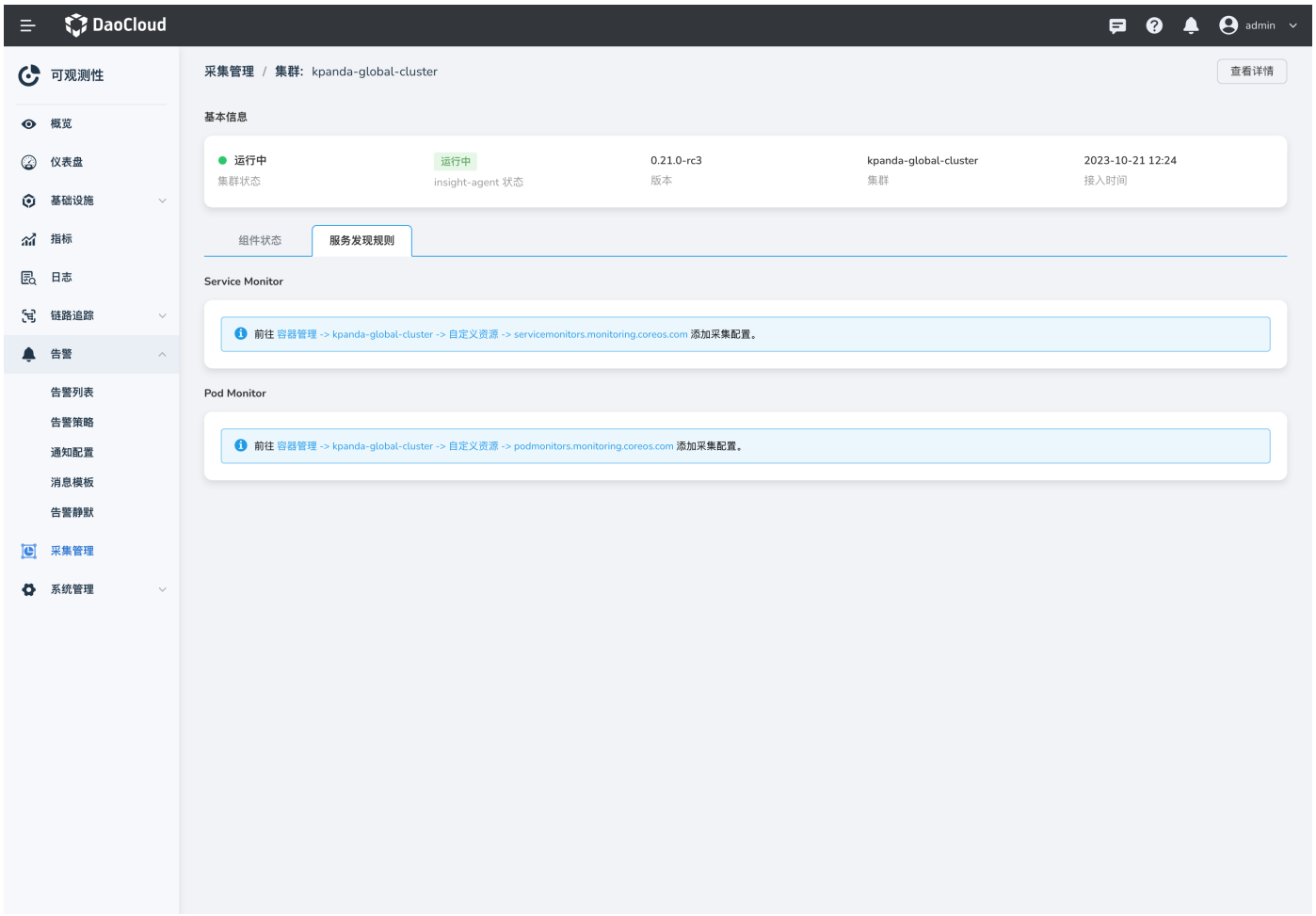
2. 选择左侧导航栏的 采集管理 ， 查看全部集群采集插件的状态。



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。



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。


DaoCloud
admin
可观测性
概览
仪表盘
基础设施
指标
日志
链路追踪
告警
告警列表
告警策略
通知配置
消息模板
告警静默
采集管理
系统管理
采集管理 / 集群: kpanda-global-cluster
查看详情
基本信息
运行中
集群状态
运行中
insight-agent 状态
0.21.0-rc3
版本
kpanda-global-cluster
集群
2023-10-21 12:24
接入时间
组件状态
服务发现规则
Service Monitor
前往 容器管理 -> kpanda-global-cluster -> 自定义资源 -> servicemonitors.monitoring.coreos.com 添加采集配置。
Pod Monitor
前往 容器管理 -> kpanda-global-cluster -> 自定义资源 -> podmonitors.monitoring.coreos.com 添加采集配置。

配置服务发现规则

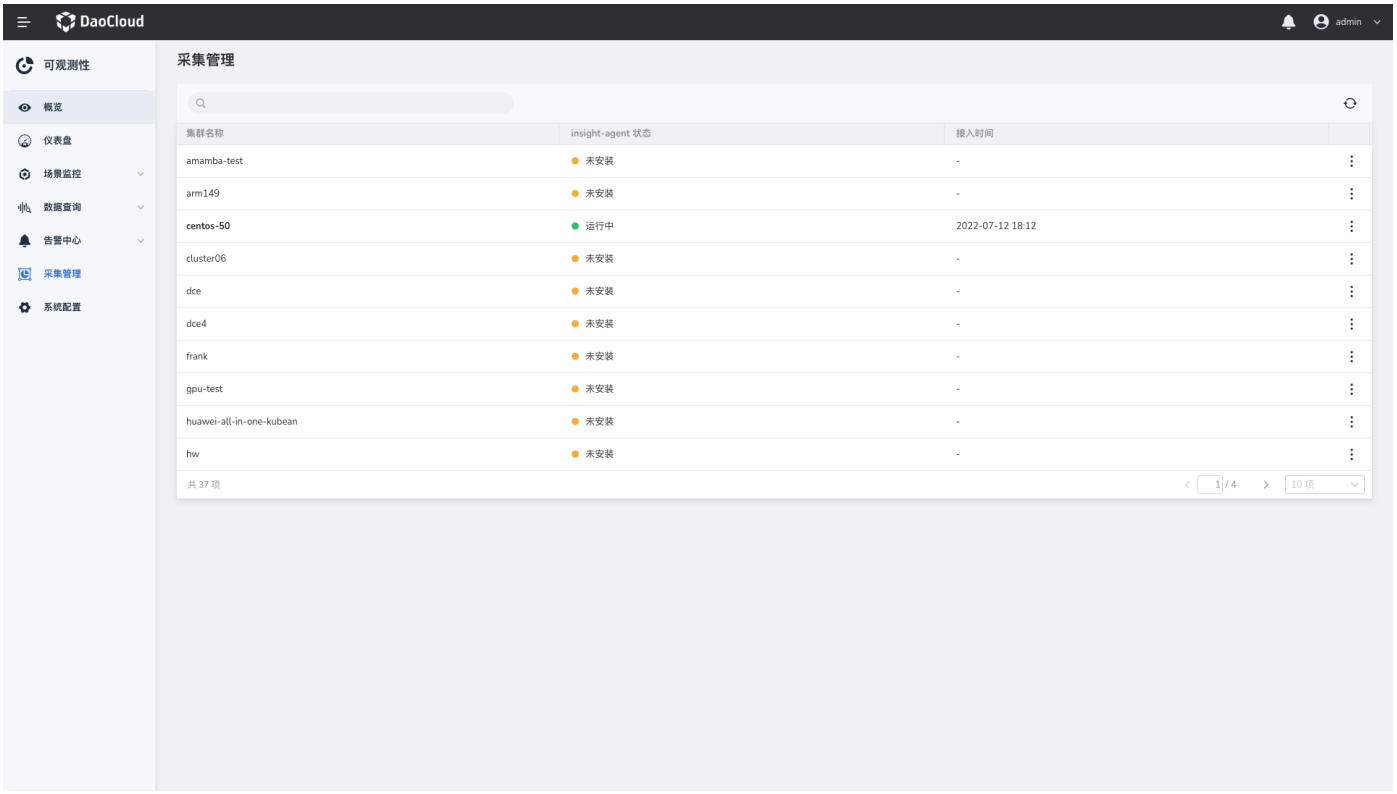
可观测 Insight 支持通过 容器管理 创建 CRD ServiceMonitor 的方式来满足您自定义服务发现的采集需求。 用户可以通过使用 ServiceMonitor 自行定义 Pod 发现的 Namespace 范围以及通过 **matchLabel** 来选择监听的 Service。

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。



2. 点击列表中的某个集群名称进入采集配置详情。

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  # (5)
  selector: # (6)
    matchLabels:
         micrometer-prometheus-discovery: "true"
```

a. 指定 ServiceMonitor 的名称

b. 指定 ServiceMonitor 的命名空间

c. 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** 。

d. 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：

```
namespaceSelector:
  matchNames:
    - default
    - insight-system
```

e. 此处匹配的命名空间为需要暴露指标的应用所在的命名空间

f. 用于选择 Service

INSIGHT-AGENT 组件状态说明

在 d.run 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）。

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
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。

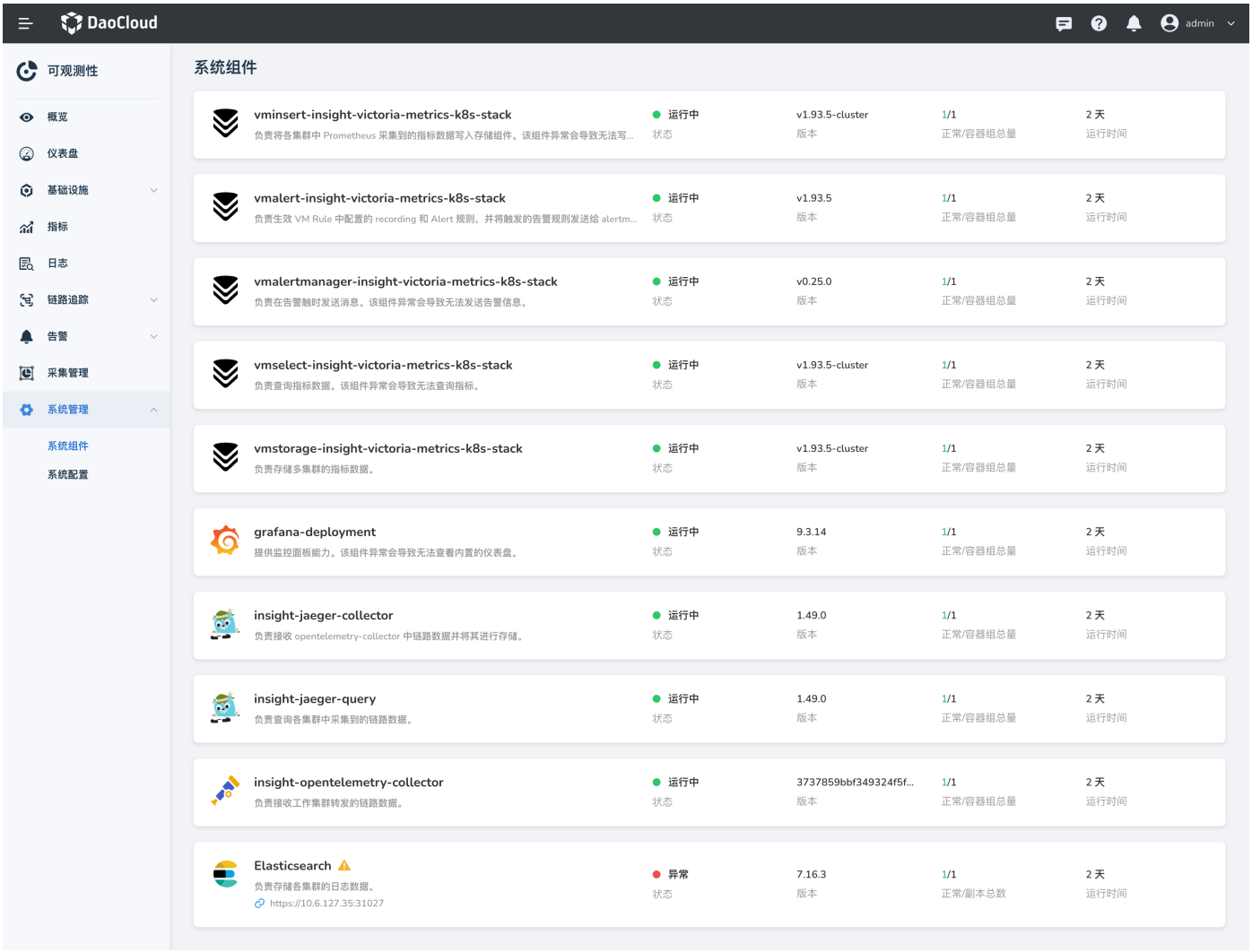
- 请联系平台管理员根据集群规模调整 vmstorage 的磁盘容量
- 根据多集群规模调整 vmstorage 磁盘

系统配置

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 。



组件说明

模块	组件名称	说明
指标	vminsert-insight-victoria-metrics-k8s-stack	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。
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。
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。

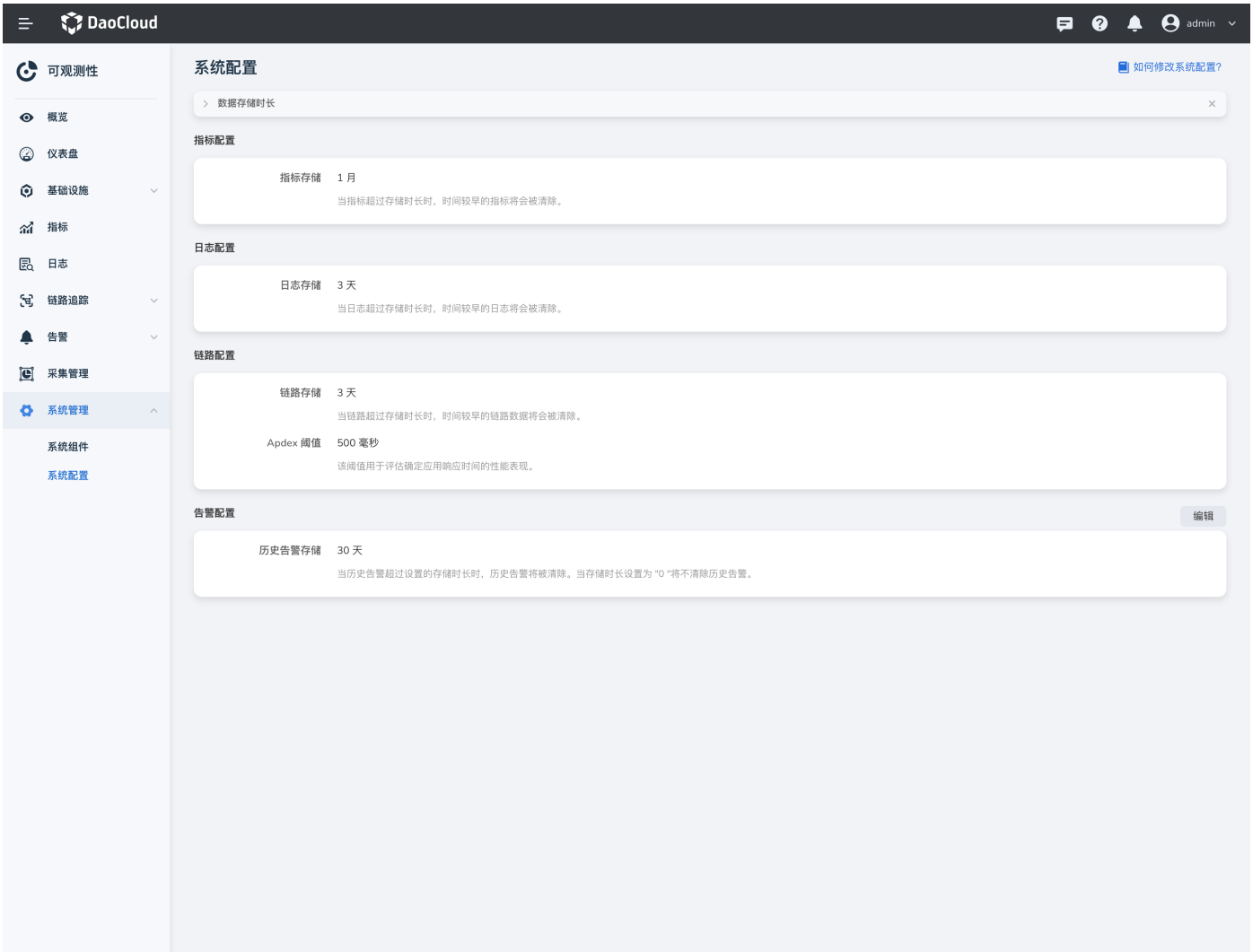
Note

若使用外部 Elasticsearch 可能无法获取部分数据以致于 Elasticsearch 的信息为空。

系统配置

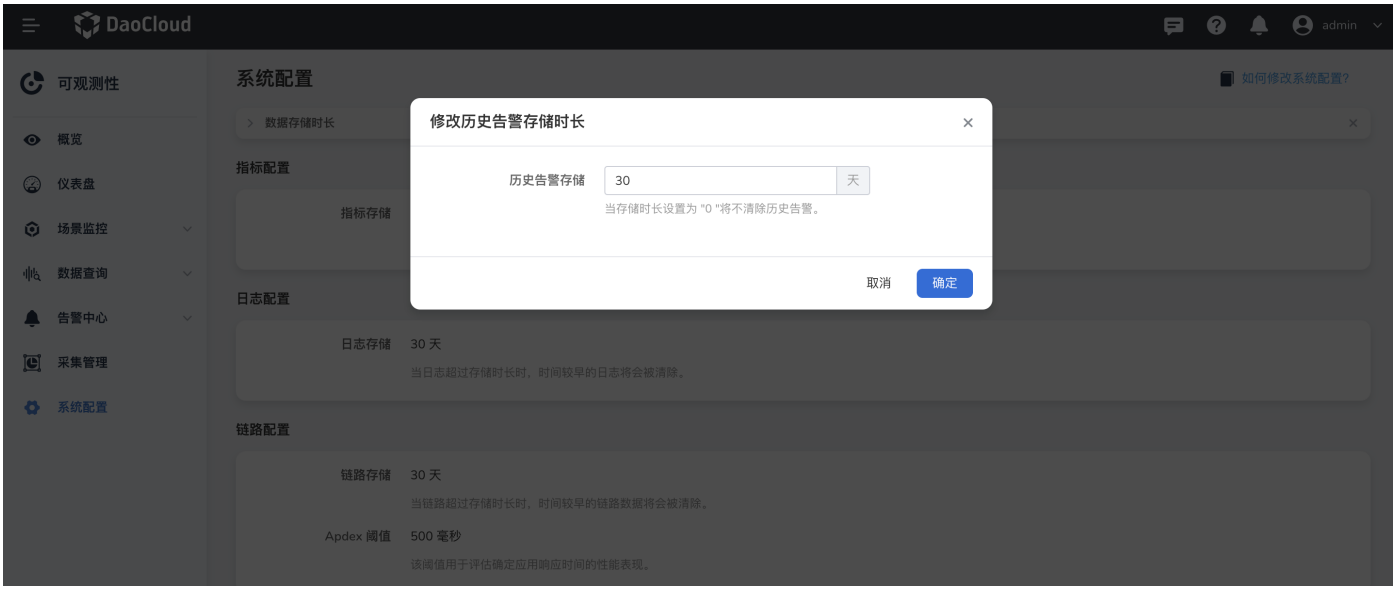
系统配置 展示指标、日志、链路默认的保存时长以及默认的 Apdex 阈值。

1. 点击右侧导航栏，选择 系统配置 。



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。



Note

修改其他配置，请点击查看如何修改系统配置？

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。该文档仅适用于内置部署的 Elasticsearch，若使用外部 Elasticsearch 可自行调整。

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。

```
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。

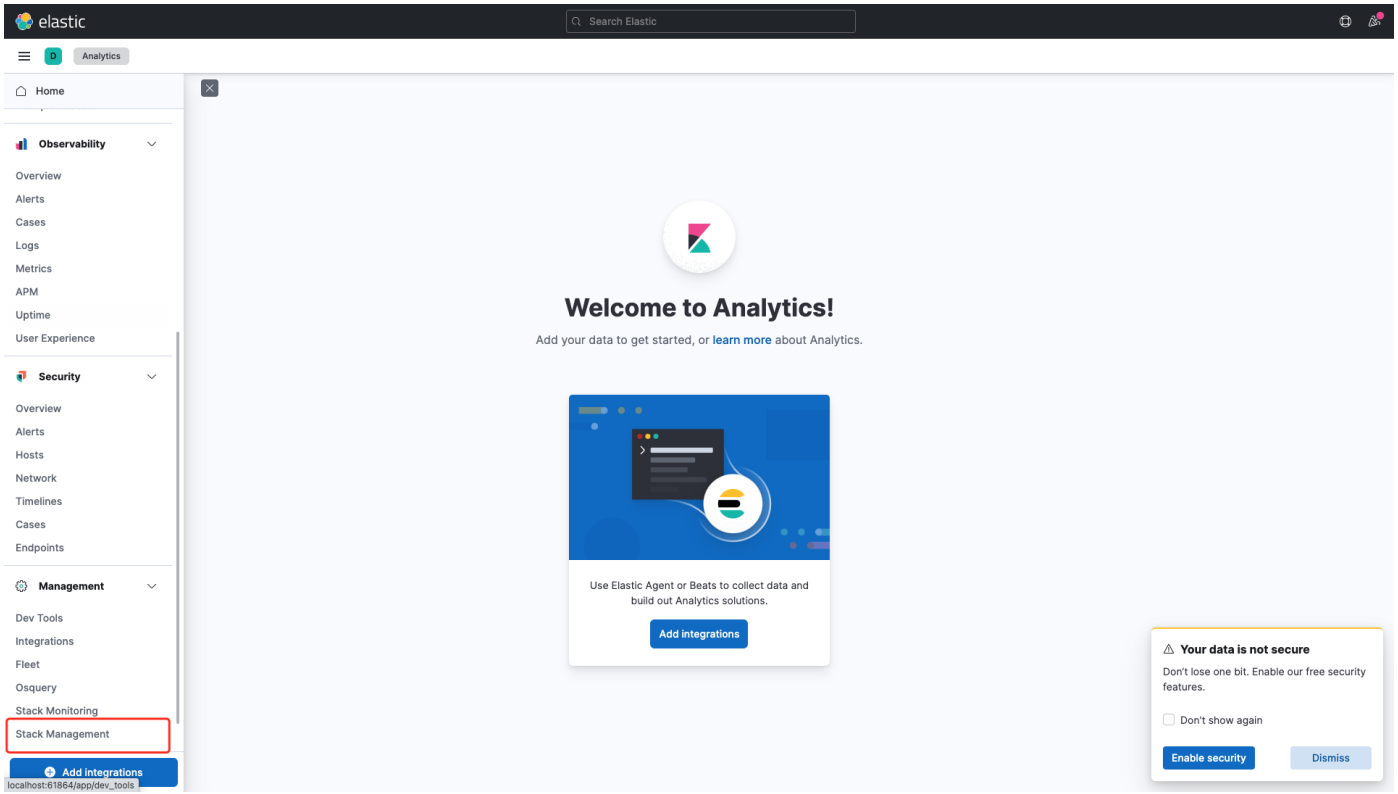
```
curl  --insecure --location -u"elastic:amyVt4o826e322TUVi13Ezw6" -X PUT "https://172.30.47.112:30468/_ilm/policy/insight-es-k8s-logs-policy?pretty" -H 'Content-
Type: application/json' -d'
{
    "policy": {
        "phases": {
            "hot": {
                "min_age": "0ms",
                "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。

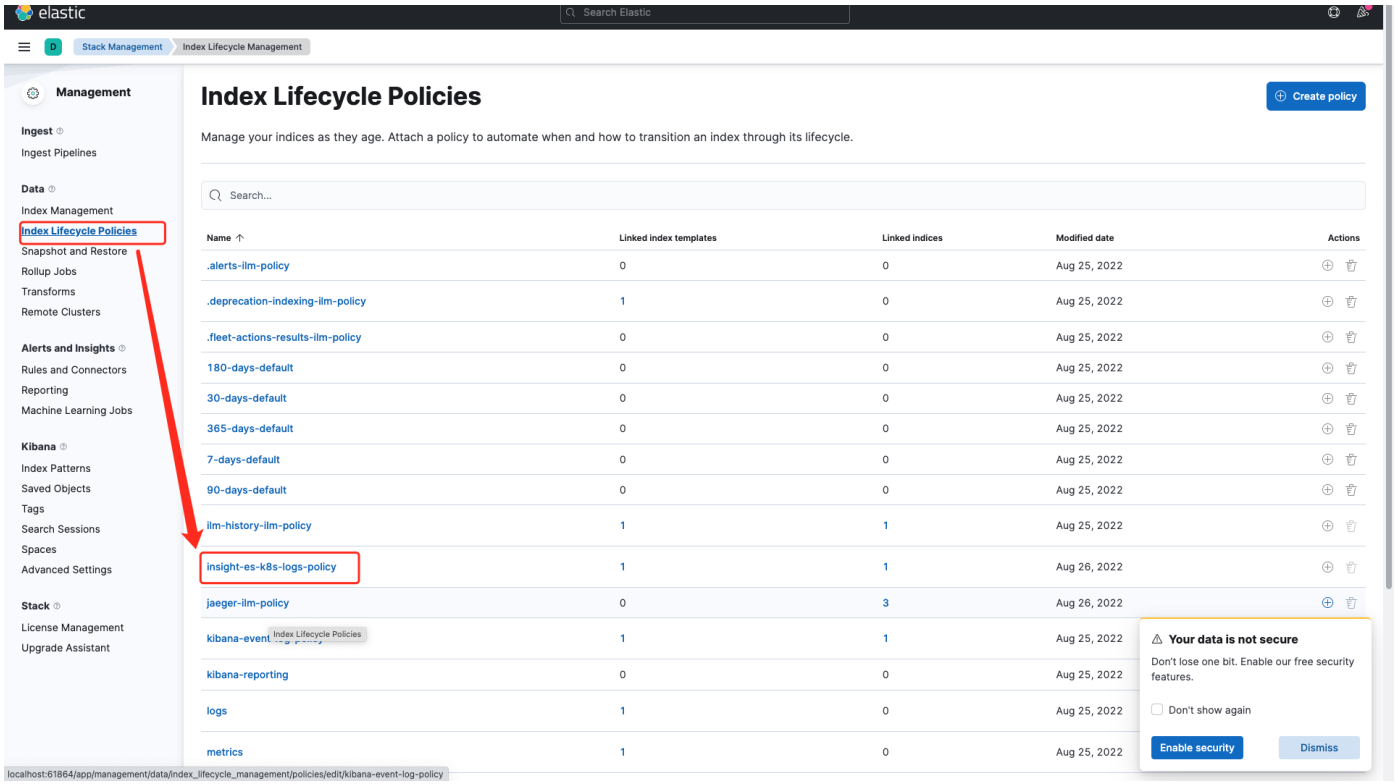
```
{
"acknowledged" : true
}
```

方法二：从 UI 修改

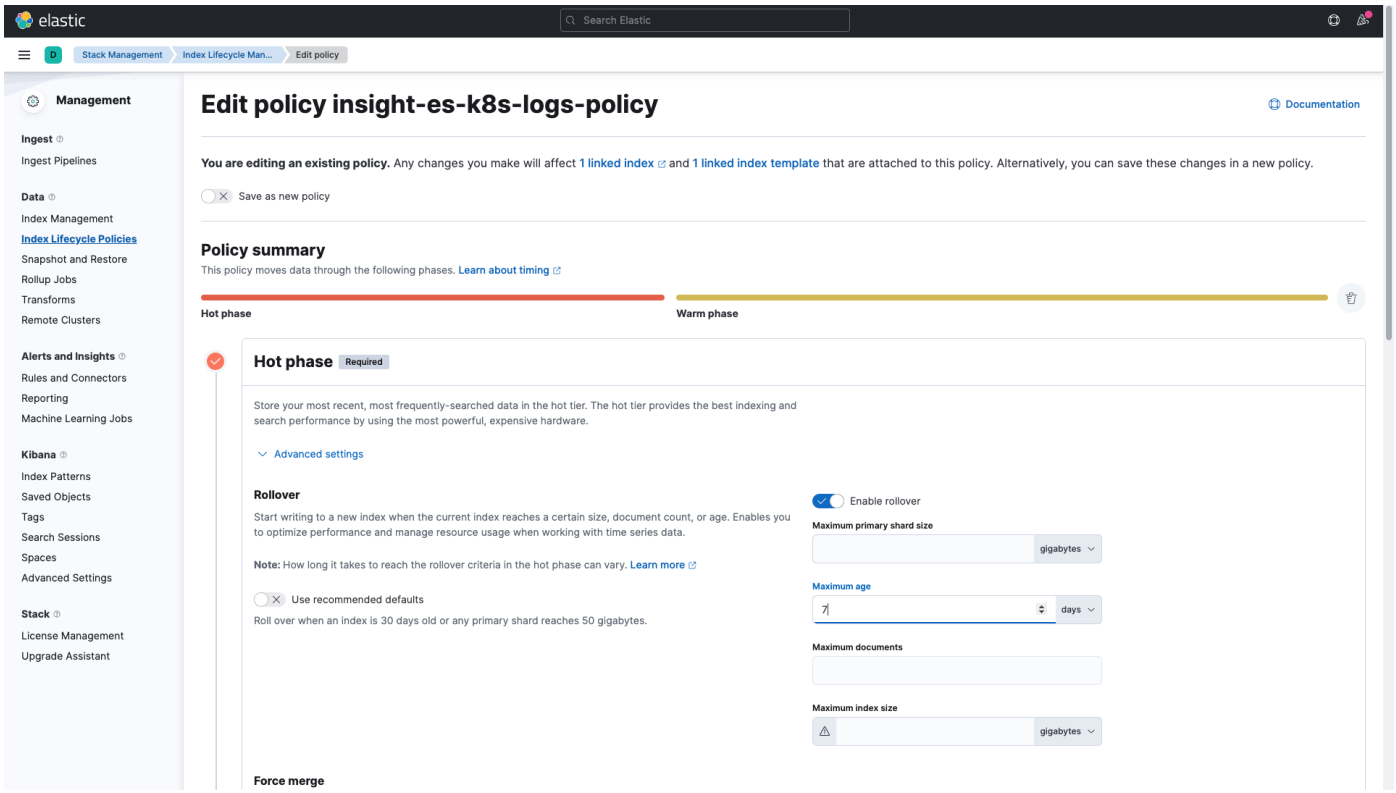
1. 登录 **kibana** ，选择左侧导航栏 **Stack Management** 。



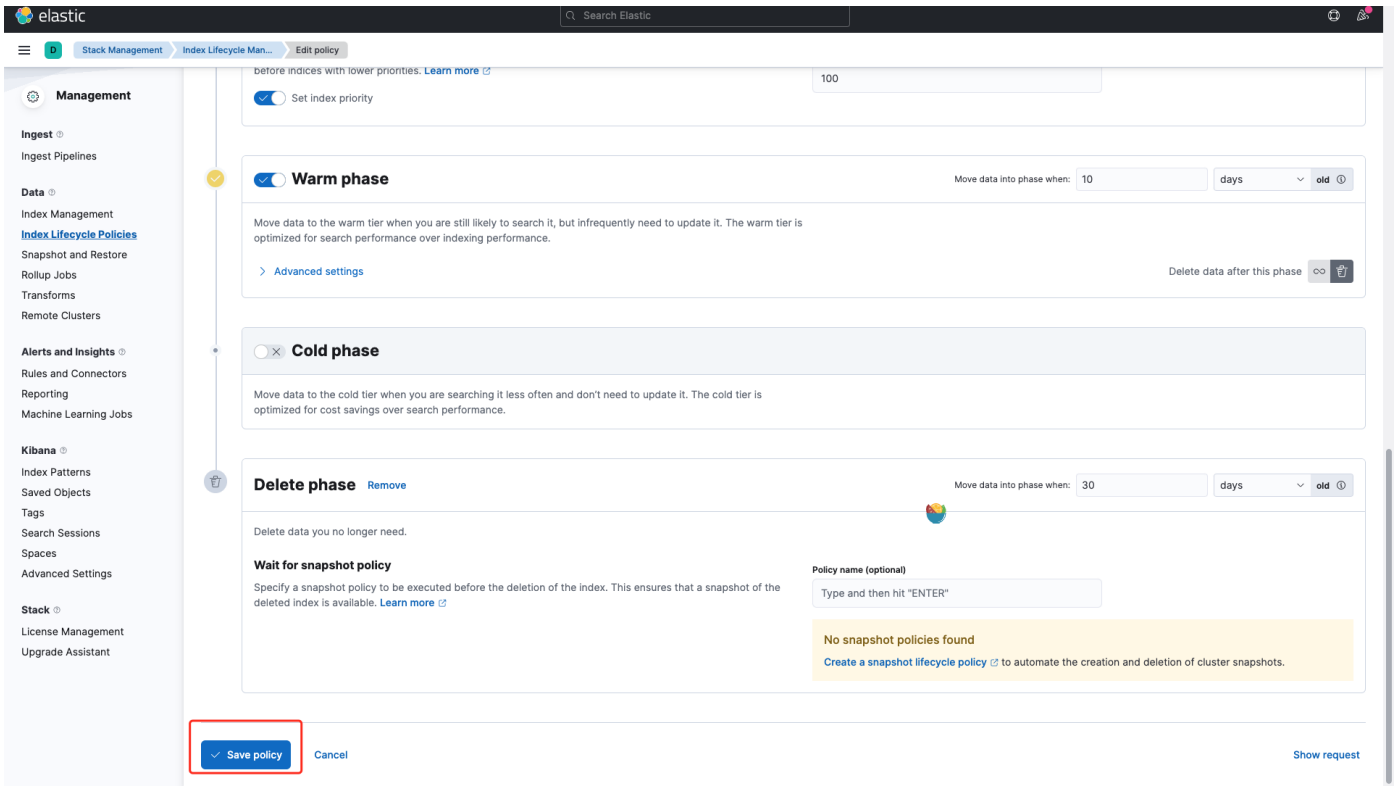
2. 选择左侧导航 **Index Lifecycle Polices** ，并找到索引 **insight-es-k8s-logs-policy** ，点击进入详情。



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** 。



4. 修改完后，点击页面底部的 **Save policy** 即修改成功。



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。

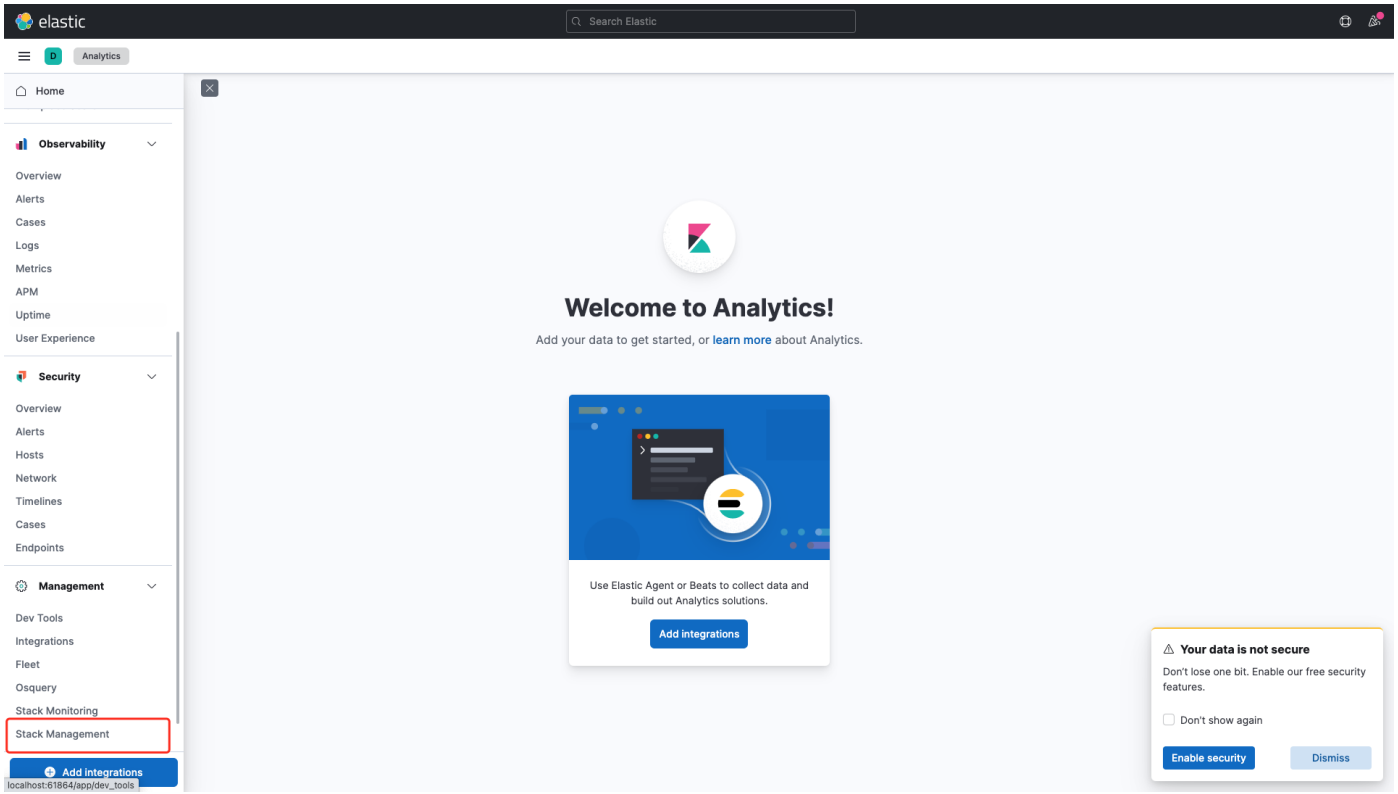
```
curl --insecure --location -u"elastic:amyVt4o826e322TUVi13Ezw6" -X PUT "https://172.30.47.112:30468/_ilm/policy/jaeger-ilm-policy?pretty" -H 'Content-Type:
application/json' -d'
{
    "policy": {
        "phases": {
            "hot": {
                "min_age": "0ms",
                "actions": {
                    "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。

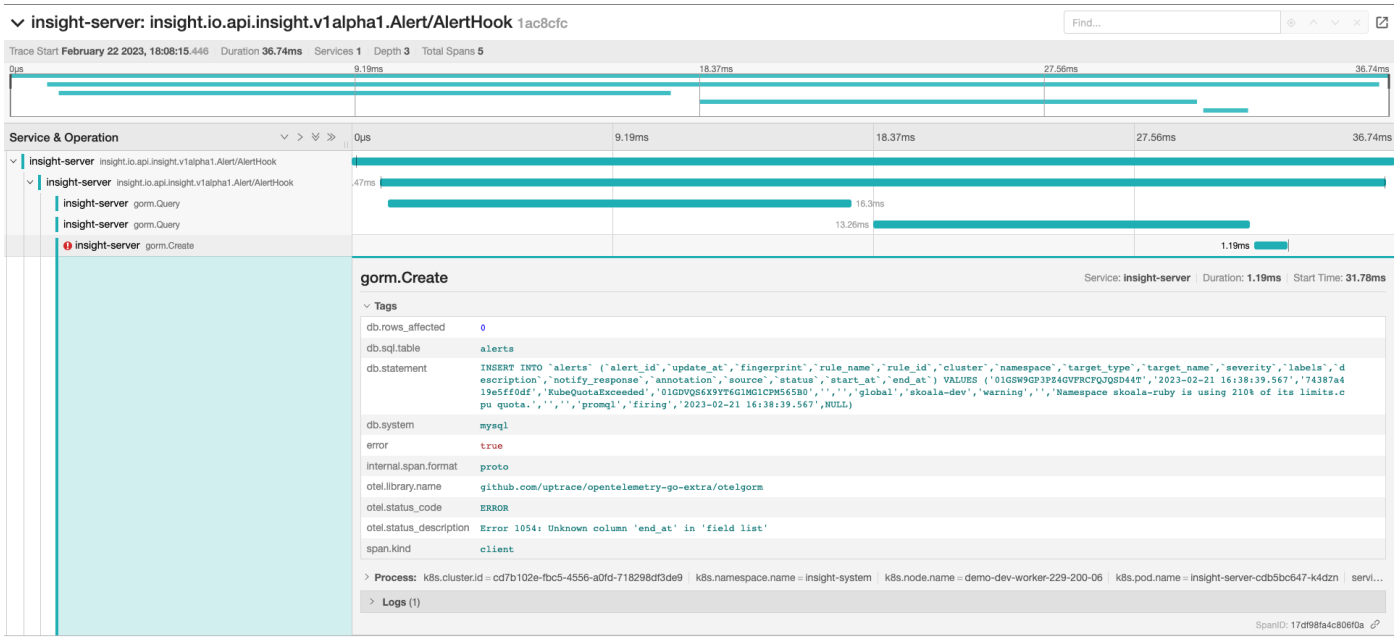
```
{
"acknowledged" : true
}
```

方法二：从 UI 修改

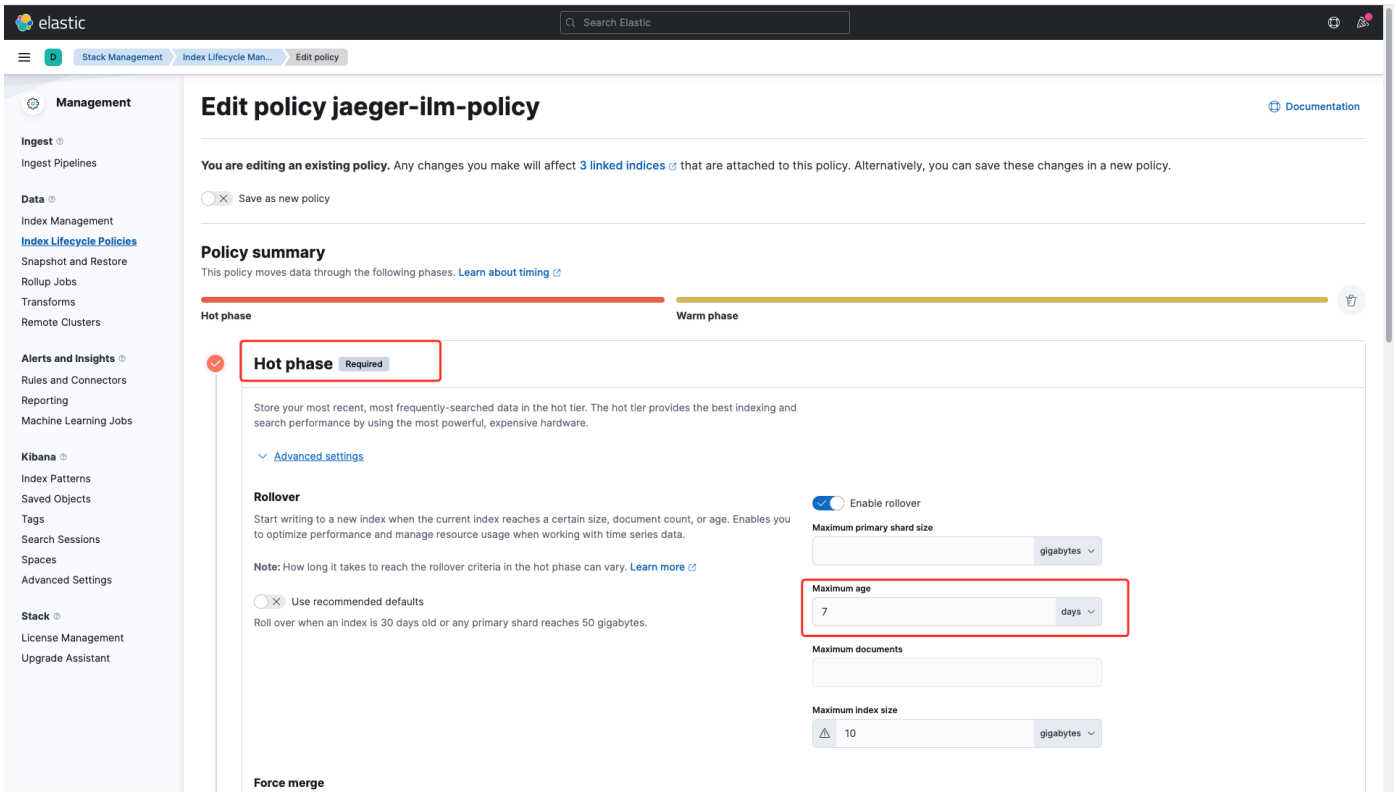
1. 登录 **kibana** ，选择左侧导航栏 **Stack Management** 。



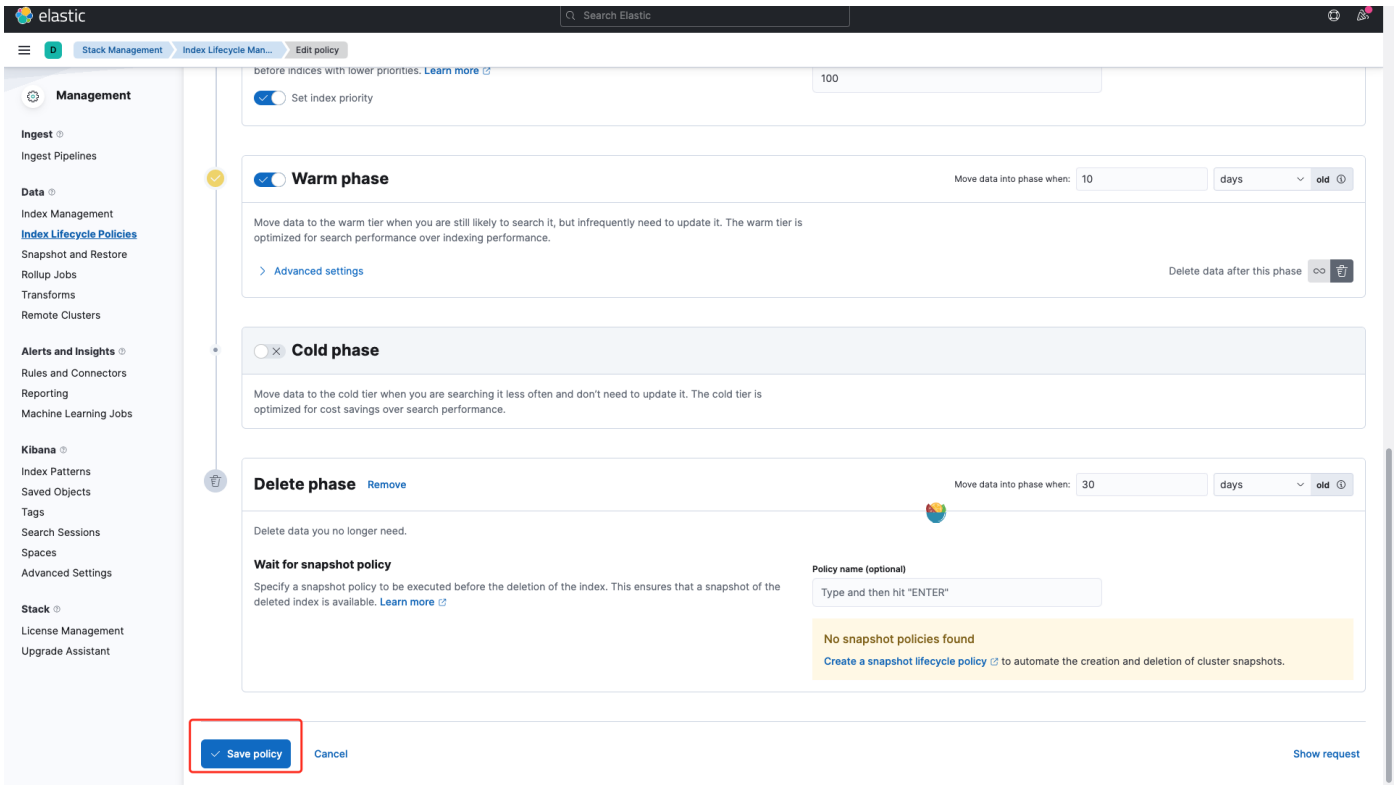
2. 选择左侧导航 **Index Lifecycle Polices** ，并找到索引 **jaeger-ilm-policy** ，点击进入详情。



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** 。



4. 修改完后，点击页面底部的 **Save policy** 即修改成功。



参考文档

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。

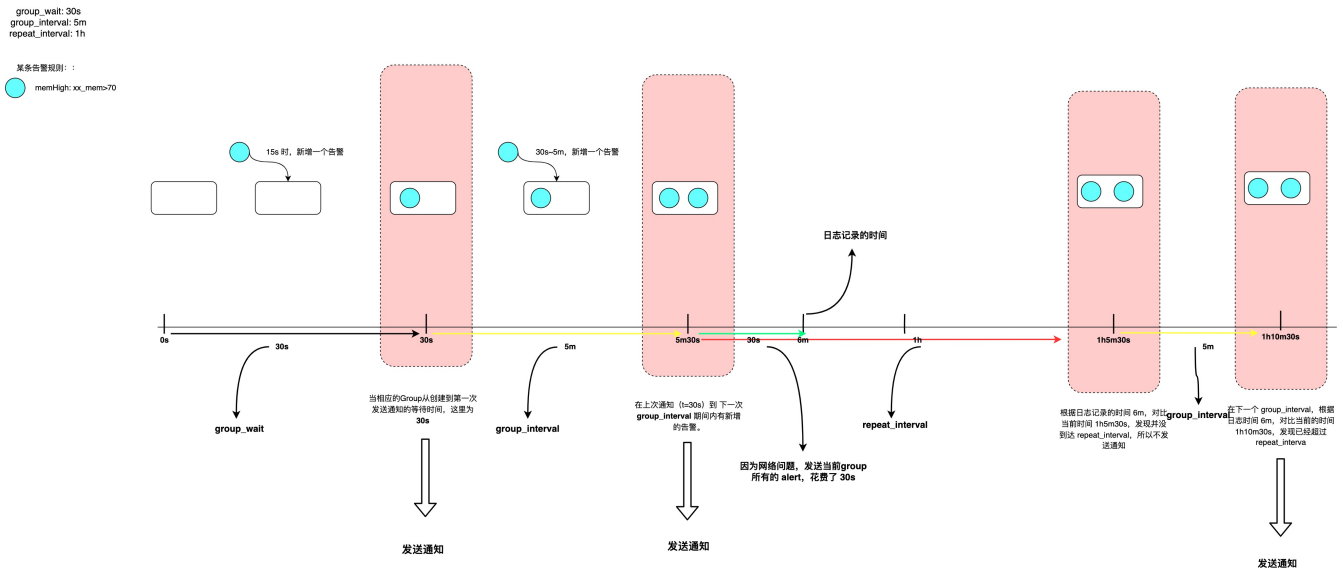
举例说明

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。



通知模板使用说明

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-prometheus-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}}
   ```

函数说明 FUNCTIONS

Insight 的”通知模板“和”短信模板“支持 70 多个 sprig 函数，以及自研的函数。

Sprig 函数

Sprig 内置了 70 多种常见的模板函数帮助渲染数据。以下列举常见函数：

- 时间操作
- 字符串操作
- 类型转换操作
- 整数的数学计算

更多细节可以查看官方文档。

自研函数 toClusterName

toClusterName 函数根据“集群唯一标示 Id”查询“集群名”；如果查询不到对应的集群，将直接返回传入的集群的唯一标示。

```
func toClusterName(id string) (string, error)
```

示例：

```
{{ toClusterName "clusterId" }}
{{ "clusterId" | toClusterName }}
```

toClusterId

toClusterId 函数根据“集群名”查询“集群唯一标示 Id”；如果查询不到对应的集群，将直接返回传入的集群名。

```
func toClusterId(name string) (string, error)
```

示例：

```
{{ toClusterId "clusterName" }}
{{ "clusterName" | toClusterId }}
```

toDateInZone

toDateInZone 根据字符串时间转换成所需的时间，并进行格式化。

```
func toDateInZone(fmt string, date interface{}, zone string) string
```

示例 **1** ：

```
{{ toDateInZone "2006-01-02T15:04:05" "2022-08-15T05:59:08.064449533Z" "Asia/Shanghai" }}
```

将获得返回值 **2022-08-15T13:59:08** 。此外，也可以通过 sprig 内置的函数达到 **toDateInZone** 的效果：

```
{{ dateInZone "2006-01-02T15:04:05" (toDate "2006-01-02T15:04:05Z07:00" .StartsAt) "Asia/Shanghai" }}
```

示例 **2** ：

```
{{ toDateInZone "2006-01-02T15:04:05" .StartsAt "Asia/Shanghai" }}

## 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。

```text
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 }}
```

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。

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。

自定义探测方式

在本文中，我们将介绍如何在已有的 Blackbox ConfigMap 中配置自定义的探测方式。 我们将以 HTTP 探测方式作为示例，展示如何修改 ConfigMap 以实现自定义的 HTTP 探测。

操作步骤

1. 进入 容器管理 的集群列表，点击进入目标集群的详情；
2. 点击左侧导航，选择 配置与密钥 -> 配置项 ；
3. 找到名为 **insight-agent-prometheus-blackbox-exporter** 的配置项，点击 编辑 **YAML**；

在 **modules** 下添加自定义探测方式。此处添加 HTTP 探测方式为例：

```
module:
  http_2xx:
    prober: http
    timeout: 5s
    http:
      valid_http_versions: [HTTP/1.1, HTTP/2]
      valid_status_codes: []  # Defaults to 2xx
      method: GET
```

Info

更多探测方式可参考 blackbox_exporter Configuration。

友情参考

以下 YAML 文件中包含了 HTTP、TCP、SMTP、ICMP、DNS 等多种探测方式，可根据需求自行修改 `insight-agent-prometheus-blackbox-exporter` 的配置文件。

点击查看完整的 **YAML** 文件

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: insight-agent-prometheus-blackbox-exporter
  namespace: insight-system
  labels:
    app.kubernetes.io/instance: insight-agent
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: prometheus-blackbox-exporter
    app.kubernetes.io/version: v0.24.0
    helm.sh/chart: prometheus-blackbox-exporter-8.8.0
  annotations:
    meta.helm.sh/release-name: insight-agent
    meta.helm.sh/release-namespace: insight-system
data:
  blackbox.yaml: |
    modules:
      HTTP_GET:
        prober: http
        timeout: 5s
        http:
          method: GET
          valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
          follow_redirects: true
          preferred_ip_protocol: "ip4"
      HTTP_POST:
        prober: http
        timeout: 5s
        http:
          method: POST
          body_size_limit: 1MB
      TCP:
        prober: tcp
        timeout: 5s
      ICMP:
        prober: icmp
        timeout: 5s
        icmp:
          preferred_ip_protocol: ip4
      SSH:
        prober: tcp
        timeout: 5s
        tcp:
          query_response:
          - expect: "^SSH-2.0-"
      POP3S:
        prober: tcp
        tcp:
          query_response:
          - expect: "^+OK"
          tls: true
          tls_config:
            insecure_skip_verify: false
      http_2xx_example:               # http 探测示例
        prober: http
        timeout: 5s                   # 探测的超时时间
        http:
          valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]                 # 返回信息中的 Version，一般默认即可
          valid_status_codes: []  # Defaults to 2xx                     # 有效的返回码范围，如果请求的返回码在该范围内，视为探测成功
          method: GET                 # 请求方法
          headers:                    # 请求的头部
            Host: vhost.example.com
            Accept-Language: en-US
            Origin: example.com
          no_follow_redirects: false  # 是否允许重定向
          fail_if_ssl: false
          fail_if_not_ssl: false
          fail_if_body_matches_regexp:
            - "Could not connect to database"
          fail_if_body_not_matches_regexp:
            - "Download the latest version here"
          fail_if_header_matches: # Verifies that no cookies are set
            - header: Set-Cookie
              allow_missing: true
              regexp: '.*'
          fail_if_header_not_matches:
            - header: Access-Control-Allow-Origin
              regexp: '(\*|example\.com)'
          tls_config:                 # 针对 https 请求的 tls 的配置
            insecure_skip_verify: false
          preferred_ip_protocol: "ip4" # defaults to "ip6"               # 首选的 IP 协议版本
          ip_protocol_fallback: false  # no fallback to "ip6"
      http_post_2xx:                  # 带 Body 的 http 探测的示例
        prober: http
        timeout: 5s
        http:
          method: POST                # 探测的请求方法
          headers:
            Content-Type: application/json
          body: '{"username":"admin","password":"123456"}'                # 探测时携带的 body
      http_basic_auth_example:        # 带用户名密码的探测的示例
        prober: http
        timeout: 5s
        http:
          method: POST
```

```
      headers:
        Host: "login.example.com"
      basic_auth:                  # 探测时要加的用户名密码
        username: "username"
        password: "mysecret"
  http_custom_ca_example:
    prober: http
    http:
      method: GET
      tls_config:                  # 指定探测时使用的根证书
        ca_file: "/certs/my_cert.crt"
  http_gzip:
    prober: http
    http:
      method: GET
      compression: gzip            # 探测时使用的压缩方法
  http_gzip_with_accept_encoding:
    prober: http
    http:
      method: GET
      compression: gzip
      headers:
        Accept-Encoding: gzip
  tls_connect:                     # TCP 探测的示例
    prober: tcp
    timeout: 5s
    tcp:
      tls: true                    # 是否使用 TLS
  tcp_connect_example:
    prober: tcp
    timeout: 5s
  imap_starttls:                   # 探测 IMAP 邮箱服务器的配置示例
    prober: tcp
    timeout: 5s
    tcp:
      query_response:
        - expect: "OK.*STARTTLS"
        - send: ". STARTTLS"
        - expect: "OK"
        - starttls: true
        - send: ". capability"
        - expect: "CAPABILITY IMAP4rev1"
  smtp_starttls:                   # 探测 SMTP 邮箱服务器的配置示例
    prober: tcp
    timeout: 5s
    tcp:
      query_response:
        - expect: "^220 ([^ ]+) ESMTP (.+)$"
        - send: "EHLO prober\r"
        - expect: "^250-STARTTLS"
        - send: "STARTTLS\r"
        - expect: "^220"
        - starttls: true
        - send: "EHLO prober\r"
        - expect: "^250-AUTH"
        - send: "QUIT\r"
  irc_banner_example:
    prober: tcp
    timeout: 5s
    tcp:
      query_response:
        - send: "NICK prober"
        - send: "USER prober prober prober :prober"
        - expect: "PING :([^ ]+)"
          send: "PONG ${1}"
        - expect: "^:[^ ]+ 001"
  icmp_example:                    # ICMP 探测配置的示例
    prober: icmp
    timeout: 5s
    icmp:
      preferred_ip_protocol: "ip4"
      source_ip_address: "127.0.0.1"
  dns_udp_example:                 # 使用 UDP 进行 DNS 查询的示例
    prober: dns
    timeout: 5s
    dns:
      query_name: "www.prometheus.io"                # 要解析的域名
      query_type: "A"              # 该域名对应的类型
      valid_rcodes:
      - NOERROR
      validate_answer_rrs:
        fail_if_matches_regexp:
        - ".*127.0.0.1"
        fail_if_all_match_regexp:
        - ".*127.0.0.1"
        fail_if_not_matches_regexp:
        - "www.prometheus.io.\t300\tIN\tA\t127.0.0.1"
        fail_if_none_matches_regexp:
        - "127.0.0.1"
      validate_authority_rrs:
        fail_if_matches_regexp:
        - ".*127.0.0.1"
      validate_additional_rrs:
        fail_if_matches_regexp:
        - ".*127.0.0.1"
  dns_soa:
    prober: dns
    dns:
      query_name: "prometheus.io"
      query_type: "SOA"
```

```
    dns_tcp_example:              # 使用 TCP 进行 DNS 查询的示例
      prober: dns
      dns:
        transport_protocol: "tcp" # defaults to "udp"
        preferred_ip_protocol: "ip4" # defaults to "ip6"
        query_name: "www.prometheus.io"
```

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
# 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 app，
        # 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.containers.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 ]
```

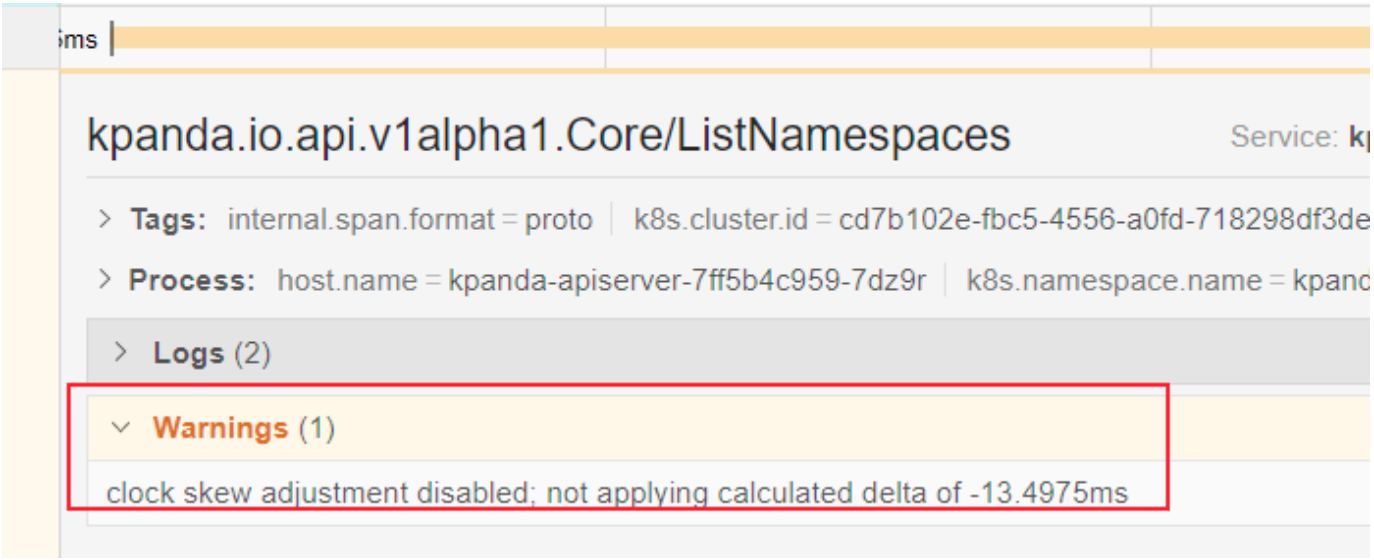
链路数据中的时钟偏移

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：

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：



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

1.3 模型应用

1.3.1 智能问答

什么是智能问答

d.run 智能问答产品运用了 RAG 技术，将用户语料向量化，再利用大语言模型结合相似度搜索用户语料库，实现知识问答。 这一方法解决了用户隐私数据的保护和模型幻觉的问题，使得回答更为精准和个性化。

您在这里可以创建应用和插件，为 AI 对话构建语料库，然后通过数据分析反馈来优化用户体验。

注册并体验 **d.run**

功能特性

智能问答的功能特性参见下表：

主要功能	细分项
对话	支持基于已发布的应用关联语料库，实现智能问答功能
	对AI提供的回答，用户可以进行评价，复制，重新生成，删除对话或提交反馈，高度互动
应用中心	支持创建 RAG 知识问答应用，提供应用的全生命周期管理，以及绑定或解绑工作空间，实现环境隔离
	支持使用 GLM、Llama 等模型服务作为应用的模型服务，实现 RAG
	提供多种配置选项，包括 AI 配置、关联语料库、检索策略、召回策略和提示词模板，以优化 AI 回答的效果
	支持创建应用的密钥，用于 OpenAPI 对话，保证安全性
语料库	支持使用北京智源人工智能研究院的 bge-large-zh-v1.5、bge-large-en-v1.5 对语料进行特征提取
	支持多种导入方式，包括标准导入、格式化导入、手动导入、图文导入
	支持多种分片方式，包括按照分割符、分片大小进行分片
	支持使用插件对文件进行自定义分片
	支持设置语料库的访问级别，保证数据隔离
	支持 CSV 和 Excel 格式导出语料库分片，方便后续处理和分析
数据分析	提供问答质量、问答次数、分片质量、分片命中率等数据分析
	收集并处理用户反馈意见，以便改进服务
我的反馈	保存用户提交过的反馈信息，记录用户的使用体验
系统配置	提供配置聊天记忆轮数、提示词模板等系统配置选项

应用中心

创建应用

本页介绍如何在应用中心创建一个应用。

1. 在 应用中心 页面中，点击 创建 按钮

2. 填写应用基础信息

- 上传应用图标 ： 选择 jpg、jpeg 或 png 格式的图片，确保文件大小不超过 10MB。
- 输入应用名称 ： 限制在 20 个字符以内。
- 填写应用简介 ： 简介应用，不超过 100 个字符。

3. 配置 AI

- 选择大语言模型服务 ： 决定是使用本地模型服务还是在线模型服务（例如 Azure Open AI 或文心一言）。
- 设置随机度 ： 调整模型回答的创造性/发散性程度。
- 向量化模型 ： 应用选择的向量化模型及向量化模型服务。
- 命中语料库时的模版 ： 为模型提供检索和回答问题的模板。
- 未命中语料库时的模版 ： 提供模型在检索不到相似语料时的通用提示词。

> **Note**
>
> - 命中语料库时的模版：当检索到相似语料的时候，会使用语料库提示词模板。模板内容包含通过相似度搜索得到的 知识块 **{corpus_search_content}**、用户的输入 **{corpus_search_content}** 。这些变量将被替换成对应的文本，发送给大模型，进行问答。
>
> ```
> 做一个知识问答游戏：
> 1. 回答内容必须在"{corpus_search_content}"中。
> 2. 如果问题在所提供的资料信息内无法找到，您会回答:"抱歉,资料库已知的信息中，没有找到您需要的结果,请尝试修改引用的资料库或使用模型自有能力回答。"
> {user_inputs_content}
> ```
>
> - 未命中语料库时的模版：当检索不到相似语料的时候，会使用嵌入提示词模板。嵌入的提示词拼接在用户的问题之前，作为通用的约定提示来引导应用模型进行回答输出。

4. 关联语料库

- 选择向量化模型 ： 确定用于问题向量化的模型。
- 选择语料库 ： 从已有的语料库中选择一个或多个用于模型检索。

5. 设置检索策略

- 知识块数 ： 决定提供给模型的知识块数量。
- 相似度 ： 设置知识块匹配的严格程度。
- 重排序 ： 启用或禁用重排序模型以改进结果排序。
- 图文模式 ： 控制是否以及如何输出图文内容。
- 图文分片最大数 ： 确定在一个对话中最多输出多少图片。

6. 设置召回策略

- 选择回答方式 ： 确定应用在找到匹配语料时的回答方式
- 聊天记忆轮数 ： 设置对话历史的轮数或回合数。

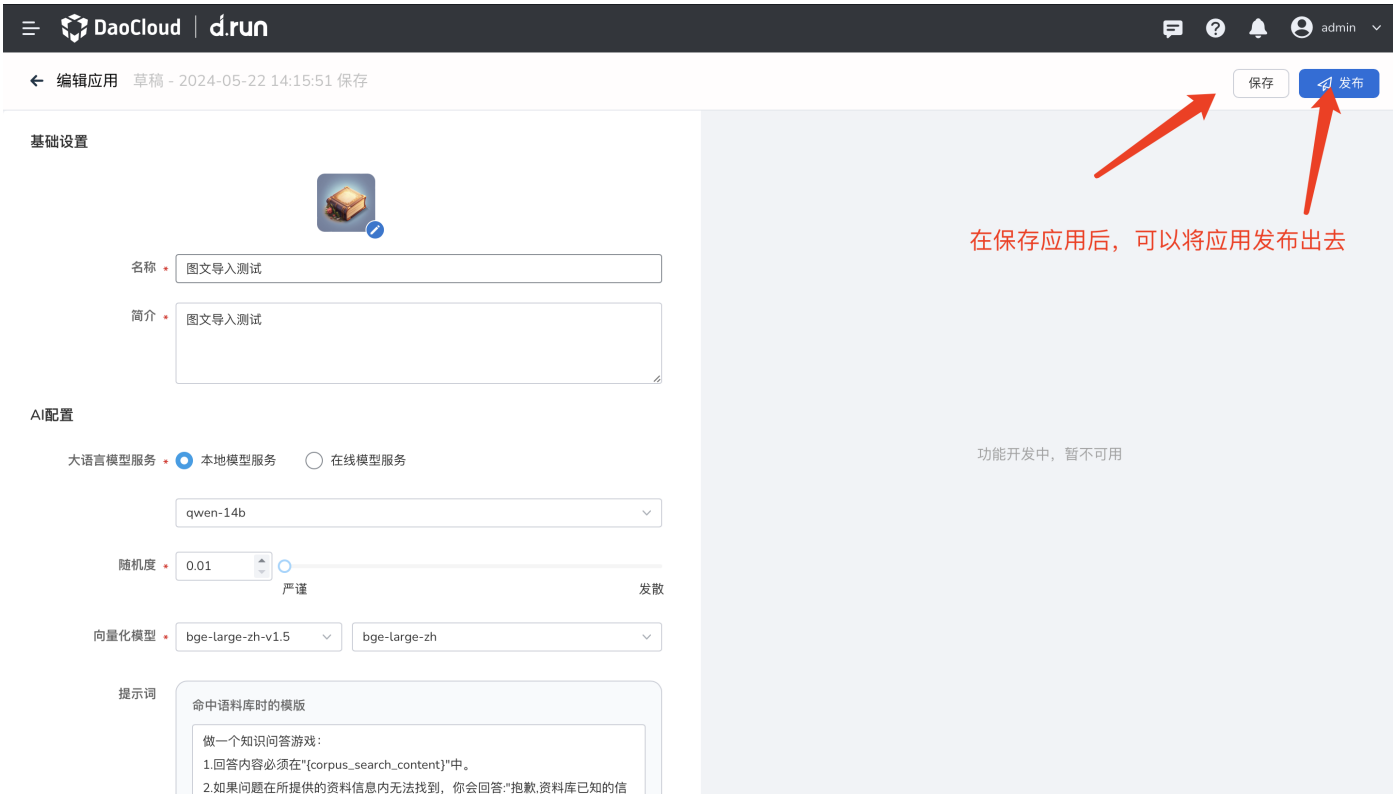
7. 保存并调试应用

- 点击 保存 按钮，应用设置完成，应用会保存成草稿状态。
- 在右侧区域进行应用调试，确保 AI 模型按预期工作。

> **Note**
>
> 请确保按照上述步骤仔细填写和配置应用的各个方面，以确保应用能够准确和高效地响应用户的需求。 在调试阶段，您可能需要根据应用的表现调整某些参数，以达到最佳性能。

8. 发布应用

点击 发布 按钮，应用将会被发布到应用中心，用户可以在应用中心中找到并使用您的应用。



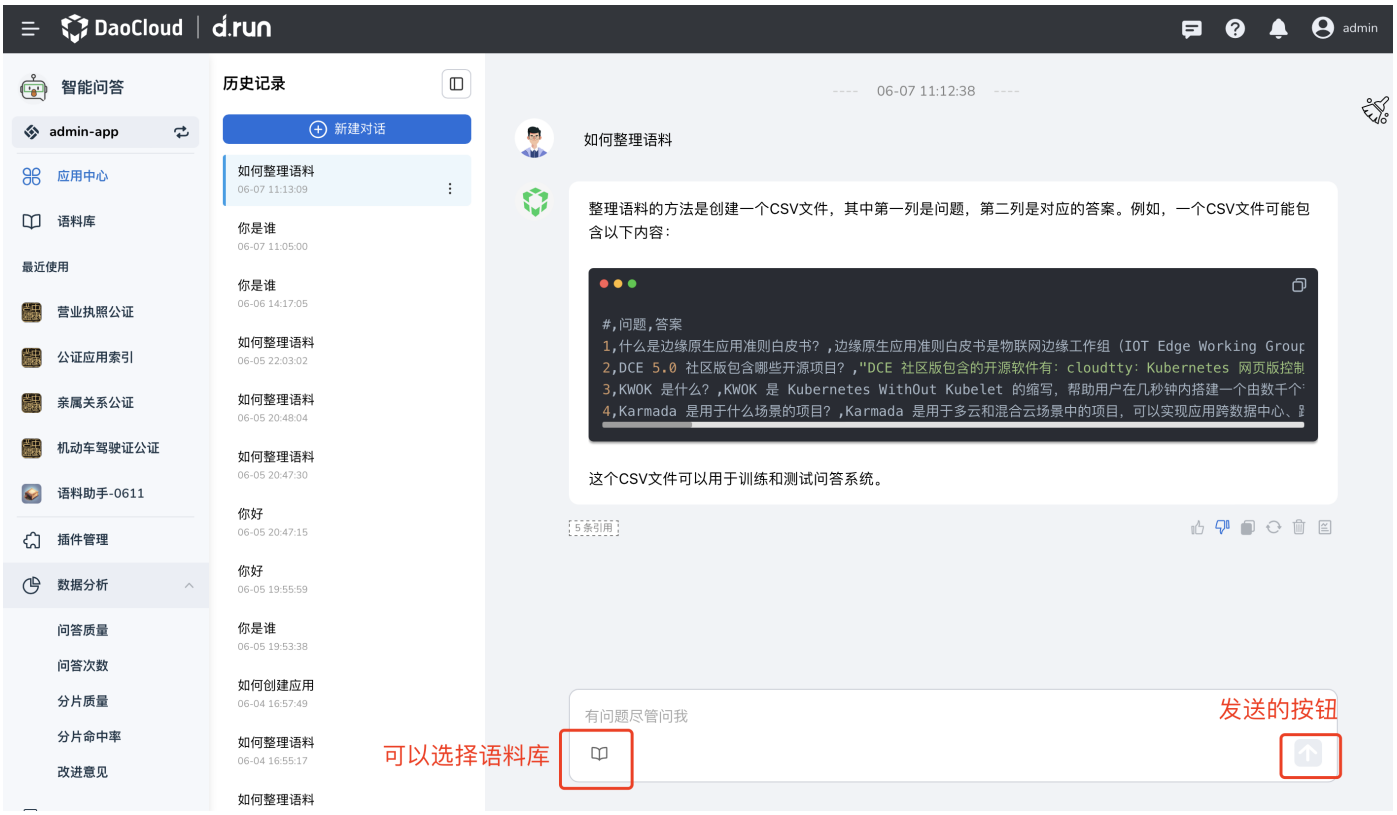
应用对话说明

对话 是 ChatGPT 出现后最常见的获取资讯方式 在 d.run 中，一个应用被发布之后，即可使用对话功能 您可以在对话中关联语料库后自由提问，可以随时查阅历史记录，另外导航栏还列出了最近使用的对话。

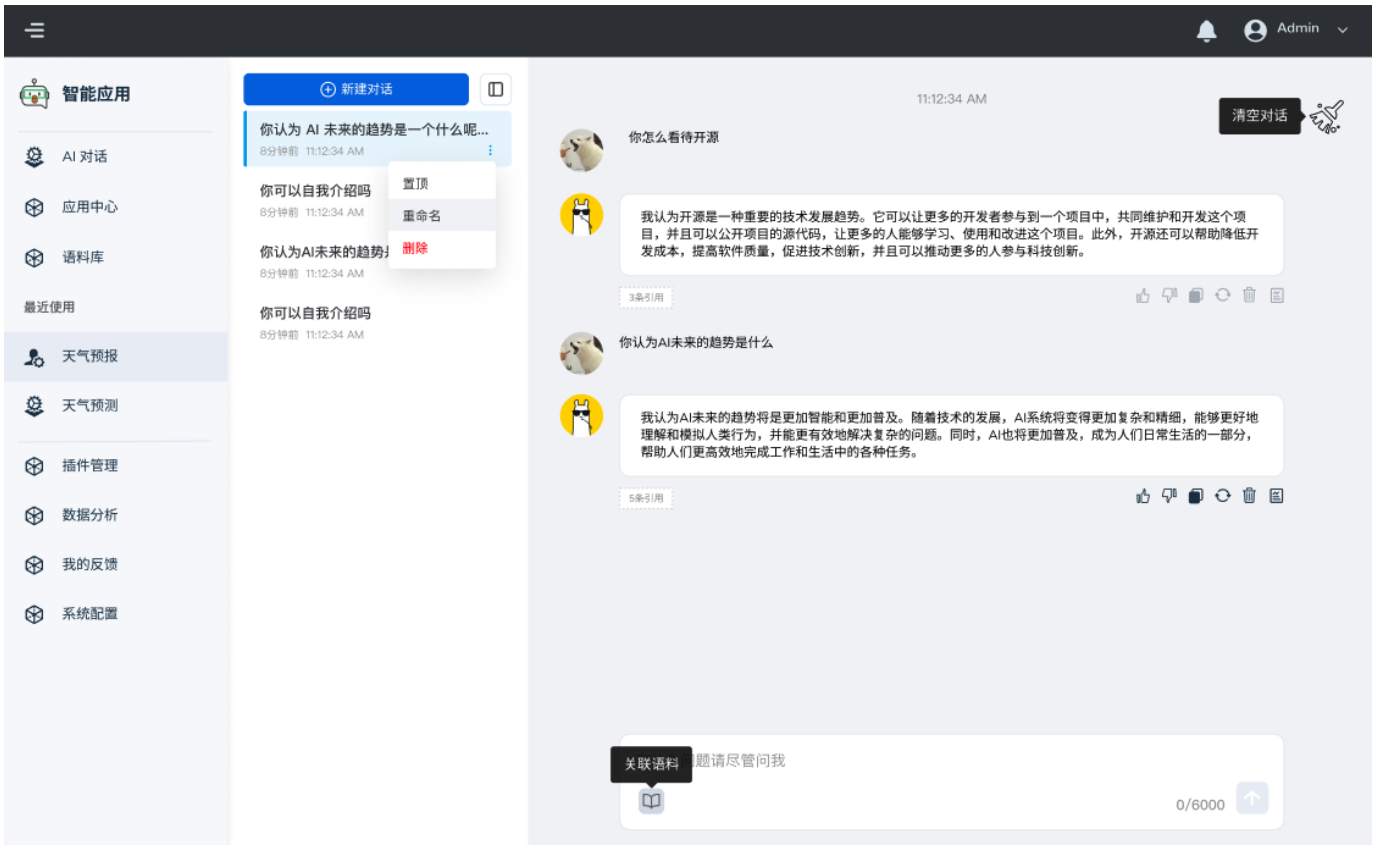
日常对话

1. 在左侧导航栏，点击 应用中心 ，在已发布应用的右下角，点击对话图标。

2. 在对话框中输入问题，点击 发送 ，或敲击回车键，进行对话。


DaoCloud | d.run
admin
智能问答
admin-app
应用中心
语料库
最近使用
营业执照公证
公证应用索引
亲属关系公证
机动车驾驶证公证
语料助手-0611
插件管理
数据分析
问答质量
问答次数
分片质量
分片命中率
改进意见
历史记录
新建对话
如何整理语料
06-07 11:13:09
你是谁
06-07 11:05:00
你是谁
06-06 14:17:05
如何整理语料
06-05 22:03:02
如何整理语料
06-05 20:48:04
如何整理语料
06-05 20:47:30
你好
06-05 20:47:15
你好
06-05 19:55:59
你是谁
06-05 19:53:38
如何创建应用
06-04 16:57:49
如何整理语料
06-04 16:55:17
如何整理语料
06-07 11:12:38
如何整理语料
整理语料的方法是创建一个CSV文件，其中第一列是问题，第二列是对应的答案。例如，一个CSV文件可能包含以下内容：
#,问题,答案
1,什么是边缘原生应用准则白皮书？,边缘原生应用准则白皮书是物联网边缘工作组（IOT Edge Working Group
2,DCE 5.0 社区版包含哪些开源项目？,"DCE 社区版包含的开源软件有：cloudtty：Kubernetes 网页版控制
3,KWOK 是什么？,KWOK 是 Kubernetes WithOut Kubelet 的缩写，帮助用户在几秒钟内搭建一个由数千个
4,Karmada 是用于什么场景的项目？,Karmada 是用于多云和混合云场景中的项目，可以实现应用跨数据中心、
这个CSV文件可以用于训练和测试问答系统。
5条引用
有问题尽管问我
可以选择语料库
发送的按钮

对话管理



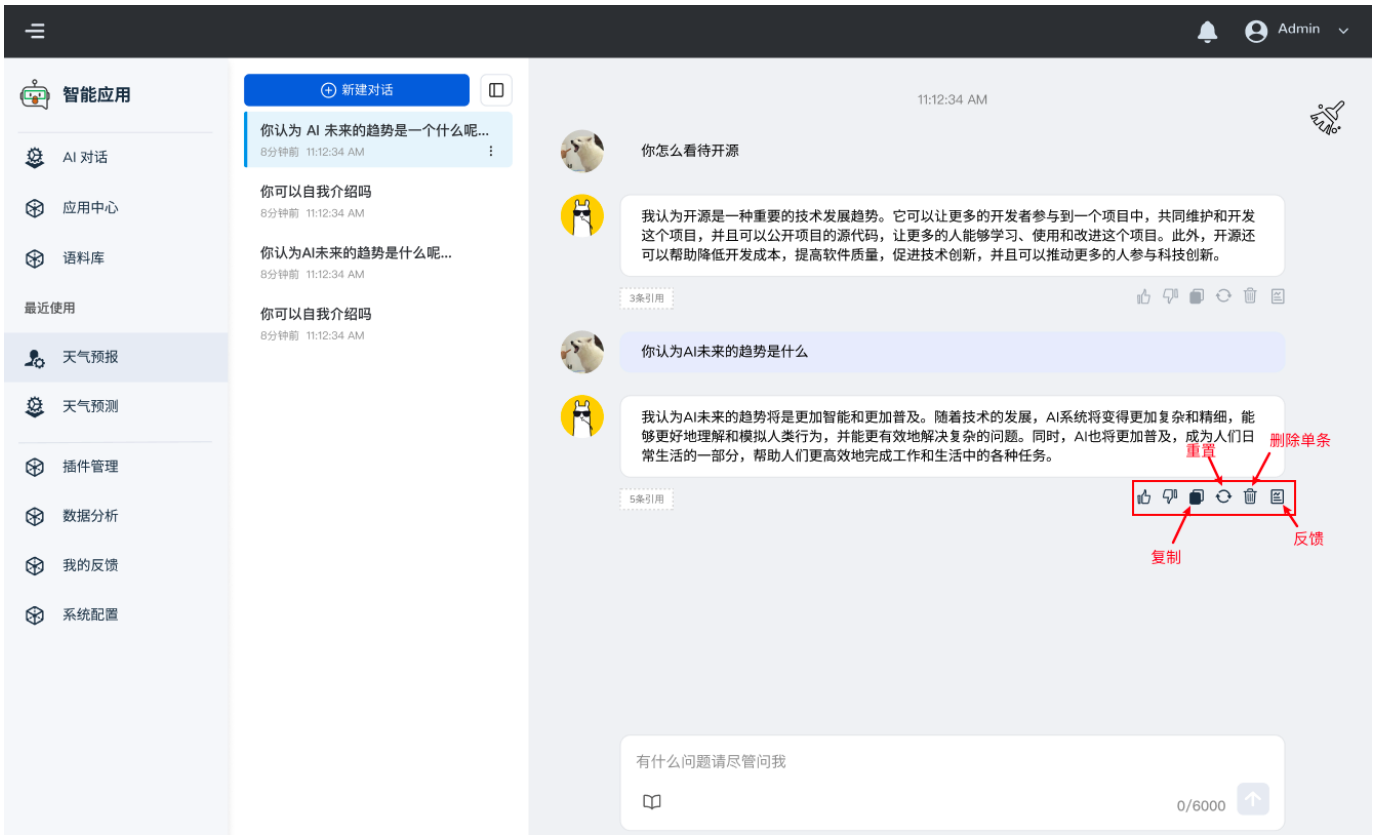
- 置顶 、 重命名 和 删除 ：在历史记录窗格中，点击某一条对话右侧的 ⋮ ，可以置顶、重命名和删除对话。
- 清空 ：在对话页面左上角，点击 🧹 图标，可以 清空 对话。
- 关联语料库 ：在输入框左下角，点击 📖 图标，可以关联语料库。

  语料库的选择： 聊天应用 会在选中的语料库中去匹配最相似的语料作为参考，回答您的问题。 显示操作成功表示已经选中语料库。也可以取消选中某个或某些语料库。

> **Info**
>
> 如果无法选择和更改关联的语料库，只能添加语料库。应用的向量化模型和语料库的向量化模型一致才可以使用。

- 停止对话 ：提问后，点击输入框右侧的图标，可以在回答过程中停止对话，让助手停止输出内容。

几个实用图标



- 评价 ： 可以点赞 👍 或点踩 👎 某条回答，取决于您对回答内容是否满意。
- 复制 ： 可以复制某条回答。
- 重置 ： 可以重置、重新生成某一条回答。

> **随机度**
>
> 对于重置的内容，可以通过管理员调节模型的 随机度 来控制聊天助手多次回答的一致性。
>
> 若随机度高，聊天助手多次回答的结果，可能会有不一样的答案。 如果要求回答的准确度高，可以降低随机度，这样聊天助手每次生成的结果会接近一致。

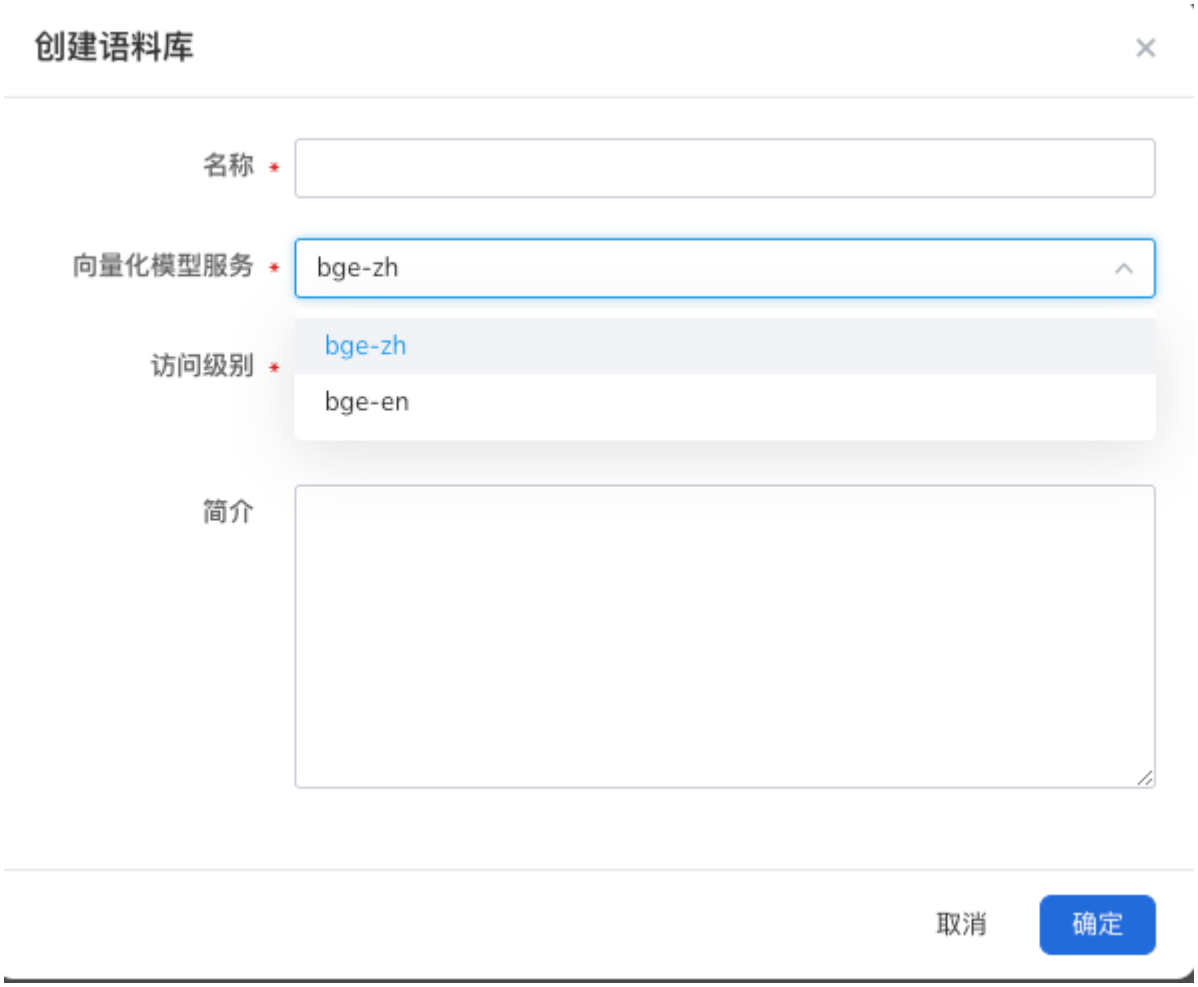
- 删除 ： 点击垃圾桶 🗑 图标，可以删除某条回答，此后在聊天助手上下文对话时，将不会对删除的对话有记忆。
- 改进意见 ： 点击某条回答的最后一个图标 📝，可以提交反馈，根据聊天助手的回答结果好坏，在反馈中提出反馈意见。

语料库

如何创建语料库

1. 在 我的语料 页面中，点击 创建语料 按钮。
2. 参考下列要求填写语料库基本信息，并点击 下一步 。

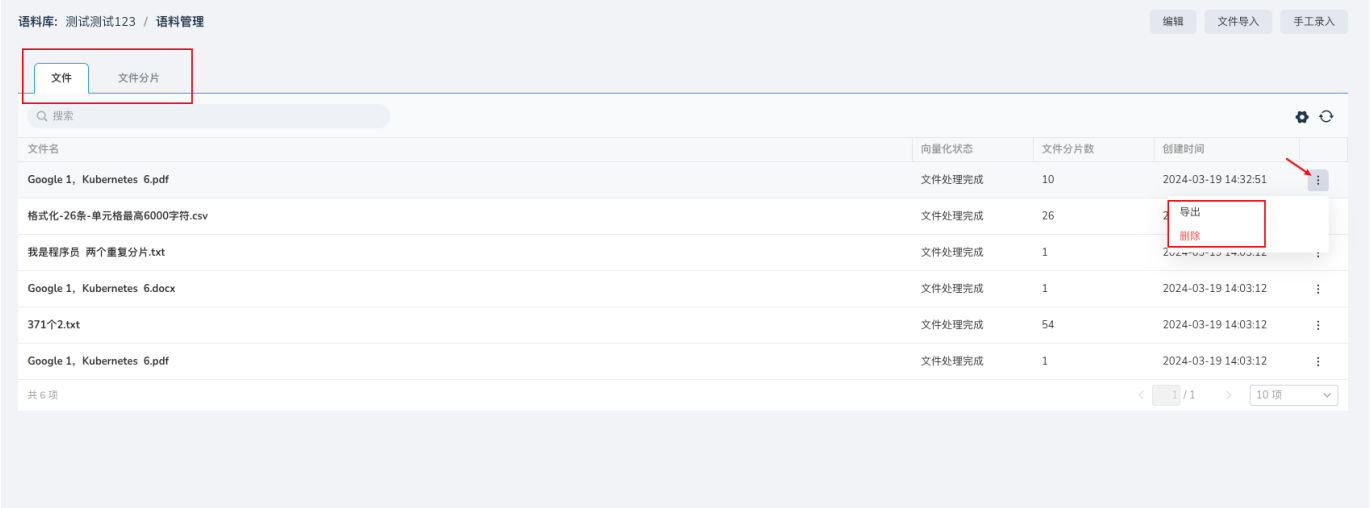
- 语料库名称：名称包含大小写字母、数字和符号（限制20个字符）。
- 向量化模型服务：可选择 `bge-large-zh` 和 `bge-large-en`。
- 访问级别：可选择公开/私有/指定工作空间。
- 简介：简要描述语料库中的内容信息，可包含中英文、数字，（限制 100 个字符）。



管理语料库内容

点击 语料库 创建时间旁的 ⋮ 按钮进入语料库整理页面。

- 查看语料库中文件名称，向量化状态，文件分片数以及创建时间。
- 选择点击操作中 导出 按钮，可以将某一文件导出。
- 点击操作中 删除 按钮，可以将文件在语料库中删除。
- 点击上方的 文件分片 进入到文件分片详情，可以查看文件的所有分块信息。
- 输入文件分片描述来搜索具体的分片，其中文本相似度用于衡量搜索内容和语料库中分块的相似度。这里用【欧氏距离】来作为相似度计算指标，故相似度越小，则距离越接近，文本越相似。
- 在文件分片中点击某一分片的 ⋮ 按钮，即可编辑语料分片内容/删除某一分片。



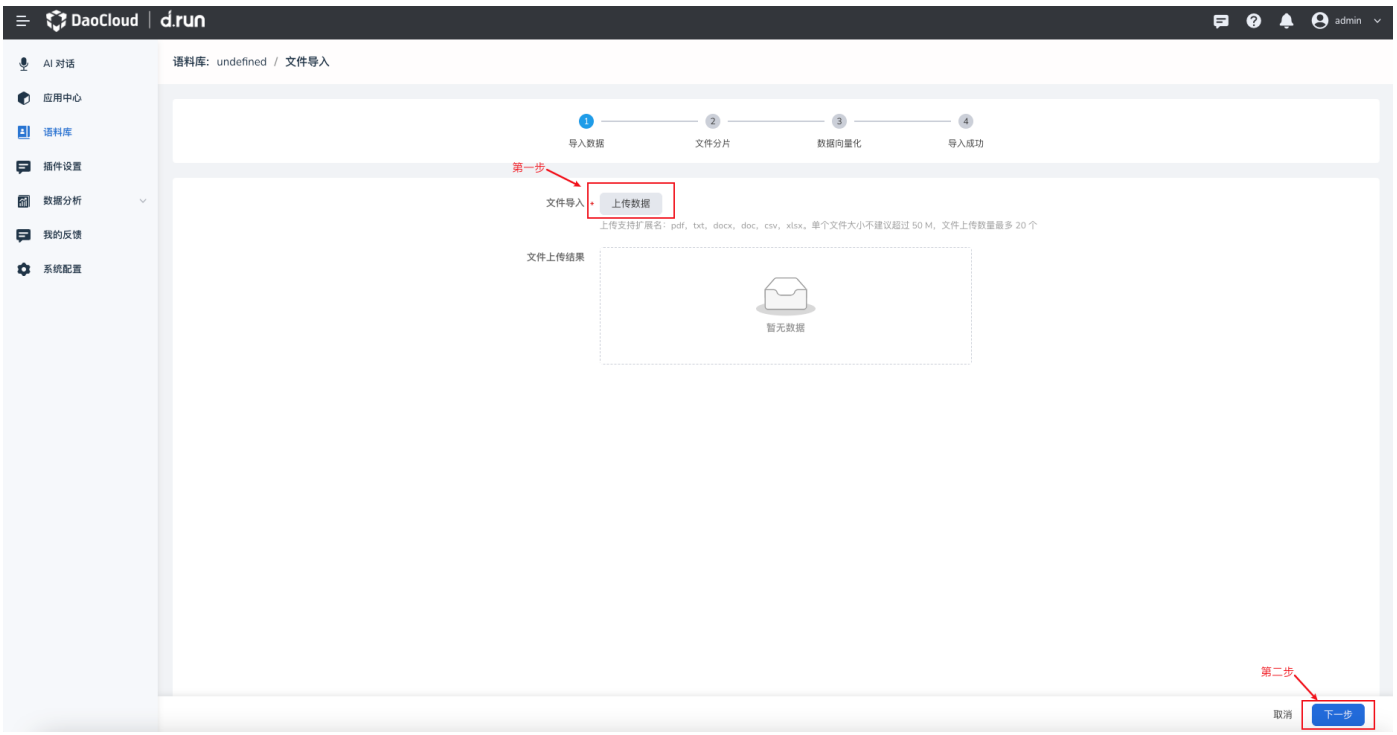
语料导入

文件导入

上传数据

1. 点击语料库旁的 ⋮ 按钮
2. 点击 语料导入 ，选择上传的数据
3. 在导入数据界面，点击 上传数据

   选择自己想要上传的文件，目前支持 pdf、txt、docx、doc、csv、xlsx 单个文件的大小不建议超过 50 M，文件上传数量限制为50个。
4. 将文件上传完成后，可以在 文件上传结果 中查看



5. 上传成功后，点击 下一步
6. 选择文件分片的处理类型：标准处理、自定义处理（即插件处理，请到插件接入处查看）
7. 数据向量化过程后，查看文件分片数量、重复分片数量、本次导入分片数以及向量化状态
8. 当向量化处理成功后，点击 下一步
9. 待文件状态为文件处理完成后，点击 关闭 即可

标准处理

- PDF、TXT、DOC、DOCX 支持自定义分隔符
- CSV、xlsx 按照一行分片
- 设置分隔符，不设置分片大小，仅根据分隔符划分文档
- 不设置分隔符，设置分片大小，仅根据分片大小拆分文档
- 设置分隔符并设置分片大小，在分片大小内，最终根据分隔符匹配进行分割

手工录入

1. 在语料导入时可以选择使用 手工录入 的方式导入分片
2. 点击 手工录入 后，会弹出 新增文件分片 弹框

在箭头位置录入信息，如对信息有备注可在附加分片文本中录入。

导入图文

在导入图文前，需要将导入的语料进行处理后再导入（目前仅支持 Word 和 Excel 的图文处理）。

预处理 Docx 文档

1. 直接支持带图文的 Docx 文档按照约定的字符长度分割

2 或 3 个 AE 堆叠实现 SAE,并用 Softmax 对最终的 EEG 特征 进行分类.另一种 AE 的变[illegible] 降噪自编码器(denoising autoencoder, DAE)[55] 通过向输入层加入噪声干扰,训练网络能够对含噪数据进行编码、解码,从而提高隐层特征的表达能力.Xu 等人利用堆栈 DAE (stacked DAE, SDAE)进行特征学习并利用 Softmax 分类.此外,Li 等人[illegible]利用变分 AE (variational AE, VAE)无监督地学 习EEG 信号的低维表示.

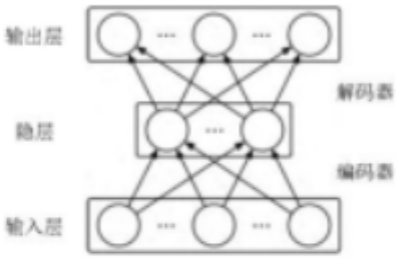


图 5 自编码器框架

深度信念网络(deep belief networks, DBN)[illegible]是一种概率生成模型,它由若干层受限玻尔兹曼机(restricted Boltzmann machines, RBM)组成.其深度结构能够有效学习高层特征表示.一个 RBM 只有两层神经元可视层和 隐层,分别用于输入数据和特征检测.两层之间双向连接,层内神经元之间无连接.将多个 RBM 堆叠则构成 DBN,其中低层 RBM 的输出作为高层 RBM 的输入.DBN 既可用于无监督学习,也可用于有监督学习.如图 6 所示,通 常采用含有两个隐层的 DBN 进行 EEG 情绪识别[47],其训练主要包括 3 个步骤:首先,对每层进行无监督的预训练;其次,将 RBM 展开为编码器和解码器,利用反向传播算法对各层进行无监督的微调,学习使得输入和重构向量尽可能相似的网络权重和偏置;最后,在顶端 RBM 的可视层加入表示类别的神经元,反向传播更新参数,实现有 监督的微调.然而DBN 的隐层缺少约束,使得其很难挖掘 EEG 电极间和频段间的相关信息.Chao 等人[illegible]引入胶质细胞链(glia chain),利用 DBN-GCs 网络学习隐层神经元的关联.胶质细胞能够调整隐层神经元的激活概率,并 向相邻胶质细胞传递信号.此外,文章利用集成学习,将特征子集分别送入 5 个 DBN-GCs,拼接各模块的隐层输 出,作为判别 RBM 的输入.

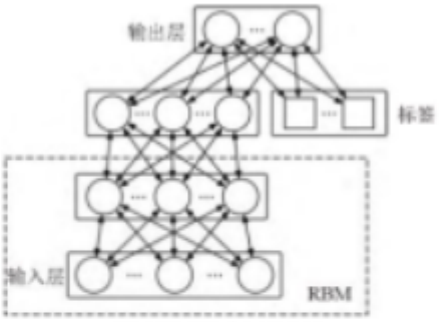


图 6 深度信念网络框架

3.2 卷积神经网络与循环神经网络

CNN[illegible]是一种前馈神经网络,利用反向传播算法进行训练.其中一个典型层包含 3 级:卷积、非线性变换、池化.卷积运算具有局部连接、权值共享、等变表示的特性.通常在同一卷积层内使用多个不同的卷积核以学习 不同特征.非线性变换,也称为激活函数,能够给模型加入非线性因素,提高表达能力.目前大部分 CNN 采用线性 整流(rectified linear unit, ReLU)函数[illegible]作为激活函数.池化是一种过滤细节的方法,使用某一位置周围的总体特征代替网络在该位置的输出.常用的池化函数包括最大池化、平均池化等.现有研究设计或应用不同的 CNN 结构 进行 EEG 情绪识别.Wang 等人[illegible]将 LeNet 用和 ResNet[illegible]这两种典型网络架构分别用于情绪识别.Cimtay 等人[illegible]在预训练的 InceptionResNetV2 模型的基础上增加 5

2. 也支持手工用 `<split></split>` 标签，提前规划好文档分割段落。

开始标签

<split>

中间的图文内容，为一个分片

2 语音情感的认知神经科学研究进展

2.1 情感的神经机制

情感产生的脑机理研究经历了一个较长的过程，受到神经解剖学、神经生理与认知心理学等相关科学发展的影响.思想家和科学家对情绪奥秘的探讨可以追溯到古代的臆测和神秘主义.直到文艺复兴以后，如霍布斯(Hobbes)、洛克(Locke)、笛卡儿(Descartes)等带有唯物主义色彩的哲学家才把知觉、思维、知识、情绪等和神经与脑的活动联系了起来.1872 年,达尔文(Darwin)在《人和动物的表情》一书里论述了情绪的生物学基础，强调了环境对情绪行为的作用，形成了情绪生理心理学的雏形.其后的詹姆斯(James)提出了最早的情绪生理-心理学理论，为探讨情绪的性质指出了一条必由之路.James-Lang 理论(1885 年)即情绪外周理论，强调情绪的产生是植物神经系统活动的产物.1912 年,Mills 首次提出了情感的大脑右半球假说，右脑更多地决定了人的空间感、抽象思维、音乐感与艺术性.1931 年,Cannon 提出了情绪的丘脑学说,认为丘脑对情绪调节起着重要作用.随后,Papez 提出了 Papez 环路理论,认为下丘脑是情绪表达中心，边缘系统是情绪体验部位.但当时,这一回路并没得到科学研究证实.Maclean 于 1952 年提出了情绪脑的概念，划分了较为精细的情绪相关脑区网络，得到研究者的广泛认同(如图 6 所示).

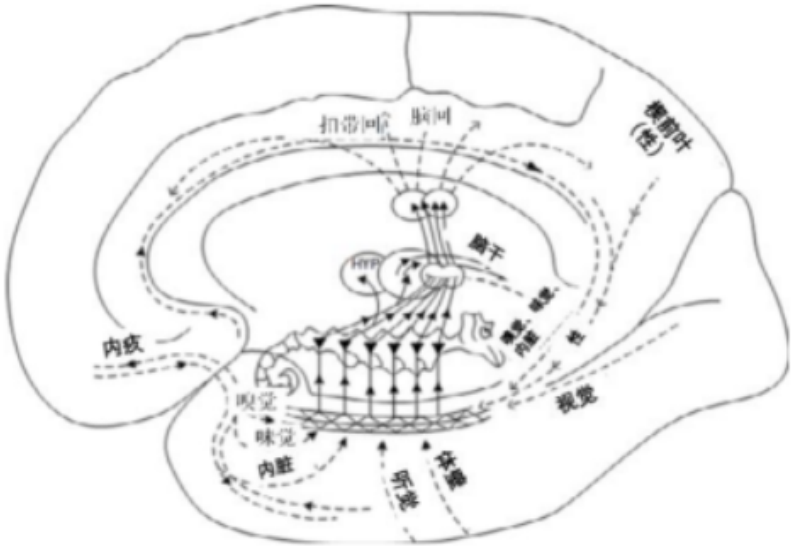


Fig.6 MacLean's limbic system theory"" 图 6 MacLean 提出的边缘系统理论〔%〕

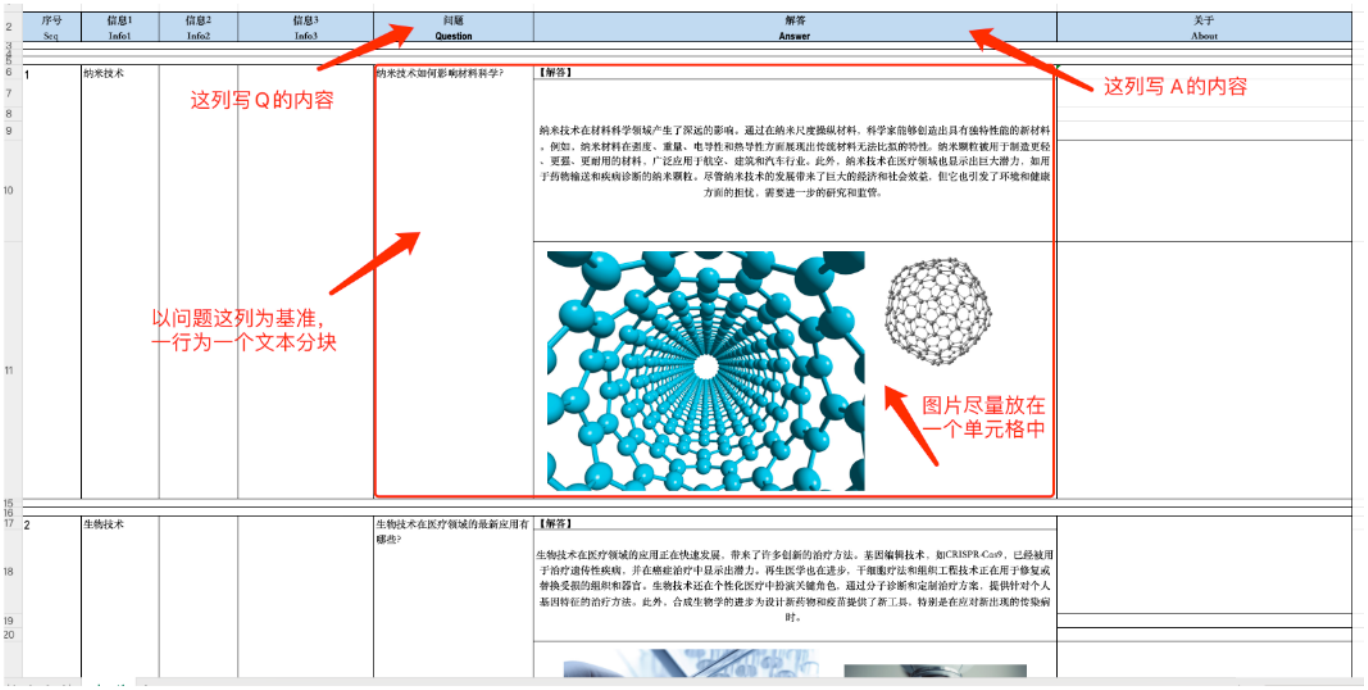
</split>

结束标签

对于 Docx 文档中的图片信息，整理的时候请直接粘贴到文档（不要使用形状或者文本框包裹图片）以免程序无法检测从而遗漏图片的处理。

预处理 xlsx 文档

xlsx 文件需要符合固定的模板格式：



Q：问题，A：答案。

对于 xlsx 文档，请按照模板要求整理，插图请尽量放一个在单元格中，尽量不要横跨几个单元格放置。

处理语料 准备环境

我们提供了基础镜像来处理图文：`release.daocloud.io/aigc/aitools:1.0`

- `/home/aitools/input` 替换成实际输入文件的目录
- `/home/aitools/output` 替换成实际输出处理后文件的目录

```
# 主机上创建输入、输出目录
mkdir -p /home/aitools/output /home/aitools/input
chmod 777 -R /home/aitools/output /home/aitools/input

# 运行常驻服务到后台
docker run -d -p 8888:8888 --name aitools \
    -v /home/aitools/input:/app/corpus_processing/input \
    -v /home/aitools/output:/app/corpus_processing/output \
    -e JUPYTER_TOKEN=aitools \
    --restart=always release.daocloud.io/aigc/aitools:1.0
```

处理数据

1. 文件上传到预设的输入目录 `/home/aitools/input`
2. 使用以下命令运行工具镜像中的脚本

```
docker exec aitools sh run.sh
```

导入处理好的文件

1. 点击 语料导入 -> 图文导入
2. 将处理好的文件上传，并进行向量化，等待处理成功即可

格式化导入

1. 在导入语料时选择 格式化导入。
2. 上传格式化文件，将数据向量化，这与上传数据的过程相同。

> **Note**
>
> 格式化导入 目前只支持 csv、xlsx 文件，并且要求文件内容格式如下。 如果是其他类型的文件，请正常使用上传数据的方式导入。

	A	B	C	D
1	seq	orgDoc	extDoc	
2	1	根据《中国人民银行 国家金融监督管理总局关于降低存	一、调整范围部分 1、2023年8月31日前我行已发放的和已签订合同但未发放的首套住房商业性个人住房贷款。 2、2023年8月31日前贷款发放或签订合同时套数性质为二套及以上，但借款人实际住房情况已符合所在城市首套住房标准的其他存量住房商业性个人住房贷款，具体情况包含： （1）房屋购买时家庭在当地没有其他成套住房，但因当地政府采取“认房又认贷”政策导致按照二套房贷利率标准办理，现在当地政府执行“认房不认贷”政策的； （2）房屋购买时不是家庭在当地唯一成套住房，但后期通过交易等方式出售了其他成套住房，本住房成为家庭在当地唯一成套住房且当地政府执行“认房不认贷”政策的； （3）其他满足所在城市首套住房利率标准的存量住房商业性个人住房贷款。	

分片插件实现与集成指南

本文档介绍了如何实现一个分片插件，并将其集成到系统中。

例如，您可以部署一个插件，该插件将接收 Markdown 格式的文本，然后将其按照标题进行分片，最后将插件添加到系统。现在就可以在语料库中导入 Markdown 格式的文本，并选择该插件进行分片。

实现分片插件

设计插件功能

在设计插件功能时，需要确定以下几点：

- 输入格式 ：插件将接收什么类型的文本作为输入？在本示例，我们将接收 Markdown 格式的文本作为输入。
- 分片标准 ：根据什么标准将文本进行分片？在本例中，我们将根据 Markdown 文档的标题进行分片。

实现插件功能

编写代码实现插件功能。下面是一个示例代码的框架：

```
from markdown_splitter import MarkdownSplitter

def import_markdown_file(file_path):
    # 读取 Markdown 文件内容
    with open(file_path, 'r', encoding='utf-8') as file:
        markdown_text = file.read()

    # 使用插件进行分片
    splitter = MarkdownSplitter()
    splitted_text = splitter.split_by_heading(markdown_text)

    # 处理分片后的文本
    # ...
```

将分片功能暴露成 http 服务，接口如下：

请求信息

- **HTTP** 方法 ：POST
- 请求 **URL** ： `/embeddings/FileSplitAndLocalEmbedDoc`
- 支持格式 ：JSON

请求体

请求体应包含待处理文档的内容。文档应采用如下 JSON 格式：

```
{
  "fileGetPath": "http://xxx/file.md",
  "params": {
    "key1": "value1",
    "key2": "value2"
  }
}
```

名称	类型	必填	描述
fileGetPath	string	是	文件路径
params	object	否	自定义参数， key 为 string，value 为 string

响应信息

- 状态码 ：200 OK
- 内容类型 ：JSON

响应体

响应体将包含处理后的嵌入结果，格式如下：


```
{
  "content": [
    {
      "orgDoc": "h1: 流沙之战\n作者：Micky Neilson\n\n正午的太阳坚定地凝视着希利苏斯的流沙，... 暴力。",
      "pageIndex": 0
    }
  ],
  "resultCode": "200800000001",
  "resultMessage": "success"
}
```


名称	类型	必选	中文名 说明
content	object	true	分片列表
orgDoc	string	true	分片内容
pageIndex	integer	true	分片页码
resultCode	string	true	状态码 200800000001 成功， 200800000002 失败
resultMessage	string	true	状态描述 success/failed

测试插件功能

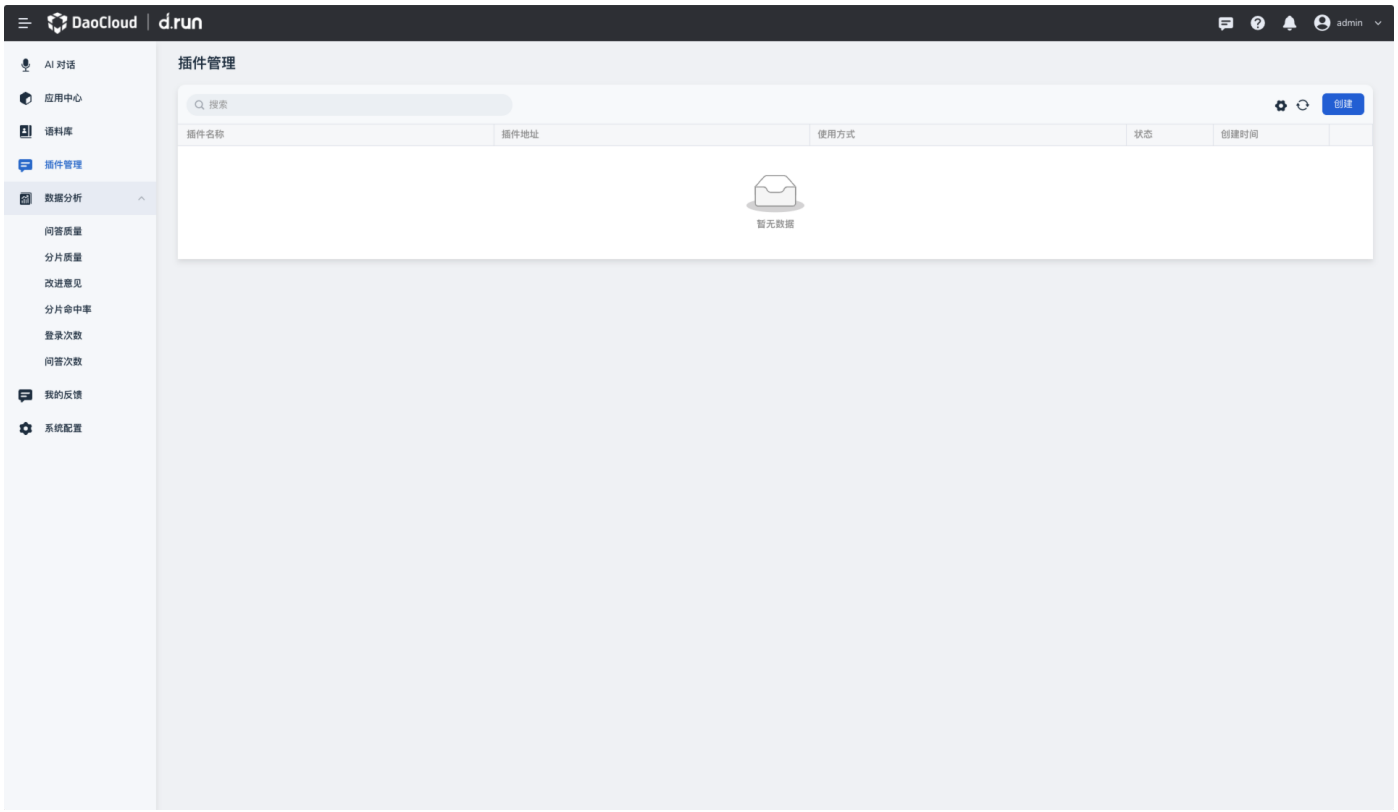
在实现插件功能后，务必进行测试以确保其正常运行。可以编写单元测试或手动测试来验证插件的正确性。

将插件集成到系统中

在系统中添加插件接口，使得语料库模块可以调用插件的功能。

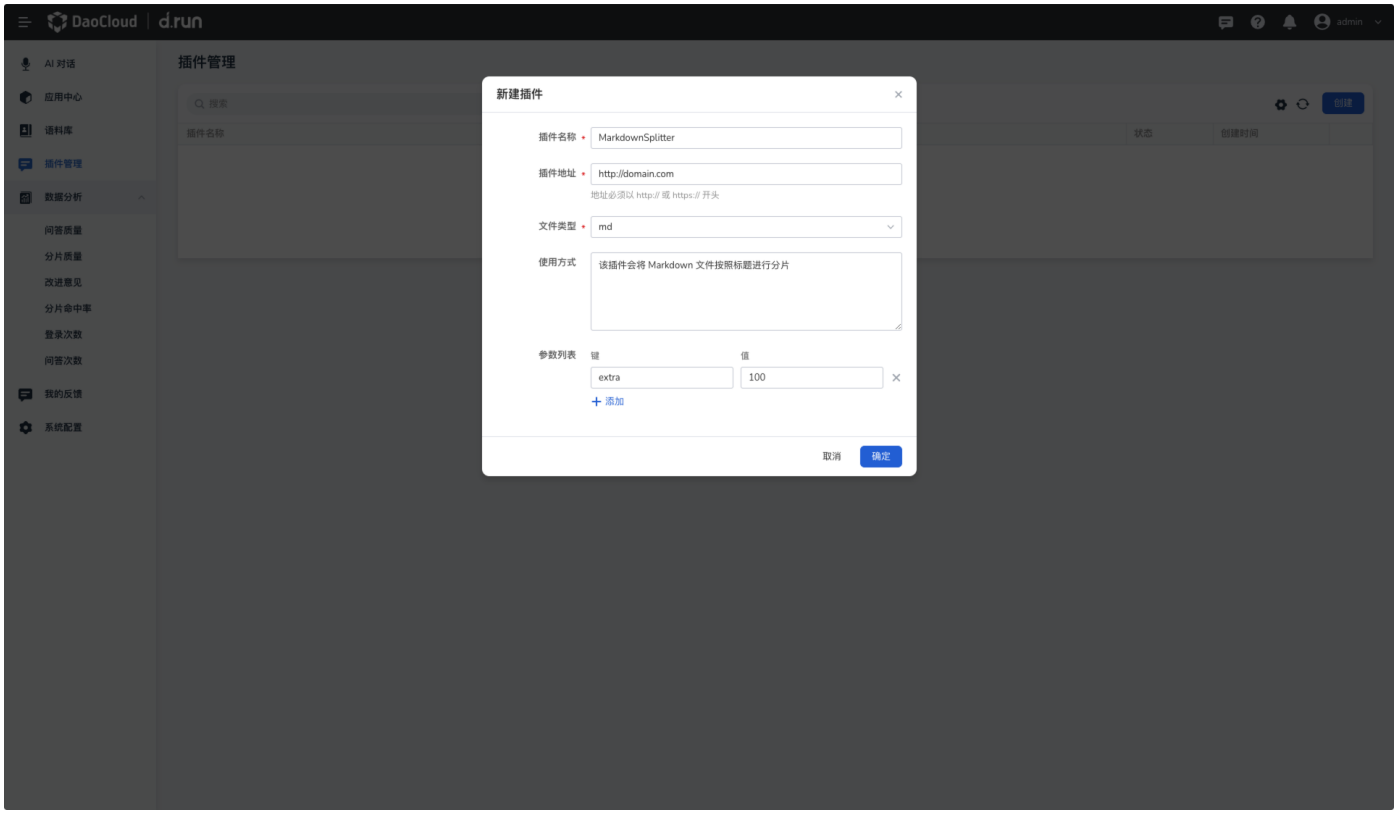
添加插件接口

1. 在 插件管理 页面中，点击 创建 按钮。



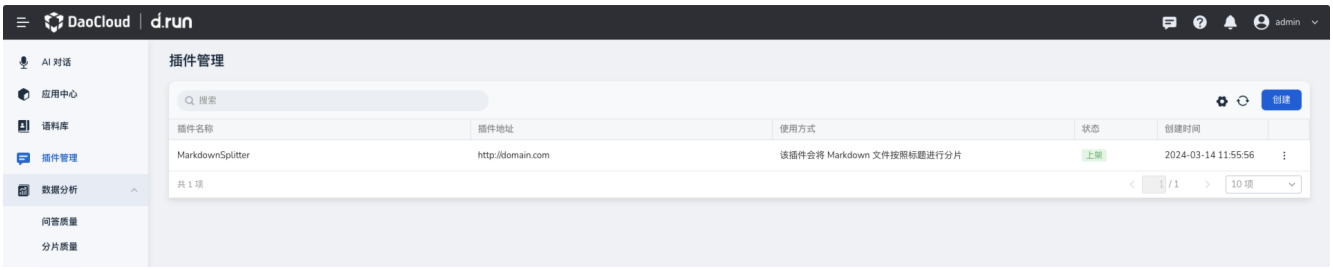
2. 参考下列要求填写插件信息，并点击 确定 。

- 插件名称 ：名称可以包含大小写字母、数字、连字符（"-"）、中文，最长 63 个字符
- 插件地址 ：插件的地址应以 `https://` 或者 `http://` 开头
- 文件类型 ：插件处理的文件类型，当前支持的文件类型包括 pdf、txt、docx、doc、csv、xlsx、ppt、md
- 使用方式 ：插件的功能介绍，可以包括分片的方法和其他功能说明
- 参数列表 ：键值对，描述在分片的时候需要填写对应参数的值，例如 key1:value1, key2:value2



Success

填写正确信息并点击确定后，页面会提示创建成功。



使用插件

在本例中，可以在语料库导入模块中调用插件功能，以在导入 Markdown 类型文件时进行分片。

数据分析

问答质量

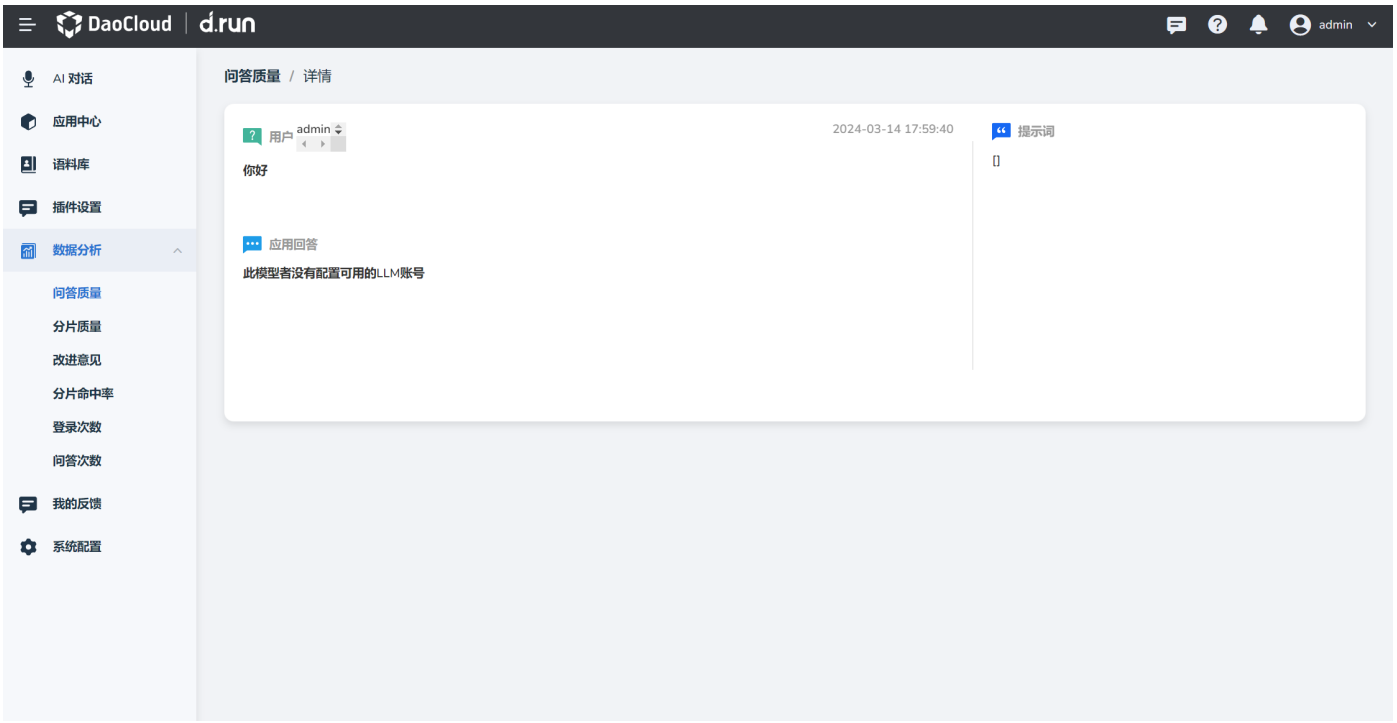
d.run 支持统计用户对问答质量的反馈，并以报表的方式展示，同样也支持将数据导出至xlsx中。

查看问答质量详情

1. 在 数据分析 栏中点击 问答质量 ,找到需要查看的问答,点击该问答记录可进入详情页面。

2. 可以查看以下内容：

- 用户提问内容，包括提问用户名称。
- 应用回答：应用针对用户问题产生的答案。
- 该问题使用的提示词。



3. 当有新的问答生成，可点击右上角 刷新 按钮查看最新问答详情。

导出问答质量

您可以将该工作空间内所有用户对问答质量的反馈汇总成表格导出。

1. 在 问答质量 页面点击右上角 导出 按键。

2. 导出在该工作该空间创建的应用，将对这些应用的问答的反馈汇总成xlsx文件并下载。

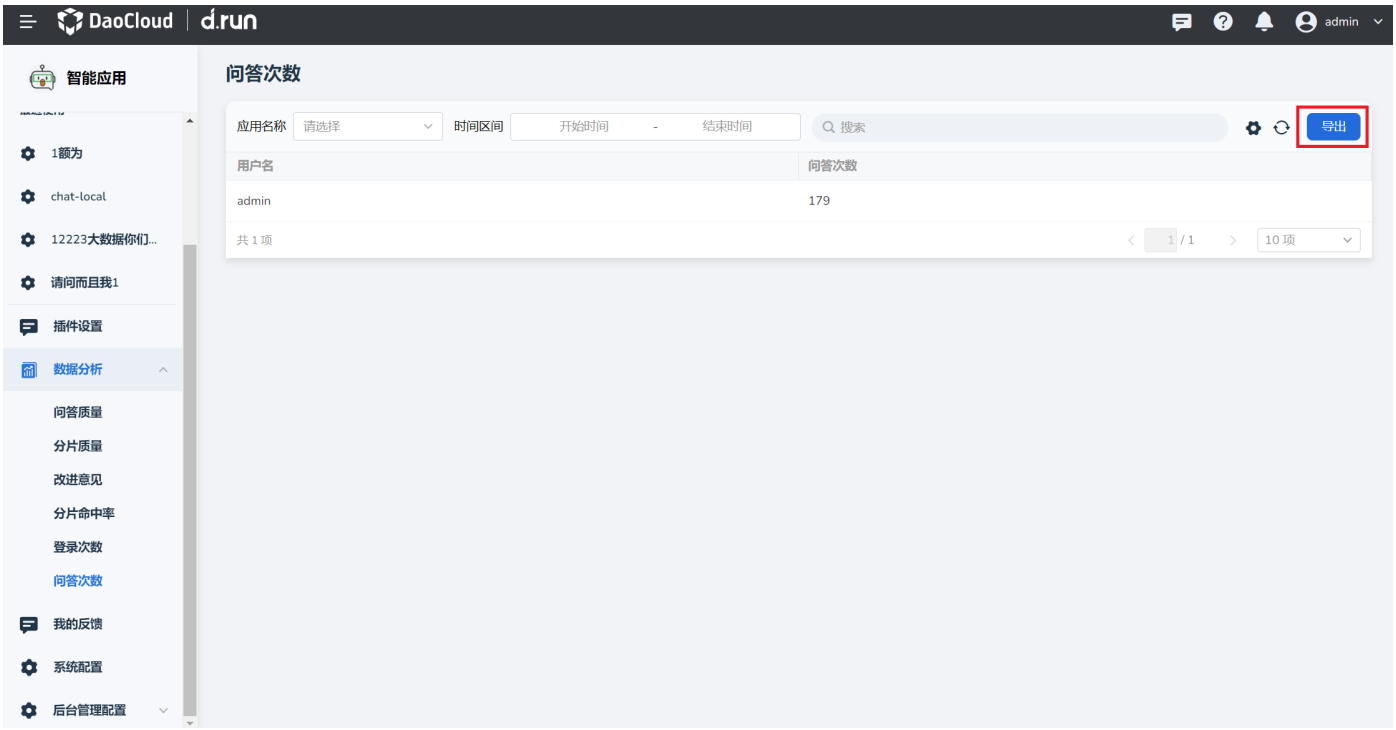
问答次数

d.run 支持记录在该工作空间创建的应用问答的次数。

当有用户进行了问答，记录会保存在创建该应用的工作空间下，点击右上角 刷新 按钮查看最新问答次数数据。

您可以将该工作空间内所有用户的问答次数汇总成表格导出。

1. 在 问答次数 页点击右上角 导出 按键。

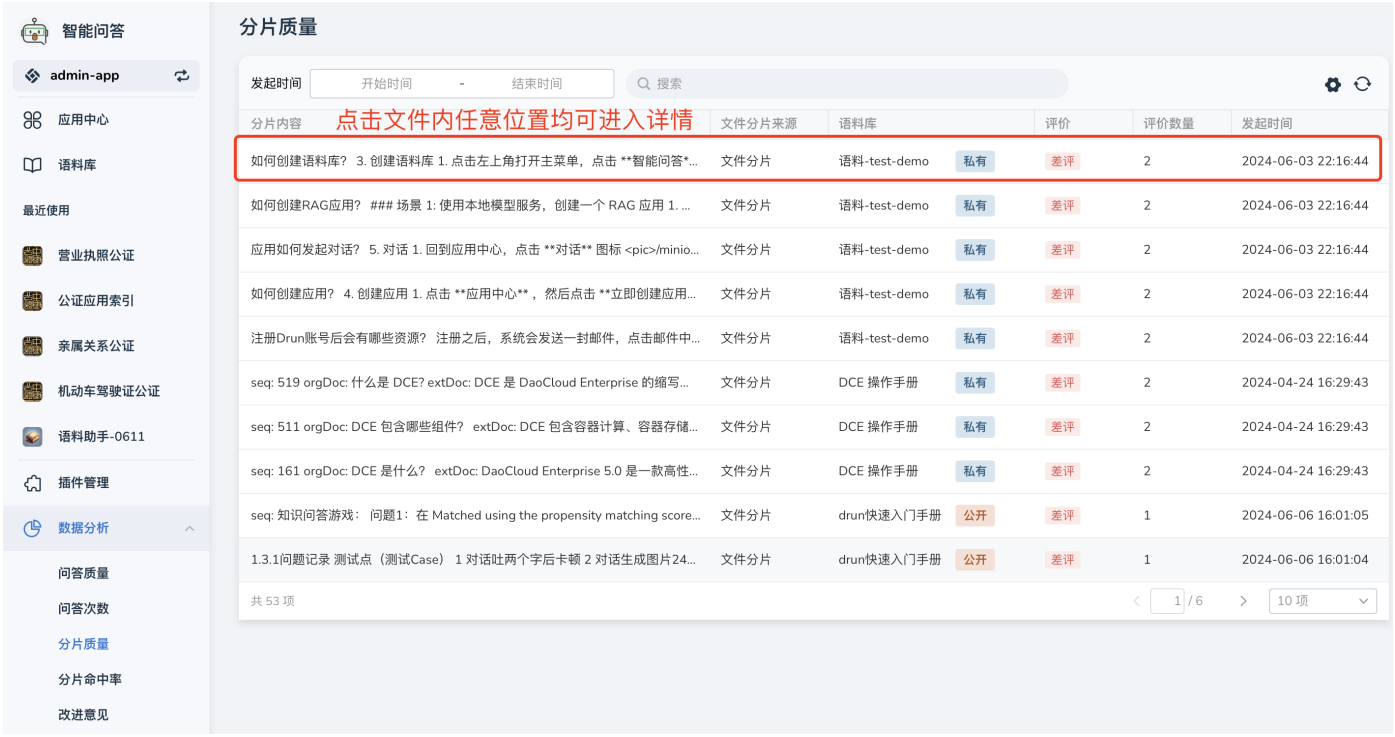


2. 将该工作空间下所有用户的问答次数汇总成xlsx文件并下载。

分片质量

分片是指将较大的数据切分为多个片段，便于训练模型查找和命中。 d.run 支持查看分片的质量。具体步骤如下：

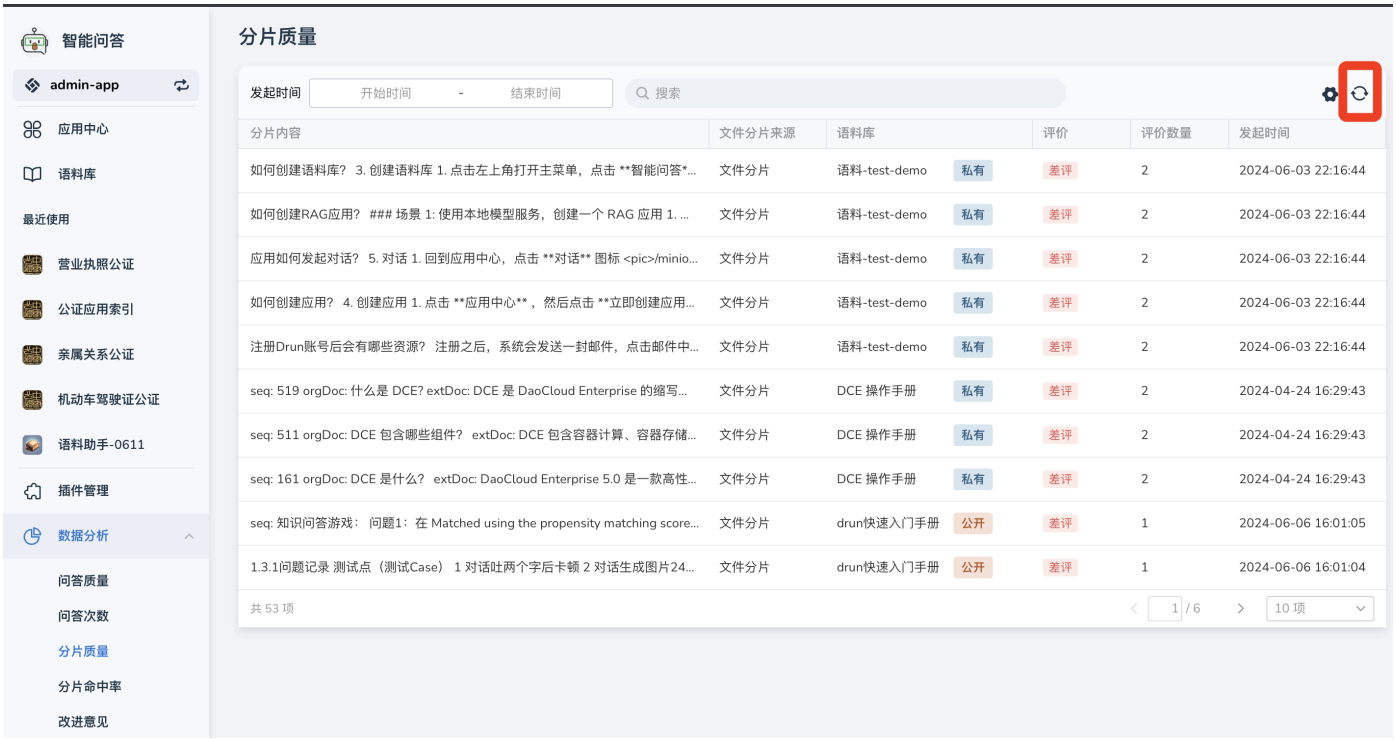
1. 在 数据分析 栏中点击 分片质量 ，通过 搜索 找到您关注的分片,点击该分片可进入详情页面，可查看该分片的详细内容。



2. 可以查看以下内容：

- 语料库：分片属于哪个语料库。
- 更新时间：该分片文件的更新时间。
- 分片 ID：分片的唯一识别码。
- 文件切片内容：切片后该分片的具体内容。
- 关联分片文本：与该分片有关的分片附加内容。

3. 当有新的分片文件被评价，可点击右上角 刷新 按钮查看最新分片文件。


智能问答
admin-app
应用中心
语料库
最近使用
营业执照公证
公证应用索引
亲属关系公证
机动车驾驶证公证
语料助手-0611
插件管理
数据分析
问答质量
问答次数
分片质量
分片命中率
改进意见
分片质量
发起时间
开始时间
结束时间
搜索
分片内容
文件分片来源
语料库
评价
评价数量
发起时间
如何创建语料库？ 3. 创建语料库 1. 点击左上角打开主菜单，点击 **智能问答*... 文件分片 语料-test-demo 私有 差评 2 2024-06-03 22:16:44
如何创建RAG应用？ ### 场景 1: 使用本地模型服务，创建一个 RAG 应用 1. ... 文件分片 语料-test-demo 私有 差评 2 2024-06-03 22:16:44
应用如何发起对话？ 5. 对话 1. 回到应用中心，点击 **对话** 图标 <pic>/minio... 文件分片 语料-test-demo 私有 差评 2 2024-06-03 22:16:44
如何创建应用？ 4. 创建应用 1. 点击 **应用中心**，然后点击 **立即创建应用... 文件分片 语料-test-demo 私有 差评 2 2024-06-03 22:16:44
注册Drun账号后会有哪些资源？ 注册之后，系统会发送一封邮件，点击邮件中... 文件分片 语料-test-demo 私有 差评 2 2024-06-03 22:16:44
seq: 519 orgDoc: 什么是 DCE? extDoc: DCE 是 DaoCloud Enterprise 的缩写... 文件分片 DCE 操作手册 私有 差评 2 2024-04-24 16:29:43
seq: 511 orgDoc: DCE 包含哪些组件？ extDoc: DCE 包含容器计算、容器存储... 文件分片 DCE 操作手册 私有 差评 2 2024-04-24 16:29:43
seq: 161 orgDoc: DCE 是什么？ extDoc: DaoCloud Enterprise 5.0 是一款高性... 文件分片 DCE 操作手册 私有 差评 2 2024-04-24 16:29:43
seq: 知识问答游戏： 问题1：在 Matched using the propensity matching score... 文件分片 drun快速入门手册 公开 差评 1 2024-06-06 16:01:05
1.3.1问题记录 测试点（测试Case） 1 对话吐两个字后卡顿 2 对话生成图片24... 文件分片 drun快速入门手册 公开 差评 1 2024-06-06 16:01:04
共 53 项
1 / 6
10 项

分片命中率

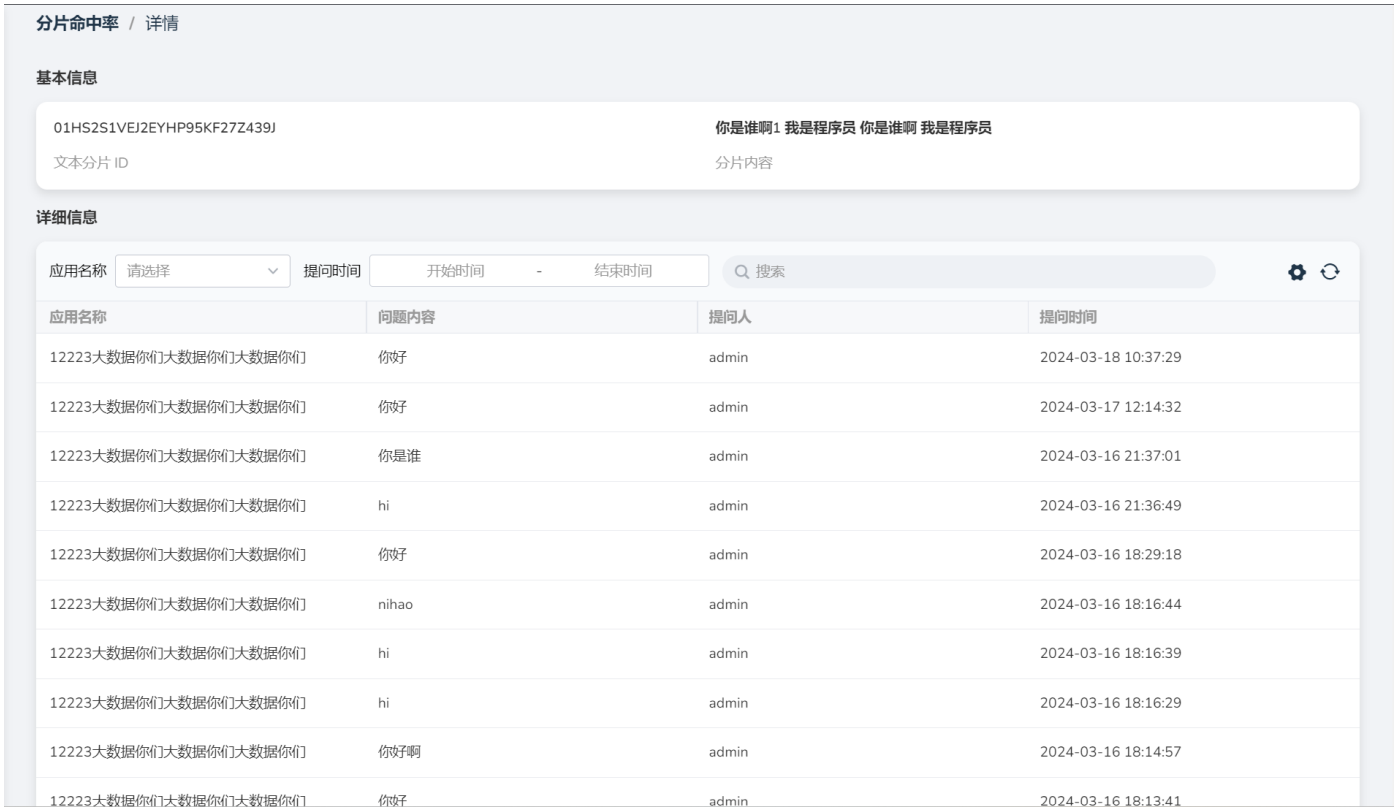
d.run 统计了用户对于分片数据的使用情况，将分片按命中次数排序，支持查看分片内容并将分片的使用情况导出到xlsx中。

查看分片命中率详情

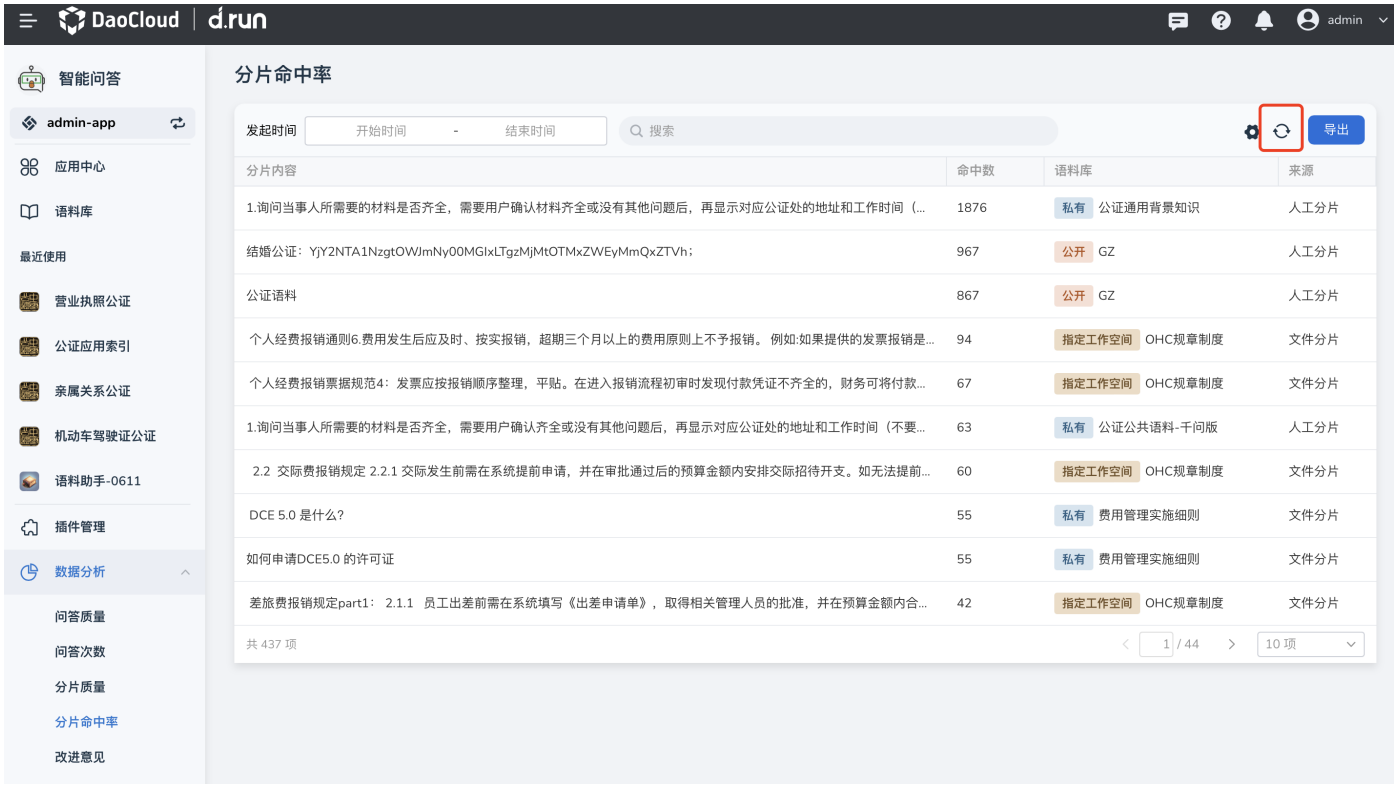
1. 在 数据分析 栏中点击 分片命中率 ，通过 搜索 找到您关注的分片,点击该分片可进入详情页面，可查看该分片被引用的详细情况。

2. 可以查看以下内容：

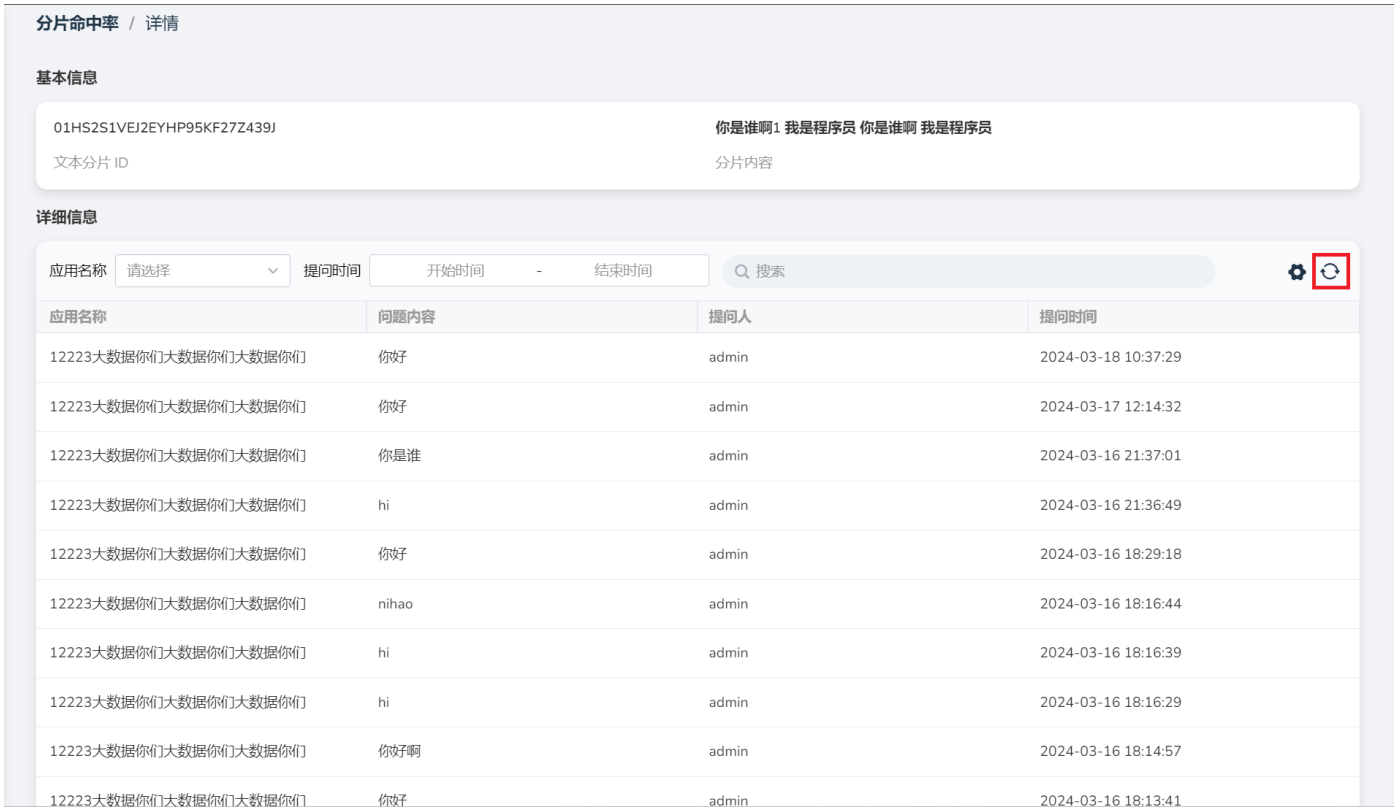
- 分片的基本信息：包括文本分片ID以及具体的分片内容。
- 详细信息：该分片被问题命中的问题内容、应用名称、提问人以及提问时间。



3. 当有新的问答生成，可点击右上角 刷新 按钮查看最新分片命中率结果。

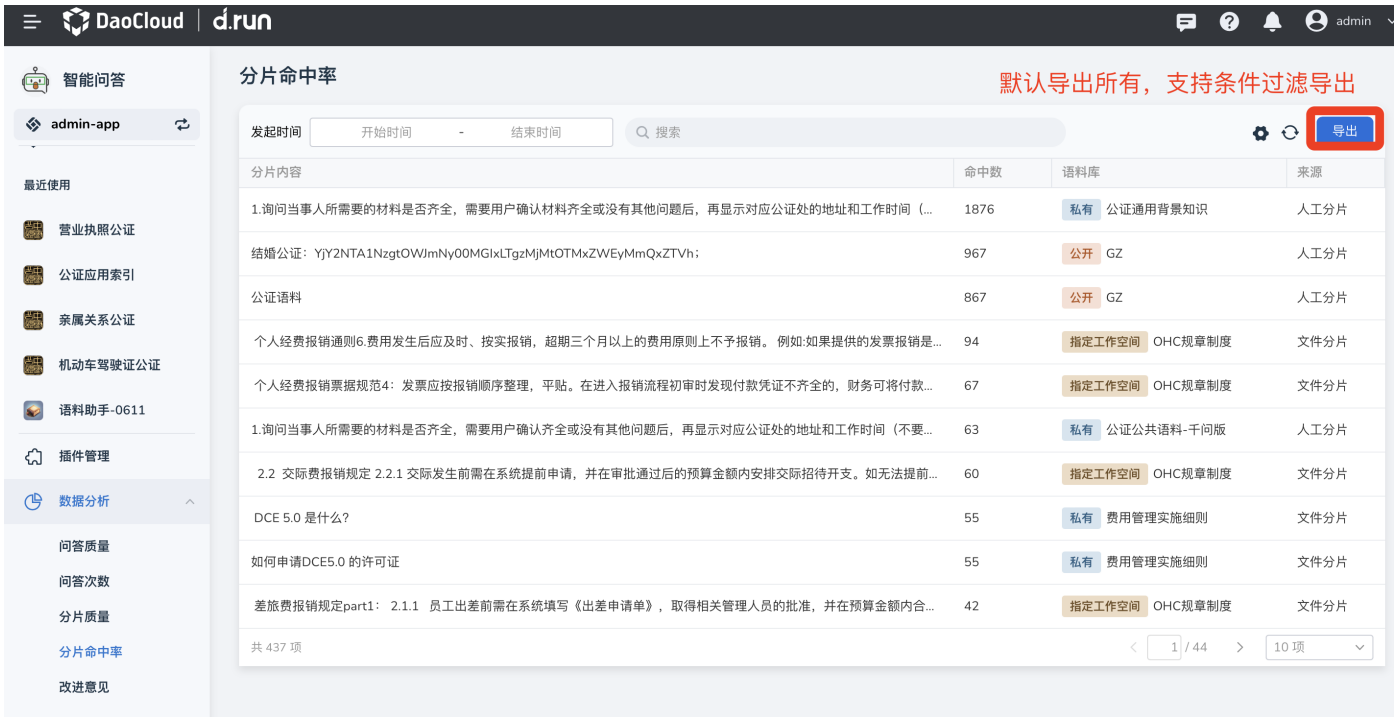


分片命中率详情中也可 刷新 查看最新结果。



导出分片命中率

1. 在 分片命中率 页点击右上角 导出 按键。



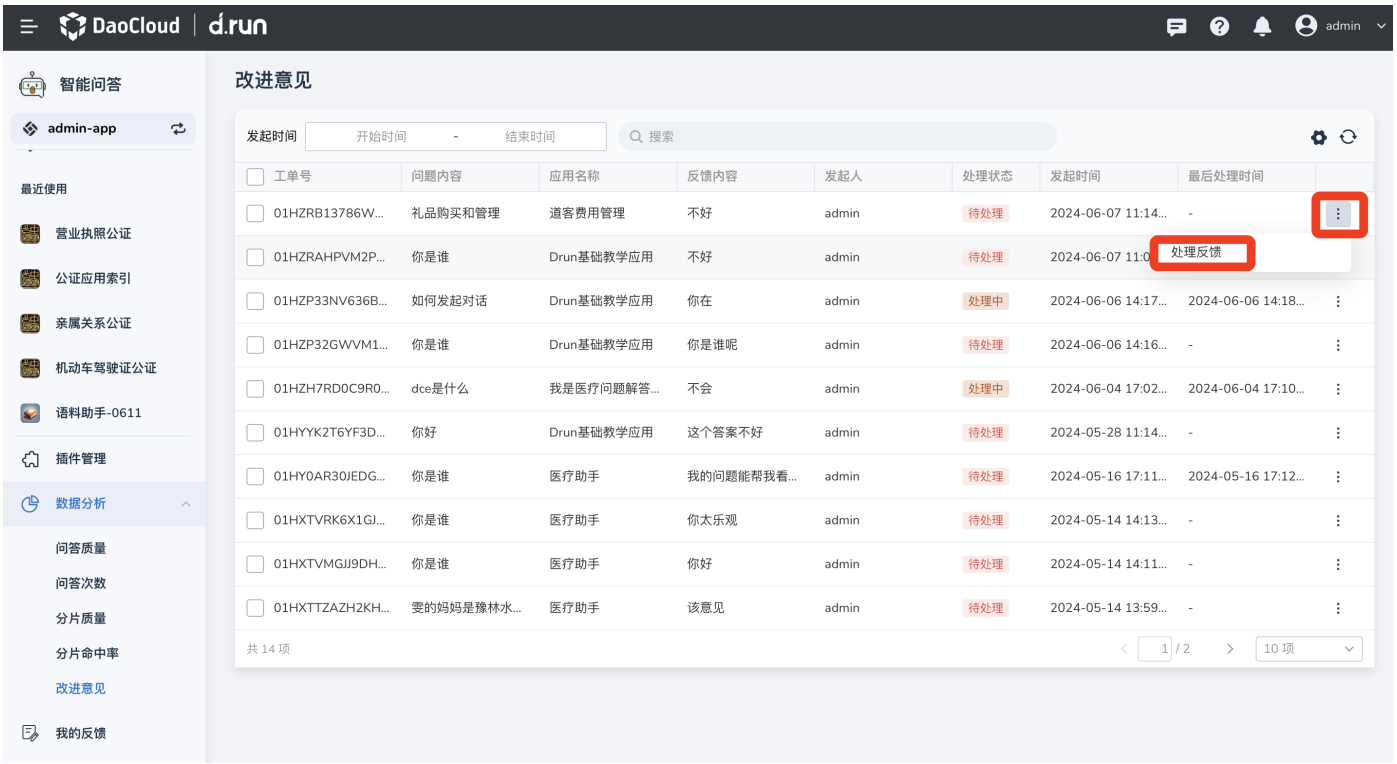
2. 导出在该工作该空间创建的分片，将对这些对分片使用情况的反馈汇总成xlsx文件并下载。

改进意见

d.run 提供了用户提交反馈的入口，管理员可在此处看到在这个工作空间的应用在对话上有哪些问题需要改进。

处理反馈

1. 在 数据分析 栏中点击 改进意见 ，找到需要处理的反馈，点击右侧的 ⋮ 按钮。
2. 在弹出菜单中选择 处理反馈（在范围内点击工单信息可进入详情界面）。

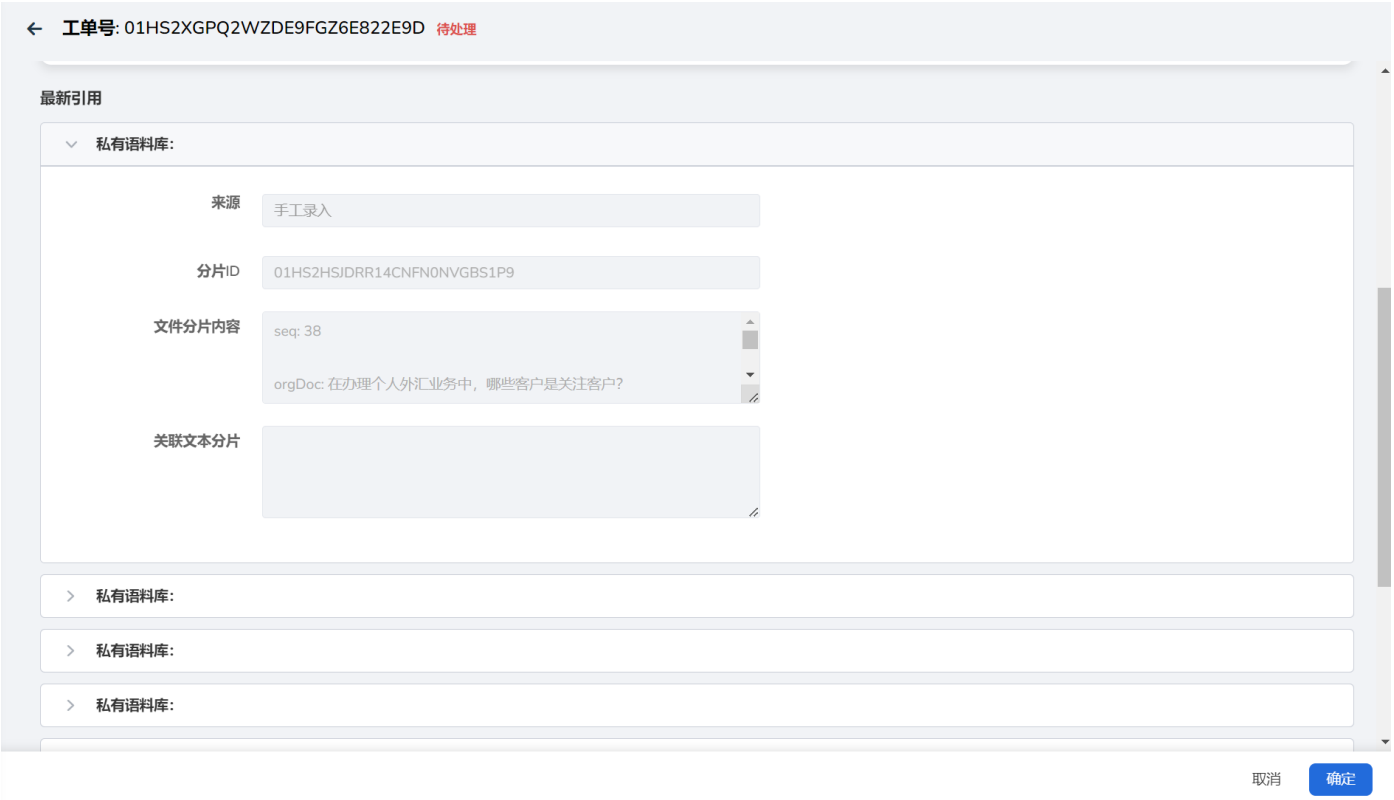


3. 可查看以下内容：

• 用户对问答的反馈内容。

• 问答的相关信息：助手名称、模型名称、引用条数、处理时间以及问答详情。

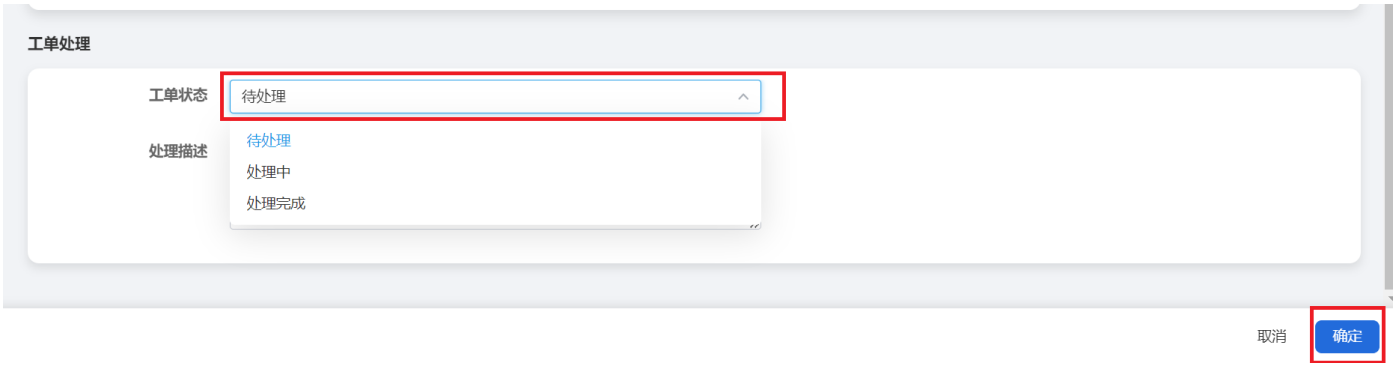
• 最新引用：可查看回答所有引用的分片详情。



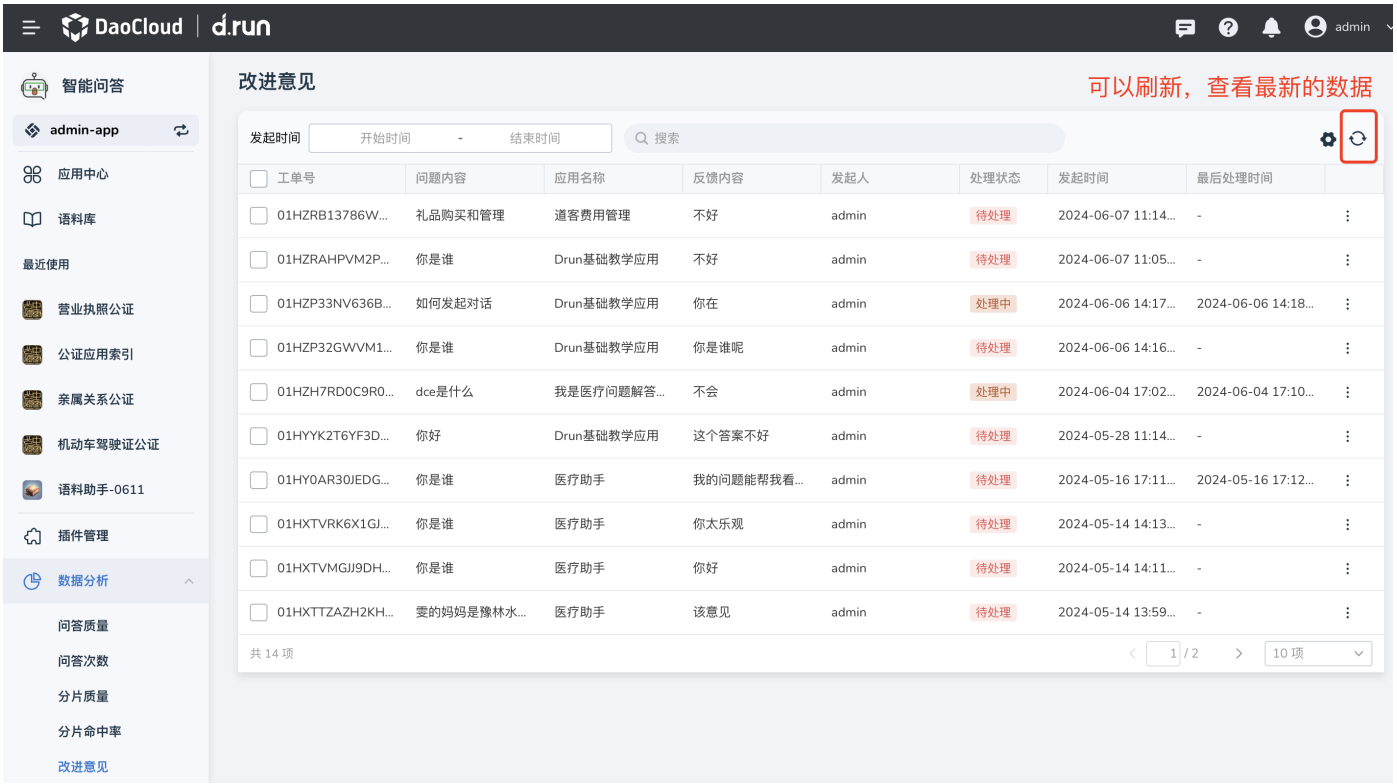
• 工单处理的详情。

4. 点击工单状态后的 待处理 ，可将状态调整为 处理中 或 处理完成 ，并填写下方的 处理描述 。

5. 点击右下角 确定 即可修改反馈处理状态。



6. 当有新的改进意见提交，可点击右上角 刷新 按钮查看最新的改进意见。



导出反馈

将所有问答的详细内容以及问答反馈汇总成表格导出。

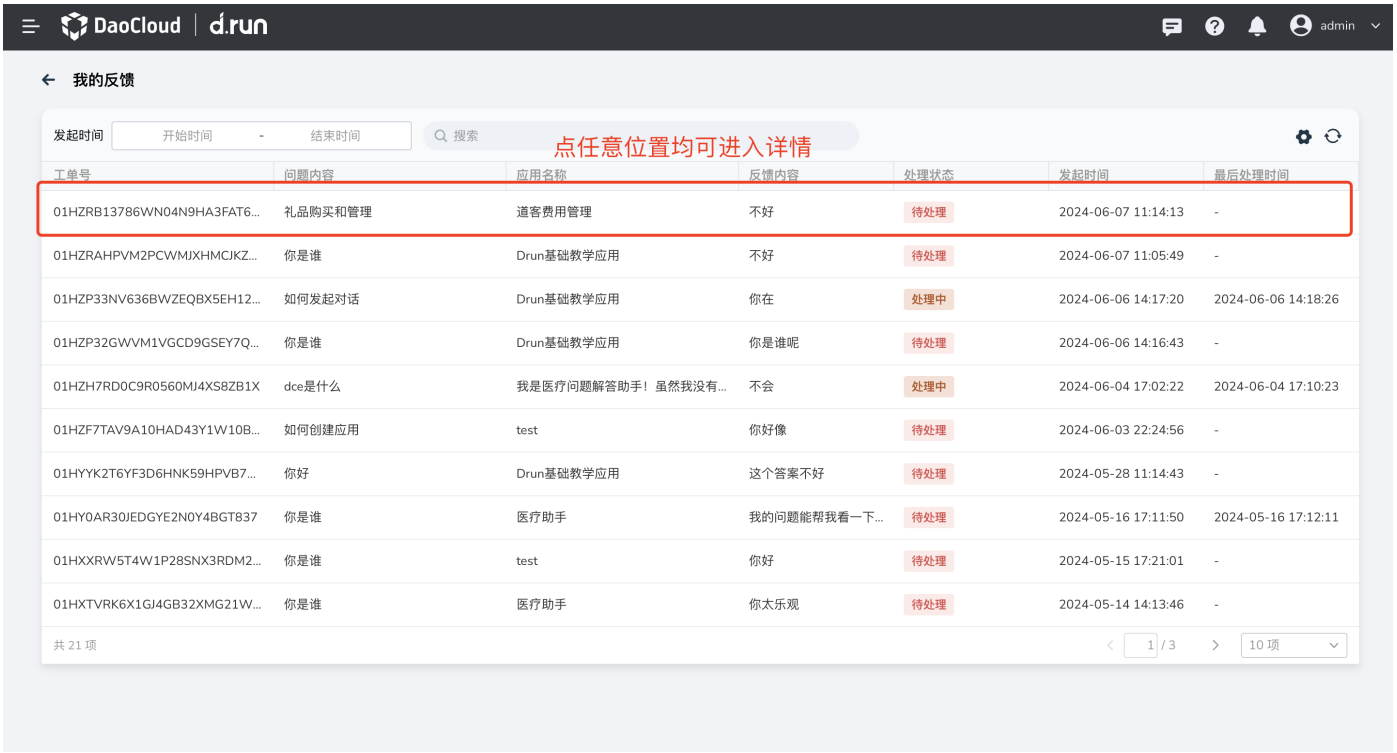
1. 点击 问答反馈左方 ，选取需要导出的反馈，点击右上角 导出 按键。
2. 将问答内容汇总为 xlsx 文件并下载。

我的反馈

d.run 记录了当前用户发出的反馈信息，以方便查看后续该反馈得到的回馈。

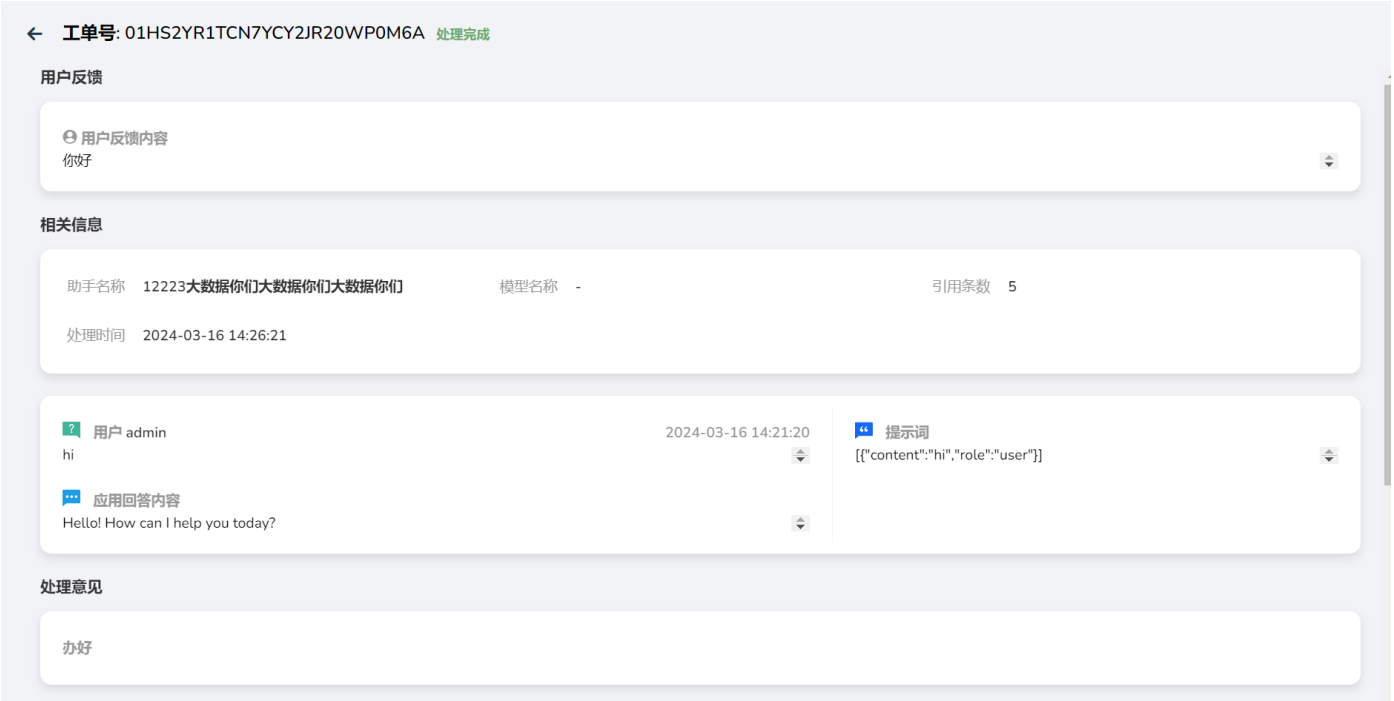
查看-我的反馈-详情

1. 点击 我的反馈 ，可通过搜索找到需要查看的反馈，点击范围内任意位置均可进入详情。

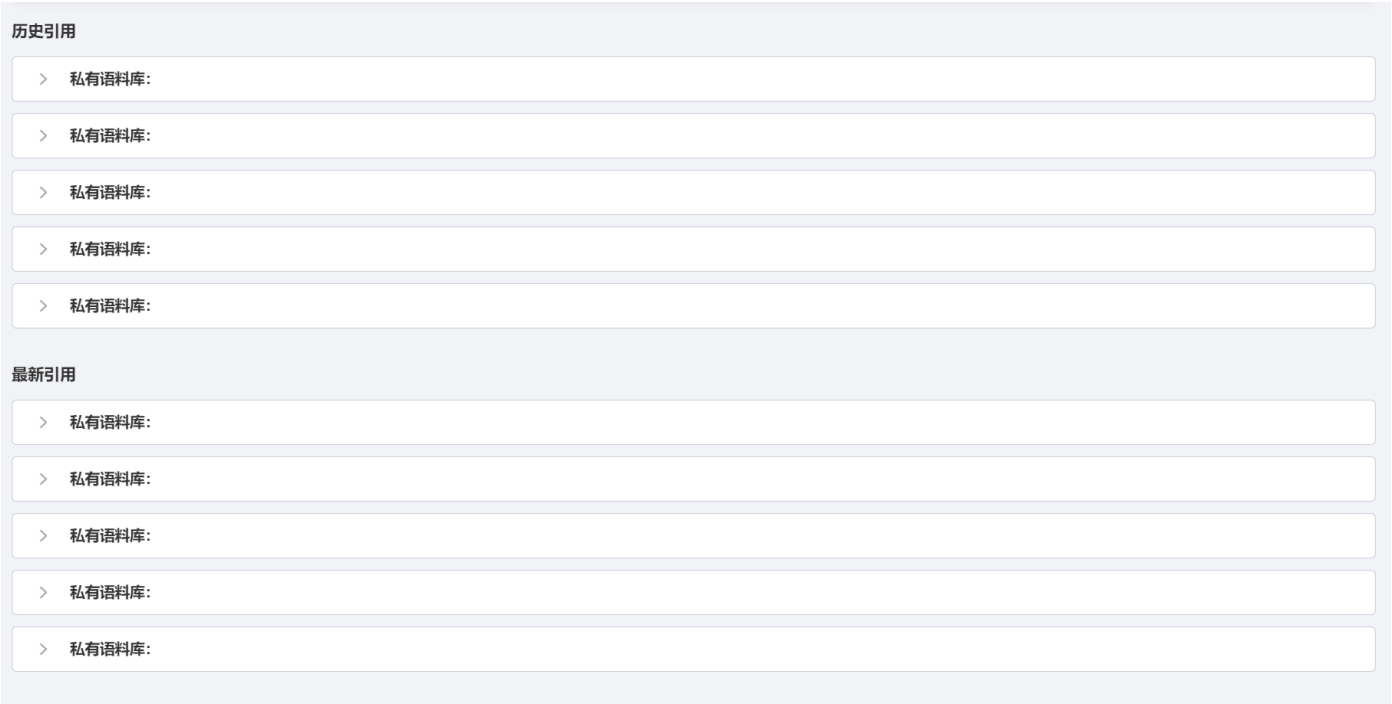


2. 可以查看以下内容：

- 用户反馈：用户本人发出的反馈内容。
- 相关信息：助手名称、模型名称、引用条数、处理时间以及问答详情。
- 处理意见：管理员处理反馈后发出的意见。

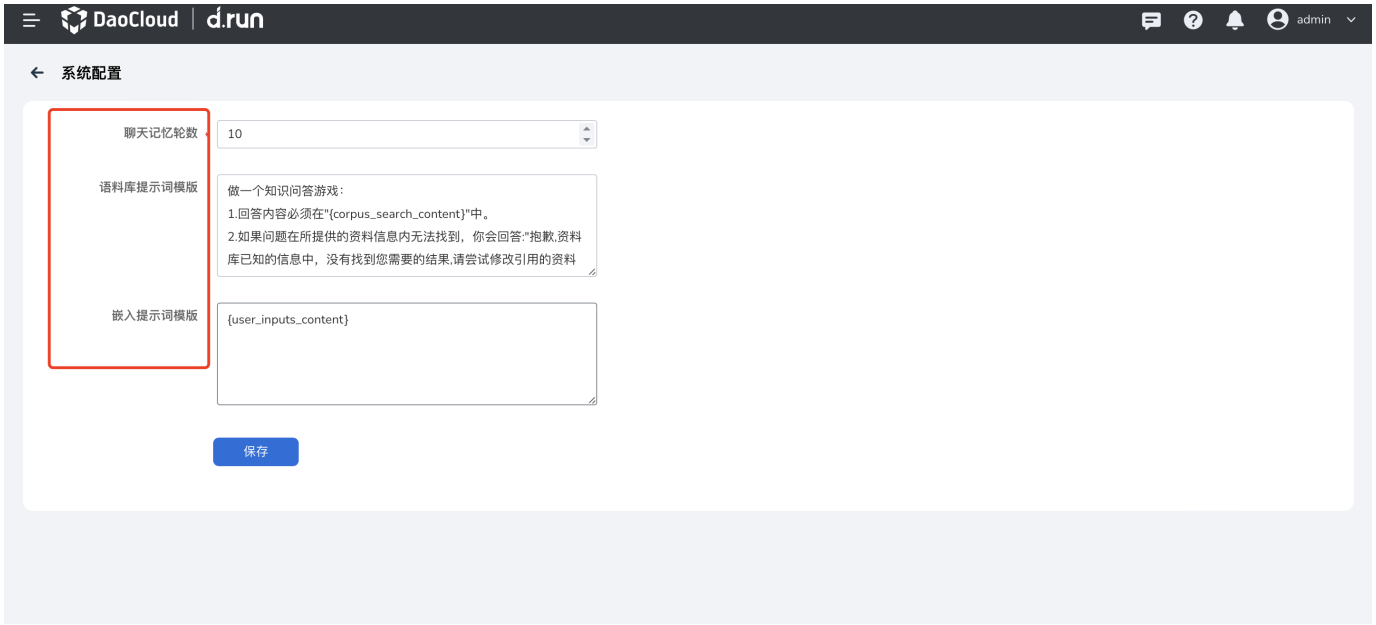


- 历史引用。
- 最新引用。



系统配置

> **Note**
>
> 系统配置中的参数将会作为创建应用时的默认参数。



聊天记忆轮数

指在一个对话系统中，用于追踪和管理对话历史的轮数或回合数。每当用户与系统进行一次交互，对话的轮数就会增加一次。 此处指用户与聊天应用进行交互时，应用会在一个小记忆窗口中保持一段时间的对话历史，记忆轮数大小即保持时间的长短。

- 默认值为 10
- 最低为 0

语料库提示词模板

该模板在应用关联使用模板时启用，模板内容包含通过相似度搜索得到的知识块、用户的输入和嵌入提示词模板。

嵌入提示词模板

嵌入的提示词拼接在用户的问题之前，作为通用的约定提示来引导应用模型输出针对问题的回答。

1.3.2 流程编排

流程编排

这是一个流程编排工具，通过智能化流程创建 AI 应用。

大模型为核心的生成式人工智能有机会产生巨大的经济价值。 但基于 ChatGPT 等现有产品直接使用，很难满足具体的大模型应用需求，痛点体现在：与真实业务场景脱节、无法连接业务逻辑和已有应用、难以协同优化、数据安全不可控等。

DaoCloud 围绕大模型应用构建生命周期管理，提供端到端解决方案，十倍加速大模型应用需求产生、构建、迭代和高可用部署， 实现 有了需求之后，当天完成应用初步构建，当周实现应用商品级可用，当月创造商业价值 。

在需求产生环节，DaoCloud 团队基于对行业头部客户的服务经验，结合技术、产品和商业价值，提供可实现的潜在应用场景参考， 通过工作坊等共创方式引导一线员工产生有效需求，并进入 应用构建-应用迭代-应用部署 的周期。

- 在应用构建环节，DaoCloud 流程编排基于 DaoCloud 首创的「工具-流程-应用」的新范式， 实现将大模型与企业已有的业务系统和业务数据连接起来，形成有效流程，进而变成解决问题的表单、对话或者 API 应用。
- 在应用迭代环节，DaoCloud 流程编排基于流程应用天然适合可观测的特点，用「黑箱 TDD 迭代-白箱逐步精调」的方式优化流程， 再结合经过学术界验证的「构建指标-构建多条流程-自动化测评-找到最优流程」的方法论，实现多人协同的可视化应用迭代，快速接近生产可用。
- 在应用高可用部署环节，DaoCloud 流程编排平台在团队对云原生的深刻理解基础上，构建最优大模型应用系统架构， 实现大模型相关服务的弹性伸缩和多副本的负载均衡，最终实现计算资源的高效利用和高可用。

参阅演示视频：

注册并体验 **d.run**

功能特性

懂业务的大模型流程引擎，零代码构建高价值 AI 应用，提供流程可视编排、标准流程定义、构建终端应用、应用可观测性、统一工具实现。 具体的功能特性见下表：

一级功能	二级功能	三级功能	描述
应用	应用构建	构建流程应用	在流程中直接观测每个节点的输出
		构建表单应用	基于一条流程的输入和输出，实现填写表单后生成对应的结果
		构建对话应用	基于一条流程的输入和输出，实现一个聊天机器人
		构建 RESTful API 应用	基于一条流程的输入和输出，实现一个 API
		构建 OpanAI API 应用	当一个流程只有一条输出时，形成兼容 OpenAI 的 API
	应用集成	将应用集成到企业微信	基于 RESTful API 接入企业微信
		将应用集成到微信客服	基于 RESTful API 接入微信客服
		将应用集成到微信公众号	基于 RESTful API 接入微信公众号
	应用可观测性	查看单个应用的总体运行情况	查看整体运行次数、运行时间等信息
		查看单个应用的单次运行情况	查看单次运行对应的流程中，每个流程中的节点输入、输出和运行时间等信息
	应用智能	需求自动匹配已有工具	基于 Function Call 特性
		需求创建新流程（直接生成）	基于 APA 方法，同时通过 RAG 利用已有数据
		需求创建新流程（自回归生成）	基于 ReAct 方法
	典型应用示例	多模态内容生成（AIGC）	连接文本生成、图像生成、视频生成等工具节点，形成应用
		信息检索增强生成（RAG）	连接关系数据搜索、向量搜索、稀疏搜索、大语言模型等工具节点，形成应用
		流程自动化（RPA）	连接已有业务系统和大语言模型等工具节点，形成应用
		数据清洗、模型训练和测评（ETL）	连接文档处理、大语言模型、模型训练、模型测评等工具节点，形成应用
流程	流程声明式语法	标准 Schema 定义	基于主流流程语法定义，结合大语言模型特点定义语法
		支持循环、选择等 DAG 能力	基于 DAG（有向无环图），实现流程编排的图灵完备性
	流程可视化配置	流程配置	快速组装流程形成流程应用
		流程参数传递	将不同的工具节点进行参数传递，实现数据流转
	流程控制	控制流程启停	根据用户在流程应用、表单应用、对话应用的输入，启动流程或者停止流程
		流程节点超时重试	根据单个工具节点的运行情况进行控制，确保流程的顺利运行
		流程日志监控	记录流程运行的全部数据并进行可视化呈现
	流程智能	需求创建新流程（直接生成）	基于 APA 方法，同时通过 RAG 利用已有数据

一级功能	二级功能	三级功能	描述
		需求创建新流程（自回归生成）	基于 ReAct 方法
		需求修改已有流程	利用模型的推理能力，基于自然语言进行修改
工具	工具声明式语法	标准 Schema 定义	基于主流工具语法定义，结合大语言模型特点定义语法
	工具列表	工具列表	将预置工具和自定义工具统一管理并调用
	预置工具（大语言模型）	多种模型兼容	兼容 Llama、Command 等海外模型；兼容 Baichuan、GLM、Qwen 等国内模型
		多种量化方式兼容	兼容 INT16、INT4 等；兼容 GPTQ 等
		模型列表	统一管理全部模型，包括微调后的模型
	预置工具（多模态模型）	视觉语言模型	兼容 QwenV 等多种视觉语言模型
		图像生成模型	兼容 Stable Diffusion 等多种图像生成模型
		视频生成模型	兼容 Stable Video Diffusion 等多种视频生成模型
	预置工具（数据处理）	关系数据处理	将已有的关系数据库储存到可被搜索的表格数据集中
		文本数据处理	将已有的文本储存到可被搜索的预置文本数据集中
		多种数据结构转换	实现字符串等多种数据结构的互相转换和存储
	预置工具（私有数据搜索）	关系数据搜索	支持通过 SQL 查询已有数据库（SQL 可以通过大语言模型生成）
		文本数据语义搜索	支持通过一段文本查询已有数据库（向量搜索）
		文本数据稀疏搜索	支持通过一段文本查询已有数据库（关键词搜索）
	预置工具（文件处理）	文件读取	支持读取 pdf、docx、xlsx 等数据格式
		文件写入	支持写入 pdf、docx、xlsx 等数据格式
	预置工具（自动化）	营销自动化	支持实现对主流公域营销渠道和私域渠道的自动化推送
		办公自动化	支持实现对主流信息化系统的自动化打通
	预置工具（模型训练）	增量预训练	支持通过增量预训练方式向模型引入数据
		指令微调	支持通过指令微调方式向模型引入数据
		强化学习	支持通过强化学习方式向模型引入数据
	预置工具（模型测评）	客观测评	支持基于有明确答案的问题进行测评，并输出测评报告
		主观测评	支持对于主观问题进行测评，并指定任意的大语言模型进行测评，并输出测评报告
	自定义工具	子流程	将已有流程作为一个节点
		API	将任意满足 OpenAPI 条件的 API 作为一个节点
		Python 代码	将一段 Python 代码作为一个节点
		JavaScript 代码	将一段 JavaScript 代码作为一个节点

一级功能	二级功能	三级功能	描述
		外部服务	在外部服务提供统一接口的情况下，接入作为一个节点

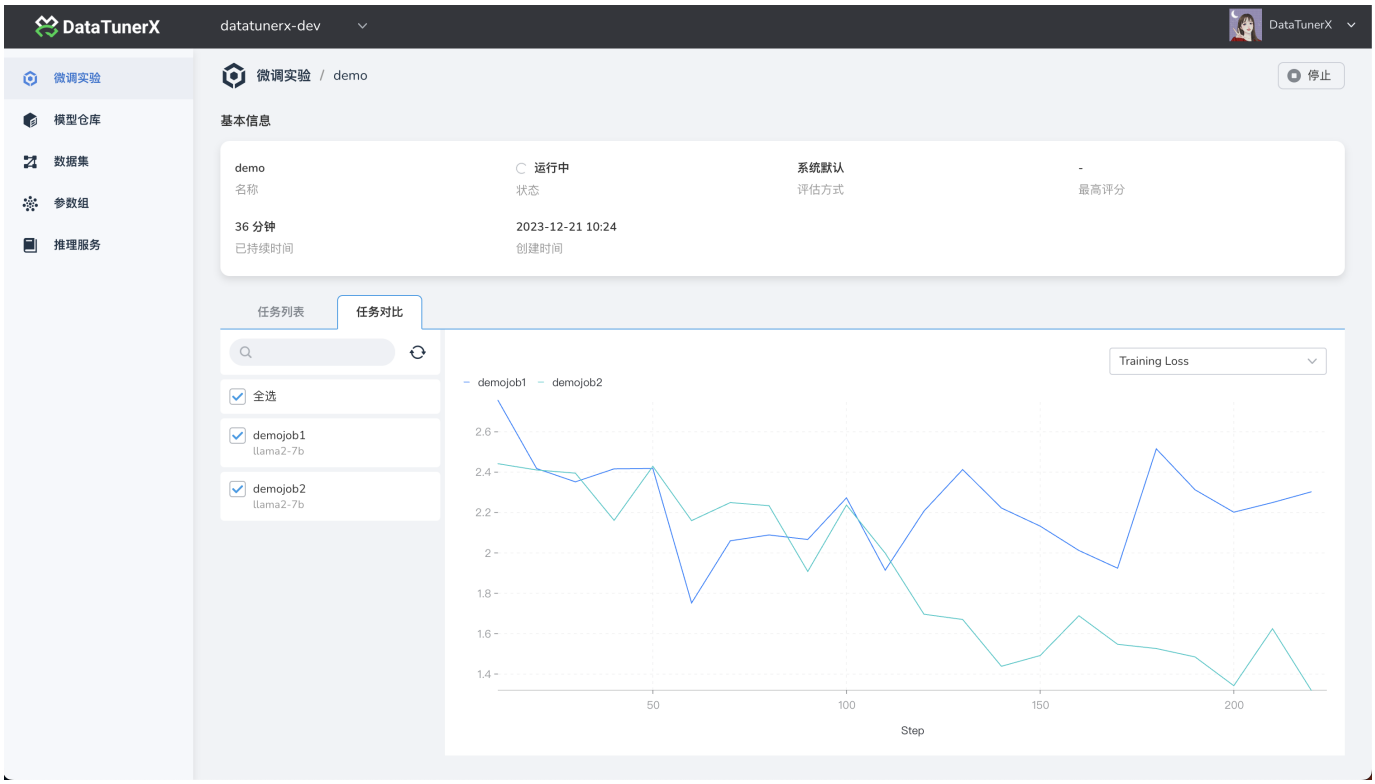
1.4 模型工具

1.4.1 模型微调

什么是模型微调

DaoCloud DataTunerX（DTX）是一站式自动化平台，专注于大型语言模型微调。 涵盖了数据集、超参组、模型仓库、模型微调、模型评估和模型推理的全生命周期，实现了高度自动化。 通过高效利用底层分布式算力资源，DTX 能够进行矩阵式的模型微调实验，从而推动大型模型的敏捷和自动化迭代。

DTX 模型微调是一个云原生解决方案，旨在与分布式计算框架集成。 利用可扩展的 GPU 资源，该平台专注于高效微调 LLM 模型。 其核心优势在于促进批量微调任务，使用户能够在单个实验中同时进行多个任务。



几个术语

云原生（**Cloud Native**）： 基于 Kubernetes 的 CRD 和 Operator 设计和开发的，一套围绕大语言模型、数据处理、 大语言模型微调、评估、并行等能力的开源解决方案。让大模型的微调能力变成云原生应用的能力。

数据集（**Dataset**）： 与大语言模型微调中的数据集定义基本差不多，这里主要提到的是训练集（用于大语言模型微调的主要微调数据，也就是想让大语言模型能够学会的数据）， 验证集（在微调过程中用于验证大语言模型微调效果的数据），测试集（用于在大语言模型这次微调结束之后，用于验证本地微调的效果的数据）。

基础大语言模型（**Base Large Language Model**）： 大语言模型（LLM）是指使用大量文本数据训练的深度学习模型，是一种用于自然语言处理的深度学习模型， 它通过大规模的预训练来学习自然语言的表示和语言模式。这些模型可以用于各种任务，如文本生成、文本分类、机器翻译等，以生成或处理自然语言文本。 基础大语言模型就是微调过程中，被微调的底座的大语言模型。

参数组（**Hyperparameter Group**）： 代表一组超参的集合。在大语言模型的微调过程中，可以根据需要设置很多不同的超参， 如 Scheduler、Optimizer、LearningRate、Epochs、BlockSize、BatchSize 等，这里的参数组就是各种超参的组合。

评估（**Scoring / Evaluation**）： 用于对大语言模型进行评定好与不好的一种方式。还有一种专业点的说法叫 metrics。 评估中支持很多种不同的 metric 类型，如 Bleu、Accuracy、Precision、Recall、Rouge、F1 等，不同类型的大模型任务需要使用不同的 metric 算法进行评估。评估的时候也需要特定格式的数据来进行评估。

微调实验（**FinetuneExperiment**）： 实验，顾名思义就是要做一次尝试。可以使用数据集、大语言模型、参数组进行不同的组合进行实验， 指定本次实验的评估方式，每个组合就是一个微调任务。最终根据这次实验的所有组合进行微调之后，给每个微调任务进行打分，评估出这次实验微调出来最好的模型。 在这个过程中是并行地使用所有可以使用的算力，我们知道大语言模型的微调是存在随机性，在这个背景下，卡越多，每次实验并行数越多，每次实验结束， 出现优秀模型的概率就越大。这是用算力资源换取时间的方式，来提升整体的微调效率。下次微调就可以基于上次评估出来最好的模型进行再次微调。

微调任务（**FinetuneJob**）： 一次微调任务就是使用指定的数据集、指定的大语言模型、指定的参数组、指定的评估方式，进行微调的运行， 运行结束就会产生一个微调版本的大语言模型，并且得到对这次微调出来的模型的评分。

微调（**Finetune**）： 使用指定的数据集、指定的大语言模型、指定的参数组进行真正的微调工作，包含了具体的微调工作的实现， 在微调过程中会有相关的微调的日志、监控数据、微调出来的权重文件等。

数据插件（**DataPlugin**）： 数据是大语言模型的基石。微调需要的数据是多种多样的，所以处理数据的能力也无法统一标准化，这种情况下， 满足数据处理的多样化能力就需要系统有插件化的能力，提供可以根据不同的数据进行不同处理的能力。

评估插件（**ScoringPlugin**）： 不同类型的模型评估方式可能是不同的，不同的场景评估方式也可能是不同的， 这个时候无法内置一种万能的评估能力，所以需要开放评估能力，以插件的方式实现这种能力。

模型仓库（**LLM / LLMCheckpoint**）： 不管是基础大模型，或者是微调出来的模型，在这里它都是一个模型，这里的模型仓库就是存放模型的地方，可以是基础大模型（LLM），也可以是微调出来的模型（LLMCheckpoint），基于这些模型可以进行后续的推理服务的部署或者对比等。

能力介绍

数据集处理

- 根据训练的数据地址，验证的数据地址，测试的数据地址等信息定义数据集。目前支持对象存储 (S3) 和本地文件的方式。
- 修改数据集的信息
- 展示数据集的列表信息和详细信息
- 删除已经存在的数据集
- 支持使用数据集插件的方式创建数据集

参数组

- 提供了基于 lora 的机制所需要大模型的微调参数来创建参数组
- 修改参数组的信息
- 展示参数组的列表信息和详细信息
- 删除已经存在的参数组

微调实验

- 根据不同的数据集、基础大模型、参数组的组合并行的创建和运行微调实验
- 支持设置一个微调实验使用的评估方式，以相同的评估方式给这次实验的所有任务进行模型评估，得到本次实验的最佳微调模型
- 支持查看每一个微调实验中的每一个微调任务的详情、日志、监控信息
- 目前支持 lora 的方式进行大模型的微调能力
- 目前支持 llama2 的模型微调能力

模型评估

- 创建一个评估对象，内置支持设置相关的问题和期望的答案，基于此来评估微调出来的模型的评分
- 查看、修改、删除评估对象
- 支持使用评估插件的方式创建评估对象

大语言模型仓库

- 显示所有实验微调出来的模型
- 利用微调出来的模型进行部署推理服务
- 支持根据分类进行过滤

推理服务

- 显示从模型仓库创建出来的推理服务，包括列表的方式显示以及详情
- 选择多个推理服务，使用相同的问题，让所有选择的推理服务同时回答这个问题，对比不同模型的推理服务的回答效果 支持在对比的过程中，展示计算性能数据，包括回复的 token 总数、每秒生成的 token 数、总的耗时是多少等性能数据

数据插件

- 创建一个数据集的插件
- 查看、修改、删除数据集的插件
- 支持开发自己的数据插件，同时在数据集创建的时候进行使用

评估插件

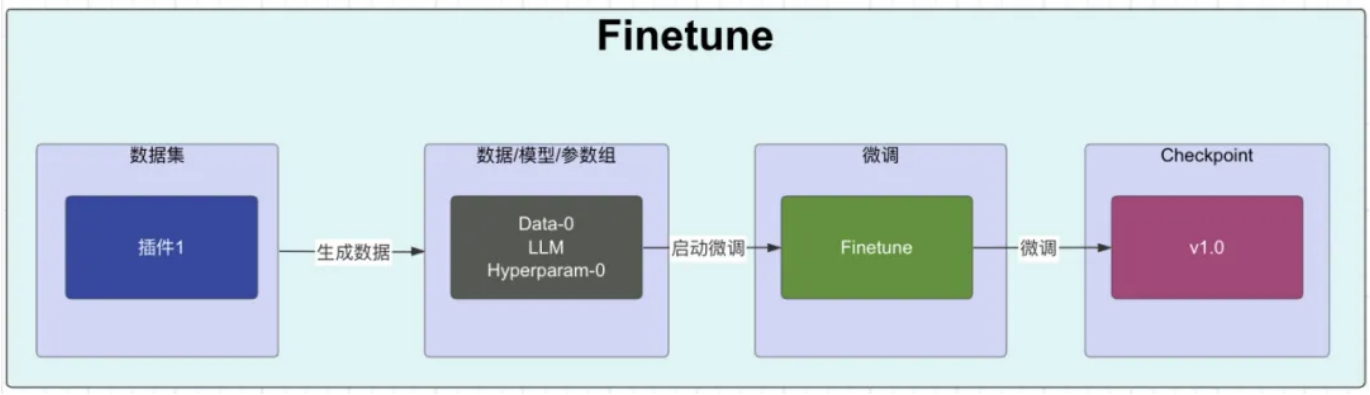
- 创建一个评估的插件
- 查看、修改、删除评估的插件
- 支持开发自己的评估插件，同时在微调实验创建的时候进行使用

控制台/命令行

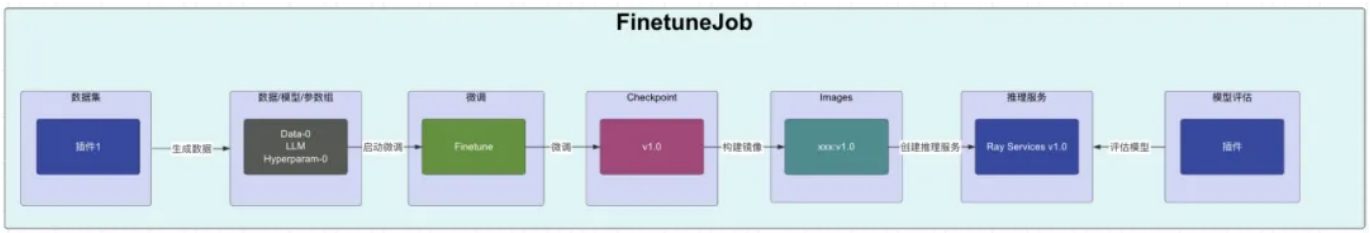
- 数据集管理
- 参数组管理
- 微调实验管理
- 模型仓库管理
- 推理服务管理

核心思路

Finetune: 这里的微调专注于基于数据、大模型、参数组进行大模型微调的核心业务处理能力。 只关注微调本身，最后的输出就是 checkpoint。是一个很典型的微调的流程。微调过程中的状态也会记录在资源对象的状态中。

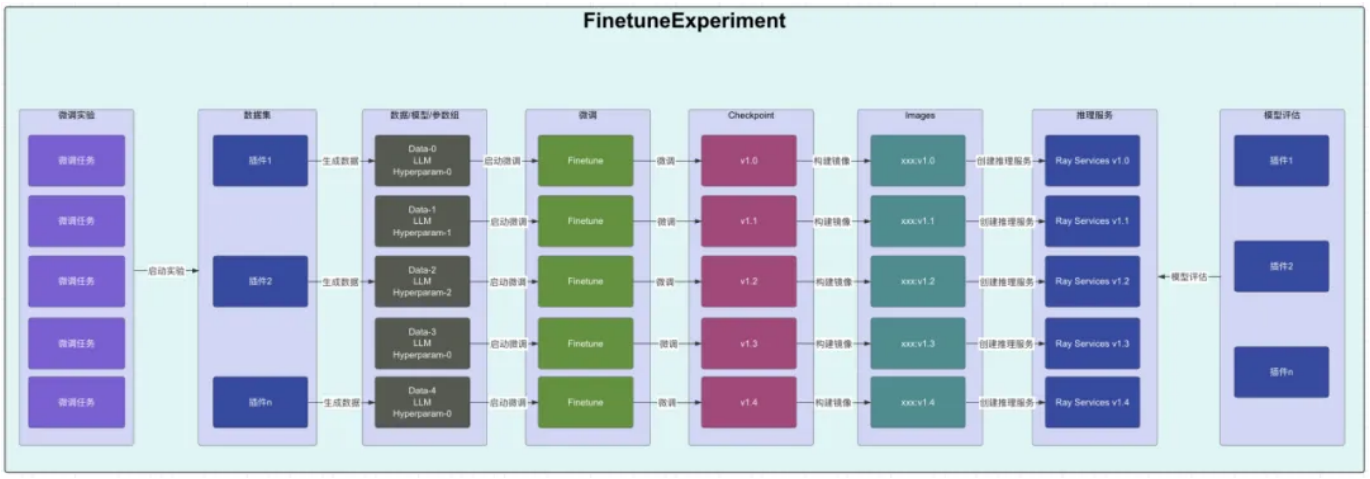


FinetuneJob: 这里的微调任务，不仅仅有上面的微调的能力，还会基于云原生的方式构建对应的镜像，以及启动对应微调出来的大模型的推理服务，并基于推理服务来完成本次的微调模型的模型评估能力，最后还会保存评估结果。FinetuneJob 中会包含 Finetune 的能力。 FinetuneJob 在运行过程中的状态也会记录在资源对象的状态中，同时还会同步 Finetune 的状态到 FinetuneJob 中的状态中。



FinetuneExperiment: 因为大模型微调存在的不确定性，所以在资源充分的情况下，并行微调就变得很多必要。 这里的微调实验就是利用数据、大模型、参数组这三个配置灵活组合，利用底层的多个 GPU 的卡，一次操作，并行完成多次微调， 然后基于所有微调模型进行总体的评估，最后在这次

实验中，使用相同的评估标准，找出最佳的微调模型作为本次微调实验的推荐的微调模型。 FinetuneExperiment 中其实就是包含了多个 FinetuneJob。也是用户侧使用的入口。FinetuneExperiment 在运行过程中的状态也会记录在资源对象的状态中， 同时还会同步 FinetuneJob 的状态到 FinetuneExperiment 中的状态中。



模型微调快速入门手册

大致流程为：

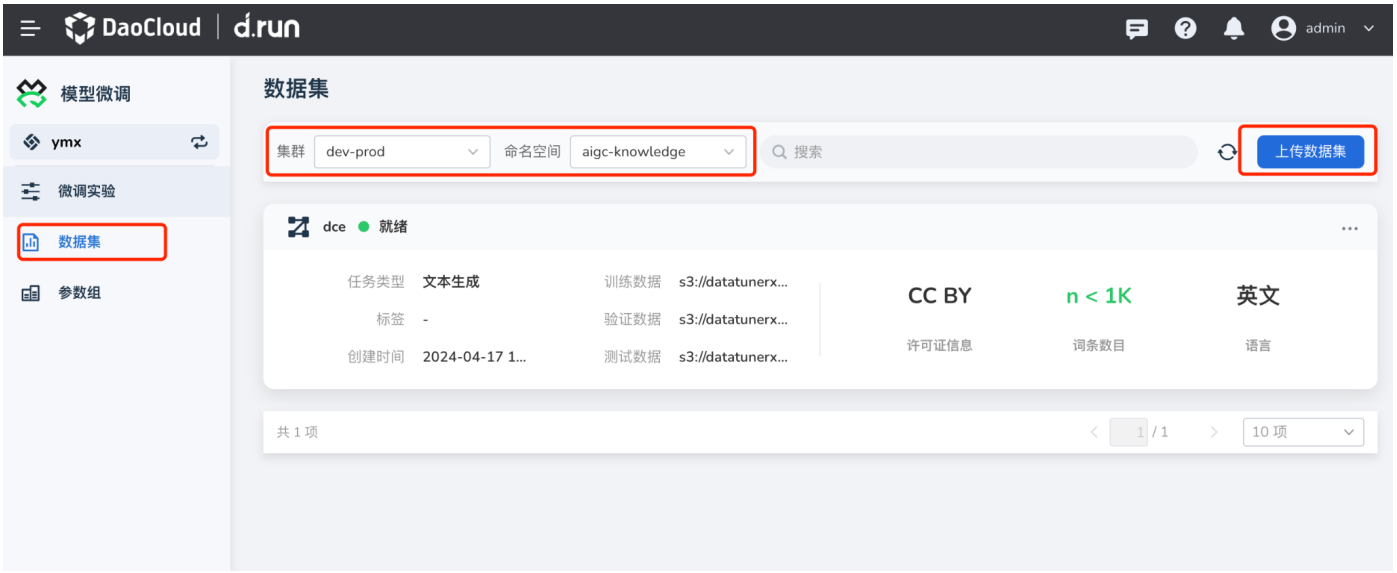
前置要求准备 -> 创建数据集 -> 创建参数组 -> 创建微调实验 -> 部署微调模型 -> 对话

前置要求

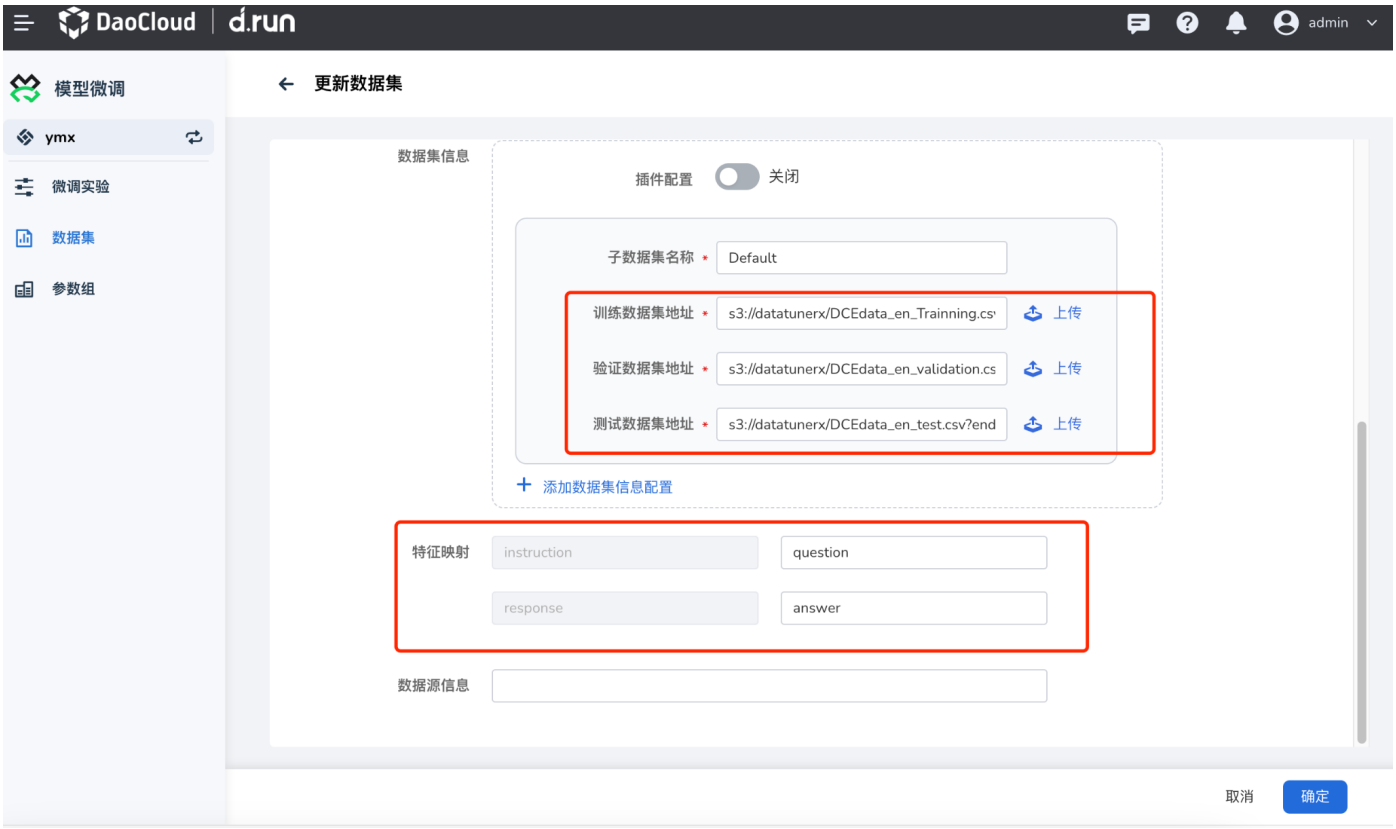
1. 已购买算力集群，并且该算力集群被加入当前用户所在的 workspace 中
2. 集群有可用的大模型（目前微调只适用与 llama2-7b 模型）
3. 已准备本地/远程数据集，数据集格式为 Q&A（目前仅支持 CSV 格式且文件不超过 50M）

创建数据集

1. 在 模型微调 -> 数据集 中，点击 上传数据集 按钮。



2. 参照以下说明填写表单后，点击 确认



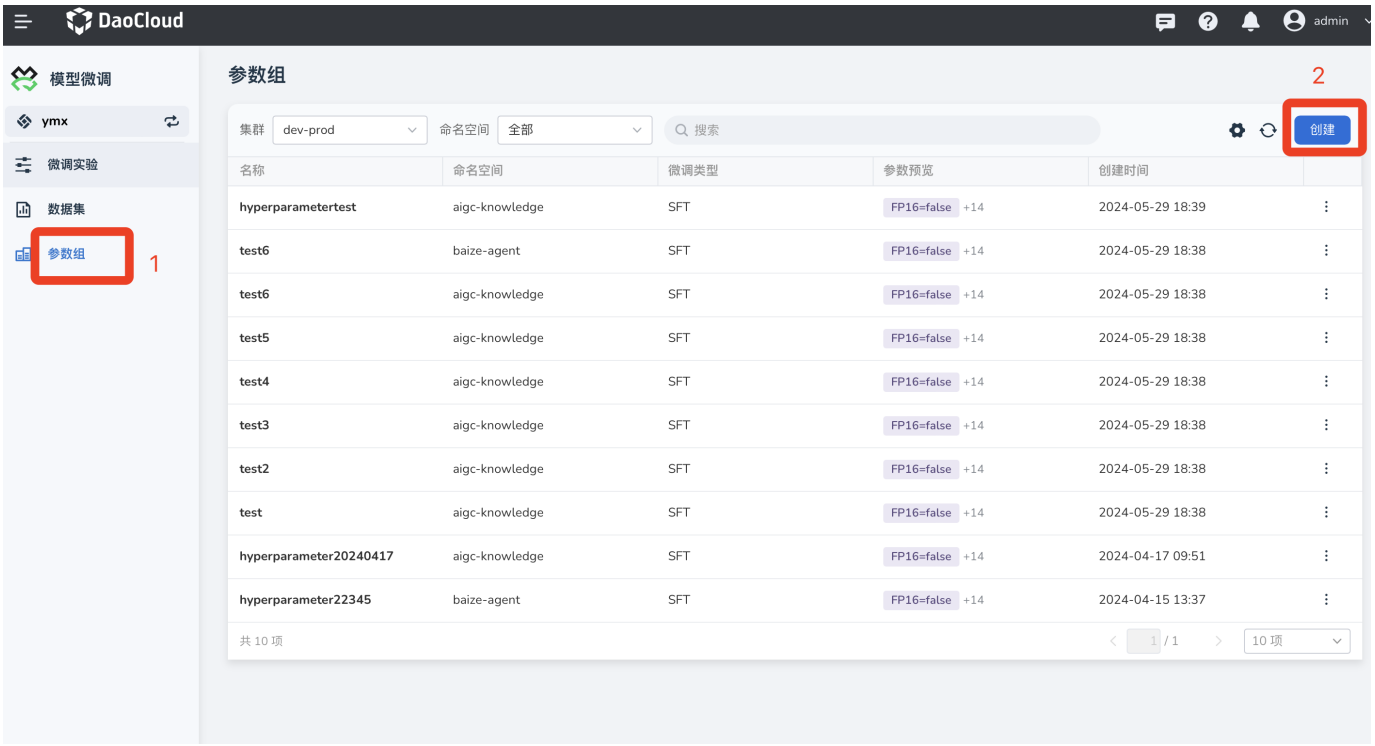
参数说明	详细描述
数据集名称	不支持中文，长度限制 63 字符
自定义标签	为数据集添加自定义标签
数据集语言	当前支持中文/英文
集群及命名空间	选择数据集所属的集群以及命名空间，注意：数据集所在集群/命名空间应该和微调实验所属集群/命名空间保持一致。
授权协议	根据数据集的属性，设置授权协议
词条数目	根据数据集的大小，选择词条的数目
任务类型	根据数据集属性，选择数据集被应用的任务的类型，同时支持添加子类型
数据集信息	配置数据集的信息，可以上传本地数据集文件，配置三种类型的数据集地址，或者设置插件，配置参数让插件拉取数据集
特征映射	填写数据集中文件的表头
列名	该数据集分为两列，列名分别为：question/answer

Tip

- 点击下载 DCEdata_en_test.csv 测试集文件
- 点击下载 DCEdata_en_Trainning.csv 训练集文件
- 点击下载 DCEdata_en_validation.csv 验证集文件

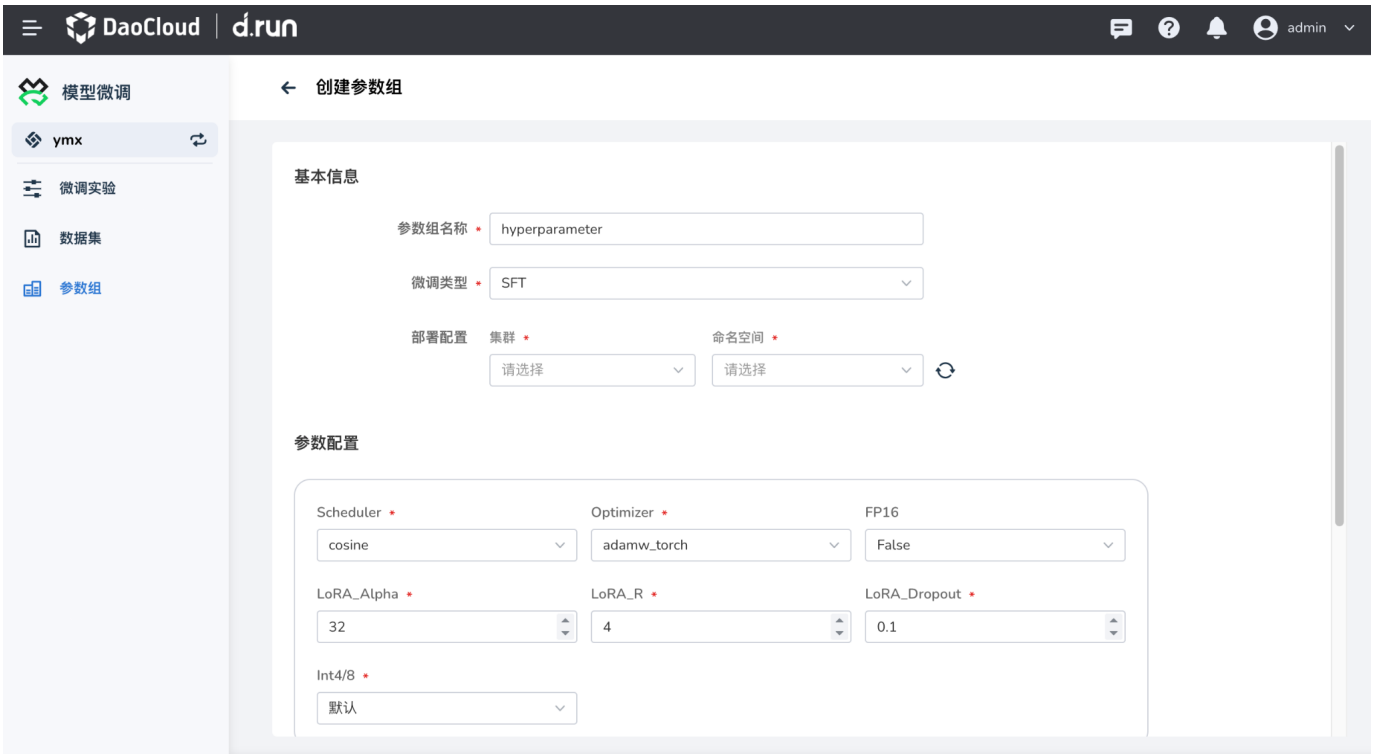
创建参数组

在 模型微调 -> 参数组 中，点击 创建 按钮。

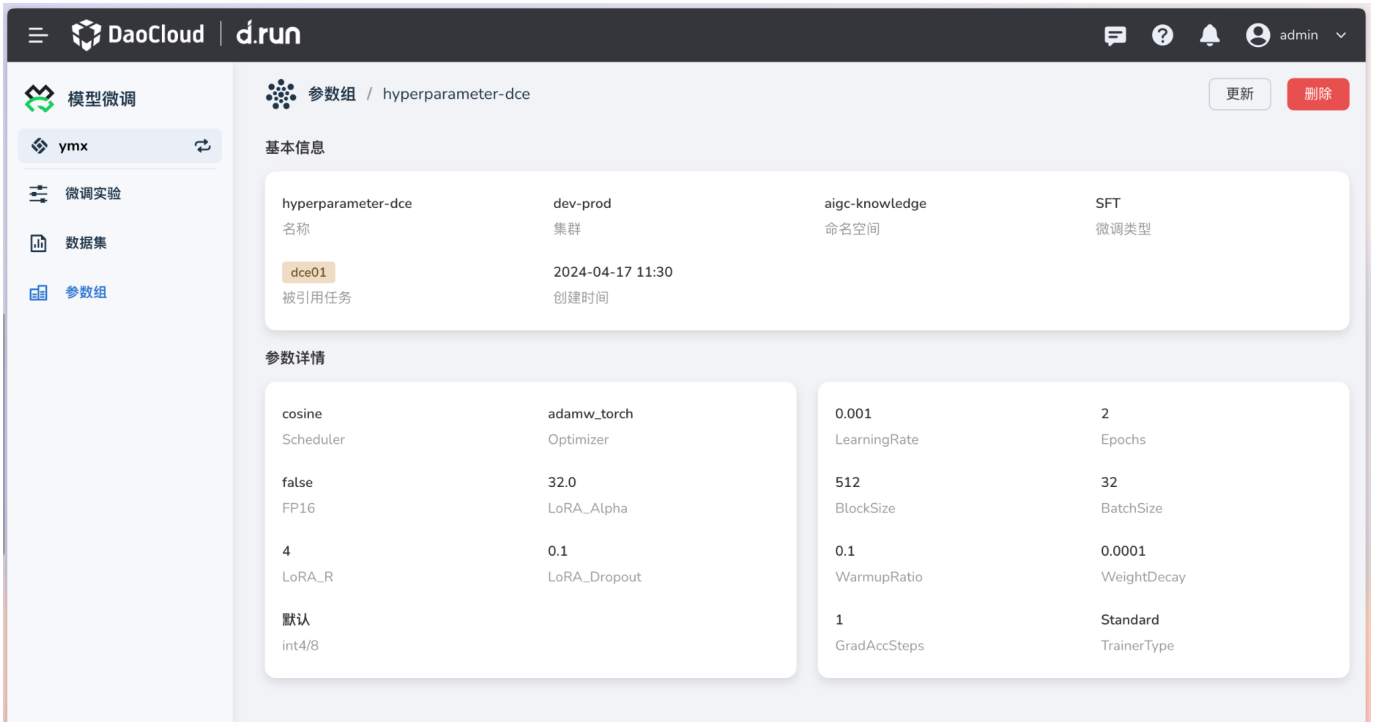


- 设置参数组基本信息，参数组名称以及微调类型和所属集群以及命名空间，目前支持的微调类型为：SFT
- 按照需要配置参数组信息，点击右下角确认完成参数组的创建

点击 确认 创建参数组。

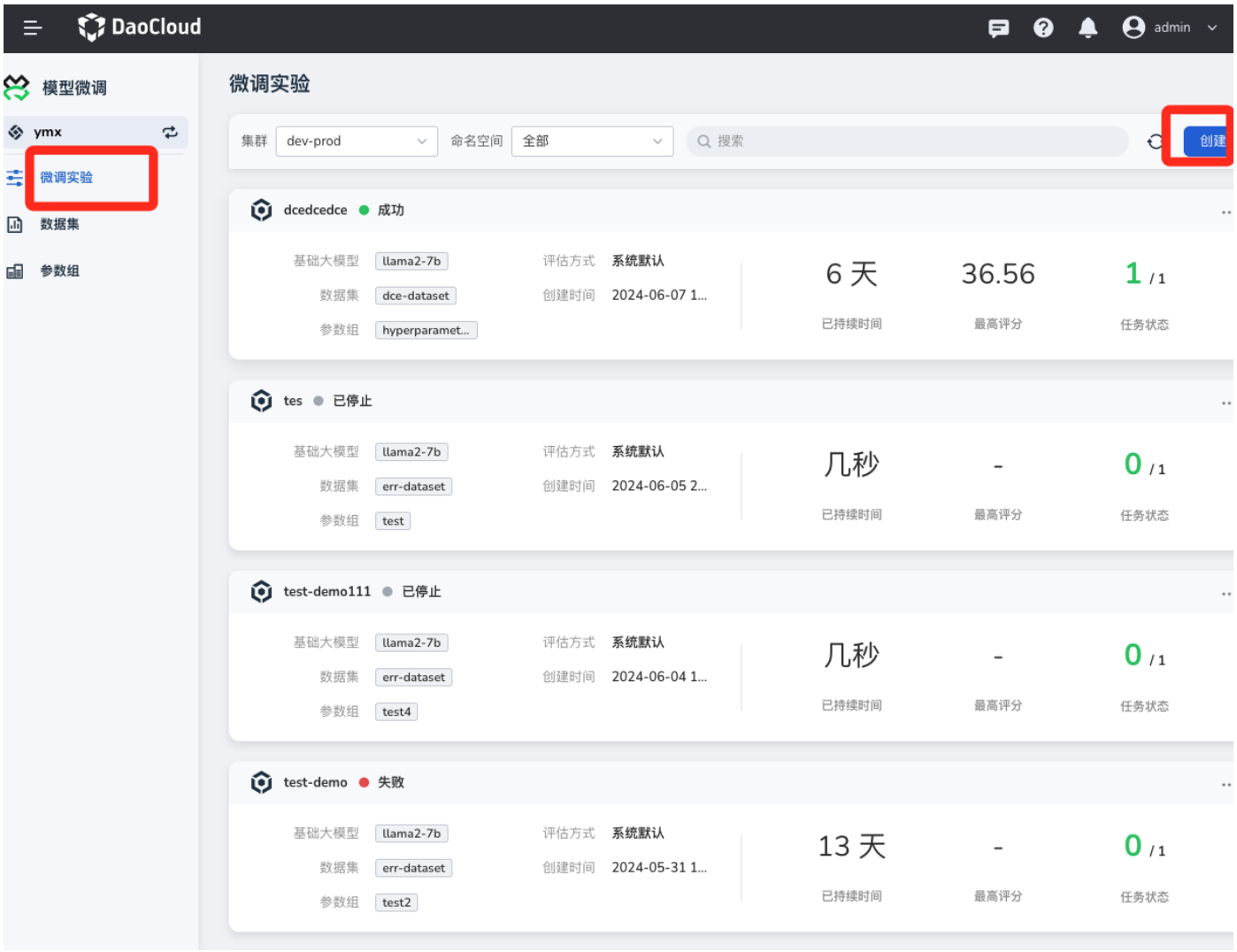


以 DCE 参数组为例：



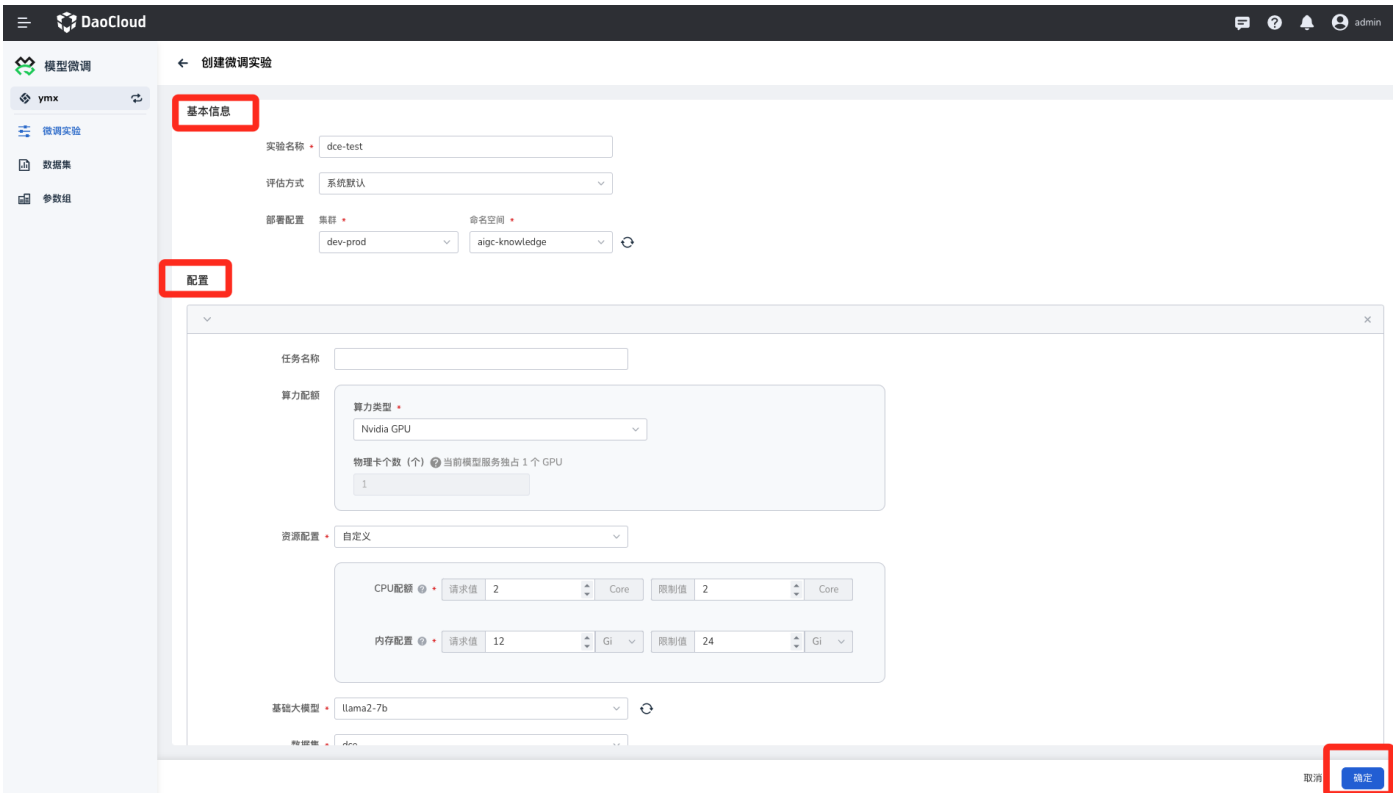
创建微调实验

1. 在 模型微调 -> 微调实验 中，点击 创建微调实验 按钮



2. 填写表单

- 实验名称：由小写字母、数字字符或“-”组成，并且必须以字母或数字字符开头及结尾。
- 选择评估方式：实验中对模型使用的评分准则。
- 选择命名空间。



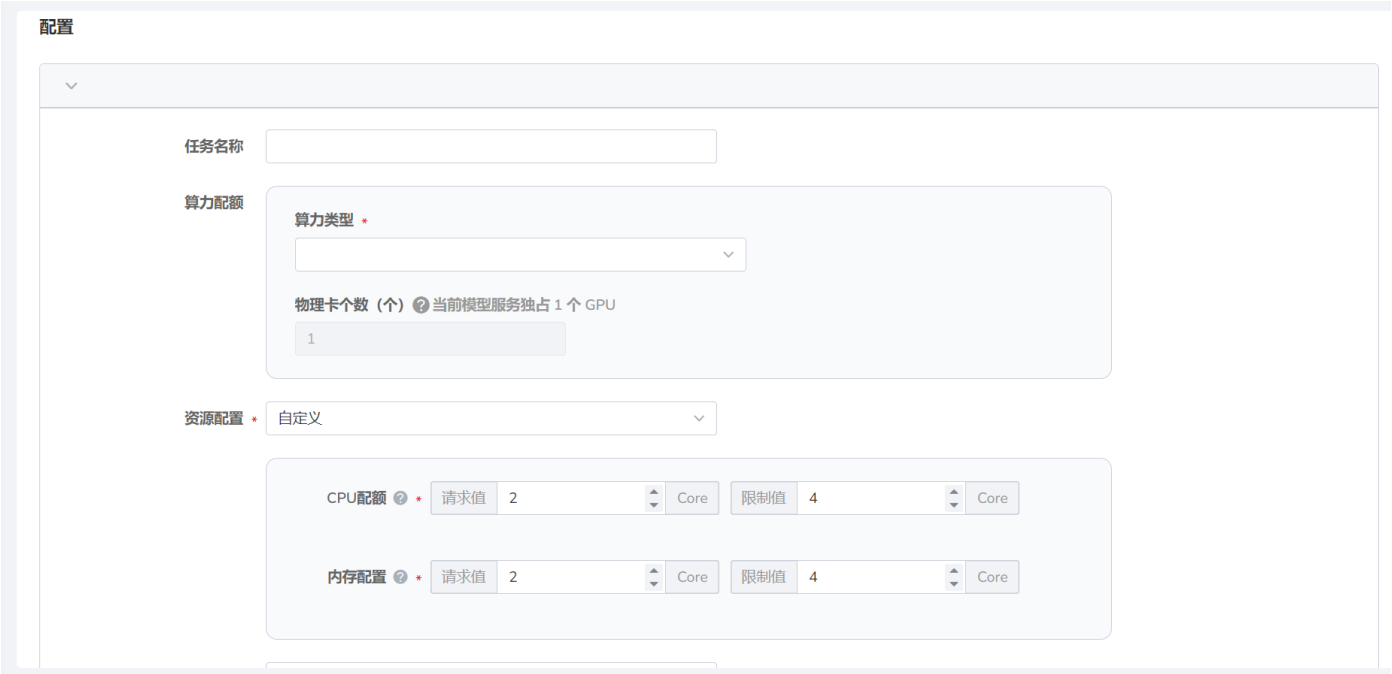
- 任务名称：由小写字母、数字字符或“-”组成，并且必须以字母或数字字符开头及结尾。
- 选择算力类型并填写物理卡个数。

> **Info**
>
> 当前模型服务仅支持 Nvidia 的 GPU，模型会根据物理卡个数在GPU上进行分布式微调。

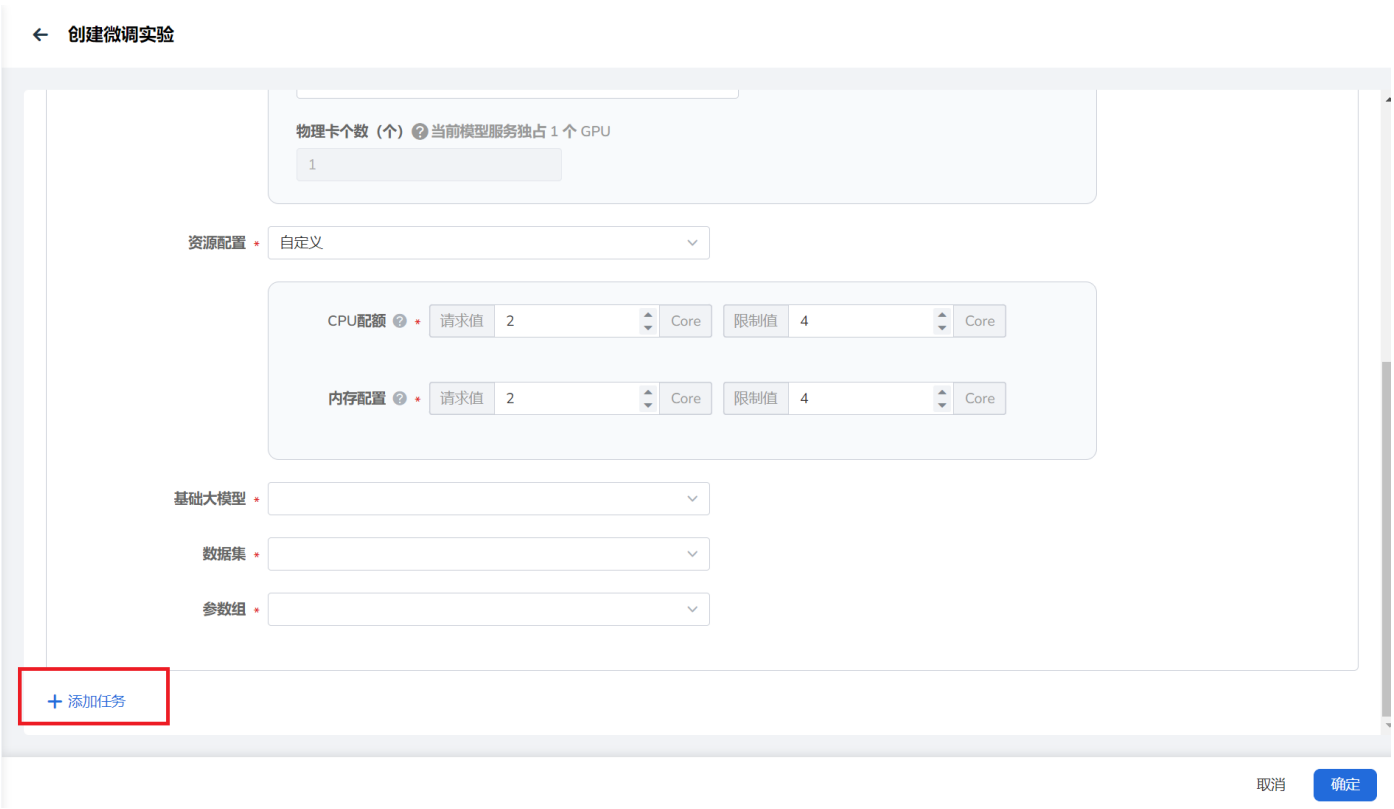
- CPU 配额：通常需要使用多核 CPU 来加速训练和推理过程。具体的 CPU 配额需要根据任务的需求和可用的硬件资源来确定。
- 内存配置：根据模型的大小和数据集的大小来确定内存需求，并根据需要调整内存配置。

推荐 CPU 配额和内存配置请求值为 **16** Core，限制值为 **32** Core。

> **Note**
>
> 两者的请求值皆不可超过限制值。



- 选择实验中要使用的 基础大模型 、 数据组 以及 参数组 。
- 若要设置多个微调任务，可点击左下角 添加任务 创建新任务。

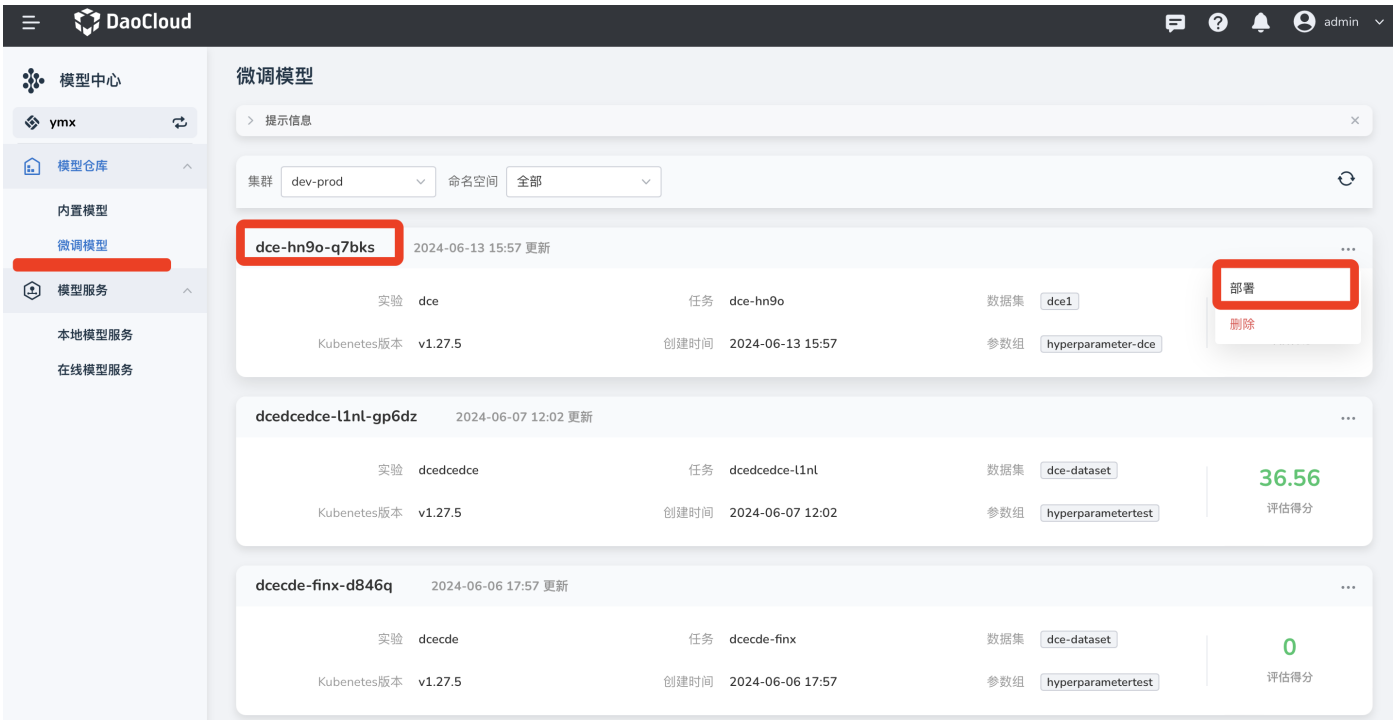


3. 点击右下角 确认 按钮创建微调实验。

部署微调模型

在 模型仓库 -> 内置模型 的微调模型中，可以查看运行成功的实验结果。

1. 点击右侧的 ... ，在弹出的选项中选择 部署 。

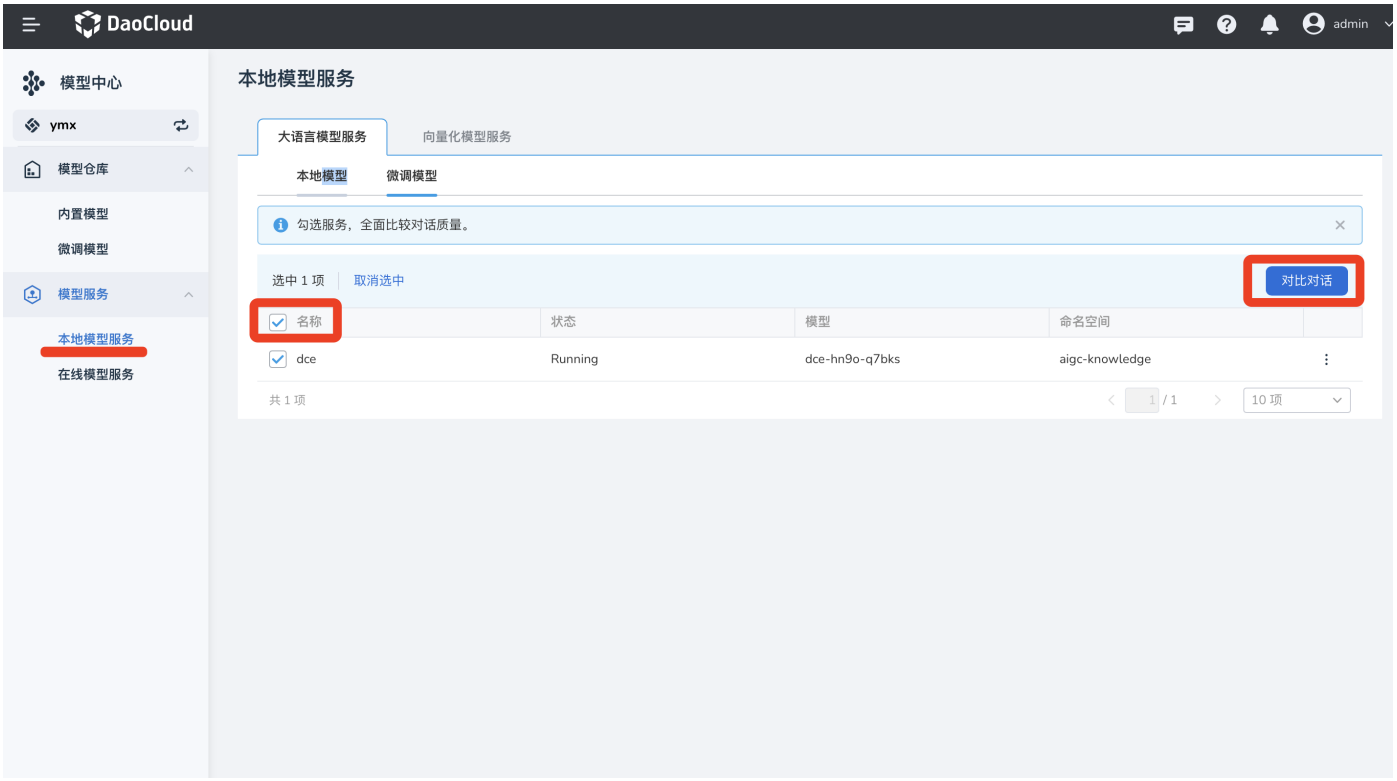


2. 填写模型服务名称、命名空间、算力配额、资源配置后点击 确定 （注：推荐 CPU 配额和内存配置请求值为 **16** Core， **32** Gi）。

3. 创建成功，接下来可以通过部署的模型提供服务。

模型服务对话

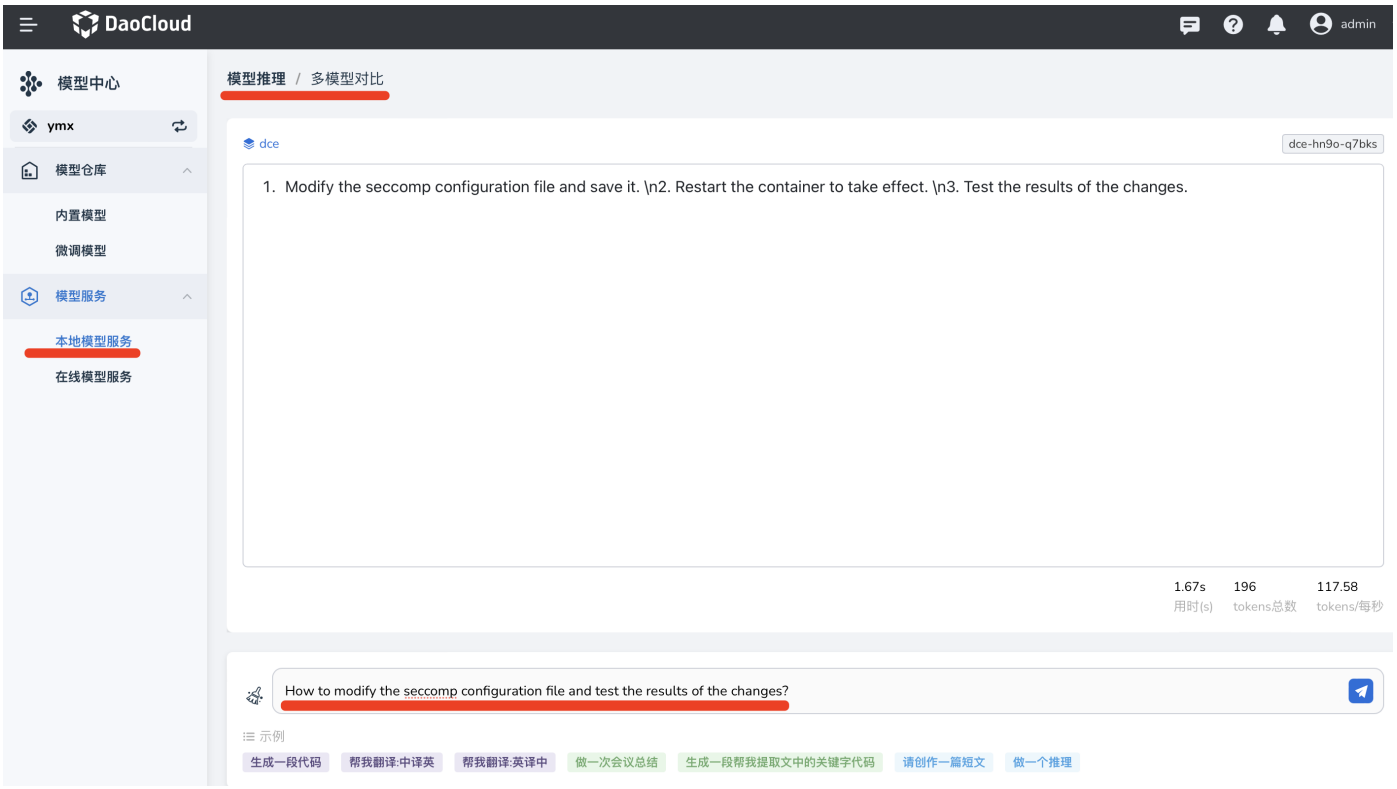
1. 微调模型部署成功后，可以在 模型服务 -> 本地模型服务 -> 微调模型 勾选模型服务后， 点击右侧 对话 即可与该模型对话。



Tip

最多可选中三个微调模型进行对话。

2. 你可以输入训练集中的问题，发送给微调模型，以此验证模型微调的效果，微调模型会回答用户提出的问题。


DaoCloud
admin
模型中心
ymx
模型仓库
内置模型
微调模型
模型服务
本地模型服务
在线模型服务
模型推理 / 多模型对比
dce
dce-hn9o-q7bks
1. Modify the seccomp configuration file and save it. \n2. Restart the container to take effect. \n3. Test the results of the changes.
1.67s
用时(s)
196
tokens总数
117.58
tokens/每秒
How to modify the seccomp configuration file and test the results of the changes?
示例
生成一段代码
帮我翻译:中译英
帮我翻译:英译中
做一次会议总结
生成一段帮我提取文中的关键字代码
请创作一篇短文
做一个推理

功能特性

模型微调的功能特性参见下表：

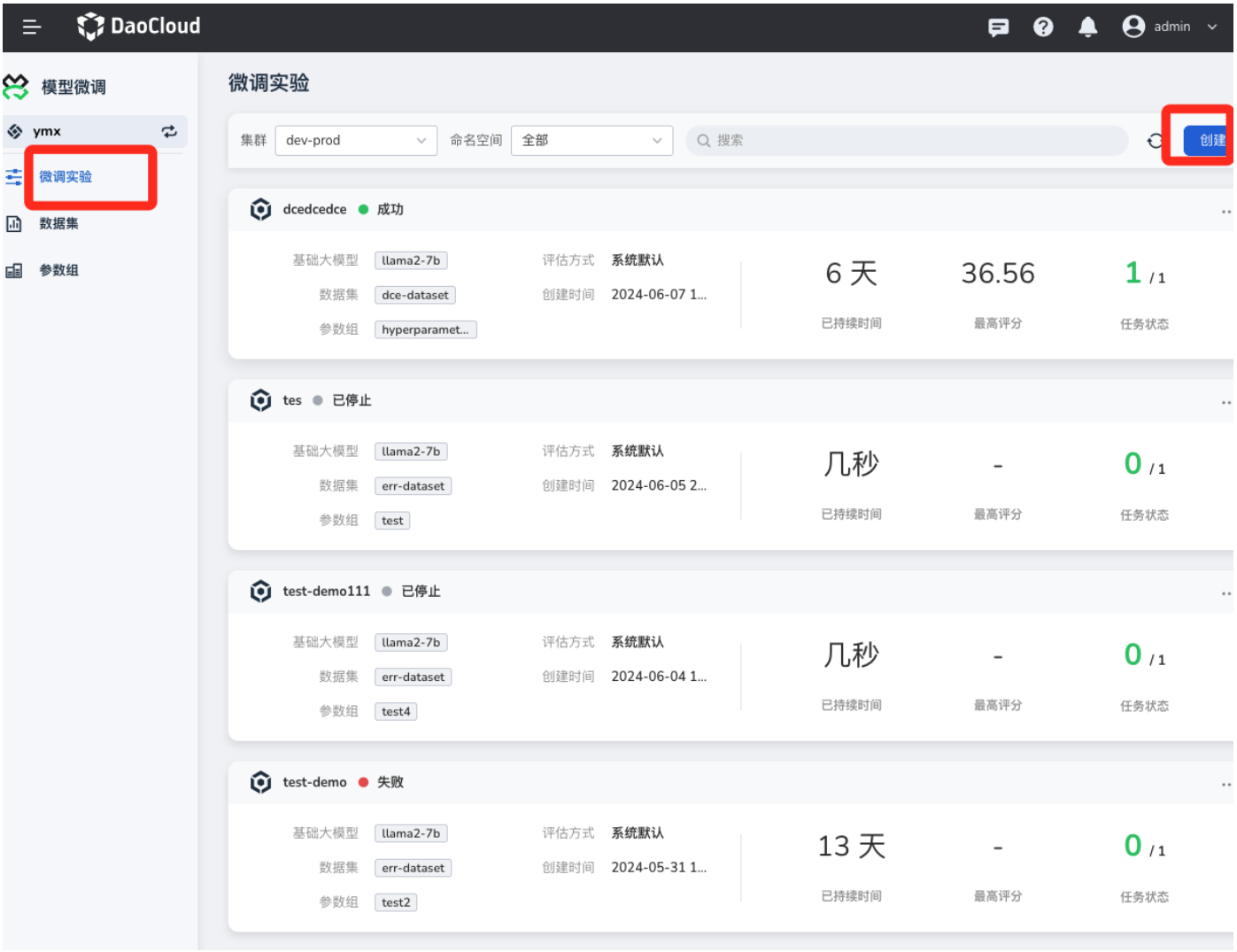
主要功能	细分项
微调实验	提供直观的界面，可视化地进行Llama系列模型（7B/13B）的指令微调
	支持跨GPU的模型分布式微调，提升训练效率
	支持使用不同的参数组和数据集创建微调实验，进行交叉微调
	支持查看微调过程中的学习率、训练损失和验证损失等关键数据，实时监控模型训练状态
	支持对检查点（checkpoint）进行评估打分，保证模型质量
	支持将模型存储到模型中心，提供可视化的模型列表和训练参数查看
数据集	提供直观的界面，可视化地创建数据集，包括训练、验证和测试数据集
	支持直接从S3存储中拉取文件，方便快捷
	支持本地文件上传，提供多种数据接入方式
参数组	提供直观的界面，可视化地创建超参数组，包括调度器（Scheduler）、优化器（Optimizer）、学习率（LearningRate）、训练周期（Epochs）和批次大小（BatchSize）等

微调实验

微调实验是使用 数据组 、 大模型 、 参数组 这三个配置灵活组合，利用底层的多个 GPU 卡，一次操作，并行完成多次微调， 然后基于所有微调模型进行总体的评估。最后在微调实验中，使用相同的评估标准，找出最佳的微调模型作为本次微调实验的推荐微调模型。

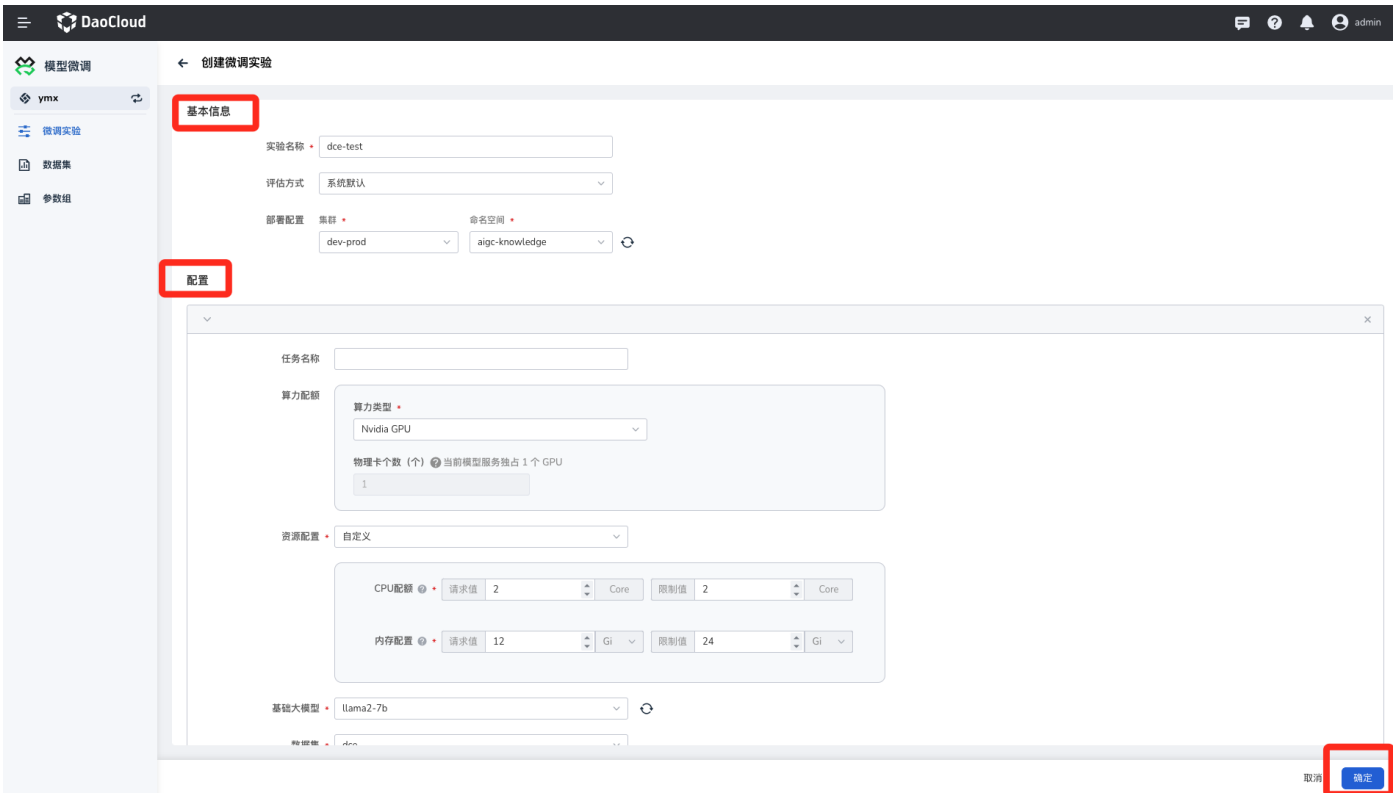
创建微调实验

1. 在 模型微调 中，点击右上角 创建微调实验 按钮。



2. 填写实验基本信息：

- 实验名称：由小写字母、数字字符或“-”组成，并且必须以字母或数字字符开头及结尾。
- 选择评估方式：实验中对模型使用的评分准则。
- 选择命名空间


DaoCloud
admin
模型微调
ymx
微调实验
数据集
参数组
创建微调实验
基本信息
实验名称 *
dce-test
评估方式
系统默认
部署配置
集群 *
命名空间 *
dev-prod
aigc-knowledge
配置
任务名称
算力配额
算力类型 *
Nvidia GPU
物理卡个数（个）当前模型服务独占 1 个 GPU
1
资源配置 *
自定义
CPU配额 *
请求值
2
Core
限制值
2
Core
内存配置 *
请求值
12
Gi
限制值
24
Gi
基础大模型 *
llama2-7b
取消
确定

3. 填写微调任务配置：

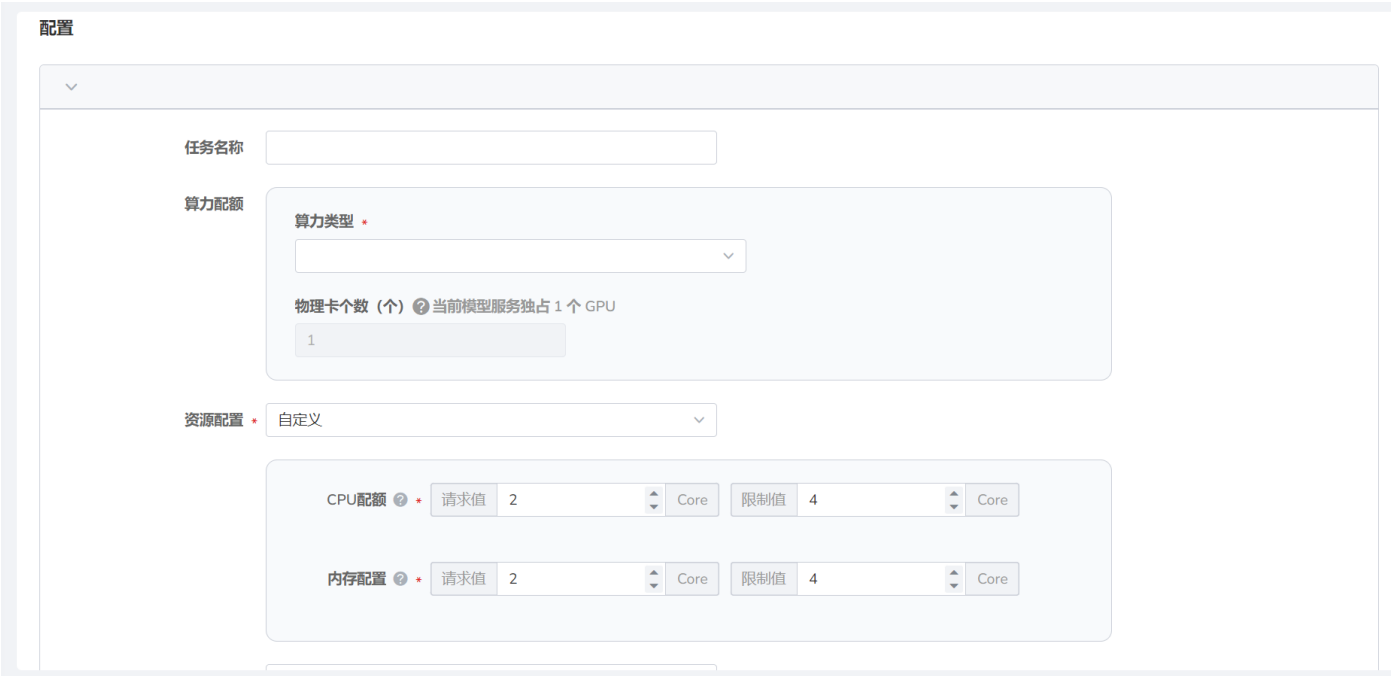
- 任务名称：由小写字母、数字字符或“-”组成，并且必须以字母或数字字符开头及结尾。
- 选择算力类型并填写物理卡个数。

> **Info**
>
> 当前模型服务仅支持Nvidia的GPU，模型会根据物理卡个数在GPU上进行分布式微调。

- 资源配置：
- CPU配额：通常需要使用多核CPU来加速训练和推理过程。具体的 CPU 配额需要根据任务的需求和可用的硬件资源来确定。
- 内存配置：根据模型的大小和数据集的大小来确定内存需求，并根据需要调整内存配置。

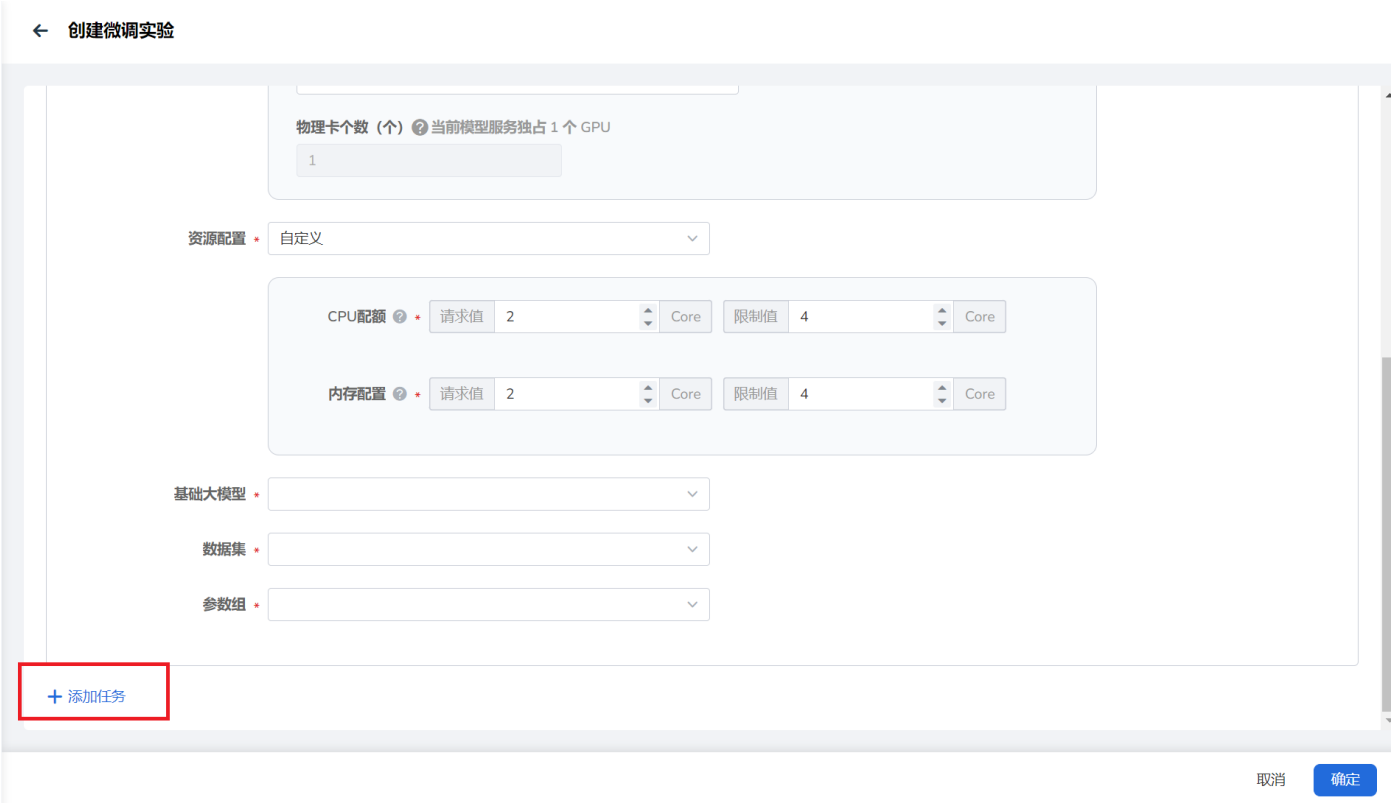
推荐CPU配额和内存配置请求值为 **16** Core，限制值为 **32** Core。

> **Note**
>
> 两者的请求值皆不可超过限制值。



- 选择实验中要使用的 基础大模型 、 数据组 以及 参数组 。

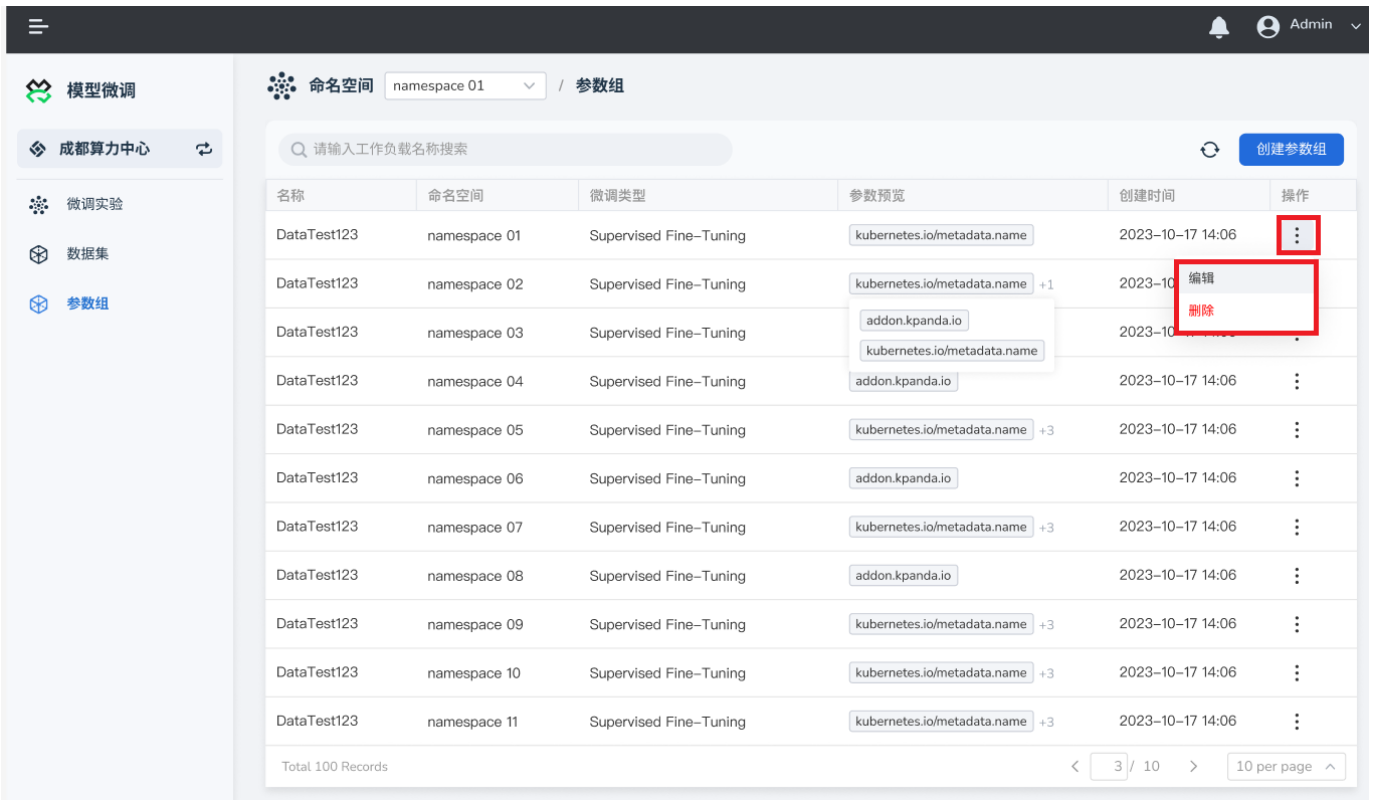
若要设置多个微调任务，可点击左下角 添加任务 创建新任务。



4. 点击右下角 确认 按钮即可创建微调实验。

停止或删除微调实验

若想停止或删除某个微调实验，可点击右侧 ⋮ ，在弹出菜单中选择 停止 或 删除 。



> **Note**
>
> 停止及删除操作不可逆，请谨慎操作。

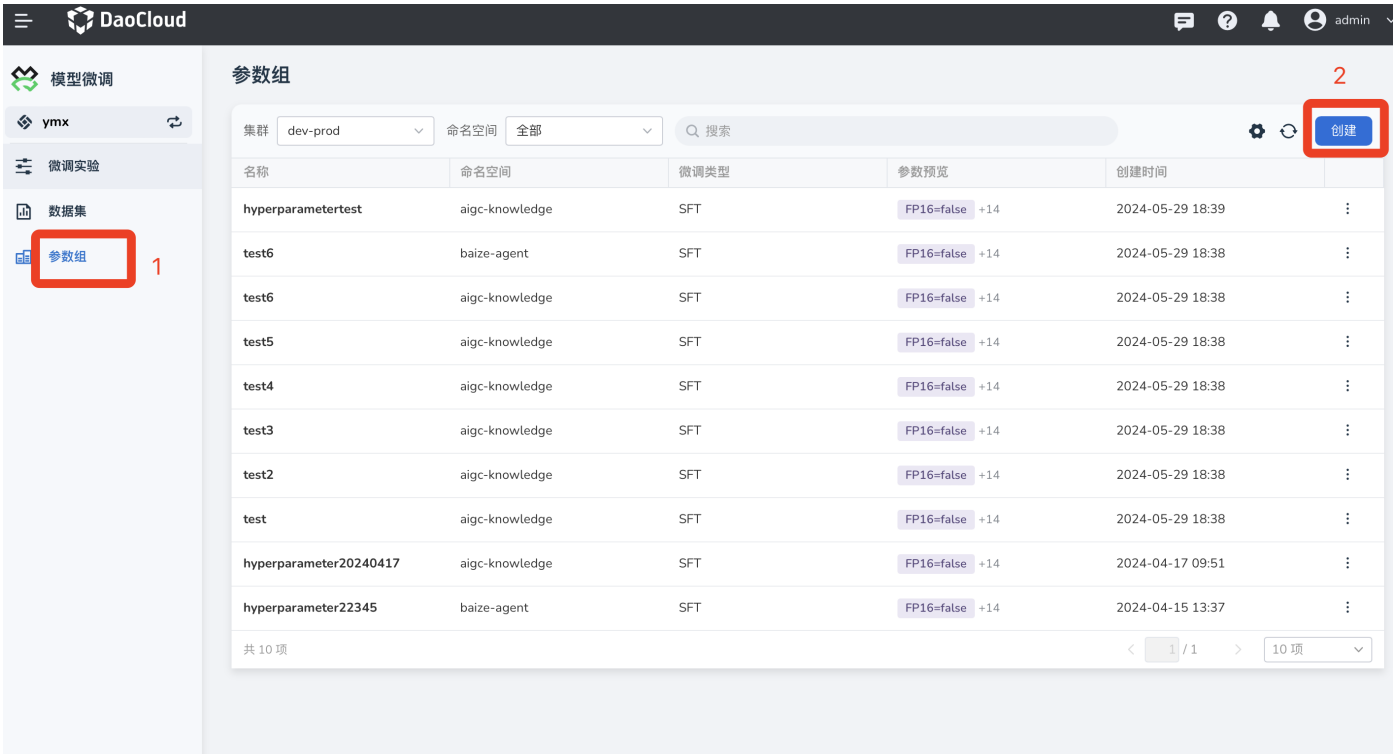
参数组

在微调模型之前，可以先设置一些参数组，不同的微调类型对应不同的的参数组。

参数组（Hyperparameter Group）代表一组超参的集合。在大语言模型的微调过程中， 可以根据需要设置很多不同的超参，如 Scheduler、Optimizer、LearningRate、Epochs、BlockSize、BatchSize 等，本文的参数组就是各种超参的组合。

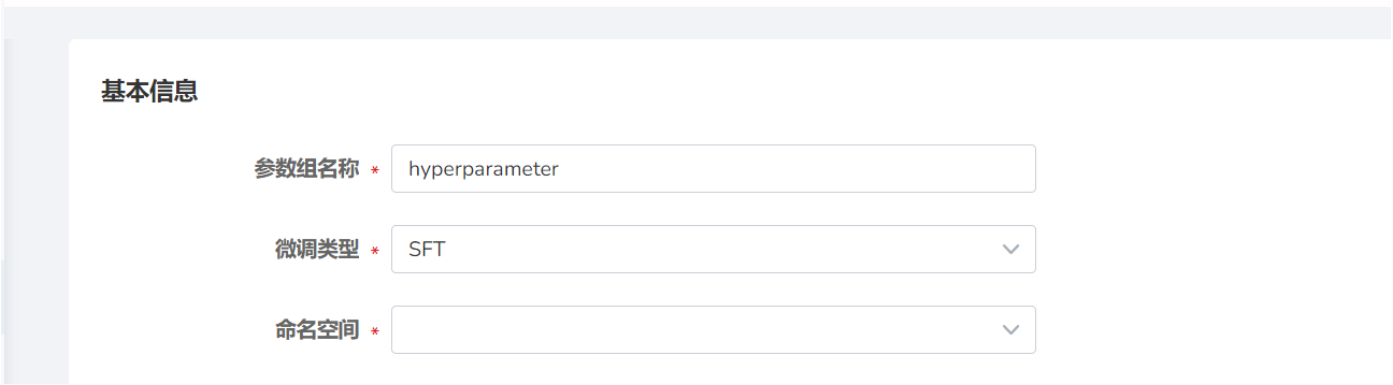
创建参数组

1. 点击右上角 创建 按钮。



2. 填写参数组的基本信息：

- 参数组名称
- 选择微调类型：默认设置为 **SFT**
- 选择参数组的命名空间

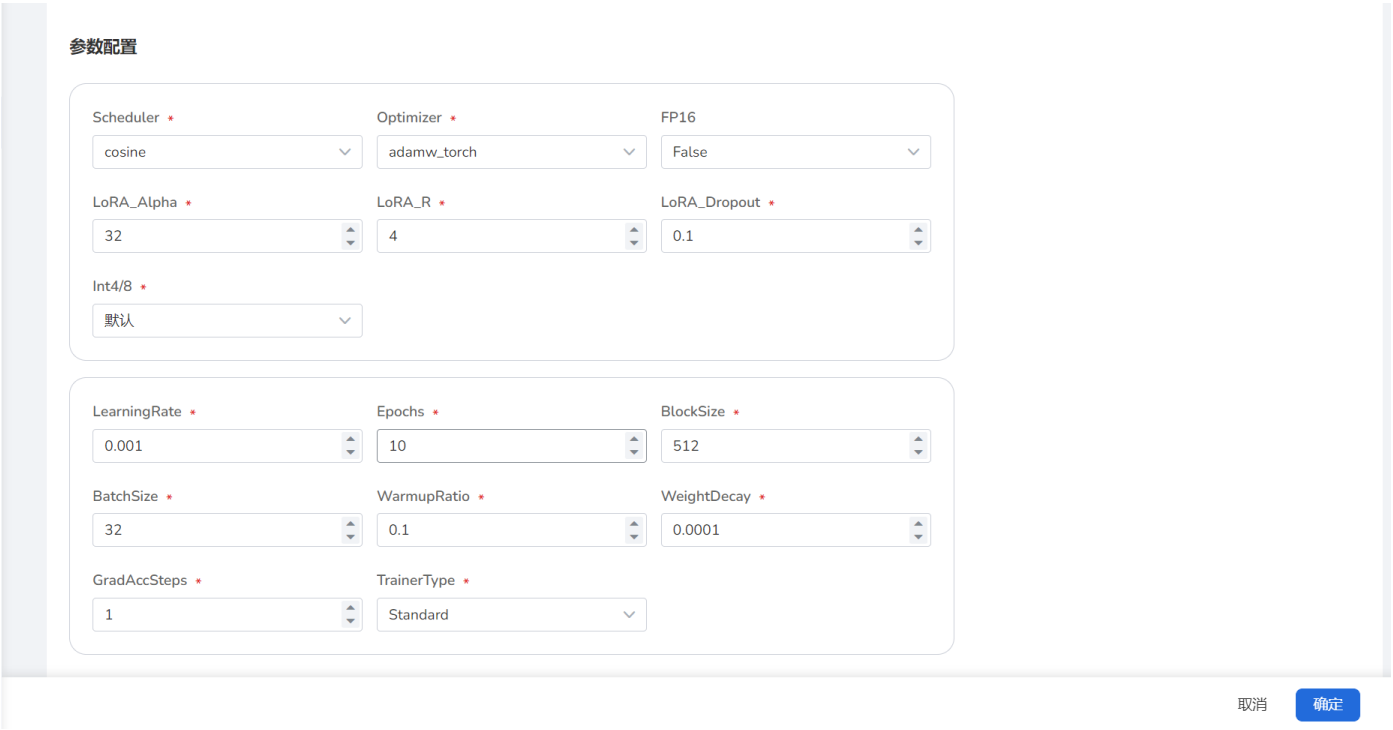


3. 填写参数配置：

- Scheduler：训练中使用的调度器类型，指如何设置学习率的变化方式，默认设置为 **cosine** 。可选项包括：
- **cosine** ： 通过绘制余弦曲线，将 **LearningRate** 逐渐减少到 0。
- **linear** ： 初始 **LearningRate** 为设置值，以直线形式减少到 0。
- **constant** ： **LearningRate** 始终保持不变。
- Optimizer：训练中使用的优化器类型，默认设置为 **adamw_torch** 。包含以下选项：
- **adamw_torch** ： 是 Adam 优化器的一种变体，它使用了一种称为权重衰减的正则化方法，可以有效地防止过拟合。AdamW_torch 是 PyTorch 中的 AdamW 实现。
- **adamw_hf** ： AdamW_hf 是 Hugging Face Transformers 库中的 AdamW 实现，与 AdamW_torch 相比，它使用了一种不同的权重衰减方法，可以更好地适应自然语言处理任务。
- **sgd** ： 是 随机梯度下降（**Stochastic Gradient Descent**） 的缩写，是一种基本的优化器。它每次迭代使用一个样本来更新模型参数，可以快速收敛，但容易陷入局部最优解。
- **adafactor** ： Adafactor 是一种自适应学习率优化器，它使用了一种称为自适应二阶矩估计的方法来调整学习率。它可以自适应地调整学习率和权重衰减，适用于大规模训练任务。
- **adagrad** ： Adagrad 是一种自适应学习率优化器，它根据每个参数的历史梯度大小来调整学习率。它适用于稀疏数据集和非平稳目标函数，但可能会导致学习率过小，导致收敛速度慢。

总结来说， **adamw_torch** 和 **adamw_hf** 适用于大多数任务， **sgd** 适用于简单的模型和数据集， **adafactor** 和 **adagrad** 适用于大规模训练任务。

参数	说明	默认值
FP16	选择是否使用 16 位浮点数进行训练，这可以减少内存占用和加速训练过程。	False
LoRA_Alpha	数值越大，LoRA 输出对模型的影响越大。建议根据微调数据量调整数值。	32
LoRA_R	数值越大，LoRA 的参数越多。建议根据模型表现调整数值。	4
LoRA_Dropout	训练过程中会以这一概率随机选择忽略神经元以防止过拟合。数值在 0 到 1 之间。	0.1
Int4/8	可选择 **Int4** 或 **Int8** 形式储存数据。选择不同形式会影响储存效率和计算速率。	-
LearningRate	每次参数更新的步长，数值越小参数更新越快。	0.001
Epochs	对同一组数据重复训练次数。	10
BlockSize	模型中块的大小。数值越大，模型表现可能会更好，但占用内存和计算量也会增加。	512
BatchSize	同时学习的批次大小。数值越大，训练速度越快，但占用内存也会增加。	32
WarmupRatio	训练过程中预热的比例，数值范围大多在 0 到 1 之间。	0.1
WeightDecay	表示模型的正则化程度。数值越小，正则化越低，可能出现过拟合。	0.0001
GradAccSteps	权重更新前积累的梯度数目。在硬件条件不变的情况下，该数值越大，可使用的 **BatchSize** 越大。	1
TrainerType	训练器类型。默认为 **Standard** ，指使用标准优化算法进行训练。	Standard



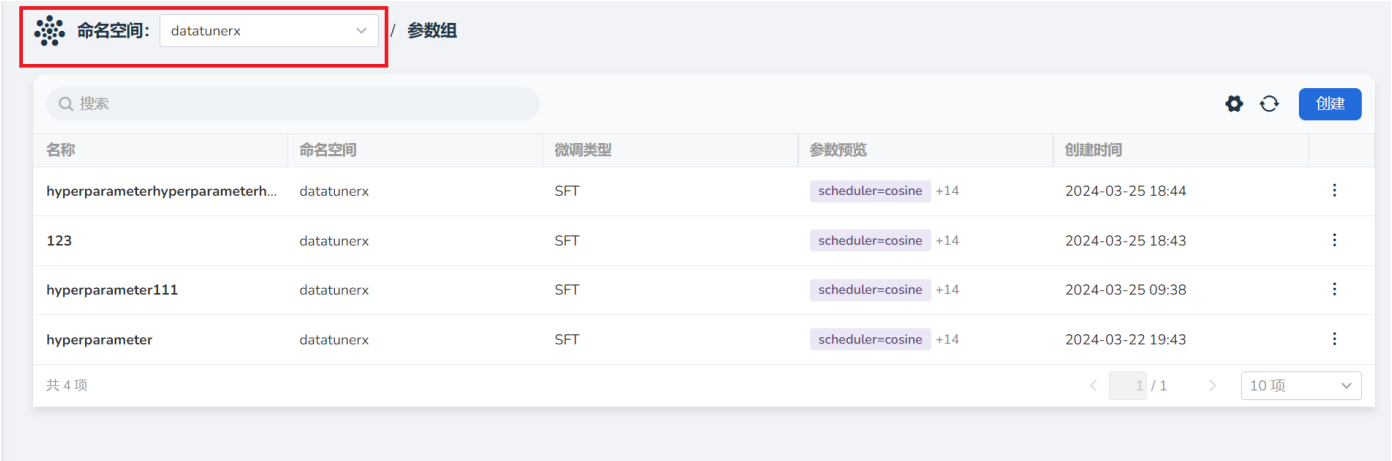
4. 点击右下角 确定 即可创建参数组供微调实验中使用。

下一步：微调实验

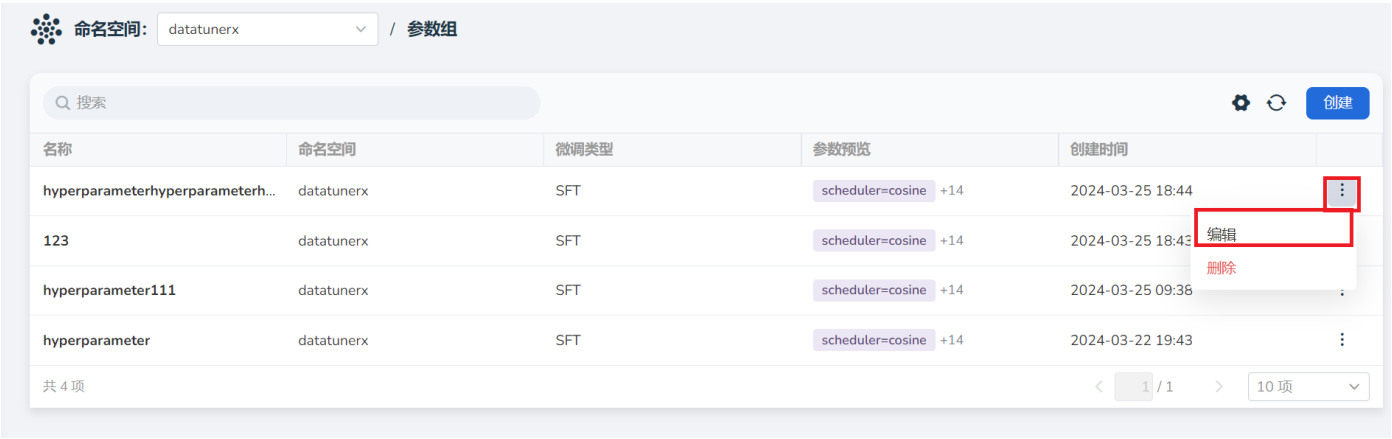
编辑参数组

若希望将已创建的参数组做出调整：

1. 可在左上角选择一个想要修改参数组的 命名空间 。



2. 找到想要查看的参数组，点击右侧的 ⋮ 按钮，选择 编辑。



3. 修改参数组内的所有参数配置。

删除参数组

找到想要删除的参数组，点击右侧的 ⋮ 按钮，选择 删除。

> **Note**
>
> 删除操作不可逆，请谨慎操作。

1.4.2 算法开发

什么是算法开发

d.run 算法开发是 DaoCloud 推出的基于云原生操作系统的 AI 算力平台，能够提供软硬一体的 AI 智算体验， 整合异构算力，优化 GPU 性能，实现算力资源统一调度和运营，最大化算力效用并降低算力开销， 并且还提供了优化的 AI 开发框架，简化 AI 开发和部署，加速推动各行业的 AI 应用场景落地。

功能特性

- 算力资源全托管 ：依托于 DCE（DaoCloud Enterprise），提供强大的基础设施能力，支持超大规模算力集群、异构 GPU 等一站式托管，并提供一系列如 vGPU 等软硬一体加速方案。
- 数据编排 ：支持模型开发生命周期中数据管理与编排能力，提供多数据源接入，数据集管理、超大训练数据预热等能力，并且从底层存储引擎优化，保障数据的安全与高效利用。
- 开发管理 ：提供主流的模型开发工具，满足 MLOps 和 LLMOps 工程师和科学家们对开发工具的需求，支持快速拉起高性能开发环境，一键申请高性能 GPU、软件依赖、训练数据等资源。
- 任务管理 ：支持训练任务的全生命周期管理，提供多种快速创建任务的方式；支持 Pytorch、TensorFlow、PaddlePaddle 等主流任务框架，天然支持单机、分布式、多节点、多卡等多种类任务调度。
- 模型推理 ：提供便捷的模型服务 Serving 能力，支持传统 NLP 模型应用和LLM 大模型一键部署，自带模型安全、审计等管理能力，支持模型服务的弹性扩容与持续可观测监控预警。
- 运维看板 ：支持监控看板，算力资源与任务资源一览无余，实现全自动平台可观测，更提供详细的指标监控与运维产品能力。
- GPU 管理 ：支持自动化 GPU 硬件发现，实现 GPU 套件全自动安装，用户无需处理繁琐的 GPU 驱动、CUDA 等部署工作，提供提供统一 GPU 资源看板与 调度情况分析。
- 队列管理 ：支持大模型算力资源的统一调度队列管理平台，支持用户全自动队列隔离能力，实现不同业务与算力需求互不干扰，从基础设施层面实现数据与算力资源安全隔离。

逻辑架构图

注册并体验 **d.run**

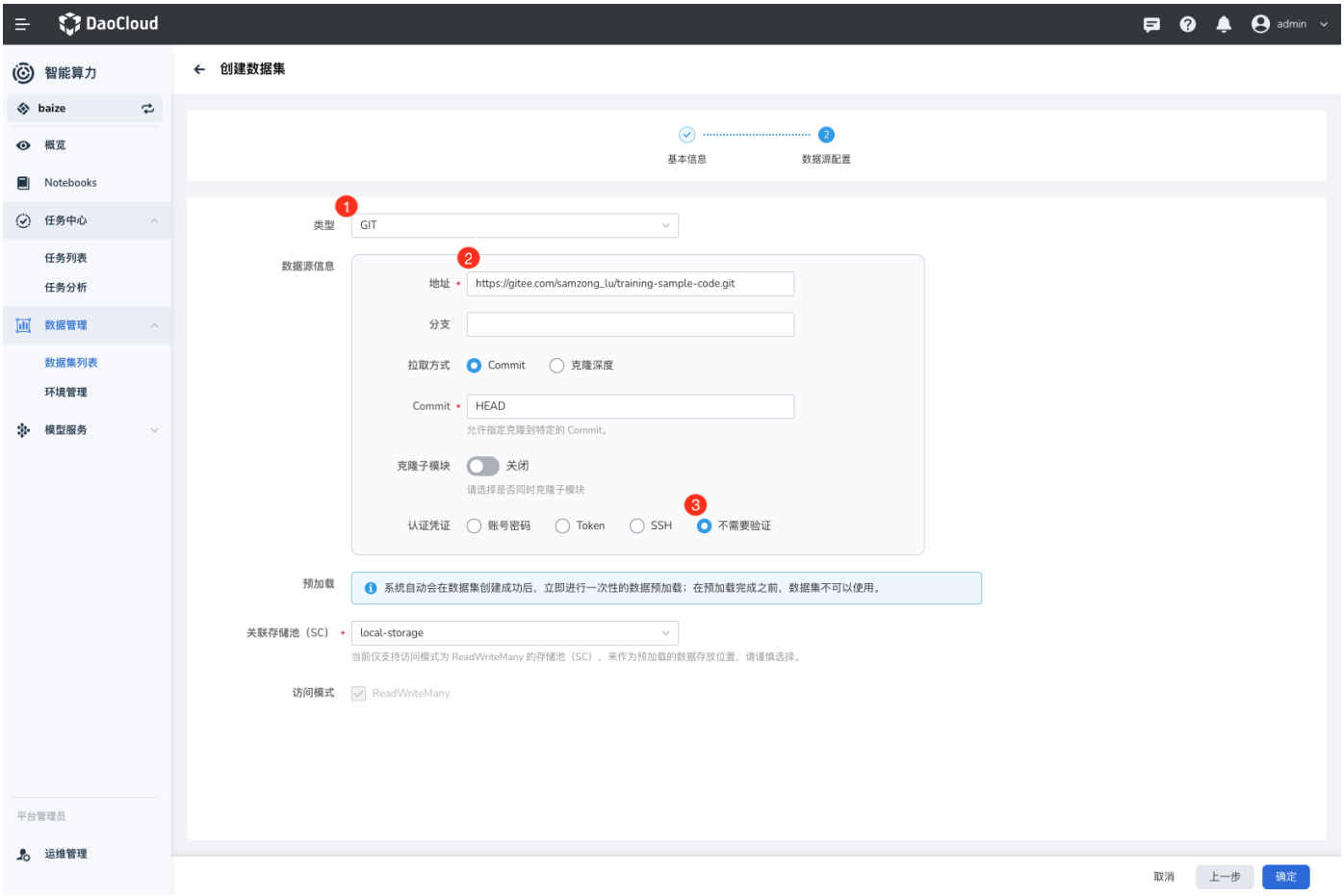
快速入门

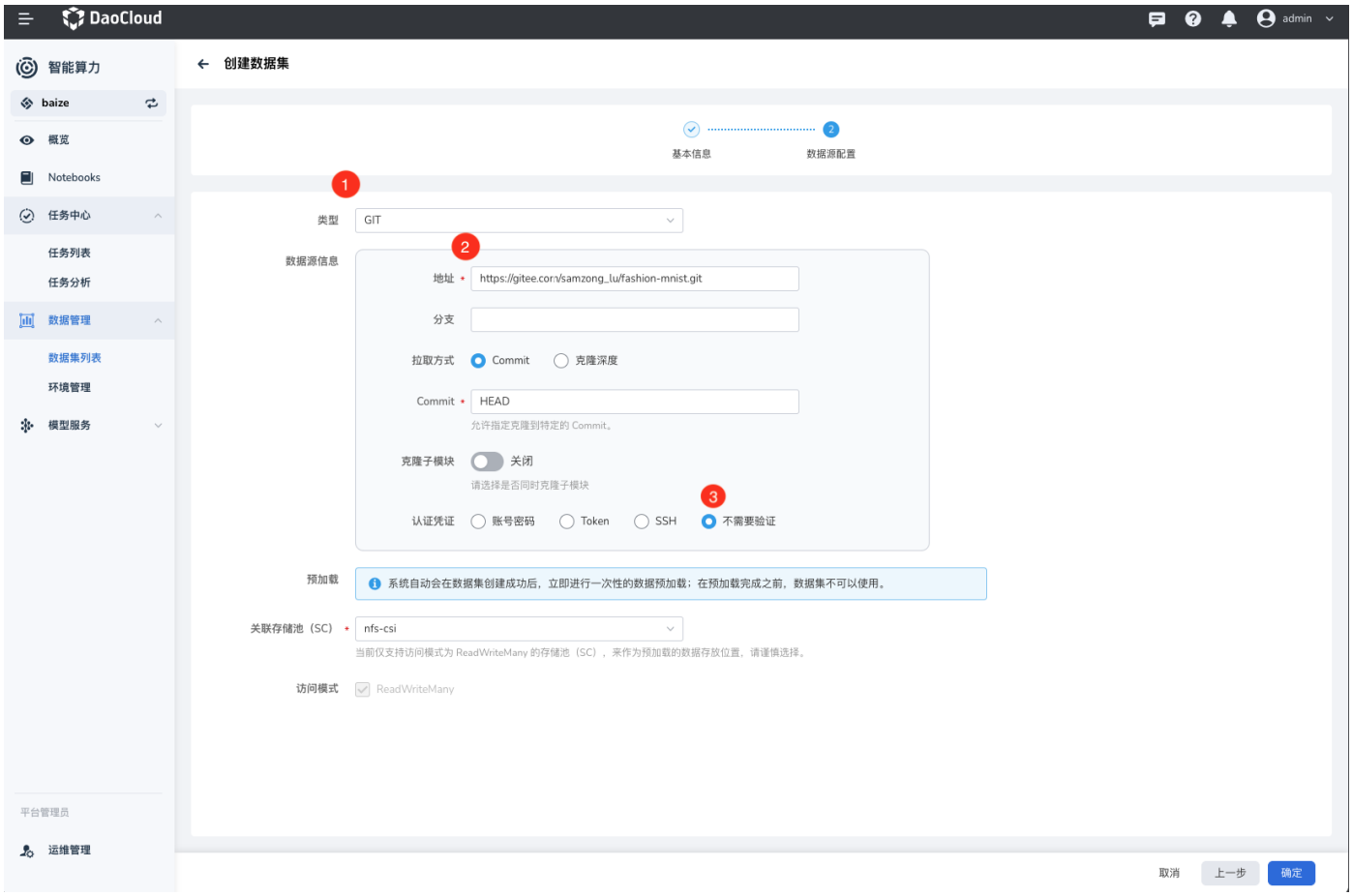
本文提供了简单的操作手册以便用户使用 d.run 算法开发进行数据集、Notebook、任务训练的整个开发、训练流程。

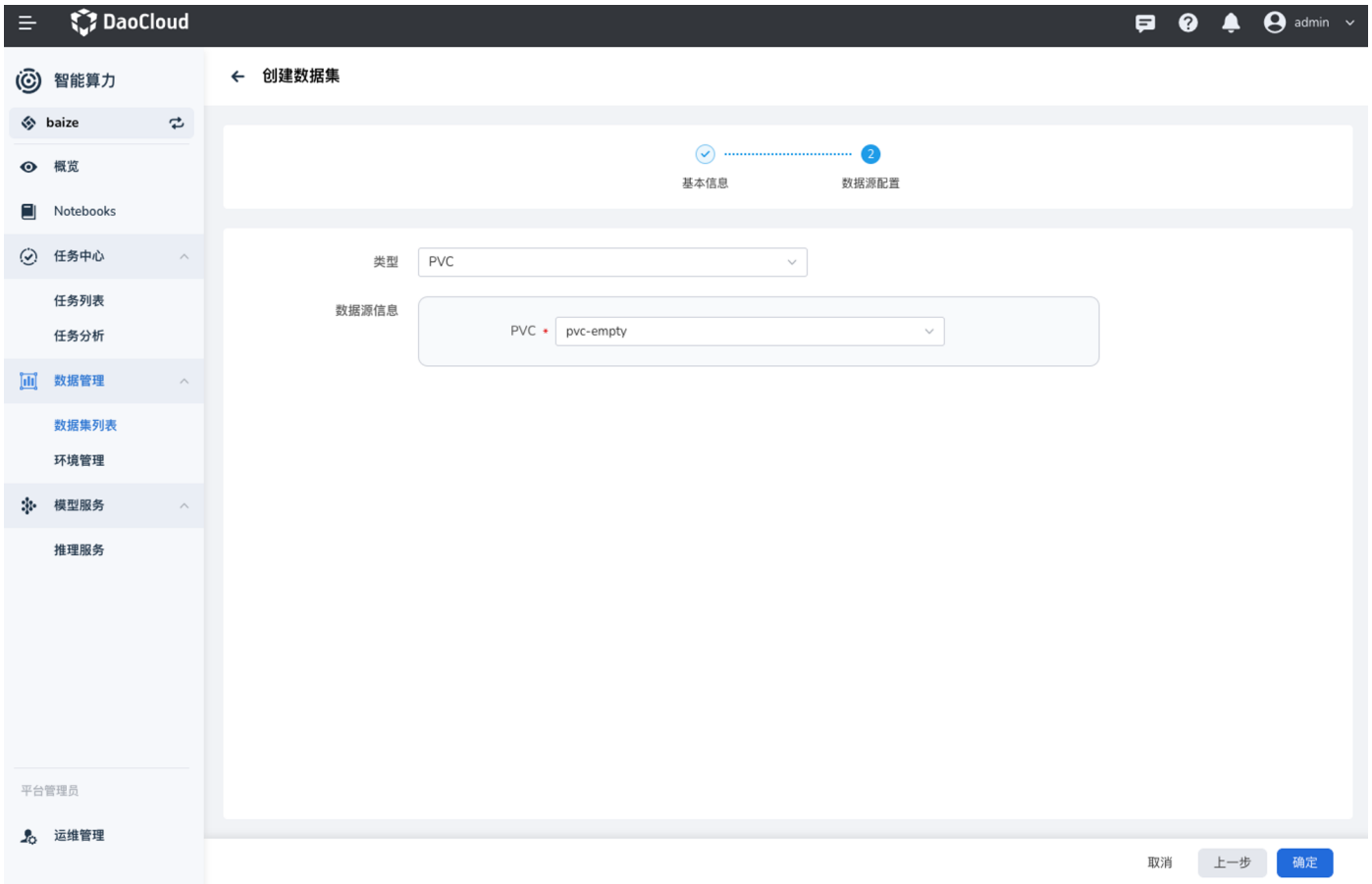
1. 点击导航栏的 数据管理 -> 数据集列表 ，点击 创建 。创建三个数据集，分别为：

- 代码： https://github.com/d-run/drun-samples
- 国内慢可以使用 gitee 加速： https://gitee.com/samzong_lu/training-sample-code.git
- 数据： https://github.com/zalandoresearch/fashion-mnist
- 国内慢可以使用 gitee 加速： https://gitee.com/samzong_lu/fashion-mnist.git
- 空 PVC：创建一个空的 PVC 以便输出训练结束的模型和日志。

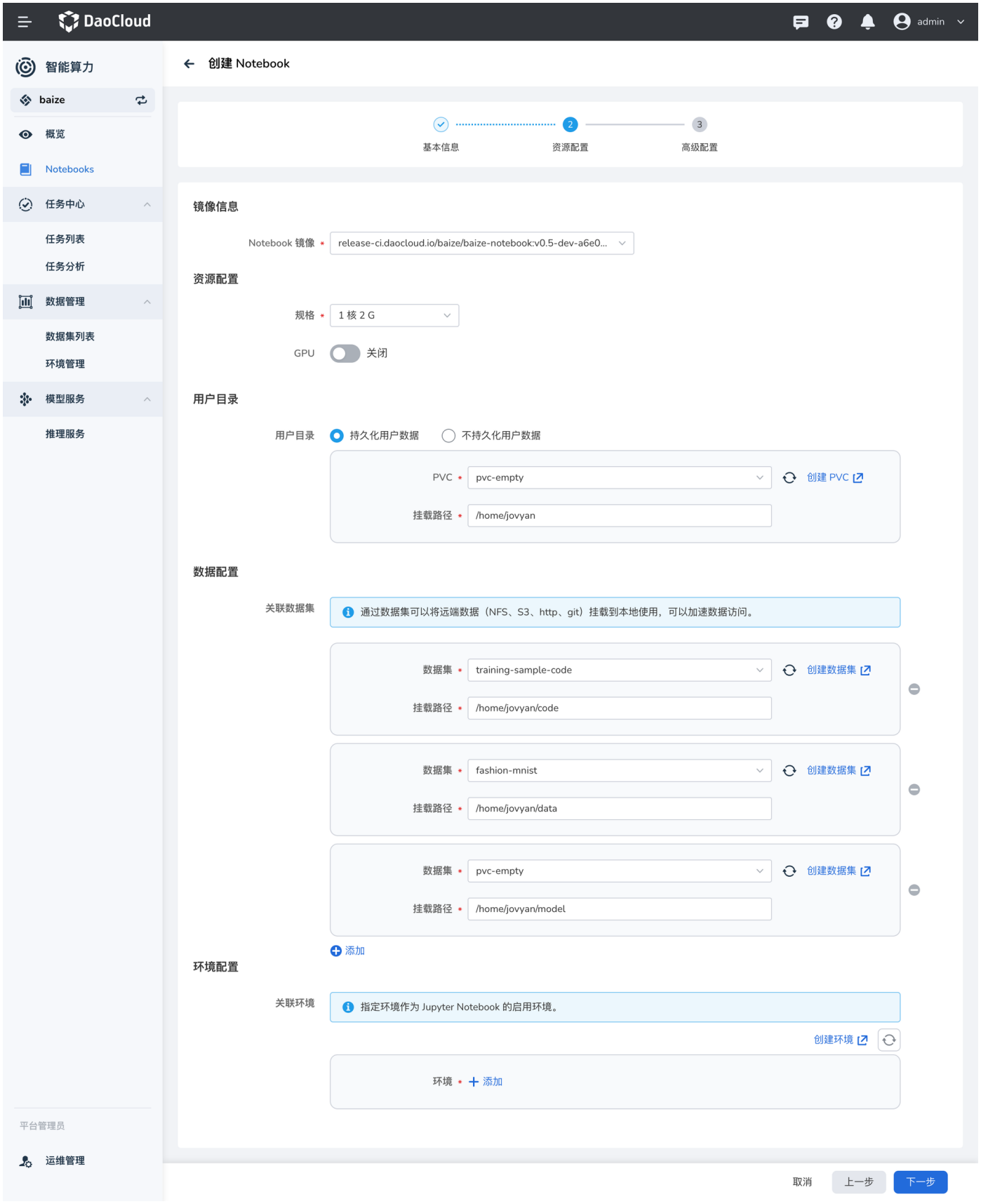
> **Note**
>
> 目前仅支持读写模式为 `ReadWriteMany` 的 `StorageClass`，请使用 NFS 或者推荐使用 JuiceFS。




DaoCloud
admin
智能算力
baize
概览
Notebooks
任务中心
任务列表
任务分析
数据管理
数据集列表
环境管理
模型服务
平台管理员
运维管理
创建数据集
基本信息
数据源配置
1
类型
GIT
数据源信息
2
地址 *
https://gitee.com/samzong_lu/fashion-mnist.git
分支
拉取方式
Commit
克隆深度
Commit *
HEAD
允许指定克隆到特定的 Commit。
克隆子模块
关闭
请选择是否同时克隆子模块
3
认证凭证
账号密码
Token
SSH
不需要验证
预加载
系统自动会在数据集创建成功后，立即进行一次性的数据预加载；在预加载完成之前，数据集不可以使用。
关联存储池（SC） *
nfs-csi
当前仅支持访问模式为 ReadWriteMany 的存储池（SC），来作为预加载的数据存放位置，请谨慎选择。
访问模式
ReadWriteMany
取消
上一步
确定

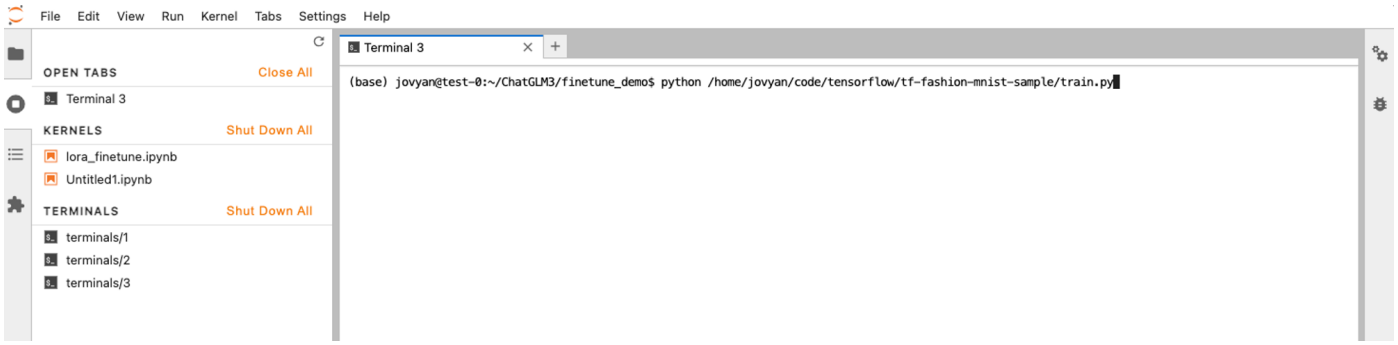

DaoCloud
admin
智能算力
baize
概览
Notebooks
任务中心
任务列表
任务分析
数据管理
数据集列表
环境管理
模型服务
推理服务
平台管理员
运维管理
创建数据集
基本信息
2
数据源配置
类型
PVC
数据源信息
PVC *
pvc-empty
取消
上一步
确定

2. 准备开发环境，点击导航栏的 **Notebooks** ，点击 创建 。将上一步中创建的三个数据集进行关联，挂载路径请参照下图填写：



3. 等待 Notebook 创建成功，点击列表中的访问地址，进入 Notebook。并在 Notebook 的终端中执行以下命令进行任务训练。

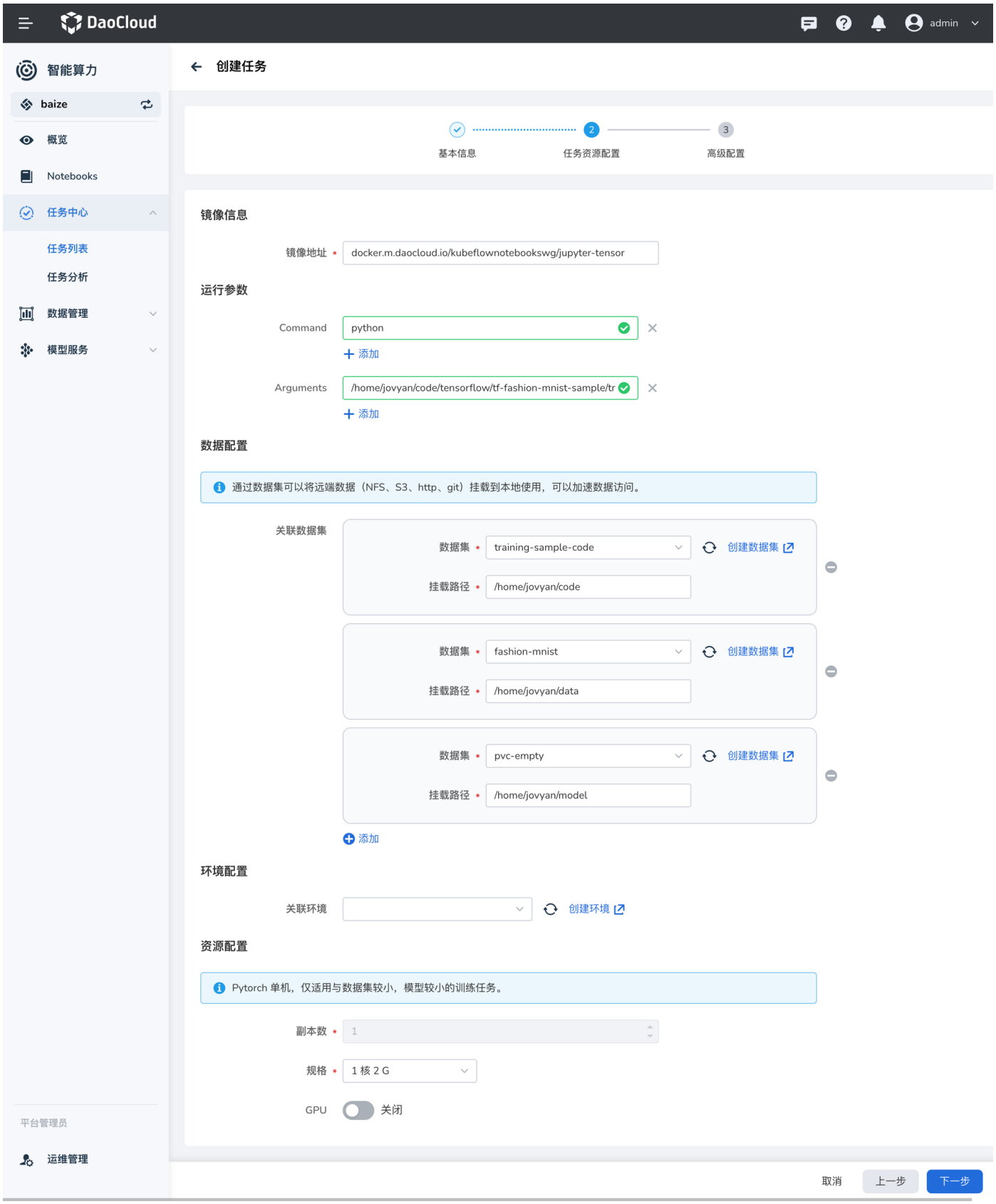
```
python /home/jovyan/code/tensorflow/tf-fashion-mnist-sample/train.py
```



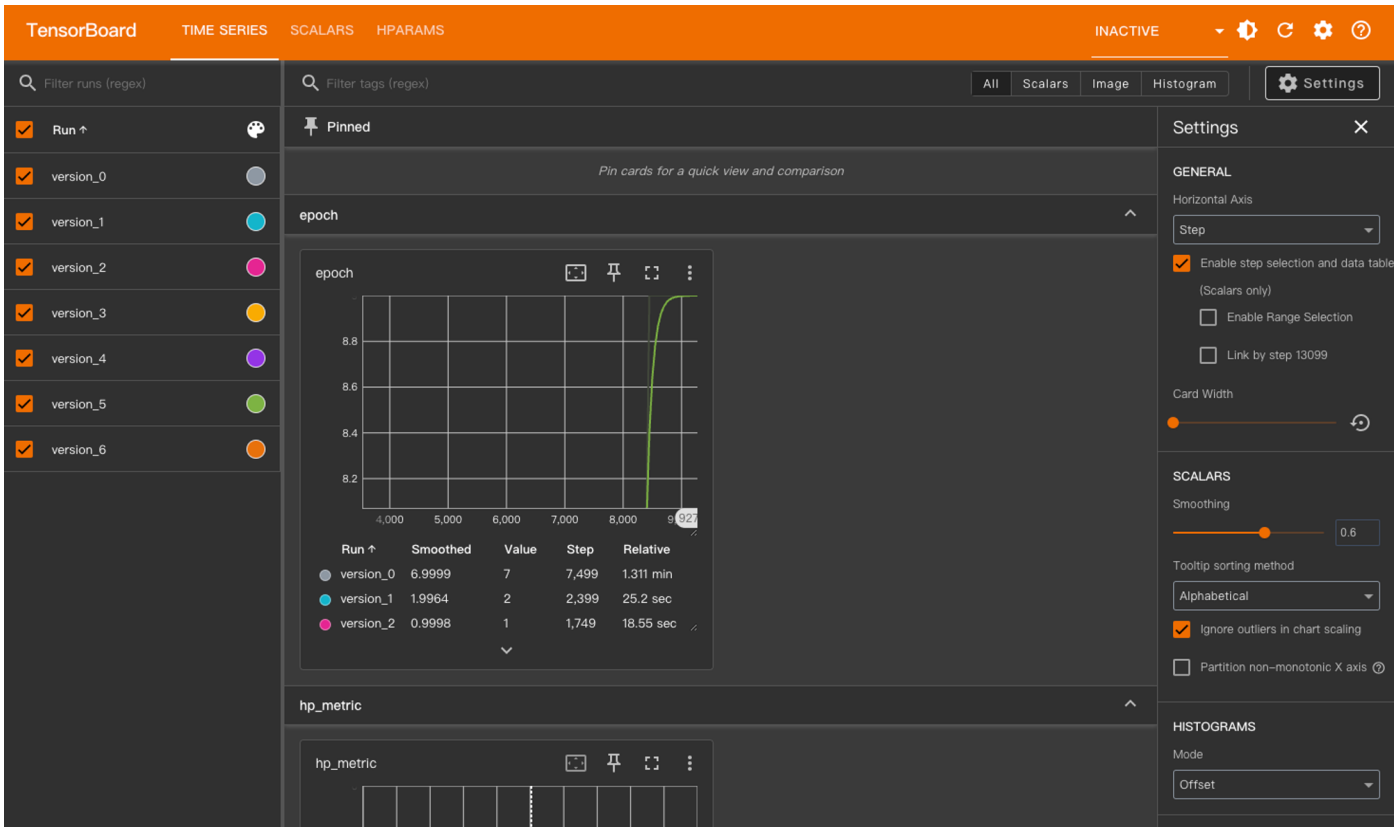
4. 点击导航栏的 任务中心 -> 任务管理 ，创建一个 `Tensorflow` 单机任务，任务配置参考下图， 并开启 任务分析（**Tersorboard**） 功能，点击 创建 后等待状态完成。

- 镜像地址填写：`release.daocloud.io/baize/jupyter-tensorflow-full:v1.8.0-baize`
- Command：`python`
- Arguments：`/home/jovyan/code/tensorflow/tf-fashion-mnist-sample/train.py`

> **Note**
>
> 数据集或模型较大时，建议在第二步资源配置中开启 GPU 配置。



5. 在上一步创建成功的任务，即可在任务分析中，点击对应任务分析中的访问查看，查看任务状态并对其进行任务训练的调优。


TensorBoard
TIME SERIES
SCALARS
HPARAMS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run
version_0
version_1
version_2
version_3
version_4
version_5
version_6
Pinned
Pin cards for a quick view and comparison
epoch
8.8
8.6
8.4
8.2
4,000
5,000
6,000
7,000
8,000
Run
Smoothed
Value
Step
Relative
version_0 6.9999 7 7,499 1.311 min
version_1 1.9964 2 2,399 25.2 sec
version_2 0.9998 1 1,749 18.55 sec
hp_metric
Settings
GENERAL
Horizontal Axis
Step
Enable step selection and data table
(Scalars only)
Enable Range Selection
Link by step 13099
Card Width
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset

算法开发功能特性

作为训推一体化算力平台，依托于 DCE AI 算力调度，提供数据编排、开发环境管理、任务管理、GPU 管理、队列管理， 最大化算力效用并降低算力开销，并且还提供了优化的 AI 开发框架，简化 AI 开发和部署。

功能模块	描述
算力资源全托管	依托于 DCE（DaoCloud Enterprise），提供强大的基础设施能力，支持超大规模算力集群、异构 GPU 等一站式托管，并提供一系列如 vGPU 等软硬一体加速方案。
数据编排	支持模型开发生命周期中数据管理与编排能力，提供多数据源接入，数据集管理、超大训练数据预热等能力，并且从底层存储引擎优化，保障数据的安全与高效利用。
开发管理	提供主流的模型开发工具，满足 MLOps 和 LLMOps 工程师和科学家们对开发工具的需求，支持快速拉起高性能开发环境，一键申请高性能 GPU、软件依赖、训练数据等资源。
任务管理	支持训练任务的全生命周期管理，提供多种快速创建任务的方式；支持 Pytorch、TensorFlow、PaddlePaddle 等主流任务框架，天然支持单机、分布式、多节点、多卡等多种类任务调度。
模型推理	提供便捷的模型服务 Serving 能力，支持传统 NLP 模型应用和 LLM 大模型一键部署，自带模型安全、审计等管理能力，支持模型服务的弹性扩容与持续可观测监控预警。
运维看板	支持监控看板，算力资源与任务资源一览无余，实现全自动平台可观测，更提供详细的指标监控与运维产品能力。
GPU 管理	支持自动化 GPU 硬件发现，实现 GPU 套件全自动安装，用户无需处理繁琐的 GPU 驱动、CUDA 等部署工作，提供提供统一 GPU 资源看板与调度情况分析。
队列管理	支持大模型算力资源的统一调度队列管理平台，支持用户全自动队列隔离能力，实现不同业务与算力需求互不干扰，从基础设施层面实现数据与算力资源安全隔离。

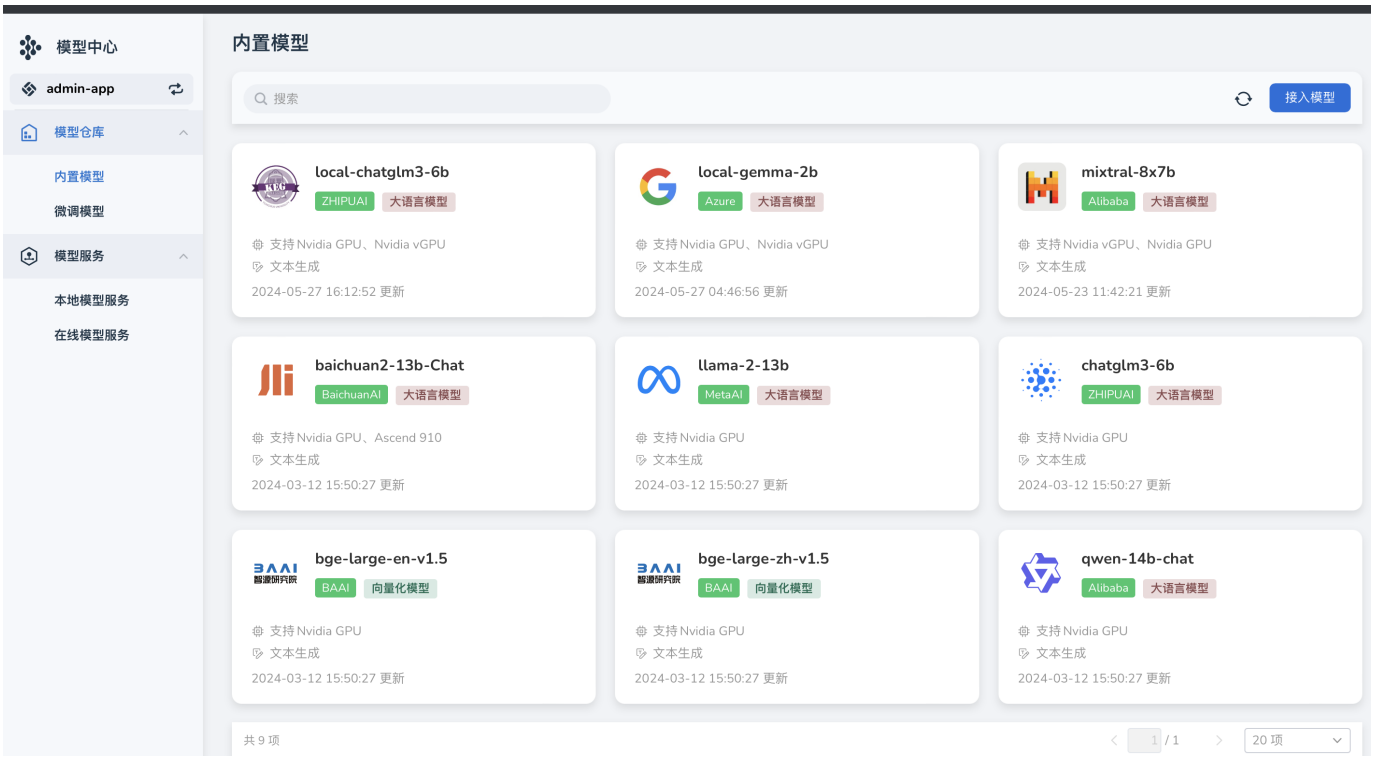
1.4.3 模型中心

什么是模型中心

d.run 模型中心是一款功能强大的模型管理和服务平台，旨在为用户提供便捷、高效的模型管理和使用体验。以下是产品的主要特点和优势：

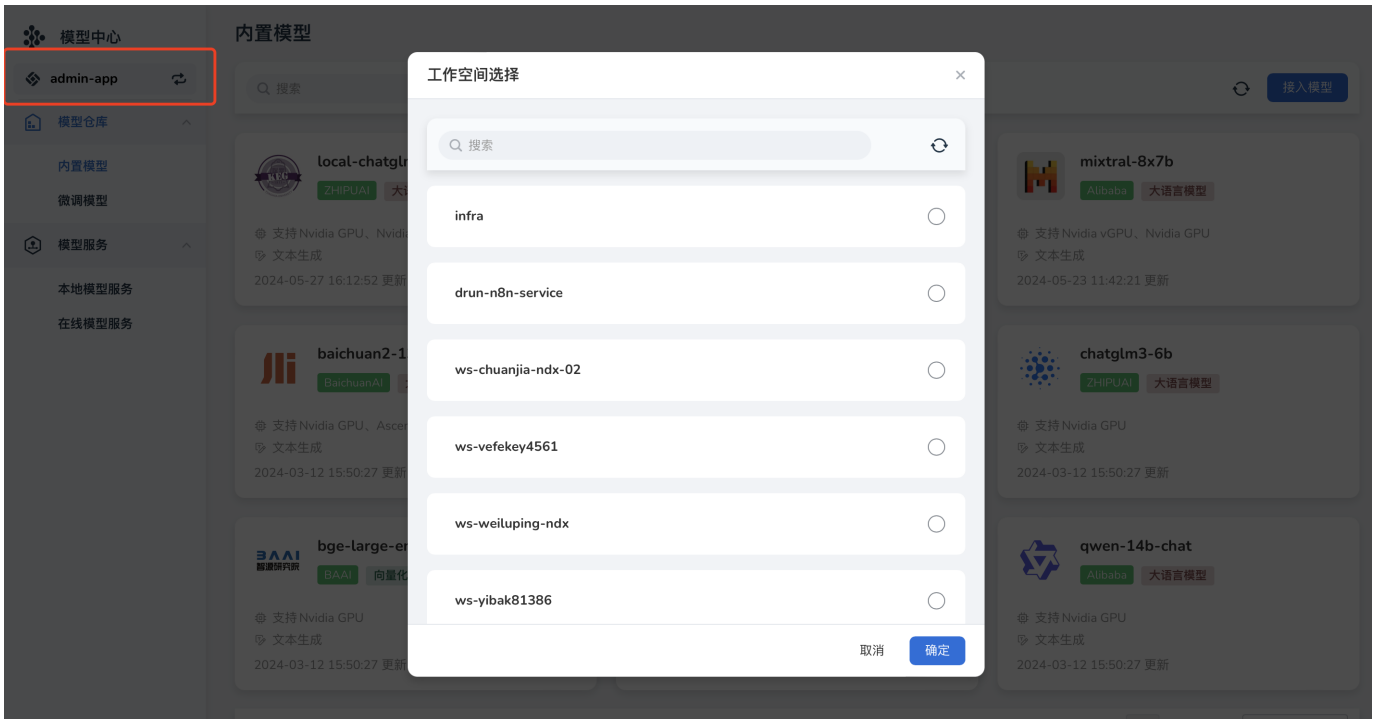
- 多模型支持 ： 模型中心支持 GLM、Llama、百川、文心一言等系列模型，以及 Transformer 架构的模型，满足用户在不同任务和场景下的需求。
- 直观图形界面 ： 提供直观的图形化界面，让用户轻松地进行模型推理和管理，无需复杂的操作和编程知识。
- 微调支持 ： 用户可以对模型进行微调，并通过可视化界面进行推理，以实现更精准和个性化的模型效果。
- 对话内容对比 ： 提供对话内容对比功能，让用户能够直观地比较不同模型推理结果的对话内容，以便评估模型效果和调整策略。
- 全面的模型服务管理 ： 提供全面的本地和在线模型服务管理，覆盖模型服务的整个生命周期，包括部署、监控和更新。
- 智能负载均衡 ： 实现智能负载均衡，优化模型服务的使用效率，提高用户访问模型的速度和稳定性。
- **API Key** 管理 ： 提供在线模型服务的 API Key 管理功能，保证模型服务的安全性和可控性。

d.run 模型中心内置了模型仓库，可以一站式管理各类大模型，按需部署。 另外通过模型服务，您可以从本地部署大语言模型服务和向量化模型服务，也可接入在线的大语言模型服务。



切换工作空间

在导航栏左上角，可以切换工作空间。


模型中心
admin-app
模型仓库
内置模型
微调模型
模型服务
本地模型服务
在线模型服务
内置模型
搜索
接入模型
工作空间选择
搜索
infra
drun-n8n-service
ws-chuanjia-ndx-02
ws-vefekey4561
ws-weiluping-ndx
ws-yibak81386
取消
确定
local-chatglm
ZHIPUAI
支持 Nvidia GPU、Nvidia
文本生成
2024-05-27 16:12:52 更新
mixtral-8x7b
Alibaba
大语言模型
支持 Nvidia vGPU、Nvidia GPU
文本生成
2024-05-23 11:42:21 更新
baichuan2-13
BaichuanAI
支持 Nvidia GPU、Ascend
文本生成
2024-03-12 15:50:27 更新
chatglm3-6b
ZHIPUAI
大语言模型
支持 Nvidia GPU
文本生成
2024-03-12 15:50:27 更新
bge-large-en
BAAI
向量化
支持 Nvidia GPU
文本生成
2024-03-12 15:50:27 更新
qwen-14b-chat
Alibaba
大语言模型
支持 Nvidia GPU
文本生成
2024-03-12 15:50:27 更新

功能特性

模型中心的功能特性参见下表：

主要功能	细分项
模型管理	提供图形化界面，能够直观地进行GLM系列模型的推理
	提供图形化界面，能够直观地进行Llama系列模型的推理
	提供图形化界面，能够直观地进行百川系列模型的推理
	提供图形化界面，能够直观地进行文心一言系列模型的推理
	支持查看模型微调出来的模型
	可视化进行微调模型推理
	提供图形化界面，能够直观地进行微调模型推理的对话内容对比
模型服务	提供全面的本地模型服务管理，覆盖模型服务的整个生命周期
	提供在线模型服务的API key管理，并实现智能负载均衡，提升使用效率

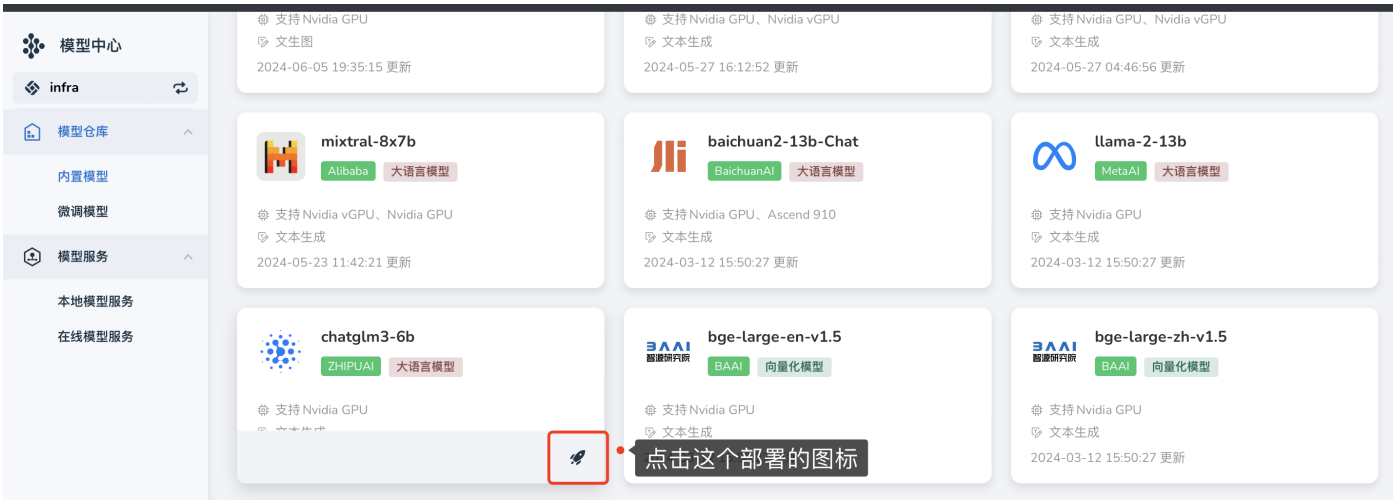
模型仓库

内置模型

内置模型是 d.run 按照规范适配验证后上线到模型中心的，您可以直接部署。

如果想要部署某个模型，

1. 在模型卡片右下角，点击 部署 按钮。



2. 填写模型服务名称、部署配置、算力类型、资源配置后点击 确定 。

> **支持国产 GPU**
>
> 其中算力类型支持 Nvidia GPU 和 Ascend 等国产 GPU。

3. 创建成功后，可以通过部署的模型提供服务。

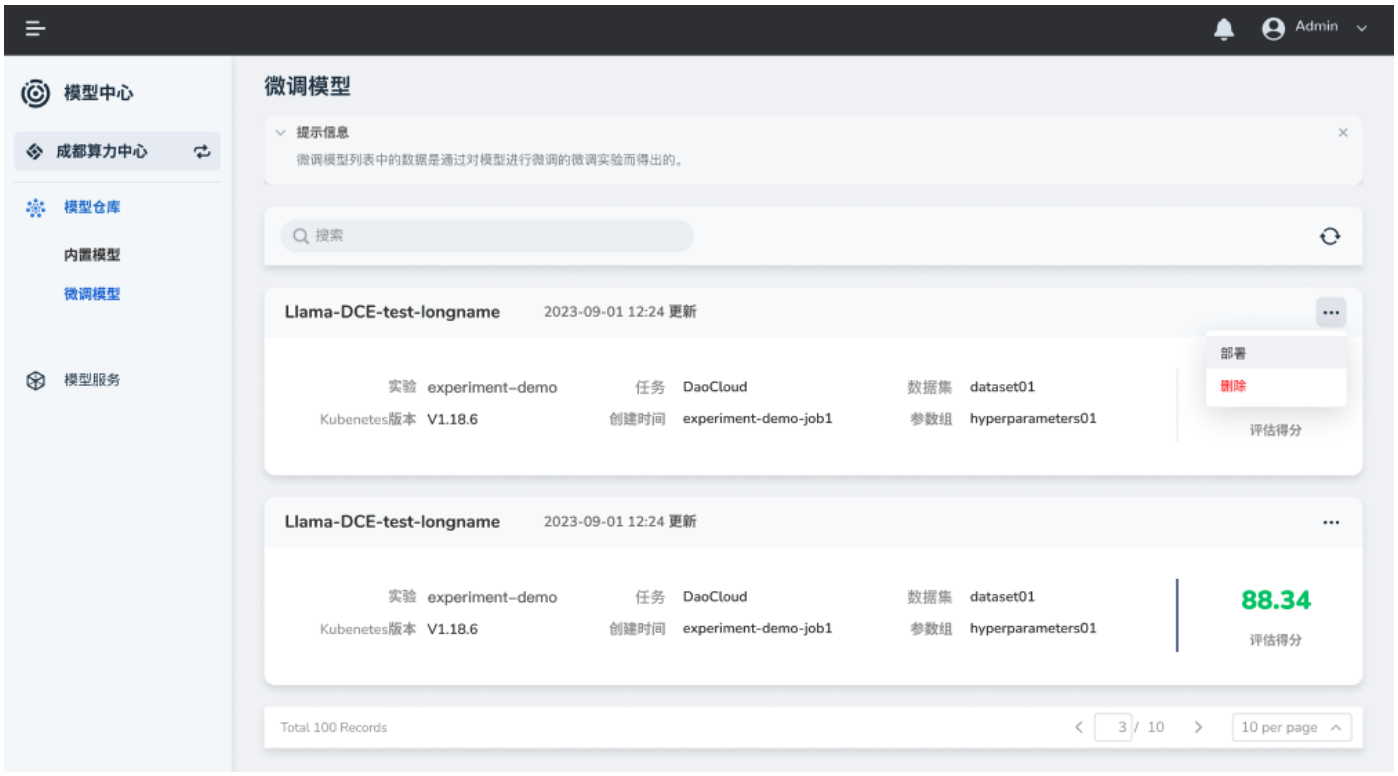
下一步：模型服务

微调模型

微调模型是指通过 DataTunerX 等工具的微调得出的模型，部署该模型的服务后，可通过问答的方式检验微调的效果

部署

1. 点击右侧的 ... ，在弹出的选项中选择 部署 。



2. 填写模型服务名称、命名空间、算力配额、资源配置后点击 确定 （注：资源给的太少服务会起不来）。

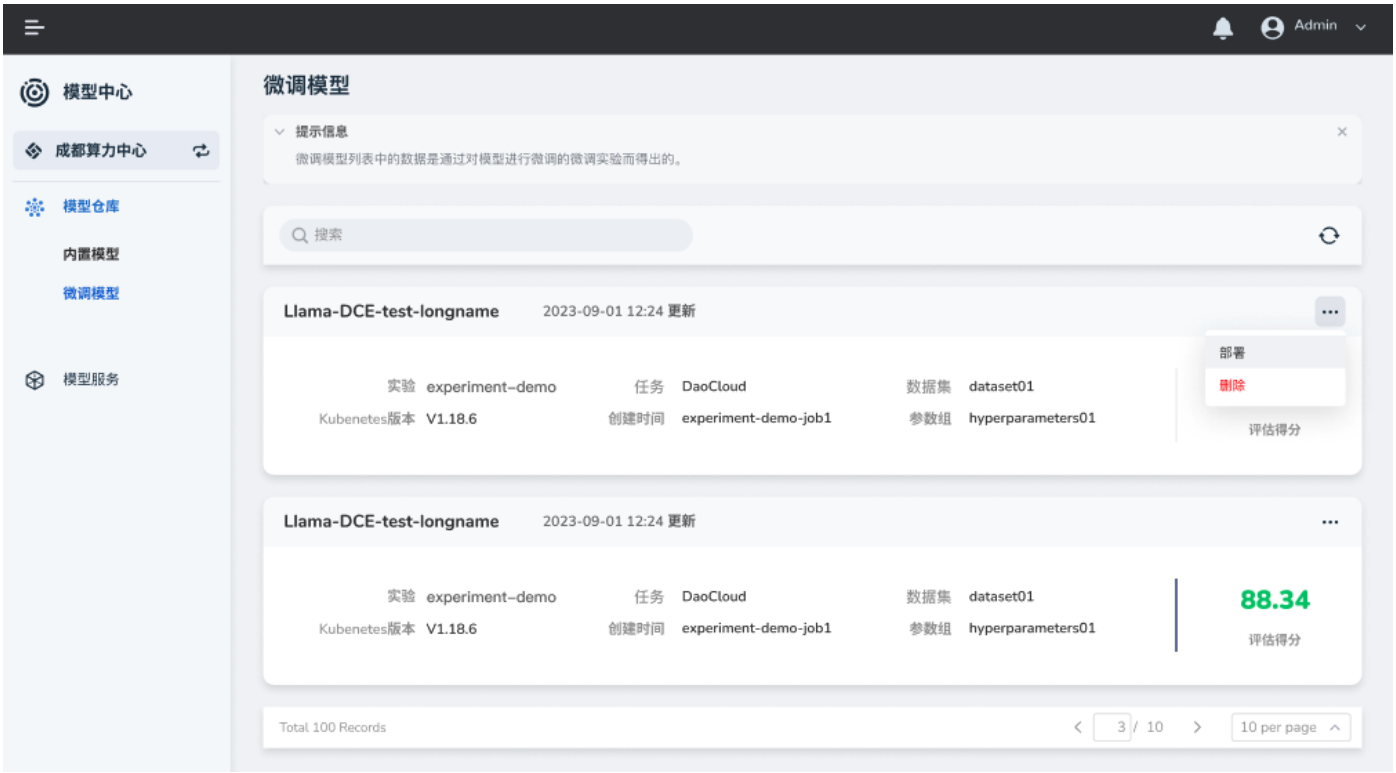
> **支持国产 GPU**
>
> 其中算力类型支持 Nvidia GPU 和 Ascend 等国产 GPU。

3. 创建成功，接下来可以通过部署的模型提供服务。

下一步：模型服务

删除

1. 点击右侧的 **...** ，在弹出的选项中选择 **删除** 。



2. 输入模型名称，确认无误后删除。

> **Note**
>
> 删除操作不可逆，请谨慎操作。

接入模型镜像

d.run 模型中心支持接入 HuggingFace Transformers 上托管的各种生成式 Transformer 模型。

以下是目前支持的模型架构列表。

架构	模型	**HuggingFace** 模型示例	`LoRA <lora>`
`AquilaForCausalLM`	Aquila	`BAAI/Aquila-7B`, `BAAI/AquilaChat-7B` 等	?
`BaiChuanForCausalLM`	Baichuan	`baichuan-inc/Baichuan2-13B-Chat`, `baichuan-inc/Baichuan-7B` 等	?
`ChatGLMModel`	ChatGLM	`THUDM/chatglm2-6b`, `THUDM/chatglm3-6b` 等	?
`CohereForCausalLM`	Command-R	`CohereForAI/c4ai-command-r-v01` 等	
`DbrxForCausalLM`	DBRX	`databricks/dbrx-base`, `databricks/dbrx-instruct` 等	
`DeciLMForCausalLM`	DeciLM	`Deci/DeciLM-7B`, `Deci/DeciLM-7B-instruct` 等	
`BloomForCausalLM`	BLOOM, BLOOMZ, BLOOMChat	`bigscience/bloom`, `bigscience/bloomz` 等	
`FalconForCausalLM`	Falcon	`tiiuae/falcon-7b`, `tiiuae/falcon-40b`, `tiiuae/falcon-rw-7b` 等	
`GemmaForCausalLM`	Gemma	`google/gemma-2b`, `google/gemma-7b` 等	?
`GPT2LMHeadModel`	GPT-2	`gpt2`, `gpt2-xl` 等	
`GPTBigCodeForCausalLM`	StarCoder, SantaCoder, WizardCoder	`bigcode/starcoder`, `bigcode/gpt_bigcode-santacoder`, `WizardLM/WizardCoder-15B-V1.0` 等	
`GPTJForCausalLM`	GPT-J	`EleutherAI/gpt-j-6b`, `nomic-ai/gpt4all-j` 等	
`GPTNeoXForCausalLM`	GPT-NeoX, Pythia, OpenAssistant, Dolly V2, StableLM	`EleutherAI/gpt-neox-20b`, `EleutherAI/pythia-12b`, `OpenAssistant/oasst-sft-4-pythia-12b-epoch-3.5`, `databricks/dolly-v2-12b`, `stabilityai/stablelm-tuned-alpha-7b` 等	
`InternLMForCausalLM`	InternLM	`internlm/internlm-7b`, `internlm/internlm-chat-7b` 等	?
`InternLM2ForCausalLM`	InternLM2	`internlm/internlm2-7b`, `internlm/internlm2-chat-7b` 等	
`JAISLMHeadModel`	Jais	`core42/jais-13b`, `core42/jais-13b-chat`, `core42/jais-30b-v3`, `core42/jais-30b-chat-v3` 等	
`LlamaForCausalLM`	LLaMA, Llama 2, Meta Llama 3, Vicuna, Alpaca, Yi	`meta-llama/Meta-Llama-3-8B-Instruct`, `meta-llama/Meta-Llama-3-70B-Instruct`, `meta-llama/Llama-2-13b-hf`, `meta-llama/Llama-2-70b-hf`, `openlm-research/open_llama_13b`, `lmsys/vicuna-13b-v1.3`, `01-ai/Yi-6B`, `01-ai/Yi-34B` 等	?
`MiniCPMForCausalLM`	MiniCPM	`openbmb/MiniCPM-2B-sft-bf16`, `openbmb/MiniCPM-2B-dpo-bf16` 等	
`MistralForCausalLM`	Mistral, Mistral-Instruct	`mistralai/Mistral-7B-v0.1`, `mistralai/Mistral-7B-Instruct-v0.1` 等	?
`MixtralForCausalLM`	Mixtral-8x7B, Mixtral-8x7B-Instruct	`mistralai/Mixtral-8x7B-v0.1`, `mistralai/Mixtral-8x7B-Instruct-v0.1`, `mistral-community/Mixtral-8x22B-v0.1` 等	?
`MPTForCausalLM`	MPT, MPT-Instruct, MPT-Chat, MPT-StoryWriter	`mosaicml/mpt-7b`, `mosaicml/mpt-7b-storywriter`, `mosaicml/mpt-30b` 等	
`OLMoForCausalLM`	OLMo	`allenai/OLMo-1B-hf`, `allenai/OLMo-7B-hf` 等	
`OPTForCausalLM`	OPT, OPT-IML	`facebook/opt-66b`, `facebook/opt-iml-max-30b` 等	

架构	模型	**HuggingFace** 模型示例	`LoRA <lora>`
`OrionForCausalLM`	Orion	`OrionStarAI/Orion-14B-Base`, `OrionStarAI/Orion-14B-Chat` 等	
`PhiForCausalLM`	Phi	`microsoft/phi-1_5`, `microsoft/phi-2` 等	
`Phi3ForCausalLM`	Phi-3	`microsoft/Phi-3-mini-4k-instruct`, `microsoft/Phi-3-mini-128k-instruct` 等	
`QWenLMHeadModel`	Qwen	`Qwen/Qwen-7B`, `Qwen/Qwen-7B-Chat` 等	
`Qwen2ForCausalLM`	Qwen2	`Qwen/Qwen2-beta-7B`, `Qwen/Qwen2-beta-7B-Chat` 等	?
`Qwen2MoeForCausalLM`	Qwen2MoE	`Qwen/Qwen1.5-MoE-A2.7B`, `Qwen/Qwen1.5-MoE-A2.7B-Chat` 等	
`StableLmForCausalLM`	StableLM	`stabilityai/stablelm-3b-4e1t/` , `stabilityai/stablelm-base-alpha-7b-v2` 等	

如何为模型构建镜像

以下是完整的指导流程，包含拉取模型、构建 Docker 镜像、配置私有仓库，以及将镜像推送到自定义仓库的步骤。

拉取 Hugging Face 上的模型 获取模型的 Clone 地址

1. 前往 Hugging Face
2. 找到目标模型，例如 "chatglm3-6b"
3. 复制模型的 Git URL 以用于克隆，例如：

```
https://huggingface.co/THUDM/chatglm3-6b
```

Clone 模型

使用以下命令克隆模型（确保已安装 Git 和 Git LFS）：

```
git lfs install
git clone https://huggingface.co/THUDM/chatglm3-6b
```

如果克隆失败，请尝试以下步骤：

```
# 跳过 smudge - 我们会在后面的步骤中以更快的方式批量下载二进制文件
git lfs install --skip-smudge
# 在这里执行 git clone
git clone https://huggingface.co/THUDM/chatglm3-6b
# 进入克隆的目录（如果是其他模型，注意替换这个目录）
cd chatglm3-6b
# 在新的克隆中获取所有的二进制文件
git lfs pull
# 重新设置 smudge
git lfs install --force
```

构建镜像 创建 Dockerfile

创建一个名为 `Dockerfile` 的文件，并粘贴以下内容。确保根据实际需求设置 `MODEL_NAME` 。

```
# 基于vllm/vllm-openai:v0.4.1构建
FROM vllm/vllm-openai:v0.4.1

# 安装Python包
RUN pip install tiktoken

# 复制模型到容器
COPY . /data

# 使用shell形式的CMD
ENTRYPOINT ["python3", "-m", "vllm.entrypoints.openai.api_server", "--model", "/data", "--trust-remote-code"]
```

构建 Docker 镜像

构建 Docker 镜像时，示例中使用 `vllm-openai-tiktoken-chatglm3-6b-server` 作为镜像名称， `/data/chatglm3-6b/Dockerfile` 是上面的 Dockerfile 文件位置， `/data/llms/chatglm3-6b` 是下载的大模型文件的目录。

```
docker build -t vllm-openai-tiktoken-chatglm3-6b-server:v2.0.1 -f /data/chatglm3-6b/Dockerfile /data/llms/chatglm3-6b
```

为了减小镜像大小，确保 .dockerignore 文件在 /data/llms/chatglm3-6b 目录中，并且其中包含需要忽略的文件或目录，如：

```
.git
```

推送镜像到自定义 Docker 仓库 登录到 Docker 仓库

如果要推送到 Docker Hub 或其他私有仓库，请确保已登录到相应的 Docker 仓库。

```
docker login
```

对于其他仓库，例如 Google Container Registry 或 Amazon ECR，需要根据其文档指引进行登录和身份验证。

配置镜像标签

在推送镜像之前，将镜像标签配置为目标仓库地址。以下示例使用自定义的私有仓库地址：

```
# 假设你的仓库地址是 myregistry.example.com
docker tag vllm-openai-tiktoken-chatglm3-6b-server myregistry.example.com/vllm-openai-tiktoken-chatglm3-6b-server
```

推送镜像到仓库

将镜像推送到目标仓库：

```
docker push myregistry.example.com/vllm-openai-tiktoken-chatglm3-6b-server
```

验证推送成功

确认推送成功后，可以在 Docker 仓库的仪表盘上查看已推送的镜像。

模型服务

本地模型服务

顾名思义，此类模型部署在本地或内网，在限定地理范围内提供服务。 在本地可提供大语言和向量化两种模型服务。

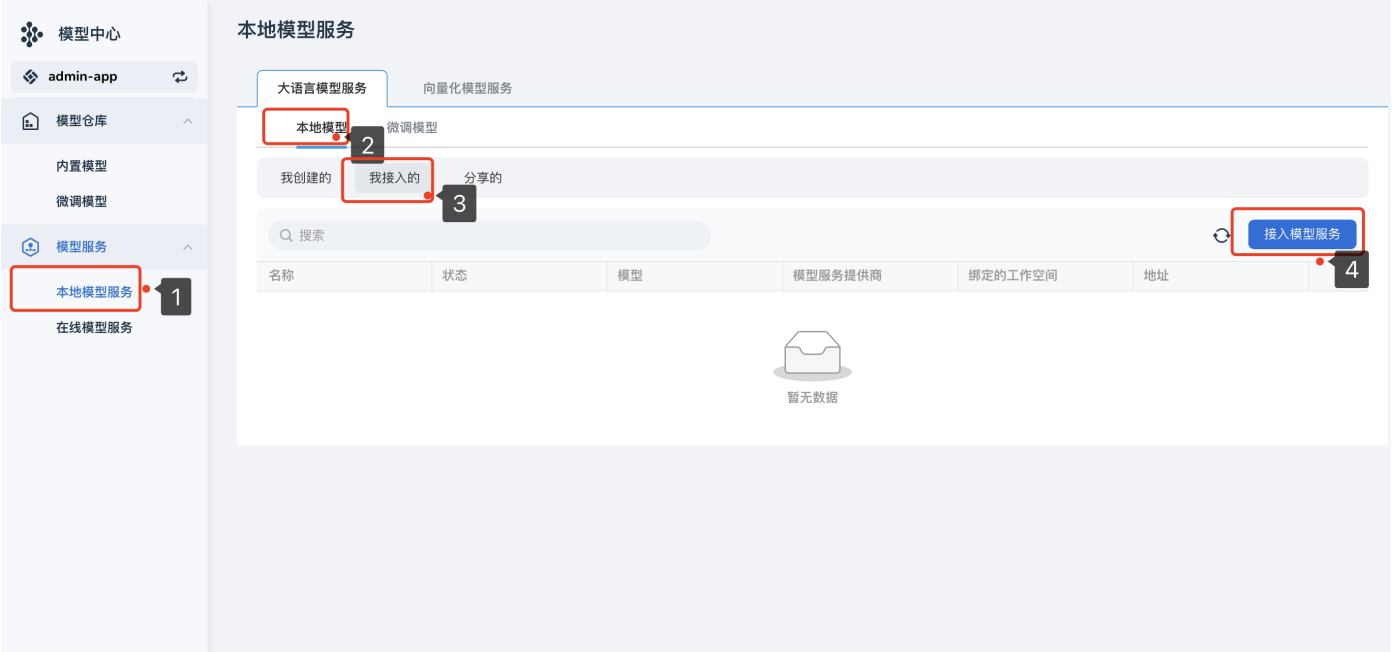
大语言模型服务

大语言模型即 Large Language Model (LLM)，这是基于大量数据进行预训练的超大型深度学习模型。 可采用内置和微调两种模型提供服务。

内置模型

此类模型可直接接入，提供服务。

1. 点击中间 我接入的 ，点击右侧的 接入模型服务 按钮。

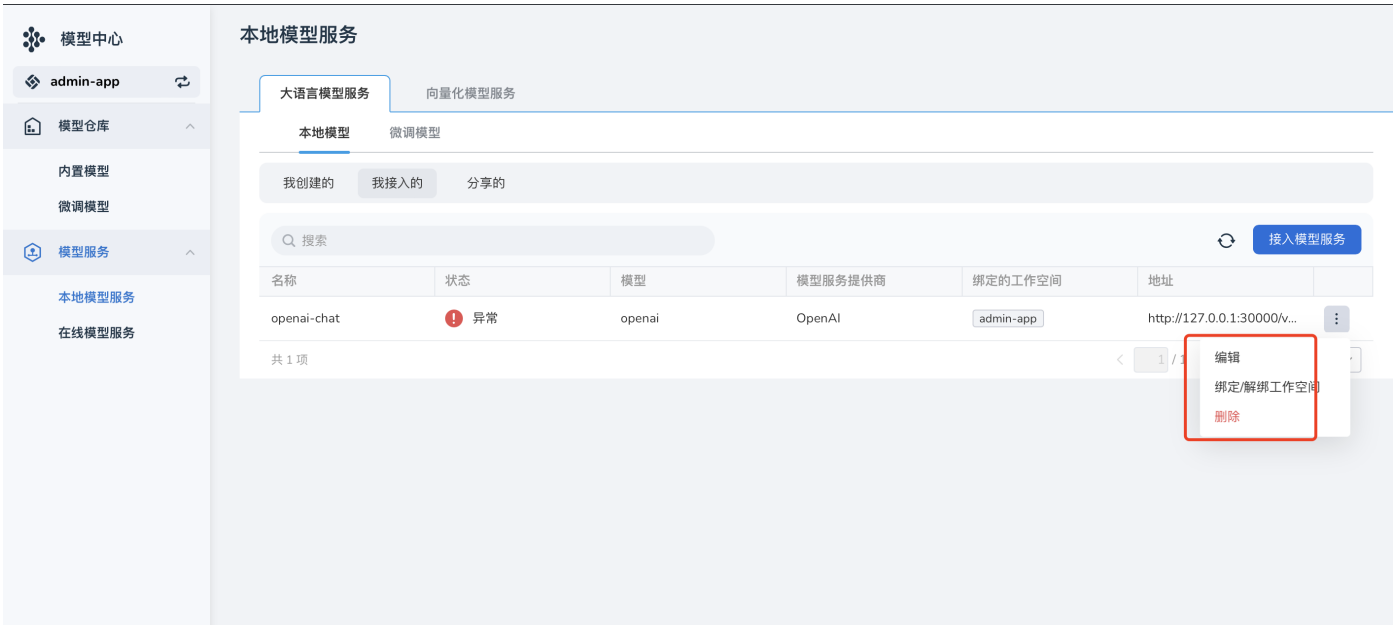


2. 填写名称，模型名称，选择模型服务提供商，输入访问地址后，点击 确定 进行连接测试（注：大语言模型访问地址需要在url后加上 **/v1/chat/completions**）。

点击 确定 后，屏幕上会有几种提示：

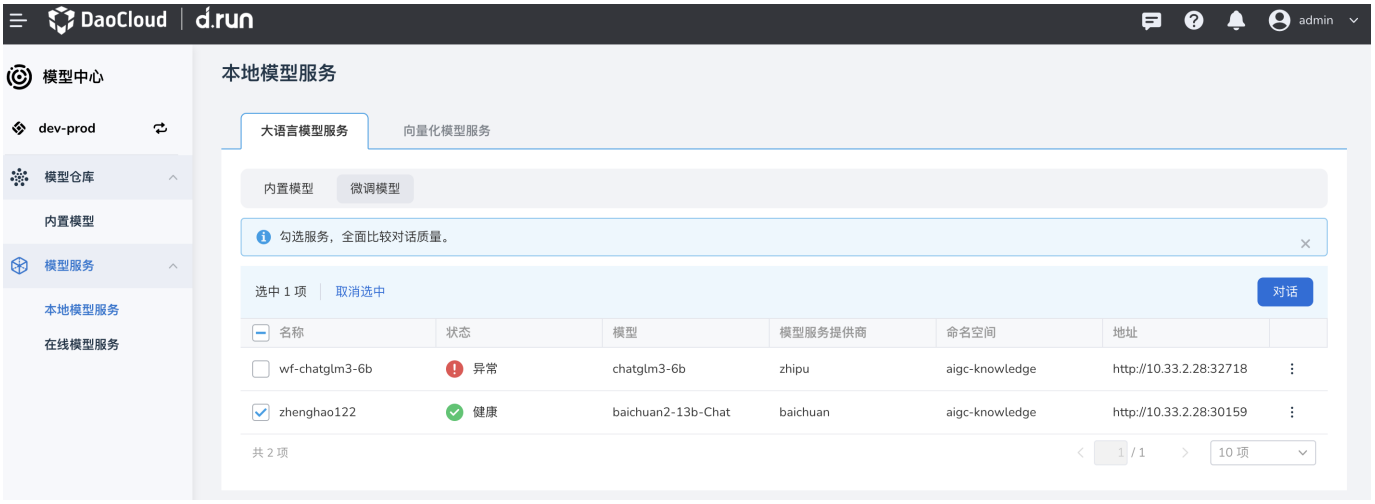
- 接入中：测试连接中，请稍候
- 接入成功：连接成功
- 接入失败：抱歉，模型服务当前不可用，请稍后重试，或联系系统管理员获取更多帮助

3. 创建成功后，在列表中，您可点击右侧的 ⋮ ，执行 编辑 、 绑定/解绑工作空间 、 删除 操作。



微调模型

对于微调模型，勾选模型服务后，点击右侧 对比对话 即可与该模型 对话(注：最多选中三个微调模型进行对话)。



向量化模型服务

向量化模型：此类模型将数据表示成计算机可识别的实数向量（vector），根据粒度大小不同可将数据特征表示分为字、词、句子或篇章几个层次。 数据向量化的方法主要分为离散表示和分布式表示。也就是说，数据向量化是将原始数据转换为数值向量的过程，以便计算机可以理解和处理数据。

此类服务不用区分内置模型或微调模型，接入服务过程与内置模型的接入步骤相同（注：向量化模型直接填写正确的访问地址即可使用）

如果要删除某个服务，点击列表右侧的 ⋮ ，在弹出菜单中选择 删除 ，在弹窗中输入服务的名称，确认无误后点击 删除 。

Caution

删除之前，请确认没有应用使用该模型服务。 此操作不可逆，请谨慎操作。

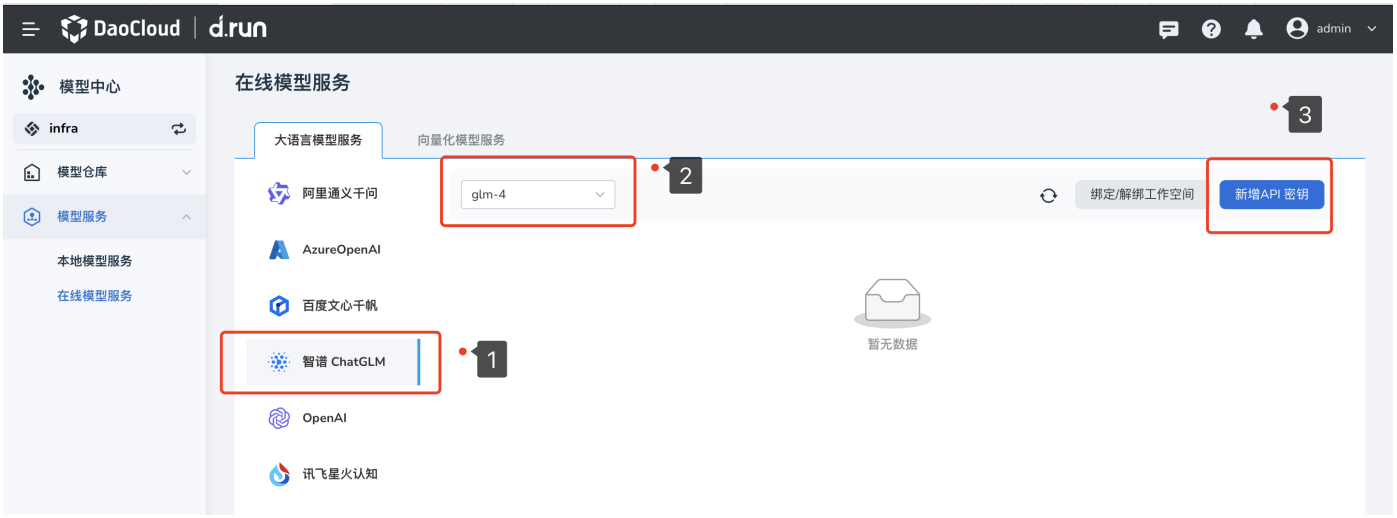
在线模型服务

该模型可通过 API 密钥的形式，向公众提供在线服务。 获取到 API 密钥的用户，即可使用模型提供的服务。

大语言模型服务

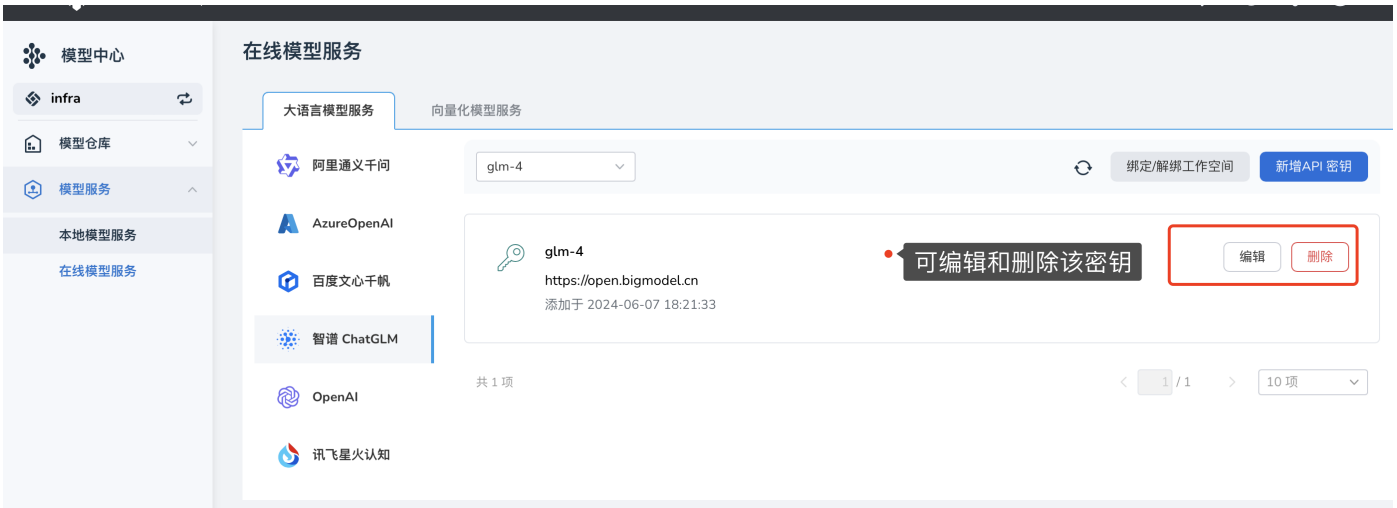
D.run平台暂时支持接入的模型服务有：阿里通义千问、AzureOpenAI、百度文心千帆、智谱ChatGLM、OpenAI、讯飞星火认知六种公有云的服务。 如需新增 API 密钥（此处以智谱ChatGLM服务为例），步骤如下所示：

1. 选择 智谱**ChatGLM** 的模型服务，中部下拉框中切换至 **glm-4** 点击右侧的 新增 **API** 密钥 按钮



2. 填写密钥名称、APIkey后点击 确定 。(注⚠：模型服务地址 非必填，填写错误会连接失败，不填写系统有默认的URL，故不建议用户自行填写)

3. 提示连接成功后，您可以执行 编辑 、 删除 等更多操作。



向量化模型服务（正在开发中，敬请期待）

……

1.5 管理

1.5.1 全局管理

介绍

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、工作空间与层级、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。
- 工作空间与层级：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 d.run 平台管理员（Admin）或具有管理员权限的用户登录 d.run 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

注册并体验 **d.run**

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。

d.run 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。

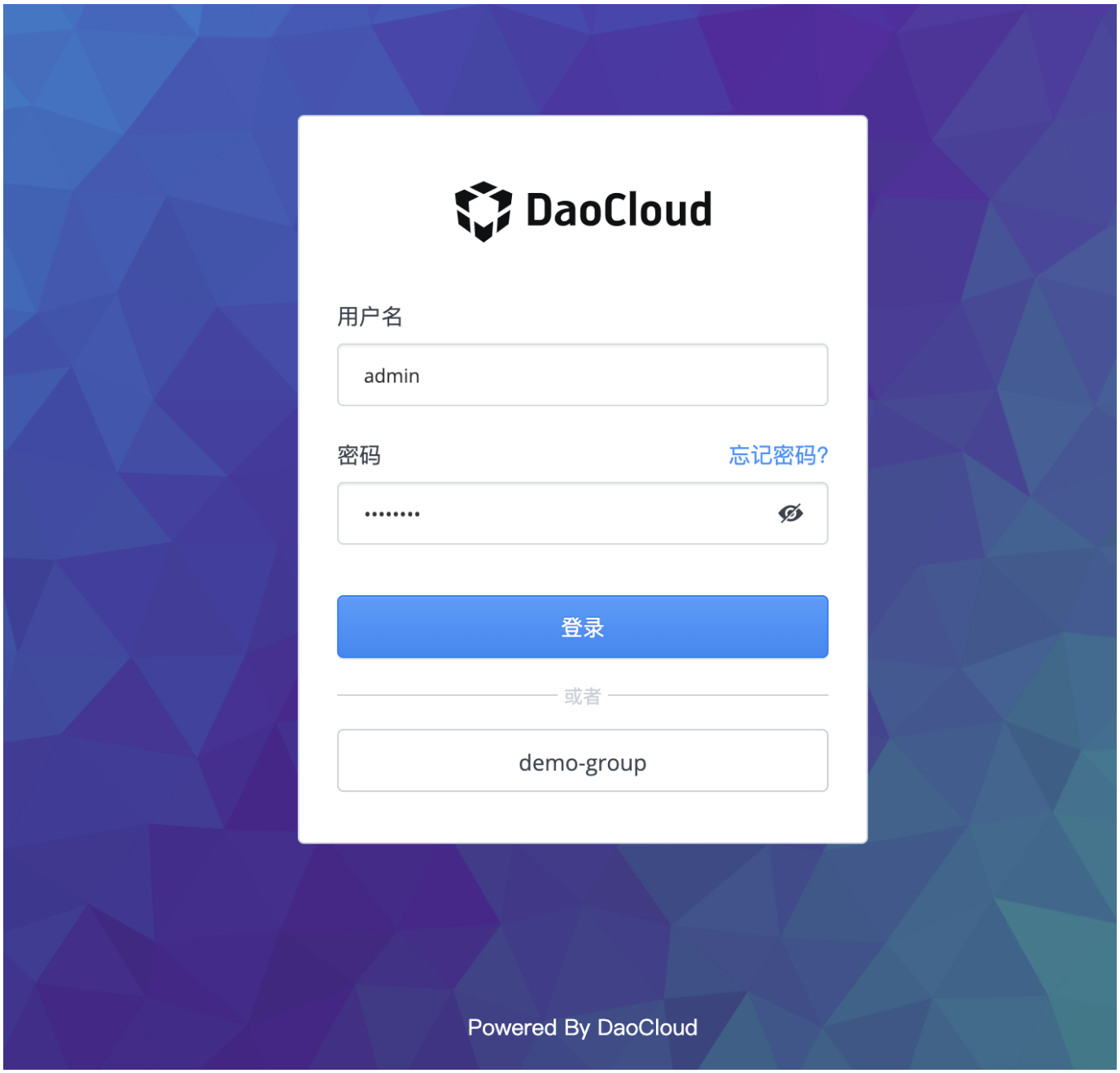
用户通过账号/密码进行登录，系统验证身份是否合法，如果验证合法，则用户成功登录。

> **Note**
>
> 如果用户登录后 24 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。


DaoCloud
用户名
admin
密码
忘记密码?
登录
或者
demo-group
Powered By DaoCloud

功能特性

全局管理的功能特性参见下表：

主要功能	细分项
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	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
	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
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

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 Owner 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 d.run，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 Owner：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，d.run 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。

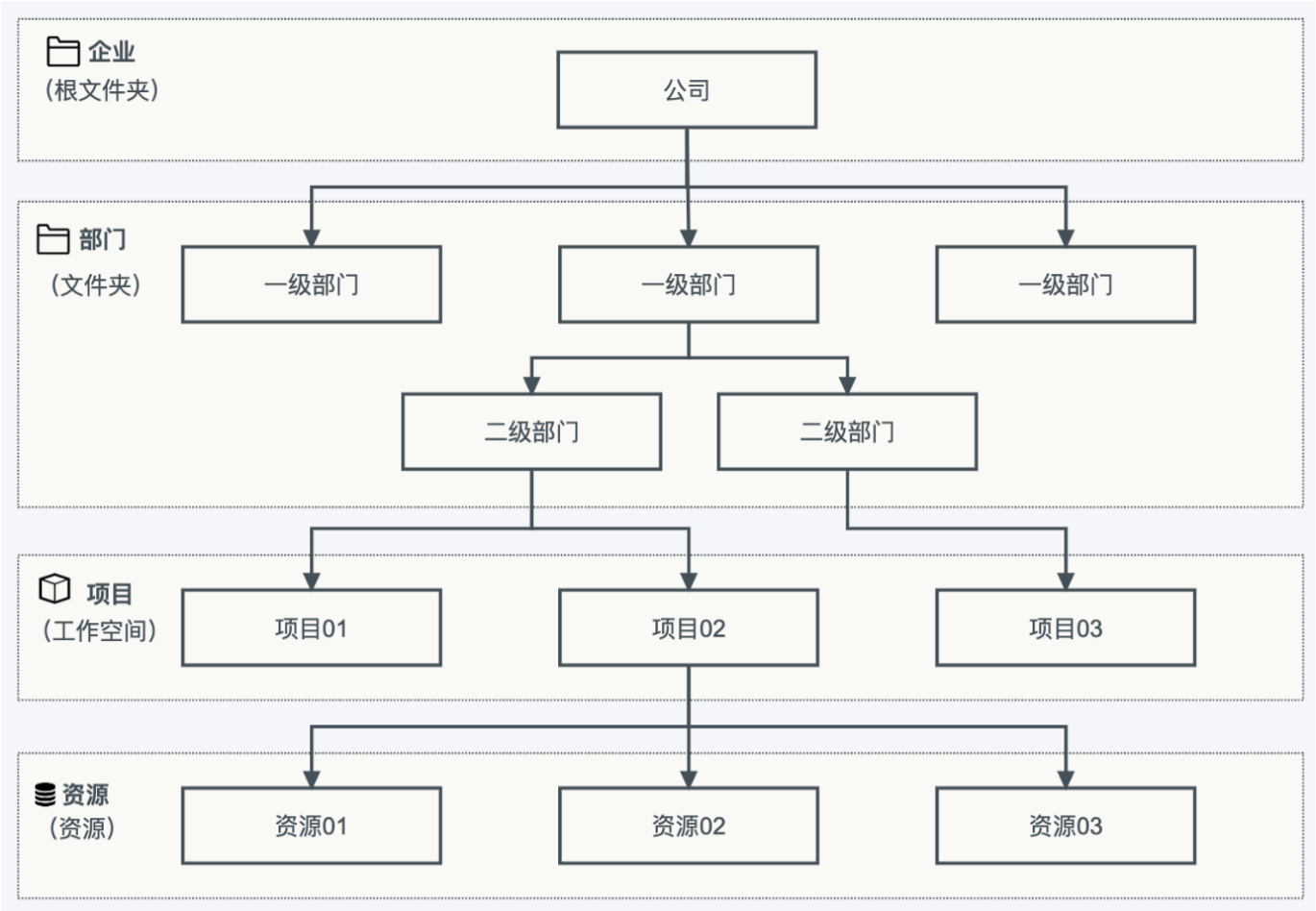
默认情况下，管理员创建的 IAM 用户没有任何角色权限，需要对其单独授予角色或将其加入用户组并给用户组授予角色，才能使得用户获得对应的角色权限，这一过程称为授权。 授权后，用户就可以基于被授予的角色权限对平台资源进行操作。

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。



层级

参见上图，为了满足企业内各个部门的分支划分，d.run 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 d.run 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 Admin。 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 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

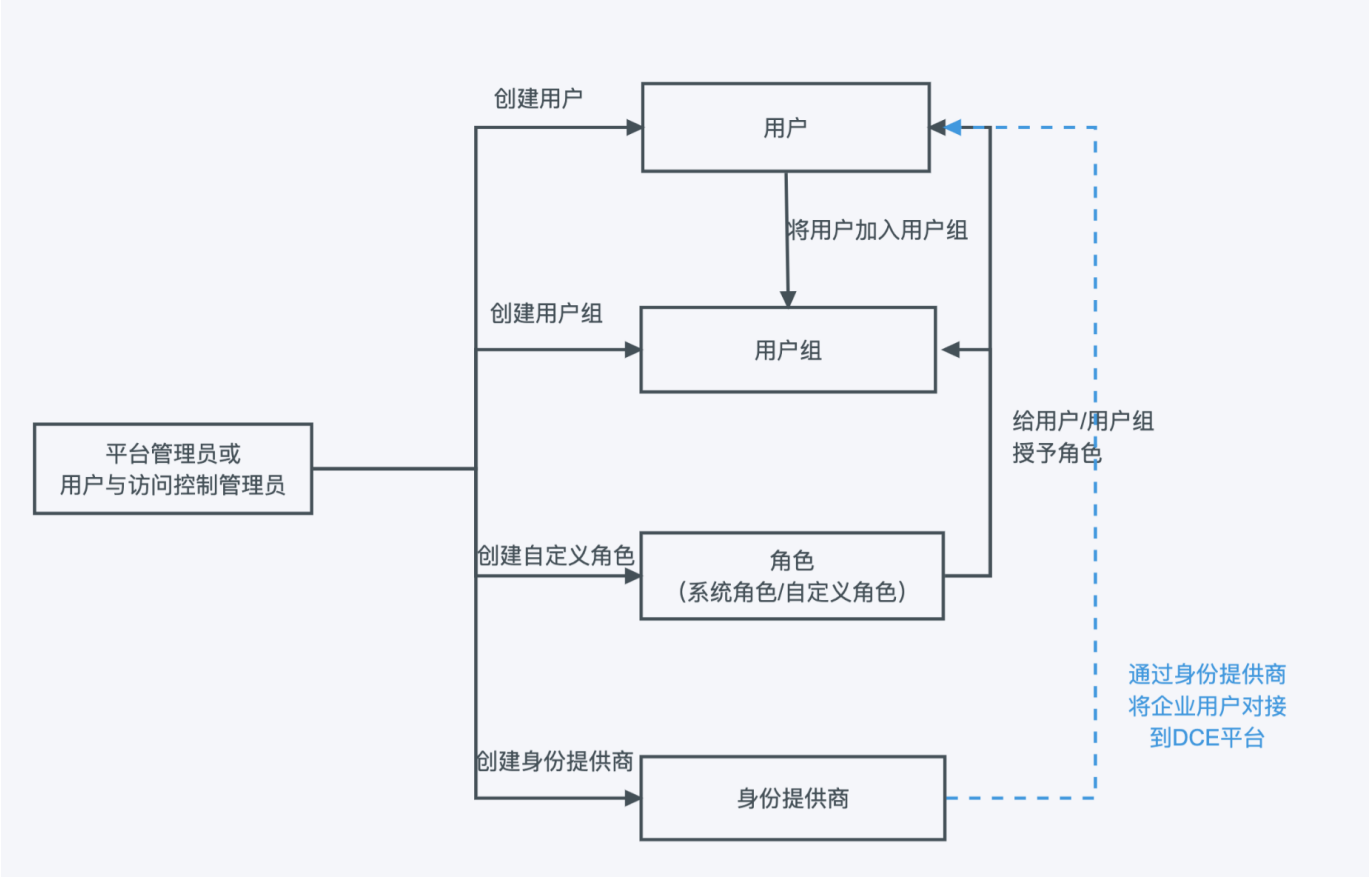
身份凭证是识别用户身份的依据，登录和使用资源时，需要有身份凭证才能通过系统的鉴权认证。 身份凭证包括密码和访问密钥，可以通过 IAM 管理自己以及下属 IAM 用户的身份凭证。

有关更多术语，请参阅本站收录的云原生词汇大全。

用户与访问控制

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 d.run 平台的权限。



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 d.run 平台，而不需要在 d.run 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 d.run 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 d.run 平台，实现单点登录。

使用流程

1. 使用 d.run 平台主账号（ Admin）或具有管理员权限的用户账号登录 d.run 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。

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 Owner 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。

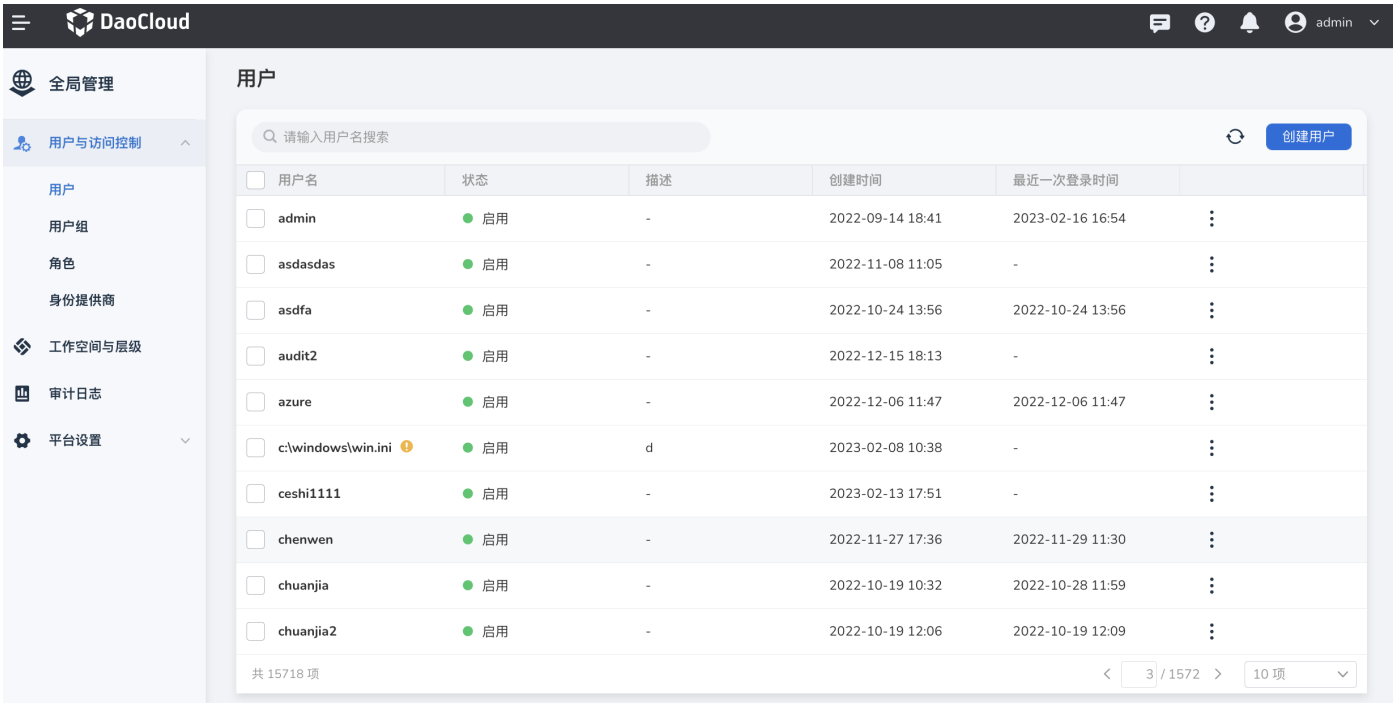
拥有一个用户账号是用户访问平台的前提。新建的用户默认没有任何权限，例如您需要给用户赋予相应的角色权限，比如在 用户列表 或 用户详情 授予子模块的管理员权限。 子模块管理员拥有该子模块的最高权限，能够创建、管理、删除该模块的所有资源。 如果用户需要被授予具体资源的权限，比如某个资源的使用权限，请查看资源授权说明。

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 Owner 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 。



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。

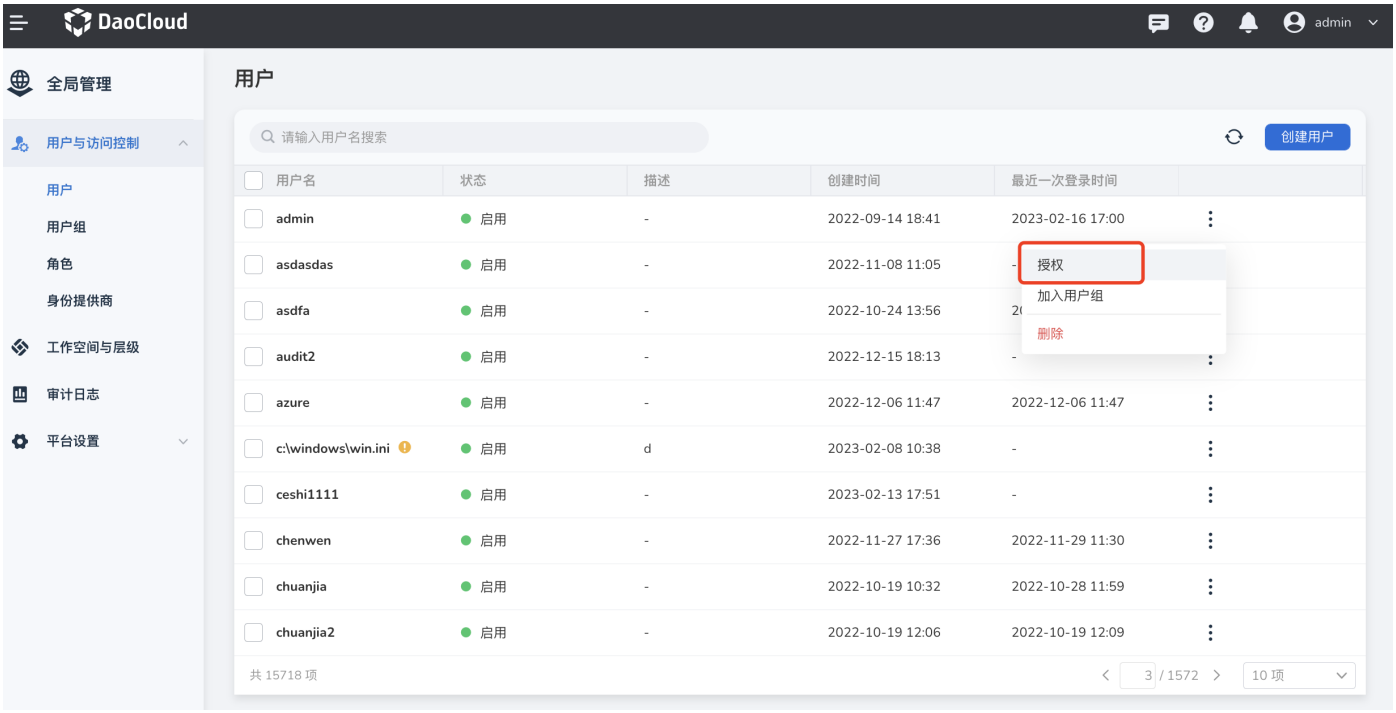
3. 点击 确定 ，创建用户成功，返回用户列表页。

> **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 。



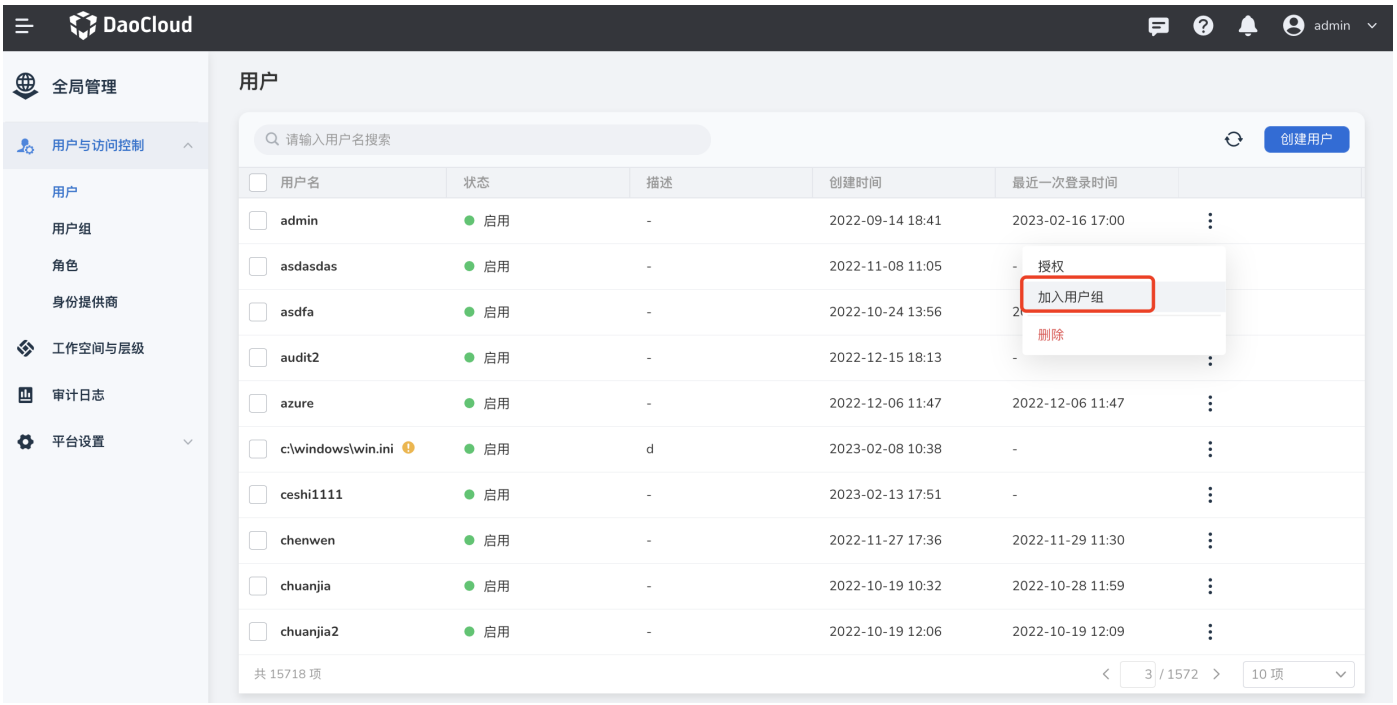
2. 在 授权 页面勾选需要的角色权限（可多选）。

3. 点击 确定 完成为用户的授权。

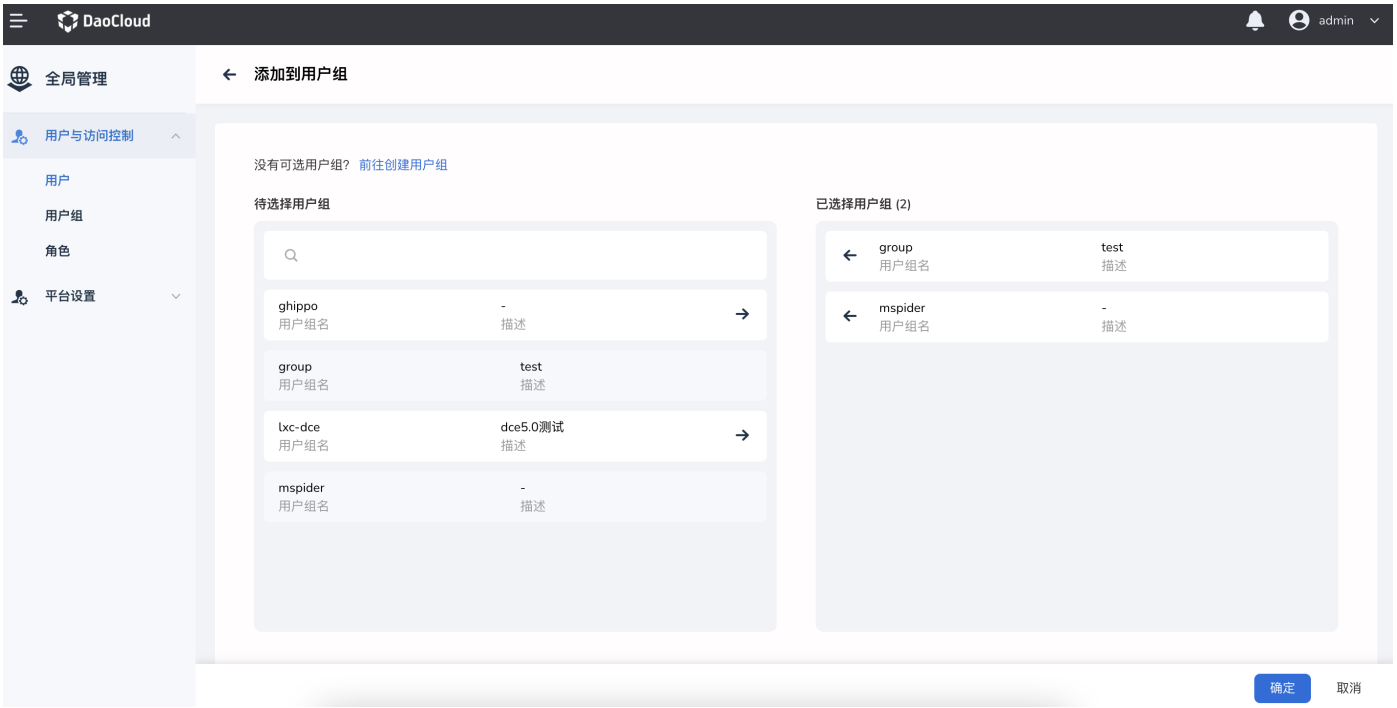
> **Note**
>
> 在用户列表中，点击某个用户，可以进入用户详情页面。

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 。



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。



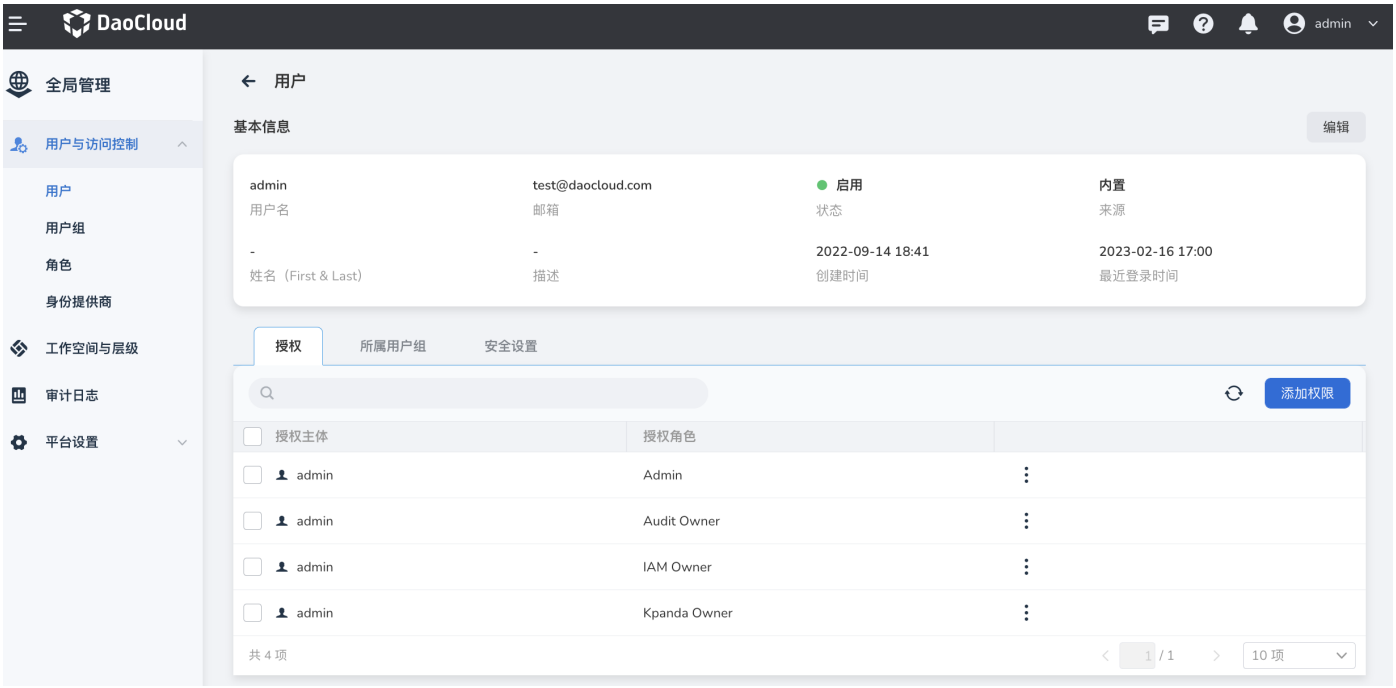
3. 点击 确定 将用户加入用户组。

> **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。



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。

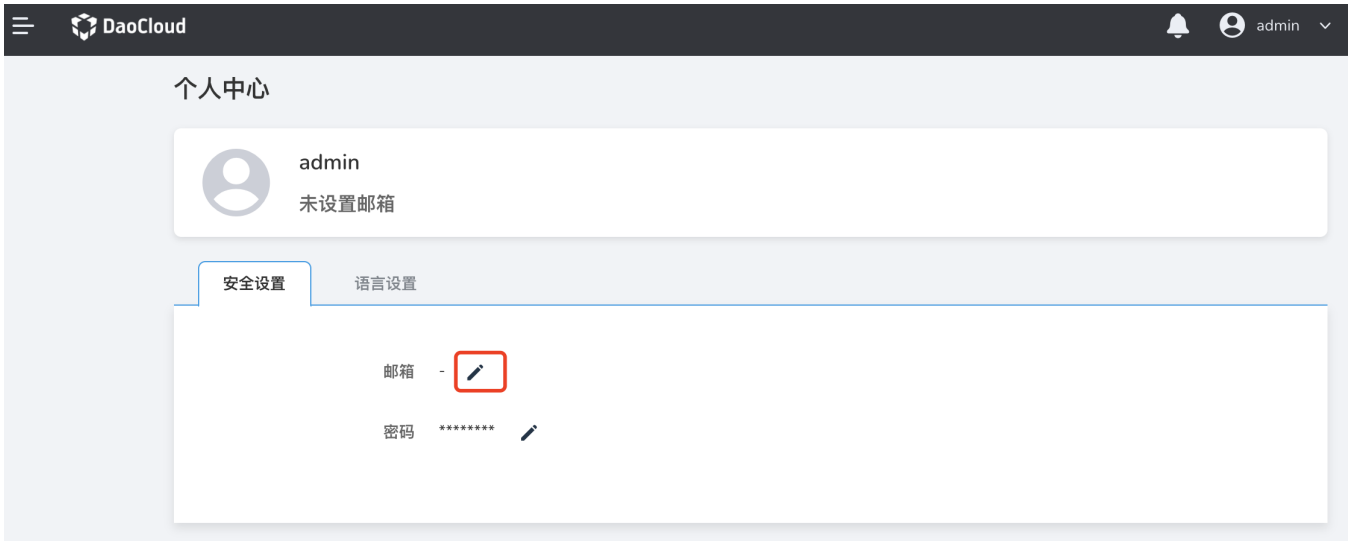
3. 点击 确定 完成禁用用户的操作。

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。

- 用户还可以进入 个人中心 ，在 安全设置 页面设置邮箱地址。



如果用户登录时忘记密码，请参考重置密码。