

大模型服务平台 使用手册

DaoCloud 研发中心

文档内容请以网站内容为准

<https://docs.daocloud.io/hydra/>

如有更改，素不另行通知

目录

1	什么是大模型服务平台	4
1.1	普通视图和运维管理使用逻辑	6
1.2	审计功能	7
2	部署大模型服务平台（WS 模式）	8
2.1	全局服务集群	9
2.2	工作集群	9
2.3	工作集群初始化	14
2.4	工作集群模型预热	15
2.5	体验模型	16
2.6	模型部署	16
3	离线升级大模型服务平台	19
3.1	解压下载的包得到 bundle 包	20
3.2	从安装包中加载镜像	20
3.3	升级	22
4	用户视图 - 模型广场	23
4.1	筛选模型	23
4.2	模型详情	24
4.3	体验模型	25
4.4	部署模型	25
5	用户视图 - 模型体验	25
5.1	体验入口	26
5.2	体验说明	26
5.3	模型类型	27
5.4	对话的更多功能	28
5.5	模型对比	28
5.6	模型切换	29
5.7	模型参数设置	29
6	用户视图 - 模型部署	31
6.1	基本信息	32
6.2	鉴权方式	32
6.3	调用 API 示例	33
6.4	扩缩容	36
6.5	删除	37
7	用户视图 - 部署新模型	37
8	用户视图 - 使用 Dataset 管理模型文件	39
8.1	自动下载	40
8.2	手动下载	43
8.3	Dataset Spec Reference	45
9	用户视图 - API Key 管理	47
9.1	功能说明	47
9.2	创建 API Key	48
9.3	查看 API Key	48
9.4	删除 API Key	49
9.5	使用 API Key 调用服务	49
10	运维管理 - 模型广场管理	50
10.1	批量导入模型	50
10.2	创建模型	52
10.3	编辑模型	53

10.4	模型上线/下线.....	55
11	运维管理 - MaaS 模型管理.....	56
11.1	创建 MaaS 模型.....	56
11.2	MaaS 模型管理列表.....	58
12	运维管理 - 模型部署管理.....	59
12.1	模型部署列表.....	59
12.2	创建模型.....	60
12.3	查看模型详情.....	61
13	大模型服务平台使用示例场景.....	61
14	在 VSCode 和 Cline 中使用 Hydra.....	62
14.1	安装 Cline.....	62
14.2	配置 Cline.....	63
14.3	Cline 使用演示.....	65
15	在 Cherry Studio 中使用 Hydra.....	67
15.1	安装 Cherry Studio.....	67
15.2	配置 Cherry Studio.....	67
15.3	Cherry Studio 使用演示.....	69
16	在 Bob Translate 中使用 Hydra.....	69
16.1	安装 Bob Translate.....	70
16.2	配置 Bob Translate.....	70
16.3	Bob Translate 使用演示.....	72
17	在 LobeChat 中使用 Hydra.....	73
17.1	安装 Lobe Chat.....	74
17.2	配置 Lobe Chat.....	74
17.3	Lobe Chat 使用演示.....	75
18	自定义大模型推理运行时.....	76
18.1	全局服务集群中添加自定义运行时.....	76
18.2	工作集群中添加镜像信息.....	81
18.3	开始使用.....	82

1 什么是大模型服务平台

大模型服务平台是一个专为企业级用户打造的综合性人工智能模型服务管理平台，旨在解决企业在大模型应用过程中所面临的一系列核心挑战，包括模型部署的复杂性、模型选择的困难性、运行稳定性不足以及潜在的安全风险等。通过提供从模型部署到运维管理的全生命周期服务，该平台能够帮助企业和开发者高效地接入和使用各类大模型能力，从而加速企业数字化转型和智能化升级的进程。

功能特性

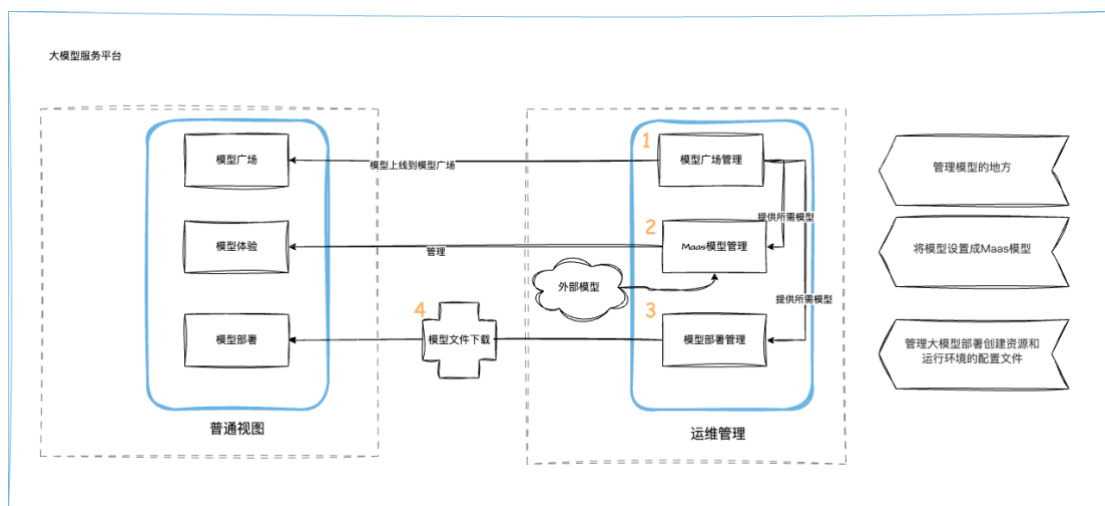
- 一键部署与简化运维
 - 图形化界面与 API 双支持：提供直观的 Web 界面和完整的 API 接口
 - 模型一键部署：支持主流大模型分钟级快速上线
 - 动态推理后端：支持 vLLM、SGLang 等多种推理引擎
 - 实时扩缩容：根据业务需求灵活调整实例数量
 - 多地域部署：支持按需选择部署地域，就近服务
- 流量治理与稳定性保障
 - 智能流量策略引擎：基于权重、QPS 限制等多维度流量控制
 - 多层限流机制：
 - 全局限流：控制整体平台负载

- API Key 限流：精细化管理不同应用访问频次
- 租户级限流：企业级用户独立限流保障
- 分布式推理能力
 - 多机多卡部署：支持 DeepSeek、GLM 等超大参数模型
 - 异构 GPU 支持：兼容 NVIDIA、壁仞、沐曦、昇腾等多种 GPU
 - 负载均衡策略：
 - 轮询策略：流量均匀分配
 - 随机策略：快速分散请求
 - 权重策略：基于权重分配策略
- 精准计费与统计
 - Token 精确计量：支持主流大模型的计费逻辑
 - 多维度统计：
 - 调用总量、输入/输出 Token 统计
 - 按 API Key、模型类型、时间维度筛选
- 多模态统一管理
 - 模型广场：提供文本、图像等各类模型的展示与介绍

- 模型对比体验：一次输入，多模型同步响应对比
- API 调用示例：提供丰富的 Demo 和接入文档

1.1 普通视图和运维管理使用逻辑

本平台通过“运维管理视图 + 普通用户视图”协同配合，实现模型的统一管理、试用体验和快速部署。如下图所示，模型的使用流程主要分为三个步骤 + 模型文件下载



1 模型广场管理：添加并上线模型

- 运维管理：
 - 在 **模型广场管理** 中导入或创建模型。
 - 模型上线后，自动同步到用户视图。
- 普通用户视图：
 - 在 **模型广场** 中可浏览已上线的模型，作为后续体验或部署的入口。

2 Maas 模型管理：配置模型试用服务

- **运维管理:**
 - 将模型设置为 **MaaS 模型**，配置其试用运行环境。
- **普通用户视图:**
 - 在 **模型体验** 中选择配置好的模型进行在线试用，无需本地部署。

3 模型部署管理：配置模型部署参数

- **运维管理:**
 - 在 **模型部署管理** 中，为模型创建资源和环境的部署配置文件，如 GPU/内存/运行框架等。
- **普通用户视图:**
 - 在 **模型部署** 页面中，选择模型快速完成部署。
 - 简化操作流程，无需手动填写复杂参数。

4 模型文件下载：支持部署的底层基础

- 需由运维人员将模型文件下载到指定位置

1.2 审计功能

审计功能是大模型服务模块的核心组成部分，旨在提供全面的操作追踪与安全监控，以满足合规性、可追溯性和安全审计的需求。

审计功能通过记录关键操作事件，确保用户行为和系统变更可追溯。主要审计项包括：

- **模型管理:** 如创建、更新、删除模型（Model）、启用/禁用 MAAS 模型体验功能（EnabledMAASModel）、发布/下线模型（PublishModel/UnPublishModel）等。
- **部署模板管理:** 如添加、更新模型部署模板（AddDeployTemplates/UpdateDeployTemplates）。
- **模型提供商管理:** 如创建、更新、删除模型提供商（ModelProvider）。
- **工作空间 API 密钥管理:** 如创建、删除工作空间 API 密钥（CreateWSAPIKey/DeleteWSAPIKey）。
- **模型服务管理:** 如创建、删除工作空间模型服务（CreateWSModelServing/DeleteWSModelServing）、更新副本数（UpdateWSModelServingReplicas）、执行服务操作（DoModelWSServingAction）。

这些审计项涵盖了模型生命周期的各个方面，确保每个操作都有迹可循。审计功能是保障大模型服务平台安全性、合规性和高效运营的关键工具。

有关审计项列表参见[大模型服务平台审计项](#)。

2 部署大模型服务平台（WS 模式）

本文说明如何部署私有化 WS（Workspace）模式的大模型服务平台 Hydra，支持无卡进行推理。

2.1 全局服务集群

在[全局服务集群](#)中，需要安装：

- Hydra：手动安装或通过安装器安装
- [服务网格](#)（依赖 Istio 创建网关进行路由）
- [MySQL](#)
- [Redis](#)

Tip

- 全局服务集群默认安装了通用的 MySQL 和 Redis。
- 如果通过安装器安装 Hydra，则默认使用通用的 MySQL 和 Redis。
- 如果手动安装 Hydra，需要在安装参数中指定 MySQL 和 Redis。

安装好的效果如下图：



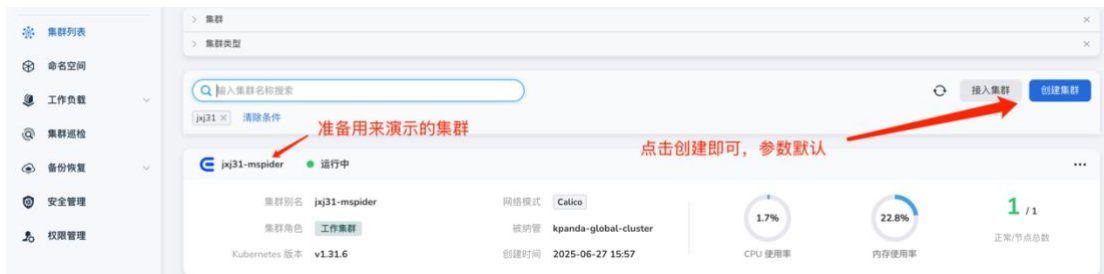
应用名称	状态	命名空间	Chart	仓库	更新时间	安装时间
mspider	已部署	mspider-system	mspider-v0.36-dev-765eaf2d	mspider-ci	2025-07-08 11:47	2025-07-08 11:47
hydra	已部署	hydra-system	hydrav0.7.0-rc1	-	2025-07-04 16:32	2025-07-04 16:32
amamba	已部署	amamba-system	amamba.0.37.0	-	2025-07-04 16:30	2025-07-04 16:30

2.2 工作集群

工作集群需要安装 Hydra Agent 和 metallb。

2.2.1 安装 Hydra Agent

1. 创建工作集群，用来部署 Hydra Agent（资源紧张时，也可以直接使用全局服务集群）



2. 在工作集群部署 Hydra Agent

需要注意以下参数：

global:

config:

cluster_name: 'jxj31-mspider'

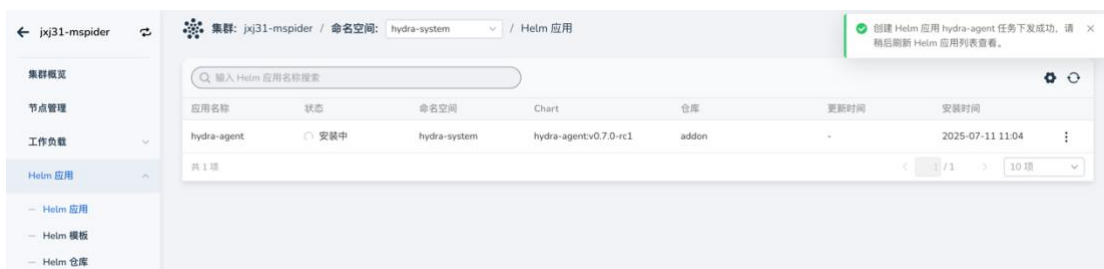
agent_base_url: 'http://cn-sh-a1' # 需要工作集群可访问的网关地址

agent_server:

address: example.com:443 # 全局服务集群 dce 地址

plaintext: false

insecure: true # 如果是测试环境，可以将其内置为 true，不走证书



3. 在工作集群部署 metallb（用于访问模型的路由），分配 LB



2.2.2 创建服务网格（Istio + Gateway API）

1. 在工作集群创建专有网格（创建网格时，参数全部默认即可）

注意：托管网格当前不支持 gateway api。



2. 初始化 Gateway API CRD

```
root@controller-node-1:~# kubectl kustomize "github.com/kubernetes-sigs/gateway-api/config/crd?ref=v1.2.1" | kubectl apply -f -;
```

```
customresourcedefinition.apiextensions.k8s.io/gatewayclasses.gateway.networking.k8s.io
created
customresourcedefinition.apiextensions.k8s.io/gateways.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/grpccroutess.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/httpproutess.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/referencegrants.gateway.networking.k8s.io
created
```

3. 创建网关，路由规则

```
root@controller-node-1:~# cat gateway.yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: gateway
  namespace: default
spec:
  gatewayClassName: istio
```

```

listeners:
- allowedRoutes:
  namespaces:
    from: All
  name: default
  port: 80
  protocol: HTTP
- allowedRoutes:
  namespaces:
    from: All
  hostname: 'cn-sh-a1'
  name: https
  port: 443
  protocol: HTTPS
  tls:
    certificateRefs:
      - group: ""
        kind: Secret
        name: cn-sh-a1
    mode: Terminate
root@controller-node-1:~# kubectl apply -f gateway.yaml

```

gateway.gateway.networking.k8s.io/gateway created

```
root@controller-node-1:~# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
gateway-istio-5c497d4b6d-9xmqp	1/1	Running	0	14s

```
root@controller-node-1:~# kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
gateway-istio	LoadBalancer	10.233.1.140	10.64.24.211	15021:32565/TCP,80:32053/TCP,443:32137/TCP	45s

```
root@controller-node-1:~# cat httproute.yaml
```

apiVersion: gateway.networking.k8s.io/v1

kind: HTTPRoute

metadata:

labels:

app.kubernetes.io/managed-by: Helm

name: hydra-agent-knoway

namespace: hydra-system

spec:

parentRefs:

- group: gateway.networking.k8s.io

kind: Gateway

```

name: gateway
namespace: default
rules:
- backendRefs:
  - group: ""
    kind: Service
    name: knoway-gateway
    port: 8080
    weight: 1
filters:
- responseHeaderModifier:
  add:
  - name: Access-Control-Allow-Headers
    value: '*'
  - name: Access-Control-Allow-Methods
    value: '*'
  type: ResponseHeaderModifier
matches:
- path:
  type: PathPrefix
  value: /v1
root@controller-node-1:~# kubectl apply -f httproue.yaml

```

httproue.gateway.networking.k8s.io/hydra-agent-knoway created

4. 配置域名解析，域名映射到 Ingress 网关的 LB，工作集群和

浏览器的电脑 /etc/hosts 追加域名映射：

```
echo "10.64.xx.xx cn-sh-a1" | sudo tee -a /etc/hosts
```

设置本地电脑信任证书：



2.3 工作集群初始化

操作数据库插入厂商数据。对于新版本的 Hydra，可以不用操作，因为后续 Hydra 可以自动创建厂商。

实现（只有批量上传支持）时使用 mcamel-system 的 mcamel-common-mysql-cluster-mysql 工作负载。



示例：

```
{"enUS": "Alibaba", "zhCN": "通义千问"}
```

2.3.1 Hydra 运维平台上架模型

配置模型部署参数（根据实际情况调整模型部署参数）



2.3.2 安装 nfs drive

下载模型实体时需要用到 nfs drive。

```
wget http://example.com/nfs-install.tar # 改成你的下载地址
```

```
tar xvf nfs-install.tar
./install.sh
```

2.4 工作集群模型预热

模型预热指的是提前下载模型镜像。

在 Hydra 和 AI Lab 同时存在时，会同时存在两个 dataset CRD。 注意

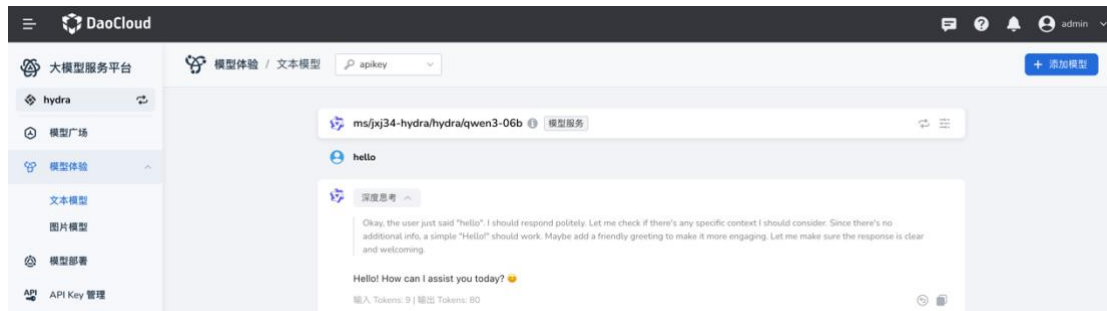
需要使用 `dataset.baizeai.io`

```
root@controller-node-1:~# cat dataset-qwen3-06b.yaml
apiVersion: dataset.baizeai.io/v1alpha1 # 这里应该是 AI Lab 组
kind: Dataset
metadata:
  name: qwen3-0.6b
  namespace: public # 必须 public 命名空间
  labels:
    hydra.io/model-id: qwen3-0.6b # 需要与模型名称一致
spec:
  dataSyncRound: 1
  share: true
  source:
    type: HTTP
    uri: http://example.com:81/model/qwen3-06b/ # 需要改成你的地址
    options:
      repoType: MODEL
  mountOptions:
    uid: 1000
    gid: 1000
    mode: "0774"
    path: /
  volumeClaimTemplate:
    spec:
      storageClassName: nfs-csi
root@jxj:~/hydra-deploy# kubectl create ns public

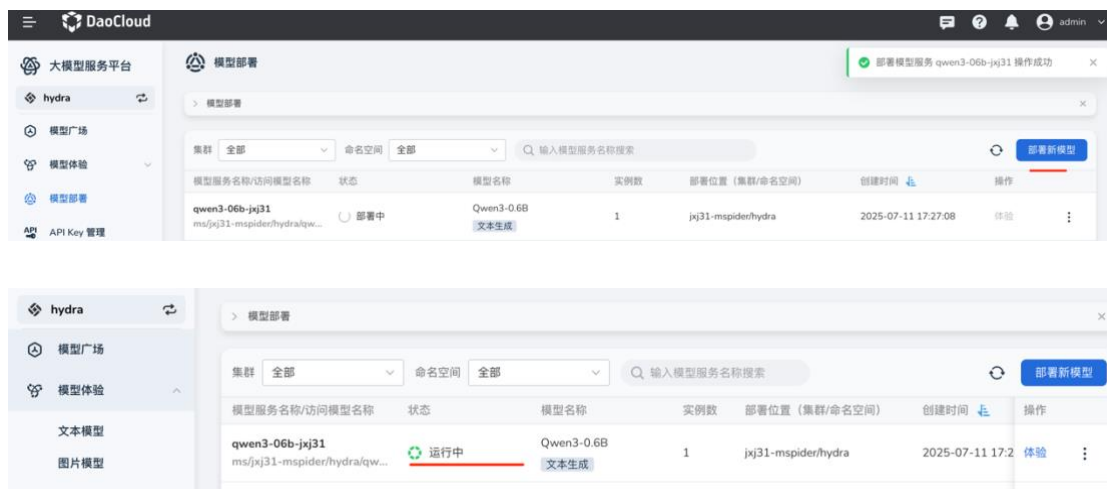
namespace/public created
root@jxj:~/hydra-deploy# kubectl apply -f dataset-qwen3-06b.yaml

dataset.dataset.baizeai.io/qwen3-0.6b created
```

2.5 体验模型

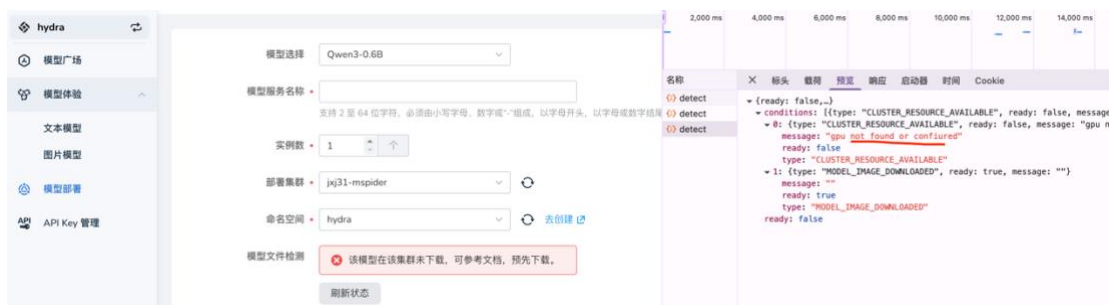


2.6 模型部署



如果是无卡的情况，需要注意：

1. 无 GPU 进行模型部署（无卡情况，部署模型会检测到没有 GPU，无法部署）



所以部署 `gpu-operator-fake`（拉取外网镜像需要设置代理）：

- a. 配置 Node
- b. `kubectll label node <node-name> run.ai/simulated-gpu-node-pool=default`
- c. 安装 `fake-gpu-operator`

对于离线安装，可以先下载离线包一键部署：

```

root@controller-node-1:~/fakegpu-install# ls
device-plugin-0.0.63.tar      install.sh      status-exporter-0.0.63.tar  topology-server-0.0.63.tar  ubuntu-24.04.tar
fake-gpu-operator-0.0.63.tgz  kwok-gpu-device-plugin-0.0.63.tar  status-updater-0.0.63.tar  ubuntu-2204.tar
root@controller-node-1:~/fakegpu-install# ./install.sh
正在导入镜像：device-plugin-0.0.63.tar 到命名空间：k8s.io      fake-gpu 离线环境一键部署
ghcr.io/run-ai/fake-gpu-operator/device-plugin saved
application/vnd.oci.image.manifest.v1+json sha256:ecb9ed50880e6fc45a34a887684e430127389db20dc6618bc6cfb7f6e7dcb97
Importing      elapsed: 1.8 s total:  0.0 B (0.0 B/s)
成功导入：device-plugin-0.0.63.tar
正在导入镜像：kwok-gpu-device-plugin-0.0.63.tar 到命名空间：k8s.io
ghcr.io/run-ai/fake-gpu-operator/kwok-gpu saved
application/vnd.oci.image.manifest.v1+json sha256:8520516f30d91cca810df6aa8caedc9c2eb07a13bf89e9cf6b4c1b836a73b480
Importing      elapsed: 1.3 s total:  0.0 B (0.0 B/s)
成功导入：kwok-gpu-device-plugin-0.0.63.tar
正在导入镜像：status-exporter-0.0.63.tar 到命名空间：k8s.io
ghcr.io/run-ai/fake-gpu-operator/status-exporter saved
application/vnd.oci.image.manifest.v1+json sha256:9fb83b71ab9d6eb411a15c8f4664df02c0d2eebec751058d7f342842bbc3a23
Importing      elapsed: 1.3 s total:  0.0 B (0.0 B/s)
成功导入：status-exporter-0.0.63.tar
正在导入镜像：status-updater-0.0.63.tar 到命名空间：k8s.io
ghcr.io/run-ai/fake-gpu-operator/status-updater saved
application/vnd.oci.image.manifest.v1+json sha256:2a2888821b0f7db535bd3297011f1287243c23389150c17a43ed6cb6d747aa6
Importing      elapsed: 1.3 s total:  0.0 B (0.0 B/s)
成功导入：status-updater-0.0.63.tar
正在导入镜像：topology-server-0.0.63.tar 到命名空间：k8s.io
ghcr.io/run-ai/fake-gpu-operator/topology-server saved
application/vnd.oci.image.manifest.v1+json sha256:30bd15701cf270e2f71dac2e1b760319c8f635ed5be7f723c2f3e2f774a4df3b
Importing      elapsed: 1.2 s total:  0.0 B (0.0 B/s)
成功导入：topology-server-0.0.63.tar
正在导入镜像：ubuntu-2204.tar 到命名空间：k8s.io
docker.io/library/ubuntu:22.04 saved
application/vnd.oci.image.manifest.v1+json sha256:918dbfc3bef0dd020950e8dbb7ae978d05a8322fb304c33093711d41319f7dc3
Importing      elapsed: 1.0 s total:  0.0 B (0.0 B/s)
成功导入：ubuntu-2204.tar
正在导入镜像：ubuntu-24.04.tar 到命名空间：k8s.io
docker.io/library/ubuntu:24.04 saved
application/vnd.oci.image.manifest.v1+json sha256:46b6d3729155f48828b138d0932a78a13a1a0e9986a80dc25e6cf705fd728664
Importing      elapsed: 1.2 s total:  0.0 B (0.0 B/s)
成功导入：ubuntu-24.04.tar
所有镜像导入操作完成。
Release "gpu-operator" does not exist. Installing it now.
NAME: gpu-operator
LAST DEPLOYED: Fri Jul 18 15:36:40 2025
NAMESPACE: gpu-operator
STATUS: deployed
REVISION: 1

```

对于在线安装，运行以下命令：

```

helm upgrade -i gpu-operator oci://ghcr.io/run-ai/fake-gpu-operator/fake-gpu-operator --namespace gpu-operator --create-namespace --version=0.0.63

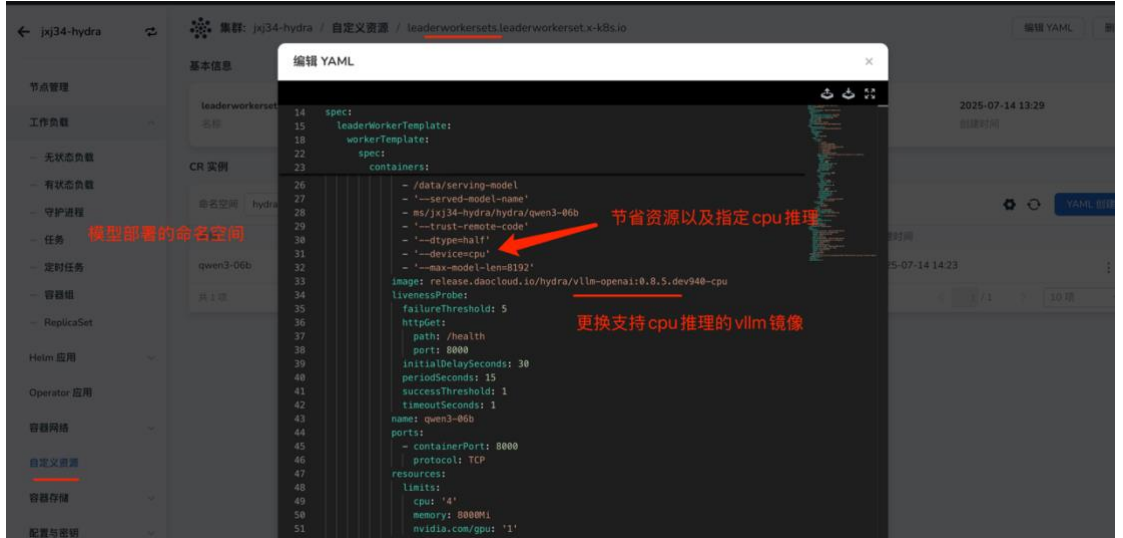
```

部署完成后，等待数分钟，查看 Node 存在 GPU 标签，刷新状态发现检测 GPU 通过。





2. 创建模型部署任务后修改部署参数：



```

--dtype=half
--device=cpu
--max-model-len=8192
release.daocloud.io/hydra/vllm-openai:0.8.5.dev940-cpu

```

3. 使用 CPU 推理时模型 Qwen3 0.6b 占用 7g 左右内存，给的 CPU 决定 token 速度



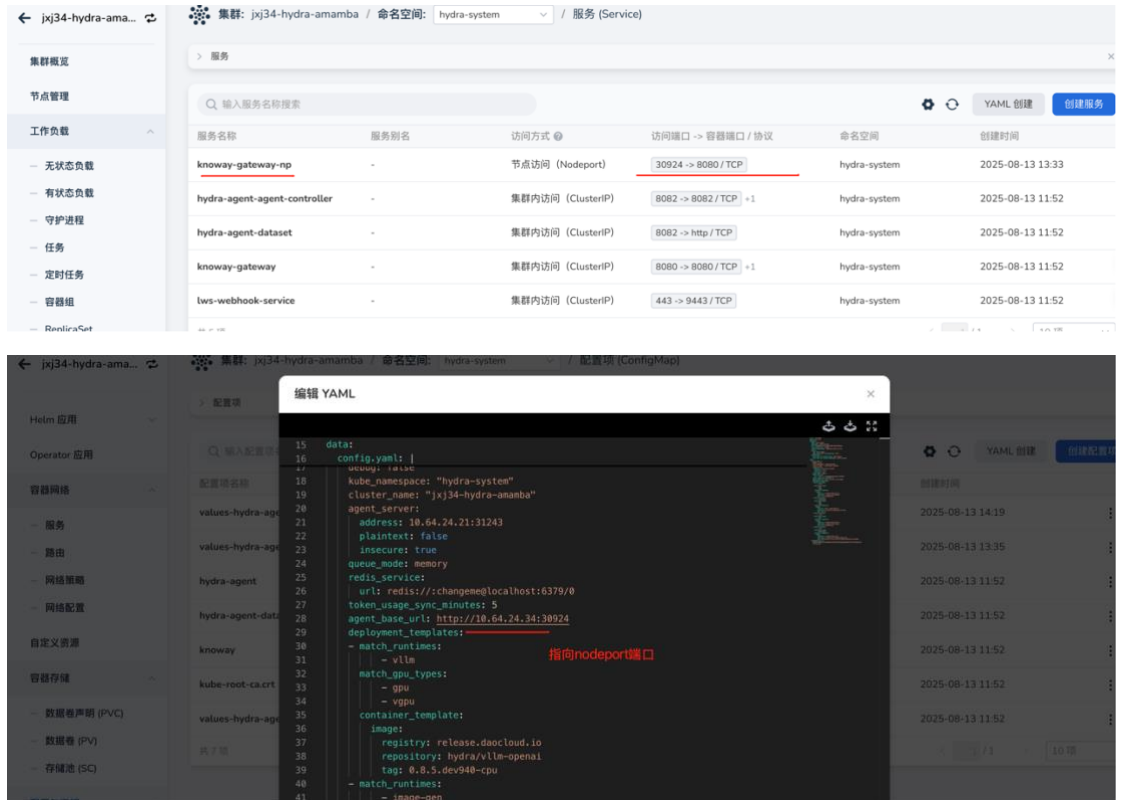
```

root@controller-node-1:~# k top po -n hydra
NAME                CPU(cores)   MEMORY(bytes)
qwen3-06b-0         17m          6589Mi

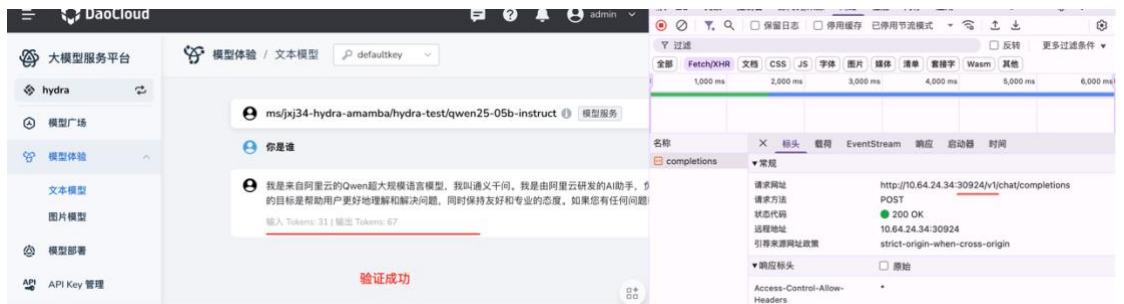
```

对于测试环境的模型流量，直接走 knoway-gateway 网关。

4. 创建网关的 NodePort 类型 svc 服务



5. 访问 DCE 时只能走 HTTP 端口



3 离线升级大模型服务平台

本页说明[下载大模型服务平台模块](#)后，应该如何安装或升级。

下述命令或脚本内出现的 **hydra** 字样是大模型服务平台模块的内部开发代号。

3.1 解压下载的包得到 bundle 包

```
tar -xvf hydra_v0.9.0_amd64.tar
```

解压后可得到 `hydra`、`hydra-agent`、`web-search-agent` 这 3 个 `bundle.tar` 包。

```
$ ll hydra_v0.9.0_amd64
-rw-r--r--@ 1 nicole  staff   72M Sep 12 12:05 hydra_v0.9.0.bundle.tar
-rw-r--r--@ 1 nicole  staff  502M Sep 12 12:05 hydra-agent_v0.9.0.bundle.tar
-rw-r--r--@ 1 nicole  staff  105M Sep 12 12:05 web-search-agent_v0.9.0.bundle.tar
```

3.2 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 **chart-syncer** 同步镜像到镜像仓库，该方法更加高效便捷。

3.2.1 chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

[已安装 chart repo](#) [未安装 chart repo](#)

若当前环境已安装 `chart repo`，`chart-syncer` 也支持将 `chart` 导出为 `tgz` 文件。

load-image.yaml

```
source:
  intermediateBundlesPath: hydra-offline
target:
  containerRegistry: 10.16.10.111
  containerRepository: release.daocloud.io/hydra
repo:
  kind: HARBOR
  url: http://10.16.10.111/chartrepo/release.daocloud.io
  auth:
    username: "admin"
    password: "Harbor12345"
containers:
  auth:
    username: "admin"
    password: "Harbor12345"
```

2. 执行同步镜像命令。
3. `charts-syncer sync --config load-image.yaml`

3.2.2 Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。
2. `tar xvf hydra.bundle.tar`

解压成功后会得到 3 个文件：

- hints.yaml
 - images.tar
 - original-chart
3. 从本地加载镜像到 Docker 或 containerd。

[Dockercontainerd](#)

```
docker load -i images.tar
```

Note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

3.3 升级

升级前注意预先备份网格的配置文件，也就是 `--set` 参数，避免升级时配置丢失导致的问题。

检查本地是否存在 `hydra-release` 仓库

```
helm repo list | grep hydra-release
```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步，直接进行更新即可。

```
Error: no repositories to show
```

添加 Helm 仓库

```
helm repo add hydra-release http://{harbor_url}/chartrepo/{project}
```

更新大模型服务平台的 `Helm` 仓库。

```
helm repo update hydra-release
```

选择您想安装的大模型服务平台版本（建议安装最新版本）。

```
# 更新 hydra-release 仓库内的镜像版本
```

```
helm update repo
```

```
# 获取最新大模型服务平台的版本
```

```
helm search repo hydra-release/hydra --versions
```

```
# 工作集群大模型服务平台组件
```

```
helm search repo hydra-release/hydra-agent --versions
```

```
# 工作集群可选支持网页搜索组件
```

```
helm search repo hydra-release/web-search-agent --versions
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
hydra-release/hydra	v0.9.0	v0.9.0	A Helm chart for Kubernetes
...			

3.3.1 备份 `--set` 参数

在升级大模型服务平台版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
helm get values hydra -n hydra-system -o yaml > bak.yaml
```

3.3.2 执行 `helm upgrade`

升级前建议您覆盖 `bak.yaml` 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
export imageRegistry={YOUR_IMAGE_REGISTRY}
helm upgrade hydra hydra-release/hydra \
  -n hydra-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry \
  --version v0.9.0
```

4 用户视图 - 模型广场

Hydra 模型广场提供多样化的模型选择，涵盖文本生成、图像生成、图片理解等领域，系统预置了通义千问、Meta、DeepSeek 等主流提供商提供的各类大语言模型。用户可根据业务需求灵活部署。

4.1 筛选模型

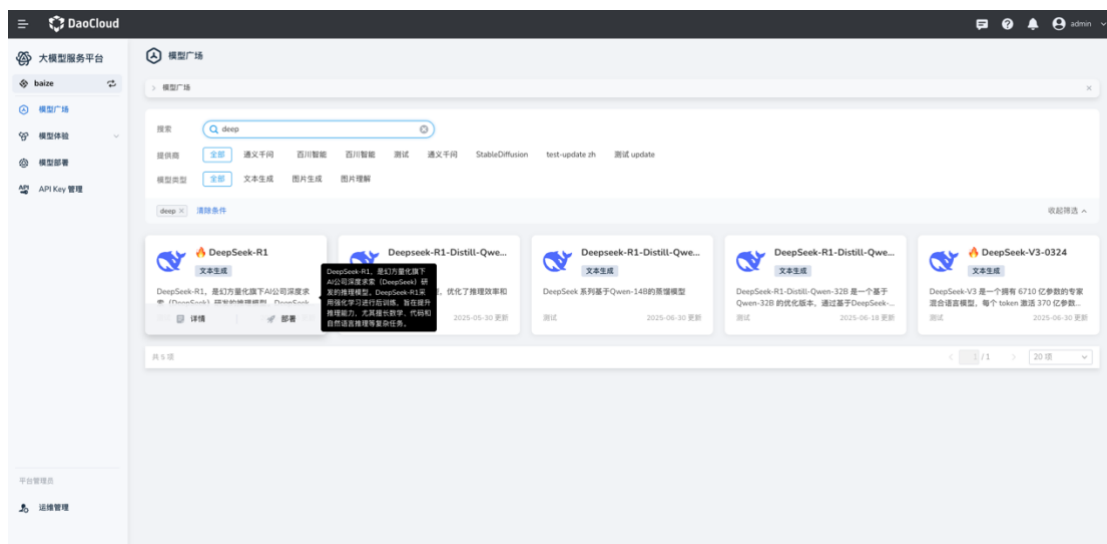
通过模型广场的标签，可以快速检索模型列表，支持标签多选，也可以随时清空筛选条件。



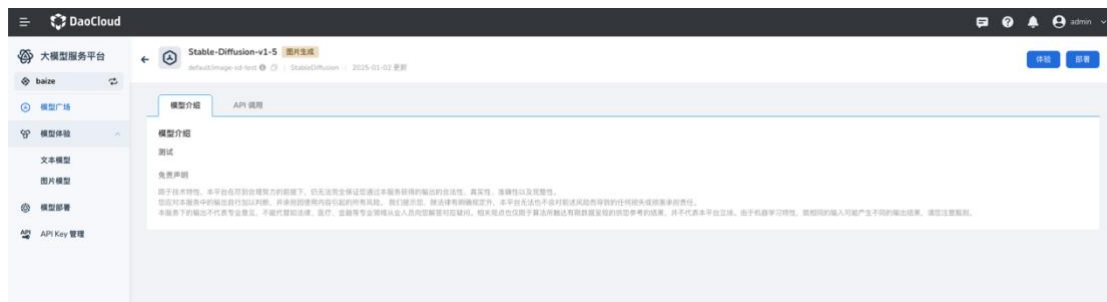
- 支持的筛选标签有：
 - 提供商：如通义千问、百川智能、GLM、Meta 等
 - 模型类型：如文本生成、图片生成、图片理解等
- 也可以输入关键词快速定位某一种模型

4.2 模型详情

光标悬浮到某个模型卡介绍字段上，即可在右侧黑色框中显示完整模型的概况。

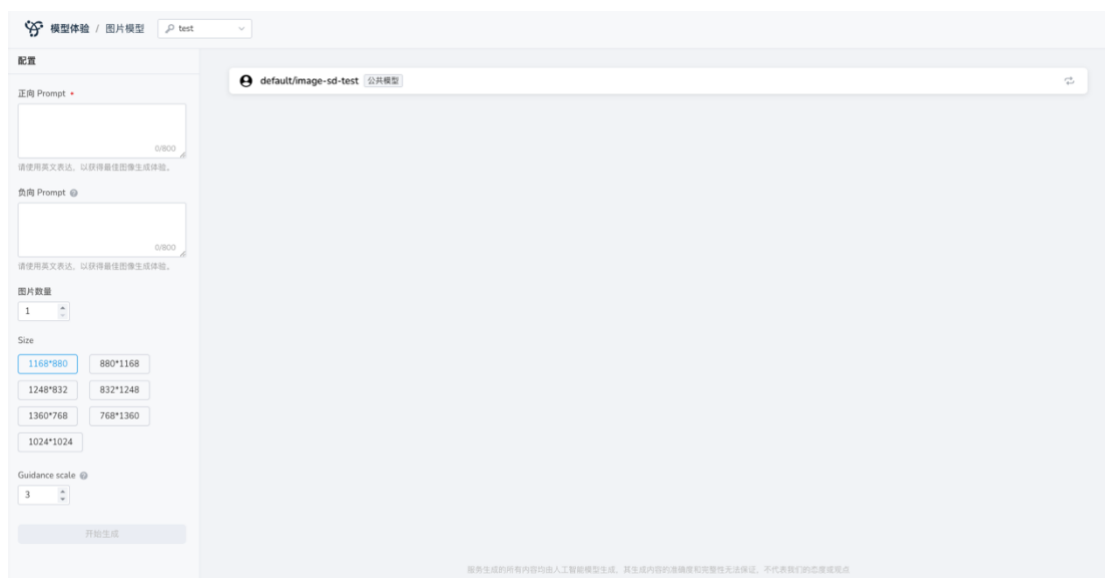


悬浮在模型卡片上，点击 **详情** 图标，可以查看模型介绍。



4.3 体验模型

在模型卡片上，点击 **体验** 图标，即可使用默认的语料库，开始体验 AI 对话。



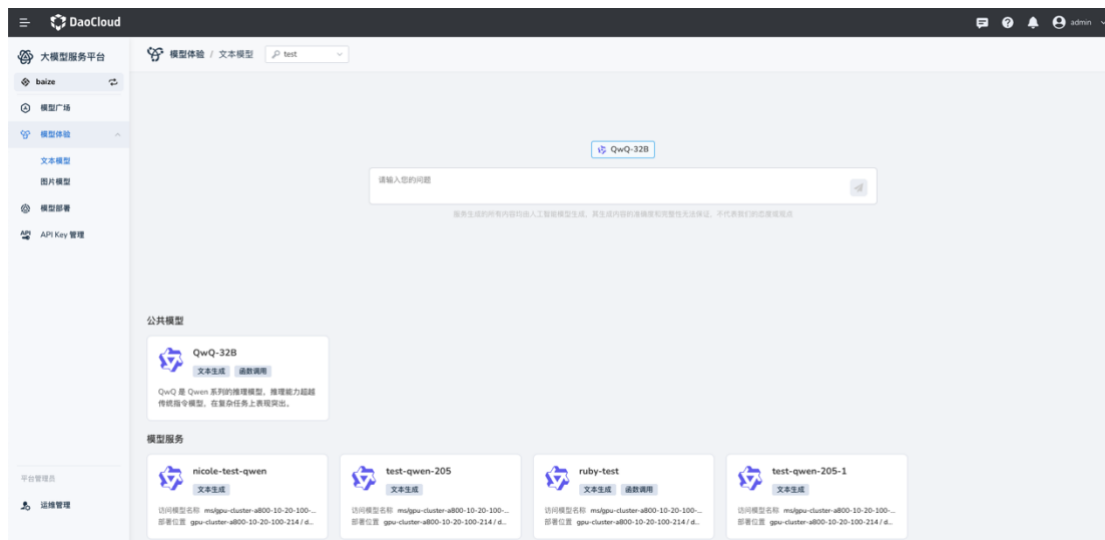
也可以从 **模型体验** 页面体验模型。参见[各项说明](#)。

4.4 部署模型

可以从 **模型广场** 或者 **模型服务** 页面轻松模型部署。参见[各项参数说明](#)。

5 用户视图 - 模型体验

Hydra 大部分模型不仅支持用户查看说明信息，还提供模型推理体验功能。例如，您可以通过与大语言模型进行日常对话，直观感受其能力。



5.1 体验入口

平台提供了两种便捷的体验入口，您可根据需求灵活选择：

- 入口一：在[模型广场](#)页面，选择喜欢的模型，点击 **体验**，即可进入对应的体验页面。
- 入口二：通过左侧菜单栏，点击 **模型体验**，选择模型类型后即可开启模型体验流程。目前可选 **文本模型** 和 **图片模型** 两种类型的模型体验。

5.2 体验说明

首次进入 **体验中心** 页面时，系统会推荐目前比较火爆的优质模型供您选择。您可以：

- 点击推荐的模型，直接体验大语言模型的对话功能；
- 或从 **公共模型** 列表中挑选其他感兴趣的模型进行体验；

- 如果你已经部署了自己的大模型，也可以从 **模型服务** 列表中挑选已经部署的模型进行体验。

5.3 模型类型

文本模型类型

体验中心支持使用预置的对话类模型，也支持选择您自己创建的服务，另外在线体验的模型可以选择多个。

图片模型类型

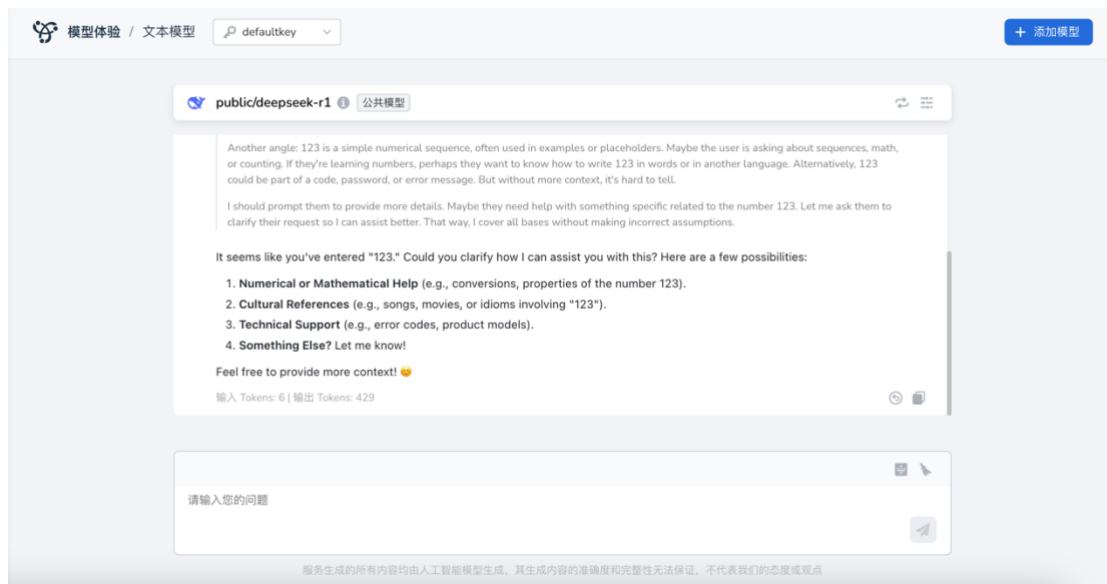
体验中心支持使用预置的图像生成模型 **Hidream-11-Dev**，也支持选择您自己创建的服务，另外在线体验的模型暂时支持 1 个。

完成一次完整的对话

完成一次对话，只需三步：

1. 首次进入，选择一个 **API Key**，如果没有，可以授权平台自动创建。
2. 需先选择模型：在列表中选择您要体验的模型，点击模型进行体验。您已添加模型，可跳过此步骤，直接体验感兴趣的模型。
3. 如果您对模型的回答不满意，还可以点击 **刷新** 重新生成答案。您还可以点击 **复制**，复制模型生成的内容。

此外，对话底部还展示了此轮回答调用 token 数的信息，您可以比较模型的性能。



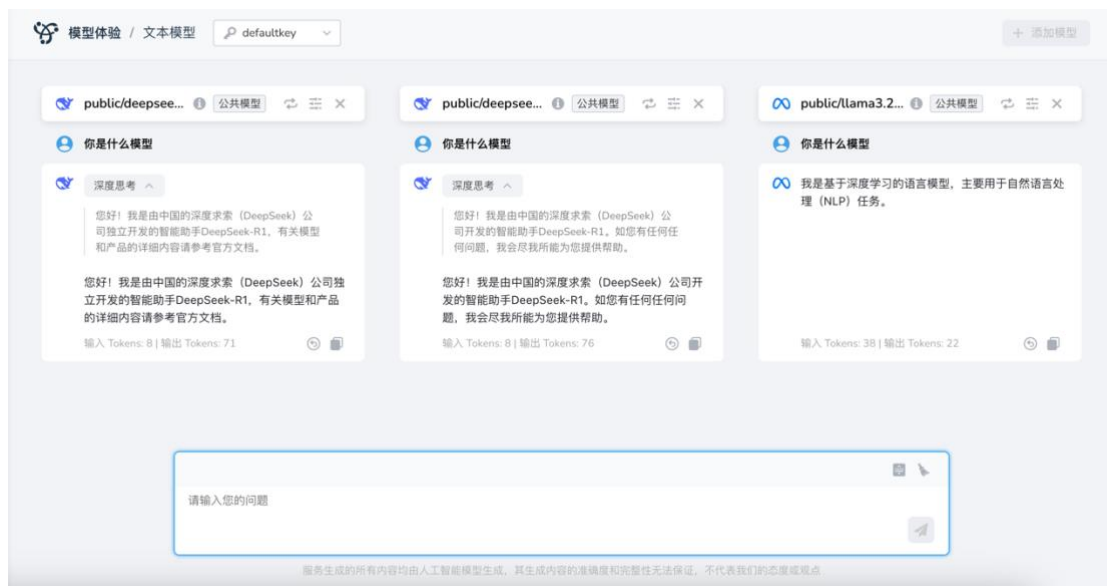
5.4 对话的更多功能

您可以点击文本对话框中右上角 **清除上下文**，结束此轮对话，并清空上下文关系，下方对话将不受上方内容影响。



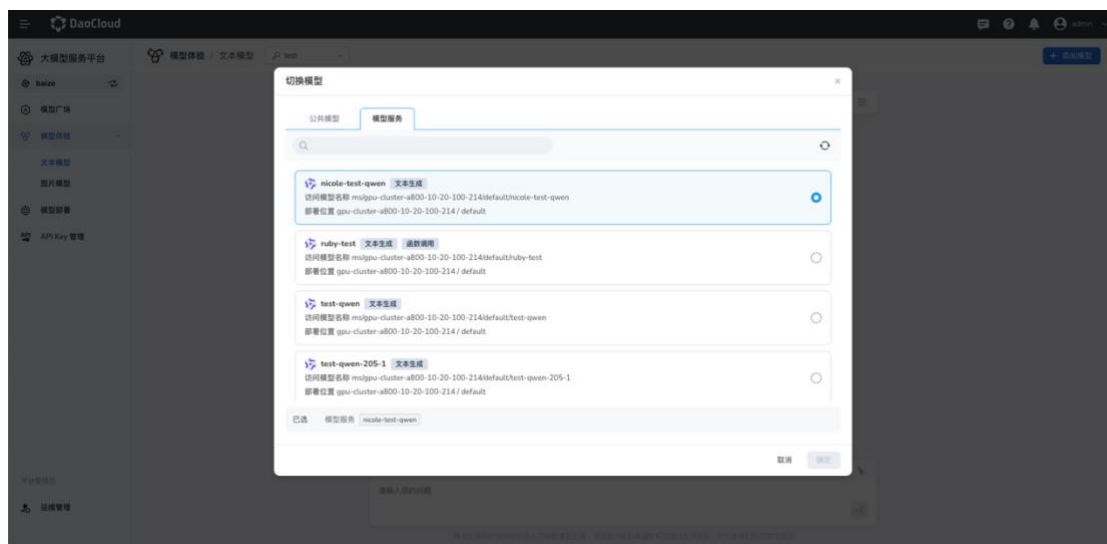
5.5 模型对比

点击页面右上角的 **添加模型**，文本生成类型的模型可以最多支持 3 个模型对比。



5.6 模型切换

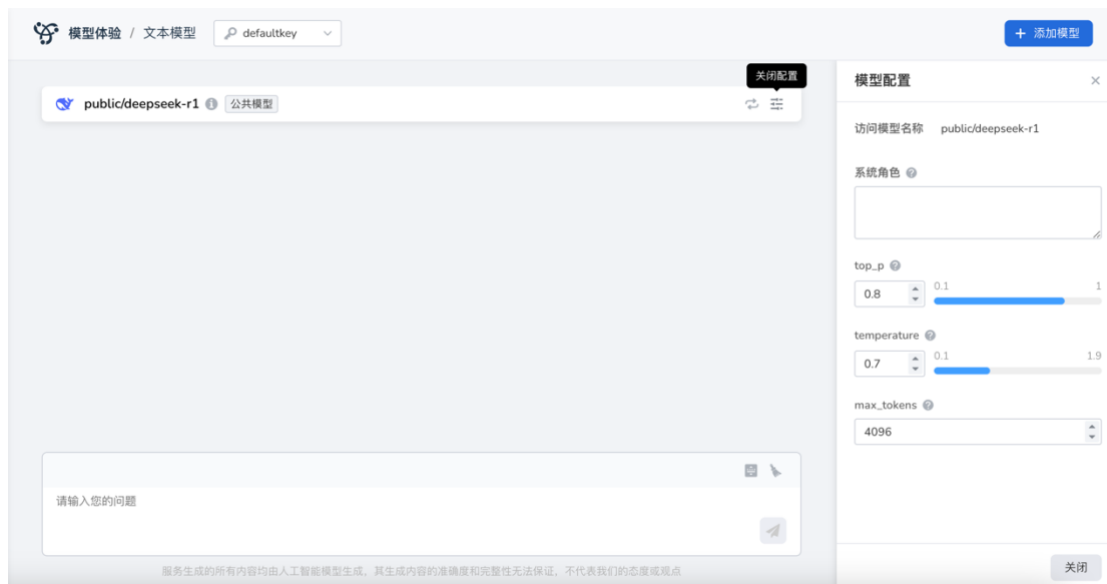
在顶部的模型信息栏，点击右侧的 **切换模型** 按钮，可以切换至其他模型。



5.7 模型参数设置

平台向用户展示了模型的若干可调参数，不同的参数设置会影响模型生成的回答，您可以根据自己的需求进行设置。

在顶部的模型信息栏，点击右侧的 **模型配置**，可以调整模型的参数。



每个参数名后面紧跟着？，将鼠标悬浮在上面，即可展示该参数的详细说明，帮助您理解该参数。

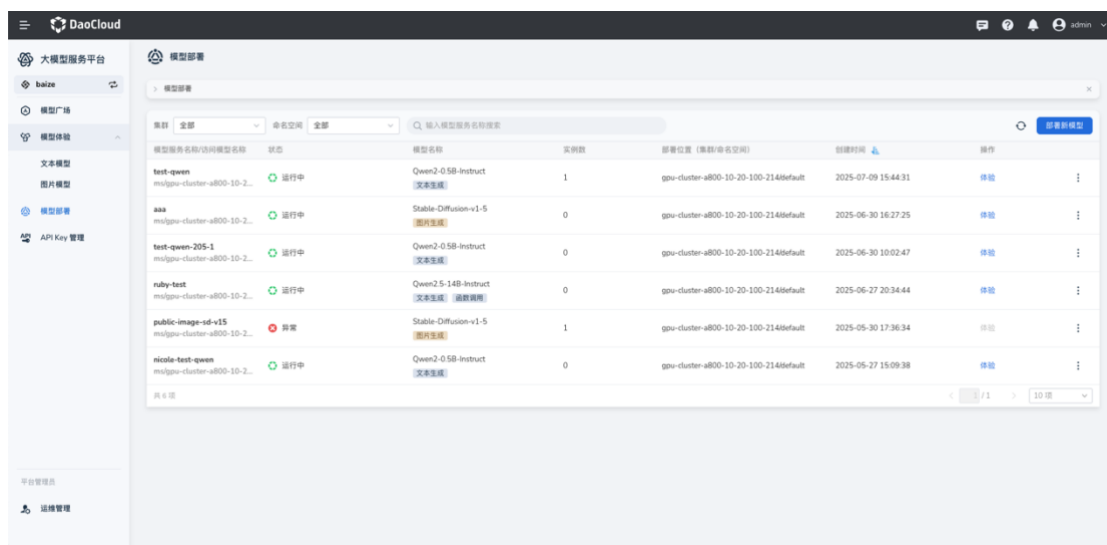
参数名	参数说明
System	系统角色：定义模型的行为准则和背景信息，明确模型需要承担的职责和角色，例如“AI 助手”。
Temperature	值越高，输出结果越随机多样；值越低，输出会更加集中且确定。建议仅设置一个。
TopP	控制输出文本的多样性，值越大，生成的文本越丰富多样。建议该参数和 Temperature 一起使用。
Max_tokens	模型可以生成的最大 Token 数，如设置为 0，则表示不限制。普通聊天建议 800-2000；代码生成建议：2000-3600；长文生成建议超过 4000。

参数名	参数说明
*	必须填写内容
负向 Prompt	描述你不希望包含在图像中的内容字数限制。
Guidance scale	决定了文本描述对生成图像的影响力度。较高的数值会使生成的图像更贴近描述，较低的数值则允许图像在保持文本核心要素的同时，展现出更多的创造性和变化。

6 用户视图 - 模型部署

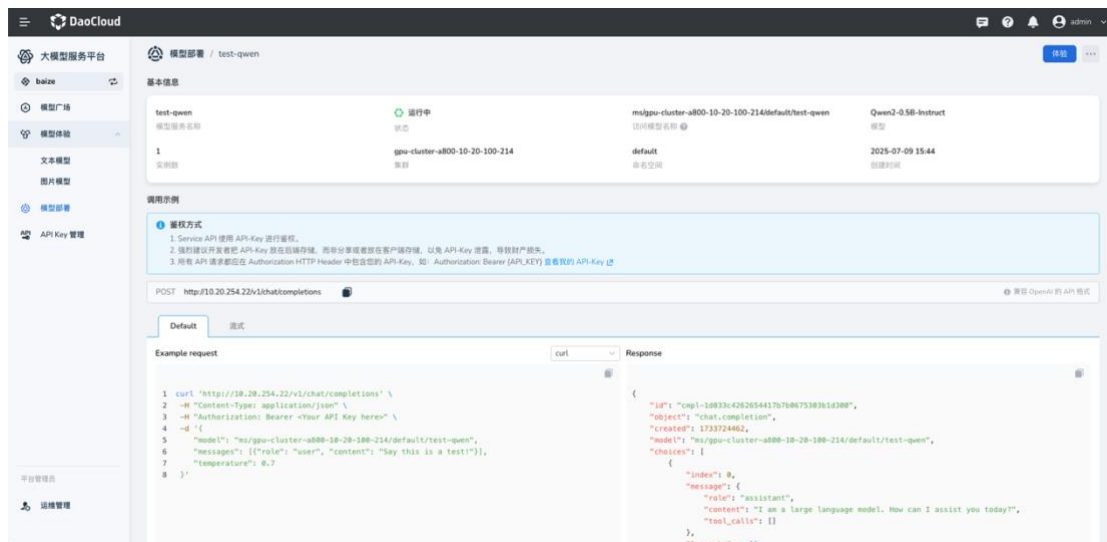
模型服务是一项将开源或微调后的大语言模型快速部署为可调用服务的解决方案。通过一键部署，将复杂的模型管理简化为标准化的服务形式，适配主流模型服务的 API 调用能力，满足即开即用的需求。

- 模型服务允许用户调用所选模型执行任务，如文本生成、图像理解、图像生成。
- 支持模型在线体验。



点击模型服务名称（需要处于运行中状态），可以进入服务详情页面。

模型服务详情中包含了该服务的基本信息、授权方式以及调用示例。



6.1 基本信息

- 模型服务名称：当前服务的名称，用于标识该模型服务
- 访问模型名称：每个模型服务都有唯一的路径名称，用于调用模型服务
- 模型服务 ID：用于账单查询
- 模型：当前服务使用的基座模型
- 实例数：该服务使用实例数
- 状态：当前服务的状态

6.2 鉴权方式

- API-Key 授权：

- 所有 API 请求均需要在 HTTP Header 中添加 Authorization 字段，用于验证身份
- 格式：Authorization: Bearer {API_KEY}
- 您可以通过页面中的“查看我的 API-Key”链接获取密钥
- 安全建议：将 API-Key 存储在后端服务器，避免将密钥暴露在客户端代码中，防止泄露

6.3 调用 API 示例

- 请求地址：POST 请求地址为 `https://sh-02.d.run/v1/chat/completions`

请求示例：使用 curl 调用 API

```
curl 'https://sh-02.d.run/v1/chat/completions' \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer <Your API Key here>" \  
-d '{  
  "model": "u-8105f7322477/test",  
  "messages": [{"role": "user", "content": "Say this is a test!"}],  
  "temperature": 0.7  
'
```

参数说明：

- model: 模型服务的访问路径名称（如 u-8105f7322477/test）。
- messages: 对话历史列表，包含用户输入，例如：
• [{"role": "user", "content": "Say this is a test!"}]

- **temperature:** 控制生成结果的随机性，值越高生成越有创意，值越低生成越稳定。

响应示例

```
{
  "id": "cmp-1d033c426254417b7b0675303b1d300",
  "object": "chat.completion",
  "created": 1733724462,
  "model": "u-8105f7322477/test",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "I am a large language model. How can I assist you today?"
      },
      "tool_calls": []
    }
  ],
  "usage": {
    "prompt_tokens": 25,
    "completion_tokens": 15,
    "total_tokens": 40
  }
}
```

响应字段说明:

- **id:** 生成结果的唯一标识符。
- **model:** 所调用的模型服务 ID。
- **choices:** 模型生成的结果数组。
 - **message:** 生成的内容。
 - **content:** 模型生成的文本内容。

- **usage:** 本次调用的 Token 使用情况:
 - **prompt_tokens:** 用户输入的 Token 数量。
 - **completion_tokens:** 生成结果的 Token 数量。
 - **total_tokens:** 总使用量。
- 集成开发示例

Python 示例代码

```
# Compatible with OpenAI Python library v1.0.0 and above
```

```
from openai import OpenAI
```

```
client = OpenAI(  
    base_url="https://sh-02.d.run/v1/",  
    api_key="<Your API Key here>"  
)
```

```
messages = [  
    {"role": "user", "content": "hello!"},  
    {"role": "user", "content": "Say this is test?"}  
]
```

```
response = client.chat.completions.create(  
    model="u-8105f7322477/test",  
    messages=messages  
)
```

```
content = response.choices[0].message.content
```

```
print(content)
```

node.js 示例代码

```
const OpenAI = require('openai');
```

```
const openai = new OpenAI({
```

```
    baseUrl: 'https://sh-02.d.run/v1',
    apiKey: '<Your API Key here>',
  });


  async function getData() {
    try {
      const chatCompletion = await openai.chat.completions.create({
        model: 'u-8105f7322477/test',
        messages: [
          { role: 'user', content: 'hello!' },
          { role: 'user', content: 'how are you?' },
        ],
      });

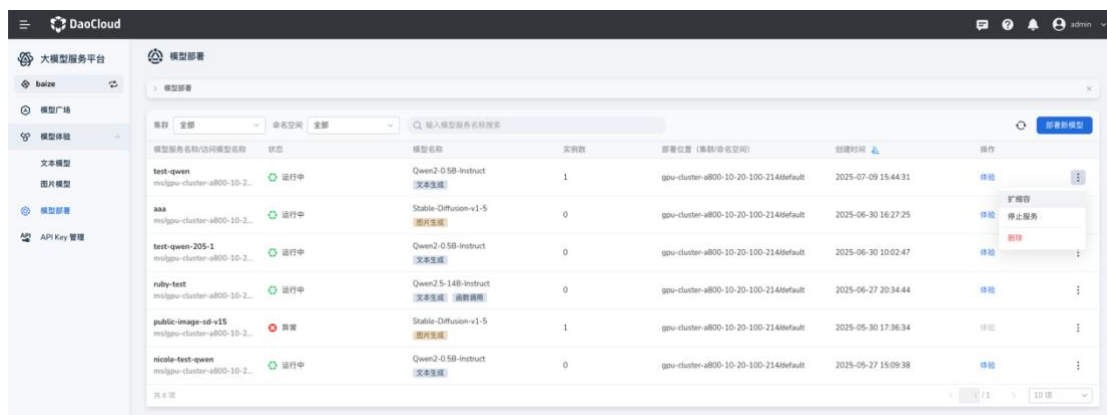
      console.log(chatCompletion.choices[0].message.content);
    } catch (error) {
      if (error instanceof OpenAI.APIError) {
        console.error('API Error:', error.status, error.message);
        console.error('Error details:', error.error);
      } else {
        console.error('Unexpected error:', error);
      }
    }
  }

  getData();
```

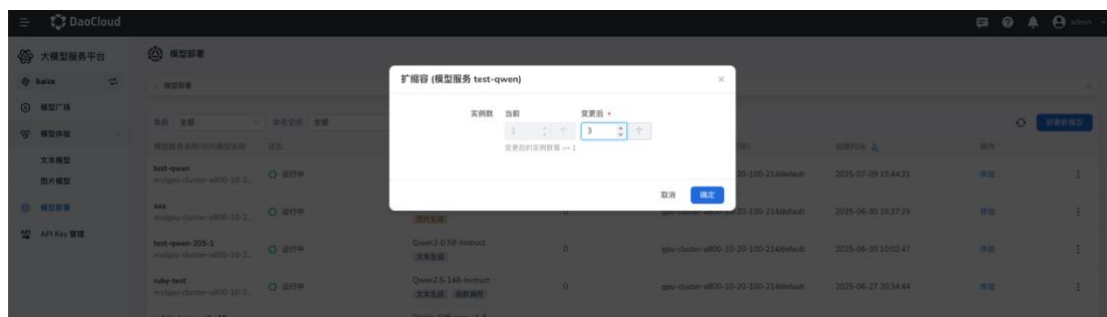
6.4 扩缩容

如果在模型使用过程中，发现资源不足或出现卡顿现象，可以对模型扩容。

在模型服务列表中，点击右侧的 ，在弹出菜单中选择 **扩缩容**。

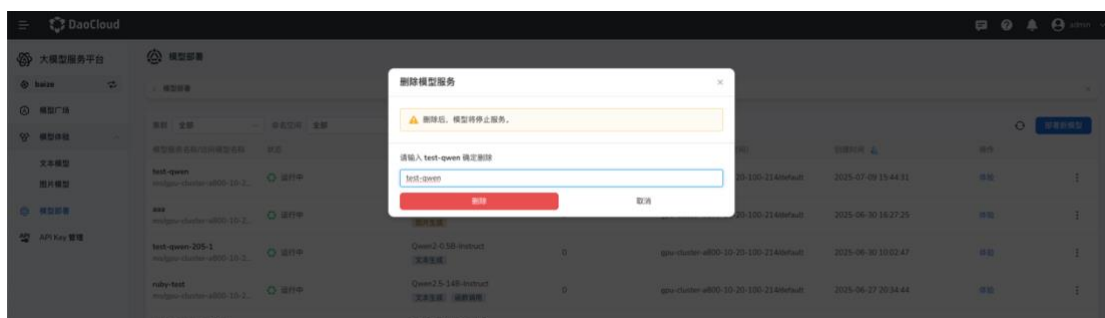


输入要增加的实例数，比如 2 个后，点击确定。



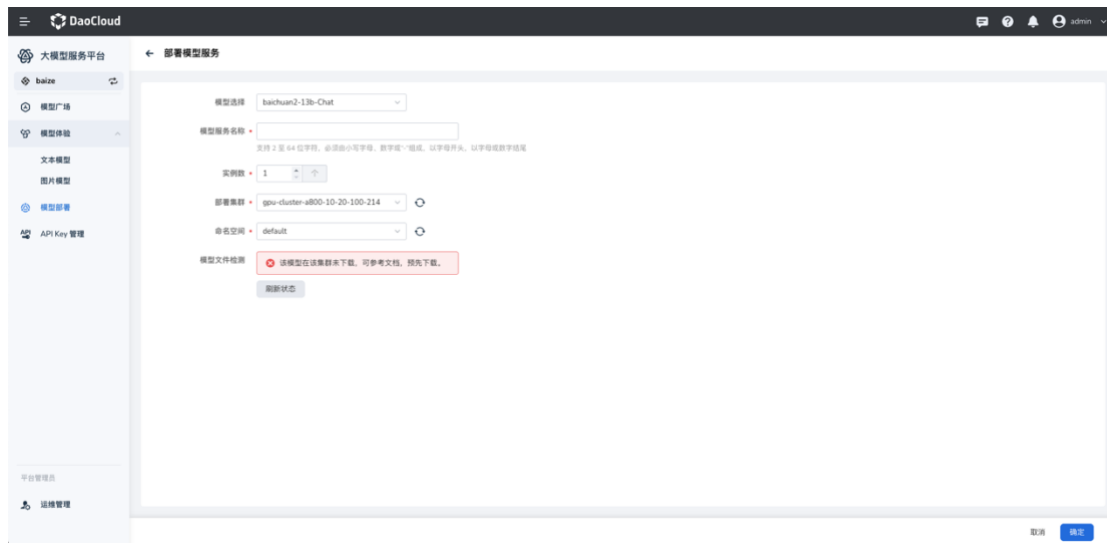
6.5 删除

1. 在模型服务列表中，点击右侧的 **⋮**，在弹出菜单中选择 **删除**
2. 输入要删除的模型服务名称，确认无误后点击 **删除**

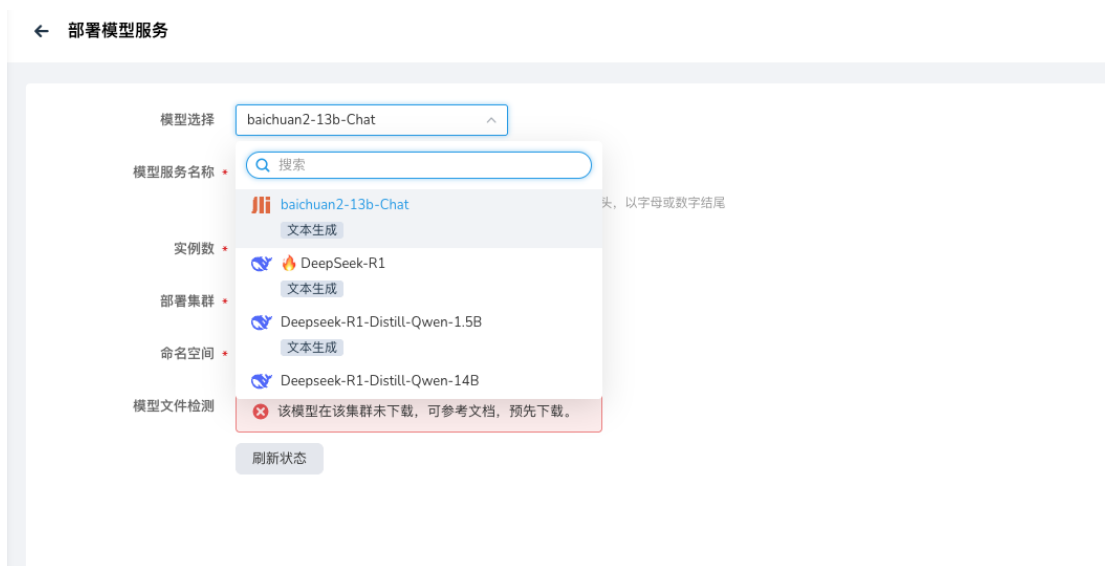


7 用户视图 - 部署新模型

可以从[模型广场](#)或者[模型部署](#)中进行模型部署。各个参数说明如下：



- **模型选择:** 选择需要部署的模型（如 DeepSeek-R1），可通过下拉菜单快速选择符合您业务需求和任务场景的模型。



- **模型服务名称:** 需满足以下要求：
 - 长度限制：2 - 64 个字符
 - 字符限制：仅支持小写字母、数字、短横线（-），且必须以小写字母或数字开头和结尾

- 示例: text-gen-service 或 model-01
- 实例数
 - 配置需要部署的实例数量。默认值: 1
 - 实例说明: 实例数量越多, 服务的并发能力越强, 但成本也会相应增加
- 部署集群: 选择部署到哪个集群, 建议优先选择距离较近的集群。
- 命名空间: 指定要部署到的目标命名空间。
- 模型文件检测: 选择模型、集群和命名空间后, 系统将自动执行模型文件检测。

8 用户视图 - 使用 Dataset 管理模型文件

Hydra 基于 BaizeAI/Dataset 来管理模型的权重文件。Dataset 抽象了 Kubernetes Volume 的使用, 极大简化了 PV/PVC 的创建和维护流程, 并支持多种数据源类型:

DatasetType	说明
GIT	通过 Git 协议下载
S3	文件存储在 S3 或 S3 协议兼容的对象存储里

DatasetType	说明
PVC	通过提前创建的 PVC 访问 Volume 的数据
NFS	通过 NFS 协议访问
HTTP	通过 HTTP 协议下载
CONDA	通过 Conda 下载 python package
REFERENCE	通过引用其他的 Dataset 来访问对应的数据
HUGGING_FACE	从 Hugging Face 下载对应的模型文件
MODEL_SCOPE	从 ModelScope 下载对应的模型文件

Dataset 还支持自动预加载：对于支持的类型会创建一个预处理 Job，将模型数据下载并存储到挂载的 PV 中，实现模型的快速初始化与复用。

8.1 自动下载

通过 Hugging Face 和 ModelScope

我们已经在 [BaizeAI/ModelHub](#) 将 Hydra 部署大模型需要的元信息都规整好了，以 qwen2-0.5b-instruct 为例，找到对应的 [metadata.yaml](#)：

```
apiVersion: model.hydra.io/v1alpha1
kind: ModelSpec
metadata:
  name: qwen2-0.5b-instruct
spec:
  descriptor:
    description:
```

```
enUS:
  A 0.5B parameter instruction-tuned model from the Qwen2 series, suitable
  for multilingual text generation and understanding.
zhCN: Qwen2 系列的 0.5B 参数指令微调模型，适用于多语言文本生成和理解。
display: Qwen2-0.5B-Instruct
source:
  huggingface:
    name: Qwen/Qwen2-0.5B-Instruct
  modelscope:
    name: Qwen/Qwen2-0.5B-Instruct
```

我们可以看到 `spec.source` 列出了从 [Hugging Face](#) 和 [ModelScope](#) 下载该模型的路径。我们根据这个信息可以创建如下的 Dataset:

从 Hugging Face 下载 qwen2-0.5b-instruct:

```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
    name: qwen2-5-0-5b-instruct
    namespace: public
spec:
  share: true
  source:
    options:
      repoType: MODEL
      type: HUGGING_FACE
    uri: huggingface://Qwen/Qwen2.5-0.5B-Instruct
```

从 ModelScope 下载 qwen2-0.5b-instruct:

```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
    name: qwen2-5-0-5b-instruct
    namespace: public
spec:
```

```
share: true
source:
  options:
    repoType: MODEL
  type: MODEL_SCOPE
uri: modelscope://Qwen/Qwen2.5-0.5B-Instruct
```

需要特别注意其中的字段设置：

- 需要指定 `metadata.labels.hydra.io/model-id` 为对应的模型 ID 来关联对应的模型服务
- 如果是模型体验里的服务，需要指定 `namespace` 为 `public`。如果是模型部署，则需要指定到对应的命名空间下。
- 通过指定 `spec.share` 为 `true`，我们允许其他的模型服务可以通过 `REFERENCE` 的方式直接引用该 `Dataset` 及对应的文件，避免重复下载。具体配置参考当前使用的 `Dataset` 管理模型文件。

通过 Git

你也可以使用 Git 下载，以 ModelScope 上的地址为例：

```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
  name: qwen2-5-0-5b-instruct
  namespace: public
spec:
  share: true
  source:
    options:
      repoType: MODEL
    type: GIT
```

```
uri: git://www.modelscope.cn/Qwen/Qwen2.5-0.5B-Instruct.git
secretRef: qwen-git-secret
```

对于需要凭证访问的地址，需要提前创建 Secret，结构如下：

```
kind: Secret
type: Opaque
metadata:
  name: qwen-git-secret
  namespace: public
data:
  username: xxx # 当类型为 MODEL_SCOPE、HTTP 和 GIT 时使用
  password: xxx # 当类型为 HTTP 和 GIT 时使用
  ssh-privatekey: xxx # 当类型为 GIT 时使用
  ssh-privatekey-passphrase: xxx # 当类型为 GIT 时使用
  token: xxx # 当类型为 HUGGING_FACE、MODEL_SCOPE 和 GIT 时使用
  access-key: xxx # 当类型为 S3 时使用
  secret-key: xxx # 当类型为 S3 时使用
```

8.2 手动下载

当网络访问不便的时候，你也可以通过提前下载并创建好对应的资源的方式来准备 Dataset。

使用 NFS

提前准备模型文件到 NFS，以路径 `nfs://192.168.1.11/dataset/Qwen/Qwen2.5-0.5B-Instruct` 为例，创建 Dataset：

```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
  name: qwen2-5-0-5b-instruct
  namespace: public
spec:
  share: true
```

```
source:  
  type: NFS  
  uri: nfs://192.168.1.11/dataset/Qwen/Qwen2.5-0.5B-Instruct
```

使用 Minio 等 S3 存储

提前将模型文件上传到诸如 Minio 之类的 S3 协议兼容的存储系统里，也可以直接在 Dataset 直接申明地址和凭证，自动加载模型文件：

```
apiVersion: dataset.baizeai.io/v1alpha1  
kind: Dataset  
metadata:  
  labels:  
    hydra.io/model-id: "qwen2-0.5b-instruct"  
  name: qwen2-5-0-5b-instruct  
  namespace: public  
spec:  
  share: true  
  source:  
    type: S3  
    uri: s3://minio-svc/dataset/Qwen/Qwen2.5-0.5B-Instruct  
    secretRef: minio-accesskey
```

当然，在外网可以访问的情况下，也可以直接填写 AWS S3、Azure **Blob Storage** 等服务商提供的存储地址。

使用提前创建的 PV/PVC

对于使用像 JuiceFS、local 等数据可以预先填充的持久化存储时，可以提前创建 PV 和 PVC，然后在 Dataset 中引用该 PVC 即可：

```
apiVersion: dataset.baizeai.io/v1alpha1  
kind: Dataset  
metadata:  
  labels:  
    hydra.io/model-id: "qwen2-0.5b-instruct"
```

```
name: qwen2-5-0-5b-instruct
namespace: public
spec:
  share: true
  source:
    type: PVC
    uri: pvc://your-pvc-name/path/to/model
```

引用其他的 Dataset

也可以直接引用其他创建过的 Dataset，避免重复下载。前提是被引用的 Dataset 必须 `share=true`，且 `shareToNamespaceSelector` 必须为空或者包含该 namespace：

```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
  name: qwen2-5-0-5b-instruct
  namespace: another-namepsace
spec:
  source:
    type: REFERENCE
    uri: dataset://public/qwen2-5-0-5b-instruct
```

8.3 Dataset Spec Reference

完整的 Dataset 结构以及各个字段的含义如下：

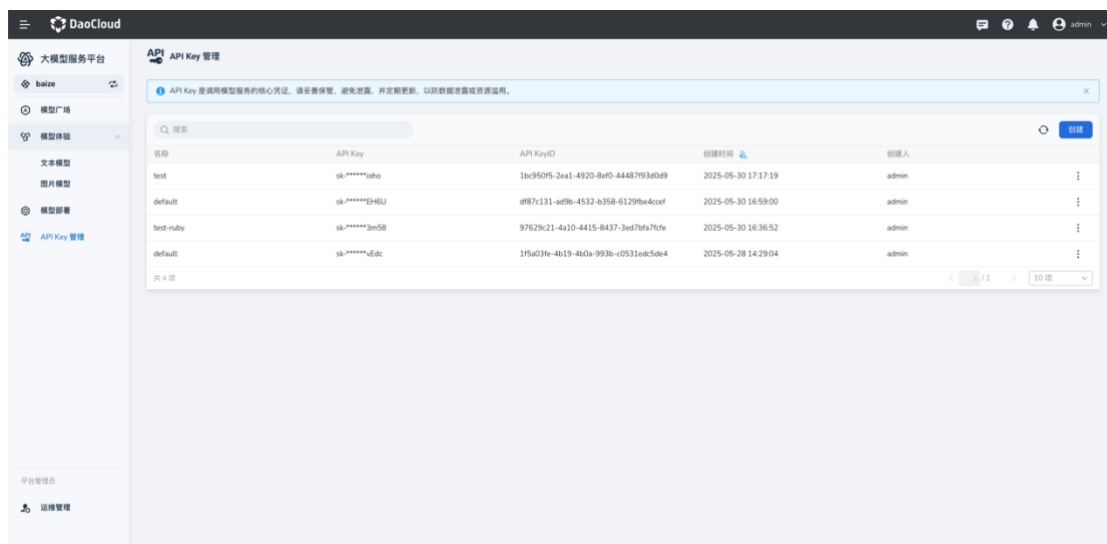
```
apiVersion: dataset.baizeai.io/v1alpha1
kind: Dataset
metadata:
  labels:
    hydra.io/model-id: "qwen2-0.5b-instruct"
  name: qwen2-5-0-5b-instruct
  namespace: public
spec:
  # Share indicates whether the model is shareable with others.
```

```
# When set to true, the model can be shared according to the specified selector.
share: true
# ShareToNamespaceSelector defines a label selector to specify the namespaces
# to which the model can be shared. Only namespaces that match the selector will have access to the
model.
# If Share is true and ShareToNamespaceSelector is empty, that means all namespaces can access
this.
shareToNamespaceSelector:
  matchExpressions:
    - key: env
      operator: In
      values: ["prod", "test"]
  matchLabels:
    region: sh-cn
# dataSyncRound is the number of data sync rounds to be performed.
#
dataSyncRound: 1
source:
  # options is a map of key-value pairs that can be used to specify additional options for the dataset
source, e.g. {"branch": "master"}
  # supported keys for each type of dataset source are:
  # - GIT: branch, commit, depth, submodules
  # - S3: region, endpoint, provider
  # - HTTP: any key-value pair will be passed to the underlying http client as http headers
  # - PVC:
  # - NFS:
  # - CONDA: requirements.txt, environment.yaml
  # - REFERENCE:
  # - HUGGING_FACE: repo, repoType, endpoint, include, exclude, revision
  # - MODEL_SCOPE: repo, repoType, include, exclude, revision
  options:
    repoType: MODEL
    type: MODEL_SCOPE
  # uri is the location of the dataset.
  # each type of dataset source has its own format of uri:
  # - GIT: http[s]://<host>/<owner>/<repo>[.git] or git://<host>/<owner>/<repo>[.git]
  # - S3: s3://<bucket>/<path/to/directory>
  # - HTTP: http[s]://<host>/<path/to/directory>?<query>
  # - PVC: pvc://<name>/<path/to/directory>
  # - NFS: nfs://<host>/<path/to/directory>
  # - CONDA: conda://<name>?[python=<python_version>]
  # - REFERENCE: dataset://<namespace>/<dataset>
  # - HUGGING_FACE: huggingface://<repoName>?[repoType=<repoType>]
  # - MODEL_SCOPE: modelscope://<namespace>/<model>
```

```
uri: modelscope://Qwen/Qwen2.5-0.5B-Instruct
# secretRef is the name of the secret that contains credentials for accessing the dataset source.
secretRef: secret-name
mountOptions:
  # path is the path to the directory to be mounted.
  # if set to "/", the dataset will be mounted to the root of the dest volume.
  # if set to a non-empty string, the dataset will be mounted to a subdirectory of the dest volume.
  path: /data
  mode: "0774"
  uid: 1000
  gid: 1000
# volumeClaimTemplate defines the PVC spec generated by dataset controller,
# except for type `REFERENCE`
volumeClaimTemplate: {}
```

9 用户视图 - API Key 管理

API Key 是调用模型服务的核心凭证，用于验证用户身份并保护数据安全。



9.1 功能说明

- API Key 作用:

- API Key 是调用模型服务的必要凭证，用于身份验证
- 通过 API Key，您可以安全地调用已部署的模型服务
- 安全提示：
 - 请妥善保管 API Key，避免暴露到客户端代码或公共环境中
 - 如果 API Key 泄露，请及时删除并重新生成新的 Key

9.2 创建 API Key

1. 在 **API Key 管理** 页面中，点击右上角的 **创建** 按钮
2. 在弹出的窗口中，填写 API Key 的名称（如 test-key），用于标识该 Key 的用途或所属项目
3. 点击 **确定**，系统将生成一个新的 API Key

创建完成后请在首次显示时保存 API Key，后续不会再次显示完整密钥。

9.3 查看 API Key

- 在 API Key 列表中，会显示已创建的所有 API Key：
 - 名称：API Key 的标识名称，便于用户区分不同用途的 Key

- **API Key:** 部分显示密钥内容，仅用于参考
- **创建时间:** API Key 的生成时间
- 点击右上角的刷新 按钮可以更新 Key 列表

9.4 删除 API Key

1. 在列表中找到需要删除的 API Key。
2. 点击某一行，可以执行删除操作
3. 在弹窗中确认删除操作
4. 删除后，该 API Key 将立即失效，所有依赖于此 Key 的服务调用将被拒绝

9.5 使用 API Key 调用服务

在调用模型服务时，需在 HTTP 请求头中添加以下字段：

```
Authorization: Bearer {API_KEY}
```

示例：

```
curl 'https://sh-02.d.run/v1/chat/completions' \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer sk-x1VDTAFB7Ra1hldATbncOa_dddVttDvRHQibTA-Oi7ucU" \  
-d '{  
  "model": "u-8105f7322477/test",  
  "messages": [{"role": "user", "content": "Hello, model!"}],  
  "temperature": 0.7  
}'
```

9.5.1 注意事项

- **API Key 数量限制：**每个账号允许创建的 API Key 数量有限，请根据需要合理分配
- **密钥泄露处理：**如果发现密钥泄露，请立即删除旧密钥并重新创建新密钥
- **Key 的权限管理：**不同的 API Key 可用于不同的项目或服务，便于权限隔离
- **定期更新密钥：**为了安全性，建议定期删除旧 Key 并生成新 Key

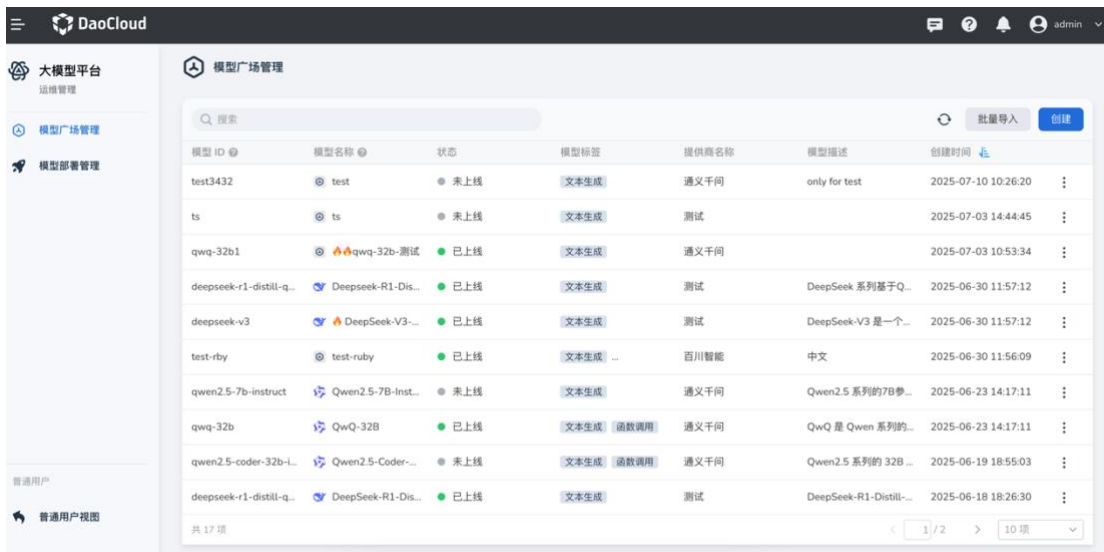
10 运维管理 - 模型广场管理

平台管理员可以通过 **运维管理** 功能对模型广场中的大模型进行全面管理，包括批量导入、模型创建、参数编辑、以及模型的上线和下线操作。此功能可帮助平台管理员快速构建和维护模型生态，提升平台整体运营效率。

10.1 批量导入模型

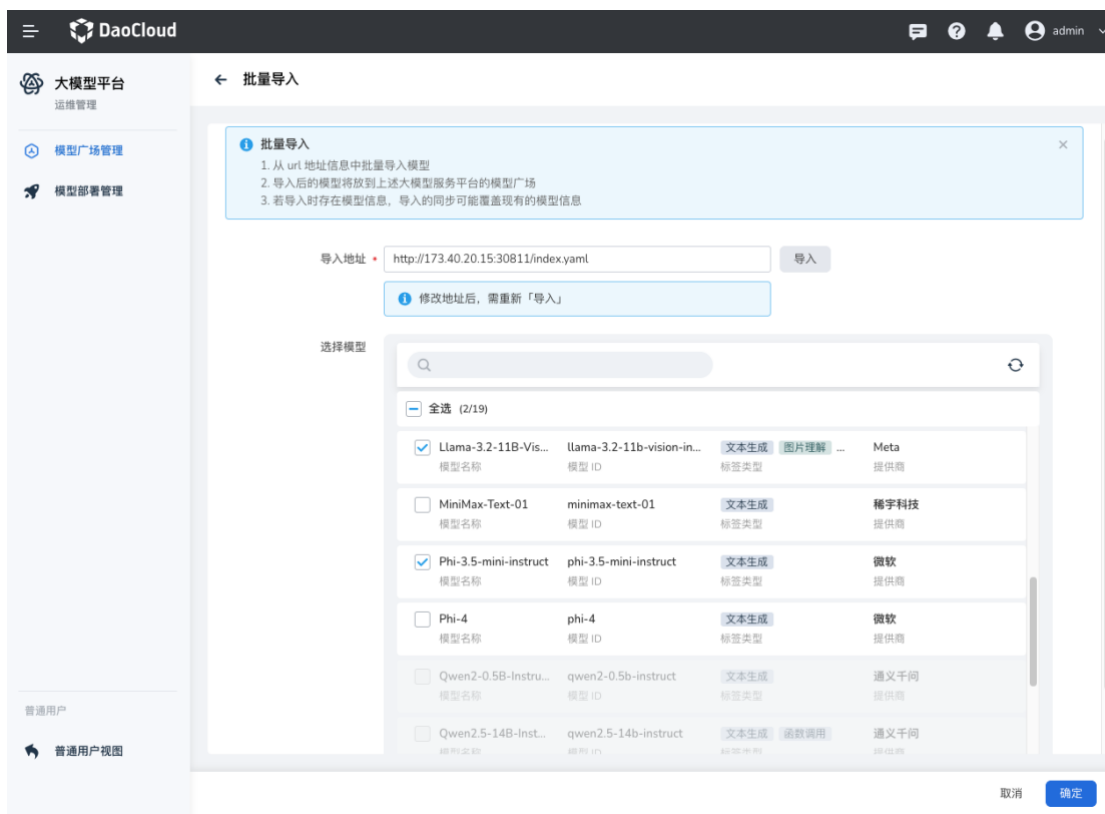
支持通过远程 URL 批量导入多个模型，简化模型引入流程。

1. 在 **模型广场管理** 页面中，点击页面右上角的 **批量导入** 按钮。



2. 在弹出的导入窗口中：

- 粘贴包含模型文件地址的 URL；
- 平台会自动解析并展示可导入的模型列表；
- 勾选一个或多个模型；
- 点击 **导入** 按钮。



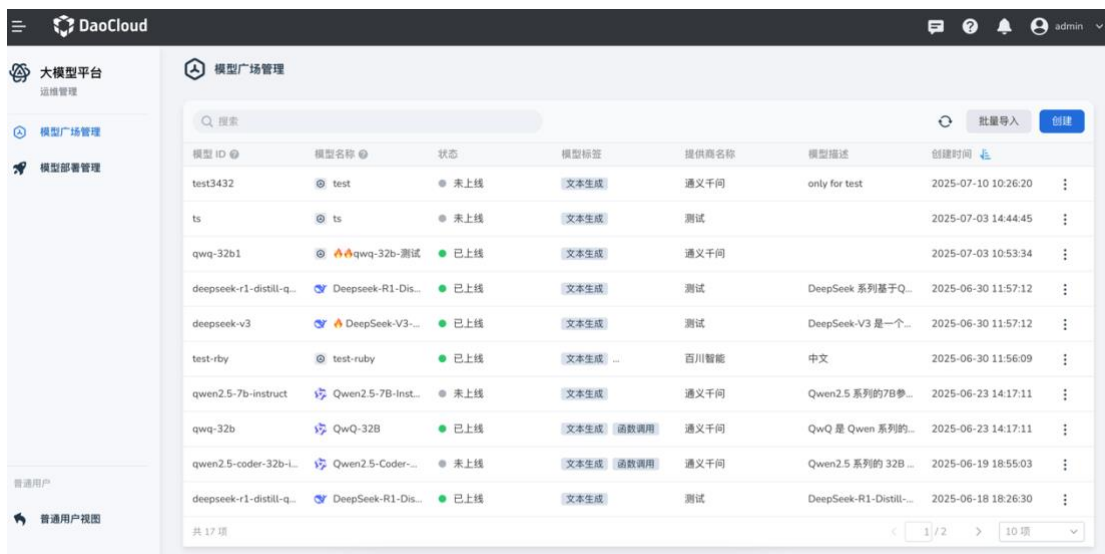
已存在于系统中的模型无法重复导入。

3. 返回模型列表页面，新导入的模型会默认显示在列表的顶部，方便后续操作。

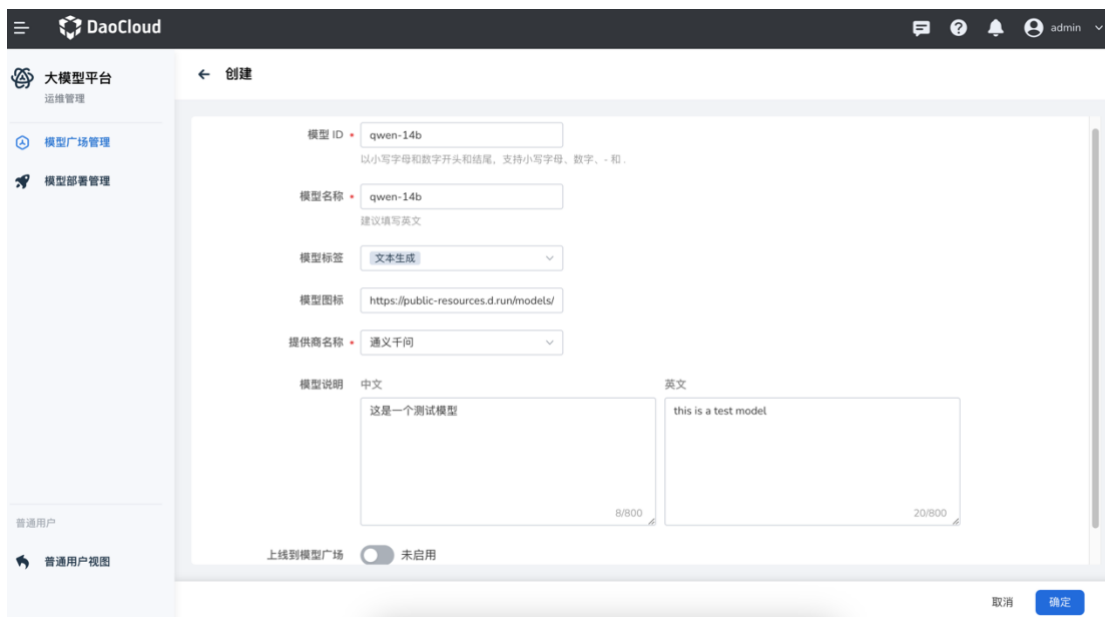
10.2 创建模型

如果需要手动添加模型，可以使用模型创建功能。

1. 在 **模型广场管理** 页面中，点击页面右上角的 **创建** 按钮。




2. 在弹出的创建模型窗口中，依次填写参数信息后，点击 **确定**

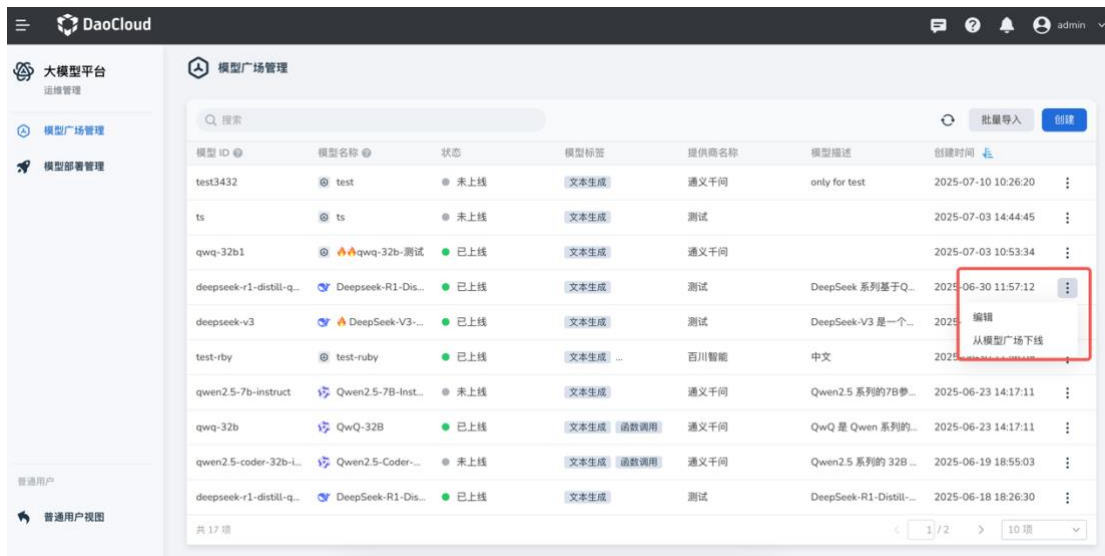


3. 返回模型列表页面，新创建的模型将显示在顶部。

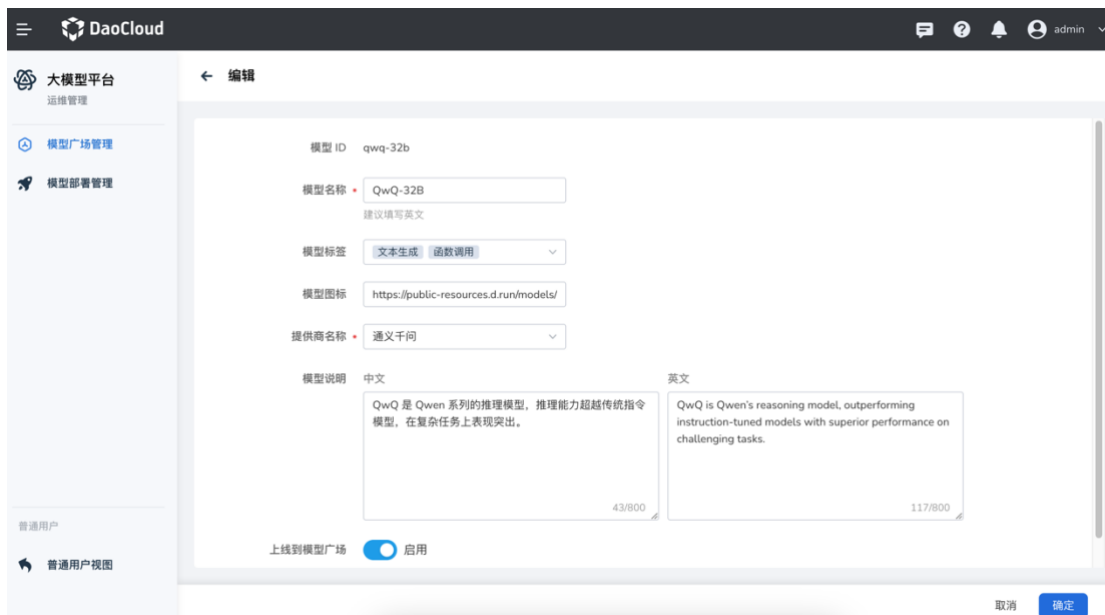
10.3 编辑模型

管理员可以在模型创建后，随时编辑模型的参数信息，以满足实际运行或业务变更需求。

1. 在模型列表中，点击对应模型右侧的  菜单图标，选择 **编辑** 选项。



2. 在弹出的编辑窗口中，根据需要修改模型的各项参数，点击 **确定** 保存变更。



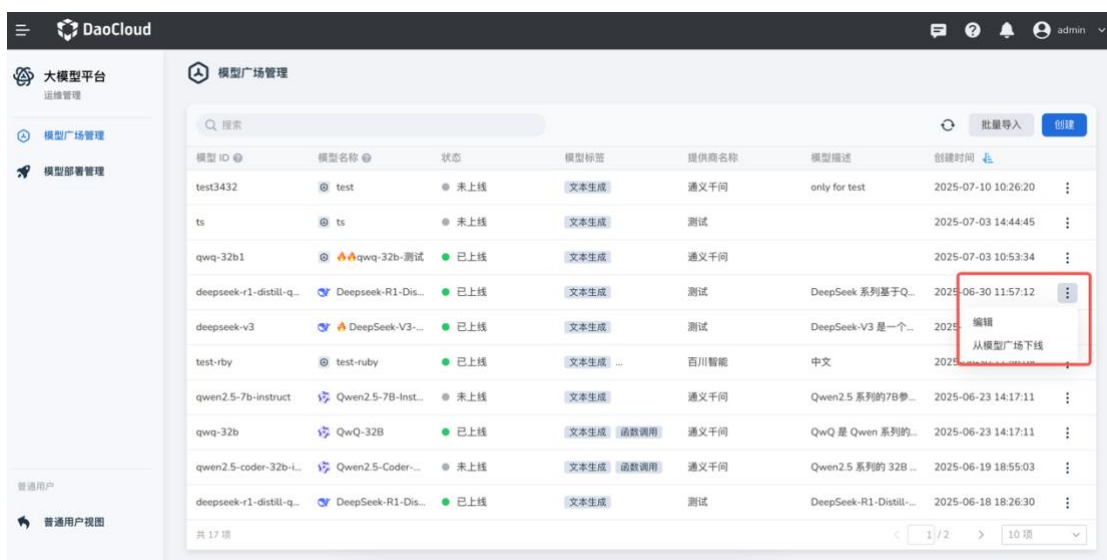
3. 返回模型列表后，模型位置保持不变，已更新的参数立即生效。

修改模型参数不会影响模型当前的上线状态，但建议在空闲时段进行编辑操作。

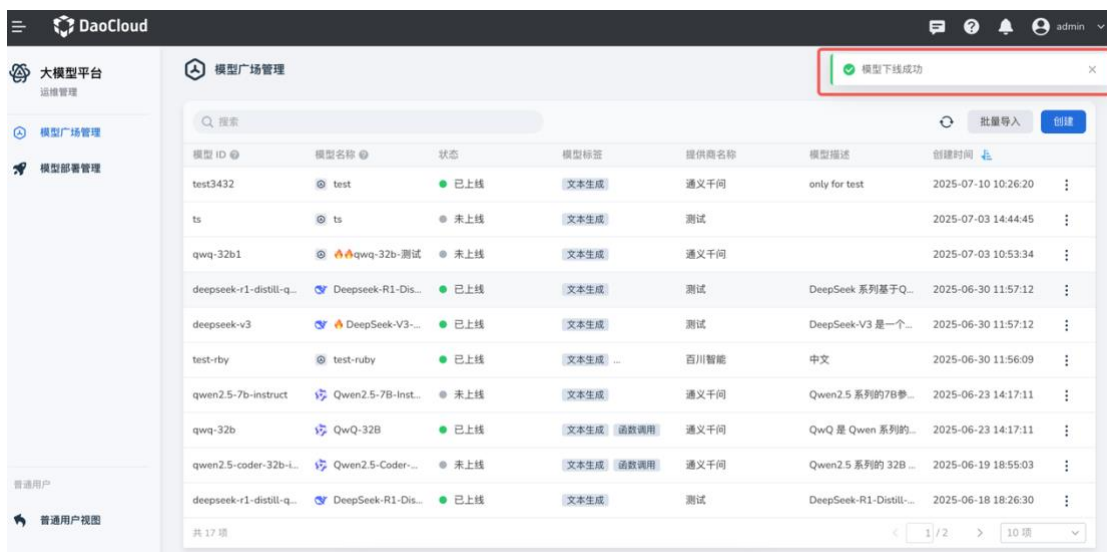
10.4 模型上线/下线

管理员可根据运营策略或资源调度情况，手动控制模型在模型广场中的展示状态。

1. 在模型列表中，点击目标模型右侧的 **⋮** 菜单，选择 **上线到模型广场** 或 **从模型广场下线**



2. 以下线模型为例，选择 **从模型广场下线** 后，页面将显示下线成功的提示，模型也会从模型广场的页面中隐藏。



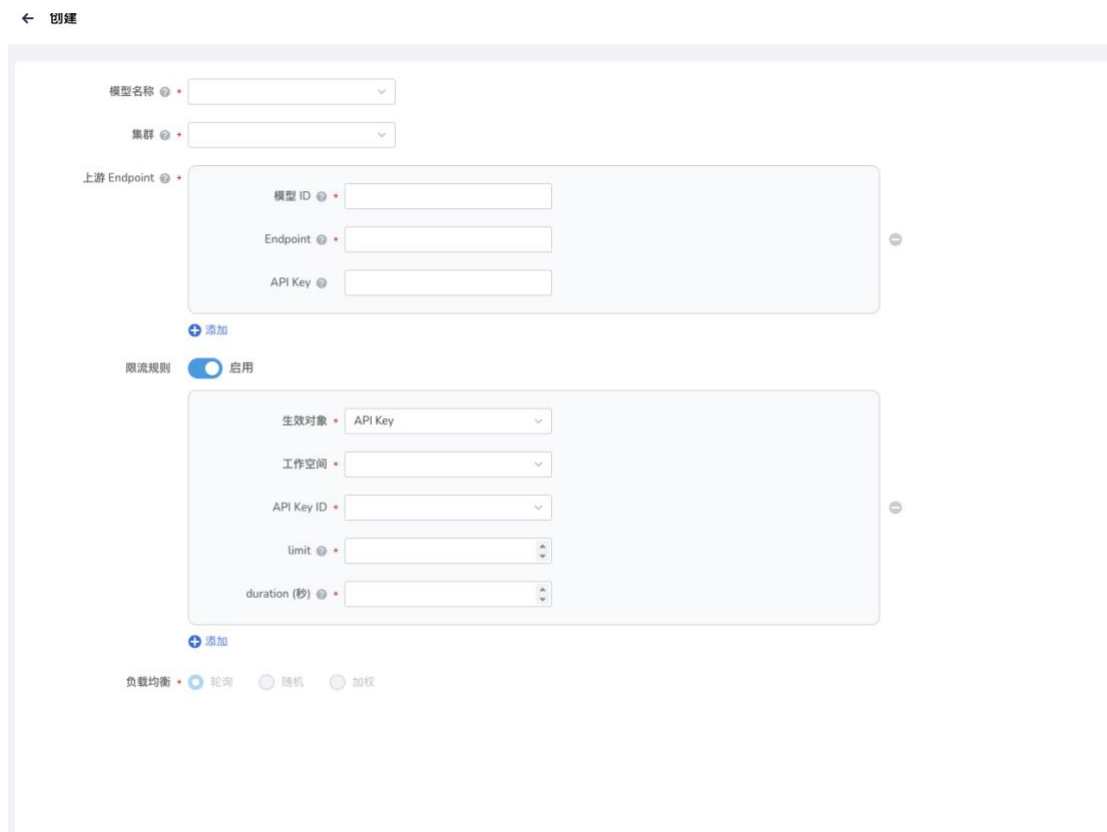
模型下线仅影响其在模型广场前台的可见性，不会删除模型本体或其配置。如需彻底移除，请使用“删除”功能。

通过以上操作，平台管理员可以高效地完成模型的导入、配置与发布，确保平台模型广场内容的更新及时、配置准确，满足多样化的业务需求。

11 运维管理 - MaaS 模型管理

MaaS 模型管理 是大模型平台的核心模块，运维管理平台面向平台运维管理者，提供模型接入、启停、限流、负载均衡等全生命周期管理能力。通过本模块，管理员可将私有或开源模型快速接入平台，并以标准 API 形式对外提供服务。

11.1 创建 MaaS 模型



字段说明

字段名称	说明
模型名称	从模型广场的模型列表中选择模型
集群	在配置 MaaS 模型时，必须为其选择一个"入口集群"，平台会根据这个集群自动生成模型的访问路径，
模型 ID	这个上游 endpoint 所提供的模型的名称，例 deepseek-chat
Endpoint	上游服务的完整 http(s) 地址，例 https://api.deepseek.com
API Key	若上游服务需要 Token 认证，请填写

限流规则（可选）：

如需要限流，可打开该配置。

生效对象：可用于 API Key 或者现有工作空间。

Note

生效对象为 API Key：工作空间下单个 API Key 对该模型的调用总量受以下限制。

生效对象为现有工作空间：限制当前工作空间下所有 API Key 的模型调用总量。

- 工作空间：选择要生效的工作空间
- API Key ID：选择要生效的 API Key。
- limit：单个 Key 在 duration 秒内最大请求次数，例 100。
- duration：秒，例 60。
- 可添加多条规则

负载均衡策略：当前仅支持轮询策略

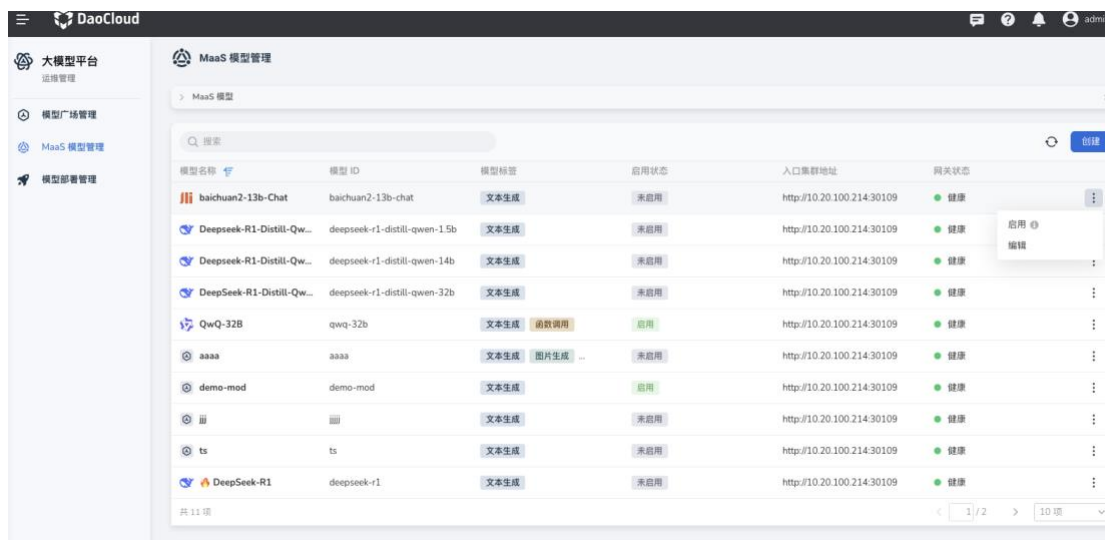
1. 轮询（默认）：顺序转发到多个 Endpoint
2. 随机：随机挑选
3. 加权：需在下拉框输入权重值，总和为 100

11.2 MaaS 模型管理列表

在 MaaS 模型管理列表中，用户可以查看所有已创建的 MaaS 模型，包括模型名称、模型 ID、模型标签、启用状态、入口集群地址、网关状态等。点击 启用 即可在 普通用户视图 中的模型广场查看到该模型。

Note

前提条件：需要在运维管理平台的模型广场管理中将该模型 **上线到模型广场**



12 运维管理 – 模型部署管理

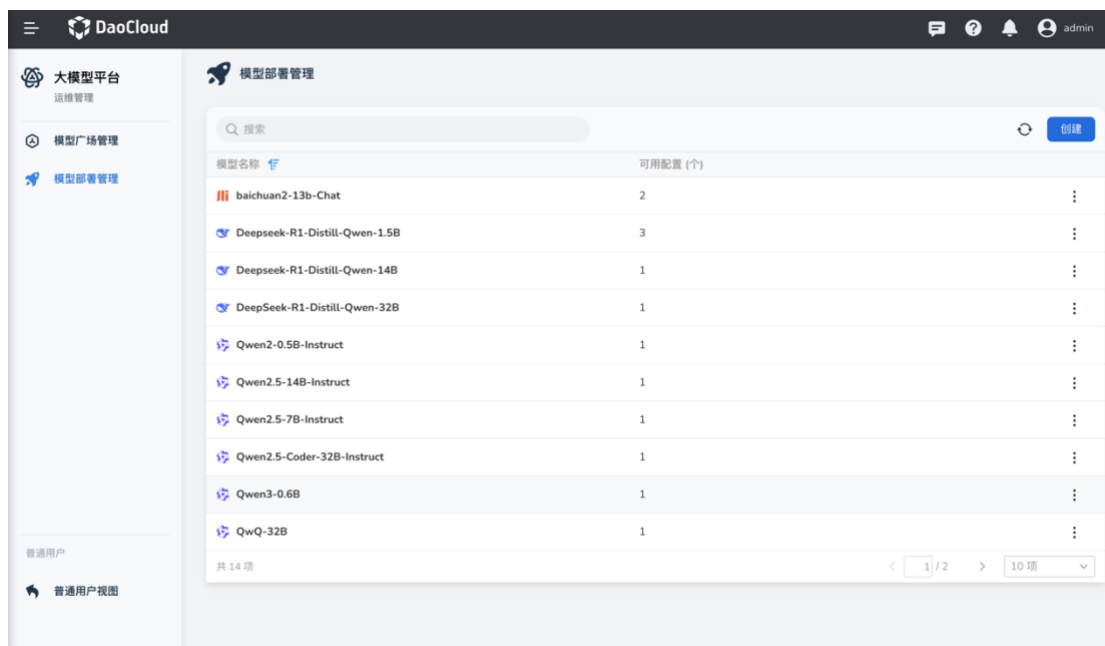
平台管理员可以在 **运维管理** 中管理模型部署，包括模型部署配置文件的创建、编辑、删除、启动和停止等操作。

Note

此处创建的是 **模型部署的配置文件**，并非模型本体文件。如需管理或下载模型文件，请参考使用 [Dataset 管理模型文件](#)。

12.1 模型部署列表

列表展示了所有已创建的模型部署配置，包括模型名称及其对应的可用配置数量。



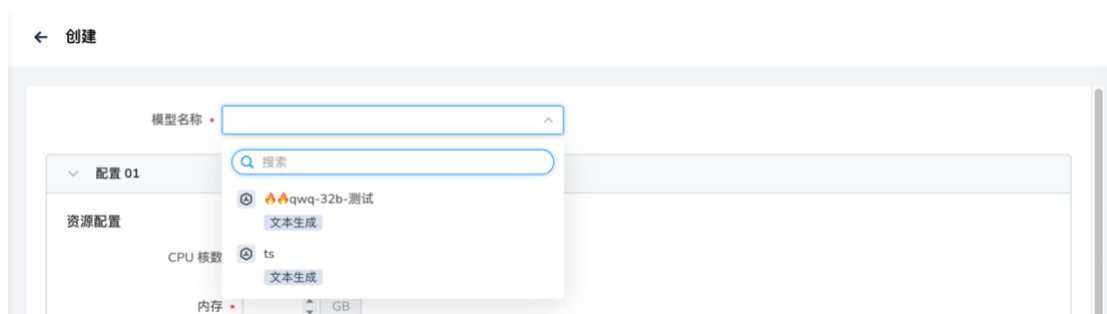
12.2 创建模型

在 **模型部署管理** 页面，点击右上角的 **创建** 按钮，进入模型部署配置创建页面。

在创建页面中，用户需要：

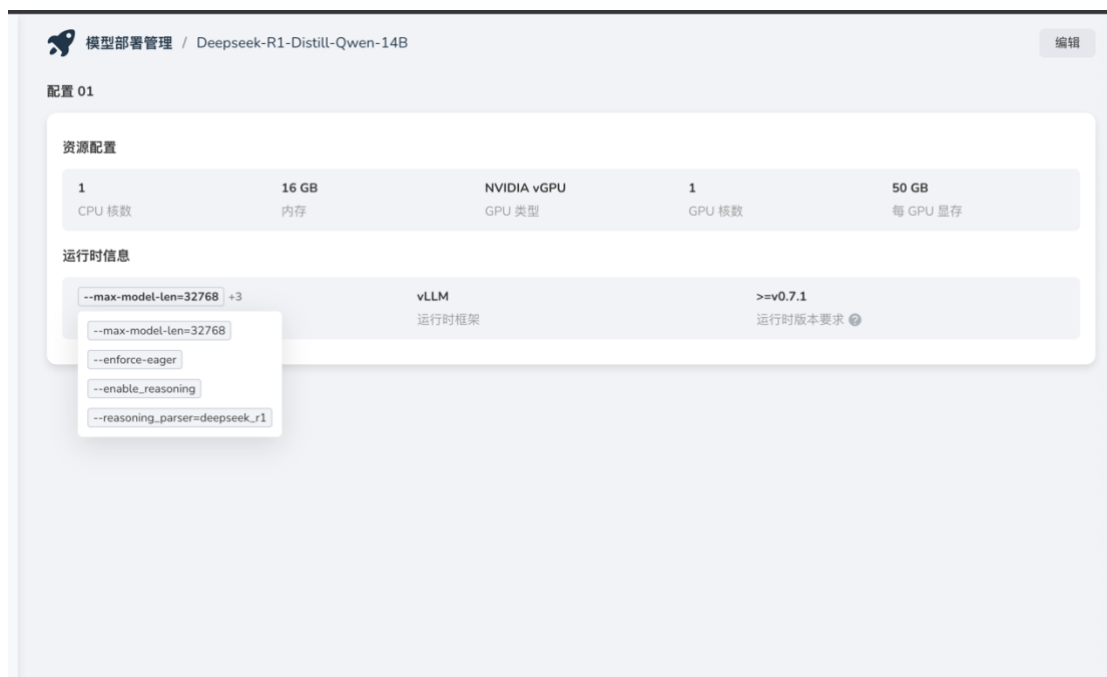
- 选择模型**：该模型列表与“模型广场管理”中的模型列表一致。
如需新增模型，可前往“模型广场管理”添加。
- 配置资源与运行时信息**：包括资源配额（CPU 核数、内存大小、GPU 类型、GPU 数量、单卡显存等）和运行时参数（如最大模型长度、运行时框架、框架版本等）。
- 支持多配置创建**：用户可为同一个模型添加多个不同的部署配置。

配置完成后，点击 **确定** 即可完成创建。



12.3 查看模型详情

可点击列表中的某个模型名称，可以查看其配置信息，包括资源配置、运行时信息等。



13 大模型服务平台使用示例场景

您可以参考以下示例场景，在开发工作中配置并使用 Hydra 提供的模型服务。

模型调用示例

- 参考[模型调用示例](#)，选择模型调用的方式。
- 参考[获取 API Key](#)，获取密钥。

场景介绍如下：

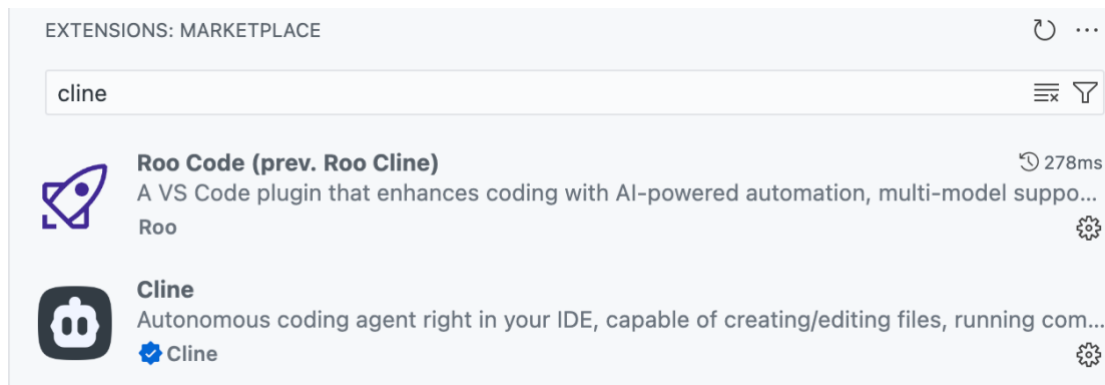
应用场景	操作说明
Cline in VSCode	在 VSCode 中通过 Cline/RooCode 使用 Hydra 的模型服务
Cherry Studio	在 Cherry Studio 中使用 Hydra 的模型服务
Bob Translate	在 Bob Translate 中使用 Hydra 的模型服务
Lobe Chat	在 Lobe Chat 中使用 Hydra 的模型服务
自定义大模型推理运行时	在自定义大模型推理运行时中使用 Hydra 的模型服务

14 在 VSCode 和 Cline 中使用 Hydra

[Cline](#) 是一个 VSCode 插件，它可以帮助您在 VSCode 中使用 [Hydra](#) 的模型服务。

14.1 安装 Cline

在 VSCode 中搜索 [Cline](#) 插件并安装。



在这里也可以下载使用 RooCode 插件，它是 Cline 的一个分支。

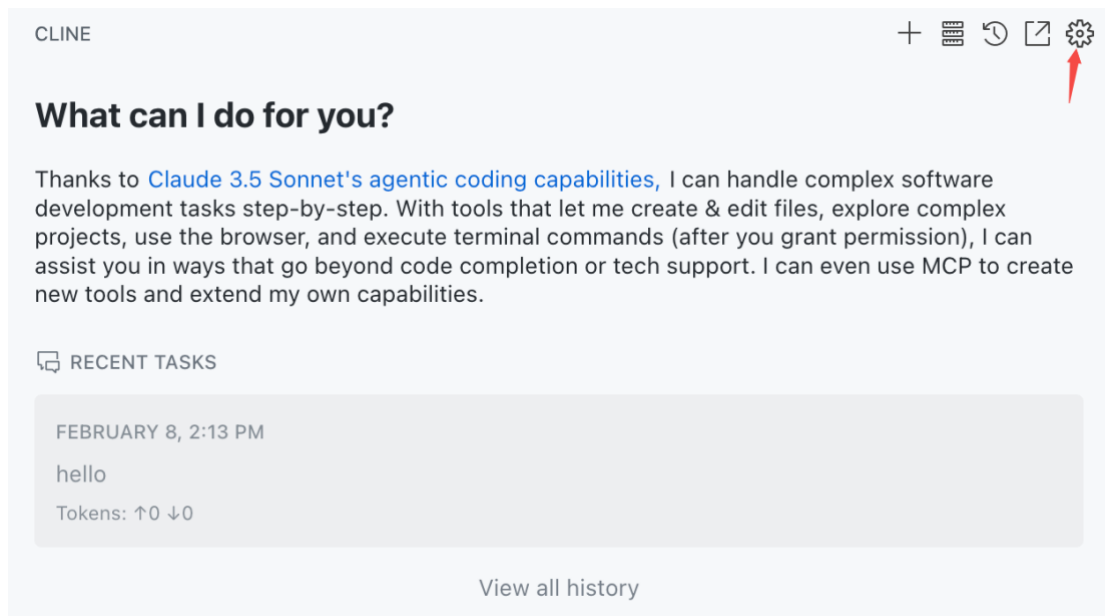
Cline 的前身是 Claude Dev, RooCode(RooCline) 是基于 Cline 的分支。

如果您的网络无法直接下载插件，可以考虑去 VSCode 插件市场下载插件的 `.vsix` 文件，然后在 VSCode 中选择 `Install from VSIX` 安装。





- [Cline](#)
- [RooCode](#): 这是 Cline 的一个分支

14.2 配置 Cline

打开 Cline 的配置页面：



- API Provider: 选择 OpenAI Compatible
- Base URL: 输入 `https://chat.d.run`
- API Key: 输入您的 API Key
- Model ID: 输入您的 Model ID
 - 可以从 Hydra 的模型广场获取，MaaS 模型开头为 `public/deeepseek-r1`
 - 独立部署的模型服务，可以从模型服务列表获取

CLINE +    

Settings Done

API Provider

Base URL

API Key

Model ID

Set Azure API version

(Note: Cline uses complex prompts and works best with Claude models. Less capable models may not work as expected.)

Custom Instructions

These instructions are added to the end of the system prompt sent with every request.

14.3 Cline 使用演示

CLINE + ☰ ↶ 🗑️ ⚙️

Task ×

Hello

Tokens: ↑ 25.9k ↓ 703 🗑️ 21.2 MB

Context Window: 13.6k ▬ 128.0k

✓ API Request ▾

Reasoning: ... user feels comfortable providing the details I need to help them best. >

⊗ Error

Cline tried to use `plan_mode_response` without value for required parameter 'response'. Retrying...

✓ API Request ▾

Reasoning: ...效地引导用户提供更多信息，进而制定合适的计划来帮助他们解决问题。 >

❓ Cline has a question:

您好！我是由 saoudrizwan.claude-dev 开发的 Cline，一个功能强大的 AI 软件工程师。请问有什么我可以帮您完成的任务或问题吗？我擅长编写代码、数据分析、系统操作、调试问题、自动化脚本编写、文档编写、以及各种问题解决。无论是简单的脚本编写还是复杂的系统集成，我都能为您提供帮助。请问您需要我协助什么？

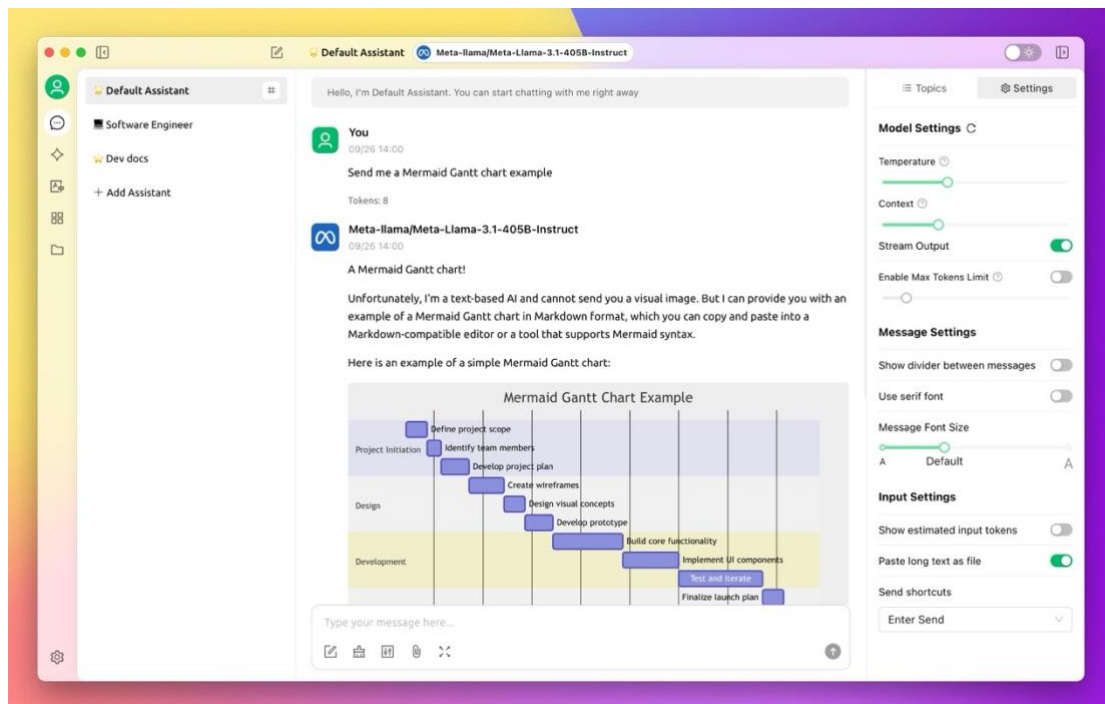
Auto-approve: None >

Approve Reject

Type a message... ➤

15 在 Cherry Studio 中使用 Hydra

[Cherry Studio](#) 是一个 LLM 桌面客户端，支持多 LLM 服务商集成，包括 OpenAI、GPT-3、Hydra 等。



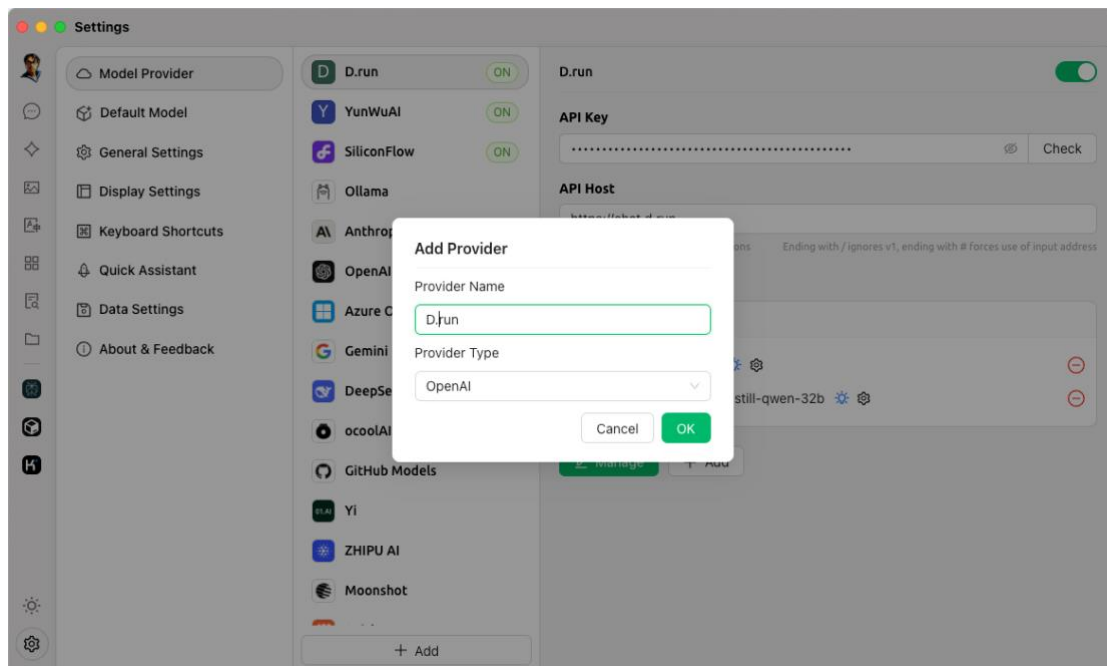
15.1 安装 Cherry Studio

您可以访问 [Cherry Studio 官网](#) 下载安装包。

支持 MacOS、Windows、Linux 三种客户端版本。

15.2 配置 Cherry Studio

打开 Cherry Studio 的配置页面，添加模型服务商，例如命名为 `d.run`，服务商类型为 `OpenAI`。

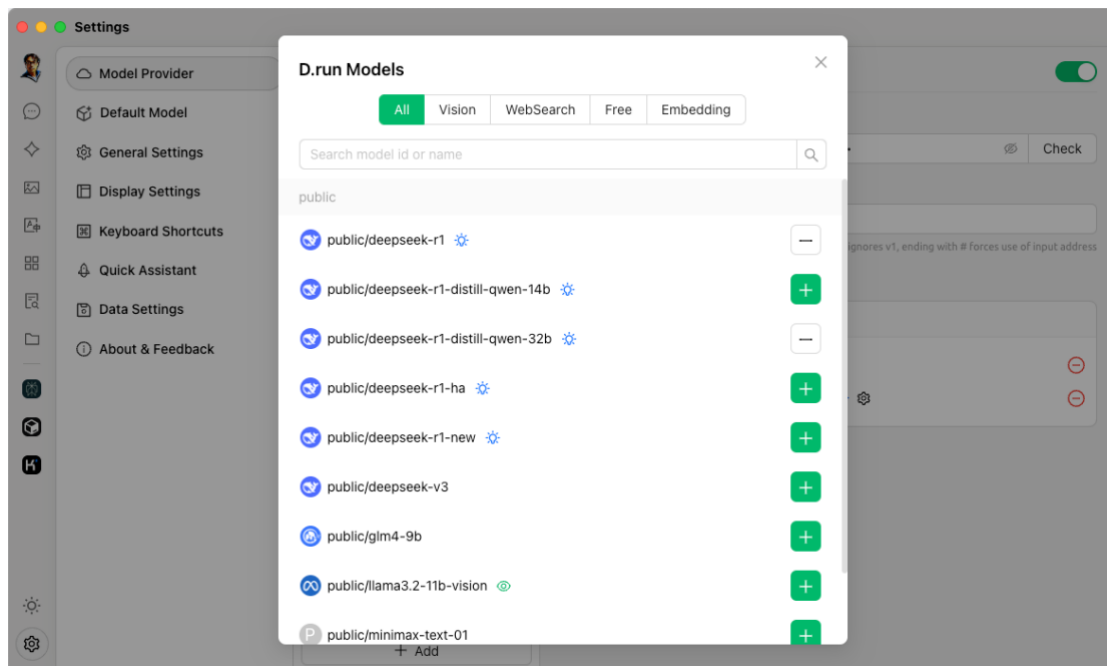


添加您从 d.run 获取的 API Key 和 API Host。

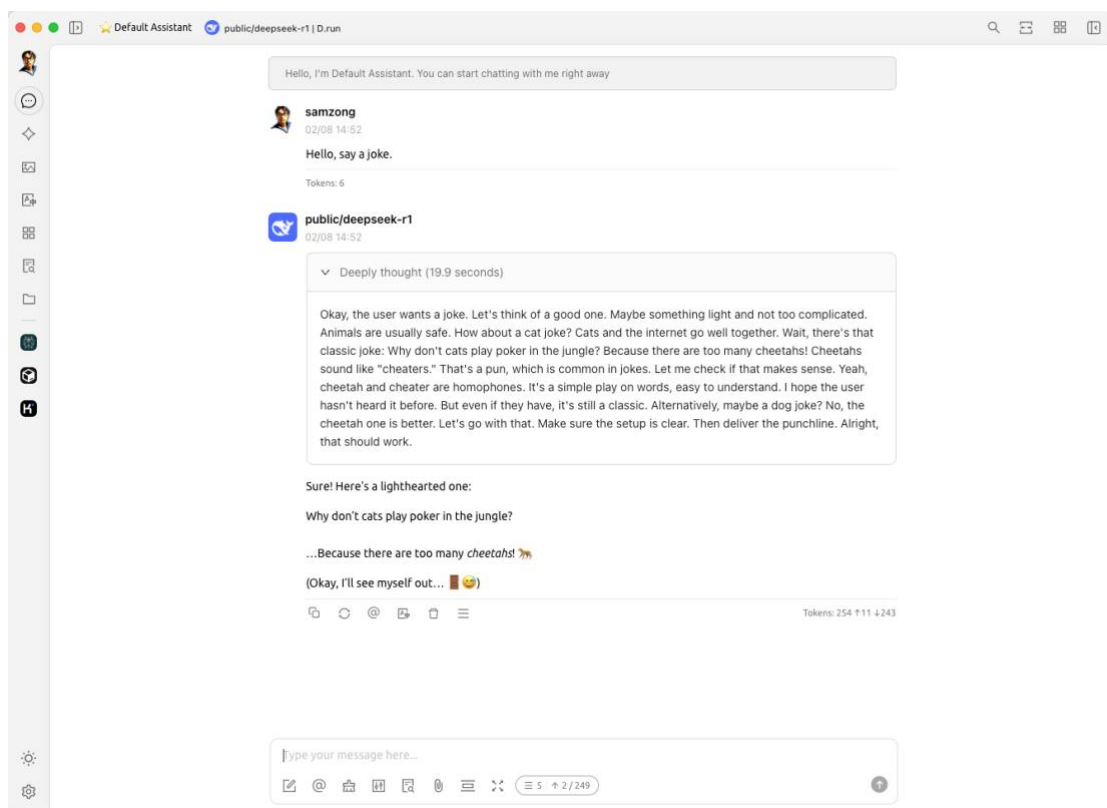
- API Key: 输入您的 API Key
- API Host:
 - MaaS 输入使用 `https://chat.d.run`
 - 独立部署的模型服务，查看模型实例详情，一般是 `https://<region>.d.run`

15.2.1 管理可用的模型

在 Cherry Studio 中，会自动检测模型列表，您可以在模型列表中启用所需要的模型。



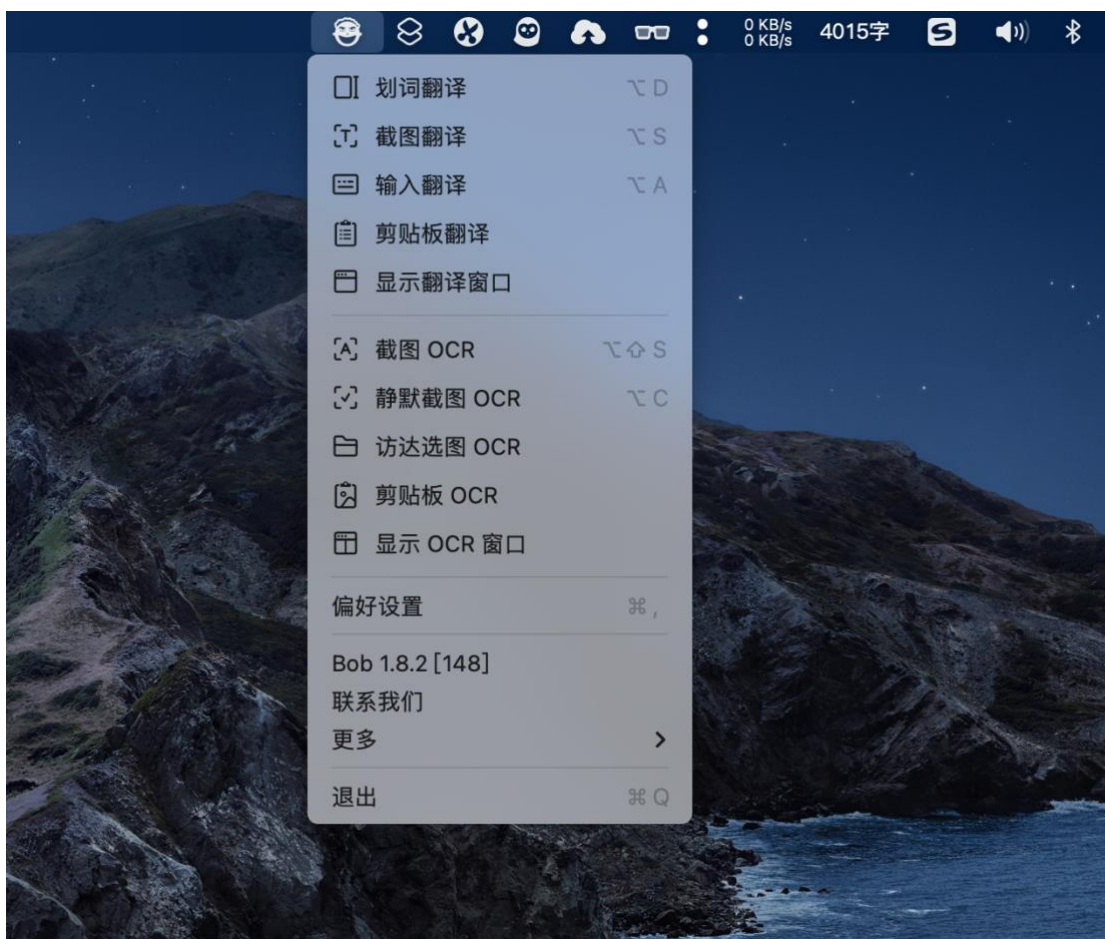
15.3 Cherry Studio 使用演示



16 在 Bob Translate 中使用 Hydra

本文说明如何在 Bob Translate 中调用 Hydra 中的模型服务。

Bob 是一款 macOS 平台的翻译和 OCR 软件，您可以在任何应用程序中使用 Bob 进行翻译和 OCR，即用即走，简单、快捷、高效！

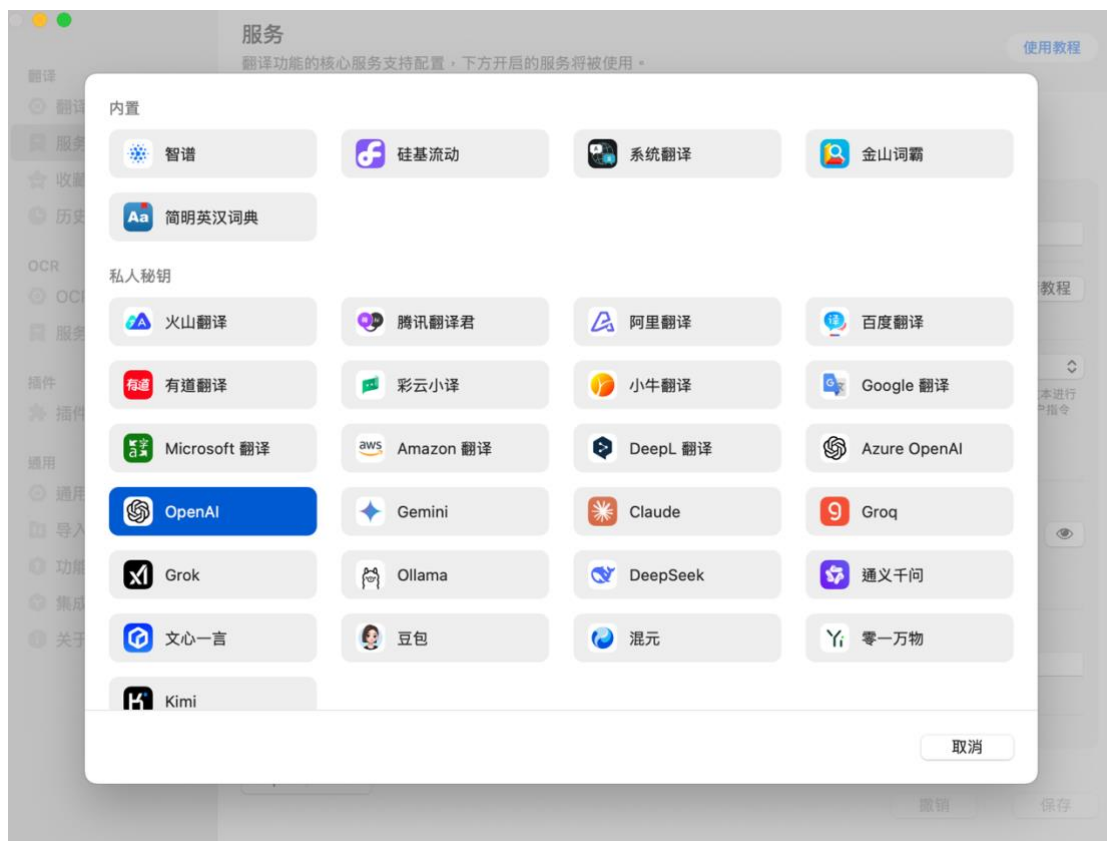


16.1 安装 Bob Translate

您可以在 [Mac App Store](#) 下载安装 Bob Translate。

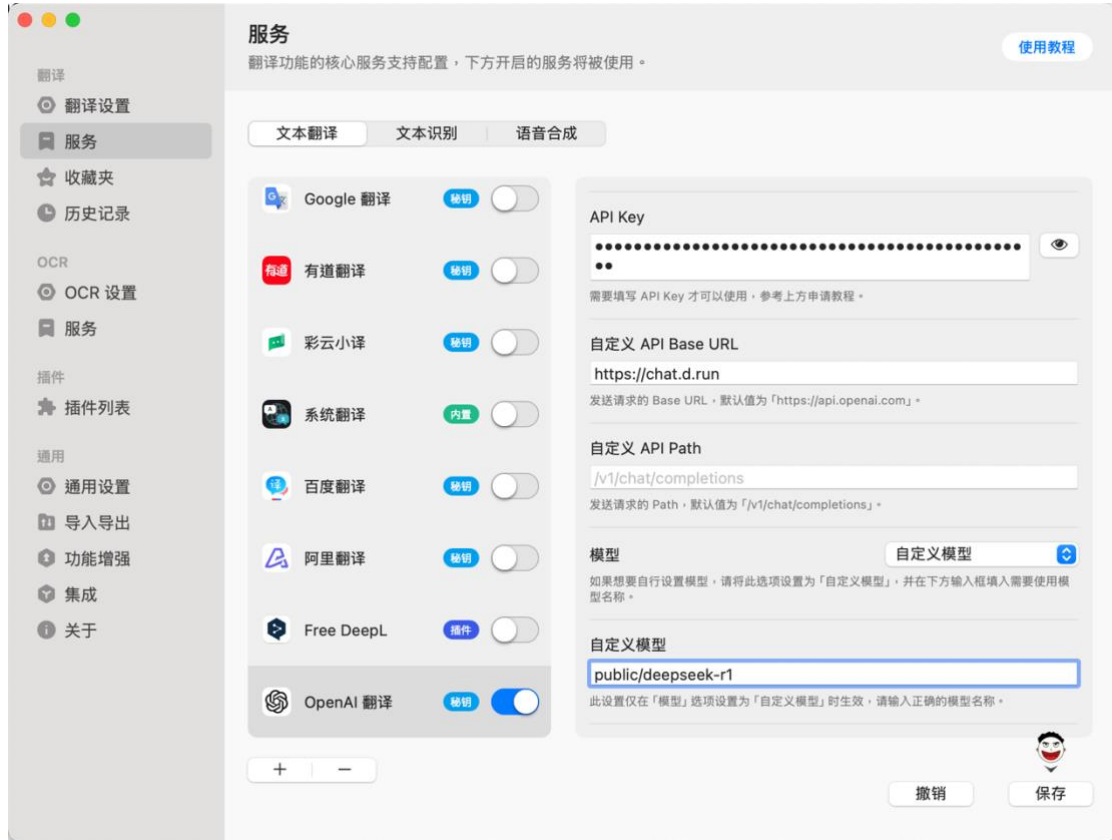
16.2 配置 Bob Translate

打开 Bob Translate 的配置页面，添加翻译服务，选择服务类型为 OpenAI。



添加您从 Hydra 获取的 API Key 和 API Host。

- API Key: 输入您的 API Key
- API Host:
 - MaaS 输入使用 `https://chat.d.run`
 - 独立部署的模型服务，查看模型实例详情，一般是 `https://<region>.d.run`
- 配置自定义模型：如 `public/deepseek-r1`



16.3 Bob Translate 使用演示



17 在 LobeChat 中使用 Hydra

Lobe Chat 是一个开源的现代设计 AI 聊天框架。支持多 AI 提供商（OpenAI/Claude 3/Gemini/Ollama/Qwen/DeepSeek）、知识库（文件上传/知识管理/RAG）、多模态（视觉/TTS/插件/艺术品）。一键免费部署您的私有 ChatGPT/Claude 应用。



17.1 安装 Lobe Chat

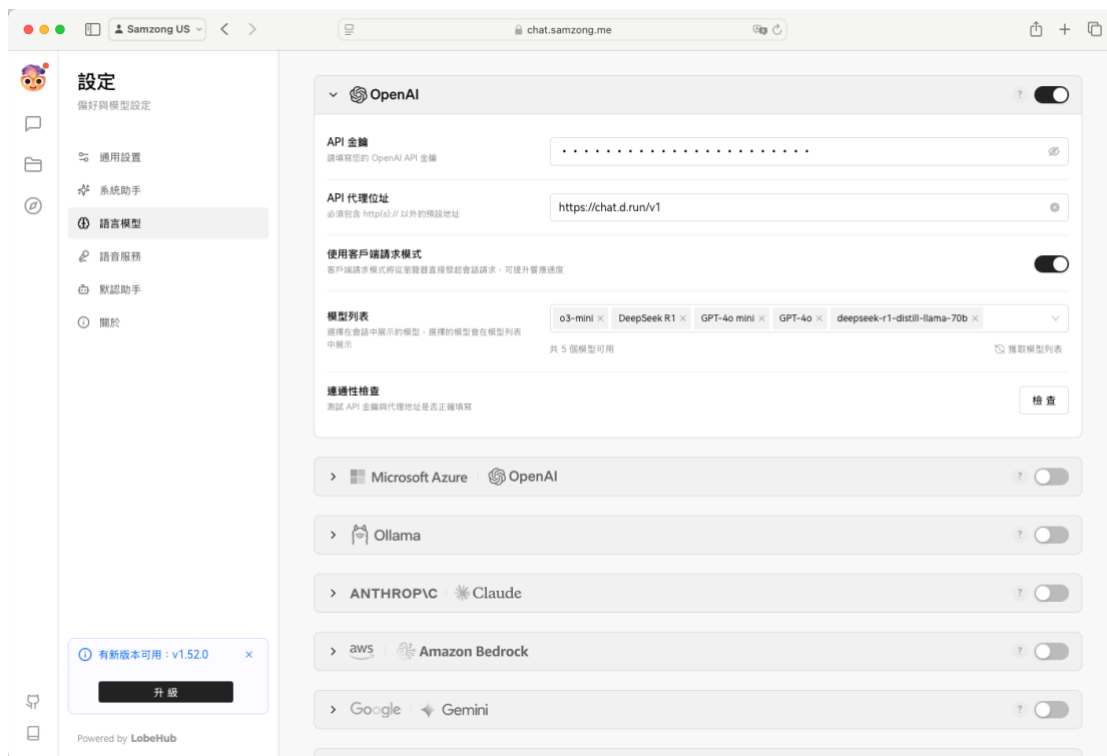
您可以参考 [Lobe Chat 官方文档](#) 下载并安装， Lobe Chat 提供了多种部署安装方式。

本文以 Docker 为例，主要介绍如何使用 Hydra 的模型服务。

```
# lobechat 支持在部署时直接配置 API Key 和 API Host
$ docker run -d -p 3210:3210 \
  -e OPENAI_API_KEY=sk-xxxx \ # 输入您的 API Key
  -e OPENAI_PROXY_URL=https://chat.d.run/v1 \ # 输入您的 API Host
  -e ENABLED_OLLAMA=0 \
  -e ACCESS_CODE=drun \
  --name lobe-chat \
  lobehub/lobe-chat:latest
```

17.2 配置 Lobe Chat

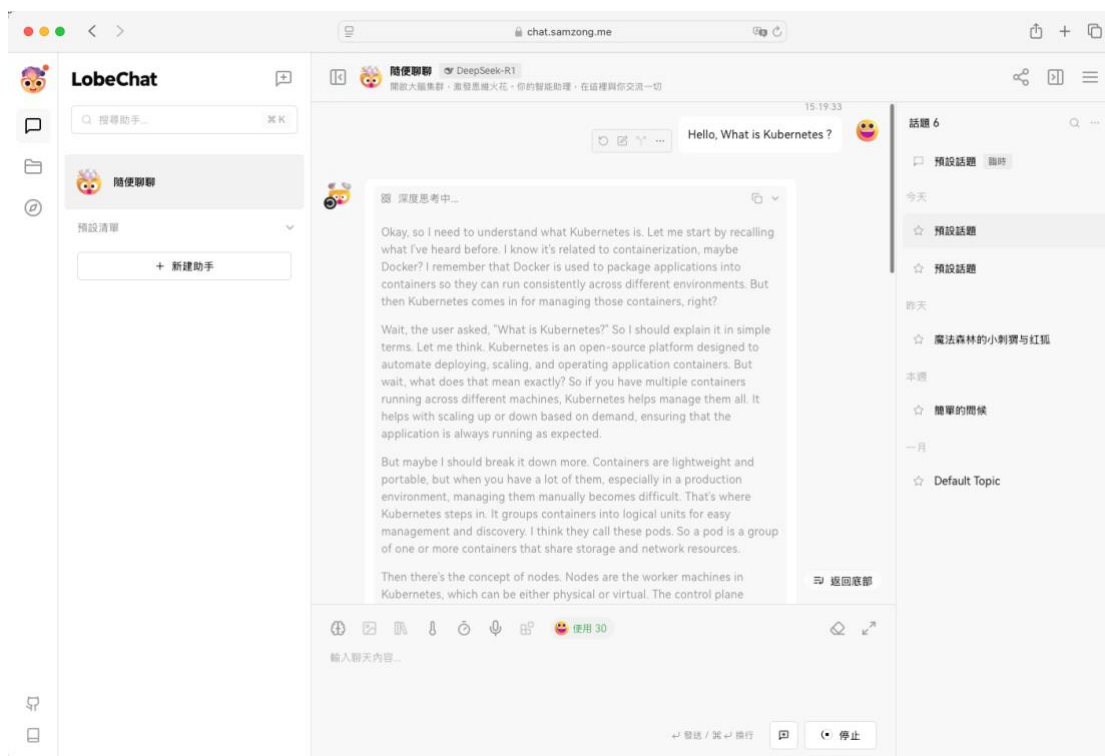
Lobe Chat 也支持在部署后，用户自行去添加模型服务商配置。



添加您从 Hydra 获取的 API Key 和 API Host。

- API Key: 输入您的 API Key
- API Host:
 - MaaS 输入使用 `https://chat.d.run`
 - 独立部署的模型服务，查看模型实例详情，一般是 `https://<region>.d.run`
 - 配置自定义模型：如 `public/deepseek-r1`

17.3 Lobe Chat 使用演示



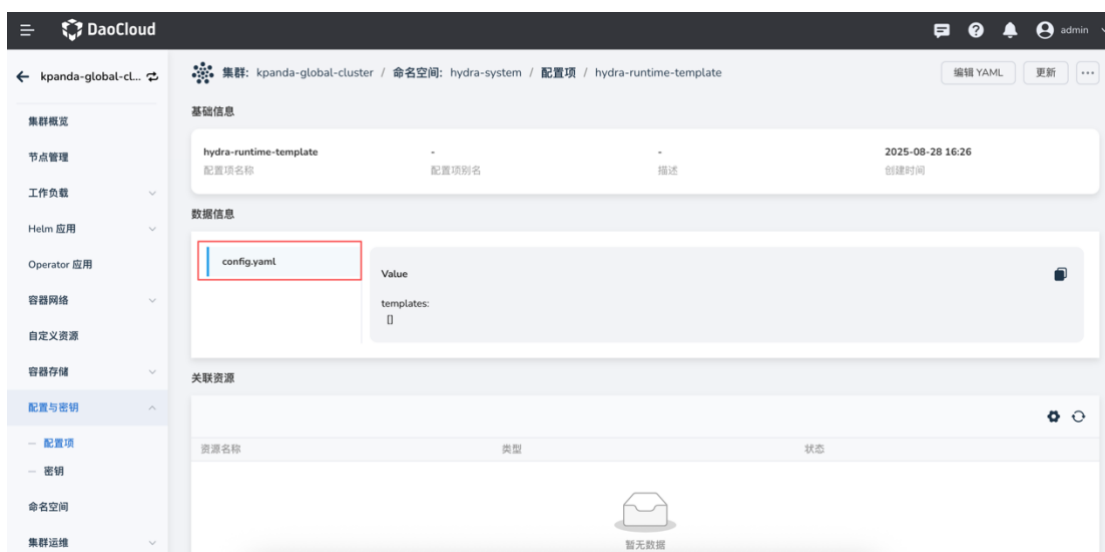
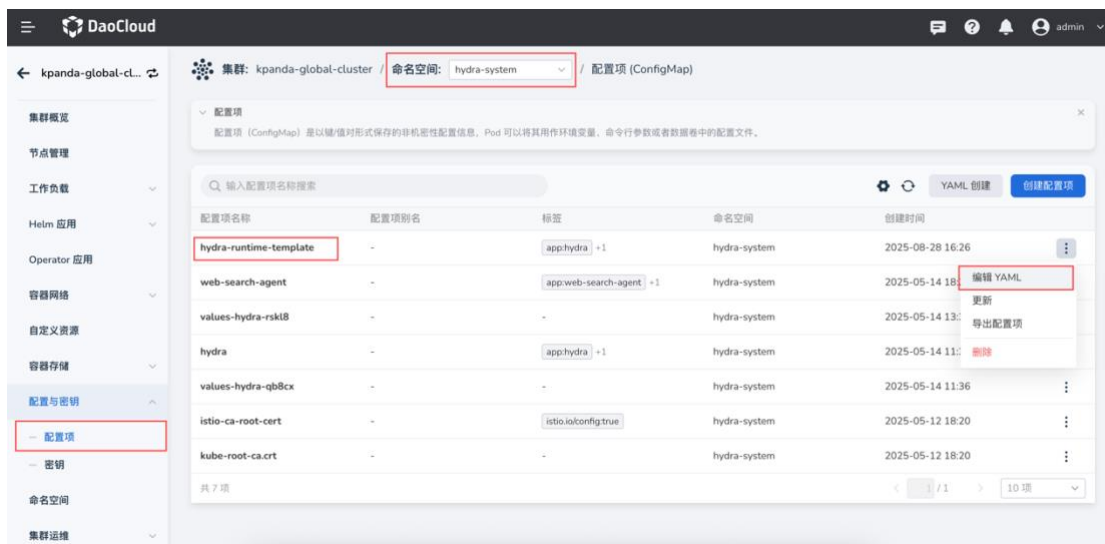
18 自定义大模型推理运行时

当前平台大模型推理服务支持 vLLM、SGLang 以及图像生成三种内置运行时。为了满足更多场景需求，提供自定义运行时的能力，用户可以根据自身业务需求，自由定义新的运行时类型，并配置其启动脚本、运行参数等相关信息，从而实现更灵活的推理服务部署。

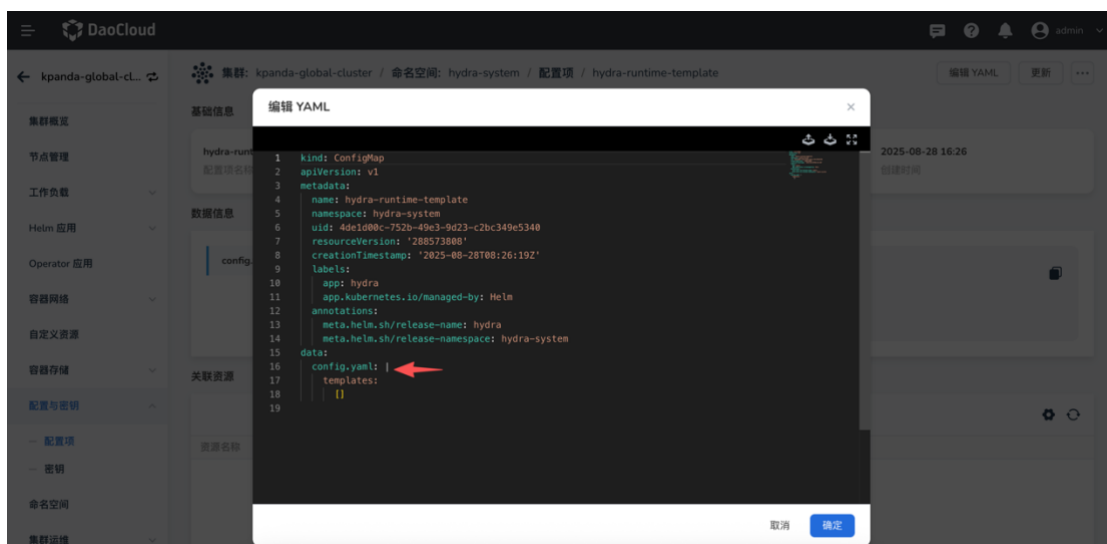
您可以参考下列操作步骤来配置并使用 Hydra 提供的自定义运行时功能。

18.1 全局服务集群中添加自定义运行时

1. 进入全局服务集群，点击 **配置与密钥** -> **配置项**，搜索并查找 `hydra-runtime-template`，其中 `config.yaml` 文件用于定义运行时模板。



2. 点击 **编辑 YAML**，修改 `config.yaml` 文件，添加自定义运行时。



config.yaml 对应的结构如下：

templates:

- runtime: string	# 运行时类型英文 (如: vllm), 必填
runtimeZH: string	# 运行时类型中文
podTemplate:	# Pod 模板定义
initContainers: []	# 初始化容器, 同 k8s 资源定义
podSecurityContext: {}	# Pod 安全上下文, 同 k8s 资源定义
volumes: []	# 卷定义, 同 k8s 资源定义
containerTemplate:	# 容器模板
commandTemplate:	# 启动命令模板, 数组, 使用 go template 语法
argsTemplate:	# 启动参数模板, 数组, 使用 go template 语法
volumeMounts: []	# 卷挂载, 同 k8s 资源定义
ports: []	# 端口, 同 k8s 资源定义
securityContext: {}	# 容器安全上下文, 同 k8s 资源定义

添加 vllm-cpu 运行时模版示例

templates:

- runtime: vllm-cpu
runtimeZH: vLLM
podTemplate:
containerTemplate:
commandTemplate:
- "/bin/bash"
- "-c"
argsTemplate:

```

-|-
  {{- if .IS_DISTRIBUTED -}}
  {{- if .IS_LEADER -}}
    ray start --head --port={{ .RAY_PORT }} && vllm serve
    {{ .MODEL_PATH }} --served-model-name {{ .MODEL_NAME }} --trust-remote-
    code --tensor-parallel-size={{ .TP_SIZE }} --pipeline-parallel-size={{ .PP_SIZE }}
  {{- else -}}
    ray start --block --
    address=$(LWS_LEADER_ADDRESS):{{ .RAY_PORT }}
  {{- end -}}
  {{- else -}}
    vllm serve {{ .MODEL_PATH }} --served-model-name
    {{ .MODEL_NAME }} --trust-remote-code {{- if gt .TP_SIZE 1 }} --tensor-parallel-
    size {{ .TP_SIZE }} {{- end -}}
  {{- end -}}
  {{- if .CUSTOM_ARGS -}} {{ range .CUSTOM_ARGS }} {{ . }} {{-
end -}} {{- end -}}

```

可用变量说明

模板中的变量请根据实际情况设置，可用变量包括：

- **RUN_TIME**：运行时类型，如 `vllm`, `image-gen`, `sglang`, `mindie` 等
- **MODEL_NAME**：模型名称，比如 `vllm` 的 `--served-model-name` 参数
- **IS_DISTRIBUTED**：是否分布式部署
- **MODEL_PATH**：模型路径，目前是固定的 `/data/serving-model`
- **IS_LEADER**：是否是分布式的 `leader` 节点
- **TP_SIZE**：张量并行数
- **PP_SIZE**：流水线并行数

- **MODEL_ID**: 模型 ID
- **CLUSTER**: 部署集群
- **NAMESPACE**: 部署的命名空间
- **MODEL_HOST**: 模型部署的服务地址, 目前是固定的 0.0.0.0
- **MODEL_PORT**: 模型部署的服务端口, 目前是固定的 8000

CUSTOM_ARGS 参数说明

为了减少配置复杂度,目前会将部署模版中配置的参数和 `hydra-agent` 中配置的参数, 单独作为一个 `CUSTOM_ARGS` 的变量传入模板中, 建议使用下面的方式:

- 在 `commandTemplate` 中只定义启动命令, 例如 `"/bin/bash -c"`, 它会被渲染成 `container` 的 `command`。在 `argsTemplate` 的最后添加 `{{- if .CUSTOM_ARGS -}} {{ range .CUSTOM_ARGS }} {{.}} {{- end -}} {{- end -}}`, 以此来保证自定义的参数也可以被正确渲染。
- 如果不定义 `template`, 则使用 `model_deployment` 和 `hydra-agent` 中配置的参数做为 `container` 的参数。
- 如果没有定义 `commandTemplate`, 则需要注意 `argsTemplate` 中应该是多行, 而不是单行的 `args`, 并

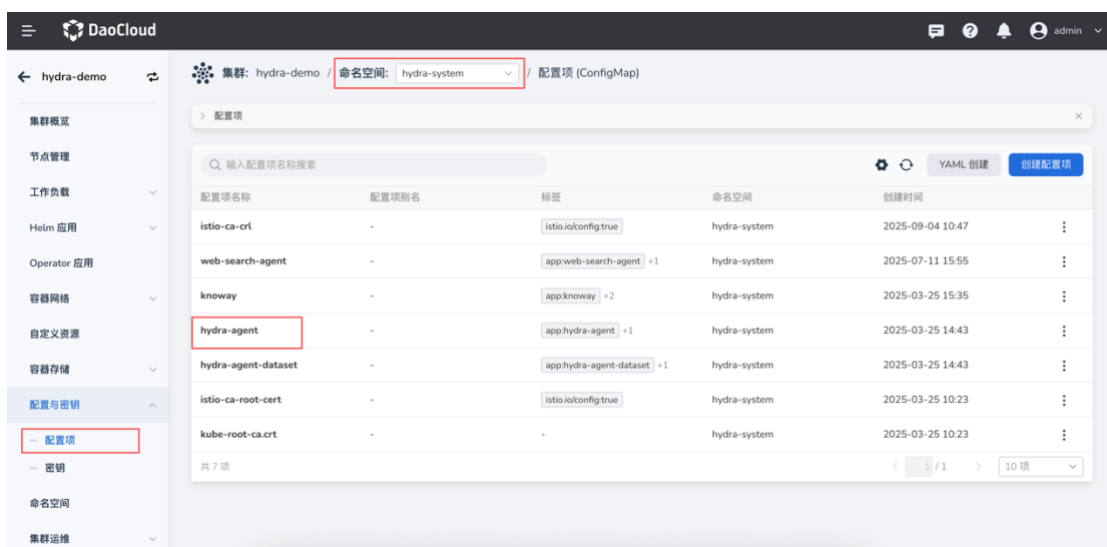
且 `args` 之间使用 **等号** 连接，如 --

```
model={{ .MODEL_PATH }}
```

- `command` 采用了 `/bin/bash -c` 的方式，因为这种方式要求 `args` 是单行的字符串，因此需要将 `CUSTOM_ARGS` 添加到模版中。
- 没有指定 `command`，`args` 是多行的数组，这种方式不需要将 `CUSTOM_ARGS` 添加到模版中，我们会自动将 `CUSTOM_ARGS` 添加到最终渲染出的 `args` 中，避免解析 `args` 出错。

18.2 工作集群中添加镜像信息

1. 进入工作集群，点击 **配置与密钥** -> **配置项**，搜索并查找 `hydra-agent`，在 `configmap/deployment_templates` 中添加自定义运行时的镜像信息和算力匹配信息。



2. 点击**编辑 YAML**，修改 configmap 文件，添加自定义运行时的镜像信息。

YAML 示例

deployment_templates:

- match_runtimes: [vllm-cpu] # 自定义运行时名称

match_gpu_types: [cpu] # 运行环境算力

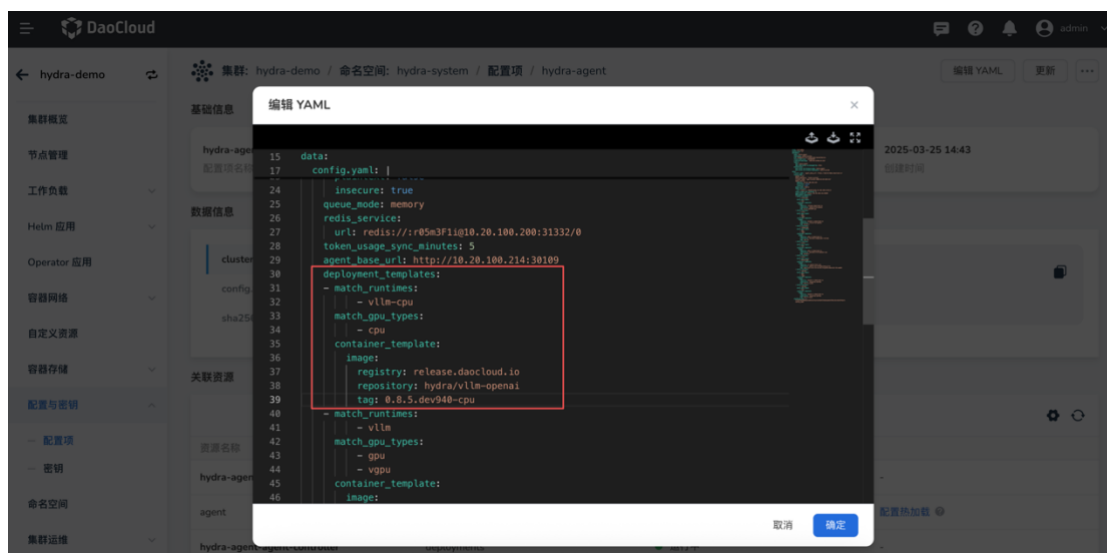
container_template:

image: # 自定义运行时镜像

registry: swr.cn-south-1.myhuaweicloud.com

repository: ascendhub/mindie

tag: 2.1.RC1



18.3 开始使用

完成以上操作后，运维管理员可以在部署配置中选择自定义运行时，并配置模型部署信息。

