天翼云·批量计算

产品文档

天翼云科技有限公司

目 录

1 产品介绍

1.1 产品概述

批量计算作业平台是应对海量批处理作业的分布式作业任务管理平台。可支持海量作业并发，系统智能的对任意规模批处理作业进行资源分配、作业调度和任务管理。广泛应用于电影动画渲染、生物数据分析、多媒体转码、科学计算、人工智能等领域。

1.2 产品优势

1.2.1 简单易用

- **编排可视化**：编排可视，便于复杂任务流程的呈现、精细调整与管理，任务依赖关系（DAG），轻松组建工作流。
- **多种算力形态调度**：支持虚机和容器两种算力形态，并且支持不同业务的算力调度需求。
- **任务热点视图**：进度和热点实时呈现，方便运维，方便客户基于运行状态的反馈调整任务编排。

1.2.2 智能调度

- **支持资源共池**：面向资源跨域调度，全网竞价。
- **自研调度器**：面向容器/应用的调度算法优化，TPC-DS 测试较原生 K8S 性能提升 27%，调度性能 1000 pods/s , 10 倍于开源调度器。
- **多作业调度/管理**：支持多种作业类型统一调度管理，自动高效完成数据分发和计

算调度。

1.2.3 极致弹性

- **秒级弹性伸缩**：基于容器技术，提供秒级弹性伸缩能力，在流量突增时能快速弹性扩容，保障业务的连续性和高稳定性。
- **弹性扩容**：根据作业需求动态分配计算资源。

1.2.4 按量付费

- **按需使用**：按照计算资源实际使用量付费。
- **精细计费**：支持专属节点和共享节点，细粒度话单，精细成本管理。

1.3 应用场景

批量计算广泛应用于电影动画渲染、生物数据分析、多媒体转码、金融保险分析等领域。

- **HPC**

  适用于视频渲染、视频转码（视频格式转换、视频分辨率变化、添加水印/logo的）、科研教育等领域。

- **AI/大数据**

  适用于内容审核、OCR、图像识别、图片处理、美颜、语音识别、推荐、搜索、智能客服、游戏 AGI 等领域。

- **生物分析**

  适用于基因测序、药物检测等领域。

1.4 名词解释

1.4.1 资源池

资源池是批量计算服务所需要使用的计算资源的集合。批量计算的资源池由 ECK 和 ECX 提供，分为共享资源池、专属资源池。专属资源池即该用户专用，1 个专属资源池对应 1 个 ECK 集群，用户拥有该 K8s 集群的所有权限。共享资源池用户无需预先创建 ECK 集群，底层的 K8s 集群由平台统一管理和维护，平台用户共享，用户可创建虚拟的共享资源池运行计算作业。

1.4.2 队列

批量任务投递时所选择的队列，在进行任务调度时，会根据队列的权重进行优先调度。

1.4.3 作业

用户的每个特定的计算需求在批量计算中被描述为一个作业。作业是计算任务实例集合。

1.4.4 作业模板

根据一定的规则来描述计算任务的运行依赖或者先后顺序的文件，使用作业模板可以快速创建作业实例。

1.4.5 任务

一个作业由一组任务（Task）及其依赖关系组成。批量计算支持能以有向无环图 DAG（directed acyclic graph）形式描述的作业。任务间的依赖关系只能在作业提交时指定，提交完成后不能修改。

1.4.6 实例

每个任务可以有一个或多个执行实例（Instance）。同一任务的各个实例并行处理各自的输入数据。实例是批量计算调度与执行的最小单元，这些实例会动态的运行在系统分配的节点上。

1.4.7 集群

集群是容器运行所需云资源的集合，包含了若干台服务器节点、虚拟私有云等云资源。

1.4.8 镜像

镜像（Image）是虚机资源创建的模板。它是一个标准的或者自定义的 虚拟机镜像。

1.4.9 容器镜像

容器镜像是一种容器化标准交付物，用于打包应用程序及其依赖的环境。可以基于 Dockerfile 文件将应用构建为容器镜像并上传到容器镜像仓库中，然后您可以在测试或者生产环境中拉取容器镜像并启动容器。

1.4.10 存储源

批量计算平台支持将用户的文件存储/对象存储作为存储源接入，批量计算平台将存储的共享目录或者存储桶挂载到容器中。

1.4.11 数据集

批量计算通过数据集将文件系统中存在的某个目录或者对象存储中的 bucket 与容器内部的

文件系统上的某一目录建立绑定关系。这就意味着，当我们在容器中的这个目录下写入数据时，容器会将其内容直接写入到存储源中。

1.5约束与限制

开通限制

对于批量计算服务开通，您的账户需要通过实名认证并且要有 100 元的余额，否则无法开通批量计算服务。

区域限制

当前共享资源池仅支持在**华南一区**区域，其他区域请持续关注。

配额限制

使用批量计算时，需注意默认区域有配额限制，具体如下表所示。如果默认区域或者配额不能满足业务使用，可通过工单申请或联系客户经理调整区域和配额。

分类	限制数量	单位
CPU	500	核
内存	1000	GB
GPU	100	块

2 计费说明

2.1 计费项

在批量计算中进行全流程使用时，涉及到计费项主要包括资源费用和存储费用。

- 资源费用：使用批量计算平台的计算资源产生的费用。
- 存储费用：使用天翼云对象存储产生的费用、使用天翼云智能边缘云文件存储等存储服务产生的费用。

资源费用

批量计算的资源池包括共享资源池和专属资源池，在使用时，您可根据业务需要自行选择。

- 共享资源池：批量计算提供的共享资源池是按需计费的资源池，若您选择共享资源池，则默认您的资源是按需计费的。
- 专属资源池：若您选择专属资源池，其计费方式由您购买的专属资源池决定。例如，您购买的专属资源池是按需的资源池，则使用时就按照按需的模式计费。

资源池类型	计费项	价格	单位
共享资源池	CPU	0.09	元/核/小时
	内存	0.03	元/GB/小时
	GPU（NVIDIA T4 16G）	4.09	元/块/小时
	GPU（NVIDIA A10 24G）	5.55	元/块/小时
	GPU（NVIDIA V100 32G）	16.07	元/块/小时
	GPU（NVIDIA A100 40G）	18.9	元/块/小时
	GPU（Atlas 300I 32G）	5.55	元/块/小时
	GPU（Cambricon MLU370 24G）	2.47	元/块/小时
专属资源池	边缘容器集群（ECK 专有版）	边缘容器集群（ECK 专有版）价格详情	

存储费用

存储类型	计费说明
对象存储	● 使用天翼云对象存储服务，存储数据和模型，会产生相应的费用，具体费用可参见对象存储价格详情。 ● 使用第三方对象存储服务，不收取额外费用。
文件存储	● 使用天翼云智能边缘云文件存储服务，存储数据和模型，会产生相应的费用，具体费用

	可参见智能边缘云价格详情。 ● 使用第三方对象存储服务，不收取额外费用。

2.2计费方式

计费说明

批量计算资源包含 CPU、内存、GPU 等，根据您申请的实际实例资源规格，以及每个实例实际运行时长按需计费，计费时长从开始下载容器镜像 (docker pull) 到实例停止使用进行计算。以小时为出账周期，在每个结算周期生成账单并从账户中扣除相应费用。

计费周期

系统以“小时”为单位统计实际用量（单位取整点区间，如 8:00-9:00），根据实际用量生成话单并结算扣费。

账单出账时间通常在当前计费周期结束后，具体出账时间以系统为准。

计费公式

实例费用的计算公式为：实例费用=实例 CPU 资源量 x CPU 单价 x 运行时长+实例内存资源量 x 内存单价 x 运行时长+实例 GPU 资源量 x GPU 单价 x 运行时长。

产品定价

批量计算产品定价信息可参考 2.1 节[计费项]说明。

计费案例

用户 A 在 2023 年 6 月 1 日 8:00~9:00 的计费周期内，创建一个 2 vCPU、4 GB 内存、1 块 NVIDIA T4 16G GPU 卡的开发环境实例，并在实例运行 1 小时后删除。则用户 A 在此计费

周期内的费用如下：

- CPU 费用=2 核 x 0.09 元/核/小时 x 1 = 0.18 元
- 内存费用=4GB x 0.03 元/GB x 1 = 0.12 元
- GPU 费用=1 x 4.09 元/块/小时 x 1 = 4.09 元
- 实例总费用=CPU 费用+内存费用+GPU 费用=0.18+0.12+4.09=4.39 元

3 快速入门

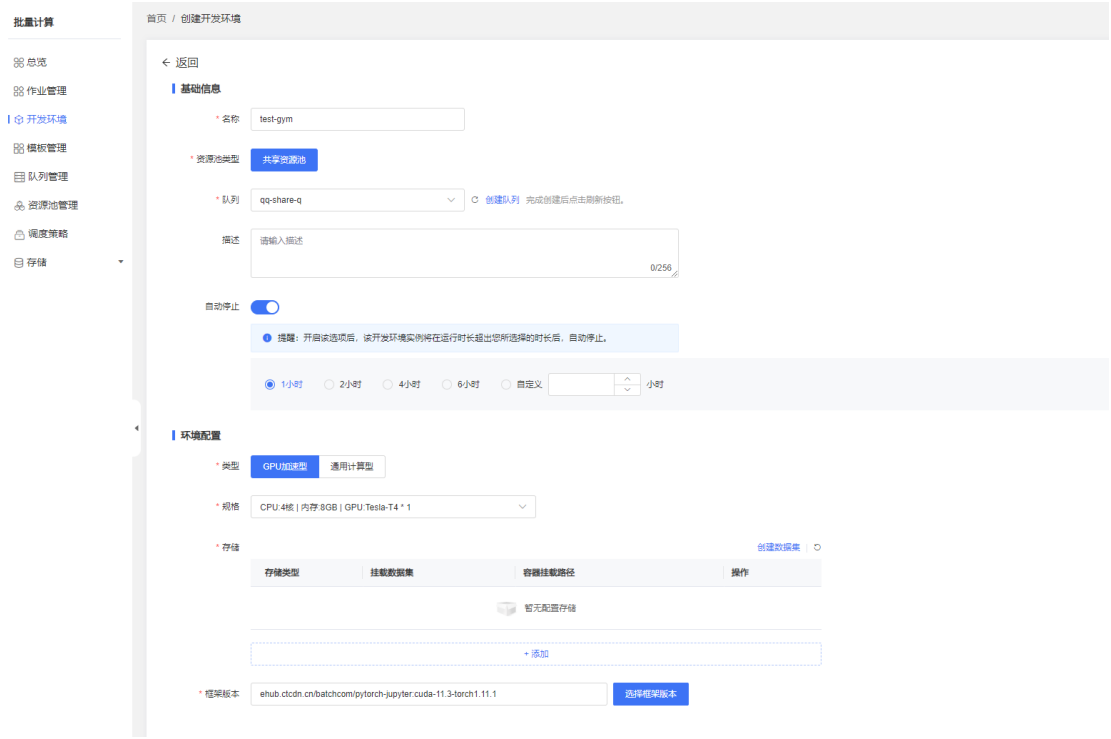
3.1 准备工作

1. 创建天翼云账号。如果您还没有天翼云账号，请登录天翼云官网，点击右上角“免费注册”创建天翼云账号。
2. 完成账号实名制认证且账户余额大于 100 元，使用注册成功的天翼云账号登录，进入批量计算产品详情页点击【立即开通】按钮，进入业务开通页开通批量计算服务。
3. 为了更好的在批量计算平台进行的操作，可先了解批量计算中的几个概念，具体请参见 1.2 节名词解释。
4. 登录批量计算管理控制台。
5. 创建专属资源池。在左侧导航栏单击【资源池管理】，在【专属资源池】页签下，在左上方单击【创建专属资源池】，创建专属资源池。
6. 创建共享资源池。在左侧导航栏单击【资源池管理】，在【共享资源池】页签下，在左上方单击“创建共享资源池”，创建共享资源池。
7. 创建队列。在左侧导航栏单击“队列管理”，在左上方单击“创建队列”，创建队列，并确认队列内有足够的资源配额。

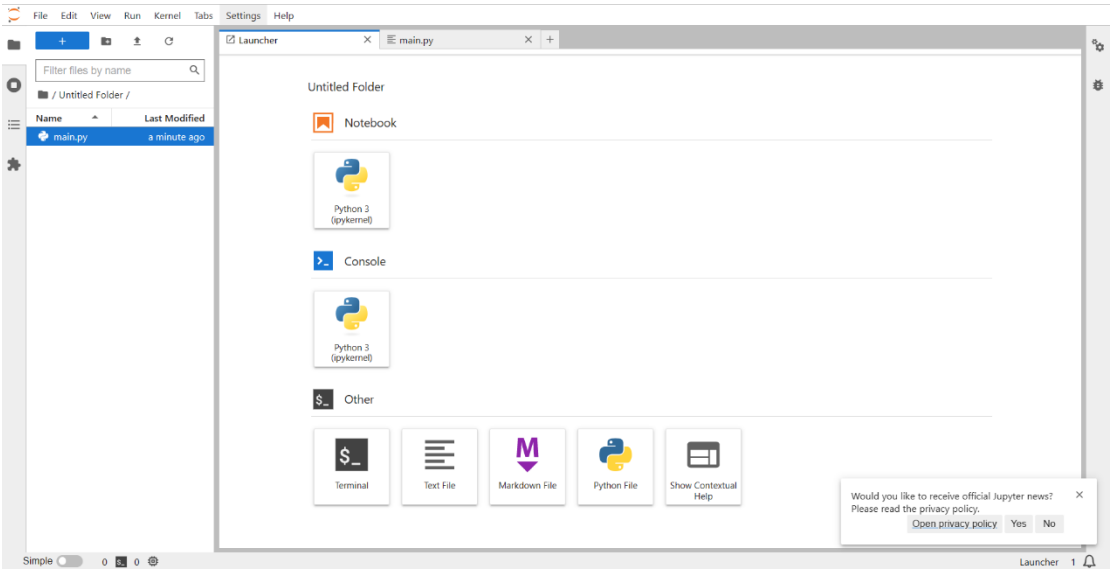
3.2 使用开发环境实现 AI 模型开发和部署

操作步骤

1. 在控制台左侧导航栏中，选择【开发环境】。进入【开发环境】列表，点击左上角【创建开发环境】，设置基本信息和环境配置，规格为选择 2 核 CPU，4G 内存和一张 GPU 显卡，IDE 框架选择 Jupyter。



2. 在控制台左侧导航栏中，选择【开发环境】。在【开发环境】列表，点击运行中的目标开发环境右侧操作栏的【打开】按钮，进入开发环境。

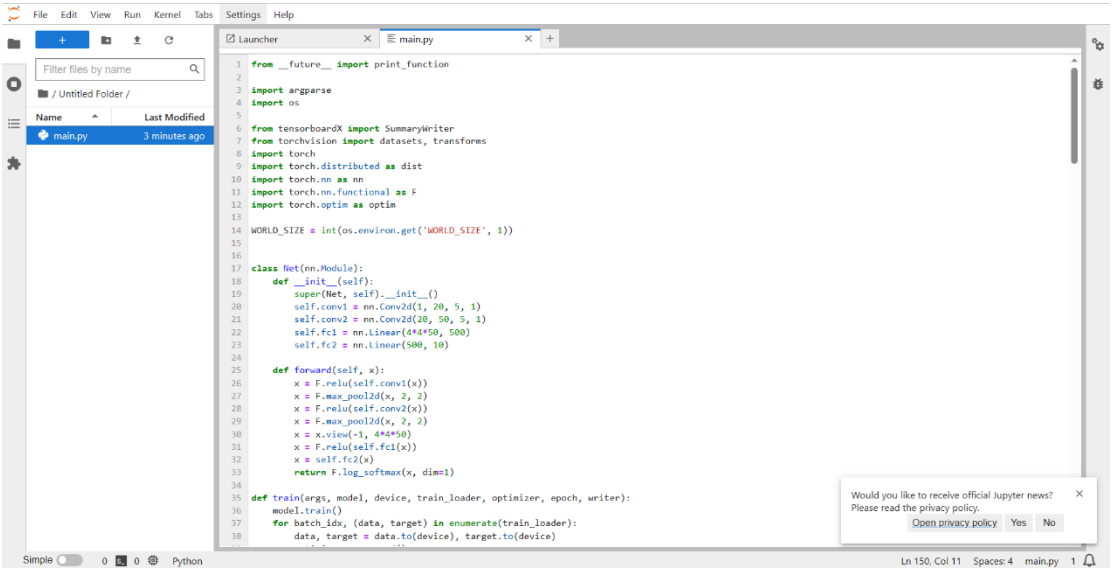


3. 点击开发环境左上角的【New Folder】按钮，新建一个文件夹，并且命名为“test”。

4. 双击 test 文件夹，进入文件，在文件夹内新建一个 python 文件，并且命名为“main.py” 并且把 github 链接

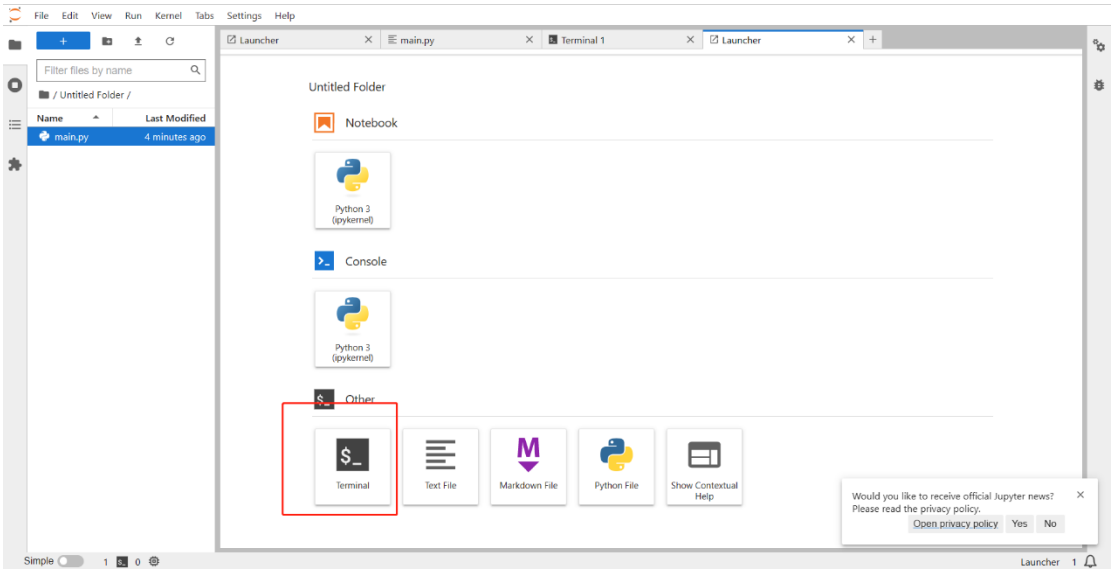
   https://github.com/kubeflow/training-operator/blob/master/examples/pytorch/mnist/mnist.py

   代码复制到新建的 main.py 文件里面，并保存文件。



5. 点击“Other”的 Terminal 打开一个新的终端，输入命令 pip install tensorboardX -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com 安装外部依赖包，依赖包根据代码里面的需求而定，不是必

须的。




```
$ pip install tesn^C
$ pip install tensorboardX -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
Looking in indexes: http://mirrors.aliyun.com/pypi/simple/
Collecting tensorboardX
  Downloading http://mirrors.aliyun.com/pypi/packages/60/9f/d532d37f10ac7af136d4c2ba71e1fe7af0f3cc0cc076dfc05826171e9737/tensorboardX-2.6-py2.py3-none-any.whl (114 kB)
     |████████████████████████████████| 114 kB 4.4 MB/s
Requirement already satisfied: packaging in /home/user/miniconda/lib/python3.8/site-packages (from tensorboardX) (23.1)
Requirement already satisfied: numpy in /home/user/miniconda/lib/python3.8/site-packages (from tensorboardX) (1.21.5)
Collecting protobuf<4,>=3.8.0
  Downloading http://mirrors.aliyun.com/pypi/packages/da/e4/4d62585593e9f962cb02614534f62f930de6a80a0a3784282094a01919b2/protobuf-3.20.3-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.0 MB)
     |████████████████████████████████| 1.0 MB 19.3 MB/s
Installing collected packages: protobuf, tensorboardX
Successfully installed protobuf-3.20.3 tensorboardX-2.6
```

6. 在终端输入以下命令，进入 test 目录

   cd test

   输入以下命令启动训练

   python main.py

   终端中会显示训练的过程，首先会下载数据集，然后处理数据，接着将数据送到模型训练，这时候会打印训练日志，包括周期（epoch），损失函数（loss），最后将打印测试的准确率（accuracy）。如果需要更改训练参数，只需要更改代码中的参数，然后再执行 python main.py 命令即可。

```
$ python main.py
Using CUDA
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz
26422272it [00:22, 1175416.44it/s]
Extracting ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
29696it [00:00, 177475.71it/s]
Extracting ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
4422656it [00:03, 1387412.02it/s]
Extracting ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
6144it [00:00, 19976592.07it/s]
Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

Processing...
/home/user/miniconda/lib/python3.8/site-packages/torchvision/datasets/mnist.py:502: UserWarning: The given NumPy array is not writeable, and PyTorch does not support non-write
able tensors. This means you can write to the underlying (supposedly non-writeable) NumPy array using the tensor. You may want to copy the array to protect its data or make it
 writeable before converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at  /opt/conda/conda-bld/pytorch_16165
54793803/work/torch/csrc/utils/tensor_numpy.cpp:143.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
Done!
Train Epoch: 1 [0/60000 (0%)]   loss=2.2893
Train Epoch: 1 [640/60000 (1%)] loss=2.2348
Train Epoch: 1 [1280/60000 (2%)]        loss=2.1075
Train Epoch: 1 [1920/60000 (3%)]        loss=1.9258
Train Epoch: 1 [2560/60000 (4%)]        loss=1.6066
Train Epoch: 1 [3200/60000 (5%)]        loss=1.4615
Train Epoch: 1 [3840/60000 (6%)]        loss=1.2695
Train Epoch: 1 [4480/60000 (7%)]        loss=1.1991
Train Epoch: 1 [5120/60000 (9%)]        loss=1.0132
Train Epoch: 1 [5760/60000 (10%)]       loss=0.9738
```

Launcher | main.py | Terminal 1 | Terminal 2

```
Train Epoch: 1 [37120/60000 (62%)]      loss=0.5700
Train Epoch: 1 [37760/60000 (63%)]      loss=0.7946
Train Epoch: 1 [38400/60000 (64%)]      loss=0.6927
Train Epoch: 1 [39040/60000 (65%)]      loss=0.7607
Train Epoch: 1 [39680/60000 (66%)]      loss=0.4932
Train Epoch: 1 [40320/60000 (67%)]      loss=0.5422
Train Epoch: 1 [40960/60000 (68%)]      loss=0.5013
Train Epoch: 1 [41600/60000 (69%)]      loss=0.5820
Train Epoch: 1 [42240/60000 (70%)]      loss=0.4423
Train Epoch: 1 [42880/60000 (71%)]      loss=0.5409
Train Epoch: 1 [43520/60000 (72%)]      loss=0.3148
Train Epoch: 1 [44160/60000 (74%)]      loss=0.5362
Train Epoch: 1 [44800/60000 (75%)]      loss=0.5567
Train Epoch: 1 [45440/60000 (76%)]      loss=0.4668
Train Epoch: 1 [46080/60000 (77%)]      loss=0.4772
Train Epoch: 1 [46720/60000 (78%)]      loss=0.5452
Train Epoch: 1 [47360/60000 (79%)]      loss=0.4785
Train Epoch: 1 [48000/60000 (80%)]      loss=0.4849
Train Epoch: 1 [48640/60000 (81%)]      loss=0.5360
Train Epoch: 1 [49280/60000 (82%)]      loss=0.6677
Train Epoch: 1 [49920/60000 (83%)]      loss=0.4967
Train Epoch: 1 [50560/60000 (84%)]      loss=0.6184
Train Epoch: 1 [51200/60000 (85%)]      loss=0.6146
Train Epoch: 1 [51840/60000 (86%)]      loss=0.5881
Train Epoch: 1 [52480/60000 (87%)]      loss=0.3743
Train Epoch: 1 [53120/60000 (88%)]      loss=0.5875
Train Epoch: 1 [53760/60000 (90%)]      loss=0.7067
Train Epoch: 1 [54400/60000 (91%)]      loss=0.4856
Train Epoch: 1 [55040/60000 (92%)]      loss=0.6245
Train Epoch: 1 [55680/60000 (93%)]      loss=0.4305
Train Epoch: 1 [56320/60000 (94%)]      loss=0.5612
Train Epoch: 1 [56960/60000 (95%)]      loss=0.6511
Train Epoch: 1 [57600/60000 (96%)]      loss=0.4341
Train Epoch: 1 [58240/60000 (97%)]      loss=0.4351
Train Epoch: 1 [58880/60000 (98%)]      loss=0.6117
Train Epoch: 1 [59520/60000 (99%)]      loss=0.3480

accuracy=0.7825

$ ^X
/usr/bin/sh: 5: : not found
$
```

7. 显存占用查看命令：watch -n 1 nvidia-smi

每秒钟显示一次显卡的状态，这样可以方便查看程序是否使用了 GPU。

```
Every 1.0s: nvidia-smi                  jupyter-gpu: Wed May 24 11:39:58 2023

Wed May 24 11:39:58 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.106.00   Driver Version: 460.106.00   CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:05:00.0 Off |                    0 |
| N/A   29C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

3.3 使用控制台快速提交作业

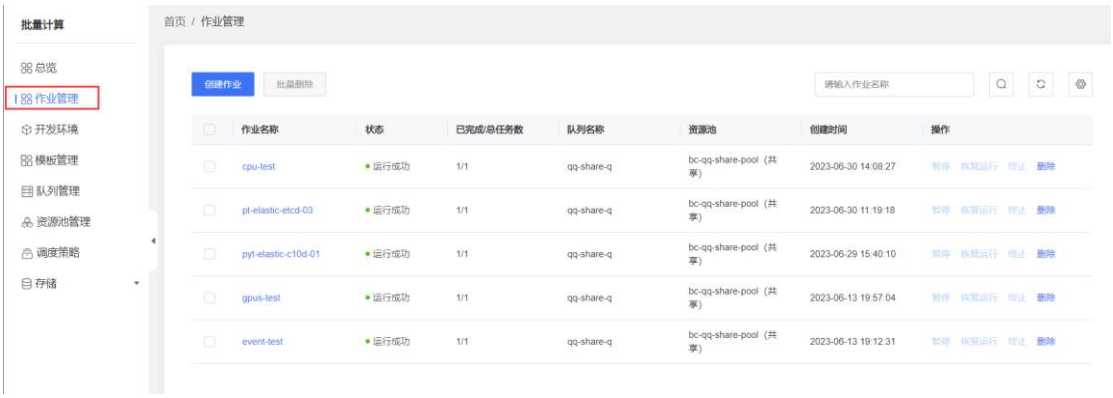
介绍如何使用控制台提交一个作业。

步骤预览

1. 作业准备
    - 制作深度学习框架训练镜像，如 Pytorch 训练镜像
2. 使用控制台提交作业
    - 创建作业
    - 编排作业
    - 编辑任务
    - 提交作业
3. 查看作业状态
    - 查看作业
    - 查看任务
    - 查看 Pod

步骤一：创建作业

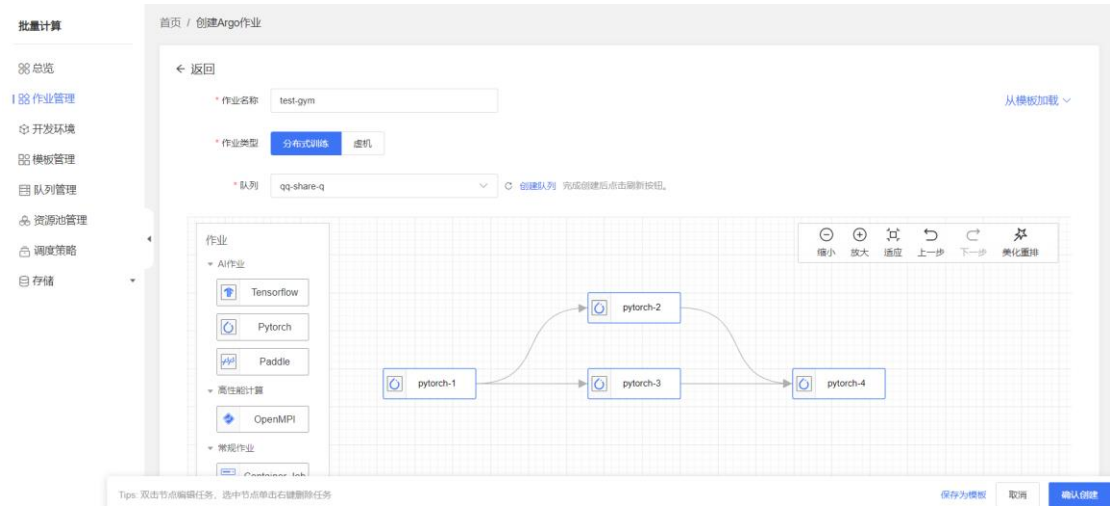
1. 点击导航栏【作业管理】菜单。



2. 点击【创建作业】按钮。

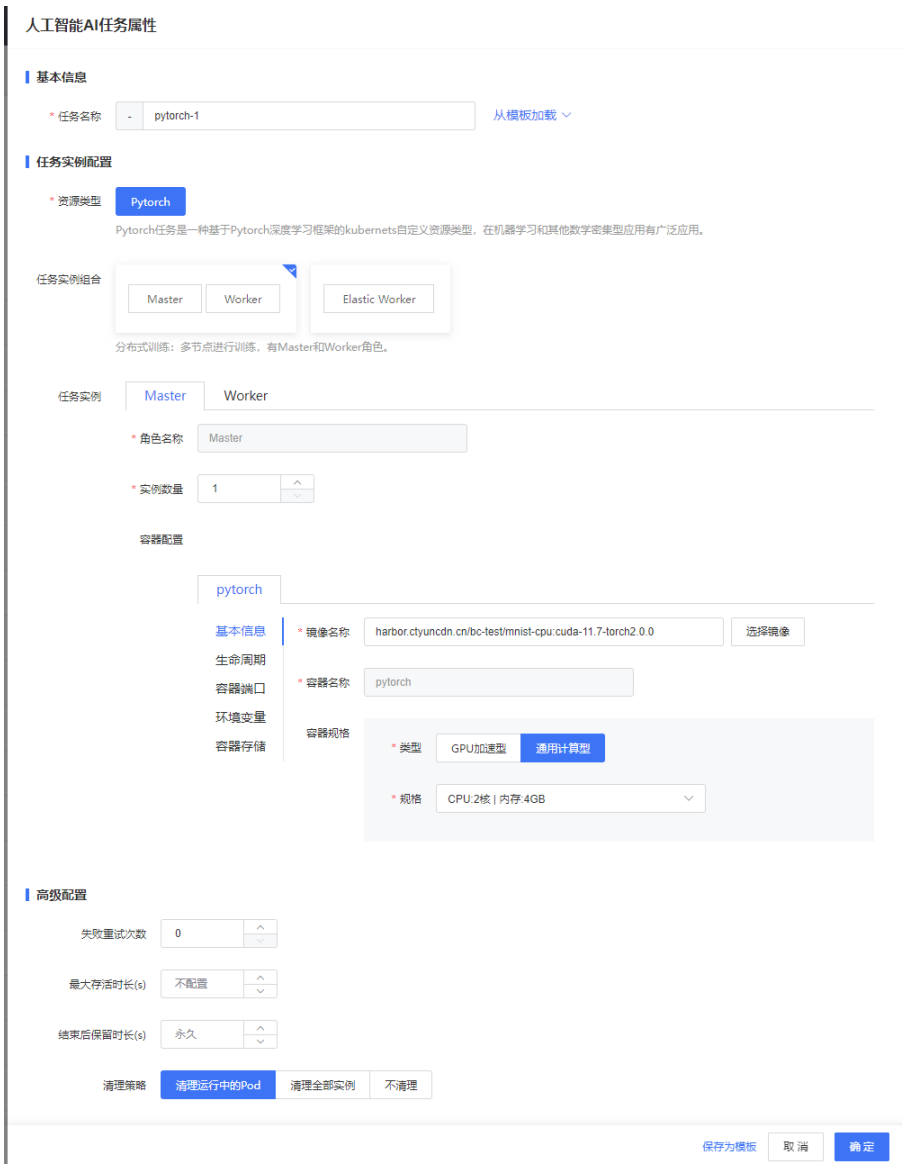
步骤二：编排作业

1. 输入作业名称，选择队列，拖拽任务图标，关联任务。

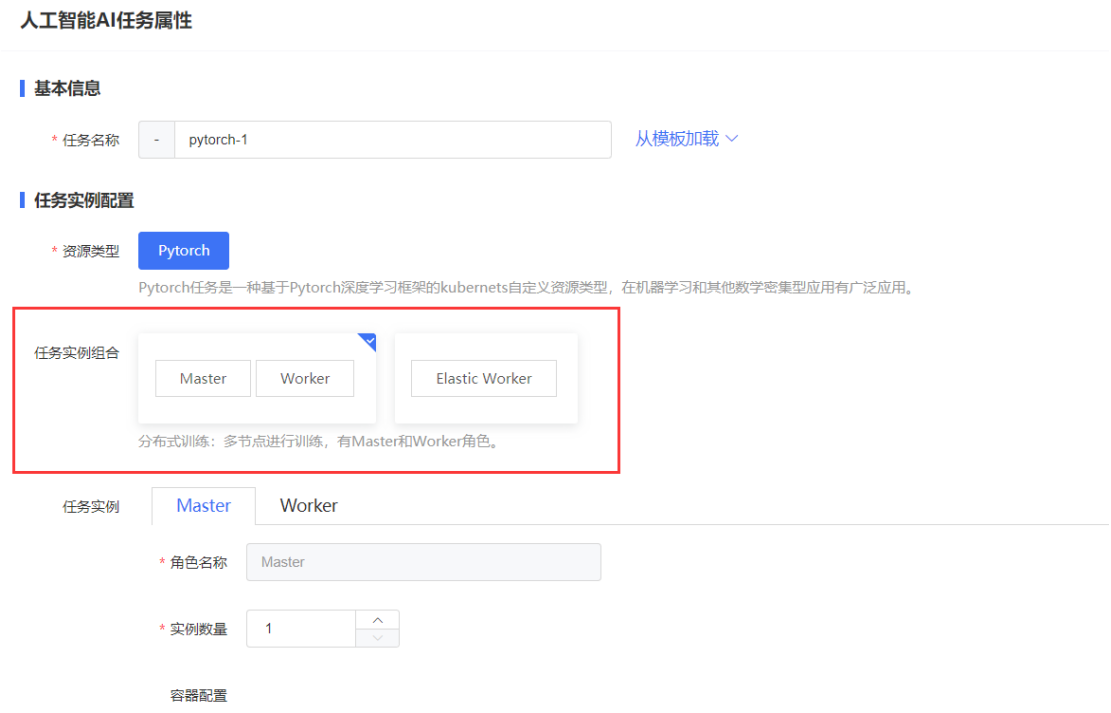


步骤三：编辑任务

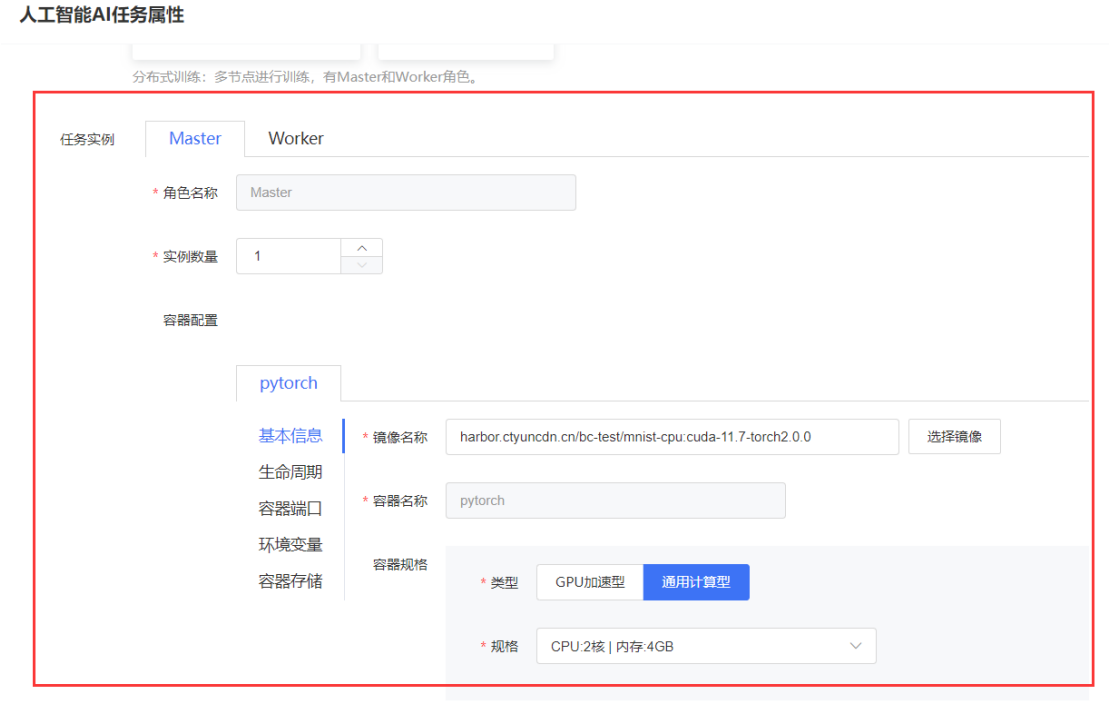
1. 双击任务图标编辑任务


人工智能AI任务属性
基本信息
* 任务名称
pytorch-1
从模板加载
任务实例配置
* 资源类型
Pytorch
Pytorch任务是一种基于Pytorch深度学习框架的kubernets自定义资源类型，在机器学习和其他数学密集型应用有广泛应用。
任务实例组合
Master
Worker
Elastic Worker
分布式训练：多节点进行训练，有Master和Worker角色。
任务实例
Master
Worker
* 角色名称
Master
* 实例数量
1
容器配置
pytorch
基本信息
生命周期
容器端口
环境变量
容器存储
* 镜像名称
harbor.ctyuncdn.cn/bc-test/mnist-cpu:cuda-11.7-torch2.0.0
选择镜像
* 容器名称
pytorch
容器规格
* 类型
GPU加速型
通用计算型
* 规格
CPU:2核 | 内存:4GB
高级配置
失败重试次数
0
最大存活时长(s)
不配置
结束后保留时长(s)
永久
清理策略
清理运行中的Pod
清理全部实例
不清理
保存为模板
取 消
确 定

2. 选择任务实例组合

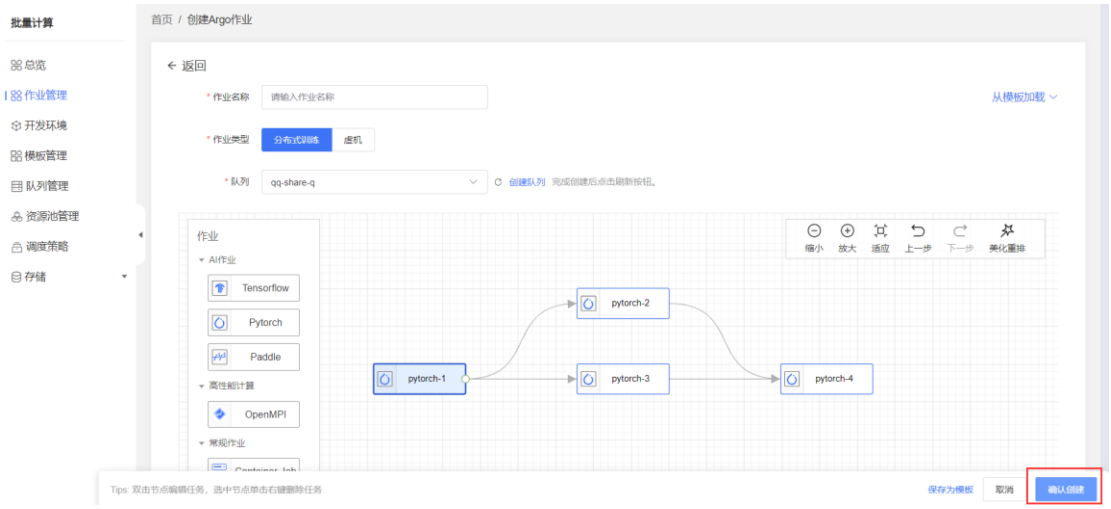


3. 配置任务实例



4. 配置高级配置

5. 提交作业

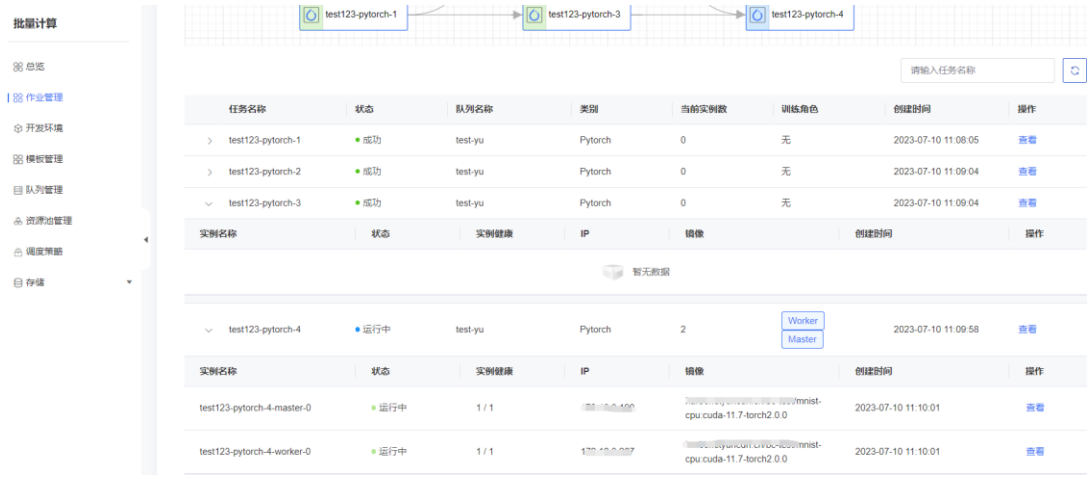


步骤四：查看作业

1. 点击导航栏【作业管理】菜单，进入作业列表，在点击作业名称，进作业详情。

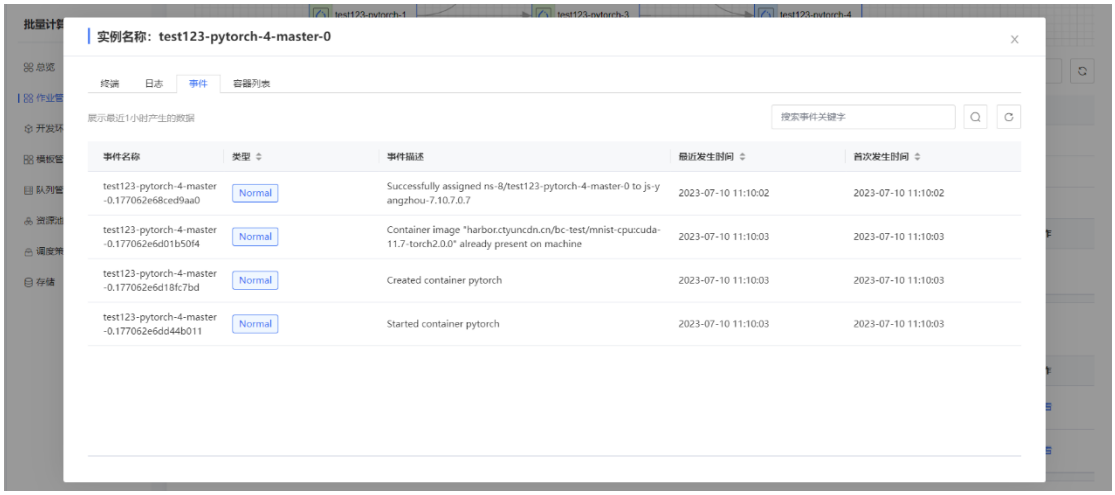
步骤五：查看任务

1. 在作业详情，点击任务列表左侧的下转按钮，展开任务。



步骤六：查看任务实例

1. 点击 Pod 列表右侧操作栏的【查看】按钮。可查看 Pod 的终端、日志、事件和容器列表。



3.4使用 tensorboard 可视化训练过程

官方文档：https://www.tensorflow.org/tensorboard/get_started?hl=zh-cn

TensorBoard 是一组用于数据可视化的工具，目前支持的框架为 pytorch 和 tensorflow。

步骤预览

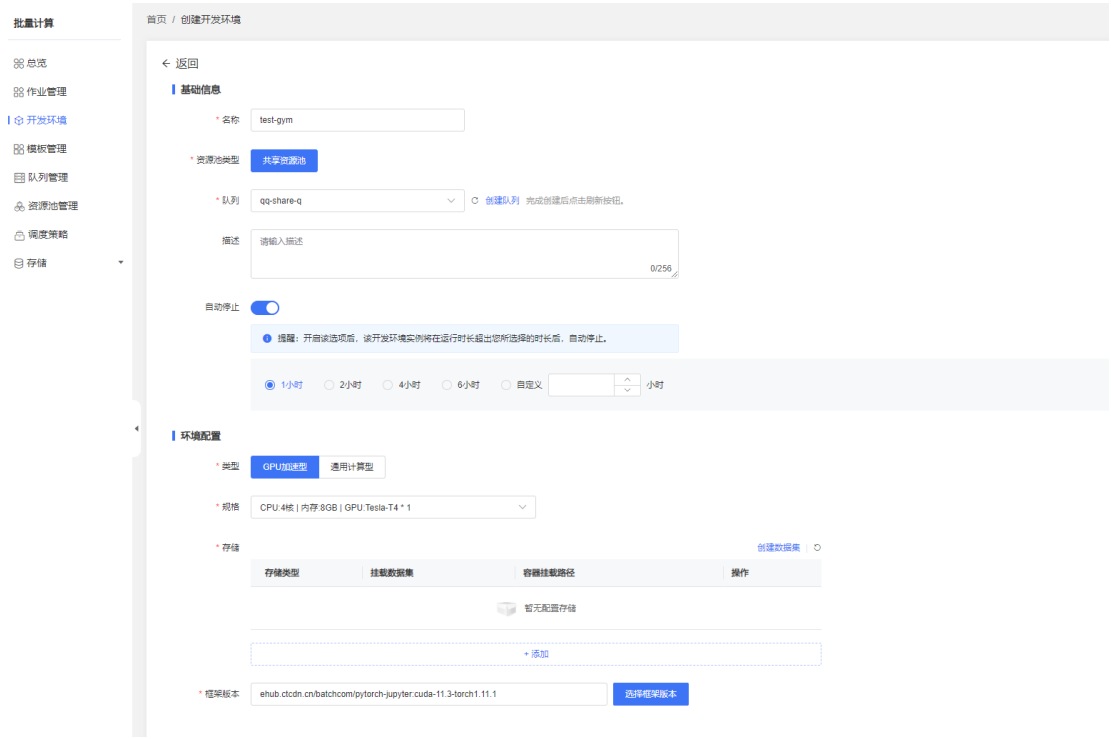
1. 创建 jupyter 开发环境

2. 训练模型并且保存训练参数

3. 安装 tensorboard 以及插件

4. 保存镜像并创建一个新的开发环境，打开 tensorboard 查看训练数据。

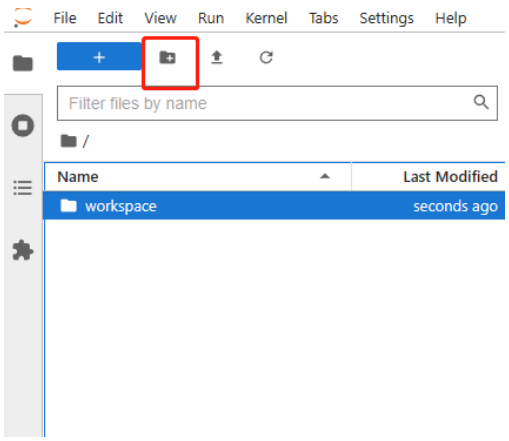
步骤一：创建 jupyter 开发环境

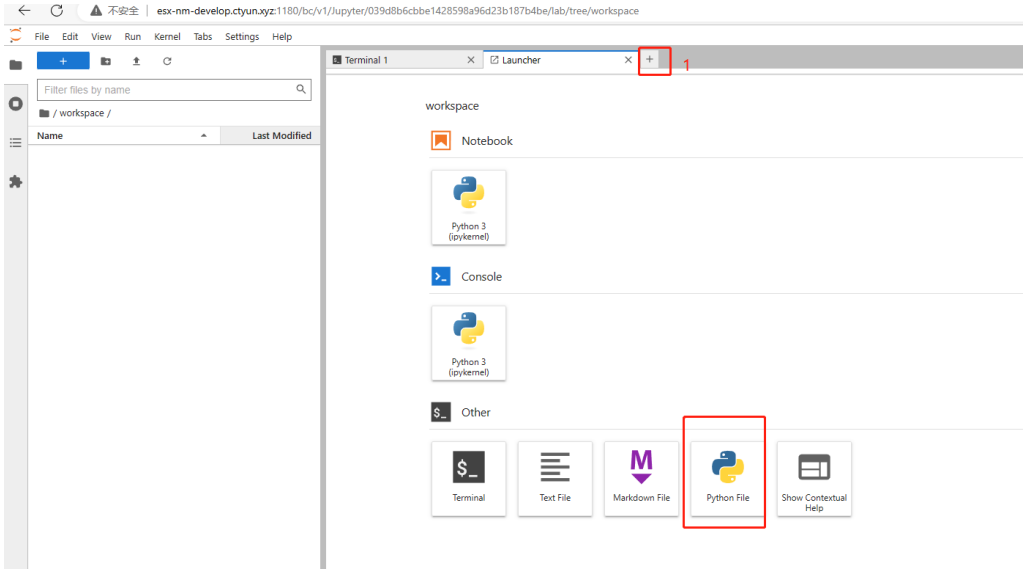


规格建议选择 GPU 加速型

步骤二：训练模型并且保存训练参数

新建”workspace”工作目录。

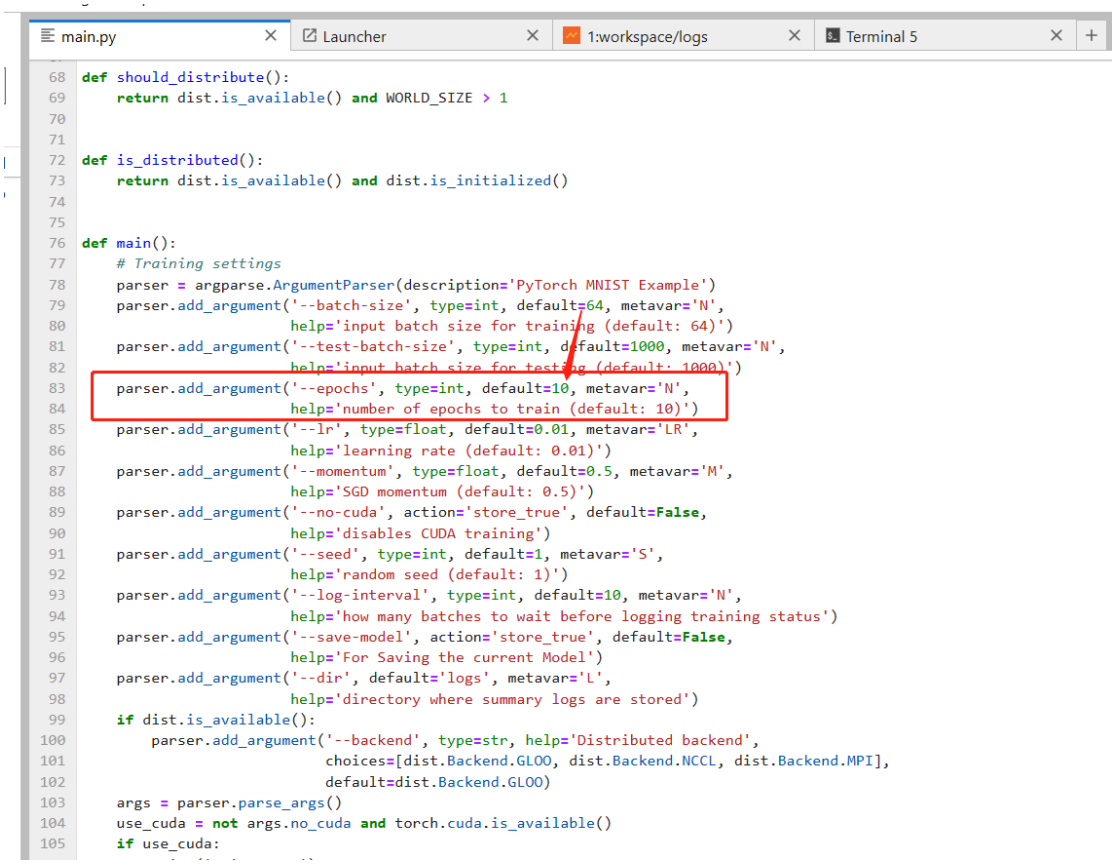




新建一个 main.py 把 github 链接

https://github.com/kubeflow/training-operator/blob/master/examples/pytorch/mnist/mnist.py

代码复制到新建的 main.py 文件里面，Epoch 可以改成 10，并保存文件。



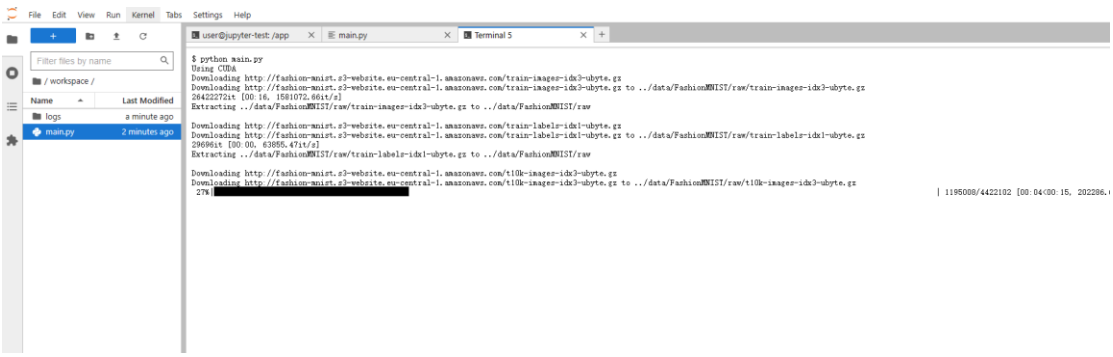
步骤三：安装 tensorboard 以及插件

在终端中安装 tensorboard，tensorboardX，

pip install tensorboardX -i https://pypi.tuna.tsinghua.edu.cn/simple

pip install tensorboard -i https://pypi.tuna.tsinghua.edu.cn/simple

```
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/6f/bb/5deac77a9af870143c684ab46a7934038a53eb4aa975bc0687ed6ca2c610/requests_oauthli
Requirement already satisfied: MarkupSafe>=2.1.1 in /home/user/miniconda/lib/python3.8/site-packages (from werkzeug>=1.0.1->tensorboard) (2.
Requirement already satisfied: idna<4,>=2.5 in /home/user/miniconda/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard) (2.9
Requirement already satisfied: certifi>=2017.4.17 in /home/user/miniconda/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard
Requirement already satisfied: charset-normalizer<4,>=2 in /home/user/miniconda/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensc
Requirement already satisfied: zipp>=0.5 in /home/user/miniconda/lib/python3.8/site-packages (from importlib-metadata>=4.4; python_version <
Collecting pyasn1>=0.1.3
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/14/e5/b56a725cbde139aa960c26a1a3ca4d4af437282e20b5314ee6a3501e7dfc/pyasn1-0.5.0-py2
     |████████████████████████████████| 83 kB 7.0 MB/s
Collecting oauthlib>=3.0.0
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/7e/80/cab10959dc1faead58dc8384a781dfbf93cb4d33d50988f7a69f1b7c9bbe/oauthlib-3.2.2-p
     |████████████████████████████████| 151 kB 80.2 MB/s
Installing collected packages: protobuf, markdown, absl-py, grpcio, pyasn1, rsa, pyasn1-modules, cachetools, google-auth, oauthlib, requests
board
Successfully installed absl-py-1.4.0 cachetools-5.3.0 google-auth-2.18.0 google-auth-oauthlib-1.0.0 grpcio-1.54.2 markdown-3.4.3 oauthlib-3.
3.1 rsa-4.9 tensorboard-2.13.0 tensorboard-data-server-0.7.0 werkzeug-2.3.4
user@jupyter-test:/app$ pip install tensorboardX -i https://pypi.tuna.tsinghua.edu.cn/simple
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting tensorboardX
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/60/9f/d532d37f10ac7af136d4c2ba71e1fe7af0f3cc0cc076dfc05826171e9737/tensorboardX-2.6
     |████████████████████████████████| 114 kB 1.8 MB/s
Requirement already satisfied: numpy in /home/user/miniconda/lib/python3.8/site-packages (from tensorboardX) (1.21.5)
Requirement already satisfied: packaging in /home/user/miniconda/lib/python3.8/site-packages (from tensorboardX) (23.1)
Collecting protobuf<4,>=3.8.0
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/da/e4/4d62585593e9f962cb02614534f62f930de6a80a0a3784282094a01919b2/protobuf-3.20.3-
     |████████████████████████████████| 1.0 MB 7.6 MB/s
Installing collected packages: protobuf, tensorboardX
  Attempting uninstall: protobuf
    Found existing installation: protobuf 4.23.0
    Uninstalling protobuf-4.23.0:
      Successfully uninstalled protobuf-4.23.0
Successfully installed protobuf-3.20.3 tensorboardX-2.6
```

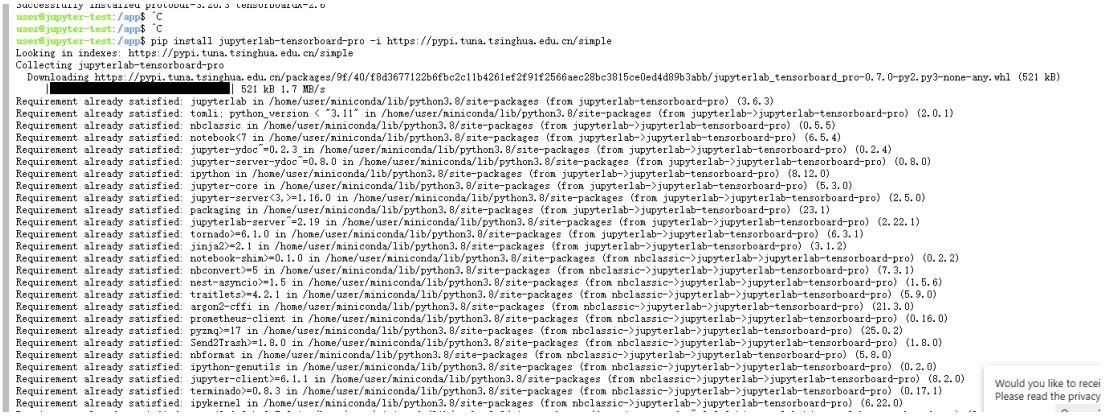


新建一个终端，输入 python main.py，开始训练，同时 workspace 目录下有 logs 文件生成。

更新 tornado 版本，pip install tornado==6.2

安装 tensorboard 插件：

pip install jupyterlab-tensorboard-pro -i https://pypi.tuna.tsinghua.edu.cn/simple



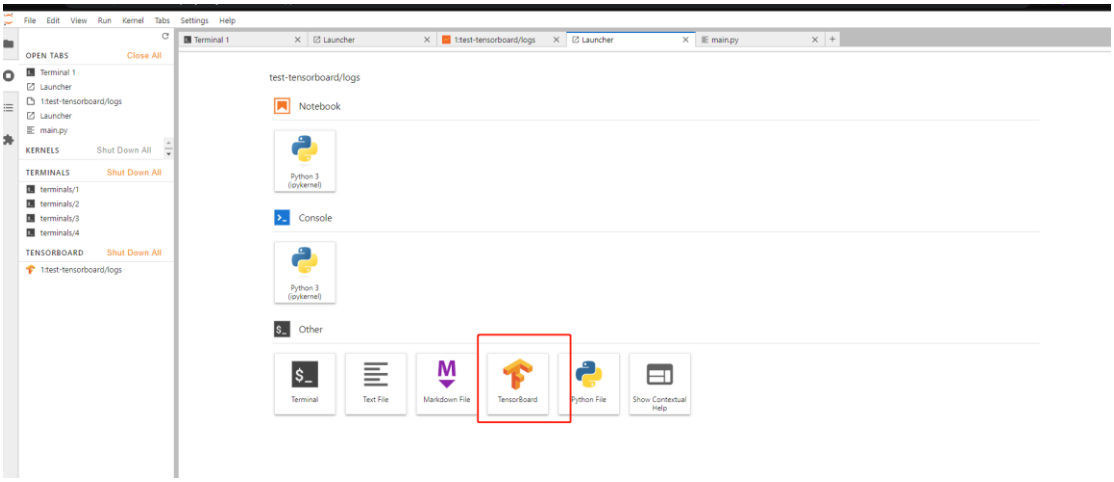
```
user@jupyter-test:/app$ ^C
user@jupyter-test:/app$ ^C
user@jupyter-test:/app$ pip install jupyterlab-tensorboard-pro -i https://pypi.tuna.tsinghua.edu.cn/simple
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting jupyterlab-tensorboard-pro
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/9f/40/f8d3677122b6fbc2c11b4261ef2f91f2566aec28bc3815ce0ed4d89b3abb/jupyterlab_tensorboard_pro-0.7.0-py2.py3-none-any.whl (521 kB)
     |████████████████████████████████| 521 kB 1.7 MB/s
Requirement already satisfied: jupyterlab in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab-tensorboard-pro) (3.6.3)
Requirement already satisfied: tomli; python_version < "3.11" in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (2.0.1)
Requirement already satisfied: nbclassic in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (0.5.5)
Requirement already satisfied: notebook<7 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (6.5.4)
Requirement already satisfied: jupyter-ydoc~=0.2.3 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (0.2.4)
Requirement already satisfied: jupyter-server-ydoc~=0.8.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (0.8.0)
Requirement already satisfied: ipython in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (8.12.0)
Requirement already satisfied: jupyter-core in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (5.3.0)
Requirement already satisfied: jupyter-server<3,>=1.16.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (2.5.0)
Requirement already satisfied: packaging in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (23.1)
Requirement already satisfied: jupyterlab-server~=2.19 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (2.22.1)
Requirement already satisfied: tornado>=6.1.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (6.3.1)
Requirement already satisfied: jinja2>=2.1 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab->jupyterlab-tensorboard-pro) (3.1.2)
Requirement already satisfied: notebook-shim>=0.1.0 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (0.2.2)
Requirement already satisfied: nbconvert>=6 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (7.3.1)
Requirement already satisfied: nest-asyncio>=1.5 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (1.5.6)
Requirement already satisfied: traitlets>=4.2.1 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (5.9.0)
Requirement already satisfied: argon2-cffi in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (21.3.0)
Requirement already satisfied: prometheus-client in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (0.16.0)
Requirement already satisfied: pyzmq>=17 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (25.0.2)
Requirement already satisfied: Send2Trash>=1.8.0 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (1.8.0)
Requirement already satisfied: nbformat in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (5.8.0)
Requirement already satisfied: ipython-genutils in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (0.2.0)
Requirement already satisfied: jupyter-client>=6.1.1 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (8.2.0)
Requirement already satisfied: terminado>=0.8.3 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (0.17.1)
Requirement already satisfied: ipykernel in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab->jupyterlab-tensorboard-pro) (6.22.0)
```

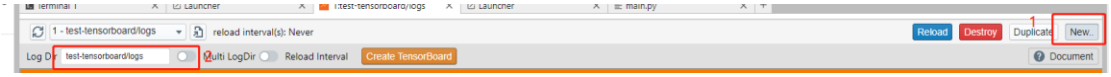
步骤四：4. 保存镜像并创建一个新的开发环境，打开 tensorboard 查看训练数据

因为安装完 tensorboard 插件需要重新创建开发环境，所以安装之后需要保存镜像。

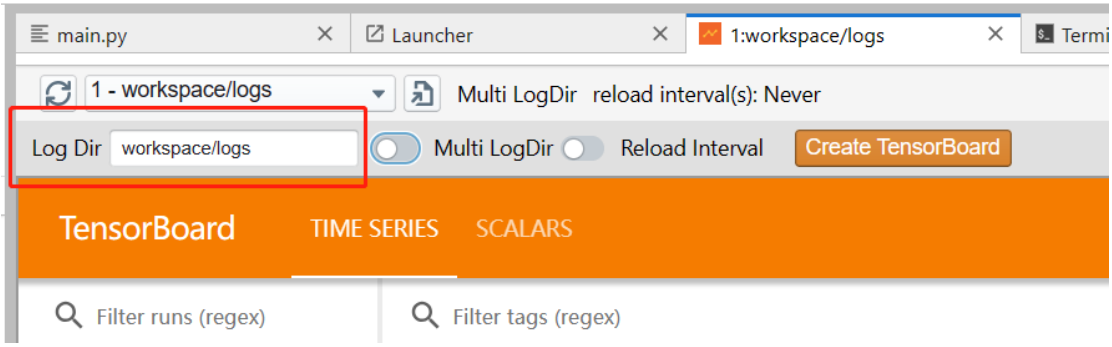
然后新建一个 jupyter 开发环境。

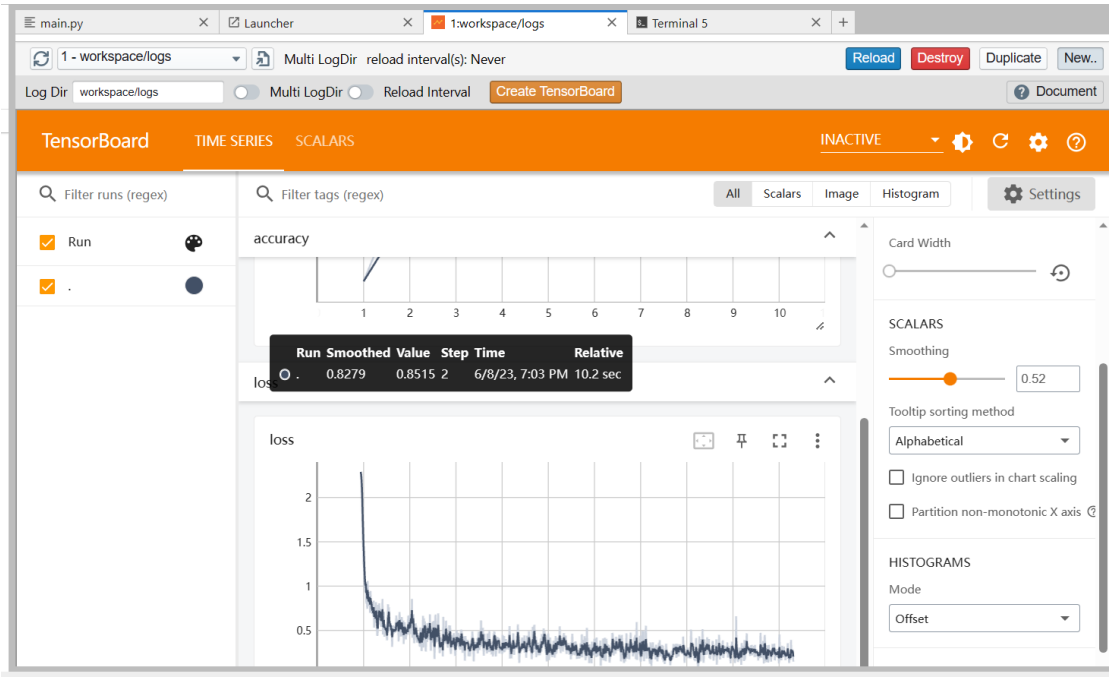
打开 tensorboard





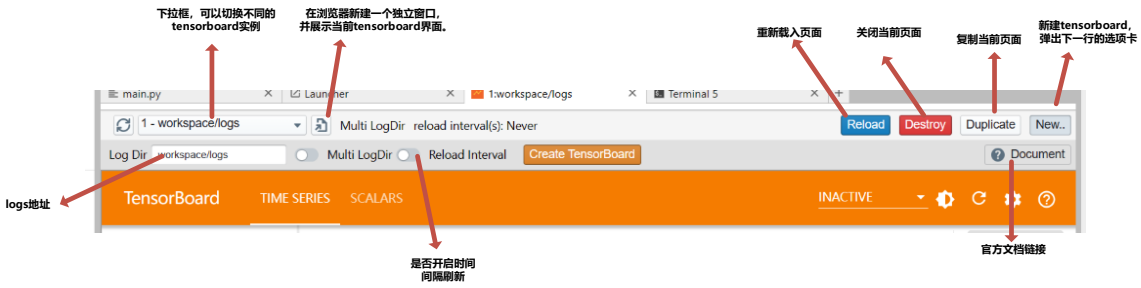
在 log dir 输入日志文件的地址





图中展示 accuracy 和 loss 的值变化过程，训练过程中一般关心 loss 值的变化趋势，观察趋势对训练参数做出对应调整。

Tensorboard 插件界面常用操作介绍



4 操作指导

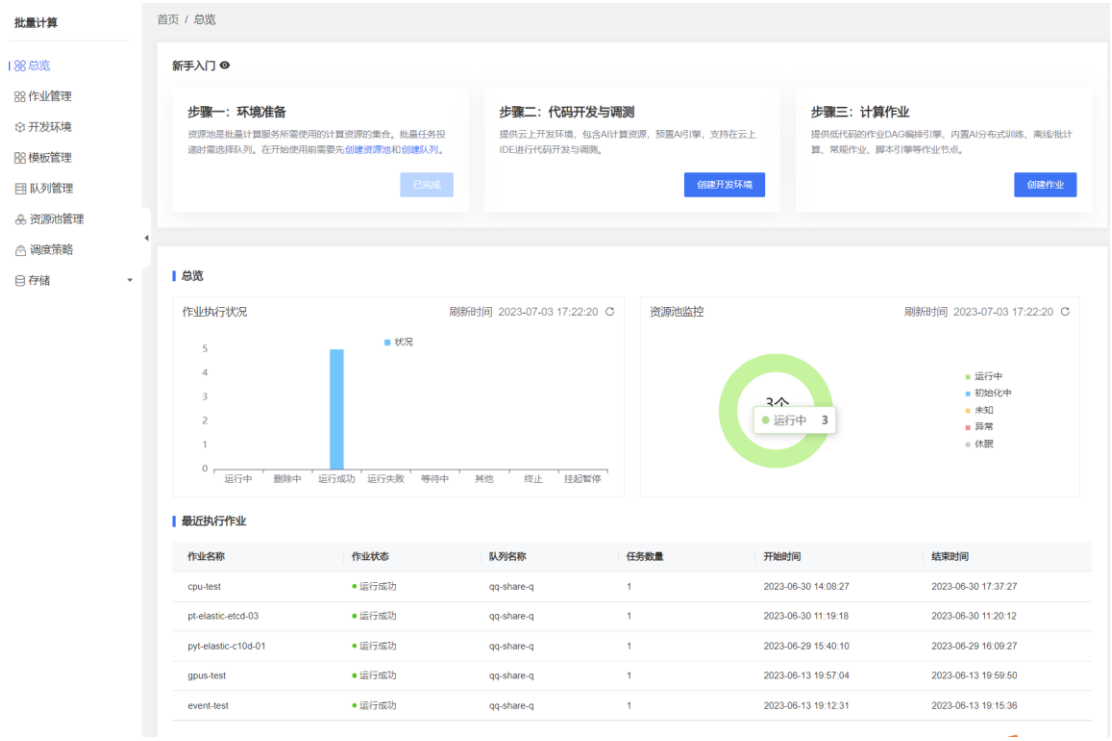
4.1 总览

在总览页面可查看作业的执行状况、资源池的监控信息和最近执行作业等信息。

操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击“总览”。

2. 在总览页面，可查看作业的执行状况、资源池的监控信息和最近执行作业。



4.2 资源池管理

资源池是批量计算服务所需要使用的计算资源的集合。批量计算的资源池由 ECK 和 ECX 提供，分为共享资源池、专属资源池。

4.2.1 专属资源池

专属资源池由边缘容器集群 ECK 支撑，ECK 提供高可靠高性能的企业级容器应用管理服务。使用边缘容器集群，您可以管理集群和节点资源。

4.2.1.1 创建专属资源池

前提条件

1. 通过边缘容器集群 ECK 创建 K8s 集群。
2. 确保以上 K8s 集群状态和网络通信正常。

操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。

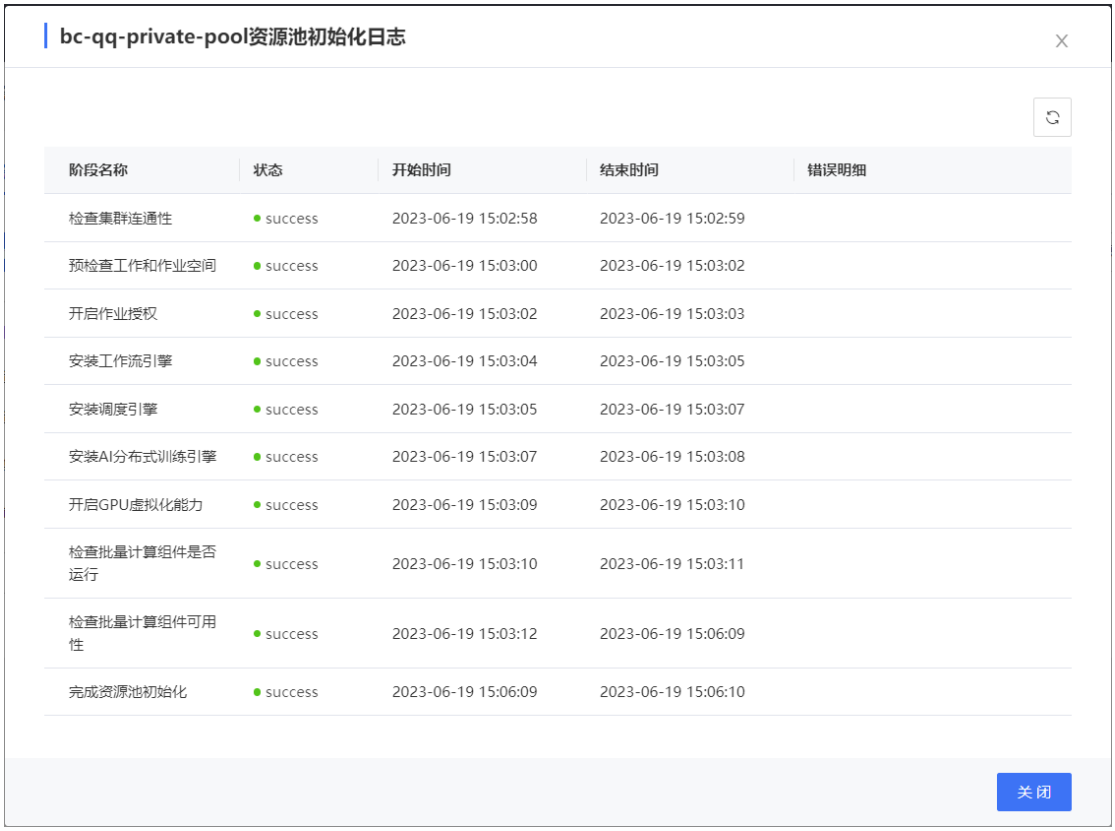
2. 在【专属资源池】页签下，在左上方单击【创建专属资源池】。

3. 在【创建专属资源池】页面中，填写基础信息，并选择资源池所在的集群，具体配置如下表所示

参数	说明
资源池名称	填写资源池名称，默认以“bc-”为前缀。
资源池描述	填写资源池的描述信息，资源池描述由除 <> 以外的字符组成，且长度为 0-1024 个字符。
选择集群	• 如果您在 ECK 中已有可用集群，则选择对应的集群。 • 如果您在 ECK 中没有可用的集群，或不想使用已有集群，请选择“创建集群”，跳转至边缘容器集群的页面进行创建，创建完成后返回本页面单击 即可选择新建的集群。

4. 在【创建专属资源池】页面中，单击右下角【立即创建】。

5. 单击【确认】后，将在该 K8s 集群安装批量计算调度插件，可点击资源池卡片中的【日志】按钮，查看安装进度。



4.2.1.2 管理专属资源池

创建专属资源池后，用户可执行查看、编辑和删除资源池的操作。

查看专属资源池

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。
2. 在【专属资源池】页签下，可查看资源池信息。

3. 点击资源池名称，可跳转至 ECK 集群详情。

编辑专属资源池

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。

2. 在【专属资源池】页签下，展示用户当前所有专属资源池列表。

3. 选择要操作的资源池，单击【编辑】。

4. 在【编辑专属资源池】页面下，可修改资源池描述信息及调度策略。若没有可选的调度策略，可点击【创建调度策略】按钮，创建自定义调度策略。

5. 在【编辑专属资源池】页面下，单击右下角【确认编辑】即可完成编辑。

删除专属资源池

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。

2. 在【专属资源池】页签下，点击目标资源池卡片右下角的【删除】按钮，在二次确认的弹框输入框中输入“确认删除 “，并且点击【确认】完成删除。弹框【删除资源池】页面，如不勾选“强制删除”删除资源池之前请先删除与之关联的队列；勾选“强制删除”将同时删除资源池、资源池关联的队列和队列中的任务。

4.2.2 共享资源池

共享资源池底层集群和服务器由批量计算平台进行维护和支撑。让您无需创建和管理服务器集群即可直接运行作业。如果作业是成熟稳定的，建议您使用共享资源池，可以省去对资源的关注。

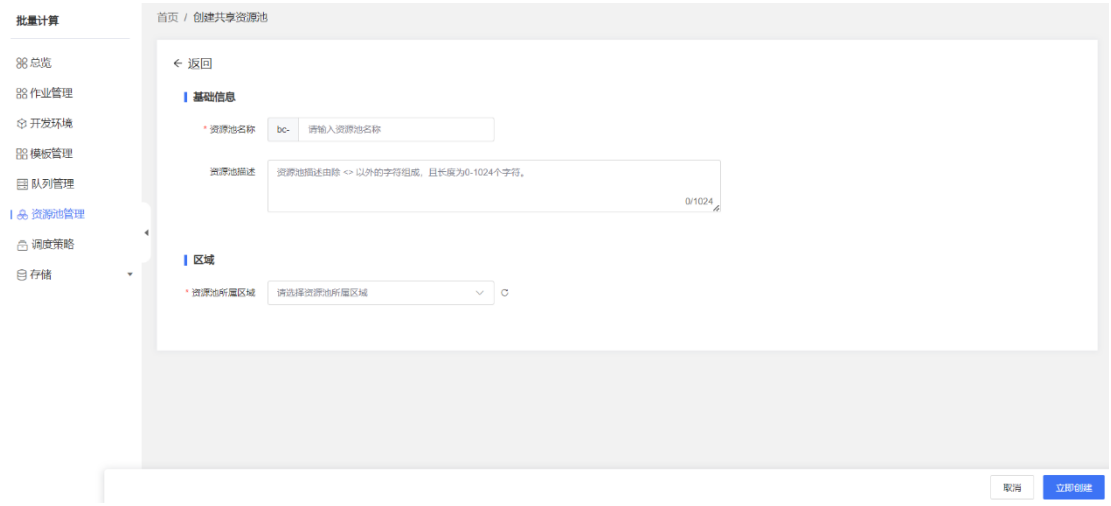
4.2.2.1 创建共享集群

操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。
2. 在【资源池管理】页面下，选择【共享资源池】。
3. 在【共享资源池】页面中，单击右上角【创建共享资源池】。

4. 在【创建共享资源池】页面中，输入资源池信息，选择资源池区域。

5. 在【创建共享资源池】页面中，单击右下角【立即创建】。



4.2.2.2 管理共享资源池

创建共享资源池后，用户可执行编辑、删除资源池的操作。

编辑专属资源池

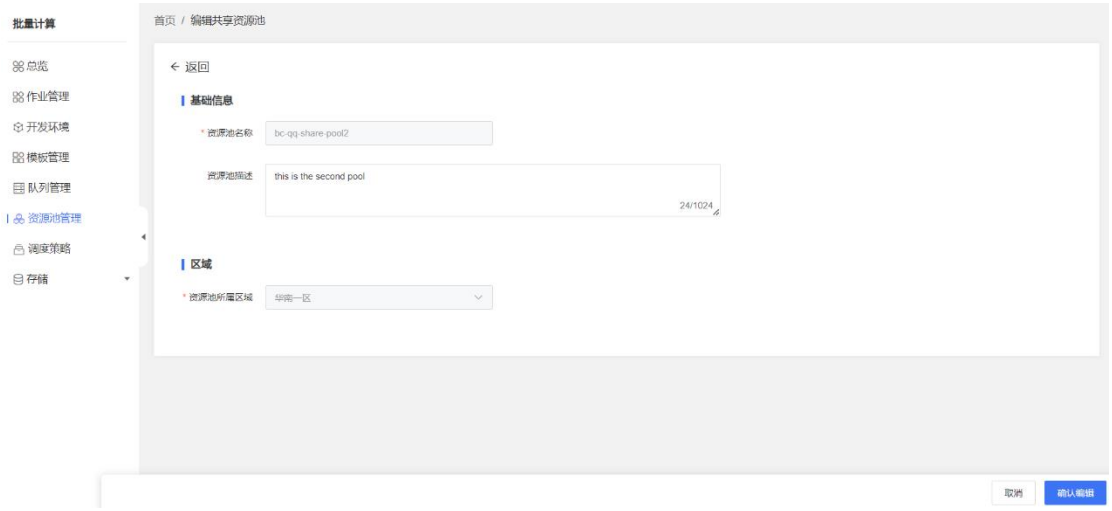
1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。

2. 在【共享属资源池】页签下，展示用户当前所有共享资源池列表。

3. 选择要编辑的资源池，单击【编辑】。

4. 在【编辑共享资源池】页面下，可修改资源池描述信息。

5. 在【编辑共享资源池】页面下，单击右下角【确认编辑】。



删除专属资源池

1. 登录批量计算管理控制台，在左侧导航栏单击【资源池管理】。

2. 在【共享资源池】页签下，展示用户当前所有共享资源池列表。

3. 选择要删除的资源池，单击【删除】。

4. 弹框【删除资源池】页面，如不勾选“强制删除”删除资源池之前请先删除与之关联的队列；勾选“强制删除”将同时删除资源池、资源池关联的队列和队列中的任务。在二次弹窗页面输入“确认删除”，单击右下角【确认】完成删除操作。

4.3 队列管理

队列管理可以查看用户创建的队列，用户投递的任务会被相应的队列接收。

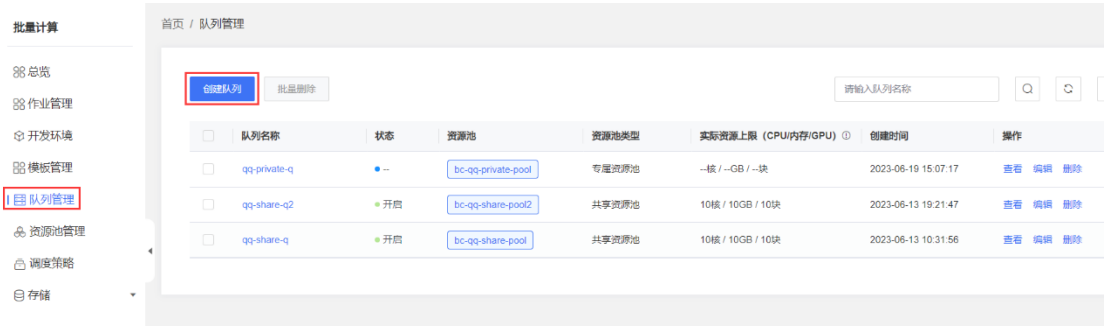
4.3.1 创建队列

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【队列管理】。

3. 在【队列管理】页面中，点击左上角的【创建队列】按钮。



4. 在【创建队列】页面中，配置参数，具体如下表所示。

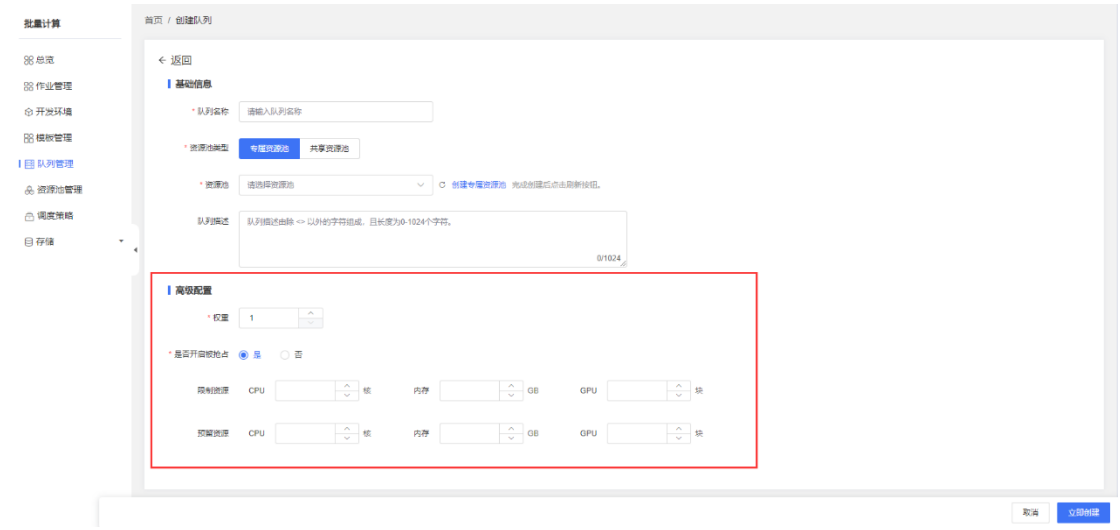
【基础信息】:

基础信息参数	说明	是否必选
队列名称	输入作业名称。要求如下： • 长度范围为 4~50 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 • 以小写字母或数字结尾。	是
资源池类型	选择资源池类型，选择“专属资源池”或者“共享资源池”的资源池类型。	是
资源池	在选择资源池类型后，再选择资源池，如果还未创建资源池，可单击“创建专属/共享资源池”进行创建，具体操作请参见资源池管理。	是
队列描述	选填，输入对该队列的描述。要求如下： • 由<>以外的字符组成。 • 长度为 0~1024 个字符组成。	否

【高级配置】:

- 专属资源池：

高级配置参数	说明	是否必选
权重	输入大于 0 的整数，默认值为 1。权重表示该队列在专属资源池中，所占资源的比重。	是
是否开启被抢占	选择“是”：表示若使用了超出该队列在专属资源池中所占比例的资源时，且资源池队列不足时，将被回收超出的部分资源；选择“否”，表示若使用了超出该队列在专属资源池中所占比例的资源时，且资源池队列不足时，不会被回收超出的部分资源。	是
限制资源	该队列的资源的上限，队列中的正在运行的各种资源累计不能超过限制资源的上限，若应用超出该上限，超出部分的资源将不会被调度成功。可填资源为“CPU”核心数（核）、内存（Gi）和“GPU”数量（块），保留小数点后两位为有效数字。	否
预留资源	该队列的资源的预留资源，资源池中将为该队列预留相关资源，预留资源将不会被其他队列占据。可填资源包括“CPU”核心数（核）、内存（Gi）和“GPU”数量（块）保留小数点后两位为有效数字 。	否



- 共享资源池：

高级配置参数	说明	是否必选
资源上限	该队列的资源限制，不同的专属资源池中配有不同规格的最大资源上限。可填资源包括“CPU”核心数（核）、内存（Gi）和“GPU”数量（块），保留小数点后两位为有效数字。	是

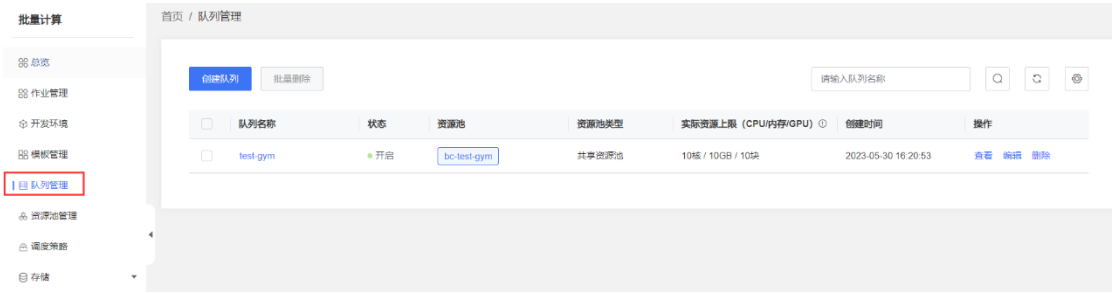
5. 填写必要的参数之后，单击右下角的【立即创建】，即可创建队列。

4.3.2 查看队列列表

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【队列管理】。



3. 此时展示的页面即为【队列列表】页面。以下【队列列表】展示的字段说明：

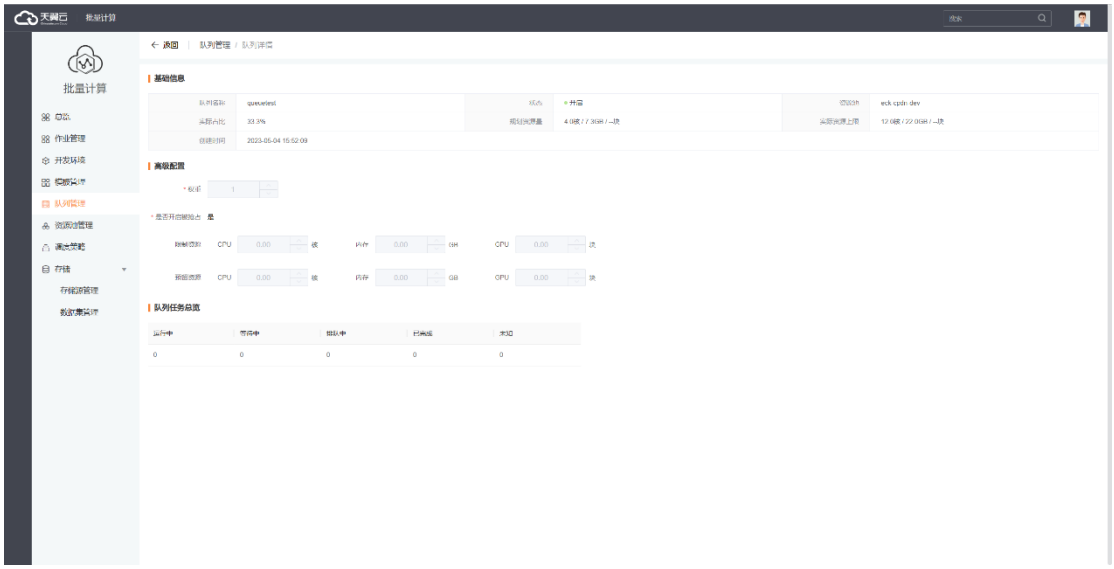
参数	说明
搜索框	输入队列名称，支持输入部分名称和全名称进行检索。
检索	根据输入队列名称进行检索筛选。
刷新	刷新当前队列列表展示页面。
选项设置	可勾选不同的字段，对列表页面进行筛选。
队列名称	队列的名称。
状态	队列的状态，共有 4 种，分别为：开启、关闭、关闭中以及未知。
资源池	队列所属的资源池的名称。
资源池类型	队列所属的资源池的类型，共有 2 种，分别为专属资源池和共享资源池。

实际占比	队列在所属资源池中所占据的比重，计算公式为：实际占比=该队列权重/（该队列权重+资源池已有的其他队列权重总和）。
规划资源量（CPU/内存/GPU）	队列在所属资源池中所规划得到的资源量，计算公式为：规划资源量=该队列的权重/所属资源池所有队列权重总和）*资源池总资源
实际资源上限（CPU/内存/GPU）	队列在所属资源池中能得到的实际资源量，计算公式为：实际资源上限=该队列所属资源池下所有的资源 - 其他所有队列的预留资源
是否开启被抢占	表示该队列在资源使用量超过该队列所应得的资源份额时，是否允许其他队列回收该队列使用超额的资源，默认值为“是”即允许回收 。
创建时间	队列的创建时间。
操作	查看：详见 3.3.3 编辑：详见 3.3.4 删除：详见 3.3.5

4.3.3 查看队列详情

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【队列管理】。
3. 单击队列名称或者右侧的【查看】按钮，即可查看队列的详细信息。以下为【队列详情】页面展示的字段说明：
   - 专属资源池的队列：



【基础信息】:

参数	说明
队列名称	队列的名称。
状态	队列的状态，共有 4 种，分别为：开启、关闭、关闭中以及未知。
资源池	队列所属的资源池的名称。
实际占比	队列在所属资源池中所占据的比重，计算公式为：实际占比=该队列权重/（该队列权重+资源池已有的其他队列权重总和)。
规划资源量	队列在所属资源池中所规划得到的资源量，计算公式为：规划资源量=该队列的权重/所属资源池所有队列权重总和）*资源池总资源
实际资源上限	队列在所属资源池中能得到的实际资源量，计算公式为：实际资源上限=该队列所属资源池下所有的资源 - 其他所有队列的预留资源
创建时间	队列的创建时间。

【高级配置】:

参数	说明
权重	队列在资源池中所占的权重。
是否开启被抢占	表示该队列在资源使用量超过该队列所应得的资源份额时，是否允许其他队列回收该队列使用超额的资源，默认值为“是”即允许回收 。
限制资源	该队列的资源的上限，队列中的正在运行的各种资源累计不能超过限制资源的上限，若应用超出该上限，超出部分的资源将不会被调度成功。
预留资源	该队列的资源的预留资源，资源池中将为该队列预留相关资源，预留资源将不会被其他队列占据。

【队列任务总览】:

参数	说明
运行中	作业在该队列中创建的强关联 Pod 组合处于运行中状态的数量。
等待中	作业在该队列中创建的强关联 Pod 组合处于等待中状态的数量。
排队中	作业在该队列中创建的强关联 Pod 组合处于排队中状态的数量。
已完成	作业在该队列中创建的强关联 Pod 组合处于已完成中状态的数量。
未知	作业在该队列中创建的强关联 Pod 组合处于未知状态的数量。

- 共享资源池的队列：

【基础信息】:

参数	说明
队列名称	队列的名称。
状态	队列的状态，共有 4 种，分别为：开启、关闭、关闭中以及未知。
资源池	队列所属的资源池的名称。
创建时间	队列的创建时间。

【高级配置】:

参数	说明
资源上限	该队列的资源的上限，队列中的正在运行的各种资源累计不能超过限制资源的上限，若应用超出该上限，超出部分的资源将不会被调度成功。

【队列任务总览】:

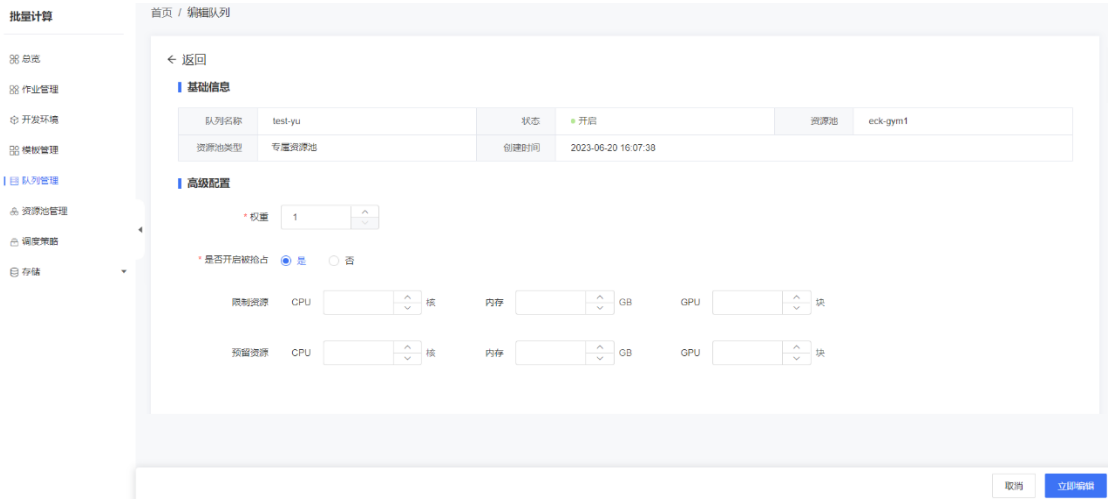
参数	说明
运行中	作业在该队列中创建的强关联 Pod 组合处于运行中状态的数量。
等待中	作业在该队列中创建的强关联 Pod 组合处于等待中状态的数量。
排队中	作业在该队列中创建的强关联 Pod 组合处于排队中状态的数量。
已完成	作业在该队列中创建的强关联 Pod 组合处于已完成中状态的数量。
未知	作业在该队列中创建的强关联 Pod 组合处于未知状态的数量。

4.3.4 编辑队列

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【队列管理】。
3. 单击队列右侧的【编辑】，即可对该队列进行编辑操作，队列名称无法更改，其他操作与创建队列操作相同。

- 专属资源池的队列的【编辑队列】页面字段说明如下：



【基础信息】:

参数	说明
队列名称	队列的名称。
状态	队列的状态，共有 4 种，分别为：开启、关闭、关闭中以及未知。
资源池	队列所属的资源池的名称。

实际占比	队列在所属资源池中所占据的比重，计算公式为：实际占比=该队列权重/（该队列权重+资源池已有的其他队列权重总和）。
规划资源量	队列在所属资源池中所规划得到的资源量，计算公式为：规划资源量=该队列的权重/所属资源池所有队列权重总和）*资源池总资源
实际资源上限	队列在所属资源池中能得到的实际资源量，计算公式为：实际资源上限=该队列所属资源池下所有的资源 - 其他所有队列的预留资源
创建时间	队列的创建时间。

【高级配置】:

参数	说明
权重	队列在资源池中所占的权重。
是否开启被抢占	表示该队列在资源使用量超过该队列所应得的资源份额时，是否允许其他队列回收该队列使用超额的资源，默认值为“是”即允许回收 。
限制资源	该队列的资源的上限，队列中的正在运行的各种资源累计不能超过限制资源的上限，若应用超出该上限，超出部分的资源将不会被调度成功。
预留资源	该队列的资源的预留资源，资源池中将为该队列预留相关资源，预留资源将不会被其他队列占据。

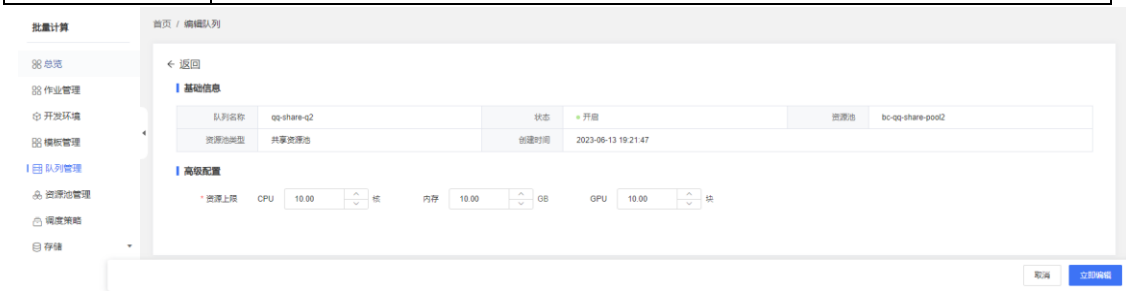
- 共享资源池的队列的【编辑队列】页面字段说明如下：

【基础信息】:

参数	说明
队列名称	队列的名称。
状态	队列的状态，共有 4 种，分别为：开启、关闭、关闭中以及未知。
资源池	队列所属的资源池的名称。
创建时间	队列的创建时间。

【高级配置】:

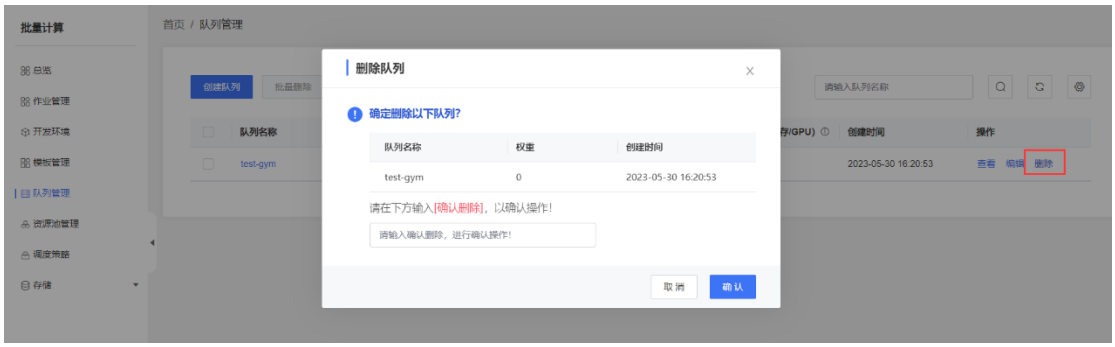
参数	说明
资源上限	该队列的资源的上限，队列中的正在运行的各种资源累计不能超过限制资源的上限，若应用超出该上限，超出部分的资源将不会被调度成功。



4.3.5 删除队列

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【队列管理】。
3. 选择一个队列，在右边单击【删除】。
4. 在二次确认的弹框输入框中输入“确认删除“，并且点击【确认】完成删除。队列下没有作业任务和开发机任务时才可删除成功。



4.4 作业管理

4.4.1 作业类型

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 在【作业管理】页面中，单击左上角的【创建作业】。
4. 在【创建作业】页面中，左侧可看到支持的多种任务类型，具体支持的类型如下表所示。

大类	任务类型	描述
AI 分布式训练	Tensorflow	Tensorflow 任务是一种基于 Tensorflow 开源框架的 kubernetes 自定义资源类型，多种角色可以配置，可更简单地实现 Tensorflow 的单机或分布式训练
	Pytorch	Pytorch 任务是一种基于 Pytorch 深度学习框架的 kubernets 自定义资源类型，在机器学习和其他数学密集型应用有广泛应用，Pytorch 任务支持 Master-Worker 模式，也支持弹性自动扩缩容模式（自动选主）
	Paddle	飞桨 PaddlePaddle,国产深度学习平台,是基于业务实践打造的千亿规模参数超大规模并行训练框架，PaddlePaddle 任务支持 Master-Worker 模式，也支持弹性自动扩缩容模式（自动选主）。
常规作业	Container-Job	Container-Job 即 K8s 中的 job 类型工作负载，负责批量处理短暂的一次性任务（short lived one-off tasks），即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束，可以用来完成数据集下载，预处理或模型上传等任务
	VM-Job	VM-Job 即以虚拟机的形式运行任务。
高性能计算	OpenMPI	MPI 任务是一种高性能大规模并行计算框架，OpenMPI 是一个强大且广泛使用的 MPI 实现。

4.4.1 创建作业

前提条件

1. 作业投递的队列状态为“运行中”。

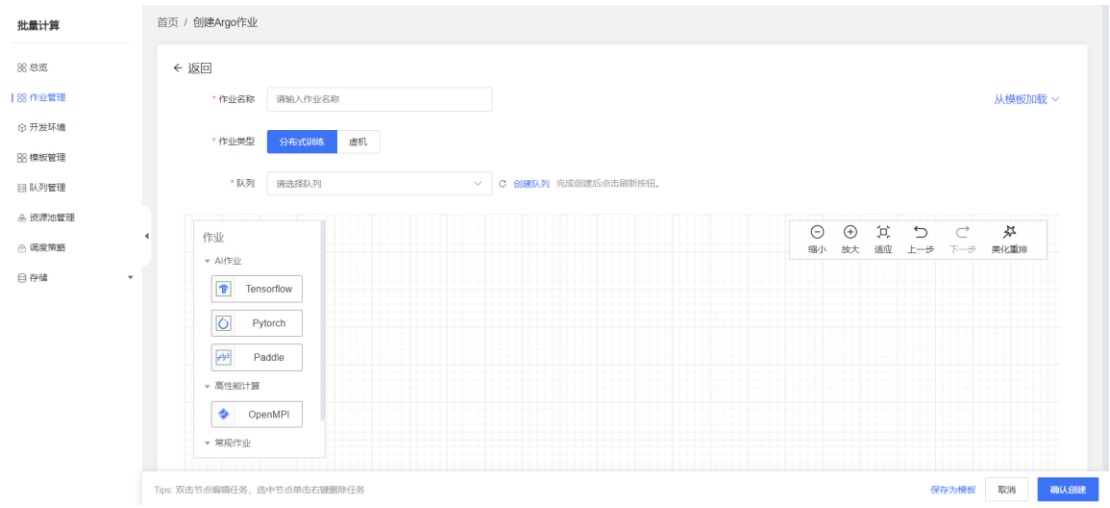
2. 队列配额充足。

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 在【作业管理】页面中，单击左上角的【创建作业】。

4. 在【创建作业】页面中，配置参数，具体如下表所示。

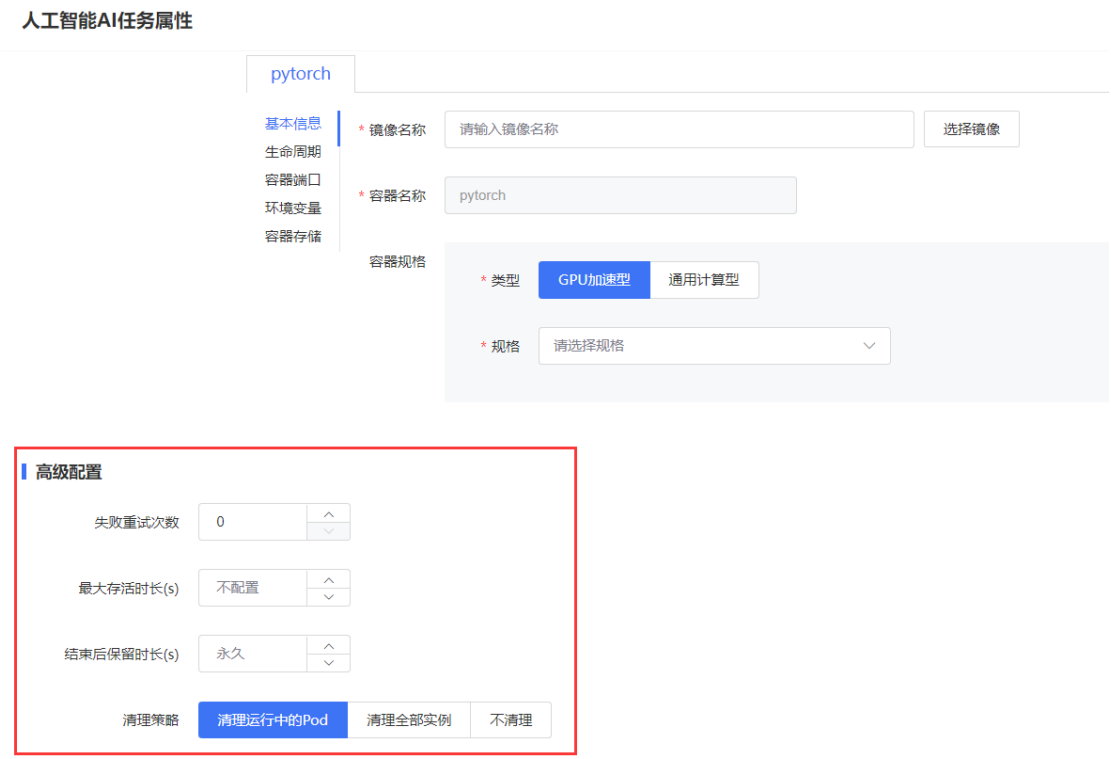
参数	说明
作业名称	输入作业名称。要求如下： • 长度范围为 4~32 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 以小写字母或数字结尾。
队列	选择队列，如果还未创建队列，可单击“创建队列”创建，具体操作请参见 3.3 队列管理。
任务组件	可将任务用鼠标拖动至画布中，任务间可以连线串起来，组成一个有向无环图(DAG)



5. 在画布中，双击任务名称，编辑任务，编辑完成后，单击“确定”。任务参数详情介绍如下：

- **通用任务参数**

通用参数用于控制任务执行过程中的失败重试策略，作业结束后的清理策略，是 OpenMPI，TensorFlow，PaddlePaddle，Pytorch 任务都具备的任务参数，具体参数如下：



失败重试次数： 任务失败后，会再次运行，直到达到重试上限，默认为 0 代表不重试

最大存活时长： 任务运行最长的时间，达到最大存活时间后任务会被终止

结束后保留时长： 任务运行结束后相应容器被保留的时长，如果任务结束后还需要查看日志，请选择适当的保留时长。

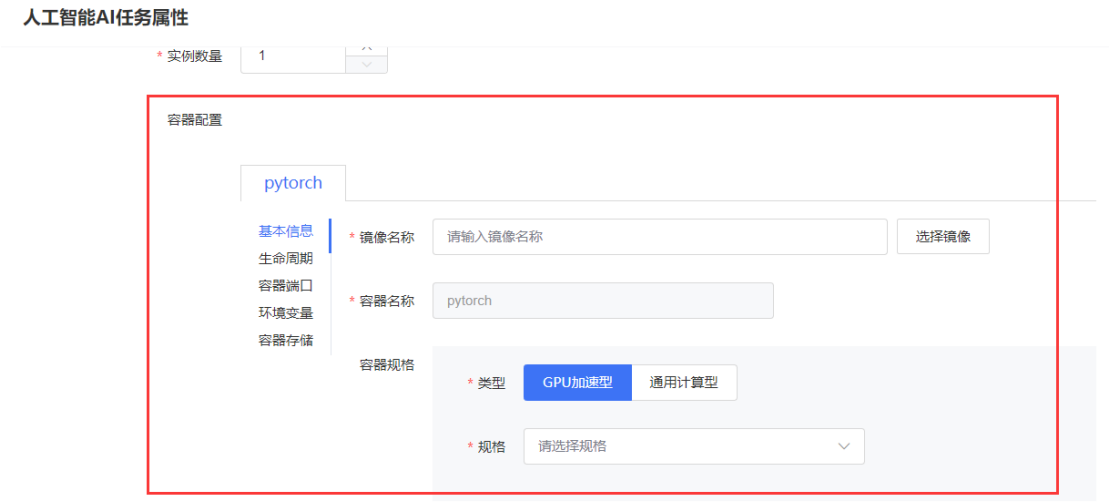
清理策略： 这里用于控制 pod 的清理策略，可以根据任务情况选择不同的清理策略

1. 不清理： 会保留所有相关的 pod，更方便排查任务失败原因，但是会继续占用资源，

可能产生额外的资费（容器处于运行时会占用资源，结束状态不会占用资源）

2. 清理全部实例： 会删除所有的 pod，资源彻底释放
3. 清理运行中的 Pod： 分布式任务运行时，通常有一个 Master 角色，用于控制整个分布式任务的训练过程，在 Master 容器成功运行后，通常代表着任务成功了。Worker 角色有时是启动后挂起等待 Master 的指令，这时候 Worker 是没有常规的退出动作的，可以配置清理运行中的 pod，确保任务结束后，不会再占用资源，同时可以查看 Master 上的日志和相关信息。

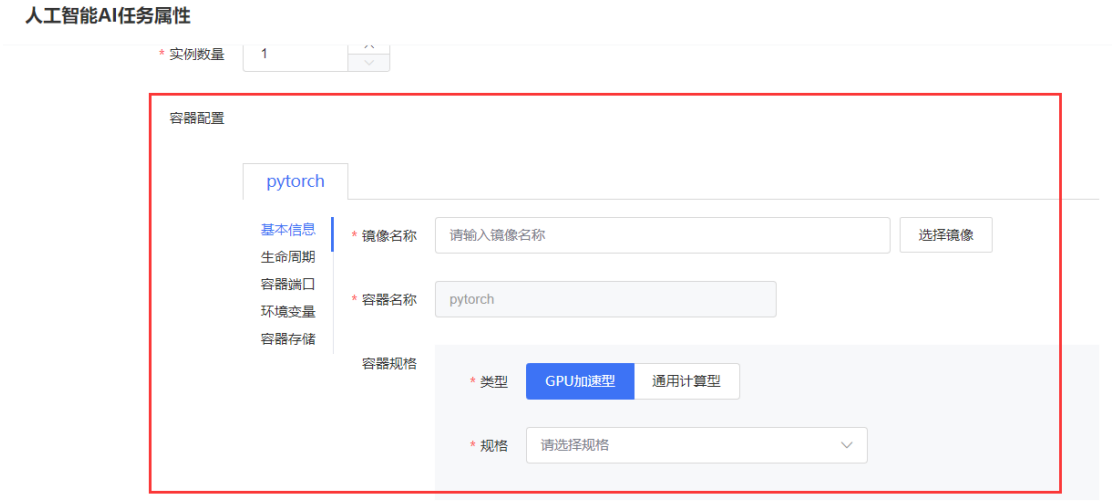
- **通用 Pod（容器）参数**



模块	参数名	说明	是否可选
基本信息	镜像名称	镜像的地址，可以手动填写第三方镜像，也可以选择存储在天翼云的镜像	必填
	容器规格	选择运行时容器的资源规格，请根据具体训练任务选择合适的资源配置，避免任务因为资源不足而失败（内存和显存是不可压缩资源，重点关注）	必填

生命周期	启动命令	容器启动后要执行的命令，分为启动命令和运行参数，系统会自动将启动命令和运行参数拼接为具体执行的参数。ls -la 命令的正确填入姿势为：命令填入“ls”，参数处填入“-la”	非必填
	启动后处理	用于执行启动后的一些脚本，执行顺序在容器创建后，主进程启动后。	非必填
	停止前处理	用于做停止前的善后处理，在容器收到删除 pod 事件前执行，然后才会执行容器终止的指令	非必填
容器端口	端口名称	• 长度范围为 4~32 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 以小写字母或数字结尾。 同一个 pod 内不可以重复	如果添加了容器端口选项，就是必填，否则可选
	容器端口	1-65535，通常 1000 以上的整数，具体看容器需要	同上
	端口协议	TCP/UDP，默认为 TCP，根据实际选择	同上
环境变量	变量名称	长度范围为 255，字母开头，同一个 pod 内不可以重复	非必选，添加环境变量选项后必选

	变量/变量引用	当前只支持固定的变量值	非必选
容器存储	存储类型	当前只支持 nfs，后续将扩充支持的存储类型	添加数据集选项后必选
	挂载数据集	是个下拉列表，具体见左侧数据集 Tab 页面	同上
	容器挂载路径	数据集要挂载到容器里面的哪个目录	同上



- **Tensorflow 任务参数**

TensorFlow 分布式训练有以下几种角色：

1. PS：参数服务器，保存各 worker 最新的参数,提供参数同步，可以有多个

2. Worker: 执行前向和反向传播计算,上传最新参数至 chief，可以有多个

3. Chief: 初始化全局参数,然后广播给所有的 worker；保存检查点和事件；保存最终的模型等，只能有一个

4. evaluator： 对最新的模型参数进行评估

PS 和 Chief 的差异如下

chief:用于协调全局训练过程,主要职责是:

- 初始化全局参数并广播给 worker

- 接收 worker 上传的最新参数并维护全局最新参数

- 保存检查点文件和事件文件

- 选取最优超参数

- 保存最终的模型参数

ps:用于参数服务器,主要职责是:

- 接收各个 worker 计算得到的最新参数

- 聚合各个 worker 的参数,生成全局最新的参数值

- 为 worker 和 chief 提供最新的全局参数值

所以,chief 和 ps 的主要差异在于:

1. 职责不同

chief 主要用于协调全局训练流程,维护最新最优的超参数和模型参数。ps 主要用于同步各个 worker 的参数,生成全局最新的参数状态。

2. ps 数量可多 chief 只有一个

3. ps 通常需更强硬件

由于 ps 负责参数同步和更新,它通常需要更高性能的 CPU、GPU 和网络来满足计算需求。而 chief 作为 coordinator,硬件要求会相对较低。

4. ps 无需保存训练信息和最终模型

ps 只需要同步最新的参数即可,无需保存检查点文件、事件文件和最终模型。这些信息由 chief 来维护。

所以总结来说,虽然 chief 和 ps 都发挥着重要作用,但其职责差异还是比较明显的:

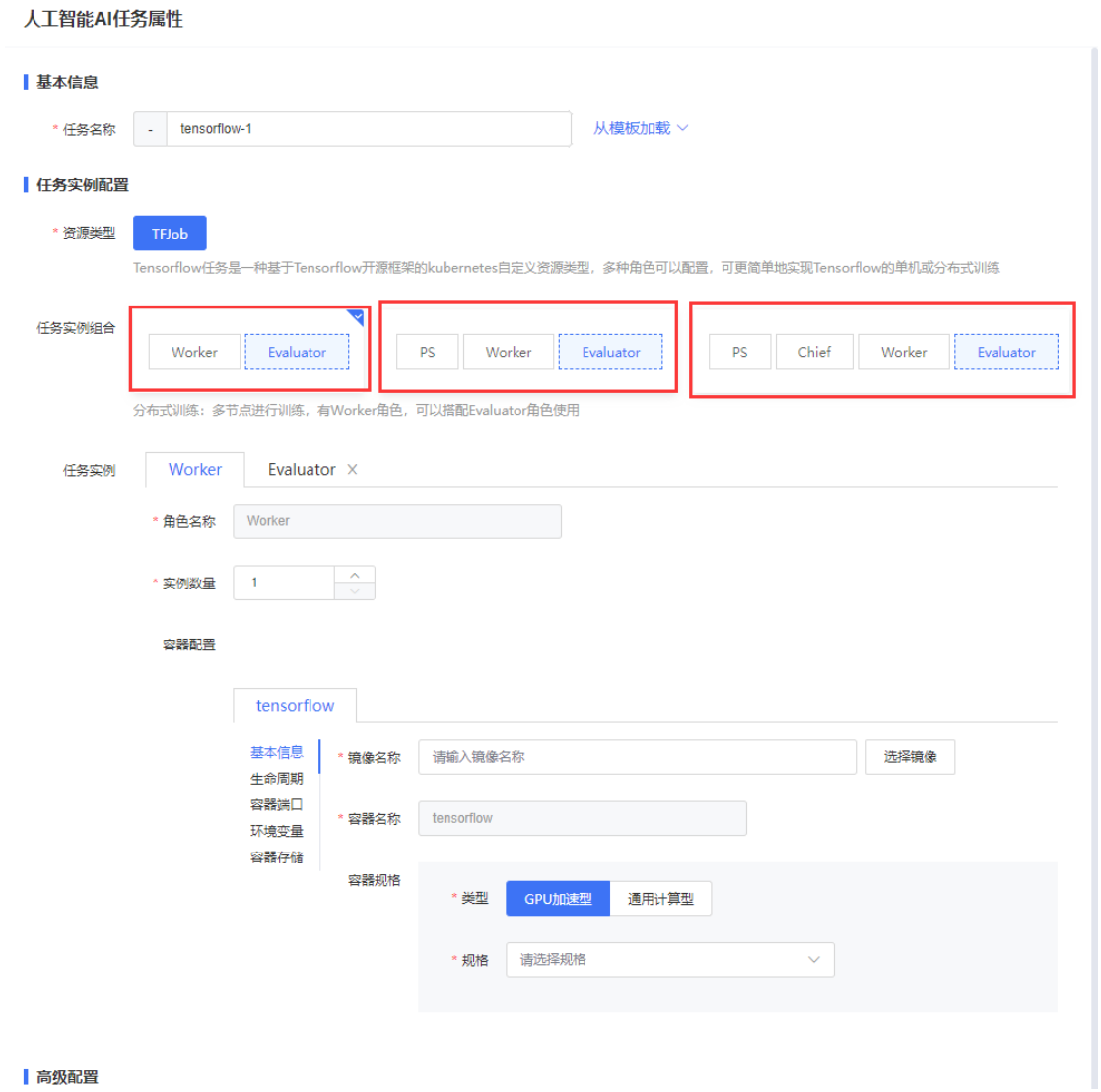
chief:负责全局训练协调与最终模型生成

ps:负责高效的参数同步与更新

二者相互配合,才能实现 TensorFlow 高性能的分布式深度学习训练。

【训练组合说明】

分组名	参数	说明
任务名称	长度范围为 4~32 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 以小写字母或数字结尾。	TensorFlow 任务的名称（选填）
任务实例组合	Worker+Evaluator	训练中的任务角色组合类型，单机训练可以选择 Worker+Evaluator 模式，其中 Evaluator 角色是可选的，
	PS+Worker+Evaluator	训练中的任务角色组合类型，分布式训练可以选择该模式，其中 Evaluator 角色是可选的
	PS+Chief+Worker+Evaluator	训练中的任务角色组合类型，分布式训练可以选择该模式，其中 Evaluator 角色是可选的



- **Pytorch 任务参数**

Pytorch 分布式训练中，有 Master 和 Worker 的角色分工，分工如下：

master:负责全局训练协调,主要职责是:

- 定义模型和优化器

- 初始化全局参数并广播给 worker

- 按轮次平均 worker 的梯度

- 根据 loss 选择最优超参数(如学习率)

- 保存检查点和最终模型

worker:负责前向和反向传播计算,主要职责是:

- 接收 master 下发的初始化参数和最新超参数

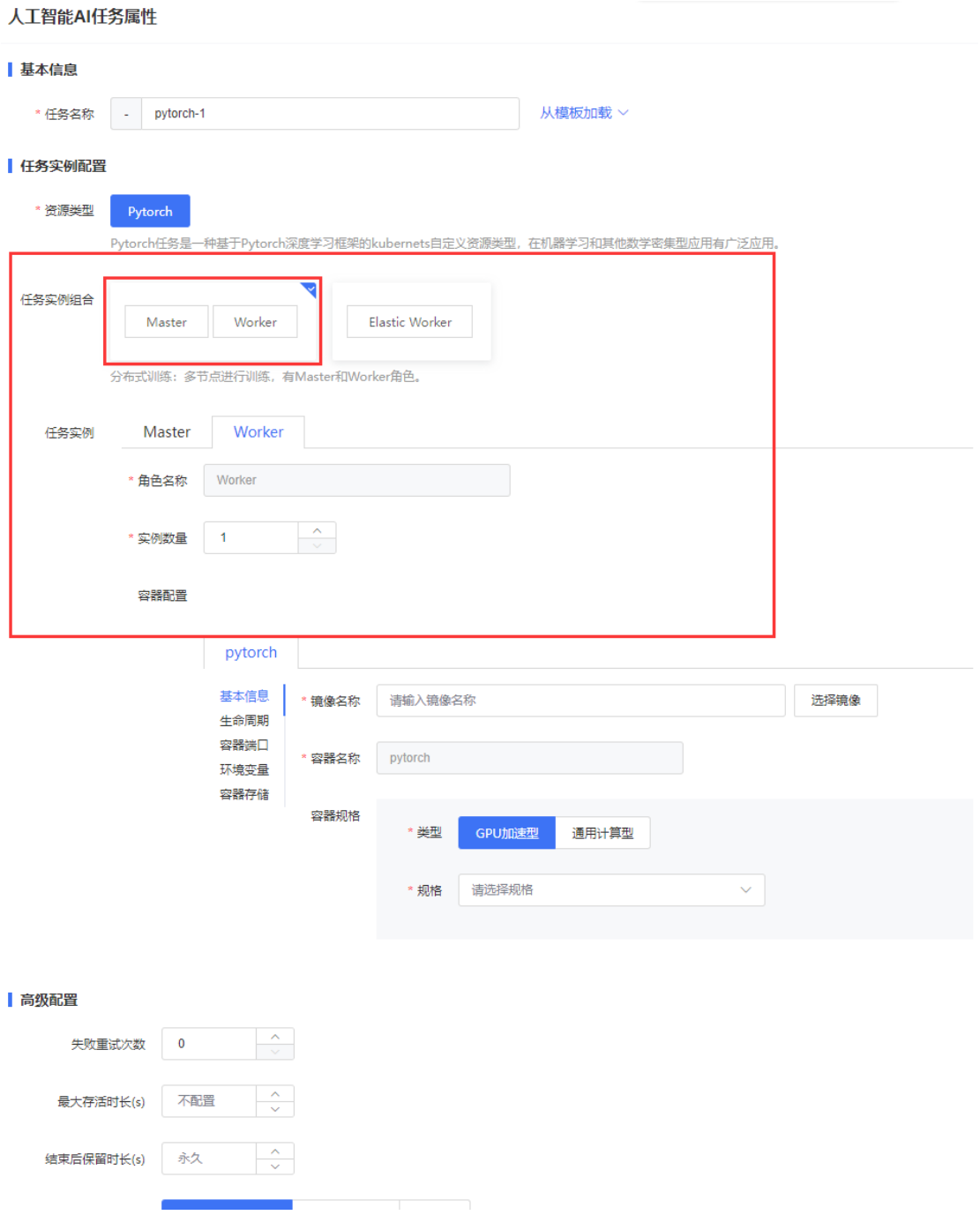
- 根据接收到的全局参数计算 loss 和梯度

- 将计算得到的梯度发送给 master

- 接收 master 同步最新的全局参数

模式	说明
Master+Worker	标准的分布式训练，Worker 数量固定，不同节点会有不一样的 rank，选择这个模式时，批量计算平台会自动给容器注入 torchrun 命令所需的以下环境变量： - MASTER_ADDR：master 节点的地址 - MASTER_PORT：master 节点开放的端口 - WORLD_SIZE：这个是节点总数，如果是多机多卡的任务，请重新为该变量赋值 - RANK：节点在所有节点中的排名
Elastic Worker	自动弹性扩缩容模式，可以在启动时设置一个最少副本数，会根据配置的资源使用率进行扩容，每次扩容会造成训练的暂停，并在节点间重新分配参数。该模式可以更好的使用资源进行计算，记得及时保存每轮迭代的训练结果，并在启动时加载模型，避免从零开始训练。Worker 启动时会选举出一个充当 master 的角色，弹性分布式训练启动时不需要关注 Master+worker 的相关参数。

Master+Worker 模式参数


人工智能AI任务属性
基本信息
* 任务名称
pytorch-1
从模板加载
任务实例配置
* 资源类型
Pytorch
Pytorch任务是一种基于Pytorch深度学习框架的kubernets自定义资源类型，在机器学习和其他数学密集型应用有广泛应用。
任务实例组合
Master
Worker
Elastic Worker
分布式训练：多节点进行训练，有Master和Worker角色。
任务实例
Master
Worker
* 角色名称
Worker
* 实例数量
1
容器配置
pytorch
基本信息
生命周期
容器端口
环境变量
容器存储
* 镜像名称
请输入镜像名称
选择镜像
* 容器名称
pytorch
容器规格
* 类型
GPU加速型
通用计算型
* 规格
请选择规格
高级配置
失败重试次数
0
最大存活时长(s)
不配置
结束后保留时长(s)
永久

弹性扩缩容模式

参数名	说明	是否必填
模式	Pytorch 默认实现 c10d，也可以选择 etcd 模式	必填，选择默认模式即可
指标	当前仅支持 cpu 使用率，后续会增加指标种类	必填，不用修改
期望值	资源的使用率，当资源使用率超过该值时会触发扩容	必填
最小副本数	训练时最少的副本数，当扩容失败后，会减少副本数，	必填

	直到该值	
最大副本数	训练时副本数的上限	必填
扩缩容超时取消时间	当集群资源不足时，扩容出来的副本可能无法运行，当等待时间超过该值时，会取消扩容	可选

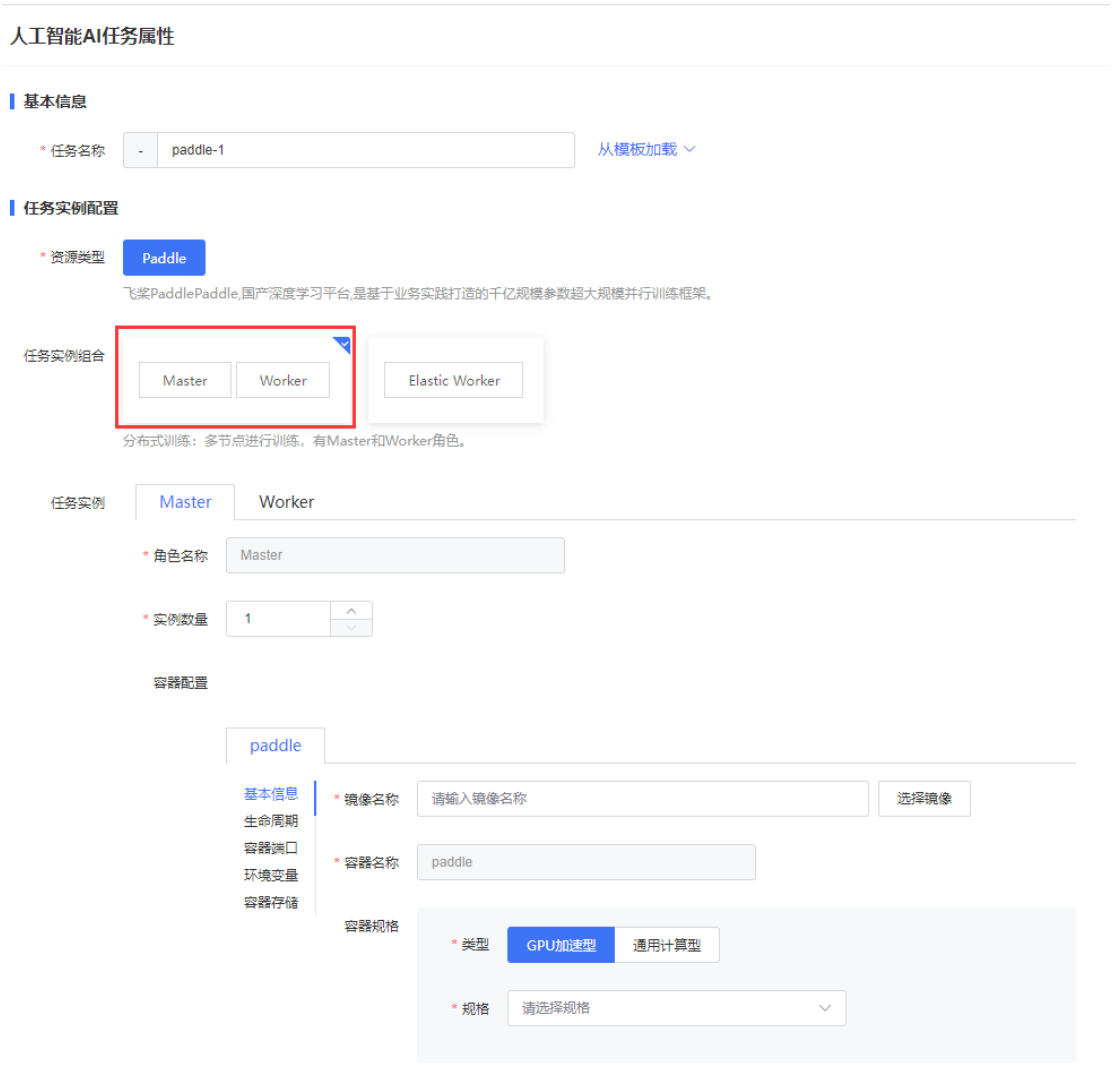
- **Paddle 任务参数**

Paddle 是百度推出的 AI 训练框架，在国内产业界应用广泛，对标的是 pytorch，训练模式同样支持 PS+Worker 模式，也支持弹性分布式训练

参数服务器（ParameterServer）模式采用了一种将模型参数中心化管理的方式来实现模型参数的分布式存储和更新。该模式下的节点/进程有两种不同的角色：

训练节点（Trainer/Worker）：该节点负责完成数据读取、从服务节点拉取参数、前向计算、反向梯度计算等过程，并将计算出的梯度上传至服务节点。

服务节点（Server）：在收到所有训练节点传来的梯度后，该节点会将梯度聚合并更新参数，供训练节点拉取进行下一轮的训练。

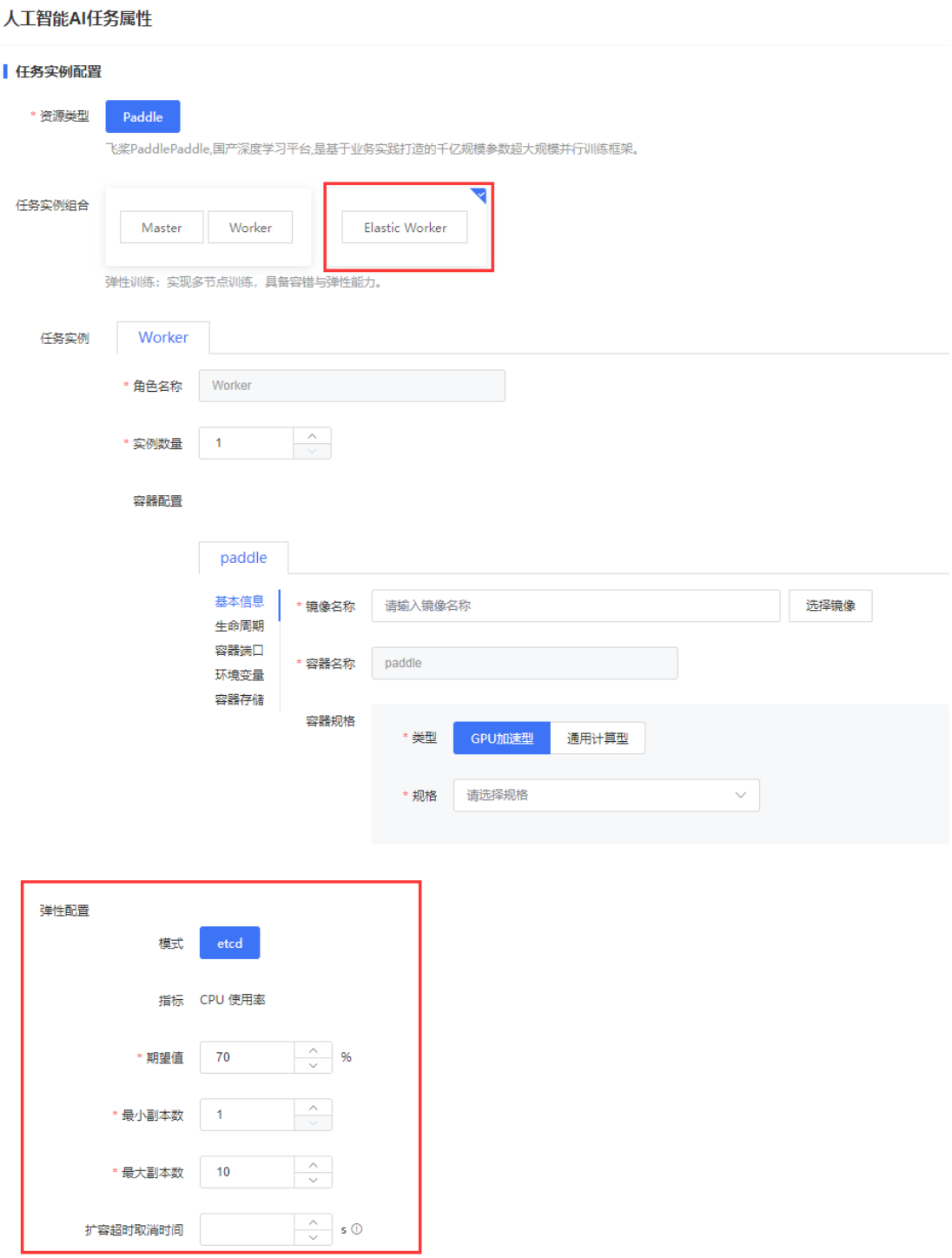


弹性模式

在分布式训练中，除了容错外，集群的资源剩余情况可能随时间而不同、任务的优先级也可能有不同， 基于这样的场景，实现弹性训练即任务可以在运行时动态调整训练资源而不影响或尽可能小地影响训练进程，能够最大限度地实现资源利用率提升同时提升训练任务质量。

paddle 目前已支持 Collective 训练模式基于热重启的弹性训练方案。热重启即用户的任务进程会被重启，所以需要用户代码中做好 checkpoint 逻辑，同时如 batchsize 和 learning rate 这样需要随节点数变化的参数也需要用户进程自动调整。参考样例：https://github.com/PaddlePaddle/PaddleFleetX/blob/old_develop/examples/resne

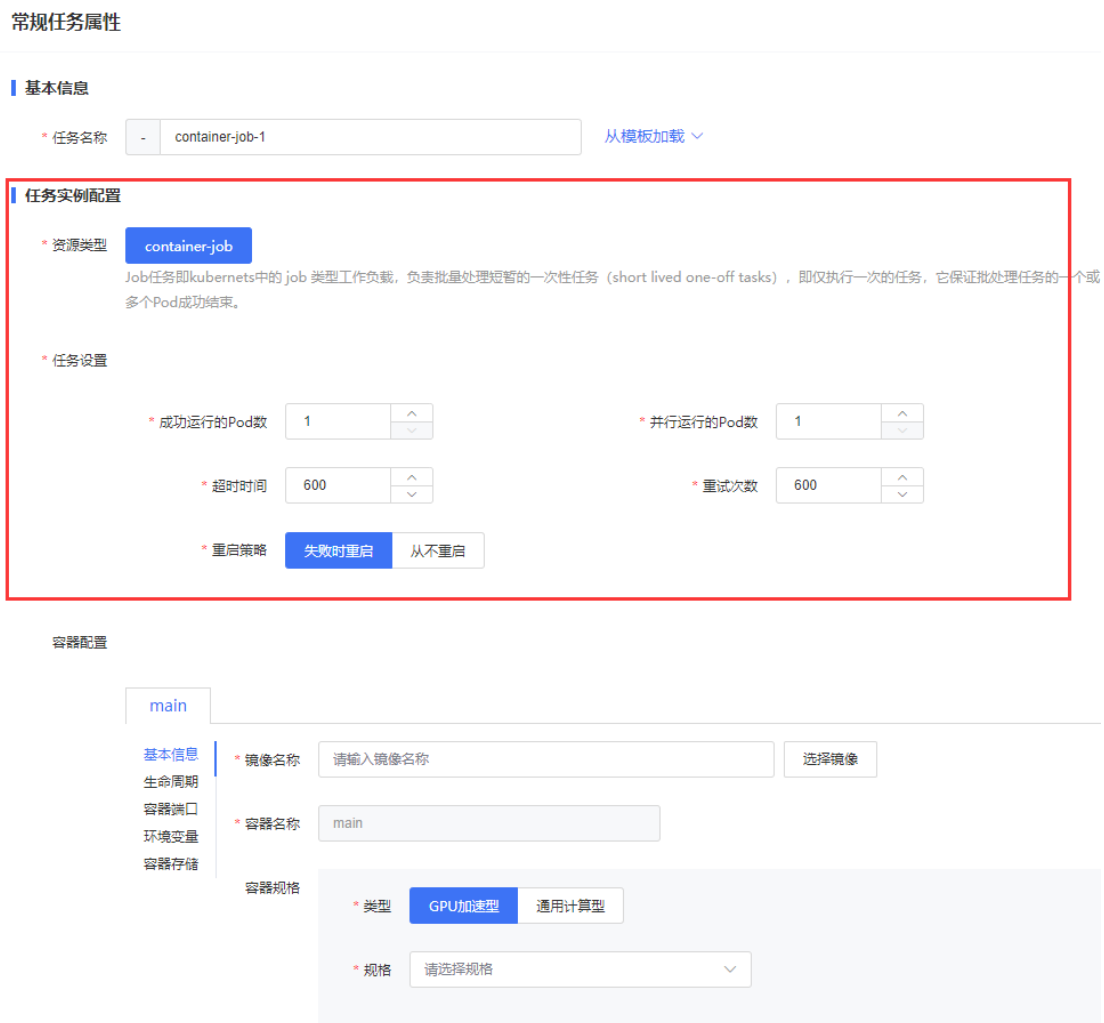
t/train_fleet_dygraph_ckpt.py



参数名	说明	是否必填
模式	Paddle 当前只支持 etcd 模式	必填
指标	当前仅支持 cpu 使用率，后续会增加指标种类	必填，不用修改

期望值	资源的使用率，当资源使用率超过该值时会触发扩容	必填
最小副本数	训练时最少的副本数，当扩容失败后，会减少副本数，直到该值	必填
最大副本数	训练时副本数的上限	必填
扩缩容超时取消时间	当集群资源不足时，扩容出来的副本可能无法运行，当等待时间超过该值时，会取消扩容	可选

- **Container-Job 任务参数**



参数	说明
运行成功的Pod数	任务要运行多少次
并行运行的	任务可以并行运行的数量

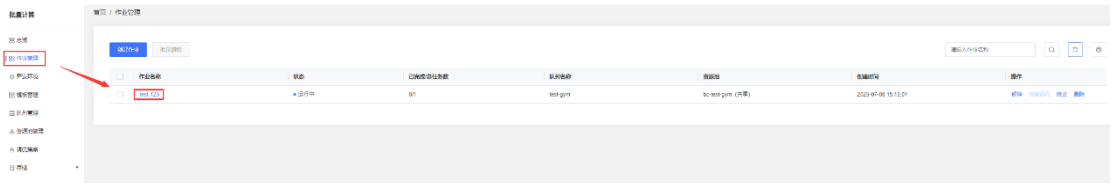
Pod 数

超时时间	任务执行的最长时长，单位秒，超过时长将被停掉重新运行，请设置足够长的值
重试次数	任务运行失败后，最多重试多少次
重启策略	任务在什么时候重试

4.4.2 查看作业详情

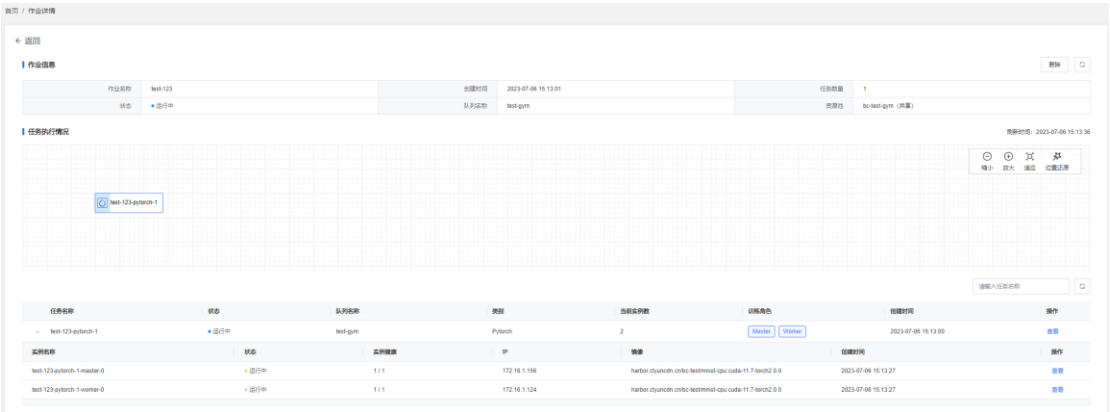
操作步骤

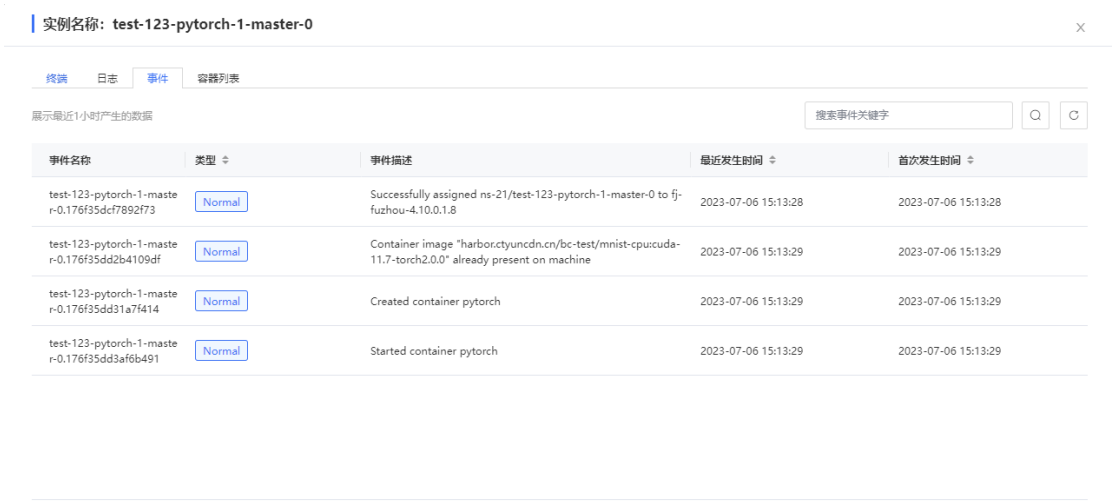
1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 单击作业名称，可查看作业的详细信息。



详情页面分为三部分：

1. 作业基本信息。
2. 作业执行情况，包含任务间的依赖关系，不同节点任务的执行情况。
3. 任务执行详情 包含任务创建出来的 Pod 的运行情况，镜像，创建时间等，点击 Pod 所在行，会弹出 Pod 的终端，日志和事件窗口。





小技巧

如果想在线调试分布式计算代码，可以选择给容器的启动命令设置为 sleep 365d，然后进入 pod 的终端执行 torchrun，进行代码调试。

4.4.3 暂停和恢复作业

作业执行过程中，可以进行暂停，暂停的生效范围是当前任务的后一个阶段会暂停，当前任务不受影响。

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 进入【作业管理】列表页，在需要停止的作业右侧操作栏单击【暂停】按钮。在弹框二次确认后，即可停止作业。

4. 在需要恢复运行的作业右侧操作栏单击【恢复运行】按钮。在弹框二次确认后，即可

恢复作业运行。

4.4.4 终止作业

作业执行过程中，可以进行终止，作业终止后无法再恢复运行。终止分为终止和强制终止两种终止模式，请根据实际情况选择。

- 终止：允许运行当前节点并执行用户定义的清理操作。
- 强制终止：直接强制删除，当前任务节点不再运行。

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 进入【作业管理】列表页，在需要终止的作业右侧操作栏单击【终止】按钮。在弹框二次选择终止模式并在输入框中输入“确认删除“，并且点击【确认】完成终止操作。

4.4.5 删除作业

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 进入【作业管理】列表页，在需要清理的作业列中单击“删除”，也可以选择多个作业，单击“批量删除”。

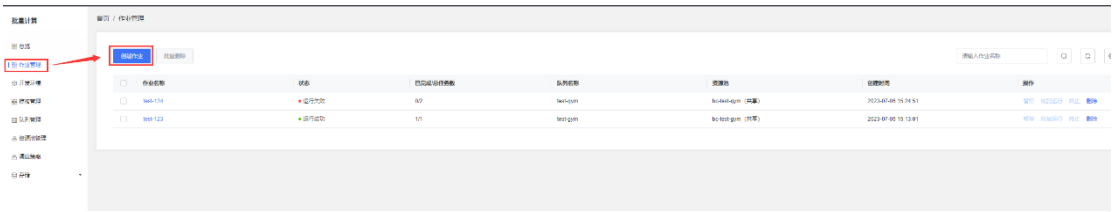
4. 在二次确认的弹框输入框中输入“确认删除“，并且点击【确认】完成删除。

4.5 模板管理

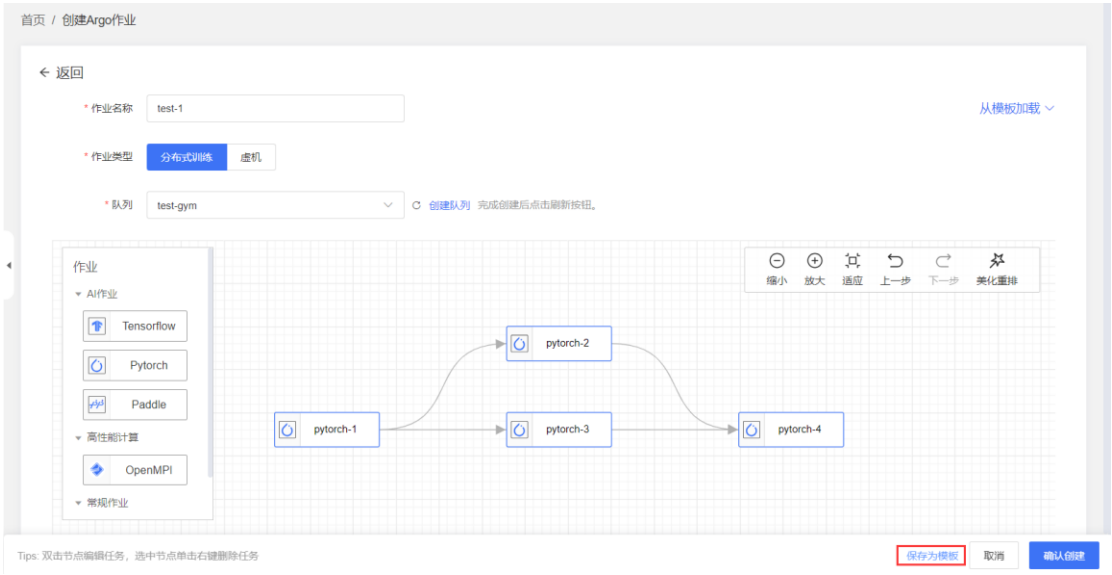
4.5.1 创建作业模板

操作步骤

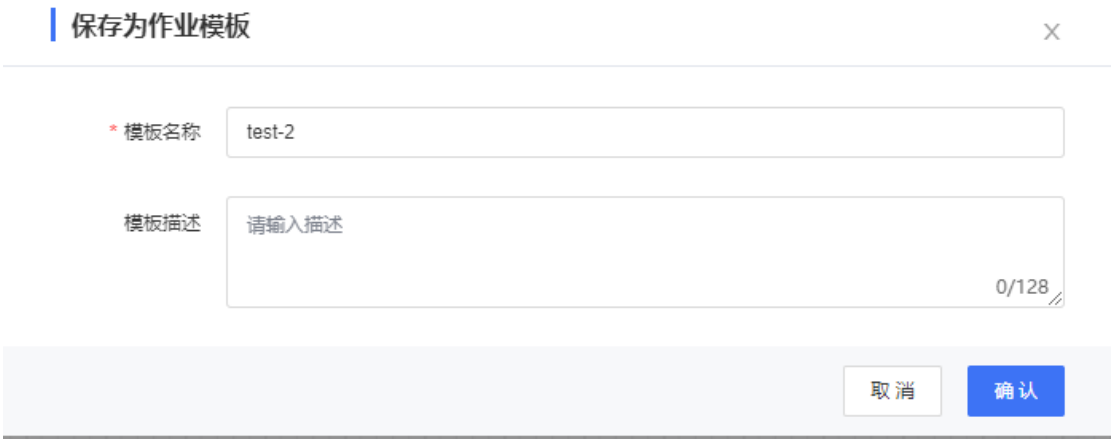
1. 登录批量计算管理控制台，在左侧导航栏单击【作业管理】。
2. 在【作业管理】页面下，单击右上角【创建作业】。



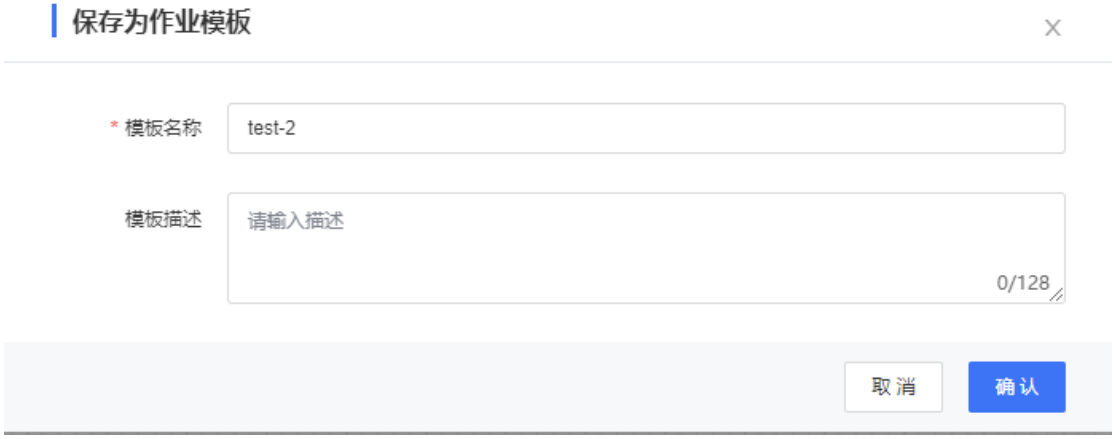
3. 在【创建 Argo 作业】页面下，输入作业名称，选择作业队列，拖动作业类型并配置作业。



4. 在【创建 Argo 作业】页面下，单击右下角【保存为模板】。



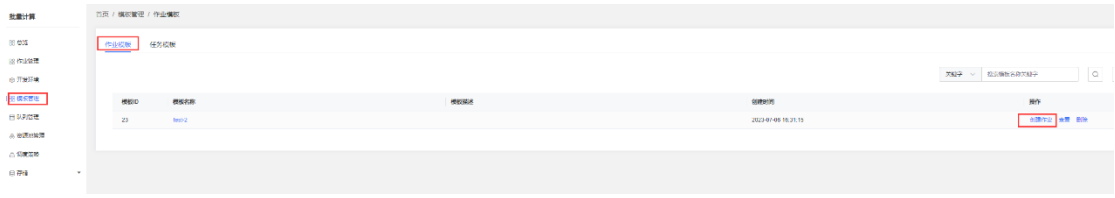
5. 在保存为作业模板弹窗中，填写模板名称和模板描述，点击【确认】，完成模板保存。



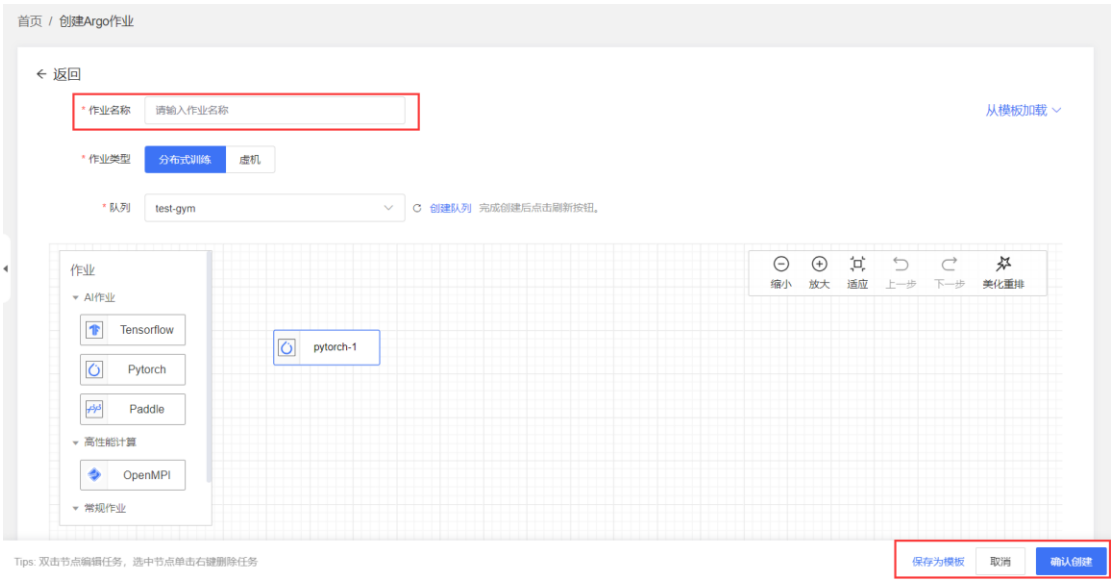
4.5.2 通过作业模板创建作业

操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击【模板管理】。
2. 在【作业管理】页面下，选择【作业模板】页签，展示作业模板列表。
3. 在【作业管理】页面下，选择一个作业模板，单击【创建作业】。



4. 跳转到【创建 Argo 作业】页面下，修改配置，单击右下角【确认创建】。



4.5.3 管理作业模板

操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击【模板管理】。
2. 在【作业管理】页面下，选择【作业模板】页签，展示作业模板列表。

3. 在【作业管理】页面下，选择一个作业模板，单击【查看】可查看模板配置。

4. 在【作业管理】页面下，选择一个作业模板，单击【删除】。

5. 弹框【删除作业模板】，二次确认即可删除作业模板。

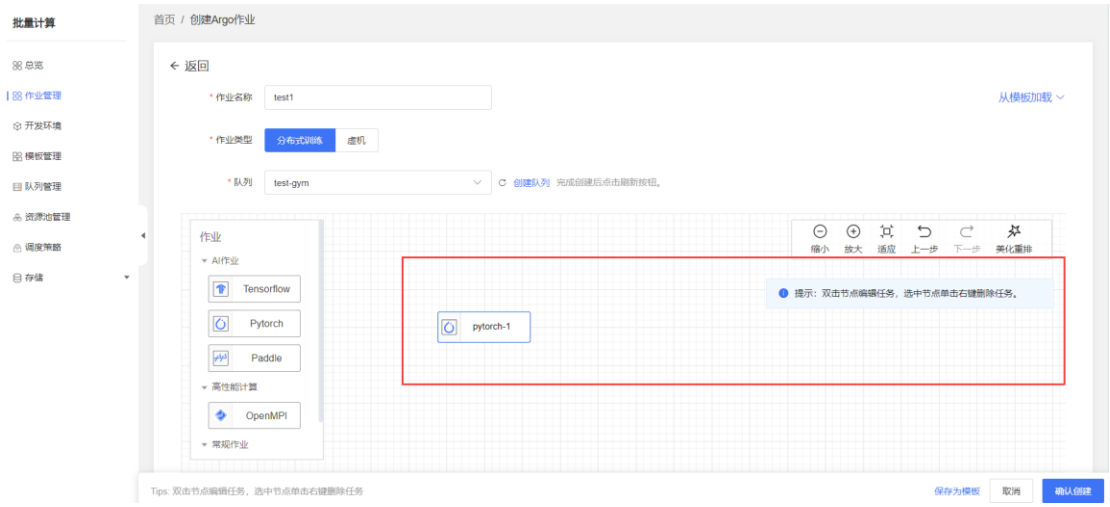
4.5.4 创建任务模板

操作步骤

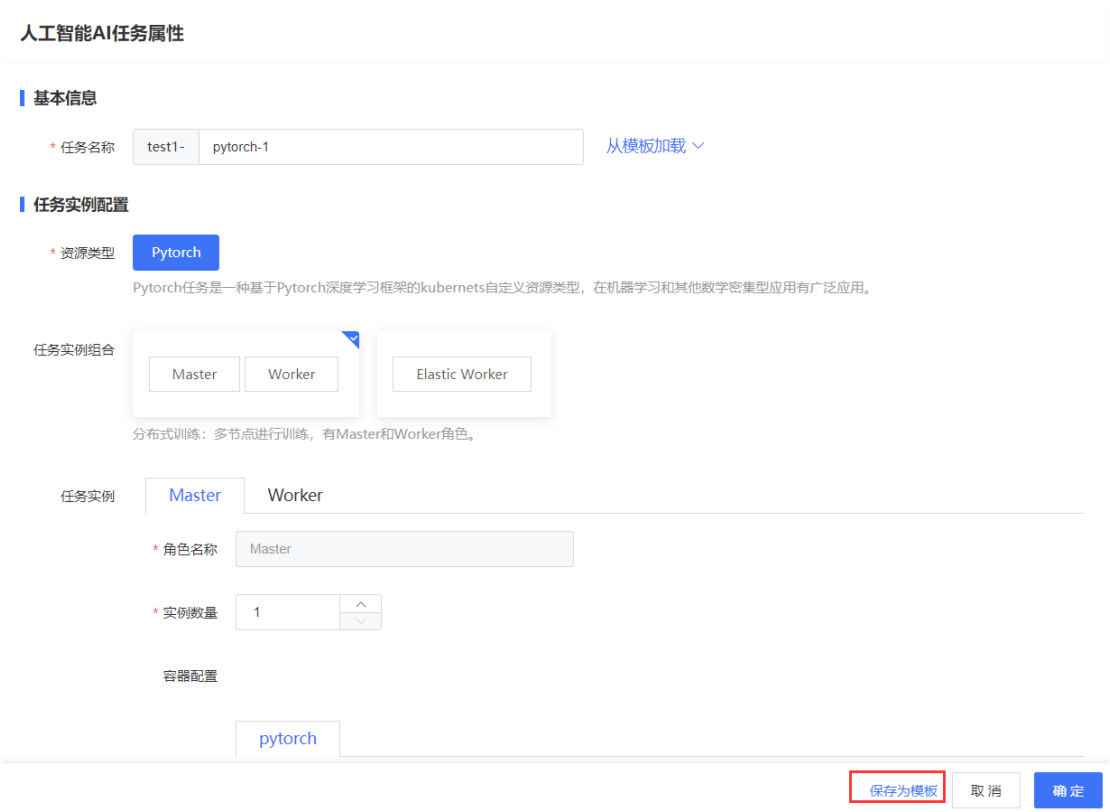
1. 登录批量计算管理控制台，在左侧导航栏单击【作业管理】。

2. 在【作业管理】页面下，单击右上角【创建作业】。

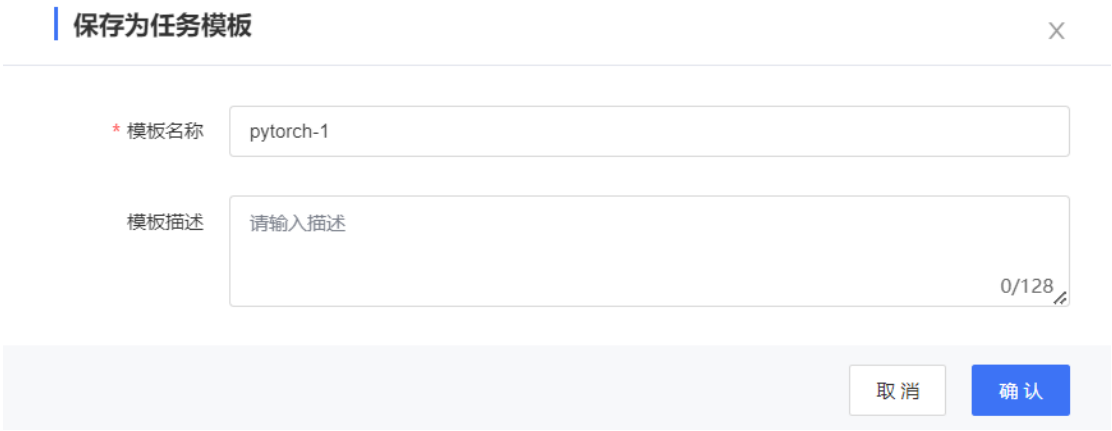
3. 在【创建 Argo 作业】页面下，输入作业名称，选择作业队列，拖动任意作业并双击配置。



4. 在【任务属性】页面，配置任务参数，单击右下角【保存为模板】。



5. 在保存为任务模板弹窗中，填写模板名称和模板描述，点击【确认】，完成模板保存。



4.5.5 通过任务模板创建作业

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【作业管理】。

3. 在【作业管理】页面中，单击左上角的【创建作业】。

4. 在【创建作业】页面中，输入作业名称，选择作业队列，拖动作业类型，双击任务名称，编辑任务。

5. 在任务配置页面中，点击任务名称右侧的【从模板加载】，在下拉列表中选择所需的模板即可直接加载到任务配置中。

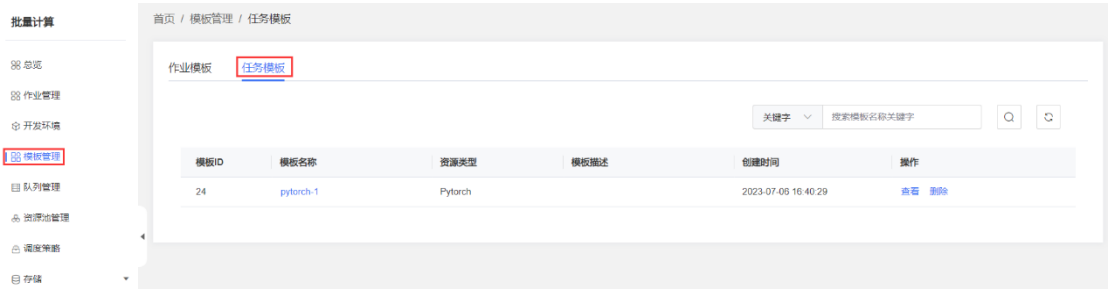
6. 任务配置完成后，提交作业完成作业创建。

4.5.6 管理任务模板

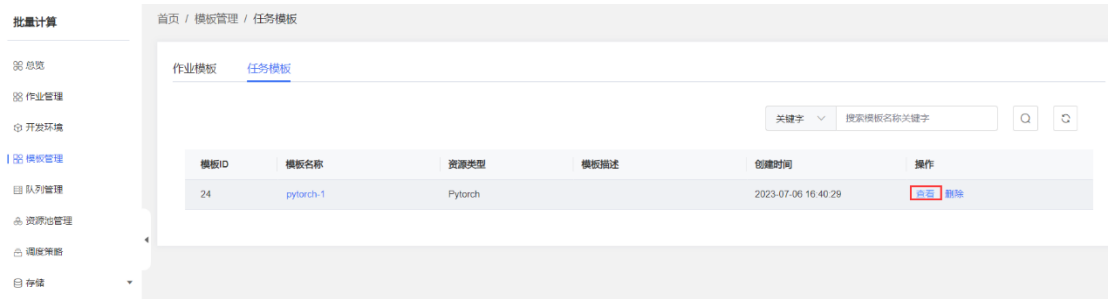
操作步骤

1. 登录批量计算管理控制台，在左侧导航栏单击【模板管理】。

2. 在【模板管理】页面下，选择【任务模板】页签，展示任务模板列表。



3. 在【任务管理】页面下，选择一个任务模板，单击【查看】可查看模板配置。



4. 在【任务管理】页面下，选择一个任务模板，单击【删除】。

5. 弹框【删除任务模板】，二次确认即可删除任务模板。

4.6 开发环境

开发环境，以云原生的资源使用和开发工具链的集成，为不同类型 AI 开发、探索、教学用户，提供更好云化 AI 开发体验，适用于高校或科研机构等进行 AI 模型开发调试等场合。

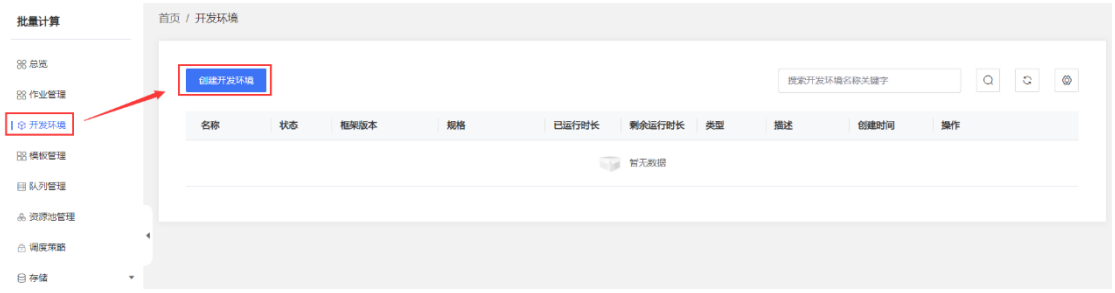
4.6.1 创建开发环境

前提条件

1. 开发环境投递的队列状态为“运行中”。
2. 队列配额充足。

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【开发环境】。
3. 在【开发环境】页面中，单击左上角的【创建开发环境】。

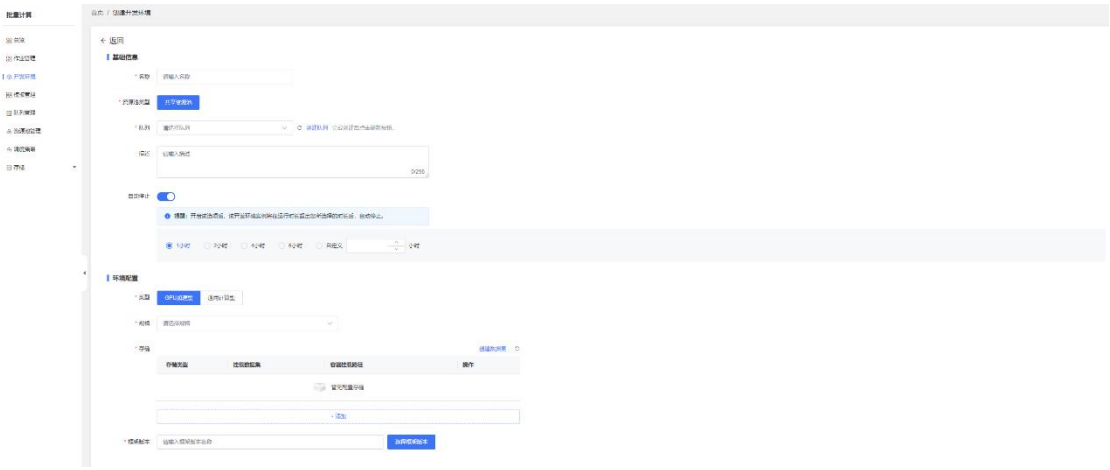


4. 在【创建开发环境】页面中，配置详细参数，具体如下表所示。

参数		说明
基础信息	开发环境名称	输入开发环境名称。要求如下： • 长度范围为 4~63 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 • 以小写字母或数字结尾。
	资源池类型	共享资源池：用户可以直接选择资源池，并且创建和使用在线 IDE。 专属资源池：目前暂不支持专属资源池，请使用共享资源池
	队列	选择队列，队列能自定义资源容量和隔离资源。如果未创建队列，可单击“创建队列”创建，具体操作请参见 3.3 小节队列管理。
	描述	可添加对开发环境的描述。
	自动停止	如果开启开关，开发环境将在设定的时间后自动停止；如果关闭开关，开发环境由用户手动停止。
	自动停止时长	开启自动停止开关需设置自动停止时长，单位小时。时长可选择 1 小时，2 小时，4 小时，6 小时，自定义时长，自定义时长数值为大于 0 的整数。默认时长为一个小时。
环境配置	类型	计算资源类型。可选： • GPU 加速型。使用 GPU 加速 AI 模型训练或者推断。 • 通用计算型。仅使用 CPU 进行模型训练或者推断。GPU 加速型能配置显卡数量，能够获得更好的体验。
	规格	计算资源规格。 • 通用计算型。包括 CPU（核），内存(GB)。 • GPU 加速型。包括 CPU（核），内存(GB)，显卡数量（块）
	存储	挂载数据集数据。 • 存储类型：支持文件存储和对象存储。 • 挂载数据集：可从数据集列表选择。如果没有数据集可点击“创建数据集”创建，具体操作请参见 3.7.2 小节数据集管理。 • 容器挂载路径：填入挂载的目标完全路径。

	框架版本	开发环境框架版本。点击选择“选择框架版本”，有以下参数： • 名称：例如“pytorch-jupyter:cuda-11.1-torch1.8.1”。 • 框架：例如：“Pytorch”。 • CUDA 版本：例如：“cuda-11.1-torch1.8.1”。 • 接入方式：显示 Jupyter 或者 VsCode 的图标。 当前支持的开发环境类型有 Jupyter 和 VsCode，框架有 Pytorch，Tensorflow 和 PaddlePaddle，CUDA 版本为 11.2 和 11.7。

5. 点击【创建开发环境】，填入参数，确认无误后点击【确认】。



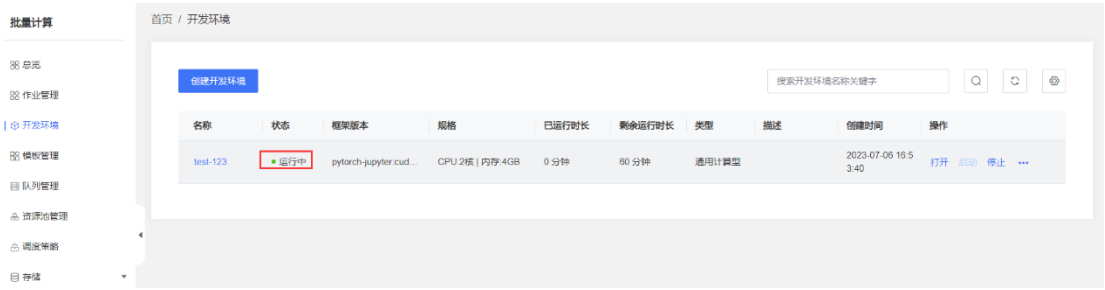
6. 进入开发环境列表，此时刚创建的开发环境处于“启动中”的状态。当开发环境的状态从“启动中”变成“运行中”就表示开发环境启动完成并且可以使用。下图为状态为运行中的开发环境。

4.6.2 开发环境列表

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。



上图为开发环境列表，列表展示了一个运行中的开发环境，显示的参数有：

参数	说明
搜索框	输入开发环境名称关键字，单击【🔍】搜索。
名称	开发环境名称，点击名称可查看开发环境详情。
状态	开发环境状态。包含的状态有： • 待创建。 • 启动中。包括资源调度，拉取镜像，启动容器流程等流程。 • 运行中。在线 IDE 实例正常运行中，可以进行“打开”和“停止”。 • 停止。在线 IDE 实例处于不可用状态，可点击“启动”重新开启在线 IDE。 • 运行失败。 • 运行结束。在线 IDE 实例在自动停止时长到期后自动停止，并处于不可用状态，也不可启动。 如果是运行中，将放置在绿色小圆点上可以查看自动停止的剩余时长，未设置自动停止，将不会显示自动停止时长。
框架版本	开发环境框架版本。显示开发环境类型，AI 开发框架类型和版本，CUDA 版本。例如：“pytorch-jupyter:cuda-11.1-torch1.8.1”，“pytorch”表示 AI 开发框架，“jupyter”表示开发环境类型，”cuda-11.1”表示 cuda 版本，“torch1.8.1”表示 AI 开发框架版本。
规格	计算资源规格。 • 通用计算型。包括 CPU（核），内存(GB)。 • GPU 加速型。包括 CPU（核），内存(GB)，显卡数量（块）

描述		开发环境描述。
创建时间		创建开发环境的时间。
操作	打开	打开开发环境，将会跳转到一个新窗口，显示对应开发环境的界面，用户可在新界面内进行代码开发和调试。该功能只有在开发环境运行中状态下才能点击。
	启动	启动开发环境。该功能只有在开发环境是暂停的状态下才能点击。点击“启动”后，开发环境的状态将变成启动中。
	停止	停止开发环境。停止开发环境将使开发环境变成不可用的状态。
	变更规格	变更计算资源规格。可自由选择 GPU 加速型和通用计算型： • GPU 加速型。使用 GPU 加速 AI 模型训练或者推断。 • 通用计算型。仅使用 CPU 进行模型训练或者推断。 再选择符合需求的计算资源即可。
	删除	删除开发环境。

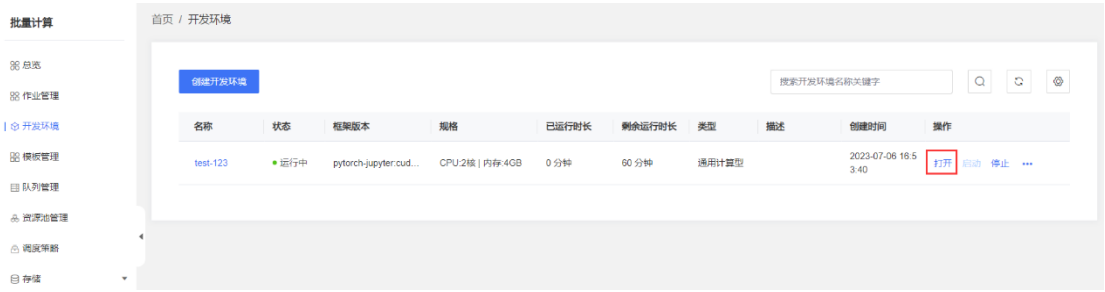
4.6.3 打开开发环境

前提条件

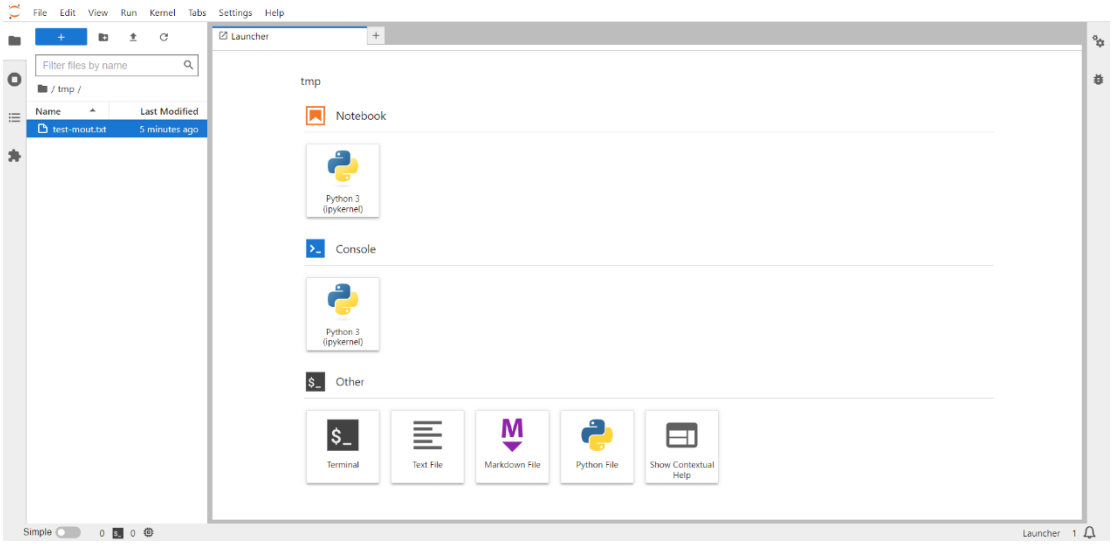
开发环境实例状态必须为“运行中”才可以打开。

操作步骤

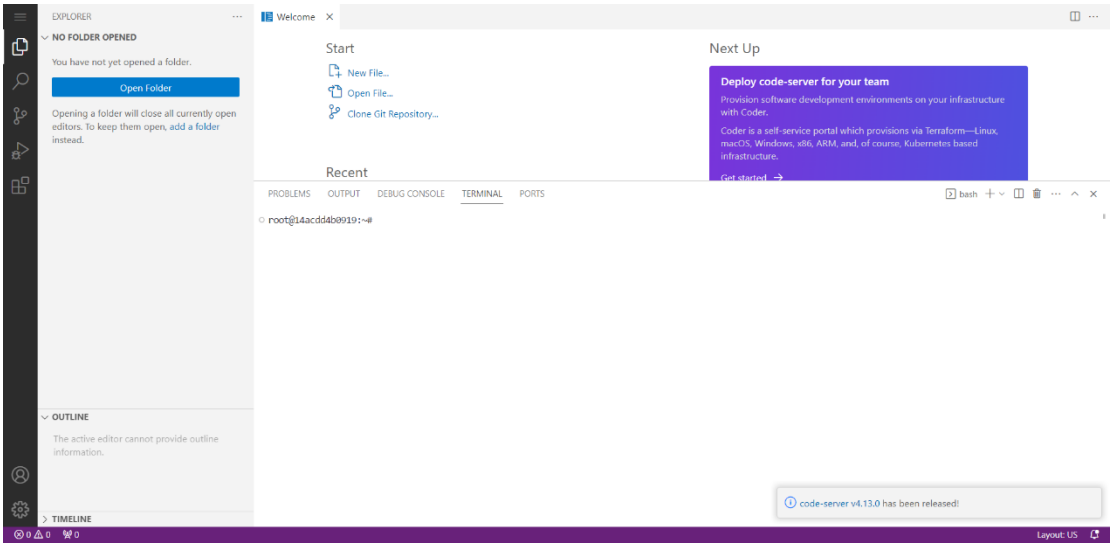
1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【开发环境】。
3. 在开发环境列表中定位到目标开发环境并单击【打开】。浏览器自动创建一个标签页，并在标签页内显示开发环境界面。



JupyterLab 界面：



VsCode 界面：



4. 切换到开发环境标签页，可以进行代码编写和调试。 在线 IDE 操作介绍请参考 3.6.9 小节介绍。

4.6.4 删除开发环境

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【开发环境】。
3. 在开发环境列表中定位到想要删除的开发环境，点击【删除】。确认删除界面如下图所

示：

4. 删除开发环境有二次确认的弹框，在输入框中输入“确认删除 “，并且点击【确认】完成删除。

4.6.5 保存自定义镜像

前提条件

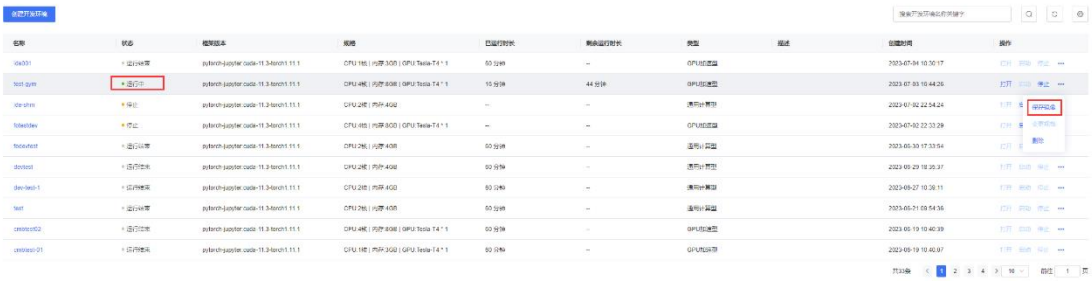
开发环境实例状态必须为“运行中”才可以一键进行镜像保存。

操作步骤

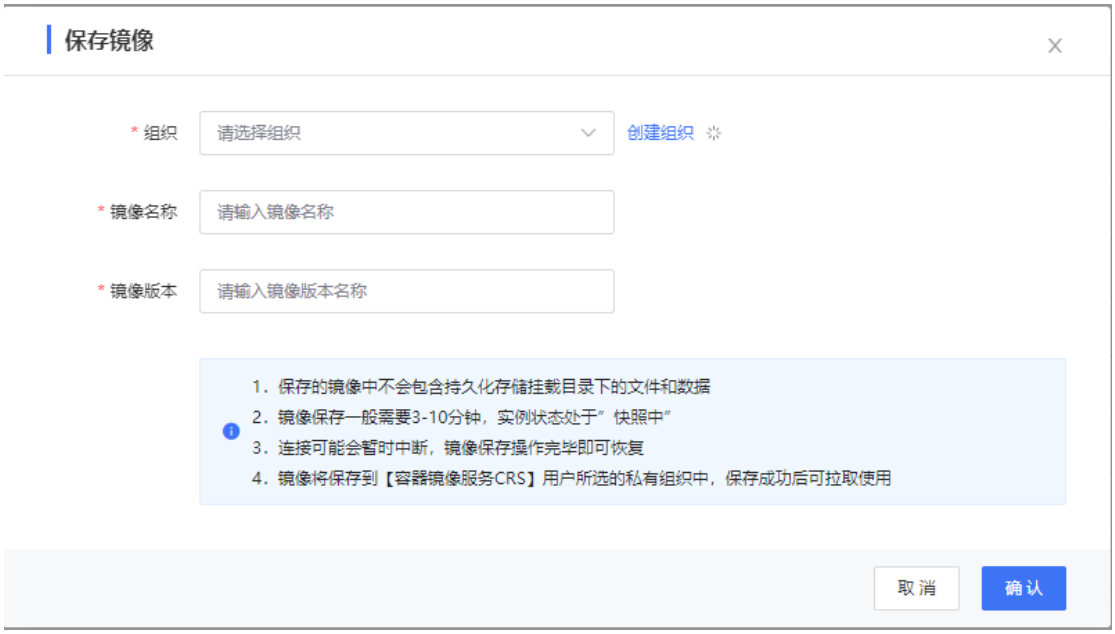
1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

3. 在开发环境列表中定位到想要保存为镜像的开发环境实例，点击右侧操作栏【保存镜像】，进入【保存镜像弹窗】。



4. 在保存镜像对话框中，设置组织、镜像名称和镜像版本信息。单击“确认”保存镜像。

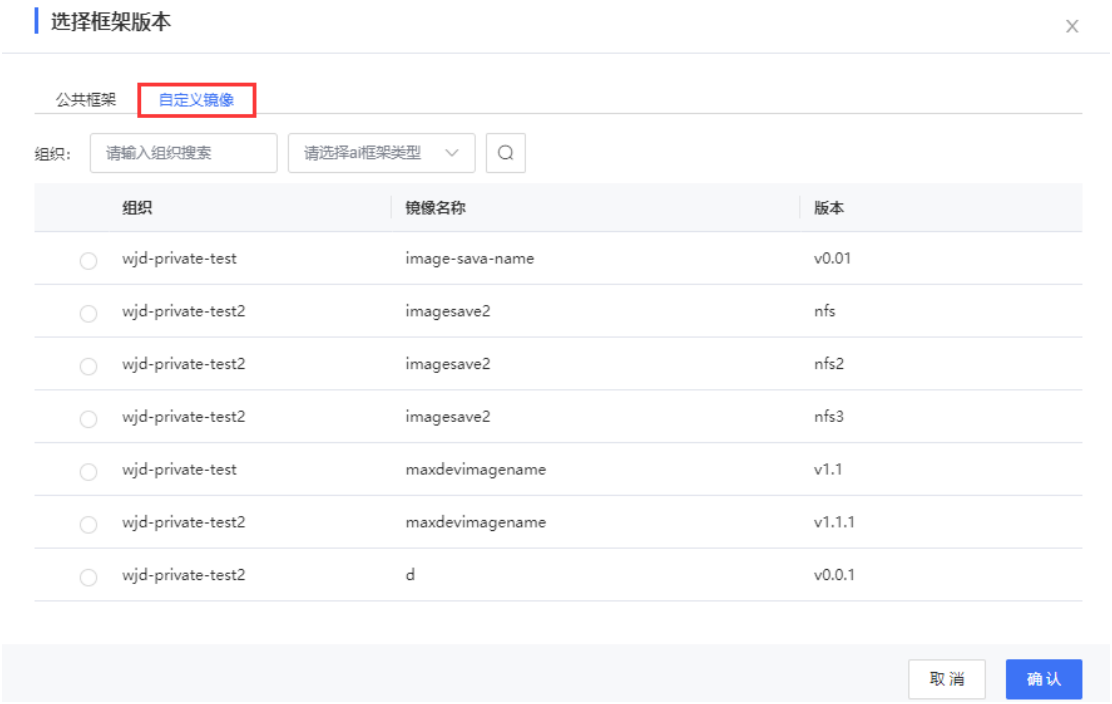


在“组织”下拉框中选择一个组织。如果没有组织，可以单击右侧的“创建组织”（可免费创建 3 个组织，每个组织有 10GB 容量）。同一个组织内的用户可以共享使用该组织内的所有镜像，首次登录容器镜像服务控制台需先设置访问凭证。

5. 镜像保存过程约 5-10 分钟，请耐心等待。此时不可再操作实例（对于打开的 JupyterLab 界面仍可操作）。

6. 镜像保存成功后，实例状态变为“运行中”，用户可在容器镜像服务（CRS）查看到该镜像详情。

7. 再次创建开发环境，在框架版本的弹窗中选择“自定义镜像”，即可找到保存的镜像。



4.6.6 查看开发环境详情

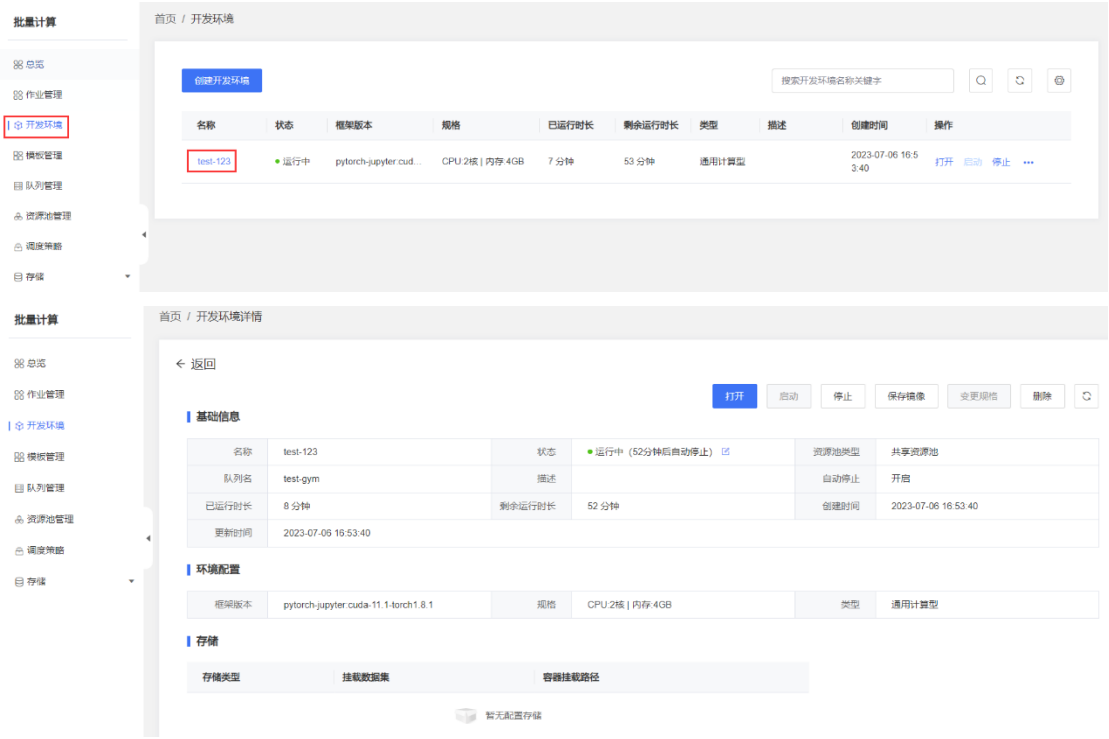
操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

3. 在开发环境列表中定位到想要查看的开发环境，点击“开发环境的名称”，例如

“test-123”。



4. 开发环境详情展示以下参数：

参数		说明
基础信息	名称	开发环境名称。
	状态	开发环境状态。包含的状态有： • 待创建。 • 启动中。包括资源调度，拉取镜像，启动容器流程等流程。 • 运行中。在线 IDE 实例正常运行中，可以进行“打开”和“停止”。 • 停止。在线 IDE 实例处于不可用状态，可点击“启动”重新开启在线 IDE。 • 运行失败。 • 运行结束。在线 IDE 实例在自动停止时长到期后自动停止，并处于不可用状态，也不可启动。 如果是运行中，还将显示自动停止的剩余时长。
	资源类型	目前只支持共享资源池。
	队列名	绑定队列的名称。
	自动停止	是否开启自动停止。
	描述	开发环境描述。
	创建时间	创建开发环境的时间。

	更新时间	上一次更改开发环境的时间。
环境配置	框架版本	开发环境框架版本。显示开发环境类型，AI 开发框架类型和版本，CUDA 版本。例如：“pytorch-jupyter:cuda-11.1-torch1.8.1”，“pytorch”表示 AI 开发框架，“jupyter”表示开发环境类型，”cuda-11.1”表示 cuda 版本，“torch1.8.1”表示 AI 开发框架版本。
	规格	计算资源规格。 • 通用计算型。包括 CPU（核），内存(GB)。 • GPU 加速型。包括 CPU（核），内存(GB)，显卡数量（块）
	类型	计算资源类型： • GPU 加速型。使用 GPU 加速 AI 模型训练或者推断。 • 通用计算型。仅使用 CPU 进行模型训练或者推断。
存储	存储	挂载数据集数据。有以下参数： • 存储类型 • 挂载数据集 • 容器挂载路径
操作	打开	打开开发环境，将会跳转到一个新窗口，显示对应开发环境的界面，用户可在新界面内进行代码开发和调试。该功能只有在开发环境运行中状态下才能点击。
	启动	启动开发环境。该功能只有在开发环境是暂停的状态下才能点击。点击“启动”后，开发环境的状态将变成启动中。
	停止	停止开发环境。停止开发环境将使开发环境变成不可用的状态。
	变更规格	变更计算资源规格。可自由选择 GPU 加速型和通用计算型： • GPU 加速型。使用 GPU 加速 AI 模型训练或者推断。 • 通用计算型。仅使用 CPU 进行模型训练或者推断。 再选择符合需求的计算资源即可。
	删除	删除开发环境。

4.6.7 变更开发环境规格

前提条件

开发环境实例状态必须为“停止”才可以变更规格。

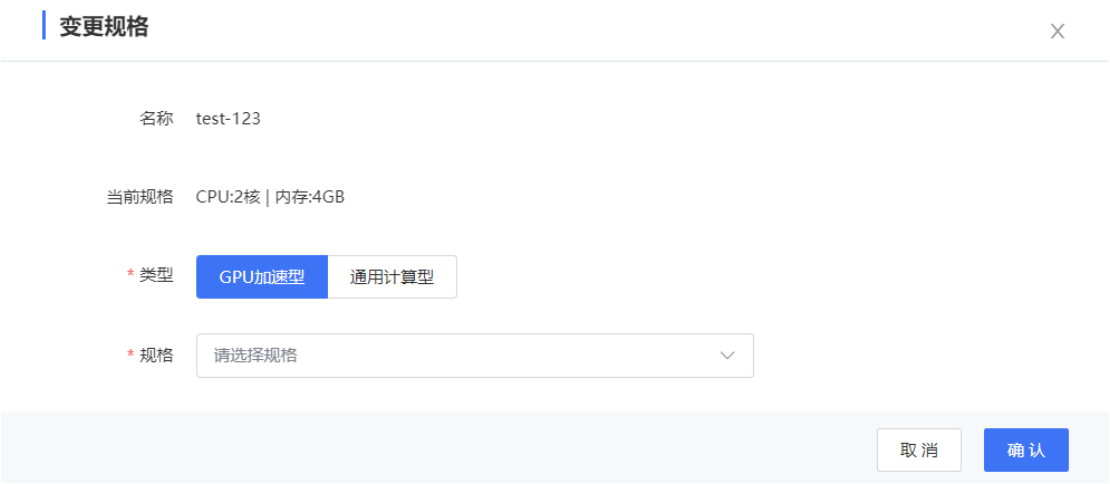
操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

3. 在开发环境列表中定位到想要变更规格的开发环境，点击【变更规格】（“停止”状态的开发环境实例才能变更规格）。

4. 在变更规格弹窗重选择目标规格，点击【确认】完成规格变更。



5. 变更规格将改变计算资源规格，具有以下参数：

参数	说明
类型	计算资源类型。可选： • GPU 加速型。使用 GPU 加速 AI 模型训练或者推断。 • 通用计算型。仅使用 CPU 进行模型训练或者推断。GPU 加速型能配置显卡数量，能够获得更好的体验。
规格	计算资源规格。 • 通用计算型。包括 CPU（核），内存(GB)。 • GPU 加速型。包括 CPU（核），内存(GB)，显卡数量（块）

6. 更改完规格之后，需要点击“启动”才能使用开发环境实例。

4.6.8 停止开发环境

前提条件

开发环境实例状态必须为“运行中”才可以停止。

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【开发环境】。
3. 在开发环境列表中定位到想要删除的开发环境，点击【停止】。停止开发环境确认界面如下图所示：

4. 停止开发环境有二次确认的弹框，点击【确认】即可停止开发环境运行。
5. 开发环境的状态由“运行中”变成“停止”。需要注意的是，除了挂载目录，停止开发环境将清楚其他目录下内容，包括用户在开发环境中安装的外部依赖和组件，请谨慎操作。

4.6.9 启动开发环境

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

3. 在开发环境列表中定位到想要删除的开发环境，点击【启动】。启动开发环境确认界面如下图所示：

4. 启动开发环境参数如下：

参数	说明
自动停止	自动停止开关，如果开启开关，开发环境将在设定的时间后自动停止；如果关闭开关，开发环境由用户手动停止。
自动停止时长	设置自动停止时长，单位小时。可用条件是开启自动停止开关。时长可选择 1 小时，2 小时，4 小时，6 小时，自定义时长，自定义时长数值应该为大于 0 的整数。

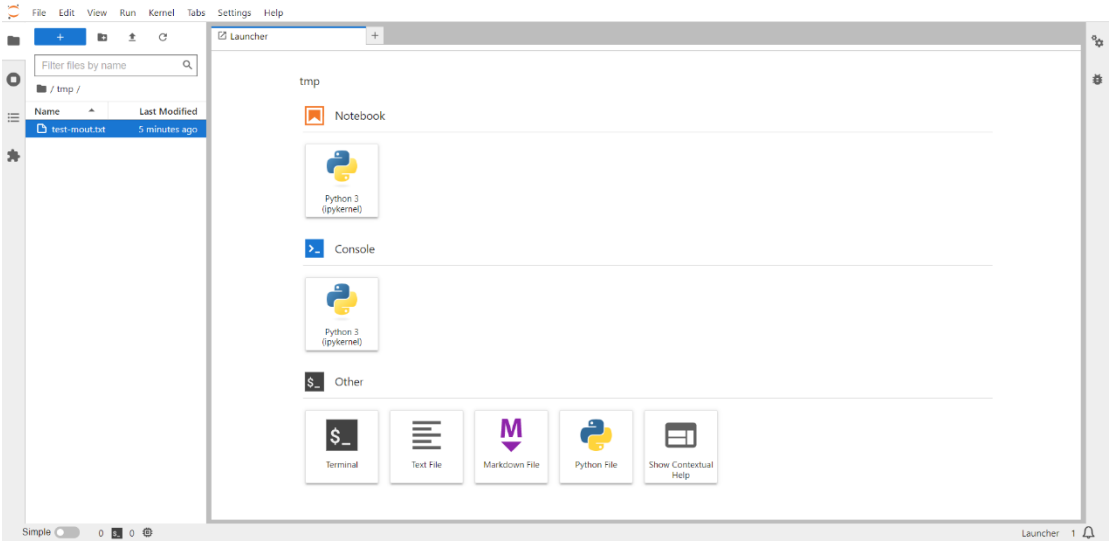
5. 启动之后开发环境的状态由“停止”变成“运行中”，用户可以打开开发环境实例。

4.6.10 JupyterLab

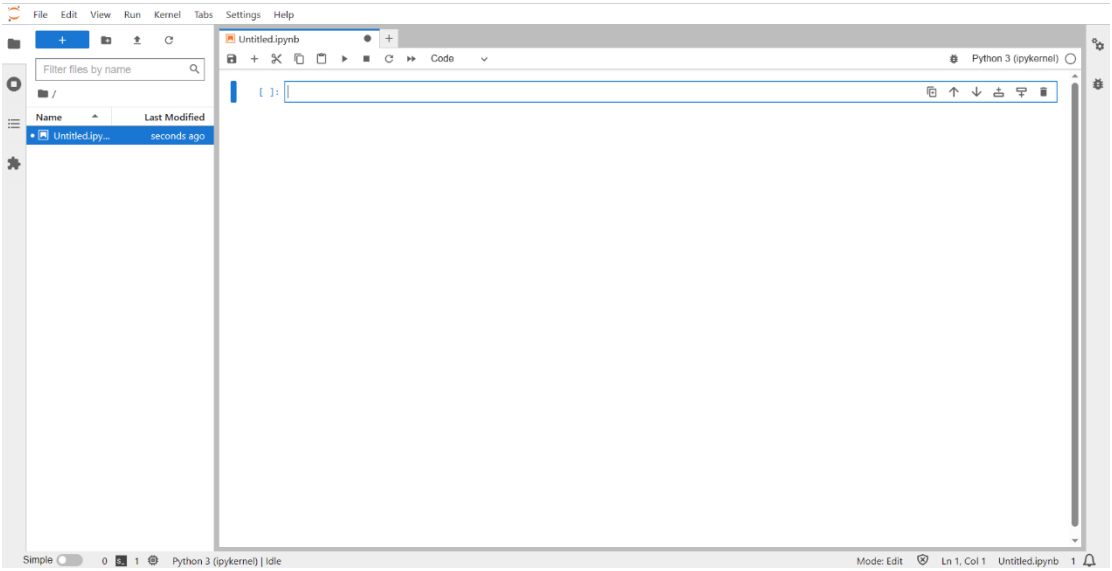
简介及常用操作。

1. 创建 JupyterLab 实例可参考 4.6.1 小节创建。详细操作可以参考官方文档：https://jupyterlab.readthedocs.io/en/stable/

2. JupyterLab 主页如下：

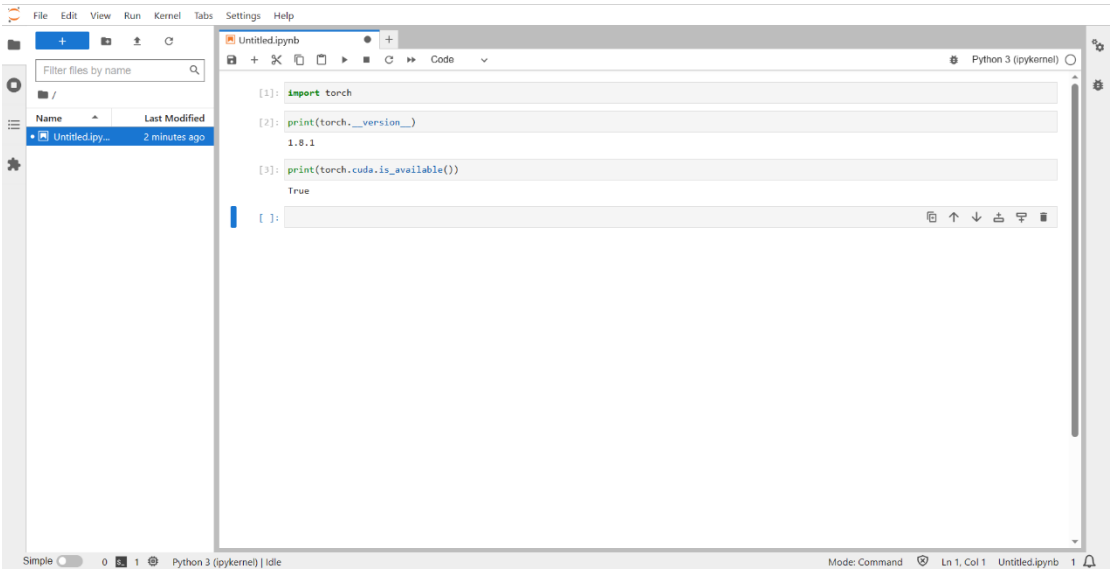


3. 运行 Notebook 的 Python3，将会自动创建一个 Untitled.ipynb。

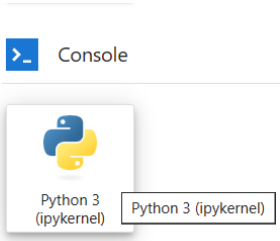


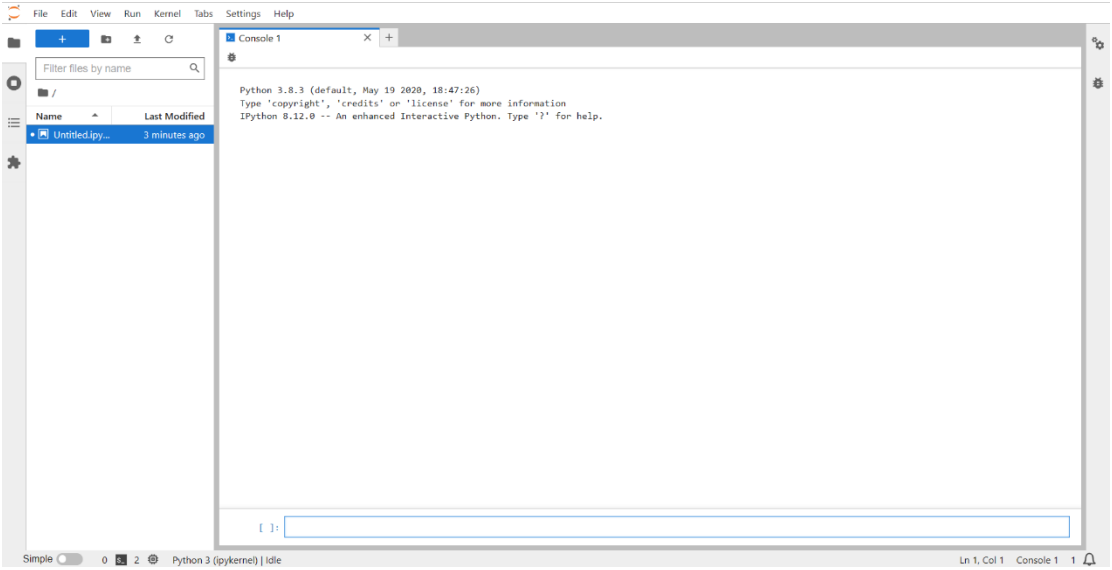
4. 菜单栏，从左到右分别表示保存、新增一个单元格、剪切单元格、复制单元格、粘贴单元格、运行选中的单元格、终止、重启内核、重启内核并且运行整个 notebook。

5. 可在 Untitled.ipynb 进行 python 代码调试：

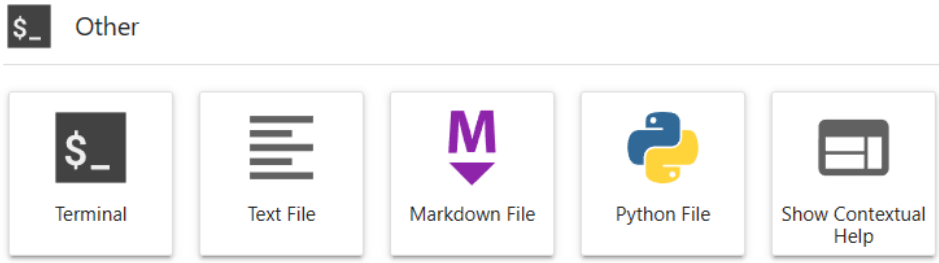


6. “Console“的”python3“提供 python 终端可进行命令控制

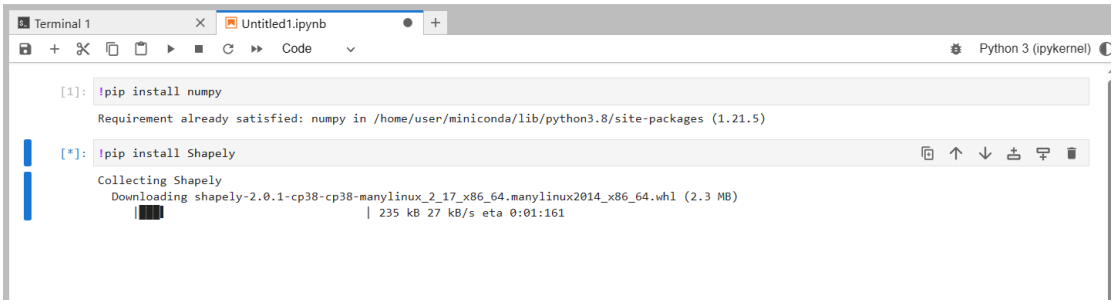




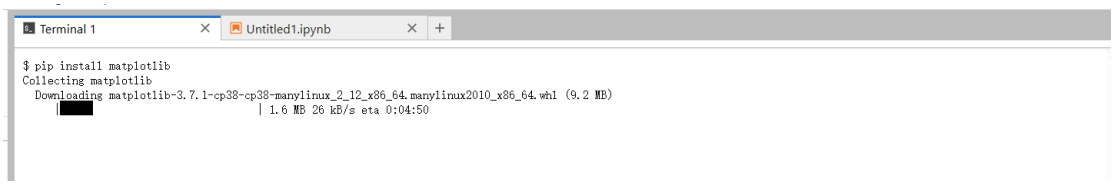
7. “Other“一栏提供系统终端服务，普通文本文本，Markdown 文本，python 文本编辑。



8. 安装外部依赖包：可在 Untitled.ipynb 文件内用在 pip 安装命令前增加“！“符号：

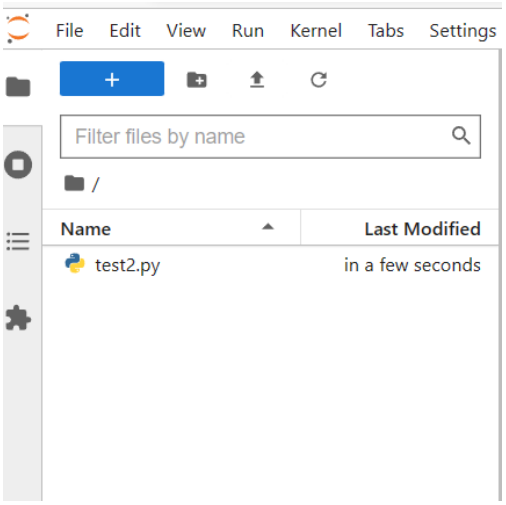


或者可以开启系统终端，在命令行中输入 pip 安装命令：

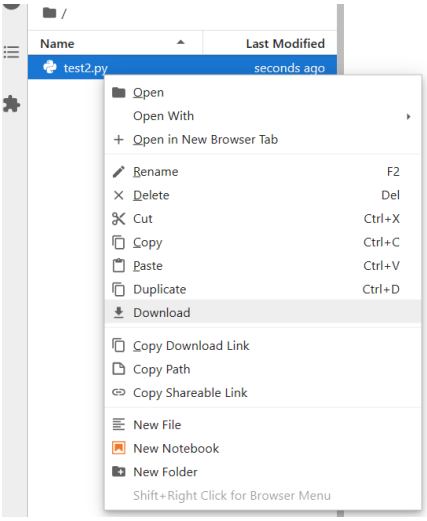


9. JupyterLab 的上传和下载（大小限制 4M 以内）：

上传：



下载：



4.6.11 监控插件

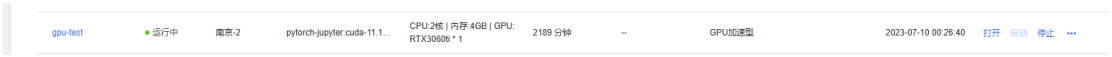
监控插件以及参考链接:

JupyterLab System Monitor : https://github.com/jtpio/jupyterlab-system-monitor

jupyterlab gpu 监控: https://developer.nvidia.com/blog/gpu-dashboards-in-jupyter-lab/

监控插件安装和使用介绍:

1. 创建 Jupyter 实例可参考 4.6.1 小节。



2. 打开 jupyter 实例，本教程以“gpu-test”为例。顶栏没有监控数据。

3. 安装插件 JupyterLab System Monitor 以及配置 jupyterlab 配置文件。

   命令:

   # 设置 pip 源:

   pip config set global.index-url https://pypi.douban.com/simple/

   # 安装 cpu 和内存的监控组件

pip install jupyterlab-system-monitor

```
user@gpu-test:/app$ pip install jupyterlab-system-monitor
Looking in indexes: https://pypi.douban.com/simple/
Collecting jupyterlab-system-monitor
  Downloading https://pypi.doubanio.com/packages/e4/cf/6b635049df82a79f0b5da5b8e4df3d936bd544a84050ae89582dd50ad445/jupyterlab_system_monitor-0.8.0-py3-none-any.whl (63 kB)
     |████████████████████████████████| 63 kB 793 kB/s
Requirement already satisfied: jupyterlab~=3.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab-system-monitor) (3.6.3)
Collecting jupyterlab-topbar~=0.6
  Downloading https://pypi.doubanio.com/packages/77/03/f3c1c92c9223309934de6d49166f6cb73fa38f9772a601cd5e6c4ca655c0/jupyterlab_topbar-0.6.1-py3-none-any.whl (56 kB)
     |████████████████████████████████| 56 kB 1.3 MB/s
Collecting jupyter-resource-usage~=0.5
  Downloading https://pypi.doubanio.com/packages/27/87/a505fb907a6e8609edd2ee8049f746fc8de91807b50ba99dc6e28024424e/jupyter_resource_usage-0.7.2-py3-none-any.whl (38 kB)
Requirement already satisfied: nbclassic in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (0.5.5)
Requirement already satisfied: jupyter-core in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (5.3.0)
Requirement already satisfied: jupyter-server<3,>=1.16.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (2.5.0)
Requirement already satisfied: tomli; python_version < "3.11" in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (2.0.1)
Requirement already satisfied: jinja2>=2.1 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (3.1.2)
Requirement already satisfied: jupyterlab-server~=2.19 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (2.22.1)
Requirement already satisfied: ipython in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (8.12.0)
Requirement already satisfied: notebook<7 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (6.5.4)
Requirement already satisfied: jupyter-ydoc~=0.2.3 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (0.2.4)
Requirement already satisfied: packaging in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (23.1)
Requirement already satisfied: jupyter-server-ydoc~=0.8.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (0.8.0)
Requirement already satisfied: tornado>=6.1.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (6.3.1)
Requirement already satisfied: psutil~=5.6 in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (5.9.5)
Requirement already satisfied: pyzmq>=19 in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (25.0.2)
Requirement already satisfied: prometheus-client in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (0.16.0)
Requirement already satisfied: argon2-cffi in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab~=3.0->jupyterlab-system-monitor) (21.3.0)
Requirement already satisfied: notebook-shim>=0.1.0 in /home/user/miniconda/lib/python3.8/site-packages (from nbclassic->jupyterlab~=3.0->jupyterlab-system-monitor) (0.2.2)
```

```
Requirement already satisfied: soupsieve>1.2 in /home/user/miniconda/lib/python3.8/site-packages (from beautifulsoup4
Requirement already satisfied: webencodings>=0.4 in /home/user/miniconda/lib/python3.8/site-packages (from tinycss2->
Requirement already satisfied: pycparser in /home/user/miniconda/lib/python3.8/site-packages (from cffi>=1.0.1->argon
Installing collected packages: jupyterlab-topbar, jupyter-resource-usage, jupyterlab-system-monitor
Successfully installed jupyter-resource-usage-0.7.2 jupyterlab-system-monitor-0.8.0 jupyterlab-topbar-0.6.1
user@gpu-test:/app$
```

安装完成之后，插件默认开启内存的监控，需要在 jupyterlab 的配置文件中修改配置，开启 cpu 的监控。

命令：

生成配置文件，

```
jupyter notebook --generate-config
```

```
user@gpu-test:/app$ jupyter notebook --generate-config
Writing default config to: /home/user/.jupyter/jupyter_notebook_config.py
```

安装 vim 工作

sudo apt update && sudo apt install vim

```
$ sudo apt update && sudo apt install vim
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [3346 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1369 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [2866 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1070 kB]
Fetched 8987 kB in 30s (297 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
93 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
vim is already the newest version (2:8.1.2269-1ubuntu5.15).
```

上图是已经安装过。

进入文件路径修改配置

复制下面这个配置到配置文件 jupyter_notebook_config.py 中。

c.ResourceUseDisplay.track_cpu_percent = True

```
jupyter_notebook_config.py  lab  migrated
user@gpu-test:~/.jupyter$ pwd
/home/user/.jupyter
user@gpu-test:~/.jupyter$ ls
jupyter_notebook_config.py  lab  migrated
user@gpu-test:~/.jupyter$
```

```
# Configuration file for jupyter-notebook.

c = get_config()  #noqa


# CPU
c.ResourceUseDisplay.track_cpu_percent = True
```

4. 安装 jupyterlab gpu 监控组件。

命令：

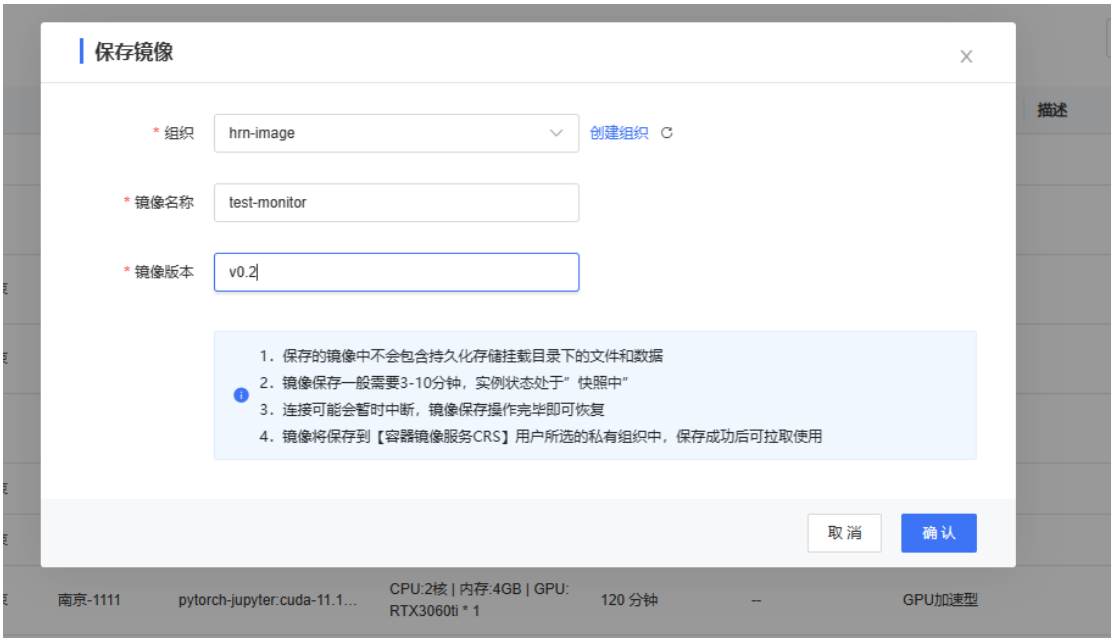
pip install jupyterlab-system-monitor
pip install jupyterlab-nvdashboard bokeh==2.4

```
user@gpu-test:~/.jupyter$ pip install jupyterlab-system-monitor
Looking in indexes: https://pypi.douban.com/simple/
Requirement already satisfied: jupyterlab-system-monitor in /home/user/miniconda/lib/python3.8/site-packages (0.8.0)
Requirement already satisfied: jupyter-resource-usage~=0.5 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab-system-monitor) (0.7.2)
Requirement already satisfied: jupyterlab~=3.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab-system-monitor) (3.6.3)
Requirement already satisfied: jupyterlab-topbar~=0.6 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab-system-monitor) (0.6.1)
Requirement already satisfied: pyzmq>=19 in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (25.0.2)
Requirement already satisfied: jupyter-server>=1.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (2.5.0)
Requirement already satisfied: prometheus-client in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (0.16.0)
Requirement already satisfied: psutil~=5.6 in /home/user/miniconda/lib/python3.8/site-packages (from jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (5.9.5)
Requirement already satisfied: notebook<7 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (6.5.4)
Requirement already satisfied: jupyter-ydoc~=0.2.3 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (0.2.4)
Requirement already satisfied: packaging in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (23.1)
Requirement already satisfied: jupyter-core in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (5.3.0)
Requirement already satisfied: jupyter-server-ydoc~=0.8.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (0.8.0)
Requirement already satisfied: ipython in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (8.12.0)
Requirement already satisfied: jinja2>=2.1 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (3.1.2)
Requirement already satisfied: jupyterlab-server~=2.19 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (2.22.1)
Requirement already satisfied: tornado>=6.1.0 in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (6.3.1)
Requirement already satisfied: tomli; python_version < "3.11" in /home/user/miniconda/lib/python3.8/site-packages (from jupyterlab~=3.0->jupyterlab-system-monitor) (2.0.1)
Requirement already satisfied: webencodings>=0.4 in /home/user/miniconda/lib/python3.8/site-packages (from tinycss2->nbconvert>=6.4.4->jupyter-server>=1.0->jupyter-resource-usage~=0.5->jupyterlab-system-monitor) (0
.5.1)
Requirement already satisfied: pycparser in /home/user/miniconda/lib/python3.8/site-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->jupyter-server>=1.0->jupyter-resource-usage~=0.5->jupyterlab-system
-monitor) (2.20)
user@gpu-test:~/.jupyter$ pip install jupyterlab-nvdashboard bokeh==2.4
Looking in indexes: https://pypi.douban.com/simple/
Collecting jupyterlab-nvdashboard
  Downloading https://pypi.doubanio.com/packages/8a/84/8ea7cf5abde83fc5cde80be9d055ed66f6f92446288d18340f23e37d02f4/jupyterlab_nvdashboard-0.8.0-py3-none-any.whl (37 kB)
Collecting bokeh==2.4
  Downloading https://pypi.doubanio.com/packages/fa/d3/da92cabfdfecd47b9efaea27833aa8e9512f9ad4818236be511f3486cda1/bokeh-2.4.0-py3-none-any.whl (18.4 MB)
     |                                | 2.3 MB 90 kB/s eta 0:02:58
```

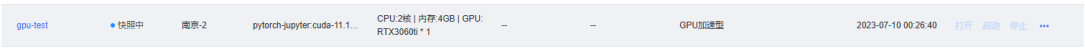
```
Requirement already satisfied: six in /home/user/miniconda/lib/python3.8/site-packages (from rfc3339-validator->jupyter-events>=0.4.0->jupyter-server>=1.0->jupyter-server-proxy->jupyterlab-nvdashboard) (1.14.0)
Requirement already satisfied: soupsieve>1.2 in /home/user/miniconda/lib/python3.8/site-packages (from beautifulsoup4->nbconvert>=6.4.4->jupyter-server>=1.0->jupyter-server-proxy->jupyterlab-nvdashboard) (2.4.1)
Requirement already satisfied: webencodings in /home/user/miniconda/lib/python3.8/site-packages (from bleach->nbconvert>=6.4.4->jupyter-server>=1.0->jupyter-server-proxy->jupyterlab-nvdashboard) (0.5.1)
Requirement already satisfied: cffi>=1.0.1 in /home/user/miniconda/lib/python3.8/site-packages (from argon2-cffi-bindings->argon2-cffi->jupyter-server>=1.0->jupyter-server-proxy->jupyterlab-nvdashboard) (1.14.0)
Requirement already satisfied: pycparser in /home/user/miniconda/lib/python3.8/site-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->jupyter-server>=1.0->jupyter-server-proxy->jupyterlab-nvdashboard)
(2.20)
Installing collected packages: bokeh, pynvml, frozenlist, aiosignal, multidict, async-timeout, yarl, aiohttp, simpervisor, jupyter-server-proxy, jupyterlab-nvdashboard
Successfully installed aiohttp-3.8.4 aiosignal-1.3.1 async-timeout-4.0.2 bokeh-2.4.0 frozenlist-1.3.3 jupyter-server-proxy-4.0.0 jupyterlab-nvdashboard-0.8.0 multidict-6.0.4 pynvml-11.5.0 simpervisor-1.0.0 yarl-1.9
.2
```

5. 保存镜像

点击保存镜像。

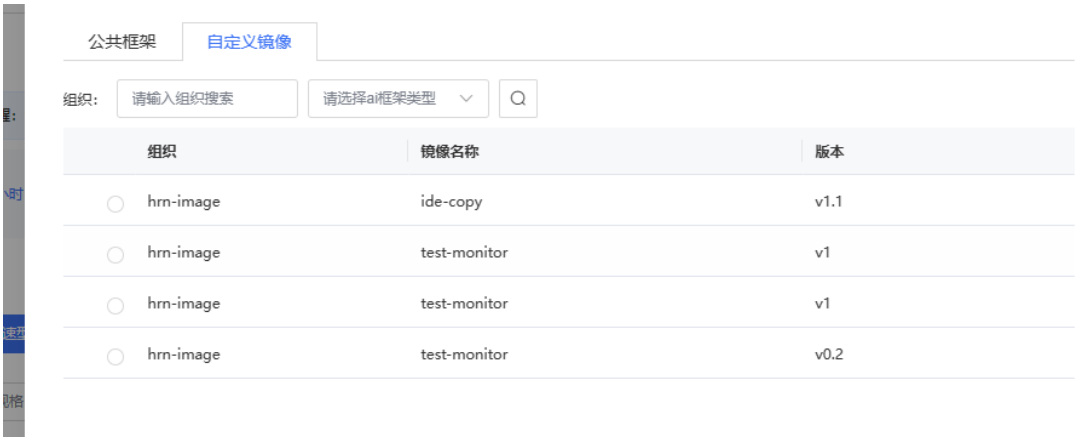


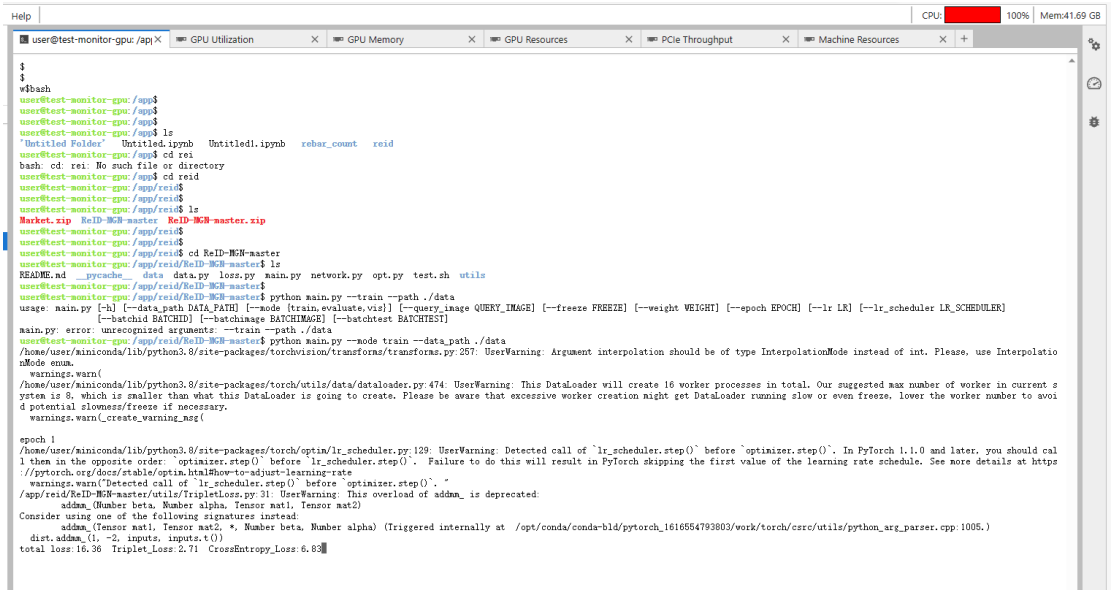
快照中，也就是保存镜像中，保存之后状态会变成运行中。



6. 新建一个 jupyter 开发实例。

选择自定义镜像，创建开发环境。开发环境启动中，需要等待镜像拉取。





右上角有 cpu 和内存的监控，内存大小会随着代码线程数变大，实际会看到很大的值。

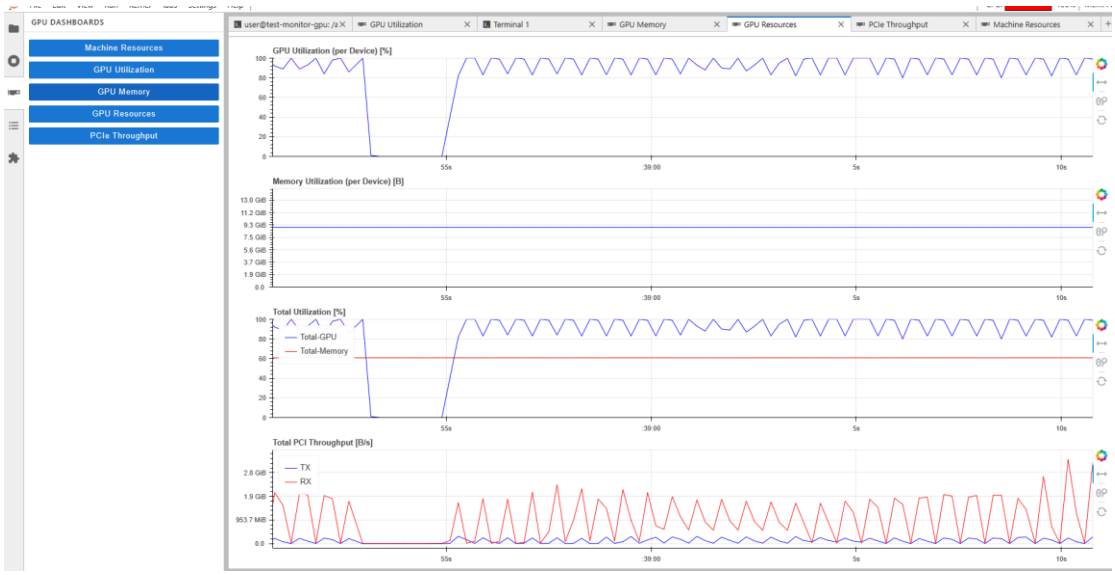
Gpu 监控选择最左列的选项：

首先是设备资源，包括 cpu，内存，磁盘 io 带宽，网络 io 带宽。

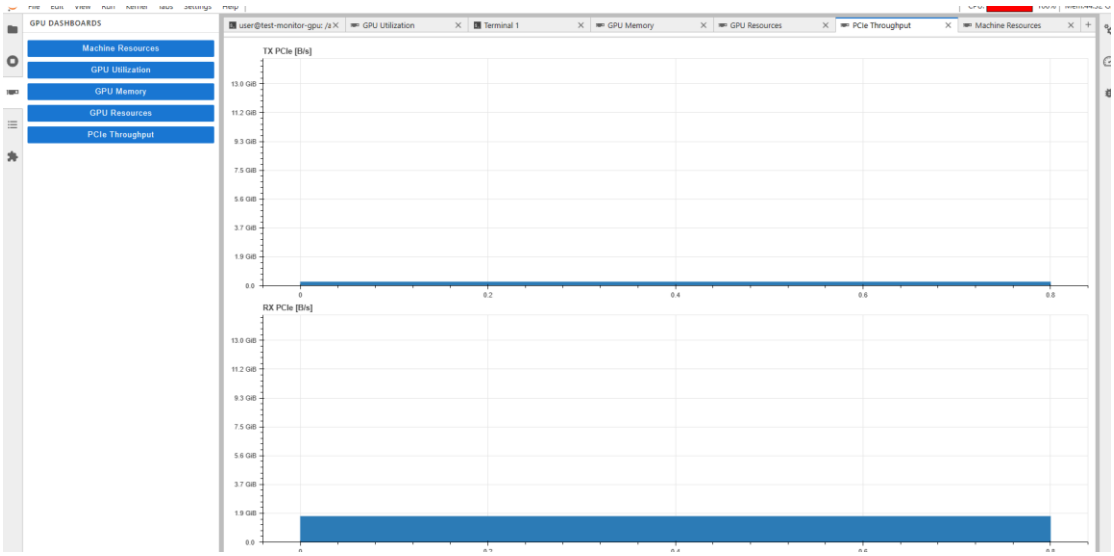
然后是 gpu 利用率。

Gpu 显存

Gpu 资源信息



PCIE 吞吐量

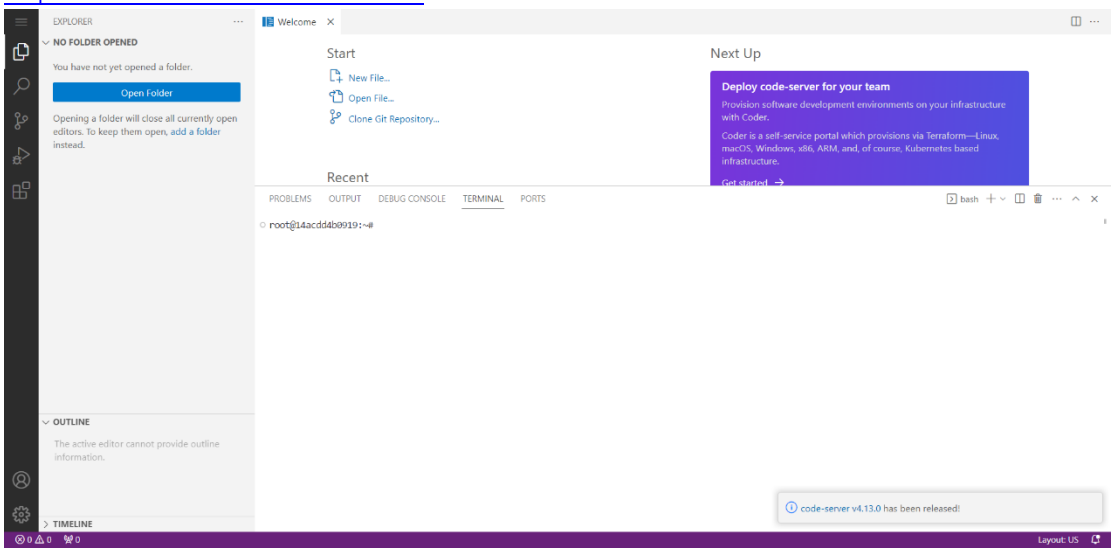


4.6.12 VsCode

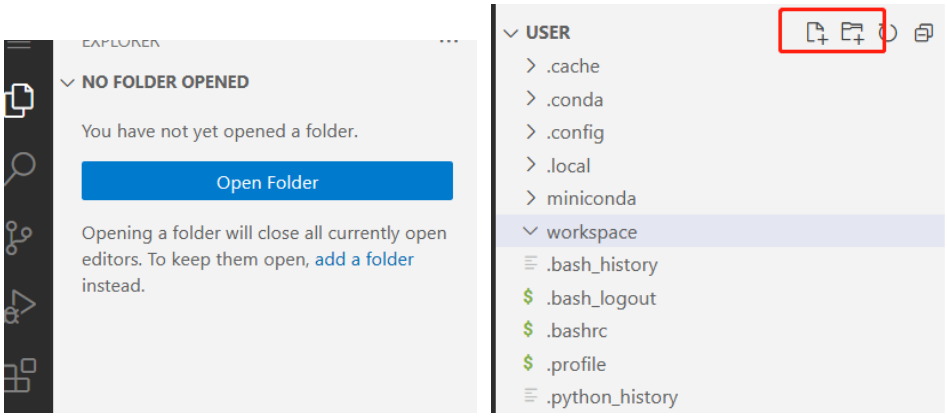
Vscode 简介及常用操作。

1. 创建 VsCode 实例可参考 4.6.1 小节创建。详细操作可以参考官方文档：

https://code.visualstudio.com/docs

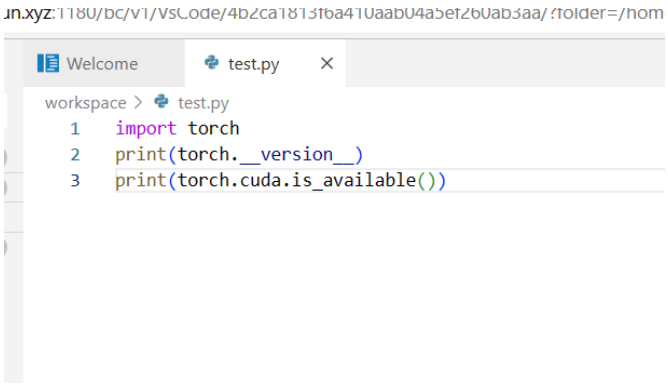


2. 打开文件夹，创建工作目录和工作文件：

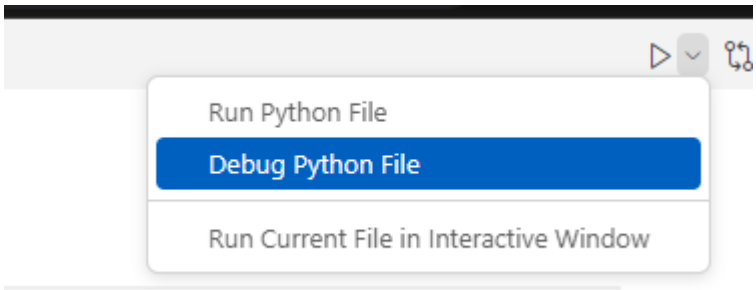


3. 安装 python 插件进行调试：

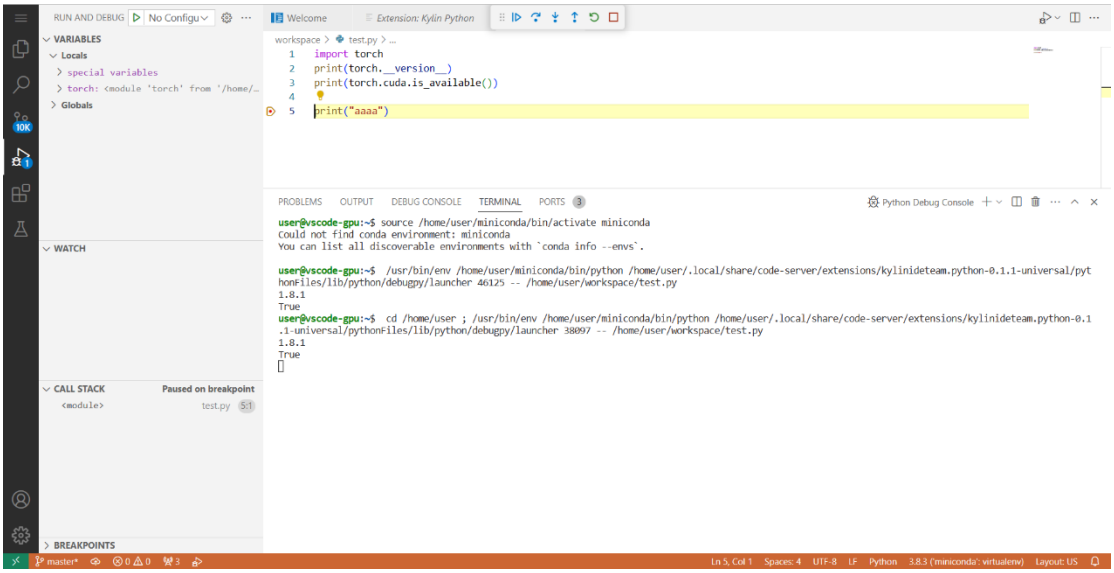
4. 编写测试代码：



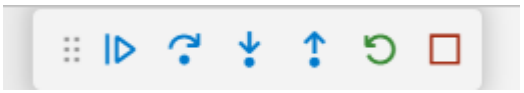
5. 点击右上角的“debug python file”可以进行调试：



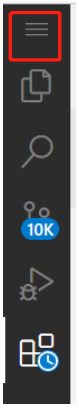
6. 在代码中打断点并且调试，输出结果在终端中显示：

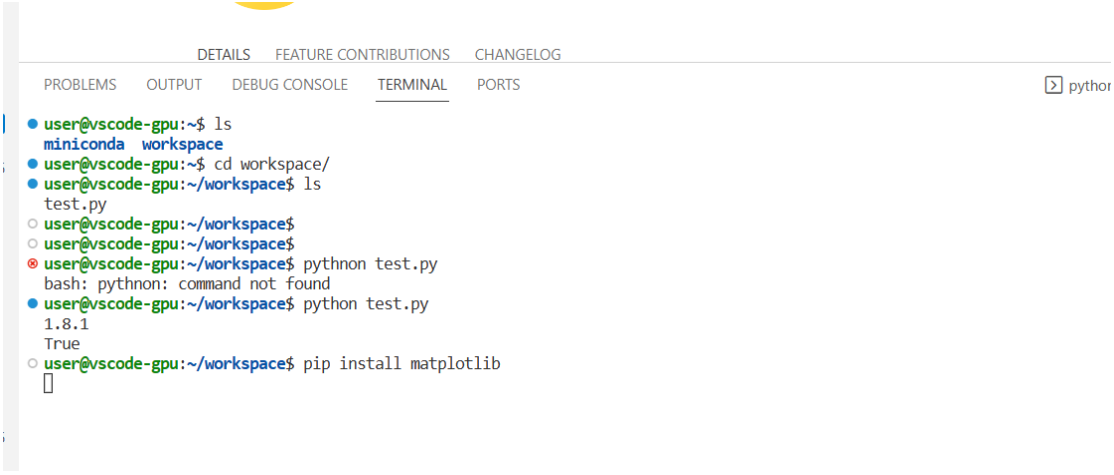


7. 代码 debug 菜单：

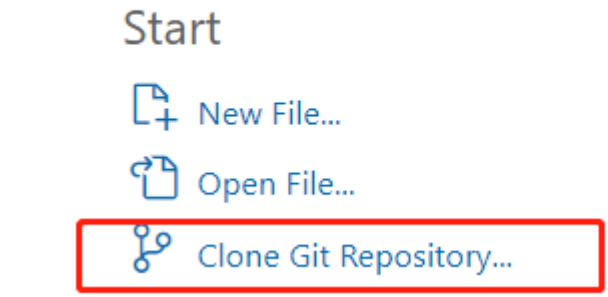


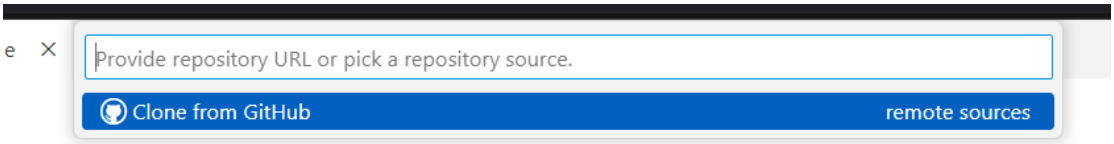
8. 安装外部依赖包：选择功能栏第一个，再点击“terminal”，新建一个终端。



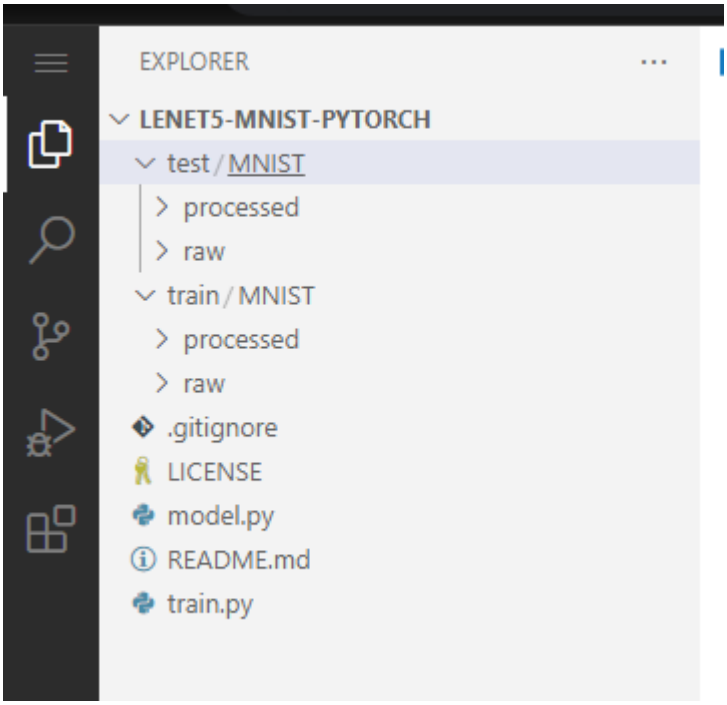


9. 可以 clone git 工程并且进行代码调试。

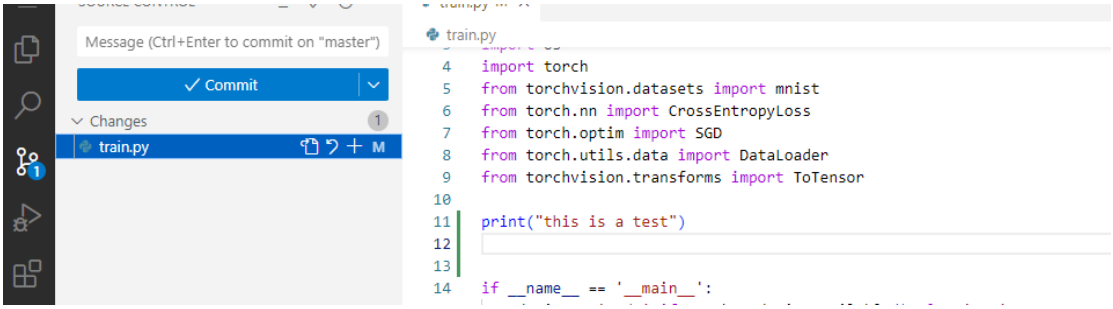




10. Git clone github 路径



11. 更改文件内容，



4.7 存储管理

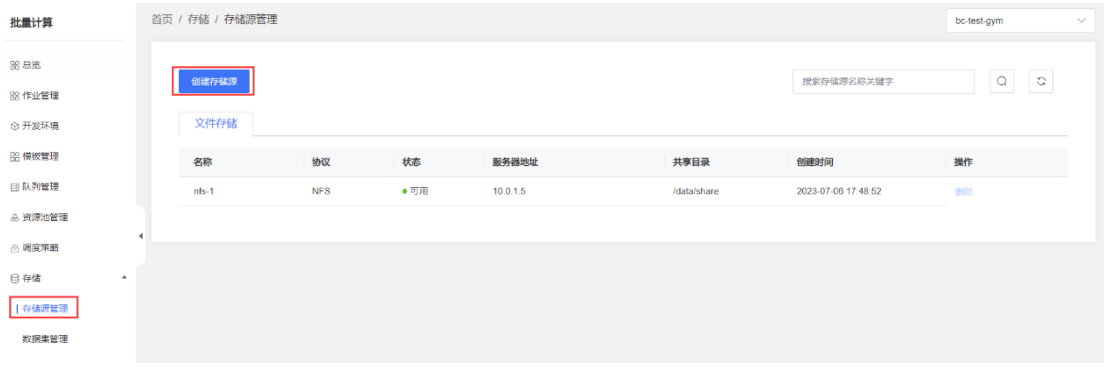
批量计算的数据存储管理，用于对接存储服务，实现数据管理入口的统一。

4.7.1 存储源管理

4.7.1.1 创建存储源

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【存储】。
3. 在【存储】弹出的选项中，选择【存储源管理】。
4. 在【存储源管理】页面，点击左上角的【创建存储源】按钮。



5. 在弹出的【创建存储源】页面中，配置参数，具体如下表所示。

参数	说明	是否必选
名称	输入作业名称。要求如下： • 长度范围为 4~53 个字符。 • 名称由小写字母、数字、中划线（-）和点（.）组成。 • 以小写字母开头。 • 以小写字母或数字结尾。	是
资源池	在选择资源池类型后，再选择资源池，如果还未创建资源池，可单击“创建专属/共享资源池”进行创建，具体操作请参见资源池	是

	管理。	
存储类型	当前只能使用网络文件系统（NFS）类型的存储。NFS 是一个分布式文件系统协议，它允许通过网络共享远程文件夹。通过 NFS，可将远程文件夹挂载到系统上，并且操作远程机器的文件，就像本地文件一样方便。	是
服务器地址	若是专属资源池，此处请确认所有节点安装了 nfs-utils，NFS 服务器地址与作业集群能够互通（内网或公网均可）。	是
共享目录	此处请确认 NFS 共享目录的权限是可以让其他用户正常读写执行（rwx）的。	是

6. 点击右下角的【确认】，即可完成存储源的创建。

4.7.1.2 查看存储源列表

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【存储】。

3. 在【存储】弹出的选项中，选择【存储源管理】。

4. 此时显示的页面即为【存储源列表】页面，在页面上方可下拉选择资源池，通过资源池来筛选存储源列表。以下【存储源列表】展示的字段说明：

参数	说明
搜索框	输入存储源名称或者关键字，支持输入部分名称和全名称进行检索。
检索	根据输入存储源名称或者关键字进行检索筛选。
刷新	刷新当前存储源列表展示页面。
名称	存储源的名称。
存储类型	目前仅支持 NFS 存储类型。
状态	存储源的状态，共有 2 种，分别为可用和不可用。可用状态表示此存储源是可联通的，可在此存储源上创建数据集；而不可用状态表示此存储源不可联通，不可在此存储源上创建数据集。
服务器地址	存储源所挂载的服务器地址。
共享目录	存储源挂载的共享目录路径。
创建时间	存储源的创建时间。
操作	删除：详见 3.7.1.3

4.7.1.3 删除存储源

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【存储】。

3. 在【存储】弹出的选项中，选择【存储源管理】。

4. 在【存储源管理】页面，选择一个存储源的【删除】按钮，注意，当有数据集和该存储源关联时，该存储源的【删除】按钮将置灰。

5. 此时弹出【删除存储源】页面，确认信息无误且没有数据集与该存储源关联后，填入【确认删除】。

6. 单击弹窗的【确认】按钮，此时删除存储源成功。

4.7.2 数据集管理

4.7.2.1 创建数据集

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【存储】。

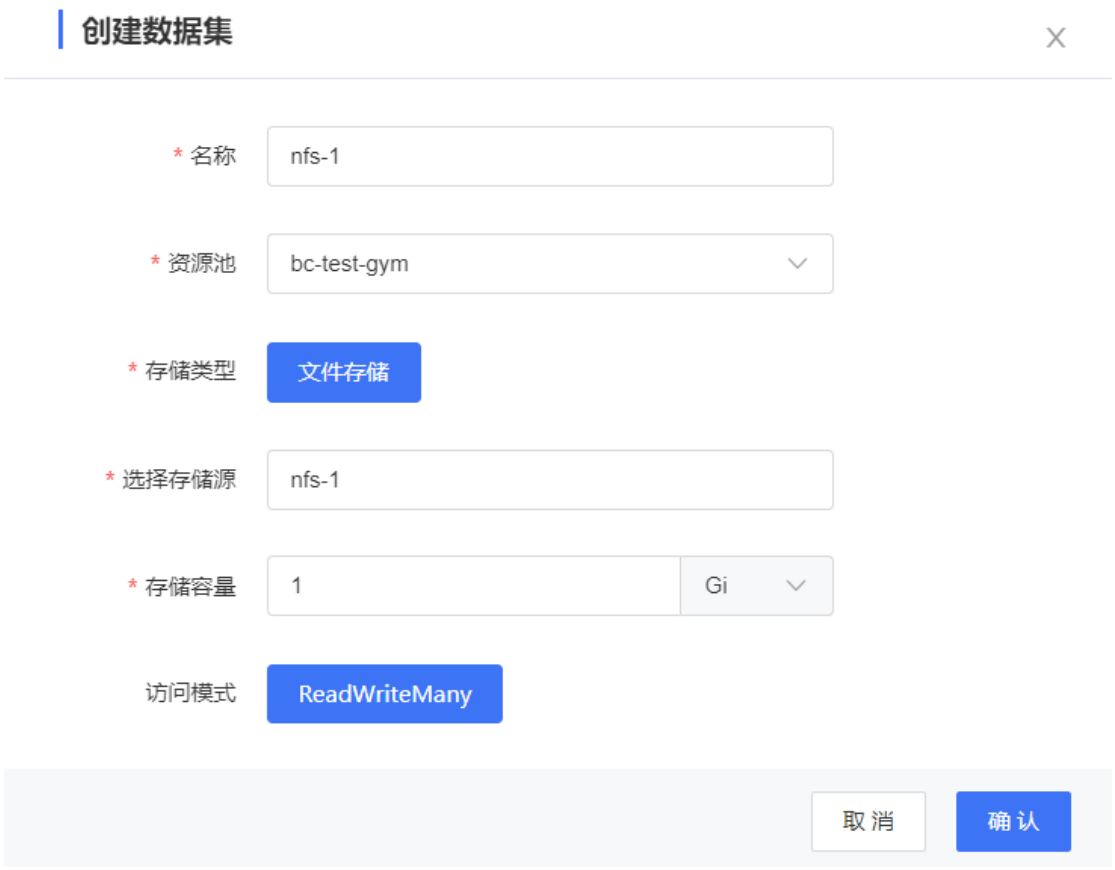
3. 在【存储】弹出的选项中，选择【数据集管理】。

4. 在【数据集管理】页面，点击左上角的【创建数据集】按钮。

5. 在弹出的【创建数据集】页面中，配置参数，具体如下表所示。

参数	说明	是否必选
名称	输入作业名称。要求如下： • 长度范围为 4~63 个字符。 • 名称由小写字母、数字、中划线（-）组成。 • 以小写字母开头。 • 以小写字母或数字结尾。	是
资源池	在选择资源池类型后，再选择资源池，如果还未创建资源池，可单击“创建专属/共享资源池”进行创建，具体操作请参见资源池管理。	是
选择存储源	选择已有的且状态为可用的存储源作为数据集的挂载存储源，如果还未创建存储源，可单击“新建存储源”进行创建，具体操作详见 3.7.1.1。	是
存储容量	数据集的存储容量大小，要求输入为数字，可选单位为 Gi/Ti。	是
访问模式	目前仅支持 ReadWriteMany 的访问模式。	是

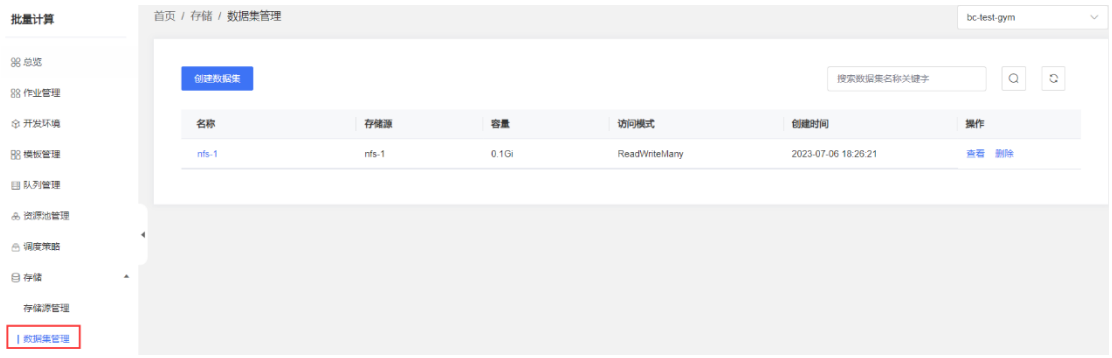
6. 点击右下角的【确认】，即可完成数据集的创建。



4.7.2.2 查看数据集列表

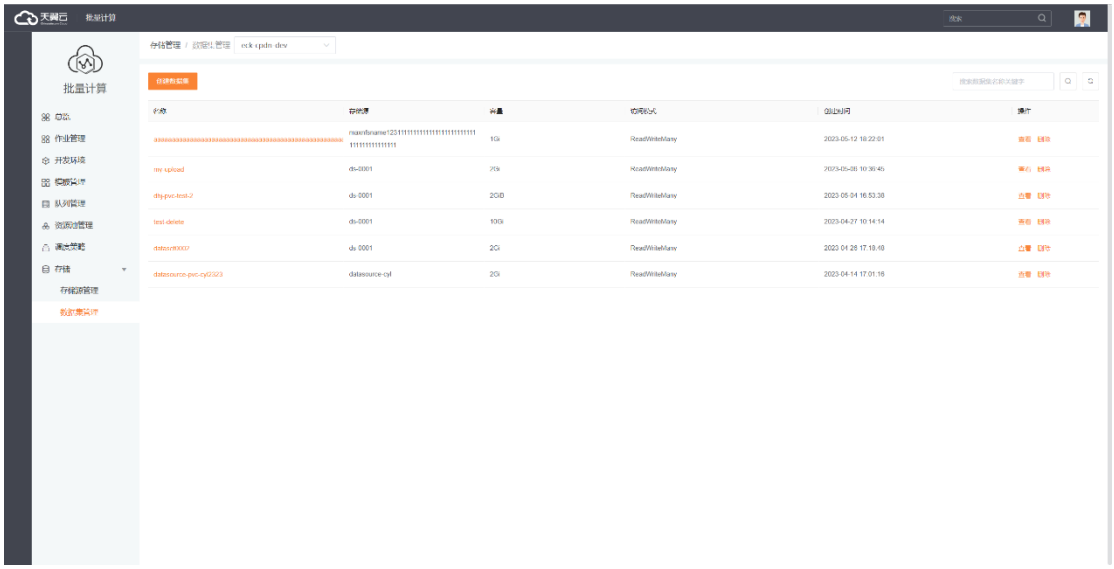
操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【存储】。
3. 在【存储】弹出的选项中，选择【数据集管理】。
4. 此时显示的页面即为【数据集列表】页面，在页面上方可下拉选择资源池，通过资源池来筛选数据集列表。



5. 以下【数据集列表】展示的字段说明：

参数	说明
搜索框	输入数据集名称或者关键字，支持输入部分名称和全名称进行检索。
检索	根据输入数据集名称或者关键字进行检索筛选。
刷新	刷新当前数据集列表展示页面。
名称	数据集的名称。
存储源	数据集绑定的存储源的名称。
存储容量	数据集的存储容量大小，单位为 Gi/Ti。
访问模式	目前仅支持 ReadWriteMany 的访问模式。
创建时间	存储源的创建时间。
操作	查看：详见 3.7.2.3 删除：详见 3.7.2.4



4.7.2.3 查看数据集详情

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【存储】

3. 在【存储】弹出的选项中，选择【数据集管理】。

4. 选择一个数据集，在操作栏中单击【查看】按钮。

5. 此时可查看数据集中的存储的内容，即为【存储文件管理】页面。以下【存储文件管理】页面展示的字段和功能说明：

参数	说明
刷新	刷新当前存储文件管理展示页面。
文件名	存储的文件名或者文件夹名。
文件类型	若为文件夹时，文件类型为文件夹；若为具体文件时，文件类型为文件后缀名。
大小	若为文件夹时，不显示；若为具体文件时，为文件大小。。
修改时间	文件的最后修改时间。
操作	上传：详见以下说明 下载：点击【下载】按钮，可将对应文件下载到本地磁盘。

4.7.2.4 删除数据集

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【存储】
3. 在【存储】弹出的选项中，选择【数据集管理】。
4. 选择一个数据集，在操作栏中单击【删除】按钮。
5. 此时弹出【删除数据集】页面，确认信息无误后，填入【确认删除】。
6. 单击弹窗的【确认】按钮，此时删除数据集成功。

4.7.2.5 在作业中引用数据集

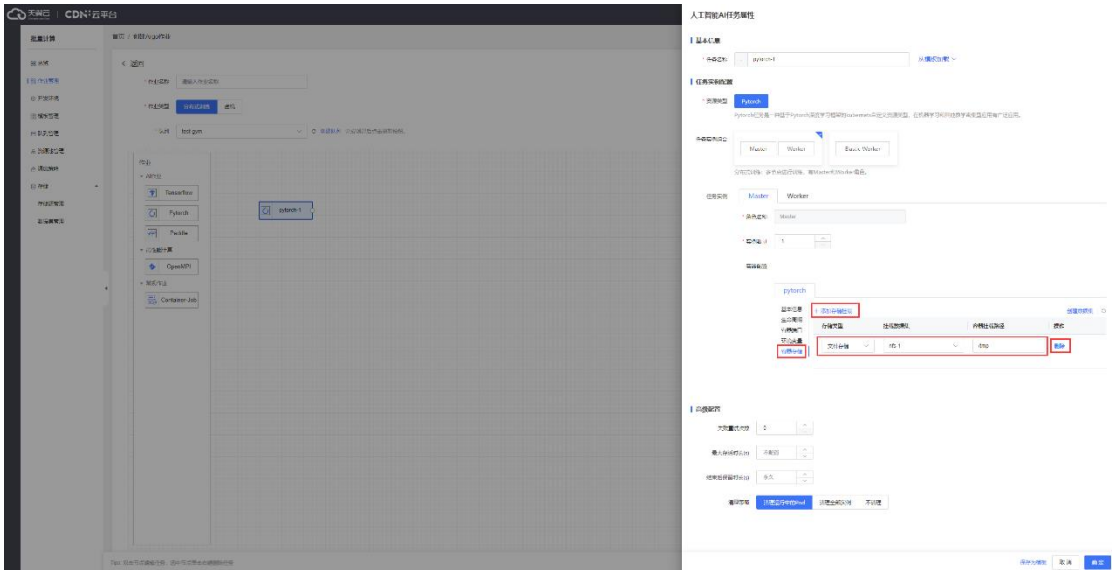
操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【作业管理】。
3. 在【作业管理】页面中，单击左上角的【创建作业】。
4. 在【创建作业】页面中先选择队列，再选择任意一个 AI 作业并对 AI 作业双击，到【任务实例配置】-【任务实例】-【容器配置】中，点击【添加存储挂载】按钮。
5. 此时在下拉条中，可以选择之前已经创建好的数据集，若还未创建数据集，可单击

【添加数据集】按钮，详见 3.7.2.1；

6. 在选择完挂载数据集之后，用户可自定义填入例如“/data”类似的容器挂在路径。

7. 若不需要挂载数据集，也可点击右方删除操作，取消挂载数据集。



4.7.2.6 在开发环境中引用数据集

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

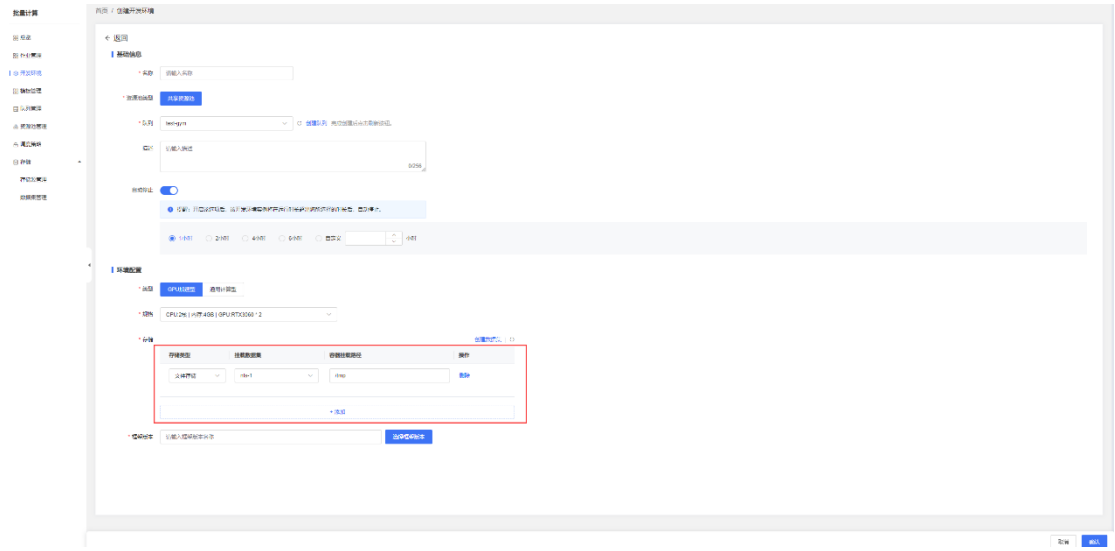
3. 在【开发环境】页面中，点击左上角【创建开发环境】

4. 在【创建开发环境】页面，【环境配置】-【存储】中，点击【添加】按钮。

5. 此时在下拉条中，可以选择之前已经创建好的数据集，若还未创建数据集，可单击【创建数据集】按钮。

6. 在选择完挂载数据集之后，用户可自定义填入例如“/tmp”类似的容器挂载路径。

7. 若不需要挂载数据集，也可点击右方删除操作，取消挂载数据集。



4.7.3 上传/下载数据

1. 批量计算控制台上传/下载数据，该方法操作简单而且不用额外工具，但是网页不支持上传文件夹且文件大小不能超过 3M。

2. JupyterLab 上传/下载数据，该方法适用于在开发环境中使用 JupyterLab 时上传/下载数据。优点是使用简便，但是只支持文件不支持文件夹，文件大小不能超过 20M。

3. Sftp 上传/下载数据（FileZilla），需安装客户端，支持大文件上传且同时支持文件和文件夹，推荐使用。

4.7.3.1 控制台上传/下载数据

前提条件

1. 数据大小不超过 3M。

2. 准备好需要上传的文件，该方法支持文件不支持文件夹。

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【存储】

3. 在【存储】弹出的选项中，选择【数据集管理】。

4. 选择一个数据集，在操作栏中单击【查看】按钮。

5. 在【存储文件管理】页面点击【上传】按钮。

6. 在弹出的【文件上传】页面中，配置参数，具体如下表所示。

参数	说明
上传路径	上传到数据集中存储的路径，若路径不存在，则会根据所填写的上传路径创建一个路径。
上传文件	可通过点击选择文件或者通过将文件拖到【文件上传】的对应方框中进行上传，文件的大小不超过 3M。

7. 填写必要的参数之后，单击右下角的【确认】，即可上传文件。

8. 在【存储文件管理】页面点击【下载】按钮，进行文件下载。

4.7.3.2 JupyterLab 上传/下载数据

前提条件

1. 已创建接入方式为 JupyterLab 的开发环境。
2. 准备好需要上传的文件，该方法支持文件不支持文件夹，文件大小不能超过 20M。

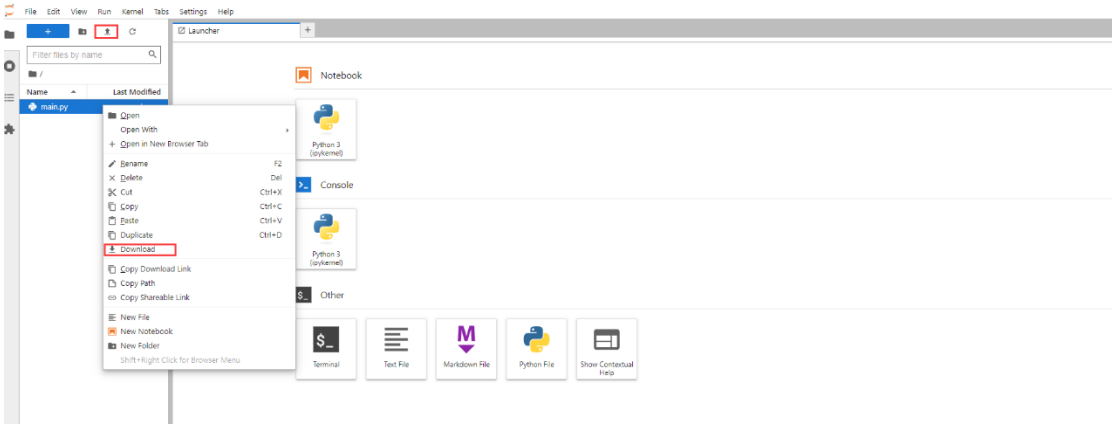
操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【开发环境】。

3. 在【开发环境】页面中，打开一个运行中的 JupyterLab 开发环境。

4. 按照下图操作指引，在 JupyterLab 页面上传或下载数据。



4.7.3.3 Sftp 上传/下载数据（FileZilla）

前提条件

1. 安装好 sftp 客户端工具。这里以 FileZilla 为例，使用以下命令在 linux 系统上进行安装：

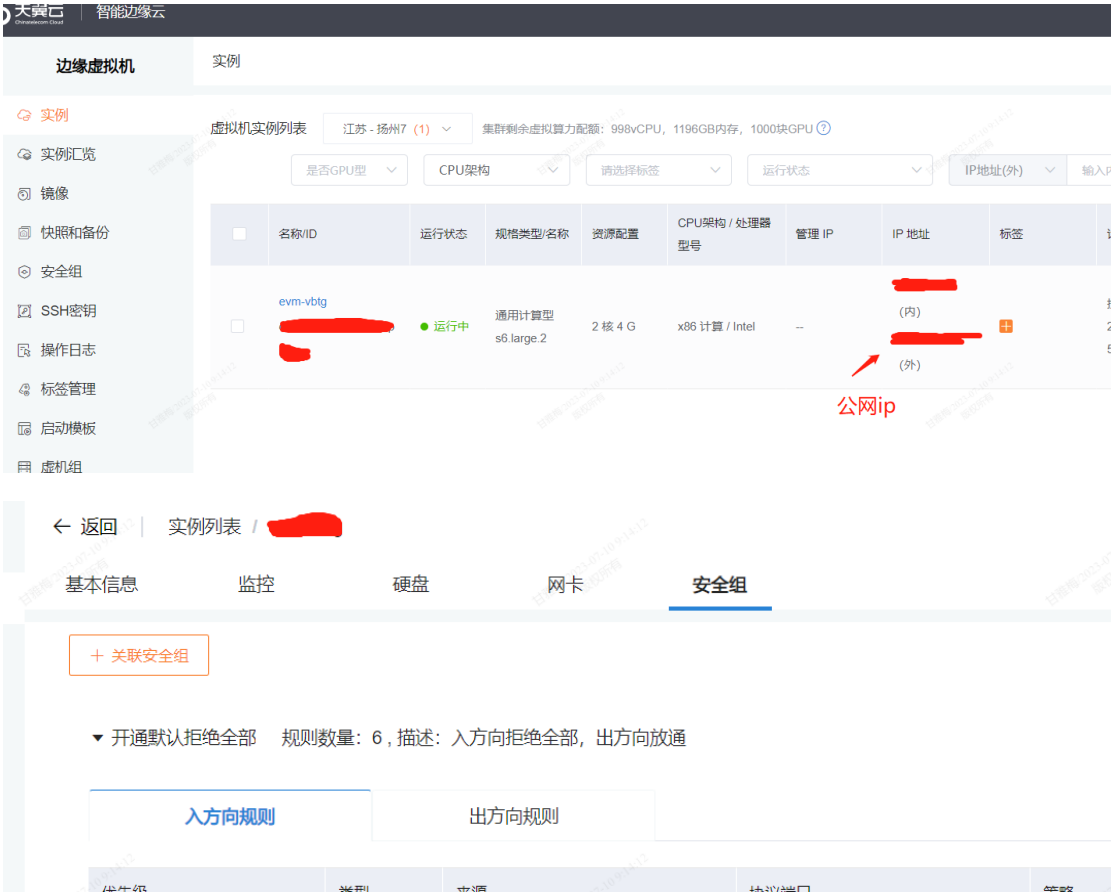
   sudo apt install filezilla

2. 准备好需要上传的文件，该方法支持文件和文件夹上传下载，且支持大文件。

操作步骤

1. 获取连接 SFTP 目标服务器的 IP 和登录方式。以在 ECX 集群创建虚机搭建 NFS 作为存储源为例，获取 IP 和登录方式的步骤如下：

   a. 登录 ecx 控制台，找到部署 nfs 的虚机，获取虚机的公网 ip 及对应安全组。



b. 获取本地的出口 ip（windows 用浏览器访问 http://www.ip138.com/，linux 请求 https://ipinfo.io/）。

c. 在虚机对应的安全组添加入方向规则，将本地的出口 ip 添加到允许的规则中。

d. 若不清楚虚机的登陆密码，可以在虚机页面进行重置，获取密码。

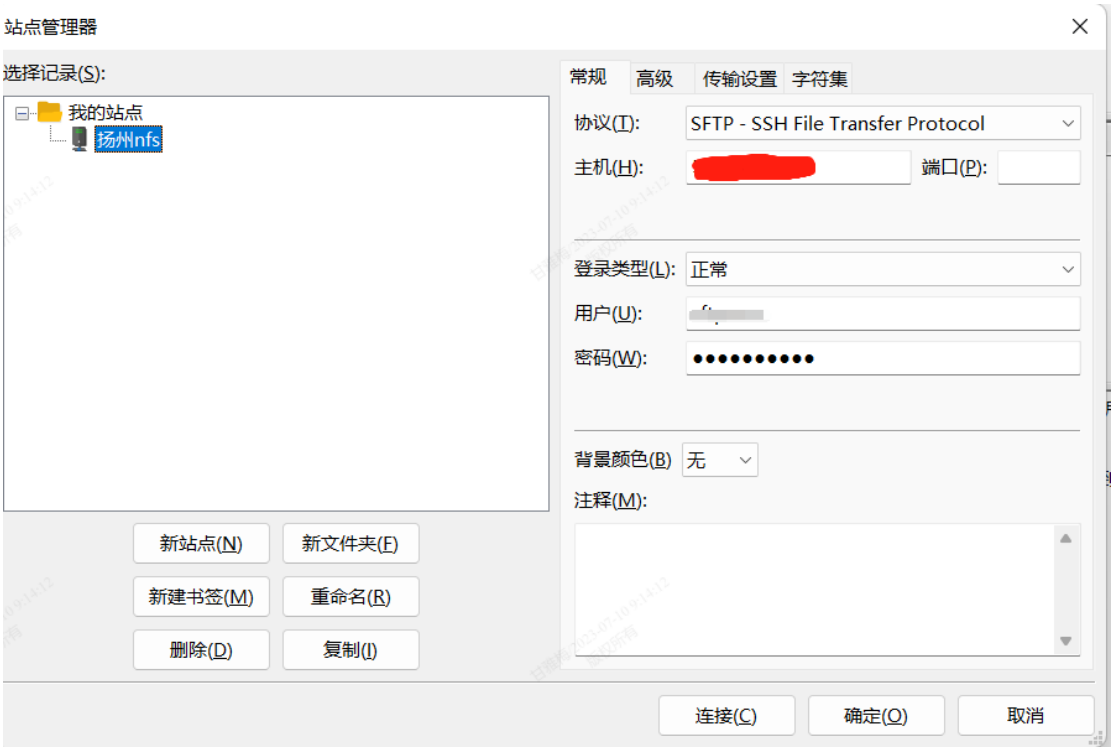
2. 登陆批量计算控制台，在控制台左侧导航栏中，选择【存储】->【数据集管理】。选择目标数据集，获取数据集挂载的目录。

3. 使用 FileZilla 上传/下载数据

   a. 新增站点，配置传输协议 sftp

   b. 输入虚机的公网 ip 及 ssh 端口

   c. 输入虚机登陆的账号及密码进行连接



   d. 连接成功后，找到数据集挂载的目录（即从批量计算平台获取的目录），即可进行文件上传下载。

4.8 调度策略

调度策略是批量计算平台提供的一种调度策略定制化以及可视化的工具，用户可根据自身

场景和需求配置定制调度策略，使得作业平台的资源池的调度器能够适配相关业务。目前，调度策略可在编辑专属资源池的页面中选择并应用。

4.8.1 创建调度策略

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【调度策略】。
3. 在【调度】页面中，点击左上角的【创建策略】按钮。

4. 在【创建策略】页面中，配置参数，具体如下表所示。

参数	说明
名称	输入自定义调度策略名称。要求如下： • 长度范围为 2~64 个字符。 • 名称由任意字符组成，包括中文。
描述	对该调度策略进行描述，由长度为 0~1024 个字符组成。
动作	包括 6 种调度动作，包括入列、分配、抢占、回收、回填以及再平衡。 各个动作和基本作用如下： ● 入列：只有入列作业才会作为调度过程的备选项。当集群提交的作业很多时，可以增加入列过程，预估作业无法调度时，阻止作业创建 Pod，提高集群调度的性能。 ● 分配：通过节点预选过滤不符合要求的节点，通过节点优先对节点进行打分并选出得分最高的节点，并判断作业是否满足就绪条件（比如 Gang 约束）。 ● 抢占与回收：通过公平分享来支持借贷模型，一些作业或者队列在空闲时会过度使用资源。但是，如果有任何进一步的资源请求，资源“所有者”将“收回”。资源可以在队列或作业之间共享：回收（Reclaim）用于队列之

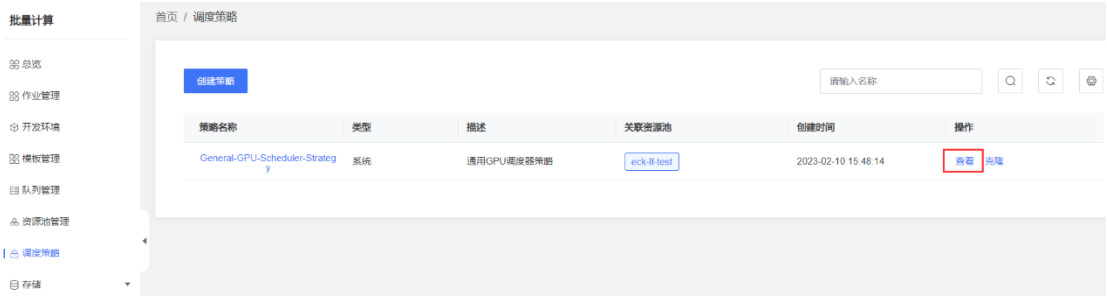
	间的资源平衡，抢占（Preemption）用于作业之间的资源平衡。 ● 回填：通过小资源和未指明资源量的作业填补，忽略队列预留资源配置，尽可能多地将节点的空闲资源分配出去。 ● 再平衡：不同节点间进行负载再平衡。
插件	调度插件，目前共有 14 种调度插件，各个调度插件和其描述如下： ● 最小碎片调度：尽量将 Pod 绑定到资源利用率高的节点上， 以减少碎片化。 ● 关键资源保护：跳过关键 Pod，而不是驱逐它们。 ● DRF 调度：【公平调度】确保在多种类型资源共存的环境 下,尽可能满足分配的公平原则。 ● 批调度：【批调度】为作业分配资源时，重点考虑作业的最低资源需求和 pod 最小运行数量，执行“All or nothing”的调度策略。 ● 节点打分：通过用户配置的打分参数来从各个维度为节点打分，从而找到最适合当前作业的节点 ● Numa 感知：【numa 感知】在将 pod 绑定到 node 节点时重点考虑 CPU Numa 因素。（需额外部署采集程序）超配调度： ● 初选调度：将可用资源设置为集群整体资源的指定倍数。 ● 优先级调度：【优先级调度】定义调度作业的优先级。 ● 资源比例调度：【资源比例】根据 queue 的配置，将集群的整个资源按比例划分到所有的 queue。 ● 服务等级约束：【服务水平】根据 SLA 设置对工作负载进行排序。 ● 任务拓扑：【应用拓扑感知】根据给定策略，将不同角色的 Pod 绑定到节点上。 ● 分时复用：【分时复用】在不同的时间段内，允许部分节点承接 K8s 和其他集群的作业调度。 ● 重调度：【重平衡】周期性评估和平衡节点间资源利用率的合理性，并驱逐不合理的作业。

5. 在用户自定义选择搭配好调度策略之后，点击右下角的【确认】，即可完成调度策略的创建。

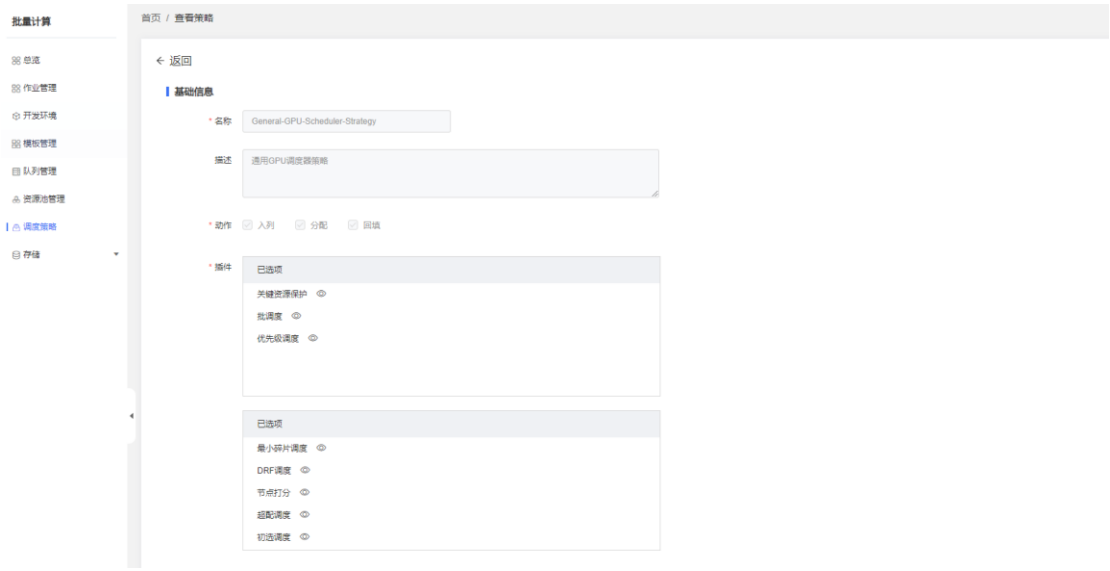
4.8.2 查看调度策略

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【调度策略】。
3. 在【调度策略】页面中，点击调度策略的名称或者右边的【查看】按钮。



4. 在【查看策略】页面中，即可查看该调度策略所选择的参数。



4.8.3 编辑调度策略

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【调度策略】。

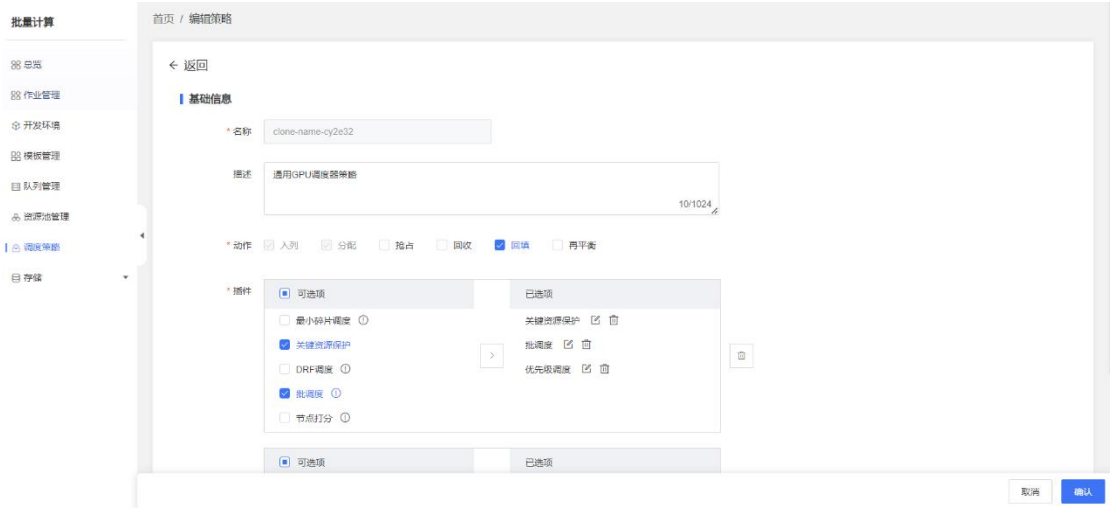
3. 在【调度策略】列表页面中，点击目标策略右侧操作栏的【编辑】按钮。

4. 在【编辑策略】页面中，配置参数，具体如下表所示。

参数	说明
描述	对该调度策略进行描述，由长度为 0~1024 个字符组成。
动作	包括 6 种调度动作，包括入列、分配、抢占、回收、回填以及再平衡。 各个动作和基本作用如下： ● 入列：只有入列作业才会作为调度过程的备选项。当集群提交

	的作业很多时，可以增加入列过程，预估作业无法调度时，阻止作业创建 Pod，提高集群调度的性能。 ● 分配：通过节点预选过滤不符合要求的节点，通过节点优先对节点进行打分并选出得分最高的节点，并判断作业是否满足就绪条件（比如 Gang 约束）。 ● 抢占与回收：通过公平分享来支持借贷模型，一些作业或者队列在空闲时会过度使用资源。但是，如果有任何进一步的资源请求，资源“所有者”将“收回”。资源可以在队列或作业之间共享：回收（Reclaim）用于队列之间的资源平衡，抢占（Preemption）用于作业之间的资源平衡。 ● 回填：通过小资源和未指明资源量的作业填补，忽略队列预留资源配置，尽可能多地将节点的空闲资源分配出去。 ● 再平衡：不同节点间进行负载再平衡。
插件	调度插件，目前共有 14 种调度插件，各个调度插件和其描述如下： ● 最小碎片调度：尽量将 Pod 绑定到资源利用率高的节点上， 以减少碎片化。 ● 关键资源保护：跳过关键 Pod，而不是驱逐它们。 ● DRF 调度：【公平调度】确保在多种类型资源共存的环境 下,尽可能满足分配的公平原则。 ● 批调度：【批调度】为作业分配资源时，重点考虑作业的最低资源需求和 pod 最小运行数量，执行“All or nothing”的调度策略。 ● 节点打分：通过用户配置的打分参数来从各个维度为节点打分，从而找到最适合当前作业的节点 ● Numa 感知：【numa 感知】在将 pod 绑定到 node 节点时重点考虑 CPU Numa 因素。（需额外部署采集程序）超配调度： ● 初选调度：将可用资源设置为集群整体资源的指定倍数。 ● 优先级调度：【优先级调度】定义调度作业的优先级。 ● 资源比例调度：【资源比例】根据 queue 的配置，将集群的整个资源按比例划分到所有的 queue。 ● 服务等级约束：【服务水平】根据 SLA 设置对工作负载进行排序。 ● 任务拓扑：【应用拓扑感知】根据给定策略，将不同角色的 Pod 绑定到节点上。 ● 分时复用：【分时复用】在不同的时间段内，允许部分节点承接 K8s 和其他集群的作业调度。 ● 重调度：【重平衡】周期性评估和平衡节点间资源利用率的合理性，并驱逐不合理的作业。

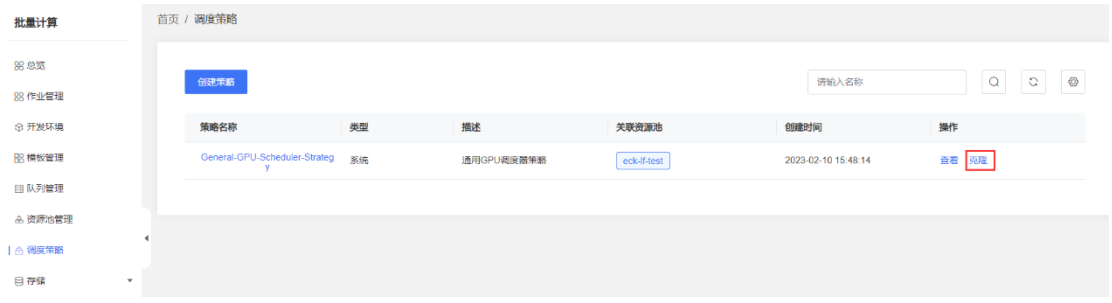
5. 在用户自定义选择搭配好调度策略之后，点击右下角的【确认】，即可完成调度策略的编辑。



4.8.4 克隆调度策略

操作步骤

1. 登录批量计算管理控制台。
2. 在控制台左侧导航栏中，选择【调度策略】。
3. 在【调度策略】页面中，选择某个调度策略右边的【克隆】按钮。



4. 在【克隆策略】页面中，可以快速拷贝一份调度策略，用户可以在克隆策略的基础上，新增或者减少对应的动作和插件，具体操作见创建调度策略。

5. 在【克隆策略】页面中，点击右小角的【确认】，即可成功克隆一份调度策略。

4.8.5 删除调度策略

操作步骤

1. 登录批量计算管理控制台。

2. 在控制台左侧导航栏中，选择【调度策略】。

3. 选择一个调度策略，在右边单击【删除】。

4. 在二次确认的弹框输入框中输入“确认删除 “，并且点击【确认】完成删除。

5 常见问题

5.1 定义类

5.1.1 批量计算适用哪些场景?

- HPC：适用于视频渲染、视频转码（视频格式转换、视频分辨率变化、添加水印/logo 的）、科研教育等领域。
- AI/ 大数据：适用于内容审核、OCR、图像识别、图片处理、美颜、语音识别、推荐、搜索、智能客服、游戏 AGI 等领域。
- 生物分析：适用于基因测序、药物检测等领域。

5.1.2 批量计算的核心功能有哪些?

- **多资源池管理：** 底层资源池 ECK 专有资源、Serverless 共享容器资源池、ECX 虚拟机资源池满足不同价格区间的业务需求。
- **作业调度：** 提供了强大的作业管理及作业调度能力，用户创建批量计算作业即可运行各种类型的工作负载。大规模数据及工作负载分散到多个计算节点上并发执行，提高数据处理速度，缩短交付时间，支持断点重启。
- **作业模板：** 支持创建作业模板，作业由至少一个工具组成。作业中的各个工具由其前后顺序关系形成数据流，前序工具为后序工具提供输。
- **队列管理：** 提供对作业队列管理，支持 CPU/GPU 作业和队列管理。
- **开发环境：** 以云原生的资源使用和开发工具链的集成，为不同类型 AI 开发、探索、教学用户，提供更好云化 AI 开发体验。

5.1.3 批量计算的专属资源池和共享资源池的区别是什么?

资源池是批量计算服务所需要使用的计算资源的集合，分为共享资源池、专属资源池。

- 专属资源池：即该用户专用，用户在边缘容器集群（简称 ECK）上创建自己的 K8s 集群，拥有该 K8s 集群和集群内的 ECX 虚机的所有权限。在批量计算平台上创建专属资源池时绑定 ECK 集群，1 个专属资源池对应 1 个 ECK 集群。
- 共享资源池：用户无需预先在 ECK 创建集群，底层的 K8s 集群由平台统一管理和维护，集群用户共享。用户可以在共享集群上创建虚拟的共享资源池运行计算作业。

5.2 购买开通类

5.2.1 批量计算支持哪些资源池?

- 专属资源池：只要客户在边缘容器集群（简称 ECK）上创建出 K8s 集群并正常运行，即可加载到批量计算上使用。I
- 共享资源池：目前控制台开放的区域为华南一区，如您有其他区域的使用需求，请联系您的客户经理，或线上咨询或拨打客服热线电话寻求帮助，热线话：400-810-9889 转 1。后续会随业务需求和资源布局会逐步开放自助区域。

5.2.2 批量计算是否支持试用?

支持试用，如需试用请联系您的客户经理进行申请。

5.2.3 批量计算支持哪些计费方式?

支持按需计费。

5.2.4 批量计算服务开通是否需要收费?

服务开通不收取费用。

5.2.5 批量计算收取哪些费用?

- 专属资源池收取边缘容器集群 ECK 的费用。
- 共享资源池收取部署计算任务占用的资源（例如 CPU、内存、GPU 等）费用。如用到天翼云其他产品，需另外支付对应产品费用。

5.3功能类

5.3.1 部署业务前有哪些前置准备工作?

首次使用批量计算服务，用户需要先创建资源池和队列，作业、开发环境等都需要归属到队列内。如需要使用自定义镜像功能，需提前登录容器镜像服务（CRS）设置访问凭证并创建私有组织。

5.3.2 批量计算的开发环境可以提供哪些快捷工具?

当前可提供 Jupyter 和 VSCode 在线代码编辑器，同时用户可自行安装所需的其他插件。

5.3.3 批量计算的开发环境支持哪些公共框架?

当前支持 Pytorch，Tensorflow 和 PaddlePaddle。

5.4 技术运维问题

5.4.1 作业运行常见问题

1. 作业提交后，为什么没有立刻启动?

   作业提交后会根据优先级获取或者排队等待资源，等待时间依资源池繁忙程度而定。

   有可能出现作业中某些实例获取了资源开始运行，其它实例还在等待的情况。

2. 实例执行结束后，其中的数据是否还在?

   实例执行结束后，其中的数据将被销毁，下次启动时恢复为初始镜像状态。

3. 通过批量计算提交的作业，是否支持节点间通信?

   同一个集群内的实例，可以通过内网 IP 进行通信，不同集群间实例不能互相通信。

5.4.2 调试和查错常见问题

1. 如何进行开发调试?

批量计算提供了开发环境，可现在开发环境进行单机调试。

2. 如果程序执行失败怎么调试?

程序失败时可以通过查看作业和实例事件获取更多的作业运行信息。如果问题还未能解决，请联系售后技术支持。