

# DCE 5.0 之 AI Lab

## 用户手册

本文内容如有变更，恕不另行通知，请以官网内容为准：

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

DaoCloud 研发部门

2024 年 12 月 31 日

|                                |     |
|--------------------------------|-----|
| 组件部署 .....                     | 4   |
| 快速体验 .....                     | 8   |
| 开发控制台 .....                    | 10  |
| 快速入门 .....                     | 11  |
| 创建 Notebook .....              | 19  |
| 启动和暂停 Notebook .....           | 21  |
| Notebook 工作负载 .....            | 21  |
| 删除 Notebook .....              | 23  |
| Notebook SSH 访问指南 .....        | 24  |
| 在 Notebook 中使用环境 .....         | 29  |
| baizectl 命令行工具使用指南 .....       | 33  |
| baizess 换源工具使用指南 .....         | 49  |
| Notebook 闲置超时自动关机 .....        | 50  |
| 创建任务 (Job) .....               | 53  |
| Pytorch 任务 .....               | 55  |
| Tensorflow 任务 .....            | 61  |
| MPI 任务 .....                   | 69  |
| MXNet 任务 .....                 | 76  |
| PaddlePaddle 任务 .....          | 85  |
| 删除任务 (Job) .....               | 92  |
| 查看任务 (Job) 工作负载 .....          | 93  |
| 任务分析介绍 .....                   | 99  |
| 数据集列表 .....                    | 104 |
| 管理环境 .....                     | 109 |
| 模型支持情况 .....                   | 114 |
| 创建 Triton 推理服务 .....           | 116 |
| 创建 vLLM 推理服务 .....             | 121 |
| 运维管理 .....                     | 124 |
| GPU 列表 .....                   | 124 |
| 创建队列 .....                     | 125 |
| 最佳实践 .....                     | 126 |
| 部署 NFS 做数据集预热 .....            | 126 |
| 更新 Notebook 内置镜像 .....         | 131 |
| Checkpoint 机制及使用介绍 .....       | 137 |
| 使用 AI Lab 微调 ChatGLM3 模型 ..... | 142 |
| 如何提交 DeepSpeed 训练任务 .....      | 159 |
| 增加任务调度器 .....                  | 160 |
| 部署 Label Studio .....          | 164 |
| 故障排查 .....                     | 170 |
| 集群下拉列表中找不到集群 .....             | 170 |
| Notebook 不受队列配额控制 .....        | 172 |
| 本地队列初始化失败 .....                | 174 |

AI Lab 是 DaoCloud 推出的基于云原生操作系统的 AI 算力平台（曾用名智能算力），AI Lab 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  
支持模型开发过程中数据管理与编排能力，提供如数据集管理、多数据源接入、数据集预热等功能，从底层容器存储引擎进行优化，保证数据的高效与稳定。
- 开发环境管理  
满足 MLOps 和 LLMOps 工程师对开发环境的需求，提供多种开发环境，包括 JupyterLab、VSCode(进行中) 等，支持自定义开发环境，一键挂载各种 GPU、数据集等资源。
- 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。
- GPU 管理  
可以查看全部的 GPU 资源和 GPU 使用情况，支持 GPU 中当前和历史运行的任务情况查看，方便进行 GPU 压力评估。
- 队列管理  
支持创建队列，并将队列与工作空间进行关联，保障在各个集群中的队列资源的统筹与隔离。

## 产品逻辑架构



## 组件部署

在 DCE 5.0 的安装器 v0.17.0 之后商业版安装时可以同步安装 AI Lab 模块，无需自行安装；请联系交付支持团队获取商业版安装包。

## 全局服务集群

AI Lab 模块仅需安装在

打开全局服务集群，然后在 **Helm** 应用 -> **Helm** 模板 找到 baize 执行安装步骤。

### 注意事项

- 命名空间为 baize-system
- 替换环境地址后打开 <YOUR\_DCE\_HOST>/kpanda/clusters/kpanda-global-cluster/helm/charts/addon/baize
- kpanda-global-cluster 是全局服务集群名称

## baize-agent 工作集群

### Warning

如果在 AI Lab 中对应的集群无法选择或提示缺少 baize-agent，也就是该工作集群的组件并未成功部署。

在每个有算力资源的工作集群内，需要部署对应的基础组件，主要组件包含：

- gpu-operator 初始化集群中的 GPU 资源，这部分会因 **GPU** 资源类型安装方式不同，详情参考 [GPU 管理](#)
- insight-agent 可观测组件，用于采集集群的基础设施信息，包含日志、指标、事件
- baize-agent 包含了 AI Lab 的核心组件，调度、监控、Pytorch、Tensorflow 等算力组件
- 【可选】nfs 存储服务，用于数据集的预热

## Danger

以上组件必须安装，否则会导致功能使用不正常。

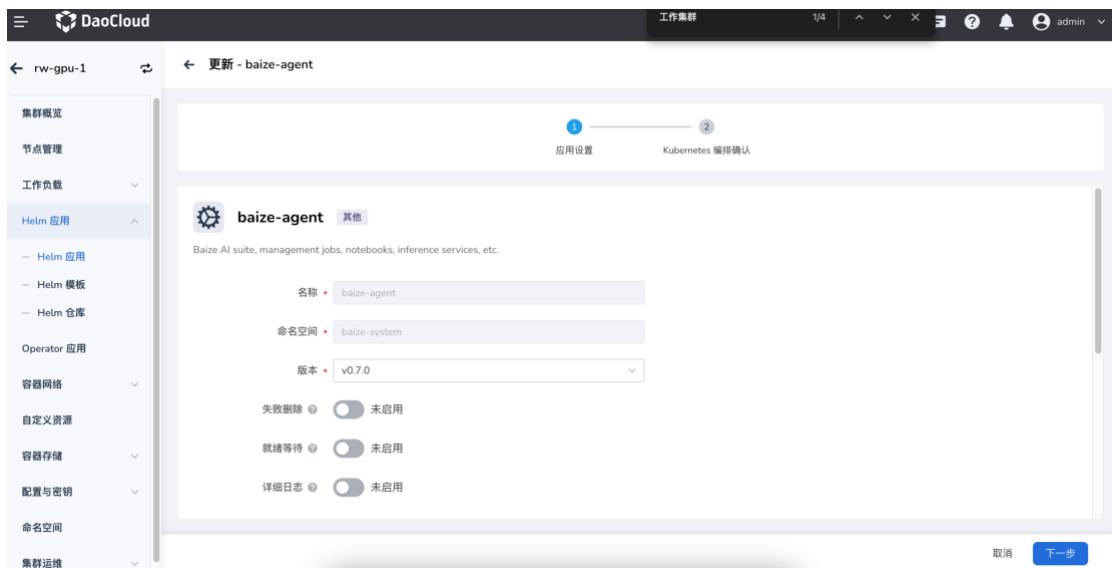
## 界面化安装 baize-agent

baize-agent 需要在工作集群部署。

按照下方提示，进入工作集群，然后在 **Helm** 应用 -> **Helm 模板** 找到 baize-agent 执行安装步骤。

### 注意事项

- 命名空间为 baize-system
- 替换环境地址后打开 `<YOUR_DCE_HOST>/kpanda/clusters/<cluster_name>/helm/charts/addon/baize`
- cluster\_name 是对应工作集群的名称



## YAML 示例：

```
cluster-controller:
  image:
    registry: ""
    repository: baize/baize-cluster-controller
    tag: v0.4.1
  global:
    cluster:
      schedulers: []
    config:
      cluster_name: ""
      dataset_job_spec: {}
      inference_config:
        triton_image: m.daocloud.io/nvcr.io/nvidia/tritonserver:24.01-py3
        triton_images_map:
          VLLM: m.daocloud.io/nvcr.io/nvidia/tritonserver:24.01-vllm-python-py3
    debug: false
    high_available: false
    imagePullPolicy: IfNotPresent
    imagePullSecrets: []
    imageRegistry: release.daocloud.io
    prod: baize-agent
    resources: {}
  kubeRbacProxy:
    image:
      registry: ""
      repository: baize/kube-rbac-proxy
      tag: v0.8.0
  kueue:
    enablePlainPod: false
    fullnameOverride: kueue
    image:
      registry: ""
      repository: baize/kueue
      tag: v0.6.2
  loader:
    image:
      registry: ""
      repository: baize/baize-data-loader
      tag: v0.4.1
  notebook:
    image:
      registry: "
```

```
repository: baize/baize-notebook
tag: v0.4.1
notebook-controller:
  image:
    registry: ""
    repository: baize/notebook-controller
    tag: v1.8.0
priority:
  high:
    value: 100000
  low:
    value: 1000
  medium:
    value: 10000
training-operator:
  image:
    registry: ""
    repository: baize/training-operator
    tag: v1-5525468
```

## Helm 安装 baize-agent

确保全局服务集群内已经安装了 AI Lab 组件，可以通过在管理界面查看是否有 AI Lab 模块。

### Info

需要在一级导航栏有 AI Lab 入口，保障管理组件部署成功。

```
# baize 是 AI Lab 组件的开发代号
helm repo add baize https://release.daocloud.io/chartrepo/baize
helm repo update baize
helm search repo baize # 获取最新的版本编号
export VERSION=<version> # 注意使用当前最新版本
helm upgrade --install baize-agent baize/baize-agent \
  --create-namespace \
  -n baize-system \
  --set global.imageRegistry=release.daocloud.io \
  --version=${VERSION}
```

以上工作完成后，工作集群初始就成功了，可以在 AI Lab 模块，进行任务训练和模型开发。

## 预热组件介绍

AI Lab 模块提供的数据管理中，数据集的预热能力依赖存储服务，推荐使用 NFS 服务：

- 部署 NFS Server
  - 如果已存在 NFS 可以跳过此步骤
  - 如果不存在，可以参考最佳实践中的 [NFS 服务部署](#)
- 部署 nfs-driver-csi
- 部署 StorageClass

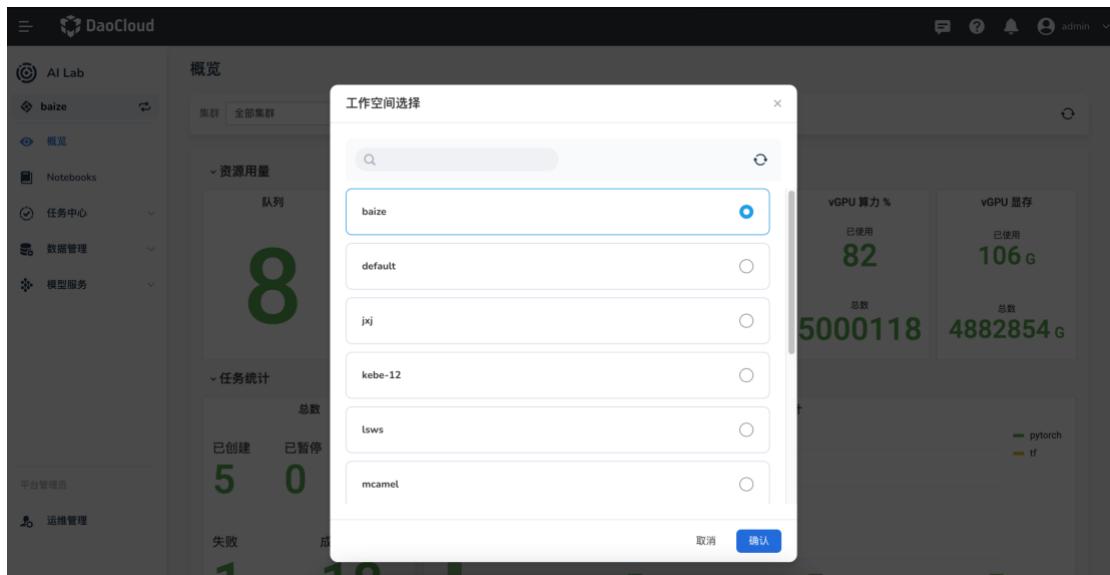
## 快速体验

第一次进入 DCE 5.0 AI Lab 时，需要：

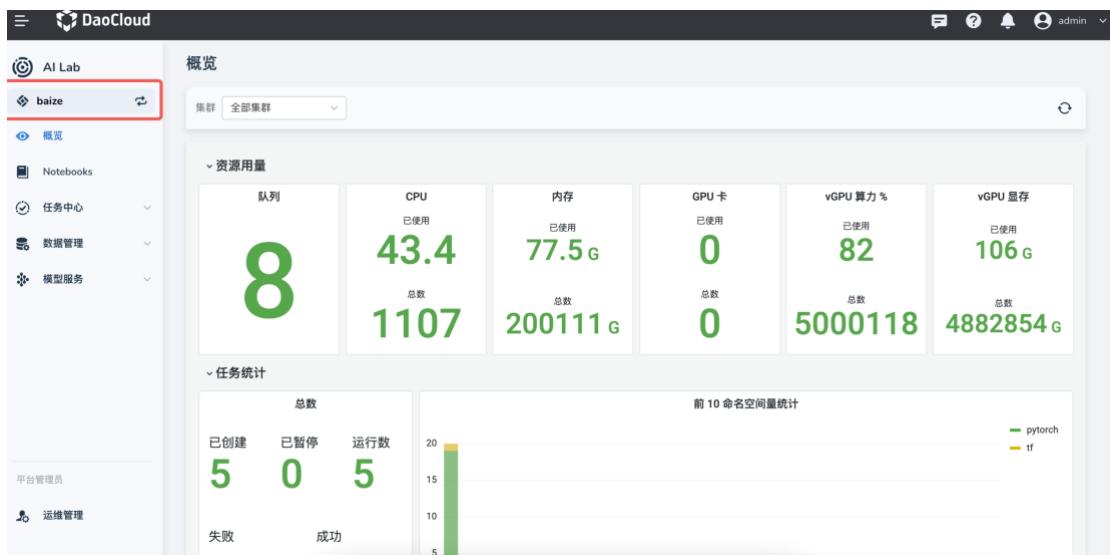
- 选择一个工作空间
- 选择一个集群
- 确定自己的角色

### 选择工作空间

首次进入 DCE 5.0 AI Lab 时，首先必须选择一个工作空间。



如需更改当前所在的工作空间，可以在左侧边栏点击更换图标重新选择工作空间。

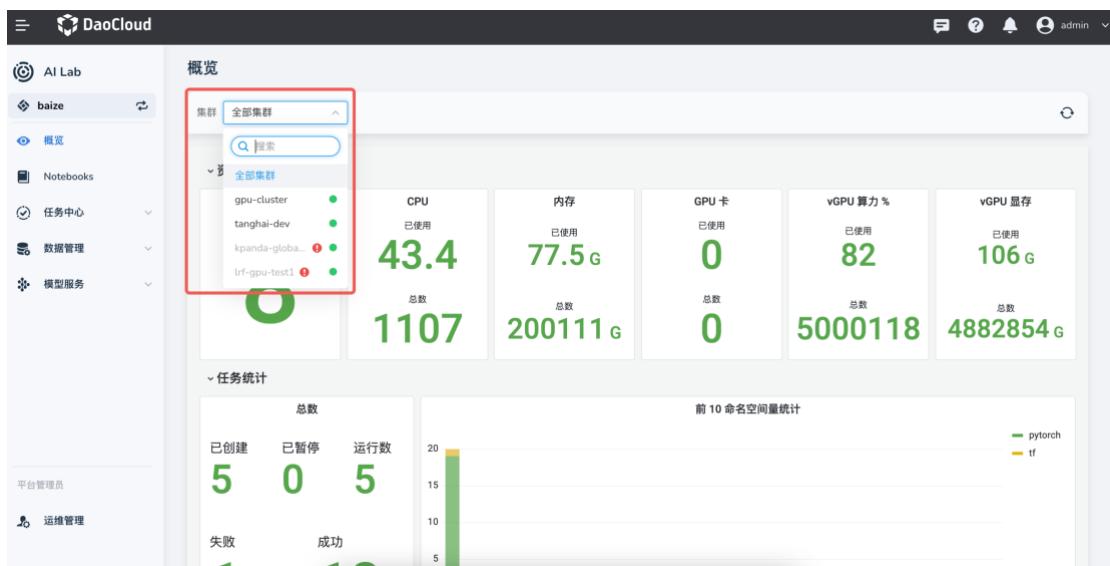


## Note

如果当前没有可选的工作空间，需要先联系管理员创建一个工作空间。

## 选择集群

您可以选择在哪个集群部署和执行 AI Lab 相关的操作。



如果您想要的集群不在集群列表中，需要先去绑定集群和命名空间。

1. 点击左上角的图标，打开导航栏，点击 全局管理 -> 工作空间与层级
2. 从列表中找到你的工作空间（例如 baize），点击 资源组 页签，点击右侧的 绑定资源 按钮。

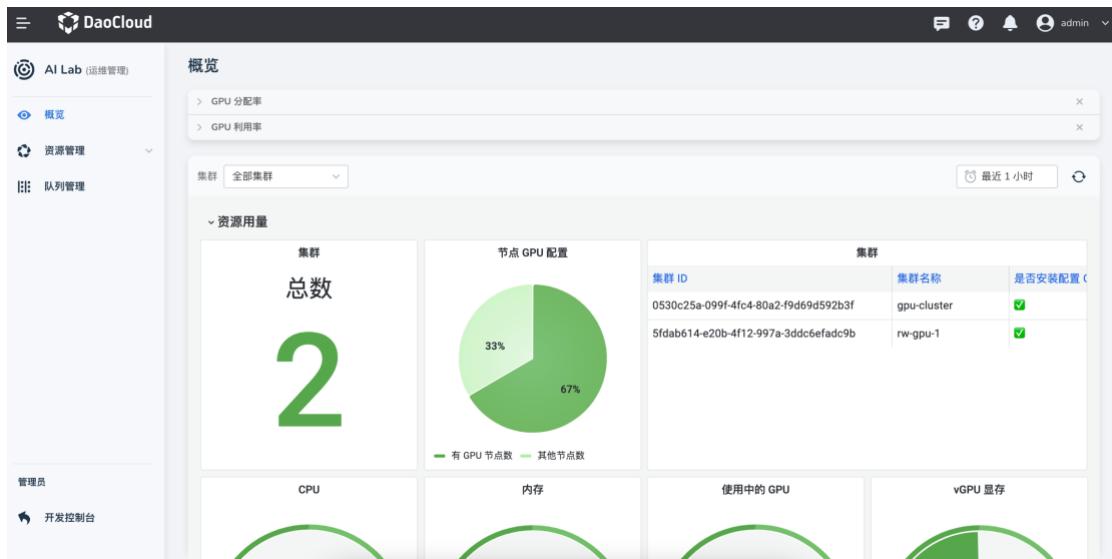
3. 勾选要绑定的命名空间、集群后，点击 **确定**。
4. 屏幕提示绑定成功，被绑定的集群和命名空间将显示在列表中。您还可以在这个页面授权给某个用户，添加更多资源组，创建共享资源等。

## 角色

DCE 5.0 AI Lab 提供了两种角色，可以点击左下角的菜单项切换两种管理员角色：

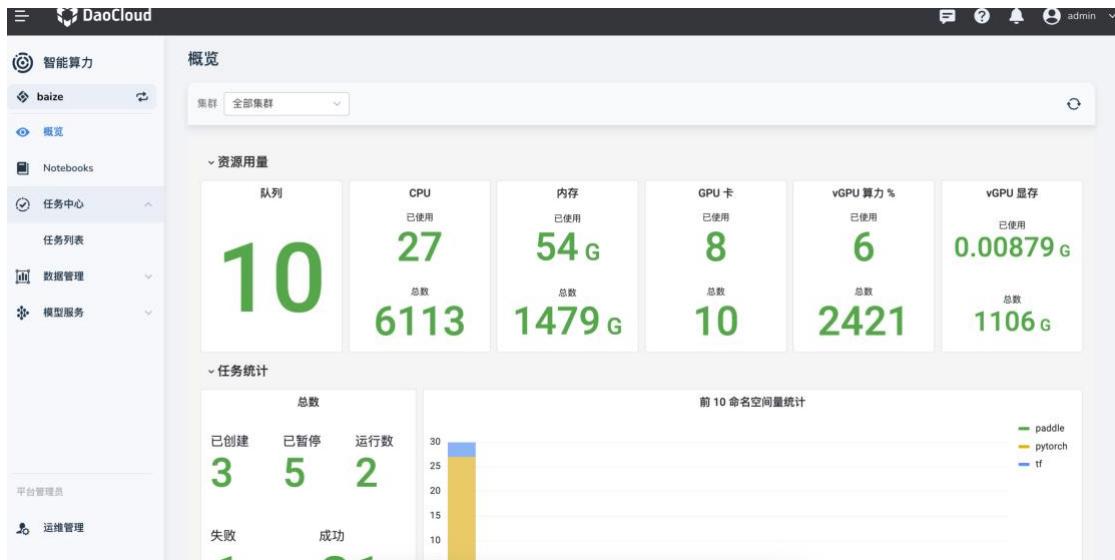
- 管理员 - 开发控制台：可以处理 Notebook、训练任务和数据集等。
- 平台管理员 - 运维管理：可以管理 GPU 资源、队列等

每个角色都有一个概览页面，通过图形仪表展示当前可以处理的数据。



## 开发控制台

开发控制台是开发者日常执行 AI 推理、大模型训练等任务的控制台。



方便用户通过概览快速了解，当前工作空间的资源及用量情况，包含了 GPU 资源、Notebook、任务以及数据集的数量信息。

## 快速入门

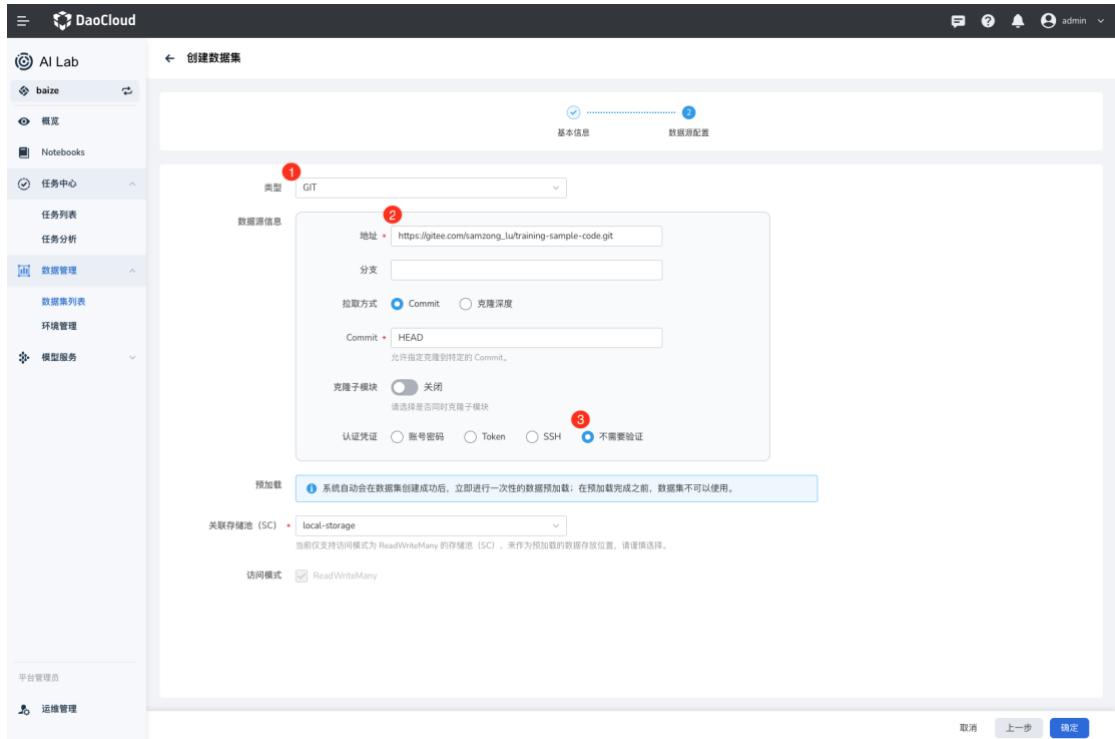
本文提供了简单的操作手册以便用户使用 DCE 5.0 AI Lab 进行数据集、Notebook、任务训练的整个开发、训练流程。

## 准备数据集

点击 **数据管理 -> 数据集**，选择 **创建** 按钮，分别创建以下三个数据集。

### 数据集：训练代码

- 代码数据源：<https://github.com/samzong/training-sample-code.git>，主要是一个简单的 Tensorflow 代码。
- 如果是中国境内的用户，可以使用 Gitee 加速：[https://gitee.com/samzong\\_lu/training-sample-code.git](https://gitee.com/samzong_lu/training-sample-code.git)
- 代码路径为 tensorflow/tf-fashion-mnist-sample



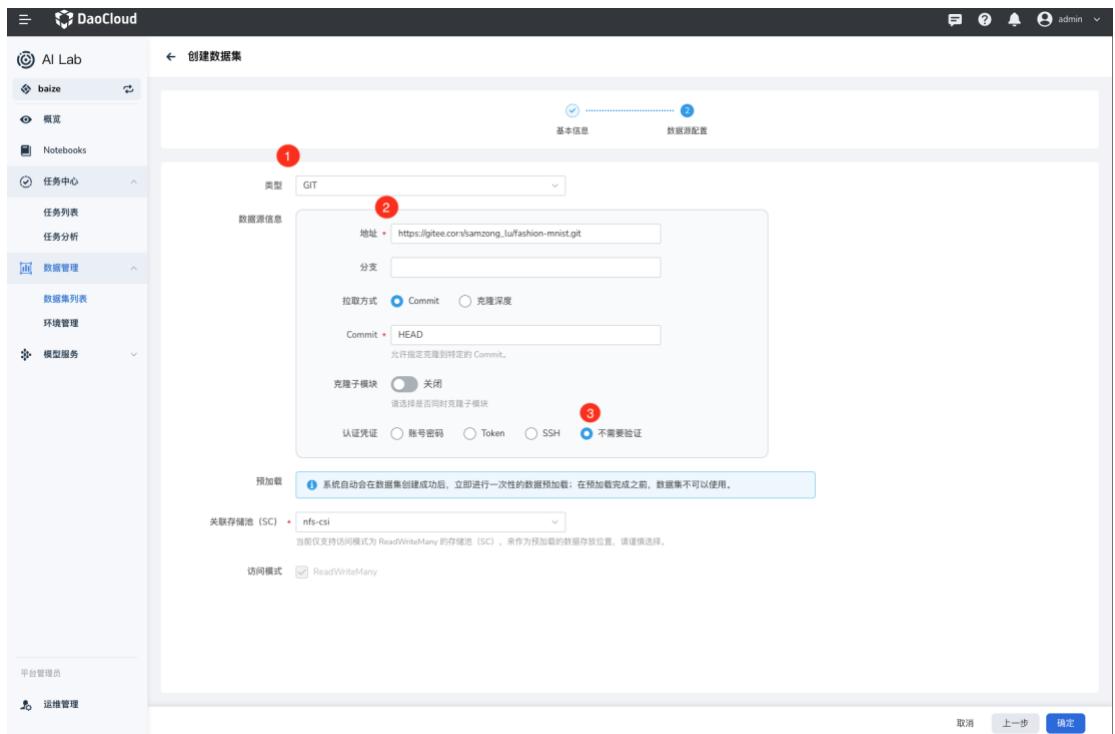
## Note

目前仅支持读写模式为 `ReadWriteMany` 的 `StorageClass`，请使用 `NFS` 或者推荐的 [JuiceFS](#)。

## 数据集：训练数据

本次训练使用的数据为 <https://github.com/zalandoresearch/fashion-mnist.git>，这是 `Fashion-MNIST` 数据集。

如果是中国境内的用户，可以使用 `Gitee` 加速：[https://gitee.com/samzong\\_lu/fashion-mnist.git](https://gitee.com/samzong_lu/fashion-mnist.git)

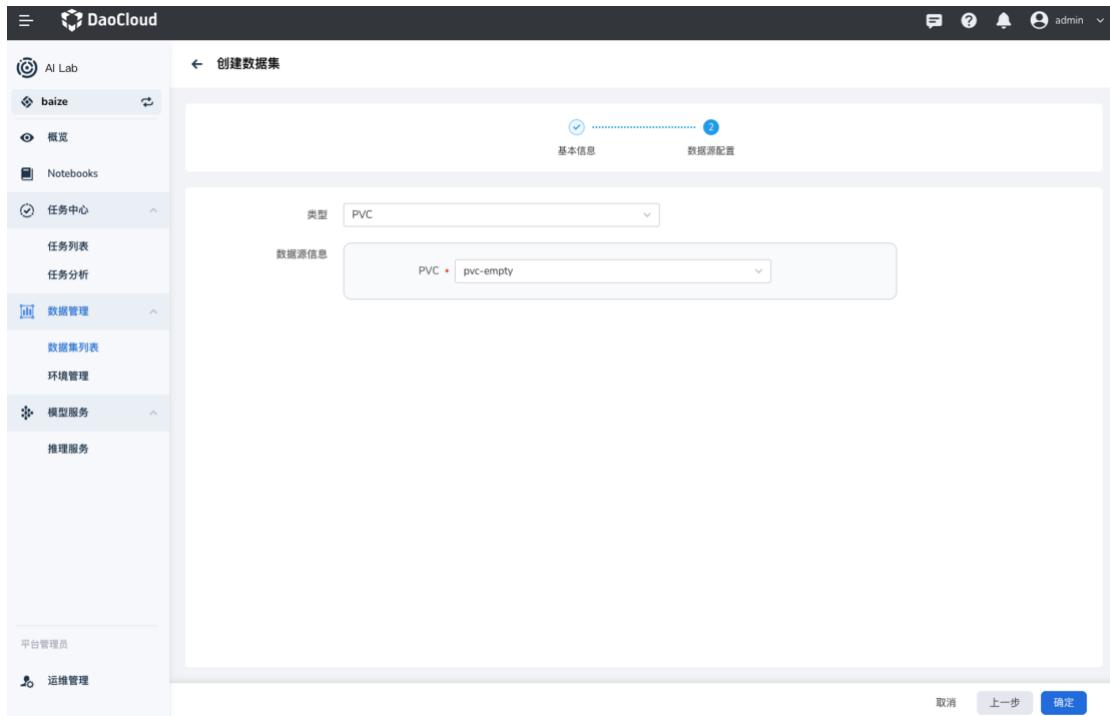


## Note

如果未创建训练数据的数据集，通过训练脚本也会自动下载；提前准备训练数据可以提高训练速度。

## 数据集：空数据集

AI Lab 支持将 PVC 作为数据集的数据源类型，所以创建一个空 PVC 绑定到数据集后，可将空数据集作为存放后续训练任务的输出数据集，存放模型和日志。

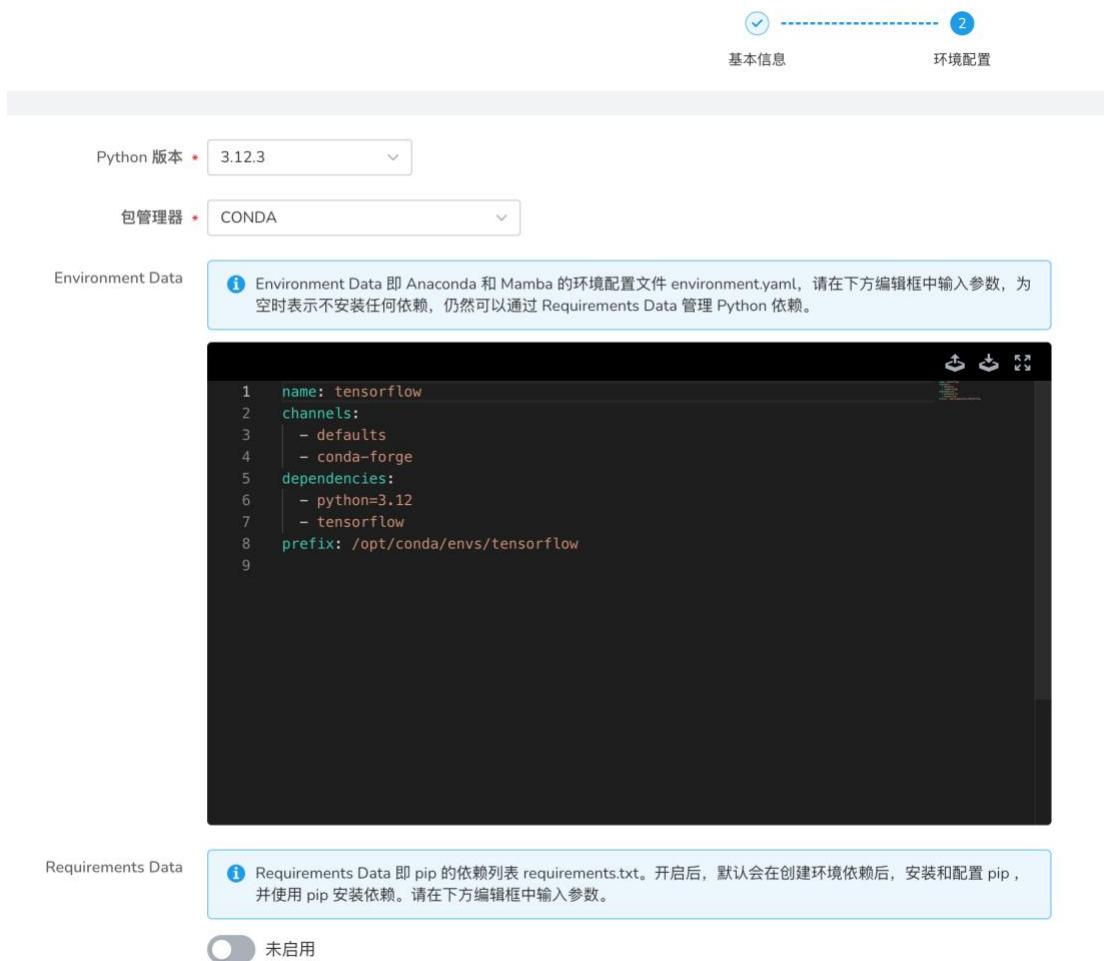


## 环境依赖: Tensorflow

脚本在运行时，需要依赖 Tensorflow 的 Python 库，可以使用 AI Lab 的环境依赖管理功能， 提前将需要的 Python 库下载和准备完成，无需依赖镜像构建。

参考[环境依赖](#)的操作方式，添加一个 CONDA 环境。

```
name: tensorflow
channels:
  - defaults
  - conda-forge
dependencies:
  - python=3.12
  - tensorflow
prefix: /opt/conda/envs/tensorflow
```



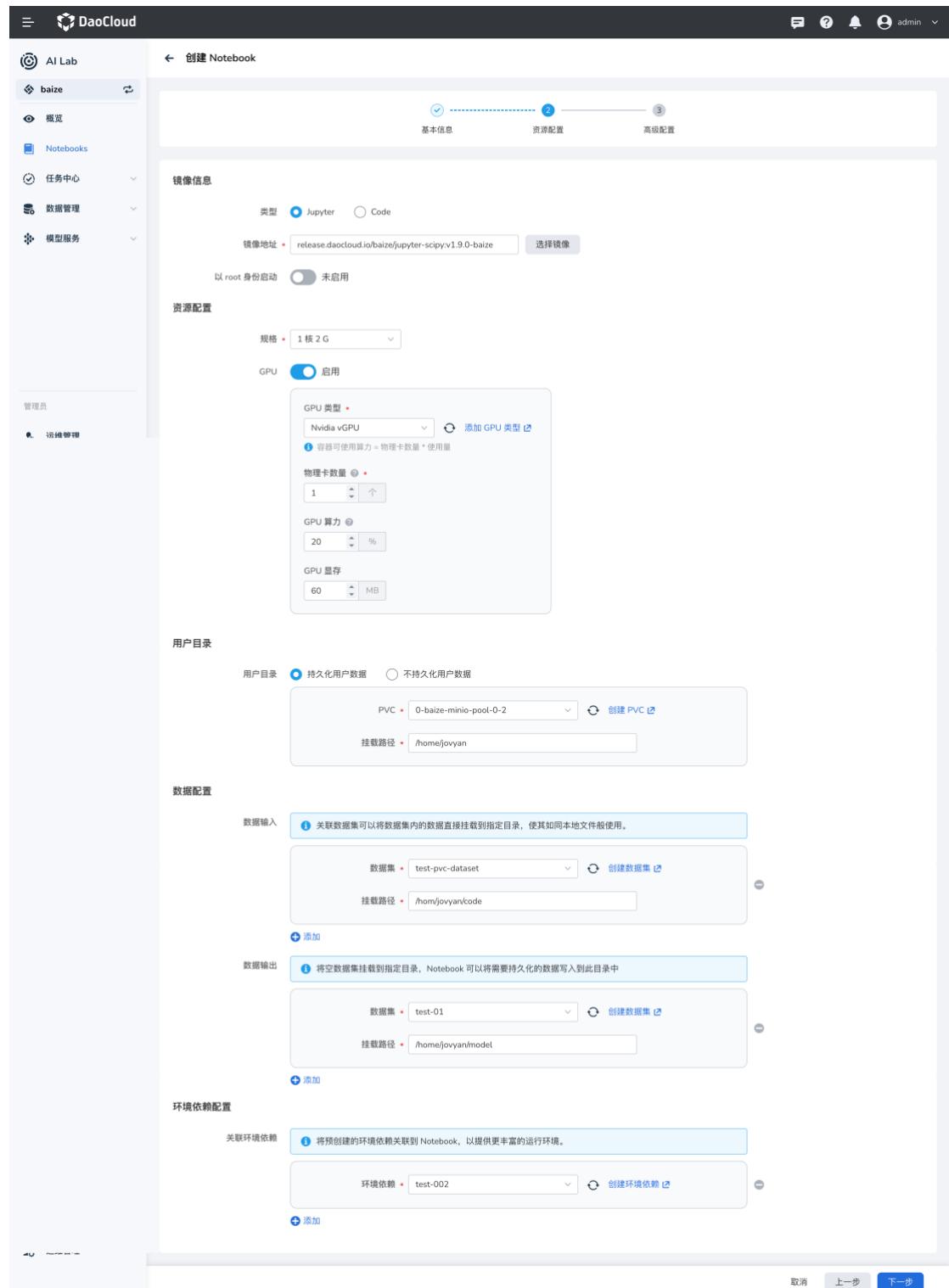
## Note

等待环境预热成功后, 只需要将此环境挂载到 Notebook、训练任务中, 使用 AI Lab 提供的基础镜像

## 使用 Notebook 调试脚本

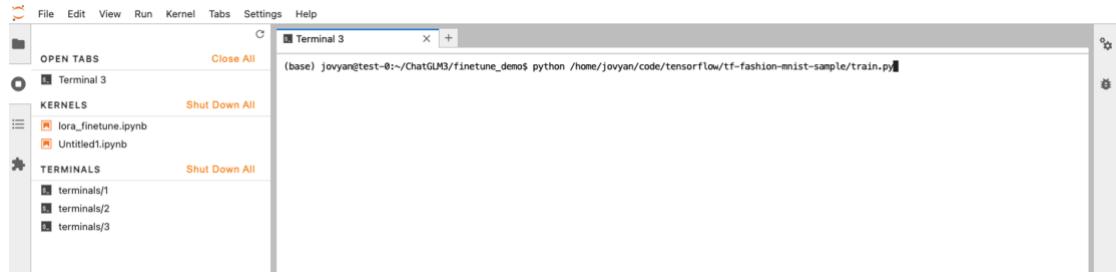
准备开发环境, 点击导航栏的 **Notebooks**, 点击 **创建**

- 将准备好的三个数据集进行关联, 挂载路径请参照下图填写, 注意需要配置好要使用的空数据集



- 选择并绑定环境依赖包

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。



### Note

脚本使用 Tensorflow，如果忘记关联依赖库，也可以临时用 pip install tensorflow 安装。

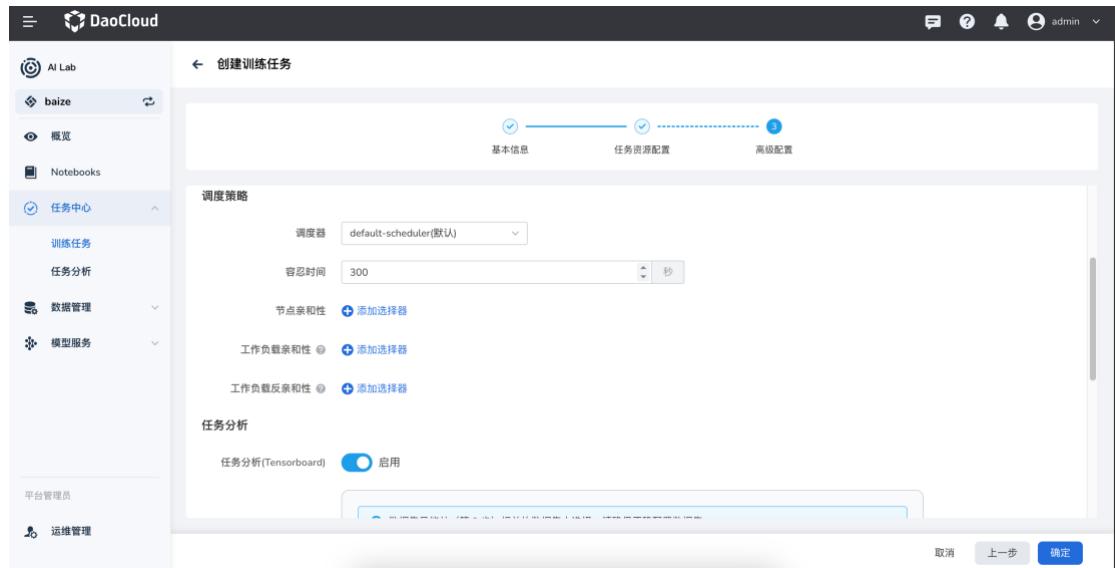
```
python /home/jovyan/code/tensorflow/tf-fashion-mnist-sample/train.py
```

## 创建训练任务

1. 点击导航栏的 **任务中心 -> 训练任务**， 创建一个 Tensorflow 单机任务
2. 先填写基本参数后，点击 **下一步**
3. 在任务资源配置中，正确配置任务资源后，点击 **下一步**
  - 镜像：如果前序环境依赖包准备好了，使用默认镜像即可；如果未准备，要确认镜像内有 tensorflow 的 Python 库
  - shell：使用 bash 即可
  - 启用命令：
  - `python /home/jovyan/code/tensorflow/tf-fashion-mnist-sample/train.py`
4. 在高级配置中，启用 **任务分析（Tensorboard）**，点击 **确定**。

### Note

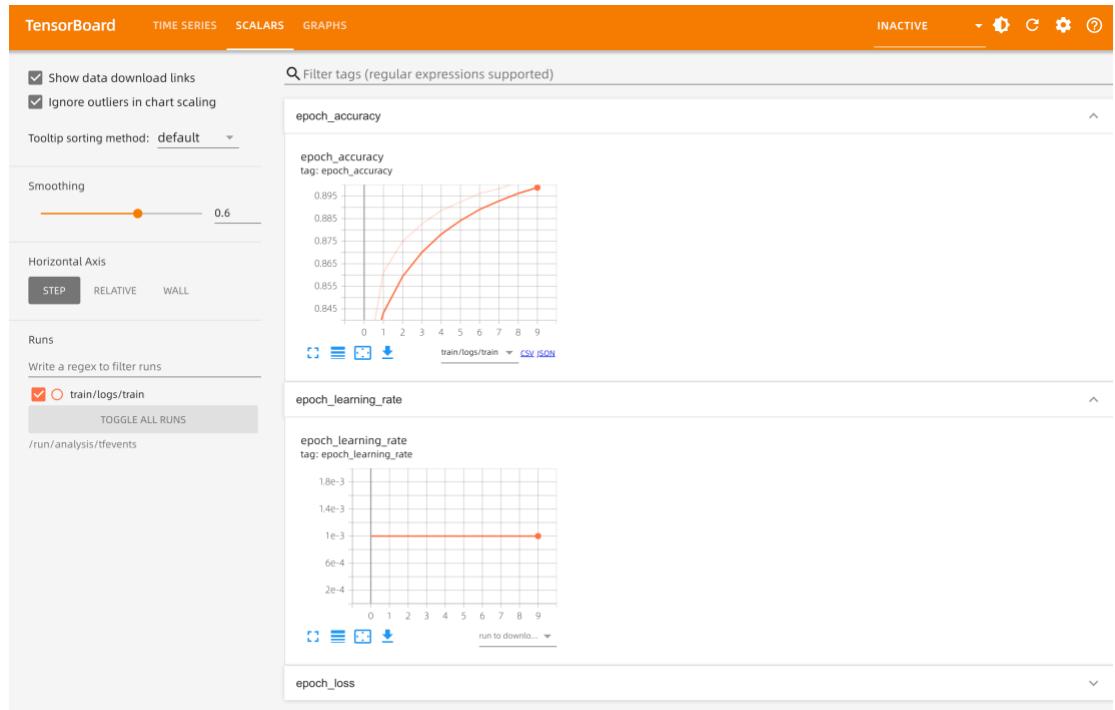
日志所在位置为输出数据集的 `/home/jovyan/model/train/logs/`



5. 返回训练任务列表，等到状态变为成功。点击列表右侧的 |，可以查看详情、克隆任务、更新优先级、查看日志和删除等操作。

| 名称                   | 任务类型          | 状态  | 队列      | 优先级       | 资源配置            | 命名空间 | 创建时间             | 更多 |
|----------------------|---------------|-----|---------|-----------|-----------------|------|------------------|----|
| cli-sh-c-env-slee... | Pytorch 分布式   | 成功  | default | 高 (10000) | cpu:2Core +1    | kebe | 2024-09-09 14:00 |    |
| test2                | Pytorch 单机    | 运行中 | default | 中 (10000) | cpu:1Core +1    | demo | 2024-09-03 15:53 |    |
| dfsd                 | Pytorch 单机    | 成功  | default | 中 (10000) | cpu:1Core +1    | demo | 2024-09-03 15:53 |    |
| dsf-2024082914...    | Pytorch 单机    | 运行中 | default | 中 (10000) | cpu:1Core +1    | demo | 2024-08-29 14:53 |    |
| dsf                  | Pytorch 单机    | 运行中 | default | 中 (10000) | cpu:1Core +1    | demo | 2024-08-29 14:53 |    |
| dfdf                 | Pytorch 单机    | 成功  | default | 中 (10000) | cpu:1Core +4    | demo | 2024-08-29 14:53 |    |
| test-202408211...    | Pytorch 单机    | 运行中 | default | 中 (10000) | cpu:1Core +1    | demo | 2024-08-21 15:29 |    |
| test-06              | Tensorflow 单机 | 失败  | default | 中 (10000) | cpu:1Core +1    | demo | 2024-08-22 10:53 |    |
| test-05              | Pytorch 单机    | 运行中 | default | 中 (10000) | cpu:1.10Core +1 | demo | 2024-08-21 16:43 |    |
| fff                  | Pytorch 单机    | 已创建 | default | 中 (10000) | cpu:1Core +4    | demo | 2024-08-21 16:43 |    |

6. 成功创建任务后，在左侧导航栏点击 任务分析，可以查看任务状态并对任务训练进行调优。



## 创建 Notebook

Notebook 提供了一个在线交互式编程环境，方便开发者快速进行数据科学和机器学习实验。为迎接 AI 浪潮，AI Lab 已集成了当前最流行的 Jupyter Notebook 数据科学交互式开发环境 (IDE)。

您进入 AI Lab 开发者控制台后，可以在不同集群、命名空间中创建和管理 Notebook。

1. 在左侧导航栏中点击 **Notebooks**，进入 Notebook 列表。点击右侧的 **创建** 按钮。

The screenshot shows the DaoCloud developer control panel with the following interface elements:

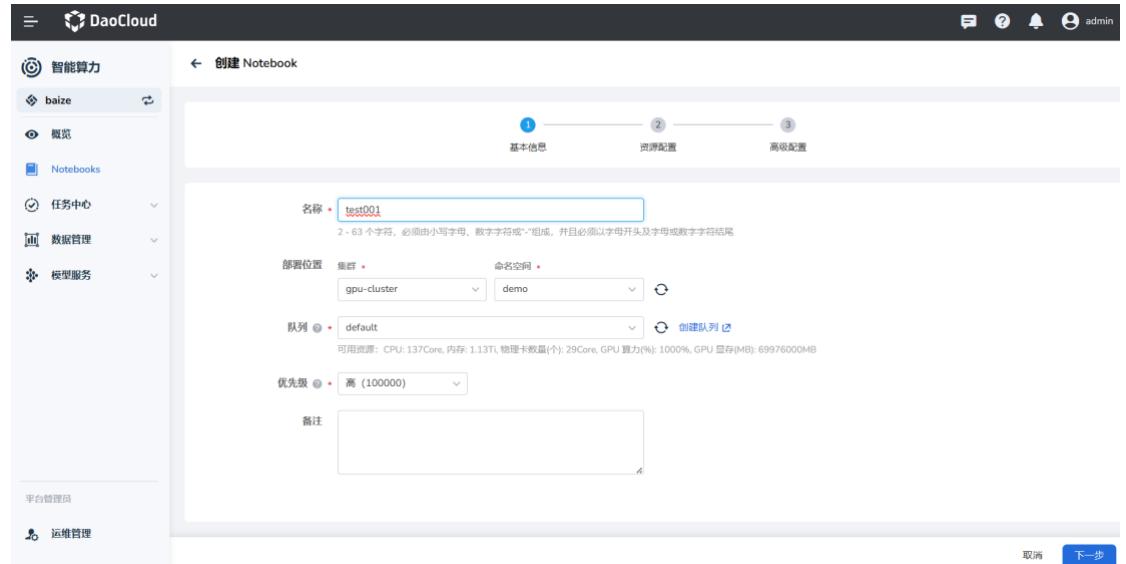
- 左侧导航栏 (Left Navigation Bar):**
  - 智能算力 (Intelligent Computing Power)
  - baize (Cluster Name)
  - 概览 (Overview)
  - Notebooks (Notebooks)
  - 任务中心 (Task Center)
  - 数据管理 (Data Management)
  - 模型服务 (Model Service)
  - 平台管理员 (Platform Administrator)
  - 运维管理 (Operations and Maintenance)
- 顶部工具栏 (Top Toolbar):**
  - 消息 (Messages)
  - 帮助 (Help)
  - 通知 (Notifications)
  - admin (Admin)
- 右侧内容区 (Right Content Area):**

### Notebooks

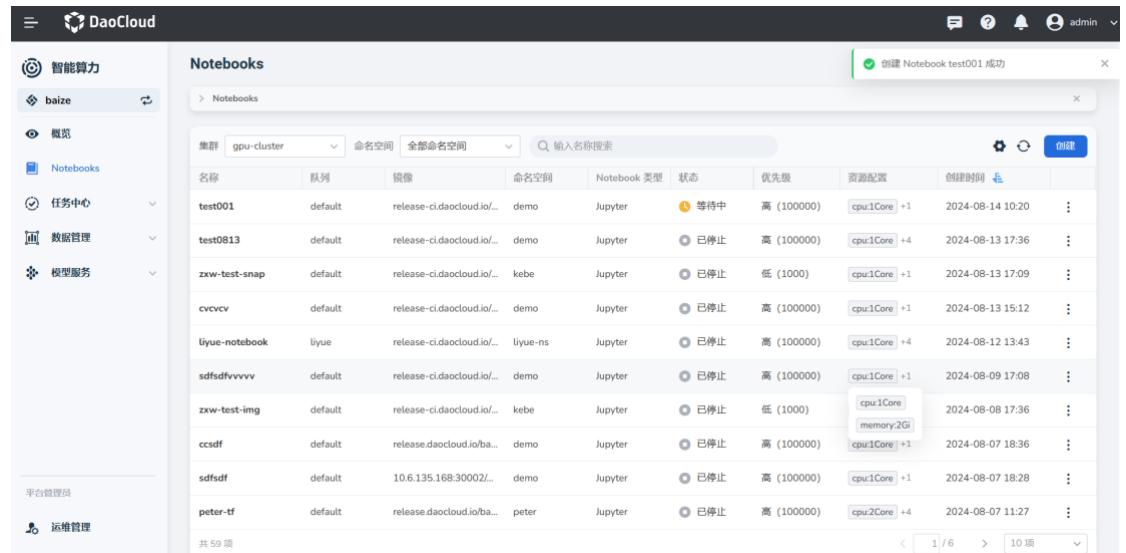
Notebook 是一个提供一个 Web 在线的交互式编程环境，方便开发者快速进行数据科学和机器学习实验。

| 名称             | 队列      | 镜像                         | 命名空间     | Notebook 类型 | 状态  | 优先级        | 资源配置         | 创建时间             | 操作 |
|----------------|---------|----------------------------|----------|-------------|-----|------------|--------------|------------------|----|
| test0813       | default | release-ci.daocloud.io/... | demo     | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +4 | 2024-08-13 17:36 | ⋮  |
| zxw-test-snap  | default | release-ci.daocloud.io/... | kebe     | Jupyter     | 已停止 | 低 (1000)   | cpu:1Core +1 | 2024-08-13 17:09 | ⋮  |
| cvcvcv         | default | release-ci.daocloud.io/... | demo     | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +1 | 2024-08-13 15:12 | ⋮  |
| liyue-notebook | liyue   | release-ci.daocloud.io/... | liyue-ns | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +4 | 2024-08-12 13:43 | ⋮  |
| sdfsdfvvvv     | default | release-ci.daocloud.io/... | demo     | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +1 | 2024-08-09 17:08 | ⋮  |
| zxw-test-img   | default | release-ci.daocloud.io/... | kebe     | Jupyter     | 已停止 | 低 (1000)   | cpu:1Core +1 | 2024-08-08 17:36 | ⋮  |
| ccsdf          | default | release.daocloud.io/ba...  | demo     | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +1 | 2024-08-07 18:36 | ⋮  |
| sdfsdf         | default | 10.6.135.168:30002/...     | demo     | Jupyter     | 已停止 | 高 (100000) | cpu:1Core +1 | 2024-08-07 18:28 | ⋮  |
| peter-tf       | default | release.daocloud.io/ba...  | peter    | Jupyter     | 已停止 | 高 (100000) | cpu:2Core +4 | 2024-08-07 11:27 | ⋮  |
| sdf            | default | release-ci.daocloud.io/... | demo     | Code        | 已停止 | 高 (100000) | cpu:1Core +1 | 2024-08-07 10:46 | ⋮  |

- 系统会预先填充基础配置数据，包括要部署的集群、命名空间、Notebook 镜像地址、队列、资源、用户目录等。调整这些参数后点击 确定。



- 刚创建的 Notebook 状态为 等待中 ，片刻后将变为 运行中 ，默认最新的位于列表顶部。



- 点击右侧的 | ，可以执行更多操作：更新参数、启动/暂停、克隆 Notebook 、查看工作负载详情和删除。

## Note

如果选择纯 CPU 资源后，发现挂载了节点上的所有 GPU 卡，可以尝试添加一条 container env 来解决此问题：

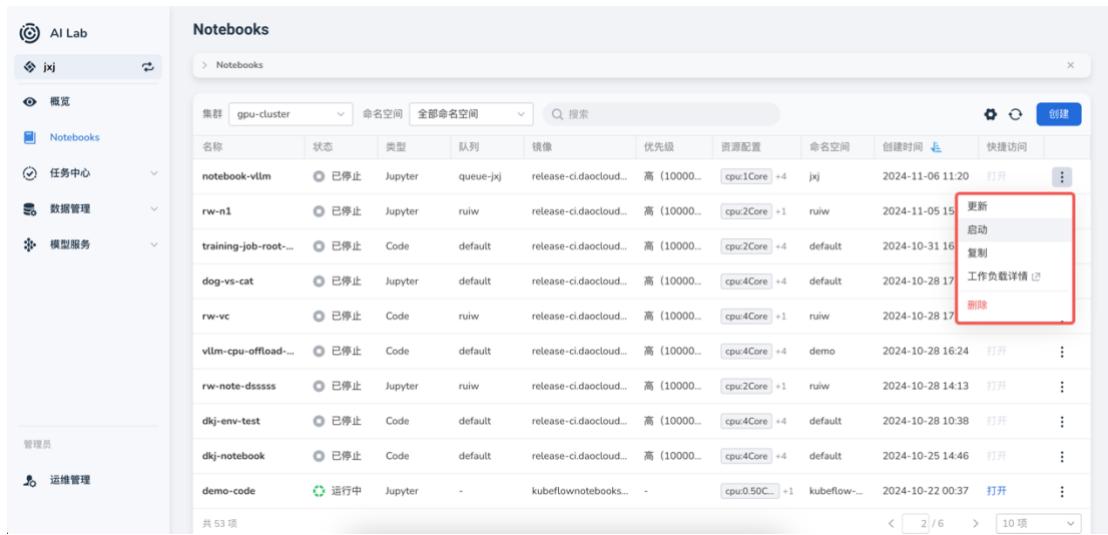
```
NVIDIA_VISIBLE_DEVICES=""
```

# 启动和暂停 Notebook

Notebook 创建成功后，通常会有几个状态：

- 等待中
- 运行中
- 已停止

如果某个 Notebook 的状态为 已停止，在列表右侧点击 ，在弹出菜单中选择 启动。



The screenshot shows the AI Lab interface with the 'Notebooks' section selected. The list displays various notebooks, some in a stopped state. A red box highlights the context menu for a specific notebook named 'rw-n1'. The menu options include '更新' (Update), '启动' (Start), '复制' (Copy), '工作负载详情' (Workload Details), and '删除' (Delete). The 'rw-n1' entry in the list has a status of '已停止' (Stopped) and a type of 'Jupyter'.

| 名称                    | 状态  | 类型      | 队列        | 镜像                     | 优先级          | 资源配置         | 命名空间         | 创建时间             | 快捷访问 |
|-----------------------|-----|---------|-----------|------------------------|--------------|--------------|--------------|------------------|------|
| notebook-vlrm         | 已停止 | Jupyter | queue-jxj | release-ci.daocloud... | 高 (10000...) | cpu:1Core +4 | jxj          | 2024-11-06 11:20 |      |
| rw-n1                 | 已停止 | Jupyter | ruiw      | release-ci.daocloud... | 高 (10000...) | cpu:2Core +1 | ruiw         | 2024-11-05 15    |      |
| training-job-root-... | 已停止 | Code    | default   | release-ci.daocloud... | 高 (10000...) | cpu:2Core +4 | default      | 2024-10-31 16    |      |
| dog-vs-cat            | 已停止 | Jupyter | default   | release-ci.daocloud... | 高 (10000...) | cpu:4Core +4 | default      | 2024-10-28 17    |      |
| rw-vc                 | 已停止 | Code    | ruiw      | release-ci.daocloud... | 高 (10000...) | cpu:4Core +1 | ruiw         | 2024-10-28 17    |      |
| vlim-cpu-offload-...  | 已停止 | Code    | default   | release-ci.daocloud... | 高 (10000...) | cpu:4Core +4 | demo         | 2024-10-28 16:24 |      |
| rw-note-dsssss        | 已停止 | Jupyter | ruiw      | release-ci.daocloud... | 高 (10000...) | cpu:2Core +1 | ruiw         | 2024-10-28 14:13 |      |
| dki-env-test          | 已停止 | Code    | default   | release-ci.daocloud... | 高 (10000...) | cpu:4Core +4 | default      | 2024-10-28 10:38 |      |
| dki-notebook          | 已停止 | Code    | default   | release-ci.daocloud... | 高 (10000...) | cpu:4Core +4 | default      | 2024-10-25 14:46 |      |
| demo-code             | 运行中 | Jupyter | -         | kubeflownotebooks...   | -            | cpu:0.50C +1 | kubeflow-... | 2024-10-22 00:37 |      |

此 Notebook 将进入运行队列中，状态变为 等待中，如果一切正常，片刻后其状态将变为 运行中。

如果使用结束，可以从菜单中选择 暂停，将其状态变为 已停止。

## Notebook 工作负载

如果想要查看某个 Notebook 的工作负载，可以执行以下操作：

1. 在 Notebook 列表右侧点击 ，在弹出菜单中选择 工作负载详情。

2. 跳转到有状态工作负载（StatefulSet）列表，可以查看：

- 容器组 Pod 的运行状态、IP、资源请求和使用情况
- 容器配置信息
- 访问方式：ClusterIP、NodePort
- 调度策略：节点和工作负载的亲和性、反亲和性
- 标签与注解：工作负载、Pod 的标签与注解键值对
- 弹性伸缩：支持 HPA、CronHPA、VPA 等方式
- 事件列表：警告、通知等消息

3. 在容器组列表，点击右侧的 |，可以针对 Pod 执行更多操作。

# 删除 Notebook

如果发现某个 Notebook 夗余、过期或因其他缘故不再需要，可以从 Notebook 列表中删除。

- 在 Notebook 列表右侧点击 ，在弹出菜单中选择 **删除**。

- 在弹窗中输入 Notebook 名称，确认无误后点击 **删除**。

- 屏幕提示删除成功，该 Notebook 从列表中消失。

The screenshot shows the DaoCloud AI Lab web interface. On the left is a sidebar with navigation links: 智能算力, baize (selected), 概览, Notebooks (selected), 任务中心, 数据管理, 平台管理员, and 运维管理. The main area is titled 'Notebooks' and displays a table of notebook entries. The table columns are: 名称, 队列, 镜像, 访问地址, 状态, 资源配置, and 创建时间. The table contains 9 items, each with a delete icon. A success message 'notebook-02 删除成功' is displayed at the top right of the table area.

| 名称            | 队列            | 镜像                       | 访问地址 | 状态  | 资源配置     | 创建时间             |
|---------------|---------------|--------------------------|------|-----|----------|------------------|
| yk-test       | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:1 +2 | 2024-01-15 14:36 |
| notebook-02   | -             | -                        | -    | 已停止 | -        | 2024-01-19 12:38 |
| notebook-01   | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:1 +1 | 2024-01-19 10:45 |
| new           | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:1 +1 | 2024-01-16 10:55 |
| neka-notebook | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:2 +2 | 2024-01-04 16:44 |
| kebe-gpu      | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:8 +2 | 2024-01-11 18:58 |
| kebe          | -             | docker.m.daocloud.io/... | -    | 已停止 | -        | 2024-01-10 16:08 |
| kay           | kebe-1c2g     | docker.m.daocloud.io/... | -    | 已停止 | cpu:1 +1 | 2023-12-28 18:29 |
| as            | cluster-queue | docker.m.daocloud.io/... | -    | 已停止 | cpu:1 +1 | 2023-12-28 11:36 |

## Caution

Notebook 一旦被删除将不可恢复，请谨慎操作。

# Notebook SSH 访问指南

AI Lab 提供的 Notebook 支持在本地通过 SSH 的方式访问；

通过简单的配置，即可使用 SSH 访问 Jupyter Notebook 的功能。无论您是使用 Windows、Mac 还是 Linux 操作系统，都可以按照以下步骤进行操作。

## 配置 SSH 访问凭证

### 生成 SSH 密钥对

首先，您需要在您的计算机上生成 SSH 公钥和私钥对。这个密钥对将用于认证过程，确保安全访问。

#### [Mac/Linux](#)[Windows](#)

1. 打开终端
2. 输入命令：
3. `ssh-keygen -t rsa -b 4096`
4. 当系统提示您“Enter a file in which to save the key”，您可以直接敲击 `Enter` 键使用默认路径，或者指定一个新的路径。

- 接下来，系统会提示您输入密码（可选），这将增加一个额外的安全层。如果选择输入密码，请记住这个密码，因为每次使用密钥时都会需要它。

## 添加 SSH 公钥到个人中心

### Note

具体操作可以参考：[配置 SSH 公钥](#)

- 打开生成的公钥文件，通常位于 `~/.ssh/id_rsa.pub`（如果您没有更改默认路径）
- 复制公钥内容
- 登录到 DCE 5.0，从右上角进入 个人中心
- 在 SSH 公钥配置页，添加你本地生成的公钥文件，保存更改

The screenshot shows the 'Personal Center' interface. In the top navigation bar, there is a user profile icon labeled 'admin' and the message '未设置邮箱'. Below the navigation bar, there are four tabs: 'Security Settings', 'Access Keys', 'SSH Public Key' (which is highlighted in blue), and 'Language Settings'. A blue banner at the top of the main content area says 'In order to import an SSH public key, you need to generate an SSH public key' with a link to 'Generate SSH Public Key'. The main content is a table listing imported SSH public keys:

| SSH Public Key Name | Status      | Expiration Time | Import Time      | Operations |
|---------------------|-------------|-----------------|------------------|------------|
| peter-laptop        | ● Effective | Permanent       | 2024-09-04 19:05 | ⋮          |
| dkj                 | ● Effective | Permanent       | 2024-09-13 11:26 | ⋮          |
| dkj-mini            | ● Effective | Permanent       | 2024-09-14 21:58 | ⋮          |
| rw-ssh              | ● Effective | Permanent       | 2024-09-24 17:55 | ⋮          |
| jxj-mac             | ● Effective | Permanent       | 2024-09-26 14:04 | ⋮          |
| peter-again-tmp     | ● Effective | Permanent       | 2024-09-26 14:40 | ⋮          |
| jxj-22              | ● Effective | Permanent       | 2024-09-26 14:47 | ⋮          |

## 在 Notebook 中开启 SSH 访问

- 登录到 Jupyter Notebook 的 Web 界面。

The screenshot shows the AI Lab interface with the 'Notebooks' section open. A list of notebooks is displayed, including 'zs2-notebooks-03' (running), 'zs2-notebooks-01' (stopped), 'rw-code', 'rw-jupyter', 'test005', 'gpt2-notebook', 'notebook-vilm', 'training-job-root-t...', 'dog-vs-cat', and 'vilm-cpu-offload-t...'. The 'zs2-notebooks-03' row has a red box around its 'SSH' icon. Below the list is a search bar and a pagination area. The terminal window shows the command 'pytorch-distributed-sample.py' being run, displaying dependency installation logs and training sample code execution.

2. 寻找您想要启用 SSH 访问的 Notebook。
3. 在 Notebook 的设置或详情页面，找到开启 SSH 访问的选项并启用它。

The screenshot shows the 'Update Notebook' dialog for 'zs2-notebooks-03'. The 'SSH Access' section is highlighted with a red box. It contains a toggle switch labeled '启用' (Enable) and a note: '使用当前用户在个人中心注册的用户名密码、公钥登录 Notebook。配置和访问方式可参考 Notebook SSH 访问指南。' (Use the current user's registered username and password, or public key to log in to the Notebook. Configuration and access methods can be referred to in the Notebook SSH access guide.)

4. 记录或复制显示的 SSH 访问命令。这个命令将用于后续步骤中的 SSH 连接。

The screenshot shows the AI Lab interface with the 'Notebooks' tab selected. A specific notebook named 'zsz-notebooks-03' is displayed. In the 'SSH 访问' (SSH Access) section, a command is listed: 'ssh -p 443 admin@gpu-cluster/jx.zsz-notebooks-03@10.20.100.201'. This command is highlighted with a red box.

## 不同环境下的 SSH 访问方式

### 访问示例

假设您获得的 SSH 访问命令如下：

```
# ssh  
{DCE5_USERNAME}@{CLUSTER}.{NAMESPACE}.{NOTEBOOK_NAME}@{DCE5_UI_LOG  
IN_IP} -p {DCE5_UI_LOGIN_IP}  
ssh baizeuser01@gpu-cluster.demo.demo-notebook@10.20.100.201 -p 80 -i private_key
```

- {DCE5\_USERNAME} 替换为您的用户名
- {DCE5\_UI\_LOGIN\_IP} 替换为实际的主机名
- {DCE5\_UI\_LOGIN\_IP} 替换为实际的端口号

### Windows

推荐使用 PuTTY 或 Git Bash 进行 SSH 连接。

#### [PuTTY](#)[Git Bash](#)

1. 打开 PuTTY
2. 在 **Host Name (or IP address)** 栏输入 mockhost (实际的主机名)
3. 输入端口号 2222 (实际的端口号)

4. 点击 **Open** 开始连接
5. 第一次连接时，可能会提示验证服务器的身份，点击 **Yes**

## Mac/Linux

1. 打开终端。
2. 输入访问命令：
3. # ssh  
{DCE5\_USERNAME}@{CLUSTER}.{NAMESPACE}.{NOTEBOOK\_NAME}@{D  
CE5\_UI\_LOGIN\_IP} -p {DCE5\_UI\_LOGIN\_IP}
4. ssh baizeuser01@gpu-cluster.demo.demo-notebook@10.20.100.201 -p 80 -i  
private\_key
5. 如果系统提示您接受主机的身份，请输入 yes。

## 配合 IDE 实现远程开发

除了使用命令行工具进行 SSH 连接，您还可以利用现代 IDE 如 Visual Studio Code (VSCode) 和 PyCharm 的 SSH 远程连接功能，直接在本地 IDE 中开发并利用远程服务器的资源。

[在 VSCode 中使用 SSH 远程连接](#) [在 PyCharm 中使用 SSH 远程连接](#)

VSCode 通过 **Remote - SSH** 扩展支持 SSH 远程连接，允许您直接在本地 VSCode 环境中编辑远程服务器上的文件，并运行命令。

操作步骤为：

1. 确保您已安装 VSCode 和 **Remote - SSH** 扩展。
2. 打开 VSCode，点击左侧活动栏底部的远程资源管理器图标。
3. 选择 **Remote-SSH: Connect to Host...** 选项，然后点击 **+ Add New SSH Host...**
4. 输入 SSH 连接命令，例如：
5. # ssh  
{DCE5\_USERNAME}@{CLUSTER}.{NAMESPACE}.{NOTEBOOK\_NAME}@{D  
CE5\_UI\_LOGIN\_IP} -p {DCE5\_UI\_LOGIN\_IP}
6. ssh baizeuser01@gpu-cluster.demo.demo-notebook@10.20.100.201 -p 80 -i  
private\_key
7. 敲击 Enter 键。请将 username、mockhost 和 2222 替换为实际的用户名、主机名和端口号。

8. 选择一个配置文件来保存此 **SSH** 主机，通常选择默认即可。

完成后，您的 **SSH** 主机将添加到 **SSH** 目标列表中。点击您的主机进行连接。如果是第一次连接，可能会提示您验证主机的指纹。接受后，您将被要求输入密码（如果 **SSH** 密钥设置了密码）。连接成功后，您可以像在本地开发一样在 **VSCode** 中编辑远程文件，并利用远程资源。

## 安全限制

在同一个 **Workspace** 内，任意用户都可以通过自己的 **SSH** 访问凭证来登录到启用了 **SSH** 的 **Notebook**。这意味着，只要用户配置了自己的 **SSH** 公钥到个人中心，并且 **Notebook** 启用了 **SSH** 访问，就可以使用 **SSH** 进行安全连接。

请注意，不同用户的访问权限可能会根据 **Workspace** 的配置而有所不同。确保您了解并遵守您所在组织的安全和访问策略。

---

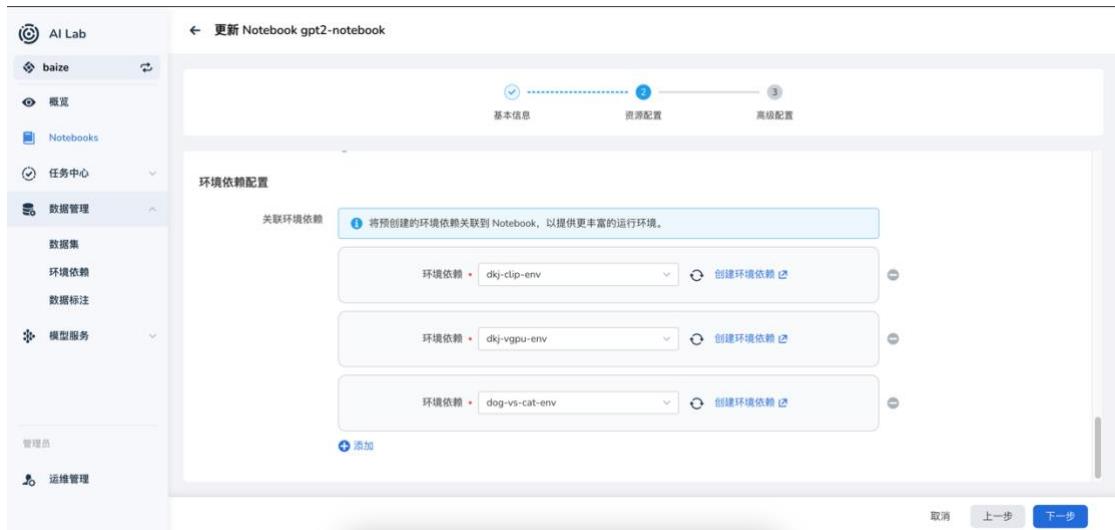
通过遵循上述步骤，您应该能够成功配置并使用 **SSH** 访问 **Jupyter Notebook**。如果遇到任何问题，请参考系统帮助文档或联系系统管理员。

## 在 **Notebook** 中使用环境

环境管理是 **AI Lab** 的重要功能之一，通过在 **Notebook** 中关联对应的环境，可以快速切换不同的环境，方便用户进行开发和调试。

### 创建 **Notebook** 时选择环境

在创建 **Notebook** 时，可以选择一个或多个的环境 **Envs**



如果没有合适的环境，可以去[创建一个新的环境](#)。

## 在 Notebook 使用环境

### Note

在 Notebook 中，我们同时提供了 conda 和 mamba 两种，用户可以根据自己的需求选择合适的环境管理工具。

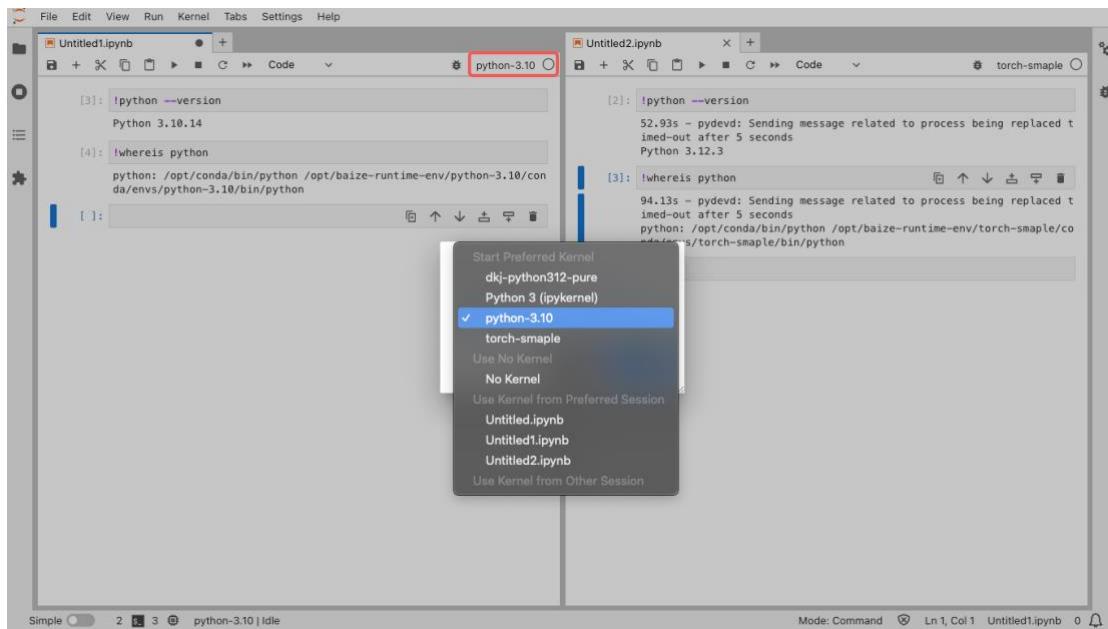
AI Lab 中，我们采用了 conda 环境管理工具，用户可以在 Notebook 中通过 `!conda env list` 命令查看当前环境列表。

```
(base) jovyan@chuanjia-jupyter-0:~/yolov8$ conda env list
# conda environments:
#
dkj-python312-pure      /opt/baize-runtime-env/dkj-python312-pure/conda/envs/dkj-
python312-pure
python-3.10              /opt/baize-runtime-env/python-3.10/conda/envs/python-3.10
torch-smapple            /opt/baize-runtime-env/torch-smapple/conda/envs/torch-smapple
base                     * /opt/conda      # 当前激活的环境
baize-base               /opt/conda/envs/baize-base
```

这个命令会列出所有的 conda 环境，并在当前激活的环境前面加上一个星号 (\*)。

# JupyterLab 的 Kernel 环境管理

在 Jupyterlab 中，我们自动将 Notebook 关联的环境绑定到 Kernel 列表中，用户可以通过 Kernel 快速切换环境。



通过以上办法，可以同时在一个 Notebook 中使用不同编写和调试算法。

## Terminal 切换环境

AI Lab 的 Notebook 目前也已经支持了 VSCode。

如果您更喜欢在 Terminal 中管理和切换环境，可以安装如下步骤：

在首次启动并使用 Notebook 时，需要先执行 conda init，然后再执行 conda activate <env\_name> 切换到对应的环境。

```
(base) jovyan@chuanjia-jupyter-0:~/yolov8$ conda init bash# 初始化 bash 环境，仅首次使用需要执行
no change      /opt/conda/condabin/conda
change       /opt/conda/bin/conda
change       /opt/conda/bin/conda-env
change       /opt/conda/bin/activate
change       /opt/conda/bin/deactivate
change       /opt/conda/etc/profile.d/conda.sh
change       /opt/conda/etc/fish/conf.d/conda.fish
change       /opt/conda/shell/condabin/Conda.psm1
change       /opt/conda/shell/condabin/conda-hook.ps1
```

```
change      /opt/conda/lib/python3.11/site-packages/xontrib/conda.xsh
change      /opt/conda/etc/profile.d/conda.csh
change      /home/jovyan/.bashrc
action taken.

Added mamba to /home/jovyan/.bashrc
```

==> For changes to take effect, close and re-open your current shell. <==

```
(base) jovyan@chuanjia-jupyter-0:~/yolov8$ source ~/.bashrc # 重新加载 bash 环境
(base) jovyan@chuanjia-jupyter-0:~/yolov8$ conda activate python-3.10 # 切换到 python-3.10 环境
(python-3.10) jovyan@chuanjia-jupyter-0:~/yolov8$ conda env list
```

```
mamba version : 1.5.1

# conda environments:
#
dkj-python312-pure      /opt/baize-runtime-env/dkj-python312-pure/conda/envs/dkj-
python312-pure
python-3.10              * /opt/baize-runtime-env/python-3.10/conda/envs/python-3.10    #
当前激活的环境
torch-smapple            /opt/baize-runtime-env/torch-smapple/conda/envs/torch-smapple
base                     /opt/conda
baize-base               /opt/conda/envs/baize-base
```

如果您更喜欢使用 mamba , 这里需要使用 mambainit 和 mambactivate <env\_name>。

## 查看环境中的包

通过不同环境管理的一个很重要的功能是，可以在一个 Notebook 中通过快速切换环境，使用不用的包。

我们可以通过下方的命令来使用 conda 查看当前环境中的所有包。

```
(python-3.10) jovyan@chuanjia-jupyter-0:~/yolov8$ conda list
# packages in environment at /opt/baize-runtime-env/python-3.10/conda/envs/python-3.10:
#
# Name          Version       Build  Channel
_libgcc_mutex   0.1           main   defaults
_openmp_mutex   5.1           1_gnu  defaults
... # 省略部分输出
idna            3.7           py310h06a4308_0  defaults
ipykernel        6.28.0        py310h06a4308_0  defaults
ipython          8.20.0        py310h06a4308_0  defaults
```

```
ipython_genutils      0.2.0          pyhd3eb1b0_1    defaults
jedi                 0.18.1         py310h06a4308_1  defaults
jinja2                3.1.4          py310h06a4308_0    defaults
jsonschema            4.19.2         py310h06a4308_0    defaults
jsonschema-specifications 2023.7.1  py310h06a4308_0    defaults
jupyter_client        7.4.9          py310h06a4308_0    defaults
jupyter_core          5.5.0          py310h06a4308_0    defaults
jupyter_events        0.8.0          py310h06a4308_0    defaults
jupyter_server        2.10.0         py310h06a4308_0    defaults
jupyter_server_terminals 0.4.4       py310h06a4308_1    defaults
jupyterlab_pygments   0.2.2          py310h06a4308_0    defaults
... # 省略部分输出
xz                   5.4.6          h5eee18b_1    defaults
yaml                 0.2.5          h7b6447c_0    defaults
zeromq               4.3.5          h6a678d5_0    defaults
zlib                 1.2.13         h5eee18b_1    defaults
```

## 更新环境的包

目前，可以通过在 **AI Lab** 的界面中 [环境依赖](#) 来更新环境中的包。

## baizectl 命令行工具使用指南

**baizectl** 是在 **DCE 5.0 AI Lab** 模块中专门服务于模型开发者与数据科学家们使用的命令行工具。它提供了一系列命令来帮助用户管理分布式训练作业、查看任务状态、管理数据集等操作，同时支持连接 **Kubernetes** 工作集群和 **DCE 5.0** 工作空间，帮助用户更高效地使用和管理 **Kubernetes** 平台资源。

目前，**baizectl** 已经集成在 **DCE 5.0 AI Lab** 中。你在创建 **Notebook** 后，即可在 **Notebook** 中直接使用 **baizectl**。

## 快速上手

### 基本信息

**baizectl** 命令的基本格式如下：

```
jovyan@19d0197587cc:/$ baizectl
AI platform management tool
```

Usage:

`baizectl [command]`

Available Commands:

|            |                                                        |
|------------|--------------------------------------------------------|
| completion | Generate the completion script for the specified shell |
| data       | Management datasets                                    |
| help       | Help about any command                                 |
| job        | Manage jobs                                            |
| login      | Login to the platform                                  |
| version    | Show cli version                                       |

Flags:

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| --cluster string       | Cluster name to operate                                                             |
| -h, --help             | help for baizectl                                                                   |
| --mode string          | Connection mode: auto, api, notebook (default "auto")                               |
| -n, --namespace string | Namespace to use for the operation. If not set, the default Namespace will be used. |
| -s, --server string    | DCE5 access base url                                                                |
| --skip-tls-verify      | Skip TLS certificate verification                                                   |
| --token string         | DCE5 access token                                                                   |
| -w, --workspace int32  | Workspace ID to use for the operation                                               |

Use "`baizectl [command] --help`" for more information about a command.

以上是 `baizectl` 的基本信息，用户可以通过 `baizectl --help` 查看帮助信息，或者通过 `baizectl [command] --help` 查看具体命令的帮助信息。

## 查看版本信息

`baizectl` 支持通过 `version` 命令查看版本信息。

```
(base) jovyan@den-0:~$ baizectl version
baizectl version: v0.5.0, commit sha: ac0837c4
```

## 命令格式

`baizectl` 命令的基本格式如下：

`baizectl [command] [flags]`

其中，`[command]` 是具体的操作命令，如 `data`、`job` 等，`[flags]` 是可选的参数，用于指定操作的详细信息。

## 常用选项

- **--cluster string:** 指定要操作的集群名称。
  - **-h, --help:** 显示帮助信息。
  - **--mode string:** 连接模式，可选值为 `auto`、`api`、`notebook`（默认值为 `auto`）。
  - **-n, --namespace string:** 指定操作的命名空间。如果未设置，将使用默认命名空间。
  - **-s, --server string:** DCE5 访问基础 URL。
  - **--skip-tls-verify:** 跳过 TLS 证书验证。
  - **--token string:** DCE5 访问令牌。
  - **-w, --workspace int32:** 指定操作的工作区 ID。
- 

## 功能介绍

### 任务管理

`baizectl` 提供了一系列命令来管理分布式训练任务，包含了查看任务列表，提交任务、查看日志、重启任务、删除任务等。

```
jovyan@19d0197587cc:/$ baizectl job  
Manage jobs
```

Usage:

```
baizectl job [command]
```

Available Commands:

|         |                    |
|---------|--------------------|
| delete  | Delete a job       |
| logs    | Show logs of a job |
| ls      | List jobs          |
| restart | restart a job      |
| submit  | Submit a job       |

Flags:

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| -h, --help          | help for job                                               |
| -o, --output string | Output format. One of: table, json, yaml (default "table") |
| --page int          | Page number (default 1)                                    |
| --page-size int     | Page size (default -1)                                     |

```
--search string    Search query
--sort string      Sort order
--truncate int     Truncate output to the given length, 0 means no truncation (default
50)
```

Use "baizectl job [command] --help" for more information about a command.

## 提交训练任务

baizectl 支持使用 submit 命令提交一个任务，用户可以通过 baizectl job submit --help 查看详细信息。

```
(base) jovyan@den-0:~$ baizectl job submit --help
Submit a job
```

Usage:

```
baizectl job submit [flags] -- command ...
```

Aliases:

```
submit, create
```

Examples:

```
# Submit a job to run the command "torchrun python train.py"
baizectl job submit -- torchrun python train.py
# Submit a job with 2 workers(each pod use 4 gpus) to run the command "torchrun python
train.py" and use the image "pytorch/pytorch:1.8.1-cuda11.1-cudnn8-runtime"
baizectl job submit --image pytorch/pytorch:1.8.1-cuda11.1-cudnn8-runtime --workers 2 --
requests-resources nvidia.com/gpu=4 -- torchrun python train.py
# Submit a tensorflow job to run the command "python train.py"
baizectl job submit --tensorflow -- python train.py
```

Flags:

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --annotations stringArray  | The annotations of the job, the<br>format is key=value                                                                                                                                                                                                                                                                                                                                                                                                                |
| --auto-load-env            | It only takes effect when executed<br>in Notebook, the environment variables of the current environment will be automatically read<br>and set to the environment variables of the Job, the specific environment variables to be read<br>can be specified using the BAIZE_MAPPING_ENVS environment variable, the default is<br>PATH,CONDA_*,*PYTHON*,NCCL_*, if set to false, the environment variables of the current<br>environment will not be read. (default true) |
| --commands stringArray     | The default command of the job                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| -d, --datasets stringArray | The dataset bind to the job, the<br>format is datasetName:mountPath, e.g. mnist:/data/mnist                                                                                                                                                                                                                                                                                                                                                                           |

|                                                                                                                                                   |                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -e, --envs stringArray<br>job, the format is key=value                                                                                            | The environment variables of the job, the format is key=value                                                                                                   |
| -x, --from-notebook string<br>configuration of the current Notebook and directly create tasks, including images, resources, Dataset, etc.         | Define whether to read the configuration of the current Notebook and directly create tasks, including images, resources, Dataset, etc.                          |
|                                                                                                                                                   | auto: Automatically determine the mode according to the current environment. If the current environment is a Notebook, it will be set to notebook mode.         |
|                                                                                                                                                   | false: Do not read the configuration of the current Notebook.                                                                                                   |
|                                                                                                                                                   | true: Read the configuration of the current Notebook. (default "auto")                                                                                          |
| -h, --help<br>--image string<br>specified if fromNotebook is false.                                                                               | help for submit<br>The image of the job, it must be specified if fromNotebook is false.                                                                         |
| -t, --job-type string<br>TENSORFLOW, PADDLE (default "PYTORCH")<br>--labels stringArray<br>key=value<br>--max-retries int32<br>this job failed    | Job type: PYTORCH, TENSORFLOW, PADDLE (default "PYTORCH")<br>The labels of the job, the format is key=value<br>number of retries before marking this job failed |
| --max-run-duration int<br>relative to the startTime that the job may be active before the system tries to terminate it                            | Specifies the duration in seconds relative to the startTime that the job may be active before the system tries to terminate it                                  |
| --name string<br>name will be generated automatically.                                                                                            | The name of the job, if empty, the name will be generated automatically.                                                                                        |
| --paddle<br>priority than --job-type                                                                                                              | PaddlePaddle Job, has higher priority than --job-type                                                                                                           |
| --priority string<br>support baize-medium-priority, baize-low-priority, baize-high-priority                                                       | The priority of the job, current support baize-medium-priority, baize-low-priority, baize-high-priority                                                         |
| --pvcs stringArray<br>is pvcName:mountPath, e.g. mnist:/data/mnist<br>--pytorch<br>than --job-type                                                | The pvcs bind to the job, the format is pvcName:mountPath, e.g. mnist:/data/mnist<br>Pytorch Job, has higher priority than --job-type                           |
| --queue string<br>--requests-resources stringArray<br>resources of requests                                                                       | The queue to used<br>Similar to resources, but sets the resources of requests                                                                                   |
| --resources stringArray<br>string in the format of cpu=1,memory=1Gi,nvidia.com/gpu=1, it will be set to the limits and requests of the container. | The resources of the job, it is a string in the format of cpu=1,memory=1Gi,nvidia.com/gpu=1, it will be set to the limits and requests of the container.        |
| --restart-policy string<br>--failure()                                                                                                            | The job restart policy (default "on-failure")                                                                                                                   |
| --runtime-envs baizectl data ls --runtime-env                                                                                                     | The runtime environment to use for the job, you can use baizectl data ls --runtime-env to get the runtime environment                                           |

|                                 |                                                                                                                                                                   |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --shm-size int32                | The shared memory size of the job, default is 0, which means no shared memory, if set to more than 0, the job will use the shared memory, the unit is MiB         |
| --tensorboard-log-dir string    | The tensorboard log directory, if set, the job will automatically start tensorboard, else not. The format is /path/to/log, you can use relative path in notebook. |
| --tensorflow<br>than --job-type | Tensorflow Job, has higher priority                                                                                                                               |
| --workers int                   | The workers of the job, default is 1, which means single worker, if set to more than 1, the job will be distributed. (default 1)                                  |
| --working-dir string            | The working directory of job container, if in notebook mode, the default is the directory of the current file                                                     |

## Note

提交任务的命令参数说明：

- **--name:** 任务名称，如果为空，则会自动生成
- **--image:** 镜像名称，必须指定
- **--priority:** 任务优先级，支持 高=baize-high-priority、中=baize-medium-priority、低=baize-low-priority
- **--requests-resources: requests** 的任务资源，格式为 cpu=1 memory=1Gi,nvidia.com/gpu=1
- **--resources: limits** 的任务资源，格式为 cpu=1 memory=1Gi,nvidia.com/gpu=1
- **--workers:** 任务工作节点数，默认为 1，当设置大于 1 时，任务将会分布式运行
- **--queue:** 任务队列，需要提前创建队列资源
- **--working-dir:** 工作目录，如果在 Notebook 模式下，会默认使用当前文件目录
- **--datasets:** 数据集，格式为 datasetName:mountPath，例如 mnist:/data/mnist
- **--shm-size:** 共享内存大小，在分布式训练任务时，可以启用，表示使用共享内存，单位为 MiB
- **--labels:** 任务标签，格式为 key=value
- **--max-retries:** 最大重试次数，任务失败后重试次数，失败后会重启任务，默认不限制
- **--max-run-duration:** 最大运行时间，任务运行时间超过指定时间后，会被系统终止，默认不限制
- **--restart-policy:** 重启策略，支持 on-failure、never、always，默认为 on-failure

- `--from-notebook`: 是否从 Notebook 中读取配置，支持 auto、true、false，默认为 auto

### PYTORCH 单机任务示例

提交训练任务示例，用户可以根据实际需求修改参数，以下为创建一个 PyTorch 任务的示例：

```
baizectl job submit --name demojob-v2 -t PYTORCH \
--image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--priority baize-high-priority \
--requests-resources cpu=1,memory=1Gi \
--workers 1 \
--queue default \
--working-dir /data \
--datasets fashion-mnist:/data/mnist \
--labels job_type=pytorch \
--max-retries 3 \
--max-run-duration 60 \
--restart-policy on-failure \
-- sleep 1000
```

### PYTORCH 分布式任务示例

提交训练任务示例，用户可以根据实际需求修改参数，以下为创建一个 PyTorch 任务的示例：

```
baizectl job submit --name demojob-v2 -t PYTORCH \
--image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--priority baize-high-priority \
--requests-resources cpu=1,memory=1Gi \
--workers 2 \ # 多任务副本会自动创建分布式任务
--shm-size 1024 \
--queue default \
--working-dir /data \
--datasets fashion-mnist:/data/mnist \
--labels job_type=pytorch \
--max-retries 3 \
--max-run-duration 60 \
--restart-policy on-failure \
-- sleep 1000
```

### TENSORFLOW 任务示例

使用 `-t` 参数指定任务类型，以下为创建一个 Tensorflow 任务的示例：

```
baizectl job submit --name demojob-v2 -t TENSORFLOW \
--image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--priority baize-high-priority \
--from-notebook auto \
--workers 1 \
--queue default \
--working-dir /data \
--datasets fashion-mnist:/data/mnist \
--labels job_type=pytorch \
--max-retries 3 \
--max-run-duration 60 \
--restart-policy on-failure \
-- sleep 1000
```

也可以使用 `--job-type` 或者 `--tensorflow` 参数指定任务类型

## PADDLE 任务示例

```
baizectl job submit --name demojob-v2 -t PADDLE \
--image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--priority baize-high-priority \
--queue default \
--working-dir /data \
--datasets fashion-mnist:/data/mnist \
--labels job_type=pytorch \
--max-retries 3 \
--max-run-duration 60 \
--restart-policy on-failure \
-- sleep 1000
```

## 查看任务列表

`baizectl job` 支持通过 `ls` 命令查看任务列表，默认显示 `pytorch` 任务，用户可以通过 `-t` 指定任务类型。

```
(base) jovyan@den-0:~$ baizectl job ls # 默认查看 pytorch 任务
NAME      TYPE      PHASE      DURATION   COMMAND
demong    PYTORCH  SUCCEEDED  1m2s       sleep 60
demo-sleep PYTORCH  RUNNING    1h25m28s  sleep 7200
(base) jovyan@den-0:~$ baizectl job ls demo-sleep # 查看指定任务
NAME      TYPE      PHASE      DURATION   COMMAND
demo-sleep PYTORCH  RUNNING    1h25m28s  sleep 7200
(base) jovyan@den-0:~$ baizectl job ls -t TENSORFLOW # 查看 tensorflow 任务
NAME      TYPE      PHASE      DURATION   COMMAND
```

```
demotfjob TENSORFLOW CREATED 0s           sleep 1000
```

任务列表默认情况下使用 table 作为展示形式，如果希望查看更多信息，可使用 json 或 yaml 格式展示，可以通过 -o 参数指定。

```
(base) jovyan@den-0:~$ baizectl job ls -t TENSORFLOW -o yaml
- baseConfig:
  args:
    - sleep
    - "1000"
  image: release.daocloud.io/baize/baize-notebook:v0.5.0
  labels:
    app: den
  podConfig:
    affinity: {}
    kubeEnvs:
      - name: CONDA_EXE
        value: /opt/conda/bin/conda
      - name: CONDA_PREFIX
        value: /opt/conda
      - name: CONDA_PROMPT_MODIFIER
        value: '(base)'
      - name: CONDA_SHLVL
        value: "1"
      - name: CONDA_DIR
        value: /opt/conda
      - name: CONDA_PYTHON_EXE
        value: /opt/conda/bin/python
      - name: CONDA_PYTHON_EXE
        value: /opt/conda/bin/python
      - name: CONDA_DEFAULT_ENV
        value: base
      - name: PATH
        value:
          /opt/conda/bin:/opt/conda/condabin:/command:/opt/conda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    priorityClass: baize-high-priority
    queue: default
  creationTimestamp: "2024-06-16T07:47:27Z"
  jobSpec:
    runPolicy:
      suspend: true
    tfReplicaSpecs:
      Worker:
        replicas: 1
```

```

restartPolicy: OnFailure
template:
  metadata:
    creationTimestamp: null
  spec:
    affinity: {}
    containers:
      - args:
          - sleep
          - "1000"
        env:
          - name: CONDA_EXE
            value: /opt/conda/bin/conda
          - name: CONDA_PREFIX
            value: /opt/conda
          - name: CONDA_PROMPT_MODIFIER
            value: '(base) '
          - name: CONDA_SHLVL
            value: "1"
          - name: CONDA_DIR
            value: /opt/conda
          - name: CONDA_PYTHON_EXE
            value: /opt/conda/bin/python
          - name: CONDA_PYTHON_EXE
            value: /opt/conda/bin/python
          - name: CONDA_DEFAULT_ENV
            value: base
          - name: PATH
            value:
        /opt/conda/bin:/opt/conda/condabin:/command:/opt/conda/bin:/usr/local/sbin:/usr/local/bin:/usr
        /sbin:/usr/bin:/sbin:/bin
        image: release.daocloud.io/baize/baize-notebook:v0.5.0
        name: tensorflow
      resources:
        limits:
          memory: 1Gi
        requests:
          cpu: "1"
          memory: 2Gi
      workingDir: /home/joyyan
      priorityClassName: baize-high-priority
    name: demotfjob
    namespace: ns-chuanjia-ndx
    phase: CREATED

```

```
roleConfig:  
  TF_WORKER:  
    replicas: 1  
    resources:  
      limits:  
        memory: 1Gi  
      requests:  
        cpu: "1"  
        memory: 2Gi  
  totalResources:  
    limits:  
      memory: "1073741824"  
    requests:  
      cpu: "1"  
      memory: "2147483648"  
  trainingConfig:  
    restartPolicy: RESTART_POLICY_ON_FAILURE  
  trainingMode: SINGLE  
  type: TENSORFLOW
```

## 查看任务日志

baizectl job 支持使用 logs 命令查看任务日志，用户可以通过 baizectl job logs --help 查看详细信息。

```
(base) jovyan@den-0:~$ baizectl job logs --help  
Show logs of a job
```

Usage:

```
baizectl job logs <job-name> [pod-name] [flags]
```

Aliases:

```
logs, log
```

Flags:

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| -f, --follow          | Specify if the logs should be streamed.                   |
| -h, --help            | help for logs                                             |
| -t, --job-type string | Job type: PYTORCH, TENSORFLOW, PADDLE (default "PYTORCH") |
| --paddle              | PaddlePaddle Job, has higher priority than --job-type     |
| --pytorch             | Pytorch Job, has higher priority than --job-type          |
| --tail int            | Lines of recent log file to display.                      |
| --tensorflow          | Tensorflow Job, has higher priority than --job-type       |
| --timestamps          | Show timestamps                                           |

## Note

- `--follow` 参数实时查看日志
- `--tail` 参数指定查看日志的行数，默认为 50 行
- `--timestamps` 参数显示时间戳

示例查看任务日志：

```
(base) jovyan@den-0:~$ baizectl job log -t TENSORFLOW tf-sample-job-v2-202406161632-  
evgrbrhn -f  
2024-06-16 08:33:06.083766: I tensorflow/core/util/port.cc:110] oneDNN custom operations  
are on. You may see slightly different numerical results due to floating-point round-off errors  
from different computation orders. To turn them off, set the environment variable  
'TF_ENABLE_ONEDNN_OPTS=0'.  
2024-06-16 08:33:06.086189: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda  
drivers on your machine, GPU will not be used.  
2024-06-16 08:33:06.132416: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda  
drivers on your machine, GPU will not be used.  
2024-06-16 08:33:06.132903: I tensorflow/core/platform/cpu_feature_guard.cc:182] This  
TensorFlow binary is optimized to use available CPU instructions in performance-critical  
operations.  
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other  
operations, rebuild TensorFlow with the appropriate compiler flags.  
2024-06-16 08:33:07.223046: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT  
Warning: Could not find TensorRT  
Model: "sequential"
```

| Layer (type)                        | Output Shape      | Param # |
|-------------------------------------|-------------------|---------|
| Conv1 (Conv2D)                      | (None, 13, 13, 8) | 80      |
| flatten (Flatten)                   | (None, 1352)      | 0       |
| Softmax (Dense)                     | (None, 10)        | 13530   |
| <hr/>                               |                   |         |
| Total params: 13610 (53.16 KB)      |                   |         |
| Trainable params: 13610 (53.16 KB)  |                   |         |
| Non-trainable params: 0 (0.00 Byte) |                   |         |
| ...                                 |                   |         |

## 删除任务

baizectl job 支持使用 `delete` 命令删除任务，并且同时支持删除多个任务。

```
(base) jovyan@den-0:~$ baizectl job delete --help
Delete a job
```

Usage:

```
baizectl job delete [flags]
```

Aliases:

```
delete, del, remove, rm
```

Flags:

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| -h, --help            | help for delete                                           |
| -t, --job-type string | Job type: PYTORCH, TENSORFLOW, PADDLE (default "PYTORCH") |
| --paddle              | PaddlePaddle Job, has higher priority than --job-type     |
| --pytorch             | Pytorch Job, has higher priority than --job-type          |
| --tensorflow          | Tensorflow Job, has higher priority than --job-type       |

示例删除任务：

```
(base) jovyan@den-0:~$ baizectl job ls
NAME      TYPE      PHASE      DURATION   COMMAND
demong    PYTORCH  SUCCEEDED  1m2s       sleep 60
demo-sleep PYTORCH  RUNNING    1h20m51s  sleep 7200
demojob    PYTORCH  FAILED     16m46s    sleep 1000
demojob-v2 PYTORCH  RUNNING    3m13s    sleep 1000
demojob-v3 PYTORCH  CREATED    0s       sleep 1000
(base) jovyan@den-0:~$ baizectl job delete demojob      # 删除单个任务
Delete job demojob in ns-chuanjia-ndx successfully
(base) jovyan@den-0:~$ baizectl job delete demojob-v2 demojob-v3      # 删除多个任务
Delete job demojob-v2 in ns-chuanjia-ndx successfully
Delete job demojob-v3 in ns-chuanjia-ndx successfully
```

## 重启任务

baizectl job 支持使用 `restart` 命令重启任务，用户可以通过 `baizectl job restart --help` 查看详细信息。

```
(base) jovyan@den-0:~$ baizectl job restart --help
restart a job
```

Usage:

```
baizectl job restart [flags] job
```

Aliases:

```
restart, rerun
```

Flags:

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| -h, --help            | help for restart                                          |
| -t, --job-type string | Job type: PYTORCH, TENSORFLOW, PADDLE (default "PYTORCH") |
| --paddle              | PaddlePaddle Job, has higher priority than --job-type     |
| --pytorch             | Pytorch Job, has higher priority than --job-type          |
| --tensorflow          | Tensorflow Job, has higher priority than --job-type       |

## 数据集管理

baizectl 支持管理数据集，目前支持查看数据集列表，方便在任务训练时，快速绑定数据集。

```
(base) jovyan@den-0:~$ baizectl data
Management datasets
```

Usage:

```
baizectl data [flags]
baizectl data [command]
```

Aliases:

```
data, dataset, datasets, envs, runtime-envs
```

Available Commands:

```
ls           List datasets
```

Flags:

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| -h, --help          | help for data                                                           |
| -o, --output string | Output format. One of: table, json, yaml (default "table")              |
| --page int          | Page number (default 1)                                                 |
| --page-size int     | Page size (default -1)                                                  |
| --search string     | Search query                                                            |
| --sort string       | Sort order                                                              |
| --truncate int      | Truncate output to the given length, 0 means no truncation (default 50) |

Use "baizectl data [command] --help" for more information about a command.

## 查看数据集列表

baizectl data 支持通过 ls 命令查看数据集列表， 默认显示 table 格式， 用户可以通过 -o 参数指定输出格式。

```
(base) jovyan@den-0:~$ baizectl data ls
  NAME      TYPE    URI
PHASE
fashion-mnist  GIT    https://gitee.com/samzong_lu/fashion-mnist.git      READY
sample-code     GIT    https://gitee.com/samzong_lu/training-sample-code....  READY
training-output PVC    pvc://training-output                                READY
```

在提交训练任务时，可以使用 -d 或者 --datasets 参数指定数据集，例如：

```
baizectl job submit --image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--datasets sample-code:/home/jovyan/code \
-- sleep 1000
```

同时挂载多个数据集，可以按照如下格式：

```
baizectl job submit --image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--datasets sample-code:/home/jovyan/code fashion-mnist:/home/jovyan/data \
-- sleep 1000
```

## 查看依赖库（环境）

环境 runtime-env 是 DCE 的特色环境管理能力，通过将模型开发、训练任务以及推理中所需的依赖库解耦， 提供了一种更加灵活的依赖库管理方式，无需重复构建复杂的 Docker 镜像，只需选择合适的环境即可。

同时 runtime-env 支持热更新，动态升级，无需重新构建镜像，即可更新环境依赖库。

baizectl data 支持通过 runtime-env 命令查看环境列表， 默认显示 table 格式， 用户可以通过 -o 参数指定输出格式。

```
(base) jovyan@den-0:~$ baizectl data ls --runtime-env
  NAME      TYPE    URI
PHASE
fashion-mnist  GIT    https://gitee.com/samzong_lu/fashion-mnist.git
READY
```

```
sample-code      GIT      https://gitee.com/samzong_lu/training-sample-code....  
READY  
training-output PVC      pvc://training-output  
READY  
tensorflow-sample CONDA  conda://python?version=3.12.3  
PROCESSING
```

在提交训练任务时，可以使用 `--runtime-env` 参数指定环境，例如：

```
baizectl job submit --image release.daocloud.io/baize/baize-notebook:v0.5.0 \  
--runtime-env tensorflow-sample \  
-- sleep 1000
```

## 高级用法

`baizectl` 支持更多高级用法，例如自动补全脚本生成、使用特定集群和命名空间、使用特定工作空间等。

### 自动补全脚本生成

```
baizectl completion bash > /etc/bash_completion.d/baizectl
```

上述命令生成 `bash` 的自动补全脚本，并将其保存到 `/etc/bash_completion.d/baizectl` 目录中，用户可以通过 `source /etc/bash_completion.d/baizectl` 加载自动补全脚本。

### 使用特定集群和命名空间

```
baizectl job ls --cluster my-cluster --namespace my-namespace
```

该命令将列出 `my-cluster` 集群中 `my-namespace` 命名空间下的所有作业。

### 使用特定工作空间

```
baizectl job ls --workspace 123
```

## 常见问题

- 问题：为什么无法连接到服务器？

**解决方法：**检查 `--server` 参数是否正确设置，并确保网络连接正常。如果服务器使用自签名证书，可以使用 `--skip-tls-verify` 跳过 TLS 证书验证。

- **问题：**如何解决权限不足的问题？

**解决方法：**确保使用正确的 `--token` 参数登录，并检查当前用户是否具有相应的操作权限。

- **问题：**为什么无法列出数据集？

**解决方法：**检查命名空间和工作区是否正确设置，确保当前用户有权限访问这些资源。

## 结语

通过以上指南，用户可以快速上手 `baizectl` 命令，并在实际应用中高效地管理 AI 平台资源。如果有任何疑问或问题，建议参考 `baizectl [command] --help` 获取更多详细信息。

# baizess 换源工具使用指南

`baizess` 是 DCE 5.0 AI Lab 模块中 Notebook 内置的开箱即用的换源小工具。它提供了简洁的命令行界面，方便用户管理各种编程环境的包管理器源。通过 `baizess`，用户可以轻松切换常用包管理器的源，确保顺利访问最新的库和依赖项。该工具通过简化包源管理流程，提升了开发者和数据科学家的工作效率。

目前，`baizess` 已经集成在 DCE 5.0 AI Lab 中。你在创建 Notebook 后，即可在 Notebook 中直接使用 `baizess`。

## 快速上手

`baizess` 命令的基本信息如下：

```
jovyan@19d0197587cc:$ baizess  
source switch tool
```

Usage:

```
baizess [command] [package-manager]
```

Available Commands:

```
set      Switch the source of specified package manager to current fastest source  
reset   Reset the source of specified package manager to default source
```

Available Package-managers:

apt (require root privilege)  
conda  
pip

## 命令格式

baizess 命令的基本格式如下：

```
baizess [command] [package-manager]
```

其中，[command] 是具体的操作命令，[package-manager] 用于指定操作对应的包管理器。

### command

- **set**: 备份源，测速，将所指定的包管理器的源切换为测速结果最快的国内源。
- **reset**: 将所指定的包管理器重置为默认源。

### 目前支持的 package-manager

- apt (源的切换与重置需要 root 权限)
- conda (原先的源将被备份在/etc/apt/backup/)
- pip (更新后源信息将被写入~/.condarc)

## Notebook 闲置超时自动关机

在默认情况下，为优化资源利用率，AI Lab 启用了 Notebook 闲置超时自动关机功能；当 Notebook 长时间无操作时，系统会自动关机 Notebook，释放资源。

- 优点：通过这种方式，可以极大减少因为长时间无操作导致的资源浪费，提高资源利用效率。
- 缺点：如果 Notebook 未配置相关备份策略，可能导致数据丢失。

### Note

当前，此功能为集群级别配置，默认开启，默认超时时长为 30 分钟。

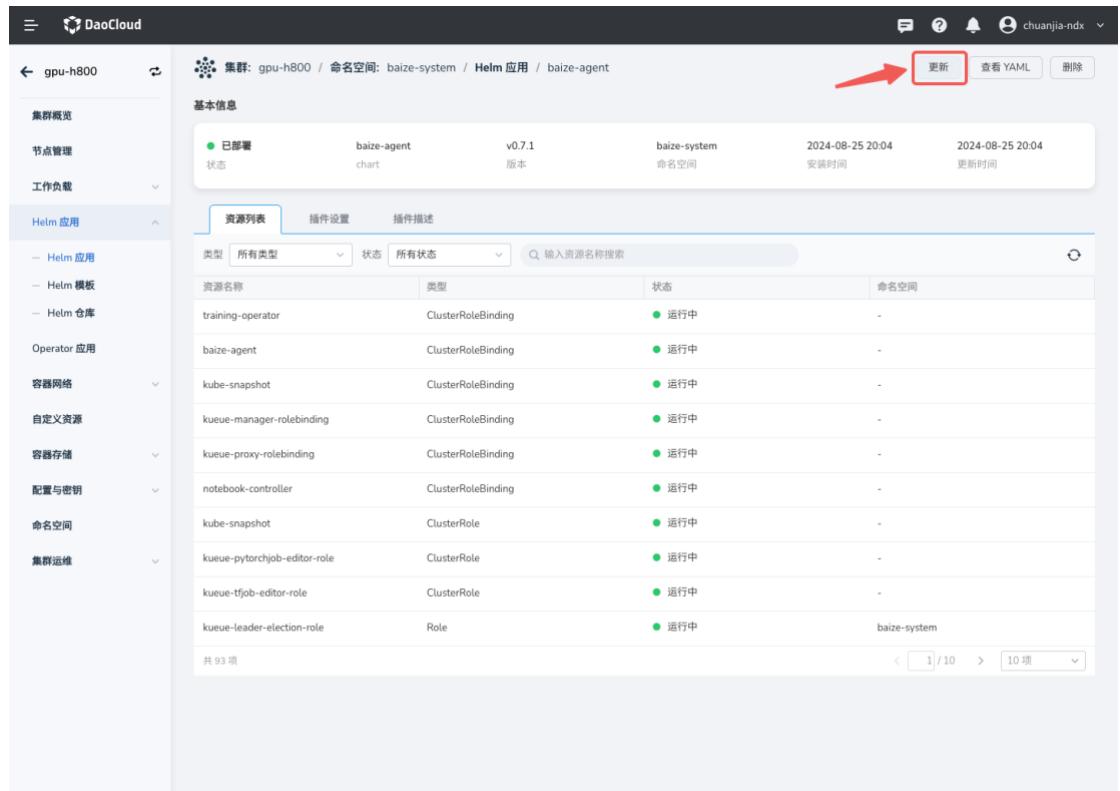
# 配置变更

目前配置修改方式为手动修改，后续会提供更加便捷的配置方式。

修改工作集群中 baize-agent 的部署参数，正确的修改方式为更新 Helm 应用，

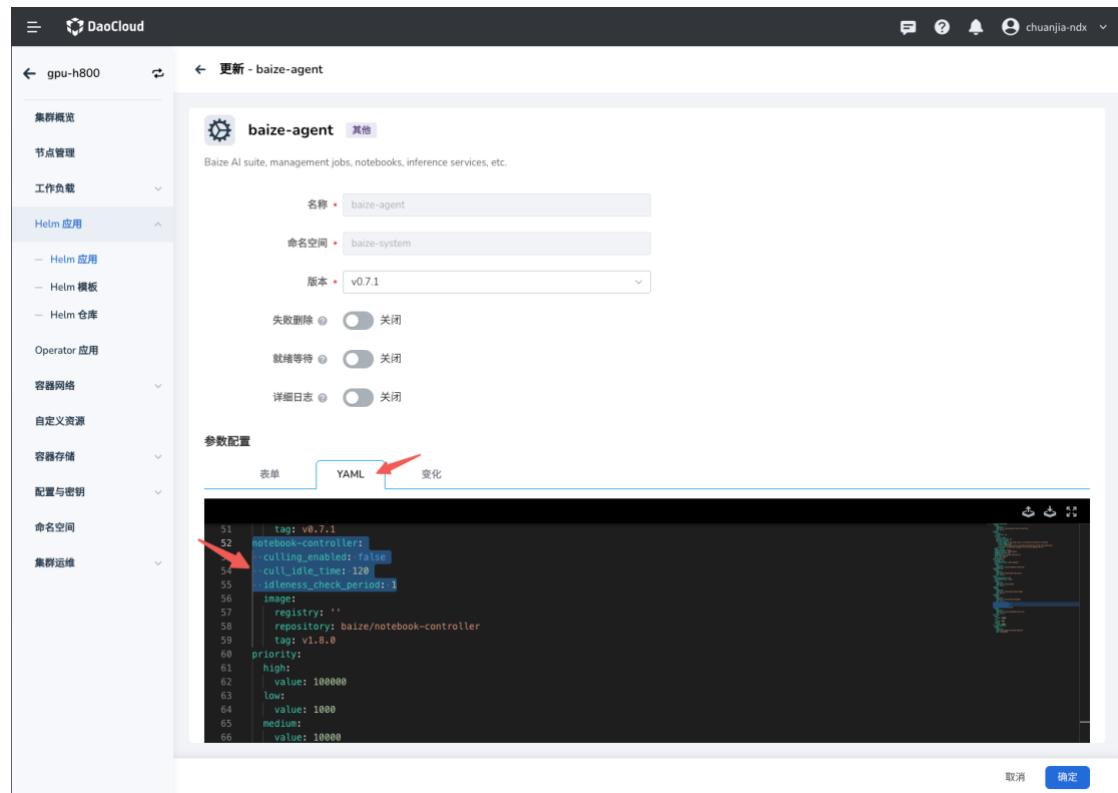
## 界面化修改

- 在集群管理界面找到对应的工作集群，进入集群详情，选择 **Helm 应用**，在 baize-system 命名空间下找到 baize-agent，在右上角点击 **更新** 按钮：



The screenshot shows the DaoCloud cluster management interface. On the left, there's a sidebar with options like '集群概览', '节点管理', '工作负载', 'Helm 应用' (which is currently selected), 'Helm 模板', and 'Helm 仓库'. In the main area, the path is '集群: gpu-h800 / 命名空间: baize-system / Helm 应用 / baize-agent'. The '基本信息' card shows the deployment status: '已部署' (baize-agent, chart v0.7.1, baize-system, 2024-08-25 20:04). Below it, the '资源列表' tab is active, displaying a table of resources. The table has columns for '资源名称', '类型', '状态', and '命名空间'. It lists several ClusterRoleBindings and ClusterRoles, all in '运行中' (running) state. The '命名空间' column shows 'baize-system' for the last item. At the bottom of the table, it says '共 93 项'.

- 如图修改 YAML 代码：



```

...
notebook-controller:
  culling_enabled: false
  cull_idle_time: 120
  idleness_check_period: 1
...

```

3. 确认参数修改成功后，点击 **下一步** 和 **确定**。

## 命令行修改

进入控制台以后，使用 `helm upgrade` 命令更改配置：

```

# 设定版本号
export VERSION=0.8.0

# 更新 Helm Chart
helm upgrade --install baize-agent baize/baize-agent \
--namespace baize-system \
--create-namespace \
--set global.imageRegistry=release.daocloud.io \
--set notebook-controller.culling_enabled=true \      # 开启自动关机，默认为 true
--set notebook-controller.cull_idle_time=120 \        # 设置闲置超时时间为 120 分钟,
默认为 30 分钟

```

```
--set notebook-controller.idleness_check_period=1 \# 设置检查间隔为 1 分钟，默认为  
1 分钟  
--version=$VERSION
```

## Note

为了避免自动关机后丢失数据，您可以将 AI Lab 升级到 v0.8.0 及更高版本，在 Notebook 配置中启用关机自动保存功能。

# 创建任务（Job）

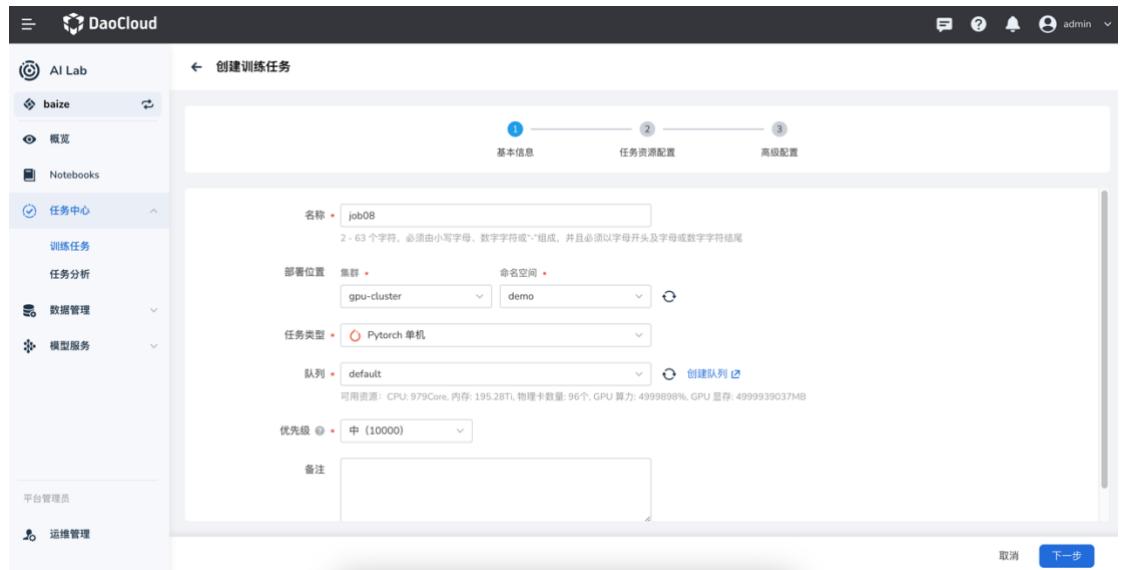
任务管理是指通过作业调度和管控组件来创建和管理任务生命周期的功能。

DCE 5.0 AI Lab 采用 Kubernetes 的 Job 机制来调度各项 AI 推理、训练任务。

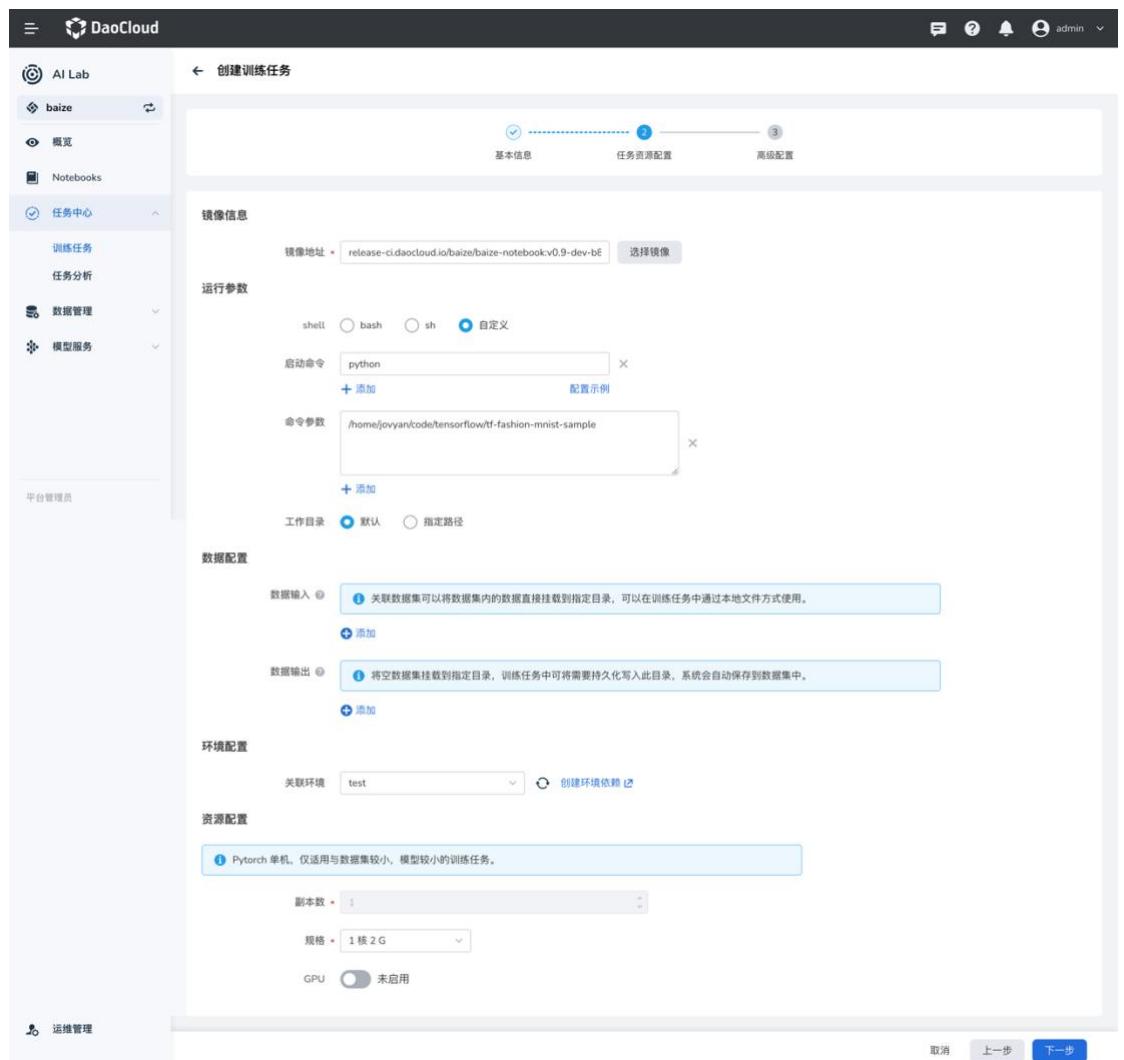
1. 在左侧导航栏中点击 任务中心 -> 训练任务，点击右侧的 创建 按钮。

| 名称                | 任务类型          | 状态    | 队列          | 优先级       | 资源配置         | 命名空间     | 创建时间             |
|-------------------|---------------|-------|-------------|-----------|--------------|----------|------------------|
| liyue-tensorflow  | Tensorflow 单机 | ● 暂停  | liyue-queue | 中 (10000) | cpu:1Core +1 | liyue-ns | 2024-09-13 16:13 |
| test-009          | Pytorch 单机    | ● 已创建 | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 16:31 |
| dfdfdf-2024091... | Pytorch 单机    | ● 已创建 | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:54 |
| dfdfdf            | Pytorch 单机    | ● 成功  | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:53 |
| cccvvvv           | Pytorch 单机    | ● 成功  | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:46 |
| fdcvcvc-202409... | Pytorch 单机    | ● 已创建 | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:34 |
| fdcvcvc           | Pytorch 单机    | ● 已创建 | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:33 |
| dfdf              | Pytorch 单机    | ● 成功  | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:26 |
| dfdfcvcv          | Pytorch 单机    | ● 成功  | default     | 中 (10000) | cpu:1Core +1 | demo     | 2024-09-12 14:19 |

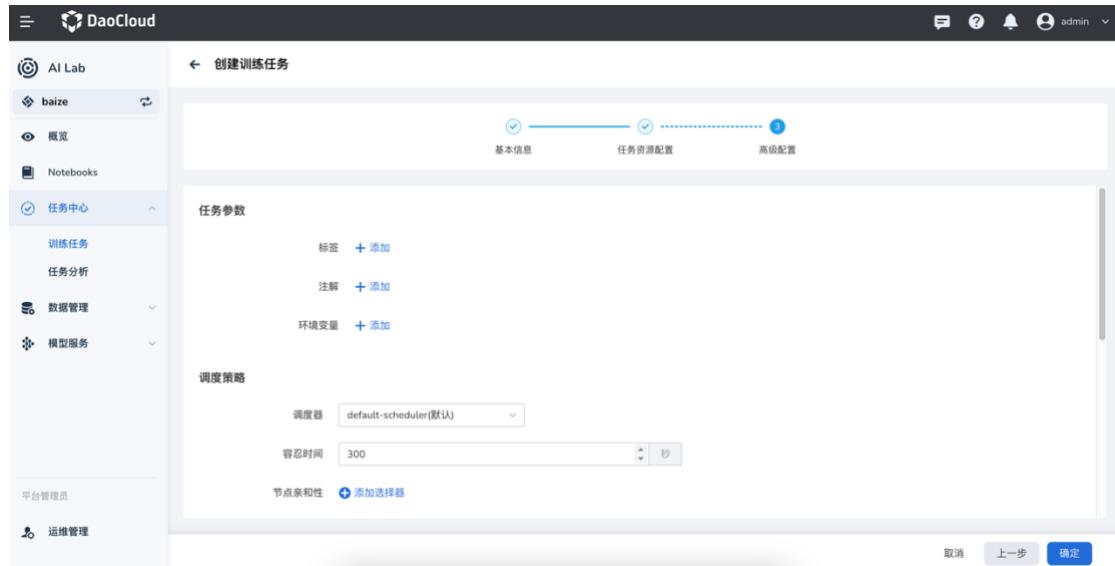
2. 系统会预先填充基础配置数据，包括要部署的集群、命名空间、任务类型、队列、优先级等。调整这些参数后点击 下一步 。



3. 配置镜像地址、运行参数以及关联的数据集、环境和资源后，点击 **下一步**。



4. 按需添加标签、注解、环境变量等任务参数，选择调度策略后点击确定。



5. 任务创建成功后，会有几种运行状态：

- 运行中
- 排队中
- 提交成功、提交失败
- 任务成功、任务失败

## Pytorch 任务

Pytorch 是一个开源的深度学习框架，它提供了一个灵活的训练和部署环境。Pytorch 任务是一个使用 Pytorch 框架的任务。

在 AI Lab 中，我们提供了 Pytorch 任务支持和适配，您可以通过界面化操作，快速创建 Pytorch 任务，进行模型训练。

## 任务配置介绍

- 任务类型同时支持 Pytorch 单机 和 Pytorch 分布式 两种模式。
- 运行镜像内已经默认支持 Pytorch 框架，无需额外安装。

## 任务运行环境

在这里我们使用 baize-notebook 基础镜像 和 关联环境 的方式来作为任务基础运行环境。

了解如何创建环境，请参考 [环境列表](#)。

## 创建任务

### Pytorch 单机任务

#### 运行参数

shell  bash  sh  自定义

启动命令 \*

-C

配置示例

命令参数

工作目录  默认  指定路径

#### 数据配置

通过数据集可以将远端数据（NFS、S3、http、git）挂载到本地使用，可以加速数据访问。

关联数据集 [+ 添加](#)

#### 环境配置

关联环境  [刷新](#) [创建环境](#)

#### 资源配置

Pytorch 单机，仅适用与数据集较小，模型较小的训练任务。

副本数 \*

规格 \*

GPU  启用

GPU 类型 \*  [刷新](#)

容器可使用算力 = 物理卡数量 \* 使用量

1. 登录 AI Lab 平台，点击左侧导航栏中的 **任务中心**，进入 **训练任务** 页面。
2. 点击右上角的 **创建** 按钮，进入任务创建页面。
3. 选择任务类型为 **Pytorch 单机**，点击 **下一步**。

4. 填写任务名称、描述后点击 确定 。

## 运行参数

- 启动命令 使用 bash
- 命令参数使用

```
import torch
import torch.nn as nn
import torch.optim as optim

# 定义一个简单的神经网络
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc = nn.Linear(10, 1)

    def forward(self, x):
        return self.fc(x)

# 创建模型、损失函数和优化器
model = SimpleNet()
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# 生成一些随机数据
x = torch.randn(100, 10)
y = torch.randn(100, 1)

# 训练模型
for epoch in range(100):
    # 前向传播
    outputs = model(x)
    loss = criterion(outputs, y)

    # 反向传播和优化
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/100], Loss: {loss.item():.4f}')

print('Training finished.')
```

## 运行结果

任务提交成功，我们可以进入任务详情查看到资源的使用情况，从右上角去往 **工作负载详情**，可以查看训练过程中的日志输出

```
[HAMI-core Warn(1:140244541377408:utils.c:183)]: get default cuda from (null)
[HAMI-core Msg(1:140244541377408:libvgpu.c:855)]: Initialized
Epoch [10/100], Loss: 1.1248
Epoch [20/100], Loss: 1.0486
Epoch [30/100], Loss: 0.9969
Epoch [40/100], Loss: 0.9611
Epoch [50/100], Loss: 0.9360
Epoch [60/100], Loss: 0.9182
Epoch [70/100], Loss: 0.9053
Epoch [80/100], Loss: 0.8960
Epoch [90/100], Loss: 0.8891
Epoch [100/100], Loss: 0.8841
Training finished.
[HAMI-core Msg(1:140244541377408:multiprocess_memory_limit.c:468)]: Calling exit handler
1
```

## Pytorch 分布式任务

1. 登录 AI Lab 平台，点击左侧导航栏中的 **任务中心**，进入 **任务列表** 页面。
2. 点击右上角的 **创建** 按钮，进入任务创建页面。
3. 选择任务类型为 Pytorch 分布式，点击 **下一步**。
4. 填写任务名称、描述后点击 **确定**。

## 运行参数

- 启动命令 使用 bash
- 命令参数使用

```
import os
import torch
import torch.distributed as dist
import torch.nn as nn
import torch.optim as optim
from torch.nn.parallel import DistributedDataParallel as DDP

class SimpleModel(nn.Module):
```

```

def __init__(self):
    super(SimpleModel, self).__init__()
    self.fc = nn.Linear(10, 1)

def forward(self, x):
    return self.fc(x)

def train():
    # 打印环境信息
    print(f'PyTorch version: {torch.__version__}')
    print(f'CUDA available: {torch.cuda.is_available()}')
    if torch.cuda.is_available():
        print(f'CUDA version: {torch.version.cuda}')
        print(f'CUDA device count: {torch.cuda.device_count()}')

    rank = int(os.environ.get('RANK', '0'))
    world_size = int(os.environ.get('WORLD_SIZE', '1'))

    print(f'Rank: {rank}, World Size: {world_size}')

    # 初始化分布式环境
    try:
        if world_size > 1:
            dist.init_process_group('nccl')
            print('Distributed process group initialized successfully')
        else:
            print('Running in non-distributed mode')
    except Exception as e:
        print(f'Error initializing process group: {e}')
        return

    # 设置设备
    try:
        if torch.cuda.is_available():
            device = torch.device(f'cuda:{rank % torch.cuda.device_count()}')
            print(f'Using CUDA device: {device}')
        else:
            device = torch.device('cpu')
            print('CUDA not available, using CPU')
    except Exception as e:
        print(f'Error setting device: {e}')
        device = torch.device('cpu')
        print('Falling back to CPU')

```

```

try:
    model = SimpleModel().to(device)
    print('Model moved to device successfully')
except Exception as e:
    print(f'Error moving model to device: {e}')
    return

try:
    if world_size > 1:
        ddp_model = DDP(model, device_ids=[rank % torch.cuda.device_count()] if
torch.cuda.is_available() else None)
        print('DDP model created successfully')
    else:
        ddp_model = model
        print('Using non-distributed model')
except Exception as e:
    print(f'Error creating DDP model: {e}')
    return

loss_fn = nn.MSELoss()
optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)

# 生成一些随机数据
try:
    data = torch.randn(100, 10, device=device)
    labels = torch.randn(100, 1, device=device)
    print('Data generated and moved to device successfully')
except Exception as e:
    print(f'Error generating or moving data to device: {e}')
    return

for epoch in range(10):
    try:
        ddp_model.train()
        outputs = ddp_model(data)
        loss = loss_fn(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if rank == 0:
            print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
    except Exception as e:
        print(f'Error during training epoch {epoch}: {e}')

```

```
break

if world_size > 1:
    dist.destroy_process_group()

if __name__ == '__main__':
    train()
```

## 任务副本数

注意 Pytorch 分布式训练任务会创建一组 Master 和 Worker 的训练 Pod，Master 负责协调训练任务，Worker 负责实际的训练工作。

### Note

本次演示中：Master 副本数为 1，Worker 副本数为 2；所以我们需要在 **任务配置** 中设置副本数为 3，即 Master 副本数 + Worker 副本数。Pytorch 会自动调谐 Master 和 Worker 的角色。

## 运行结果

同样，我们可以进入任务详情，查看资源的使用情况，以及每个 Pod 的日志输出。

## Tensorflow 任务

Tensorflow 是除了 Pytorch 另外一个非常活跃的开源的深度学习框架，它提供了一个灵活的训练和部署环境。

在 AI Lab 中，我们同样提供了 Tensorflow 框架的支持和适配，您可以通过界面化操作，快速创建 Tensorflow 任务，进行模型训练。

## 任务配置介绍

- 任务类型同时支持 Tensorflow 单机 和 Tensorflow 分布式 两种模式。
- 运行镜像内已经默认支持 Tensorflow 框架，无需额外安装。

## 任务运行环境

在这里我们使用 baize-notebook 基础镜像 和 关联环境 的方式来作为任务基础运行环境。

了解如何创建环境, 请参考 [环境列表](#)。

# 创建任务

## 示例 TFJob 单机任务

镜像地址 \*

运行参数

shell  bash  sh  自定义

启动命令 \*   
-c

命令参数

工作目录  默认  指定路径

数据配置

通过数据集可以将远端数据 (NFS、S3、http、git) 挂载到本地使用，可以加速数据访问。

关联数据集

数据集 \*

挂载路径 \*

+ 添加

环境配置

关联环境

资源配置

Tensorflow 单机，适用于数据集小、模型小的训练任务。

副本数 \*

规格 \*

GPU  启用

1. 登录 AI Lab 平台，点击左侧导航栏中的 任务中心，进入 训练任务 页面。
2. 点击右上角的 创建 按钮，进入任务创建页面。
3. 选择任务类型为 Tensorflow 单机，点击 下一步 。
4. 填写任务名称、描述后点击 确定 。

## 提前预热代码仓库

使用 **AI Lab -> 数据集列表**， 创建一个数据集，并将远端 Github 的代码拉取到数据集中，这样在创建任务时，可以直接选择数据集，将代码挂载到任务中。

演示代码仓库地址：<https://github.com/d-run/training-sample-code/>

## 运行参数

- 启动命令 使用 bash
  - 命令参数使用 python /code/tensorflow/tf-single.py
- =====  
pip install tensorflow numpy  
=====
- ```
import tensorflow as tf
import numpy as np

# 创建一些随机数据
x = np.random.rand(100, 1)
y = 2 * x + 1 + np.random.rand(100, 1) * 0.1

# 创建一个简单的模型
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_shape=(1,))
])

# 编译模型
model.compile(optimizer='adam', loss='mse')

# 训练模型，将 epochs 改为 10
history = model.fit(x, y, epochs=10, verbose=1)

# 打印最终损失
print('Final loss: {} + str(history.history['loss'])[-1]) +')'

# 使用模型进行预测
test_x = np.array([[0.5]])
prediction = model.predict(test_x)
print(f'Prediction for x=0.5: {prediction[0][0]}'
```

## 运行结果

任务提交成功后，可以进入任务详情查看到资源的使用情况，从右上角去往 **工作负载详情**，可以查看训练过程中的日志输出。

## TFJob 分布式任务

1. 登录 **AI Lab**，点击左侧导航栏中的 **任务中心**，进入 **任务列表** 页面。
2. 点击右上角的 **创建** 按钮，进入任务创建页面。
3. 选择任务类型为 **Tensorflow** 分布式，点击 **下一步**。
4. 填写任务名称、描述后点击 **确定**。

## 示例任务介绍

**镜像信息**

镜像地址 • release-ci.daocloud.io/baize/baize-notebook:v0.8-dev-ef752fd3

**运行参数**

shell  bash  sh  自定义

启动命令 • /bin/bash  
-c

配置示例

命令参数 `python /code/tensorflow/tensorflow-distributed.py`

工作目录  默认  指定路径

**数据配置**

通过数据集可以将远端数据 (NFS、S3、http、git) 挂载到本地使用，可以加速数据访问。

关联数据集

数据集 • training-sample-code

挂载路径 • /code

**环境配置**

关联环境 tensorflow

**资源配置**

**节点配置**

Tensorflow 分布式任务，可以自行添加 PS、Chief、Evaluator 角色节点。

**WORKER**

副本数 • 2  
规格 • 1 核 2 G  
GPU  启用  
GPU 类型 • Nvidia vGPU  
物理卡数量 • 1 个  
GPU 算力 • 50 %  
GPU 显存 • 200 MB

**CHIEF**

副本数 • 1  
规格 • 1 核 2 G  
GPU  启用  
GPU 类型 • Nvidia vGPU  
物理卡数量 • 1 个  
GPU 算力 • 50 %  
GPU 显存 • 200 MB

**PS**

副本数 • 1  
规格 • 2 核 4 G  
GPU  关闭

+ 添加任务角色

**共享配置**

共享内存  关闭

本次包含了三种角色： Chief、 Worker 和 Parameter Server (PS)。

- Chief: 主要负责协调训练过程和模型检查点的保存。
- Worker: 执行实际的模型训练。
- PS: 在异步训练中用于存储和更新模型参数。

为不同的角色分配了不同的资源。Chief 和 Worker 使用 GPU，而 PS 使用 CPU 和较大的内存。

## 运行参数

- 启动命令 使用 bash
- 命令参数使用 python /code/tensorflow/tensorflow-distributed.py

```
import os
import json
import tensorflow as tf

class SimpleModel(tf.keras.Model):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc = tf.keras.layers.Dense(1, input_shape=(10,))

    def call(self, x):
        return self.fc(x)

    def train():
        # 打印环境信息
        print(f"TensorFlow version: {tf.__version__}")
        print(f"GPU available: {tf.test.is_gpu_available()}")
        if tf.test.is_gpu_available():
            print(f"GPU device count: {len(tf.config.list_physical_devices('GPU'))}")

        # 获取分布式训练信息
        tf_config = json.loads(os.environ.get('TF_CONFIG') or '{}')
        task_type = tf_config.get('task', {}).get('type')
        task_id = tf_config.get('task', {}).get('index')

        print(f"Task type: {task_type}, Task ID: {task_id}")

        # 设置分布式策略
        strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()

        with strategy.scope():
```

```

model = SimpleModel()
loss_fn = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)

# 生成一些随机数据
data = tf.random.normal((100, 10))
labels = tf.random.normal((100, 1))

@tf.function
def train_step(inputs, labels):
    with tf.GradientTape() as tape:
        predictions = model(inputs)
        loss = loss_fn(labels, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss

for epoch in range(10):
    loss = train_step(data, labels)
    if task_type == 'chief':
        print(f'Epoch {epoch}, Loss: {loss.numpy():.4f}')

if __name__ == '__main__':
    train()

```

## 运行结果

同样，我们可以进入任务详情，查看资源的使用情况，以及每个 Pod 的日志输出。

# MPI 任务

**MPI** (Message Passing Interface) 是一种用于并行计算的通信协议，它允许多个计算节点之间进行消息传递和协作。MPI 任务是使用 MPI 协议进行并行计算的任务，适用于需要大规模并行处理的应用场景，例如分布式训练、科学计算等。

在 AI Lab 中，我们提供了 MPI 任务的支持，您可以通过界面化操作，快速创建 MPI 任务，进行高性能的并行计算。本教程将指导您如何在 AI Lab 中创建和运行一个 MPI 任务。

# 任务配置介绍

- **任务类型**：MPI，用于运行并行计算任务。
- **运行环境**：选用预装了 MPI 环境的镜像，或者在任务中指定安装必要的依赖。
- **MPIJob 配置**：理解并配置 MPIJob 的各项参数，如副本数、资源请求等。

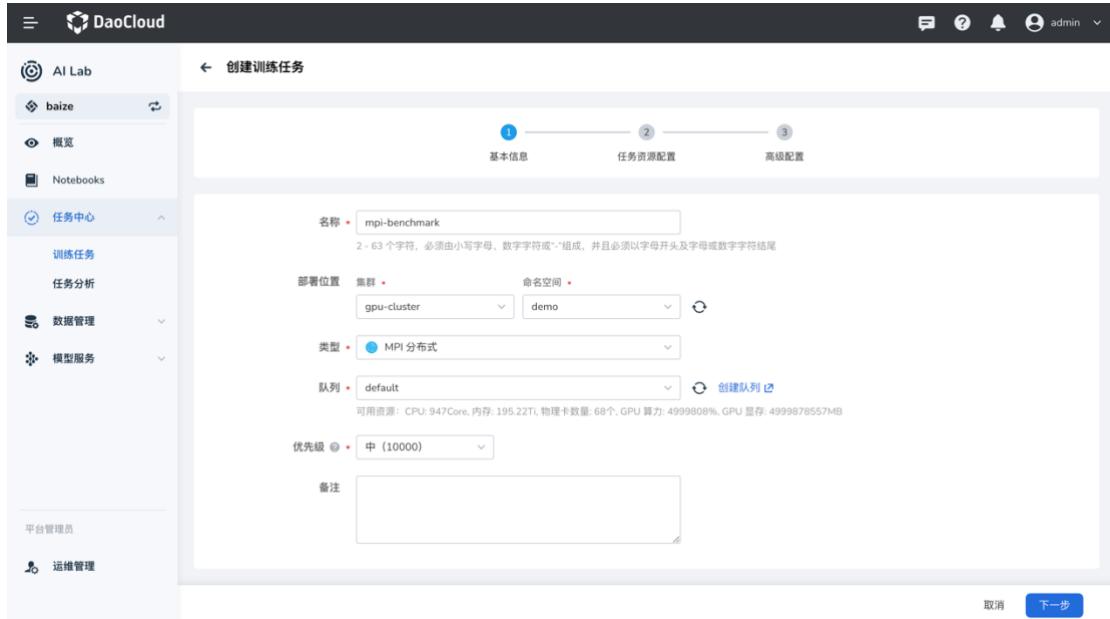
# 任务运行环境

在这里我们使用 baize-notebook 基础镜像和 **关联环境** 的方式来作为任务的基础运行环境。确保运行环境中包含 MPI 及相关库，如 OpenMPI、mpi4py 等。

**注意：**了解如何创建环境，请参考[环境列表](#)。

# 创建 MPI 任务

## MPI 任务创建步骤



1. **登录平台**：登录 AI Lab 平台，点击左侧导航栏中的**任务中心**，进入**训练任务**页面。
2. **创建任务**：点击右上角的**创建**按钮，进入任务创建页面。
3. **选择任务类型**：在弹出的窗口中，选择任务类型为**MPI**，然后点击**下一步**。

4. 填写任务信息：填写任务名称和描述，例如“benchmarks-mpi”，然后点击下一步。
5. 配置任务参数：根据您的需求，配置任务的运行参数、镜像、资源等信息。

## 运行参数

- 启动命令：使用 `mpirun`，这是运行 MPI 程序的命令。
- 命令参数：输入您要运行的 MPI 程序的参数。

### 示例：运行 TensorFlow Benchmarks

在本示例中，我们将运行一个 TensorFlow 的基准测试程序，使用 Horovod 进行分布式训练。首先，确保您使用的镜像中包含所需的依赖项，例如 TensorFlow、Horovod、Open MPI 等。

**镜像选择：** 使用包含 TensorFlow 和 MPI 的镜像，例如 `mai.daocloud.io/docker.io/mpoperator/tensorflow-benchmarks:latest`。

### 命令参数：

```
mpirun --allow-run-as-root -np 2 -bind-to none -map-by slot \
-x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH \
-mca pml ob1 -mca btl ^openib \
python scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py \
--model=resnet101 --batch_size=64 --variable_update=horovod
```

### 说明：

- `mpirun`: MPI 的启动命令。
- `--allow-run-as-root`: 允许以 `root` 用户运行（在容器中通常是 `root` 用户）。
- `-np 2`: 指定运行的进程数为 2。
- `-bind-to none, -map-by slot`: MPI 进程绑定和映射的配置。
- `-x NCCL_DEBUG=INFO`: 设置 NCCL (NVIDIA Collective Communication Library) 的调试信息级别。
- `-x LD_LIBRARY_PATH, -x PATH`: 在 MPI 环境中传递必要的环境变量。
- `-mca pml ob1 -mca btl ^openib`: MPI 的配置参数，指定传输层和消息层协议。
- `python scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py`: 运行 TensorFlow 基准测试脚本。

- `--model=resnet101, --batch_size=64, --variable_update=horovod`: TensorFlow 脚本的参数，指定模型、批量大小和使用 Horovod 进行参数更新。

## 资源配置

在任务配置中，需要为每个节点（Launcher 和 Worker）分配适当的资源，例如 CPU、内存和 GPU。

资源示例：

- **Launcher**（启动器）：
  - 副本数：1
  - 资源请求：
    - CPU: 2 核
    - 内存: 4 GiB
- **Worker**（工作节点）：
  - 副本数：2
  - 资源请求：
    - CPU: 2 核
    - 内存: 4 GiB
    - GPU: 根据需求分配

## 完整的 MPIJob 配置示例

以下是完整的 MPIJob 配置示例，供您参考。

```
apiVersion: kubeflow.org/v1
kind: MPIJob
metadata:
  name: tensorflow-benchmarks
spec:
  slotsPerWorker: 1
  runPolicy:
    cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
```

```

containers:
  - name: tensorflow-benchmarks
    image: mai.daocloud.io/docker.io/mpiooperator/tensorflow-benchmarks:latest
    command:
      - mpirun
      - --allow-run-as-root
      - -np
      - "2"
      - -bind-to
      - none
      - -map-by
      - slot
      - -x
      - NCCL_DEBUG=INFO
      - -x
      - LD_LIBRARY_PATH
      - -x
      - PATH
      - -mca
      - pml
      - ob1
      - -mca
      - btl
      - ^openib
      - python
      - scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py
      - --model=resnet101
      - --batch_size=64
      - --variable_update=horovod
resources:
limits:
  cpu: "2"
  memory: 4Gi
requests:
  cpu: "2"
  memory: 4Gi
Worker:
replicas: 2
template:
spec:
  containers:
    - name: tensorflow-benchmarks
      image: mai.daocloud.io/docker.io/mpiooperator/tensorflow-benchmarks:latest
resources:

```

```
limits:  
  cpu: "2"  
  memory: 4Gi  
  nvidia.com/gpumem: 1k  
  nvidia.com/vgpu: "1"  
requests:  
  cpu: "2"  
  memory: 4Gi
```

## 配置解析：

- **apiVersion** 和 **kind**: 表示资源的 API 版本和类型, MPIJob 是 Kubeflow 定义的自定义资源, 用于创建 MPI 类型的任务。
- **metadata**: 元数据, 包含任务的名称等信息。
- **spec**: 任务的详细配置。
  - **slotsPerWorker**: 每个 Worker 节点的槽位数量, 通常设置为 1。
  - **runPolicy**: 运行策略, 例如任务完成后是否清理 Pod。
  - **mpiReplicaSpecs**: MPI 任务的副本配置。
    - **Launcher**: 启动器, 负责启动 MPI 任务。
      - **replicas**: 副本数, 通常为 1。
      - **template**: Pod 模板, 定义容器运行的镜像、命令、资源等。
    - **Worker**: 工作节点, 实际执行任务的计算节点。
      - **replicas**: 副本数, 根据并行需求设置, 这里设置为 2。
      - **template**: Pod 模板, 同样定义容器的运行环境和资源。

## 设置任务副本数

在创建 MPI 任务时, 需要根据 **mpiReplicaSpecs** 中配置的副本数, 正确设置 **任务副本数**。

- 总副本数 = Launcher 副本数 + Worker 副本数
- 本示例中：
  - Launcher 副本数: 1
  - Worker 副本数: 2
  - 总副本数 :  $1 + 2 = 3$

因此，在任务配置中，您需要将 **任务副本数** 设置为 **3**。

## 提交任务

配置完成后，点击 **提交** 按钮，开始运行 **MPI** 任务。

## 查看运行结果

任务提交成功后，您可以进入 **任务详情** 页面，查看资源的使用情况和任务的运行状态。从右上角进入 **工作负载详情**，可以查看运行过程中每个节点的日志输出。

**示例输出：**

```
TensorFlow: 1.13
Model:      resnet101
Mode:       training
Batch size: 64
...

```

```
Total images/sec: 125.67
```

这表示 **MPI** 任务成功运行，**TensorFlow** 基准测试程序完成了分布式训练。

---

## 小结

通过本教程，您学习了如何在 **AI Lab** 平台上创建和运行一个 **MPI** 任务。我们详细介绍了 **MPIJob** 的配置方式，以及如何在任务中指定运行的命令和资源需求。希望本教程对您有所帮助，如有任何问题，请参考平台提供的其他文档或联系技术支持。

---

## 附录：

- 如果您的运行环境未预装所需的库（如 `mpi4py`、`Horovod` 等），请在任务中添加安装命令，或者使用预装了相关依赖的镜像。
- 在实际应用中，您可以根据需求修改 **MPIJob** 的配置，例如更改镜像、命令参数、资源请求等。

# MXNet 任务

## Warning

由于 Apache MXNet 项目已存档，因此 **Kubeflow MXJob** 将在未来的 **Training Operator 1.9** 版本中弃用和删除。

Apache MXNet 是一个高性能的深度学习框架，支持多种编程语言。**MXNet** 任务可以使用多种方式进行训练，包括单机模式和分布式模式。在 **AI Lab** 中，我们提供了对 **MXNet** 任务的支持，您可以通过界面化操作，快速创建 **MXNet** 任务，进行模型训练。

本教程将指导您如何在 **AI Lab** 平台上创建和运行 **MXNet** 的单机和分布式任务。

## 任务配置介绍

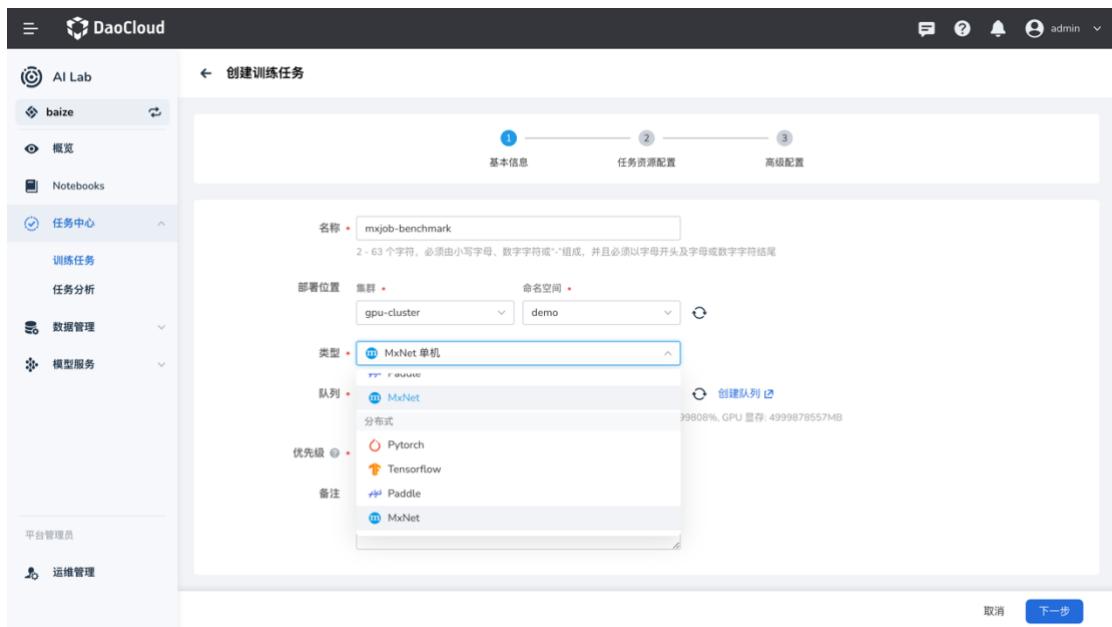
- **任务类型:** **MXNet**，支持单机和分布式两种模式。
- **运行环境:** 选择包含 **MXNet** 框架的镜像，或在任务中安装必要的依赖。

## 任务运行环境

我们使用 `release-ci.daocloud.io/baize/kubeflow/mxnet-gpu:latest` 镜像作为任务的基础运行环境。该镜像预装了 **MXNet** 及其相关依赖，支持 **GPU** 加速。

**注意：**了解如何创建和管理环境，请参考 [环境列表](#)。

# 创建 MXNet 任务



## MXNet 单机任务

### 创建步骤

1. **登录平台:** 登录 AI Lab 平台，点击左侧导航栏中的 **任务中心**，进入 **训练任务** 页面。
2. **创建任务:** 点击右上角的 **创建** 按钮，进入任务创建页面。
3. **选择任务类型:** 在弹出的窗口中，选择任务类型为 **MXNet**，然后点击 **下一步**。
4. **填写任务信息:** 填写任务名称和描述，例如“**MXNet 单机训练任务**”，然后点击 **确定**。
5. **配置任务参数:** 根据您的需求，配置任务的运行参数、镜像、资源等信息。

### 运行参数

- 启动命令: `python3`
- 命令参数:  
`./mxnet/mxnet/example/gluon/mnist/mnist.py --epochs 10 --cuda`

说明:

- `/mxnet/mxnet/example/gluon/mnist/mnist.py`: MXNet 提供的 MNIST 手写数字识别示例脚本。
- `--epochs 10`: 设置训练轮数为 10。
- `--cuda`: 使用 CUDA 进行 GPU 加速。

## 资源配置

- 副本数: 1 (单机任务)
- 资源请求:
  - CPU: 2 核
  - 内存: 4 GiB
  - GPU: 1 块

## 完整的 MXJob 配置示例

以下是单机 MXJob 的 YAML 配置:

```
apiVersion: "kubeflow.org/v1"
kind: "MXJob"
metadata:
  name: "mxnet-single-job"
spec:
  jobMode: MXTrain
  mxReplicaSpecs:
    Worker:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: release-ci.daocloud.io/baize/kubeflow/mxnet-gpu:latest
              command: ["python3"]
              args:
                [
                  "/mxnet/mxnet/example/gluon/mnist/mnist.py",
                  "--epochs",
                  "10",
                  "--cuda",
                ]
  ports:
```

```
- containerPort: 9991
  name: mxjob-port
resources:
limits:
  cpu: "2"
  memory: 4Gi
  nvidia.com/gpu: 1
requests:
  cpu: "2"
  memory: 4Gi
  nvidia.com/gpu: 1
```

## 配置解析:

- **apiVersion** 和 **kind**: 指定资源的 API 版本和类型，这里是 MXJob。
- **metadata**: 元数据，包括任务名称等信息。
- **spec**: 任务的详细配置。
  - **jobMode**: 设置为 MXTrain，表示训练任务。
  - **mxReplicaSpecs**: MXNet 任务的副本配置。
    - **Worker**: 指定工作节点的配置。
      - **replicas**: 副本数，这里为 1。
      - **restartPolicy**: 重启策略，设为 Never，表示任务失败时不重启。
      - **template**: Pod 模板，定义容器的运行环境和资源。
        - **containers**: 容器列表。
          - **name**: 容器名称。
          - **image**: 使用的镜像。
          - **command** 和 **args**: 启动命令和参数。
        - **ports**: 容器端口配置。
        - **resources**: 资源请求和限制。

## 提交任务

配置完成后，点击 **提交** 按钮，开始运行 **MXNet** 单机任务。

## 查看运行结果

任务提交成功后，您可以进入 **任务详情** 页面，查看资源的使用情况和任务的运行状态。从右上角进入 **工作负载详情**，可以查看运行过程中的日志输出。

**示例输出：**

```
Epoch 1: accuracy=0.95
Epoch 2: accuracy=0.97
...
Epoch 10: accuracy=0.98
Training completed.
```

这表示 **MXNet** 单机任务成功运行，模型训练完成。

---

## MXNet 分布式任务

在分布式模式下，**MXNet** 任务可以使用多台计算节点共同完成训练，提高训练效率。

## 创建步骤

1. **登录平台：** 同上。
2. **创建任务：** 点击右上角的 **创建** 按钮，进入任务创建页面。
3. **选择任务类型：** 选择任务类型为 **MXNet**，然后点击 **下一步**。
4. **填写任务信息：** 填写任务名称和描述，例如“**MXNet 分布式训练任务**”，然后点击 **确定**。
5. **配置任务参数：** 根据需求，配置运行参数、镜像、资源等。

## 运行参数

- **启动命令：** python3

- 命令参数:

```
• ./mxnet/mxnet/example/image-classification/train_mnist.py --num-epochs 10 --num-layers 2 --kv-store dist_device_sync --gpus 0
```

说明:

- `/mxnet/mxnet/example/image-classification/train_mnist.py`: MXNet 提供的图像分类示例脚本。
- `--num-epochs 10`: 训练轮数为 10。
- `--num-layers 2`: 模型的层数为 2。
- `--kv-store dist_device_sync`: 使用分布式设备同步模式。
- `--gpus 0`: 使用 GPU 进行加速。

## 资源配置

- 任务副本数: 3 (包括 Scheduler、Server 和 Worker)

- 各角色资源请求:

- **Scheduler** (调度器):

- 副本数: 1
    - 资源请求:
      - CPU: 2 核
      - 内存: 4 GiB
      - GPU: 1 块

- **Server** (参数服务器):

- 副本数: 1
    - 资源请求:
      - CPU: 2 核
      - 内存: 4 GiB
      - GPU: 1 块

- **Worker** (工作节点):

- 副本数: 1
    - 资源请求:
      - CPU: 2 核
      - 内存: 4 GiB
      - GPU: 1 块

## 完整的 MXJob 配置示例

以下是分布式 MXJob 的 YAML 配置：

```
apiVersion: "kubeflow.org/v1"
kind: "MXJob"
metadata:
  name: "mxnet-job"
spec:
  jobMode: MXTrain
  mxReplicaSpecs:
    Scheduler:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: release-ci.daocloud.io/baize/kubeflow/mxnet-gpu:latest
              ports:
                - containerPort: 9991
                  name: mxjob-port
              resources:
                limits:
                  cpu: "2"
                  memory: 4Gi
                  nvidia.com/gpu: 1
                requests:
                  cpu: "2"
                  memory: 4Gi
    Server:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: release-ci.daocloud.io/baize/kubeflow/mxnet-gpu:latest
              ports:
                - containerPort: 9991
                  name: mxjob-port
              resources:
                limits:
                  cpu: "2"
```

```

        memory: 4Gi
        nvidia.com/gpu: 1
      requests:
        cpu: "2"
        memory: 4Gi
    Worker:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: mxnet
              image: release-ci.daocloud.io/baize/kubeflow/mxnet-gpu:latest
              command: ["python3"]
              args:
                [
                  "/mxnet/mxnet/example/image-classification/train_mnist.py",
                  "--num-epochs",
                  "10",
                  "--num-layers",
                  "2",
                  "--kv-store",
                  "dist_device_sync",
                  "--gpus",
                  "0",
                ]
          ports:
            - containerPort: 9991
              name: mxjob-port
          resources:
            limits:
              cpu: "2"
              memory: 4Gi
              nvidia.com/gpu: 1
            requests:
              cpu: "2"
              memory: 4Gi

```

## 配置解析:

- **Scheduler (调度器):** 负责协调集群中各节点的任务调度。
- **Server (参数服务器):** 用于存储和更新模型参数，实现分布式参数同步。
- **Worker (工作节点):** 实际执行训练任务。

- **资源配置**: 为各角色分配适当的资源，确保任务顺利运行。

## 设置任务副本数

在创建 MXNet 分布式任务时，需要根据 `mxReplicaSpecs` 中配置的副本数，正确设置 **任务副本数**。

- 总副本数 = Scheduler 副本数 + Server 副本数 + Worker 副本数
- 本示例中：
  - Scheduler 副本数: 1
  - Server 副本数: 1
  - Worker 副本数: 1
  - 总副本数:  $1 + 1 + 1 = 3$

因此，在任务配置中，需要将 **任务副本数** 设置为 **3**。

## 提交任务

配置完成后，点击 **提交** 按钮，开始运行 MXNet 分布式任务。

## 查看运行结果

进入 **任务详情** 页面，查看任务的运行状态和资源使用情况。您可以查看每个角色（Scheduler、Server、Worker）的日志输出。

示例输出：

```
INFO:root:Epoch[0] Batch [50]      Speed: 1000 samples/sec   accuracy=0.85
INFO:root:Epoch[0] Batch [100]     Speed: 1200 samples/sec   accuracy=0.87
...
INFO:root:Epoch[9] Batch [100]     Speed: 1300 samples/sec   accuracy=0.98
Training completed.
```

这表示 MXNet 分布式任务成功运行，模型训练完成。

## 小结

通过本教程，您学习了如何在 AI Lab 平台上创建和运行 MXNet 的单机和分布式任务。我们详细介绍了 MXJob 的配置方式，以及如何在任务中指定运行的命令和资源需求。希望本教程对您有所帮助，如有任何问题，请参考平台提供的其他文档或联系技术支持。

---

## 附录

- **注意事项:**
  - 确保您使用的镜像包含所需的 MXNet 版本和依赖。
  - 根据实际需求调整资源配置，避免资源不足或浪费。
  - 如需使用自定义的训练脚本，请修改启动命令和参数。
- **参考文档:**
  - [MXNet 官方文档](#)
  - [Kubeflow MXJob 指南](#)

## PaddlePaddle 任务

PaddlePaddle（飞桨）是百度开源的深度学习平台，支持丰富的神经网络模型和分布式训练方式。PaddlePaddle 任务可以通过单机或分布式模式进行训练。在 AI Lab 平台中，我们提供了对 PaddlePaddle 任务的支持，您可以通  
过界面化操作，快速创建 PaddlePaddle 任务，进行模型训练。

本教程将指导您如何在 AI Lab 平台上创建和运行 PaddlePaddle 的单机和分布式任务。

## 任务配置介绍

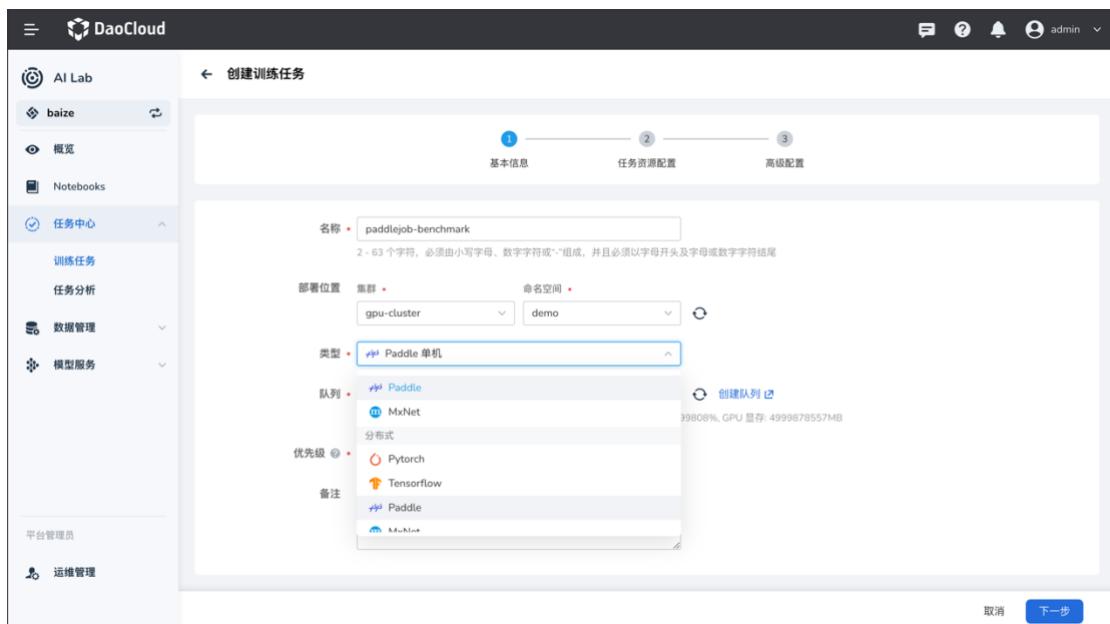
- **任务类型:** PaddlePaddle，支持单机和分布式两种模式。
- **运行环境:** 选择包含 PaddlePaddle 框架的镜像，或在任务中安装必要的依赖。

# 任务运行环境

我们使用 `registry.baidubce.com/paddlepaddle/paddle:2.4.0rc0-cpu` 镜像作为任务的基础运行环境。该镜像预装了 PaddlePaddle 框架，适用于 CPU 计算。如果需要使用 GPU，请选择对应的 GPU 版本镜像。

**注意：**了解如何创建和管理环境，请参考 [环境列表](#)。

## 创建 PaddlePaddle 任务



## PaddlePaddle 单机训练任务

### 创建步骤

- 登录平台：**登录 AI Lab 平台，点击左侧导航栏中的 **任务中心**，进入 **训练任务** 页面。
- 创建任务：**点击右上角的 **创建** 按钮，进入任务创建页面。
- 选择任务类型：**在弹出的窗口中，选择任务类型为 **PaddlePaddle**，然后点击 **下一步**。
- 填写任务信息：**填写任务名称和描述，例如“**PaddlePaddle 单机训练任务**”，然后点击 **确定**。
- 配置任务参数：**根据您的需求，配置任务的运行参数、镜像、资源等信息。

## 运行参数

- 启动命令: python
- 命令参数:  
• -m paddle.distributed.launch run\_check

说明:

- m paddle.distributed.launch: 使用 PaddlePaddle 提供的分布式启动模块，即使在单机模式下也可以使用，方便将来迁移到分布式。
- run\_check: PaddlePaddle 提供的测试脚本，用于检查分布式环境是否正常。

## 资源配置

- 副本数: 1 (单机任务)
- 资源请求:
  - CPU**: 根据需求设置，建议至少 1 核
  - 内存**: 根据需求设置，建议至少 2 GiB
  - GPU**: 如果需要使用 GPU，选择 GPU 版本的镜像，并分配相应的 GPU 资源

## 完整的 PaddleJob 配置示例

以下是单机 PaddleJob 的 YAML 配置:

```
apiVersion: kubeflow.org/v1
kind: PaddleJob
metadata:
  name: paddle-simple-cpu
  namespace: kubeflow
spec:
  paddleReplicaSpecs:
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
            - name: paddle
```

```
image: registry.baidubce.com/paddlepaddle/paddle:2.4.0rc0-cpu
command:
  [
    'python',
    '-m',
    'paddle.distributed.launch',
    'run_check',
  ]
```

## 配置解析:

- **apiVersion** 和 **kind**: 指定资源的 API 版本和类型，这里是 **PaddleJob**。
- **metadata**: 元数据，包括任务名称和命名空间。
- **spec**: 任务的详细配置。
  - **paddleReplicaSpecs**: **PaddlePaddle** 任务的副本配置。
    - **Worker**: 指定工作节点的配置。
      - **replicas**: 副本数，这里为 1，表示单机训练。
      - **restartPolicy**: 重启策略，设为 **OnFailure**，表示任务失败时自动重启。
      - **template**: Pod 模板，定义容器的运行环境和资源。
        - **containers**: 容器列表。
          - **name**: 容器名称。
          - **image**: 使用的镜像。
          - **command**: 启动命令和参数。

## 提交任务

配置完成后，点击 **提交** 按钮，开始运行 **PaddlePaddle** 单机任务。

## 查看运行结果

任务提交成功后，您可以进入 **任务详情** 页面，查看资源的使用情况和任务的运行状态。从右上角进入 **工作负载详情**，可以查看运行过程中的日志输出。

示例输出：

```
run check success, PaddlePaddle is installed correctly on this node :)
```

这表示 **PaddlePaddle** 单机任务成功运行，环境配置正常。

---

## PaddlePaddle 分布式训练任务

在分布式模式下，**PaddlePaddle** 任务可以使用多台计算节点共同完成训练，提高训练效率。

### 创建步骤

1. 登录平台：同上。
2. 创建任务：点击右上角的 **创建** 按钮，进入任务创建页面。
3. 选择任务类型：选择任务类型为 **PaddlePaddle**，然后点击 **下一步**。
4. 填写任务信息：填写任务名称和描述，例如“**PaddlePaddle** 分布式训练任务”，然后点击 **确定**。
5. 配置任务参数：根据需求，配置运行参数、镜像、资源等。

### 运行参数

- 启动命令：python
- 命令参数：  
-m paddle.distributed.launch train.py --epochs=10

说明：

- -m paddle.distributed.launch：使用 **PaddlePaddle** 提供的分布式启动模块。
- train.py：您的训练脚本，需要放在镜像中或挂载到容器内。
- --epochs=10：训练的轮数，这里设置为 10。

### 资源配置

- 任务副本数：根据 Worker 副本数设置，这里为 2。

- 资源请求:
  - **CPU**: 根据需求设置, 建议至少 1 核
  - **内存**: 根据需求设置, 建议至少 2 GiB
  - **GPU**: 如果需要使用 GPU, 选择 GPU 版本的镜像, 并分配相应的 GPU 资源

## 完整的 PaddleJob 配置示例

以下是分布式 PaddleJob 的 YAML 配置:

```
apiVersion: kubeflow.org/v1
kind: PaddleJob
metadata:
  name: paddle-distributed-job
  namespace: kubeflow
spec:
  paddleReplicaSpecs:
    Worker:
      replicas: 2
      restartPolicy: OnFailure
      template:
        spec:
          containers:
            - name: paddle
              image: registry.baidubce.com/paddlepaddle/paddle:2.4.0rc0-cpu
              command:
                [
                  'python',
                  '-m',
                  'paddle.distributed.launch',
                  'train.py',
                ]
            args:
              - '--epochs=10'
```

### 配置解析:

- **Worker**:
  - **replicas**: 副本数, 设置为 2, 表示使用 2 个工作节点进行分布式训练。
  - 其他配置与单机模式类似。

## 设置任务副本数

在创建 PaddlePaddle 分布式任务时，需要根据 paddleReplicaSpecs 中配置的副本数，正确设置 任务副本数。

- 总副本数 = Worker 副本数
- 本示例中：
  - Worker 副本数：2
  - 总副本数：2

因此，在任务配置中，需要将 任务副本数 设置为 **2**。

## 提交任务

配置完成后，点击 提交 按钮，开始运行 PaddlePaddle 分布式任务。

## 查看运行结果

进入 任务详情 页面，查看任务的运行状态和资源使用情况。您可以查看每个工作节点的日志输出，确认分布式训练是否正常运行。

示例输出：

```
Worker 0: Epoch 1, Batch 100, Loss 0.5
Worker 1: Epoch 1, Batch 100, Loss 0.6
...
Training completed.
```

这表示 PaddlePaddle 分布式任务成功运行，模型训练完成。

---

## 小结

通过本教程，您学习了如何在 AI Lab 平台上创建和运行 PaddlePaddle 的单机和分布式任务。我们详细介绍了 PaddleJob 的配置方式，以及如何在任务中指定运行的命令和资源需求。希望本教程对您有所帮助，如有任何问题，请参考平台提供的其他文档或联系技术支持。

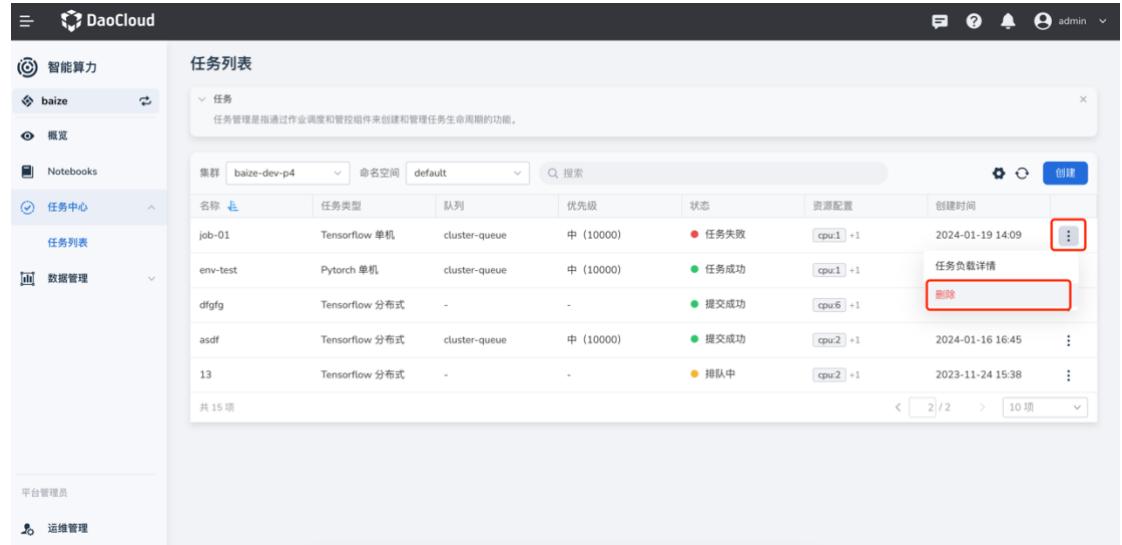
# 附录

- **注意事项:**
  - **训练脚本:** 确保 `train.py` (或其他训练脚本) 在容器内存在。您可以通过自定义镜像、挂载持久化存储等方式将脚本放入容器。
  - **镜像选择:** 根据您的需求选择合适的镜像，例如使用 GPU 时选择 `paddle:2.4.0rc0-gpu` 等。
  - **参数调整:** 可以通过修改 `command` 和 `args` 来传递不同的训练参数。
- **参考文档:**
  - [PaddlePaddle 官方文档](#)
  - [Kubeflow PaddleJob 指南](#)

## 删除任务 (Job)

如果发现任务冗余、过期或因其他缘故不再需要，可以从训练任务列表中删除。

1. 在训练任务列表右侧点击 ，在弹出菜单中选择 **删除**。



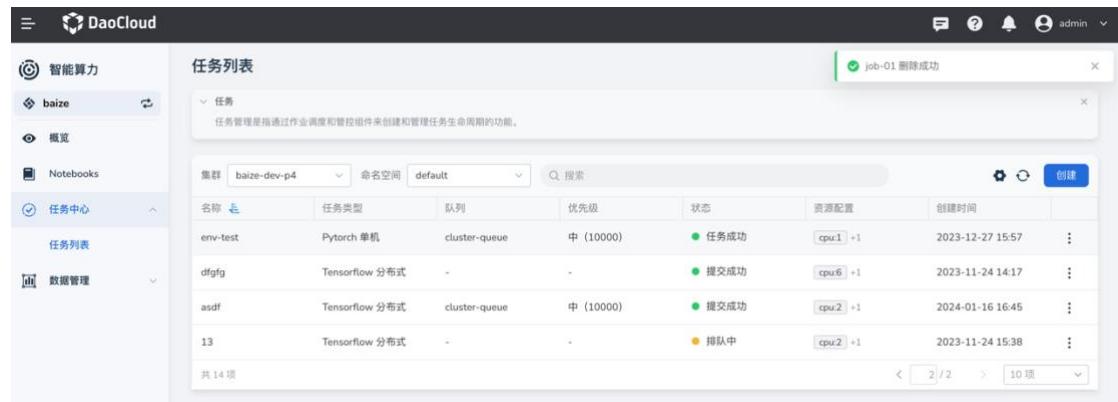
The screenshot shows the DaoCloud interface with the 'Task List' page open. The left sidebar has 'baize' selected under '智能算力'. The main area shows a table of tasks with columns: Name, Task Type, Queue, Priority, Status, Resource Configuration, and Creation Time. One task, 'dfgfg', is highlighted with a red box around its delete button in the 'Actions' column.

| 名称       | 任务类型           | 队列            | 优先级       | 状态     | 资源配置     | 创建时间             |
|----------|----------------|---------------|-----------|--------|----------|------------------|
| job-01   | Tensorflow 单机  | cluster-queue | 中 (10000) | ● 任务失败 | cpu.1 +1 | 2024-01-19 14:09 |
| env-test | Pytorch 单机     | cluster-queue | 中 (10000) | ● 任务成功 | cpu.1 +1 | 任务负载详情           |
| dfgfg    | Tensorflow 分布式 | -             | -         | ● 提交成功 | cpu.6 +1 | <b>删除</b>        |
| asdf     | Tensorflow 分布式 | cluster-queue | 中 (10000) | ● 提交成功 | cpu.2 +1 | 2024-01-16 16:45 |
| 13       | Tensorflow 分布式 | -             | -         | ● 排队中  | cpu.2 +1 | 2023-11-24 15:38 |

2. 在弹窗中确认要删除的任务，输入任务名称后点击 **删除**。



3. 屏幕提示删除成功，该任务从列表中消失。



## Caution

任务一旦删除将不可恢复，请谨慎操作。

## 查看任务（Job）工作负载

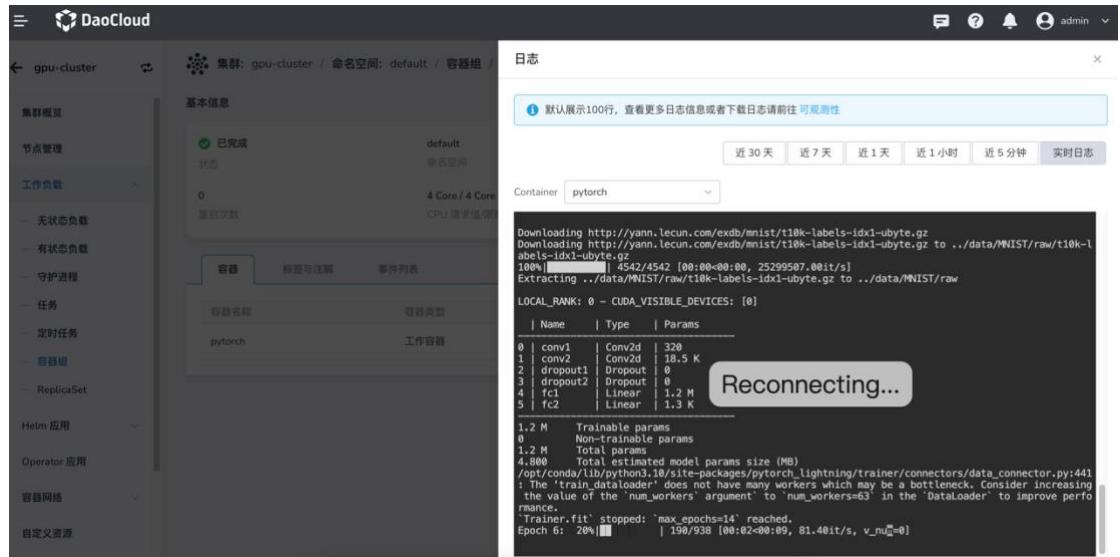
任务创建好后，都会显示在训练任务列表中。

1. 在训练训练任务列表中，点击某个任务右侧的 -> 任务负载详情。

2. 出现一个弹窗选择要查看哪个 Pod 后，点击 进入。

3. 跳转到容器管理界面，可以查看容器的工作状态、标签与注解以及发生的事件。

4. 你还可以查看当前 Pod 最近一段时间的详细日志。此处默认展示 100 行日志，如果要查看更详细的日志或下载日志，请点击顶部的蓝色 可观测性 文字。



5. 当然你还可以通过右上角的 ...，查看当前 Pod 的 YAML、上传和下载文件。以下是一个 Pod 的 YAML 示例。

```

kind: Pod
apiVersion: v1
metadata:
  name: neko-tensorboard-job-test-202404181843-skxivllb-worker-0
  namespace: default
  uid: ddedb6ff-c278-47eb-ae1e-0de9b7c62f8c
  resourceVersion: '41092552'
  creationTimestamp: '2024-04-18T10:43:36Z'
  labels:
    training.kubeflow.org/job-name: neko-tensorboard-job-test-202404181843-skxivllb
    training.kubeflow.org/operator-name: pytorchjob-controller
    training.kubeflow.org/replica-index: '0'
    training.kubeflow.org/replica-type: worker
  annotations:
    cni.projectcalico.org/containerID: 0cfbb9af257d5e69027c603c6cb2d3890a17c4ae1a145748d5aef73a10d7fbe1
    cni.projectcalico.org/podIP: ""
    cni.projectcalico.org/podIPs: ""
    hami.io/bind-phase: success
    hami.io/bind-time: '1713437016'
    hami.io/vgpu-devices-allocated: GPU-29d5fa0d-935b-2966-aff8-483a174d61d1,NVIDIA,1024,20;;
    hami.io/vgpu-devices-to-allocate: ;
    hami.io/vgpu-node: worker-a800-1
    hami.io/vgpu-time: '1713437016'
    k8s.v1.cni.cncf.io/network-status: |-[{"name": "kube-system/calico", "status": "OK"}]

```

```

    "ips": [
        "10.233.97.184"
    ],
    "default": true,
    "dns": {}
}
]

k8s.v1.cni.cncf.io/networks-status: |-  

[{
    "name": "kube-system/calico",
    "ips": [
        "10.233.97.184"
    ],
    "default": true,
    "dns": {}
}]
ownerReferences:  

- apiVersion: kubeflow.org/v1  

  kind: PyTorchJob  

  name: neko-tensorboard-job-test-202404181843-skxivllb  

  uid: e5a8b05d-1f03-4717-8e1c-4ec928014b7b  

  controller: true  

  blockOwnerDeletion: true  

spec:  

  volumes:  

- name: 0-dataset-pytorch-examples  

  persistentVolumeClaim:  

    claimName: pytorch-examples  

- name: kube-api-access-wh9rh  

  projected:  

    sources:  

      - serviceAccountToken:  

        expirationSeconds: 3607  

        path: token  

      - configMap:  

        name: kube-root-ca.crt  

        items:  

          - key: ca.crt  

          path: ca.crt  

      - downwardAPI:  

        items:  

          - path: namespace  

        fieldRef:  

          apiVersion: v1  

          fieldPath: metadata.namespace

```

```

    defaultMode: 420
  containers:
    - name: pytorch
      image: m.daocloud.io/docker.io/pytorch/pytorch
      command:
        - bash
      args:
        - '-c'
        - '>-
          ls -la /root && which pip && pip install pytorch_lightning tensorboard
          && python /root/Git/pytorch/examples/mnist/main.py'
    ports:
      - name: pytorchjob-port
        containerPort: 23456
        protocol: TCP
    env:
      - name: PYTHONUNBUFFERED
        value: '1'
      - name: PET_NNODES
        value: '1'
    resources:
      limits:
        cpu: '4'
        memory: 8Gi
        nvidia.com/gpucores: '20'
        nvidia.com/gpumem: '1024'
        nvidia.com/vgpu: '1'
      requests:
        cpu: '4'
        memory: 8Gi
        nvidia.com/gpucores: '20'
        nvidia.com/gpumem: '1024'
        nvidia.com/vgpu: '1'
    volumeMounts:
      - name: 0-dataset-pytorch-examples
        mountPath: /root/Git/pytorch/examples
      - name: kube-api-access-wh9rh
        readOnly: true
        mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    imagePullPolicy: Always
  restartPolicy: Never
  terminationGracePeriodSeconds: 30

```

```
dnsPolicy: ClusterFirst
serviceAccountName: default
serviceAccount: default
nodeName: worker-a800-1
securityContext: {}
affinity: {}
schedulerName: hami-scheduler
tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
priorityClassName: baize-high-priority
priority: 100000
enableServiceLinks: true
preemptionPolicy: PreemptLowerPriority
status:
  phase: Succeeded
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2024-04-18T10:43:36Z'
      reason: PodCompleted
    - type: Ready
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2024-04-18T10:46:34Z'
      reason: PodCompleted
    - type: ContainersReady
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2024-04-18T10:46:34Z'
      reason: PodCompleted
    - type: PodScheduled
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2024-04-18T10:43:36Z'
hostIP: 10.20.100.211
podIP: 10.233.97.184
```

```
podIPs:
  - ip: 10.233.97.184
startTime: '2024-04-18T10:43:36Z'
containerStatuses:
  - name: pytorch
    state:
      terminated:
        exitCode: 0
        reason: Completed
        startedAt: '2024-04-18T10:43:39Z'
        finishedAt: '2024-04-18T10:46:34Z'
        containerID: >-

containerd://09010214bcf3315e81d38fba50de3943c9d2b48f50a6cc2e83f8ef0e5c6eeec1
  lastState: {}
  ready: false
  restartCount: 0
  image: m.daocloud.io/docker.io/pytorch/pytorch:latest
  imageID: >-
m.daocloud.io/docker.io/pytorch/pytorch@sha256:11691e035a3651d25a87116b4f6adc113a2
7a29d8f5a6a583f8569e0ee5ff897
  containerID: >-

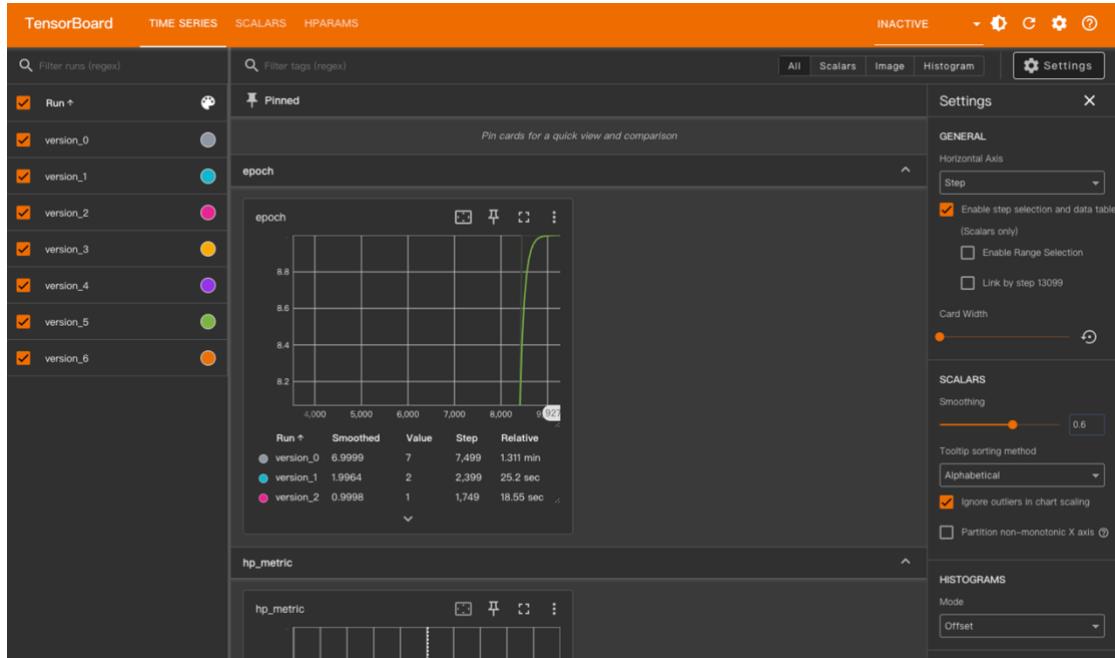
containerd://09010214bcf3315e81d38fba50de3943c9d2b48f50a6cc2e83f8ef0e5c6eeec1
  started: false
  qosClass: Guaranteed
```

## 任务分析介绍

在 DCE 5.0 AI Lab 模块中，提供了模型开发过程重要的可视化分析工具，用于展示机器学习模型的训练过程和结果。本文将介绍 任务分析（Tensorboard）的基本概念、在 AI Lab 系统中的使用方法，以及如何配置数据集的日志内容。

### Note

Tensorboard 是 TensorFlow 提供的一个可视化工具，用于展示机器学习模型的训练过程和结果。它可以帮助开发者更直观地理解模型的训练动态，分析模型性能，调试模型问题等。



Tensorboard 在模型开发过程中的作用及优势：

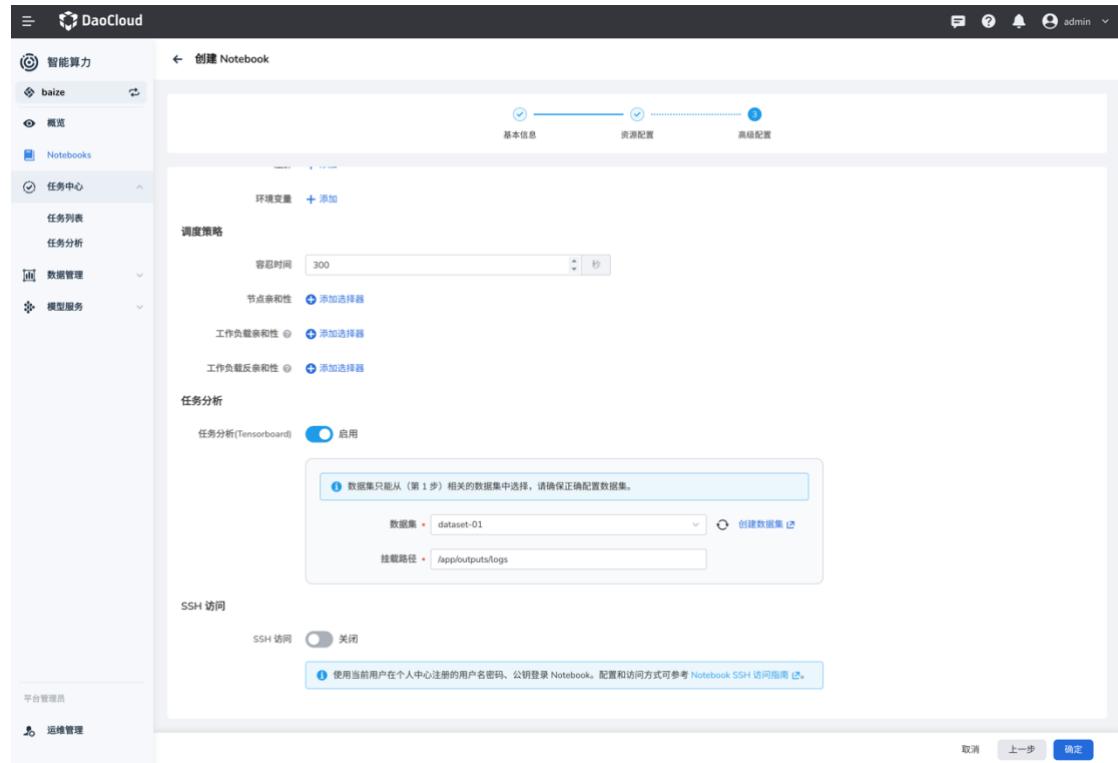
- 可视化训练过程:** 通过图表展示训练和验证的损失、精度等指标，帮助开发者直观地观察模型的训练效果。
- 调试和优化模型:** 通过查看不同层的权重、梯度分布等，帮助开发者发现和修正模型中的问题。
- 对比不同实验:** 可以同时展示多个实验的结果，方便开发者对比不同模型和超参数配置的效果。
- 追踪训练数据:** 记录训练过程中使用的数据集和参数，确保实验的可复现性。

## 如何创建 Tensorboard

在 AI Lab 系统中，我们提供了便捷的方式来创建和管理 Tensorboard。以下是具体步骤：

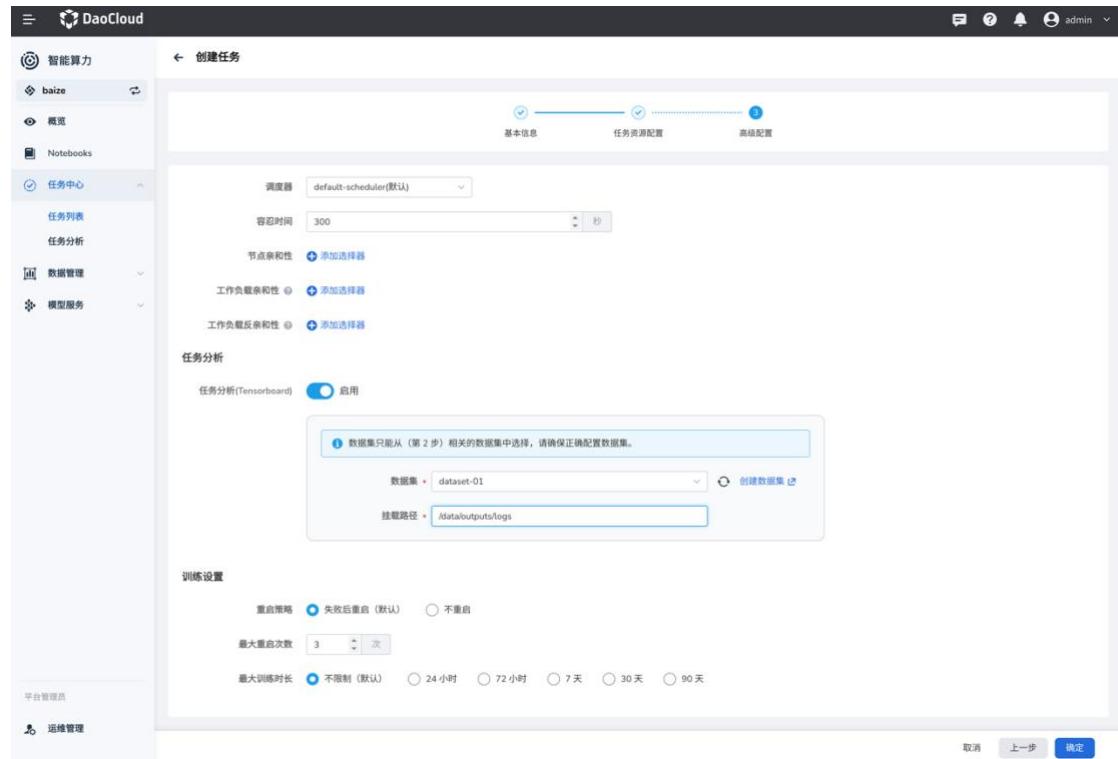
### 在创建时 Notebook 启用 Tensorboard

- 1. 创建 Notebook:** 在 AI Lab 平台上创建一个新的 Notebook。
- 2. 启用 Tensorboard:** 在创建 Notebook 的页面中，启用 **Tensorboard** 选项，并指定数据集和日志路径。



## 在分布式任务创建及完成后启用 Tensorboard

1. **创建分布式任务:** 在 AI Lab 平台上创建一个新的分布式训练任务。
2. **配置 Tensorboard:** 在任务配置页面中，启用 **Tensorboard** 选项，并指定数据集和日志路径。
3. **任务完成后查看 Tensorboard:** 任务完成后，可以在任务详情页面中查看 **Tensorboard** 的链接，点击链接即可查看训练过程的可视化结果。



## 在 Notebook 中直接引用 Tensorboard

在 Notebook 中，可以通过代码直接启动 Tensorboard。以下是一个示例代码：

```
# 导入必要的库
import tensorflow as tf
import datetime

# 定义日志目录
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

# 创建 Tensorboard 回调
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

# 构建并编译模型
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 训练模型并启用 Tensorboard 回调
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test),
           callbacks=[tensorboard_callback])
```

## 如何配置数据集的日志内容

在使用 **Tensorboard** 时，可以记录和配置不同的数据集和日志内容。以下是一些常见的配置方式：

### 配置训练和验证数据集的日志

在训练模型时，可以通过 **TensorFlow** 的 `tf.summary API` 来记录训练和验证数据集的日志。以下是一个示例代码：

```
# 导入必要的库
import tensorflow as tf

# 创建日志目录
train_log_dir = 'logs/gradient_tape/train'
val_log_dir = 'logs/gradient_tape/val'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
val_summary_writer = tf.summary.create_file_writer(val_log_dir)

# 训练模型并记录日志
for epoch in range(EPOCHS):
    for (x_train, y_train) in train_dataset:
        # 训练步骤
        train_step(x_train, y_train)
        with train_summary_writer.as_default():
            tf.summary.scalar('loss', train_loss.result(), step=epoch)
            tf.summary.scalar('accuracy', train_accuracy.result(), step=epoch)

    for (x_val, y_val) in val_dataset:
        # 验证步骤
        val_step(x_val, y_val)
        with val_summary_writer.as_default():
            tf.summary.scalar('loss', val_loss.result(), step=epoch)
            tf.summary.scalar('accuracy', val_accuracy.result(), step=epoch)
```

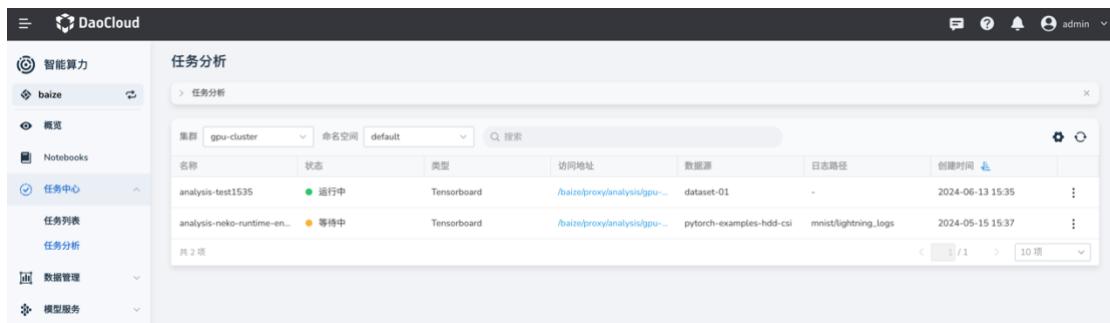
## 配置自定义日志

除了训练和验证数据集的日志外，还可以记录其他自定义的日志内容，例如学习率、梯度分布等。以下是一个示例代码：

```
# 记录自定义日志
with train_summary_writer.as_default():
    tf.summary.scalar('learning_rate', learning_rate, step=epoch)
    tf.summary.histogram('gradients', gradients, step=epoch)
```

## Tensorboard 管理

在 AI Lab 中，通过各种方式创建出来的 Tensorboard 会统一展示在任务分析的页面中，方便用户查看和管理。



| 名称                          | 状态  | 类型          | 访问地址                          | 数据源                      | 日志路径                 | 创建时间             |
|-----------------------------|-----|-------------|-------------------------------|--------------------------|----------------------|------------------|
| analysis-test1535           | 运行中 | Tensorboard | /baize/proxy/analysis/gpu-... | dataset-01               | -                    | 2024-06-13 19:35 |
| analysis-neko-runtime-en... | 等待中 | Tensorboard | /baize/proxy/analysis/gpu-... | pytorch-examples-hdd-csi | mnist/lightning_logs | 2024-05-15 15:37 |

用户可以在任务分析页面中查看 Tensorboard 的链接、状态、创建时间等信息，并通过链接直接访问 Tensorboard 的可视化结果。

## 数据集列表

AI Lab 提供模型开发、训练以及推理过程所有需要的数据集管理功能。目前支持将多种数据源统一接入能力。

通过简单配置即可将数据源接入到 AI Lab 中，实现数据的统一纳管、预热、数据集管理等功能。

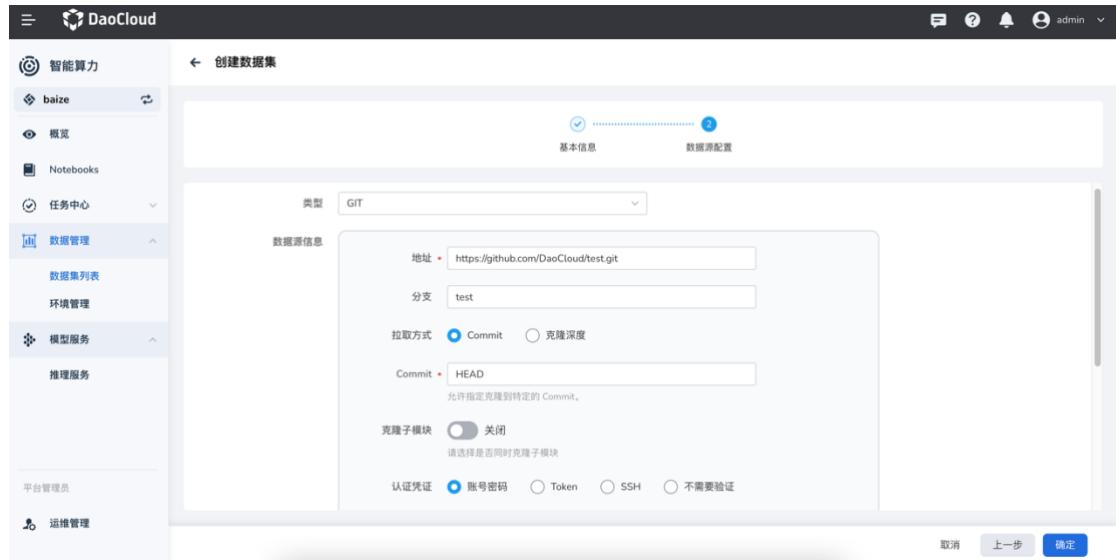
## 创建数据集

1. 在左侧导航栏中点击 **数据管理 -> 数据集列表**，点击右侧的 **创建** 按钮。

| 名称                         | 类型   | 数据源                            | 状态    | 预加载 | 创建时间             |
|----------------------------|------|--------------------------------|-------|-----|------------------|
| test-s3                    | S3   | s3://ba                        | ● 预热中 | 启用  | 2024-01-08 11:43 |
| test-138                   | PVC  | pvc://demo                     | ● 运行中 | 不启用 | 2023-12-20 14:28 |
| test-117                   | S3   | s3://d                         | ● 预热中 | 启用  | 2024-01-08 11:18 |
| test-116                   | GIT  | git://asd.c                    | ● 预热中 | 启用  | 2024-01-08 11:17 |
| test-114                   | HTTP | http://d.c                     | ● 预热中 | 启用  | 2024-01-08 11:15 |
| test-105                   | HTTP | https://wwd.c                  | ● 预热中 | 启用  | 2024-01-08 10:41 |
| test-103                   | S3   | s3://d                         | ● 预热中 | 启用  | 2024-01-08 10:38 |
| test-101                   | S3   | s3://234                       | ● 预热中 | 启用  | 2024-01-08 10:36 |
| ssss                       | HTTP | https://cs.biomerieuxcs.cn/... | ● 失败  | 启用  | 2023-12-28 00:23 |
| python3.12-pytorch-cuda... | NFS  | nfs://10.20.2.21/data/pytho... | ● 运行中 | 不启用 | 2023-12-27 15:32 |

## 2. 选择数据集归属的工作集群、命名空间 下一步。

## 3. 配置目标数据的数据源类型，然后点击 确定。



目前支持这几种数据源：

- **GIT:** 支持 GitHub、GitLab、Gitee 等仓库
- **S3:** 支持 Amazon 云等对象存储
- **HTTP:** 直接输入一个有效的 HTTP 网址
- **PVC:** 支持预先创建的 Kubernetes PersistentVolumeClaim
- **NFS:** 支持 NFS 共享存储

4. 数据集创建成功将返回数据集列表。你可以通过右侧的 ⋮ 执行更多操作。

| 名称                         | 类型   | 数据源                           | 状态    | 预加载 | 创建时间             |
|----------------------------|------|-------------------------------|-------|-----|------------------|
| test-s3                    | S3   | s3://aa                       | ● 预热中 | 启用  | 2024-01-08 11:43 |
| test-138                   | PVC  | pvc://demo                    | ● 运行中 | 不启用 | 2023-12-20 14:28 |
| test-117                   | S3   | s3://d                        | ● 预热中 | 启用  | 2024-01-08 11:18 |
| test-116                   | GIT  | git://sd.c                    | ● 预热中 | 启用  | 2024-01-08 11:17 |
| test-114                   | HTTP | http://d.c                    | ● 预热中 | 启用  | 2024-01-08 11:15 |
| test-105                   | HTTP | https://wwd.c                 | ● 预热中 | 启用  | 2024-01-08 10:41 |
| test-103                   | S3   | s3://d                        | ● 预热中 | 启用  | 2024-01-08 10:38 |
| test-101                   | S3   | s3://234                      | ● 预热中 | 启用  | 2024-01-08 10:36 |
| ssss                       | HTTP | https://cs.biomerieuxcs.cn/_  | ● 失败  | 启用  | 2023-12-28 00:23 |
| python3.12-pytorch-cuda... | NFS  | nfs://10.20.2.21/data/pyth... | ● 运行中 | 不启用 | 2023-12-27 15:32 |

## Info

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。

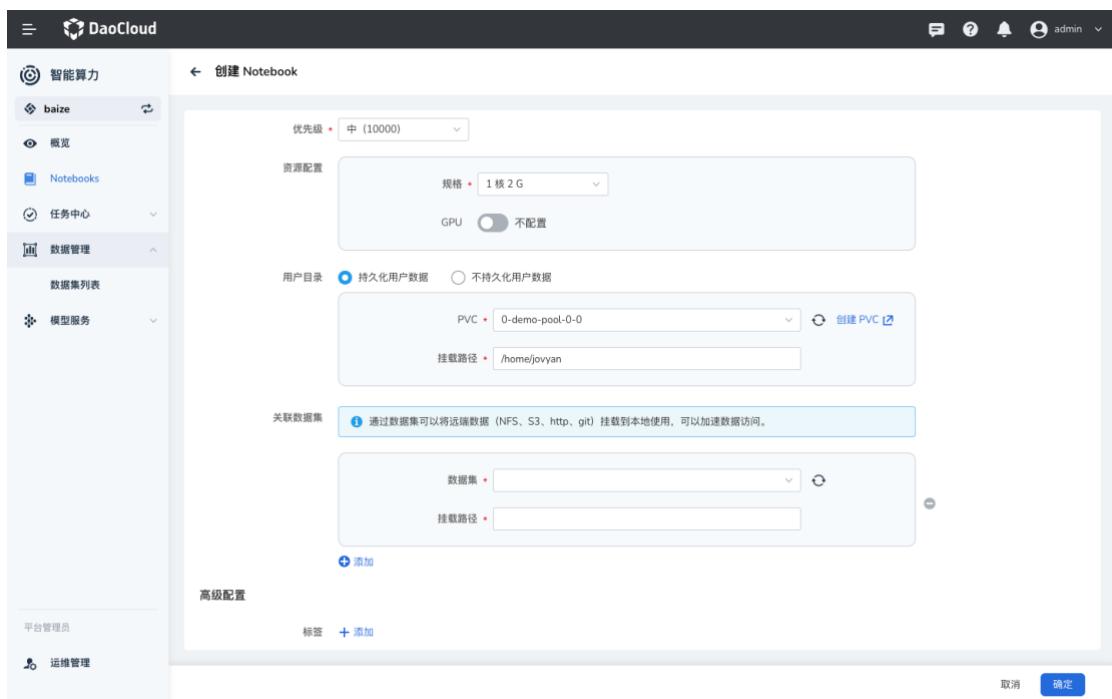
# 数据集使用

数据集创建成功后，可以在模型训练、推理等任务中使用。

## 在 Notebook 中使用

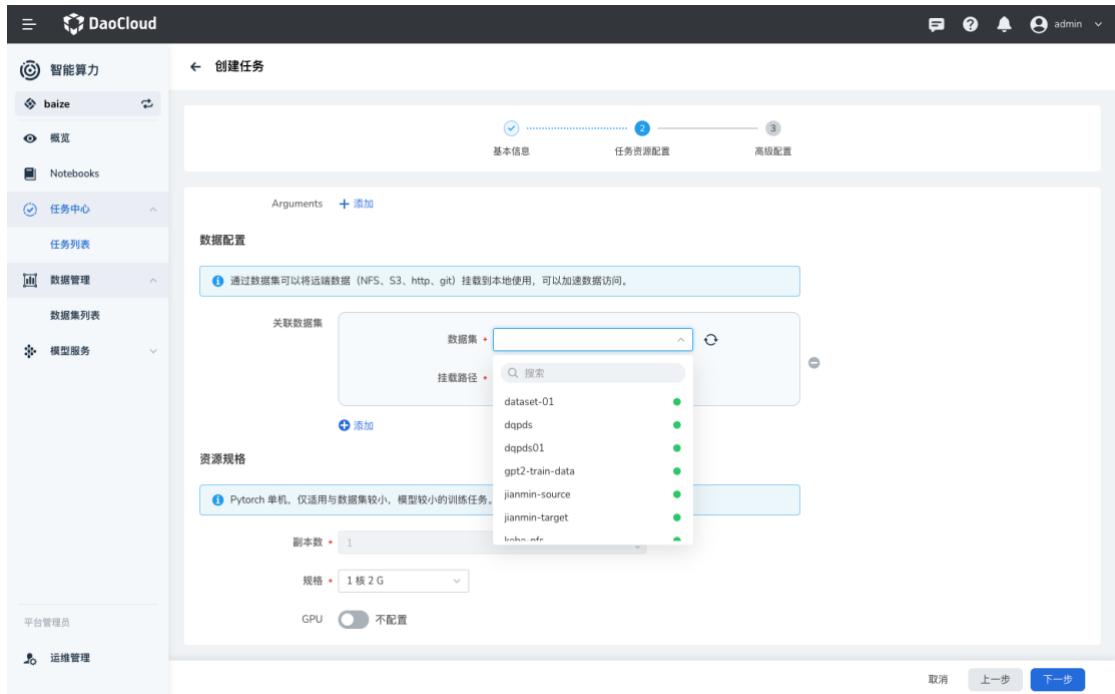
在创建 Notebook 中，可以直接使用数据集；使用方式如下：

- 使用数据集做训练数据挂载
- 使用数据集做代码挂载



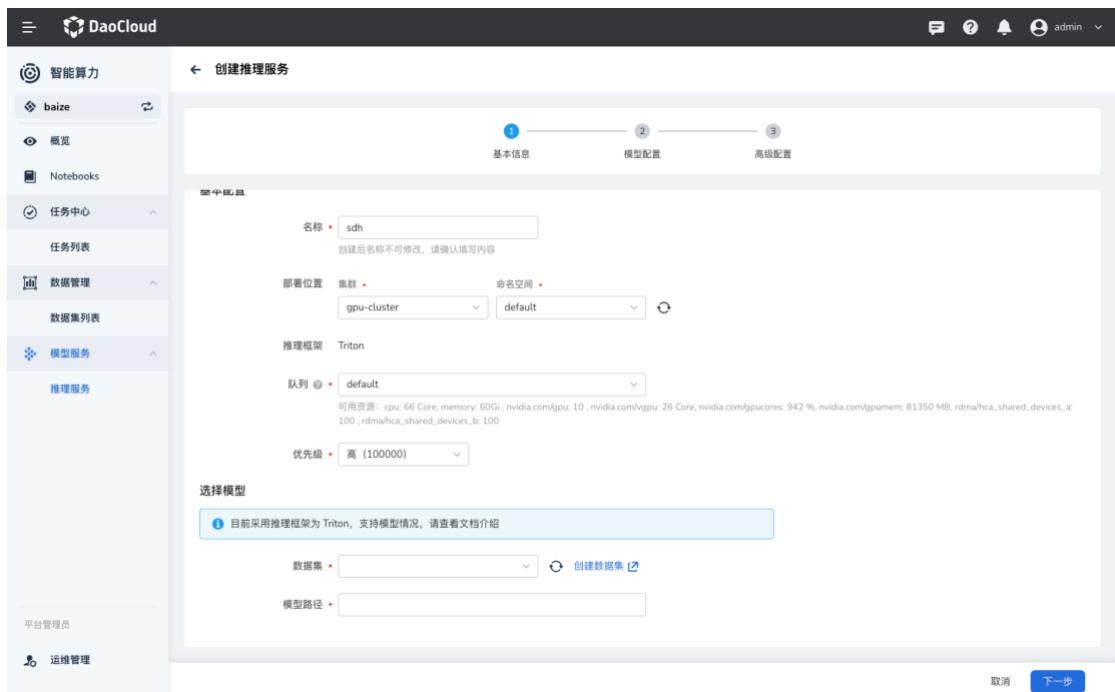
## 在 训练任务 中使用

- 使用数据集指定任务输出
- 使用数据集指定任务输入
- 使用数据集指定 TensorBoard 输出



## 在推理服务 中使用

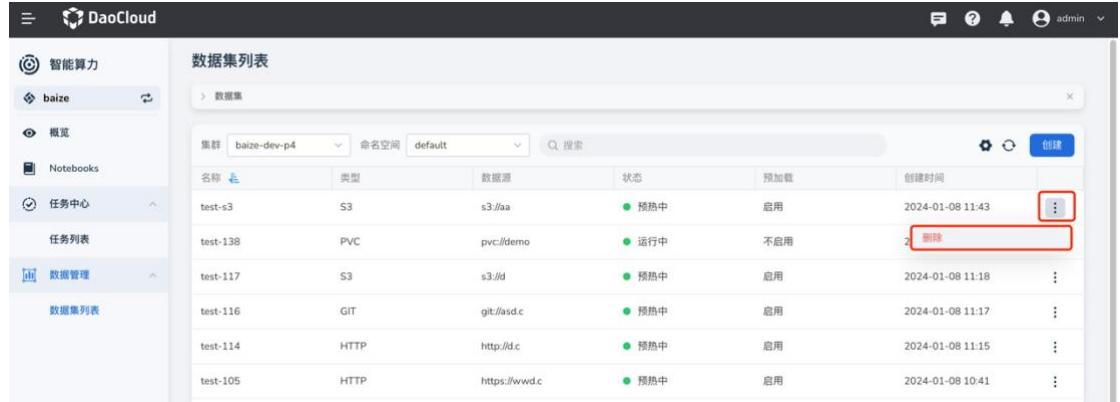
- 使用数据集挂载模型



# 删除数据集

如果发现数据集冗余、过期或因其他缘故不再需要，可以从数据集列表中删除。

1. 在数据集列表右侧点击 ，在弹出菜单中选择 **删除**。



The screenshot shows the DaoCloud interface with the 'Data Set' list. A red box highlights the three-dot menu icon next to the first data set in the list, and another red box highlights the 'Delete' button in the confirmation dialog.

| 名称       | 类型   | 数据源           | 状态    | 预加载 | 创建时间             |
|----------|------|---------------|-------|-----|------------------|
| test-s3  | S3   | s3://aa       | ● 预热中 | 启用  | 2024-01-08 11:43 |
| test-138 | PVC  | pvc://demo    | ● 运行中 | 不启用 | 2024-01-08 11:43 |
| test-117 | S3   | s3://d        | ● 预热中 | 启用  | 2024-01-08 11:18 |
| test-116 | GIT  | git://asd.c   | ● 预热中 | 启用  | 2024-01-08 11:17 |
| test-114 | HTTP | http://d.c    | ● 预热中 | 启用  | 2024-01-08 11:15 |
| test-105 | HTTP | https://wwd.c | ● 预热中 | 启用  | 2024-01-08 10:41 |

2. 在弹窗中确认要删除的数据集，输入数据集名称后点击 **删除**。



The screenshot shows a confirmation dialog box asking if you want to delete the data set 'test-s3'. It contains a message about losing data, an input field with 'test-s3', and a red 'Delete' button.

3. 屏幕提示删除成功，该数据集从列表中消失。

## Caution

数据集一旦删除将不可恢复，请谨慎操作。

# 管理环境

本文说明如何在 **DCE AI Lab** 中管理你的环境依赖库，以下是具体操作步骤和注意事项。

1. [环境管理概述](#)
2. [创建新环境](#)
3. [配置环境](#)

#### 4. 故障排除

## 环境管理概述

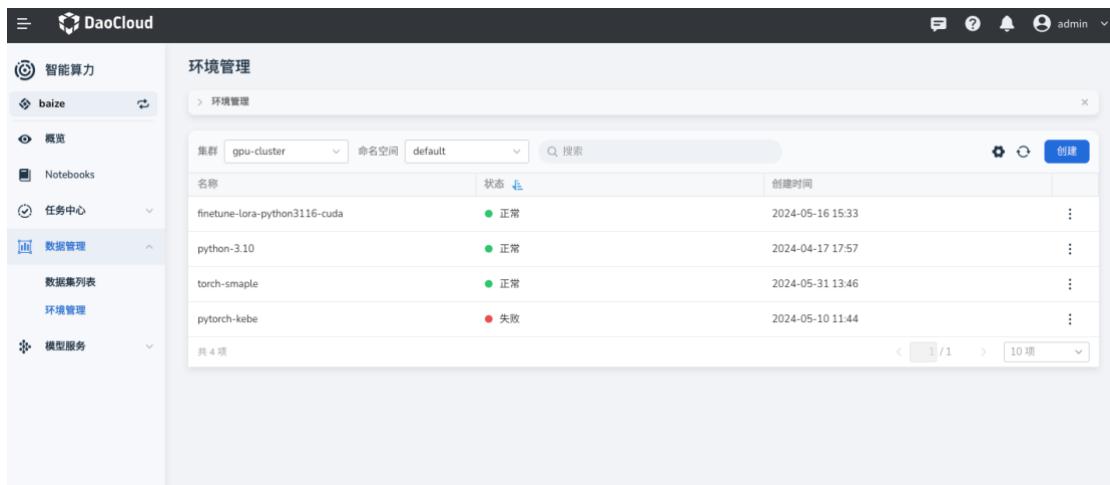
传统方式，一般会将 Python 环境依赖在镜像中构建，镜像带有 Python 版本和依赖包的镜像，维护成本较高且更新不方便，往往需要重新构建镜像。

而在 DCE 5.0 AI Lab 中，用户可以通过 **环境管理** 模块来管理纯粹的环境依赖，将这部分从镜像中解耦，带来的优势有：

- 一份环境多处使用，同时可以在 Notebook、分布式训练任务、乃至推理服务中使用。
- 更新依赖包更加方便，只需要更新环境依赖即可，无需重新构建镜像。

以下为环境管理的主要组成部分：

- **集群**：选择需要操作的集群。
- **命名空间**：选择命名空间以限定操作范围。
- **环境列表**：展示当前集群和命名空间下的所有环境及其状态。



| 名称                            | 状态 | 创建时间             |
|-------------------------------|----|------------------|
| finetune-lora-python3116-cuda | 正常 | 2024-05-16 15:33 |
| python-3.10                   | 正常 | 2024-04-17 17:57 |
| torch-smaple                  | 正常 | 2024-05-31 13:46 |
| pytorch-kebe                  | 失败 | 2024-05-10 11:44 |

### 字段 描述

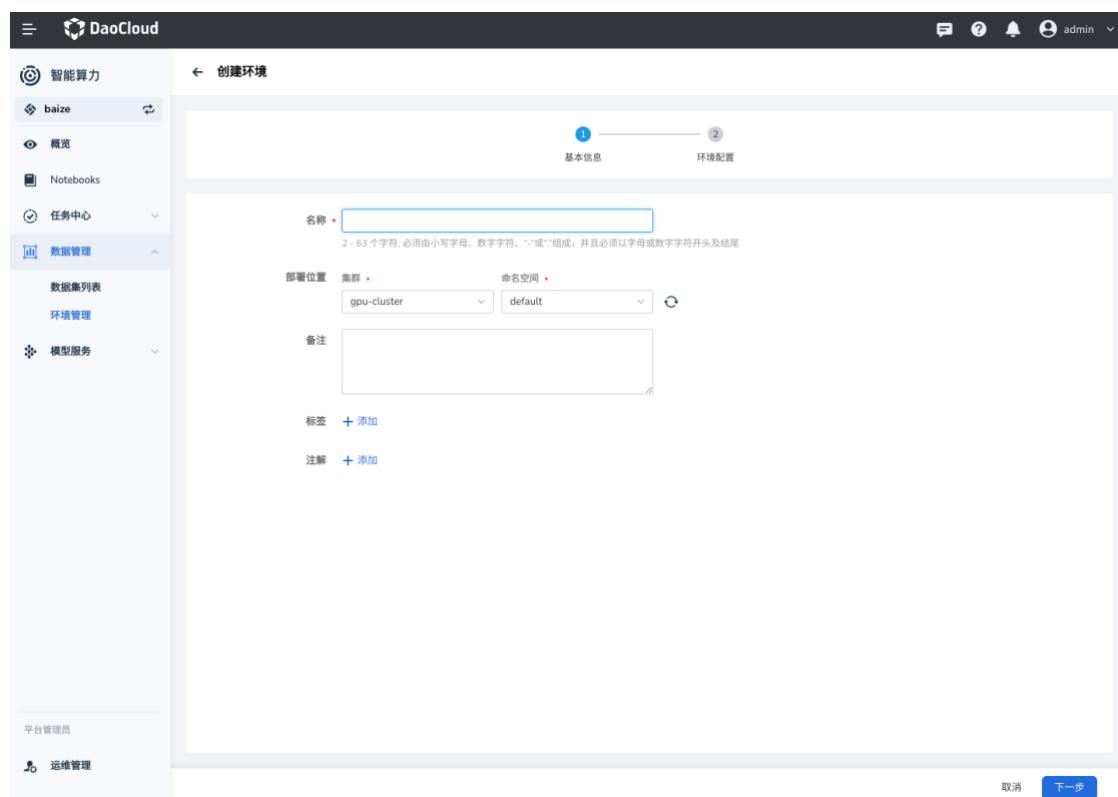
名称 环境的名称

状态 环境当前的状态（正常或失败），新创建环境有一个预热过程，预热成功后即可在其他任务中使用

| 字段   | 描述      | 举例值                 |
|------|---------|---------------------|
| 创建时间 | 环境创建的时间 | 2021-06-10 10:10:10 |
| 间    |         | 10:10:10            |

## 创建新环境

在 **环境管理** 界面，点击右上角的 **创建** 按钮，进入创建环境的流程。

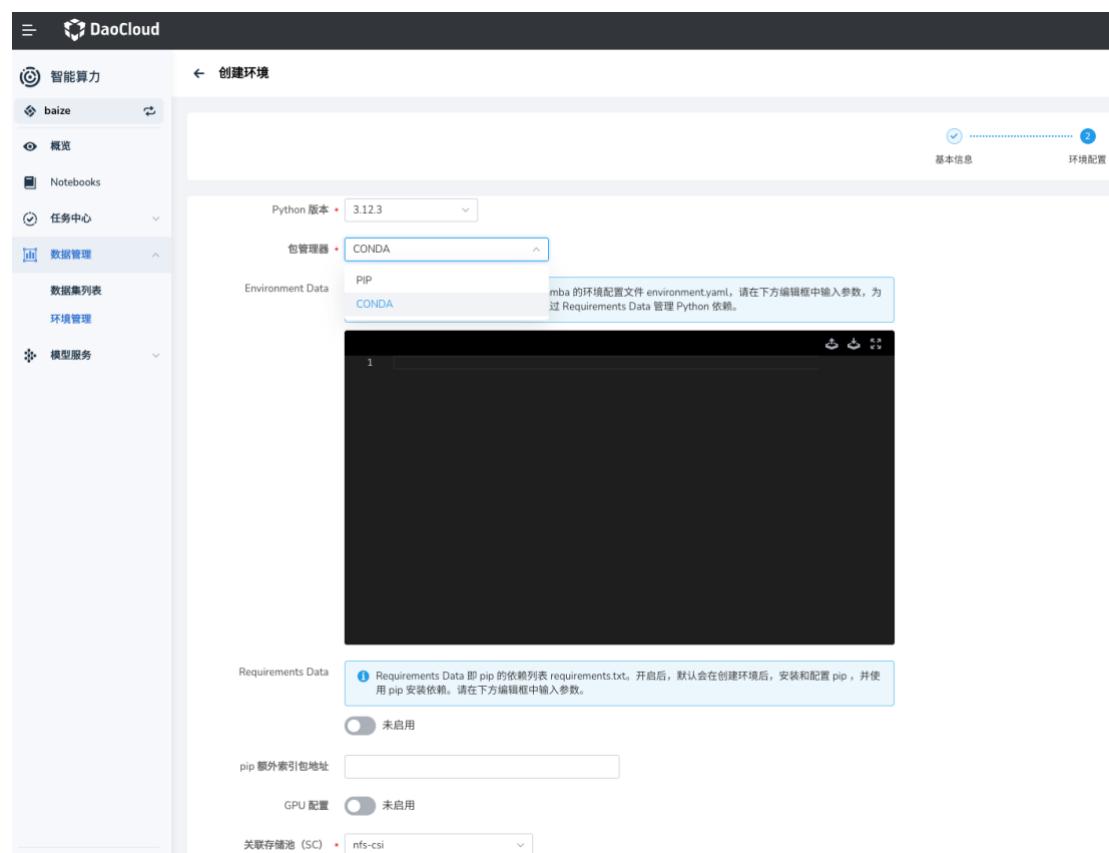


| 字段   | 描述                                    | 举例值            |
|------|---------------------------------------|----------------|
| 名称   | 输入环境的名称，长度为 2-63 个字符，必须以小写字母、数字开头和结尾。 | my-environment |
| 部署位置 | <b>集群：</b> 选择需要部署的集群                  | gpu-cluster    |

| 字段 | 描述                                  | 举例值      |
|----|-------------------------------------|----------|
|    | 命名空间：选择命名空间                         | default  |
| 备注 | 填写备注信息。                             | 这是一个测试环境 |
| 标签 | 为环境添加标签。                            | env:test |
| 注解 | 为环境添加注解。填写完成后，点击 <b>下一步</b> 进入环境配置。 | 注解示例     |

## 配置环境

在环境配置步骤中，用户需要配置 Python 版本和依赖包管理工具。



| 字段               | 描述                                                                                                                                                                                                         | 举例值                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Python 版本        | 选择所需的 Python 版本                                                                                                                                                                                            | 3.12.3                                         |
| 包管理器             | 选择包管理工具，可选 PIP 或 CONDA                                                                                                                                                                                     | PIP                                            |
| Environment Data | 如果选择 PIP：在下方编辑器中输入 requirements.txt 格式的依赖包列表。<br><br>如果选择 CONDA：在下方编辑器中输入 environment.yaml 格式的依赖包列表。                                                                                                       | numpy                                          |
| 其他选项             | <b>pip 额外索引地址</b> ：配置 pip 额外的索引地址；适用于企业内部有自己的私有仓库或者 PIP 加速站点。<br><br><b>GPU 配置</b> ：启用或禁用 GPU 配置；部分涉及到 GPU 的依赖包需要在预加载时配置 GPU 资源。<br><br><b>关联存储</b> ：选择关联的存储配置；环境依赖包会存储在关联存储中。注意：需要使用支持 ReadWriteMany 的存储。 | <a href="https://">https://</a><br>启用<br>my-st |

配置完成后，点击 **创建** 按钮，系统会自动创建并配置新的 Python 环境。

这里有一个 YAML 示例：

```
environment.yaml
name: tensorflow
channels:
- defaults
- conda-forge
dependencies:
- python=3.12
- tensorflow
prefix: /opt/conda/envs/tensorflow
```

## 故障排除

- 如果环境创建失败：

- 检查网络连接是否正常。
  - 确认填写的 Python 版本和包管理器配置无误。
  - 确保所选集群和命名空间可用。
  - 如果依赖预热失败：
    - 检查 requirements.txt 或 environment.yaml 文件格式是否正确。
    - 确认依赖包名称和版本是否正确无误。如遇到其他问题，请联系平台管理员或查看平台帮助文档获取更多支持。
- 

以上即为在 DCE 5.0 AI Lab 中管理 Python 依赖库的基本操作步骤和注意事项。

## 模型支持情况

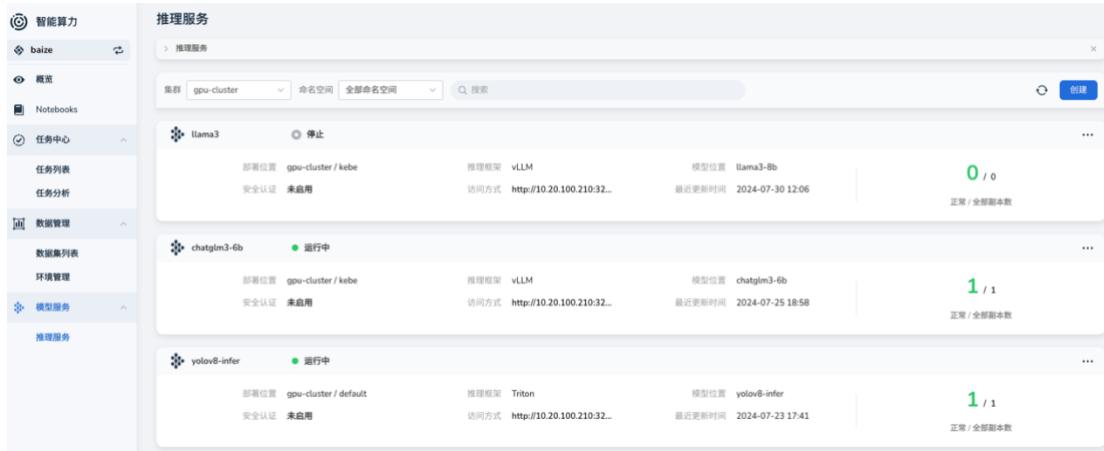
随着 AI Lab 的快速迭代，我们已经支持了多种模型的推理服务，您可以在这里看到所支持的模型信息。

- AI Lab v0.3.0 上线了模型推理服务，针对传统的深度学习模型，方便用户可以直接使用 AI Lab 的推理服务，无需关心模型的部署和维护。
- AI Lab v0.6.0 支持了完整版本的 vLLM 推理能力，支持诸多大语言模型，如 LLaMA、Qwen、ChatGLM 等。

### Note

推理能力的支持与 AI Lab 的版本有关，请查阅 [Release Notes](#) 了解最新版本并及时更新。

您可以在 AI Lab 中使用经过 DCE 5.0 验证过的 GPU 类型；更多细节参阅 [GPU 支持矩阵](#)。



## Triton Inference Server

通过 Triton Inference Server 可以很好的支持传统的深度学习模型，我们目前支持主流的推理后端服务：

| Backend          | 支持模型格式                        | 介绍                                                         |
|------------------|-------------------------------|------------------------------------------------------------|
| pytorch          | TorchScript、PyTorch 2.0 格式的模型 | <a href="#">triton-inference-server/pytorch_backend</a>    |
| tensorflow       | TensorFlow 2.x                | <a href="#">triton-inference-server/tensorflow_backend</a> |
| vLLM(Deprecated) | 与 vLLM 一致                     | 支持的模型和 vLLM support Model 一致                               |

## Danger

使用 Triton 的 Backend vLLM 的方式已被弃用，推荐使用最新支持 vLLM 来部署您的大语言模型。

## vLLM

通过 vLLM 我们可以很快的使用大语言模型，您可以在[这里](#)看到我们支持的模型列表，这通常和 vLLM Support Models 保持一致。

- HuggingFace 模型：我们支持了 HuggingFace 的大部分模型，您可以在 [HuggingFace Model Hub](#) 查看更多模型。
- [vLLM 支持模型](#)列出了支持的大语言模型和视觉语言模型。
- 使用 vLLM 支持框架的模型进行微调后的模型。

## vLLM 新特性

目前，AI Lab 还支持在使用 vLLM 作为推理工具时的一些新特性：

- 在推理模型时，启用 Lora Adapter 来优化模型推理服务
- 提供兼容 OpenAI 的 OpenAPI 接口，方便用户切换到本地推理服务时，可以低成本的快速切换

## 创建 Triton 推理服务

AI Lab 目前提供以 Triton、vLLM 作为推理框架，用户只需简单配置即可快速启动一个高性能的推理服务。

### Danger

使用 Triton 的 Backend vLLM 的方式已被弃用，推荐使用最新支持 vLLM 来部署您的大语言模型。

## Triton 介绍

Triton 是由 NVIDIA 开发的一个开源推理服务器，旨在简化机器学习模型的部署和推理服务。它支持多种深度学习框架，包括 TensorFlow、PyTorch 等，使得用户能够轻松管理和部署不同类型的模型。

## 前提条件

准备模型数据：在数据集管理中纳管模型代码，并保证数据成功预加载，下面以 mnist 手写数字识别的 PyTorch 模型为例。

### Note

待推理的模型在数据集中需要遵以下目录格式：

```
<model-repository-name>
└── <model-name>
```

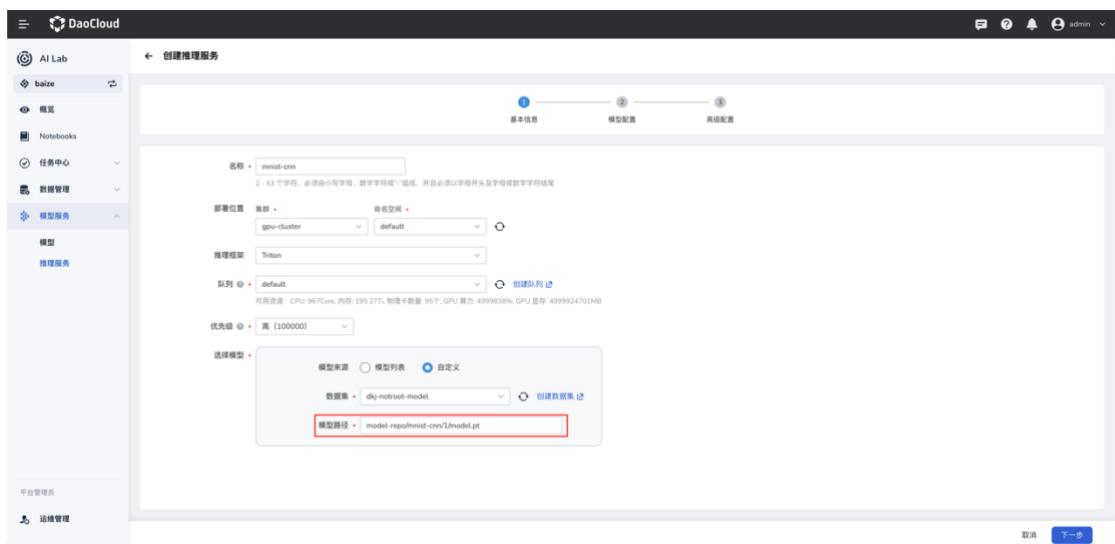
```
└─ <version>
    └─ <model-definition-file>
```

本例中的目录格式为：

```
model-repo
└─ mnist-cnn
    └─ 1
        └─ model.pt
```

## 创建推理服务

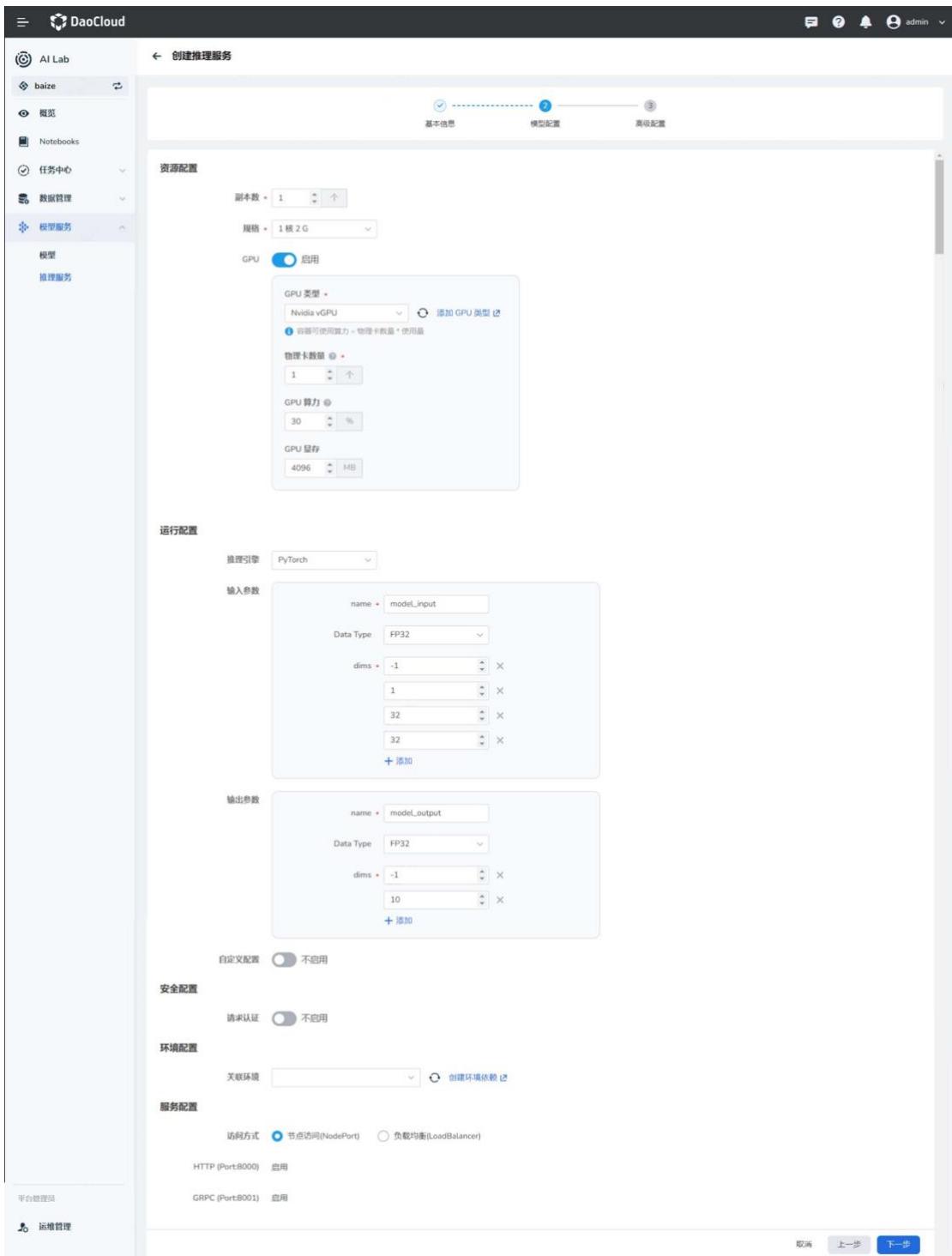
目前已经支持表单创建，可以界面字段提示，进行服务创建。



## 配置模型路径

模型路径 `model-repo/mnist-cnn/1/model.pt` 需要和数据集中的模型目录格式一致。

# 模型配置



配置输入和输出参数

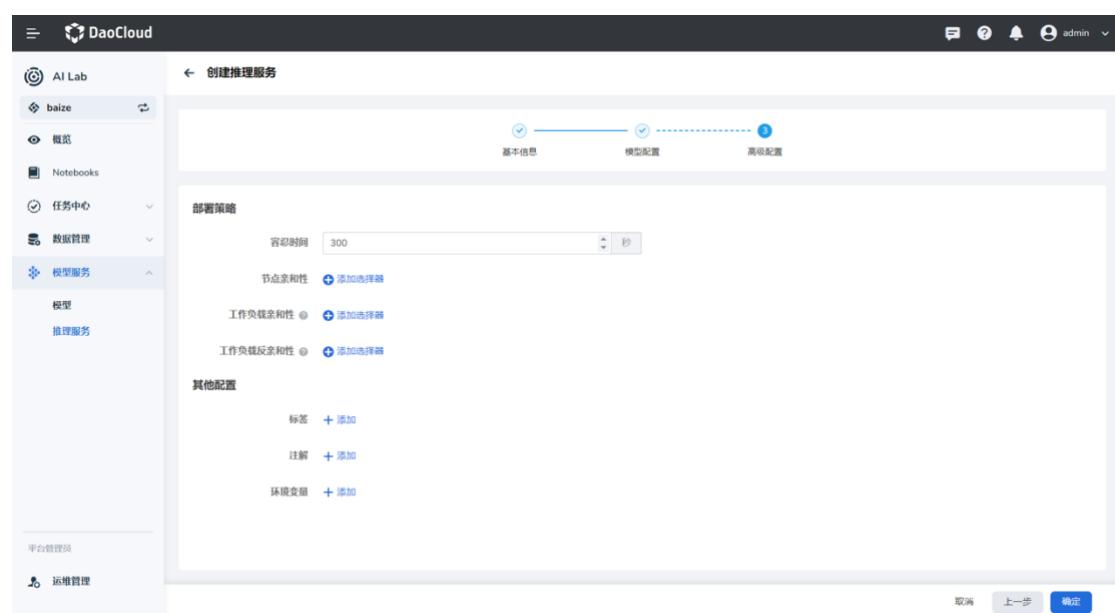
## Note

输入和输出参数的第一个维度默认为 `batchsize` 的大小，设置为 `-1` 可以根据输入的推理数据自动计算 `batchsize`。参数其余维度和数据类型需要与模型输入匹配。

## 配置环境

可以导入 [环境管理](#) 中创建的环境作为推理时的运行环境。

## 高级配置



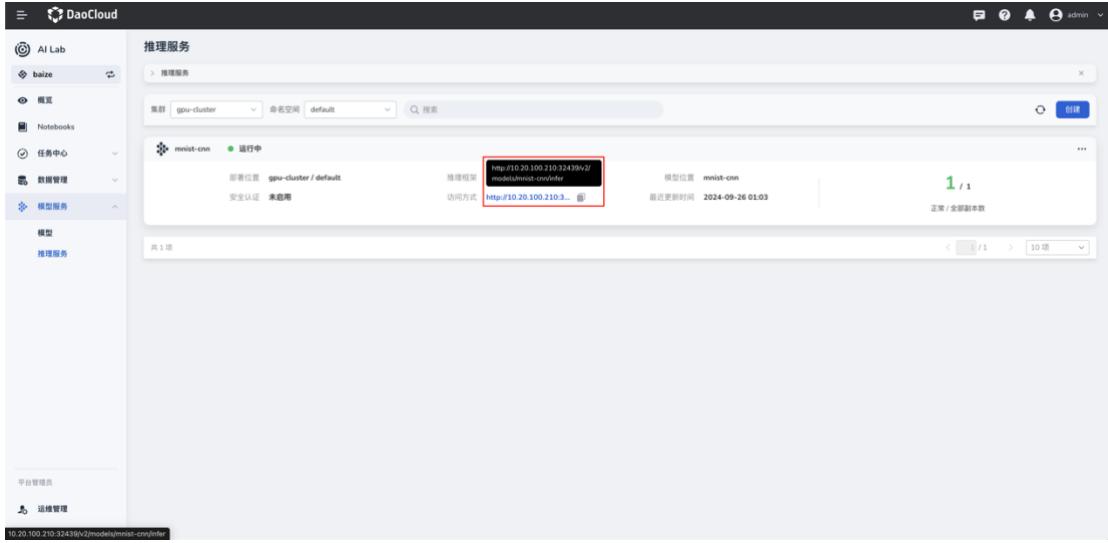
## 配置认证策略

支持 API key 的请求方式认证，用户可以自定义增加认证参数。

## 亲和性调度

支持 根据 GPU 资源等节点配置实现自动化的亲和性调度，同时也方便用户自定义调度策略。

# 访问



## API 访问

- Triton 提供了一个基于 REST 的 API，允许客户端通过 HTTP POST 请求进行模型推理。
- 客户端可以发送 JSON 格式的请求体，其中包含输入数据和相关的元数据。

## HTTP 访问

1. **发送 HTTP POST 请求：** 使用工具如 curl 或 HTTP 客户端库（如 Python 的 requests 库）向 Triton Server 发送 POST 请求。
2. **设置 HTTP 头：** 根据用户配置项自动生成的配置，包含模型输入和输出的元数据。
3. **构建请求体：** 请求体通常包含要进行推理的输入数据，以及模型特定的元数据。

### 示例 CURL 命令

```
curl -X POST "http://<ip>:<port>/v2/models/<inference-name>/infer" \
-H "Content-Type: application/json" \
-d '{
  "inputs": [
    {
      "name": "model_input",
      "shape": [1, 1, 32, 32],
      "datatype": "FP32",
    }
  ]
}'
```

```
        "data": [
            [0.1234, 0.5678, 0.9101, ... ]
        ]
    }
]
```

- <ip> 是 Triton Inference Server 运行的主机地址。
- <port> 是 Triton Inference Server 运行的主机端口号。
- <inference-name> 是所创建的推理服务的名称。
- "name" 要与模型配置中的输入参数的 name 一致。
- "shape" 要与模型配置中的输入参数的 dims 一致。
- "datatype" 要与模型配置中的输入参数的 Data Type 一致。
- "data" 替换为实际的推理数据。

请注意，上述示例代码需要根据你的具体模型和环境进行调整，输入数据的格式和内容也需要符合模型的要求。

## 创建 vLLM 推理服务

AI Lab 支持以 vLLM 作为推理服务，提供全部 vLLM 的能力，同时提供了完全适配 OpenAI 接口定义。

### vLLM 介绍

vLLM 是一个快速且易于使用的用于推理和服务的库，vLLM 旨在极大地提升实时场景下的语言模型服务的吞吐与内存使用效率。vLLM 在速度、灵活性方面具有以下部分特点：

- 连续批处理传入请求；
- 使用 PagedAttention 高效管理注意力键和值内存；
- 与流行的 HuggingFace 型号无缝集成；
- 兼容 OpenAI 的 API 服务器。

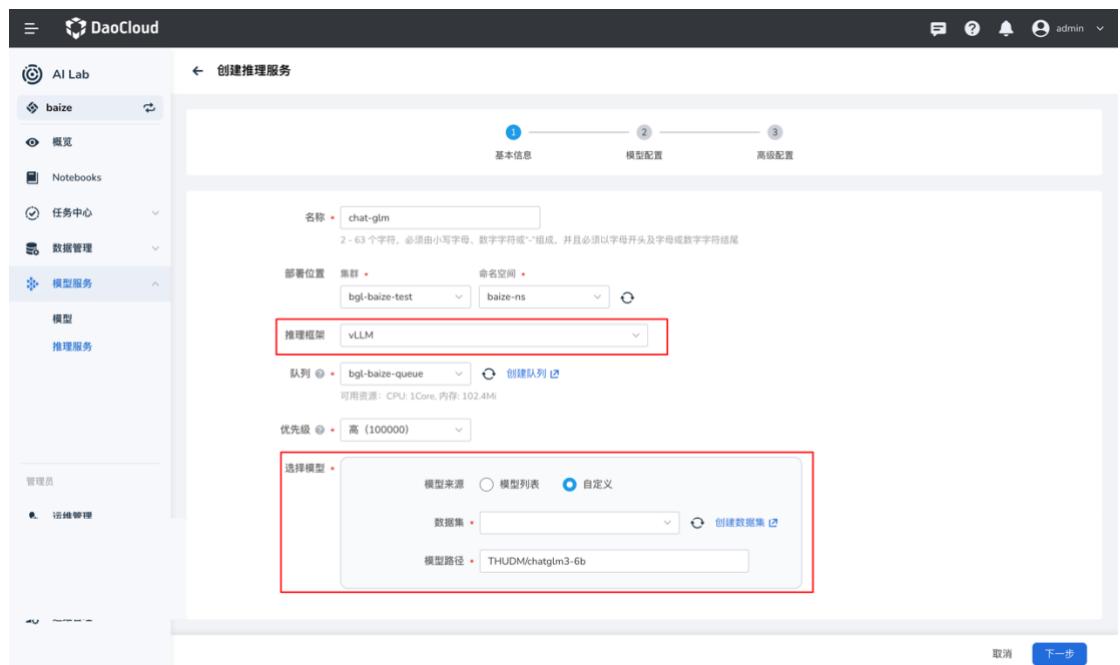
### 前提条件

准备模型数据：在数据集管理中纳管模型代码，并保证数据成功预加载。

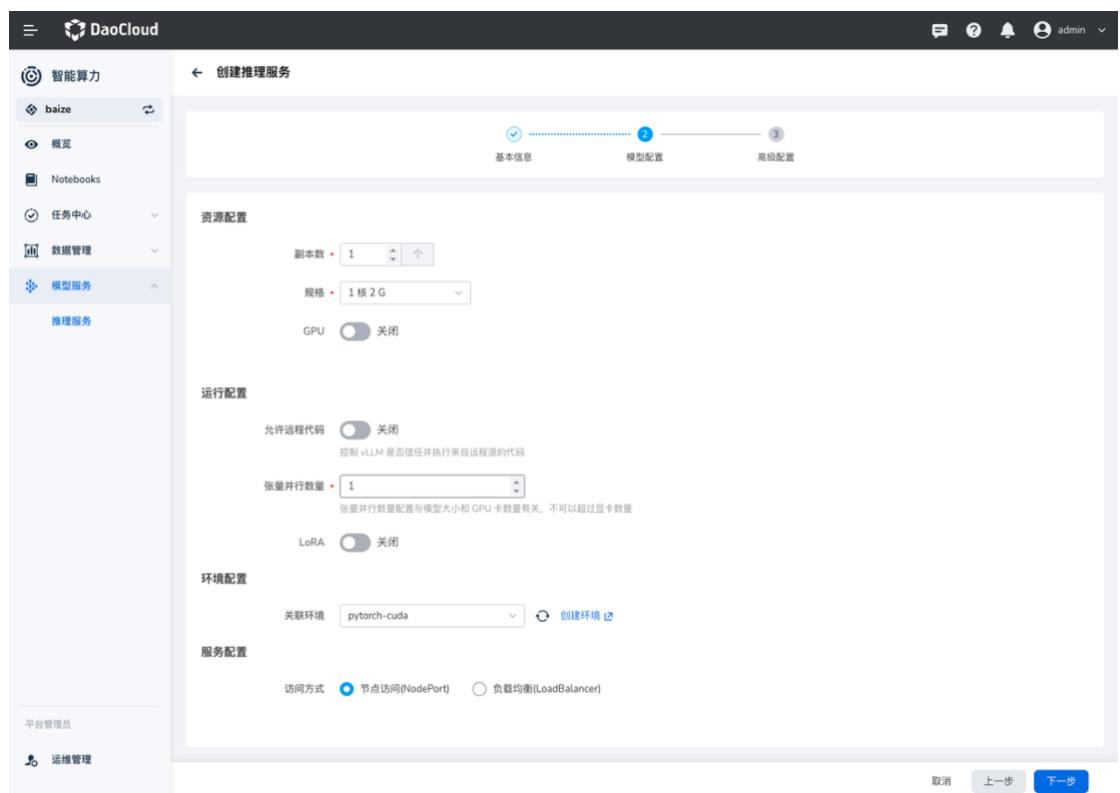
# 创建推理服务

1. 选择 vLLM 推理框架，选择提前创建好的数据集，填写数据集中模型所在的路径信息。

本文推理服务的创建使用 ChatGLM3 模型。



2. 配置推理服务的资源，并调整推理服务运行的参数。



| 参数名    | 描述                                                                                                                                                                                                                                                                                                                                |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GPU 资源 | 根据模型规模以及集群资源可以为推理配置 GPU 资源。                                                                                                                                                                                                                                                                                                       |
| 允许远程代码 | 控制 vLLM 是否信任并执行来自远程源的代码                                                                                                                                                                                                                                                                                                           |
| LoRA   | <b>LoRA</b> 是一种针对深度学习模型的参数高效调整技术。它通过将原始模型参数减少参数数量和计算复杂度。<br>1. <b>--lora-modules</b> : 用来指定特定模块或层进行低秩分解。<br>2. <b>lora_rank</b> : 用来指定 LoRA 模型中每个适配层的最大秩，对于简单的任务，可以选择较小的秩值，而对于复杂的任务可能需要较大的秩值来保证模型性能。<br>3. <b>max_loras</b> : 表示模型中可以包含的 LoRA 模块数量，由模型大小、推理复杂度等因素自定。<br>4. <b>max_cpu_loras</b> : 用于指定在 CPU 环境中可以同时运行的 LoRA 模块数量。 |
| 关联环境   | 通过选择环境预定义推理时所需的环境依赖。                                                                                                                                                                                                                                                                                                              |

## Info

支持配置 LoRA 参数的模型可参考 [vLLM 支持的模型](#)。

3. 在 **高级配置** 中，支持根据 GPU 资源等节点配置实现自动化的亲和性调度，同时也方便用户自定义调度策略。

## 验证推理服务

推理服务创建完成之后，点击推理服务名称进入详情，查看 API 调用方法。通过使用 Curl、Python、Nodejs 等方式验证执行结果。

拷贝详情中的 curl 命令，并在终端中执行命令发送一条模型推理请求，预期输出：

```

~ % curl 'http://10.20.100.210:32178/v1/chat/completions' \
-H "Content-Type: application/json" \
-d '{
  "model": "chatglm3-6b",
  "messages": [{"role": "user", "content": "翻译成中文 hello world!"}],
  "temperature": 0.7
}'
{"id":"cmpl-e6414a863fc1420e99eef899aa99035f","object":"chat.completion","created":1721899395,"model":"chatglm3-6b","choices":[{"index":0,"message":{"role":"assistant","content":"\n你好，世界！"}],"logprobs":null,"finish_reason":"stop","stop_reason":null,"usage":{"prompt_tokens":13,"total_tokens":21,"completion_tokens":8}}'

```

## 运维管理

运维管理是 IT 运维人员日常管理 IT 资源，处理工作的空间。



在这里可以直观地了解当前集群、节点、CPU、GPU、vGPU 等资源的使用状况。

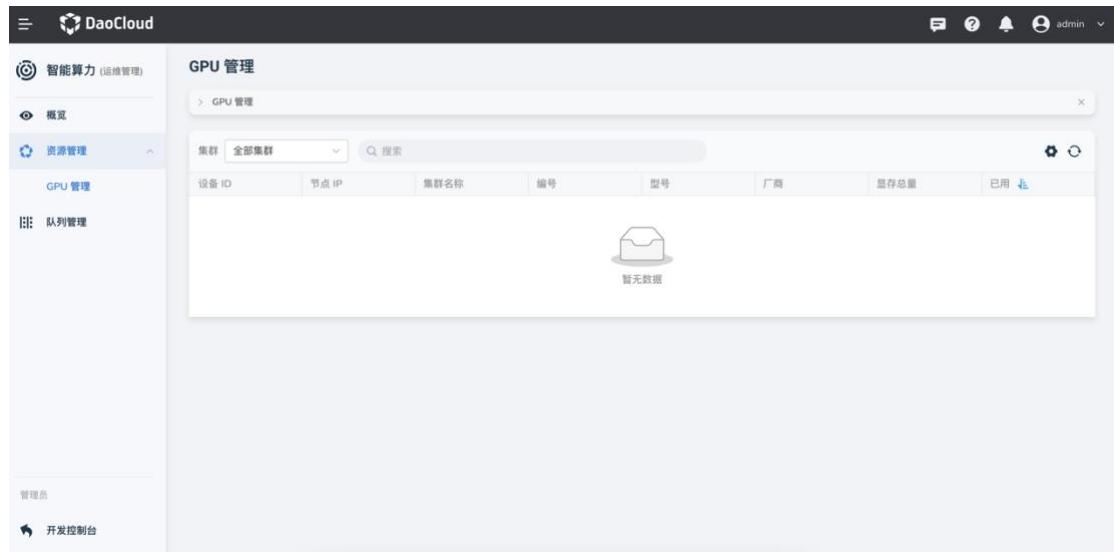
## 常见术语

- **GPU 分配率**: 统计当前集群内所有未完成的任务的 GPU 分配情况，统计请求的 GPU (Request) 与总资源量 (Total) 之间的比例。
- **GPU 利用率**: 统计当前集群中所有运行中的任务的实际资源利用情况，统计实际使用的 GPU (Usage) 与总资源量 (Total) 之间的比例。

## GPU 列表

自动化汇总整个平台中的 GPU 资源信息，提供详尽的 GPU 设备信息展示，可查看各种 GPU 卡的负载统计和任务运行信息。

进入 运维管理 后，点击左侧导航栏的 资源管理 -> GPU 管理，可以查看 GPU 卡和各种训练任务。

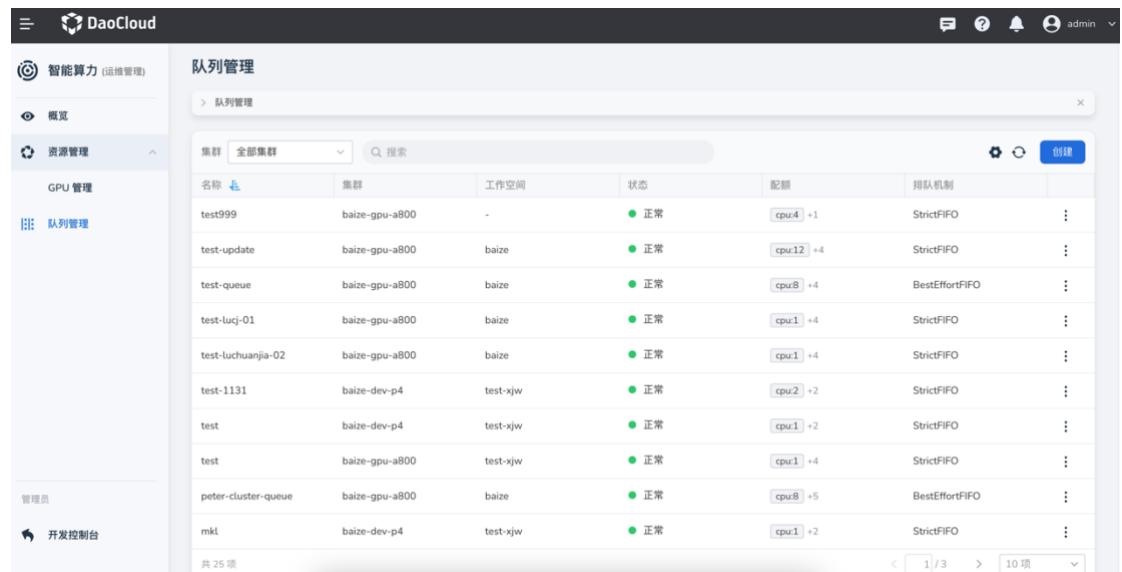


The screenshot shows the DaoCloud interface with the navigation bar at the top. The left sidebar has sections for '智能算力 (运维管理)', '概览', '资源管理' (selected), 'GPU 管理', and '队列管理'. The main content area is titled 'GPU 管理' and shows a search bar with dropdowns for '集群' (All Clusters) and '设备 ID', and fields for '节点 IP', '集群名称', '编号', '型号', '厂商', '显存总量', and '已用'. Below the search bar is a message '暂无数据' (No data available) with a small GPU icon.

## 创建队列

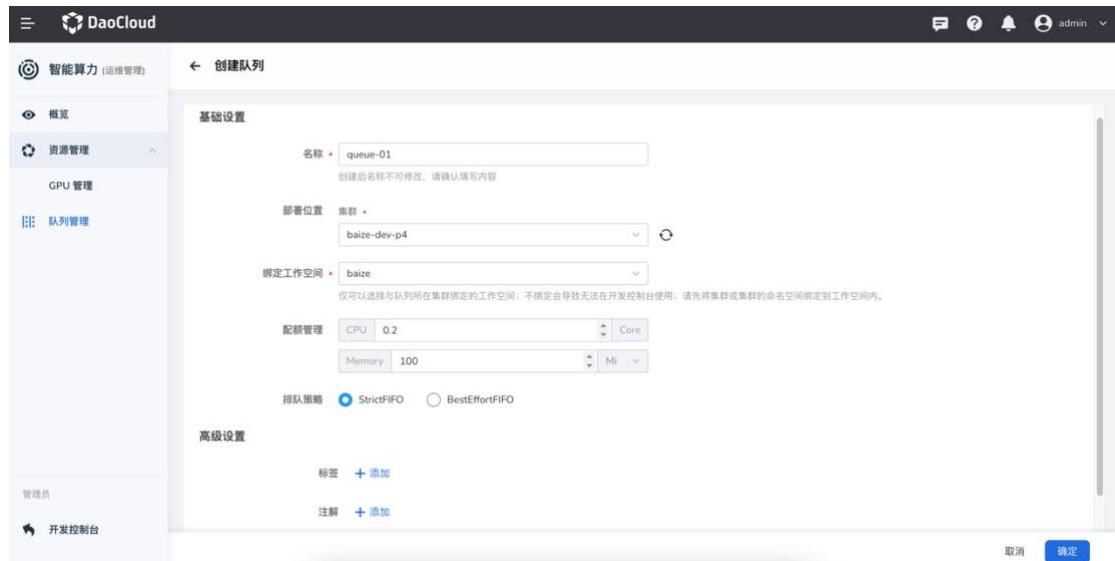
在运维管理模式中，队列可用于调度和优化批处理工作负载，它可以有效地管理在集群上运行的多个任务，通过队列系统来优化资源利用率。

1. 在左侧导航栏中点击 队列管理，点击右侧的 创建 按钮。



The screenshot shows the DaoCloud interface with the navigation bar at the top. The left sidebar has sections for '智能算力 (运维管理)', '概览', '资源管理' (selected), 'GPU 管理', and '队列管理' (selected). The main content area is titled '队列管理' and shows a search bar with dropdowns for '集群' (All Clusters) and '名称', and a '搜索' button. Below the search bar is a table with columns: '名称' (Name), '集群' (Cluster), '工作空间' (Workspace), '状态' (Status), '配额' (Quota), and '排队机制' (Queueing Mechanism). The table lists several entries, such as 'test999' (cluster baize-gpu-a800, workspace -, status 正常, quota cpu:4 +1, queueing mechanism StrictFIFO), 'test-update' (cluster baize-gpu-a800, workspace baize, status 正常, quota cpu:12 +4, queueing mechanism StrictFIFO), etc. A blue '创建' (Create) button is visible in the top right corner of the table header.

2. 系统会预先填充基础设置数据，包括要部署的集群、工作空间、排队策略等。调整这些参数后点击 确定。



3. 屏幕提示创建，返回队列管理列表。点击列表右侧的 |，可以执行更新、删除等更多操作。

| 名称               | 集群           | 工作空间     | 状态 | 配额         | 排队机制           | 操作 |
|------------------|--------------|----------|----|------------|----------------|----|
| test-1131        | baize-dev-p4 | test-xjw | 正常 | cpu2 +2    | StrictFIFO     | ⋮  |
| test             | baize-dev-p4 | test-xjw | 正常 | cpu1 +2    | StrictFIFO     | ⋮  |
| mkl              | baize-dev-p4 | test-xjw | 正常 | cpu1 +2    | StrictFIFO     | ⋮  |
| mk               | baize-dev-p4 | test-xjw | 正常 | cpu1 +2    | StrictFIFO     | ⋮  |
| kebe-1c2g        | baize-dev-p4 | baize    | 正常 | cpu10 +2   | StrictFIFO     | ⋮  |
| kebe             | baize-dev-p4 | test-xjw | 正常 | cpu10 +2   | StrictFIFO     | ⋮  |
| cluster-queue111 | baize-dev-p4 | baize    | 正常 | cpu10 +3   | BestEffortFIFO | ⋮  |
| cluster-queue    | baize-dev-p4 | baize    | 正常 | cpu90 +3   | BestEffortFIFO | ⋮  |
| ccv              | baize-dev-p4 | test-xjw | 正常 | cpu1 +2    | StrictFIFO     | ⋮  |
| asdd             | baize-dev-p4 | test-xjw | 正常 | cpu600m +2 | StrictFIFO     | ⋮  |

## 最佳实践

### 部署 NFS 做数据集预热

**网络文件系统 (NFS)** 允许远程主机通过网络挂载文件，并像本地文件系统一样进行交互。这使系统管理员能够将资源集中到网络服务器上进行管理。

**数据集** 是 DCE 5.0 AI Lab 中的核心数据管理功能，将 MLOps 生命周期中对于数据的依赖统一抽象为数据集；支持用户将各类数据纳管到数据集内，以便训练任务可以直接使用数据集中的数据。

当远端数据不在工作集群内时，数据集提供了自动进行预热的能力，支持 Git、S3、HTTP 等数据提前预热到集群本地。

数据集需要一个支持 `ReadWriteMany` 模式的存储服务对远端数据进行预热，推荐在集群内部署 NFS。

本文主要介绍了如何快速部署一个 NFS 服务，并将其添加为集群的存储类。

## 准备工作

- NFS 默认使用节点的存储作为数据缓存点，因此需要确认磁盘本身有足够的磁盘空间。
- 安装方式使用 Helm 与 Kubectl，请确保已经安装好。

## 部署过程

一共需要安装几个组件：

- NFS Server
- csi-driver-nfs
- StorageClass

### 初始化命名空间

所有系统组件会安装到 nfs 命名空间内，因此需要先创建此命名空间。

```
kubectl create namespace nfs
```

### 安装 NFS Server

这里是一个简单的 YAML 部署文件，可以直接使用。

#### Note

注意检查 `image:`，根据集群所在位置情况，可能需要修改为国内镜像。

#### nfs-server.yaml

```
kind: Service
apiVersion: v1
```

```

metadata:
  name: nfs-server
  namespace: nfs
  labels:
    app: nfs-server
spec:
  type: ClusterIP
  selector:
    app: nfs-server
  ports:
    - name: tcp-2049
      port: 2049
      protocol: TCP
    - name: udp-111
      port: 111
      protocol: UDP
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-server
  namespace: nfs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nfs-server
  template:
    metadata:
      name: nfs-server
      labels:
        app: nfs-server
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
        - name: nfs-server
          image: itsthenetwork/nfs-server-alpine:latest
          env:
            - name: SHARED_DIRECTORY
              value: "/exports"
      volumeMounts:
        - mountPath: /exports
          name: nfs-vol

```

```
    securityContext:
      privileged: true
    ports:
      - name: tcp-2049
        containerPort: 2049
        protocol: TCP
      - name: udp-111
        containerPort: 111
        protocol: UDP
    volumes:
      - name: nfs-vol
        hostPath:
          path: /nfsdata
          type: DirectoryOrCreate
```

将上述 YAML 保存为 `nfs-server.yaml`，然后执行以下命令进行部署：

```
kubectl -n nfs apply -f nfs-server.yaml
```

```
# 检查部署结果
kubectl -n nfs get pod,svc
```

## 安装 csi-driver-nfs

安装 `csi-driver-nfs` 需要使用 Helm，请注意提前安装。

```
# 添加 Helm 仓库
helm repo add csi-driver-nfs
https://mirror.ghproxy.com/https://raw.githubusercontent.com/kubernetes-csi/csi-driver-
nfs/master/charts
helm repo update csi-driver-nfs

# 部署 csi-driver-nfs
# 这里参数主要优化了镜像地址，加速国内下载
helm upgrade --install csi-driver-nfs csi-driver-nfs/csi-driver-nfs \
  --set image.nfs.repository=k8s.m.daocloud.io/sig-storage/nfsplugin \
  --set image.csiProvisioner.repository=k8s.m.daocloud.io/sig-storage/csi-provisioner \
  --set image.livenessProbe.repository=k8s.m.daocloud.io/sig-storage/livenessprobe \
  --set image.nodeDriverRegistrar.repository=k8s.m.daocloud.io/sig-storage/csi-node-
driver-registrar \
  --namespace nfs \
  --version v4.5.0
```

## Warning

csi-nfs-controller 的镜像并未全部支持 helm 参数，需要手工修改 deployment 的 image 字段。将 image: registry.k8s.io 改为 image: k8s.dockerproxy.com 以加速国内下载。

## 创建 StorageClass

将以下 YAML 保存为 nfs-sc.yaml:

### nfs-sc.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-csi
provisioner: nfs.csi.k8s.io
parameters:
  server: nfs-server.nfs.svc.cluster.local
  share: /
  # csi.storage.k8s.io/provisioner-secret is only needed for providing mountOptions in
DeleteVolume
  # csi.storage.k8s.io/provisioner-secret-name: "mount-options"
  # csi.storage.k8s.io/provisioner-secret-namespace: "default"
reclaimPolicy: Retain
volumeBindingMode: Immediate
mountOptions:
  - nfsvers=4.1
```

然后执行以下命令进行部署：

```
kubectl apply -f nfs-sc.yaml
```

## 测试

创建数据集，并将数据集的 **关联存储类**，预热方式设置为 NFS，即可将远端数据预热到集群内。

数据集创建成功后，可以看到数据集的状态为 预热中，等待预热完成后即可使用。

## 常见问题

### 缺少必要的 NFS 客户端软件 /sbin/mount

bad option; for several filesystems (e.g. nfs, cifs) you might need a /sbin/mount.<type> helper program.

在运行 Kubernetes 的节点机器上，确保已安装 NFS 客户端：

#### [Ubuntu/DebianCentOS/RHEL](#)

运行以下命令安装 NFS 客户端：

```
sudo apt-get update  
sudo apt-get install nfs-common
```

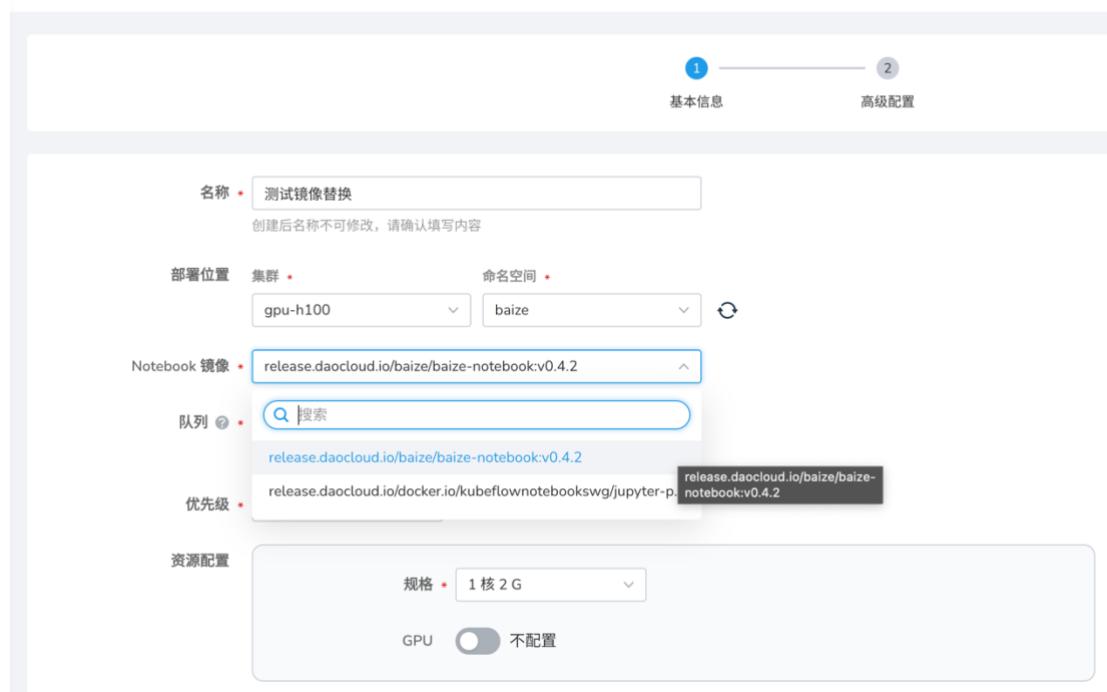
检查 NFS 服务器配置，确保 NFS 服务器正在运行且配置正确。你可以尝试运行以下命令手动挂载来测试：

```
sudo mkdir -p /mnt/test  
sudo mount -t nfs <nfs-server>:/nfsdata /mnt/test
```

## 更新 Notebook 内置镜像

在 Notebook 中，默认提供了多个可用的基础镜像，供开发者选择；大部分情况下，这会满足开发者的使用。

[← 创建 Notebook](#)



DaoCloud 提供了一个默认的 Notebook 镜像，包含了所需的任何开发工具和资料。

`baize/baize-notebook`

这个 Notebook 里面包含了基础的开发工具，以 `baize-notebook:v0.5.0`（2024 年 5 月 30 日）为例，相关依赖及版本如下：

| 依赖           | 版本 编号   | 介绍                                |
|--------------|---------|-----------------------------------|
| Ubuntu       | 22.04.3 | 默认 OS                             |
| Python       | 3.11.6  | 默认 Python 版本                      |
| pip          | 23.3.1  |                                   |
| conda(mamba) | 23.3.1  |                                   |
| jupyterlab   | 3.6.6   | JupyterLab 镜像，提供完整的 Notebook 开发体验 |

| 依赖         | 版本编号    | 介绍                                              |
|------------|---------|-------------------------------------------------|
| codeserver | v4.89.1 | 主流 Code 开发工具，方便用户使用熟悉的工具进行开发体验                  |
| *baizectl  | v0.5.0  | DaoCloud 内置 CLI 任务管理工具                          |
| *SSH       | -       | 支持本地 SSH 直接访问到 Notebook 容器内                     |
| *kubectl   | v1.27   | Kubernetes CLI，可以使用 kubectl 在 Notebook 内 管理容器资源 |

## Note

随着版本发展，DCE 5.0 会主动维护并在每次版本迭代时更新。

但有时用户可能需要自定义镜像，本文介绍了如何更新镜像，并增加到 Notebook 创建界面中进行选择。

## 构建自定义镜像（仅供参考）

### Note

注意，构建新镜像 需要以 `baize-notebook` 作为基础镜像，以保证 Notebook 的正常运行。

在构建自定义镜像时，建议先了解 `baize-notebook` 镜像的 Dockerfile，以便更好地理解如何构建自定义镜像。

### baize-notebook 的 Dockerfile

```
ARG BASE_IMG=docker.m.daocloud.io/kubeflow-notebookswg/jupyter:v1.8.0
```

```
FROM $BASE_IMG
```

```
USER root
```

```
# install - useful linux packages
```

```

RUN export DEBIAN_FRONTEND=noninteractive \
  && apt-get -yq update \
  && apt-get -yq install --no-install-recommends \
    openssh-server git git-lfs bash-completion \
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*

# remove default s6 jupyterlab run script
RUN rm -rf /etc/services.d/jupyterlab

# install - useful jupyter plugins
RUN mamba install -n base -y jupyterlab-language-pack-zh-cn \
  && mamba clean --all -y

ARG CODESERVER_VERSION=4.89.1
ARG TARGETARCH

RUN curl -fsSL "https://github.com/coder/code-
server/releases/download/v$CODESERVER_VERSION/code-
server_${CODESERVER_VERSION}_${TARGETARCH}.deb" -o /tmp/code-server.deb \
  && dpkg -i /tmp/code-server.deb \
  && rm -f /tmp/code-server.deb

ARG CODESERVER_PYTHON_VERSION=2024.4.1
ARG CODESERVER_JUPYTER_VERSION=2024.3.1
ARG CODESERVER_LANGUAGE_PACK_ZH_CN=1.89.0
ARG CODESERVER_YAML=1.14.0
ARG CODESERVER_DOTENV=1.0.1
ARG CODESERVER_EDITORCONFIG=0.16.6
ARG CODESERVER_TOML=0.19.1
ARG CODESERVER_GITLENS=15.0.4

# configure for code-server extensions
# #
https://github.com/kubeflow/kubeflow/blob/709254159986d2cc99e675d0fad5a128ddeb0917/c
omponents/example-notebook-servers/codeserver-python/Dockerfile
# # and
# #
https://github.com/kubeflow/kubeflow/blob/709254159986d2cc99e675d0fad5a128ddeb0917/c
omponents/example-notebook-servers/codeserver/Dockerfile
RUN code-server --list-extensions --show-versions \
  && code-server --list-extensions --show-versions \
  && code-server \

```

```

--install-extension MS-CEINTL.vscode-language-pack-zh-
hans@$CODESERVER_LANGUAGE_PACK_ZH_CN \
--install-extension ms-python.python@$CODESERVER_PYTHON_VERSION \
--install-extension ms-toolsai.jupyter@$CODESERVER_JUPYTER_VERSION \
--install-extension redhat.vscode-yaml@$CODESERVER_YAML \
--install-extension mikestead.dotenv@$CODESERVER_DOTENV \
--install-extension EditorConfig.EditorConfig@$CODESERVER_EDITORCONFIG \
--install-extension tamasfe.even-better-toml@$CODESERVER_TOML \
--install-extension eamodio.gitlens@$CODESERVER_GITLENS \
--install-extension catppuccin.catppuccin-vsc-pack \
--force \
&& code-server --list-extensions --show-versions

# configure for code-server
RUN mkdir -p /home/${NB_USER}/.local/share/code-server/User \
    && chown -R ${NB_USER}:users /home/${NB_USER} \
    && cat <<EOF > /home/${NB_USER}/.local/share/code-server/User/settings.json
{
    "gitlens.showWelcomeOnInstall": false,
    "workbench.colorTheme": "Catppuccin Mocha",
}
EOF

RUN mkdir -p /tmp_home/${NB_USER}/.local/share \
    && mv /home/${NB_USER}/.local/share/code-server /tmp_home/${NB_USER}/.local/share

# set ssh configuration
RUN mkdir -p /run/sshd \
    && chown -R ${NB_USER}:users /etc/ssh \
    && chown -R ${NB_USER}:users /run/sshd \
    && sed -i "/#\?Port/s/^.*$/Port 2222/g" /etc/ssh/sshd_config \
    && sed -i "/#\?PasswordAuthentication/s/^.*$/PasswordAuthentication no/g" \
/etc/ssh/sshd_config \
    && sed -i "/#\?PubkeyAuthentication/s/^.*$/PubkeyAuthentication yes/g" \
/etc/ssh/sshd_config \
    && rclone_version=v1.65.0 && \
        arch=$(uname -m | sed -E 's/x86_64/amd64/g;s/aarch64/arm64/g') && \
        filename=rclone-$rclone_version-linux-$arch && \
        curl -fsSL \
https://github.com/rclone/rclone/releases/download/${rclone_version}/${filename}.zip -o \
${filename}.zip && \
        unzip ${filename}.zip && mv ${filename}/rclone /usr/local/bin && rm -rf ${filename} \
${filename}.zip

```

```

# Init mamba
RUN mamba init --system

# init baize-base environment for essential python packages
RUN mamba create -n baize-base -y python \
    && /opt/conda/envs/baize-base/bin/pip install tensorboard \
    && mamba clean --all -y \
    && ln -s /opt/conda/envs/baize-base/bin/tensorboard /usr/local/bin/tensorboard

# prepare baize-runtime-env directory
RUN mkdir -p /opt/baize-runtime-env \
    && chown -R ${NB_USER}:users /opt/baize-runtime-env

ARG APP
ARG PROD_NAME
ARG TARGETOS

COPY out/$TARGETOS/$TARGETARCH/data-loader /usr/local/bin/
COPY out/$TARGETOS/$TARGETARCH/baizectl /usr/local/bin/

RUN chmod +x /usr/local/bin/baizectl /usr/local/bin/data-loader && \
    echo "source /etc/bash_completion" >> /opt/conda/etc/profile.d/conda.sh && \
    echo "source <(baizectl completion bash)" >> /opt/conda/etc/profile.d/conda.sh && \
    echo "source <(kubectl completion bash)" >> /opt/conda/etc/profile.d/conda.sh && \
    echo '[ -f /run/baize-env ] && export $(cat /run/baize-env | xargs)' >> \
    /opt/conda/etc/profile.d/conda.sh && \
    echo 'alias conda="mamba"' >> /opt/conda/etc/profile.d/conda.sh

USER ${NB_UID}

```

## 构建你的镜像

```

ARG BASE_IMG=release.daocloud.io/baize/baize-notebook:v0.5.0

FROM $BASE_IMG
USER root

# Do Customization
RUN mamba install -n baize-base -y pytorch torchvision torchaudio ccpuonly -c pytorch \
    && mamba install -n baize-base -y tensorflow \
    && mamba clean --all -y

USER ${NB_UID}

```

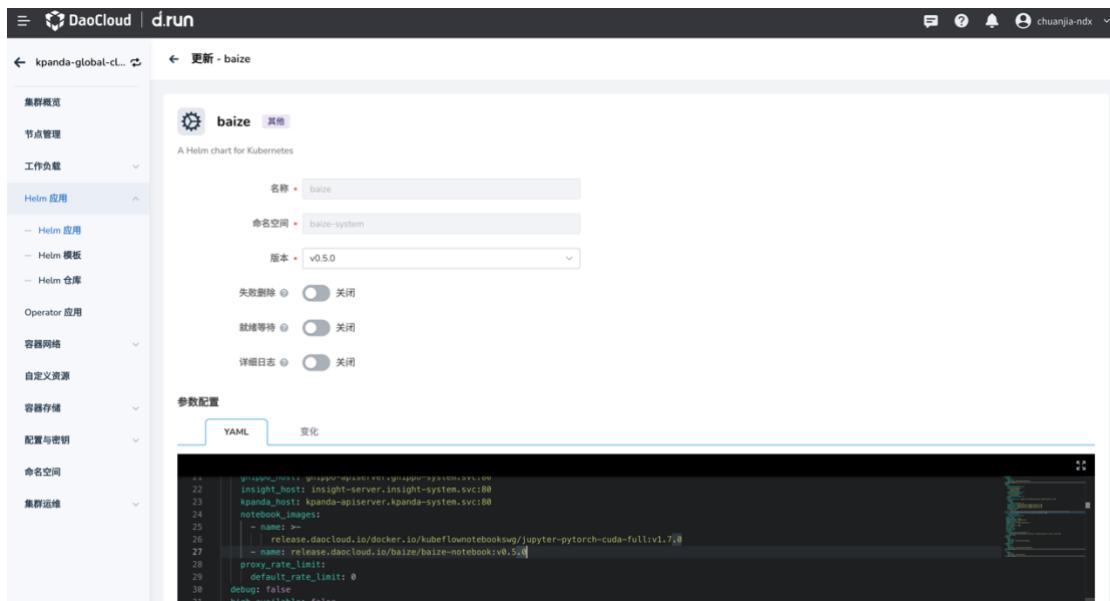
# 增加到 Notebook 镜像列表（Helm）

## Warning

注意，必须由平台管理员操作，谨慎变更。

目前，镜像选择器需要通过更新 baize 的 Helm 参数来修改，具体步骤如下：

在 kpanda-global-cluster 全局服务集群的 Helm 应用列表，找到 baize，进入更新页面，在 YAML 参数中修改 Notebook 镜像：



注意参数修改的路径如下 global.config.notebook\_images：

```
...
global:
...
config:
  notebook_images:
    ...
    names: release.daocloud.io/baize/baize-notebook:v0.5.0
    # 在这里增加你的镜像信息
```

更新完成之后，待 Helm 应用重启成功之后，可以在 Notebook 创建界面中的选择镜像看到新的镜像。

# Checkpoint 机制及使用介绍

在深度学习的实际场景中，模型训练一般都会持续一段时间，这对分布式训练任务的稳定性和效率提出了更高的要求。而且，在实际训练的过程中，异常中断会导致训练过程中的模型状态丢失，需要重新开始训练，这不仅浪费了时间和资源，这在 LLM 训练中尤为明显，而且也会影响模型的训练效果。

能够在训练过程中保存模型的状态，以便在训练过程中出现异常时能够恢复模型状态，变得至关重要。Checkpoint 就是目前主流的解决方案，本文将介绍 Checkpoint 机制的基本概念和在 PyTorch 和 TensorFlow 中的使用方法。

## 什么是 Checkpoint？

Checkpoint 是在模型训练过程中保存模型状态的机制。通过定期保存 Checkpoint，可以在以下情况下恢复模型：

- 训练过程中断（如系统崩溃或手动中断）
- 需要在某个训练阶段进行评估
- 希望在不同的实验中复用模型

## PyTorch

在 PyTorch 中，`torch.save` 和 `torch.load` 是用于保存和加载模型的基本函数。

### PyTorch 保存 Checkpoint

在 PyTorch 中，通常使用 `state_dict` 保存模型的参数。以下是一个简单的示例：

```
import torch
import torch.nn as nn

# 假设我们有一个简单的神经网络
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc = nn.Linear(10, 2)

    def forward(self, x):
        return self.fc(x)

# 初始化模型和优化器
model = SimpleModel()
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# 训练模型...
# 保存 Checkpoint
checkpoint_path = 'model_checkpoint.pth'
torch.save({
    'epoch': 10,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'loss': 0.02,
}, checkpoint_path)
```

## PyTorch 恢复 Checkpoint

加载模型时，需要恢复模型参数和优化器状态，并继续训练或推理：

```
# 恢复 Checkpoint
checkpoint = torch.load('model_checkpoint.pth')
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']

# 继续训练或推理...
```

- `model_state_dict`: 模型参数
- `optimizer_state_dict`: 优化器状态
- `epoch`: 当前训练轮数
- `loss`: 损失值
- `learning_rate`: 学习率
- `best_accuracy`: 最佳准确率

## TensorFlow

TensorFlow 提供了 `tf.train.Checkpoint` 类来管理模型和优化器的保存和恢复。

## TensorFlow 保存 Checkpoint

以下是一个在 TensorFlow 中保存 Checkpoint 的示例：

```
import tensorflow as tf

# 假设我们有一个简单的模型
model = tf.keras.Sequential([
    tf.keras.layers.Dense(2, input_shape=(10,))
])
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# 定义 Checkpoint
checkpoint = tf.train.Checkpoint(optimizer=optimizer, model=model)
checkpoint_dir = './checkpoints'
checkpoint_prefix = f'{checkpoint_dir}/ckpt'

# 训练模型...
# 保存 Checkpoint
checkpoint.save(file_prefix=checkpoint_prefix)
```

## Note

使用 DCE 5.0 AI Lab 的用户，可以直接将高性能存储挂载为 Checkpoint 目录，以提高 Checkpoint 保存和恢复的速度。

## TensorFlow 恢复 Checkpoint

加载 Checkpoint 并恢复模型和优化器状态：

```
# 恢复 Checkpoint
latest_checkpoint = tf.train.latest_checkpoint(checkpoint_dir)
checkpoint.restore(latest_checkpoint)

# 继续训练或推理...
```

## TensorFlow 在分布式训练的 Checkpoint 管理

TensorFlow 在分布式训练中管理 Checkpoint 的主要方法如下：

- 使用 `tf.train.Checkpoint` 和 `tf.train.CheckpointManager`
- `checkpoint = tf.train.Checkpoint(model=model, optimizer=optimizer)`
- `manager = tf.train.CheckpointManager(checkpoint, directory='/tmp/model', max_to_keep=3)`
- 在分布式策略中保存 Checkpoint

- strategy = tf.distribute.MirroredStrategy()
- with strategy.scope():
  - checkpoint = tf.train.Checkpoint(model=model, optimizer=optimizer)
  - manager = tf.train.CheckpointManager(checkpoint, directory='/tmp/model', max\_to\_keep=3)
  - 只在主节点 (chief worker) 保存 Checkpoint
- if strategy.cluster\_resolver.task\_type == 'chief':
  - manager.save()
- 使用 MultiWorkerMirroredStrategy 时的特殊处理
- strategy = tf.distribute.MultiWorkerMirroredStrategy()
- with strategy.scope():
  - # 模型定义
  - ...
  - checkpoint = tf.train.Checkpoint(model=model, optimizer=optimizer)
  - manager = tf.train.CheckpointManager(checkpoint, '/tmp/model', max\_to\_keep=3)
  - 
  - def \_chief\_worker(task\_type, task\_id):
    - return task\_type is None or task\_type == 'chief' or (task\_type == 'worker' and task\_id == 0)
    - 
    - if \_chief\_worker(strategy.cluster\_resolver.task\_type, strategy.cluster\_resolver.task\_id):
      - manager.save()
  - 使用分布式文件系统
 

确保所有工作节点都能访问到同一个 Checkpoint 目录，通常使用分布式文件系统如 HDFS 或 GCS。
  - 异步保存
 

使用 tf.keras.callbacks.ModelCheckpoint 并设置 save\_freq 参数可以在训练过程中异步保存 Checkpoint。
  - Checkpoint 恢复
 

```
status = checkpoint.restore(manager.latest_checkpoint)
status.assert_consumed()
```
  - 性能优化
    - 使
 

用 tf.train.experimental.enable\_mixed\_precision\_graph\_rewrite()  
启用混合精度训练
    - 调整保存频率，避免过于频繁的 I/O 操作
    - 考虑使用 tf.saved\_model.save() 保存整个模型，而不仅仅是权重

## 注意事项

1. 定期保存：根据训练时间和资源消耗，决定合适的保存频率。如每个 epoch 或每隔一定的训练步数。
2. 保存多个 **Checkpoint**：保留最新的几个 Checkpoint 以防止文件损坏或不适用的情况。
3. 记录元数据：在 Checkpoint 中保存额外的信息，如 epoch 数、损失值等，以便更好地恢复训练状态。
4. 使用版本控制：保存不同实验的 Checkpoint，便于对比和复用。
5. 验证和测试：在训练的不同阶段使用 Checkpoint 进行验证和测试，确保模型性能和稳定性。

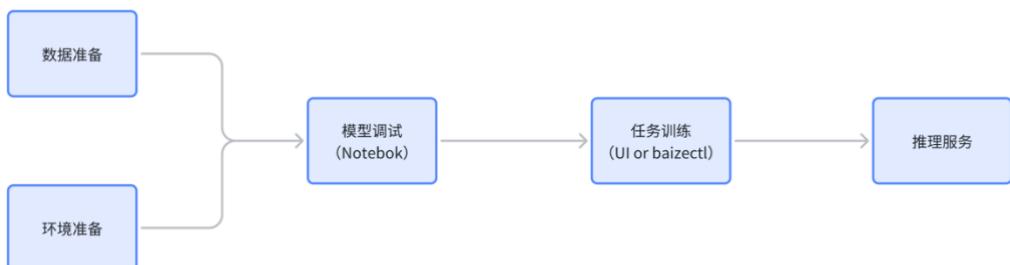
## 结论

Checkpoint 机制在深度学习训练中起到了关键作用。通过合理使用 PyTorch 和 TensorFlow 中的 Checkpoint 功能，可以有效提高训练的可靠性和效率。希望本文所述的方法和最佳实践能帮助你更好地管理深度学习模型的训练过程。

## 使用 AI Lab 微调 ChatGLM3 模型

本文以 ChatGLM3 模型为例，演示如何在 DCE 5.0 AI Lab 中使用 LoRA（Low-Rank Adaptation，低秩自适应）微调 ChatGLM3 模型。Demo 程序来自 [ChatGLM3 官方案例](#)。

微调的大致流程为：



## 环境依赖

- GPU 显存至少 20GB，推荐使用 RTX4090、NVIDIA A/H 系列显卡

- 可用磁盘空间至少 200GB
- CPU 至少 8 核, 推荐 16 核
- 内存 64GB, 推荐 128GB

## Info

在开始体验之前, 请检查 DCE 5.0 以及 [AI Lab 部署](#)正确, GPU 队列资源初始化成功, 且算力资源充足。

## 数据准备

利用 DCE 5.0 AI Lab 提供的数据集管理功能, 快速将微调大模型所需的数据进行预热及持久化, 减少因为准备数据导致的 GPU 资源占用, 提高资源利用效率。

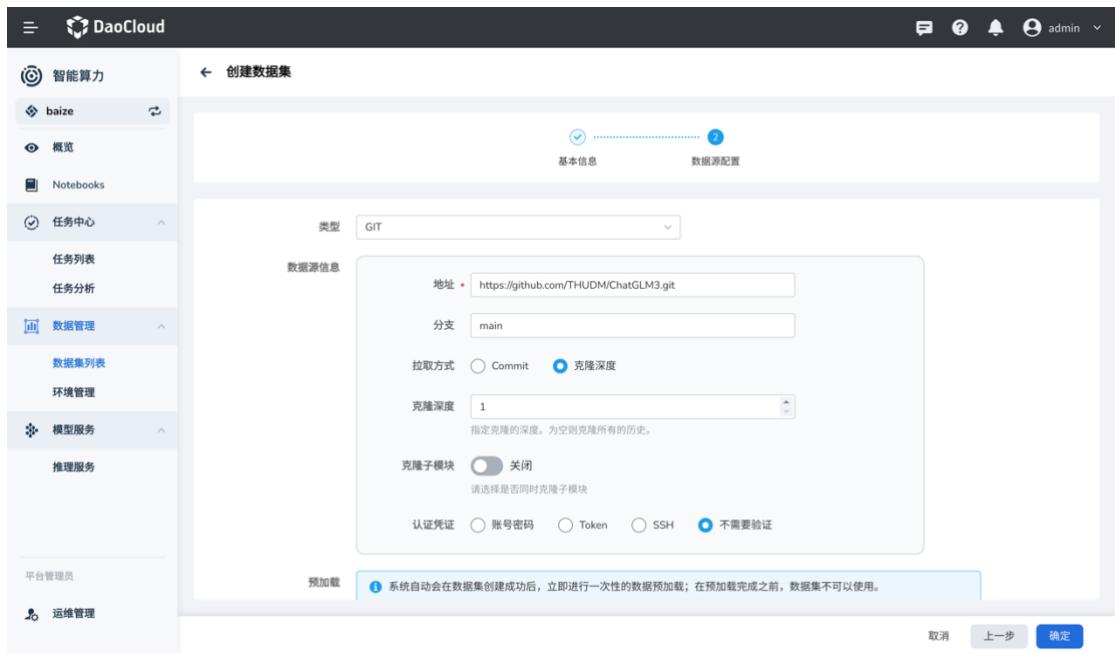
| 名称             | 类型   | 数据源                                   | 状态    | 预加载 | 创建时间             |
|----------------|------|---------------------------------------|-------|-----|------------------|
| bbbb           | GIT  | http://baidu.com                      | ● 同步中 | 启用  | 2024-05-17 14:20 |
| baize-tools    | HTTP | http://baize-ai.daocloud.io/playgr... | ● 同步中 | 启用  | 2024-05-16 15:21 |
| finetune       | PVC  | pvc://0-demo-pool-0-0                 | ● 正常  | 不启用 | 2024-05-16 13:49 |
| nfs-models     | NFS  | nfs://10.64.65.91/data/ndxlgpu/...    | ● 正常  | 不启用 | 2024-05-16 11:59 |
| test-05        | GIT  | http://jqc.c                          | ● 同步中 | 启用  | 2024-05-14 11:46 |
| 123            | GIT  | http://jqc.c                          | ● 同步中 | 启用  | 2024-05-14 11:26 |
| test-03        | GIT  | http://asd.c                          | ● 同步中 | 启用  | 2024-05-13 18:12 |
| merbridge-code | GIT  | https://github.com/merbridge/mer...   | ● 正常  | 启用  | 2024-05-13 18:12 |
| test-02        | GIT  | http://jvc.c                          | ● 同步中 | 启用  | 2024-05-13 18:06 |
| test-01        | GIT  | http://ica.c                          | ● 同步中 | 启用  | 2024-05-13 15:27 |

在数据集列表页面, 创建需要的数据资源, 这些资源包含了 ChatGLM3 代码, 也可以是数据文件, 所有这些数据都可以通过数据集列表来统一管理。

## 代码及模型文件

[ChatGLM3](#) 是[智谱 AI](#) 和清华大学 KEG 实验室联合发布的对话预训练模型。

先拉取 ChatGLM3 代码仓库, 下载预训练模型, 用于后续的微调任务。



DCE 5.0 AI Lab 会在后台进行全自动数据预热，以便后续的任务能够快速访问数据。

## AdvertiseGen 数据集

国内数据可以从 [Tsinghua Cloud](#) 直接获取，这里使用 HTTP 的数据源方式。

注意创建完成后，需要等待数据集预热完成，一般很快，根据您的网络情况而定。

## 微调输出数据

同时，您需要准备一个空的数据集，用于存放微调任务完成后输出的模型文件，这里创建一个空的数据集，以 PVC 为例。

### Warning

注意需要使用支持 `ReadWriteMany` 的存储类型，以便后续的任务能够快速访问数据。

## 环境准备

对于模型开发者来说，准备模型开发需要的 `Python` 环境依赖是非常重要的，传统做法将环境依赖直接打包到开发工具的镜像中，或者直接在本地环境中安

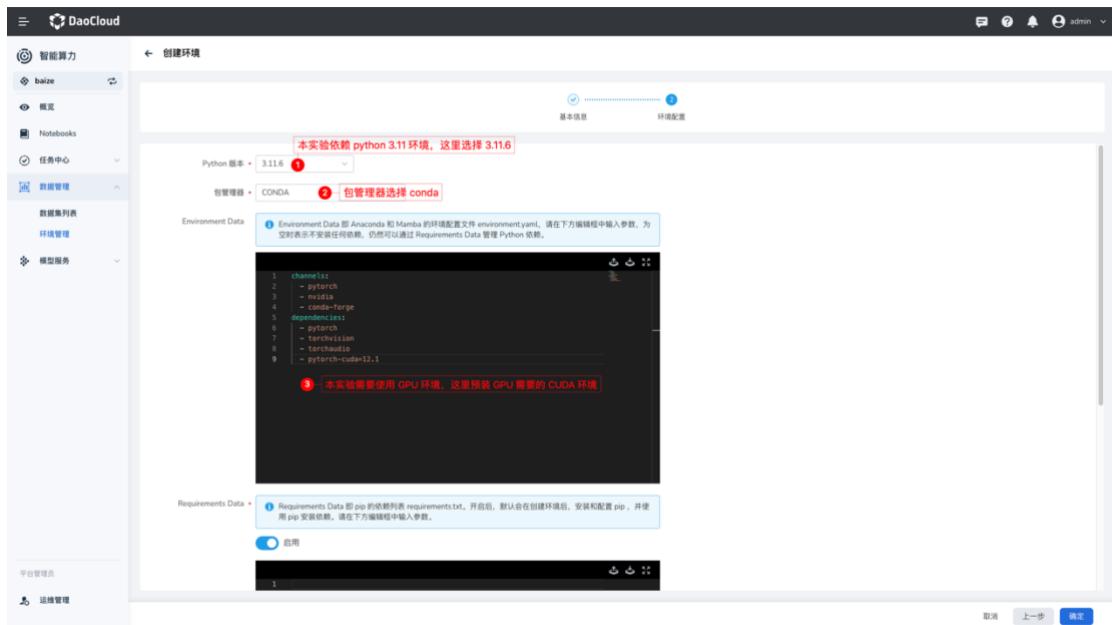
装，但是这样做会导致环境依赖的不一致，而且不利于环境的管理和依赖更新及同步。

DCE 5.0 AI Lab 提供了环境管理的能力，将 Python 环境依赖包管理和开发工具、任务镜像等进行解耦，解决了依赖管理混乱，环境不一致等问题。

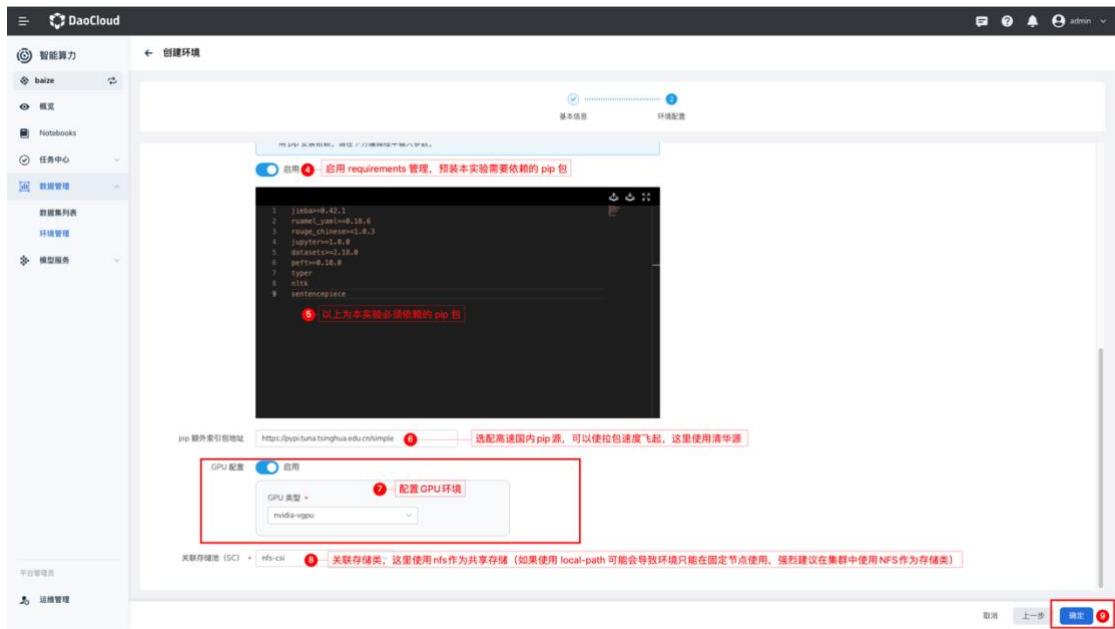
这里使用 DCE 5.0 AI Lab 提供的环境管理功能，创建 ChatGLM3 微调所需的环境，以备后续使用。

## Warning

1. ChatGLM 仓库内有 requirements.txt 文件，里面包含了 ChatGLM3 微调所需的环境依赖
2. 本次微调没有用到 deepspeed 和 mpi4py 包，建议从 requirements.txt 文件中将其注释掉，否则可能出现包编译不通过的情况



在环境管理列表，您可以快速创建一个 Python 环境，并通过简单的表单配置来完成环境的创建；这里需要一个 Python 3.11.x 环境，



因为本实验需要使用 CUDA，所以在这里需要配置 GPU 资源，用于预热需要资源的依赖库。

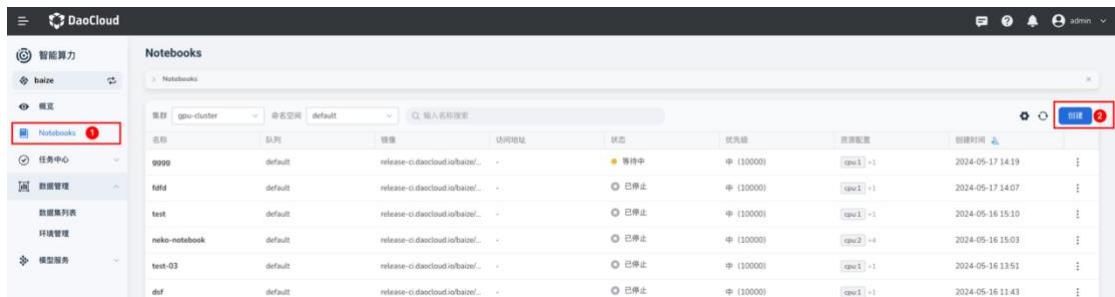
创建环境，需要去下载一系列的 Python 依赖，根据您的实际位置不同，可能会有不同的下载速度，这里使用了国内的镜像加速，可以加快下载速度。

## 使用 Notebook 作为 IDE

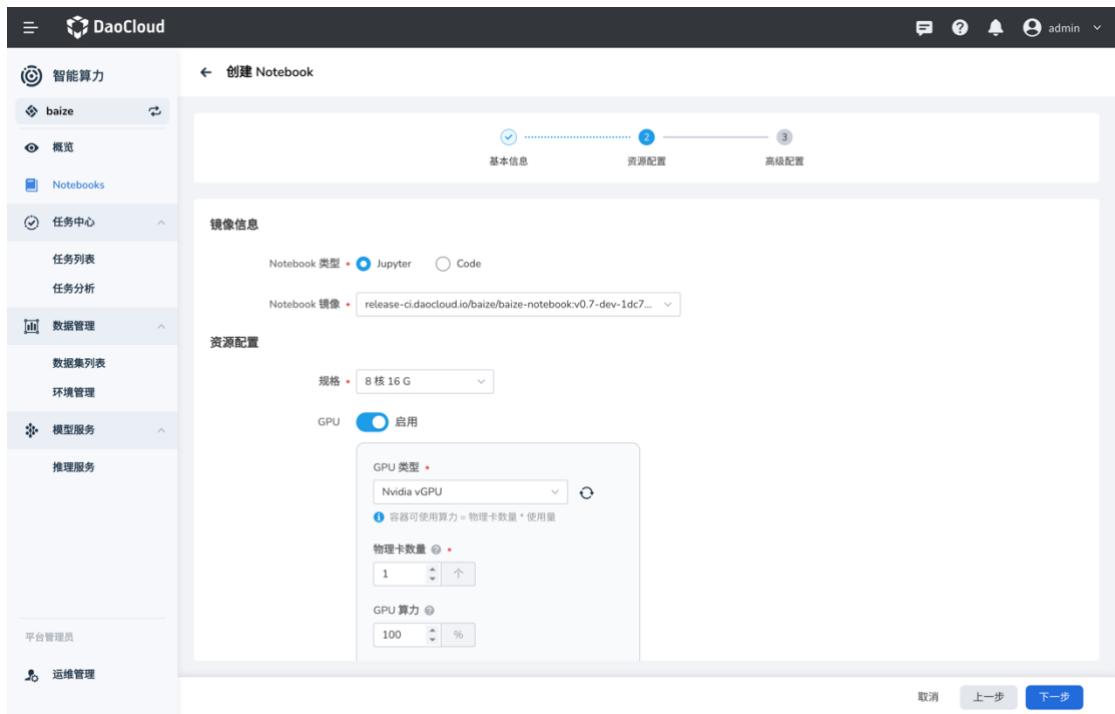
DCE 5.0 AI Lab 提供了 Notebook 作为 IDE 的功能，可以让用户在浏览器中直接编写代码，运行代码，查看代码运行结果，非常适合于数据分析、机器学习、深度学习等领域的开发。

您可以使用 AI Lab 提供的 JupyterLab Notebook 来进行 ChatGLM3 的微调任务。

## 创建 JupyterLab Notebook



在 Notebook 列表中，可以根据页面操作指引，创建一个 Notebook。注意您需要根据前文提到的资源要求来配置对应的 Notebook 资源参数，避免后续因为资源问题，影响微调过程。

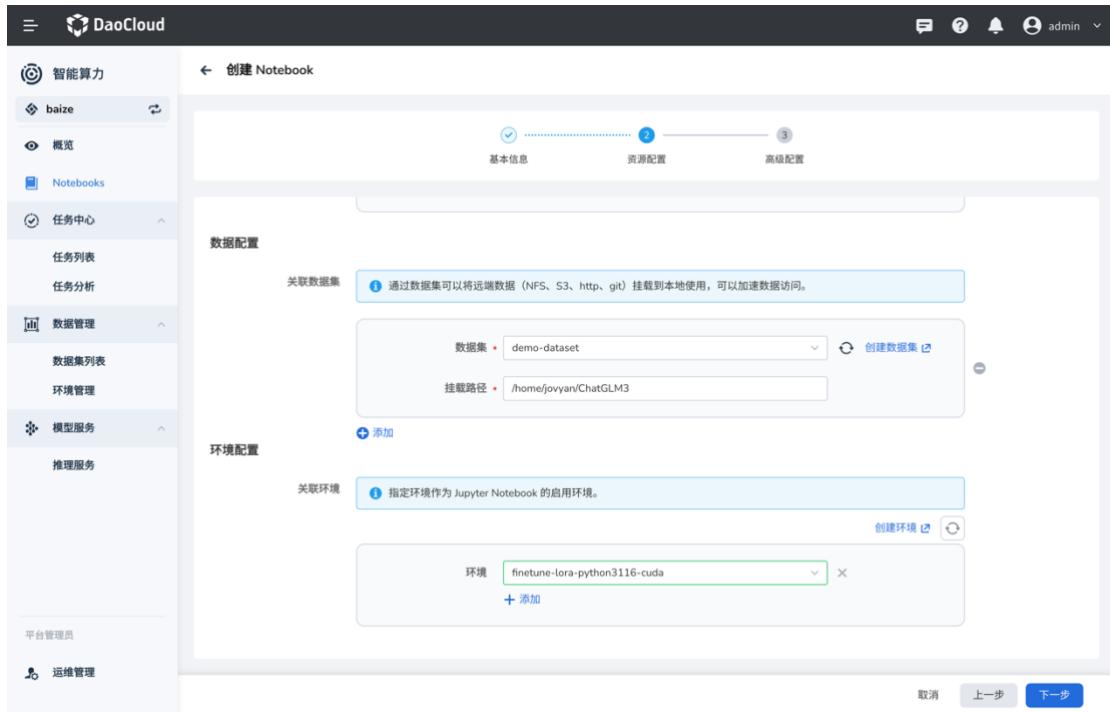


## Note

在创建 Notebook 时，可以将之前预加载的模型代码数据集和环境，直接挂载到 Notebook 中，极大节省了数据准备的时间。

## 挂载数据集和代码

注意：ChatGLM3 的代码文件挂载到了 /home/jovyan/ChatGLM3 目录下，同时您也需要将 AdvertiseGen 数据集挂载到 /home/jovyan/ChatGLM3/finetune\_demo/data/AdvertiseGen 目录下，以便后续的微调任务能够访问数据。



## 挂载 PVC 到模型输出文件夹

本次使用的模型输出位置在 `/home/jovyan/ChatGLM3/finetune_demo/output` 目录下，可以将之前创建的 PVC 数据集挂载到这个目录下，这样训练输出的模型就可以保存到数据集中，后续模型推理等任务可以直接访问。

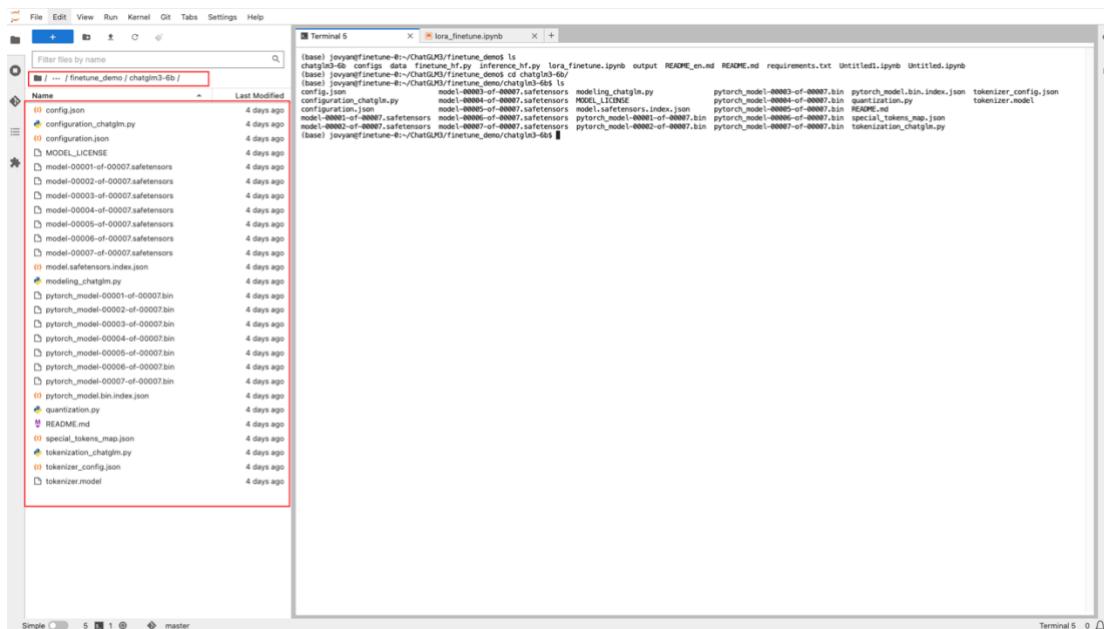
创建完成后，可以看到 Notebook 的界面，您可以直接在 Notebook 中编写代码，运行代码，查看代码运行结果。

| 名称            | 队列      | 镜像                               | 访问地址                                        | 状态    | 优先级        | 资源配置    | 创建时间             | 操作 |
|---------------|---------|----------------------------------|---------------------------------------------|-------|------------|---------|------------------|----|
| testssss      | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 高 (100000) | cpu2 +1 | 2024-05-19 12:35 | ⋮  |
| 9999          | default | release-ci.daocloud.io/baize/... | -                                           | ● 等待中 | 中 (10000)  | cpu1 +1 | 2024-05-17 14:19 | ⋮  |
| ffffd         | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +1 | 2024-05-17 14:07 | ⋮  |
| test          | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +1 | 2024-05-16 15:10 | ⋮  |
| neko-notebook | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu2 +4 | 2024-05-16 15:03 | ⋮  |
| test-03       | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +1 | 2024-05-16 13:51 | ⋮  |
| df            | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +1 | 2024-05-16 11:43 | ⋮  |
| finetune      | default | release-ci.daocloud.io/baize/... | <a href="#">/baize/proj1/finetunegpu...</a> | ● 运行中 | 高 (100000) | cpu8 +4 | 2024-05-16 17:50 | ⋮  |
| ffffdf        | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +3 | 2024-05-15 14:27 | ⋮  |
| df            | default | release-ci.daocloud.io/baize/... | -                                           | ● 已停止 | 中 (10000)  | cpu1 +1 | 2024-05-14 17:31 | ⋮  |

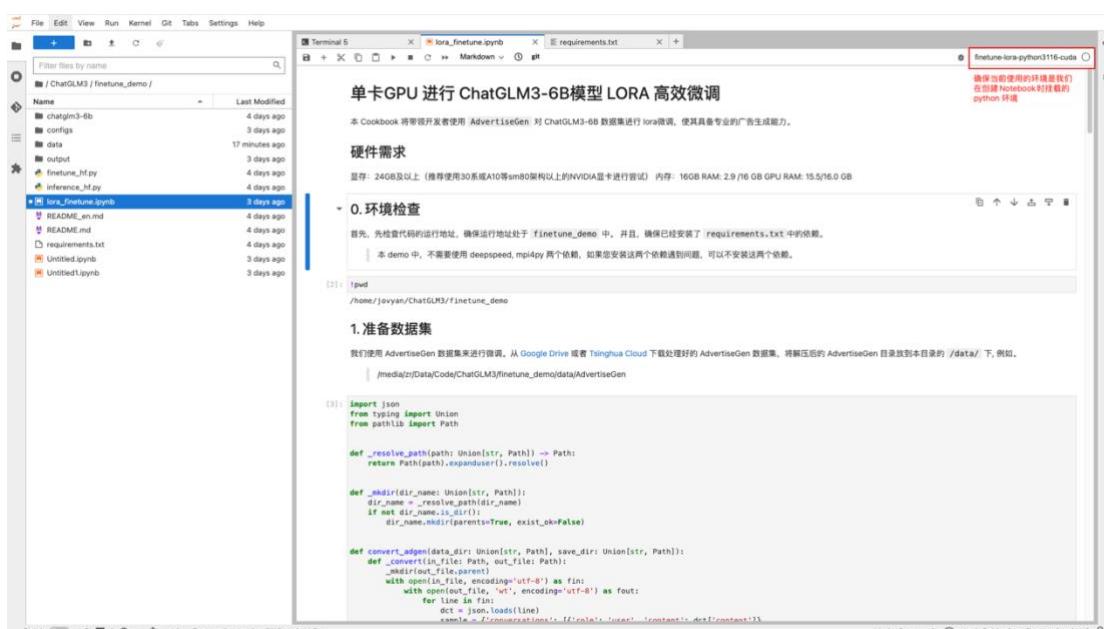
# 微调 ChatGLM3

当您进入到 Notebook 中后，可以在 Notebook 侧边栏会发现有一个 File Browser 的选项，可以看到之前挂载的数据集和代码，在这里找到 ChatGLM3 的文件夹。

您可以看到 ChatGLM3 的微调代码在 finetune\_demo 文件夹中，这里可以直接打开 lora\_finetune.ipynb 文件，这是 ChatGLM3 的微调代码。



首先，根据 README.md 的说明，您可以了解到整个微调的过程，建议先阅读一遍，确保基础的环境依赖和数据准备工作都已经完成。



打开终端，并使用 conda 切换到您提前预热的环境中，此环境与 JupyterLab Kernel 保持一致，以便后续的代码运行。



## 数据预处理

首先，您需要将 AdvertiseGen 数据集进行预处理，对数据进行标准化处理，使其符合 Lora 预训练的标准格式要求；这里将处理后的数据保存到 AdvertiseGen\_fix 文件夹中。

```
import json
from typing import Union
from pathlib import Path

def _resolve_path(path: Union[str, Path]) -> Path:
    return Path(path).expanduser().resolve()

def _mkdir(dir_name: Union[str, Path]):
    dir_name = _resolve_path(dir_name)
    if not dir_name.is_dir():
        dir_name.mkdir(parents=True, exist_ok=False)

def convert_adgen(data_dir: Union[str, Path], save_dir: Union[str, Path]):
    def _convert(in_file: Path, out_file: Path):
        _mkdir(out_file.parent)
        with open(in_file, encoding='utf-8') as fin:
            with open(out_file, 'wt', encoding='utf-8') as fout:
                for line in fin:
```

```

dct = json.loads(line)
sample = {'conversations': [{'role': 'user', 'content': dct['content']},
                           {'role': 'assistant', 'content':
dct['summary']}]}
fout.write(json.dumps(sample, ensure_ascii=False) + '\n')

data_dir = _resolve_path(data_dir)
save_dir = _resolve_path(save_dir)

train_file = data_dir / 'train.json'
if train_file.is_file():
    out_file = save_dir / train_file.relative_to(data_dir)
    _convert(train_file, out_file)

dev_file = data_dir / 'dev.json'
if dev_file.is_file():
    out_file = save_dir / dev_file.relative_to(data_dir)
    _convert(dev_file, out_file)

convert_adgen('data/AdvertiseGen', 'data/AdvertiseGen_fix')

```

为了节省调试的时间，您可以将 `/home/jovyan/ChatGLM3/finetune_demo/data/AdvertiseGen_fix/dev.json` 中的数据量缩减到 50 条，这里的数据是 JSON 格式，处理起来也比较方便的。

The screenshot shows a Jupyter Notebook interface. On the left, there's a file browser window titled 'finetune\_demo / data' showing two files: 'AdvertiseGen' and 'AdvertiseGen\_fix'. A red box highlights the 'AdvertiseGen\_fix' file. On the right, there's a terminal window titled 'Terminal 6' and a code editor window titled 'lora\_finetune.pyrb'. The code editor contains the Python script provided above. A blue box highlights the first few lines of the script.

```

1. 准备数据集
我们使用 AdvertiseGen 数据集来进行微调。从 Google Drive 或者 Tsinghua Cloud 下载处理好的 AdvertiseGen 数据集，将解压后的 AdvertiseGen 目录放到本目录的 ./data/ 下，例如。
./media2/Data/Code/ChatGLM3/finetune_demo/data/AdvertiseGen

import json
from typing import Union
from pathlib import Path

def _resolve_path(path: Union[str, Path]) -> Path:
    return Path(path).expanduser().resolve()

def _mkdir(dir_name: Union[str, Path]):
    dir_name = _resolve_path(dir_name)
    if not dir_name.is_dir():
        dir_name.mkdir(parents=True, exist_ok=False)

def convert_adgen(data_dir: Union[Path, str], save_dir: Union[Path, str]):
    def _convert(in_file: Path, out_file: Path):
        _mkdir(out_file.parent)
        with open(in_file, encoding='utf-8') as fin:
            with open(out_file, 'w', encoding='utf-8') as fout:
                for line in fin:
                    dct = json.loads(line)
                    sample = {'conversations': [
                        {'role': 'user', 'content': dct['content']},
                        {'role': 'assistant', 'content': dct['summary']}]}
                    fout.write(json.dumps(sample, ensure_ascii=False) + '\n')

    data_dir = _resolve_path(data_dir)
    save_dir = _resolve_path(save_dir)

    train_file = data_dir / 'train.json'
    if train_file.is_file():
        out_file = save_dir / train_file.relative_to(data_dir)
        _convert(train_file, out_file)

    dev_file = data_dir / 'dev.json'
    if dev_file.is_file():
        out_file = save_dir / dev_file.relative_to(data_dir)
        _convert(dev_file, out_file)

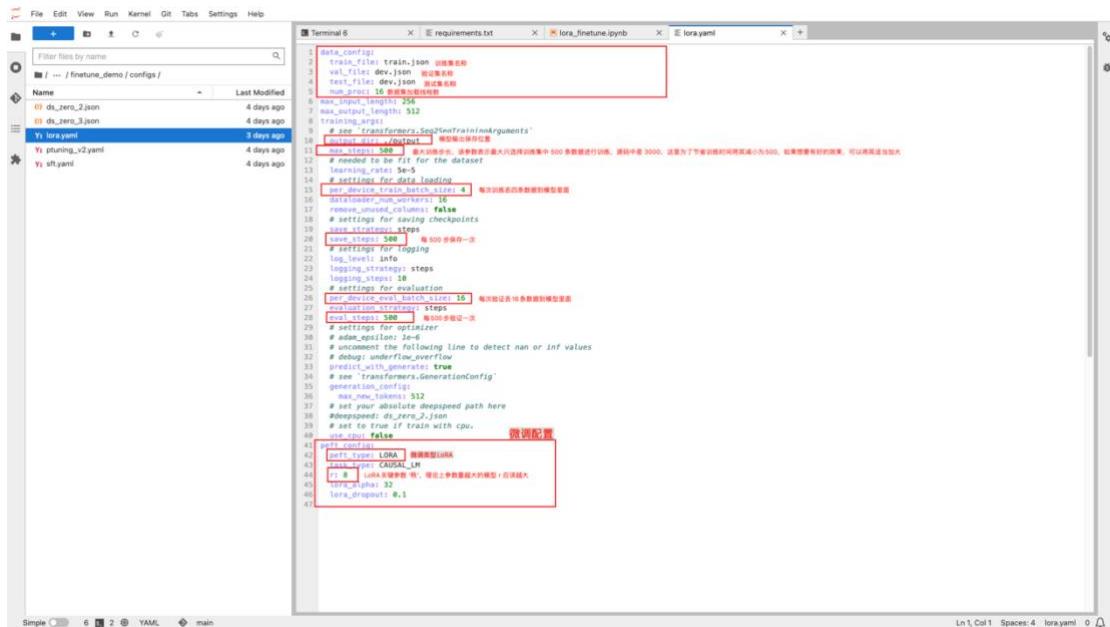
convert_adgen('data/AdvertiseGen', 'data/AdvertiseGen_fix')

```

2. 使用命令行开始微调, 我们使用 lora 进行微调

## 本地 LoRA 微调测试

完成数据的预处理之后，基本上您就可以直接微调测试了，可以在 `/home/jovyan/ChatGLM3/finetune_demo/configs/lora.yaml` 文件中配置微调的参数，一般需要关注的参数基本如下：



```
train_file: train.json
dev_file: dev.json
test_file: dev.json
max_input_length: 256
max_output_length: 512
# see "transformers.Seq2SeqTrainingArguments"
# need to be fit for the dataset
learning_rate: 5e-5
# settings for data loading
data_loader_num_workers: 16
remove_unused_columns: False
save_steps: 1000
# save steps: 500 每 500 步保存一次
# save_strategy: steps
# logging
logging_steps: 10
logging_strategy: steps
# per_device_eval_batch_size: 16 每次验证 16 条数据到模型里面
per_device_train_batch_size: 16
# save_strategy: steps
# settings for optimizer
# adam_epsilon: 1e-6
# debug: underflow_overflow
predict_with_generate: true
# see "transformers.GenerationConfig"
generate_max_length: 512
max_new_tokens: 512
# set your absolute designed path here
# set your absolute designed path here
# set to True if train with cpu.
use_tpu: false
# lora
lora_type: LORA
lora_alpha: 16
lora_beta: 16
lora_rank: 16
lora_dropout: 0.1
```

新开一个终端窗口，使用如下命令即可进行本地微调测试，请确保参数配置和路径正确：

```
!CUDA_VISIBLE_DEVICES=0 NCCL_P2P_DISABLE="1" NCCL_IB_DISABLE="1" python
finetune_hf.py data/AdvertiseGen_fix ./chatglm3-6b configs/lora.yaml
```

在这条命令中，

- `finetune_hf.py` 是 ChatGLM3 代码中的微调脚本
- `data/AdvertiseGen_fix` 是您预处理后的数据集
- `./chatglm3-6b` 是您预训练模型的路径
- `configs/lora.yaml` 是微调的配置文件

```

2. 使用命令行开始微调,我们使用lora进行微调
接着,我们只需要将配置好的参数以命令行的形式作参数程序,就可以使用命令行进行高效微调。
[8]: [CUDA_VISIBLE_DEVICES=0 NCCL_P2P_DISABLE=1] python finetune_hf.py data/AdvertiseGen_file ./chatglm3-6b configs/lora.yaml
[HMMI-core Msg[15162:148147194547972:libvgpu.c:836]]: Initializing.....
[HMMI-core Warn[15162:148147194547972:libvgpu.c:851]]: recursive diag:omp_start_tool
[HMMI-core Warn[15162:148147194547972:libvgpu.c:194]]: get default cuda from 0
[HMMI-core Warn[15162:148147194547972:libvgpu.c:851]]: initialized
Setting eos_token is not supported, use the default one.
Setting pad_token is not supported, use the default one.
Setting max_length is not supported, use the default one.
Loading checkpoint shards: 100% [██████████] 7/7 [00:11:00:00, 1.58s/it]
trainable params: 1,949,696 || all params: 6,245,533,696 || trainable%: 0.031217444255383614
--> Model
--> model has 1,949,696 params

Setting num_proc from 16 back to 1 for the train split to disable multiprocessing as it only contains one shard.
Generating train split: 114599 examples [00:00, 172572.43 examples/s]
Setting num_proc from 16 back to 1 for the validation split to disable multiprocessing as it only contains one shard.
Generating validation split: 59 examples [00:00, 2696.78 examples/s]
Setting num_proc from 16 back to 1 for the test split to disable multiprocessing as it only contains one shard.
Generating test split: 58 examples [00:00, 2696.78 examples/s]
Map (num_rows: 114599) 114599/114599 [00:05:00:00, 22109.86 examples/s]
train_dataset: Dataset({
    features: ['input_ids', 'labels'],
    num_rows: 114599
})
Map (num_proc=16): 100% [██████████] 58/58 [00:00:00:00, 63.81 examples/s]
val_dataset: Dataset({
    features: ['input_ids', 'output_ids'],
    num_rows: 58
})
Map (num_proc=16): 100% [██████████] 58/58 [00:00:00:00, 66.53 examples/s]
test_dataset: Dataset({
    features: ['input_ids', 'output_ids'],
    num_rows: 58
})
--> Sanity check
'@MSK': 64798 ~ -100
'@': 64792 ~ -100
'<user>': 64795 ~ -100
'.' : 38918 ~ -100
'\\': 38919 ~ -100
'!': 38910 ~ -100
'@#': 33467 ~ -100
'@#': 38918 ~ -100
'@#': 55899 ~ -100
'@#': 38919 ~ -100
'@#': 55898 ~ -100
'@#': 55899 ~ -100

```

微调过程中可以使用 nvidia-smi 命令查看 GPU 显存使用情况:

| GPU Name             | Persistence-M | Bus-Id               | Disp.A | Volatile Uncorr. | ECC | GPU-H11 | Compute M | MIS |
|----------------------|---------------|----------------------|--------|------------------|-----|---------|-----------|-----|
| NVIDIA A100 8GB PCIe | On            | 00000000:1C:00.0 Off |        |                  |     |         |           |     |
| N/A 59C P0           | 19m / 38m     | 19680MIB / 38720MIB  | 88%    | Default          | 0   |         |           |     |

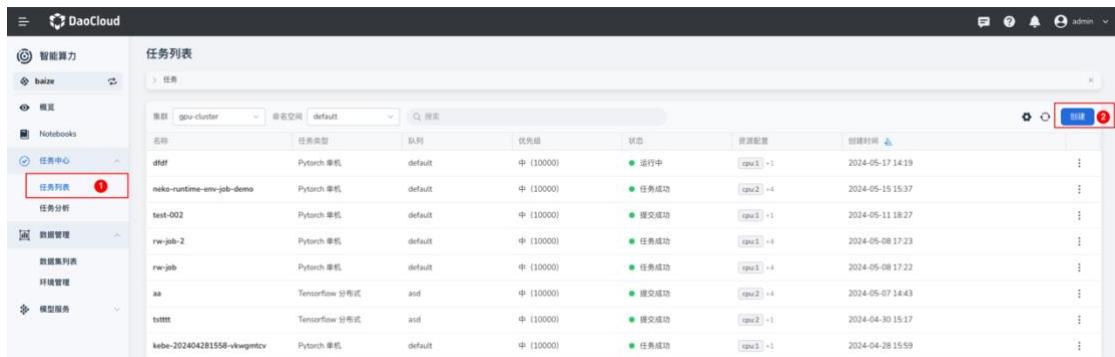
在微调完成后，在 finetune\_demo 目录下会生成一个 output 目录，里面包含了微调的模型文件，这样微调的模型文件就直接保存到您之前创建的 PVC 数据集中了。

## 微调任务提交

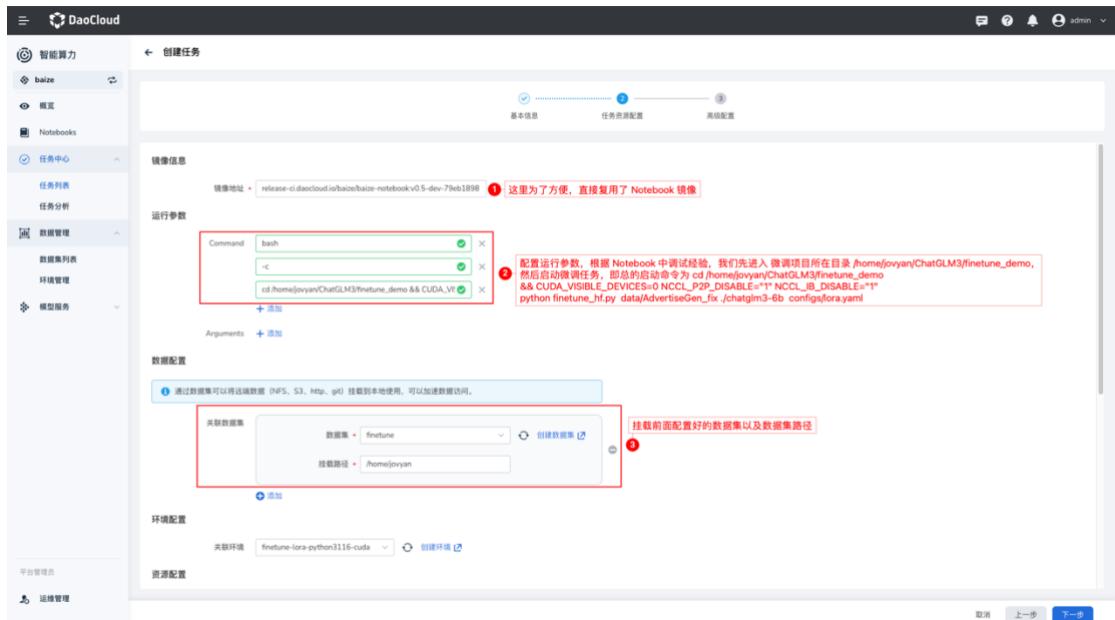
在本地微调测试完成后，确保您的代码和数据没有问题，接下来可以将微调任务提交到 AI Lab 中，进行大规模的训练和微调任务。

这也是推荐的模型开发和微调流程，先在本地进行微调测试，确保代码和数据没有问题。

## 使用界面提交微调任务



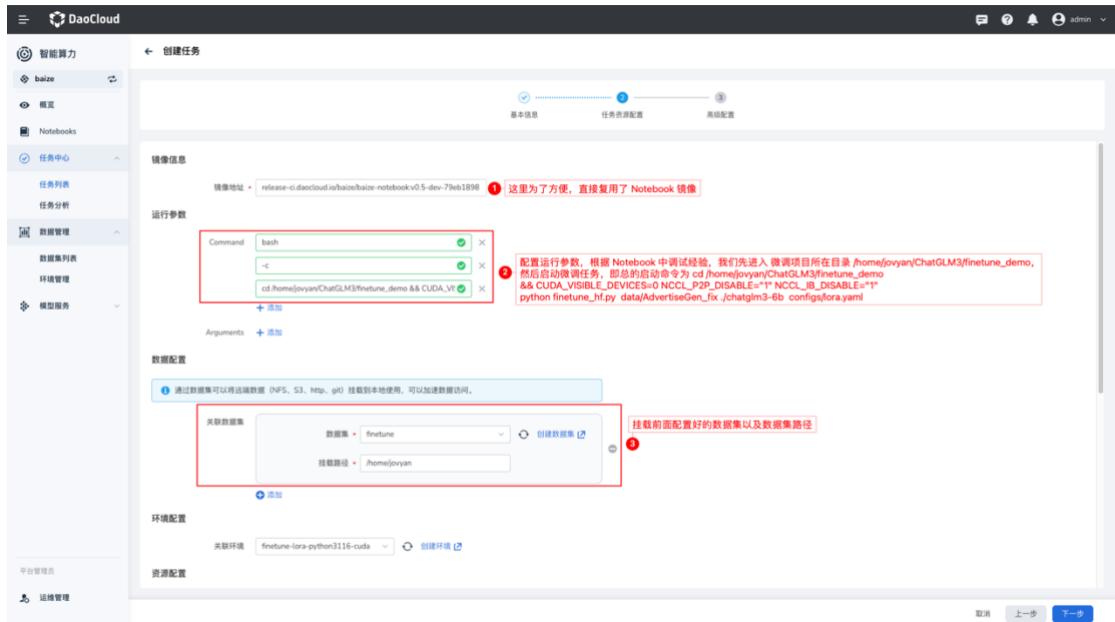
这里使用 Pytorch 来创建微调任务，根据您的实际情况，选择需要使用哪个集群的资源，注意需要满足前面资源准备中提及的资源要求。



- 镜像：可直接使用 `baizectl` 提供的模型镜像
- 启动命令，根据您在 Notebook 中使用 LoRA 微调的经验，代码文件和数据在 `/home/jovyan/ChatGLM3/finetune_demo` 目录下，所以您可以直接使用这个路径：

```
• bash -c "cd /home/jovyan/ChatGLM3/finetune_demo &&
CUDA_VISIBLE_DEVICES=0 NCCL_P2P_DISABLE=1
NCCL_IB_DISABLE=1" python finetune_hf.py
data/AdvertiseGen_fix ./chatglm3-6b configs/lora.yaml"
```

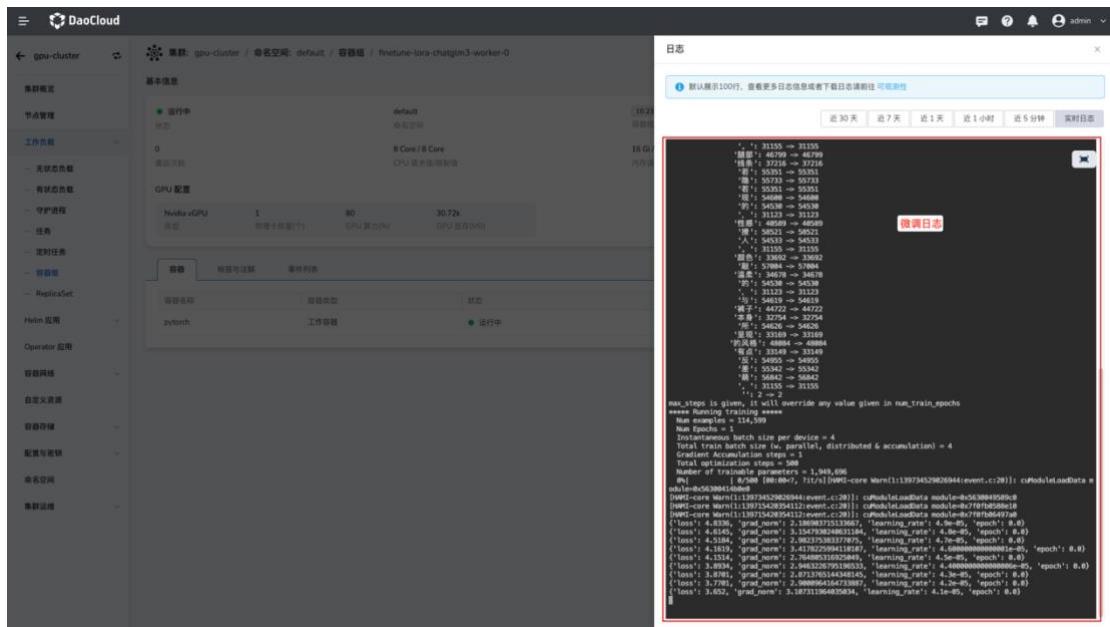
- 挂载环境，这样之前预加载的环境依赖不仅可以在 Notebook 中使用，同时也可以在任务中使用
- 数据集：直接使用之前预热的数据集
  - 将模型输出路径设置为之前创建的 PVC 数据集
  - 将 AdvertiseGen 数据集挂载到 /home/jovyan/ChatGLM3/finetune\_demo/data/AdvertiseGen 目录下
- 配置足够的 GPU 资源，确保微调任务能够正常运行



## 查看任务状态

任务成功提交后，您可以在任务列表中实时查看任务的训练进展，这里您可以看到任务的状态、资源使用情况、日志等信息。

## 查看任务日志



任务运行完成后，您可以在数据输出的数据集中查看微调的模型文件，这样就可以使用这个模型文件进行后续的推理任务。

## 使用 baizectl 提交任务

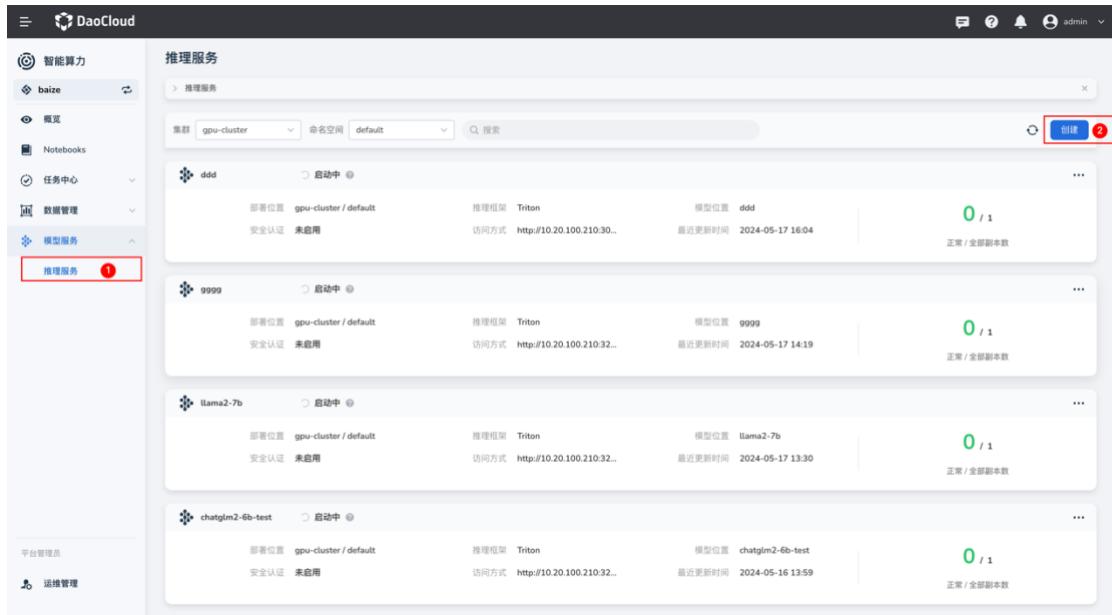
DCE 5.0 AI Lab 的 Notebook 支持免认证直接使用 baizectl 命令行工具，如果您喜欢使用 CLI，那么可以直接使用 baizectl 提供的命令行工具，提交任务。

```
baizectl job submit --name finetunel-chatglm3 -t PYTORCH \
--image release.daocloud.io/baize/baize-notebook:v0.5.0 \
--priority baize-high-priority \
--resources cpu=8,memroy=16Gi,nvidia.com/gpu=1 \
--workers 1 \
--queue default \
--working-dir /home/jovyan/ChatGLM3 \
--datasets AdvertiseGen:/home/jovyan/ChatGLM3/fine_tune_demo/data/AdvertiseGen \
--datasets output:/home/jovyan/ChatGLM3/fine_tune_demo/output \
--labels job_type=pytorch \
--restart-policy on-failure \
-- bash -c "cd /home/jovyan/ChatGLM3/fine_tune_demo && CUDA_VISIBLE_DEVICES=0 \
NCCL_P2P_DISABLE='1' NCCL_IB_DISABLE='1' python finetune_hf.py \
data/AdvertiseGen_fix ./chatglm3-6b configs/lora.yaml"
```

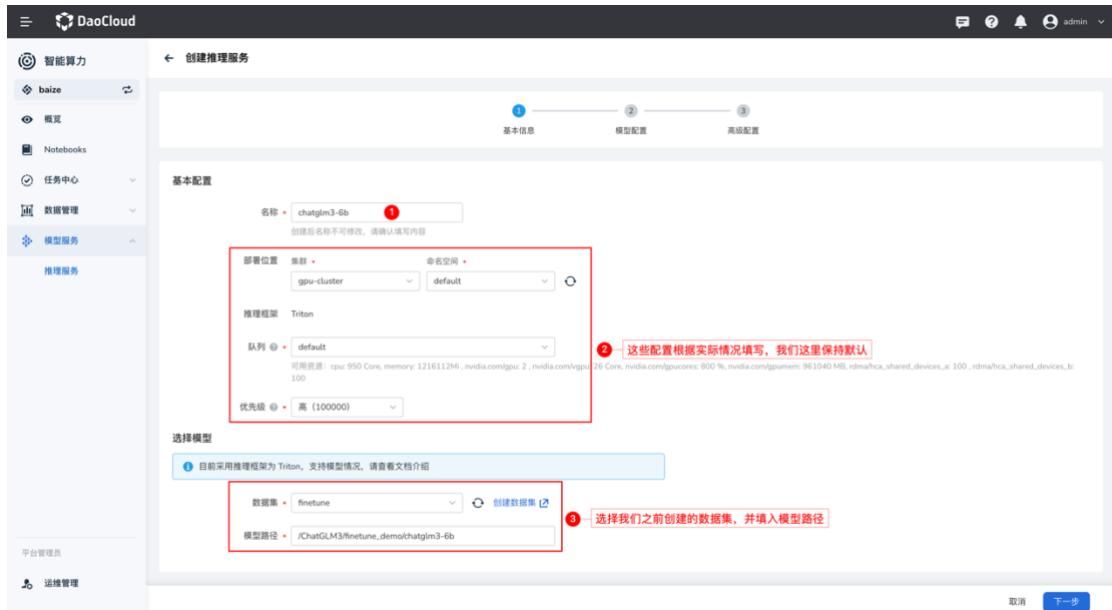
如果希望了解更多 baizectl 的使用说明，可以查看 [baizectl 使用文档](#)。

# 模型推理

在微调任务完成后，您可以使用微调的模型进行推理任务，这里您可以使用 AI Lab 提供的推理服务，将输出后的模型创建为推理服务。



在推理服务列表中，您可以创建一个新的推理服务，在选择模型的位置，选择之前推理输出的数据集，并配置模型路径。

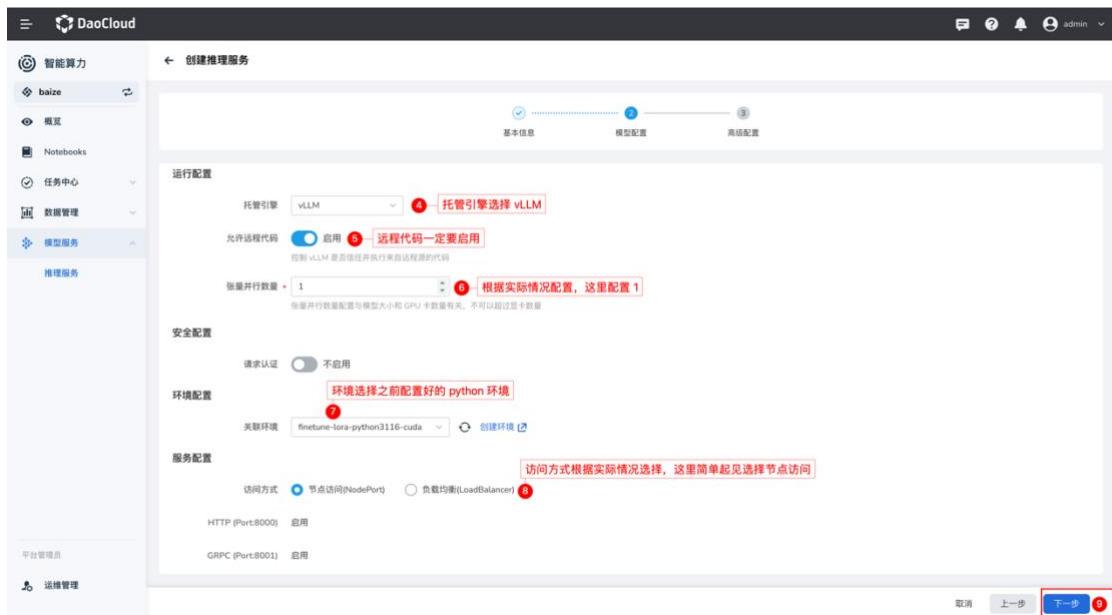


有关模型资源要求、推理服务的 GPU 资源要求，需要根据模型的大小和推理的并发量来配置，这里您可以根据之前微调任务的资源配置来配置。

## 配置模型运行时

配置模型的运行时尤为重要，目前 DCE 5.0 AI Lab 已经支持 vLLM 作为模型推理服务的运行时，可以直接选择 vLLM。

vLLM 支持非常丰富的大语言模型，建议访问 [vLLM](#) 了解更多信息，这些模型都可以很方便地在 AI Lab 中使用。



创建完成后，您可以在推理服务列表中看到您创建的推理服务，在模型服务列表，您可以直接获取模型的访问地址

## 使用模型服务测试

简单在终端中尝试，使用 curl 命令来测试模型服务，这里您可以看到返回的结果，这样就可以使用模型服务进行推理任务了。

```
curl -X POST http://10.20.100.210:31118/v2/models/chatglm3-6b/generate \
-d '{"text_input": "hello", "stream": false, "sampling_parameters": {"temperature": 0.7,
"top_p": 0.95, "max_tokens": 1024} } '
```

```
(base) tangming@MacBook-Pro ~ % curl -X POST http://10.20.100.210:31118/v2/models/chatglm3-6b/generate -d '{"text_input": "hello", "stream": false, "sampling_parameters": {"temperature": 0.7, "top_p": 0.95, "max_tokens": 1024}}'
(base) tangming@MacBook-Pro ~ % curl -X POST http://10.20.100.210:31118/v2/models/chatglm3-6b/generate -d '{"text_input": "hello", "stream": false, "sampling_parameters": {"temperature": 0.7, "top_p": 0.95, "max_tokens": 1024}}'
(base) tangming@MacBook-Pro ~ % curl -X POST http://10.20.100.210:31118/v2/models/chatglm3-6b/generate -d '{"text_input": "你好！我是你的人工智能助手。很高兴为你服务。请问有什么问题我可以帮助解答吗？"}'
("model_name": "chatglm3-6b", "model_version": "1", "text_output": "你好！我是你的人工智能助手。很高兴为你服务。请问有什么问题我可以帮助解答吗？")
(base) tangming@MacBook-Pro ~ % curl -X POST http://10.20.100.210:31118/v2/models/chatglm3-6b/generate -d '{"text_input": "kbs是什么？", "stream": false, "sampling_parameters": {"temperature": 0.7, "top_p": 0.95, "max_tokens": 1024}}'
("model_name": "chatglm3-6b", "model_version": "1", "text_output": "kbs是什么？!KBS是一个开源的容器编排系统，用于自动化应用部署、扩缩容和管理。K8s使用容器技术来包装应用代码和其他配置，使其能够在不同的环境中移动和运行。它已经成为容器编排领域的事实标准，被许多大型企业和组织用于管理和编排其现代化的应用程序。")
(base) tangming@MacBook-Pro ~ % 
```

## 结语

本文以 ChatGLM3 为例，带您快速了解和上手 **AI Lab** 的模型微调，使用 LoRA 微调了 ChatGLM3 模型。

DCE 5.0 AI Lab 提供了非常丰富的功能，可以帮助模型开发者快速进行模型开发、微调、推理等任务，同时也提供了丰富的 OpenAPI 接口，可以方便地与第三方应用生态进行结合。

## 如何提交 DeepSpeed 训练任务

根据 [DeepSpeed 官方文档](#)，我们推荐使用修改代码的方式实现。

即使用 `deepspeed.init_distributed()` 代替 `torch.distributed.init_process_group(...)`。然后运行命令使用 `torchrun`，提交为 Pytorch 分布式任务，既可运行 DeepSpeed 任务。

是的，你可以使用 `torchrun` 运行你的 DeepSpeed 训练脚本。`torchrun` 是 PyTorch 提供的一个实用工具，用于分布式训练。你可以结合 `torchrun` 和 DeepSpeed API 来启动你的训练任务。

以下是一个使用 `torchrun` 运行 DeepSpeed 训练脚本的示例：

1. 编写训练脚本：

```
train.py
import torch
import deepspeed
from torch.utils.data import DataLoader

# 模型和数据加载
model = YourModel()
train_dataset = YourDataset()
train_dataloader = DataLoader(train_dataset, batch_size=32)

# 配置文件路径
deepspeed_config = "deepspeed_config.json"

# 创建 DeepSpeed 训练引擎
model_engine, optimizer, _, _ = deepspeed.initialize(
    model=model,
    model_parameters=model.parameters(),
    config_params=deepspeed_config
```

```
)  
  
    # 训练循环  
    for batch in train_dataloader:  
        loss = model_engine(batch)  
        model_engine.backward(loss)  
        model_engine.step()
```

## 2. 创建 DeepSpeed 配置文件:

```
deepspeed_config.json  
{  
    "train_batch_size": 32,  
    "gradient_accumulation_steps": 1,  
    "fp16": {  
        "enabled": true,  
        "loss_scale": 0  
    },  
    "optimizer": {  
        "type": "Adam",  
        "params": {  
            "lr": 0.00015,  
            "betas": [0.9, 0.999],  
            "eps": 1e-08,  
            "weight_decay": 0  
        }  
    }  
}
```

## 3. 使用 torchrun 或者 baizectl 运行训练脚本:

### 4. torchrun train.py

通过这种方式，你可以结合 PyTorch 的分布式训练功能和 DeepSpeed 的优化技术，从而实现更高效的训练。您可以在 Notebook 中，使用 baizectl 提交命令：

```
baizectl job submit --pytorch --workers 2 -- torchrun train.py
```

## 增加任务调度器

DCE 5.0 AI Lab 提供了任务调度器，可以帮助您更好地管理任务，除了提供基础的调度器之外，目前也支持用户自定义调度器。

# 任务调度器介绍

在 **Kubernetes** 中，任务调度器负责决定将 Pod 分配到哪个节点上运行。它考虑多种因素，如资源需求、硬件/软件约束、亲和性/反亲和性规则、数据局部性等。

默认调度器是 **Kubernetes** 集群中的一个核心组件，负责决定将 Pod 分配到哪个节点上运行。让我们深入了解它的工作原理、特性和配置方法。

## 调度器的工作流程

默认调度器的工作流程可以分为两个主要阶段：过滤（Filtering）和评分（Scoring）。

### 过滤阶段

调度器会遍历所有节点，排除不满足 Pod 要求的节点，考虑的因素包括：

- 资源需求
- 节点选择器
- 节点亲和性
- 污点和容忍

以上参数，我们可以通过创建任务时的高级配置来设置，如下图所示：



## 评分阶段

对通过过滤的节点进行打分，选择得分最高的节点来运行 Pod，考虑因素包括：

- 资源使用率
- Pod 亲和性/反亲和性
- 节点亲和性等。

## 调度器插件

除了基础的一些任务调度能力之外，我们还支持使用 Scheduler Plugins：Kubernetes SIG Scheduling 维护的一组调度器插件，包括 Coscheduling (Gang Scheduling) 等功能。

### 部署调度器插件

在工作集群中部署第二调度器插件，请参考[部署第二调度器插件](#)。

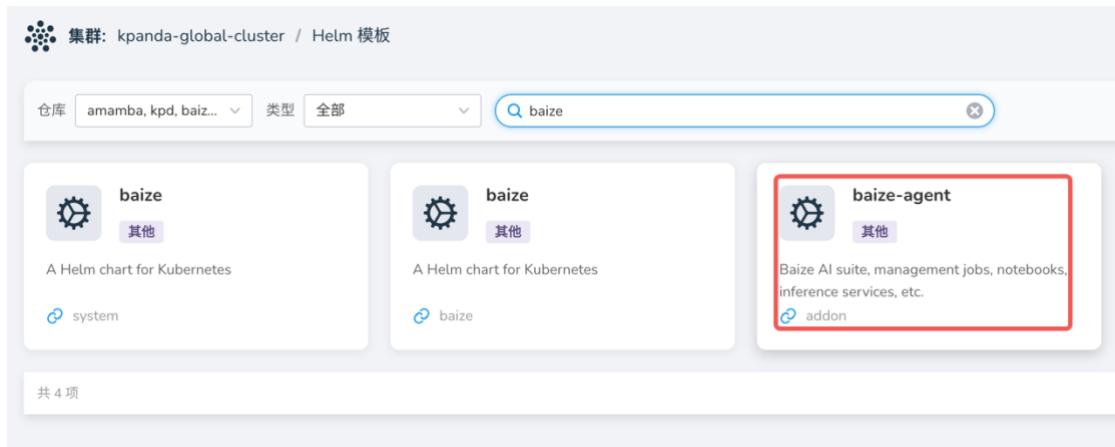
### 在 AI Lab 中启用调度器插件

#### Danger

增加调度器插件若操作不当，可能会影响到整个集群的稳定性，建议在测试环境中进行测试；或者联系我们的技术支持团队。

注意，如果希望在训练任务中使用更多的调度器插件，需要事先手工在工作集群中成功安装，然后在集群中部署 baize-agent 时，增加对应的调度器插件配置。

通过容器管理提供的界面 **Helm** 应用 管理能力，可以方便地在集群中部署调度器插件，如下图所示：



然后，在右上角点击 **安装**，（若已部署了 baize-agent，可以到 Helm 应用列表去更新），根据如下图所示的配置，增加调度器。

```
1 cluster-controller;
2   image:
3     registry: ''
4     repository: baize/baize-cluster-controller
5     tag: v0.6.0-rc0
6   global:
7   <cluster:
8     schedulers:
9       - Coscheduling
10      - Binpack
11     config:
12       cluster_name: kpanda-global-cluster
13       dataset_job_spec: {}
14       inference_config:
15         triton_image: m.daocloud.io/nvcr.io/tritonserver:24.04-py3
16       triton_images_map:
```

注意调度器的参数层级，添加完成后，点击 **确定** 即可。

注意以后在更新 baize-agent 时，不要遗漏这个配置。

## 在创建任务时指定调度器

当您在集群中成功部署了对应的调度器插件，并且在 baize-agent 也正确增加了对应的调度器配置后，可以在创建任务时，指定调度器。

一切正常的情况下，您可以在调度器下拉框中看到您部署的调度器插件。



以上，就是我们在 AI Lab 中，为任务增加调度器选项的配置使用说明。

## 部署 Label Studio

### Note

参阅视频教程：[数据标注和数据集使用说明](#)

[Label Studio](#) 是一个开源的数据标注工具，用于各种机器学习和人工智能任务。以下是 Label Studio 的简要介绍：

- 支持图像、音频、视频、文本等多种数据类型的标注
- 可用于目标检测、图像分类、语音转录、命名实体识别等多种任务
- 提供可定制的标注界面
- 支持多种标注格式和导出选项

Label Studio 通过其灵活性和功能丰富性，为数据科学家和机器学习工程师提供了强大的数据标注解决方案。

# 部署到 DCE 5.0

要想在 AI Lab 中使用 Label Studio，需将其部署到[全局服务集群](#)，你可以通过 Helm 的方式快速部署。

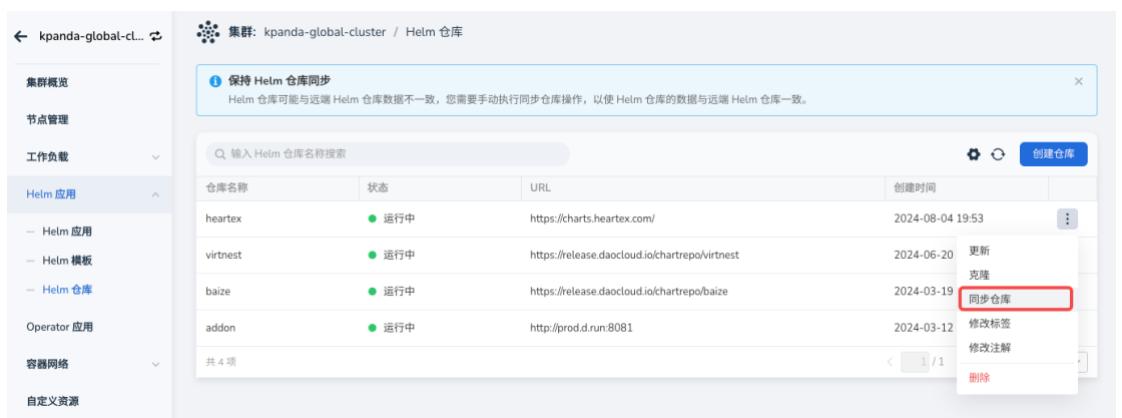
## Note

更多部署详情，请参阅 [Deploy Label Studio on Kubernetes](#)。

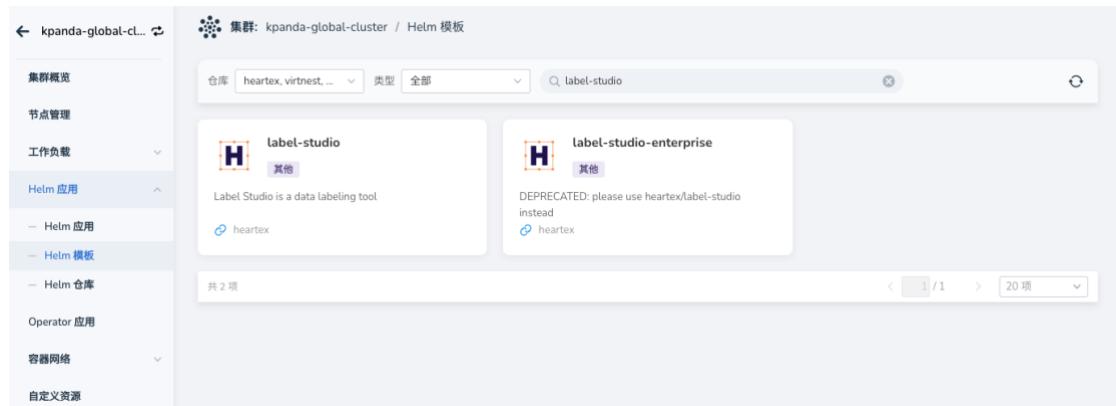
1. 打开全局服务集群界面，从左侧导航栏找到 **Helm 应用 -> Helm 仓库**，选择 **创建仓库** 按钮，填写如下参数：



2. 添加成功后，点击列表右侧的 **!**，选择 **同步仓库**，稍等片刻后完成同步。（后续更新 Label Studio 也会用到这个同步操作）。

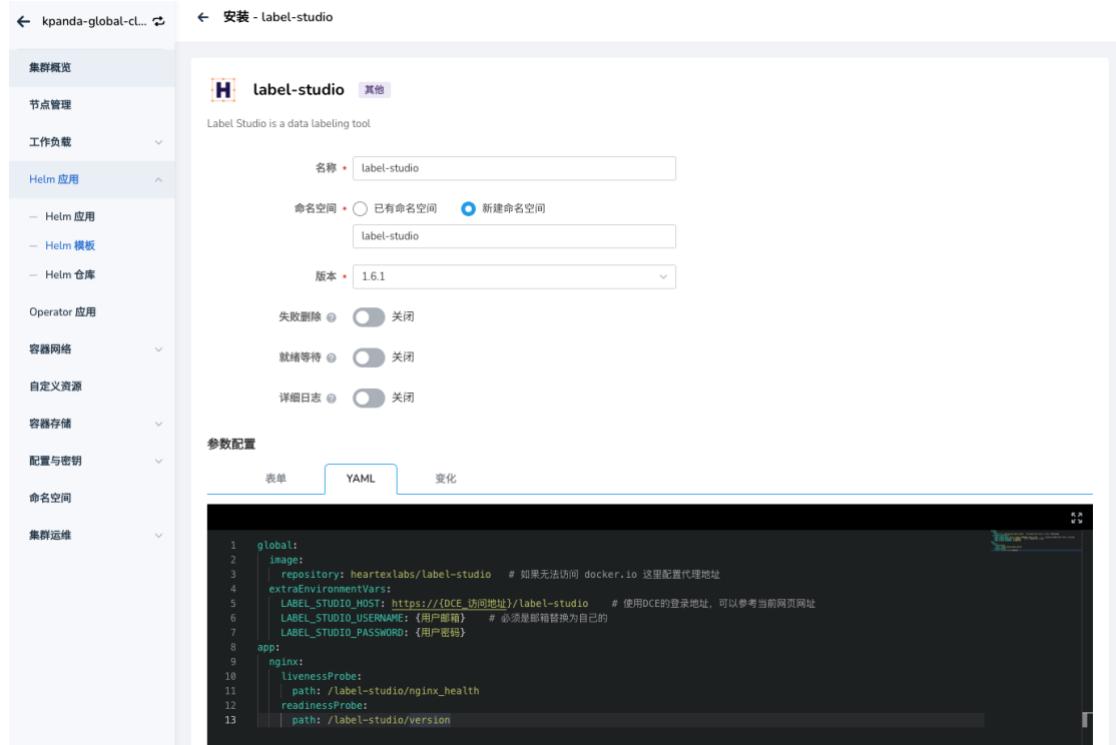


3. 然后跳转到 **Helm 模板** 页面，你可以搜索找到 label-studio，点击卡片。



4. 选择最新的版本，如下图配置安装参数，名称为 label-studio，建议创建新的命令空间，配置参数切换到 YAML，根据说明修改其中配置。

```
global:  
  image:  
    repository: heartexlabs/label-studio    # 如果无法访问 docker.io，在此处配置  
    代理地址  
    extraEnvironmentVars:  
      LABEL_STUDIO_HOST: https://{{DCE_访问地址}}/label-studio    # 使用 DCE  
      5.0 的登录地址，请参阅当前网页 URL  
      LABEL_STUDIO_USERNAME: {用户邮箱}    # 必须是邮箱，替换为自己的  
      LABEL_STUDIO_PASSWORD: {用户密码}  
app:  
  nginx:  
    livenessProbe:  
      path: /label-studio/nginx_health  
    readinessProbe:  
      path: /label-studio/version
```



至此，完成了 Label studio 的安装。

## Warning

默认会安装 PostgreSQL 作为数据服务中间件，如果镜像拉取失败，可能是 docker.io 无法访问，注意切换到可用代理即可。

如果你有自己的 PostgreSQL 数据服务中间件，可以使用如下参数配置：

```

global:
image:
  repository: heartexlabs/label-studio # 如果无法访问 docker.io, 在此处配置代理地址
extraEnvironmentVars:
  LABEL_STUDIO_HOST: https://{{DCE_访问地址}}/label-studio # 使用 DCE 5.0 的登录地址, 参阅当前网页 URL
  LABEL_STUDIO_USERNAME: {{用户邮箱}} # 必须是邮箱, 替换为自己的
  LABEL_STUDIO_PASSWORD: {{用户密码}}
app:
nginx:
  livenessProbe:
    path: /label-studio/nginx_health
  readinessProbe:
    path: /label-studio/version
postgresql:
  enabled: false # 禁用内置的 PostgreSQL
externalPostgresql:

```

```
host: "postgres-postgresql" # PostgreSQL 地址  
port: 5432  
username: "label_studio" # PostgreSQL 用户名  
password: "your_label_studio_password" # PostgreSQL 密码  
database: "label_studio" # PostgreSQL 数据库名
```

## 添加 GProduct 到导航栏

如果要添加 Label Studio 到 DCE 5.0 导航栏，可以参考[全局管理 OEM IN](#) 的方式。以下案例是增加到 AI Lab 二级导航的添加方式。

### 添加代理访问

```
apiVersion: ghippo.io/v1alpha1  
kind: GProductProxy  
metadata:  
  name: label-studio  
spec:  
  gproduct: label-studio  
  proxies:  
    - authnCheck: false  
      destination:  
        host: label-studio-ls-app.label-studio.svc.cluster.local  
        port: 80  
      match:  
        uri:  
          prefix: /label-studio
```

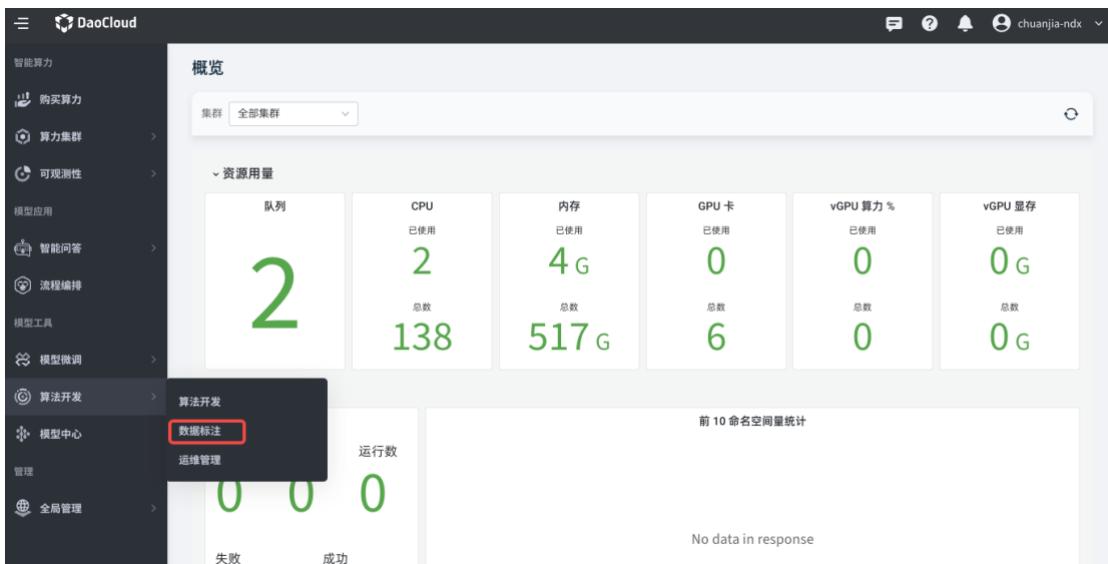
### 添加到 AI Lab

修改 CRD 为 GProductNavigator 的 CR baize，然后在现有配置中进行如下变更：

```
apiVersion: ghippo.io/v1alpha1  
kind: GProductNavigator  
metadata:  
  annotations:  
    meta.helm.sh/release-name: baize  
    meta.helm.sh/release-namespace: baize-system  
  labels:  
    app.kubernetes.io/managed-by: Helm
```

```
gProductName: baize
name: baize
spec:
  category: cloudnativeai
  gproduct: baize
  iconUrl: ./ui/baize/logo.svg
  isCustom: false
  localizedName:
    en-US: AI Lab
    zh-CN: AI Lab
  menus:
    - iconUrl: "
      isCustom: false
      localizedName:
        en-US: AI Lab
        zh-CN: AI Lab
      name: workspace-view
      order: 1
      url: ./baize
      visible: true
    - iconUrl: "
      isCustom: false
      localizedName:
        en-US: Operator
        zh-CN: 运维管理
      name: admin-view
      order: 1
      url: ./baize/admin
      visible: true
    # 添加开始
    - iconUrl: "
      localizedName:
        en-US: Data Annotation
        zh-CN: 数据标注
      name: label-studio
      order: 1
      target: blank    # 控制新开页
      url: https://DCE_访问地址}/label-studio    # 访问地址
      visible: true
    # 添加结束
  name: AI Lab
  order: 10
  url: ./baize
  visible: true
```

## 添加效果



## 结语

以上，就是如何添加 Label Studio 并将其作为 AI Lab 的标注组件，通过将标注后的数据添加到 AI Lab 的数据集中， 联动算法开发，完善算法开发流程，后续如何使用请关注其他文档参考。

## 故障排查

本文将持续统计和梳理 AI Lab 使用过程可能因环境或操作不规范引起的报错，以及在使用过程中遇到某些报错的问题分析、解决方案。

### Warning

本文档仅适用于 DCE 5.0 版本，若遇到 AI Lab 的使用问题，请优先查看此排障手册。

AI Lab 在 DCE 5.0 中模块名称 baize，提供了一站式的模型训练、推理、模型管理等功能。

## 集群下拉列表中找不到集群

## 问题现象

在 **AI Lab** 开发控制台、运维控制台，功能模块的集群搜索条件的下拉列表找不到想要的集群。

## 问题分析

在 **AI Lab** 中，集群下拉列表如果缺少了想要的集群，可能是由于以下原因导致的：

- **baize-agent** 未安装或安装不成功，导致 **AI Lab** 无法获取集群信息
- 安装 **baize-agent** 未配置集群名称，导致 **AI Lab** 无法获取集群信息
- 工作集群内可观测组件异常，导致无法采集集群内的指标信息

## 解决办法

### baize-agent 未安装或安装不成功

**AI Lab** 有一些基础组件需要在每个工作集群内进行安装，如果工作集群内未安装 **baize-agent** 时，可以在界面上选择安装，可能会导致一些非预期的报错等问题。

所以，为了保障使用体验，可选择的集群范围仅包含了已经成功安装了 **baize-agent** 的集群。

如果是因为 **baize-agent** 未安装或安装失败，则使用 **容器管理 -> 集群管理 -> Helm 应用 -> Helm 模板**，找到 **baize-agent** 并安装。

#### Note

此地址快速跳

转 [https://<dce\\_host>/kpanda/clusters/<cluster\\_name>/helm/charts/addon/baize-agent](https://<dce_host>/kpanda/clusters/<cluster_name>/helm/charts/addon/baize-agent)。  
注意将 **<dce\_host>** 替换为实际的 DCE 控制台地址，**<cluster\_name>** 替换为实际的集群名称。

### 安装 **baize-agent** 时未配置集群名称

在安装 **baize-agent** 时，需要注意配置集群的名称，这个名称会用于可观测指标采集，**默认为空，需手工配置**。

The screenshot shows the '安装 - baize-agent' (Install - baize-agent) page. On the left, there's a sidebar with sections like '集群概览', '节点管理', '工作负载' (with sub-options: 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' (with sub-options: 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', and '集群运维'. The main area has tabs for '全局' (Global) and '集群' (Cluster). Under '集群', it says '集群名称' (Cluster Name) is 'baize-agent'. Below that are fields for '命名空间' (Namespace) 'baize-system', '版本' (Version) 'v0.7.0-rc1', and several toggle switches for '失败删除' (Delete on Failure), '就绪等待' (Ready Waiting), and '详细日志' (Detailed Log). A large '参数配置' (Parameter Configuration) section shows a YAML file with code. Two red arrows point to line 10: 'dataset\_job\_spec: {}' and line 11: 'cluster\_name: '''. The line 11 comment '手工配置集群的名称' (Manually configure cluster name) is also highlighted with a red box.

```

global:
  imageRegistry: release.daocloud.io
  prod: baize-agent
  resources: {}
  imagePullPolicy: IfNotPresent
  imagePullSecrets: []
  highAvailable: false
  debug: false
  config:
    datasetJobSpec: {}
    clusterName: '' // 手工配置集群的名称
    inferenceConfig:
      tritonImage: m.daocloud.io/nvcr.io/nvidia/tritonserver:24.04-py3
      tritonImagesMap:
        VLLM: m.daocloud.io/nvcr.io/nvidia/tritonserver:24.04-vllm-python-py3
        vLLM: m.daocloud.io/docker.io/vllm/vllm-openai:v0.5.0
    cluster:
      schedulers: []
    notebookController:
      image:
        registry: ''
        repository: baize/notebook-controller
        tag: v1.8.0
    trainingOperator:
      image:
        registry: ''
        repository: baize/training-operator
        tag: v1-5525468

```

## 工作集群内可观测组件异常

如果集群内可观测组件异常，可能会导致 AI Lab 无法获取集群信息，请检查平台的可观测服务是否正常运行及配置。

- 检查[全局服务集群](#)内 insight-server 组件是否正常运行
- 检查[工作集群](#)内 insight-agent 组件是否正常运行

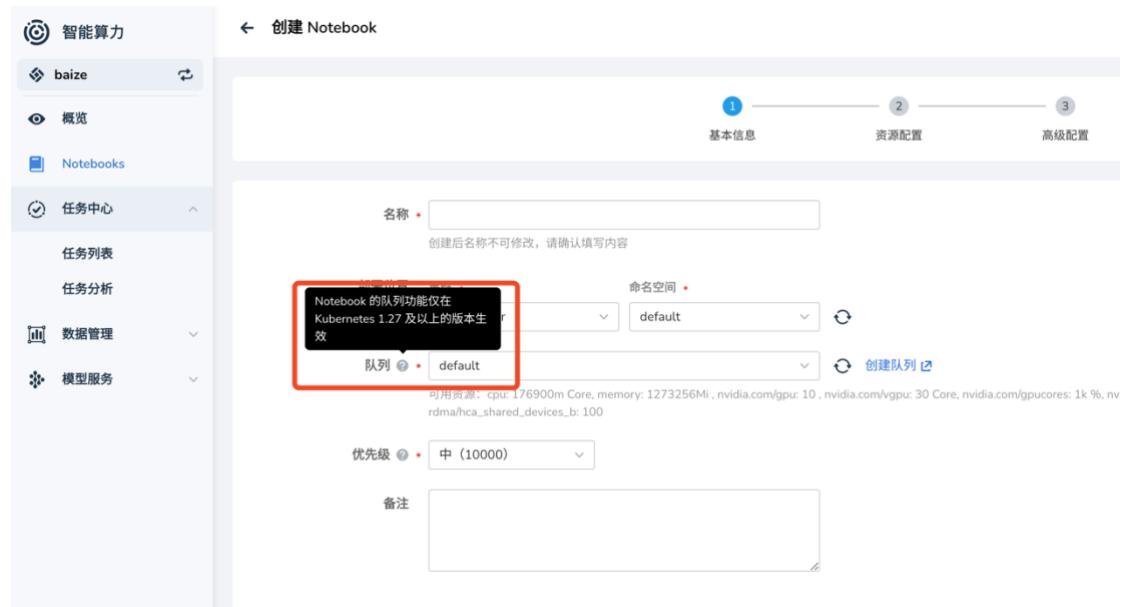
## Notebook 不受队列配额控制

在 AI Lab 中，用户在创建 Notebook 时，发现选择的队列即使资源不足，Notebook 依然可以创建成功。

## 问题 01: Kubernetes 版本不支持

- 分析:

AI Lab 中的队列管理能力由 [Kueue](#) 提供, Notebook 服务是通过 [JupyterHub](#) 提供的。 JupyterHub 对 Kubernetes 的版本要求较高, 对于低于 v1.27 的版本, 即使在 DCE 5.0 中设置了队列配额, 用户在创建 Notebook 时也选择了配额, 但 Notebook 实际也不会受到队列配额的限制。



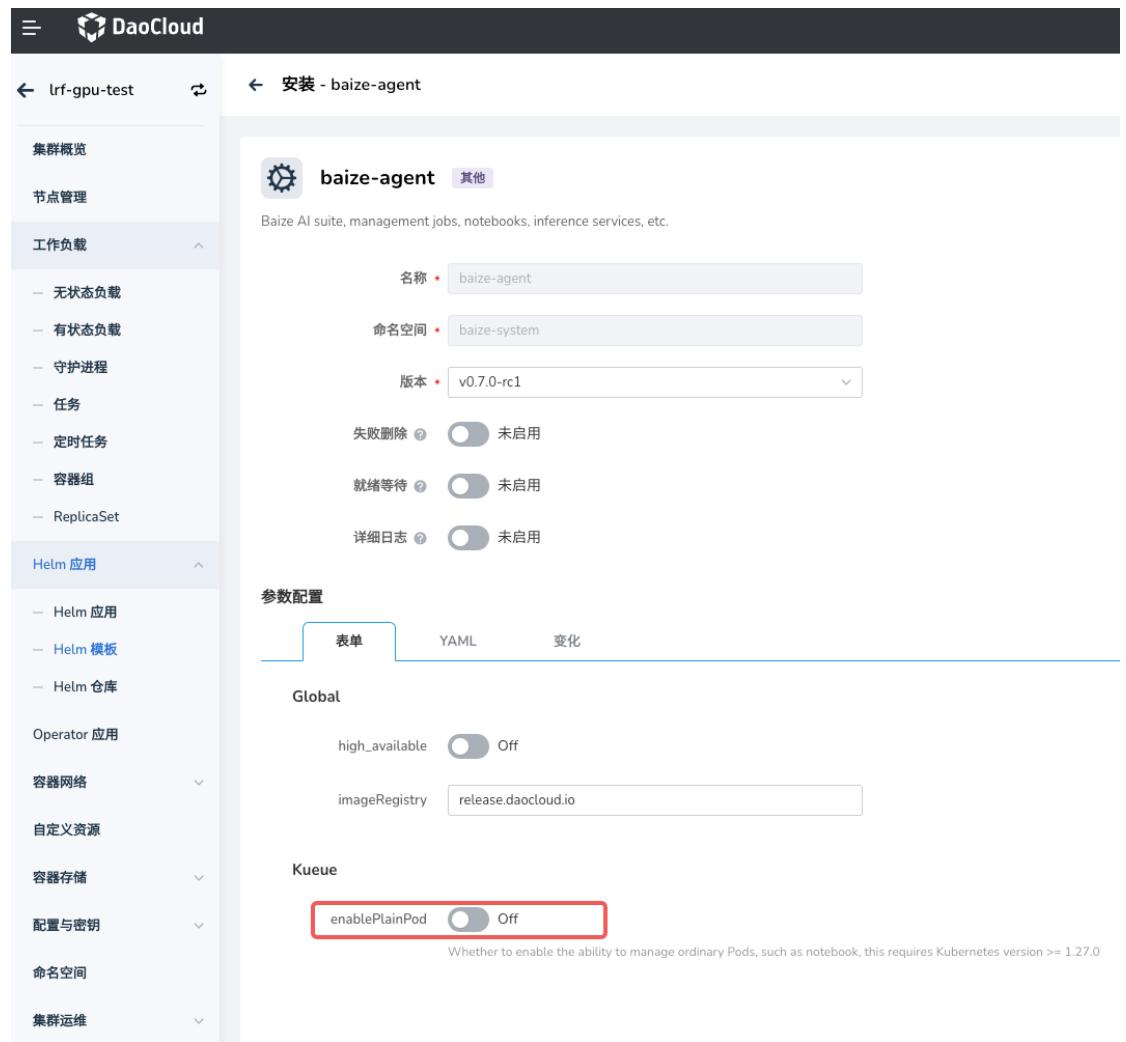
- 解决办法: 提前规划, 生产环境中建议使用 Kubernetes 版本 v1.27 以上。
- 参考资料: [Jupyter Notebook Documentation](#)

## 问题 02: 配置未启用

- 分析:

当 Kubernetes 集群版本 大于 v1.27 时, Notebook 仍无法受到队列配额的限制。

这是因为, Kueue 需要启用对 enablePlainPod 支持, 才会对 Notebook 服务生效。

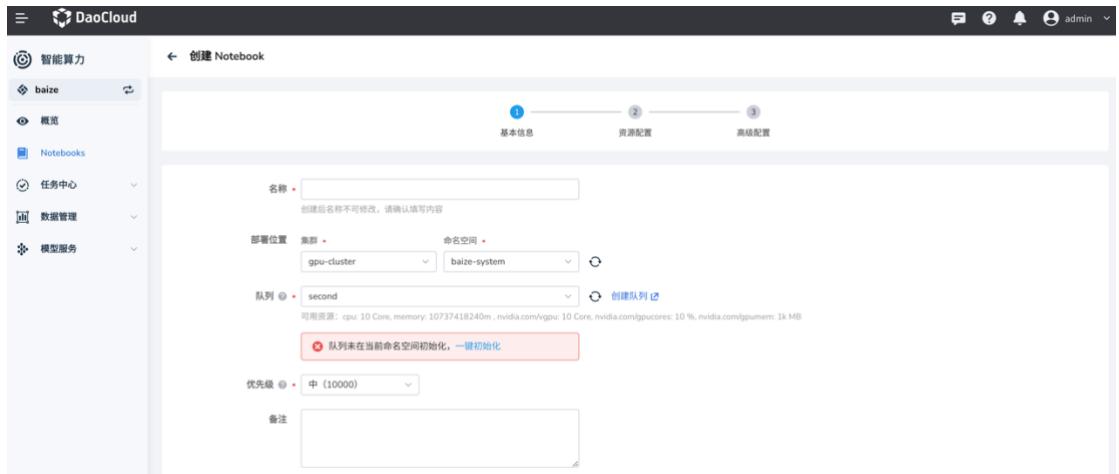


- 解决办法：在工作集群中部署 baize-agent 时，启用 Kueue 对 enablePlainPod 的支持。
- 参考资料：[Run Plain Pods as a Kueue-Managed Job](#)

## 本地队列初始化失败

### 问题现象

在创建 Notebook、训练任务或者推理服务时，当队列是首次在该命名空间使用时，会提示需要一键初始化队列，但是初始化失败。



## 问题分析

在 **AI Lab** 中，队列管理能力由 [Kueue](#) 提供，而 [Kueue](#) 提供了两种队列管理资源：

- **ClusterQueue** 是集群级别的队列，主要用于管理队列中的资源配额，包含了 CPU、内存、GPU 等资源
- **LocalQueue** 是命名空间级别的队列，需要指向到一个 **ClusterQueue**，用于使用队列中的资源分配

在 **AI Lab** 中，如果创建服务时，发现指定的命名空间不存在 **LocalQueue**，则会提示需要初始化队列。

在极少数情况下，可能由于特殊原因会导致 **LocalQueue** 初始化失败。

## 解决办法

检查 **Kueue** 是否正常运行，如果 **kueue-controller-manager** 未运行，可以通过以下命令查看。

```
kubectl get deploy kueue-controller-manager -n baize-sysatem
```

如果 **kueue-controller-manager** 未正常运行，请先修复 **Kueue**。