华北电力大学高性能计算平台项目

AI 部分使用手册

阿里云计算有限公司

日　　期：　2023 年 12 月 5 日

目　录

1.1 进入工作空间

用户登录平台后，自动进入一个**工作空间**，并打开该**工作空间**的**概览**页，显示缺省的**计算类型组**下的资源与任务的统计情况。如果用户拥有多个**工作空间**的权限，那么还可以通过控制台左上角的下拉框切换**工作空间**。如果该工作空间下有多个**计算类型组**，可以通过概览页右上角的**计算类型组**下拉框切换。

一. 工作空间概览

说明

1. 用户能够拥有多少个工作空间的权限，取决于他所隶属的项目，一个项目可以与一个或多个工作空间相关联，从而允许该项目内的成员使用对应的工作空间的资源。
2. 上图左侧菜单项是平台开发者功能的全集，其中工作空间管理功能只有工作空间管理员才可以看到。
3. 在概览页，可以看到在当前工作空间下某个计算类型组内的资源与任务

情况。用户可以通过计算类型组下拉框切换不同的计算类型组，从而查看对应计算类型组的信息。这些信息包括：

4. 计算资源的用量与总量，包括 CPU、内存、GPU。
5. 资源水位概览，包括 CPU、内存、GPU，还可以通过选择图表右上角的下拉框选择资源统计的间隔时间。
6. 任务状态统计：按照任务不同状态统计的任务的数量。
7. 在概览页上有多个视图的页签，分别展示不同视图下的工作空间信息，包括：

二. 节点视图

节点视图：展示本工作空间下每个节点的基本信息、资源用量、I/O 读写速度、关联用户等信息。

三. 用户视图

用户视图：展示本工作空间下每个用户的资源使用情况、I/O 读写速度、节点使用情况等信息。

四. 任务视图

任务视图：展示本工作空间下每个任务的资源使用情况、I/O 读写速度、节点使用情况等信息。

重要

- PAI 灵骏智算平台通过工作空间来隔离和管理不同的资产和任务。如果您拥有多个工作空间的权限，那么需要通过控制台左上角的工作空间下拉框选项，切换到不同的工作空间分别开发和管理。
- 您在不同工作空间下看到的内容是不同的，因此在进行开发和管理时，需要随时关注您当前工作在哪个工作空间下。

1.2 准备数据

本节介绍如何准备任务运行所需要的数据。

您可以将任务运行所需的数据上传到指定的存储路径下。上传和管理数据的途径有两种：

一. 文件管理

通过资产管理->文件管理，进入文件管理页面。一般情况使用这种方式即可，您可以方便地将本地文件上传到自己有权限的文件目录下。

二. 数据集管理

通过资产管理->数据集管理，进入数据集配置页面。如果您希望通过物理机

的挂载点访问数据，则通过这种方式配置数据。

1.3 AI 开发环境开发与调试

PAI 灵骏智算平台提供交互式 AI 开发环境 DSW（Data Science Workshop），用于开发、调试 AI 程序。

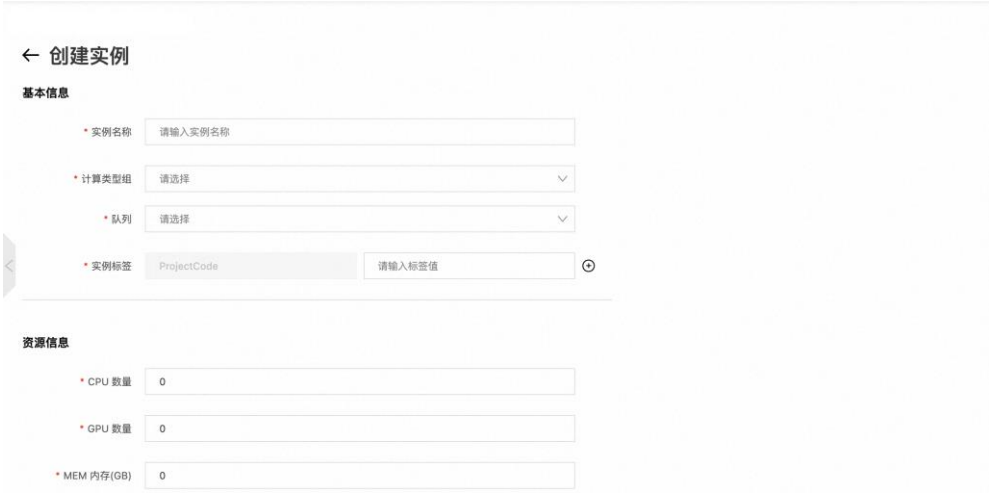
一. DSW 实例列表

在 PAI 灵骏智算平台控制台点击左侧菜单开发环境-> AI 开发环境（DSW），可以查看 AI 开发环境（DSW）实例列表：

二. 创建 DSW 实例

在交互式建模（DSW）页面，单击创建实例。在创建实例页面，填写基本信息、资源信息、存储配置。

- 基本信息



（1）基本信息

您可以在基本信息区域填写任务的基本信息，这些字段是任务的元数据，需要提供的字段和字段的取值范围说明如下表格：

参数	描述
实例名称	创建的开发环境实例名称只能包含英文字母、数字和下划线，且不能超过 27 个字符。
计算类型组	可以从下拉菜单中选择一个计算类型组。计算类型组与调度器相关联，选定了某种调度器后自动对计算类型组做筛选，并限制该用户的计算资源限制情况，包括各类资源的总量和当前已用量。
队列	从下拉框选择您可以使用的队列的名字。
实例标签	实例标签目前包括固定的“ProjectCode”以及其它通用标签对。 总共最多支持 20 个标签对，每个标签名和标签值最大长度为 20 个字符。 固定的“ProjectCode”用于设置作业所属的计费的项目，您可以从 ProjectCode 对应的标签值下拉框中选择项目。 ProjectCode 是必选的，通用标签是可选项。

说明

计算类型组是由作业中心的工作空间管理员创建的，如果您在相应字段中没有可选值，请联系您的工作空间管理员帮助创建。

实例标签的 Project Code 的值是运营中心管理员管理和维护，如果您在提交任务时下拉列表中没有可选值，请联系运营中心管理员创建。

队列是您申请开发环境资源参与排队的通道，您可以选择希望参与排队的队列，如果在下拉框列表中没有可选队列，请联系您的工作空间管理员。

关于这些参数的详细说明，您可以参考基本概念部分。

（2）资源信息

参数	描述
CPU 数量	开发环境需要占用的 CPU 核心数量。
GPU 数量	开发环境需要占用的 GPU 数量。
MEM 内存（GB）	开发环境需要占用的内存大小，单位为 GB。
节点名称	您可以通过指定节点的方式来创建 DSW 实例， 不填则表示不指定节点。
镜像来源	开发环境使用的 Docker 镜像，支持官方镜像和自定义镜像。 如果使用自定义镜像，您需要提供 Docker 镜像地址。例如，master0:5000/pai-dsw/tensorflow-torch:tf2.3torch1.8-gpu-py36-cu101-ubuntu18.04。

（3）存储配置

参数	描述
存储配置-数据集	您可以选择已经创建的数据集进行挂载。此项非必选。
存储配置-本地盘挂载	如果选择了指定节点创建 DSW，那么可以配置本地来源路径和容器内路径。此项功能

	位高阶功能，建议谨慎使用。

关于存储配置的说明：

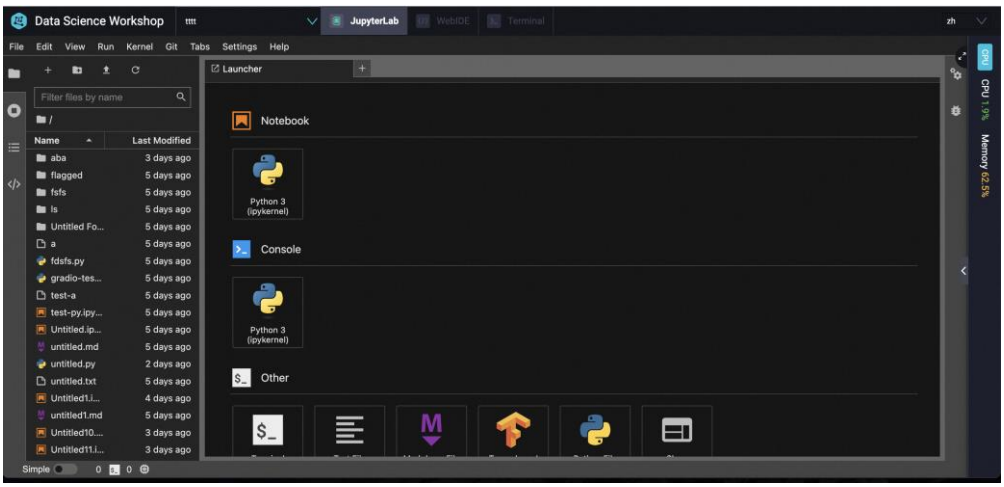
一般情况下您不需要选择专门的数据集，只需要通过文件管理上传自己的数据即可。

如果您希望通过物理机挂载点方式访问数据集，则从下拉列表中选择您提前创建好的数据集，您可以通过数据集配置页面创建数据集。

填写完创建 DSW 实例所需的信息后，点击创建，回到 DSW 实例列表页。当 DSW 实例的状态变为“运行中”后，就可以打开并进入 DSW 开发环境。

三. 进入 DSW 开发环境

单击状态处于“运行中”的实例操作列下的打开，即可进入相应的开发环境。DSW 开发环境基于 Jupyter Notebook，您可以在开发环境中打开 Terminal，或者运行 Notebook，实现交互式 AI 建模。



说明

进入 DSW 开发环境后，左侧目录缺省打开的目录，就是与您创建实例时选择的 ProjectCode（项目）所对应的个人存储目录。您可以提前在文件管理页面上传您进行开发所需要的数据。

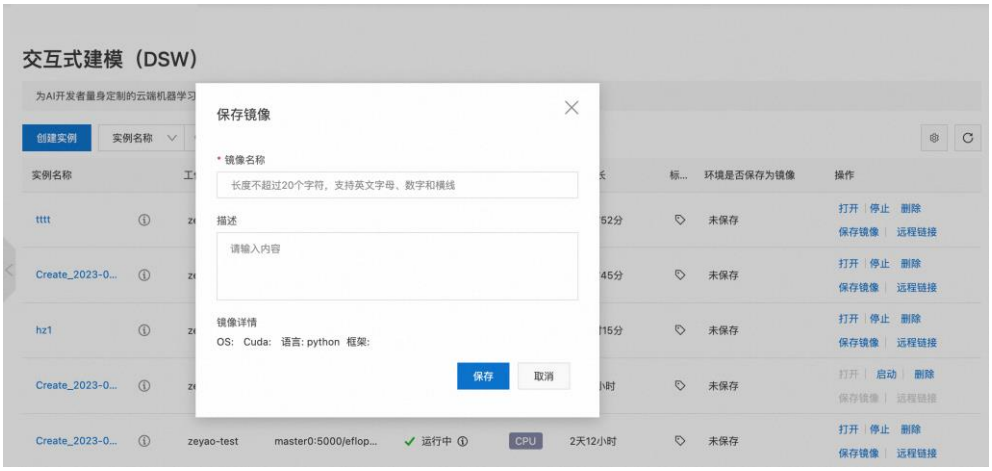
四. DSW 实例操作

（1）保存镜像

您可以将一个DSW开发环境实例保存为镜像，并在其它任务中使用该镜像。具体做法如下：

在 DSW 实例列表页，单击目标实例操作列下的保存镜像，即可将开发环境保存为镜像。

将鼠标悬停在目标实例保存于后面的问号图标，即可显示保存的详细信息。



说明

根据生成的镜像仓库地址，您可以基于当前保存的镜像创建一个新实例。在创建实例时，在镜像来源输入框填入该镜像地址即可。此外，您也可以基于该镜像仓库地址，在 AI 任务管理中创建一个新的 AI 任务，具体参见创建 AI 任务部分。

（2）停止实例

在 DSW 实例列表页，单击目标实例操作列下的停止。

在弹出的对话框中，单击保存环境然后停止或直接停止，即可停止运行中的实例，如下图所示：

实例停止后，其状态会切换为**停止**。

说明

两种停止方式的区别如下，您可以根据实际情况选择：

- **保存环境然后停止**：下次重启实例时，系统会恢复上次关闭时的开发环境。如果您对默认环境进行了修改，比如安装了新的包，那么建议选择该方式。
- **直接停止**：如果未修改默认环境，通常选择该方式停止实例。

（3）删除实例

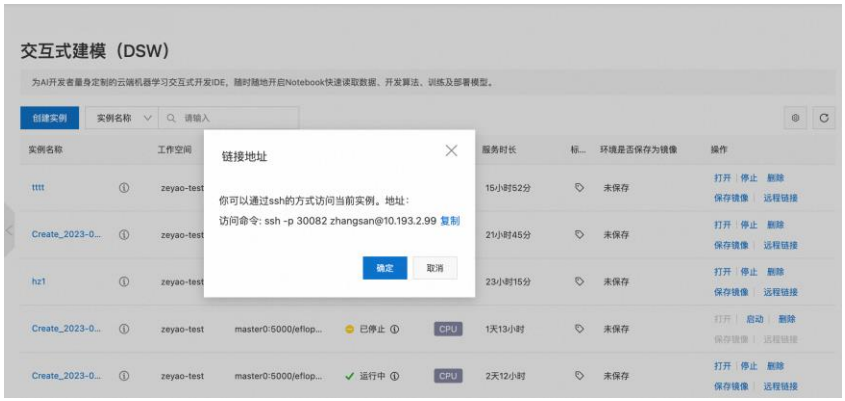
如果开发环境实例不再需要，则可以删除该示例。注意：删除后的实例不可恢复。

1. 在 DSW 实例列表页，单击目标实例**操作**列下的**删除**。
2. 在删除确认对话框中，单击**确定**，即可删除该实例。

五. 远程连接进入 DSW 实例

（1）获取 SSH 连接方式

在 DSW 实例列表页面，单击目标实例**操作**列下的**远程链接**。在**链接地址**对话框中，单击访问命令后的**复制**，即可获取连接到实例的 SSH 命令。



（2）控制台连接 SSH

1.打开支持 SSH 连接的 Windows、Linux 或 Mac 控制台。

2.将连接到实例的 SSH 命令输入至控制台中，并按回车键确认。关于如何获取连接到实例的 SSH 命令，请参见以上获取 SSH 连接方式。

3.根据提示信息输入密码，即可连接成功。

重要

通过 SSH 登录实例所需要的用户名和密码，是您登录作业中心所使用的用户名和密码，如果登录不成功，请联系运营中心管理员。

（3）VS Code 连接 SSH

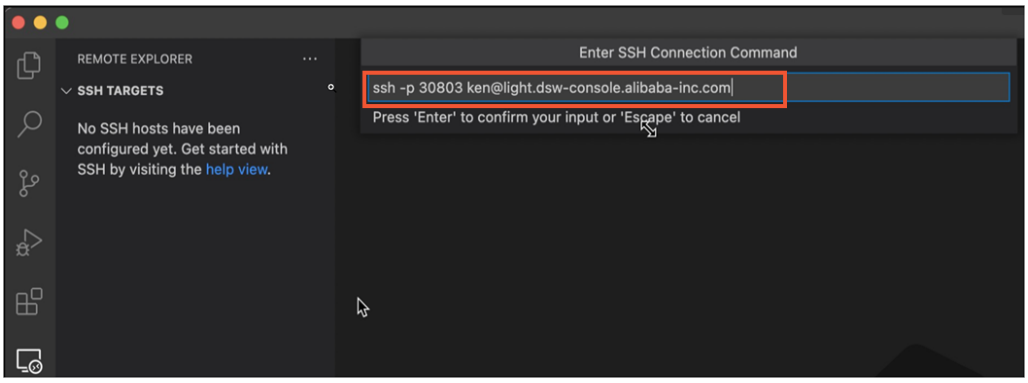
1.打开 Visual Studio Code，单击左侧的 Remote Explorer 菜单。

2.单击右上方的添加。

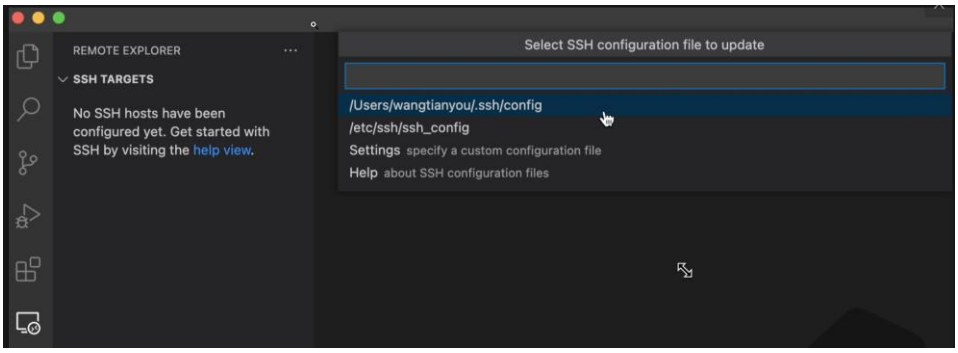
3.将连接地址输入到 Command 中，如下图所示。关于如何获取连接到实例的 SSH 命令，请参见以上获取 SSH 连接方式。

说明

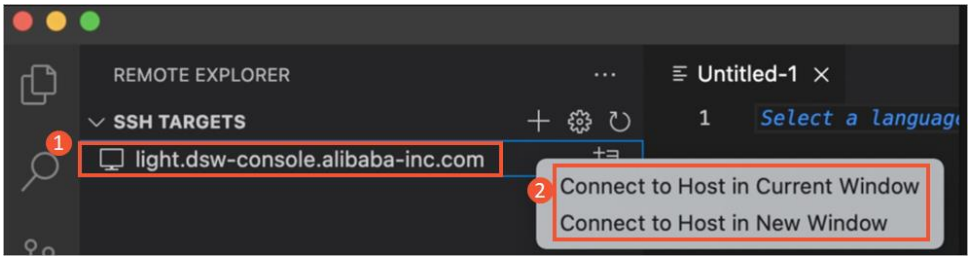
请用以上获得的 SSH 链接地址，代替以下红框标出来的链接地址。



4.选择一个配置文件进行保存。



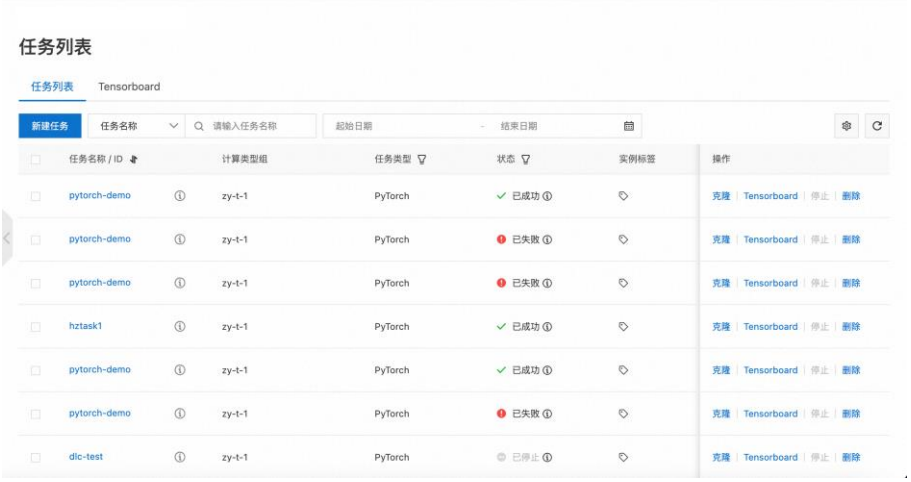
5.在 Remote Explorer 中会出现一台新的 Host 主机，您可以右键单击该主机，在弹出的快捷菜单中选择连接呈现方式，并根据提示输入密码。



1.4 提交任务

一. 任务列表

进入 PAI 灵骏平台控制台后，您可以点击左侧菜单栏**任务管理->AI 任务管理**，打开 AI 任务列表：



二. 创建任务

创建 AI 任务的步骤如下：

（1）新建任务

在 **AI 任务列表**页面，点击“**新建任务**按钮，可以创建新任务，如下图所示：

（2）填写基本信息

您可以在**基本信息**区域填写任务的基本信息，这些字段是任务的元数据，需要提供的字段和字段的取值范围说明如下表格：

参数	描述
计算类型组	可以从下拉菜单中选择一个计算类型组。计算类型组与调度器相关联，选定了某种调度器后自动对计算类型组做筛选，并限制该用户的计算资源限制情况，包括各类资源的总量和当前已用量。
任务队列	从下拉框选择您可以使用的任务队列的名字。
实例标签	实例标签目前包括固定的“ProjectCode”以及其它通用标签对。 总共最多支持 20 个标签对，每个标签名和标签值最大长度为 20 个字符。 固定的“ProjectCode”用于设置作业所属的计费的项目，您可以从 ProjectCode 对应

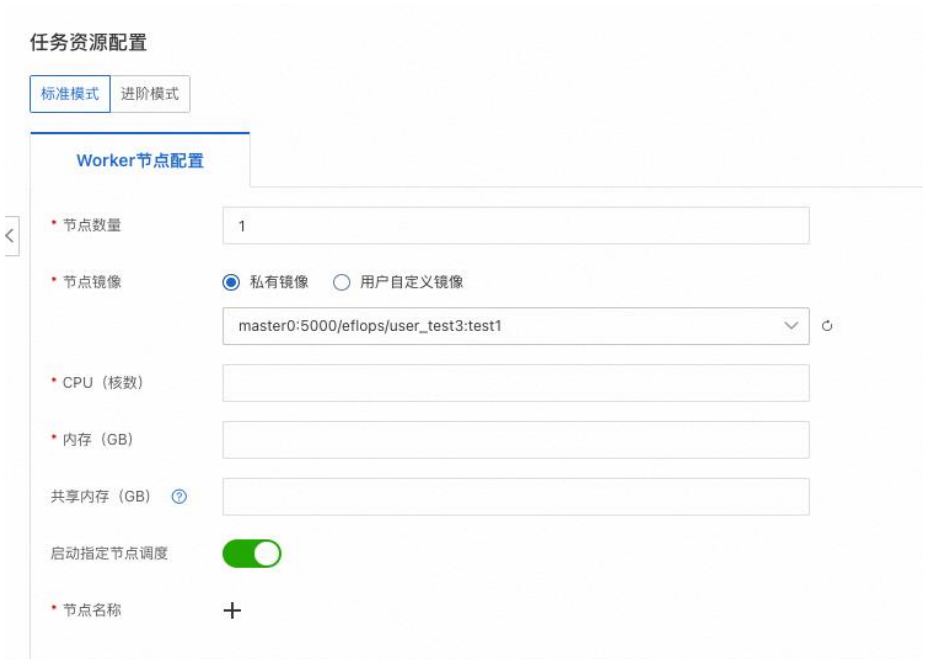
	的标签值下拉框中选择项目。 ProjectCode 是必选的，通用标签是可选项。
任务名称	任务名称为字符串，长度不得超过 20 字符
任务类型	您可以选择一种任务类型，包括 TensorFlow，PyTorch，XGBoost，Batch 批处理。未来将扩展到更多类型。
数据集配置	您可以选择已经创建的数据集进行挂载。此项非必选。
执行命令	填写任意 Shell 命令，例如使用 python -c "print('Hello World')"运行 Python。
容错配置	打开后启动容错训练功能，即如果任务失败可以自动重启实例。具体用法参见容错训练部分。

说明

- **计算类型组**是由作业中心的工作空间管理员创建的，如果您在相应字段中没有可选值，请联系您的工作空间管理员帮助创建。
- **实例标签**的 Project Code 的值是运营中心管理员管理和维护，如果您在提交任务时下拉列表中没有可选值，请联系运营中心管理员创建。
- **任务队列**是您申请开发环境资源参与排队的通道，您可以选择希望参与排队的队列，如果在下拉框列表中没有可选队列，请联系您的工作空间管理员。
- **数据集配置**：
  - 一般情况下您不需要选择专门的数据集，只需要通过文件管理上传自己的数据即可。
  - 如果您希望通过物理机挂载点方式访问数据集，则从下拉列表中选择您提前创建好的数据集，您可以通过数据集配置页面创建数据集。

（3）配置任务资源

标准模式



参数	描述
节点数量	当前镜像和规格的节点数量。
节点镜像	工作节点的镜像，包括私有镜像和自定义镜像。对于私有镜像可以从镜像列表中选择。对于自定义镜像，填入镜像链接即可，如 master0:5000/kuande/deepops/bert:rdma-pytorch-mnist
CPU (核数)	节点 CPU 核数。
内存 (GB)	节点内存大小。
共享内存 (GB)	Shared Memory 内存设置。
启动指定节点调度	如果不打开，则系统自动分配节点。如果打开，可以点击“+”号在弹出对话框中选择需要的节点。
最长运行时长	设置任务的最长运行时长。
实例保持时长	设置实例结束后还需要保留的时长。

进阶模式

进阶模式支持指定节点单独配置参数，包括 Worker、PS、Chief、Evaluator、GraphLearn 等类型的节点。配置参数说明与“标准模式”相同。

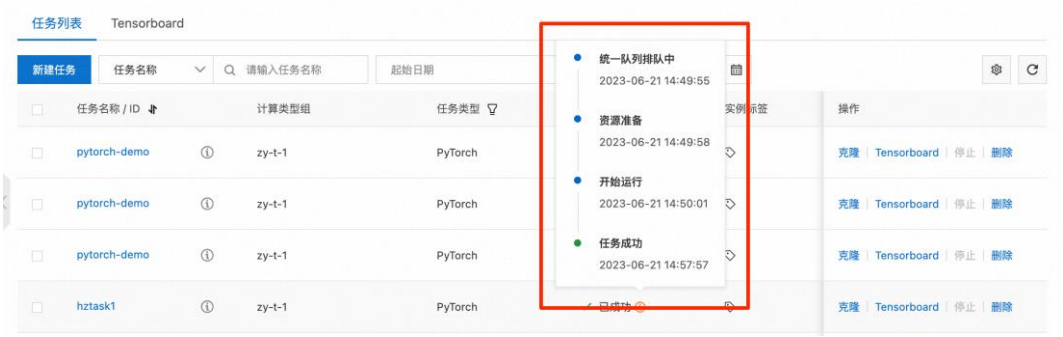
（4）提交任务

完成以上信息填写后，点击**提交**按钮，提交任务，并回到 AI **任务列表**页。

三. 管理任务

（1）任务状态

在 AI **任务列表**页，可以查看当前用户在所有工作空间下的所有任务和对应的状态。您可以将鼠标悬停至**任务状态**后的感叹号图标，概览任务生命周期流转状态，新创建的任务会经历队列排队、资源准备、开始运行、任务成功等阶段。如下图所示：



（2）克隆任务

在 AI 任务列表查看任务时，可以单击任务的**操作**列下的**克隆**，即可重新创建一个与该任务完全相同的任务。

（3）停止任务

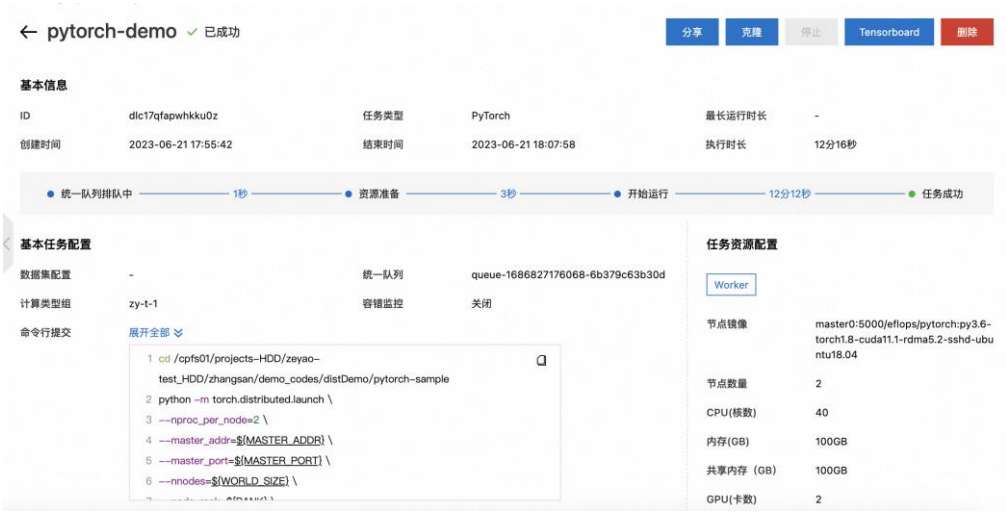
在 AI 任务列表查看任务时，可以单击正在运行的任务的**操作**列下的**停止**，即可停止该任务。

四. 任务详情

（1）详细信息

在 AI 任务列表，单击任务的任务名称进入任务详情页面，即可查看任务基本信息包括

ID、任务类型、创建时间、结束时间、运行时长、基本任务配置、任务资源配置，以及任务配置参数等详细信息：

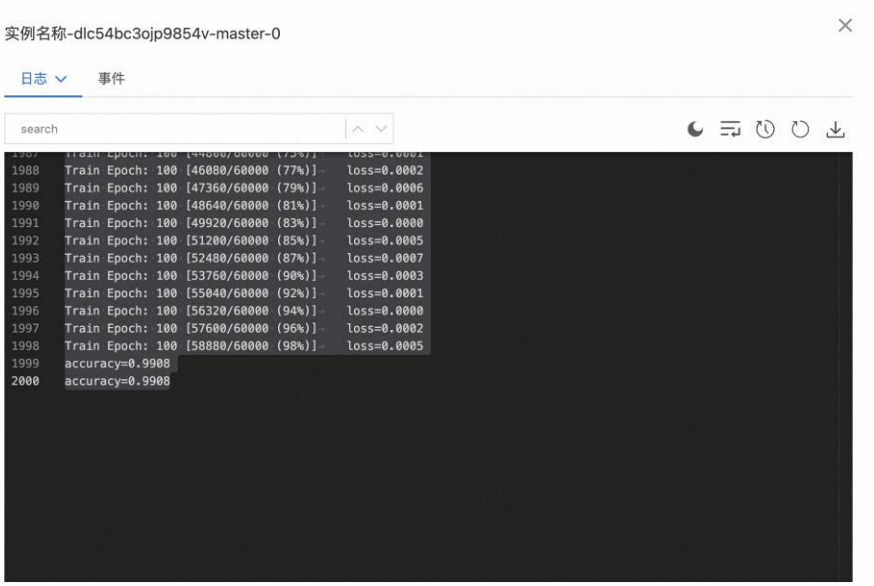


（2）实例日志

在任务详情页面底部的**实例**页签，可以查看该任务下的所有实例的列表。

单击**事件**页签则打开事件信息。单击**资源**页签查看资源使用的相关指标。

单击实例**操作**列下的**日志**，即可查看详细的任务日志和事件。



五. Tensorboard

（1）创建 TensorBoard

您可以通过一下任意一种方式进入创建 Tensorboard 对话框。

1. 在 AI 任务列表页，单击某个运行中的任务对应**操作**列下的 **Tensorboard**。

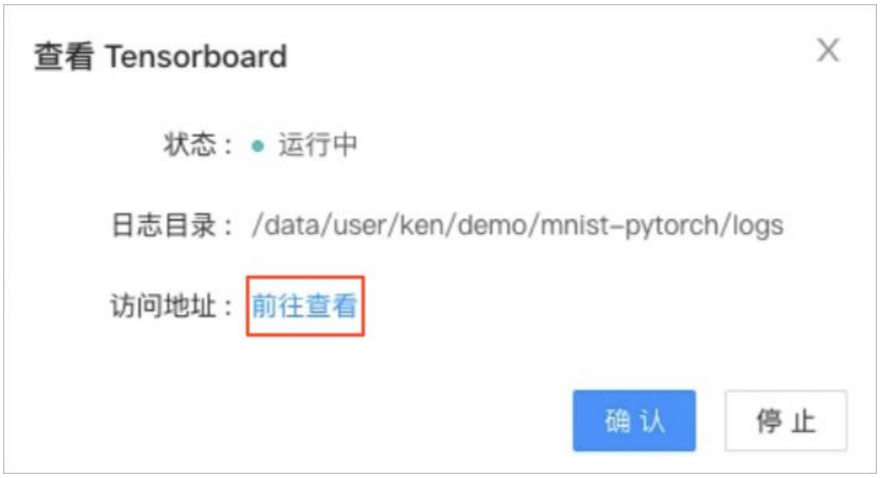
2. 通过任务详情页面进入：在任务详情页面，单击右上方的 **Tensorboard**。

（2）使用 TensorBoard

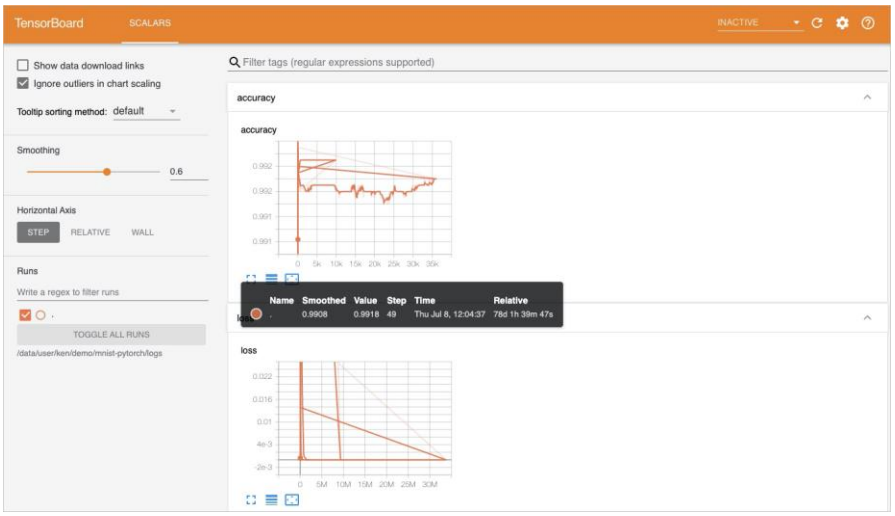
1. 在**创建 Tensorboard** 对话框，输入 TensorBoard 的**名称**和 **Summary 文件路径**。

2. 单击**确认**，即可启动 Tensorboard 服务。

Tensorboard 服务运行后，弹出**查看 Tensorboard** 对话框，单击**前往查看**即可跳转到对应页面。



3.Tensorboard 服务运行后，在**操作**列单击查看 Tensorboard 也可以打开 Tensorboard 页面，如下图所示：



1.5 通知管理

PAI 灵骏智算平台 AI 任务支持基于钉钉 Webhook 或企业微信 Webhook 进行消息推送，您可以通过创建通知渠道并关联通知规则，实现消息推送，更及时的跟踪您的训练任务状态的变化。本节介绍如何创建通知渠道及关联通知规则。

一. 创建通知渠道

下面以钉钉 Webhook 为例，介绍如何创建通知渠道，具体操作步骤如下所示。

1.登录到 PAI 灵骏智算平台。

2.按照下图操作指引，进入**通知渠道**页面。

PAI-DLC 支持基于钉钉 Webhook 或企业微信 Webhook 进行消息推送，您可以通过创建通知渠道并关联通知规则，实现消息推送，更及时的跟踪您的训练任务状态的变化。本文为您介绍如何创建通知渠道及关联通知规则。

3.在**通知渠道**页签，单击**新建通知渠道**。

4.在**新建通知渠道**对话框，配置以下参数。

参数	描述
渠道名称	通知渠道的名称。
渠道类型	支持以下两种渠道类型： DingTalkWebhook WechatWebhook
渠道地址	如果您选择的是 Webhook 类型，此处填写 access_token。关于如何获取 access_token，详情请参见附录：获取 Access_token。

二. 关联通知规则

通知规则是指您希望在何种条件下，触发何种消息渠道的通知，关联通知规则的操作步骤如下所示。

1. 按照下图操作指引，进入**通知规则**页面。

2. 在**通知规则**页签，单击**新建通知规则**。

3. 在**新建通知规则**对话框，配置以下参数。

其中：

- **所属工作空间**：只通知该工作空间内的训练任务的状态。
- **通知渠道**：选择您创建的通知渠道。

4. 单击**确定**。

当通知规则创建成功后，**启用状态**默认为**开启**状态。当训练任务状态变化时或子任务状态失败时，您会收到钉钉消息通知。

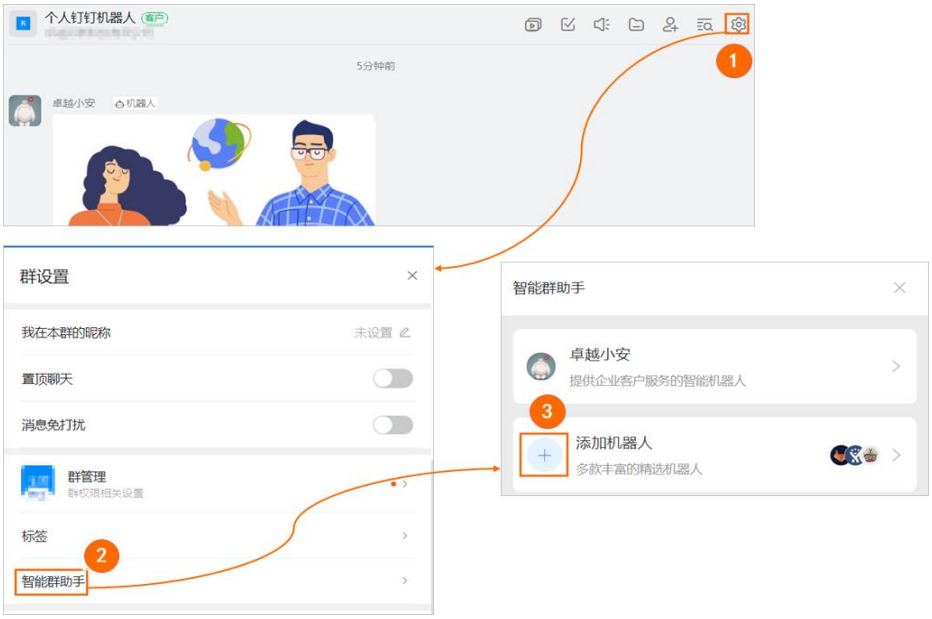
三. 附录：如何获取 Access_token

下面以钉钉群组机器人为例，介绍如何获取 access_token，具体操作步骤如下所示。

1. 在钉钉页面，按照下图操作指引，发起群聊。

2. 在**发起群聊**对话框，输入**群名称**并选择**企业名**，单击**立即创建**。

3. 在**个人钉钉机器人**群聊页面，按照下图操作指引，进入**群机器人**对话框。



4. 按照下图操作指引，进入**添加机器人**对话框。

5. 在**添加机器人**对话框，配置以下参数，并单击完成。

重要

其中**安全设置**需要选中**自定义关键词**，并配置为 PAI，否则将无法收到消息。

6. 在**添加机器人**对话框，单击**复制**。

其中Webhook链接中**access_token=**后的内容，即为您需要获取的access_token。

企业微信 Webhook 的 access_token 获取方式同钉钉 Webhook，最终会生成 Webhook 链接，比如 **https://qyapi.weixin.qq.com/cgi-bin/webhook/send?key=a412d913-335a-4973-a36d-ffc78282xxxx**，其中 **key=**后的内容即为 access_token。

1.6 高级功能

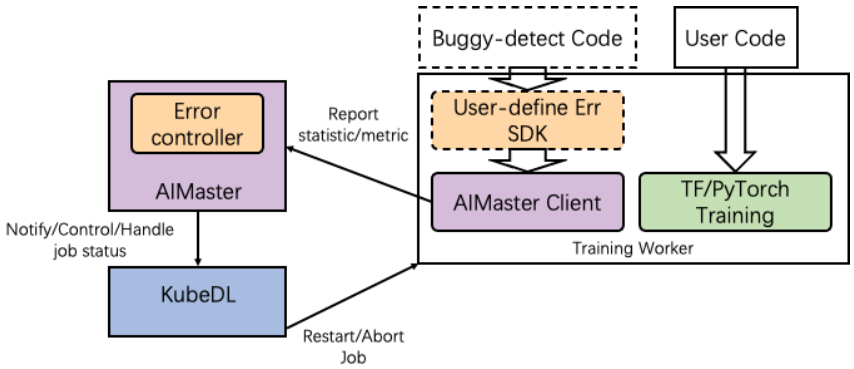
1.6.1 容错训练

本文介绍如何使用 PAI 灵骏智算平台 **AI 任务管理**提供的基于 AIMaster 的容错训练功能。

容错训练 AIMaster

当前深度学习已被广泛使用，随着模型和数据规模越来越大，常采用分布式方式运行深度学习任务。当同一个任务运行实例个数增大后，由于依赖的软件栈和硬件环境都有可能出现偶发异常，会导致任务停止运行。

为了保障大规模分布式深度学习任务稳定运行，PAI-DLC 提供了基于 AIMaster 的容错训练功能。整个基于 AIMaster 的容错训练系统架构如下图所示。



AIMaster 是一个任务级别的组件，当任务开启 AIMaster 的容错训练功能后，会拉起一个 AIMaster 实例和任务其他实例一起运行，起到任务监控、容错判断、资源控制的作用。

在作业容错训练方面，AIMaster 中集成了两个模块，一个是沉淀了我们的经验的通用错误监控和规避模块，二是 AIMaster 通过向用户提供 python 的 sdk，允许用户自定义容错信息，AIMaster 会自动扫描出错 pod 的日志并和自定义容错信息对比，若满足容错条件会触发容错。

使用流程

步骤一：配置容错训练参数

首先根据 AIMaster 支持的全量参数说明，并参考容错训练参数配置示例，来配置额外参数。

步骤二：开启容错训练功能

您可以在提交 PAI-DLC 训练任务时，通过控制台或 SDK 的方式开启容错训练控功能。任务运行异常时，会根据配置的额外参数，对任务进行相应的处理，从而保障任务能正常运行。

步骤三：配置容错训练增强功能

如果当前的容错训练配置不能满足您的要求，您可以使用容错训练增强功能，通过 AIMaster SDK 自定义容错关键字。任务运行过程中，AIMaster 会自动扫描出错节点的日志，并和自定义容错信息对比，如果满足容错条件，也会触发容错。

步骤一：配置容错训练参数

当前容错训练功能支持配置的参数如下，您可以提前规划好要为任务配置的容错训练内容。后续开启容错训练功能时，将已规划的容错训练内容配置到额外参数即可。

配置分类	功能介绍	配置参数	参数说明	默认值
通用配置	任务运行类型	--job-execution-mode	配置任务运行类型，取值如下： Sync：同步任务。 Async：异步任务。 不同任务类型容错行为不同。对于可重试错误： 同步任务需要重启任务。 异步任务一般只需重启失败的 Worker 实例。	Sync
	任务最大运行时长	--job-max-running-time	配置任务允许的最大运行时长，正整数，单位为分钟。 如果任务运行时长超过该值后，则	-1

			将任务标记为失败。默认值为-1，表示不开启该功能。 说明： 当该参数取值为-1 时，任务最大运行时长以控制台配置时间为准。 当控制台和容错训练参数均配置了任务最大运行时长时，以最短的时长为准。	
	任务重启设置	--enable-job-restart	在满足容错条件或检测到运行时异常时，是否允许任务重启。取值如下： False：不重启任务。 True：重启任务。	False
		--max-num-of-job-restart	配置任务最大重启次数。超过最大重启次数后，会将任务标记为失败。	3
运行时配置（针对没有 Worker 实例失败的场景）	任务 Hang（停止执行）异常检测	--enable-job-hang-detection	是否开启任务 Hang 异常检测，只支持同步任务。取值如下： False：表示不开启。 True：表示开启。如果所有 Worker 实例的 Stdout 和 Stderr 日志在指定时间内没有更新，将触发任务重启。	False
		--job-hang-interval	配置任务暂停执行的持续时长，正整数，单位为秒。 当任务暂停执行时长超过该值时，则将任务标记为异常，并触发任务	1800

			重启。	
	任务退出时Hang（停止执行）异常检测	--enable-job-exit-hang-detection	是否开启任务退出时 Hang 异常检测，只支持同步任务。取值如下： False：表示不开启。 True：表示开启。当任务某个 Worker 实例执行成功后，如果在指定时间内任务没有结束，将触发任务重启。	False
		--job-exit-hang-interval	配置任务退出时停止执行的持续时长，正整数，单位为秒。 当任务退出时长超过该值时，则将任务标记为异常，并触发任务重启。	600
容错配置（针对有 Worker 实例失败的场景）	容错策略	--fault-tolerant-policy	容错策略参数取值如下： OnFailure：任务实例出现异常时：异步任务会无条件重启失败的 Worker 实例。 同步任务会无条件重启任务。 ExitCodeAndErrorMsg：任务出现异常时，判断失败 Worker 实例的退出码及错误日志信息，如果满足重试条件： 异步任务会重启失败的 Worker 实例。 同步任务会重启任务。 Never：对失败任务不做任何处理，直接将任务标记为失败。	Never

	相同错误最大允许出现次数	--max-num-of-same-error	配置单个 Worker 实例上同一错误允许出现的最大次数。 当错误出现次数超过该值时，直接将任务标记为失败。	10
	最大容错率	--max-tolerated-failure-rate	配置最大容错率。 当失败 Worker 实例比例超过该值时，直接将任务标记为失败。 默认值值为-1，表示不开启该功能。	-1

步骤二：开启容错训练功能

您可以在提交 PAI-DLC 训练任务时，开启**容错训练**功能。

通过控制台开启容错训练功能在控制台提交 PAI-DLC 训练任务时，您可以在**基本信息**区域，打开**容错训练**开关，并配置**额外参数**，详情请参见新建任务（通过控制台）。

其中：**额外参数**的配置详情，请参见以上步骤一的说明。

步骤三：配置容错训练增强功能

您可以通过 AIMaster 的 SDK，在您自己的代码里根据不同的场景配置容错训练增强功能。

1、使用以下命令安装 AIMaster SDK

```
pip install -U https://pai-dlc.oss-cn-zhangjiakou.aliyuncs.com/aimaster/aimaster-1.1.6-cp36-cp36m-linux_x86_64.whl
```

2、配置容错训练增强功能

（1）配置自定义容错关键字

容错训练功能已内置了常见的可重试错误的监控模块，如果您希望异常 Worker 实例日志中出现某些关键字时也进行容错，则可以在您的代码中使用以下方法进行配置。配置完成后，容错训练模块会扫描失败的 Worker 实例尾部日志进行关键信息匹配。

说明

配置自定义容错关键字时，容错策略需要配置为 ExitCodeAndErrorMsg。

PyTorch 任务自定义容错关键字配置示例：

```
from aimaster import job_monitor as jm
jm_config_params = {}
jm_config = jm.PyTorchConfig(**jm_config_params)
monitor = jm.Monitor(config=jm_config)
monitor.set_retryable_errors(["connect timeout", "error_yyy", "error_zzz"])
```

说明

其中：monitor.set_retryable_errors 中配置的参数即为自定义容错关键字。

（2）分阶段自定义任务 Hang 异常检测

目前任务 Hang 异常检测的配置是针对整个任务的，但是任务状态是分阶段的。例如：构建通信训练任务时，规模大且耗时长，但训练阶段日志更新比较快。为了在训练过程中能快速发现任务 Hang 异常的节点,PAI-DLC 提供了分阶段自定义任务 Hang 异常检测功能，支持您在不同训练阶段配置不同的任务 Hang 异常检测时长，具体配置方法如下。

```
    monitor.reset_config(jm_config_params)

# Example:
# monitor.reset_config(job_hang_interval=10)
# or
# config_params = {"job_hang_interval": 10, }
# monitor.reset_config(**config_params)
```

PyTorch 任务分阶段自定义任务 Hang 异常检测示例如下。

```
    import torch
import torch.distributed as dist
from aimaster import job_monitor as jm

jm_config_params = {
 "job_hang_interval": 1800 # 全局 30min 检测
}
jm_config = jm.PyTorchConfig(**jm_config_params)
monitor = jm.Monitor(config=jm_config)

dist.init_process_group('nccl')

...
```

```
# impl these two funcs in aimaster sdk
# user just need to add annotations to their func
def reset_hang_detect(hang_seconds):
 jm_config_params = {
 "job_hang_interval": hang_seconds
 }
 monitor.reset_config(**jm_config_params)

def hang_detect(interval):
 reset_hang_detect(interval)
 ...

@hang_detect(180) # reset hang detect to 3 min, only for func scope
def train():
 ...

@hang_detect(-1) # disable hang detect temperally, only for func scope
def test():
 ...

for epoch in range(0, 100):
 train(epoch)
 test(epoch)
 self.scheduler.step()
```

1.6.2 AI 工作流

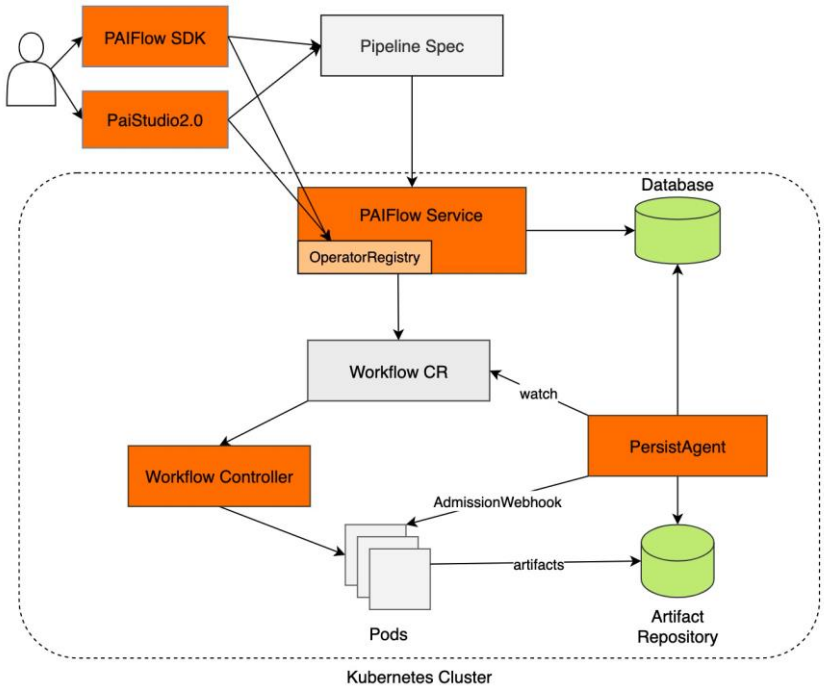
本节介绍 AI 工作流的使用。典型的机器学习任务应用，不单单是模型的训练，还可能包括数据清洗、数据处理、特征工程、模型训练、模型验证和离线推理等任务，开发者可以将这些流程使用离线脚本的形式进行拼接串联，或是将这些任务组合为工作流，提交到

工作流服务中执行。

一. 背景信息

PAI 灵骏智算平台提供了基于 Kubernetes 的工作流服务 PAIFlow，支持定义参数化的工作流，支持工作流和组件的注册机制。PAIFlow 使用 YAML 格式的 Manifest 描述工作流的定义（Pipeline Spec），可以通过 HTTP API 提交到后端执行，由 PAIFlow 后端负责解析对应的 PipelineSpec，然后在 Kubernetes 集群上执行和任务编排调度。

PAIFlow 的整体架构如下图所示。



PAI 提供了一个 Python SDK，用于工作流的构建，运行以及管理。以下的文档介绍了，用户如何通过 PAI 提供的 SDK 创建组件，使用组件构建工作流，然后提交到 PAIFlow 执行。

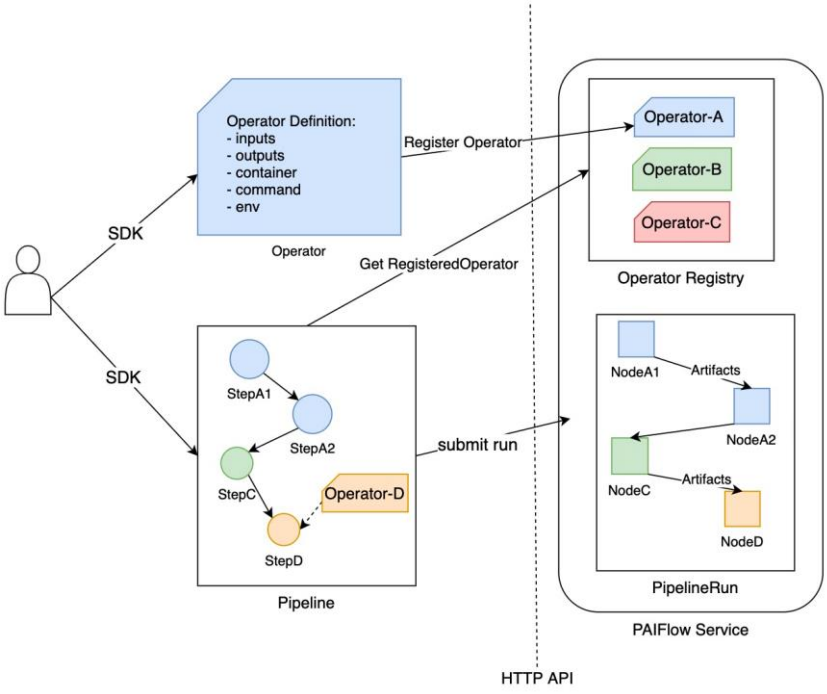
二. 基本概念

在通过 SDK 使用工作流前，需要首先了解 PAIFlow 的一些基础概念。

- Operator：组件，对应 PAIFlow 的一个组件，包含了组件的输入输出定义信息，具体的实现支持镜像执行，或是工作流执行。如果组件的实现是镜像，则还应当包含使用的镜像，执行命令等信息。如果组件的实现是多个节点组合而成的 DAG，则对应的组件是一个工作流（Pipeline）。用户构建的组件，可以注册到

PAIFlow 后端，或是用于工作流构建，也可以在提供组件所需参数后，直接提交到 PAIFlow 后端执行。

- Parameter：参数，组件的定义参数化，用户可以通过 PipelineParameter 定义一个输入参数信息，参数类型支持 int，float，bool，map，string 等，如果没有设置默认值，则默认为必须的参数，用户在提交组件执行，或是将组件使用到工作流作为一个工作节点时，必须填写所需的参数。
- Pipeline：工作流，用户可以使用注册在 PAIFlow 的组件或是本地定义的组件串联构建工作流。工作流支持参数化，用户填写了必须的参数后，可以将工作流提交到 PAIFlow 后端运行。同时，工作流也支持注册到 PAIFlow 后端，作为一个组件复用。
- Step：工作流的节点，由一个组件实例而来。用户在工作流中实例组件时，需要将组件所需的输入参数以及数据赋值给到节点。
- PipelineRun：工作流任务，由工作流或是组件提交到 PAIFlow 后端执行的任务实例，支持使用注册的组件提交，也支持用户将本地临时构建的组件提交。
- Artifact：工作流节点之间传递的数据信息，当工作流中的一个节点执行结束后，由 PAIFlow 将组件定义的输出数据搬运保存，然后加载到一个下游的依赖于这个数据的组件中。



三. 准备工作

1.安装 SDK。

请通过 pip 安装所需的 SDK，执行命令如下。

说明

虽然 SDK 支持 Python 2 和 Python 3 版本，但是 Python 社区已经停止维护 Python 2 版本，因此推荐您使用 Python 3 环境运行 SDK。

```
python -m pip install --upgrade pip --index-url=https://mirrors.aliyun.com/pypi/simple/
```

```
python -m pip install https://pai-sdk.oss-cn-shanghai.aliyuncs.com/alipai/dist/alipai-0.3.2b2-py2.py3-none-any.whl --index-url=https://mirrors.aliyun.com/pypi/simple/
```

2.初始化默认的 SDK Session。

提交工作流任务，或是保存工作流组件需要与 PAIFlow 后端交互，需要用户提供相应的认证信息。用户可以通过登录 PAI 的控制台点击右上角游用户信息，查看相应的认证凭证，以及 endpoint 信息，如下图所示。

轻量化场景下，请使用 setup_light_default_session 初始化默认的 session，如以下的代码示例。

```
import os
```

```
import pai
print(pai.__version__)
print(pai.__file__)
from pai.core.session import setup_light_default_session
# PAI-EFLOPS(PAI 轻量化输出）使用以下方法，初始化执行环境
# 其中 username/token 信息请从 PAI-EFLOPS 产品控制台.
# ak_id 即当前的用户登录名.
ak_id = os.environ.get("PAI_EFLOPS_DEMO_USERNAME") or "<UserName>"
# ak_secret 即当前的用户使用的鉴权 token，见控制台的用户信息
ak_secret = os.environ.get("PAI_EFLOPS_DEMO_PASSOWRD") or
"<Token/AccessKseySecret"
# PAIFlow 使用的 endpoint，请从控制台查看
endpoint = os.environ.get("PAI_EFLOPS_DEMO_ENDPOINT") or "<PAIFlow
Endpoint>"
sess = setup_light_default_session(
    # 轻量化场景下的用户登录名
    access_key_id=ak_id,
    # 用户 Token，请在控制台查看用户信息获取
    access_key_secret=ak_secret,
    # PAIFlow 服务的 Endpoint
    endpoint=endpoint,
    # 轻量化场景下，默认使用 HTTP
    # protocol="http",
)
sess.provider
```

四. 自定义组件

(1) 创建组件

在 PAIFlow 中，用户通过串联多个组件(Operator)，构建一个工作流(Pipeline)。 工

作流提交到 PAIFlow 的一次执行，对应一个工作流任务(PipelineRun)，由 PAIFlow 服务完成参数的解析渲染，调度到 Kubernetes 上执行，工作流任务中的一个节点，对应到 Kubernetes 上的一个 Pod。

用户可以使用本地自定义的组件，或是已经注册的组件，构建一个工作流。用户可以通过 SDK 提供的 ContainerOperator 构建一个 PAIFlow 组件。

以下 ContainerOperator 的一个使用样例，它通过一段 Python 代码，打印出运行容器的环境变量。

- image_uri，command，env 则分别表示节点执行使用的容器镜像，运行命令，和注入容器的环境变量信息。
- inputs 和 outputs 参数是用于表示组件的输入和输出信息定义。用户在 inputs 中定义的参数（PipelineParameter）在可以在 command，以及 env 中，以{{inputs.parameters.parameter_name}}的形式引用，其中，parameter_name 是组件 inputs 中参数的名称，PAIFlow 根据提交任务时的实际参数信息渲染出完整的执行命令，环境变量等信息。

```
    from pai.operator import ContainerOperator
from pai.operator.types import PipelineParameter
image_uri = "python:3"
# 构建 PAIFlow 组件
container_op = ContainerOperator(
    inputs=[
        PipelineParameter(name="foo"),
        PipelineParameter(name="bar", default="Bob")
    ],
    image_uri=image_uri,
    command=[
        "python",
```

```
        ],
        args=[
            "-c",
             "import os; print(os.environ); print('Parameter Bar is
    {{inputs.parameters.bar}}');",
        ],
        env={
            "Hello": "World",
            "ParameterFoo": "{{inputs.parameters.foo}}"
        }
    )
    # 打印组件定义的 YAML
    print(container_op.to_manifest(identifier="Example", version="v1.0"))
    # 导出组件定义的 YAML 文件
    container_op.export_manifest(file_path="/tmp/component_def.yaml",
    identifier="Example", version="v1.0")
```

PAIFlow 使用 YAML 格式文件的表示 Pipeline 的定义，可以通过 op.to_manifest 获取对应的 YAML 定义或是 op.export_manifest 导出相关的定义到本地文件，对应的定义文件可以用于在 PAI 的工作流控制台上传注册新的组件。

下面的 YAML 文件来自于以上的 ContainerOperator 的例子，其中：

- .spec 是组件的运行定义，包含了使用的容器，执行命令，环境变量（spec.container），以及输入输出信息定义(spec.inputs 和 spec.outputs)。
- .metadata 是组件的元信息，当上传注册后，用户可以通过这些原先从后端获取对应的组件。

```
    apiVersion: core/v1
metadata:
```

```
    identifier: Example
    version: v1.0
  spec:
    container:
      args:
      - -c
      - import os; print(os.environ); print('Parameter Bar is
  {{inputs.parameters.bar}}');
      command:
      - python
      envs:
        Hello: World
        ParameterFoo: '{{inputs.parameters.foo}}'
      image: python:3
    inputs:
      parameters:
      - name: foo
        type: String
      - name: bar
        type: String
        value: Bob
    outputs:
      artifacts: []
```

（2）组件注册和运行

通过 SDK 定义的组件，可以直接提交到 PAIFlow 执行，也可以注册到 PAIFlow，以便后续复用，具体示例可以参考以下代码。

通过提供组件定义时所需的参数(inputs)，对应的组件可以直接提交在 PAIFlow 中运行。以上构建的组件，定义了两个参数 foo, bar，其中 bar 参数提供了默认值，用户提交

执行时，可以无需提供。通过 SDK 提交执行后，会打印作业的 URL 信息，用户可以通过作业 URL 查看任务执行进度。

PAIFlow 也支持组件的注册机制，用户通过调用.save 接口，可以将本地构建的组件（或是工作流）注册到 PAIFlow。通过注册时填写的 identifier, version 信息，用户可以调用 SavedOperator.get_by_identifier 接口获取已注册的组件。

```
    import time
from pai.operator import SavedOperator
# 直接运行构造的组件
run = container_op.run(job_name="jobExample", arguments={
    "foo": "Foo",
}, wait=False)
print(run)
# 组成组件到后端
op_version = "v-%s" % int(time.time())
registered_op      =      container_op.save(identifier="containerOpExample",
version=op_version)
print(registered_op)
# 获取以上步骤注册的组件
remote_op=  SavedOperator.get_by_identifier(identifier="containerOpExample",
version=op_version)
print(remote_op)
assert registered_op == remote_op
# 获取已经注册的所有组件列表
for op in SavedOperator.list():
    print(op)
# 运行注册的组件
remote_op.run(job_name="RegisterOpRunExample",
```

```
    arguments={
        "foo": "ParamFoo",
        "bar":"ParamBar",
    },
    wait=False,
)
```

五. 构建工作流

通过将组件进行串联构建 DAG，构建一个工作流。这里使用的组件，支持 PAIFlow 中注册的组件，也支持使用 SDK 构建的未注册组件。以下为代码示例。

```
    from pai.pipeline import Pipeline
# 本地构建组件
local_op = ContainerOperator(
    inputs=[
        PipelineParameter(name="x", typ=int),
        PipelineParameter(name="y", typ=int, default=10)
    ],
    image_uri=image_uri,
    command=[
        "python",
    ],
    args=[
        "-c",
         "import os; print('op result is', {{inputs.parameters.x}} + {{inputs.parameters.y}}, os.environ)"
    ],
)
def create_pipeline():
```

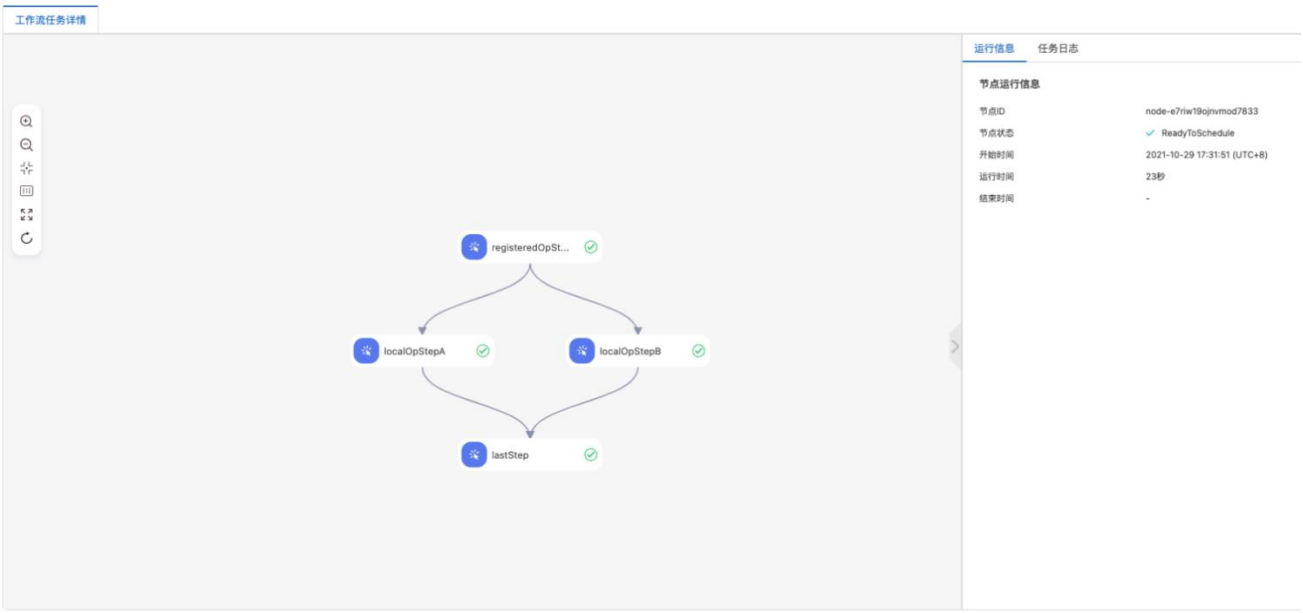
```
    x = PipelineParameter(name="x",typ=int)
    # step1 是使用以上已注册的组件
    step1 = registered_op.as_step(name="registeredStep1", inputs={
        "foo": "ParameterFoo"
    })
    # step2 和 step3 是使用未注册的组件构建
    step2 = local_op.as_step(name="localOpStepA", inputs={
        "x": x,
    })
    step3 = local_op.as_step(name="localOpStepB", inputs={
        "x": x,
        "y": 1000,
    })
    step4 = registered_op.as_step(name="lastStep", inputs={
        "foo": "ParameterFoo",
    })
    step2.after(step1)
    step3.after(step1)
    step4.after(step2, step3)
    # Pipeline 根据依赖关系推导出所需的 steps,以及 inputs 参数
    return Pipeline(steps=[step4])


# 构建 Pipeline
p = create_pipeline()
# 通过 graphviz 可视化工作流的 DAG，需要本地安装 Graphviz: https://graphviz.org/download/#executable-packages
p.dot()
# 打印 Pipeline 的 Manifest 定义.
```

```
pipeline_version = "v-%s" % int(time.time())
print(p.to_manifest(identifier="example-pipeline",
version=pipeline_version))
p.export_manifest("/tmp/example_pipeline.yaml",          identifier="example-
pipeline", version=pipeline_version)
# 打印 Pipeline 所需的输入
print(p.inputs)
# 提交 Pipeline 运行
p.run(job_name="pipeline_run_example", arguments={
    "x": 22,
})


# 注册 Pipeline
reg_pipeline              =              p.save(identifier="example-pipeline",
version=pipeline_version)
remote_pipeline     =     SavedOperator.get_by_identifier("example-pipeline",
version=pipeline_version)
# 运行保存的 Pipeline1
remote_pipeline.run(job_name="remote_pipeline_run", arguments={
    "x": 200,
})
```

通过 Pipeline 提交后打印的任务 URL，用户可以去控制台查看工作流的执行进度，节点日志等信息。



如同使用 ContainerOperator 构建的组件，以上构建的 Pipeline 可以直接提交到 PAIFlow 运行，也可以通过注册到 PAIFlow 后端。

PAIFlow 中的工作流，同样也支持使用一个 YAML 表示，以上的构建的工作流，对应的 YAML 格式的工作流定义如下。

```
    apiVersion: core/v1
metadata:
  guid: 75ebcdbd631c4a2497c46cb125304fdf
  name: tmp-n4r8xtqlu13dialr
spec:
  container:
    args:
    - -c
    -     print('op     result     is',     {{inputs.parameters.x}}     +
{{inputs.parameters.y}})
    command:
    - python
    image: python:3
```

```
    imageRegistryConfig: {}
  inputs:
    artifacts: []
    parameters:
    - name: x
      type: Int
    - name: y
      type: Int
      value: 10
  outputs:
    artifacts: []
    parameters: []
---
apiVersion: core/v1
metadata:
  guid: 1a20943d0ff744488ce55e7c9510bb54
  identifier: example_pipeline
  name: tmp-i2vup9fcipzlad24
  version: v-1635434866
spec:
  inputs:
    artifacts: []
    parameters:
    - name: x
      type: Int
  outputs:
    artifacts: []
    parameters: []
  pipelines:
```

```
- metadata:
    guid: 75ebcdbd631c4a2497c46cb125304fdf
    name: local_op_step_2
  spec:
    arguments:
      artifacts: []
      parameters:
      - from: '{{inputs.parameters.x}}'
        name: x
      - name: y
        value: 1000
- metadata:
    guid: 75ebcdbd631c4a2497c46cb125304fdf
    name: local_op_step_1
  spec:
    arguments:
      artifacts: []
      parameters:
      - from: '{{inputs.parameters.x}}'
        name: x
- metadata:
    identifier: containerOpExample
    name: registered_op_step
    provider: '6'
    version: v-1635434597
  spec:
    arguments:
      artifacts: []
      parameters:
```

```
    - name: foo
      value: ParameterFoo
  dependencies:
  - local_op_step_1
  - local_op_step_2
```

六. 使用脚本构建组件

通过 ContainerOperator 直接构建的组件，需要依赖用户将代码，或是执行逻辑持久化到镜像中，或是通过在 command 或是 args 等方式传递，构建的成本较高。ContainerOperator 提供了一个便利的方法 ContainerOperator.create_with_soure_snapshot，用户可以通过一段脚本，简易得构建一个组件。

说明

目前支持用户使用 Shell 或是 Python 脚本。

以下的样例中，我们使用一个 shell script 构建一个提交 Dlc 作业的组件节点。 组件执行时接受，两个参数 batch_size,epochs，然后使用 dlc-cli 提交作业。

```
%%writefile "./tmp_torch_job.sh"
set -e
# torchJob.sh
# 以下 operator 提交时，会以以下方式执行
#       torchJob.sh       --epochs       {input_param_epochs}       --batch_size
{input_param_batch_size}
epochs=${2:-1}
batch_size=${4:-32}
cat << EOT > torchJobFile
```

```
name=cli_pytorch_job
kind=PyTorchJob
worker_count=3
worker_cpu=10
worker_gpu=1
worker_memory=10
worker_image=master0:5000/eflops/deepops/bert:rdma-pytorch-mnist
workspace_id=ws1oydurlpapu726
priority=0
command=export NCCL_IB_DISABLE=1 && /opt/conda/bin/python3
/data/user/ken/demo/mnist-pytorch/mnist.py --backend nccl --epochs=$epochs --
-batch_size=$batch_size
EOT


# dlc-cli 通过 PAIFlow 注入到 POD 的环境变量，获得 username/token/Endpoint 信息
# 增加 -w
echo "Use Golang Dlc Cli"
dlc create job --params_file=torchJobFile --interactive

# sleep 600
echo "job run complete"
```

使用 ContainerOperator.create_with_source_snapshot 构建组件，提交到 PAIFlow 执行。用户可以通过执行任务输出的 Job URL 跳转到 DLC 作业的控制台页面。

说明

create_with_source_snapshot 使用的默认镜像已安装了常见的一些 Python

Package，以及 PAI-DLC CLI 工具。

```
    from pai.operator import ContainerOperator
from pai.operator.types import PipelineParameter
import time
script_file = "./tmp_torch_job.sh"
op = ContainerOperator.create_with_source_snapshot(
    script_file=script_file,
    # 组件的 inputs 是组件的
    inputs=[
        PipelineParameter("epochs", default=2),
        PipelineParameter("batch_size", default=32),
    ],
)
print(op.to_manifest(identifier="hello", version="world"))
version = "v-%s" % (int(time.time()))
op.run("script_op_run", arguments={
    "batch_size": 1,
    "epochs": 30
})
```

七. 工作流的节点间传递数据

PAIFlow 支持上下游的节点间传递数据，对应的数据，称之为 artifact。 当用户构建的 Pipeline 声明下游节点依赖上游节点的输出 artifact 时，会有 PAIFlow 的调度器，完成数据的搬运。

组件内的程序运行前，PAIFlow 完成 artifact 的搬运准备，默认输入的 artifact 会在/work/inputs/artifacts/{{artifact_name}}/value 就绪。而在组件运行结束后，PAIFlow 会尝试搬运组件定义时的输出 artifact，然后搬运到 PAIFlow 的 artifact 仓库，默认输

出 artifact 的地址是 work/outputs/artifacts/{{artifact_name}}/value。

```
/work
├── inputs
│   └── artifacts
│       └── foo
│           ├── metadata
│           └── value
└── outputs
    └── artifacts
        └── bar
            └── value
```

以下的脚本是执行在节点上的逻辑，他会尝试去读取输入 artifact(foo)，修改后写出一个新的 artifact(bar)。

```
%%writefile "./tmp_artifact_read_write.py"
import os
import time
def main():
    base_dir = "/work/"
    input_path = os.path.join(base_dir, "inputs", "artifacts", "foo", "value")
    input_value = open(input_path, "r").read()
    print("input value is", input_value)
    if input_value == "Hello":
        output_value = input_value + "Meta"
```

```
    else:
        output_value = input_value + "World"
    output_path = os.path.join(base_dir, "outputs", "artifacts", "bar",
"value")
    # exist_ok only work in Python3
    os.makedirs(os.path.dirname(output_path), exist_ok=True)
    with open(output_path, "w+") as f:
        f.write(output_value)
if __name__ == "__main__":
    main()
```

我们使用以上的脚本构建一个读写 artifact 的组件，并将组件串联为一个 Pipeline 执行。可以通过执行的 run.get_outputs 获取最终组件的输出 artifact 信息。

```
    from pai.operator.types import PipelineArtifact, MetadataBuilder
script_file = "./tmp_artifact_read_write.py"
art_op = ContainerOperator.create_with_source_snapshot(
    script_file=script_file,
    inputs=[
        PipelineArtifact(
            name="foo",
            metadata=MetadataBuilder.raw(),
        ),
    ],
    outputs=[
        PipelineArtifact(
            name="bar",
            metadata=MetadataBuilder.raw(),
        )
```

```
    ],
)
# run = op.run("rawArtifactAppend", arguments={"input1": "helloWorld"})
def create_pipeline():
    # step1 的初始化输入的 artifact 是 "Hello"
    step1 = art_op.as_step(name="step1",
                inputs={
                    "foo": "Hello"
                })
    # step2 在运行时，会使用 step1 的输出 artifact(bar)，因而 step2 的运行也是依赖于 step1 的，等同于`run_after`语义.
    step2 = art_op.as_step(name="step2",
                           inputs= {
                           "foo": step1.outputs["bar"]
                       })
    return Pipeline(steps=[step1, step2], outputs=step2.outputs[:])
art_pipeline = create_pipeline()
run = art_pipeline.run(
    "artifactExample",
    wait=True,
)
# 获取 Pipeline 的最终输出信息
output = run.get_outputs()[0].value
print(output)
assert output == "HelloMetaWorld"
```

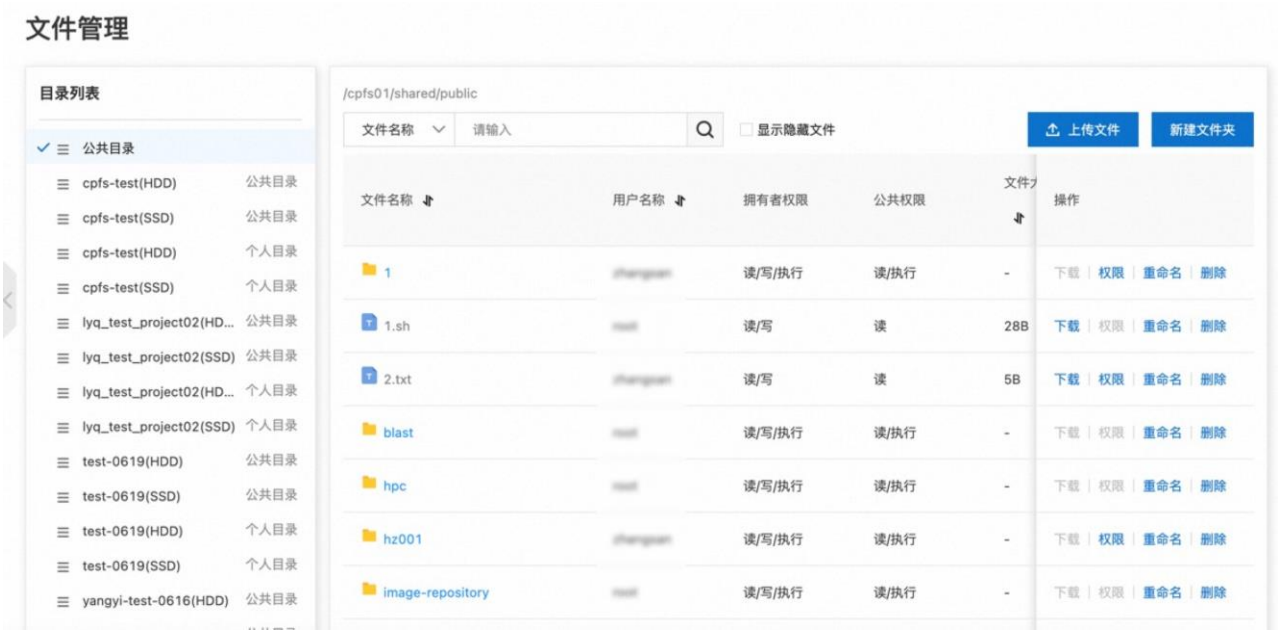
1.7 2.2.6. 管理资产

1.7.1 文件管理

本节介绍如何使用**文件管理**功能。

一. 进入文件管理

在控制台点击左侧菜单**资产管理**->**文件管理**，进入文件管理页面，如下所示：



二. 使用文件管理

在左侧的目录列表，可以看到您拥有权限的所有目录，点击任何一个目录，右侧显示该目录下的文件。您可以做如下操作：

- **上传文件**
- **新建文件夹**
- **下载**文件
- 修改**权限**
- **重命名**文件或文件夹
- **删除**文件或文件夹

1.7.2 数据集配置

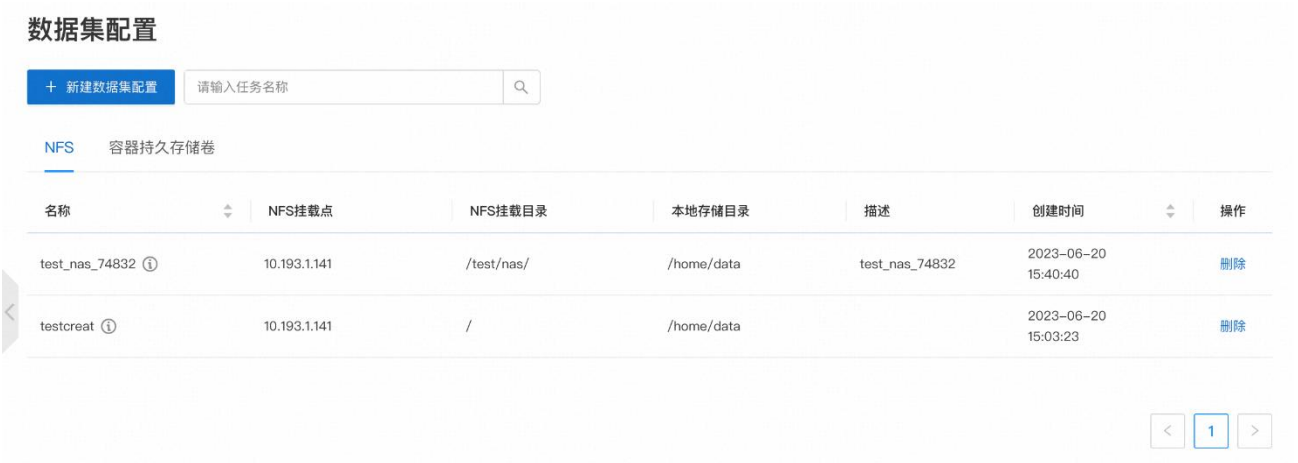
本节介绍如何创建和使用数据集。

说明

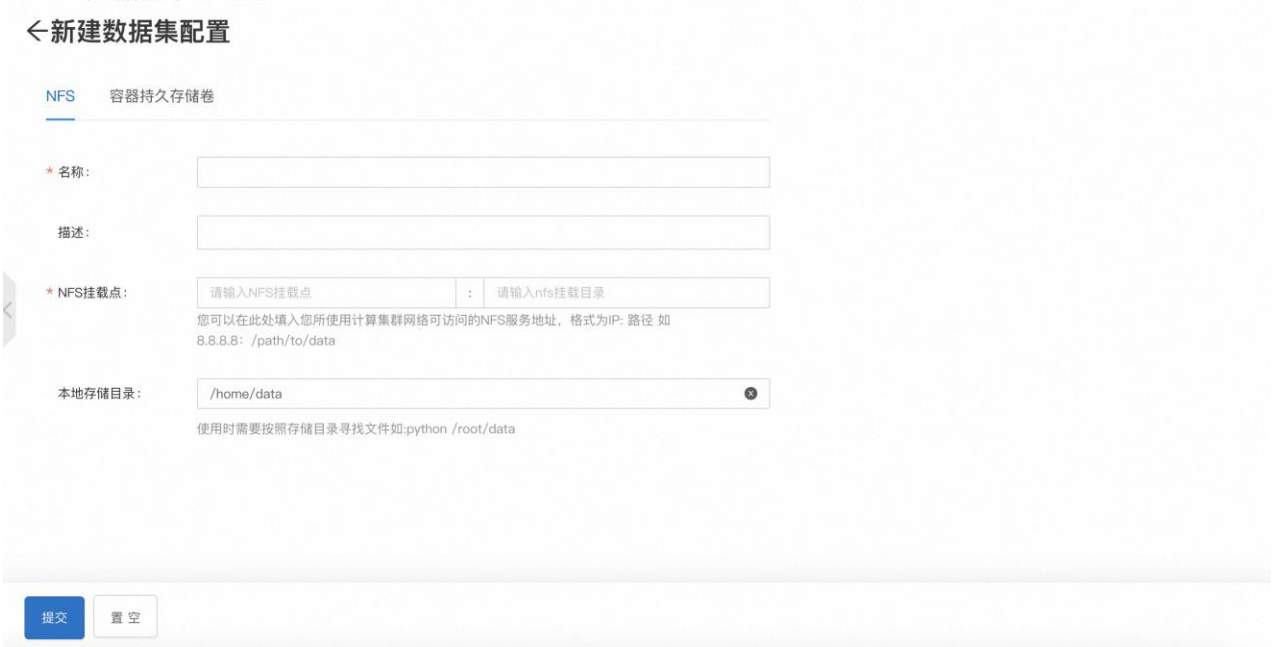
一般情况下您只需要通过文件管理上传和管理数据即可。如果您需要通过物理机挂载点访问数据集，则可以通过数据集配置创建数据集，您可以在控制台点击左侧菜单**资产管理->数据集配置**，进入数据集配置列表。PAI 灵骏智算平台支持 NFS 和容器持久存储卷两类数据集配置。

一. NFS 数据集配置

（1）进入**数据集配置**，选择 **NFS** 标签页，打开 NFS 数据集列表。



（2）点击**新建数据集配置**，进入创建 NFS 数据集页面：

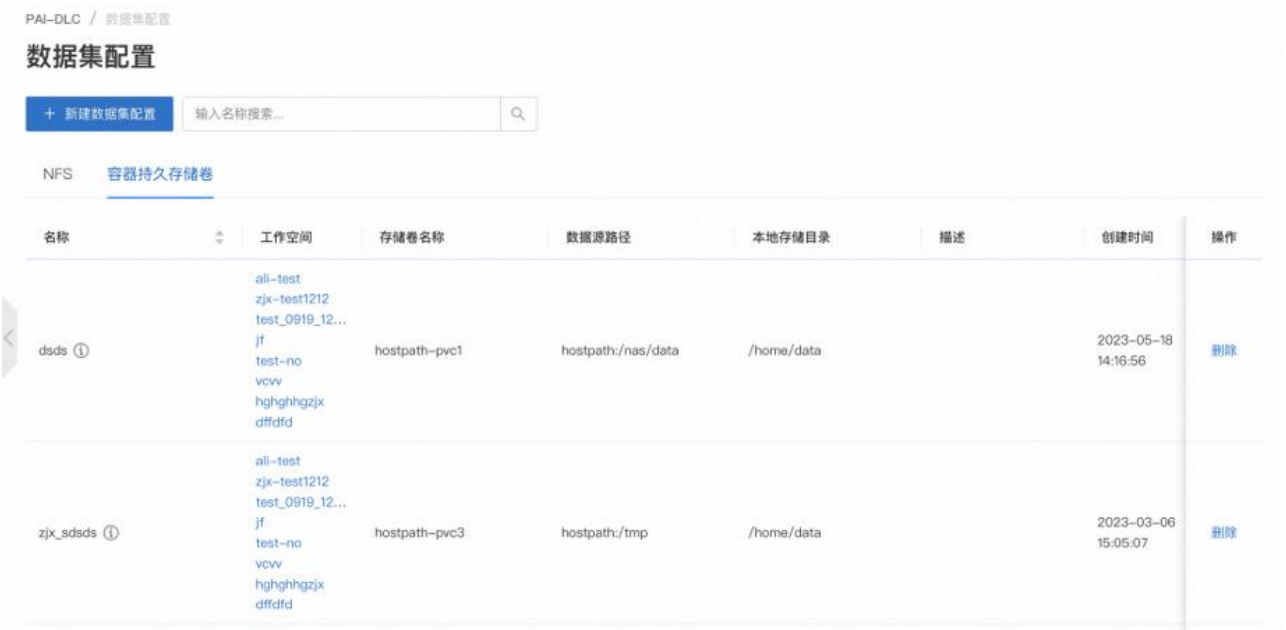


（3）输入参数说明：

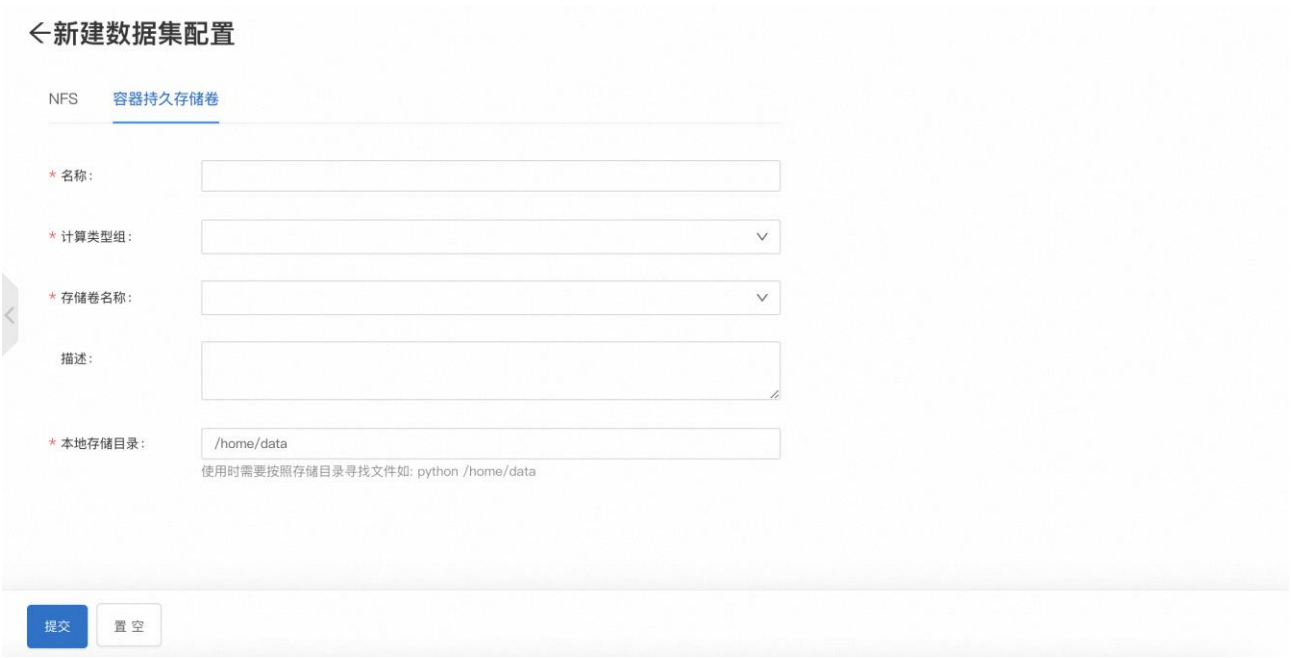
参数	描述
名称	名称长度 2～64 字符之间，允许数字、字母、下划线、小数点、短横线。
描述	描述长度 2～1024 字符之间。
NFS 挂载点	您可以在此处填入您所使用计算集群网络可访问的 NFS 服务地址，格式为 IP：路径，如 8.8.8.8：/path/to/data
本地存储目录	使用时需要按照存储目录寻找文件如: python /home/data

二. 容器持久存储卷

（1）进入**数据集配置**，点击**容器持久存储卷**，打开容器持久存储卷列表：



（2）点击**新建数据集配置**，并选择**容器持久存储卷**，进入创建页面：



（3）输入参数说明：

参数	描述
名称	名称长度 2～64 字符之间，允许数字、字母、下划线、小数点、短横线。
计算类型组	从列表中选择您可以使用的计算类型组。
存储卷名称	从列表中选择您可以使用的存储卷。
描述	描述长度 2～1024 字符之间。
本地存储目录	使用时需要按照存储目录寻找文件如:python /home/data

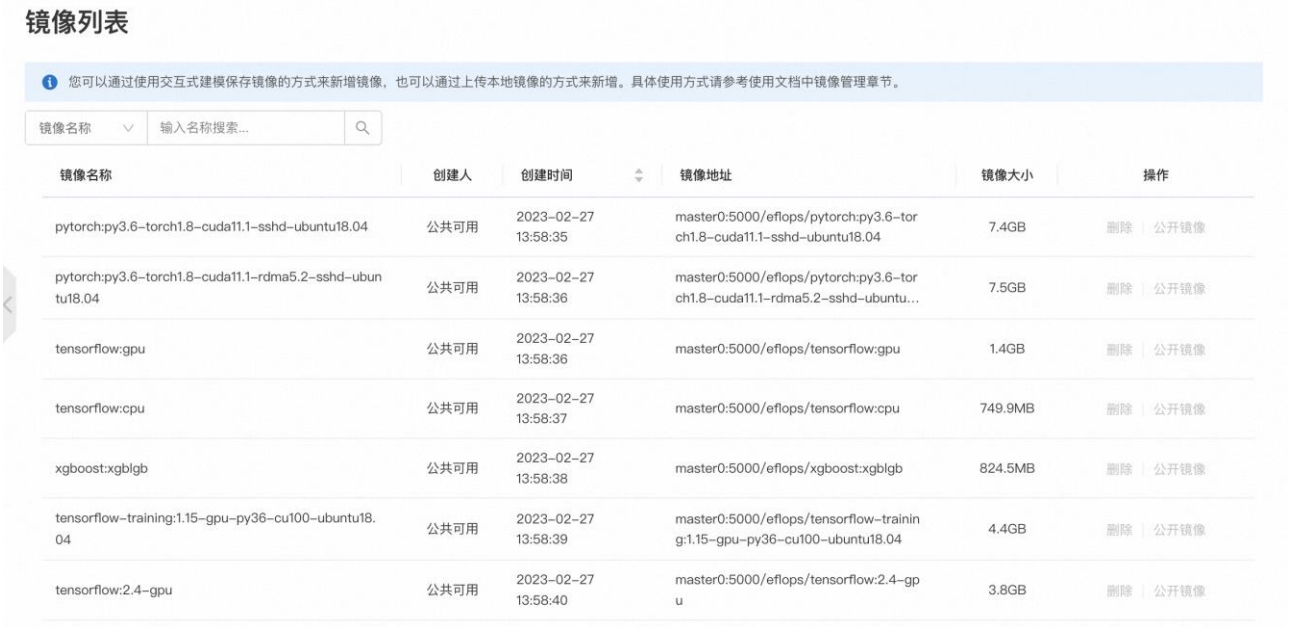
1.7.3 AI 镜像管理

一. 查看镜像

AI 镜像目前只包含 Docker 镜像，您可以在**镜像列表**查看当前可使用的 Docker 镜像：

1. 在控制台左侧导航栏，选择**资产管理->AI 镜像管理**。

2. 在**镜像列表**页面，即可查看当前可使用的镜像，包括**镜像名称**、**创建人**、**镜像地址**、**镜像大小**等信息。



镜像的创始人会指明该镜像的所有者，其中**公共可用**表示该镜像可被所有用户共享使用。

二. 制作镜像

如果您需要从集群中一个现成的容器来制作镜像快照，并推送到您内部的镜像仓库，可以通过 DLC 命令行工具执行如下命令。关于命令行工具的下载安装，请参考准备工具。

```
dlc-cli create-image \
--identity_file=/cpfs/user/ken/auth \
--pod_id=dlc-20210308111132-pid8uevpvrk22-master-0 \
--image_tag=chenkun/test:v1 \
--is_shared=True
```

说明

- 调用该命令前请先确保已经有一个处于 Running 状态的 Worker（即 container）。

●image_tag 只用填写镜像信息即可，前面不需要指定镜像仓库的域名和 namespace 前缀，系统后端会自动为您加上。

●is_shared 用来表示是否为公共镜像。如果是 False，则表示仅提交人可见。

三. 同步镜像

制作完基础镜像并执行完 Docker Push 后，使用该命令将该基础镜像的信息同步至数据库，进而所有用户都可以在提交作业时的下拉列表查看并使用该镜像。

```
dlc-cli sync-image \
--identity_file=/cpfs/user/ken/auth \
--image_tag=chenkun/test:v1
```

四. 删除镜像

对于确认不再使用的镜像，您可以删除，具体做法：

1. 通过控制台删除：单击镜像**操作**列下的**删除**，即可删除该镜像。或者，
2. 通过 dlc-cli 命令行工具：

```
dlc-cli delete-image \
--identity_file=/cpfs/user/ken/auth \
--image_tag=chenkun/test:v1
```

说明

管理员可以删除任何镜像，普通用户只能删除**创建人**为自己的镜像。