

ModelArts

使用自定义镜像

文档版本 01
发布日期 2024-04-30



版权所有 © 华为云计算技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目录

| | |
|---------------------------------------|-----------|
| 1 镜像管理 | 1 |
| 2 使用预置镜像 | 4 |
| 2.1 统一镜像介绍 | 4 |
| 2.2 Notebook 基础镜像介绍 | 11 |
| 2.2.1 Notebook 基础镜像功能 | 11 |
| 2.2.2 Notebook 基础镜像列表 | 12 |
| 2.2.3 Notebook 基础镜像 x86 PyTorch | 13 |
| 2.2.4 Notebook 基础镜像 x86 Tensorflow | 20 |
| 2.2.5 Notebook 基础镜像 x86 MindSpore | 25 |
| 2.2.6 Notebook 基础镜像 x86 自定义专用镜像 | 33 |
| 2.2.7 Notebook 基础镜像 ARM MindSpore | 35 |
| 2.2.8 Notebook 基础镜像 ARM TenSorFlow | 38 |
| 2.3 训练基础镜像详情介绍 | 40 |
| 2.3.1 训练基础镜像列表 | 41 |
| 2.3.2 训练基础镜像详情 (PyTorch) | 41 |
| 2.3.3 训练基础镜像详情 (TensorFlow) | 42 |
| 2.3.4 训练基础镜像详情 (Horovod) | 43 |
| 2.3.5 训练基础镜像详情 (MPI) | 45 |
| 2.3.6 预置框架启动流程说明 | 45 |
| 2.3.6.1 PyTorch | 45 |
| 2.3.6.2 Tensorflow | 48 |
| 2.3.6.3 Ascend-Powered-Engine | 51 |
| 2.3.6.4 Horovod/MPI/MindSpore-GPU | 53 |
| 2.4 推理基础镜像介绍 | 56 |
| 2.4.1 推理基础镜像列表 | 56 |
| 2.4.2 推理基础镜像详情 TensorFlow (CPU/GPU) | 58 |
| 2.4.3 推理基础镜像详情 Pytorch (CPU/GPU) | 62 |
| 2.4.4 推理基础镜像详情 MindSpore (CPU/GPU) | 66 |
| 3 Notebook 中使用自定义镜像 | 72 |
| 3.1 Notebook 自定义镜像约束 | 72 |
| 3.2 在 ModelArts 中进行镜像注册 | 72 |
| 3.3 Notebook 制作自定义镜像方法 | 74 |

| | |
|---|------------|
| 3.4 将 Notebook 实例保存为自定义镜像..... | 74 |
| 3.4.1 保存 Notebook 镜像环境..... | 74 |
| 3.4.2 基于自定义镜像创建 Notebook 实例..... | 76 |
| 3.5 在 Notebook 中构建自定义镜像并使用..... | 76 |
| 3.5.1 使用场景和构建流程说明..... | 76 |
| 3.5.2 Step1 制作自定义镜像..... | 77 |
| 3.5.3 Step2 注册新镜像..... | 78 |
| 3.5.4 Step3 创建开发环境并使用..... | 79 |
| 3.6 在 ECS 上构建自定义镜像并在 Notebook 中使用..... | 81 |
| 3.6.1 使用场景和构建流程说明..... | 81 |
| 3.6.2 Step1 准备 Docker 机器并配置环境信息..... | 81 |
| 3.6.3 Step2 制作自定义镜像..... | 82 |
| 3.6.4 Step3 注册新镜像..... | 86 |
| 3.6.5 Step4 创建开发环境并使用..... | 86 |
| 3.7 Notebook 自定义镜像故障基础排查..... | 88 |
| 4 使用自定义镜像训练模型（模型训练）..... | 89 |
| 4.1 训练管理中使用自定义镜像介绍..... | 89 |
| 4.2 示例：从 0 到 1 制作自定义镜像并用于训练..... | 91 |
| 4.2.1 示例：从 0 到 1 制作自定义镜像并用于训练（PyTorch+CPU/GPU）..... | 91 |
| 4.2.2 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU）..... | 97 |
| 4.2.3 示例：从 0 到 1 制作自定义镜像并用于训练（Horovod-PyTorch+GPU）..... | 105 |
| 4.2.4 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+GPU）..... | 116 |
| 4.2.5 示例：从 0 到 1 制作自定义镜像并用于训练（Tensorflow+GPU）..... | 124 |
| 4.2.6 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+Ascend）..... | 130 |
| 4.2.6.1 场景描述..... | 130 |
| 4.2.6.2 Step1 创建 OBS 桶和文件夹..... | 131 |
| 4.2.6.3 Step2 准备脚本文件并上传至 OBS 中..... | 132 |
| 4.2.6.4 Step3 制作自定义镜像..... | 142 |
| 4.2.6.5 Step4 上传镜像至 SWR..... | 145 |
| 4.2.6.6 Step5 在 ModelArts 上创建 Notebook 并调试..... | 146 |
| 4.2.6.7 Step6 在 ModelArts 上创建训练作业..... | 146 |
| 4.3 准备训练镜像..... | 147 |
| 4.3.1 训练作业自定义镜像规范..... | 147 |
| 4.3.2 已有镜像如何适配迁移至 ModelArts 训练平台..... | 148 |
| 4.3.3 使用基础镜像构建新的训练镜像..... | 149 |
| 4.3.4 在容器镜像中安装 MLNX_OFED..... | 150 |
| 4.4 使用自定义镜像创建算法..... | 151 |
| 4.5 使用自定义镜像创建训练作业（CPU/GPU）..... | 155 |
| 4.6 使用自定义镜像创建训练作业（Ascend）..... | 159 |
| 4.7 自定义镜像训练作业失败定位思路..... | 160 |
| 5 使用自定义镜像创建 AI 应用（推理部署）..... | 162 |
| 5.1 创建 AI 应用的自定义镜像规范..... | 162 |

| | |
|---|------------|
| 5.2 从 0-1 制作自定义镜像并创建 AI 应用..... | 164 |
| 5.3 在开发环境中构建并调试推理镜像..... | 167 |
| 5.3.1 场景说明..... | 167 |
| 5.3.2 Step1 在 Notebook 中构建一个新镜像..... | 168 |
| 5.3.3 Step2 构建成功的镜像注册到镜像管理模块..... | 173 |
| 5.3.4 Step3 在 Notebook 中变更镜像并调试..... | 173 |
| 5.3.5 Step4 使用调试成功的镜像用于推理部署..... | 176 |
| 5.4 无需构建直接在开发环境中调试并保存镜像用于推理..... | 179 |
| 5.4.1 场景说明..... | 179 |
| 5.4.2 Step1 在 Notebook 中拷贝模型包..... | 179 |
| 5.4.3 Step2 在 Notebook 中调试模型..... | 183 |
| 5.4.4 Step3 Notebook 中保存镜像..... | 185 |
| 5.4.5 Step4 使用保存成功的镜像用于推理部署..... | 186 |
| 6 FAQ..... | 189 |
| 6.1 如何登录并上传镜像到 SWR..... | 189 |
| 6.2 如何给镜像设置环境变量..... | 190 |
| 6.3 如何通过 docker 启动 Notebook 保存后的镜像..... | 190 |
| 6.4 如何在 Notebook 开发环境中配置 Conda 源..... | 191 |
| 6.5 自定义镜像软件版本匹配注意事项..... | 192 |
| 6.6 镜像在 SWR 上显示只有 13G，安装少量的包，然后镜像保存过程会提示超过 35G 大小保存失败，为什么？..... | 192 |
| 6.7 镜像保存如何保证能正常保存，不会因为超过 35G 而保存失败？..... | 193 |
| 6.8 本地/ECS 构建镜像，如何减小目的镜像的大小？..... | 193 |
| 6.9 镜像过大，卸载原来的包重新打包镜像，或者把原有的数据集从镜像中删除，最终镜像会变小吗？..... | 194 |
| 6.10 在 ModelArts 镜像管理注册镜像报错 ModelArts.6787..... | 194 |
| 7 修订记录..... | 195 |

1 镜像管理

ModelArts 镜像管理简介

在AI业务开发以及运行的过程中，一般都会有复杂的环境依赖需要进行调测并固化。面对开发中的开发环境的脆弱和多轨切换问题，在ModelArts的AI开发最佳实践中，通过容器镜像的方式，将运行环境进行固化，以这种方式不仅能够进行依赖管理，而且可以方便的完成工作环境切换。配合ModelArts提供的云化容器资源使用，可以更加快速、高效的进行AI开发与模型实验的迭代等。

ModelArts默认提供了一组预置镜像供开发使用，这些镜像有以下特点：

- 零配置，即开即用，面向特定的场景，将AI开发过程中常用的依赖环境进行固化，提供合适的软件、操作系统、网络等配置策略，通过在硬件上的充分测试，确保其兼容性和性能最合适。
- 方便自定义，预置镜像已经在SWR仓库中，通过对预置镜像的扩展完成自定义镜像注册。
- 安全可信，基于安全加固最佳实践，访问策略、用户权限划分、开发软件漏洞扫描、操作系统安全加固等方式，确保镜像使用的安全性。

当用户对深度学习引擎、开发库有特殊需求场景的时候，预置镜像已经不能满足用户需求。ModelArts提供自定义镜像功能支持用户自定义运行引擎。

ModelArts底层采用容器技术，自定义镜像指的是用户自行制作容器镜像并在ModelArts上运行。自定义镜像功能支持自由文本形式的命令行参数和环境变量，灵活性比较高，便于支持任意计算引擎的作业启动需求。

ModelArts 的预置镜像使用场景

ModelArts给用户提供了预置镜像，用户可以直接使用预置镜像创建Notebook实例，在实例中进行依赖安装与配置后，保存为自定义镜像，可直接用于ModelArts训练，而不需要做适配。同时也可以使用预置镜像直接提交训练作业、创建AI应用等。

ModelArts提供的预置镜像版本是依据用户反馈和版本稳定性决定的。当用户的功能开发基于ModelArts提供的版本能够满足的时候，比如用户开发基于MindSpore1.X，建议用户使用预置镜像，这些镜像经过充分的功能验证，并且已经预置了很多常用的安装包，用户无需花费过多的时间来配置环境即可使用。

ModelArts 的自定义镜像使用场景

- **Notebook中使用自定义镜像**
当Notebook预置镜像不能满足需求时，用户可以制作自定义镜像。在镜像中自行安装与配置环境依赖软件及信息，并制作作为自定义镜像，用于创建新的Notebook实例。
- **使用自定义镜像训练作业**
如果您已经在本地完成模型开发或训练脚本的开发，且您使用的AI引擎是ModelArts不支持的框架。您可以制作自定义镜像，并上传至SWR服务。您可以在ModelArts使用此自定义镜像创建训练作业，使用ModelArts提供的资源训练模型。
- **使用自定义镜像创建AI应用**
如果您使用了ModelArts不支持的AI引擎开发模型，也可通过制作自定义镜像，导入ModelArts创建为AI应用，并支持进行统一管理和部署为服务。

📖 说明

用户制作的自定义镜像，使用的场景不同，镜像规则也不同，具体如下：

- 通用规则：SWR镜像类型为“私有”时，才可以共享给他人，适用于开发环境、训练作业、AI应用。
- 开发环境：SWR镜像类型为“公开”时，其它用户才可以在ModelArts镜像管理页面注册使用。
- 训练作业：SWR镜像类型为“公开”时，在使用自定义镜像创建训练作业时，在“镜像”输入框内直接填写“组织名称/镜像名称:版本名称”即可。例如：公开镜像的SWR地址为“swr.cn-north-4.myhuaweicloud.com/test-image/tensorflow2_1_1:1.1.1”，则在创建训练作业的“镜像”输入框里内直接填“test-images/tensorflow2_1_1:1.1.1”。

自定义镜像功能关联服务介绍

使用自定义镜像功能可能涉及以下服务：

- **容器镜像服务**
容器镜像服务（Software Repository for Container，SWR）是一种支持镜像全生命周期管理的服务，提供简单易用、安全可靠的镜像管理功能，帮助您快速部署容器化服务。您可以通过界面、社区CLI和原生API上传、下载和管理容器镜像。
您制作的自定义镜像需要上传至SWR服务。ModelArts训练和创建AI应用使用的自定义镜像需要从SWR服务管理列表获取。

图 1-1 获取镜像列表



- **对象存储服务**
对象存储服务（Object Storage Service，OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。
在使用ModelArts时存在与OBS的数据交互，您需要使用的数据可以存储至OBS。
- **弹性云服务器**

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的基础的计算组件。弹性云服务器创建成功后，您就可以像使用自己的本地PC或物理服务器一样，使用弹性云服务器。

在制作自定义镜像时，您可以在本地环境或者ECS上完成自定义镜像制作。

说明

在您使用自定义镜像功能时，ModelArts可能需要访问您的容器镜像服务SWR、对象存储服务OBS等依赖服务，如果没有授权，这些功能将不能正常使用。建议您使用委托授权功能，将依赖服务操作权限委托给ModelArts服务，让ModelArts以您的身份使用依赖服务，代替您进行一些资源操作。详细操作参见使用[委托授权](#)。

2 使用预置镜像

2.1 统一镜像介绍

ModelArts提供了ARM+Ascend规格的统一镜像，包括MindSpore、PyTorch，适用于开发环境，模型训练，服务部署，请参考[统一镜像列表](#)。

统一镜像列表

ModelArts提供了ARM+Ascend规格的统一镜像，包括MindSpore、PyTorch。适用于开发环境，模型训练，服务部署，请参考[统一镜像列表](#)。[表2-1](#)、[表2-2](#)所示镜像仅发布在西南-贵阳一区域。

表 2-1 MindSpore

| 预置镜像 | 适配芯片 | 适用范围 |
|---|--------------|------------------|
| mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |
| mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |
| mindspore_2.2.10-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |

表 2-2 PyTorch

| 预置镜像 | 适配芯片 | 适用范围 |
|---|--------------|------------------|
| pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |
| pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |

| 预置镜像 | 适配芯片 | 适用范围 |
|---|--------------|------------------|
| pytorch_1.11.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b | Ascend snt9b | Notebook、训练、推理部署 |

镜像一： mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

表 2-3 mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

| AI引擎 框架 | URL | 包含的依赖项 | |
|---|--|--|--|
| | | PyPI 程序包 | Yum 软件包 |
| mindspore 2.2.0 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC 1 | swr.<region>.myhuaweicloud.com/atelier/ mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 | mindspore 2.2.0 ipykernel 6.7.0 ipython 8.17.2 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.22.0 pandas 1.2.5 pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.3.3 mindinsight 2.2.0 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |
| | 例如： 西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/ mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 | | |

镜像二：mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b

表 2-4 mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|---|---|--|---|
| mindspore 2.1.0 + mindspore-lite 2.1.0 + Ascend CANN Toolkit 6.3.RC 2 | swr.<region>.myhuaweicloud.com/atelier/ mindspore_2_0_ascend:mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b-20231009152946-e7b7e70 例如： 西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/ mindspore_2_0_ascend:mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b-20231009152946-e7b7e70 | PyPI 程序包 mindspore 2.1.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.9 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.21.6 pandas 1.3.5 pillow 9.5.0 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.7.3 scikit-learn 1.0.2 tornado 6.2 mindinsight 2.1.0 | Yum 软件包 cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像三：pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b

表 2-5 pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|---|---|---|--|
| | | PyPI 程序包 | Yum 软件包 |
| pytorch 1.11 + minds pore-lite 2.1.0 + Ascend CANN Toolkit 6.3.RC 2 | swr.<region>.myhuaweicloud.com/atelier/ pytorch_1_11_ascend:pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b-20231009152946-e7b7e70 例如： 西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/ pytorch_1_11_ascend:pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b-20231009152946-e7b7e70 | torch-1.11.0 apex 0.1-ascend-20230719 torch_npu 1.11.0.post1.dev20230719 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.9 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.0.1.rc0.ff1c0c8 numpy 1.21.6 pandas 1.3.5 pillow 9.5.0 pip 21.0.1 psutil 5.9.5 PyYAML 5.3.1 scipy 1.7.3 scikit-learn 1.0.2 tornado 6.2 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像四: mindspore_2.2.10-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b

表 2-6 mindspore_2.2.10-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|--|--|--|--|
| mindspore 2.2.10 + mindspore-lite 2.2.10 + Ascend CANN Toolkit 7.0.0 | swr.<region>.myhuaweicloud.com/atelier/ mindspore_2_2_ascend:mindspore_2.2.10-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240228101133-dc2f966 例如: 西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/ mindspore_2_2_ascend:mindspore_2.2.10-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240228101133-dc2f966 | PyPI 程序包 | Yum 软件包 |
| | | ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.22.0 pandas 1.3.5 pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像五: pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b

表 2-7 pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|---|--|--|--|
| | | PyPI 程序包 | Yum 软件包 |
| pytorch 2.1.0 + mindspore-lite 2.2.10 + Ascend CANN Toolkit 7.0.0 | swr.<region>.myhuaweicloud.com/atelier/ pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240228101133-dc2f966 例如: 西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/ pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240228101133-dc2f966 | ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.22.0 pandas 1.3.5 pillow 10.2.0 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像六：pytorch_1.11.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b

表 2-8 pytorch_1.11.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|---|---|--|--|
| | | PyPI 程序包 | Yum 软件包 |
| pytorc h 1.11.0 + minds pore- lite 2.2.10 + Ascend CANN Toolkit 7.0.0 | swr.<region>.myhuaweicloud.co m/atelier/ pytorch_1_11_ascend:pytorch_1.1 1.0-cann_7.0.0-py_3.9- hce_2.0.2312-aarch64- snt9b-20240228101133-dc2f966 例如：西南-贵阳一 swr.cn- southwest-2.myhuaweicloud.com /atelier/ pytorch_1_11_ascend:pytorch_1.1 1.0-cann_7.0.0-py_3.9- hce_2.0.2312-aarch64- snt9b-20240228101133-dc2f966 | ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.22.0 pandas 1.3.5 pillow 10.2.0 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像安全加固

预置的基础镜像中存在cpp、gcc等调试/编译工具，如果您不需要使用这些工具，可以通过运行脚本删除。

创建一个run.sh脚本文件，文件中的代码内容如下。然后在容器中执行sh run.sh命令运行脚本。

```
#!/bin/bash
delete_sniff_compiler()
{
    echo "[+][001] start remove debug tools"
    rm -rf /usr/bin/readelf
    rm -rf /usr/bin/gcc-nm
    #readelf根据需要决定是否删除
    #rm -rf /usr/local/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/aarch64-target-linux-gnu/bin/readelf
    rm -rf /usr/bin/gcc
    rm -rf /usr/bin/cpp
    rm -rf /usr/bin/objdump
    echo "[+] complete"
```

```
}  
hardening_ssh_config()  
{  
  sed -i "s/Subsystem/#Subsystem/g" /etc/ssh/sshd_config #关闭sftp服务  
  sed -i "s/^.MaxAuthTries.*/MaxAuthTries 6/g" /etc/ssh/sshd_config #开启防暴力机制  
  sed -i "s/^.ClientAliveInterval.*/ClientAliveInterval 300/g" /etc/ssh/sshd_config #开启会话超时机制  
  systemctl restart /etc/ssh/sshd_config  
  chmod 600 /home/ma-user/.ssh/id_rsa* #收缩公私钥文件权限  
  chmod 600 /home/ma-user/etc/ssh_host_rsa_key* #收缩公私钥文件权限  
  sed -i "s/ma-user/#ma-user/g" /etc/sudoers #不允许ma-user用户免密执行所有命令  
}  
delete_sniff_compiler  
hardening_ssh_config
```

Ascend镜像中存在hcc编译器，具体说明请参见昇腾社区提供的[HCC编译器说明文档](#)。

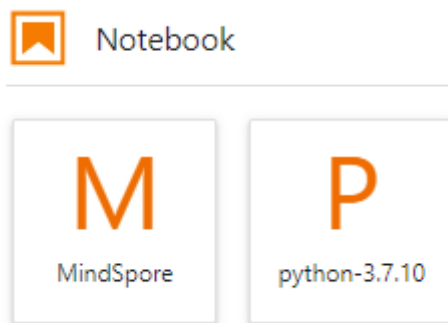
2.2 Notebook 基础镜像介绍

2.2.1 Notebook 基础镜像功能

预置镜像功能

ModelArts开发环境提供的预置镜像主要包含：

- 常用预置包：基于标准的Conda环境，预置了常用的AI引擎，常用的数据分析软件包，例如Pandas，Numpy等，常用的工具软件，例如cuda，cudnn等，满足AI开发常用需求。
- 预置Conda环境：每个预置镜像都会创建一个相对应的Conda环境和一个基础Conda环境python（不包含任何AI引擎），如预置MindSpore所对应的Conda环境如下。



用户可以根据是否使用AI引擎Mindspore参与功能调试，选择不同的Conda环境。

- Notebook：是一款Web应用，用户能够在界面编写代码，并且将代码、数学方程和可视化内容组合到一个文档中。
- JupyterLab插件：插件包括规格切换，分享案例到AI Gallery进行交流，停止实例（实例停止后CPU、Memory不再计费）等，提升用户体验。
- 支持SSH远程连接功能：通过SSH连接启动实例，在本地调试就可以操作实例，方便调试。
- 预置镜像支持功能开发：基于ModelArts预置镜像进行依赖安装配置后，保存为自定义镜像，能直接在ModelArts用于训练作业。

ModelArts提供的预置镜像是以ma-user启动的，用户进入实例后，工作目录默认是“/home/ma-user/work”。

创建实例，持久化存储挂载存放路径为“/home/ma-user/work”目录，存放在work目录的内容，在实例停止，重新启动后依然保留，其他目录下的内容不会保留，使用开发环境时建议将需要持久化的数据放在“/home/ma-user/work”目录。

```
Terminal 1
( . ) [ma-user work]$pwd
/home/ma-user/work
(l, ) [ma-user work]$
```

Notebook中提供的常见预置镜像列表和镜像详情介绍，请参见[Notebook基础镜像列表](#)。

使用预置镜像创建 Notebook 实例

创建Notebook实例时选择预置镜像，创建成功后，打开Notebook即可使用。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”页面。
2. 单击“创建”，进入“创建Notebook”页面，选择公共镜像，选择资源、规格等参数并提交。详细参数请参见[创建Notebook实例](#)。
3. Notebook实例状态为“运行中”时，即可以打开Notebook使用创建的镜像。

2.2.2 Notebook 基础镜像列表

ModelArts开发环境提供Docker容器镜像，可作为预构建容器运行。预置镜像里面包含PyTorch, Tensorflow, MindSpore等常用AI引擎框架，镜像命名以AI引擎为主，并且每个镜像里面都预置了很多常用包，用户可以直接使用而无需重新安装。

开发环境预置镜像分为X86和ARM两类：

表 2-9 X86 预置镜像列表

| 引擎类型 | 镜像名称 |
|------------|--|
| PyTorch | pytorch1.8-cuda10.2-cudnn7-ubuntu18.04 |
| | pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 |
| | pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 |
| Tensorflow | tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04 |
| | tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04 |
| MindSpore | mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04 |
| | mindspore1.7.0-py3.7-ubuntu18.04 |
| | mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04 |
| | mindspore1.2.0-openmpi2.1.1-ubuntu18.04 |

| 引擎类型 | 镜像名称 |
|----------------------|------------------------------------|
| 无AI引擎（专用于自定义镜像的基础镜像） | conda3-cuda10.2-cudnn7-ubuntu18.04 |
| | conda3-ubuntu18.04 |

表 2-10 ARM 预置镜像列表

| 引擎类型 | 镜像名称 |
|------------|--|
| TensorFlow | tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64 |
| | tensorflow1.15-cann5.1.0-py3.7-euler2.8.3 |
| MindSpore | mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3 |
| | mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3 |
| | mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3 |

2.2.3 Notebook 基础镜像 x86 PyTorch

PyTorch包含三种镜像：pytorch1.8-cuda10.2-cudnn7-ubuntu18.04，pytorch1.10-cuda10.2-cudnn7-ubuntu18.04，pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

镜像一：pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

表 2-11 pytorch1.8-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|-------------|------------------|--|----------|------------|
| Pytorch 1.8 | 是 (cuda 10.2) | swr. {region_id}.myhuaweicloud.com/atelier/ pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|---------------------|-----|--|--|
| | | | torch 1.8.0 torchvision 0.9.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip |

镜像二：pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

表 2-12 pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------------|---------------------|--|----------|------------|
| Pytorch 1.10 | 是 (cuda 10.2) | swr. {region_id}.myhuaweicloud.com/atelier/ pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|---------------------|-----|---|--|
| | | | torch 1.10.2 torchvision 0.11.3 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip |

镜像三：pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

表 2-13 pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|-------------|---------------------|---|----------|------------|
| Pytorch 1.4 | 是 (cuda 10.1) | swr. {region_id}.myhuaweicloud.com/atelier/ pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e 华东上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|---------------------|-----|--|--|
| | | | torch 1.4.0 torchvision 0.5.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip |

2.2.4 Notebook 基础镜像 x86 Tensorflow

Tensorflow包含两种镜像：tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04,
tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

镜像一： tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

表 2-14 tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|----------------|---------------------|--|----------|------------|
| Tensorflow 2.1 | 是 (cuda 10.1) | swr. {region_id}.myhuaweicloud.com/atelier/ tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|---------------------|-----|--|--|
| | | | tensorflow 2.1.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

镜像二： tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

表 2-15 tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|---------------------|---------------------|--|----------|------------|
| Tensorflow 1.13-gpu | 是 (cuda 10.0) | swr. {region_id}.myhuaweicloud.com/atelier/ tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|-----|---|--|
| | | | tensorflow-gpu 1.13.1 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.0.1.rc0.ff1c0c8 numpy 1.17.0 opencv-python 4.1.2.30 pandas 1.1.5 pillow 6.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.2.2 scikit-learn 0.22.1 tornado 6.2 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

2.2.5 Notebook 基础镜像 x86 MindSpore

MindSpore包含四种镜像：mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04, mindspore1.7.0-py3.7-ubuntu18.04, mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04, mindspore1.2.0-openmpi2.1.1-ubuntu18.04

镜像一：mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

表 2-16 mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|-----------------------|------------------|--|----------|------------|
| Mind spore -gpu 1.7.0 | 是 (cuda 10.1) | swr. {region_id}.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|-----|---|--|
| | | | mindspore-gpu 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.17.0 pandas 1.1.5 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.1 mindinsight 1.7.0 mindvision 0.1.0 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

镜像二：mindspore1.7.0-py3.7-ubuntu18.04

表 2-17 mindspore1.7.0-py3.7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|------------------|---------------------|--|----------|------------|
| Mind spore 1.7.0 | 无 | swr. {region_id}.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|-----|---|--|
| | | | mindspore 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.17.0 pandas 1.1.5 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.1 mindinsight 1.7.0 mindvision 0.1.0 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

镜像三：mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

表 2-18 mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|----------------------|---------------------|--|----------|------------|
| Mind spore-gpu 1.2.0 | 是 (cuda 10.1) | swr. {region_id}.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|-----|---|--|
| | | | mindspore-gpu 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 pillow 6.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

镜像四: mindspore1.2.0-openmpi2.1.1-ubuntu18.04

表 2-19 mindspore1.2.0-openmpi2.1.1-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|------------------|---------------------|--|----------|------------|
| Mind spore 1.2.0 | 无 | swr. {region_id}.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e 例如: 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e | PyPI 程序包 | Ubuntu 软件包 |

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|-----|---|--|
| | | | mindspore 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 6.2.0 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0 | automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip |

2.2.6 Notebook 基础镜像 x86 自定义专用镜像

自定义镜像包含两种镜像：conda3-cuda10.2-cudnn7-ubuntu18.04，conda3-ubuntu18.04，该类镜像是无AI引擎以及相关的软件包，镜像较小，只有2~5G。用户

使用此类镜像做基础镜像，安装自己需要的引擎版本和依赖包，可扩展性更高。并且这些镜像预置了一些开发环境启动所必要的配置，用户无需对此做任何适配，安装所需的软件包即可使用。

此类镜像为最基础的镜像，主要应对用户做自定义镜像时基础镜像太大的问题，所以镜像中未安装任何组件；如果需使用OBS SDK相关功能，推荐使用ModelArts SDK进行文件复制等操作，详细操作请参考[文件传输](#)。

镜像一：conda3-cuda10.2-cudnn7-ubuntu18.04

表 2-20 conda3-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|--|--|---|
| 无 | 是 (cuda 10.2) | swr. {region_id}.myhuaweicloud.com/atelier/ user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c | PyPI 程序包 | Ubuntu 软件包 |
| | | | ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.21.6 pandas 1.3.5 pillow 9.2.0 pip 20.3.3 psutil 5.9.1 PyYAML 6.0 scipy 1.7.3 tornado 6.2 | automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep libcudnn7 libcudnn7-dev nginx python3 rpm tar unzip vim wget zip |

镜像二：conda3-ubuntu18.04

表 2-21 conda3-ubuntu18.04 镜像介绍

| AI引擎框架 | 是否使用GPU (CUDA版本) | URL | 包含的依赖项 | |
|--------|------------------|--|--|---|
| 无 | 否 | swr. {region_id}.myhuaweicloud.com/atelier/ user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/ user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/ user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/ user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c | PyPI 程序包 | Ubuntu 软件包 |
| | | | ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.21.6 pandas 1.3.5 pillow 9.2.0 pip 20.3.3 psutil 5.9.1 PyYAML 6.0 scipy 1.7.3 tornado 6.2 | automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep nginx python3 rpm tar unzip vim wget zip |

2.2.7 Notebook 基础镜像 ARM MindSpore

ARM MindSpore包含三种镜像，mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3，mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3，mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3

镜像一：mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3

表 2-22 mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|------------------------------------|--|---|--|
| Minds pore- Ascend 1.10.0 | {region_id}.myhuaweicloud.com/ atelier/ mindspore_1_10_ascend:mindspo re_1.10.0-cann_6.0.1-py_3.7- euler_2.8.3-aarch64- d910-20230303173945-815d627 例如： 华北-北京四 swr.cn- north-4.myhuaweicloud.com/ atelier/ mindspore_1_10_ascend:mindspo re_1.10.0-cann_6.0.1-py_3.7- euler_2.8.3-aarch64- d910-20230303173945-815d627 | PyPI 程序包 | Yum 软件包 |
| | | mindspore-ascend 1.10.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.5 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.2.1 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.0.1.rc0.ff1c0c8 numpy 1.21.2 pandas 1.1.3 pillow 9.3.0 pip 22.3.1 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.2 mindinsight 1.9.0 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像二：mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3

表 2-23 mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|-----------------|--|--|--|
| | | PyPI 程序包 | Yum 软件包 |
| MindSpore 1.9.0 | swr. {region_id}.myhuaweicloud.com/ atelier/ mindspore_1_9_ascend:mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64-d910-20221116111529 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/ atelier/ mindspore_1_9_ascend:mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64-d910-20221116111529 | mindspore-ascend 1.9.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.5 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.21.2 pandas 1.1.3 pillow 9.3.0 pip 22.3.1 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.2 mindinsight 1.9.0 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像三：mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3

表 2-24 mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|-----------------------------------|--|--|--|
| | | PyPI 程序包 | Yum 软件包 |
| Minds pore- Ascend 1.7.0 | swr. {region_id}.myhuaweicloud.com/ atelier/ mindspore_1_7_0:mindspore_1.7. 0-cann_5.1.0-py_3.7-euler_2.8.3- aarch64-d910-20220906 例如： 华北-北京四 swr.cn- north-4.myhuaweicloud.com/ atelier/ mindspore_1_7_0:mindspore_1.7. 0-cann_5.1.0-py_3.7-euler_2.8.3- aarch64-d910-20220906 | mindspore-ascend 1.7.0 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.0.1.rc0.ff1c0c8 numpy 1.21.2 pandas 1.1.3 pillow 9.2.0 pip 22.1.2 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.1 tornado 6.2 mindinsight 1.7.0 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

2.2.8 Notebook 基础镜像 ARM TensorFlow

ARM TensorFlow镜像包含两种，tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64、tensorflow1.15-cann5.1.0-py3.7-euler2.8.3

镜像一： tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64

表 2-25 tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64 镜像介绍

| AI引擎 框架 | URL | 包含的依赖项 | |
|-----------------------------------|--|---|--|
| | | PyPI 程序包 | Yum 软件包 |
| Minds pore- Ascend 1.7.0 | swr. {region_id}.myhuaweicloud.com/ atelier/notebook2.0-mul-kernel- arm-ascend-cp37:5.0.1- c81-20220726 例如： 华北-北京四 swr.cn- north-4.myhuaweicloud.com/ atelier/notebook2.0-mul-kernel- arm-ascend-cp37:5.0.1- c81-20220726 | mindspore-ascend 1.7.0 ipykernel 6.7.0 ipython 7.29.0 jupyter-client 7.0.6 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.1.2 modelarts 1.4.22 moxing-framework 2.0.0.rc2.4b57a67b numpy 1.17.5 pandas 1.1.3 pillow 7.0.0 pip 21.2.4 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.1 tornado 6.1 mindinsight 1.7.0 | cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip |

镜像二： tensorflow1.15-cann5.1.0-py3.7-euler2.8.3

表 2-26 tensorflow1.15-cann5.1.0-py3.7-euler2.8.3 镜像介绍

| AI引擎框架 | 是否使用昇腾 (CANN 版本) | URL | 包含的依赖项 | |
|-----------------|------------------|---|---|--|
| Tensorflow 1.15 | 是 (CANN 5.1) | swr.{region-id}.{局点域名}/atelier/ tensorflow_1_15_ascend:tensorflow_1.15-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906 | PyPI 程序包 | Yum 软件包 |
| | | | tensorflow 1.15.0 tensorboard 1.15.0 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.2 ma-cau-adapter 1.1.2 ma-cli 1.1.3 matplotlib 3.5.1 modelarts 1.4.7 moxing-framework 2.0.1.rc0.ff1c0c8 numpy 1.17.5 pandas 0.24.2 pillow 9.2.0 pip 22.1.2 psutil 5.7.0 PyYAML 5.3.1 scipy 1.3.3 scikit-learn 0.20.0 tornado 6.2 | ca-certificates. noarch cmake cpp curl gcc-c++ gcc gdb grep nginx python3 rpm tar unzip vim wget zip |

2.3 训练基础镜像详情介绍

2.3.1 训练基础镜像列表

ModelArts平台提供了Tensorflow, PyTorch, MindSpore等常用深度学习任务的基础镜像, 镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时, 您可以基于这些基础镜像制作一个新的镜像并进行训练。

训练基础镜像列表

ModelArts中预置的训练基础镜像如下表所示。

表 2-27 ModelArts 训练基础镜像列表

| 引擎类型 | 版本名称 |
|------------|--|
| PyTorch | pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |
| TensorFlow | tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 |
| Horovod | horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 |
| | horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |
| MPI | mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64 |

📖 说明

不同区域支持的AI引擎有差异, 请以实际环境为准。

2.3.2 训练基础镜像详情 (PyTorch)

介绍预置的PyTorch镜像详情。

引擎版本: `pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64`

- 镜像地址: `swr.{region}.myhuaweicloud.com/aip/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-roma-20220309171256-40adcc1`
- 镜像构建时间: 20220309171256 (yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本: Ubuntu 18.04.4 LTS
- cuda: 10.2.89
- cudnn: 7.6.5.32
- Python解释器路径及版本: `/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10`
- 三方包安装路径: `/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages`
- 部分三方包版本信息列表:

Cython 0.27.3
 dask 2022.2.0
 easydict 1.9

```
enum34 1.1.10
torch 1.8.0
Flask 1.1.1
grpcio 1.44.0
gunicorn 20.1.0
idna 3.3
torchtext 0.5.0
imageio 2.16.0
imgaug 0.4.0
lxml 4.8.0
matplotlib 3.5.1
torchvision 0.9.0
mmcv 1.2.7
numba 0.47.0
numpy 1.21.5
opencv-python 4.1.2.30
toml 0.10.2
pandas 1.1.5
Pillow 9.0.1
pip 21.2.2
protobuf 3.19.4
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 6.0
requests 2.27.1
scikit-image 0.19.2
...
```

- 历史版本：无

2.3.3 训练基础镜像详情（TensorFlow）

介绍预置的TensorFlow镜像详情。

引擎版本：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/tensorflow_2_1:train-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- 镜像构建时间：20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python，python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- 部分三方包版本信息列表：

```
Cython 0.29.21
dask 2021.9.0
easydict 1.9
enum34 1.1.10
tensorflow 2.1.0
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
tensorflow-estimator 2.1.0
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
```

```
matplotlib 3.4.3
termcolor 1.1.0
scikit-image 0.18.3
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
tiff file 2021.8.30
pandas 1.1.5
Pillow 6.2.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
...
```

- 历史版本：无

2.3.4 训练基础镜像详情（Horovod）

介绍预置的Horovod镜像详情。

引擎版本一：horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：`swr.{region}.myhuaweicloud.com/aip/horovod_tensorflow:train-horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d`
- 镜像构建时间：20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：`/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python`，python 3.7.10
- 三方包安装路径：`/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages`
- 部分三方包版本信息列表：

```
Cython 0.29.21
dask 2021.9.0
easydict 1.9
enum34 1.1.10
horovod 0.20.0
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
tensorboard 2.1.1
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
matplotlib 3.4.3
tensorflow-gpu 2.1.0
tensorboardX 2.0
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
toml 0.10.2
pandas 1.1.5
Pillow 6.2.0
```



```
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
scikit-image 0.18.3
...
```

- 历史版本：无

引擎版本二： horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址： swr.{region}.myhuaweicloud.com/aip/horovod_pytorch:train-horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- 镜像构建时间： 20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本： Ubuntu 18.04.4 LTS
- cuda： 11.1.1
- cudnn： 8.0.5.39
- Python解释器路径及版本： /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python, python 3.7.10
- 三方包安装路径： /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages
- 部分三方包版本信息列表：

```
Cython 0.27.3
dask 2021.9.0
easydict 1.9
enum34 1.1.10
horovod 0.22.1
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
mmcv 1.2.7
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
matplotlib 3.4.3
torch 1.8.0
tensorboardX 2.0
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
torchtext 0.5.0
pandas 1.1.5
Pillow 6.2.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
scikit-image 0.18.3
torchvision 0.9.0
...
```

- 历史版本：无

2.3.5 训练基础镜像详情（MPI）

介绍预置的mindspore_1.3.0镜像详情。

引擎版本：mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/mindspore_1_3_0:train-mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-roma-20211104202338-f258e59
- 镜像构建时间：20211104202338(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python，python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages
- 部分三方包版本信息列表：

```
requests 2.26.0
dask 2021.9.0
easydict 1.9
enum34 1.1.10
mindspore-gpu 1.3.0
Flask 1.1.1
grpcio 1.41.1
gunicorn 20.1.0
idna 3.3
PyYAML 5.1
imageio 2.10.1
imgaug 0.4.0
lxml 4.6.4
matplotlib 3.4.2
psutil 5.8.0
scikit-image 0.18.3
numba 0.47.0
numpy 1.17.0
opencv-python 4.5.2.54
tifffile 2021.11.2
pandas 1.1.5
Pillow 8.4.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
...
```
- 历史版本：无

2.3.6 预置框架启动流程说明

2.3.6.1 PyTorch

ModelArts训练服务支持了多种AI引擎，并对不同的引擎提供了针对性适配，用户在使用这些引擎进行模型训练时，训练的启动命令也需要做相应适配，本文讲解了使用PyTorch引擎所需要做的适配。

PyTorch 框架启动原理

规格和节点个数

下面以选择“GPU: 8*GP-Vnt1 | CPU: 72核 | 内存: 512GB”规格为例，介绍在单机和分布式场景下ModelArts规格资源的分配情况。

单机作业时（即选择的节点数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

分布式作业时（即选择的节点数大于1），worker的数量和创建作业时选择的节点数一致，每个worker将被分配到所选规格对应的计算资源。例如计算节点个数为“2”时，将启动2个worker，每个worker拥有“GPU: 8*GP-Vnt1 | CPU: 72核 | 内存: 512GB”的计算资源。目前PyTorch引擎仅支持GPU和CPU类型的规格，如果需要使用昇腾规格，请参考[Ascend-Powered-Engine](#)。

网络通信介绍

- 单机作业不涉及网络通信情况。
- 分布式作业的涉及网络通信则可以分为节点内网络通信和节点间网络通信。

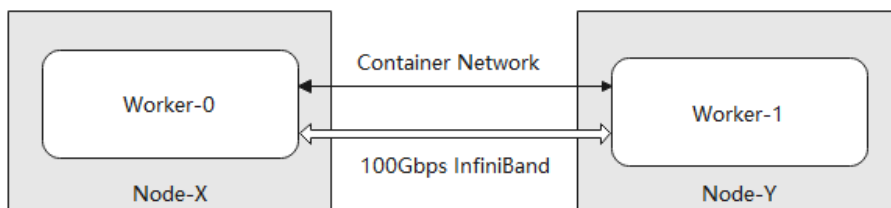
节点内网络

使用NVLink和共享内存通信。

节点间网络

当计算节点个数大于1时，将启动PyTorch引擎分布式训练模式。PyTorch引擎的分布式模式如下图所示，worker之间可通过容器网络和100Gbps的InfiniBand网卡或RoCE网卡通信，部分规格中使用RoCE网卡，将在规格中额外提示。其中，容器网络可以通过DNS域名通信，但网络性能一般，可用于点对点小规模通信需求；InfiniBand网络和RoCE网络为高性能网络，可用于集合通信等分布式训练的场景。

图 2-1 分布式模式



启动命令

训练服务使用作业镜像中默认的python解释器启动训练脚本（即“which python”命令指向的可执行文件），启动时的工作目录（即pwd命令或python中“os.getcwd()”返回的目录）为“/home/ma-user/user-job-dir/<代码目录最外层目录名>”。

• 单机单卡启动命令

```
python <启动文件相对路径> <作业运行参数>
```

- 启动文件相对路径：启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。
- 作业运行参数：训练作业中配置的运行参数。

图 2-2 创建训练作业详情

* 创建方式 自定义算法

* 启动方式 预置框架 自定义

PyTorch pytorch_1.8.2-cuda_10.2-py_3...

* 代码目录 /modelarts-train-test/gpu-train/ 选择

* 启动文件 /modelarts-train-test/gpu-train/train.py 选择

本地代码目录 /home/ma-user/modelarts/user-job-dir

工作目录 /home/ma-user/modelarts/user-job-dir 选择

输入 增加训练输入

输出 增加训练输出

超参 epochs = 5 增加超参

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

● 单机多卡启动命令

```
python <启动文件相对路径> --init_method "tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" <作业运行参数>
```

- 启动文件相对路径：启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。
- `${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}`：worker-0所在容器的域名，请参考[系统默认加载的环境变量说明](#)。
- port: worker-0所在容器的默认通信端口。
- 作业运行参数：训练作业中配置的运行参数。

图 2-3 创建训练作业详情

* 创建方式 自定义算法

* 启动方式 预置框架 自定义

PyTorch pytorch_1.8.2-cuda_10.2-py_3...

* 代码目录 /modelarts-train-test/gpu-train/ 选择

* 启动文件 /modelarts-train-test/gpu-train/train.py 选择

本地代码目录 /home/ma-user/modelarts/user-job-dir

工作目录 /home/ma-user/modelarts/user-job-dir 选择

输入 增加训练输入

输出 增加训练输出

超参 epochs = 5 增加超参

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --init_method "tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" --epochs 5
```

- **多机多卡启动命令**

```
python <启动文件相对路径> --init_method "tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" --rank <rank_id> --world_size <node_num> <作业运行参数>
```

- 启动文件相对路径：启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。
- \${MA_VJ_NAME}-\${MA_TASK_NAME}-0.\${MA_VJ_NAME}：worker-0所在容器的域名，请参考[系统默认加载的环境变量说明](#)。
- port：worker-0所在容器的默认通信端口。
- rank：worker的序号。
- node_num：worker的数量。
- 作业运行参数：训练作业中配置的运行参数。

图 2-4 创建训练作业详情

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --init_method "tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" --rank "${rank_id}" --world_size "${node_num}" --epochs 5
```

2.3.6.2 Tensorflow

ModelArts训练服务支持了多种AI框架，并对不同的引擎提供了针对性适配，用户在使用这些框架进行模型训练时，训练的启动命令也需要做相应适配。本文介绍了Tensorflow框架启动原理、控制台上创建训练任务时后台对应的启动命令。

Tensorflow 框架启动原理

规格和节点个数

下面以选择“GPU: 8*GP-Vnt1 | CPU: 72核 | 内存: 512GB”规格为例，介绍在单机和分布式场景下ModelArts规格资源的分配情况。

单机作业时（即选择的节点数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

分布式作业时（即选择的节点数大于1），ModelArts会优先在相同节点上启动一个parameter server（以下简称ps）和一个worker，其中ps将分配一半的CPU和内存资源，即ps拥有“CPU: 36核 | 内存: 256GB”的计算资源，worker拥有“GPU: 8*GP-Vnt1 | CPU: 36核 | 内存: 256GB”的计算资源。

需要注意的是ps只会分配到CPU和内存资源，而worker除CPU和内存外，还可能分配到加速卡（纯CPU规格除外）。如本例中，每个worker将分配到八张GP Vnt1加速卡，如果ps和worker在相同节点上启动，则磁盘资源由ps和worker共享。

网络通信介绍

- 单机作业不涉及网络通信情况。
- 分布式作业的涉及网络通信则可以分为节点内网络通信和节点间网络通信。

节点内网络

节点内网络通信即同一个节点上的ps和woker间的网络通信，又可以分为两种情况：容器网络和主机网络。

- 在使用公共规格进行训练时，使用的是容器网络。
- 在使用专属池训练时，如果节点配置的是RoCE网卡，使用的是主机网络；如果节点配置的是Infiniband网卡，使用的是容器网络。

节点间网络

分布式作业存在节点间ps和worker的通信，当前ModelArts主要提供了Infiniband网卡或RoCE网卡，带宽高达100Gb/s。

启动命令

训练服务使用作业镜像中默认的python解释器启动训练脚本（即“which python”命令指向的可执行文件），启动时的工作目录（即pwd命令或python中“os.getcwd()”返回的目录）为“/home/ma-user/user-job-dir/<代码目录最外层目录名>”。

- **单机启动命令**
python <启动文件相对路径> <作业运行参数>
 - 启动文件相对路径：启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。
 - 作业运行参数：训练作业中配置的运行参数。

图 2-5 创建训练作业详情

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

- **分布式启动命令**

```
python --task_index ${VC_TASK_INDEX} --ps_hosts ${TF_PS_HOSTS} --worker_hosts ${TF_WORKER_HOSTS} --job_name ${MA_TASK_NAME} <启动文件相对路径> <作业运行参数>
```

- VC_TASK_INDEX: task序号，如0/1/2。
- TF_PS_HOSTS : ps节点地址数组，如[xx-ps-0.xx:TCP_PORT,xx-ps-1.xx:TCP_PORT]，TCP_PORT是一个在5000~10000的随机端口。
- TF_WORKER_HOSTS: worker节点地址数组，如[xx-worker-0.xx:TCP_PORT,xx-worker-1.xx:TCP_PORT]，TCP_PORT是一个在5000~10000的随机端口。
- MA_TASK_NAME: 任务名称，ps或worker。
- 启动文件相对路径: 启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。
- 作业运行参数: 训练作业中配置的运行参数。

图 2-6 创建训练作业详情

The screenshot shows the configuration for a training job. Key settings include:

- Creation Method:** Custom Algorithm
- Startup Method:** Pre-configuration Framework
- Code Directory:** /modelarts-train-test/gpu-train/
- Startup File:** /modelarts-train-test/gpu-train/train.py
- Local Code Directory:** /home/ma-user/modelarts/user-job-dir
- Working Directory:** /home/ma-user/modelarts/user-job-dir
- Hyperparameters:** epochs = 5

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python --task_index "$VC_TASK_INDEX" --ps_hosts "$TF_PS_HOSTS" --worker_hosts "$TF_WORKER_HOSTS" --job_name "$MA_TASK_NAME" /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

2.3.6.3 Ascend-Powered-Engine

ModelArts训练服务支持了多种AI引擎，并对不同的引擎提供了针对性适配，用户在使用这些引擎进行模型训练时，训练的算法代码也需要做相应适配，本文讲解了使用Ascend引擎所需要做的代码适配。

Ascend-Powered-Engine 框架启动原理

在ModelArts算法建界面选择AI引擎时，能够看到一个叫做Ascend-Powered-Engine的AI引擎，它与其他AI引擎相比有些特别。它既不是一个AI框架（如：PyTorch、TensorFlow）也不是一个并行执行框架（如：MPI），而是适配加速芯片Ascend编译的一组AI框架+运行环境+启动方式的集合。

- 由于几乎所有的Ascend加速卡都跑在ARM规格的机器上，因此上层docker镜像也都是ARM镜像。
- 针对GPU场景的镜像中安装了对应版本的CUDA（由英伟达推出的统一计算架构）驱动，而Ascend-Powered-Engine引擎的镜像中都安装了与底层硬件版本适配的CANN（华为针对AI场景推出的异构计算架构）驱动。

规格和节点个数

下面以选择“Ascend: 8 *** | ARM: 192 核 720 GB”规格为例，介绍在单机和分布式场景下ModelArts规格资源的分配情况。

单机作业时（即选择的节点数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

分布式作业时（即选择的节点数大于1），worker的数量和创建作业时选择的节点数一致，每个worker将被分配到所选规格对应的计算资源。例如计算节点个数为2时，将启

动2个worker，每个worker拥有“Ascend: 8*** | ARM: 192 核 768GB 720 GB”的计算资源。

另外，Ascend-Powered-Engine引擎的启动文件，将被平台自动启动为N个进程，N=单节点的Ascend加速卡数。

须知

PyTorch on Ascend不能使用Ascend-Powered-Engine引擎来启动训练作业，应该参考[基于训练作业启动PyTorch DDP on Ascend加速卡训练示例](#)使用自定义镜像来启动训练作业。

网络通信介绍

- 单机作业不涉及网络通信情况。
- 分布式作业的涉及网络通信则可以分为节点内网络通信和节点间网络通信。

节点内网络

使用HCCS和共享内存通信。

节点间网络

worker之间可通过容器网络和Ascend加速卡上的RoCE网络通信。

启动命令

训练服务使用作业镜像中默认的python解释器启动训练脚本（即“which python”命令指向的可执行文件），启动时的工作目录（即pwd命令或python中“os.getcwd()”返回的目录）为“/home/ma-user/user-job-dir/<代码目录最外层目录名>”

启动命令

```
python ${系统自带启动文件} python ${启动文件路径} \  
--${超参键1}=${超参值1} \  
--${超参键2}=${超参值2} \  
...  
--${训练输入参数名称}=${训练输入参数值}  
--${训练输出参数名称}=${训练输出参数值}
```

- 系统自带启动文件：一般为“/home/ma-user/modelarts/run/davincirun.py”
- 启动文件路径：启动文件相对“/home/ma-user/user-job-dir/<代码目录最外层目录名>”的路径。

图 2-7 创建训练作业详情

* 创建方式 自定义算法

* 启动方式 预置框架 自定义

Ascend-Powered-Engi... | mindspore_1.7.0-cann_5.1.0-py...

* 代码目录 选择

* 启动文件 选择

本地代码目录

工作目录 选择

输入

数据集 数据存储位置

获取方式 超参 环境变量

增加训练输入

输出

数据存储位置

获取方式 超参 环境变量

预下载至本地目录 不下载 下载

增加训练输出

超参

| | | |
|------------------|---|--|
| enable_modelarts | = | True |
| is_distribute | = | 0 |
| per_batch_size | = | 1 |
| data_dir | = | /home/ma-user/modelarts/inputs/data_url_0/ |
| pretrained | = | /home/ma-user/modelarts/inputs/data_url_0/ckpt/C |

增加超参

例如控制台上设置如上图所示，则控制台后台执行命令如下：

```
python /home/ma-user/modelarts/run/davincirun.py \
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py \
--enable_modelarts=True \
--is_distribute=0 \
--per_batch_size=1 \
--data_dir=/home/ma-user/modelarts/inputs/data_url_0/ \
--pretrained=/home/ma-user/modelarts/inputs/data_url_0/ckpt/0-320_195200.ckpt \
--data_url=/home/ma-user/modelarts/inputs/data_url_0/ \
--train_url=/home/ma-user/modelarts/outputs/train_url_0/
```

说明

Ascend-Powered-Engine框架单机启动命令和分布式启动命令无区别。

2.3.6.4 Horovod/MPI/MindSpore-GPU

ModelArts训练服务支持了多种AI引擎，并对不同的引擎提供了针对性适配，用户在使用这些引擎进行模型训练时，训练的算法代码也需要做相应适配，本文讲解了使用Horovod/MPI/MindSpore-GPU引擎所需要做的代码适配。

Horovod/MPI/MindSpore-GPU 框架启动原理

规格和节点个数

下面以选择“GPU: 8*GP-Vnt1 | CPU: 72核 | 内存: 512GB”规格为例，介绍在单机和分布式场景下ModelArts规格资源的分配情况。

单机作业时（即选择的节点数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

分布式作业时（即选择的节点数大于1），worker的数量和创建作业时选择的节点数一致，每个worker将被分配到所选规格对应的计算资源。例如计算节点个数为“2”时，将启动2个worker，每个worker拥有“GPU: 8*GP-Vnt1 | CPU: 72核 | 内存: 512GB”的计算资源。

网络通信介绍

- 单机作业不涉及网络通信情况。
- 分布式作业的涉及网络通信则可以分为节点内网络通信和节点间网络通信。

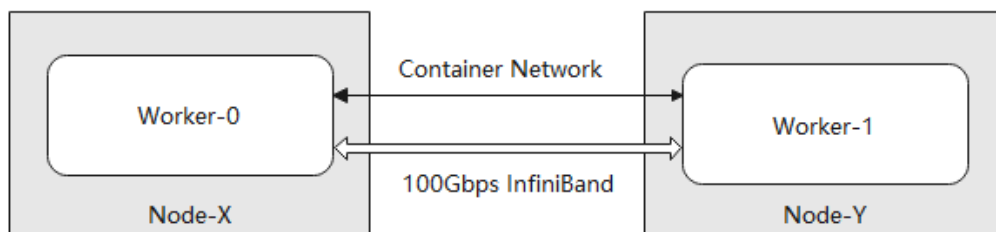
节点内网络

使用NVLink和共享内存通信。

节点间网络

当计算节点个数大于1时，将启动PyTorch引擎分布式训练模式。PyTorch引擎的分布式模式如下图所示，worker之间可通过容器网络和100Gbps的InfiniBand网卡或RoCE网卡通信，部分规格中使用RoCE网卡，将在规格中额外提示。其中，容器网络可以通过DNS域名通信，但网络性能一般，可用于点对点小规模通信需求；InfiniBand网络和RoCE网络为高性能网络，可用于集合通信等分布式训练的场景。

图 2-8 分布式模式



启动命令

训练服务使用作业镜像中默认的python解释器启动训练脚本，即which python命令指向的可执行文件，启动时的工作目录（即pwd命令或python中os.getcwd()返回的目录）为/home/ma-user/user-job-dir/<代码目录最外层目录名>。

启动命令

```
mpirun \  
-np ${OPENMPI_NP} \  
-hostfile ${OPENMPI_HOST_FILE_PATH} \  
-mca plm_rsh_args "-p ${SSHD_PORT}" \  
-tune ${TUNE_ENV_FILE} \  
${OPENMPI_BIND_ARGS} \  
${OPENMPI_X_ARGS} \  
${OPENMPI_MCA_ARGS} \  
${OPENMPI_EXTRA_ARGS} \  
python <启动文件相对路径> <作业运行参数>
```

- OPENMPI_NP: 可控制mpirun启动的进程数, 默认为“GPU 卡数 * 节点数”, 不建议修改。
- OPENMPI_HOST_FILE_PATH: 可控制hostfile参数, 不建议修改。
- SSHD_PORT: 可控制ssh登录端口, 不建议修改。
- TUNE_ENV_FILE: 将worker-0的如下env, 广播到当前训练作业的其他worker节点。
 - MA_ 前缀的env
 - SHARED_ 前缀的env
 - S3_ 前缀的env
 - PATH的env
 - VC_WORKER_ 前缀的env
 - SCC前缀的env
 - CRED前缀的env

```
env|grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED'|grep -v '=$> $'
{TUNE_ENV_FILE}
```
- OPENMPI_BIND_ARGS: 可控制mprun cpu绑核行为, 默认设置如下。
OPENMPI_BIND_ARGS="-bind-to none -map-by slot"
- OPENMPI_X_ARGS: 可控制mpirun -x参数, 默认设置如下。
OPENMPI_X_ARGS="-x LD_LIBRARY_PATH -x HOROVOD_MPI_THREADS_DISABLE=1 -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=ib0,bond0,eth0 -x NCCL_SOCKET_FAMILY=AF_INET -x NCCL_IB_DISABLE=0"
- OPENMPI_MCA_ARGS: 可控制mpirun -mca参数, 默认设置如下。
OPENMPI_MCA_ARGS="-mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true"
- OPENMPI_EXTRA_ARGS: 控制可额外传递到mpirun的参数, 默认设置为空。
- 启动文件相对路径: 启动文件相对 “/home/ma-user/user-job-dir/<代码目录最外层目录名>” 的路径。
- 作业运行参数: 训练作业中配置的运行参数。

图 2-9 创建训练作业详情

例如控制台上设置如上图所示, 则控制台后台执行命令如下:

```
mpirun \
-np ${np} \
-hostfile ${OPENMPI_HOST_FILE_PATH} \
```

```
-mca plm_rsh_args "-p ${SSHD_PORT}" \
-tune ${TUNE_ENV_FILE} \
${OPENMPI_BIND_ARGS} \
${OPENMPI_X_ARGS} \
${OPENMPI_MCA_ARGS} \
${OPENMPI_EXTRA_ARGS} \
python /home/ma-user/user-job-dir/gpu-train/train.py --datasets=obs://modelarts-train-test/gpu-train/
data_url_0
```

📖 说明

Horovod/MPI/MindSpore-GPU框架单机启动命令和分布式启动命令无区别。

2.4 推理基础镜像介绍

2.4.1 推理基础镜像列表

ModelArts的推理平台提供了一系列的基础镜像，用户可以基于这些基础镜像构建自定义镜像，用于部署推理服务。

X86 架构（CPU/GPU）的推理基础镜像

表 2-28 TensorFlow

| AI引擎版本 | 支持的运行环境 | URI |
|--------|----------------------|---|
| 2.1.0 | CPU GPU(cuda10.1) | swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817 |
| 1.15.5 | CPU GPU(cuda11.4) | swr.{region_id}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18 |
| 2.6.0 | CPU GPU(cuda11.2) | swr.{region_id}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18 |

表 2-29 Pytorch

| AI引擎版本 | 支持的运行环境 | URI |
|--------|----------------------|---|
| 1.8.0 | CPU GPU(cuda10.2) | swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817 |
| 1.8.2 | CPU GPU(cuda11.1) | swr.{region_id}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18 |

表 2-30 MindSpore

| AI引擎版本 | 支持的运行环境 | URI |
|--------|---------------|---|
| 1.7.0 | CPU | swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76 |
| 1.7.0 | GPU(cuda10.1) | swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76 |
| 1.7.0 | GPU(cuda11.1) | swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76 |

ARM + Ascend 架构的推理基础镜像

表 2-31 TensorFlow

| AI引擎版本 | 支持的运行环境 | URI |
|--------|---------|--|
| 1.15.0 | Snt9 | swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_1_15_ascend:tensorflow_1.15-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220715093657-9446c6a |

表 2-32 Pytorch

| AI引擎版本 | 支持的运行环境 | URI |
|--------|---------|--|
| 1.8.1 | Snt9 | swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8_ascend:pytorch_1.8.1-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220715093657-9446c6a |

表 2-33 MindSpore

| AI引擎版本 | 支持的运行环境 | URI |
|--------|---------|---|
| 1.7.0 | Snt9 | swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220715093657-9446c6a |

2.4.2 推理基础镜像详情 TensorFlow (CPU/GPU)

ModelArts提供了以下TensorFlow (CPU/GPU) 推理基础镜像：

- 引擎版本一： [tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64](#)
- 引擎版本二： [tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64](#)
- 引擎版本三： [tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64](#)

引擎版本一： [tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64](#)

- 镜像地址： `swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817`
- 镜像构建时间： 20220713110657(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本： Ubuntu 18.04.4 LTS
- cuda： 10.1.243
- cudnn： 7.6.5.32
- Python解释器路径及版本： `/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python`， python 3.7.10
- 三方包安装路径： `/home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages`

- 部分pip安装包列表：

| | |
|----------------------|----------------|
| Cython | 0.29.21 |
| easydict | 1.9 |
| Flask | 2.0.1 |
| grpcio | 1.47.0 |
| gunicorn | 20.1.0 |
| h5py | 3.7.0 |
| ipykernel | 6.7.0 |
| Jinja2 | 3.0.1 |
| lxml | 4.9.1 |
| matplotlib | 3.5.1 |
| moxing-framework | 2.1.0.5d9c87c8 |
| numpy | 1.19.5 |
| opencv-python | 4.1.2.30 |
| pandas | 1.1.5 |
| Pillow | 9.2.0 |
| pip | 22.1.2 |
| protobuf | 3.20.1 |
| psutil | 5.8.0 |
| PyYAML | 5.1 |
| requests | 2.27.1 |
| scikit-learn | 0.22.1 |
| scipy | 1.5.2 |
| sklearn | 0.0 |
| tensorboard | 2.1.1 |
| tensorboardX | 2.0 |
| tensorflow | 2.1.0 |
| tensorflow-estimator | 2.1.0 |
| wheel | 0.37.1 |
| zipp | 3.8.0 |
| ... | |

- 部分apt安装包列表：

```
apt
ca-certificates
```

```
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二： **tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64**

- 镜像地址：swr.{region_id}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
- 镜像构建时间：20220524162601(yyyy-mm-dd-hh-mm-ss)

- 镜像系统版本：Ubuntu 20.04.4 LTS
- cuda：11.4.3
- cudnn：8.2.4.15
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python，python 3.8.13
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages

- 部分pip安装包列表：

```
Cython 0.29.21
psutil 5.9.0
matplotlib 3.5.1
protobuf 3.20.1
tensorflow 1.15.5+nv
Flask 2.0.1
grpcio 1.46.1
gunicorn 20.1.0
Pillow 9.0.1
tensorboard 1.15.0
PyYAML 6.0
pip 22.0.4
lxml 4.7.1
numpy 1.18.5
tensorflow-estimator 1.15.1
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
```

```
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本三： tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址： swr.{region_id}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- 镜像构建时间： 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本： Ubuntu 18.04.4 LTS
- cuda： 11.2.0
- cudnn： 8.1.1.33
- Python解释器路径及版本： /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/bin/python， python 3.7.10
- 三方包安装路径： /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/lib/python3.7/site-packages
- 部分pip安装包列表：

```
Cython 0.29.21
requests 2.27.1
easydict 1.9
tensorboardX 2.0
tensorflow 2.6.0
Flask 2.0.1
grpcio 1.46.1
gunicorn 20.1.0
idna 3.3
tensorflow-estimator 2.9.0
pandas 1.1.5
Pillow 9.0.1
lxml 4.8.0
matplotlib 3.5.1
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
numpy 1.17.0
opencv-python 4.1.2.30
protobuf 3.20.1
pip 21.2.2
...
```

- 部分apt安装包列表:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

2.4.3 推理基础镜像详情 Pytorch (CPU/GPU)

ModelArts提供了以下Pytorch (CPU/GPU) 推理基础镜像:

- [引擎版本一: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64](#)
- [引擎版本二: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64](#)

引擎版本一: [pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64](#)

- 镜像地址: `swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817`
- 镜像构建时间: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本: Ubuntu 18.04.4 LTS
- cuda: 10.2.89
- cudnn: 7.6.5.32
- Python解释器路径及版本: `/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python`, python 3.7.10
- 三方包安装路径: `/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages`

- 部分pip安装包列表:

| | |
|-------------------|----------------|
| Cython | 0.27.3 |
| easydict | 1.9 |
| Flask | 2.0.1 |
| fonttools | 4.34.4 |
| gunicorn | 20.1.0 |
| ipykernel | 6.7.0 |
| Jinja2 | 3.0.1 |
| lxml | 4.9.1 |
| matplotlib | 3.5.1 |
| mmcv | 1.2.7 |
| moxing-framework | 2.1.0.5d9c87c8 |
| numpy | 1.19.5 |
| opencv-python | 4.1.2.30 |
| pandas | 1.1.5 |
| Pillow | 9.2.0 |
| pip | 22.1.2 |
| protobuf | 3.20.1 |
| psutil | 5.8.0 |
| PyYAML | 5.1 |
| requests | 2.27.1 |
| scikit-learn | 0.22.1 |
| scipy | 1.5.2 |
| sklearn | 0.0 |
| tensorboard | 2.1.1 |
| tensorboardX | 2.0 |
| torch | 1.8.0 |
| torchtext | 0.5.0 |
| torchvision | 0.9.0 |
| tornado | 6.2 |
| tqdm | 4.64.0 |
| traitlets | 5.3.0 |
| typing_extensions | 4.3.0 |
| urllib3 | 1.26.10 |
| watchdog | 2.0.0 |
| wcwidth | 0.2.5 |
| Werkzeug | 2.1.2 |
| wheel | 0.37.1 |
| yapf | 0.32.0 |
| zipp | 3.8.0 |
| ... | |

- 部分apt安装包列表:

```
apt
ca-certificates
```

```
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二：pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- 镜像构建时间：20220524162601(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS

- cuda: 11.1.1
- cudnn: 8.0.5.39
- Python解释器路径及版本: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python, python 3.7.10
- 三方包安装路径: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages

- 部分pip安装包列表:

```
Cython 0.27.3  
mmcv 1.2.7  
easydict 1.9  
tensorboardX 2.0  
torch 1.8.2+cu111  
Flask 2.0.1  
pandas 1.1.5  
gunicorn 20.1.0  
PyYAML 5.1  
torchaudio 0.8.2  
Pillow 9.0.1  
psutil 5.8.0  
lxml 4.8.0  
matplotlib 3.5.1  
torchvision 0.9.2+cu111  
pip 21.2.2  
protobuf 3.20.1  
numpy 1.17.0  
opencv-python 4.1.2.30  
scikit-learn 0.22.1  
...
```

- 部分apt安装包列表:

```
apt  
ca-certificates  
cmake  
cuda  
curl  
ethtool  
fdisk  
ffmpeg  
g++  
gcc  
git  
gpg  
graphviz  
libsm6  
libxext6  
libopencv-dev  
libxrender-dev  
libatlas3-base  
libnuma-dev  
libcap-dev  
libssl-dev  
liblz-dev  
libbz2-dev  
liblzma-dev  
libboost-graph-dev  
libsndfile1  
libcurl4-openssl-dev  
libopenblas-base  
liblapack3  
libopenblas-dev  
libprotobuf-dev  
libleveldb-dev  
libsnap-dev  
libhdf5-serial-dev  
liblapack-dev  
libgflags-dev
```

```
libgoogle-glog-dev
liblmbd-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

2.4.4 推理基础镜像详情 MindSpore (CPU/GPU)

ModelArts提供了以下MindSpore (CPU/GPU) 推理基础镜像：

- [引擎版本一：mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64](#)
- [引擎版本二：mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64](#)
- [引擎版本三：mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64](#)

引擎版本一：mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- 部分pip安装包列表：

```
cycler 0.11.0
easydict 1.9
Flask 2.0.1
grpcio 1.47.0
gunicorn 20.1.0
ipykernel 6.7.0
Jinja2 3.0.1
lxml 4.9.0
matplotlib 3.5.1
mindinsight 1.7.0
mindspore 1.7.0
mindvision 0.1.0
moxing-framework 2.1.0.5d9c87c8
numpy 1.17.0
```

```
opencv-contrib-python-headless 4.6.0.66
opencv-python-headless 4.6.0.66
pandas 1.1.5
Pillow 9.1.1
pip 22.1.2
protobuf 3.20.1
psutil 5.8.0
PyYAML 5.1
requests 2.27.1
scikit-learn 0.22.1
scipy 1.5.2
setuptools 62.6.0
sklearn 0.0
tensorboardX 2.0
threadpoolctl 3.1.0
tomli 2.0.1
tornado 6.1
tqdm 4.64.0
traitlets 5.3.0
treelib 1.6.1
urllib3 1.26.9
wheel 0.37.1
zipp 3.8.0
...
```

- 部分apt安装包列表:

```
apt
ca-certificates
cmake
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapack-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
```



```
openside-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodo
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二：mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages

- 部分pip安装包列表：

```
cycler 0.11.0
easydict 1.9
Flask 2.0.1
grpcio 1.47.0
gunicorn 20.1.0
ipykernel 6.7.0
Jinja2 3.0.1
lxml 4.9.0
matplotlib 3.5.1
mindinsight 1.7.0
mindspore 1.7.0
mindvision 0.1.0
moxing-framework 2.1.0.5d9c87c8
numpy 1.17.0
opencv-contrib-python-headless 4.6.0.66
opencv-python-headless 4.6.0.66
pandas 1.1.5
Pillow 9.1.1
pip 22.1.2
protobuf 3.20.1
psutil 5.8.0
PyYAML 5.1
requests 2.27.1
scikit-learn 0.22.1
scipy 1.5.2
setuptools 62.6.0
sklearn 0.0
tensorboardX 2.0
threadpoolctl 3.1.0
tomli 2.0.1
```

```
tornado          6.1
tqdm              4.64.0
traitlets        5.3.0
treelib          1.6.1
urllib3          1.26.9
wheel            0.37.1
zipp              3.8.0
...
```

- 部分apt安装包列表:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
```

```
zlib1g-dev  
...
```

引擎版本三：mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：11.1.1
- cudnn：8.0.5.39
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python，python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages

- 部分pip安装包列表：

```
cycler 0.11.0  
easydict 1.9  
Flask 2.0.1  
grpcio 1.47.0  
gunicorn 20.1.0  
ipykernel 6.7.0  
Jinja2 3.0.1  
lxml 4.9.0  
matplotlib 3.5.1  
mindinsight 1.7.0  
mindspore 1.7.0  
mindvision 0.1.0  
moxing-framework 2.1.0.5d9c87c8  
numpy 1.17.0  
opencv-contrib-python-headless 4.6.0.66  
opencv-python-headless 4.6.0.66  
pandas 1.1.5  
Pillow 9.1.1  
pip 22.1.2  
protobuf 3.20.1  
psutil 5.8.0  
PyYAML 5.1  
requests 2.27.1  
scikit-learn 0.22.1  
scipy 1.5.2  
setuptools 62.6.0  
sklearn 0.0  
tensorboardX 2.0  
threadpoolctl 3.1.0  
tomli 2.0.1  
tornado 6.1  
tqdm 4.64.0  
traitlets 5.3.0  
treelib 1.6.1  
urllib3 1.26.9  
wheel 0.37.1  
zipp 3.8.0  
...
```

- 部分apt安装包列表：

```
apt  
ca-certificates  
cmake  
cuda  
curl
```

```
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnapappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

3 Notebook 中使用自定义镜像

3.1 Notebook 自定义镜像约束

制作自定义镜像时，Base镜像需满足如下规范：

- 可以基于开发环境提供的[预置镜像](#)为Base镜像制作自定义镜像。
- 基于昇腾、Dockerhub官网等官方开源的镜像制作，开源镜像需要满足如下操作系统约束：

x86: Ubuntu18.04、Ubuntu20.04

ARM: Euler2.8.3、Euler2.10.7

不满足以上镜像规范，所制作的镜像使用可能会出现故障，请用户检查镜像规范，并参考[Notebook自定义镜像故障基础排查](#)自行排查，如未解决请联系华为技术工程师协助解决。

3.2 在 ModelArts 中进行镜像注册

用户的自定义镜像构建完成后，需要在ModelArts“镜像管理”页面注册后，方可在Notebook中使用。

说明

SWR镜像类型设置为“私有”时，同一账号下的子用户（IAM用户）可以注册使用。

SWR镜像类型设置为“公开”时，其它用户才可以注册使用。

镜像注册


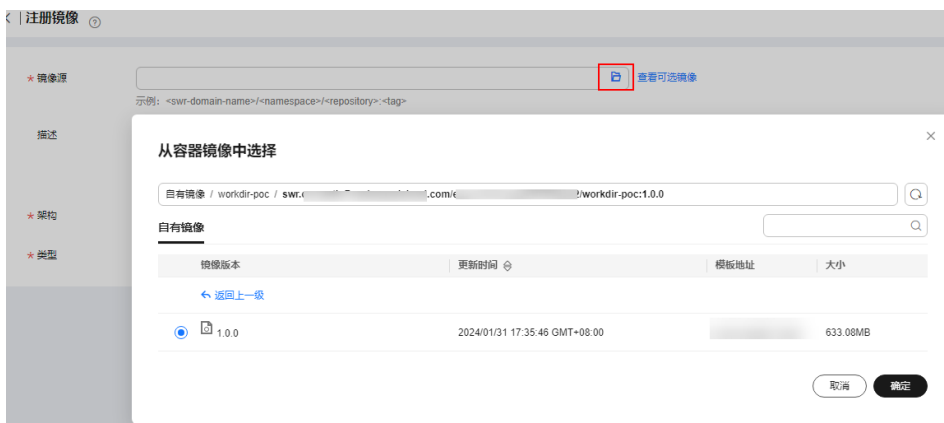
1. 进入ModelArts控制台，单击“镜像管理 > 注册镜像”，进入“注册镜像”页面。
2. 根据界面提示填写相关信息，然后单击“立即注册”。
 - “镜像源”选择构建好的镜像。可直接拷贝完整的SWR地址，或单击  选择SWR构建好的镜像进行注册。

图 3-1 选择镜像源



- “架构”和“类型”：根据自定义镜像的实际框架选择。
3. 注册后的镜像会显示在镜像管理页面。

镜像查找

在镜像列表页面，可根据“名称”、“所属组织”、“版本”等属性进行筛选，快速查找镜像。

图 3-2 查找镜像



创建 Notebook

单击镜像名称，进入镜像详情页面，单击“创建Notebook”，会跳转到基于该自定义镜像创建Notebook的页面。

图 3-3 进入镜像详情页面



镜像同步

当用户完成镜像故障排除后，进入镜像详情页，单击操作列的“镜像同步”完成镜像状态的刷新。

图 3-4 镜像同步



3.3 Notebook 制作自定义镜像方法

制作自定义镜像有以下方式：

- 方式一：使用Notebook的预置镜像创建开发环境实例，在环境中进行依赖安装与配置，配置完成后，可以通过开发环境提供的镜像保存功能，将运行实例的内容以容器镜像的方式保存下来，作为自定义镜像使用。详细操作请参考[将Notebook实例保存为自定义镜像](#)。
- 方式二：基于ModelArts提供的基础镜像以及镜像构建模板来编写Dockerfile，在Notebook中构建出完全适合自己的镜像。然后将镜像进行注册，用以创建新的开发环境，满足自己的业务需求。详细操作请参考[在Notebook中构建自定义镜像并使用](#)。
- 方式三：基于ModelArts提供的基础镜像或第三方镜像，在ECS服务器上自行编写Dockerfile构建镜像，对ModelArts基础镜像或第三方镜像进行改造，构建出符合[ModelArts要求](#)的新的自定义Docker镜像，并将镜像推送到SWR，作为自定义镜像使用。详细操作请参考[在ECS上构建自定义镜像并在Notebook中使用](#)。

3.4 将 Notebook 实例保存为自定义镜像

3.4.1 保存 Notebook 镜像环境

通过预置的镜像创建Notebook实例，在基础镜像上安装对应的自定义软件和依赖，在管理页面上进行操作，进而完成将运行的实例环境以容器镜像的方式保存下来。

保存的镜像中，安装的依赖包不丢失，持久化存储的部分（home/ma-user/work目录的内容）不会保存在最终产生的容器镜像中。VS Code远程开发场景下，在Server端安装的插件不丢失。

📖 说明

Notebook中保存的镜像大小不超过35G，镜像层数不能超过125层。否则镜像会保存失败。

若镜像保存时报错“The container size (xx) is greater than the threshold (25G)”，请参考[镜像保存时报错“The container size \(xG\) is greater than the threshold \(25G\)”如何解决？](#)处理。

前提条件

Notebook实例状态为“运行中”。

保存镜像

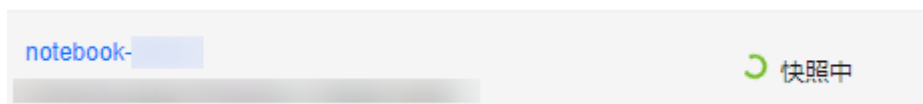
1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发环境 > Notebook”，进入新版Notebook管理页面。
2. 在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列中的“更多 > 保存镜像”，进入“保存镜像”对话框。

图 3-5 保存镜像



3. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确定”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“立即创建”，创建一个组织。创建组织的详细操作请参见[创建组织](#)。
同一个组织内的用户可以共享使用该组织内的所有镜像。
4. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例。

图 3-6 保存镜像



须知

快照中耗费的时间仍占用实例的总运行时长，若在快照中时，实例因运行时间到期停止，将导致镜像保存失败。

5. 镜像保存成功后，实例状态变为“运行中”，用户可在“镜像管理”页面查看到该镜像详情。
6. 单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

常见问题

- [镜像保存时报错“there are processes in 'D' status, please check process status using 'ps -aux' and kill all the 'D' status processes”如何解决?](#)

- 镜像保存时报错 “container size %dG is greater than threshold %dG” 如何解决？
- 保存镜像时报错 “too many layers in your image” 如何解决？
- 镜像保存时报错 “The container size (xG) is greater than the thresholds (25G)” 如何解决？

3.4.2 基于自定义镜像创建 Notebook 实例

从Notebook中保存的镜像可以在镜像管理中查询到，可以用于创建新的Notebook实例，完全继承保存状态下的实例软件环境配置。

基于自定义镜像创建Notebook实例有两种方式：

方式一：在Notebook实例创建页面，镜像类型选择“自定义镜像”，名称选择上述保存的镜像。

图 3-7 创建基于自定义镜像的 Notebook 实例



方式二：在“镜像管理”页面，单击某个镜像的镜像详情，在镜像详情页，单击“创建Notebook”，也会跳转到基于该自定义镜像创建Notebook的页面。

3.5 在 Notebook 中构建自定义镜像并使用

3.5.1 使用场景和构建流程说明

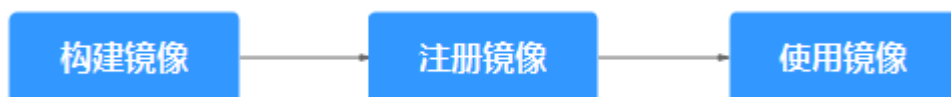
在预置镜像不满足客户使用诉求时，可以基于预置镜像自行构建容器镜像用于开发和训练。

用户在使用ModelArts开发环境时，经常需要对开发环境进行一些改造，如安装、升级或卸载一些包。但是某些包的安装升级需要root权限，运行中的Notebook实例中无root权限，所以在Notebook实例中安装需要root权限的软件，目前在预置的开发环境镜像中是无法实现的。

此时，用户可以使用ModelArts提供的基础镜像来编写Dockerfile，构建出完全适合自己的镜像。进一步可以调试该镜像，确保改造后的镜像能够在ModelArts服务中正常使用。最终将镜像进行注册，用以创建新的开发环境，满足自己的业务需求。

本案例将基于ModelArts提供的MindSpore基础镜像，并借助ModelArts命令行工具(请参考[ma-cli镜像构建命令介绍](#))制作和注册镜像，构建一个面向AI开发的自定义镜像。主要流程如下图所示：

图 3-8 构建镜像流程



3.5.2 Step1 制作自定义镜像

本节描述通过加载镜像构建模板并编写Dockerfile，构建出一个新镜像。如下的步骤都需要在“开发环境 > Notebook”的Terminal中完成，请提前创建好开发环境并打开Terminal。关于Dockerfile的具体编写方法，请参考[官网](#)。

步骤1 首先配置鉴权信息，指定profile，根据提示输入账号、用户名及密码。鉴权更多信息请查看[配置登录信息](#)。

```
ma-cli configure --auth PWD -P xxx
```

```
(MindSpore) [ma-user work]$ma-cli configure --auth PWD -P yuan
account []: hws
username []:
password:
```

步骤2 执行`env|grep -i CURRENT_IMAGE_NAME`命令查询当前实例所使用的镜像。

```
(MindSpore) [ma-user work]$env|grep -i CURRENT_IMAGE_NAME
CURRENT_IMAGE_NAME=swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906
```

步骤3 制作新镜像。

1. 获取上步查询的基础镜像的SWR地址。

```
CURRENT_IMAGE_NAME=swr.cn-north-4.myhuaweicloud.com/atelier/
mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-
d910-20220906
```

2. 加载镜像构建模板。

执行`ma-cli image get-template`命令查询镜像模板。

```
(MindSpore) [ma-user work]$ma-cli image get-template
Template Name      Description
-----
upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2  Upgrade Ascend MindSpore to 1.8.1 and CANN version to 5.1.RC2
```

然后执行`ma-cli image add-template`命令将镜像模板加载到指定文件夹下，默认路径为当前命令所在的路径。例如：加载`upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2`镜像构建模板。

```
ma-cli image add-template upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2
```

```
(MindSpore) [ma-user work]$ma-cli image add-template upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2 --force
[ OK ] Successfully add configuration template [ upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2 ] under folder [ /home/ma-user/work/.ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2 ]
```

3. 修改Dockerfile。

本例的Dockerfile将基于MindSpore基础镜像mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3，升级到cann 5.1.RC2和MindSpore1.8.1，构建一个面向AI任务的镜像。

加载镜像模板后，Dockerfile文件自动加载，在“`.ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2`”路径下，内容参考如下，根据实际需求修改：

```
#The following uses Mindspore-1.7 as an example, which can be replaced with the current notebook
image. ( 请替换成当前Notebook使用的镜像 )
FROM swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-
py_3.7-euler_2.8.3-aarch64-d910-20220715093657-9446c6a
ARG CANN=Ascend-cann-toolkit_5.1.RC2_linux-aarch64.run

# Set proxy to download internet resources ( 根据实际修改Notebook代理 )
ENV HTTP_PROXY=http://proxy.modelarts.com:80 \
```

```

http_proxy=http://proxy.modelarts.com:80 \
HTTPS_PROXY=http://proxy.modelarts.com:80 \
https_proxy=http://proxy.modelarts.com:80

USER root

# Download CANN-5.1.RC2 and install CANN package, which is a dependency package for
mindspore-1.8.1.
# For details about the mapping between Mindpore and CANN and the download address of CANN,
see the official website of Mindpore.
RUN wget https://ascend-repo.obs.cn-east-2.myhuaweicloud.com/CANN/CANN%205.1.RC2/${CANN} -
P /tmp && \
  chmod +x /tmp/${CANN} && \
  sh -x /tmp/${CANN} --quiet --full && \
  rm -f /tmp/${CANN}

ENV PYTHONPATH=/usr/local/Ascend/tfplugin/latest/python/site-packages:/usr/local/Ascend/ascend-
toolkit/latest/python/site-packages:/usr/local/Ascend/ascend-toolkit/latest/opp/op_impl/built-in/
ai_core/tbe:/usr/local/seccomponent/lib

USER ma-user

# Update mindspore version in "MindSpore" conda env by using pip.
RUN source /home/ma-user/anaconda3/bin/activate MindSpore && \
  pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.8.1/MindSpore/ascend/aarch64/
mindspore_ascend-1.8.1-cp37-cp37m-linux_aarch64.whl --upgrade && \
  echo "successfully install mindspore 1.8.1"

## [Optional] Uncomment to set default conda env
#ENV DEFAULT_CONDA_ENV_NAME=/home/ma-user/anaconda3/envs/MindSpore

```

📖 说明

如果使用的基础镜像不是ModelArts提供的公共镜像，需要在Dockerfile文件中添加ModelArts指定的用户和用户组，具体可参考[Dockerfile文件（基础镜像为非ModelArts提供）](#)。

4. 构建镜像

使用**ma-cli image build**命令从Dockerfile构建出一个新镜像。命令更多信息请参考[镜像构建命令](#)。

```
ma-cli image build .ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2/Dockerfile -swr notebook-test/my_image:0.0.1 -P XXX
```

其中“.ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2/Dockerfile”为Dockerfile文件所在路径，“notebook-test/my_image:0.0.1”为构建的新镜像的SWR路径。“XXX”为鉴权时指定的profile。

```

(MindSpore) [ma-user work]ma-cli image build .ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2/Dockerfile -swr notebook-test/my_image:0.0.1 -P yuan
[*] Building 1121.2s (8/8) FINISHED
-> [internal] load .dockerignore
-> -> transferring context: 2B
-> [internal] load build definition from Dockerfile
-> -> transferring dockerfile: 1.71kB
-> [internal] load metadata for swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1.7.0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906
-> [1/3] FROM swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1.7.0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906@sha256:c1ee0855
-> -> resolve swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1.7.0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906@sha256:c1ee0855

```

----结束

3.5.3 Step2 注册新镜像

调试完成后，将新镜像注册到ModelArts镜像管理服务中，进而能够在ModelArts中使用该镜像。

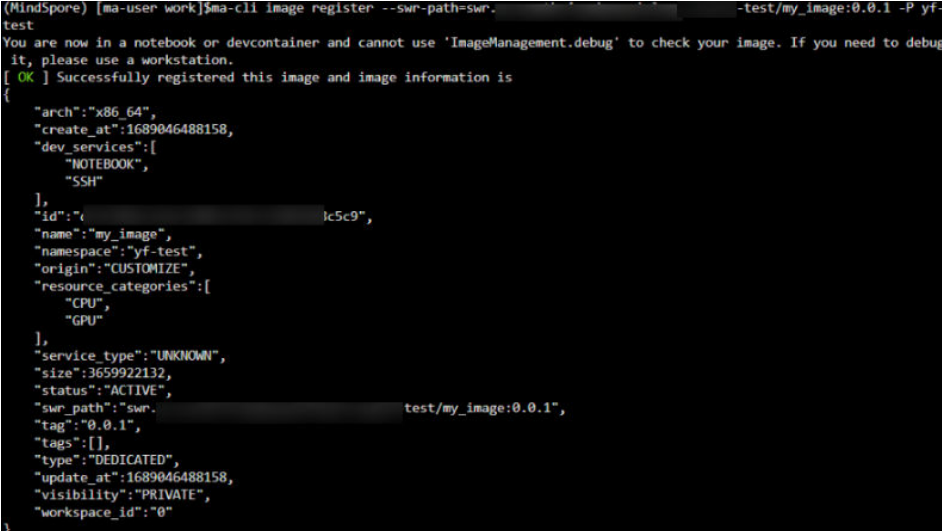
有两种方式来注册镜像。

- **方式一：使用ma-cli image register命令来注册镜像。**注册命令会返回注册好的镜像信息，包括镜像id，name等，如下图所示。该命令的更多信息可参考[注册镜像](#)。

```
ma-cli image register --swr-path=swr.cn-north-4.myhuaweicloud.com/notebook-test/my_image:0.0.1 -  
a AARCH64 -rs ASCEND -P XXX
```

-a指定该镜像支持ARM架构，-rs指定镜像支持ASCEND芯片，“XXX”为鉴权时指定的profile。


图 3-9 注册镜像



```
(MindSpore) [ma-user work]$ma-cli image register --swr-path=swr.cn-north-4.myhuaweicloud.com/notebook-test/my_image:0.0.1 -P yf-  
test  
You are now in a notebook or devcontainer and cannot use 'ImageManagement.debug' to check your image. If you need to debug  
it, please use a workstation.  
[ OK ] Successfully registered this image and image information is  
{  
  "arch": "x86_64",  
  "create_at": 1689046488158,  
  "dev_services": [  
    "NOTEBOOK",  
    "SSH"  
  ],  
  "id": "1c5c9",  
  "name": "my_image",  
  "namespace": "yf-test",  
  "origin": "CUSTOMIZE",  
  "resource_categories": [  
    "CPU",  
    "GPU"  
  ],  
  "service_type": "UNKNOWN",  
  "size": 3659922132,  
  "status": "ACTIVE",  
  "swr_path": "swr.cn-north-4.myhuaweicloud.com/notebook-test/my_image:0.0.1",  
  "tag": "0.0.1",  
  "tags": [],  
  "type": "DEDICATED",  
  "update_at": 1689046488158,  
  "visibility": "PRIVATE",  
  "workspace_id": "0"  
}
```

- **方式二：在ModelArts Console上注册镜像**

登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。

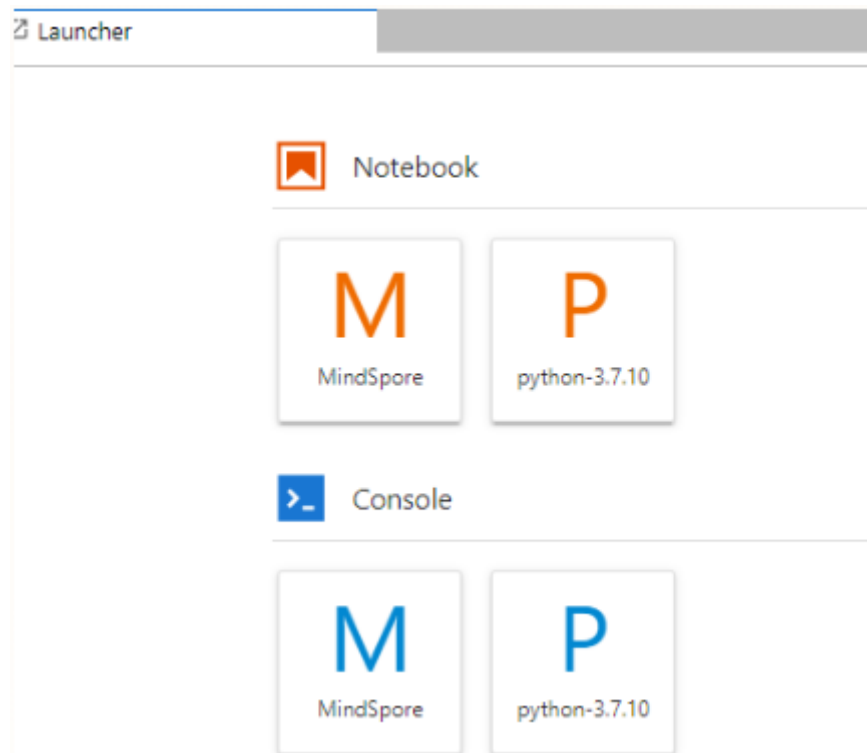
- a. 单击“注册镜像”。请将完整的SWR地址复制到这里即可，或单击  可直接从SWR选择自有镜像进行注册。
- b. “架构”和“类型”根据实际情况选择，与镜像源保持一致。

3.5.4 Step3 创建开发环境并使用

创建开发环境

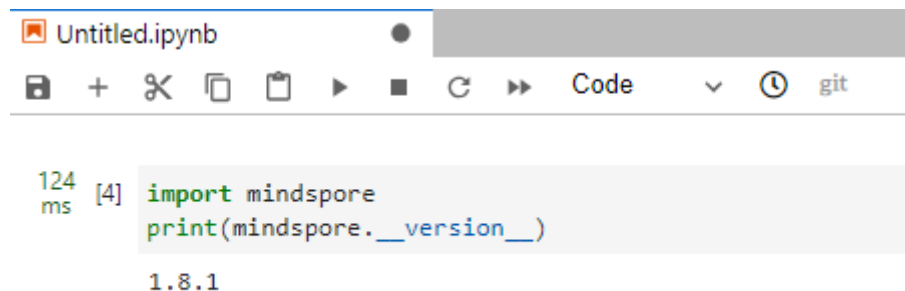
1. 镜像注册成功后，即可在ModelArts控制台的“开发环境 > Notebook”页面，创建开发环境时选择该自定义镜像。
2. 打开开发环境。

图 3-10 打开开发环境



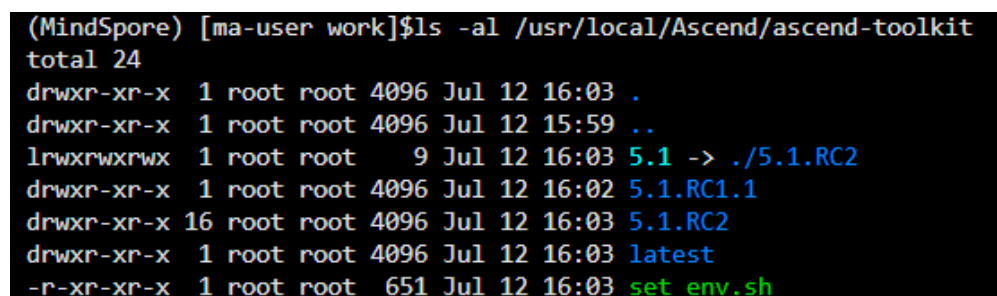
3. 单击图中的MindSpore，即可创建一个ipynb文件，导入mindspore，可以看到安装的mindspore 1.8.1已经能够使用。

图 3-11 创建一个 ipynb 文件



4. 再打开一个terminal，查看cann的版本，是Dockerfile中安装的版本。

图 3-12 查看 cann 版本



3.6 在 ECS 上构建自定义镜像并在 Notebook 中使用

3.6.1 使用场景和构建流程说明

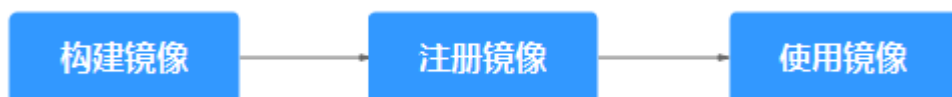
用户在使用ModelArts开发环境时，经常需要对开发环境进行一些改造，如安装、升级或卸载一些包。但是某些包的安装升级需要root权限，运行中的Notebook实例中无root权限，所以在Notebook实例中安装需要root权限的软件，这一点在预置的开发环境镜像中是无法实现的。

此时，用户可以使用ModelArts提供的基础镜像或第三方的镜像来编写Dockerfile，构建出完全适合自己的镜像。然后将镜像进行注册，用以创建新的开发环境，满足自己的业务需求。

本案例将基于ModelArts提供的PyTorch基础镜像，安装pytorch 1.8, ffmpeg 3和gcc 8，构建一个面向AI开发的新环境。

主要流程如下图所示：

图 3-13 构建与调测镜像流程



📖 说明

本案例适用于华为云-北京四Region。

3.6.2 Step1 准备 Docker 机器并配置环境信息

准备一台具有Docker功能的机器，如果没有，建议申请一台弹性云服务器并购买弹性公网IP，并在准备好的机器上安装必要的软件。

ModelArts提供了ubuntu系统的脚本，方便安装docker。

📖 说明

本地Linux机器的操作等同ECS服务器上的操作，请参考本案例。

创建 ECS 服务器

- 登录ECS控制台，购买弹性云服务器，镜像选择公共镜像，推荐使用ubuntu18.04的镜像；系统盘设置为100GiB。具体操作请参考[购买并登录弹性云服务器](#)。

图 3-14 选择镜像和磁盘



- 购买弹性公网IP并绑定到弹性云服务器。具体操作请参考[配置网络](#)。

配置 VM 环境

1. 在docker机器中，使用如下命令下载安装脚本。

```
wget https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/modelarts/custom-image-build/install_on_ubuntu1804.sh
```

📖 说明

当前仅支持ubuntu系统的脚本。

2. 在docker机器中并执行如下命令，即可完成环境配置。

```
bash install_on_ubuntu1804.sh
```

图 3-15 配置成功

```
Congratulations! The environment has been configured successfully.
```

```
source /etc/profile
```

安装脚本依次执行了如下任务：

- a. 安装docker。
- b. 如果挂载了GPU，则会安装nvidia-docker2，用以将GPU挂载到docker容器中。

3.6.3 Step2 制作自定义镜像

这一节描述如何编写一个Dockerfile，并据此构建出一个新镜像在Notebook创建实例并使用。关于Dockerfile的具体编写方法，请参考[官网](#)。

前提条件

已参考[Step1 准备Docker机器并配置环境信息](#)完成docker机器准备。

查询基础镜像（第三方镜像可跳过此步骤）

ModelArts提供的公共镜像，请参考[Notebook基础镜像列表](#)，根据预置镜像的引擎类型在对应的章节查看镜像URL。

制作新镜像

1. 连接容器镜像服务。
 - a. 登录容器镜像服务控制台。
 - b. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

图 3-16 获取登录指令

登录指令



说明

- 此处生成的登录指令有效期为24小时，若需要长期有效的登录指令，请参见[获取长期有效登录指令](#)。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
 - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。
- c. 在安装容器引擎的机器中执行上一步复制的登录指令。

登录成功会显示“Login Succeeded”。
 2. 拉取基础镜像或第三方镜像（此处以基础镜像举例，第三方镜像直接替换镜像地址）。

拉取ModelArts提供的公共镜像（请参考[预置镜像](#)）。

```
docker pull swr.cn-north-4.myhuaweicloud.com/atelier/notebook2.0-pytorch-1.4-kernel-cp37:3.3.3-release-v1-20220114
```
 3. 编写Dockerfile。

vim一个Dockerfile，如果使用的基础镜像是ModelArts提供的公共镜像，Dockerfile的具体内容可参考[Dockerfile文件（基础镜像为ModelArts提供）](#)。

如果使用的基础镜像是第三方镜像（非ModelArts提供的公共镜像），Dockerfile文件中需要添加uid为1000的用户ma-user和gid为100的用户组ma-group，具体可参考[Dockerfile文件（基础镜像为非ModelArts提供）](#)。

本例的Dockerfile将基于PyTorch基础镜像安装pytorch 1.8, ffmpeg 3和gcc 8，构建一个面向AI任务的镜像。
 4. 构建镜像
使用docker build命令从Dockerfile构建出一个新镜像。命令参数解释如下：

- “-t” 指定了新的镜像地址，包括{局点信息}/{组织名称}/{镜像名称}:{版本名称}，请根据实际填写。建议使用完整的swr地址，因为后续的调试和注册需要使用。
- “-f” 指定了Dockerfile的文件名，根据实际填写。
- 最后的“.” 指定了构建的上下文是当前目录，根据实际填写。

```
docker build -t swr.cn-north-4.myhuaweicloud.com/sdk-test/pytorch_1_8:v1 -f Dockerfile .
```

图 3-17 构建成功

```
Successfully built 495b3e4ff658  
Successfully tagged swr.cn-north-4.myhuaweicloud.com/sdk-test/pytorch_1_8:v1
```

Dockerfile 文件（基础镜像为 ModelArts 提供）

vim一个Dockerfile文件。基础镜像为ModelArts提供的镜像时，Dockerfile文件的具体内容如下：

```
FROM swr.cn-north-4.myhuaweicloud.com/atelier/notebook2.0-pytorch-1.4-kernel-cp37:3.3.3-release-v1-20220114  
  
USER root  
# section1: config apt source  
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \  
    echo -e "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb http://  
repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://repo.huaweicloud.com/ubuntu/  
bionic universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates universe\ndeb http://  
repo.huaweicloud.com/ubuntu/ bionic multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates  
multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted universe multiverse  
\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main restricted\ndeb http://  
repo.huaweicloud.com/ubuntu bionic-security universe\ndeb http://repo.huaweicloud.com/ubuntu bionic-  
security multiverse" > /etc/apt/sources.list && \  
    apt-get update  
# section2: install ffmpeg and gcc  
RUN apt-get -y install ffmpeg && \  
    apt -y install gcc-8 g++-8 && \  
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8  
&& \  
    rm $HOME/.pip/pip.conf  
USER ma-user  
# section3: configure conda source and pip source  
RUN echo -e "channels:\n - defaults\nshow_channel_urls: true\ndefault_channels:\n - https://  
mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main\n - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r  
\n - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2\nncustom_channels:\n conda-forge: https://  
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud  
\n bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n menpo: https://  
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/  
cloud\n pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n simpleitk: https://  
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud" > $HOME/.condarc && \  
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple\n[install]\ntrusted-host = https://  
pypi.tuna.tsinghua.edu.cn" > $HOME/.pip/pip.conf  
# section4: create a conda environment(only support python=3.7) and install pytorch1.8  
RUN source /home/ma-user/anaconda3/bin/activate && \  
    conda create -y --name pytorch_1_8 python=3.7 && \  
    conda activate pytorch_1_8 && \  
    pip install torch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0 && \  
    conda deactivate
```

Dockerfile 文件（基础镜像为非 ModelArts 提供）

如果使用的镜像是第三方镜像，Dockerfile文件中需要添加uid为1000的用户ma-user和gid为100的用户组ma-group。如果基础镜像中uid 1000或者gid 100已经被其他用户和用户组占用，需要将其对应的用户和用户组删除。如下Dockerfile文件已添加指定的用户和用户组，您直接使用即可。

 说明

用户需要设置uid为1000的用户ma-user和gid为100的用户组ma-group，并使/home/ma-user目录权限为750。

vim一个Dockerfile文件，添加第三方镜像（即非ModelArts提供的官方镜像）为基础镜像，如以ubuntu18.04为例。Dockerfile文件的具体内容如下：

```
# Replace it with the actual image version.
FROM ubuntu:18.04
# Set the user ma-user whose UID is 1000 and the user group ma-group whose GID is 100
USER root
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
    fi && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
    fi && \
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user && \
# Grant the read, write, and execute permissions on the target directory to the user ma-user.
chmod -R 750 /home/ma-user

#Configure the APT source and install the ZIP and Wget tools (required for installing conda).
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb http://\
repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://repo.huaweicloud.com/ubuntu/\
bionic universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates universe\ndeb http://\
repo.huaweicloud.com/ubuntu/ bionic multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates\
multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted universe multiverse\
\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main restricted\ndeb http://\
repo.huaweicloud.com/ubuntu bionic-security universe\ndeb http://repo.huaweicloud.com/ubuntu bionic-\
security multivers e" > /etc/apt/sources.list && \
apt-get update && \
apt-get install -y zip wget

#Modifying the system Configuration of the image (required for creating the Conda environment)
RUN rm /bin/sh && ln -s /bin/bash /bin/sh

#Switch to user ma-user , download miniconda from the Tsinghua repository, and install miniconda in /
home/ma-user.
USER ma-user
RUN cd /home/ma-user/ && \
    wget --no-check-certificate https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-4.6.14-\
Linux-x86_64.sh && \
    bash Miniconda3-4.6.14-Linux-x86_64.sh -b -p /home/ma-user/anaconda3 && \
    rm -rf Miniconda3-4.6.14-Linux-x86_64.sh

#Configure the conda and pip sources
RUN mkdir -p /home/ma-user/.pip && \
    echo -e "channels:\n - defaults\nshow_channel_urls: true\ndefault_channels:\n - https://\
mirrors.tuna.tsinghua.edu.cn/anaconda/pkgsg/main\n - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgsg/r/\
\n - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgsg/msys2" > /home/ma-user/.condarc && \
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple\n[install]\ntrusted-host = https://\
pypi.tuna.tsinghua.edu.cn" > /home/ma-user/.pip/pip.conf

#Create the conda environment and install the Python third-party package. The ipykernel package is
mandatory for starting a kernel.
RUN source /home/ma-user/anaconda3/bin/activate && \
    conda create -y --name pytorch_1_8 python=3.7 && \
    conda activate pytorch_1_8 && \
    pip install torch==1.8.1 torchvision==0.9.1 && \
    pip install ipykernel==6.7.0 && \
    conda init bash && \
    conda deactivate

#Install FFmpeg and GCC
USER root
```

```
RUN apt-get -y install ffmpeg && \
apt -y install gcc-8 g++-8
```

3.6.4 Step3 注册新镜像

调试完成后，将新镜像注册到ModelArts镜像管理服务中，进而能够在ModelArts中使用该镜像。

1. 将镜像推到SWR

推送前需要登录SWR，请参考[登录SWR](#)。登录后使用docker push命令进行推送，如下：

```
docker push swr.cn-north-4.myhuaweicloud.com/sdk-test/pytorch_1_8:v1
```

完成后即可在SWR上看到该镜像。

图 3-18 将镜像推到 SWR



说明

SWR地址，包括{局点信息}/{组织名称}/{镜像名称}:{版本名称}，局点信息可以参考下图查看，组织名称在“容器镜像服务>组织管理”创建。

<https://console.huaweicloud.com/swr/?agencyid=...®ion=...#/swr/warehouse/list/private>

2. 注册镜像

在ModelArts Console上注册镜像

登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。

a. 单击“注册镜像”，镜像源即为1.将镜像推到SWR中推送到SWR中的镜像。

请将完整的SWR地址复制到这里即可，或单击可直接从SWR选择自有镜像进行注册。

b. “架构”和“类型”根据实际情况选择，与镜像源保持一致。

说明

注册镜像时，“架构”和“类型”需要和镜像源保持一致，否则在使用此自定义镜像创建Notebook时会创建失败。

3.6.5 Step4 创建开发环境并使用

创建开发环境

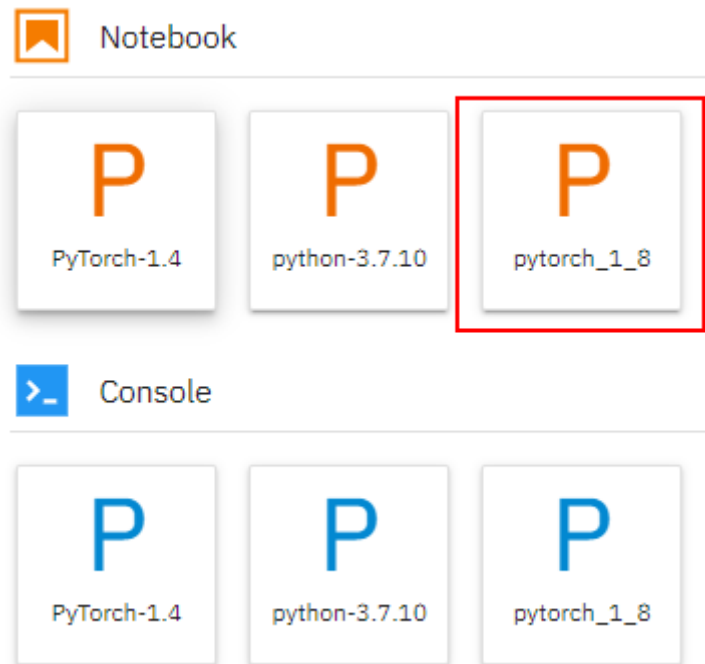
1. 镜像注册成功后，即可在ModelArts控制台的Notebook页面，创建开发环境时选择该自定义镜像。

图 3-19 创建开发环境



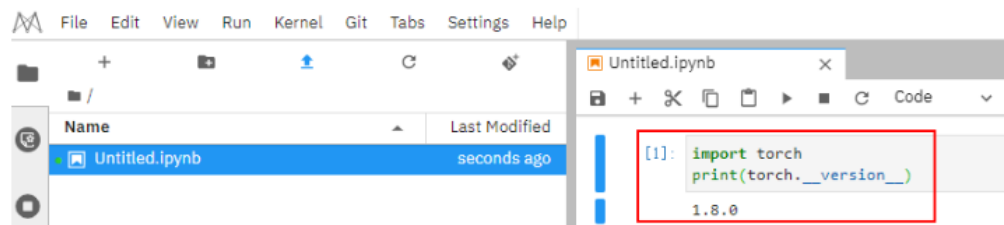
2. 打开开发环境，即可看到Dockerfile中创建的conda环境pytorch_1_8。

图 3-20 打开开发环境



3. 单击图中的pytorch_1_8，即可创建一个ipynb文件，导入torch，可以看到安装的pytorch 1.8已经能够使用。

图 3-21 创建一个 ipynb 文件



4. 再打开一个Terminal，查看ffmpeg和gcc的版本，是Dockerfile中安装的版本。

图 3-22 查看 ffmpeg 和 gcc 的版本

```
[ma-user work]$ffmpeg -v
ffmpeg version 3.4.8-0ubuntu0.2 Copyright (c) 2000-2020 the Ffmpeg developers
built with gcc 7 (Ubuntu 7.5.0-3ubuntu1-18.04)
configuration: --prefix=/usr --extra-version=0ubuntu0.2 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/
stripping --enable-avresample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libass --enable-libbluray --enable-libs2b --en
ibflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libmp3lame --enable-lib
openmpt --enable-libopus --enable-libpulse --enable-librubberband --enable-libsrt --enable-libssh --enable-libshine --enable-libsnap --enable-libsoxr
ble-libtheora --enable-libtwolame --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libx
nable-libzvi --enable-omx --enable-opengl --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enabl
ibopencv --enable-libx264 --enable-shared
libavutil      55. 78.100 / 55. 78.100
libavcodec     57.107.100 / 57.107.100
libavformat    57. 83.100 / 57. 83.100
libavdevice    57. 10.100 / 57. 10.100
libavfilter    6.107.100 / 6.107.100
libavresample  3.  7.  0 / 3.  7.  0
libswscale     4.  8.100 / 4.  8.100
libswresample  2.  9.100 / 2.  9.100
libpostproc   54.  7.100 / 54.  7.100
Missing argument for option 'y'
Error splitting the argument list: invalid argument
[ma-user work]$gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/8/lto-wrapper
OFFLOAD_TARGET_NAMES=avx512bw
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 8.4.0-1ubuntu1-18.04' --with-bugurl=file:///usr/share/doc/gcc-8/README.Bu
d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-8 --program-prefix=x86_64-linux-gnu- --enable-share
/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdc++-debug
lt-libstdc++-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-syst
enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch=32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-m
load-targets=avx512bw --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64
Thread model: posix
gcc version 8.4.0 (Ubuntu 8.4.0-1ubuntu1-18.04)
```

3.7 Notebook 自定义镜像故障基础排查

当制作的自定义镜像使用出现故障时，请用户按照如下方法排查：

- 用户自定义镜像没有ma-user用户及ma-group用户组；
- 用户自定义镜像中/home/ma-user目录，属主和用户组不是ma-user和ma-group；
- 用户自定义镜像必须满足用户目录/home/ma-user权限为750，不能为其他权限；
- 用户自定义镜像使用远程SSH功能，OpenSSH版本要兼容或高于8.0；
- 用户制作的自定义镜像，在本地执行docker run启动，无法正常运行；
- 用户自行安装了Jupyterlab服务导致冲突的，需要用户本地使用Jupyterlab命令罗列出相关的静态文件路径，删除并且卸载镜像中的Jupyterlab服务；
- 用户自己业务占用了开发环境官方的8888、8889端口的，需要用户修改自己的进程端口号；
- 用户的镜像指定了PYTHONPATH、sys.path导致服务启动调用冲突的，需在实例启动后，再指定PYTHONPATH、sys.path；
- 用户使用了已开启sudo权限的专属池，使用自定义镜像时，sudo工具未安装或安装错误；
- 用户使用的cann、cuda环境有兼容性问题；
- 用户的docker镜像配置错误、网络或防火墙限制、镜像构建问题（文件权限、依赖缺失或构建命令错误）等原因导致的。

4 使用自定义镜像训练模型（模型训练）

4.1 训练管理中使用自定义镜像介绍

订阅算法和预置框架涵盖了大部分的训练场景。针对特殊场景，ModelArts支持用户构建自定义镜像用于模型训练。

自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置框架无法满足需求，否则不推荐使用。自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练。

ModelArts上使用自定义镜像训练支持2种方式：

- 使用预置框架 + 自定义镜像：
如果先前基于预置框架且通过指定代码目录和启动文件的方式来创建的训练作业；但是随着业务逻辑的逐渐复杂，您期望可以基于预置框架修改或增加一些软件依赖的时候，此时您可以使用预置框架 + 自定义镜像的功能，即选择预置框架名称后，在预置框架版本下拉列表中选择“自定义”。
- 完全自定义镜像：
用户遵循ModelArts镜像的规范要求制作镜像，选择自己的镜像，并且通过指定代码目录（可选）和启动命令的方式来创建的训练作业。

📖 说明

当使用完全自定义镜像创建训练作业时，“启动命令”必须在“/home/ma-user”目录下执行，否则训练作业可能会运行异常。

使用预置框架 + 自定义镜像

此功能与直接基于预置框架创建训练作业的区别仅在于，镜像是由用户自行选择的。用户可以基于预置框架制作自定义镜像。基于预置框架制作自定义镜像可参考[使用基础镜像构建新的训练镜像](#)章节。

图 4-1 使用预置框架+自定义镜像创建算法

* 启动方式

1 预置框架 自定义 ?

2 3 自定义

* 镜像 选择

* 代码目录 选择

* 启动文件 选择

该功能的行为与直接基于预置框架创建的训练作业相同，例如：

- 系统将会自动注入一系列环境变量
 - `PATH=${MA_HOME}/anaconda/bin:${PATH}`
 - `LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}`
 - `PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}`
- 您选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。注意到系统自动注入的PATH环境变量，您可以参考下述命令确认训练作业最终使用的Python版本：
 - `export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V`
 - `docker run --rm {image} $(which python) -V`
- 系统将会自动添加预置框架关联的超参

完全使用自定义镜像

图 4-2 完全使用自定义镜像创建算法

* 启动方式

预置框架 自定义 ?

* 镜像 选择

代码目录 选择

* 启动命令 ?

1

训练支持的自定义镜像使用说明请参考[使用自定义镜像创建训练作业](#)。

完全使用自定义镜像场景下，指定的“conda env”启动训练方法如下：

由于训练作业运行时不是shell环境，因此无法直接使用“conda activate”命令激活指定的“conda env”，需要使用其他方式以达成使用指定“conda env”来启动训练的效果。

假设您的自定义镜像中的“conda”安装于“/home/ma-user/anaconda3”目录，“conda env”为“python-3.7.10”，训练脚本位于“/home/ma-user/modelarts/user-job-dir/code/train.py”。可通过以下方式使用指定的“conda env”启动训练：

- 方式一：为镜像设置正确的“DEFAULT_CONDA_ENV_NAME”环境变量与“ANACONDA_DIR”环境变量。

您可以使用Python命令启动训练脚本。启动命令示例如下：

```
python /home/ma-user/modelarts/user-job-dir/code/train.py
```

- 方式二：使用“conda env python”的绝对路径。

您可以使用“/home/ma-user/anaconda3/envs/python-3.7.10/bin/python”命令启动训练脚本。启动命令示例如下：

```
/home/ma-user/anaconda3/envs/python-3.7.10/bin/python /home/ma-user/modelarts/user-job-dir/code/train.py
```

- 方式三：设置PATH环境变量。

您可以将指定的“conda env bin”目录配置到PATH环境变量中。您可以使用Python命令启动训练脚本。启动命令示例如下：

```
export PATH=/home/ma-user/anaconda3/envs/python-3.7.10/bin:$PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
```

- 方式四：使用“conda run -n”命令。

您可以使用“/home/ma-user/anaconda3/bin/conda run -n python-3.7.10”命令来执行训练命令，启动命令示例如下：

```
/home/ma-user/anaconda3/bin/conda run -n python-3.7.10 python /home/ma-user/modelarts/user-job-dir/code/train.py
```

📖 说明

如果在训练时发生找不到“\$ANACONDA_DIR/envs/\$DEFAULT_CONDA_ENV_NAME/lib”目录下“.so”文件的相关报错，可以尝试将该目录加入到“LD_LIBRARY_PATH”，将以下命令放在上述启动方式命令前：

```
export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH;
```

例如，方式一的启动命令示例此时变为：

```
export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
```

4.2 示例：从 0 到 1 制作自定义镜像并用于训练

4.2.1 示例：从 0 到 1 制作自定义镜像并用于训练（PyTorch+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

📖 说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. [前提条件](#)
2. [Step1 创建OBS桶和文件夹](#)
3. [Step2 准备训练脚本并上传至OBS](#)
4. [Step3 准备镜像主机](#)
5. [Step4 制作自定义镜像](#)
6. [Step5 上传镜像至SWR服务](#)
7. [Step6 在ModelArts上创建训练作业](#)

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如[表4-1](#)所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-1 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|---|-------------|
| “obs://test-modelarts/pytorch/demo-code/” | 用于存储训练脚本文件。 |
| “obs://test-modelarts/pytorch/log/” | 用于存储训练日志文件。 |

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本“pytorch-verification.py”文件，并上传至OBS桶的“obs://test-modelarts/pytorch/demo-code/”文件夹下。

“pytorch-verification.py”文件内容如下：

```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用Ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以执行以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果docker images命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 执行如下命令确认Docker Engine版本。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
...
Engine:
Version:      18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，也可以查看“pip.conf”文件内容。

5. 下载“torch*.whl”文件。

在网站“https://download.pytorch.org/whl/torch_stable.html”搜索并下载如下whl文件。

- torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
- torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
- torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

📖 说明

“+”符号的URL编码为“%2B”，在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

8. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
```

```
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
  /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
  /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
  /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
  /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim和curl 工具 (依然使用华为开源镜像站)
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  apt-get update && \
  apt-get install -y vim curl && \
  apt-get clean && \
  mv /etc/apt/sources.list.bak /etc/apt/sources.list

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组, 因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
  PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见[Docker官方文档](#)。

9. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像
pytorch:1.8.1-cuda11.1。

```
docker build . -t pytorch:1.8.1-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:1.8.1-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。

- a. 使用docker tag命令给上传镜像打标签。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

#此处以华为云cn-north-4为例

```
sudo docker tag pytorch:1.8.1-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```
 - b. 使用docker push命令上传镜像。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

#此处以华为云cn-north-4为例

```
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```
6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考[使用委托授权](#)。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
 - 创建方式：选择“自定义算法”
 - 启动方式：选择“自定义”
 - 镜像地址：[Step5 上传镜像至SWR服务](#)中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“\${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 启动命令：“/home/ma-user/miniconda3/bin/python \${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 资源池：选择公共资源池
 - 类型：选择GPU或者CPU规格。
 - 永久保存日志：打开
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-3 GPU 规格运行日志信息

```
1 tensor([[ -0.4181,  0.8150, -0.2581],
2         [ -0.6062,  0.5347,  0.1890],
3         [  0.5751,  1.2730, -0.3907],
4         [  0.4812, -0.4064, -0.2753],
5         [  1.0377, -1.1248,  1.2977]])
6 tensor([[ -0.7440, -0.8577, -0.2340],
7         [  0.9569,  0.5516, -1.3350],
8         [-1.2878, -0.2791,  0.3486],
9         [-1.0997,  0.7627, -0.3188],
10        [-1.0865, -1.2626, -0.5900]], device='cuda:0')
11
```

图 4-4 CPU 规格运行日志信息

```
1 tensor([[ 0.8945, -0.6946,  0.3807],
2         [ 0.6665,  0.3133,  0.8285],
3         [-0.5353, -0.1730, -0.5419],
4         [ 0.4870,  0.5183,  0.2505],
5         [ 0.2679, -0.4996,  0.7919]])
6 tensor([[ 0.9692,  0.4652,  0.5659],
7         [ 2.2032,  1.4157, -0.1755],
8         [-0.6296,  0.5466,  0.6994],
9         [ 0.2353, -0.0089, -1.9546],
10        [ 0.9319,  1.1781, -0.4587]])
11
```

4.2.2 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

📖 说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. [前提条件](#)
2. [Step1 创建OBS桶和文件夹](#)
3. [Step2 准备脚本文件并上传至OBS中](#)
4. [Step3 准备镜像主机](#)
5. [Step4 制作自定义镜像](#)
6. [Step5 上传镜像至SWR服务](#)
7. [Step6 在ModelArts上创建训练作业](#)

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如[表4-2](#)所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-2 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|---------------------------------------|---------------------|
| “obs://test-modelarts/mpi/demo-code/” | 用于存储MPI启动脚本与训练脚本文件。 |
| “obs://test-modelarts/mpi/log/” | 用于存储训练日志文件。 |

Step2 准备脚本文件并上传至 OBS 中

准备本案例所需的MPI启动脚本run_mpi.sh文件和训练脚本mpi-verification.py文件，并上传至OBS桶的“obs://test-modelarts/mpi/demo-code/”文件夹下。

- MPI启动脚本run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user
```

```

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: ${MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=' > ${MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|${MY_SSHD_PORT}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/ssh_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/ssh_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<${MA_NUM_HOSTS}; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .*/1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null > /dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=${MY_MPI_SLOTS}" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-

```



```

p ${MY_SSHD_PORT}" -tune ${MY_MPI_TUNE_FILE} ... $@"

# execute mpirun at worker-0
# mpirun
mpirun \
  -np ${np} \
  -hostfile ${MY_HOME}/hostfile \
  -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
  -tune ${MY_MPI_TUNE_FILE} \
  -bind-to none -map-by slot \
  -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
  -x HOROVOD_MPI_THREADS_DISABLE=1 \
  -x PATH -x LD_LIBRARY_PATH \
  -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
  "$@"

RET_CODE=$?

if [ $RET_CODE -ne 0 ]; then
  echo "[run_mpi] exec command failed, exited with $RET_CODE"
else
  echo "[run_mpi] exec command successfully, exited with $RET_CODE"
fi

# stop 1...N worker by killing the sleep proc
sed -i '1d' ${MY_HOME}/hostfile
if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
  echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

  sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
  printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

  mpirun \
  --hostfile ${MY_HOME}/hostfile \
  --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
  -x PATH -x LD_LIBRARY_PATH \
  pkill sleep \
  > /dev/null 2>&1
fi

echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
  echo "[run_mpi] the training log is in worker-0"
  sleep 365d
  echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE

```

📖 说明

“run_mpi.sh”脚本需要以LF作为换行符。使用CRLF作为换行符会导致训练作业运行失败，日志中会打印“\$'\r': command not found”的错误信息。

- 训练脚本mpi-verification.py文件内容如下：

```

import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

# https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])

```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:
Version:      18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 下载Miniconda3安装文件。

使用地址 https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应 python 3.7.13）。

5. 下载openmpi 3.0.0安装文件。

使用地址<https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz>，下载 horovod v0.22.1已经编译好的openmpi 3.0.0文件。

6. 将上述Miniconda3安装文件、openmpi 3.0.0文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

7. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网
```

```
# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
```

```
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 复制 Miniconda3 (python 3.7.13) 安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim / curl / net-tools / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
  apt-get update && \
  apt-get install -y vim curl net-tools iputils-ping \
  openssh-client openssh-server && \
  ssh -V && \
  mkdir -p /run/sshd && \
  apt-get clean && \
  mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
  rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
  tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
  ldconfig && \
  mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
  PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd, 使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
  # setup sshd dir
  mkdir -p ${MA_HOME}/etc && \
  ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N "" -t rsa && \
  mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run && \
  # setup sshd config (listen at {{MY_SSHD_PORT}} port)
  echo "Port {{MY_SSHD_PORT}}\n\
  HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
  AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
```

```
PidFile ${MA_HOME}/var/run/sshd.pid\n
StrictModes no\n
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
# generate ssh key
ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P "" && \
cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
# disable ssh host key checking for all hosts
echo "Host *\n
StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

关于Dockerfile文件编写的更多指导内容参见 [Docker 官方文档](#)。

8. 确认已创建完成Dockerfile 文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像 mpi:3.0.0-cuda11.1。

```
docker build -t mpi:3.0.0-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
 - a. 使用docker tag命令给上传镜像打标签。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
 - b. 使用docker push命令上传镜像。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考[使用委托授权](#)。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在ModelArts管理控制台，左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
 - 创建方式：选择“自定义算法”
 - 启动方式：选择“自定义”

- 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/multi-mpi:3.0.0-cuda11.1”
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/multi-mpi/demo-code/”
 - 启动命令：bash \${MA_JOB_DIR}/demo-code/run_mpi.sh python \${MA_JOB_DIR}/demo-code/multi-mpi-verification.py
 - 环境变量：添加“MY_SSHD_PORT = 38888”
 - 资源池：选择公共资源池
 - 类型：选择GPU规格
 - 计算节点个数：选择“1”或“2”
 - 永久保存日志：打开
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/multi-mpi/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
 5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如图4-5所示。

图 4-5 1 个计算节点 GPU 规格 worker-0 运行日志信息

```

MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp      0      0 0.0.0.0:38888          0.0.0.0:*              LISTEN    60/sshd
tcp6     0      0 :::38888                :::*                    LISTEN    60/sshd
172.16.0.122 slots=1
modelarts-job-8cf8a682-21cb-4d73-9bb3-789cecdc458b-worker-0
OMPI_COMM_WORLD_SIZE: 1
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
    
```

计算节点个数选择为2，训练作业也可以运行。日志信息如图4-6和图4-7所示。

图 4-6 2 个计算节点 worker-0 运行日志信息

```

MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp      0      0 0.0.0.0:38888          0.0.0.0:*              LISTEN    61/sshd
tcp6     0      0 :::38888                :::*                    LISTEN    61/sshd
172.16.0.39 slots=1
172.16.0.123 slots=1
Warning: Permanently added '[172.16.0.123]:38888' (RSA) to the list of known hosts.
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-0
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-1
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 1
OMPI_COMM_WORLD_LOCAL_RANK: 0
    
```

图 4-7 2 个计算节点 worker-1 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 1
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888          0.0.0.0:*           LISTEN     62/sshd
tcp6       0      0 :::38888              :::*                 LISTEN     62/sshd
/home/ma-user/modelarts/user-job-dir/xxxxx/run_mpi.sh: line 109: 66 Terminated          sleep 365d
```

4.2.3 示例：从 0 到 1 制作自定义镜像并用于训练（Horovod-PyTorch+GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Horovod 0.22.1 + PyTorch 1.8.1，训练使用的资源是GPU。

📖 说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1
- horovod-0.22.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. [前提条件](#)
2. [Step1 创建OBS桶和文件夹](#)
3. [Step2 准备训练脚本并上传至OBS](#)
4. [Step3 准备镜像主机](#)
5. [Step4 制作自定义镜像](#)
6. [Step5 上传镜像至SWR服务](#)
7. [Step6 在ModelArts上创建训练作业](#)

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如表4-3所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-3 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|---|-------------|
| “obs://test-modelarts/pytorch/demo-code/” | 用于存储训练脚本文件。 |
| “obs://test-modelarts/pytorch/log/” | 用于存储训练日志文件。 |

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本pytorch_synthetic_benchmark.py和run_mpi.sh文件，并上传至OBS桶的“obs://test-modelarts/horovod/demo-code/”文件夹下。

pytorch_synthetic_benchmark.py文件内容如下：

```
import argparse
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data.distributed
from torchvision import models
import horovod.torch as hvd
import timeit
import numpy as np

# Benchmark settings
parser = argparse.ArgumentParser(description='PyTorch Synthetic Benchmark',
                                formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='resnet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

parser.add_argument('--use-adasum', action='store_true', default=False,
                    help='use adasum algorithm to do reduction')

args = parser.parse_args()
```

```

args.cuda = not args.no_cuda and torch.cuda.is_available()

hvd.init()

if args.cuda:
    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

cudnn.benchmark = True

# Set up standard model.
model = getattr(models, args.model)()

# By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
    # Move model to GPU.
    model.cuda()
    # If using GPU Adasum allreduce, scale learning rate by local_size.
    if args.use_adasum and hvd.nccl_built():
        lr_scaler = hvd.local_size()

optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)

# Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     compression=compression,
                                     op=hvd.Adasum if args.use_adasum else hvd.Average)

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Set up fixed fake data
data = torch.randn(args.batch_size, 3, 224, 224)
target = torch.LongTensor(args.batch_size).random_() % 1000
if args.cuda:
    data, target = data.cuda(), target.cuda()

def benchmark_step():
    optimizer.zero_grad()
    output = model(data)
    loss = F.cross_entropy(output, target)
    loss.backward()
    optimizer.step()

def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')

log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

# Warm-up
log('Running warmup...')
timeit.timeit(benchmark_step, number=args.num_warmup_batches)

# Benchmark

```



```
log('Running benchmark...')
img_secs = []
for x in range(args.num_iters):
    time = timeit.timeit(benchmark_step, number=args.num_batches_per_iter)
    img_sec = args.batch_size * args.num_batches_per_iter / time
    log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
    img_secs.append(img_sec)

# Results
img_sec_mean = np.mean(img_secs)
img_sec_conf = 1.96 * np.std(img_secs)
log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
log('Total img/sec on %d %s(s): %.1f +-%.1f' %
    (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))
```

run_mpi.sh文件内容如下:

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_[SHARED_]^[S3_]^[PATH|^VC_WORKER_]^[SCC|^CRED]' | grep -v '=' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<${MA_NUM_HOSTS}; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+) .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
        done
    done
fi
```

```

        fi
        sleep 1
    done

    echo "[run_mpi] the sshd of ip ${ip} is up"

    echo "${ip} slots=${MY_MPI_SLOTS}" >> ${MY_HOME}/hostfile
done
printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi
RET_CODE=0
if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... @$@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
            --hostfile ${MY_HOME}/hostfile \
            --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
            --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
            -x PATH -x LD_LIBRARY_PATH \
            pkill sleep \
            > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

```

```
exit $RET_CODE
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1
- horovod-0.22.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh  
sh get-docker.sh
```

如果docker images命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:  
Version:      18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]  
index-url = https://repo.huaweicloud.com/repository/pypi/simple  
trusted-host = repo.huaweicloud.com  
timeout = 120
```

说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载horovod源码文件。

进入网站<https://pypi.org/project/horovod/0.22.1/#files>，下载horovod-0.22.1.tar.gz文件。

6. 下载torch*.whl文件。

在网站https://download.pytorch.org/whl/torch_stable.html搜索并下载如下whl文件。

- torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
- torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
- torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

 说明

“+”符号的URL编码为“%2B”；在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

7. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

8. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder

# 安装 cmake 工具（使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y build-essential cmake g++-7 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp
COPY openmpi-3.0.0-bin.tar.gz /tmp
COPY horovod-0.22.1.tar.gz /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
RUN cd /usr/local && \
```

```

tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
ldconfig && \
mpirun --version

# 编译 Horovod with PyTorch 所需的环境变量
ENV HOROVOD_NCCL_INCLUDE=/usr/include \
HOROVOD_NCCL_LIB=/usr/lib/x86_64-linux-gnu \
HOROVOD_MPICXX_SHOW="/usr/local/openmpi/bin/mpicxx -show" \
HOROVOD_GPU_OPERATIONS=NCCL \
HOROVOD_WITH_PYTORCH=1

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
/home/ma-user/miniconda3/bin/pip install --no-cache-dir \
/tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
/tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
/tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 编译安装 horovod-0.22.1.tar.gz
RUN cd /tmp && \
/home/ma-user/miniconda3/bin/pip install --no-cache-dir \
/tmp/horovod-0.22.1.tar.gz

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed / ssh 工具 (依然使用华为开源镜像站)
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
apt-get update && \
apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
openssh-client openssh-server && \
ssh -V && \
mkdir -p /run/sshd && \
# mlnx ofed
apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1 libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-dev flex chrpath libltdl-dev && \
cd /tmp && \
tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --without-fw-update -q && \
cd - && \
rm -rf /tmp/* && \
apt-get clean && \
mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
ldconfig && \
mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组, 因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

```

```
# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd, 使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
  # setup sshd dir
  mkdir -p ${MA_HOME}/etc && \
  ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N "" -t rsa && \
  mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run && \
  # setup sshd config (listen at {{MY_SSHD_PORT}} port)
  echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
  # generate ssh key
  ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P "" && \
  cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
  # disable ssh host key checking for all hosts
  echo "Host *\n\
StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
  PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见[Docker 官方文档](#)。

9. 下载MLNX_OFED安装包。

进入[地址](#)，单击“Download”，在“Current Versions”和“Archive Versions”中选择合适的安装包下载。本文示例中选择“Archive Versions”，“Version”选择“5.4-3.5.8.0-LTS”，“OS Distribution”选择“Ubuntu”，“OS Distribution Version”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

10. 下载openmpi-3.0.0-bin.tar.gz。

使用地址<https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz>，下载openmpi-3.0.0-bin.tar.gz文件。

11. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── horovod-0.22.1.tar.gz
├── openmpi-3.0.0-bin.tar.gz
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

12. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1。

```
docker build -t horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
 - a. 使用docker tag命令给上传镜像打标签。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker tag horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1 swr:{region-id}:{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```


#此处以华为云cn-north-4为例

```
sudo docker tag horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```
 - b. 使用docker push命令上传镜像。
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker push swr:{region-id}:{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```


#此处以华为云cn-north-4为例

```
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考[使用委托授权](#)。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
 - 创建方式：选择“自定义算法”
 - 镜像来源：选择“自定义”
 - 镜像地址：[Step5 上传镜像至SWR服务](#)中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1”
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“\${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 启动命令：“bash \${MA_JOB_DIR}/demo-code/run_mpi.sh python \${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 环境变量：单击“增加环境变量”，增加环境变量：MY_SSHD_PORT=38888。
 - 资源池：选择公共资源池。
 - 资源类型：选择GPU规格。
 - 计算节点个数：1个或者2个。

- 永久保存日志：打开。
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
 5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-8 GPU 规格运行日志信息（1 个计算节点）

```
58 Iter #0: 342.4 img/sec per GPU
59 Iter #1: 342.7 img/sec per GPU
60 Iter #2: 342.8 img/sec per GPU
61 Iter #3: 342.5 img/sec per GPU
62 Iter #4: 342.9 img/sec per GPU
63 Iter #5: 342.8 img/sec per GPU
64 Iter #6: 342.9 img/sec per GPU
65 Iter #7: 343.0 img/sec per GPU
66 Iter #8: 342.6 img/sec per GPU
67 Iter #9: 342.7 img/sec per GPU
68 Img/sec per GPU: 342.7 +-0.3
69 Total img/sec on 1 GPU(s): 342.7 +-0.3
```

图 4-9 GPU 规格运行日志信息（2 个计算节点）

```
84 Iter #0: 115.1 img/sec per GPU
85 Iter #1: 123.5 img/sec per GPU
86 Iter #2: 115.7 img/sec per GPU
87 Iter #3: 117.4 img/sec per GPU
88 Iter #4: 120.6 img/sec per GPU
89 Iter #5: 126.9 img/sec per GPU
90 Iter #6: 119.4 img/sec per GPU
91 Iter #7: 118.4 img/sec per GPU
92 Iter #8: 122.0 img/sec per GPU
93 Iter #9: 118.2 img/sec per GPU
94 Img/sec per GPU: 119.7 +-6.8
95 Total img/sec on 2 GPU(s): 239.4 +-13.5
```


4.2.4 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore +GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

- [前提条件](#)
- [Step1 创建OBS桶和文件夹](#)
- [Step2 创建数据集并上传至OBS](#)
- [Step3 准备训练脚本并上传至OBS](#)
- [Step4 准备镜像主机](#)
- [Step5 制作自定义镜像](#)
- [Step6 上传镜像至SWR服务](#)
- [Step7 在ModelArts上创建训练作业](#)

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如[表4-4](#)所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-4 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|--|-------------|
| “obs://test-modelarts/mindspore-gpu/resnet/” | 用于存储训练脚本文件。 |
| “obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/” | 用于存储数据集文件。 |
| “obs://test-modelarts/mindspore-gpu/output/” | 用于存储训练输出文件。 |
| “obs://test-modelarts/mindspore-gpu/log/” | 用于存储训练日志文件。 |

Step2 创建数据集并上传至 OBS

进入网站<http://www.cs.toronto.edu/~kriz/cifar.html>，下载“CIFAR-10 binary version (suitable for C programs)”，解压后将数据上传至OBS桶的“obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/”文件夹下。

Step3 准备训练脚本并上传至 OBS

准备本案例所需的resnet文件和脚本run_mpi.sh，并上传至OBS桶的“obs://test-modelarts/mindspore-gpu/resnet/”文件夹下。

resnet文件下载地址：<https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet>

run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: ${MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=' > $MY_MPI_TUNE_FILE
# add -x to each line
sed -i 's/^-x /' ${MY_MPI_TUNE_FILE}
```

```

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<${MA_NUM_HOSTS}; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .*([0-9.]+). */\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: $ip"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip $ip is up"

        echo "${ip} slots=${MY_MPI_SLOTS}" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... @$@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    fi
fi

```

```
else
    echo "[run_mpi] exec command successfully, exited with $RET_CODE"
fi

# stop 1...N worker by killing the sleep proc
sed -i '1d' ${MY_HOME}/hostfile
if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
    echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

    sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

    mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
fi

echo "[run_mpi] exit time: "$(date +%Y-%m-%d-%H:%M:%S)"
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +%Y-%m-%d-%H:%M:%S)"
fi

exit $RET_CODE
```

“obs://test-modelarts/mindspore-gpu/resnet/”文件夹下包含resnet文件和run_mpi.sh。

Step4 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step5 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果docker images命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:  
Version:      18.09.0
```

📖 说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]  
index-url = https://repo.huaweicloud.com/repository/pypi/simple  
trusted-host = repo.huaweicloud.com  
timeout = 120
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl文件。

使用网站https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.8.1/MindSpore/gpu/x86_64/cuda-11.1/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl，mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl文件。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网  
  
# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA  
#  
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds  
# require Docker Engine >= 17.05  
#  
# builder stage  
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder  
  
# 基础容器镜像的默认用户已经是 root  
# USER root  
  
# 使用华为开源镜像站提供的 pypi 配置  
RUN mkdir -p /root/.pip/  
COPY pip.conf /root/.pip/pip.conf  
  
# 复制待安装文件到基础容器镜像中的 /tmp 目录  
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp  
COPY mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl /tmp  
  
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux  
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中  
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3  
  
# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 mindspore whl  
RUN cd /tmp && \  
/home/ma-user/miniconda3/bin/pip install --no-cache-dir \  
/tmp/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl \  

```

```

easydict PyYAML

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
  echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
  apt-get update && \
  apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
  openssh-client openssh-server && \
  ssh -V && \
  mkdir -p /run/sshd && \
  # mlnx ofed
  apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
  libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
  dev flex chrpath libltdl-dev && \
  cd /tmp && \
  tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
  MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
  without-fw-update -q && \
  cd - && \
  rm -rf /tmp/* && \
  apt-get clean && \
  mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
  rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
  tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
  ldconfig && \
  mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
  # setup sshd dir
  mkdir -p ${MA_HOME}/etc && \
  ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N "" -t rsa && \
  mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run && \
  # setup sshd config (listen at {{MY_SSHD_PORT}} port)
  echo "Port {{MY_SSHD_PORT}}\n\
  HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
  AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
  PidFile ${MA_HOME}/var/run/sshd.pid\n\
  StrictModes no\n\
  UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
  # generate ssh key
  ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P "" && \
  cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
  # disable ssh host key checking for all hosts
  echo "Host *\n\

```

```
StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

```
# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
  LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
  PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见[Docker 官方文档](#)。

8. 下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

进入地址https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/，单击“Download”，“Version”选择“5.4-3.5.8.0-LTS”，“OSDistributionVersion”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

9. 下载openmpi-3.0.0-bin.tar.gz。

使用地址<https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz>，下载openmpi-3.0.0-bin.tar.gz文件。

10. 将上述Dockerfile文件、Miniconda3安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl
├── openmpi-3.0.0-bin.tar.gz
└── pip.conf
```

11. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mindspore:1.8.1-ofed-cuda11.1。

```
docker build . -t mindspore:1.8.1-ofed-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged mindspore:1.8.1-ofed-cuda11.1
```

Step6 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。

- a. 使用docker tag命令给上传镜像打标签。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr:{region-id}.{domain}/deep-learning/
mindspore:1.8.1-ofed-cuda11.1
#此处以华为云cn-north-4为例
sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-
learning/mindspore:1.8.1-ofed-cuda11.1
```

- b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr:{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
#此处以华为云cn-north-4为例
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:1.8.1-ofed-
cuda11.1
```

6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:1.8.1-ofed-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step7 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考[使用委托授权](#)。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
 - 创建方式：选择“自定义算法”。
 - 镜像来源：选择“自定义”。
 - 镜像地址：[Step6 上传镜像至SWR服务](#)中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:1.8.1-ofed-cuda11.1”。
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mindspore-gpu/resnet/”，训练代码会被自动下载至训练容器的“\${MA_JOB_DIR}/resnet”目录中，“resnet”为OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 启动命令：“bash \${MA_JOB_DIR}/resnet/run_mpi.sh python \${MA_JOB_DIR}/resnet/train.py”，此处的“resnet”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 训练输入：单击“增加训练输入”，参数名称设置为“data_path”，选择OBS中存放数据集的目录，例如“obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/”，获取方式设置为“超参”。
 - 训练输出：单击“增加训练输出”，参数名称设置为“output_path”，选择OBS存放训练输出的路径，例如：“obs://test-modelarts/mindspore-gpu/output/”，获取方式设置为“超参”，预下载至本地目录设置为“不下载”。
 - 超参：单击“增加超参”，增加如下四个超参：
 - run_distribute=True
 - device_num=1（根据规格中的GPU卡数设置）
 - device_target=GPU
 - epoch_size=2
 - 环境变量：单击“增加环境变量”，增加环境变量：MY_SSHD_PORT=38888。
 - 资源池：选择公共资源池。
 - 资源类型：选择GPU规格。
 - 计算节点个数：1个或者2个。
 - 永久保存日志：打开。
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mindspore-gpu/log/”。

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-10 GPU 规格运行日志信息（1 个计算节点）

```
127 epoch: 1 step: 1875, loss is 1.4800076
128 Train epoch time: 369422.027 ms, per step time: 197.025 ms
129 epoch: 2 step: 1875, loss is 1.0306032
130 Train epoch time: 66996.087 ms, per step time: 35.731 ms
```

图 4-11 GPU 规格运行日志信息（2 个计算节点）

```
187 epoch: 1 step: 937, loss is 1.8482083
188 epoch: 1 step: 937, loss is 1.342748
189 Train epoch time: 488492.170 ms, per step time: 521.336 ms
190 Train epoch time: 488490.528 ms, per step time: 521.335 ms
191 epoch: 2 step: 937, loss is 0.9150252
192 Train epoch time: 180200.654 ms, per step time: 192.317 ms
193 epoch: 2 step: 937, loss is 0.9933052
194 Train epoch time: 180199.969 ms, per step time: 192.316 ms
195 172.16.0.45 slots=1
```

4.2.5 示例：从 0 到 1 制作自定义镜像并用于训练（Tensorflow +GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Tensorflow，训练使用的资源是GPU。

📖 说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.2
- python-3.7.13
- mlnx ofed-5.4
- tensorflow gpu-2.10.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. [前提条件](#)
2. [Step1 创建OBS桶和文件夹](#)
3. [Step2 创建数据集并上传至OBS](#)
4. [Step3 准备训练脚本并上传至OBS](#)
5. [Step4 准备镜像主机](#)
6. [Step5 制作自定义镜像](#)
7. [Step6 上传镜像至SWR服务](#)
8. [Step7 在ModelArts上创建训练作业](#)

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如[表4-5](#)所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-5 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|---|-------------|
| “obs://test-modelarts/tensorflow/code/” | 用于存储训练脚本文件。 |
| “obs://test-modelarts/tensorflow/data/” | 用于存储数据集文件。 |
| “obs://test-modelarts/tensorflow/log/” | 用于存储训练日志文件。 |

Step2 创建数据集并上传至 OBS

使用网站<https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>，下载“mnist.npz”文件并上传至OBS桶的“obs://test-modelarts/tensorflow/data/”文件夹下。

Step3 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本mnist.py，并上传至OBS桶的“obs://test-modelarts/tensorflow/code/”文件夹下。

mnist.py文件内容如下：

```
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='TensorFlow quick start')
parser.add_argument('--data_url', type=str, default='./Data', help='path where the dataset is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

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

model.fit(x_train, y_train, epochs=5)
```

Step4 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step5 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果docker images命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:  
Version:      18.09.0
```

📖 说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]  
index-url = https://repo.huaweicloud.com/repository/pypi/simple  
trusted-host = repo.huaweicloud.com  
timeout = 120
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manynlinux2014_x86_64.whl文件。

使用网站<https://pypi.org/project/tensorflow-gpu/2.10.0/#files>，下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manynlinux2014_x86_64.whl文件。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网  
  
# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA  
#  
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds  
# require Docker Engine >= 17.05  
#  
# builder stage  
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04 AS builder  
  
# 基础容器镜像的默认用户已经是 root  
# USER root  
  
# 使用华为开源镜像站提供的 pypi 配置  
RUN mkdir -p /root/.pip/  
COPY pip.conf /root/.pip/pip.conf  
  
# 复制待安装文件到基础容器镜像中的 /tmp 目录  
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp  
COPY tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manynlinux2014_x86_64.whl /tmp  
  
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux  
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中  
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3  
  
# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 tensorflow whl  
RUN cd /tmp && \  
/home/ma-user/miniconda3/bin/pip install --no-cache-dir \  

```

```
/tmp/tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

RUN cd /tmp && \
/home/ma-user/miniconda3/bin/pip install --no-cache-dir keras==2.10.0

# 构建最终容器镜像
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
apt-get update && \
apt-get install -y vim curl net-tools iputils-ping && \
# mlnx ofed
apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
dev flex chrpath libltdl-dev && \
cd /tmp && \
tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
without-fw-update -q && \
cd - && \
rm -rf /tmp/* && \
apt-get clean && \
mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1，以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见[Docker 官方文档](#)。

8. 下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。
进入[地址](#)，单击“Download”，“Version”选择“5.4-3.5.8.0-LTS”，“OSDistributionVersion”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。
9. 将上述Dockerfile文件、Miniconda3 安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
└── tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像tensorflow:2.10.0-ofed-cuda11.2。

```
docker build . -t tensorflow:2.10.0-ofed-cuda11.2
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged tensorflow:2.10.0-ofed-cuda11.2
```

Step6 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
 - a. 使用docker tag命令给上传镜像打标签。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/
tensorflow:2.10.0-ofed-cuda11.2
#此处以华为云cn-north-4为例
sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.cn-north-4.myhuaweicloud.com/deep-
learning/tensorflow:2.10.0-ofed-cuda11.2
```
 - b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
#此处以华为云cn-north-4为例
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/tensorflow:2.10.0-ofed-
cuda11.2
```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/tensorflow:2.10.0-ofed-cuda11.2”即为此自定义镜像的“SWR_URL”。

Step7 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考[使用委托授权](#)。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
 - 创建方式：选择“自定义算法”。
 - 镜像来源：选择“自定义”。
 - 镜像地址：[Step5 制作自定义镜像](#)中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/tensorflow:2.10.0-ofed-cuda11.2”。
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/tensorflow/code/”，训练代码会被自动下载至训练容器的“\${MA_JOB_DIR}/code”目录中，“code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 启动命令：“python \${MA_JOB_DIR}/code/mnist.py”，此处的“code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
 - 训练输入：单击“增加训练输入”，参数名称设置为“data_path”，选择OBS中存放“mnist.npz”的目录，例如“obs://test-modelarts/tensorflow/data/mnist.npz”，获取方式设置为“超参”。

- 资源池：选择公共资源池。
 - 资源类型：选择GPU规格。
 - 计算节点个数：1个。
 - 永久保存日志：打开。
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mindspore-gpu/log/”。
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
 5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-12 GPU 规格运行日志信息

```
0.9767.....
323 1503/1875 [=====>.....] - ETA: 0s - loss: 0.0741 - accuracy:
0.9769.....
324 1533/1875 [=====>.....] - ETA: 0s - loss: 0.0743 - accuracy:
0.9769.....
325 1564/1875 [=====>.....] - ETA: 0s - loss: 0.0746 - accuracy:
0.9768.....
326 1595/1875 [=====>.....] - ETA: 0s - loss: 0.0741 - accuracy:
0.9770.....
327 1624/1875 [=====>.....] - ETA: 0s - loss: 0.0742 - accuracy:
0.9770.....
328 1654/1875 [=====>.....] - ETA: 0s - loss: 0.0745 - accuracy:
0.9770.....
329 1685/1875 [=====>.....] - ETA: 0s - loss: 0.0747 - accuracy:
0.9768.....
330 1716/1875 [=====>.....] - ETA: 0s - loss: 0.0752 - accuracy:
0.9767.....
331 1747/1875 [=====>.....] - ETA: 0s - loss: 0.0755 - accuracy:
0.9767.....
332 1778/1875 [=====>.....] - ETA: 0s - loss: 0.0753 - accuracy:
0.9767.....
333 1809/1875 [=====>.....] - ETA: 0s - loss: 0.0751 - accuracy:
0.9768.....
334 1841/1875 [=====>.....] - ETA: 0s - loss: 0.0753 - accuracy:
0.9767.....
335 1872/1875 [=====>.....] - ETA: 0s - loss: 0.0753 - accuracy:
0.9767.....
336 1875/1875 [=====] - 3s 2ms/step - loss: 0.0752 - accuracy: 0.9767
```

4.2.6 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore + Ascend）

4.2.6.1 场景描述

本案例介绍如何从0到1制作Ascend容器镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是专属资源池的Ascend芯片。

约束限制

- 由于案例中需要下载商用版CANN，因此本案例仅面向有下载权限的渠道用户，非渠道用户建议参考其他自定义镜像制作教程。

- Mindspore版本与CANN版本，CANN版本与Ascend驱动/固件版本均有严格的匹配关系，版本不匹配会导致训练失败。

场景描述

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用Ascend规格资源运行训练任务。

- ubuntu-18.04
- cann-6.3.RC2 (商用版本)
- python-3.7.13
- mindspore-2.1.1

说明

- 本教程以cann-6.3.RC2、mindspore-2.1.1为例介绍。
- 本示例仅用于示意Ascend容器镜像制作流程，且在匹配正确的Ascend驱动/固件版本的专属资源池上运行通过。

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. [Step1 创建OBS桶和文件夹](#)
2. [Step2 准备脚本文件并上传至OBS中](#)
3. [Step3 制作自定义镜像](#)
4. [Step4 上传镜像至SWR](#)
5. [Step5 在ModelArts上创建Notebook并调试](#)
6. [Step6 在ModelArts上创建训练作业](#)

4.2.6.2 Step1 创建 OBS 桶和文件夹

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。如下示例中，请创建命名为“test-modelarts”的桶，并创建如[表4-6](#)所示的文件夹。

创建OBS桶和文件夹的操作指导请参见[创建桶](#)和[新建文件夹](#)。

请确保您使用的OBS与ModelArts在同一区域。

表 4-6 OBS 桶文件夹列表

| 文件夹名称 | 用途 |
|---|----------------------|
| obs://test-modelarts/ascend/demo-code/ | 用于存储Ascend训练脚本文件。 |
| obs://test-modelarts/ascend/demo-code/run_ascend/ | 用于存储Ascend训练脚本的启动脚本。 |
| obs://test-modelarts/ascend/log/ | 用于存储训练日志文件。 |

4.2.6.3 Step2 准备脚本文件并上传至 OBS 中

1. 准备本案例所需训练脚本mindspore-verification.py文件和Ascend的启动脚本文件（共5个）。
 - 训练脚本文件具体内容请参见[训练脚本mindspore-verification.py文件](#)。
 - Ascend的启动脚本文件包括以下5个，具体脚本内容请参见[Ascend的启动脚本文件](#)。
 - i. run_ascend.py
 - ii. common.py
 - iii. rank_table.py
 - iv. manager.py
 - v. fmk.py

📖 说明

mindspore-verification.py和run_ascend.py脚本文件在创建训练作业时的“启动命令”参数中调用，具体请参见[启动命令](#)。

run_ascend.py脚本运行时会调用common.py、rank_table.py、manager.py、fmk.py脚本。

2. 上传训练脚本mindspore-verification.py文件至OBS桶的“obs://test-modelarts/ascend/demo-code/”文件夹下。
3. 上传Ascend的启动脚本文件（共5个）至OBS桶的“obs://test-modelarts/ascend/demo-code/run_ascend/”文件夹下。

训练脚本 mindspore-verification.py 文件

mindspore-verification.py文件内容如下：

```
import os
import numpy as np
from mindspore import Tensor
import mindspore.ops as ops
import mindspore.context as context

print('Ascend Envs')
print('-----')
print('JOB_ID: ', os.environ['JOB_ID'])
print('RANK_TABLE_FILE: ', os.environ['RANK_TABLE_FILE'])
print('RANK_SIZE: ', os.environ['RANK_SIZE'])
print('ASCEND_DEVICE_ID: ', os.environ['ASCEND_DEVICE_ID'])
print('DEVICE_ID: ', os.environ['DEVICE_ID'])
print('RANK_ID: ', os.environ['RANK_ID'])
```

```
print('-----')

context.set_context(device_target="Ascend")
x = Tensor(np.ones([1,3,3,4]).astype(np.float32))
y = Tensor(np.ones([1,3,3,4]).astype(np.float32))

print(ops.add(x, y))
```

Ascend 的启动脚本文件

- run_ascend.py

```
import sys
import os

from common import RunAscendLog
from common import RankTableEnv

from rank_table import RankTable, RankTableTemplate1, RankTableTemplate2

from manager import FMKManager

if __name__ == '__main__':
    log = RunAscendLog.setup_run_ascend_logger()

    if len(sys.argv) <= 1:
        log.error('there are not enough args')
        sys.exit(1)

    train_command = sys.argv[1:]
    log.info('training command')
    log.info(train_command)

    if os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1) is not None:
        # new format rank table file
        rank_table_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1)
        RankTable.wait_for_available(rank_table_path)
        rank_table = RankTableTemplate1(rank_table_path)
    else:
        # old format rank table file
        rank_table_path_origin = RankTableEnv.get_rank_table_template2_file_path()
        RankTable.wait_for_available(rank_table_path_origin)
        rank_table = RankTableTemplate2(rank_table_path_origin)

    if rank_table.get_device_num() >= 1:
        log.info('set rank table %s env to %s' % (RankTableEnv.RANK_TABLE_FILE,
rank_table.get_rank_table_path()))
        RankTableEnv.set_rank_table_env(rank_table.get_rank_table_path())
    else:
        log.info('device num < 1, unset rank table %s env' % RankTableEnv.RANK_TABLE_FILE)
        RankTableEnv.unset_rank_table_env()

    instance = rank_table.get_current_instance()
    server = rank_table.get_server(instance.server_id)
    current_instance = RankTable.convert_server_to_instance(server)

    fmk_manager = FMKManager(current_instance)
    fmk_manager.run(rank_table.get_device_num(), train_command)
    return_code = fmk_manager.monitor()

    fmk_manager.destroy()

    sys.exit(return_code)
```

- common.py

```
import logging
import os

logo = 'Training'
```

```
# Rank Table Constants
class RankTableEnv:
    RANK_TABLE_FILE = 'RANK_TABLE_FILE'

    RANK_TABLE_FILE_V1 = 'RANK_TABLE_FILE_V1_0'

    HCCL_CONNECT_TIMEOUT = 'HCCL_CONNECT_TIMEOUT'

    # jobstart_hccl.json is provided by the volcano controller of Cloud-Container-Engine(CCE)
    HCCL_JSON_FILE_NAME = 'jobstart_hccl.json'

    RANK_TABLE_FILE_DEFAULT_VALUE = '/user/config/%s' % HCCL_JSON_FILE_NAME

    @staticmethod
    def get_rank_table_template1_file_dir():
        parent_dir = os.environ[ModelArts.MA_MOUNT_PATH_ENV]
        return os.path.join(parent_dir, 'rank_table')

    @staticmethod
    def get_rank_table_template2_file_path():
        rank_table_file_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE)
        if rank_table_file_path is None:
            return RankTableEnv.RANK_TABLE_FILE_DEFAULT_VALUE

        return os.path.join(os.path.normpath(rank_table_file_path),
RankTableEnv.HCCL_JSON_FILE_NAME)

    @staticmethod
    def set_rank_table_env(path):
        os.environ[RankTableEnv.RANK_TABLE_FILE] = path

    @staticmethod
    def unset_rank_table_env():
        del os.environ[RankTableEnv.RANK_TABLE_FILE]

class ModelArts:
    MA_MOUNT_PATH_ENV = 'MA_MOUNT_PATH'
    MA_CURRENT_INSTANCE_NAME_ENV = 'MA_CURRENT_INSTANCE_NAME'
    MA_VJ_NAME = 'MA_VJ_NAME'

    MA_CURRENT_HOST_IP = 'MA_CURRENT_HOST_IP'

    CACHE_DIR = '/cache'

    TMP_LOG_DIR = '/tmp/log/'

    FMK_WORKSPACE = 'workspace'

    @staticmethod
    def get_current_instance_name():
        return os.environ[ModelArts.MA_CURRENT_INSTANCE_NAME_ENV]

    @staticmethod
    def get_current_host_ip():
        return os.environ.get(ModelArts.MA_CURRENT_HOST_IP)

    @staticmethod
    def get_job_id():
        ma_vj_name = os.environ[ModelArts.MA_VJ_NAME]
        return ma_vj_name.replace('ma-job', 'modelarts-job', 1)

    @staticmethod
    def get_parent_working_dir():
        if ModelArts.MA_MOUNT_PATH_ENV in os.environ:
            return os.path.join(os.environ.get(ModelArts.MA_MOUNT_PATH_ENV),
ModelArts.FMK_WORKSPACE)
```

```

        return ModelArts.CACHE_DIR

class RunAscendLog:

    @staticmethod
    def setup_run_ascend_logger():
        name = logo
        formatter = logging.Formatter(fmt='[run ascend] %(asctime)s - %(levelname)s - %(message)s')

        handler = logging.StreamHandler()
        handler.setFormatter(formatter)

        logger = logging.getLogger(name)
        logger.setLevel(logging.INFO)
        logger.addHandler(handler)
        logger.propagate = False
        return logger

    @staticmethod
    def get_run_ascend_logger():
        return logging.getLogger(logo)

```

- rank_table.py

```

import json
import time
import os

from common import ModelArts
from common import RunAscendLog
from common import RankTableEnv

log = RunAscendLog.get_run_ascend_logger()

class Device:
    def __init__(self, device_id, device_ip, rank_id):
        self.device_id = device_id
        self.device_ip = device_ip
        self.rank_id = rank_id

class Instance:
    def __init__(self, pod_name, server_id, devices):
        self.pod_name = pod_name
        self.server_id = server_id
        self.devices = self.parse_devices(devices)

    @staticmethod
    def parse_devices(devices):
        if devices is None:
            return []
        device_object_list = []
        for device in devices:
            device_object_list.append(Device(device['device_id'], device['device_ip'], ""))

        return device_object_list

    def set_devices(self, devices):
        self.devices = devices

class Group:
    def __init__(self, group_name, device_count, instance_count, instance_list):
        self.group_name = group_name
        self.device_count = int(device_count)
        self.instance_count = int(instance_count)
        self.instance_list = self.parse_instance_list(instance_list)

    @staticmethod

```

```

def parse_instance_list(instance_list):
    instance_object_list = []
    for instance in instance_list:
        instance_object_list.append(
            Instance(instance['pod_name'], instance['server_id'], instance['devices']))

    return instance_object_list

class RankTable:
    STATUS_FIELD = 'status'
    COMPLETED_STATUS = 'completed'

    def __init__(self):
        self.rank_table_path = ""
        self.rank_table = {}

    @staticmethod
    def read_from_file(file_path):
        with open(file_path) as json_file:
            return json.load(json_file)

    @staticmethod
    def wait_for_available(rank_table_file, period=1):
        log.info('Wait for Rank table file at %s ready' % rank_table_file)
        complete_flag = False
        while not complete_flag:
            with open(rank_table_file) as json_file:
                data = json.load(json_file)
                if data[RankTable.STATUS_FIELD] == RankTable.COMPLETED_STATUS:
                    log.info('Rank table file is ready for read')
                    log.info('\n' + json.dumps(data, indent=4))
                    return True

            time.sleep(period)

        return False

    @staticmethod
    def convert_server_to_instance(server):
        device_list = []
        for device in server['device']:
            device_list.append(
                Device(device_id=device['device_id'], device_ip=device['device_ip'],
rank_id=device['rank_id']))

        ins = Instance(pod_name="", server_id=server['server_id'], devices=[])
        ins.set_devices(device_list)
        return ins

    def get_rank_table_path(self):
        return self.rank_table_path

    def get_server(self, server_id):
        for server in self.rank_table['server_list']:
            if server['server_id'] == server_id:
                log.info('Current server')
                log.info('\n' + json.dumps(server, indent=4))
                return server

        log.error('server [%s] is not found' % server_id)
        return None

class RankTableTemplate2(RankTable):

    def __init__(self, rank_table_template2_path):
        super().__init__()

```

```
json_data = self.read_from_file(file_path=rank_table_template2_path)

self.status = json_data[RankTableTemplate2.STATUS_FIELD]
if self.status != RankTableTemplate2.COMPLETED_STATUS:
    return

# sorted instance list by the index of instance
# assert there is only one group
json_data["group_list"][0]["instance_list"] = sorted(json_data["group_list"][0]["instance_list"],
                                                       key=RankTableTemplate2.get_index)

self.group_count = int(json_data["group_count"])
self.group_list = self.parse_group_list(json_data["group_list"])

self.rank_table_path, self.rank_table = self.convert_template2_to_template1_format_file()

@staticmethod
def parse_group_list(group_list):
    group_object_list = []
    for group in group_list:
        group_object_list.append(
            Group(group["group_name"], group["device_count"], group["instance_count"],
group["instance_list"]))

    return group_object_list

@staticmethod
def get_index(instance):
    # pod_name example: job94dc1dbf-job-bj4-yolov4-15
    pod_name = instance["pod_name"]
    return int(pod_name[pod_name.rfind("-") + 1:])

def get_current_instance(self):
    """
    get instance by pod name
    specially, return the first instance when the pod name is None
    :return:
    """
    pod_name = ModelArts.get_current_instance_name()
    if pod_name is None:
        if len(self.group_list) > 0:
            if len(self.group_list[0].instance_list) > 0:
                return self.group_list[0].instance_list[0]

    return None

    for group in self.group_list:
        for instance in group.instance_list:
            if instance.pod_name == pod_name:
                return instance
    return None

def convert_template2_to_template1_format_file(self):
    rank_table_template1_file = {
        'status': 'completed',
        'version': '1.0',
        'server_count': '0',
        'server_list': []
    }

    logic_index = 0
    server_map = {}
    # collect all devices in all groups
    for group in self.group_list:
        if group.device_count == 0:
            continue
        for instance in group.instance_list:
            if instance.server_id not in server_map:
                server_map[instance.server_id] = []
```

```

        for device in instance.devices:
            template1_device = {
                'device_id': device.device_id,
                'device_ip': device.device_ip,
                'rank_id': str(logic_index)
            }
            logic_index += 1
            server_map[instance.server_id].append(template1_device)

    server_count = 0
    for server_id in server_map:
        rank_table_template1_file['server_list'].append({
            'server_id': server_id,
            'device': server_map[server_id]
        })
        server_count += 1

    rank_table_template1_file['server_count'] = str(server_count)

    log.info('Rank table file (Template1)')
    log.info('\n' + json.dumps(rank_table_template1_file, indent=4))

    if not os.path.exists(RankTableEnv.get_rank_table_template1_file_dir()):
        os.makedirs(RankTableEnv.get_rank_table_template1_file_dir())

    path = os.path.join(RankTableEnv.get_rank_table_template1_file_dir(),
RankTableEnv.HCCL_JSON_FILE_NAME)
    with open(path, 'w') as f:
        f.write(json.dumps(rank_table_template1_file))
        log.info('Rank table file (Template1) is generated at %s', path)

    return path, rank_table_template1_file

def get_device_num(self):
    total_device_num = 0
    for group in self.group_list:
        total_device_num += group.device_count
    return total_device_num

class RankTableTemplate1(RankTable):
    def __init__(self, rank_table_template1_path):
        super().__init__()
        self.rank_table_path = rank_table_template1_path
        self.rank_table = self.read_from_file(file_path=rank_table_template1_path)

    def get_current_instance(self):
        current_server = None
        server_list = self.rank_table['server_list']
        if len(server_list) == 1:
            current_server = server_list[0]
        elif len(server_list) > 1:
            host_ip = ModelArts.get_current_host_ip()
            if host_ip is not None:
                for server in server_list:
                    if server['server_id'] == host_ip:
                        current_server = server
                        break
            else:
                current_server = server_list[0]

        if current_server is None:
            log.error('server is not found')
            return None
        return self.convert_server_to_instance(current_server)

    def get_device_num(self):
        server_list = self.rank_table['server_list']

```

```
device_num = 0
for server in server_list:
    device_num += len(server['device'])
return device_num
```

- **manager.py**

```
import time
import os
import os.path
import signal

from common import RunAscendLog
from fmk import FMK

log = RunAscendLog.get_run_ascend_logger()

class FMKManager:
    # max destroy time: ~20 (15 + 5)
    # ~ 15 (1 + 2 + 4 + 8)
    MAX_TEST_PROC_CNT = 4

    def __init__(self, instance):
        self.instance = instance
        self.fmk = []
        self.fmk_processes = []
        self.get_sigterm = False
        self.max_test_proc_cnt = FMKManager.MAX_TEST_PROC_CNT

    # break the monitor and destroy processes when get terminate signal
    def term_handle(func):
        def receive_term(signum, stack):
            log.info('Received terminate signal %d, try to destroyed all processes' % signum)
            stack.f_locals['self'].get_sigterm = True

        def handle_func(self, *args, **kwargs):
            origin_handle = signal.getsignal(signal.SIGTERM)
            signal.signal(signal.SIGTERM, receive_term)
            res = func(self, *args, **kwargs)
            signal.signal(signal.SIGTERM, origin_handle)
            return res

        return handle_func

    def run(self, rank_size, command):
        for index, device in enumerate(self.instance.devices):
            fmk_instance = FMK(index, device)
            self.fmk.append(fmk_instance)

            self.fmk_processes.append(fmk_instance.run(rank_size, command))

    @term_handle
    def monitor(self, period=1):
        # busy waiting for all fmk processes exit by zero
        # or there is one process exit by non-zero

        fmk_cnt = len(self.fmk_processes)
        zero_ret_cnt = 0
        while zero_ret_cnt != fmk_cnt:
            zero_ret_cnt = 0
            for index in range(fmk_cnt):
                fmk = self.fmk[index]
                fmk_process = self.fmk_processes[index]
                if fmk_process.poll() is not None:
                    if fmk_process.returncode != 0:
                        log.error('proc-rank-%s-device-%s (pid: %d) has exited with non-zero code: %d'
                                % (fmk.rank_id, fmk.device_id, fmk_process.pid, fmk_process.returncode))
            return fmk_process.returncode
```



```

        zero_ret_cnt += 1
    if self.get_sigterm:
        break
    time.sleep(period)

    return 0

def destroy(self, base_period=1):
    log.info('Begin destroy training processes')
    self.send_sigterm_to_fm_k_process()
    self.wait_fm_k_process_end(base_period)
    log.info('End destroy training processes')

def send_sigterm_to_fm_k_process(self):
    # send SIGTERM to fm_k processes (and process group)
    for r_index in range(len(self.fm_k_processes) - 1, -1, -1):
        fm_k = self.fm_k[r_index]
        fm_k_process = self.fm_k_processes[r_index]
        if fm_k_process.poll() is not None:
            log.info('proc-rank-%s-device-%s (pid: %d) has exited before receiving the term signal',
                    fm_k.rank_id, fm_k.device_id, fm_k_process.pid)
            del self.fm_k_processes[r_index]
            del self.fm_k[r_index]

    try:
        os.killpg(fm_k_process.pid, signal.SIGTERM)
    except ProcessLookupError:
        pass

def wait_fm_k_process_end(self, base_period):
    test_cnt = 0
    period = base_period
    while len(self.fm_k_processes) > 0 and test_cnt < self.max_test_proc_cnt:
        for r_index in range(len(self.fm_k_processes) - 1, -1, -1):
            fm_k = self.fm_k[r_index]
            fm_k_process = self.fm_k_processes[r_index]
            if fm_k_process.poll() is not None:
                log.info('proc-rank-%s-device-%s (pid: %d) has exited',
                        fm_k.rank_id, fm_k.device_id, fm_k_process.pid)
                del self.fm_k_processes[r_index]
                del self.fm_k[r_index]
            if not self.fm_k_processes:
                break

        time.sleep(period)
        period *= 2
        test_cnt += 1

    if len(self.fm_k_processes) > 0:
        for r_index in range(len(self.fm_k_processes) - 1, -1, -1):
            fm_k = self.fm_k[r_index]
            fm_k_process = self.fm_k_processes[r_index]
            if fm_k_process.poll() is None:
                log.warn('proc-rank-%s-device-%s (pid: %d) has not exited within the max waiting time,
                        'send kill signal',
                        fm_k.rank_id, fm_k.device_id, fm_k_process.pid)
                os.killpg(fm_k_process.pid, signal.SIGKILL)

```

- **fm_k.py**

```

import os
import subprocess
import pathlib
from contextlib import contextmanager

from common import RunAscendLog
from common import RankTableEnv
from common import ModelArts

log = RunAscendLog.get_run_ascend_logger()

```

```

class FMK:

    def __init__(self, index, device):
        self.job_id = ModelArts.get_job_id()
        self.rank_id = device.rank_id
        self.device_id = str(index)

    def gen_env_for_fmk(self, rank_size):
        current_envs = os.environ.copy()

        current_envs['JOB_ID'] = self.job_id

        current_envs['ASCEND_DEVICE_ID'] = self.device_id
        current_envs['DEVICE_ID'] = self.device_id

        current_envs['RANK_ID'] = self.rank_id
        current_envs['RANK_SIZE'] = str(rank_size)

        FMK.set_env_if_not_exist(current_envs, RankTableEnv.HCCL_CONNECT_TIMEOUT, str(1800))

        log_dir = FMK.get_log_dir()
        process_log_path = os.path.join(log_dir, self.job_id, 'ascend', 'process_log', 'rank_' + self.rank_id)
        FMK.set_env_if_not_exist(current_envs, 'ASCEND_PROCESS_LOG_PATH', process_log_path)
        pathlib.Path(current_envs['ASCEND_PROCESS_LOG_PATH']).mkdir(parents=True, exist_ok=True)

        return current_envs

    @contextmanager
    def switch_directory(self, directory):
        owd = os.getcwd()
        try:
            os.chdir(directory)
            yield directory
        finally:
            os.chdir(owd)

    def get_working_dir(self):
        fmk_workspace_prefix = ModelArts.get_parent_working_dir()
        return os.path.join(os.path.normpath(fmk_workspace_prefix), 'device%s' % self.device_id)

    @staticmethod
    def get_log_dir():
        parent_path = os.getenv(ModelArts.MA_MOUNT_PATH_ENV)
        if parent_path:
            log_path = os.path.join(parent_path, 'log')
            if os.path.exists(log_path):
                return log_path

        return ModelArts.TMP_LOG_DIR

    @staticmethod
    def set_env_if_not_exist(envs, env_name, env_value):
        if env_name in os.environ:
            log.info('env already exists. env_name: %s, env_value: %s' % (env_name, env_value))
            return

        envs[env_name] = env_value

    def run(self, rank_size, command):
        envs = self.gen_env_for_fmk(rank_size)
        log.info('bootstrap proc-rank-%s-device-%s' % (self.rank_id, self.device_id))

        log_dir = FMK.get_log_dir()
        if not os.path.exists(log_dir):
            os.makedirs(log_dir)

        log_file = '%s-proc-rank-%s-device-%s.txt' % (self.job_id, self.rank_id, self.device_id)

```

```
log_file_path = os.path.join(log_dir, log_file)

working_dir = self.get_working_dir()
if not os.path.exists(working_dir):
    os.makedirs(working_dir)

with self.switch_directory(working_dir):
    # os.setsid: change the process(forked) group id to itself
    training_proc = subprocess.Popen(command, env=envs, preexec_fn=os.setsid,
                                     stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

    log.info('proc-rank-%s-device-%s (pid: %d)', self.rank_id, self.device_id, training_proc.pid)

    # https://docs.python.org/3/library/subprocess.html#subprocess.Popen.wait
    subprocess.Popen(['tee', log_file_path], stdin=training_proc.stdout)

return training_proc
```

4.2.6.4 Step3 制作自定义镜像

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cann-6.3.RC2(商用版本)
- python-3.7.13
- mindspore-2.1.1

📖 说明

Mindspore版本与CANN版本，CANN版本和Ascend驱动/固件版本均有严格的匹配关系，版本不匹配会导致训练失败。

本示例仅用于示意Ascend容器镜像制作流程，且在匹配正确的Ascend驱动/固件版本的专属资源池上运行通过。

1. 准备一台Linux **aarch64**架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录Linux弹性云服务器](#)。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

2. 安装Docker。

以Linux **aarch64**架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见[Docker官方文档](#)。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

启动docker。

```
systemctl start docker
```

3. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:
Version:      18.09.0
```

📖 说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

4. 准备名为context的文件夹。

```
mkdir -p context
```

5. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，可以查看pip.conf文件内容。

6. 准备可用的apt源文件Ubuntu-Ports-bionic.list。本示例使用华为开源镜像站提供的apt源，执行如下命令获取apt源文件。

```
wget -O Ubuntu-Ports-bionic.list https://repo.huaweicloud.com/repository/conf/Ubuntu-Ports-bionic.list
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索Ubuntu-Ports，可以查看获取apt源文件的命令。

7. 下载CANN 6.3.RC2-linux_aarch64与mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl安装文件。

- 下载run文件“Ascend-cann-nnae_6.3.RC2_linux-aarch64.run”（[下载链接](#)）。
- 下载whl文件“mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl”（[下载链接](#)）。

📖 说明

ModelArts当前仅支持CANN商用版本，不支持社区版。

8. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.10.3-Linux-aarch64.sh，下载Miniconda3-py37-4.10.3安装文件（对应python 3.7.10）。

9. 将上述pip源文件、*.run文件、*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

10. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网
FROM arm64v8/ubuntu:18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 安装 OS 依赖（使用华为开源镜像站）
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
```

```
# utils
ca-certificates vim curl \
# CANN 6.3.RC2
gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
net-tools libblas-dev gfortran libblas3 && \
apt-get clean && \
mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
# 修改 CANN 6.3.RC2 安装目录的父目录权限，使得 ma-user 可以写入
chmod o+w /usr/local

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:100 pip.conf /home/ma-user/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY --chown=ma-user:100 Miniconda3-py37_4.10.3-Linux-aarch64.sh /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.10.3-Linux-aarch64.sh -b -p /home/ma-user/miniconda3

ENV PATH=$PATH:/home/ma-user/miniconda3/bin

# 安装 CANN 6.3.RC2 Python Package 依赖
RUN pip install numpy~=1.14.3 decorator~=4.4.0 sympy~=1.4 cffi~=1.12.3 protobuf~=3.11.3 \
    attrs pyyaml pathlib2 scipy requests psutil absl-py

# 安装 CANN 6.3.RC2 至 /usr/local/Ascend 目录
COPY --chown=ma-user:100 Ascend-cann-nnae_6.3.RC2_linux-aarch64.run /tmp
RUN chmod +x /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run && \
    /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run --install --install-path=/usr/local/Ascend

# 安装 MindSpore 2.1.1
COPY --chown=ma-user:100 mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl /tmp
RUN chmod +x /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl && \
    pip install /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl

# 构建最终容器镜像
FROM arm64v8/ubuntu:18.04

# 安装 OS 依赖（使用华为开源镜像站）
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 6.3.RC2
    gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
    net-tools libblas-dev gfortran libblas3 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3
COPY --chown=ma-user:100 --from=builder /home/ma-user/Ascend /home/ma-user/Ascend
COPY --chown=ma-user:100 --from=builder /home/ma-user/var /home/ma-user/var
COPY --chown=ma-user:100 --from=builder /usr/local/Ascend /usr/local/Ascend
```

```
# 设置容器镜像预置环境变量
# 请务必设置 CANN 相关环境变量
# 请务必设置 Ascend Driver 相关环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/usr/local/Ascend/nnae/latest/bin:/usr/local/Ascend/nnae/latest/compiler/
cccec_compiler/bin:/home/ma-user/miniconda3/bin \
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Ascend/driver/lib64:/usr/local/Ascend/driver/
lib64/common:/usr/local/Ascend/driver/lib64/driver:/usr/local/Ascend/nnae/latest/lib64:/usr/local/
Ascend/nnae/latest/lib64/plugin/opskernel:/usr/local/Ascend/nnae/latest/lib64/plugin/nnengine \
PYTHONPATH=$PYTHONPATH:/usr/local/Ascend/nnae/latest/python/site-packages:/usr/local/
Ascend/nnae/latest/opp/built-in/op_impl/ai_core/tbe \
ASCEND_AICPU_PATH=/usr/local/Ascend/nnae/latest \
ASCEND_OPP_PATH=/usr/local/Ascend/nnae/latest/opp \
ASCEND_HOME_PATH=/usr/local/Ascend/nnae/latest \
PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见[Docker 官方文档](#)。

11. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── Dockerfile
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

12. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像。
`docker build . -t mindspore:2.1.1-cann6.3.RC2`

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged mindspore:2.1.1-cann6.3.RC2
```

13. 将制作完成的镜像上传至SWR服务，具体参见[Step4 上传镜像至SWR](#)。

4.2.6.5 Step4 上传镜像至 SWR

本章节介绍如何将制作好的镜像上传至SWR服务，方便后续在ModelArts上创建训练作业时调用。

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。

- a. 使用docker tag命令给上传镜像打标签。

#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker tag mindspore:2.1.1-cann6.3.RC2 swr.{region}.{domain}/deep-learning/
mindspore:2.1.1-cann6.3.RC2
```

#以华为云北京四为例：

```
sudo docker tag mindspore:2.1.1-cann6.3.RC2 swr.cn-north-4.myhuaweicloud.com/deep-learning/
mindspore:2.1.1-cann6.3.RC2
```

- b. 使用docker push命令上传镜像。

#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。

```
sudo docker push swr.{region}.{domain}/deep-learning/mindspore:2.1.1-cann6.3.RC2
```

#以华为云北京四为例：

```
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:2.1.1-
cann6.3.RC2
```

6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
“swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:2.1.1-cann6.3.RC2”即为此自定义镜像的“SWR_URL”。

4.2.6.6 Step5 在 ModelArts 上创建 Notebook 并调试

1. 将[上传到SWR上的镜像](#)注册到ModelArts的镜像管理中。
登录ModelArts管理控制台，在左侧导航栏中选择“镜像管理”，单击“注册镜像”，根据界面提示注册镜像。注册后的镜像可以用于创建Notebook。
2. 在Notebook中使用自定义镜像创建Notebook并调试，调试成功后，保存镜像。
 - a. 在Notebook中使用自定义镜像创建Notebook操作请参见[基于自定义镜像创建Notebook实例](#)。
 - b. 保存Notebook镜像操作请参见[保存Notebook镜像环境](#)。
3. 已有的镜像调试成功后，再[使用ModelArts训练模块训练作业](#)。

4.2.6.7 Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
2. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
 - 创建方式：选择“自定义算法”
 - 启动方式：选择“自定义”
 - 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/mindspore:2.1.1-cann6.3.RC2”
 - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/ascend/demo-code/”
 - 启动命令：“python \${MA_JOB_DIR}/demo-code/run_ascend/run_ascend.py python \${MA_JOB_DIR}/demo-code/mindspore-verification.py”
 - 资源池：选择专属资源池
 - 类型：选择驱动/固件版本匹配的专属资源池Ascend规格。
 - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/ascend/log/”
3. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
4. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。
训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如[图4-13](#)所示。

图 4-13 专属资源池 Ascend 规格运行日志信息

```
75 Ascend Envs
76 -----
77 JOB_ID: modelarts-job-2436291a-8543-4ab8-84ad-2dda8f1e4f5c
78 RANK_TABLE_FILE: /home/ma-user/modelarts/rank_table/jobstart_hcc1.json
79 RANK_SIZE: 1
80 ASCEND_DEVICE_ID: 0
81 DEVICE_ID: 0
82 RANK_ID: 0
83 -----
84 [2. 2. 2. 2.]
85 [2. 2. 2. 2.]]
86
87 [[2. 2. 2. 2.]
88 [2. 2. 2. 2.]
89 [2. 2. 2. 2.]]
90
91 [[2. 2. 2. 2.]
92 [2. 2. 2. 2.]
93 [2. 2. 2. 2.]]]]
```

4.3 准备训练镜像

4.3.1 训练作业自定义镜像规范

针对您本地开发的模型及训练脚本，在制作镜像时，需满足ModelArts定义的规范。

规范要求

- 推荐自定义镜像使用ubuntu-18.04的操作系统，避免出现版本不兼容的问题。
- 自定义镜像的大小推荐15GB以内，最大不要超过资源池的容器引擎空间大小的一半。镜像过大会直接影响训练作业的启动时间。

ModelArts公共资源池的容器引擎空间为50G，专属资源池的容器引擎空间的默认为50G，支持在创建专属资源池时自定义容器引擎空间。

- 自定义镜像的默认用户必须为“uid”为“1000”的用户。
- 自定义镜像中不能安装GPU或Ascend驱动程序。当用户选择GPU资源运行训练作业时，ModelArts后台自动将GPU驱动程序放置在训练环境中的 /usr/local/nvidia 目录；当用户选择Ascend资源运行训练作业时，ModelArts后台自动将Ascend驱动程序放置在/usr/local/Ascend/driver目录。
- X86 CPU架构，ARM CPU架构的自定义镜像分别只能运行于对应CPU架构的规格中。

- 执行如下命令查看自定义镜像的CPU架构
docker inspect {自定义镜像地址} | grep Architecture

ARM CPU架构的自定义镜像，上述命令回显示意如下
"Architecture": "arm64"

- 规格中带有ARM字样的显示，为ARM CPU架构。

★ 规格 Ascend: 1*Ascend 910(32GB) | ARM: 24 核 96GB 3200GB

- 规格中未带有ARM字样的显示，为X86 CPU架构。

* 规格

GPU: 1*NVIDIA-V100(32GB) | CPU: 8 核 64GB 3200GB

- ModelArts后台暂不支持下载开源安装包，建议用户在自定义镜像中安装训练所需的依赖包。

4.3.2 已有镜像如何适配迁移至 ModelArts 训练平台

已有镜像迁移至训练管理需要关注如下步骤。

1. 为镜像增加训练管理的默认用户组ma-group，“gid = 100”。

📖 说明

如果已存在“gid = 100”用户组，可能会报错“groupadd: GID '100' already exists”。可通过命令“cat /etc/group | grep 100”查询是否已存在gid = 100用户组。

如果已存在“gid = 100”用户组，则该步骤跳过，下文Dockerfile中删除“RUN groupadd ma-group -g 100”命令。

2. 为镜像增加训练管理的默认用户ma-user，“uid = 1000”。

📖 说明

如果已存在“uid = 1000”用户，可能会报错“useradd: UID 1000 is not unique”。可通过命令“cat /etc/passwd | grep 1000”查询是否已存在uid = 1000用户。

如果已存在“uid = 1000”用户，则该步骤跳过，下文Dockerfile中删除“RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user”命令。

3. 修改镜像中相关文件权限，使得ma-user，“uid = 1000”用户可读写。

您可以参考如下Dockerfile，修改已有镜像，使其符合新版训练管理自定义镜像的规范。

```
FROM {已有镜像}

USER root

# 如果已存在 gid = 100 用户组，则删除 groupadd 命令。
RUN groupadd ma-group -g 100
# 如果已存在 uid = 1000 用户，则删除 useradd 命令。
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 修改镜像中相关文件权限，使得 ma-user, uid = 1000 用户可读写。
RUN chown -R ma-user:100 {Python软件包路径}

# 设置容器镜像预置环境变量。
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失。
ENV PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录。
USER ma-user
WORKDIR /home/ma-user
```

编写好Dockerfile后，通过执行如下所示命令进行新镜像构建。

```
docker build -f Dockerfile . -t {新镜像}
```

构建成功后将新镜像上传至SWR（参考[如何登录并上传镜像到SWR](#)）。

📖 说明

上述内容为关键代码样例，为了方便理解迁移过程，推荐您体验完整迁移案例：[示例：从0到1制作自定义镜像并用于训练（PyTorch+CPU/GPU）](#)。

4.3.3 使用基础镜像构建新的训练镜像

ModelArts平台提供了Tensorflow, PyTorch, MindSpore等常用深度学习任务的基础镜像，镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时，您可以基于这些基础镜像制作一个新的镜像并进行训练。

基于训练基础镜像构建新镜像的操作步骤

您可以参考如下步骤基于训练基础镜像来构建新镜像。

1. 安装Docker。如果docker images命令可以执行成功，表示Docker已安装，此步骤可跳过。

以linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

2. 准备名为context的文件夹。

```
mkdir -p context
```

3. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

📖 说明

在华为开源镜像站<https://mirrors.huaweicloud.com/home>中，搜索pypi，可以查看pip.conf文件内容。

4. 参考如下Dockerfile文件内容来基于ModelArts提供的训练基础镜像来构建一个新镜像。将编写好的Dockerfile文件放置在context文件夹内。训练基础镜像地址请参见[训练基础镜像列表](#)。

```
FROM {ModelArts提供的训练基础镜像地址}

# 配置pip
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:ma-group pip.conf /home/ma-user/.pip/pip.conf

# 设置容器镜像预置环境变量
# 将python解释器路径加入到PATH环境变量中
# 请务必设置PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
    PYTHONUNBUFFERED=1

RUN /home/ma-user/anaconda/bin/pip install --no-cache-dir numpy
```

5. 构建新镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像training:v1。

```
docker build . -t training:v1
```

6. 将构建好的新镜像上传至SWR（参考[如何登录并上传镜像到SWR](#)）。
7. 参考[使用自定义镜像创建训练作业（CPU/GPU）](#)章节在ModelArts上使用。

4.3.4 在容器镜像中安装 MLNX_OFED

场景描述

ModelArts GPU服务器上配置了Mellanox Technologies网卡，支持RDMA（Remote Direct Memory Access）。因此可以在容器镜像中安装MLNX_OFED，使得NCCL可以启用该网卡，提高跨节点通信效率。

NCCL启用该网卡后，跨节点通信采用的方法为NET/IB。未启用该网卡时，跨节点通信采用的方法为NET/Socket。NET/IB在时延与带宽方面都要优于NET/Socket。

表 4-7 ModelArts GPU 服务器 Mellanox Technologies 网卡和 MLNX_OFED 安装情况

| 服务器GPU型号 | Mellanox Technologies 网卡 | 服务器安装的 MLNX_OFED 版本 | 推荐容器镜像安装的MLNX_OFED版本 |
|---------------|--------------------------|-------------------------|----------------------|
| Vnt1 | ConnectX-5 | 4.3-1.0.1.0/4.5-1.0.1.0 | 4.9-6.0.6.0-LTS |
| Ant8/ Ant1 | ConnectX-6 Dx | 5.5-1.0.3.2 | 5.8-2.0.3.0-LTS |

安装 MLNX_OFED

以Ubuntu18.04的容器镜像为例，安装MLNX_OFED 4.9-6.0.6.0-LTS的Dockerfile示例如下。

说明

Dockerfile中涉及文件下载，构建容器镜像的主机要求能够连通公网。

```
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
apt-get update && \
apt-get install --no-install-recommends -y lsb-core curl && \
curl -k -o /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz https://content.mellanox.com/ofed/MLNX_OFED-4.9-6.0.6.0/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz && \
cd /tmp && \
tar xzf MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz && \
cd MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64 && \
./mlnxofedinstall --user-space-only --without-fw-update --without-neohost-backend --force && \
rm /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz && \
rm -rf /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64 && \
apt-get clean && \
mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
rm /etc/apt/apt.conf.d/00skip-verify-peer.conf
```

构建容器镜像命令示例如下：

```
docker build -f Dockerfile . -t nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-ubuntu18.04
```

构建完成后，执行如下命令可以查询容器镜像中的MLNX_OFED版本。

```
docker run -ti --rm nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-ubuntu18.04 ofed_info | head -n 1
```

命令回显示例如下：

```
MLNX_OFED_LINUX-4.9-6.0.6.0 (OFED-4.9-6.0.6):
```

4.4 使用自定义镜像创建算法

针对您在本地或使用其他工具开发的算法，支持上传至ModelArts中统一管理。

创建算法入口

在ModelArts上基于自定义镜像创建算法有2个入口：

- 入口1：在ModelArts控制台“算法管理 > 我的算法”入口，此处创建的算法可以在创建训练作业时直接使用，并且可以发布算法到AI Gallery。
- 入口2：在ModelArts控制台“训练管理 > 训练作业 > 创建训练作业”时，直接创建自定义算法，并提交训练作业。具体参见[使用自定义镜像创建训练作业（CPU/GPU）](#)。

创建算法参数设置

表 4-8 创建算法参数说明

| 参数 | 说明 |
|------|---|
| 启动方式 | 必选，选择“自定义”。 |
| 镜像 | <p>必选。容器镜像地址。</p> <ul style="list-style-type: none"> • 自有镜像或他人共享的镜像：单击右边的“选择”，可以从SWR服务选择用户的容器镜像，前提是要先上传镜像到SWR中，操作指导可参见如何登录并上传镜像到SWR。 • 公开镜像：支持手动输入SWR上的公开镜像地址（<用户镜像所属组织>/<镜像名称>），地址上不需要带域名信息（swr.<region>.xxx.com），系统会自动拼接域名地址。例如： modelarts-job-dev-image/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-euleros_2.10.1-x86_64-8.1.1 |
| 代码目录 | <p>可选，训练代码存储的OBS路径。</p> <p>以选择了OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录为例，OBS路径下的内容会被自动下载至训练容器的“\${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。</p> |

| 参数 | 说明 |
|------|---|
| 启动命令 | <p>必选，镜像的启动命令。在代码目录下载完成后，启动命令会被自动执行。</p> <ul style="list-style-type: none"> 如果训练启动脚本用的是py文件，例如train.py，启动命令可以写为<code>python \${MA_JOB_DIR}/demo-code/train.py</code>。 如果训练启动脚本用的是sh文件，例如main.sh，启动命令可以写为<code>bash \${MA_JOB_DIR}/demo-code/main.sh</code>。 <p>启动命令可支持使用“;”和“&&”拼接多条命令，但暂不支持换行拼接。命令中的demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。</p> |

输入输出管道设置

训练过程中，基于预置框架的算法需要从OBS桶或者数据集中获取数据进行模型训练，训练产生的输出结果也需要存储至OBS桶中。用户的算法代码中需解析输入输出参数实现ModelArts后台与OBS的数据交互，用户可以参考[开发自定义脚本](#)完成适配ModelArts训练的代码开发。

创建基于预置框架的算法时，用户需要配置算法代码中定义的输入输出参数。

- 输入配置

表 4-9 输入配置

| 参数 | 参数说明 |
|------|--|
| 参数名称 | <p>根据实际代码中的输入数据参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输入数据参数保持一致，否则您的算法代码无法获取正确的输入数据。</p> <p>例如，算法代码中使用argparse解析的data_url作为输入数据的参数，那么创建算法时就需要配置输入数据的参数名称为“data_url”。</p> |
| 描述 | 输入参数的说明，用户可以自定义描述。 |
| 获取方式 | 输入参数的获取方式，默认使用“超参”，也可以选择“环境变量”。 |

| 参数 | 参数说明 |
|------|---|
| 输入约束 | <p>开启后，用户可以根据实际情况限制数据输入来源。输入来源可以选择“数据存储位置”或者“ModelArts数据集”。</p> <p>如果用户选择数据来源为ModelArts数据集，还可以约束以下三种：</p> <ul style="list-style-type: none"> 标注类型。数据类型请参考标注数据。 数据格式。可选“Default”和“CarbonData”，支持多选。其中“Default”代表Manifest格式。 数据切分。仅“图像分类”、“物体检测”、“文本分类”和“声音分类”类型数据集支持进行数据切分功能。可选“仅支持切分的数据集”、“仅支持未切分数据集”和“无限制”。数据切分详细内容可参考发布数据版本。 |
| 添加 | 用户可以根据实际算法添加多个输入数据来源。 |

- 输出配置

表 4-10 输出配置

| 参数 | 参数说明 |
|------|--|
| 参数名称 | <p>根据实际代码中的训练输出参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输出参数保持一致，否则您的算法代码无法获取正确的输出路径。</p> <p>例如，算法代码中使用argparse解析的train_url作为训练输出数据的参数，那么创建算法时就需要配置输出数据的参数名称为“train_url”。</p> |
| 描述 | 输出参数的说明，用户可以自定义描述。 |
| 获取方式 | 输出参数的获取方式，默认使用“超参”，也可以选择“环境变量”。 |
| 添加 | 用户可以根据实际算法添加多个输出数据路径。 |

定义超参

使用预置框架创建算法时，ModelArts支持用户自定义超参，方便用户查阅或修改。定义超参后会体现在启动命令中，以命令行参数的形式传入您的启动文件中。

1. 导入超参

您可以单击“增加超参”手动添加超参。

图 4-14 添加超参



2. 编辑超参

超参的参数说明参见[表4-11](#)。

表 4-11 超参编辑参数

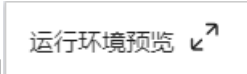
| 参数 | 说明 |
|-----|---|
| 名称 | 填入超参名称。 超参名称支持64个以内字符，仅支持大小写字母、数字、下划线和中划线。 |
| 类型 | 填入超参的数据类型。支持String、Integer、Float和Boolean。 |
| 默认值 | 填入超参的默认值。创建训练作业时，默认使用该值进行训练。 |
| 约束 | 单击“约束”。在弹出对话框中，支持用户设置默认值的取值范围或者枚举值范围。 |
| 必需 | 选择是或否。 <ul style="list-style-type: none">选择否，则在使用该算法创建训练作业时，支持在创建训练作业页面删除该超参。选择是，则在使用该算法创建训练作业时，不支持在创建训练作业页面删除该超参。 |
| 描述 | 填入超参的描述说明。 超参描述支持大小写字母、中文、数字、空格、中划线、下划线、中英文逗号和中英文句号。 |

添加训练约束

用户可以根据实际情况定义此算法的训练约束。

- 资源类型：选择适用的资源类型，支持多选。
- 多卡训练：选择是否支持多卡训练。
- 分布式训练：选择是否支持分布式训练。

运行环境预览

创建算法时，可以打开创建页面右下方的运行环境预览窗口 ，辅助您了解代码目录、启动文件、输入输出等数据配置在训练容器中的路径。

后续操作

创建算法完成后，可以使用算法快速创建训练作业，详细操作请参见[使用自定义镜像创建训练作业（CPU/GPU）](#)。

也可以将新创建的算法[发布到AI Gallery](#)，分享给其他用户使用。

4.5 使用自定义镜像创建训练作业（CPU/GPU）

模型训练是一个不断迭代和优化的过程。在训练模块的统一管理下，方便用户试验算法、数据和超参数的各种组合，便于追踪最佳的模型与输入配置，您可以通过不同版本间的评估指标比较，确定最佳训练作业。

前提条件

- 已将用于训练的数据上传至OBS目录。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 已按照ModelArts规范制作自定义镜像包，自定义镜像包规范请参见[训练作业自定义镜像规范](#)。
- 已将自定义镜像上传至SWR服务，操作指导可参见[如何登录并上传镜像到SWR](#)。

创建训练作业

1. 登录ModelArts管理控制台，在左侧导航栏中选择“训练管理 > 训练作业”，进入“训练作业”列表。
2. 单击“创建训练作业”，进入创建训练作业页面，填写训练作业相关参数。

表 4-12 创建训练作业的创建方式（使用自定义镜像）

| 参数名称 | 说明 |
|------|---|
| 创建方式 | 必选，选择“自定义算法”。 |
| 启动方式 | 必选，选择“自定义”。 |
| 镜像 | <p>必填，填写容器镜像的地址。</p> <p>容器镜像地址的填写支持如下方式。</p> <ul style="list-style-type: none"> • 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 • 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息（swr.<region>.xxx.com），系统会自动拼接域名地址。例如：某公开镜像的SWR地址为“swr.<region>.xxx.com/test-image/tensorflow2_1_1:1.1.1”，则此处填写“test-images/tensorflow2_1_1:1.1.1”。 |

| 参数名称 | 说明 |
|--------|---|
| 代码目录 | <p>选择训练代码文件所在的OBS目录。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。</p> <ul style="list-style-type: none"> 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“<code>\${MA_JOB_DIR}/demo-code</code>”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“<code>\${MA_JOB_DIR}/code</code>”目录中。 |
| 运行用户ID | <p>容器运行时的用户ID，该参数为选填参数，建议使用默认值1000。</p> <p>如果需要指定uid，则uid数值需要在规定范围内，不同资源池的uid范围如下：</p> <ul style="list-style-type: none"> 公共资源池：1000-65535 专属资源池：0-65535 |
| 启动命令 | <p>必填，镜像的启动命令。</p> <p>运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。</p> <ul style="list-style-type: none"> 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。 <code>python \${MA_JOB_DIR}/demo-code/train.py</code> 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。 <code>bash \${MA_JOB_DIR}/demo-code/main.sh</code> <p>启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。</p> |
| 本地代码目录 | <p>指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录。</p> <p>此参数可选，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。</p> |
| 工作目录 | <p>训练时，系统会自动cd到此目录下执行启动文件。</p> |

表 4-13 创建训练作业的训练参数

| 参数名称 | 子参数 | 说明 |
|------|--------|--|
| 输入 | 参数名称 | <p>算法代码需要通过“输入”的“参数名称”去读取训练的输入数据。</p> <p>建议设置为“data_url”，和训练代码中解析输入数据的参数保持一致。此处可以设置多条训练输入参数。训练输入参数名称不可以重名。</p> <p>例如，训练代码中使用argparse解析data_url为输入数据超参，则在此处需要配置输入数据代码参数名称为data_url。</p> <pre>import argparse # 创建解析 parser = argparse.ArgumentParser(description="train mnist", formatter_class=argparse.ArgumentDefaultsHelpFormatter) # 添加参数 parser.add_argument('--train_url', type=str, help='the path model saved') parser.add_argument('--data_url', type=str, help='the training data') # 解析参数 args, unknown = parser.parse_known_args()</pre> |
| | 数据集 | <p>单击“数据集”，在ModelArts数据集列表中勾选目标数据集并选择对应的版本。</p> <p>训练启动时，系统将自动下载输入路径中的数据到训练运行容器。</p> <p>说明 ModelArts数据管理模块在重构升级中，对未使用过数据管理的用户不可见。建议新用户将训练数据存放至OBS桶中使用。</p> |
| | 数据存储位置 | <p>单击“数据存储位置”，从OBS桶中选择训练输入数据的存储位置。</p> <p>训练启动时，系统将自动下载输入路径中的数据到训练运行容器。</p> |
| | 获取方式 | <p>以参数名称为“data_path”的训练输入为例，说明获取方式的作用。</p> <ul style="list-style-type: none"> 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 <pre>import argparse parser = argparse.ArgumentParser() parser.add_argument('--data_path') args, unknown = parser.parse_known_args() data_path = args.data_path</pre> 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 <pre>import os data_path = os.getenv("data_path", "")</pre> |
| 输出 | 参数名称 | <p>算法代码需要通过“输出”的“参数名称”去读取训练的输出目录。</p> <p>建议设置为“train_url”，和训练代码中解析输出数据的参数保持一致。此处也可以设置多条训练输出参数。训练输出参数名称不可以重名。</p> |

| 参数名称 | 子参数 | 说明 |
|--------|----------|---|
| | 数据存储位置 | <p>单击“数据存储位置”，从OBS桶中选择训练输出数据的存储位置。训练过程中，系统将自动从训练容器的本地代码目录下同步文件到数据存储位置。</p> <p>说明 数据存储位置仅支持OBS路径。为避免数据存储冲突，建议选择一个空目录用作“数据存储位置”。</p> |
| | 获取方式 | <p>以参数名称为“train_url”的训练输出为例，说明获取方式的作用。</p> <ul style="list-style-type: none"> 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 <pre>import argparse parser = argparse.ArgumentParser() parser.add_argument('--train_url') args, unknown = parser.parse_known_args() train_url = args.train_url</pre> 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 <pre>import os train_url = os.getenv("train_url", "")</pre> |
| | 预下载至本地目录 | <p>选择是否将输出目录下的文件预下载至本地目录。</p> <ul style="list-style-type: none"> 不下载：表示启动训练作业时不会将输出数据的存储位置中的文件下载到训练容器的本地代码目录中。 下载：表示系统会在启动训练作业时自动将输出数据的存储位置中的所有文件下载到训练容器的本地代码目录中。下载时间会随着文件变大而变长，为了防止训练时间过长，请及时清理训练容器的本地代码目录中的无用文件。如果要使用断点续训练和增量训练，则必须选择“下载”。 |
| 超参 | - | 可选，超参用于训练调优。 |
| 环境变量 | - | 根据业务需求增加环境变量。训练容器中预置的环境变量请参见 查看训练容器环境变量 。 |
| 故障自动重启 | - | <p>打开开关后，可以设置故障的重启次数，训练作业失败时会自动重新下发并运行训练作业。在训练作业详情页中，用户可以查看训练作业发生故障后的重启次数。</p> <ul style="list-style-type: none"> 此参数默认关闭。 打开故障自动重启后，需要设置重启次数，重启次数取值范围1~3次，重启次数设置后无法修改。 |

- 选择训练资源的规格。训练参数的可选范围与已有自定义镜像的使用约束保持一致。根据需要选择公共资源池或专属资源池，参数说明请参[创建训练作业](#)。
- 单击“提交”，完成训练作业的创作。
训练作业一般需要运行一段时间。
要查看训练作业实时情况，您可以前往训练作业列表，单击训练作业的名称，进入训练作业详情页，查看训练作业的基本情况，具体请参考[查看作业详情](#)。

4.6 使用自定义镜像创建训练作业（Ascend）

如果Ascend-Powered-Engine预置镜像无法满足您的需求，您可以构建一个自定义镜像，通过自定义镜像创建训练作业。Ascend自定义镜像训练作业创建流程与CPU/GPU一致，但是需要额外关注：

- Ascend HCCL RANK_TABLE_FILE文件说明

Ascend HCCL RANK_TABLE_FILE文件提供Ascend分布式训练作业的集群信息，用于Ascend芯片分布式通信，可以被HCCL集合通信库解析。该文件格式有两个版本，分别为模板一、模板二。当前ModelArts提供的是模板二格式。完整的Ascend HCCL RANK_TABLE_FILE文件内容请参见[昇腾AI处理器资源配置文件](#)。

ModelArts训练环境的Ascend HCCL RANK_TABLE_FILE文件名为jobstart_hccl.json。获取方式参考[表4-14](#)。

- ModelArts训练环境jobstart_hccl.json文件内容（模板二）示例

```
{
  "group_count": "1",
  "group_list": [{
    "device_count": "1",
    "group_name": "job-trainjob",
    "instance_count": "1",
    "instance_list": [{
      "devices": [{
        "device_id": "4",
        "device_ip": "192.1.10.254"
      }],
      "pod_name": "jobxxxxxxx-job-trainjob-0",
      "server_id": "192.168.0.25"
    }
  ]
},
  "status": "completed"
}
```

jobstart_hccl.json文件中的status字段的值在训练脚本启动时，并不一定为completed状态。因此需要训练脚本等待status字段的值等于completed之后，再去读取文件的剩余内容。

如果算法开发者期望使用模板一格式的jobstart_hccl.json文件，可以使用训练脚本，在等待status字段的值等于completed之后，将模板二格式jobstart_hccl.json文件转换为模板一格式的jobstart_hccl.json文件。

- 转换后的jobstart_hccl.json文件格式（模板一）示例

```
{
  "server_count": "1",
  "server_list": [{
    "device": [{
      "device_id": "4",
      "device_ip": "192.1.10.254",
      "rank_id": "0"
    }],
    "server_id": "192.168.0.25"
  ]
},
  "status": "completed",
  "version": "1.0"
}
```

📖 说明

转换功能的实现，可参考[示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore + Ascend）](#)中所述的Ascend训练脚本的启动脚本。

- RANK_TABLE_FILE环境变量

表 4-14 RANK_TABLE_FILE 环境变量说明

| 环境变量 | 说明 |
|-----------------|--|
| RANK_TABLE_FILE | 该环境变量指示Ascend HCCL RANK_TABLE_FILE文件所在目录，值为/user/config。 算法开发者可通过 “\${RANK_TABLE_FILE}/jobstart_hccl.json”，路径获取该文件。 |

4.7 自定义镜像训练作业失败定位思路

问题现象

使用自定义镜像训练作业时，训练失败。

定位思路

1. 确定镜像来源

- 确认该自定义镜像的基础镜像是否来源于ModelArts提供的基础镜像，推荐用户使用ModelArts的基础镜像构建自定义镜像，具体请参见[使用ModelArts的基础镜像构建新的训练镜像](#)。
- 如镜像来源于第三方，设法找到自定义镜像的制作者咨询，制作者一般对镜像如何使用更加了解。

2. 确定自定义镜像大小

自定义镜像的大小推荐15GB以内，最大不要超过资源池的容器引擎空间大小的一半。镜像过大会直接影响训练作业的启动时间。

ModelArts公共资源池的容器引擎空间为50G，专属资源池的容器引擎空间的默认为50G，支持在创建专属资源池时自定义容器引擎空间。

3. 确定错误类型

- 提示找不到文件等错误，请参见[训练作业日志中提示 “No such file or directory”](#)。
- 提示找不到包等错误，请参见[训练作业日志中提示 “No module named .*”](#)。
- Ascend启动脚本和初始化脚本问题。
确认相关脚本是否来源于官方文档并且是否严格按照官方文档使用。比如确认脚本名称是否正常、脚本路径是否正常。具体请参见[示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+Ascend）](#)。
- 驱动版本与底层驱动不兼容
当对自定义镜像的驱动进行升级时，请确定底层驱动是否兼容。当前支持哪种驱动版本，请从[基础镜像](#)中获取。
- 文件权限不足

该问题可能为自定义镜像的用户与作业容器的用户不同导致的。请修改dockerfile文件：

```
RUN if id -u ma-user > /dev/null 2>&1 ; \
then echo 'MA 用户已存在' ; \
else echo 'MA 用户不存在' && \
groupadd ma-group -g 1000 && \
```

```
useradd -d /home/ma-user -m -u 1000 -g 1000 -s /bin/bash ma-user ; fi && \  
chmod 770 /home/ma-user && \  
chmod 770 /root && \  
usermod -a -G root ma-user
```

- 其他现象，可以在已有的[训练故障案例](#)查找。

建议与总结

用户使用自定义镜像训练作业时，建议按照[训练作业自定义镜像规范](#)制作镜像。文档中同时提供了端到端的示例供用户参考。

5 使用自定义镜像创建 AI 应用（推理部署）

5.1 创建 AI 应用的自定义镜像规范

针对您本地开发的模型，在制作AI应用的自定义镜像时，需满足ModelArts定义的规范。

- 自定义镜像中不能包含恶意代码。
- 自定义镜像大小不超过50GB。
- 对于同步请求模式的AI应用，如果预测请求时延超过60s，会造成请求失败，甚至会有服务业务中断的风险，预测请求时延超过60s时，建议制作异步请求模式的镜像。

- **镜像对外接口**

设置镜像的对外服务接口，推理接口需与config.json文件中apis定义的url一致，当镜像启动时可以直接访问。下面是mnist镜像的访问示例，该镜像内含mnist数据集训练的模型，可以识别手写数字。其中listen_ip为容器IP，您可以通过启动自定义镜像，在容器中获取容器IP。

- 请求示例

```
curl -X POST \ http://{listen_ip}:8080/ \ -F images=@seven.jpg
```

图 5-1 listen_ip 获取示例

```
root@169.254.30.2:~# cat /etc/hosts
127.0.0.1    localhost
::1         localhost ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
169.254.30.2  d6211431d0e3
```

- 返回示例

```
{"mnist_result": 7}
```

- **（可选）健康检查接口**

如果在滚动升级时要求不中断业务，那么必须在config.json文件中配置健康检查的接口，供ModelArts调用，在config.json文件中配置。当业务可提供正常服务时，健康检查接口返回健康状态，否则返回异常状态。

须知

- 如果要实现无损滚动升级，必须配置健康检查接口。
- 自定义镜像如果需要在“在线服务”模块使用OBS外部存储挂载功能，需要新建一个OBS挂载专属目录如“/obs-mount/”，避免选择存量目录覆盖已有文件。OBS挂载仅开放对挂载目录文件新增、查看、修改功能不支持删除挂载目录文件对象，若需要删除文件请到OBS并行文件系统中手动删除。

健康检查接口示例如下。

- URI
GET /health
- 请求示例 `curl -X GET \ http://{listen_ip}:8080/health`
- 响应示例
`{"health": "true"}`
- 状态码

表 5-1 状态码

| 状态码 | 编码 | 状态码说明 |
|-----|----|-------|
| 200 | OK | 请求成功 |

- **日志文件输出**

为保证日志内容可以正常显示，日志信息需要打印到标准输出。

- **镜像启动入口**

如果需要部署批量服务，镜像的启动入口文件需要为“/home/run.sh”，采用CMD设置默认启动路径，例如Dockerfile如下：

CMD ["sh", "/home/run.sh"]

- **镜像依赖组件**

如果需要部署批量服务，镜像内需要安装python、jre/jdk、zip等组件包。

- **（可选）保持Http长链接，无损滚动升级**

如果需要支持滚动升级的过程中不中断业务，那么需要将服务的Http的“keep-alive”参数设置为200s。以gunicorn服务框架为例，gunicorn缺省情形下不支持keep-alive，需要同时安装gevent并配置启动参数“--keep-alive 200 -k gevent”。不同服务框架参数设置有区别，请以实际情况为准。

- **（可选）处理SIGTERM信号，容器优雅退出**

如果需要支持滚动升级的过程中不中断业务，那么需要在容器中捕获SIGTERM信号，并且在收到SIGTERM信号之后等待60秒再优雅退出容器。提前优雅退出容器可能会导致在滚动升级的过程中业务概率中断。要保证容器优雅退出，从收到SIGTERM信号开始，业务需要将收到的请求全部处理完毕再结束，这个处理时长最多不超过90秒。例如run.sh如下所示：

```
#!/bin/bash
gunicorn_pid=""

handle_sigterm() {
  echo "Received SIGTERM, send SIGTERM to $gunicorn_pid"
  if [ $gunicorn_pid != "" ]; then
    sleep 60
    kill -15 $gunicorn_pid # 传递 SIGTERM 给gunicorn进程
```



```
wait $gunicorn_pid      # 等待gunicorn进程完全终止
fi
}
trap handle_sigterm TERM
```

5.2 从 0-1 制作自定义镜像并创建 AI 应用

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为AI应用。本文详细介绍如何使用自定义镜像完成AI应用的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考[创建AI应用的自定义镜像规范](#)。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。
3. **将自定义镜像创建为AI应用**：将上传至SWR服务的镜像导入ModelArts的AI应用管理。
4. **将AI应用部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考[购买并登录弹性云服务器](#)。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 5-2 创建 ECS 服务器-选择 X86 架构的公共镜像



1. 登录主机后，安装Docker，可参考[Docker官方文档](#)。也可执行以下命令安装docker。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```
2. 获取基础镜像。本示例以Ubuntu18.04为例。

```
docker pull ubuntu:18.04
```
3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。
文件结构如下所示

```
self-define-images/
--Dockerfile
--test_app.py

- “Dockerfile”
From ubuntu:18.04
# 配置华为云的源, 安装 python、python3-pip 和 Flask
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
    apt-get update && \
    apt-get install -y python3 python3-pip && \
    pip3 install --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
repository/pypi/simple Flask

# 复制应用服务代码进镜像里面
COPY test_app.py /opt/test_app.py

# 指定镜像的启动命令
CMD python3 /opt/test_app.py

- “test_app.py”
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----- in hello func -----")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----- in goodbye func -----")
    return '\nGoodbye!\n'

@app.route('/', methods=['POST'])
def default_func():
    print("----- in default func -----")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. 进入 “self-define-images” 文件夹，执行以下命令构建自定义镜像 “test:v1” 。
docker build -t test:v1 .
5. 您可以使用 “docker images” 查看您构建的自定义镜像。

本地验证镜像并上传镜像至 SWR 服务

1. 在本地环境执行以下命令启动自定义镜像
docker run -it -p 8080:8080 test:v1

图 5-3 启动自定义镜像

```
:/opt/file# docker run -it -p 8080:8080 test:v1
* Serving Flask app "test_app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye

如果验证自定义镜像功能成功，结果如下图所示。

图 5-4 校验接口

```
root@~:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/
called default func !
{'name': 'Tom'}
root@~:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
{"response": "Hello, Tom!"}
root@~:~# curl -X GET 127.0.0.1:8080/goodbye
Goodbye!
```

3. 上传自定义镜像至SWR服务。上传镜像的详细操作可参考[如何登录并上传镜像到SWR](#)。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为 AI 应用

参考[从容器镜像中选择元模型](#)导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
 - 容器镜像所在的路径：选择已制作好的自有镜像

图 5-5 选择已制作好的自有镜像



- 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
- 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
- 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致AI应用创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见[模型配置文件说明](#)。

本样例的配置文件如下所示：

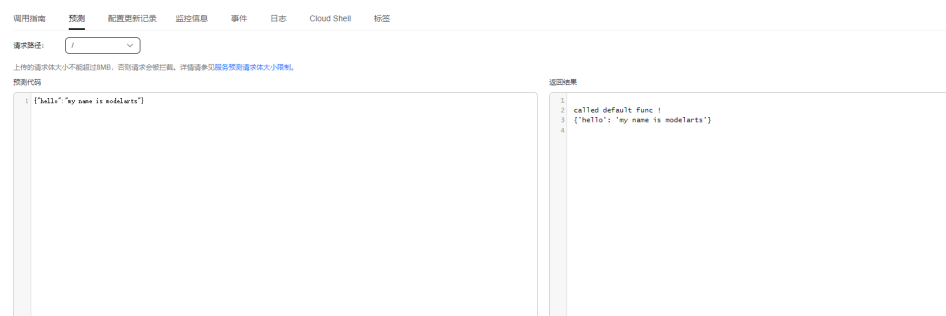
```
[[
  "url": "/",
  "method": "post",
  "request": {
```

```
    "Content-type": "application/json"
  },
  "response": {
    "Content-type": "application/json"
  }
},
{
  "url": "/greet",
  "method": "post",
  "request": {
    "Content-type": "application/json"
  },
  "response": {
    "Content-type": "application/json"
  }
},
{
  "url": "/goodbye",
  "method": "get",
  "request": {
    "Content-type": "application/json"
  },
  "response": {
    "Content-type": "application/json"
  }
}
]
```

将 AI 应用部署为在线服务

1. 参考[部署为在线服务](#)将AI应用部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。
3. 您可以通过“预测”页签访问在线服务。

图 5-6 访问在线服务



5.3 在开发环境中构建并调试推理镜像

5.3.1 场景说明

针对ModelArts目前不支持的AI引擎，您可以通过自定义镜像的方式将编写的模型导入ModelArts，创建为AI应用。

本文详细介绍如何在ModelArts的开发环境Notebook中使用基础镜像构建一个新的推理镜像，并完成AI应用的创建，部署为在线服务。本案例仅适用于华为云北京四和上海一站点。

操作流程如下：

1. **Step1 在Notebook中构建一个新镜像**：在ModelArts的开发环境Notebook中制作自定义镜像，镜像规范可参考[创建AI应用的自定义镜像规范](#)。
2. **Step2 构建成功的镜像注册到镜像管理模块**：将构建成功的自定义镜像注册到ModelArts的镜像管理模块中，方便下一步调试。
3. **Step3 在Notebook中变更镜像并调试**：在Notebook中调试镜像。
4. **Step4 使用调试成功的镜像用于推理部署**：将调试完成的镜像导入ModelArts的AI应用管理中，并部署上线。

5.3.2 Step1 在 Notebook 中构建一个新镜像

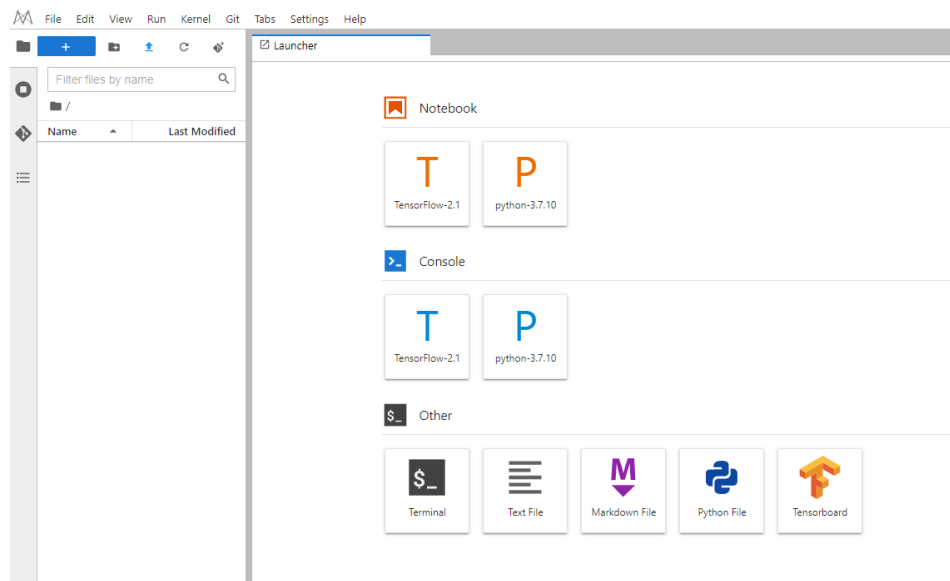
本章节以ModelArts提供的基础镜像tensorflow为例介绍如何在ModelArts的Notebook中构建一个新镜像并用于AI应用部署。

创建 Notebook 实例

1. 登录ModelArts控制台，在左侧导航栏中选择“全局配置”，检查是否配置了访问授权。若未配置，请先配置访问授权。参考[使用委托授权](#)完成操作。
2. 登录ModelArts控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。
3. 单击右上角“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
 - a. 填写Notebook基本信息，包含名称、描述、是否自动停止。
 - b. 填写Notebook详细参数，如选择镜像、资源规格等。
 - “镜像”：选择公共镜像下任意一个支持CPU类型的镜像，例如：**tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04**
 - “资源池”：选择公共资源池或专属资源池，此处以公共资源池为例。
 - “类型”：推荐选择GPU。
 - “规格”：推荐选择GP Tnt004规格，若没有再选择其他规格。
4. 参数填写完成后，单击“立即创建”进行规格确认。参数确认无误后，单击“提交”，完成Notebook的创建操作。

进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
5. 打开运行中的Notebook实例。

图 5-7 打开 Notebook 实例




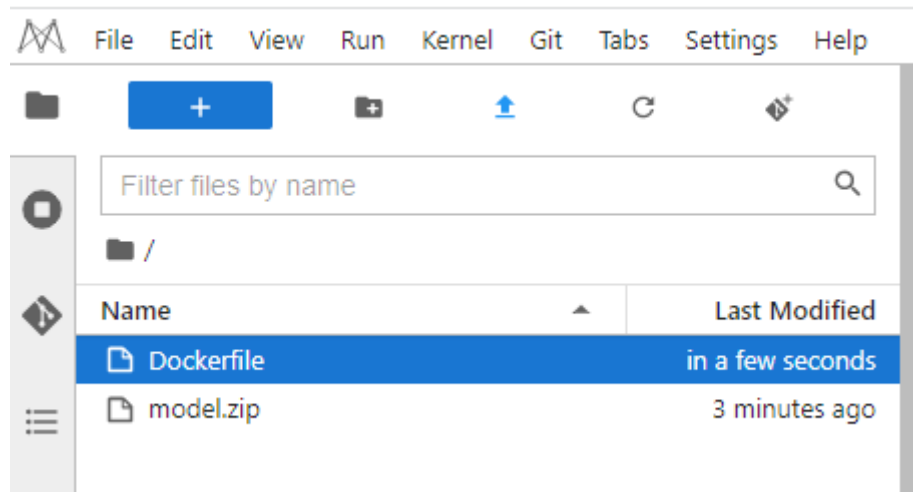
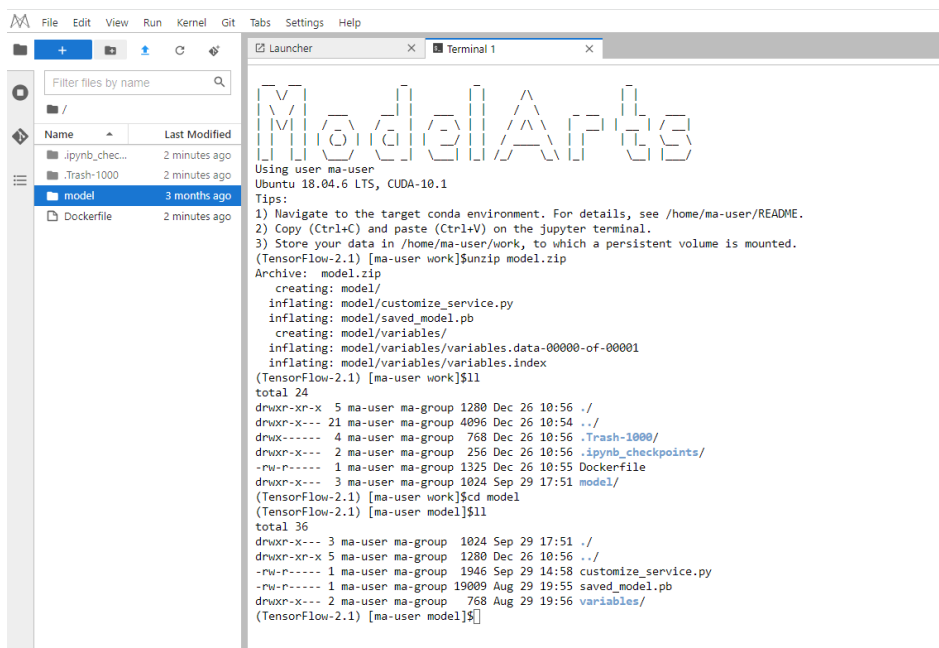
6. 通过  功能，上传dockerfile文件和模型包文件到Notebook中，默认工作目录/home/ma-user/work/。
dockerfile文件的具体内容可以参见[Dockerfile模板](#)。模型包文件需要用户自己准备，样例内容参见[模型包文件样例](#)。

图 5-8 上传 dockerfile 文件和模型包文件



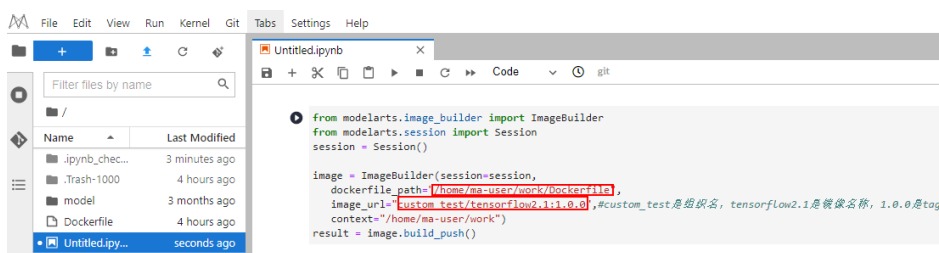
7. 打开Terminal终端，解压model.zip，解压后删除zip文件。
#解压命令
unzip model.zip

图 5-9 在 Terminal 终端中解压 model.zip



8. 打开一个新的.ipynb文件，启动构建脚本，在构建脚本中指定dockerfile文件和镜像的推送地址。构建脚本当前仅支持华为云北京四和上海一节点。

图 5-10 启动构建脚本



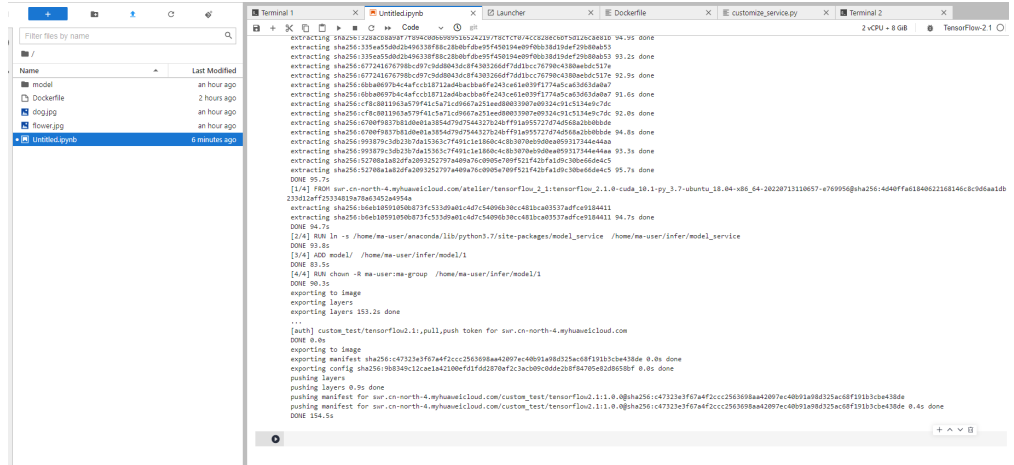
构建脚本内容如下：

```
from modelarts.image_builder import ImageBuilder
from modelarts.session import Session
session = Session()

image = ImageBuilder(session=session,
    dockerfile_path="/home/ma-user/work/Dockerfile",
    image_url="custom_test/tensorflow2.1:1.0.0",#custom_test是组织名，tensorflow2.1是镜像名称，1.0.0是tag
    context="/home/ma-user/work")
result = image.build_push()
```

等待镜像构建完成。镜像构建完成后会自动推送到SWR中。

图 5-11 等待镜像构建完成



Dockerfile 模板

Dockerfile 样例，此样例可以直接另存为一个 Dockerfile 文件使用。此处可以使用的基
础镜像列表请参见[推理基础镜像列表](#)。

```
FROM swr.cn-north-4.myhuaweicloud.com/atelier/tensorflow_2.1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
```

```
# here create a soft link from '/home/ma-user/anaconda/lib/python3.7/site-packages/model_service' to '/home/ma-user/infer/model_service'. It's the build-in inference framework code dir  
# if the installed python version of this base image is python3.8, you should create a soft link from '/home/ma-user/anaconda/lib/python3.8/site-packages/model_service' to '/home/ma-user/infer/model_service'.  
USER root  
RUN ln -s /home/ma-user/anaconda/lib/python3.7/site-packages/model_service /home/ma-user/infer/model_service  
USER ma-user
```

```
# here we supply a demo, you can change it to your own model files  
ADD model/ /home/ma-user/infer/model/1  
USER root  
RUN chown -R ma-user:ma-group /home/ma-user/infer/model/1  
USER ma-user
```

```
# default MODELARTS_SSL_CLIENT_VERIFY switch is "true". In order to debug, we set it to be "false"  
ENV MODELARTS_SSL_CLIENT_VERIFY="false"
```

```
# change your port and protocol here, default is 8443 and https  
# ENV MODELARTS_SERVICE_PORT=8080  
# ENV MODELARTS_SSL_ENABLED="false"
```

```
# add pip install here  
# RUN pip install numpy==1.16.4  
# RUN pip install -r requirements.txt
```

```
# default cmd, you can change it here  
# CMD sh /home/ma-user/infer/run.sh
```

模型包文件样例

模型包文件 model.zip 中需要用户自己准备模型文件，此处仅是举例示意说明，以一个
手写数字识别模型为例。

Model 目录下必须要包含推理脚本文件 customize_service.py，目的是为开发者提供模
型预处理和后处理的逻辑。

图 5-12 推理模型 model 目录示意图（需要用户自己准备模型文件）

| 名称 | 修改日期 | 类型 | 大小 |
|-------------------|-----------------|-------|--------|
| variables | 2022/7/22 10:30 | 文件夹 | |
| customize_service | 2022/7/22 10:32 | PY 文件 | 2 KB |
| saved_model | 2022/3/10 18:47 | PB 文件 | 105 KB |

推理脚本customize_service.py的具体写法要求可以参考[模型推理代码编写说明](#)。

本案例中提供的customize_service.py文件具体内容如下：

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tf-serving_model_service import TfServingBaseService

class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()

        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)

        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logging.warning("signatures more than one, use serving_default signature from %s", signature)
            model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        self.predict = self.model.signatures[model_signature]

    def _preprocess(self, data):
        images = []
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((28, 28, 1))
                images.append(image1)

        images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
        preprocessed_data = images

        return preprocessed_data
```

```
def _inference(self, data):  
    return self.predict(data)  
  
def _postprocess(self, data):  
    return {  
        "result": int(data["output"].numpy()[0].argmax())  
    }
```

5.3.3 Step2 构建成功的镜像注册到镜像管理模块

将**Step1 在Notebook中构建一个新镜像**中构建成功的自定义镜像注册到镜像管理中，方便后续使用。

1. 登录ModelArts控制台，在左侧导航栏中选择“镜像管理”，单击“注册镜像”，进入注册镜像页面。
2. 输入镜像源地址，选择架构和类型后，单击“立即注册”。
 - “镜像源”：地址为 `swr.cn-north-4-myhuaweicloud.com/custom_test/tensorflow2.1:1.0.0`。其中 `custom_test/tensorflow2.1:1.0.0` 为**8**镜像构建脚本中设置的镜像地址。
 - “架构”：选择 `X86_64`
 - “类型”：选择 `CPU`

图 5-13 注册镜像

The screenshot shows the 'Register Image' form in the ModelArts console. It includes the following fields and options:

- 镜像源**: A dropdown menu showing 'swr.cn' with a '查看可选镜像' (View Available Images) link. Below it is a placeholder text: '示例: <swr-domain-name>/<namespace>/<repository>:<tag>'.
- 描述**: A text input area with a character count of '0/256'.
- 架构**: Two radio buttons, 'X86_64' (selected) and 'ARM'.
- 类型**: Two radio buttons, 'CPU' (selected) and 'GPU'.

3. 注册完成后，可以在镜像管理页面查看到注册成功的镜像。

5.3.4 Step3 在 Notebook 中变更镜像并调试

使用制作完成的自定义镜像进行推理服务调试，调试成功后再导入到ModelArts的AI应用中并部署为在线服务。

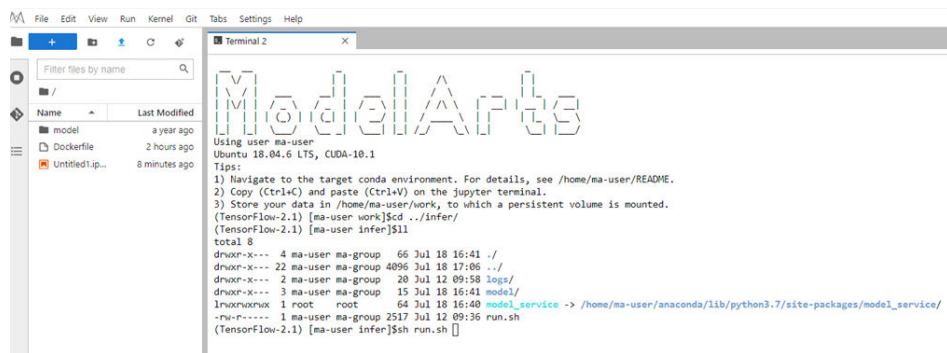
1. 登录ModelArts控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。停止在**Step1 在Notebook中构建一个新镜像**中创建的 Notebook。
2. 在Notebook对应操作列，单击“更多 > 变更镜像”，打开“变更镜像”弹出框，变更镜像选择“自定义镜像”，将当前镜像变更为**Step2 构建成功的镜像注册到镜像管理模块**注册的镜像，如**图5-14**所示。

图 5-14 变更镜像



3. 启动变更后的Notebook，并打开。进入Terminal运行界面，在工作目录，运行启动脚本run.sh，并预测模型。基础镜像中默认提供了run.sh作为启动脚本。

图 5-15 运行启动脚本

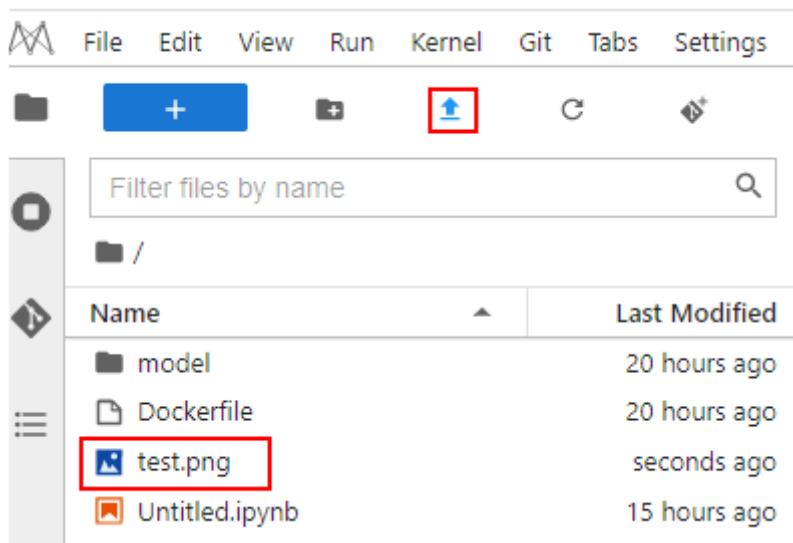


4. 上传一张预测图片（手写数字图片）到Notbook中。

图 5-16 手写数字图片

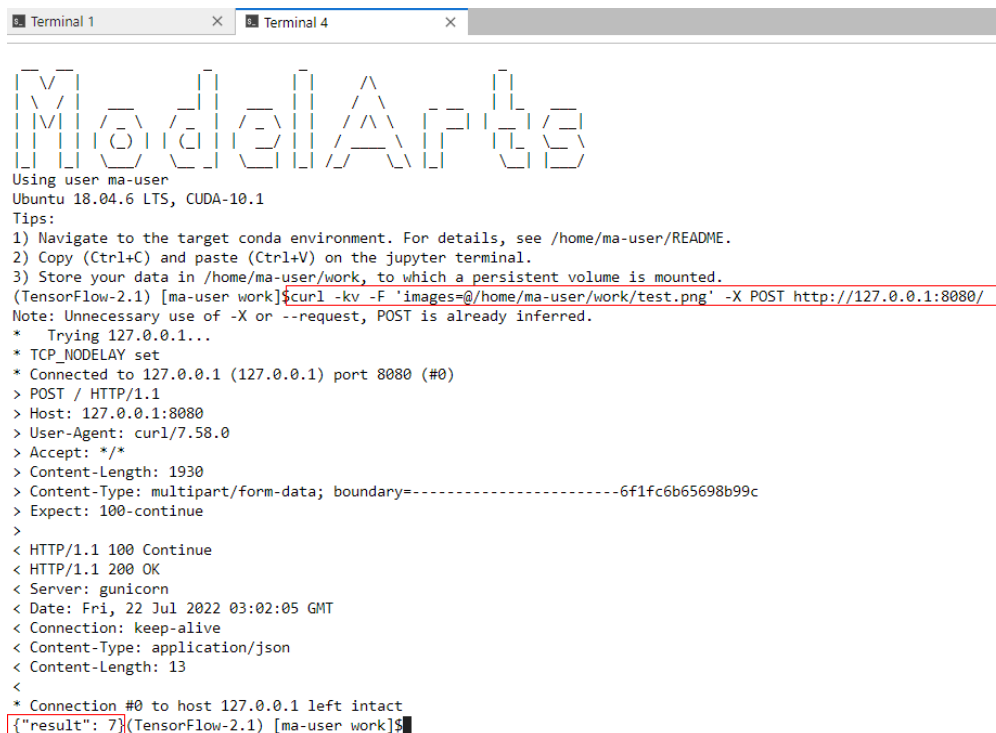


图 5-17 上传预测图片



5. 重新打开一个新的Terminal终端，执行如下命令进行预测。
`curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/`

图 5-18 预测



在调试过程中，如果有修改模型文件或者推理脚本文件，需要重启run.sh脚本。执行如下命令先停止nginx服务，再运行run.sh脚本。

```
#查询nginx进程
ps -ef |grep nginx
#关闭所有nginx相关进程
kill -9 {进程ID}
#运行run.sh脚本
sh run.sh
```

也可以执行kill nginx命令直接关闭所有nginx进程。

```
#关闭所有nginx进程
pkill nginx
#运行run.sh脚本
sh run.sh
```

图 5-19 重启 run.sh 脚本

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user  6474    1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user  6476    6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user  6477    6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user  7925   1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

5.3.5 Step4 使用调试成功的镜像用于推理部署

将Step3 在Notebook中变更镜像并调试中调试成功的自定义镜像导入到AI应用中，并部署为在线服务。


1. 登录ModelArts控制台，在左侧导航栏中选择“AI应用管理 > AI应用”，单击“创建”，进入创建AI应用。
2. 设置AI应用的参数，如图5-20所示。
 - 元模型来源：从容器镜像中选择。
 - 容器镜像所在的路径：单击  选择前面创建的镜像。
 - 容器调用接口：选择HTTPS。
 - host：设置为8443。
 - 部署类型：选择在线部署。

图 5-20 设置 AI 应用参数



3. 填写apis定义，单击“保存”生效。apis定义中指定输入为文件，具体内容参见下面代码样例。

图 5-21 填写 apis 定义



apis定义具体内容如下：

```
[[
  {
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "applicaton/json",
      "data": {
        "type": "object",
        "properties": {
          "result": {
            "type": "integer"
          }
        }
      }
    }
  }
]
```

须知

apis定义提供AI应用对外Restfull api数据定义，用于定义AI应用的输入、输出格式。

- 创建AI应用填写apis。在创建的AI应用部署服务成功后，进行预测时，会自动识别预测类型。
- 创建AI应用时不填写apis。在创建的AI应用部署服务成功后，进行预测，需选择“请求类型”。“请求类型”可选择“application/json”或“multipart/form-data”。请根据元模型，选择合适的类型。
 - 选择“application/json”时，直接填写“预测代码”进行文本预测。
 - 选择“multipart/form-data”时，需填写“请求参数”，请求参数取值等同于使用图形界面的软件进行预测（以Postman为例）Body页签中填写的“KEY”的取值，也等同于使用curl命令发送预测请求上传数据的参数名。

4. 设置完成后，单击“立即创建”，等待AI应用状态变为“正常”。
5. 单击新建的AI应用名称左侧的小三角形，展开AI应用的版本列表。在操作列单击“部署 > 在线服务”，跳转至在线服务的部署页面。
6. 在部署页面，参考如下说明填写关键参数。

“名称”：按照界面提示规则自定义一个在线服务的名称，也可以使用默认值。

“资源池”：选择“公共资源池”。

“AI应用来源”和“选择AI应用及版本”：会自动选择AI应用和版本号。

“计算节点规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

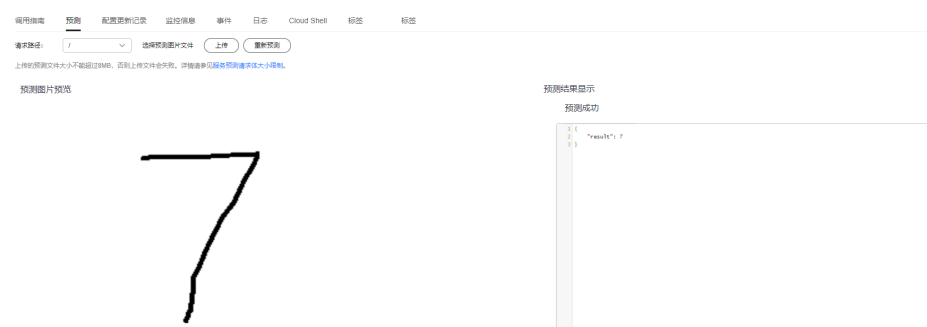
其他参数可使用默认值。

说明

若限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。

7. 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
8. 进入“部署上线 > 在线服务”页面，等待服务状态变为“运行中”时，表示服务部署成功。单击操作列的“预测”，进入服务详情页的“预测”页面。上传图片，预测结果。

图 5-22 预测



5.4 无需构建直接在开发环境中调试并保存镜像用于推理

5.4.1 场景说明

本文详细介绍如何将本地已经制作好的模型包导入ModelArts的开发环境Notebook中进行调试和保存，然后将保存后的镜像部署到推理。本案例仅适用于华为云北京四和上海一站点。

操作流程如下：

1. [在Notebook中拷贝模型包](#)
2. [在Notebook中调试模型](#)
3. [在Notebook中保存镜像](#)
4. [使用保存成功的镜像用于推理部署](#)

5.4.2 Step1 在 Notebook 中拷贝模型包

1. 登录ModelArts控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。
2. 单击右上角“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。

- a. 填写Notebook基本信息，包含名称、描述、是否自动停止。
- b. 填写Notebook详细参数，如选择镜像、资源规格等。

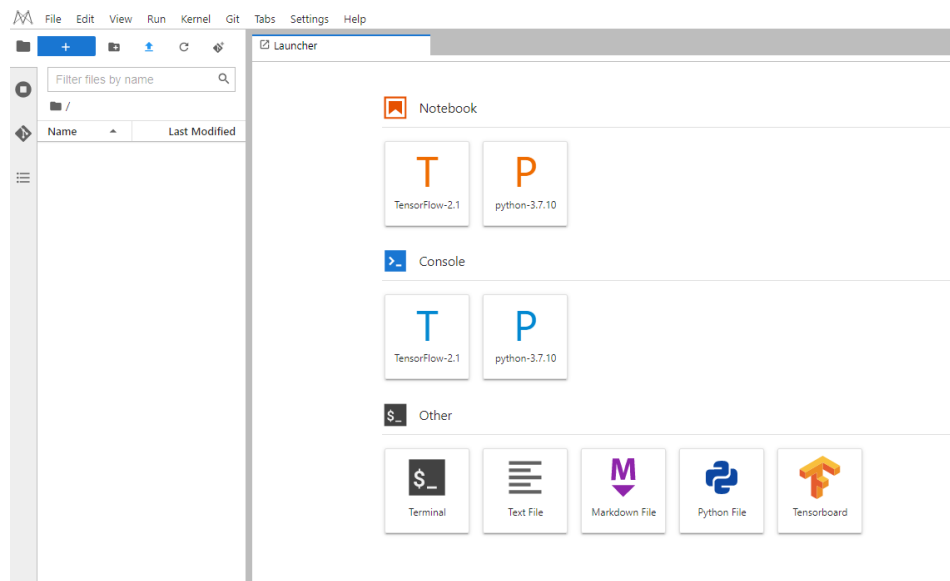
- “镜像”：选择统一镜像[tensorflow_2.1-cuda_10.1-cudnn7-ubuntu_18.04](#)（详见[引擎版本一：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64](#)）或者[pytorch1.8-cuda10.2-cudnn7-ubuntu18.04](#)（详见[引擎版本一：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64](#)）。
- “资源池”：选择公共资源池或专属资源池，此处以公共资源池为例。
- “类型”：推荐选择GPU。
- “规格”：推荐选择GP Tnt004规格，若没有再选择其他规格。

3. 参数填写完成后，单击“立即创建”进行规格确认。参数确认无误后，单击“提交”，完成Notebook的创建操作。

进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。

4. 打开运行中的Notebook实例。

图 5-23 打开 Notebook 实例




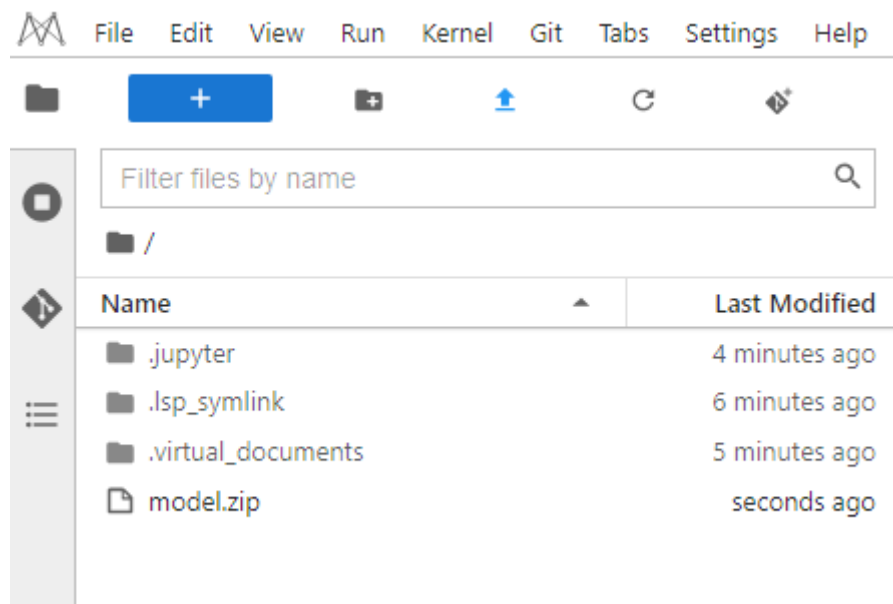
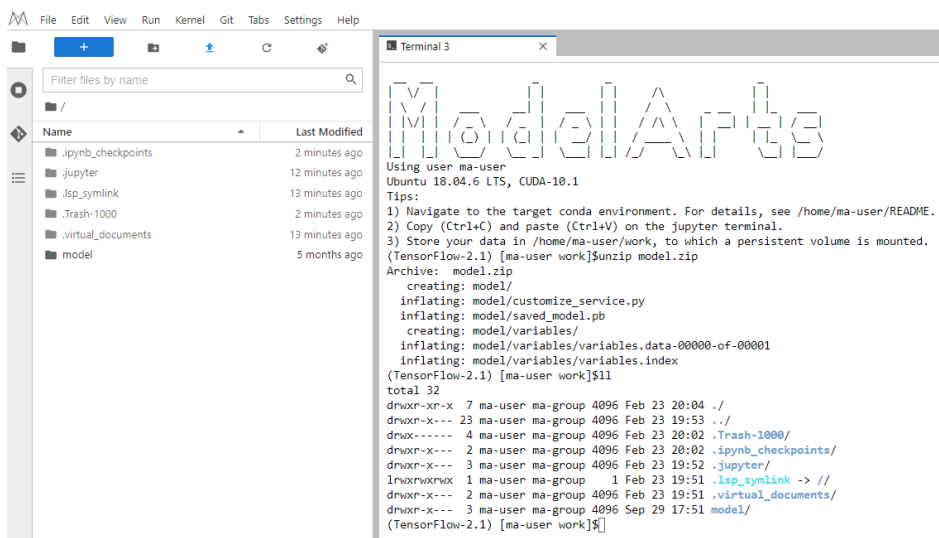
5. 通过  功能，上传模型包文件到Notebook中，默认工作目录/home/ma-user/work/。模型包文件需要用户自己准备，样例内容参见[模型包文件样例](#)。

图 5-24 上传模型包



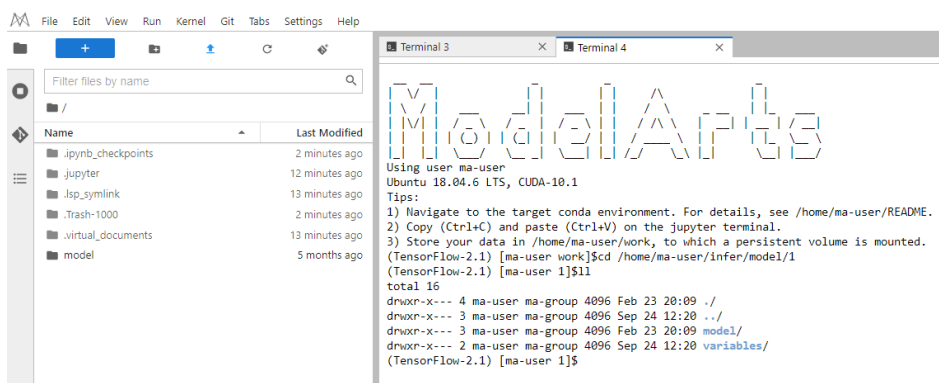
6. 打开Terminal终端，解压model.zip，解压后删除zip文件。
#解压命令
unzip model.zip

图 5-25 在 Terminal 终端中解压 model.zip



7. 在Terminal运行界面，执行拷贝命令。查看镜像文件拷贝成功。
cp -rf model/* /home/ma-user/infer/model/1

图 5-26 查看镜像文件拷贝成功



模型包文件样例

模型包文件model.zip中需要用户自己准备模型文件，此处仅是举例示意说明，以一个手写数字识别模型为例。

Model目录下必须要包含推理脚本文件customize_service.py，目的是为开发者提供模型预处理和后处理的逻辑。

图 5-27 推理模型 model 目录示意图（需要用户自己准备模型文件）

| 名称 | 修改日期 | 类型 | 大小 |
|-------------------|-----------------|-------|--------|
| variables | 2022/7/22 10:30 | 文件夹 | |
| customize_service | 2022/7/22 10:32 | PY 文件 | 2 KB |
| saved_model | 2022/3/10 18:47 | PB 文件 | 105 KB |

推理脚本customize_service.py的具体写法要求可以参考[模型推理代码编写说明](#)。

本案例中提供的customize_service.py文件具体内容如下：

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tf_serving_model_service import TfServingBaseService

class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()

        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)

        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logging.warning("signatures more than one, use serving_default signature from %s", signature)
            model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        self.predict = self.model.signatures[model_signature]

    def _preprocess(self, data):
        images = []
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((28, 28, 1))
                images.append(image1)

        images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
        preprocessed_data = images

        return preprocessed_data

    def _inference(self, data):
        return self.predict(data)

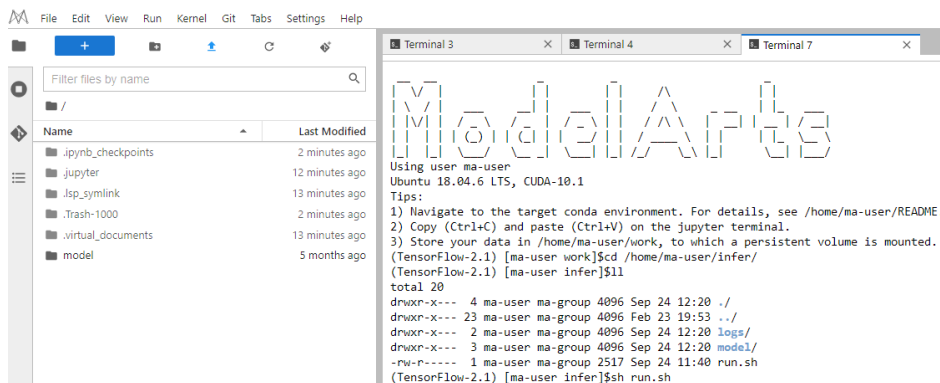
    def _postprocess(self, data):
        return {
            "result": int(data["output"].numpy()[0].argmax())
        }
```

5.4.3 Step2 在 Notebook 中调试模型

1. 打开一个新的Terminal终端，进入“/home/ma-user/infer/”目录，运行启动脚本run.sh，并预测模型。基础镜像中默认提供了run.sh作为启动脚本。启动命令如下：

```
sh run.sh
```

图 5-28 运行启动脚本

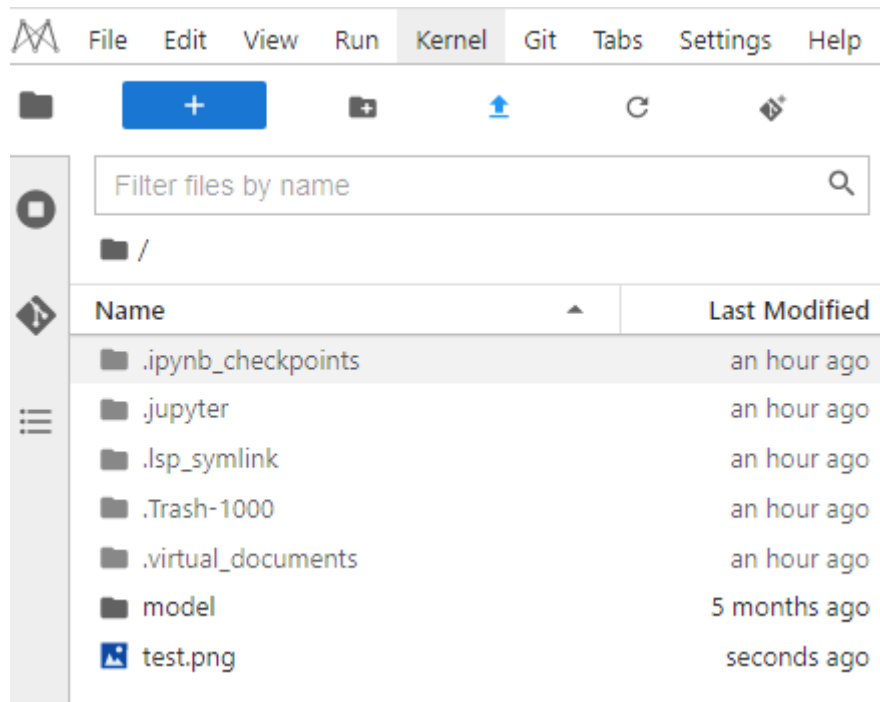


2. 上传一张预测图片（手写数字图片）到Notbook中。

图 5-29 手写数字图片

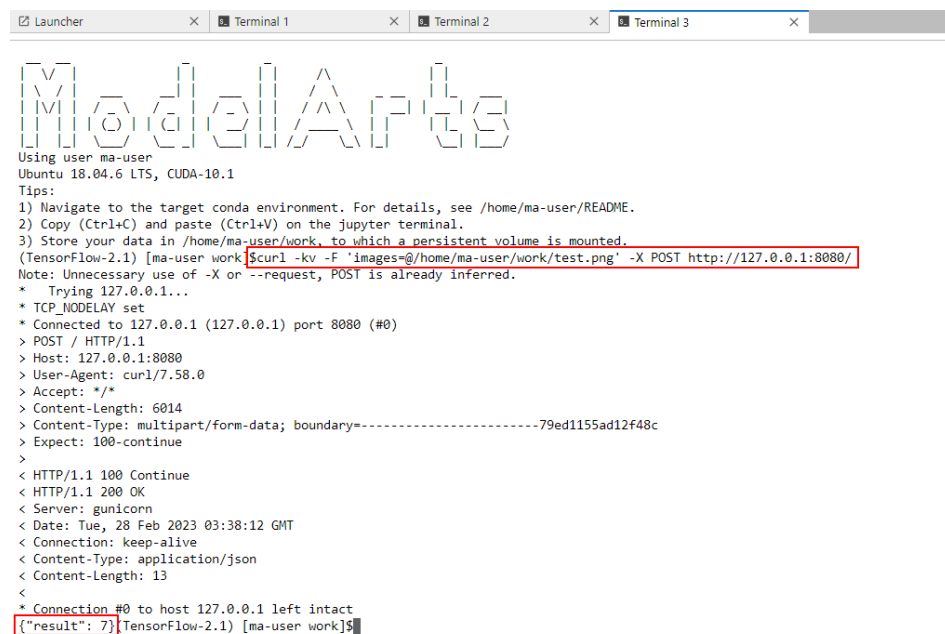


图 5-30 上传预测图片



3. 重新打开一个新的Terminal终端，执行如下命令进行预测。
`curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/`

图 5-31 预测



在调试过程中，如果有修改模型文件或者推理脚本文件，需要重启run.sh脚本。执行如下命令先停止nginx服务，再运行run.sh脚本。

```
#查询nginx进程
ps -ef |grep nginx
#关闭所有nginx相关进程
kill -9 {进程ID}
```

```
#运行run.sh脚本
sh run.sh
```

也可以执行kill nginx命令直接关闭所有nginx进程。

```
#关闭所有nginx进程
kill nginx
#运行run.sh脚本
sh run.sh
```

图 5-32 重启 run.sh 脚本

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user  6474    1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user  6476  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user  6477  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user  7925  1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

5.4.4 Step3 Notebook 中保存镜像

前提条件

Notebook实例状态必须为“运行中”才可以一键进行镜像保存。

保存镜像

1. 在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列中的“更多 > 保存镜像”，进入“保存镜像”对话框。
2. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。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“立即创建”，创建一个组织。创建组织的详细操作请参见[创建组织](#)。
同一个组织内的用户可以共享使用该组织内的所有镜像。
3. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例（对于打开的JupyterLab界面和本地IDE 仍可操作）。

须知

快照中耗费的时间仍占用实例的总运行时长，如果在快照中时，实例因运行时间到期停止，将导致镜像保存失败。

4. 镜像保存成功后，实例状态变为“运行中”，用户可在“镜像管理”页面查看到该镜像详情。
5. 单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

5.4.5 Step4 使用保存成功的镜像用于推理部署

将Step2 在Notebook中调试模型的自定义镜像导入到AI应用中，并部署为在线服务。


1. 登录ModelArts控制台，在左侧导航栏中选择“AI应用管理 > AI应用”，单击“创建”，进入创建AI应用。
2. 设置AI应用的参数，如图5-33所示。
 - 元模型来源：从容器镜像中选择。
 - 容器镜像所在的路径：单击  选择镜像文件。具体路径查看5SWR地址。
 - 容器调用接口：选择HTTPS。
 - host：设置为8443。
 - 部署类型：选择在线服务。

图 5-33 设置 AI 应用参数



3. 填写启动命令，启动命令内容如下：
sh /home/ma-user/infer/run.sh
4. 填写apis定义，单击“保存”生效。apis定义中指定输入为文件，具体内容参见下面代码样例。

图 5-34 填写 apis 定义



apis定义具体内容如下：

```
{
  "url": "/",
  "method": "post",
```

```
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "application/json",
      "data": {
        "type": "object",
        "properties": {
          "result": {
            "type": "integer"
          }
        }
      }
    }
  }
}
```

须知

apis定义提供AI应用对外Restfull api数据定义，用于定义AI应用的输入、输出格式。

- 创建AI应用填写apis。在创建的AI应用部署服务成功后，进行预测时，会自动识别预测类型。
- 创建AI应用时不填写apis。在创建的AI应用部署服务成功后，进行预测，需选择“请求类型”。“请求类型”可选择“application/json”或“multipart/form-data”。请根据元模型，选择合适的类型。
 - 选择“application/json”时，直接填写“预测代码”进行文本预测。
 - 选择“multipart/form-data”时，需填写“请求参数”，请求参数取值等同于[使用图形界面的软件进行预测（以Postman为例）](#)Body页签中填写的“KEY”的取值，也等同于[使用curl命令发送预测请求](#)上传数据的参数名。

5. 设置完成后，单击“立即创建”，等待AI应用状态变为“正常”。
6. 单击新建的AI应用名称左侧的小三角形，展开AI应用的版本列表。在操作列单击“部署 > 在线服务”，跳转至在线服务的部署页面。
7. 在部署页面，参考如下说明填写关键参数。
 - “名称”：自定义一个在线服务的名称，也可以使用默认值。
 - “资源池”：选择“公共资源池”。
 - “AI应用来源”和“选择AI应用及版本”：会自动选择AI应用和版本号。
 - “计算节点规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

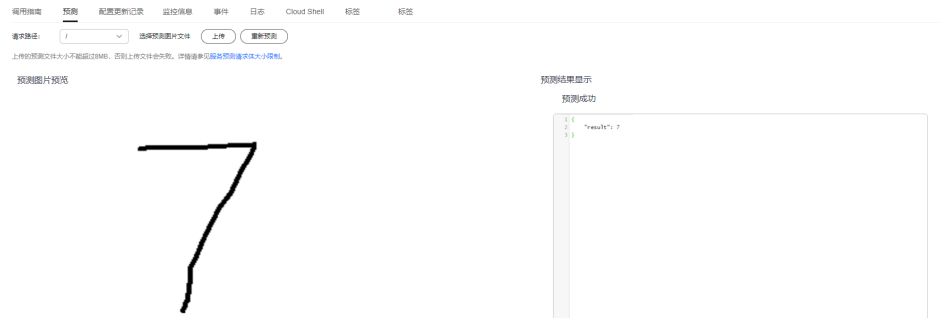
其他参数可使用默认值。

说明

若限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。

- 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
- 进入“部署上线 > 在线服务”页面，等待服务状态变为“运行中”时，表示服务部署成功。单击操作列的“预测”，进入服务详情页的“预测”页面。上传图片，预测结果。

图 5-35 预测



6 FAQ

6.1 如何登录并上传镜像到 SWR

本章节介绍如何上传镜像到容器镜像服务SWR。

Step1 登录 SWR

1. 登录容器镜像服务控制台，选择区域。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。您可以自定义组织名称，本示例使用“deep-learning”，实际操作时请重新命名一个组织名称。后续所有命令中使用到组织名称deep-learning时，均需要替换为此处实际创建的组织名称。
3. 单击右上角“登录指令”，获取登录访问指令。
4. 以root用户登录ECS环境，输入登录指令。

图 6-1 在 ECS 中执行登录指令

```
root@dy-ubuntu1804-cpu:~# docker login -u 4GVX2D1VW0J15RBC15IGUE -p swr.ci
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Step2 上传镜像到 SWR

此小节介绍如何上传镜像至容器镜像服务SWR的镜像仓库。

1. 登录SWR后，使用docker tag命令给上传镜像打标签。下面命令中的组织名称deep-learning，请替换为Step1中实际创建的组织名称，以下所有命令中的deep-learning都需要替换。
`sudo docker tag tf-1.13.2:latest swr.xxx.com/deep-learning/tf-1.13.2:latest`
2. 使用docker push命令上传镜像。
`sudo docker push swr.xxx.com/deep-learning/tf-1.13.2:latest`

图 6-2 上传镜像

```
root@ecs-7918:~# sudo docker tag tf-1.13.2:latest swr.***.com/deep-learning/tf-1.13.2:latest
root@ecs-7918:~# sudo docker push swr.***.com/deep-learning/tf-1.13.2:latest
The push refers to repository [swr.***.com/deep-learning/tf-1.13.2]
c554b77e0db: Pushed
902871f33e88: Pushed
83ade5a612e2: Pushed
20cfaf8c1ab8: Pushed
bf7955efefcb: Pushed
af6a5fe577ce: Pushed
f4c051ffa5f2: Pushed
184f790bb501: Pushed
dfbea9e01449: Pushed
f0193b2fb026: Pushed
f98177ec269a: Pushed
81e535525773: Pushed
582ab80c9f26: Pushed
4e3516398cef: Pushed
52ad947270f1: Pushed
dd841c774a30: Pushed
37b9a4b22186: Pushed
e0b3afb09dc3: Pushed
6c01b5a53aac: Pushed
2c6ac8e5063e: Pushed
cc967c529ced: Pushed
latest: digest: sha256:b19dad3d95e931eb1a87e5d010f0b987357e618eedb14fbbb3701f3144c106f7 size: 4735
```

- 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

“swr.xxx.com/deep-learning/tf-1.13.2:latest”即为此自定义镜像的“SWR_URL”。

6.2 如何给镜像设置环境变量

在Dockerfile中，可使用ENV 指令来设置环境变量，具体信息请参考[Dockerfile指导](#)。

6.3 如何通过 docker 启动 Notebook 保存后的镜像

Notebook保存后的镜像有Entrypoint参数，如图6-3。Entrypoint参数中指定的可执行文件或命令会覆盖镜像的默认启动命令，Entrypoint中指定的执行命令内容不在镜像中预置，在本地环境通过docker run启动通过Notebook保存的镜像，报错创建容器任务失败，启动文件或目录不存在，如图6-4。

因此需要设置--entrypoint参数，覆盖Entrypoint中指定的程序。使用--entrypoint参数指定的启动文件或命令启动镜像。命令示例如下：

```
docker run -it -d --entrypoint /bin/bash image:tag
```

图 6-3 Entrypoint 参数

```
},
  "Cmd": null,
  "Healthcheck": {
    "Test": [
      "NONE"
    ]
  },
  "Image": "sha256:***8bc15f25c7e1",
  "Volumes": null,
  "WorkingDir": "/home/ma-user",
  "Entrypoint": [
    "/modelarts/authoring/script/entrypoint/deps/tini/bin/tini",
    "-g",
    "-_",
    "/modelarts/authoring/script/entrypoint/notebook/boot/start.sh"
  ],
```

图 6-4 启动镜像报错

```
root@:~# docker inspect -f {{.Config.Entrypoint}} sur.cn-north-4.nghuacloud.com/hustaff_public:latest:entrypoint-test:01
/ModelArts/authoring/script/entrypoint/deps/tini/bin/tini -g -- /ModelArts/authoring/script/entrypoint/notebook/boot/start.sh
root@:~# docker run -it -d sur.cn-north-4.nghuacloud.com/hustaff_public:latest:entrypoint-test:01
5cc42a90026b5e0fc42d1115a629e6534d14a5b50b9496d5b4d3f825991b248d
docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "/ModelArts/authoring/script/entrypoint/deps/tini/bin/tini": stat /ModelArts/authoring/script/entrypoint/deps/tini/bin/tini: no such file or directory: unknown.
```

6.4 如何在 Notebook 开发环境中配置 Conda 源

用户可以在 Notebook 开发环境中自行安装开发依赖包，方便使用。常见的依赖安装支持 pip 和 Conda，pip 源已经配置好，可以直接使用安装，Conda 源需要多一步配置。

本章节介绍如何在 Notebook 开发环境中配置 Conda 源。

配置 Conda 源

Conda 软件已经预置在镜像中，具体操作可以参见 <https://mirror.tuna.tsinghua.edu.cn/help/anaconda/>。

常用 Conda 命令

全部 Conda 命令建议参考 [Conda 官方文档](#)。这里仅对常用命令做简要说明。

表 6-1 常用 Conda 命令

| 命令说明 | 命令 |
|-------------|--|
| 获取帮助 | conda --help conda update --help #获取某一命令的帮助，如update |
| 查看 conda 版本 | conda -V |
| 更新 conda | conda update conda #更新 conda conda update anaconda #更新 anaconda |
| 环境管理 | conda env list #显示所有的虚拟环境 conda info -e #显示所有的虚拟环境 conda create -n myenv python=3.7 #创建一个名为myenv环境，指定Python版本是3.7 conda activate myenv #激活名为myenv的环境 conda deactivate #关闭当前环境 conda remove -n myenv --all #删除一个名为myenv的环境 conda create -n newname --clone oldname #克隆oldname环境为newname环境 |

| 命令说明 | 命令 |
|------------|--|
| package 管理 | <pre>conda list #查看当前环境下已安装的package conda list -n myenv #指定myenv环境下安装的package conda search numpy #查找名为numpy的package的所有信息 conda search numpy=1.12.0 --info #查看版本为1.12.0的numpy的信息 conda install numpy pandas #安装numpy和pandas两个package, 此命令可同时安装一个或多个包 conda install numpy=1.12.0 #安装指定版本的numpy #install, update及remove命令使用-n指定环境, install及update命令使用-c指定源地址 conda install -n myenv numpy #在myenv的环境中安装名字为numpy的package conda install -c https://conda.anaconda.org/anaconda numpy #使用源 https://conda.anaconda.org/anaconda 安装numpy conda update numpy pandas #更新numpy和pandas两个package, 此命令可同时更新一个或多个包 conda remove numpy pandas #卸载numpy和pandas两个package, 此命令可同时卸载一个或多个包 conda update --all #更新当前环境下所有的package</pre> |
| 清理 conda | <pre>conda clean -p # 删除无用的包 conda clean -t # 删除压缩包 conda clean -y --all # 删除所有的安装包及cache</pre> |

安装完外部库后保存镜像环境

ModelArts的新版Notebook提供了镜像保存功能。支持一键将运行中的Notebook实例保存为镜像，将准备好的环境保存下来，可以作为自定义镜像，方便后续使用。保存镜像，安装的依赖包不会丢失。安装完依赖包后，推荐保存镜像，避免安装的依赖包丢失。具体操作请参见[保存Notebook镜像环境](#)。

6.5 自定义镜像软件版本匹配注意事项

如果您的自定义镜像涉及NCCL、CUDA、OFED等软件库，当您制作自定义镜像时，您需要确保镜像中的软件库和ModelArts的软件库相匹配。您镜像中的软件版本需要满足以下要求：

- NCCL版本 ≥ 2.7.8。
- OFED版本 ≥ MLNX_OFED_LINUX-5.4-3.1.0.0。
- CUDA版本需要参考专属资源池的GPU驱动版本，自主进行适配，GPU驱动版本可在专属资源池详情页面查看。

6.6 镜像在 SWR 上显示只有 13G，安装少量的包，然后镜像保存过程会提示超过 35G 大小保存失败，为什么？

问题现象

我的镜像在SWR侧看，只有13G左右，在开发环境Notebook镜像管理注册，启动Notebook实例后，安装一些包后，镜像保存过程会提示超过35G大小，保存失败？

原因分析

SWR侧看到的大小是镜像压缩后的大小，解压后实际大小一般是压缩后的2.5~3倍，所以才会安装少量的包后，镜像大小超过35G。

6.7 镜像保存如何保证能正常保存，不会因为超过 35G 而保存失败？

可以从如下几方面考虑：

1. 请选择较小的基础镜像创建Notebook实例，这样在实例中可操作的空间才会大，可自由安装的包才能更多，一般建议原始的启动Notebook的基础镜像在SWR侧查看大小不要超过6G。
2. 镜像保存主要保存/home/ma-user路径下，除挂载路径/home/ma-user/work以外的目录，请将数据集等放到work路径下，不要放到其他非work路径下。
3. 请不要将实例频繁保存镜像，建议一次将需要的安装包安装好，然后执行镜像保存，避免频繁执行镜像保存的动作，保存次数越多镜像越大，且多次保存后的镜像过大问题无法通过清理磁盘方式减少镜像的大小（ Docker保存原理机制）。

6.8 本地/ECS 构建镜像，如何减小目的镜像的大小？

减小目的镜像大小的最直接的办法就是选择尽可能小且符合自己诉求的镜像，比如您需要制作一个Pytorch2.1+cuda12.2的镜像，官方如果没有提供对应的Pytorch或者Cuda版本的镜像，优选一个没有conda环境或没有安装Cuda的镜像，而不是选择一个引擎和Cuda都不满足的镜像，如MindSpore+Cuda11.X，这样基础镜像就会很大，同样的操作最终目的镜像就很大。

此外下面举出几种常见的减少镜像大小的方式。

- 减少目的镜像层数

举例：假设需要安装两个pip包six, numpy，将安装放到同一层，而不是放到不同层：

正确方式：

```
RUN pip install six &&\  
    pip install numpy
```

不宜方式：

```
RUN pip install six  
RUN pip install numpy
```

镜像层数越多，镜像越大。

- 安装和卸载放在同一层，不要跨层删除。

举例：假设从官网下载了一个SCC包，安装后卸载：

正确方式：

```
RUN mkdir -p /tmp/scc && \  
    cd /tmp/scc && \  
    wget http://100.95.151.167:6868/aarch64/euler/dls-release/euleros-arm/compiled-software/  
seccomponent-1.1.0-release.aarch64.rpm && \  
    rpm -ivh /tmp/scc/seccomponent-1.1.0-release.aarch64.rpm --force --nodeps && \  
    rm -rf /tmp/scc
```

不宜方式：

```
RUN mkdir -p /tmp/scc && \  
    cd /tmp/scc && \  
    wget http://100.95.151.167:6868/aarch64/euler/dls-release/euleros-arm/compiled-software/  
seccomponent-1.1.0-release.aarch64.rpm && \  
    rpm -ivh /tmp/scc/seccomponent-1.1.0-release.aarch64.rpm --force --nodeps  
RUN rm -rf /tmp/scc
```

6.9 镜像过大，卸载原来的包重新打包镜像，或者把原有的数据集从镜像中删除，最终镜像会变小吗？

不会，反而会变大。因为Docker镜像的层原因，当前的镜像是基于原来的镜像制作，而原来的镜像层数是无法改变的，层不变的情况下，大小是不变的，卸载包或者删除数据集，会新增镜像层，镜像反而会变大，这和传统概念的存储不一样。

6.10 在 ModelArts 镜像管理注册镜像报错 ModelArts.6787

问题现象

在“镜像管理”界面使用提示“ModelArts.6787:镜像***无法使用，在SWR路径下***无法找到指定镜像，请在SWR控制台检查镜像及访问权限配置，或使用其他镜像并重试”。

原因分析

报错主要有如下原因：

- 该镜像为主账号注册的private镜像，子账号在使用，而主账号没有给子账号赋SWR权限，子账号从SWR Console界面看不到该镜像，需要主账号给子账号在SWR侧赋予SWR权限，使得子账号可以看到该SWR镜像地址，否则该镜像子账号不可使用。
- 该镜像不属于该租户（包括主账号和子账号），是其他人共享的public镜像，而这个镜像又被镜像所有者删除，导致不可使用，用户需要联系对应的SWR镜像负责人，确认镜像是否存在。
- 该镜像不属于该租户（包括主账号和子账号），是其他人共享的public镜像，而这个镜像又被镜像所有者设置成private，导致不可使用，用户需要联系对应的SWR镜像负责人，确认镜像的属性。

解决方案

按照原因分析分别解决。

7 修订记录

| 发布时间 | 修改内容 |
|------------|---|
| 2023-10-01 | 新增内容如下： <ul style="list-style-type: none">新增章节预置框架启动流程说明。 |
| 2023-09-07 | 新增内容如下： <ul style="list-style-type: none">新增章节在Notebook中构建自定义镜像并使用。新增章节自定义镜像软件版本匹配注意事项。 修改内容如下： <ul style="list-style-type: none">“使用自定义镜像”手册更名为“镜像管理”。“基于ModelArts提供的基础镜像构建自定义镜像在Notebook中使用”和“基于第三方镜像在ECS上构建自定义镜像并在Notebook中使用”合并为在ECS上构建自定义镜像并在Notebook中使用。预置镜像结构调整，开发环境，训练，推理基础镜像整合为使用预置镜像。镜像保存故障处理FAQ移到“故障排除 > 开发环境 > 镜像保存故障”章节。 |