ModelArts

# Image Management

| | |
|---|---|
| **Issue** | 01 |
| **Date** | 2024-04-30 |

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

## Trademarks and Permissions

## Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

# Contents

# 1 Image Management

## Overview

During the development and runtime of AI services, complex environment dependencies need to be debugged for containerization. In the best practices of AI development in ModelArts, container images are used to provide fixed runtime environments. In this way, dependencies can be managed and the runtime environments can be easily switched. The container resources provided by ModelArts enable quick and efficient AI development and model experiment iteration.

The preset images provided by ModelArts by default have the following features:

- Out-of-the-box and scenario-specific: Typical dependent environments for AI development are preset in these images to provide optimal software, OS, and network configurations. They have been fully tested on hardware to ensure optimal compatibility and performance.
- Configuration customizable: Preset images are stored in the SWR repository for you to customize and register them as your own images.
- Secure and reliable: Access policies, user permissions control, vulnerability scanning for development software, and OS are configured based on best practices for security hardening to ensure the security of images.

If you have special requirements on the deep learning engine and development library, you can use ModelArts custom images to customize runtime engines.

Based on the container technology, you can customize container images and run them on ModelArts. Custom images support CLI parameters and environment variables in free text format, featuring high flexibility for a wide range of compute engines.

## Application Scenarios of Preset Images

ModelArts provides a group of preset images. You can use a preset image to create a notebook instance. After installing and configuring dependencies on the instance, create a custom image. Then, you can directly use the image in ModelArts for training jobs without any adaptation. You can also use preset images to submit training jobs and create AI applications.

We recommend the preset image version based on your development requirements and stability of the version. If your development can be carried out using versions preset in ModelArts, for example, MindSpore 1.*X*, use the preset images. They have been fully verified and have many commonly-used installation packages, relieving you from configuring the environment.

## Application Scenarios of Custom Images

- **Using custom images on notebook instances**

  If the preset images of notebook instances cannot meet requirements, you can create a custom image by installing and configuring the software and other data required by the environment in a preset image. Then, use the custom image to create new notebook instances.

- **Using a custom image to create training jobs**

  If you have developed a model or training script locally but the AI engine you used is not supported by ModelArts, create a custom image and upload it to SWR. Then, use this image to create a training job on ModelArts and use the resources provided by ModelArts to train models.

- **Using a custom image to create AI applications**

  If you have developed a model using an AI engine that is not supported by ModelArts, to use this model to create AI applications, do as follows: Create a custom image, import the image to ModelArts, and use it to create AI applications. The AI applications created in this way can be centrally managed and deployed as services.

## Custom Image Services

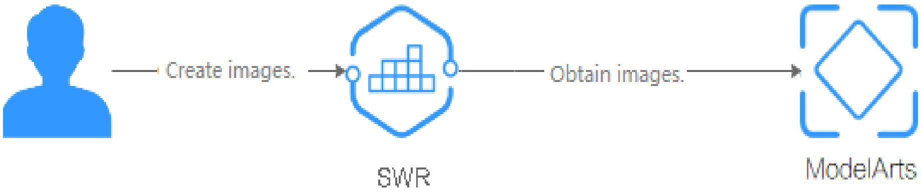
When you use a custom image, the following services may be involved:

- SWR

  Software Repository for Container (SWR) provides easy, secure, and reliable management over container images throughout their lifecycle, facilitating the deployment of containerized applications. You can upload, download, and manage container images through the SWR console, SWR APIs, or community CLI.

  Your custom images must be uploaded to SWR. The custom images used by ModelArts for training or creating AI applications are obtained from the SWR service management list.

  **Figure 1-1** Obtaining images

- OBS

  Object Storage Service (OBS) is a cloud storage service optimized for storing massive amounts of data. It provides unlimited, secure, and highly reliable storage capabilities at a relatively low cost.

ModelArts exchanges data with OBS. You can store data in OBS.

- ECS

  An Elastic Cloud Server (ECS) is a basic computing unit that consists of vCPUs, memory, OS, and Elastic Volume Service (EVS) disks. After an ECS is created, you can use it similarly to how you would use your local PC or physical server.

  You can create a custom image on premises or on an ECS.

**NOTE**

When you use a custom image, ModelArts may need to access dependent services, such as SWR and OBS. The custom image can be used only after the access is authorized. It is a good practice to use an agency for authorization. After the agency is configured, the permissions to access dependent services are delegated to ModelArts so that ModelArts can use the dependent services and perform operations on resources on your behalf. For details, see **Configuring Access Authorization (Global Configuration)**.

# 2 Using a Preset Image

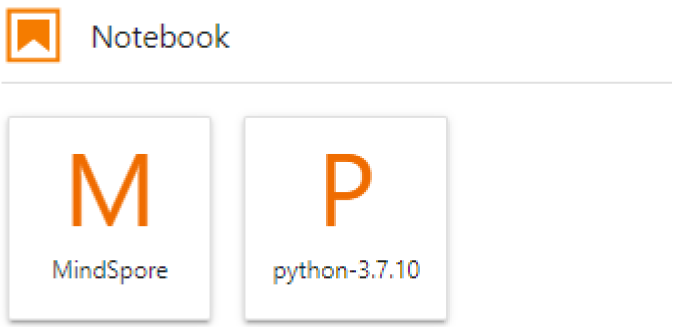
## 2.1 Images Preset in Notebook

### 2.1.1 Notebook Base Images

#### Presetting Images

The images preset in ModelArts DevEnviron include:

- Typical preset packages: AI engines based on standard Conda, data analysis software packages such as Pandas and Numpy, and tool software such as CUDA and CUDNN are included to meet your needs.
- Preset Conda environments: A Conda environment and basic Conda Python (excluding any AI engine) are created for each preset image. The following figure shows the Conda environment for a preset MindSpore image.



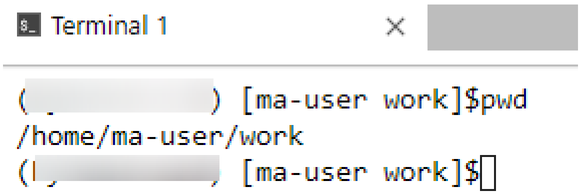
  Select a Conda environment based on whether MindSpore is used for debugging.

- Notebook: a web application that enables you to code on the GUI and combine the code, mathematical equations, and visualized content into a document.
- JupyterLab plug-ins: enable flavor changing and instance stopping to improve user experience. After a notebook instance is stopped, its CPUs and memory are no longer billed.

- Remote SSH: allows you to remotely start and debug a notebook instance from a local PC.
- Images preset in ModelArts DevEnviron: After these preset images support function development, the custom images created based on these preset images can be directly used for ModelArts training jobs.

A ModelArts preset image is started as user **ma-user**. The default working directory of an accessed notebook instance is **/home/ma-user/work**.

Create an instance and mount the persistent storage to **/home/ma-user/work**. The data stored only in the **work** directory is retained after the instance is stopped and restarted. When you use a development environment, store the data for persistence in **/home/ma-user/work**.



```
([redacted]) [ma-user work]$pwd
/home/ma-user/work
([redacted]) [ma-user work]$
```

## Creating a Notebook Instance Using a Preset Image

Select a preset image when creating a notebook instance. You can access and use the instance right after it is created.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron** > **Notebook** to access the new-version **Notebook** page.
2. Click **Create**. On the **Create Notebook** page, select a public image, configure other parameters, and submit the creation request. For details about the parameters, see **Creating a Notebook Instance**.
3. After the status of the notebook instance changes to **Running**, access the notebook to use the created image.

# 2.1.2 Notebook Base Image List

ModelArts DevEnviron provides Docker container images, which can run as preset containers. Certain preset images are built on common AI engine frameworks such as PyTorch, TensorFlow, and MindSpore. These images are named using the AI engines. Additionally, many common packages are preset in these images, relieving you from the package installation.

**Table 2-1** Preset x86 images

| AI Engine | Image |
|---|---|
| PyTorch | pytorch1.8-cuda10.2-cudnn7-ubuntu18.04 |
| | pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 |
| | pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 |

| AI Engine | Image |
|---|---|
| **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 |
| **No AI engine (base images dedicated for image customization)** | conda3-cuda10.2-cudnn7-ubuntu18.04 |
| | conda3-ubuntu18.04 |

## 2.1.3 PyTorch (x86)-powered Notebook Base Image

ModelArts provides the following notebook base images powered by PyTorch (x86): **pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**, **pytorch1.10-cuda10.2-cudnn7-ubuntu18.04**, and **pytorch1.4-cuda10.1-cudnn7-ubuntu18.04**.

### Image pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

**Table 2-2** pytorch1.8-cuda10.2-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| PyTorch 1.8 | Yes<br>(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 | PyPI package | Ubuntu package |

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

## Image pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

**Table 2-3** pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| PyTorch 1.10 | Yes<br>(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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | torch 1.10.2<br>torchvision 0.11.3<br>ipykernel 5.3.4<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.6<br>ma-cau-adapter 1.1.3<br>ma-cli 1.2.2<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.19.5<br>opencv-python 4.1.2.30<br>pandas 1.1.5<br>pillow 9.2.0<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.2 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libcudnn7<br>libcudnn7-dev<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>pandoc<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## Image pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

**Table 2-4** pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| PyTorch 1.4 | Yes<br>(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 | PyPI package | Ubuntu package |

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

## 2.1.4 Tensorflow (x86)-powered Notebook Base Image

ModelArts provides the following notebook base images powered by Tensorflow (x86): **tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04** and **tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04**

### Image tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

**Table 2-5** tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| TensorFlow 2.1 | Yes<br>(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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | tensorflow 2.1.0<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.6<br>ma-cau-adapter 1.1.3<br>ma-cli 1.2.2<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.19.5<br>opencv-python 4.1.2.30<br>pandas 1.1.5<br>pillow 9.2.0<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.2<br>tensorboard 2.1.1 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libcudnn7<br>libcudnn7-dev<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## Image tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

**Table 2-6** tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| TensorFlow 1.13-gpu | Yes<br>(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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | tensorflow-gpu 1.13.1<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.6<br>ma-cau-adapter 1.1.3<br>ma-cli 1.2.2<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.0.1.rc0.ffd1c0c8<br>numpy 1.17.0<br>opencv-python 4.1.2.30<br>pandas 1.1.5<br>pillow 6.2.0<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.2.2<br>scikit-learn 0.22.1<br>tornado 6.2 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libcudnn7<br>libcudnn7-dev<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## 2.1.5 MindSpore (x86)-powered Notebook Base Image

ModelArts provides the following notebook base images powered by MindSpore (x86): **mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04**, **mindspore1.7.0-py3.7-ubuntu18.04**, **mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04**, and **mindspore1.2.0-openmpi2.1.1-ubuntu18.04**.

### Image mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

**Table 2-7** mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| MindSpore-gpu 1.7.0 | Yes<br>(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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | mindspore-gpu 1.7.0<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.6<br>ma-cau-adapter 1.1.3<br>ma-cli 1.2.2<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.17.0<br>pandas 1.1.5<br>pillow 9.1.1<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.1<br>mindinsight 1.7.0<br>mindvision 0.1.0 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libcudnn7<br>libcudnn7-dev<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## Image mindspore1.7.0-py3.7-ubuntu18.04

**Table 2-8** mindspore1.7.0-py3.7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| MindSpore 1.7.0 | None | 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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | mindspore 1.7.0<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.6<br>ma-cau-adapter 1.1.3<br>ma-cli 1.2.2<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.17.0<br>pandas 1.1.5<br>pillow 9.1.1<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.1<br>mindinsight 1.7.0<br>mindvision 0.1.0 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## Image mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

**Table 2-9** mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| MindSpore-gpu 1.2.0 | Yes<br>(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 | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | mindspore-gpu 1.2.0<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.3<br>ma-cau-adapter 1.1.3<br>ma-cli 1.1.5<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.19.5<br>pandas 1.1.5<br>pillow 6.2.0<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.2<br>mindinsight 1.2.0 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libcudnn7<br>libcudnn7-dev<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## Image mindspore1.2.0-openmpi2.1.1-ubuntu18.04

**Table 2-10** mindspore1.2.0-openmpi2.1.1-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| MindSpore 1.2.0 | None | swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | mindspore 1.2.0<br>ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.3<br>ma-cau-adapter 1.1.3<br>ma-cli 1.1.5<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.19.5<br>pandas 6.2.0<br>pillow 9.1.1<br>pip 22.1.2<br>psutil 5.8.0<br>PyYAML 5.1<br>scipy 1.5.2<br>scikit-learn 0.22.1<br>tornado 6.2<br>mindinsight 1.2.0 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>ffmpeg<br>g++<br>gcc<br>gfortran<br>git<br>git-lfs<br>grep<br>libjpeg-dev:amd64<br>libjpeg8-dev:amd64<br>openssh-client<br>openssh-server<br>nginx<br>python3<br>rpm<br>screen<br>tar<br>tmux<br>unzip<br>vim<br>wget<br>zip |

## 2.1.6 Custom Dedicated Image (x86)-powered Notebook Base Image

ModelArts provides the following notebook base images powered by custom images (x86): **conda3-cuda10.2-cudnn7-ubuntu18.04** and **conda3-ubuntu18.04**. These images do not have AI engines or related software packages. The image size is only 2 GB to 5 GB. You can use these images as base images and install your desired engine and dependency packages, improving scalability. In addition, these images are preset with some configurations required for starting the development environment. You can use these images after installing required software packages, without the need for any adaptations.

Such images are the most basic ones and have no component installed. They are small enough to facilitate image customization. If you need to use the OBS SDK, use ModelArts SDK instead to copy files. For details, see **Transferring Files**.

### Image conda3-cuda10.2-cudnn7-ubuntu18.04

**Table 2-11** conda3-cuda10.2-cudnn7-ubuntu18.04 description

| AI Engine | Whether to Use GPUs<br>(CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| None | Yes<br>(cuda 10.2) | swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c | PyPI package | Ubuntu package |

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| | | | ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.3<br>ma-cau-adapter 1.1.3<br>ma-cli 1.1.5<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.21.6<br>pandas 1.3.5<br>pillow 9.2.0<br>pip 20.3.3<br>psutil 5.9.1<br>PyYAML 6.0<br>scipy 1.7.3<br>tornado 6.2 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>g++<br>gcc<br>gfortran<br>grep<br>libcudnn7<br>libcudnn7-dev<br>nginx<br>python3<br>rpm<br>tar<br>unzip<br>vim<br>wget<br>zip |

### Image conda3-ubuntu18.04

**Table 2-12** conda3-ubuntu18.04 description

| AI Engine | Whether to Use GPUs (CUDA Version) | URL | Dependency | |
|---|---|---|---|---|
| None | No | swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c<br>For example:<br>CN North-Beijing4<br>swr.cn-north-4.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c<br>CN East-Shanghai1<br>swr.cn-east-3.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c<br>CN South-Guangzhou<br>swr.cn-south-1.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c | PyPI package | Ubuntu package |
| | | | ipykernel 6.7.0<br>ipython 7.34.0<br>jupyter-client 7.3.4<br>ma-cau 1.1.3<br>ma-cau-adapter 1.1.3<br>ma-cli 1.1.5<br>matplotlib 3.5.1<br>modelarts 1.4.11<br>moxing-framework 2.1.0.5d9c87c8<br>numpy 1.21.6<br>pandas 1.3.5<br>pillow 9.2.0<br>pip 20.3.3<br>psutil 5.9.1<br>PyYAML 6.0<br>scipy 1.7.3<br>tornado 6.2 | automake<br>build-essential<br>ca-certificates<br>cmake<br>cpp<br>curl<br>g++<br>gcc<br>gfortran<br>grep<br>nginx<br>python3<br>rpm<br>tar<br>unzip<br>vim<br>wget<br>zip |

## 2.2 Training Base Image

# 2.2.1 Available Training Base Images

ModelArts provides deep learning-powered base images such as TensorFlow, PyTorch, and MindSpore images. In these images, the software mandatory for running training jobs has been installed. If the software in the base images cannot meet your service requirements, create new images based on the base images and use the new images to create training jobs.

## Available Training Base Images

The following table lists the preset training base images of ModelArts.

**Table 2-13** ModelArts training base images

| Engine | Version |
|---|---|
| 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 |

**NOTE**

Supported AI engines vary depending on regions.

# 2.2.2 Training Base Image (PyTorch)

This section describes preset PyTorch images.

## Engine Version: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Image address: 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
- Image creation time: 20220309171256 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- cuda: 10.2.89
- cudnn: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- The versions of some third-party packages:

```
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
...
```

- Earlier versions: none

## 2.2.3 Training Base Image (TensorFlow)

This section describes preset TensorFlow images.

### Engine Version: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Image address: 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
- Image creation time: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- The versions of some third-party packages:

```
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
tifffile 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
...
```

- Earlier versions: none

## 2.2.4 Training Base Image (Horovod)

This section describes preset Horovod images.

### Engine Version 1: horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Image address: 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
- Image creation time: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages
- The versions of some third-party packages:

```
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
...
```

- Earlier versions: none

## Engine Version 2: horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Image address: 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
- Image creation time: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- cuda: 11.1.1
- cudnn: 8.0.5.39
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages
- The versions of some third-party packages:

```
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
...
```

- Earlier versions: none

## 2.2.5 Training Base Image (MPI)

This section describes preset mindspore_1.3.0 images.

### Engine Version: mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

- Image address: 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
- Image creation time: 20211104202338(yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages
- The versions of some third-party packages:

```
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
...
```

- Earlier versions: none

## 2.2.6 Starting Training with a Preset Image

### 2.2.6.1 PyTorch

ModelArts provides multiple AI frameworks for different engines. When you use these engines for model training, the boot commands during training need to be adapted accordingly. This section introduces how to make adaptions to the PyTorch engine.

## PyTorch Startup Principle

**Specifications and number of nodes**

In this case, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** is used as an example to describe how to allocate ModelArts resources for single-node and distributed jobs.

For a single-node job (running on only one node), ModelArts starts a training container that exclusively uses the resources on the node.

For a distributed job (running on more than one node), there are as many workers as the nodes that are selected during job creation. Each worker is allocated with the compute resources of the selected specification. For example, there are **2** compute nodes, two workers will be started, and each worker owns the compute resources of **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

**Network communication**

- For a single-node job, no network communication is required.
- For a distributed job, network communications are required in nodes and between nodes.

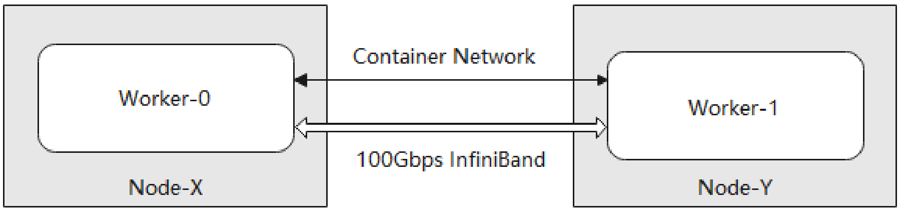
**In nodes**

NVLink and shared memory are used for communication.

**Between nodes**

If there is more than one compute node, PyTorch distributed training will be started. The following figure shows the network communications between workers in PyTorch distributed training. Workers can communicate with each other using the container network and a 100-Gbit/s InfiniBand or RoCE NIC. RoCE NICs are described specifically for certain specifications. The containers can communicate through DNS domain names, which is suitable for small-scale point-to-point communication that requires average network performance. The InfiniBand and RoCE NICs are suitable for distributed training jobs using collective communication that require high-performance network.

**Figure 2-1** Network communications for distributed training



## Boot Commands

The training service uses the default python interpreter in the job image to start the training script. To obtain the python interpreter, run the **which python** command. The working directory during startup is **/home/ma-user/user-job-dir/**<*The code directory name*>, which is the directory returned by running **pwd** or **os.getcwd()** in python.

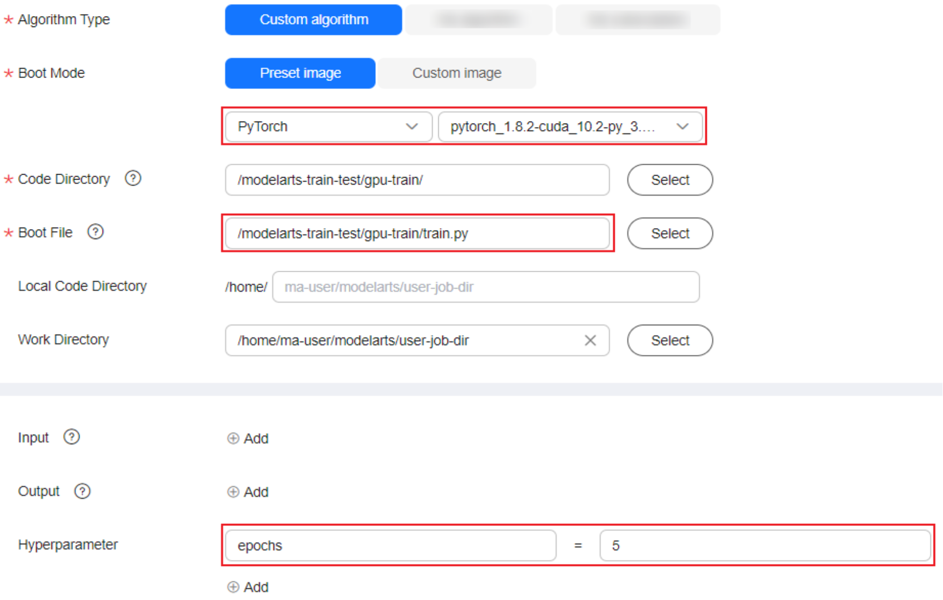
- **Boot command for single-card single-node**

  **python** *<Relative path of the startup file> <Job parameters>*

  – *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*

  – *Job parameters*: parameters configured for a training job

**Figure 2-2** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image

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

* Code Directory: /modelarts-train-test/gpu-train/

* Boot File: /modelarts-train-test/gpu-train/train.py

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir

Input

Output

Hyperparameter: epochs = 5

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

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

- **Boot command for multi-cards single-node**

  **python** *<Relative path of the startup file>* **--init_method "tcp://***${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}***:$***{port}***"** *<Job parameters>*

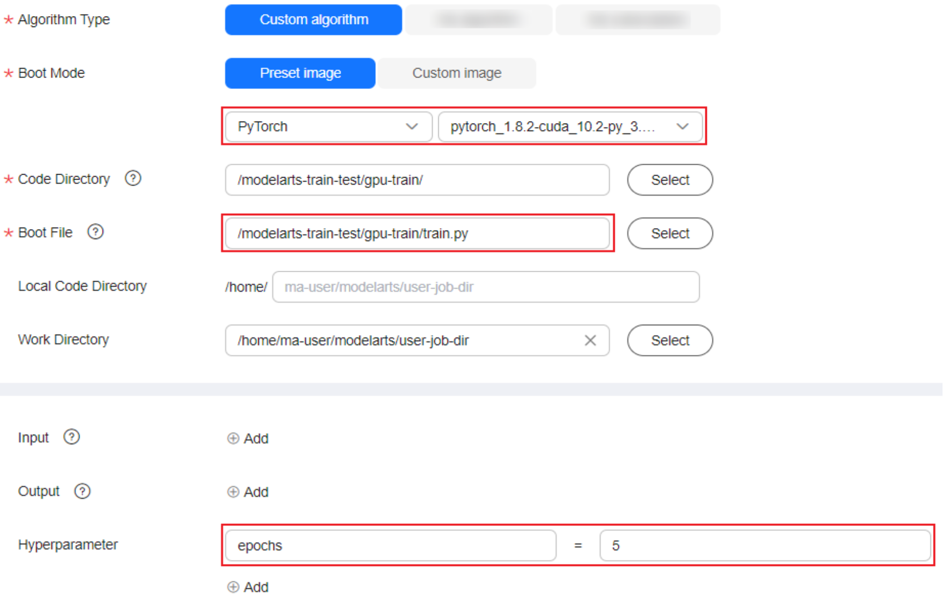
  – *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*

  – *${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}*: domain name of the container where worker-0 is located. For details, see **Default environment variables**.

  – *port*: default communication port of the container where worker-0 is located

  – *Job parameters*: parameters configured for a training job

**Figure 2-3** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image | Custom image

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

* Code Directory: /modelarts-train-test/gpu-train/ Select

* Boot File: /modelarts-train-test/gpu-train/train.py Select

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir Select

Input: ⊕ Add

Output: ⊕ Add

Hyperparameter: epochs = 5

⊕ Add

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

```
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
```

- **Boot command for multi-cards multi-nodes**

  **python** *<Relative path of the startup file>* **--init_method "tcp://***${MA_VJ_NAME}-$ {MA_TASK_NAME}***-0.${MA_VJ_NAME}:$***{port}***"** *--rank* **<rank_id> --world_size** *<node_num> <Job parameters>*

  – *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*

  – ${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}: domain name of the container where worker-0 is located. For details, see **Default environment variables**.

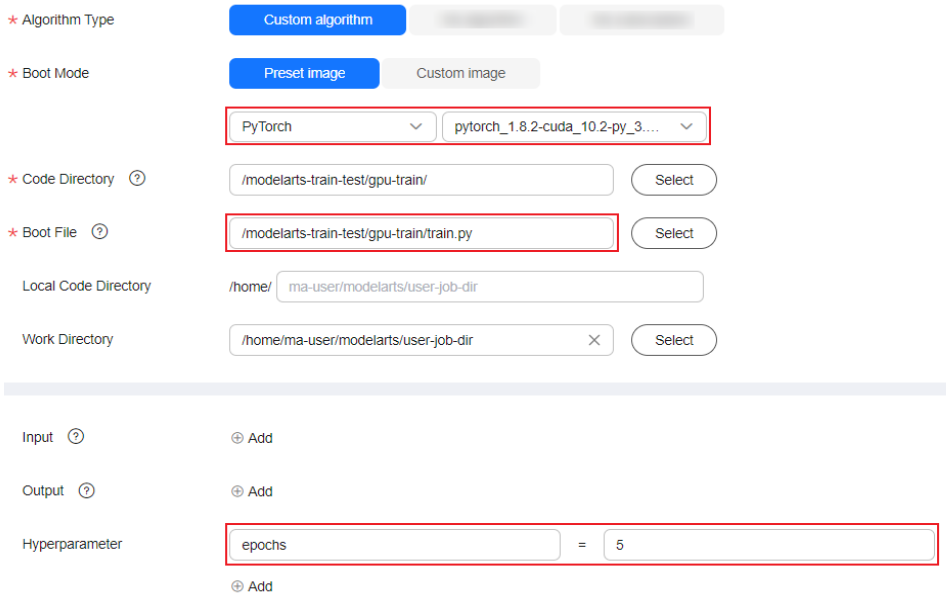
  – *port*: default communication port of the container where worker-0 is located

  – *rank*: worker serial number

  – *node_num*: number of workers

  – *Job parameters*: parameters configured for a training job

**Figure 2-4** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image

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

* Code Directory: /modelarts-train-test/gpu-train/

* Boot File: /modelarts-train-test/gpu-train/train.py

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir

Input

Output

Hyperparameter: epochs = 5

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

```
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
```

### 2.2.6.2 TensorFlow

ModelArts provides multiple AI frameworks for different engines. When you use these engines for model training, the boot commands during training need to be adapted accordingly. This section introduces how to make adaptions to the TensorFlow engine.

## TensorFlow Startup Principle

**Specifications and number of nodes**

In this case, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** is used as an example to describe how to allocate ModelArts resources for single-node and distributed jobs.

For a single-node job (running on only one node), ModelArts starts a training container that exclusively uses the resources on the node.

For a distributed job (running on more than one node), ModelArts starts a parameter server (PS) and a worker on the same node. The PS owns the compute resources of **CPU: 36 cores | Memory: 256 GB**, and the worker owns **GPU: 8 xNVIDIA-V100 | CPU: 36 cores | Memory: 256 GB**.

Only CPU and memory resources are allocated to a PS, while a worker can also own acceleration cards (except for pure CPU specifications). In this example, each worker owns eight NVIDIA V100 acceleration cards. If a PS and a worker are started on the same node, the disk resources are shared by both parties.

**Network communication**

- For a single-node job, no network communication is required.
- For a distributed job, network communications are required in nodes and between nodes.

### In nodes

A PS and a worker can communicate in nodes through a container network or host network.

- A container network is used when you run a training job on nodes using shared resources.
- When you run a training job on nodes using a dedicated pool, the host network is used if the node is configured with RoCE NICs, and the container network is used if the node is configured with InfiniBand NICs.

### Between nodes

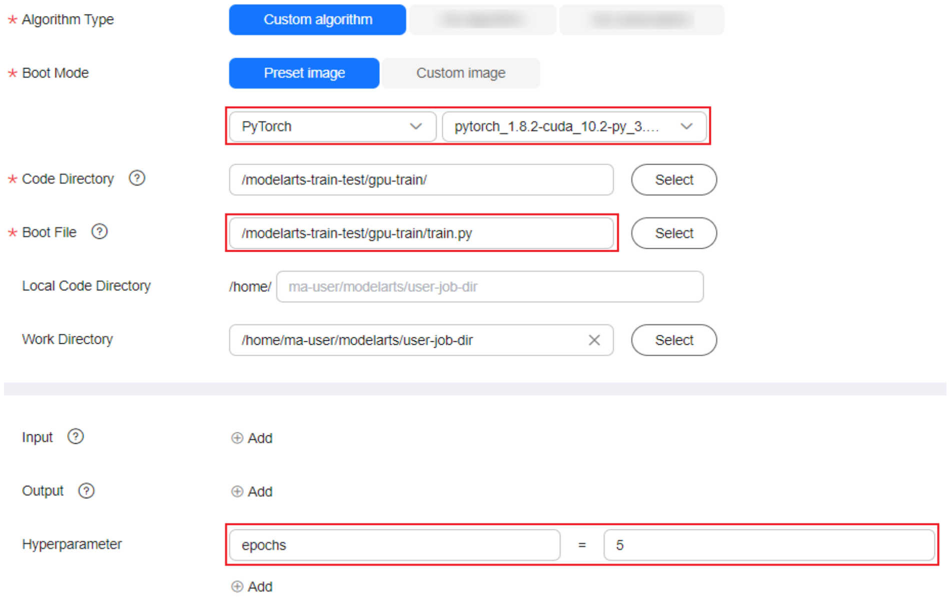
For a distributed job, a PS and a worker can communicate between nodes. ModelArts provides you with InfiniBand and RoCE NICs with a bandwidth of up to 100 Gbit/s.

## Boot Commands

By default, the training service uses the python interpreter in the job image to start up the training script. To obtain the python interpreter, run the **which python** command. The working directory during startup is **/home/ma-user/user-job-dir/***<The code directory name>*, which is the directory returned by running **pwd** or **os.getcwd()** in python.

- **Boot command for single-card single-node**
  **python** *<Relative path of the startup file> <Job parameters>*
  - *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*
  - *Job parameters*: running parameters configured for a training job

**Figure 2-5** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image | Custom image

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

* Code Directory: /modelarts-train-test/gpu-train/ Select

* Boot File: /modelarts-train-test/gpu-train/train.py Select

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir Select

Input: ⊕ Add

Output: ⊕ Add

Hyperparameter: epochs = 5

⊕ Add

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

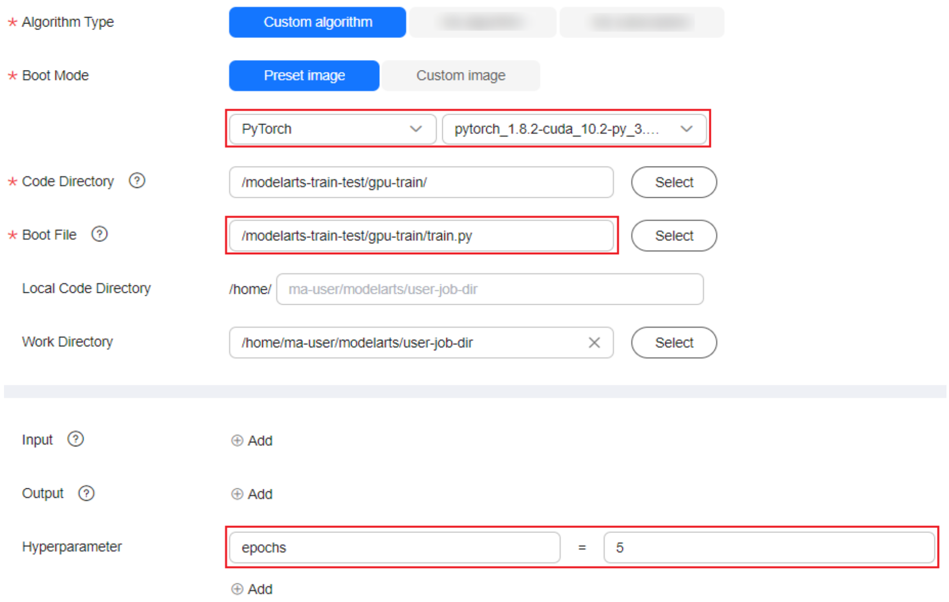
```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

- **Boot command for distributed jobs**

```
python --task_index ${VC_TASK_INDEX} --PS_hosts ${TF_PS_HOSTS} --worker_hosts ${TF_WORKER_HOSTS} --job_name ${MA_TASK_NAME} <Relative path of the startup file> <Job parameters>
```

  – *VC_TASK_INDEX*: task serial number, for example, **0/1/2**.
  – *TF_PS_HOSTS*: address array of PS nodes, for example, **[***xx***-PS-0.***xx***:***TCP_PORT***,***xx***-PS-1.***xx***:***TCP_PORT***]**. The value of **TCP_PORT** is a random port ranging from 5,000 to 10,000.
  – *TF_WORKER_HOSTS*: address array of worker nodes, for example, **[***xx***-worker-0.***xx***:***TCP_PORT***,***xx***-worker-1.***xx***:***TCP_PORT***]**. The value of **TCP_PORT** is a random port ranging from 5,000 to 10,000.
  – *MA_TASK_NAME*: task name, which can be PS or worker.
  – *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*
  – *Job parameters*: running parameters configured for a training job

**Figure 2-6** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image | Custom image

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

* Code Directory: /modelarts-train-test/gpu-train/ Select

* Boot File: /modelarts-train-test/gpu-train/train.py Select

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir Select

Input: ⊕ Add

Output: ⊕ Add

Hyperparameter: epochs = 5

⊕ Add

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

```
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
```

### 2.2.6.3 Horovod/MPI/MindSpore-GPU

ModelArts provides multiple AI frameworks for different engines. When you use these engines for model training, the algorithm codes during training need to be adapted accordingly. This section introduces how to make adptions to the Horovod/MPI/MindSpore-GPU engine.

## Horovod/MPI/MindSpore-GPU Startup Principle

**Specifications and number of nodes**

In this case, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** is used as an example to describe how to allocate ModelArts resources for single-node and distributed jobs.

For a single-node job (running on only one node), ModelArts starts a training container that exclusively uses the resources on the node.

For a distributed job (running on more than one node), there are as many workers as the nodes that are selected during job creation. Each worker is allocated with the compute resources of the selected specification. For example, if there are **2** compute nodes, two workers will be started, and each worker owns the compute resources of **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

**Network communication**

- For a single-node job, no network communication is required.
- For a distributed job, network communications are required in nodes and between nodes.

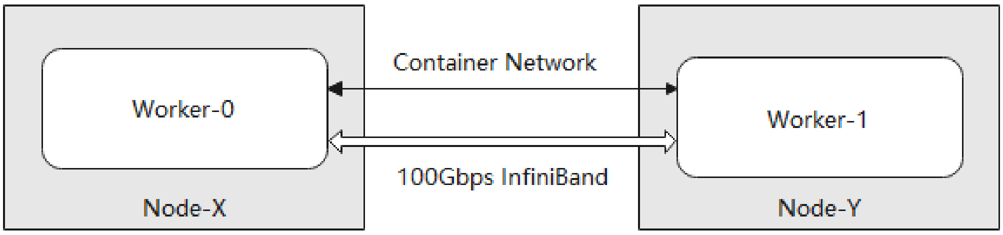
**In nodes**

NVLink and shared memory are used for communication.

**Between nodes**

If there is more than one compute node, PyTorch distributed training will be started. The following figure shows the network communications between workers in PyTorch distributed training. Workers can communicate with each other using the container network and a 100-Gbit/s InfiniBand or RoCE NIC. RoCE NICs are described specifically for certain specifications. The containers can communicate through DNS domain names, which is suitable for small-scale point-to-point communication that requires average network performance. The InfiniBand and RoCE NICs are suitable for distributed training jobs using collective communication that require high-performance network.

**Figure 2-7** Network communications for distributed training



## Boot Commands

By default, the training service uses the python interpreter in the job image to start up the training script. To obtain the python interpreter, run the **which python** command. The working directory during startup is **/home/ma-user/user-**

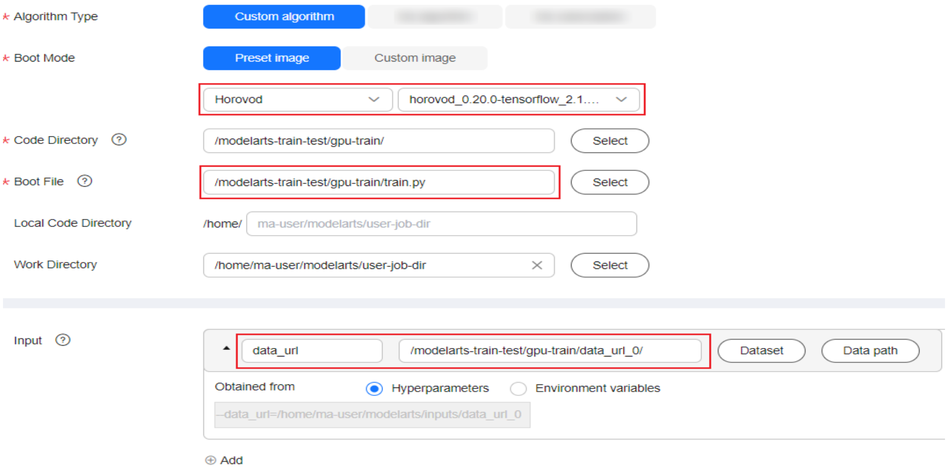
**job-dir/***<The code directory name>*, which is the directory returned by running **pwd** or **os.getcwd()** in python.

**Boot commands**

```
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 <Relative path of the startup file> <Job parameters>
```

- *OPENMPI_NP*: number of processes started by **mpirun**. The default value is the number of GPUs multiplied by the number of nodes. Do not modify this value.
- *OPENMPI_HOST_FILE_PATH*: value of **hostfile**. Do not modify this value.
- *SSHD_PORT*: Port for SSH login. Do not modify this value.
- *TUNE_ENV_FILE*: environment variables of worker-0. Broadcast the following environment variables to other worker nodes of the current training job.
  - **env** with the **MA_** prefix
  - **env** with the **SHARED_** prefix
  - **env** with the **S3_** prefix
  - **env** of **PATH**
  - **env** with the **VC_WORKER_** prefix
  - **env** with the **SCC** prefix
  - **env** with the **CRED** prefix
    ```
    env|grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED'|grep -v '=$'> $
    {TUNE_ENV_FILE}
    ```
- *OPENMPI_BIND_ARGS*: process pinning with the **mpirun cpu** command. The default settings are as follows:
  ```
  OPENMPI_BIND_ARGS="-bind-to none -map-by slot"
  ```
- *OPENMPI_X_ARGS*: **-x** parameters of the **mpirun** command. The default settings are as follows:
  ```
  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_X_ARGS*: **-mca** parameters of the **mpirun** command. The default settings are as follows:
  ```
  OPENMPI_MCA_ARGS="-mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true"
  ```
- *OPENMPI_EXTRA_ARGS*: parameters passed to **mpirun**. The default value is empty.
- *Relative path of the startup file*: path of the startup file relative to **/home/ma-user/user-job-dir/***<The code directory name>*
- *Job parameters*: running parameters configured for a training job

**Figure 2-8** Creating a training job



* Algorithm Type: Custom algorithm

* Boot Mode: Preset image | Custom image

Horovod | horovod_0.20.0-tensorflow_2.1....

* Code Directory: /modelarts-train-test/gpu-train/ Select

* Boot File: /modelarts-train-test/gpu-train/train.py Select

Local Code Directory: /home/ ma-user/modelarts/user-job-dir

Work Directory: /home/ma-user/modelarts/user-job-dir Select

Input: data_url /modelarts-train-test/gpu-train/data_url_0/ Dataset Data path

Obtained from ● Hyperparameters ○ Environment variables

--data_url=/home/ma-user/modelarts/inputs/data_url_0

⊕ Add

Configure the parameters by referring to the above figure. Then, run the following command on the console background:

```
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
```

NOTE

If you are using a Horovod, MPI, or MindSpore-GPU engine for model training, the boot commands for single-node jobs and distributed jobs are the same.

# 2.3 Inference Base Images

## 2.3.1 Available Inference Base Images

ModelArts inference provides a series of base images. You can create custom images based on these base images to deploy inference services.

## x86 (CPU/GPU)

**Table 2-14** TensorFlow

| AI Engine Version | Runtime Environment | URI |
|---|---|---|
| 2.1.0 | CPU<br>GPU (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_64-20221121111529-d65d817 |
| 1.15.5 | CPU<br>GPU (CUDA 11.4) | 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<br>GPU (CUDA 11.2) | 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 |

**Table 2-15** PyTorch

| AI Engine Version | Runtime Environment | URI |
|---|---|---|
| 1.8.0 | CPU<br>GPU (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_64-20221118143845-d65d817 |
| 1.8.2 | CPU<br>GPU (CUDA 11.1) | 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 |

**Table 2-16** MindSpore

| AI Engine Version | Runtime Environment | 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 (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_64-20220702120711-8590b76 |

| AI Engine Version | Runtime Environment | URI |
|---|---|---|
| 1.7.0 | GPU (CUDA 11.1) | 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 |

## 2.3.2 TensorFlow (CPU/GPU)-powered Inference Base Images

ModelArts provides the following inference base images powered by TensorFlow (CPU/GPU):

- **Engine Version 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Engine Version 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64**
- **Engine Version 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64**

### Engine Version 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
- Image creation time: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## Engine Version 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
- Image creation time: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 20.04.4 LTS
- CUDA: 11.4.3
- cuDNN: 8.2.4.15
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python, python 3.8.13
- Third-party package installation path: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## Engine Version 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- Image creation time: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.2.0
- cuDNN: 8.1.1.33
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## 2.3.3 PyTorch (CPU/GPU)-powered Inference Base Images

ModelArts provides the following inference base images powered by PyTorch (CPU/GPU):

- **Engine Version 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**
- **Engine Version 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

### Engine Version 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Image path: 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
- Image creation time: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.2.89
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## Engine Version 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- Image creation time: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## 2.3.4 MindSpore (CPU/GPU)-powered Inference Base Images

ModelArts provides the following inference base images powered by MindSpore (CPU/GPU):

- **Engine Version 1: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64**
- **Engine Version 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Engine Version 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

### Engine Version 1: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Image creation time: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
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
...
```

## Engine Version 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Image creation time: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

## Engine Version 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Image path: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Image creation time: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Path and version of the Python interpreter: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Third-party package installation path: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certain pip installation packages:

```
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
...
```

- Certain apt installation packages:

```
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
...
```

# 3 Using Custom Images in Notebook Instances

## 3.1 Registering an Image in ModelArts

After a custom image is created, register it on the ModelArts **Image Management** page before using it in notebook.

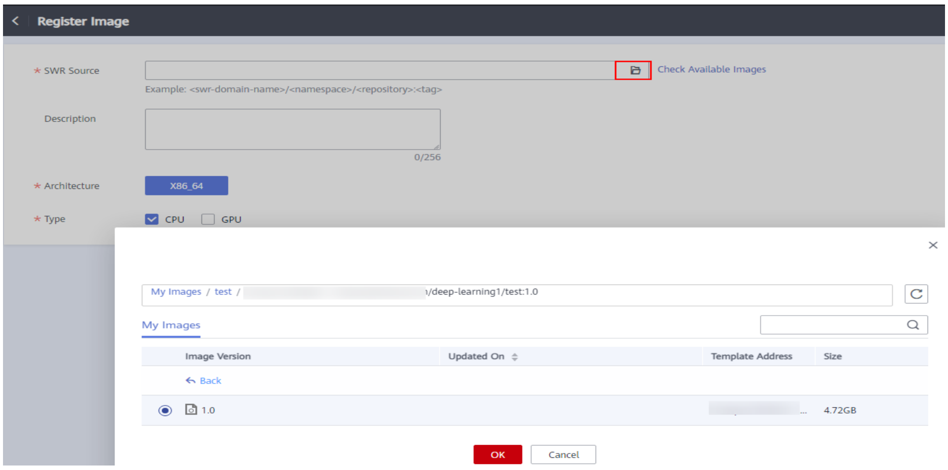
NOTE

Only the sub-users (IAM users) of the account can register and use the SWR images if the image type is **Private**.

Other users can register and use SWR images only when the image type is **Public**.

1. Log in to the ModelArts management console and choose **Image Management**. Then, click **Register**.
2. Configure parameters and click **Register**.
   - **SWR Source**: Select a built image as the image source. You can copy the complete SWR address or click 🗁 to select the target image for registration.

**Figure 3-1** Selecting an image source



Register Image

* SWR Source 🗁 Check Available Images

Example: <swr-domain-name>/<namespace>/<repository>:<tag>

Description

0/256

* Architecture X86_64

* Type ☑ CPU □ GPU

My Images / test / [redacted]/deep-learning1/test:1.0

My Images

| | Image Version | Updated On | Template Address | Size |
|---|---|---|---|---|
| | Back | | | |
| ● | 1.0 | | [redacted]... | 4.72GB |

   – **Architecture** and **Type**: Configure them based on the actual framework of the custom image.

3. View the registered image on the **Image Management** page.

**Figure 3-2** Image list

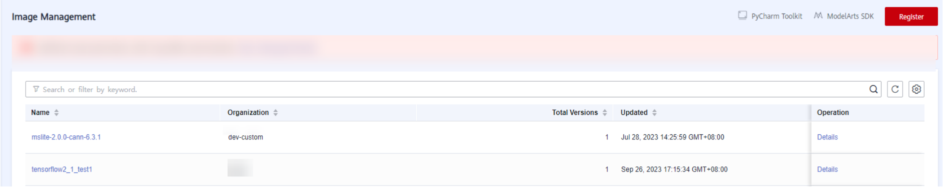


Image Management

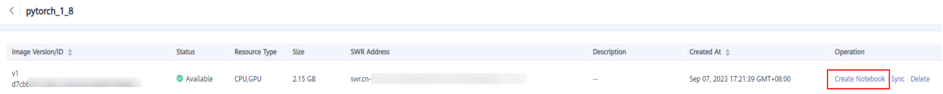
[redacted]

| Name | Organization | Total Versions | Updated | Operation |
|---|---|---|---|---|
| mslite-2.0.0-cann-6.3.1 | dev-custom | 1 | Jul 28, 2023 14:25:59 GMT+08:00 | Details |
| tensorflow2_1_test1 | [redacted] | 1 | Sep 26, 2023 17:15:34 GMT+08:00 | Details |

## Creating a Notebook Instance

Click the image name. On the image details page that appears, click **Create Notebook**. The page for creating a notebook instance using this image is displayed.

**Figure 3-3** Image details page



pytorch_1_8

| Image Version/ID | Status | Resource Type | Size | SWR Address | Description | Created At | Operation |
|---|---|---|---|---|---|---|---|
| v1<br>d7cb [redacted] | Available | CPU,GPU | 2.15 GB | swr.cn- [redacted] | -- | Sep 07, 2023 17:21:39 GMT+08:00 | Create Notebook Sync Delete |

## Synchronizing an Image

After the image fault is rectified, go to the image details page. Click **Sync** in the **Operation** column to refresh the image status.

# 3.2 Creating a Custom Image

You can create a custom image in any of the following ways:

- Method 1: Use a preset image of notebook instances to create a development environment instance. Then, install and configure dependencies in the environment. After the configuration, use the image saving function provided by the development environment to save the running instance as a custom container image. For details, see **Saving a Notebook Instance as a Custom Image**.
- Method 2: Use ModelArts base images and image creation templates to write a Dockerfile and create your own image on a notebook instance. Then, register the image to create a new development environment based on your needs. For details, see **Creating and Using a Custom Image in Notebook**.
- Method 3: Use ModelArts base images or third-party images to write a Dockerfile on an ECS and to reconstruct the ModelArts base images or third-party images. This allows you to customize Docker images that meet **ModelArts requirements** and push the images to SWR. For details, see **Creating a Custom Image on an ECS and Using It in Notebook**.

# 3.3 Saving a Notebook Instance as a Custom Image

## 3.3.1 Saving a Notebook Environment Image

To save a notebook environment image, do as follows: Create a notebook instance using a preset image, install custom software and dependencies on the base image, and save the running instance as a container image.

In the saved image, the installed dependencies are retained. The data stored in **home/ma-user/work** for persistent storage will not be stored. When you use VS Code for remote development, the plug-ins installed on the Server are retained.

NOTE

Images stored in a notebook instance cannot be larger than 35 GB and there cannot be more than 125 image layers. Otherwise, the image cannot be saved.

If error "The container size (xx) is greater than the threshold (25G)" is reported when an image is saved, handle the error by referring to **What Do I Do If Error "The container size (xG) is greater than the threshold (25G)" Is Displayed When I Save an Image?**.

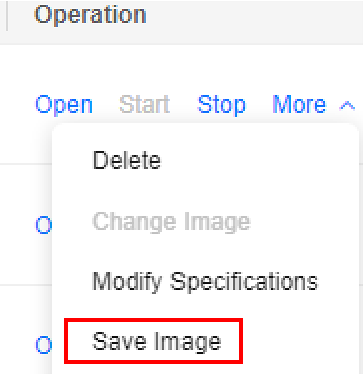
### Prerequisites

The notebook instance is in **Running** state.

### Saving an Image

1. Log in to the ModelArts management console and choose **DevEnviron** > **Notebook** in the navigation pane on the left to switch to notebook of the new version.
2. In the notebook instance list, select the target notebook instance and choose **Save Image** from the **More** drop-down list in the **Operation** column. The **Save Image** dialog box is displayed.

**Figure 3-4** Save Image

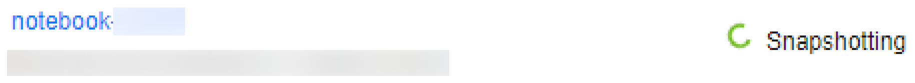


3. In the **Save Image** dialog box, configure parameters. Click **OK** to save the image.

   Choose an organization from the **Organization** drop-down list. If no organization is available, click **Create** on the right to create one.

   Users in an organization can share all images in the organization.
4. The image will be saved as a snapshot, and it will take about 5 minutes. During this period of time, do not perform any operations on the instance.

**Figure 3-5** Saving as a snapshot



notebook-[redacted] [redacted]

Snapshotting

**NOTICE**

The time required for saving an image as a snapshot will be counted in the instance running duration. If the instance running duration expires before the snapshot is saved, saving the image will fail.

5. After the image is saved, the instance status changes to **Running**. View the image on the **Image Management** page.
6. Click the name of the image to view its details.

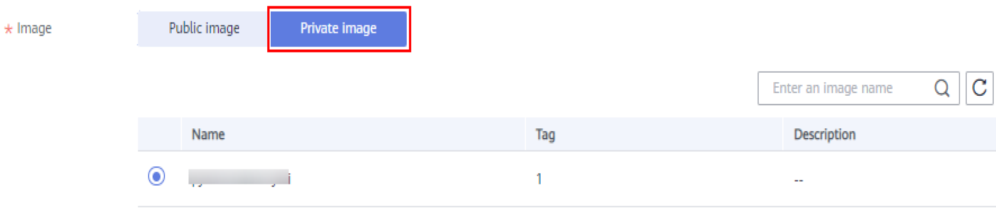
## 3.3.2 Using a Custom Image to Create a Notebook Instance

The images saved from a notebook instance can be viewed on the **Image Management** page. You can use these images to create new notebook instances, which inherit the software configurations of the original notebook instances.

You can use either of the following methods:

Method 1: On the **Create Notebook** page, click **Private Image** and select the saved image.

**Figure 3-6** Selecting a custom image to create a notebook instance



| | Name | Tag | Description |
|---|---|---|---|
| ● | [redacted] | 1 | -- |

Method 2: On the **Image Management** page, click the target image to access its details page. Then, click **Create Notebook**.

# 3.4 Creating and Using a Custom Image in Notebook

## 3.4.1 Application Scenarios and Process

If preset images cannot meet your service requirements, you can create container images based on the preset images for development and training.

Generally, you will need to reconstruct the ModelArts development environment, for example, by installing, upgrading, or uninstalling some packages. However, the root permission is required when certain packages are installed or upgraded. The running notebook instance does not have the root permission. As a result, you need to install the software that requires the root permission in the notebook instance, which is currently unavailable in the preset development environment.

You need to write a Dockerfile based on a preset public image to customize your image. Then, debug the image so that it can be used in ModelArts. At last, register the image with ModelArts so that it can be used to create development environments to meet your service requirements.

This example shows how to use **ma-cli** commands in ModelArts CLI to create and register a custom image for AI development with a PyTorch base image. For details, see **ma-cli Image Building Command**. The following figure shows the whole process.

**Figure 3-7** Creating an image

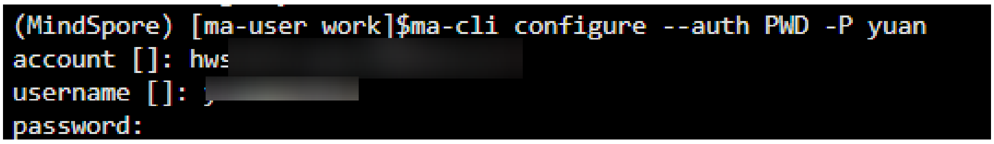
Build.
Register.
Use.

## 3.4.2 Step 1 Creating a Custom Image

This section shows you how to create an image by loading an image creation template and writing a Dockerfile. Ensure that you have created the development environment and opened a terminal on the **Notebook** page. For details about Dockerfiles, see **Dockerfile reference**.

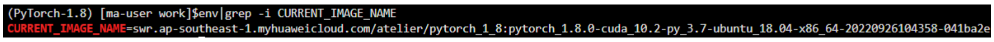
**Step 1** Configure authentication information, specify a profile, and enter the account information as prompted. For more information about authentication, see **ma-cli Authentication**.

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



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

**Step 2** Run **env|grep -i CURRENT_IMAGE_NAME** to query the image used by the current instance.



```
(PyTorch-1.8) [ma-user work]$env|grep -i CURRENT_IMAGE_NAME
CURRENT_IMAGE_NAME=swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e
```

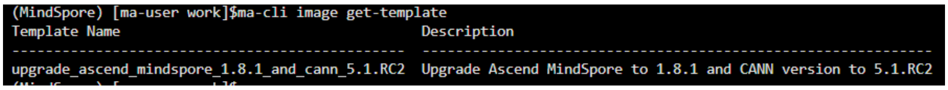
**Step 3** Create an image.

1. Obtain the SWR address of the base image.

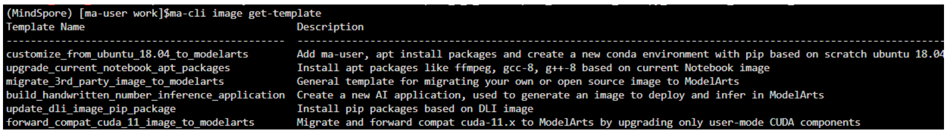
   **CURRENT_IMAGE_NAME**=swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

2. Load an image creation template.

   Run the **ma-cli image get-template** command to query the image template.



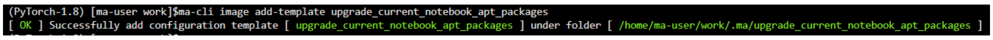
```
(MindSpore) [ma-user work]$ma-cli image get-template
Template Name                                   Description
----------------------------------------------  ------------------------------------------------------------
upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2  Upgrade Ascend MindSpore to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



```
(MindSpore) [ma-user work]$ma-cli image get-template
Template Name                                Description
-------------------------------------------  ---------------------------------------------------------------------------------
customize_from_ubuntu_18.04_to_modelarts     Add ma-user, apt install packages and create a new conda environment with pip based on scratch ubuntu 18.04
upgrade_current_notebook_apt_packages        Install apt packages like ffmpeg, gcc-8, g++-8 based on current Notebook image
migrate_3rd_party_image_to_modelarts         General template for migrating your own or open source image to ModelArts
build_handwritten_number_inference_application Create a new AI application, used to generate an image to deploy and infer in ModelArts
update_dli_image_pip_package                 Install pip packages based on DLI image
forward_compat_cuda_11_image_to_modelarts    Migrate and forward compat cuda-11.x to ModelArts by upgrading only user-mode CUDA components
```

Run the **ma-cli image add-template** command to load the image template to the specified folder. The default path is where the current command is located. For example, load the **upgrade_current_notebook_apt_packages** image creation template.

```
ma-cli image add-template upgrade_current_notebook_apt_packages
```



```
(PyTorch-1.8) [ma-user work]$ma-cli image add-template upgrade_current_notebook_apt_packages
[ OK ] Successfully add configuration template [ upgrade_current_notebook_apt_packages ] under folder [ /home/ma-user/work/.ma/upgrade_current_notebook_apt_packages ]
```

3. Modify a Dockerfile.

   The Dockerfile in this example is modified based on the base PyTorch image pytorch1.8-cuda10.2-cudnn7-ubuntu18.04, the image template upgrade_current_notebook_apt_packages is loaded, and GCC and G++ are upgraded.

   After the image template is loaded, the Dockerfile will be automatically loaded in **.ma/upgrade_current_notebook_apt_packages**. The content is as follows and you can modify it based on your needs.

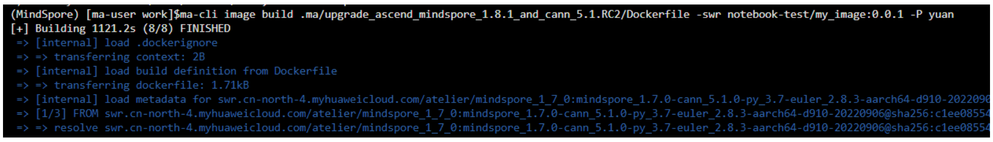
```
FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

# Set proxy to download internet resources
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

# Config apt source which can accelerate the apt package download speed. Also install ffmpeg and gcc-8 in root mode
RUN cp -f /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 update && \
    apt -y install ffmpeg && \
    apt install -y --no-install-recommends gcc-8 g++-8 && apt-get autoremove -y && \
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8

# ModelArts requires ma-user as the default user to start image
USER ma-user
```

4. Build an image.

   Run the **ma-cli image build** command to build an image with the Dockerfile. For more information, see **Creating an Image in ModelArts Notebook**.

```
ma-cli image build .ma/upgrade_current_notebook_apt_packages/Dockerfile -swr notebook-test/my_image:0.0.1 -P XXX
```

   The Dockerfile is stored in **.ma/upgrade_current_notebook_apt_package/Dockerfile** and the new image is stored in **notebook-test/my_image:0.0.1** in SWR. **XXX** indicates the profile specified for authentication.



```
(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-d910-2022090
 => [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:c1ee08554
 => => 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:c1ee08554
```

**----End**

## 3.4.3 Step 2 Registering a New Image

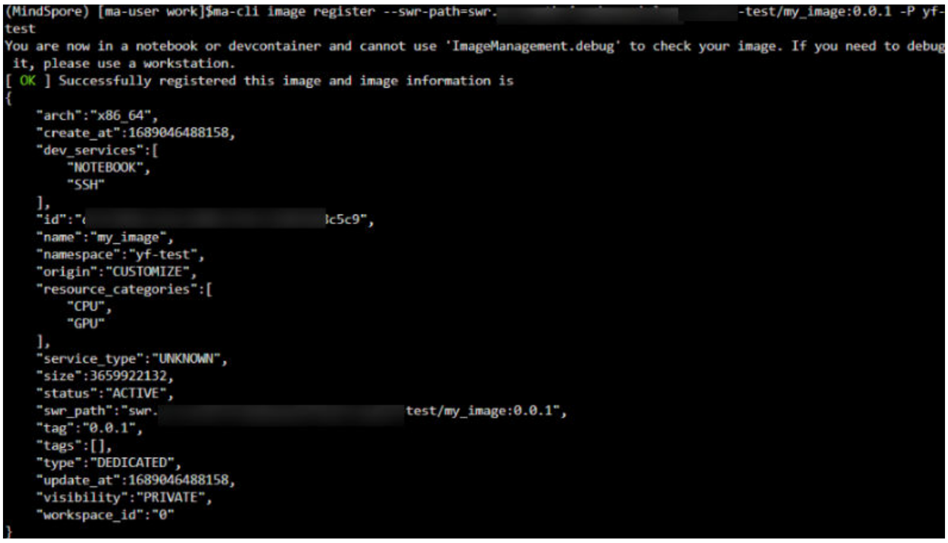
After an image is debugged, register it with ModelArts image management so that the image can be used in ModelArts.

Use either of the following methods to register the image with ModelArts:

- **Method 1**: Run the **ma-cli image register** command to register an image. Then, the information of the registered image is returned, including image ID and name, as shown in the following figure. For more information, see **Registering SWR Images with ModelArts Image Management**.

  ```
  ma-cli image register --swr-path=swr.ap-southeast-1.myhuaweicloud.com/notebook-test/
  my_image:0.0.1 -P XXX
  ```

  **Figure 3-8** Registered image

  ```
  (MindSpore) [ma-user work]$ma-cli image register --swr-path=swr.[redacted]-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":"[redacted]c5c9",
      "name":"my_image",
      "namespace":"yf-test",
      "origin":"CUSTOMIZE",
      "resource_categories":[
          "CPU",
          "GPU"
      ],
      "service_type":"UNKNOWN",
      "size":3659922132,
      "status":"ACTIVE",
      "swr_path":"swr.[redacted]test/my_image:0.0.1",
      "tag":"0.0.1",
      "tags":[],
      "type":"DEDICATED",
      "update_at":1689046488158,
      "visibility":"PRIVATE",
      "workspace_id":"0"
  }
  ```

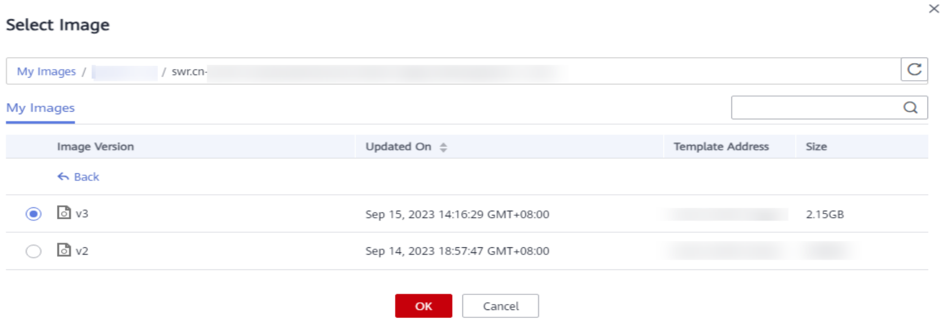
- **Method 2**: Register the image on the ModelArts management console.

  Log in to the ModelArts management console. In the navigation pane on the left, select **Image Management**. The **Image Management** page is displayed.

  Click **Register**. Paste the complete SWR address, or click 📂 to select a private image from SWR for registration, as shown in **Figure 3-9**.

  Select the architecture and type based on the site requirements. The architecture and type must be the same as those of the image source.

**Figure 3-9** Selecting an image



Select Image

My Images / [redacted] / swr.cn-[redacted]

My Images

| | Image Version | Updated On | Template Address | Size |
|---|---|---|---|---|
| | ← Back | | | |
| ● | v3 | Sep 15, 2023 14:16:29 GMT+08:00 | [redacted] | 2.15GB |
| ○ | v2 | Sep 14, 2023 18:57:47 GMT+08:00 | [redacted] | [redacted] |

## 3.4.4 Step 3 Using a New Image to Create a Development Environment

### Procedure

After an image is registered, it is available for development environment creation. You can log in to the ModelArts management console, choose **DevEnviron** > **Notebook**, and select the image during creation.

# 3.5 Creating a Custom Image on an ECS and Using It in Notebook

## 3.5.1 Application Scenarios and Process

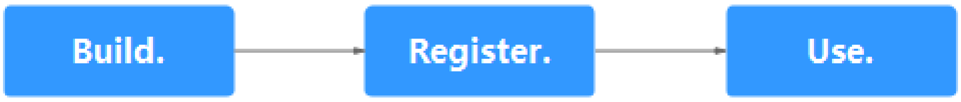
Generally, you will need to reconstruct the ModelArts development environment, for example, by installing, upgrading, or uninstalling some packages. However, the root permission is required when certain packages are installed or upgraded. The running notebook instance does not have the root permission. As a result, you need to install the software that requires the root permission in the notebook instance, which is currently unavailable in the preset development environment.

You can write a Dockerfile based on a preset base image or third-party image to customize your image. Then, you can register the image to create a new development environment based on your needs.

This section describes how to install PyTorch 1.8, FFmpeg 3, and GCC 8 on an Ubuntu image to create a new AI development environment.

The following figure shows the whole process.

**Figure 3-10** Creating and debugging an image



## 3.5.2 Step 1 Preparing a Docker Server and Configuring an Environment

Prepare a server with Docker enabled. If no such a server is available, create an ECS, buy an EIP, and install required software on it. Subsequent image building, debugging, and registration are all performed on this server.

ModelArts provides Ubuntu scripts for you to install Docker easier.

NOTE

The operations on the local Linux server are the same as those on the ECS. For details, see this case.

### Creating an ECS

- Log in to the ECS console and click **Buy ECS**. Select a public image (an Ubuntu 18.04 image is recommended) and set the system disk to 100 GiB. For more details, see **Purchasing and Logging In to a Linux ECS**.

  **Figure 3-11** Selecting an image and a disk

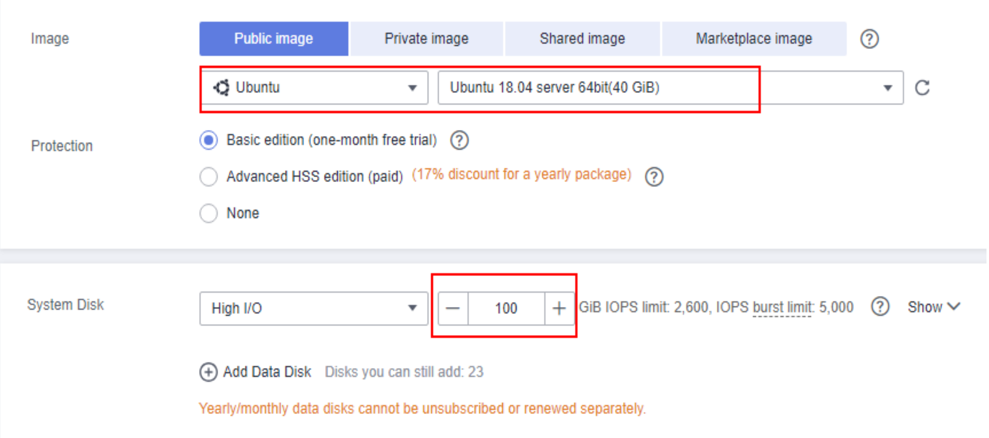
  

  Image: Public image | Private image | Shared image | Marketplace image
  Ubuntu — Ubuntu 18.04 server 64bit(40 GiB)
  Protection: ● Basic edition (one-month free trial) ○ Advanced HSS edition (paid) (17% discount for a yearly package) ○ None
  System Disk: High I/O — 100 GiB IOPS limit: 2,600, IOPS burst limit: 5,000 Show
  Add Data Disk Disks you can still add: 23
  Yearly/monthly data disks cannot be unsubscribed or renewed separately.

- Purchase an EIP and bind it to the ECS. For details, see **Configure Network**.

### Configuring an ECS

1. Run the following command on the Docker ECS to download the installation script:

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

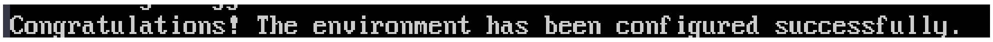
   NOTE

   Only Ubuntu scripts are supported.

2. Run the following command on the Docker ECS to configure the environment:

```
bash install_on_ubuntu1804.sh
```

   **Figure 3-12** Configured

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

```
source /etc/profile
```

The installation script is executed to:

a. Install Docker.

b. If the Docker ECS runs on GPUs, install nvidia-docker2 to mount the GPUs to the Docker container.

## 3.5.3 Step 2 Creating a Custom Image

This section describes how to edit a Dockerfile, use it to create an image, and use the created image to create a notebook instance. For details about how to edit a Dockerfile, see **Dockerfile reference**.

### Prerequisites

You have prepared a Docker server by referring to **Step 1 Preparing a Docker Server and Configuring an Environment**.

### Querying Base Images (Skip This Step for Third-Party Images)

For details about ModelArts base images, see **Notebook Base Image List**. Check the image URL in the corresponding section based on the engine type of the preset image.

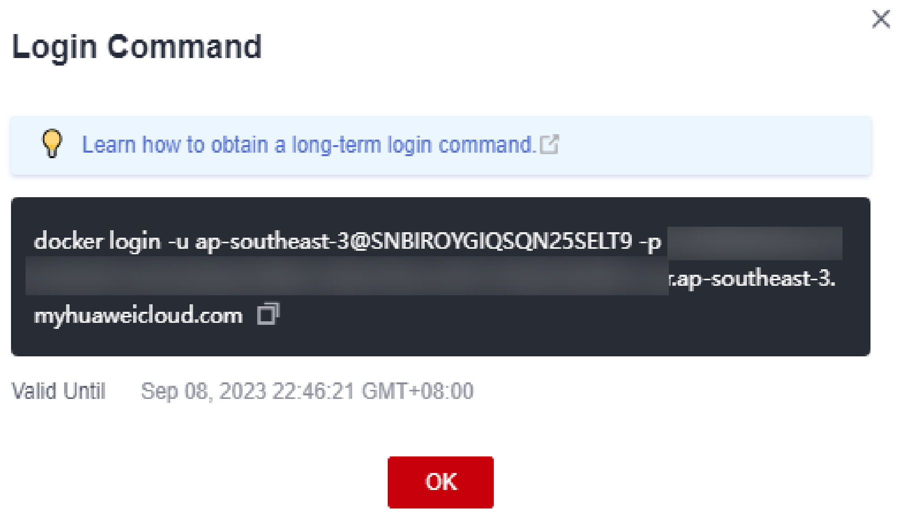
### Creating an Image

1. Access SWR.

   a. Log in to the SWR console.

   b. In the navigation pane on the left, choose **Dashboard**, and click **Generate Login Command** in the upper right corner. On the displayed page, copy the login command.

**Figure 3-13** Obtaining the login command



Login Command

Learn how to obtain a long-term login command.

```
docker login -u ap-southeast-3@SNBIROYGIQSQN25SELT9 -p [redacted] .ap-southeast-3.
myhuaweicloud.com
```

Valid Until Sep 08, 2023 22:46:21 GMT+08:00

NOTE

- The validity period of the generated login command is 24 hours. To obtain a long-term valid login command, see **Obtaining a Login Command with Long-Term Validity**. After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.
- The domain name at the end of the login command is the image repository address. Record the address for later use.

c. Run the login command on the machine where the container engine is installed.

The message "Login Succeeded" will be displayed upon a successful login.

2. Pull a base image or third-party image. The following uses a third-party image as an example.

```
docker pull swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/ubuntu:18.04 #Your organization name and image
```

3. Compile a Dockerfile.

   Run the **vim** command to create a Dockerfile. If a ModelArts base image is used, see **Dockerfile on a ModelArts Base Image** for details about the Dockerfile.

   If a third-party image is used, add user **ma-user** whose **UID** is **1000** and user group **ma-group** whose **GID** is **100**. For details, see **Dockerfile on a Non-ModelArts Base Image**.

   In this case, PyTorch 1.8, FFmpeg 3, and GCC 8 will be installed on an Ubuntu image to build an AI image.

4. Build an image.

   Run the **docker build** command to build a new image from the Dockerfile. The description of the command parameters are as follows:

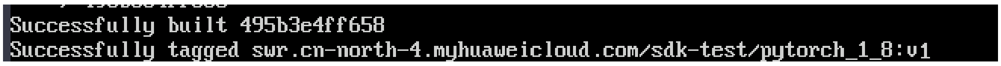
   – **-t** specifies the new image path, including region information, organization name, image name, and version. Set this parameter based on the real-life scenario. Use a complete SWR address for debugging and registration.

   – **-f** specifies the Dockerfile name. Set this parameter based on the real-life scenario.

   – **.** at the end specifies that the context is the current directory. Set this parameter based on the real-life scenario.

```
docker build -t swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/pytorch_1_8:v1 -f Dockerfile .
```

**Figure 3-14** Image created



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

## Dockerfile on a ModelArts Base Image

Run the **vim** command to create a Dockerfile. If the base image is provided by ModelArts, the content of the Dockerfile is as follows:

```
FROM swr.ap-southeast-1.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\ncustom_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 on a Non-ModelArts Base Image

If a third-party image is used, add user **ma-user** whose **UID** is **1000** and user group **ma-group** whose **GID** is **100** to the Dockerfile. If UID 1000 or GID 100 in the base image has been used by another user or user group, delete the user or user group. **The user and user group have been added to the Dockerfile in this case. You can directly use them.**

NOTE

You only need to set the user **ma-user** whose **UID** is **1000** and the user group **ma-group** whose **GID** is **100**, and grant the read, write, and execute permissions on the target directory to user **ma-user**.

Run the **vim** command to create a Dockerfile and add a third-party (non-ModelArts) image as the base image, for example, ubuntu 18.04. The content of the Dockerfile is as follows:

```
# 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 10
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/pkgs/main\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/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
```

## 3.5.4 Step 3 Registering a New Image

After an image is debugged, register it with ModelArts image management so that the image can be used in ModelArts.

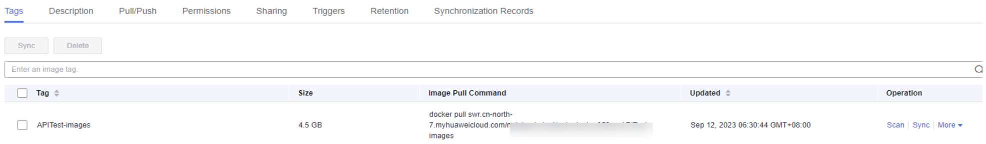
1. Push the image to SWR.

   Log in to SWR first. For details, see **Log In to SWR**. Run the following command to push the image:

   ```
   docker push swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/pytorch_1_8:v1
   ```

   The image is then available on SWR.

**Figure 3-15** Pushing the image to SWR



| ☐ Tag | Size | Image Pull Command | Updated | Operation |
|---|---|---|---|---|
| ☐ APITest-images | 4.5 GB | docker pull swr.cn-north-7.myhuaweicloud.com/n[redacted]images | Sep 12, 2023 06:30:44 GMT+08:00 | Scan \| Sync \| More ▾ |

2. Register an image.

### Registering an Image on the ModelArts Console

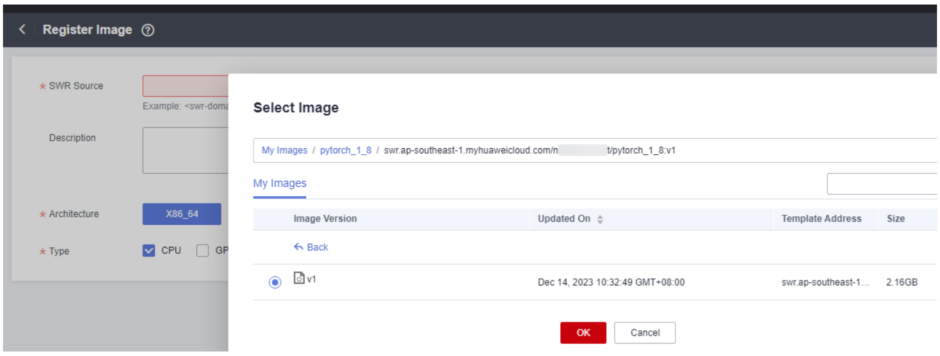
Log in to the ModelArts console. In the navigation pane on the left, select **Image Management** to access the image management page. Click **Register**.

Set **SWR Source** to the image pushed to SWR in **Step 1**. Click 📂 to select an existing image you want to register, as shown in **Figure 3-16**.

📖 NOTE

When you register a new image, ensure that the architecture and type are the same as those of the image source. Otherwise, the creation fails.

**Figure 3-16** Registering an image



My Images / pytorch_1_8 / swr.ap-southeast-1.myhuaweicloud.com/n[redacted]t/pytorch_1_8:v1

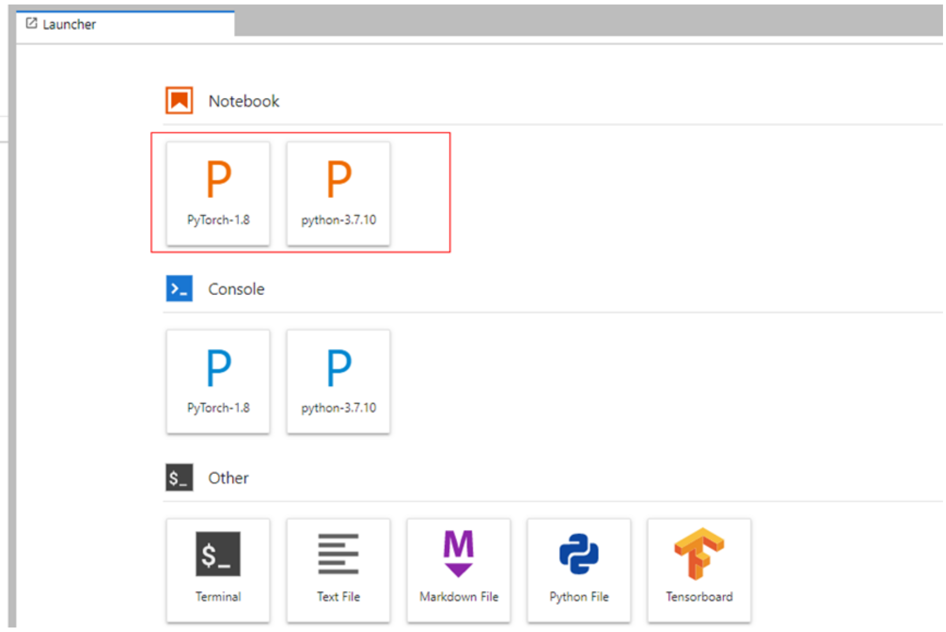
| | Image Version | Updated On | Template Address | Size |
|---|---|---|---|---|
| ● | v1 | Dec 14, 2023 10:32:49 GMT+08:00 | swr.ap-southeast-1... | 2.16GB |

## 3.5.5 Step 5 Creating and Starting a Development Environment

### Procedure

1. After the image is created, log in to the ModelArts console, go to the notebook tab, and choose the image registered in **Step 3 Registering a New Image** to create a development environment.
2. Go to the notebook list, click **Open** to start the created development environment.

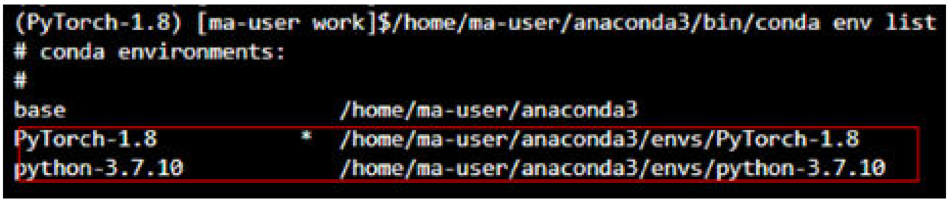
**Figure 3-17** Opening a development environment



3. Open a terminal to check the conda environment. For more information about conda, see the **official website**.

   Each kernel in the development environment is essentially a conda environment installed in **/home/ma-user/anaconda3/**. Run the **/home/ma-user/anaconda3/bin/conda env list** command to check the conda environment.

**Figure 3-18** Checking the conda environment



```
(PyTorch-1.8) [ma-user work]$/home/ma-user/anaconda3/bin/conda env list
# conda environments:
#
base                     /home/ma-user/anaconda3
PyTorch-1.8           *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
```

# 4 Using a Custom Image to Train Models (Model Training)

## 4.1 Overview

The subscribed algorithms and preset images can be used in most training scenarios. In certain scenarios, ModelArts allows you to create custom images to train models.

Customizing an image requires a deep understanding of containers. Use this method only if the subscribed algorithms and preset images cannot meet your requirements. Custom images can be used to train models in ModelArts only after they are uploaded to the Software Repository for Container (SWR).

You can use custom images for training on ModelArts in either of the following ways:

- Using a preset image with customization

  If you use a preset image to create a training job and you need to modify or add some software dependencies based on the preset image, you can customize the preset image. In this case, select a preset image and choose **Customize** from the framework version drop-down list box.
- Using a custom image

  You can create an image based on the ModelArts image specifications, select your own image and configure the code directory (optional) and boot command to create a training job.

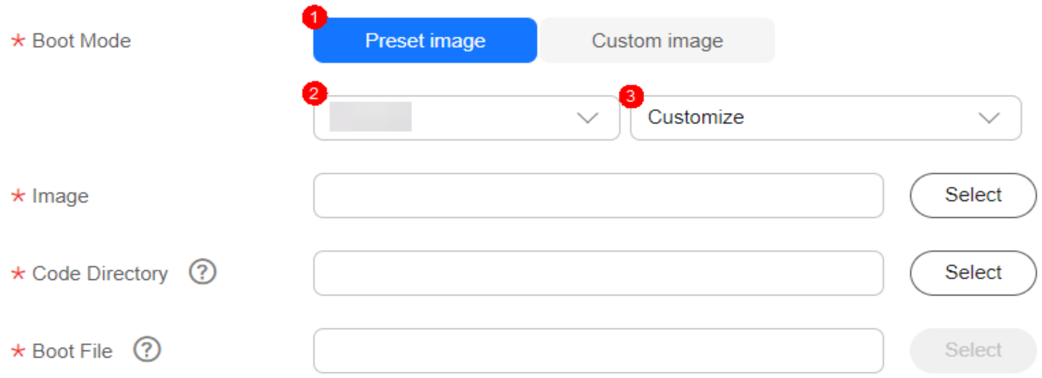
  **NOTE**

  When you use a custom image to create a training job, the boot command must be executed in the **/home/ma-user** directory. Otherwise, the training job may run abnormally.

### Using a Preset Image with Customization

The only difference between this method and creating a training job totally based on a preset image is that you must select an image. You can create a custom image based on a preset image.

**Figure 4-1** Creating an algorithm using a preset image with customization

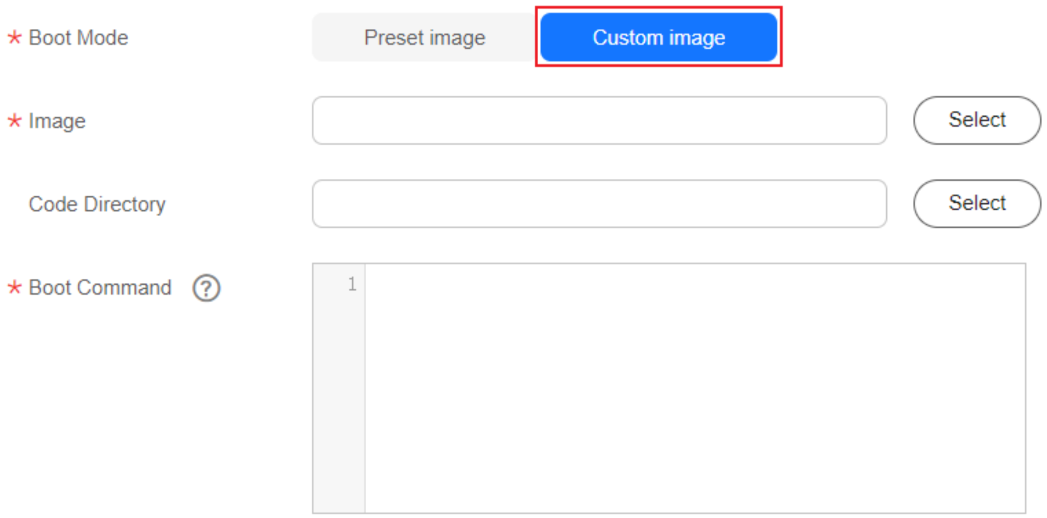


The process of this method is the same as that of creating a training job based on a preset image. For example:

- The system automatically injects environment variables.
  - PATH=${MA_HOME}/anaconda/bin:${PATH}
  - LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  - PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
- The selected boot file will be automatically started using Python commands. Ensure that the Python environment is correct. The PATH environment variable is automatically injected. Run the following commands to check the Python version for the training job:
  - export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  - docker run --rm {image} $(which python) -V
- The system automatically adds hyperparameters associated with the preset image.

## Using a Custom Image

**Figure 4-2** Creating an algorithm using a custom image



For details about how to use custom images supported by training, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

If all used images are customized, do as follows to use a specified Conda environment to start training:

Training jobs do not run in a shell. Therefore, you are not allowed to run the **conda activate** command to activate a specified Conda environment. In this case, use other methods to start training.

For example, Conda in your custom image is installed in the **/home/ma-user/anaconda3** directory, the Conda environment is **python-3.7.10**, and the training script is stored in **/home/ma-user/modelarts/user-job-dir/code/train.py**. Use a specified Conda environment to start training in one of the following ways:

- Method 1: Configure the correct **DEFAULT_CONDA_ENV_NAME** and **ANACONDA_DIR** environment variables for the image.

  Run the **python** command to start the training script. The following shows an example:

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

- Method 2: Use the absolute path of Conda environment Python.

  Run the **/home/ma-user/anaconda3/envs/python-3.7.10/bin/python** command to start the training script. The following shows an example:

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

- Method 3: Configure the path environment variable.

  Configure the bin directory of the specified Conda environment into the path environment variable. Run the **python** command to start the training script. The following shows an example:

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

- Method 4: Run the **conda run -n** command.

  Run the **/home/ma-user/anaconda3/bin/conda run -n python-3.7.10** command to execute the training. The following shows an example:

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

NOTE

If there is an error indicating that the .so file is unavailable in the **$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib** directory, add the directory to **LD_LIBRARY_PATH** and place the following command before the preceding boot command:

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

For example, the example boot command used in method 1 is as follows:

```
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 Example: Creating a Custom Image for Training

## 4.2.1 Example: Creating a Custom Image for Training (PyTorch + CPU/GPU)

This section describes how to create an image and use the image for training on the ModelArts platform. The AI engine used for training is PyTorch, and the resources are CPUs or GPUs.

**NOTE**

This section applies only to training jobs of the new version.

### Scenarios

In this example, create a custom image by writing a Dockerfile on a Linux x86_64 host running the Ubuntu 18.04 operating system.

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training on ModelArts.

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

### Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience. The following is the detailed procedure:

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing the Training Script and Uploading It to OBS**
4. **Step 3 Preparing a Host**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading an Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

### Prerequisites

You have registered a Huawei ID and enabled Huawei Cloud services, and the account is not in arrears or frozen.

### Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-1** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

**Table 4-1** Folder to create

| Name | Description |
|---|---|
| **obs://test-modelarts/pytorch/demo-code/** | Stores the training script. |
| **obs://test-modelarts/pytorch/log/** | Stores training log files. |

## Step 2 Preparing the Training Script and Uploading It to OBS

Prepare the training script **pytorch-verification.py** and upload it to the **obs://test-modelarts/pytorch/demo-code/** folder of the OBS bucket.

The **pytorch-verification.py** file contains the following information:

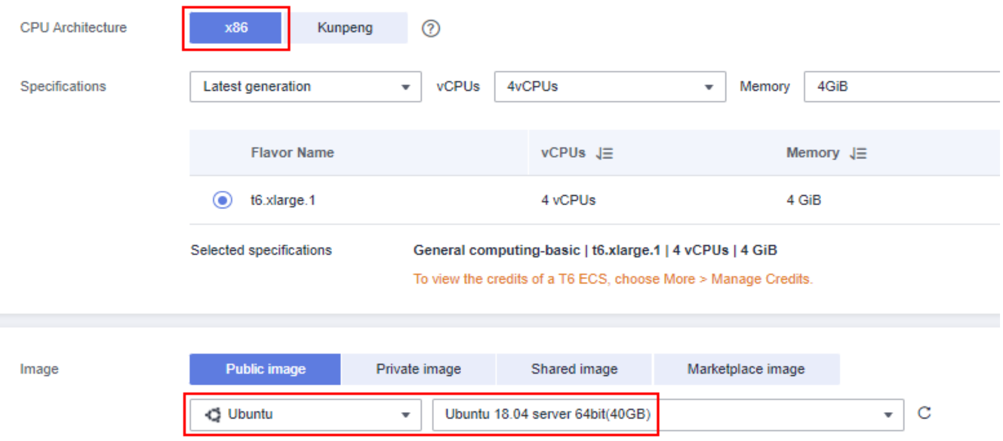
```
import torch
import torch.nn as nn

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)
```

## Step 3 Preparing a Host

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

**Figure 4-3** Creating an ECS using a public image (x86)



CPU Architecture: x86 | Kunpeng

Specifications: Latest generation; vCPUs: 4vCPUs; Memory: 4GiB

| Flavor Name | vCPUs | Memory |
|---|---|---|
| ● t6.xlarge.1 | 4 vCPUs | 4 GiB |

Selected specifications: General computing-basic | t6.xlarge.1 | 4 vCPUs | 4 GiB

To view the credits of a T6 ECS, choose More > Manage Credits.

Image: Public image | Private image | Shared image | Marketplace image

Ubuntu; Ubuntu 18.04 server 64bit(40GB)

## Step 4 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

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

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses the Linux x86_64 OS as an example to describe how to obtain the Docker installation package. For more details about how to install Docker, see **official Docker documents**.

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

   If the **docker images** command is executed, Docker has been installed. In this case, skip this step.

2. Run the following command to check the Docker Engine version:

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

   The following information is displayed:

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

   NOTE

   Use the Docker engine of the preceding version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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

   NOTE

   In Huawei Mirrors **https://mirrors.huaweicloud.com/home**, search for **pypi** to obtain the pip.conf file.

5. Download the following .whl files from https://download.pytorch.org/whl/torch_stable.html:

   - 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

   NOTE

   The URL code of the + symbol is %2B. When searching for a file in the above website, replace the + symbol in the file name with %2B.

   For example, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

6. Download the Miniconda3-py37_4.12.0-Linux-x86_64.sh installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

7. Store the pip source file, torch*.whl file, and Miniconda3 installation file in the **context** folder, which is as follows:

```
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. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The host must be connected to the public network for creating a container image.

# Base container image at 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

# The default user of the base container image is root.
# USER root

# Use the PyPI configuration provided by Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
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
# Install Miniconda3 to the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install torch*.whl using the default Miniconda3 Python environment in /home/ma-user/
miniconda3/bin/pip.
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

# Create the final container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim and cURL in Huawei Mirrors.
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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the base container image exists. User ma-user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the directory with the
same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1
```

```
# Set the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user
```

For details about how to write a Dockerfile, see **official Docker documents**.

9. Verify that the Dockerfile has been created. The following shows the **context** folder:

```
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. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **pytorch:1.8.1-cuda11.1**:

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

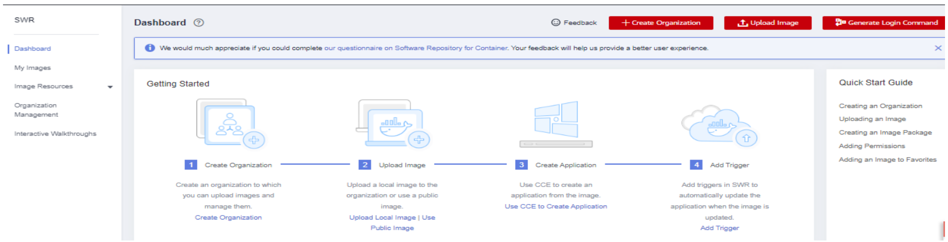
The following log information displayed during image creation indicates that the image has been created.

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

## Step 5 Uploading an Image to SWR

1. Log in to the SWR console and select the target region.

**Figure 4-4** SWR console



We would much appreciate if you could complete our questionnaire on Software Repository for Container. Your feedback will help us provide a better user experience.

Getting Started

1 Create Organization
Create an organization to which you can upload images and manage them.
Create Organization

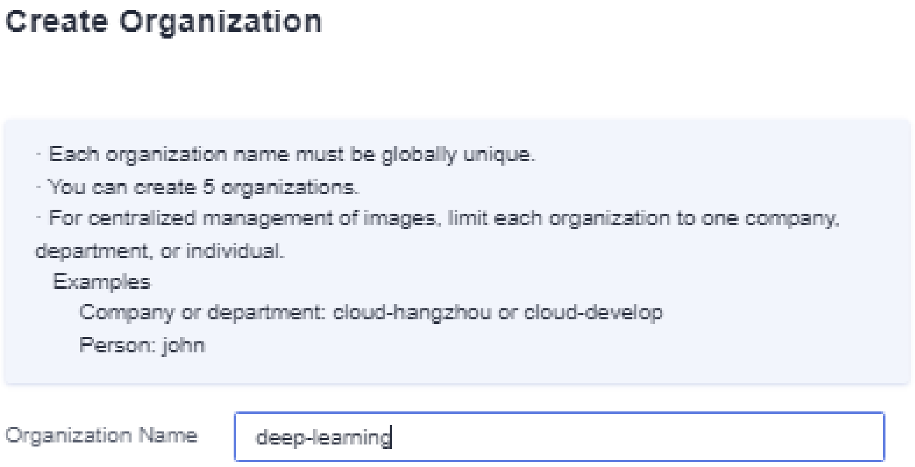
2 Upload Image
Upload a local image to the organization or use a public image.
Upload Local Image | Use Public Image

3 Create Application
Use CCE to create an application from the image.
Use CCE to Create Application

4 Add Trigger
Add triggers in SWR to automatically update the application when the image is updated.
Add Trigger

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

**Figure 4-5** Creating an organization



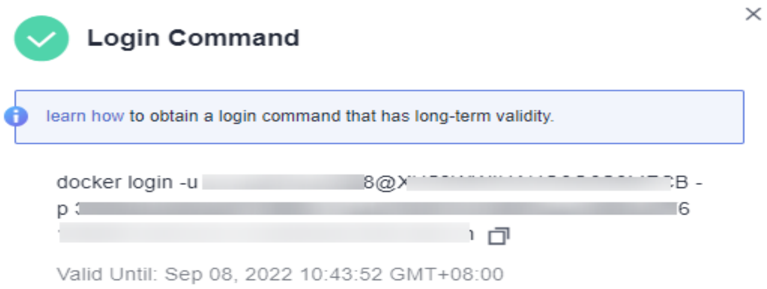
Create Organization

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.
  Examples
    Company or department: cloud-hangzhou or cloud-develop
    Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 4-6** Login Command



Login Command

learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted]8@[redacted]B -
p [redacted]6
[redacted]
```

Valid Until: Sep 08, 2022 10:43:52 GMT+08:00

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Run the following command to tag the uploaded image:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
   b. Run the following command to upload the image:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

## Step 6 Creating a Training Job on ModelArts

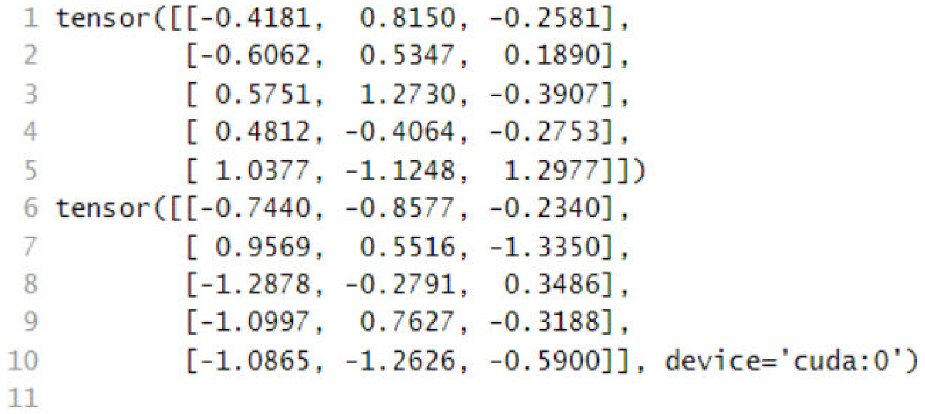
1. Log in to the ModelArts management console and check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. On the **Create Training Job** page, set required parameters and click **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Uploading an Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/pytorch/demo-code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py**. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Resource Pool**: **Public resource pools**

  - **Resource Type**: Select **CPU** or **GPU**.
  - **Persistent Log Saving**: enabled
  - **Job Log Path**: Set this parameter to the OBS path for storing training logs, for example, **obs://test-modelarts/pytorch/log/**.

4. Check the parameters of the training job and click **Submit**.
5. Wait until the training job is completed.

   After a training job is created, the operations such as container image downloading, code directory downloading, and boot command execution are automatically performed in the backend. Generally, the training duration ranges from dozens of minutes to several hours, depending on the training procedure and selected resources. After the training job is executed, the log similar to the following is output.

**Figure 4-7** Run logs of training jobs with GPU specifications



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

```

## 4.2.2 Example: Creating a Custom Image for Training (MPI + CPU/GPU)

This section describes how to create an image and use the image for training on the ModelArts platform. The AI engine used for training is MPI, and the resources are CPUs or GPUs.

NOTE

This section applies only to training jobs of the new version.

### Scenarios

In this example, create a custom image by writing a Dockerfile on a Linux x86_64 host running the Ubuntu 18.04 operating system.

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training on ModelArts.

- ubuntu-18.04
- cuda-11.1
- python-3.7.13

- openmpi-3.0.0

## Procedure

Before using a custom image to create a training job, get familiar with Docker and have development experience. The following is the detailed procedure:

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing Script Files and Uploading Them to OBS**
4. **Step 3 Preparing an Image Server**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading an Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

## Prerequisites

You have registered a Huawei ID and enabled Huawei Cloud services, and the account is not in arrears or frozen.

## Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-2** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

**Table 4-2** Folder to create

| Name | Description |
|---|---|
| **obs://test-modelarts/mpi/demo-code/** | Stores the MPI boot script and training script file. |
| **obs://test-modelarts/mpi/log/** | Stores training log files. |

## Step 2 Preparing Script Files and Uploading Them to OBS

Prepare the MPI boot script **run_mpi.sh** and training script **mpi-verification.py** and upload them to the **obs://test-modelarts/mpi/demo-code/** folder of the OBS bucket.

- The content of the MPI boot script **run_mpi.sh** is as follows:

```
#!/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/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 ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    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
```

**NOTE**

The script **run_mpi.sh** uses LF line endings. If CRLF line endings are used, executing the training job will fail, and the error "$'\r': command not found" will be displayed in logs.

- The content of the training script **mpi-verification.py** is as follows:

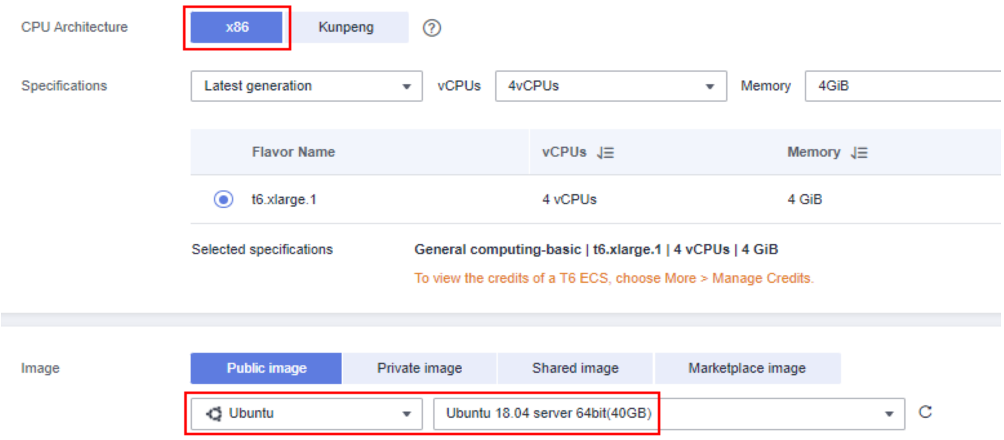
```
import os
import socket

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'])
```

## Step 3 Preparing an Image Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

**Figure 4-8** Creating an ECS using a public image (x86)



CPU Architecture: x86 | Kunpeng

Specifications: Latest generation | vCPUs: 4vCPUs | Memory: 4GiB

| | Flavor Name | vCPUs | Memory |
|---|---|---|---|
| ● | t6.xlarge.1 | 4 vCPUs | 4 GiB |

Selected specifications: General computing-basic | t6.xlarge.1 | 4 vCPUs | 4 GiB

To view the credits of a T6 ECS, choose More > Manage Credits.

Image: Public image | Private image | Shared image | Marketplace image

Ubuntu | Ubuntu 18.04 server 64bit(40GB)

## Step 4 Creating a Custom Image

Objective: Build and install container images of the following software and use the ModelArts training service to run the images.

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

The following describes how to create a custom image by writing a Dockerfile.

1. Install Docker.

   The following uses the Linux x86_64 OS as an example to describe how to obtain a Docker installation package. For more details, see **Docker official documents**. Run the following commands to install Docker:

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

   If the **docker images** command is executed, Docker has been installed. In this case, skip this step.

2. Check the Docker engine version. Run the following command:

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

   The following information is displayed:

   ```
    Engine:
     Version:          18.09.0
   ```

   NOTE

   You are advised to use Docker Engine of this version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

5. Download the openmpi 3.0.0 installation file.

   Download the openmpi 3.0.0 file edited using Horovod v0.22.1 from **https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz**.

6. Store the Miniconda3 and openmpi 3.0.0 files in the **context** folder. The following shows the **context** folder:

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

7. Write the Dockerfile of the container image.

   Create an empty file named **Dockerfile** in the **context** folder and write the following content to the file:

   ```
   # The host must be connected to the public network for creating a container image.

   # Basic container image at 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

   # The default user of the basic container image is root.
   # USER root

   # Copy the Miniconda3 (Python 3.7.13) installation files to the /tmp directory of the basic container
   image.
   COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

   # Install Miniconda3 to the /home/ma-user/miniconda3 directory of the basic container image.
   # 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

   # Create the final container image.
   FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

   # Install vim, cURL, net-tools, and the SSH tool in Huawei Mirrors.
   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

   # Install the Open MPI 3.0.0 file written using Horovod v0.22.1.
   # 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
   ```

```
# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the basic container image exists. User ma-user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# Set the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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
```

For details about how to write a Dockerfile, see **Docker official documents**.

8. Verify that the Dockerfile has been created. The following shows the **context** folder:

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

9. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **mpi:3.0.0-cuda11.1**:

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

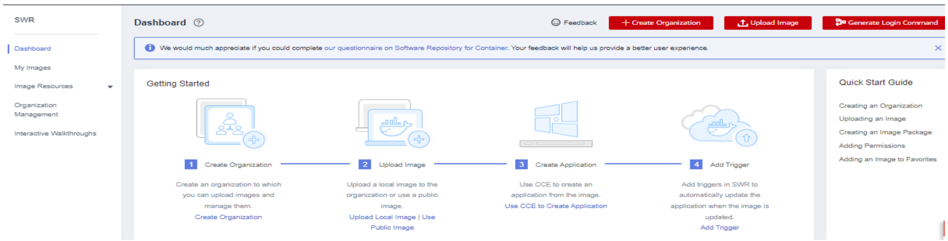
The following log information displayed during image creation indicates that the image has been created.

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

## Step 5 Uploading an Image to SWR

1. Log in to the SWR console and select the target region.

**Figure 4-9** SWR console



We would much appreciate if you could complete our questionnaire on Software Repository for Container. Your feedback will help us provide a better user experience.

Create an organization to which you can upload images and manage them.

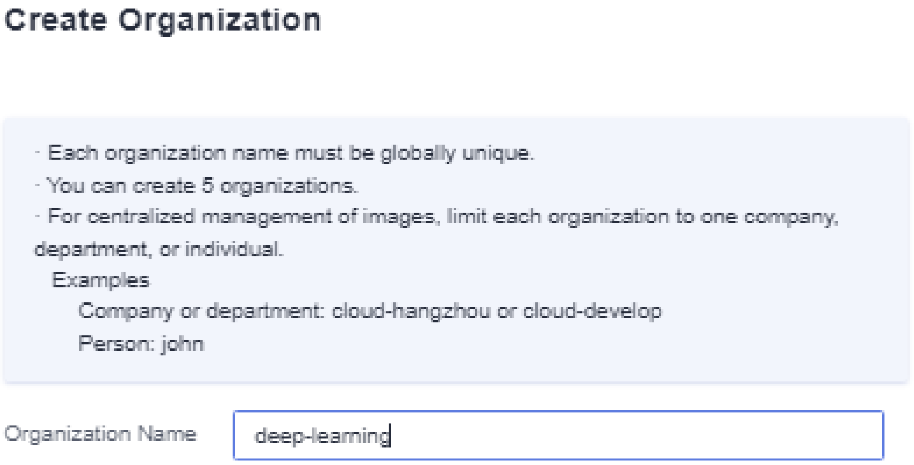
Upload a local image to the organization or use a public image.

Use CCE to create an application from the image.

Add triggers in SWR to automatically update the application when the image is updated.

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

**Figure 4-10** Creating an organization



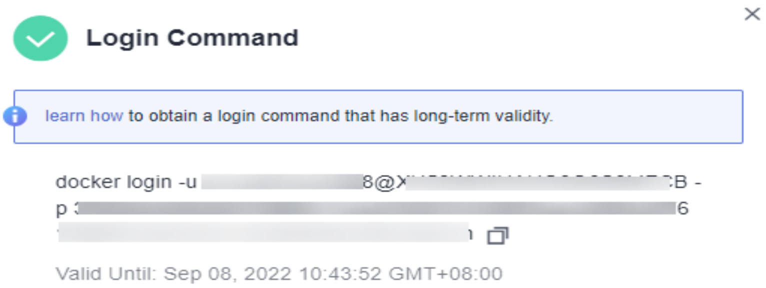
Create Organization

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.
  Examples
    Company or department: cloud-hangzhou or cloud-develop
    Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 4-11** Login Command



Login Command

learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted]
```

Valid Until: Sep 08, 2022 10:43:52 GMT+08:00

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Run the following command to tag the uploaded image:

```
#Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
```

   b. Run the following command to upload the image:

```
#Replace the region and domain information with the actual values, and replace the
organization name deep-learning with your custom value.
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
```

6. After the image is uploaded, choose **My Images** on the left navigation pane of the SWR console to view the uploaded custom images.

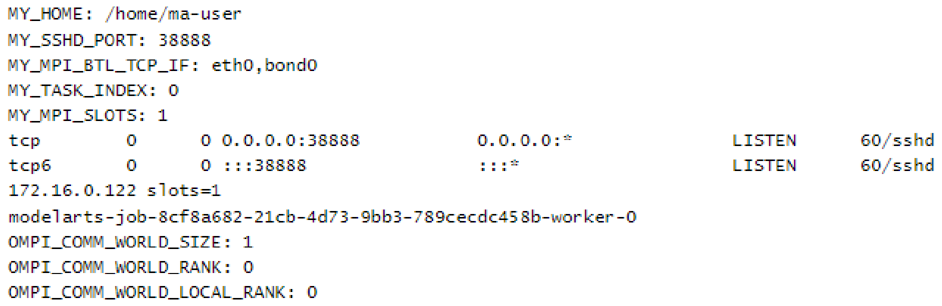
   **swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1** is the SWR URL of the custom image.

## Step 6 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management** > **Training Jobs (New)**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: **swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1**
   - **Code Directory**: OBS path to the boot script, for example, **obs://test-modelarts/mpi/demo-code/**.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py**
   - **Environment Variable**: Add **MY_SSHD_PORT = 38888**.
   - **Resource Pool**: **Public resource pools**
   - **Resource Type**: Select **GPU**.
   - **Compute Nodes**: Enter **1** or **2**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: Set this parameter to the OBS path for storing training logs, for example, **obs://test-modelarts/mpi/log/**.
4. Check the parameters of the training job and click **Submit**.
5. Wait until the training job is completed.

   After a training job is created, the operations such as container image downloading, code directory downloading, and boot command execution are automatically performed in the backend. Generally, the training duration ranges from dozens of minutes to several hours, depending on the training procedure and selected resources. After the training job is executed, the log similar to the following is output.

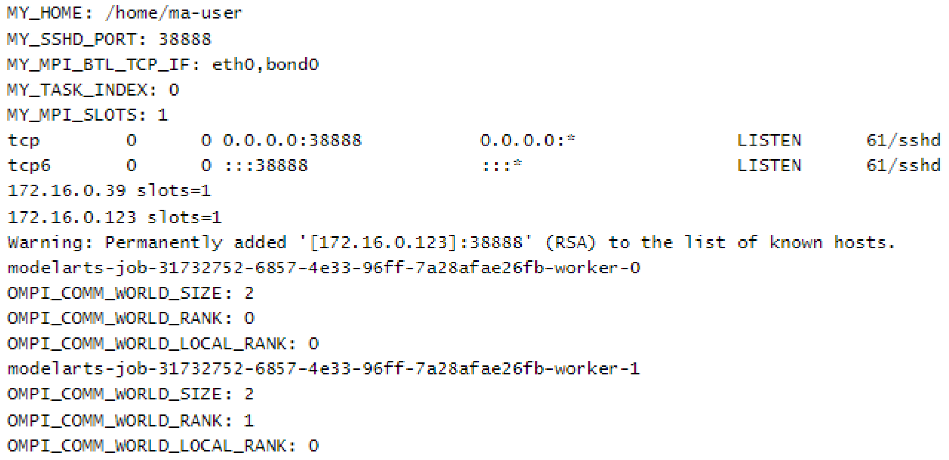
**Figure 4-12** Run logs of worker-0 with one compute node and GPU specifications



```
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
```

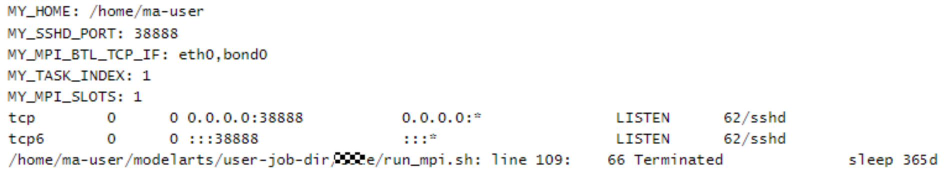
Set **Compute Nodes** to **2** and run the training job. **Figure 4-13** and **Figure 4-14** show the log information.

**Figure 4-13** Run logs of worker-0 with two compute nodes and GPU specifications



```
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
```

**Figure 4-14** Run logs of worker-1 with two compute nodes and GPU specifications



```
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/[redacted]e/run_mpi.sh: line 109:    66 Terminated              sleep 365d
```

## 4.2.3 Example: Creating a Custom Image for Training (Horovod-PyTorch and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is horovod_0.22.1-pytorch_1.8.1, and the resources used for training are GPUs.

**NOTE**

This section applies only to training jobs of the new version.

## Scenario

In this example, write a Dockerfile to create a custom image on a Linux x86_64 server running Ubuntu 18.04.

Create a container image with the following configurations and use the image to create a CPU- or GPU-powered training job on ModelArts:

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

## Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing the Training Script and Uploading It to OBS**
4. **Step 3 Preparing a Server**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading the Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

## Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

## Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-3** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

**Table 4-3** Folder to create

| Name | Description |
|---|---|
| **obs://test-modelarts/pytorch/demo-code/** | Stores the training script. |

| Name | Description |
|---|---|
| **obs://test-modelarts/pytorch/log/** | Stores training log files. |

## Step 2 Preparing the Training Script and Uploading It to OBS

Obtain training scripts **pytorch_synthetic_benchmark.py** and **run_mpi.sh** and upload them to **obs://test-modelarts/horovod/demo-code/** in the OBS bucket.

**pytorch_synthetic_benchmark.py** is as follows:

```
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** is as follows:

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

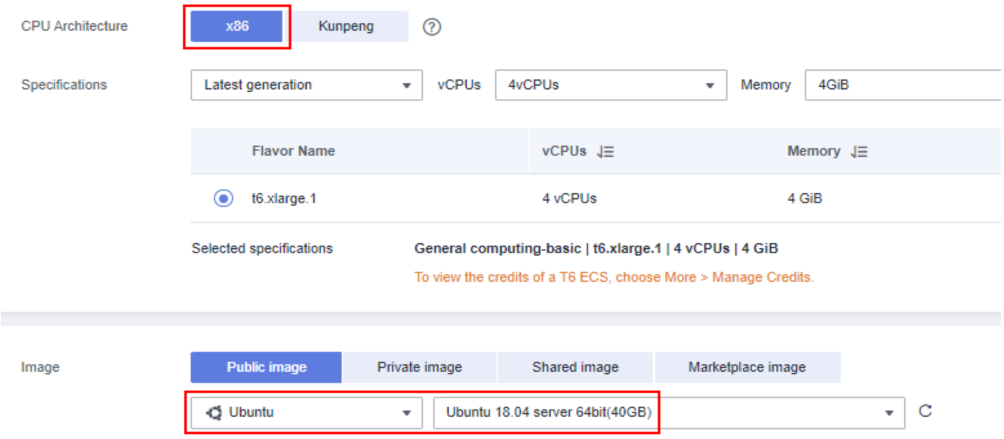
```
MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

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"
    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
```

## Step 3 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

**Figure 4-15** Creating an ECS using a public image (x86)



CPU Architecture: x86 | Kunpeng

Specifications: Latest generation | vCPUs: 4vCPUs | Memory: 4GiB

| Flavor Name | vCPUs | Memory |
|---|---|---|
| ● t6.xlarge.1 | 4 vCPUs | 4 GiB |

Selected specifications: General computing-basic | t6.xlarge.1 | 4 vCPUs | 4 GiB

To view the credits of a T6 ECS, choose More > Manage Credits.

Image: Public image | Private image | Shared image | Marketplace image

Ubuntu | Ubuntu 18.04 server 64bit(40GB)

## Step 4 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

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

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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

   If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:

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

   The following information is displayed:

   ```
    Engine:
     Version:          18.09.0
   ```

   NOTE

   Use the Docker engine of the preceding version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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

```
trusted-host = repo.huaweicloud.com
timeout = 120
```

**NOTE**

To obtain **pip.conf**, switch to Huawei Mirrors **https://mirrors.huaweicloud.com/home** and search for **pypi**.

5. Download the source Horovod code file.

   Download **horovod-0.22.1.tar.gz** from https://pypi.org/project/horovod/0.22.1/#files.

6. Download .whl files.

   Download the following .whl files from https://download.pytorch.org/whl/torch_stable.html.

   – 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

   **NOTE**

   The URL code of the plus sign (+) is %2B. When searching for files in the preceding websites, replace the plus sign (+) in the file name with %2B, for example, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

7. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

8. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The server on which the container image is created must access the Internet.

# Base container image at 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

# Install CMake obtained from Huawei Mirrors.
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

# The default user of the base container image is root.
# USER root

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
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
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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

# Environment variables required for building Horovod with PyTorch
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

# Install the .whl files using default Miniconda3 Python environment /home/ma-user/
miniconda3/bin/pip.
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

# Build and install horovod-0.22.1.tar.gz using default Miniconda3 Python environment /home/ma-
user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/horovod-0.22.1.tar.gz

# Create the container image.
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

# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from Huawei Mirrors.
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

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    PYTHONUNBUFFERED=1
```

For details about how to write a Dockerfile, see **official Docker documents**.

9. Download the MLNX_OFED installation package.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, in the **Download** tab, select a proper installation package from **Current Versions** or **Archive Versions**. In this example, choose **Archive Versions**, set **Version** to **5.4-3.5.8.0-LTS**, **OS Distribution** to **Ubuntu**, **OS Distribution Version** to **Ubuntu 18.04**, **Architecture** to **x86_64**, and download the **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz** installation package.

10. Download **openmpi-3.0.0-bin.tar.gz**.

    Download **openmpi-3.0.0-bin.tar.gz** from https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

11. Store the pip source file, .whl files, and Miniconda3 installation file in the **context** folder, which is as follows:

```
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. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **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
```

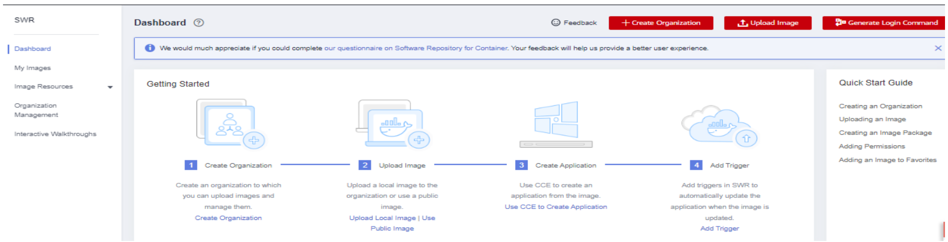
The following log shows that the image has been created.

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

## Step 5 Uploading the Image to SWR

1. Log in to the SWR console and select the target region.

**Figure 4-16** SWR console



We would much appreciate if you could complete our questionnaire on Software Repository for Container. Your feedback will help us provide a better user experience.

Create an organization to which you can upload images and manage them.
Create Organization

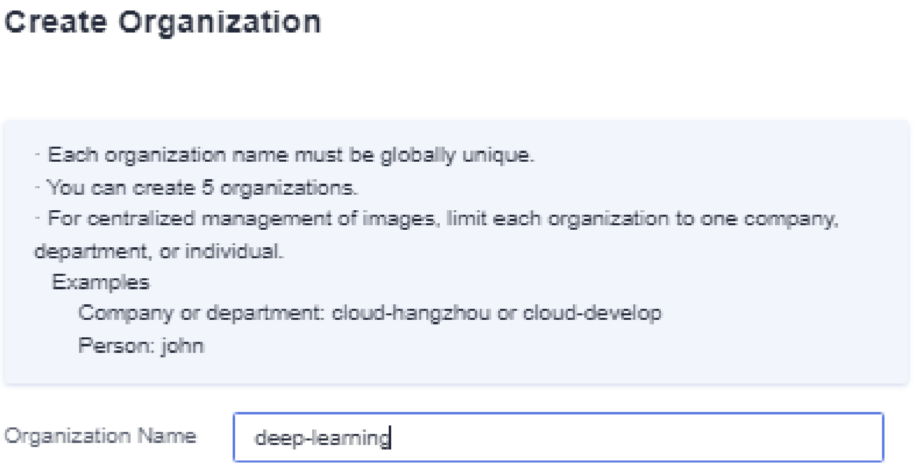
Upload a local image to the organization or use a public image.
Upload Local Image | Use Public Image

Use CCE to create an application from the image.
Use CCE to Create Application

Add triggers in SWR to automatically update the application when the image is updated.
Add Trigger

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

**Figure 4-17** Creating an organization



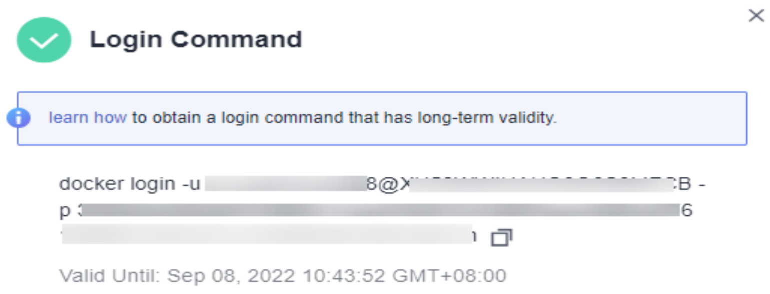
Create Organization

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.
  Examples
  Company or department: cloud-hangzhou or cloud-develop
  Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 4-18** Login Command



Login Command

learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted] -p [redacted]
```

Valid Until: Sep 08, 2022 10:43:52 GMT+08:00

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   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
   ```
   b. Run the following command to upload the image:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

## Step 6 Creating a Training Job on ModelArts

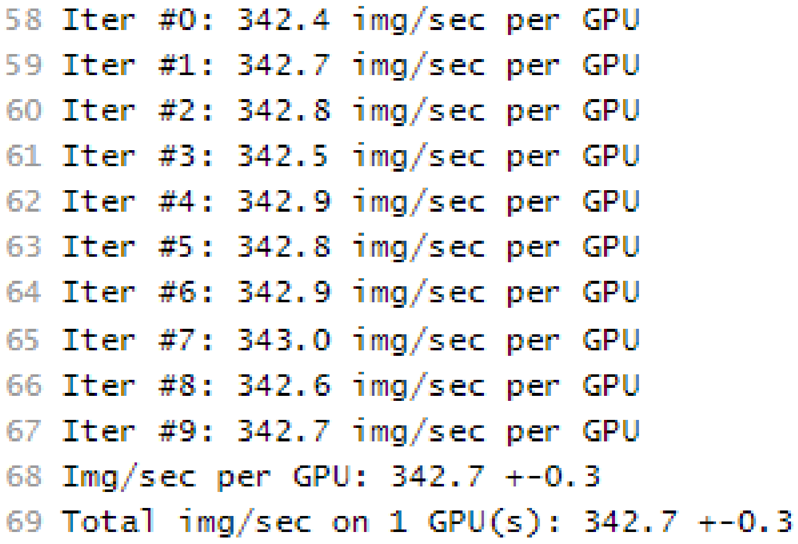
1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Uploading the Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/pytorch/demo-code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py**. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Environment Variable**: Click **Add Environment Variable** and add the environment variable **MY_SSHD_PORT=38888**.
   - **Resource Pool**: Select **Public resource pools**.

- **Resource Type**: Select **GPU**.
- **Compute Nodes**: 1 or 2
- **Persistent Log Saving**: enabled
- **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/pytorch/log/**

4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

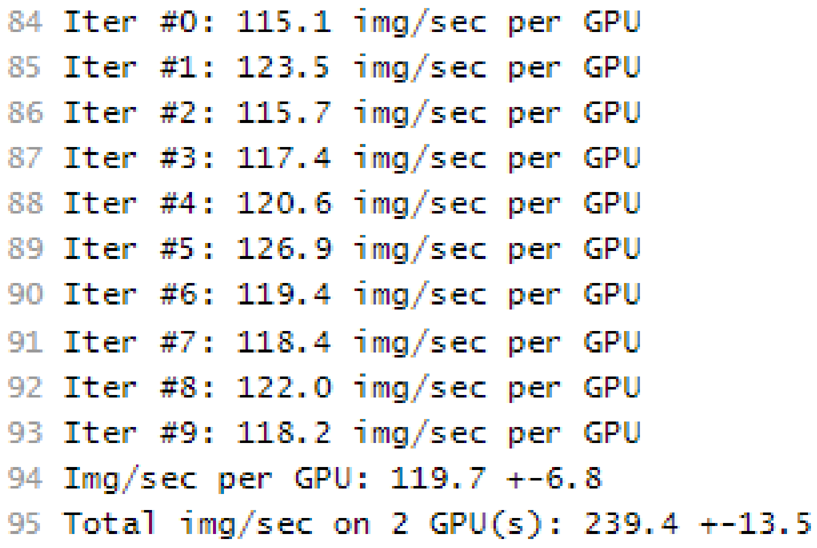
   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

**Figure 4-19** Run logs of training jobs with GPU specifications (one compute node)



```
Iter #0: 342.4 img/sec per GPU
Iter #1: 342.7 img/sec per GPU
Iter #2: 342.8 img/sec per GPU
Iter #3: 342.5 img/sec per GPU
Iter #4: 342.9 img/sec per GPU
Iter #5: 342.8 img/sec per GPU
Iter #6: 342.9 img/sec per GPU
Iter #7: 343.0 img/sec per GPU
Iter #8: 342.6 img/sec per GPU
Iter #9: 342.7 img/sec per GPU
Img/sec per GPU: 342.7 +-0.3
Total img/sec on 1 GPU(s): 342.7 +-0.3
```

**Figure 4-20** Run logs of training jobs with GPU specifications (two compute nodes)



```
Iter #0: 115.1 img/sec per GPU
Iter #1: 123.5 img/sec per GPU
Iter #2: 115.7 img/sec per GPU
Iter #3: 117.4 img/sec per GPU
Iter #4: 120.6 img/sec per GPU
Iter #5: 126.9 img/sec per GPU
Iter #6: 119.4 img/sec per GPU
Iter #7: 118.4 img/sec per GPU
Iter #8: 122.0 img/sec per GPU
Iter #9: 118.2 img/sec per GPU
Img/sec per GPU: 119.7 +-6.8
Total img/sec on 2 GPU(s): 239.4 +-13.5
```

## 4.2.4 Example: Creating a Custom Image for Training (MindSpore and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is MindSpore, and the resources used for training are GPUs.

NOTE

This section applies only to training jobs of the new version.

### Scenario

In this example, write a Dockerfile to create a custom image on a Linux x86_64 server running Ubuntu 18.04.

Create a container image with the following configurations and use the image to create a GPU-powered training job on ModelArts:

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

### Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

- **Prerequisites**
- **Step 1 Creating an OBS Bucket and Folder**

- Step 2 Creating a Dataset and Uploading It to OBS
- Step 3 Preparing the Training Script and Uploading It to OBS
- Step 4 Preparing a Server
- Step 5 Creating a Custom Image
- Step 6 Uploading the Image to SWR
- Step 7 Creating a Training Job on ModelArts

## Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

## Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-4** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS and ModelArts are in the same region.

**Table 4-4** Required OBS folders

| Folder | Description |
|---|---|
| **obs://test-modelarts/mindspore-gpu/resnet/** | Stores the training script. |
| **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** | Stores dataset files. |
| **obs://test-modelarts/mindspore-gpu/output/** | Stores training output files. |
| **obs://test-modelarts/mindspore-gpu/log/** | Store training log files. |

## Step 2 Creating a Dataset and Uploading It to OBS

Go to http://www.cs.toronto.edu/~kriz/cifar.html, download the **CIFAR-10 binary version (suitable for C programs)** package, decompress it, and upload the decompressed data to the **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** directory in the OBS bucket.

## Step 3 Preparing the Training Script and Uploading It to OBS

Obtain the ResNet file and script **run_mpi.sh** and upload them to **obs://test-modelarts/mindspore-gpu/resnet/** in the OBS bucket.

Download the ResNet file from **https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet**.

**run_mpi.sh** is as follows:

```
#!/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"
    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
```

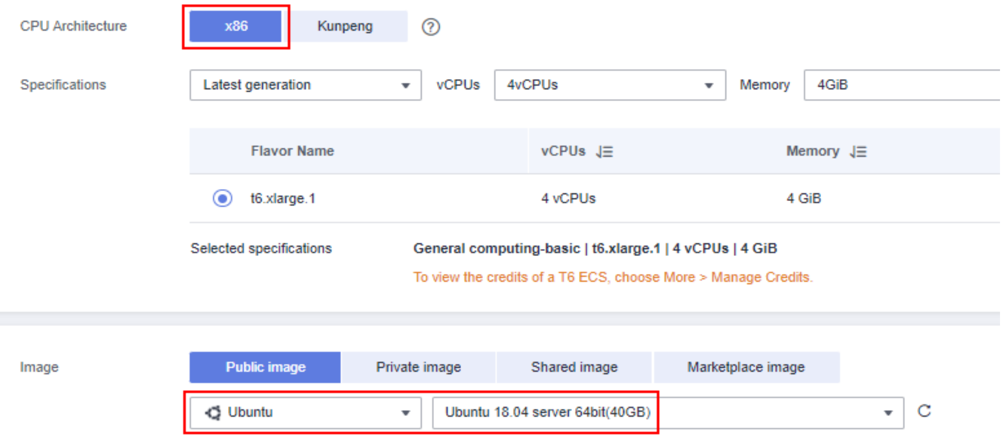
The **obs://test-modelarts/mindspore-gpu/resnet/** folder contains files **resnet** and **run_mpi.sh**.

## Step 4 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

**Figure 4-21** Creating an ECS using a public image (x86)



CPU Architecture: x86 | Kunpeng

Specifications: Latest generation · vCPUs: 4vCPUs · Memory: 4GiB

| | Flavor Name | vCPUs | Memory |
|---|---|---|---|
| ● | t6.xlarge.1 | 4 vCPUs | 4 GiB |

Selected specifications: General computing-basic \| t6.xlarge.1 \| 4 vCPUs \| 4 GiB

To view the credits of a T6 ECS, choose More > Manage Credits.

Image: Public image | Private image | Shared image | Marketplace image

Ubuntu · Ubuntu 18.04 server 64bit(40GB)

## Step 5 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

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

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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

   If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:

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

   The following information is displayed:

   ```
    Engine:
     Version:          18.09.0
   ```

   NOTE

   Use the Docker engine of the preceding version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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

NOTE

To obtain **pip.conf**, switch to Huawei Mirrors **https://mirrors.huaweicloud.com/home** and search for **pypi**.

5. Download **mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl** from **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**.
6. Download the Miniconda3 installation file.

   Download **Miniconda3-py37_4.12.0-Linux-x86_64.sh** from https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.
7. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The server on which the container image is created must access the Internet.

# Base container image at 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

# The default user of the base container image is root.
# USER root

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
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
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the whl file using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/
pip.
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

# Create the container image.
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

# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from Huawei Mirrors.
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

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-user can directly run
the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the directory with the
same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
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
```

For details about how to write a Dockerfile, see **official Docker documents**.

8. Download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, click **Download**, set **Version** to **5.4-3.5.8.0-LTS**, **OSDistributionVersion** to **Ubuntu 18.04**, and **Architecture** to **x86_64**, and download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Download **openmpi-3.0.0-bin.tar.gz**.

Download **openmpi-3.0.0-bin.tar.gz** from https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

10. Store the Dockerfile and Miniconda3 installation file in the **context** folder, which is as follows:

```
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. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **mindspore:1.8.1-ofed-cuda11.1**:

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

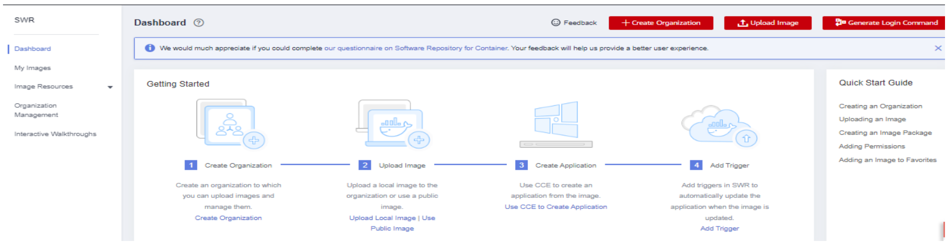
The following log shows that the image has been created.

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

## Step 6 Uploading the Image to SWR

1. Log in to the SWR console and select the target region.

**Figure 4-22** SWR console



We would much appreciate if you could complete our questionnaire on Software Repository for Container. Your feedback will help us provide a better user experience.

Getting Started

1 Create Organization

Create an organization to which you can upload images and manage them.

Create Organization

2 Upload Image

Upload a local image to the organization or use a public image.

Upload Local Image | Use Public Image

3 Create Application

Use CCE to create an application from the image.

Use CCE to Create Application

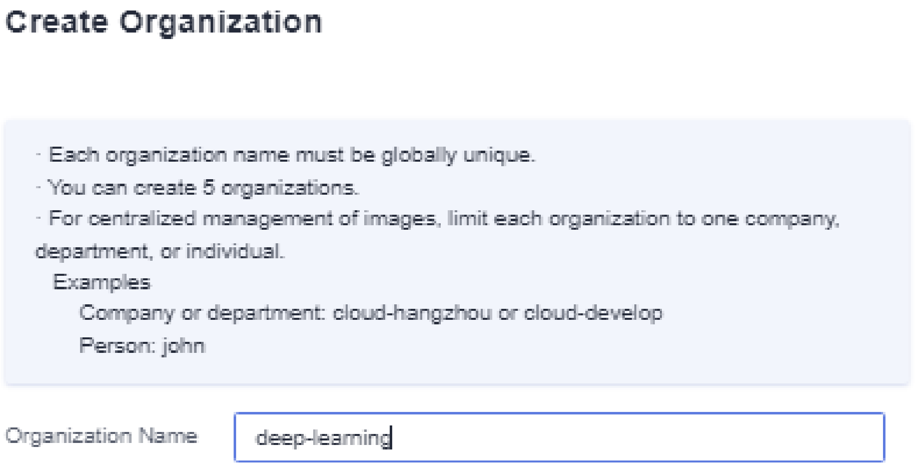
4 Add Trigger

Add triggers in SWR to automatically update the application when the image is updated.

Add Trigger

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

**Figure 4-23** Creating an organization



Create Organization

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.

Examples

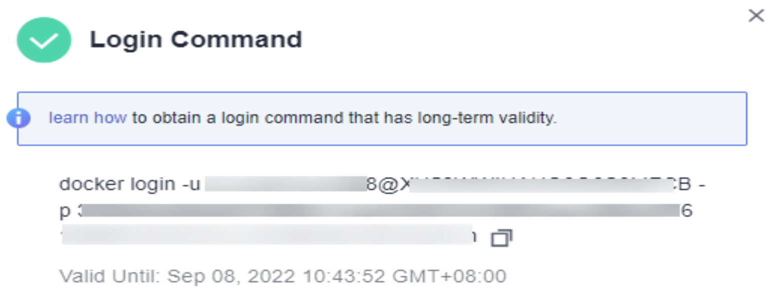
Company or department: cloud-hangzhou or cloud-develop

Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 4-24** Login Command



Login Command

learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted] -p [redacted]
```

Valid Until: Sep 08, 2022 10:43:52 GMT+08:00

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
   ```
   b. Run the following command to upload the image:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

## Step 7 Creating a Training Job on ModelArts

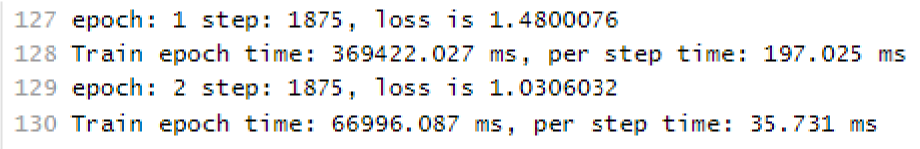
1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 6 Uploading the Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/mindspore-gpu/resnet/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/resnet** directory of the training container. **resnet** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **bash ${MA_JOB_DIR}/resnet/run_mpi.sh python ${MA_JOB_DIR}/resnet/train.py**. **resnet** (customizable) is the last-level directory of the OBS path.
   - **Training Input**: Click **Add Training Input**. Enter **data_path** for the name, select the OBS path to the target dataset, for example, **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/**, and set **Obtained from** to **Hyperparameters**.

- **Training Output**: Click **Add Training Output**. Enter **output_path** for the name, select an OBS path for storing training outputs, for example, **obs://test-modelarts/mindspore-gpu/output/**, and set **Obtained from** to **Hyperparameters** and **Predownload** to **No**.
- **Hyperparameters**: Click **Add Hyperparameter** and add the following hyperparameters:
  - run_distribute=True
  - device_num=1 (Set this parameter based on the number of GPUs in the instance flavors.)
  - device_target=GPU
  - epoch_size=2
- **Environment Variable**: Click **Add Environment Variable** and add the environment variable **MY_SSHD_PORT=38888**.
- **Resource Pool**: Select **Public resource pools**.
- **Resource Type**: Select **GPU**.
- **Compute Nodes**: 1 or 2
- **Persistent Log Saving**: enabled
- **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mindspore-gpu/log/**

4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

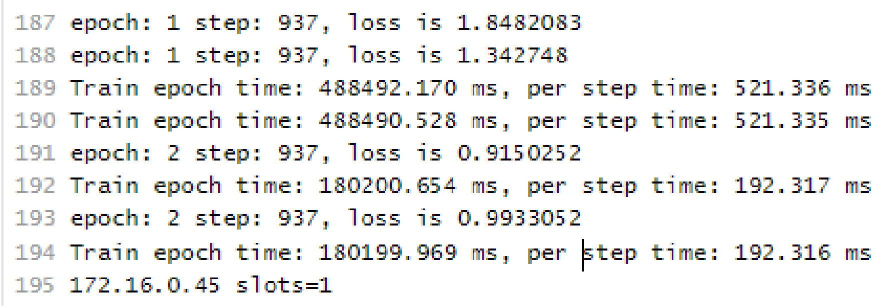
   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

   **Figure 4-25** Run logs of training jobs with GPU specifications (one compute node)



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

**Figure 4-26** Run logs of training jobs with GPU specifications (two compute nodes)



```
epoch: 1 step: 937, loss is 1.8482083
epoch: 1 step: 937, loss is 1.342748
Train epoch time: 488492.170 ms, per step time: 521.336 ms
Train epoch time: 488490.528 ms, per step time: 521.335 ms
epoch: 2 step: 937, loss is 0.9150252
Train epoch time: 180200.654 ms, per step time: 192.317 ms
epoch: 2 step: 937, loss is 0.9933052
Train epoch time: 180199.969 ms, per step time: 192.316 ms
172.16.0.45 slots=1
```

## 4.2.5 Example: Creating a Custom Image for Training (TensorFlow and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is TensorFlow, and the resources used for training are GPUs.

NOTE

This section applies only to training jobs of the new version.

### Scenario

In this example, write a Dockerfile to create a custom image on a Linux x86_64 server running Ubuntu 18.04.

Create a container image with the following configurations and use the image to create a GPU-powered training job on ModelArts:

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

### Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Creating a Dataset and Uploading It to OBS**
4. **Step 3 Preparing the Training Script and Uploading It to OBS**
5. **Step 4 Preparing a Server**
6. **Step 5 Creating a Custom Image**
7. **Step 6 Uploading the Image to SWR**
8. **Step 7 Creating a Training Job on ModelArts**

## Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

## Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-5** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS and ModelArts are in the same region.

**Table 4-5** Required OBS folders

| Folder | Description |
|---|---|
| **obs://test-modelarts/tensorflow/code/** | Stores the training script. |
| **obs://test-modelarts/tensorflow/data/** | Stores dataset files. |
| **obs://test-modelarts/tensorflow/log/** | Store training log files. |

## Step 2 Creating a Dataset and Uploading It to OBS

Download **mnist.npz** from **https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz**, and upload it to **obs://test-modelarts/tensorflow/data/** in the OBS bucket.

## Step 3 Preparing the Training Script and Uploading It to OBS

Obtain the training script **mnist.py** and upload it to **obs://test-modelarts/tensorflow/code/** in the OBS bucket.

**mnist.py** is as follows:

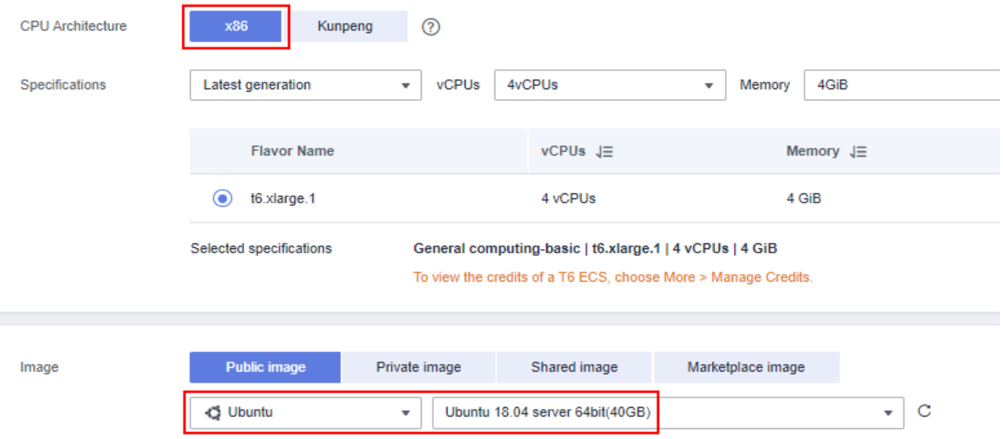
```
import argparse
import tensorflow as tf

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)
```

## Step 4 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

**Figure 4-27** Creating an ECS using a public image (x86)



CPU Architecture: x86

Specifications: Latest generation | vCPUs: 4vCPUs | Memory: 4GiB

| | Flavor Name | vCPUs | Memory |
|---|---|---|---|
| ● | t6.xlarge.1 | 4 vCPUs | 4 GiB |

Selected specifications: General computing-basic | t6.xlarge.1 | 4 vCPUs | 4 GiB

To view the credits of a T6 ECS, choose More > Manage Credits.

Image: Public image — Ubuntu — Ubuntu 18.04 server 64bit(40GB)

## Step 5 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

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

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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

   If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:

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

   The following information is displayed:

   ```
    Engine:
     Version:          18.09.0
   ```

   NOTE

   Use the Docker engine of the preceding version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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

   NOTE

   To obtain **pip.conf**, switch to Huawei Mirrors **https://mirrors.huaweicloud.com/home** and search for **pypi**.

5. Download **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl**.

   Download **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl** from **https://pypi.org/project/tensorflow-gpu/2.10.0/#files**.

6. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

7. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

   ```
   # The server on which the container image is created must access the Internet.

   # Base container image at 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

   # The default user of the base container image is root.
   # USER root

   # Use the PyPI configuration obtained from Huawei Mirrors.
   RUN mkdir -p /root/.pip/
   COPY pip.conf /root/.pip/pip.conf

   # Copy the installation files to the /tmp directory in the base container image.
   COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
   COPY tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl /tmp

   # https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
   # Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
   RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

   # Install the TensorFlow .whl file using default Miniconda3 Python environment /home/ma-user/
   ```

```
miniconda3/bin/pip.
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

# Create the container image.
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

# Install the vim, cURL, net-tools, and MLNX_OFED tools obtained from Huawei Mirrors.
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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the base container image. User ma-user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
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
```

For details about how to write a Dockerfile, see **official Docker documents**.

8. Download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, click **Download**, set **Version** to **5.4-3.5.8.0-LTS**, **OSDistributionVersion** to **Ubuntu 18.04**, and **Architecture** to **x86_64**, and download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Store the Dockerfile and Miniconda3 installation file in the **context** folder, which is as follows:

```
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. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **tensorflow:2.10.0-ofed-cuda11.2**:

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

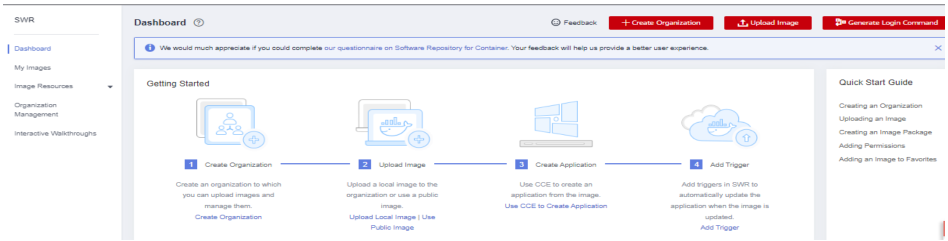
The following log shows that the image has been created.

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

## Step 6 Uploading the Image to SWR

1. Log in to the SWR console and select the target region.

**Figure 4-28** SWR console



We would much appreciate if you could complete our questionnaire on Software Repository for Container. Your feedback will help us provide a better user experience.

Create an organization to which you can upload images and manage them. Create Organization

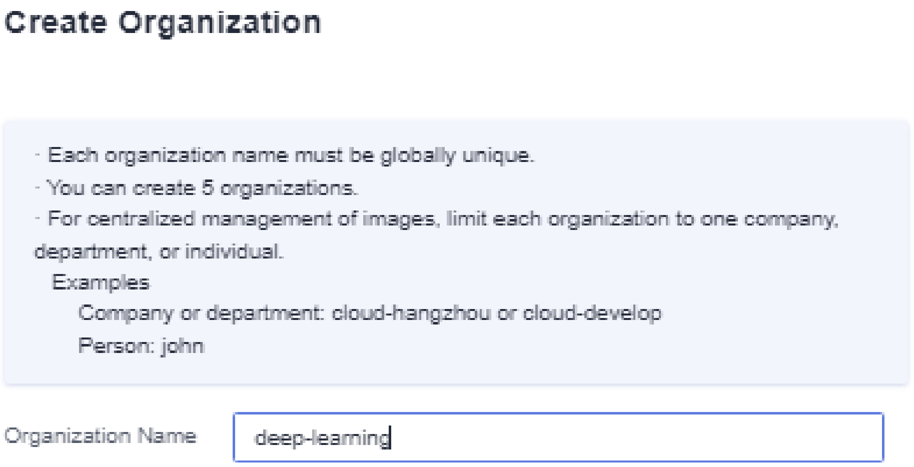
Upload a local image to the organization or use a public image. Upload Local Image | Use Public Image

Use CCE to create an application from the image. Use CCE to Create Application

Add triggers in SWR to automatically update the application when the image is updated. Add Trigger

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

**Figure 4-29** Creating an organization



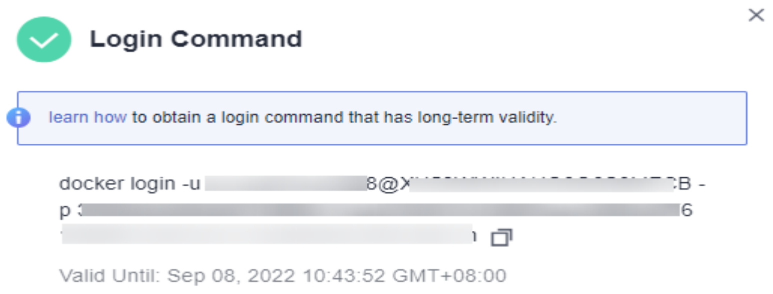
Create Organization

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.
  Examples
  Company or department: cloud-hangzhou or cloud-develop
  Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 4-30** Login Command



Login Command

learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted] -
p [redacted]
[redacted]
```

Valid Until: Sep 08, 2022 10:43:52 GMT+08:00

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
   b. Run the following command to upload the image:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

## Step 7 Creating a Training Job on ModelArts

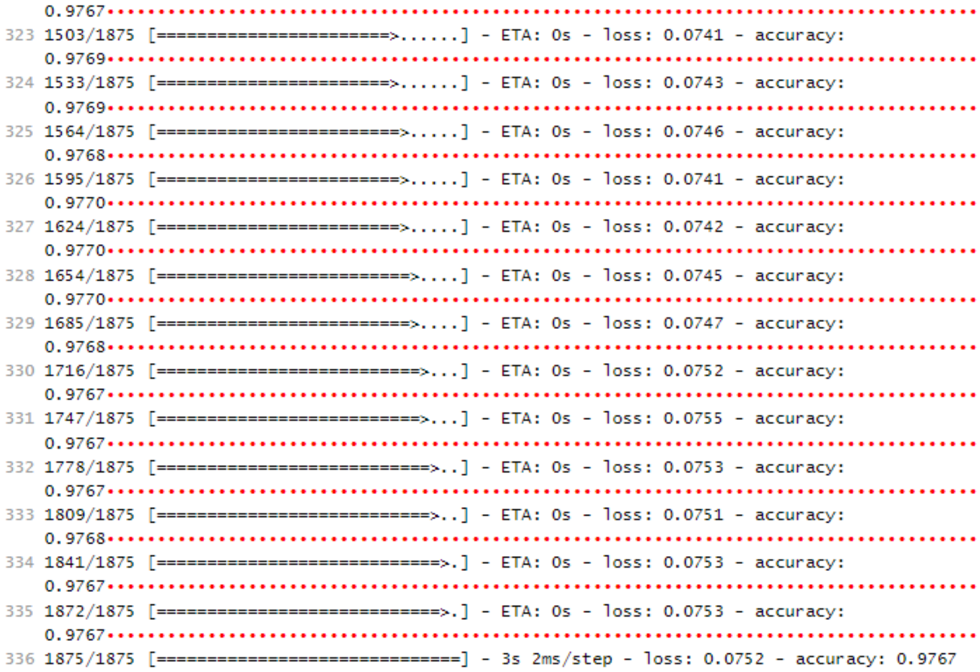
1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Creating a Custom Image**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/tensorflow/code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/code** directory of the training container. **code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **python ${MA_JOB_DIR}/code/mnist.py**. **code** (customizable) is the last-level directory of the OBS path.
   - **Training Input**: Click **Add Training Input**. Enter **data_path** for the name, select the OBS path to **mnist.npz**, for example, **obs://test-modelarts/tensorflow/data/mnist.npz**, and set **Obtained from** to **Hyperparameters**.
   - **Resource Pool**: Select **Public resource pools**.

   - **Resource Type**: Select **GPU**.
   - **Compute Nodes**: Enter **1**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mindspore-gpu/log/**

4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

**Figure 4-31** Run logs of training jobs with GPU specifications



```
     0.9767
1503/1875 [=======================>......] - ETA: 0s - loss: 0.0741 - accuracy:
     0.9769
1533/1875 [=======================>......] - ETA: 0s - loss: 0.0743 - accuracy:
     0.9769
1564/1875 [========================>.....] - ETA: 0s - loss: 0.0746 - accuracy:
     0.9768
1595/1875 [========================>.....] - ETA: 0s - loss: 0.0741 - accuracy:
     0.9770
1624/1875 [========================>.....] - ETA: 0s - loss: 0.0742 - accuracy:
     0.9770
1654/1875 [=========================>....] - ETA: 0s - loss: 0.0745 - accuracy:
     0.9770
1685/1875 [=========================>....] - ETA: 0s - loss: 0.0747 - accuracy:
     0.9768
1716/1875 [==========================>...] - ETA: 0s - loss: 0.0752 - accuracy:
     0.9767
1747/1875 [==========================>...] - ETA: 0s - loss: 0.0755 - accuracy:
     0.9767
1778/1875 [===========================>..] - ETA: 0s - loss: 0.0753 - accuracy:
     0.9767
1809/1875 [===========================>..] - ETA: 0s - loss: 0.0751 - accuracy:
     0.9768
1841/1875 [============================>.] - ETA: 0s - loss: 0.0753 - accuracy:
     0.9767
1872/1875 [============================>.] - ETA: 0s - loss: 0.0753 - accuracy:
     0.9767
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0752 - accuracy: 0.9767
```

# 4.3 Preparing a Training Image

## 4.3.1 Specifications for Custom Images for Training Jobs

When you use a locally developed model and training script to create a custom image, ensure that the custom image complies with the specifications defined by ModelArts.

### Specifications

- Use Ubuntu 18.04 for custom images to in case versions are not compatible.

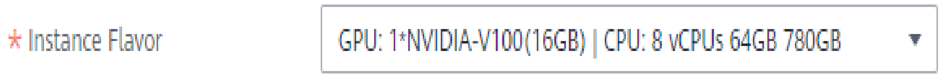
- Do not use a custom image larger than 15 GB. The size should not exceed half of the container engine space of the resource pool. Otherwise, the start time of the training job is affected.

  The container engine space of ModelArts public resource pool is 50 GB. By default, the container engine space of the dedicated resource pool is also 50 GB. You can customize the container engine space when creating a dedicated resource pool.
- The **uid** of the default user of a custom image must be **1000**.
- The GPU or Ascend driver cannot be installed in a custom image. When you select GPU resources to run training jobs, ModelArts automatically places the GPU driver in the **/usr/local/nvidia** directory in the training environment. When you select Ascend resources to run training jobs, ModelArts automatically places the Ascend driver in the **/usr/local/Ascend/driver** directory.
- x86- or Arm-based custom images can run only with specifications corresponding to their architecture.
  - Run the following command to check the CPU architecture of a custom image:

    ```
    docker inspect {Custom image path} | grep Architecture
    ```

    The following is the command output for an Arm-based custom image:

    ```
    "Architecture": "arm64"
    ```
  - If the name of a specification contains **Arm**, this specification is an Arm-based CPU architecture.
  - If the name of a specification does not contain **Arm**, this specification is an x86-based CPU architecture.



* Instance Flavor

GPU: 1*NVIDIA-V100(16GB) | CPU: 8 vCPUs 64GB 780GB

- ModelArts does not support the download of open source installation packages. Install the dependency packages required by the training job in the custom image.

## 4.3.2 Migrating an Image to ModelArts Training

To migrate an image to the training management, perform the following operations:

1. Add the default user group **ma-group** (**gid = 100**) of the training management for the image.

   **NOTE**

   If the user group whose **gid** is **100** already exists, the error message "groupadd: GID '100' already exists" may be displayed. You can use the command **cat /etc/group | grep 100** to check whether the user group whose GID is 100 exists.

   If the user group whose **gid** is **100** already exists, skip this step and delete the command **RUN groupadd ma-group -g 100** from the Dockerfile.
2. Add the default user **ma-user** (**uid = 1000**) of the training management for the image.

NOTE

> If the user whose **uid** is **1000** already exists, the error message "useradd: UID 1000 is not unique" may be displayed. You can use the command **cat /etc/passwd | grep 1000** to check whether the user whose UID is 1000 exists.
>
> If the user whose **uid** is **1000** already exists, skip this step and delete the command **RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user** from the Dockerfile.

3. Modify the permissions on files in the image to allow **ma-user** whose **uid** is **1000** to read and write the files.

You can modify an image by referring to the following Dockerfile so that the image complies with specifications for custom images of the new-version training management.

```
FROM {An existing image}

USER root

# If the user group whose GID is 100 already exists, delete the groupadd command.
RUN groupadd ma-group -g 100
# If the user whose UID is 1000 already exists, delete the useradd command.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Modify the permissions on image files so that user ma-user whose UID is 1000 can read and write the files.
RUN chown -R ma-user:100 {Path to the Python software package}

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PYTHONUNBUFFERED=1

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user
```

After editing the Dockerfile, run the following command to build a new image:

```
docker build -f Dockerfile . -t {New image}
```

Upload the new image to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**

## 4.3.3 Using a Base Image to Create a Training Image

ModelArts provides deep learning-powered base images such as TensorFlow, PyTorch, and MindSpore images. In these images, the software mandatory for running training jobs has been installed. If the software in the base images cannot meet your service requirements, create new images based on the base images and use the new images to create training jobs.

### Procedure

Perform the following operations to create an image using a training base image:

1. Install Docker. If the **docker images** command is executed, Docker has been installed. In this case, skip this step.

   The following uses Linux x86_64 as an example to describe how to obtain the Docker installation package. Run the following command to install Docker:

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

2. Create a folder named **context**.
   ```
   mkdir -p context
   ```
3. Obtain the **pip.conf** file.
   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```
4. Create a new image based on a training base image provided by ModelArts. Save the edited Dockerfile in the **context** folder. For details about how to obtain a training base image, see **Available Training Base Images**.
   ```
   FROM {Path to the training base image provided by ModelArts}

   # Configure pip.
   RUN mkdir -p /home/ma-user/.pip/
   COPY --chown=ma-user:ma-group pip.conf /home/ma-user/.pip/pip.conf

   # Configure the preset environment variables of the container image.
   # Add the Python interpreter path to the PATH environment variable.
   # Set PYTHONUNBUFFERED to 1 to prevent log loss.
   ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
       PYTHONUNBUFFERED=1

   RUN /home/ma-user/anaconda/bin/pip install --no-cache-dir numpy
   ```
5. Run the following command in the directory where the Dockerfile is stored to create a container image, for example, **training:v1**:
   ```
   docker build . -t training:v1
   ```
6. Upload the new image to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**.
7. Use the custom image to create a training job on ModelArts. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

## 4.3.4 Installing MLNX_OFED in a Container Image

### Scenarios

The Mellanox Technologies NIC has been configured on ModelArts GPU servers to support Remote Direct Memory Access (RDMA). As a result, you can install MLNX_OFED in the container image, which will allow the **NCCL** to leverage the NIC and enhance the efficiency of cross-node communication.

After this NIC is enabled for NCCL, NET/IB is used for cross-node communication. If this NIC is not enabled, NET/Socket is used for cross-node communication. NET/IB is better than NET/Socket in terms of latency and bandwidth.

**Table 4-6** Mellanox Technologies NIC and MLNX_OFED installation on ModelArts GPU servers

| GPU Model | Mellanox Technologies NIC | Installed MLNX_OFED Version | Recommended MLNX_OFED Version for Container Image |
|---|---|---|---|
| V100 | 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 |

## Installing MLNX_OFED

Take the Ubuntu18.04 container image as an example. The Dockerfile for installing MLNX_OFED 4.9-6.0.6.0-LTS is as follows:

**NOTE**

The host that is used to download files and create container images using a Dockerfile must be able to connect to the public network.

```
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
```

Create a container image by referring to this command example:

```
docker build -f Dockerfile . -t nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-ubuntu18.04
```

After the container image has been created, run this command to obtain the MLNX_OFED version in the container image:

```
docker run -ti --rm nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-ubuntu18.04 ofed_info | head -n 1
```

The command output is as follows:

```
MLNX_OFED_LINUX-4.9-6.0.6.0 (OFED-4.9-6.0.6):
```

# 4.4 Creating an Algorithm Using a Custom Image

Your locally developed algorithms or algorithms developed using other tools can be uploaded to ModelArts for unified management.

## Entries for Creating an Algorithm

You can create an algorithm using a custom image on ModelArts in either of the following ways:

- Entry 1: On the ModelArts console, choose **Algorithm Management** > **My algorithms**. Then, create an algorithm and use it in training jobs or publish it to AI Gallery.
- Entry 2: On the ModelArts console, choose **Training Management** > **Training Jobs**, and click **Create Training Job** to create a custom algorithm and submit

a training job. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

## Parameters for creating an algorithm

**Table 4-7** Parameters for creating an algorithm

| Parameter | Description |
|---|---|
| Boot Mode | Select **Custom images**. This parameter is mandatory. |
| Image | URL of an SWR image. This parameter is mandatory.<br>● Private images or shared images: Click **Select** on the right to select an SWR image. Ensure that the image has been uploaded to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**.<br>● Public images: You can also manually enter the image path in the format of "<Organization to which your image belongs>/<Image name>" on SWR. Do not contain the domain name (swr.<region>.xxx.com) in the path because the system will automatically add the domain name to the path. For example: `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` |
| Code Directory | OBS path for storing the training code. This parameter is optional.<br>Take OBS path **obs://obs-bucket/training-test/demo-code** as an example. The content in the OBS path will be automatically downloaded to **${MA_JOB_DIR}/demo-code** in the training container, and **demo-code** (customizable) is the last-level directory of the OBS path. |
| Boot Command | Command for booting an image. This parameter is mandatory. The boot command will be automatically executed after the code directory is downloaded.<br>● If the training boot script is a .py file, **train.py** for example, the boot command can be **python ${MA_JOB_DIR}/demo-code/train.py**.<br>● If the training boot script is an .sh file, **main.sh** for example, the boot command can be **bash ${MA_JOB_DIR}/demo-code/main.sh**.<br>Semicolons (;) and ampersands (&&) can be used to combine multiple boot commands, but line breaks are not supported. **demo-code** (customizable) in the boot command is the last-level directory of the OBS path. |

## Configuring Pipelines

A preset image-based algorithm obtains data from an OBS bucket or dataset for model training. The training output is stored in an OBS bucket. The input and output parameters in your algorithm code must be parsed to enable data exchange between ModelArts and OBS. For details about how to develop code for training on ModelArts, see **Developing a Custom Script**.

When you use a preset image to create an algorithm, configure the input and output pipelines.

- Input configurations

**Table 4-8** Input configurations

| Parameter | Description |
|---|---|
| Parameter Name | Set the name based on the data input parameter in your algorithm code. The code path parameter must be the same as the training input parameter parsed in your algorithm code. Otherwise, the algorithm code cannot obtain the input data.<br>For example, If you use **argparse** in the algorithm code to parse **data_url** into the data input, set the data input parameter to **data_url** when creating the algorithm. |
| Description | Customizable description of the input parameter, |
| Obtained from | Source of the input parameter. You can select **Hyperparameters** (default) or **Environment variables**. |
| Constraints | Whether data is obtained from a storage path or ModelArts dataset.<br>If you select the ModelArts dataset as the data source, the following constraints are added:<br>● **Labeling Type**: For details, see **Creating a Labeling Job**.<br>● **Data Format**, which can be **Default**, **CarbonData**, or both. **Default** indicates the manifest format.<br>● **Data Segmentation**: available only for image classification, object detection, text classification, and sound classification datasets. Possible values are **Segmented dataset**, **Dataset not segmented**, and **Unlimited**. For details, see **Publishing a Data Version**. |
| Add | Multiple data input sources are allowed. |

- Output configurations

**Table 4-9** Output configurations

| Parameter | Description |
|---|---|
| Parameter Name | Set the name based on the data output parameter in your algorithm code. The code path parameter must be the same as the training output parameter parsed in your algorithm code. Otherwise, the algorithm code cannot obtain the output path.<br>For example, If you use **argparse** in the algorithm code to parse **train_url** into the data output, set the data output parameter to **train_url** when creating the algorithm. |
| Description | Customizable description of the output parameter, |
| Obtained from | Source of the output parameter. You can select **Hyperparameters** (default) or **Environment variables**. |
| Add | Multiple data output paths are allowed. |

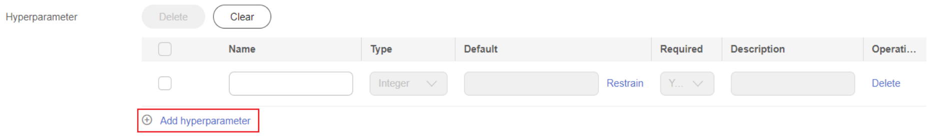
## Defining Hyperparameters

When you use a preset image to create an algorithm, ModelArts allows you to customize hyperparameters so you can view or modify them anytime. After the hyperparameters are defined, they are displayed in the startup command and transferred to your boot file as CLI parameters.

1. Import hyperparameters.

   You can click **Add hyperparameter** to manually add hyperparameters.

   **Figure 4-32** Adding hyperparameters

   Hyperparameter Delete Clear

   | □ | Name | Type | Default | | Required | Description | Operati... |
   |---|---|---|---|---|---|---|---|
   | □ | | Integer | | Restrain | Y... | | Delete |

   ⊕ Add hyperparameter

2. Edit hyperparameters.

   For details, see **Table 4-10**.

**Table 4-10** Hyperparameters

| Parameter | Description |
|---|---|
| Name | Hyperparameter name<br>Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed. |
| Type | Type of the hyperparameter, which can be **String**, **Integer**, **Float**, or **Boolean** |

| Parameter | Description |
|---|---|
| Default | Default value of the hyperparameter, which is used for training jobs by default |
| Constraints | Click **Restrain**. Then, set the range of the default value or enumerated value in the dialog box displayed. |
| Required | Select **Yes** or **No**.<br>● If you select **No**, you can delete the hyperparameter on the training job creation page when using this algorithm to create a training job.<br>● If you select **Yes**, you cannot delete the hyperparameter on the training job creation page when using this algorithm to create a training job. |
| Description | Description of the hyperparameter<br>Only letters, digits, spaces, hyphens (-), underscores (_), commas (,), and periods (.) are allowed. |

## Adding Training Constraints

You can add training constraints of the algorithm based on your needs.

- **Resource Type**: Select the required resource types.
- **Multicard Training**: Choose whether to support multi-card training.
- **Distributed Training**: Choose whether to support distributed training.

## Runtime Environment Preview

When creating an algorithm, click the arrow on in the lower right corner of the page to know the path of the code directory, boot file, and input and output data in the training container.

## Follow-Up Procedure

After an algorithm is created, use it to create a training job. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

# 4.5 Using a Custom Image to Create a CPU- or GPU-based Training Job

Model training is an iterative optimization process. Through unified training management, you can flexibly select algorithms, data, and hyperparameters to obtain the optimal input configuration and model. After comparing metrics between job versions, you can determine the most satisfactory training job.

## Prerequisites

- The data to be trained has been uploaded to an OBS directory.
- At least one empty folder for storing the training output has been created in OBS.
- A custom image has been created based on ModelArts specifications. For details about the custom image specifications, see **Specifications for Custom Images for Training Jobs**.
- The custom image has been uploaded to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**.

## Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management** > **Training Jobs**.
2. Click **Create Training Job** and set parameters.

**Table 4-11** Creating a training job using a custom image

| Parameter | Description |
|---|---|
| Algorithm Type | Select **Custom algorithm**. This parameter is mandatory. |
| Boot Mode | Select **Custom image**. This parameter is mandatory. |
| Image | Container image path. This parameter is mandatory.<br>You can set the container image path in either of the following ways:<br>● To select your image or an image shared by others, click **Select** on the right and select a container image for training. The required image must be uploaded to SWR beforehand.<br>● To select a public image, enter the address of the public image in SWR. Enter the image path in the format of "Organization name/Image name:Version name". Do not contain the domain name (swr.<region>.xxx.com) in the path because the system will automatically add the domain name to the path. For example, if the SWR address of a public image is **swr.<region>.xxx.com/test-image/tensorflow2_1_1:1.1.1**, enter **test-images/tensorflow2_1_1:1.1.1**. |

| Parameter | Description |
|---|---|
| Code Directory | Select the OBS directory where the training code file is stored. If the custom image does not contain training code, you need to set this parameter. If the custom image contains training code, you do not need to set this parameter.<br>● Upload code to the OBS bucket beforehand. The total size of files in the directory cannot exceed 5 GB, the number of files cannot exceed 1000, and the folder depth cannot exceed 32.<br>● The training code file is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container when the training job is started. **demo-code** is the last-level OBS directory for storing the code. For example, if **Code Directory** is set to **/test/code**, the training code file is downloaded to the **${MA_JOB_DIR}/code** directory of the training container. |
| User ID | User ID for running the container. The default value 1000 is recommended.<br>If the UID needs to be specified, its value must be within the specified range. The UID ranges of different resource pools are as follows:<br>● Public resource pool: 1000 to 65535<br>● Dedicated resource pool: 0 to 65535 |
| Boot Command | Command for booting an image. This parameter is mandatory.<br>When a training job is running, the boot command is automatically executed after the code directory is downloaded.<br>● If the training boot script is a .py file, **train.py** for example, the boot command is as follows.<br>`python ${MA_JOB_DIR}/demo-code/train.py`<br>● If the training boot script is a .sh file, **main.sh** for example, the boot command is as follows.<br>`bash ${MA_JOB_DIR}/demo-code/main.sh`<br>You can use semicolons (;) and ampersands (&&) to combine multiple commands. **demo-code** in the command is the last-level OBS directory where the code is stored. Replace it with the actual one. |
| Local Code Directory | Specify the local directory of a training container. When a training starts, the system automatically downloads the code directory to this directory.<br>The default local code directory is **/home/ma-user/modelarts/user-job-dir**. This parameter is optional. |

| Parameter | Description |
|---|---|
| Work Directory | During training, the system automatically runs the **cd** command to execute the boot file in this directory. |

**Table 4-12** Parameters for creating a training job

| Parameter | Sub-Parameter | Description |
|---|---|---|
| Input | Parameter | The algorithm code reads the training input data based on the input parameter name.<br>Set this parameter to **data_url**, which is the same as the parameter for parsing the input data in the training code. You can set multiple training input parameters. The name of each training input parameter must be unique.<br>For example, if you use **argparse** in the training code to parse **data_url** into the data input, set the parameter name of the training input to **data_url**.<br>`import argparse`<br>`# Create a parsing task.`<br>`parser = argparse.ArgumentParser(description="train mnist",`<br>`formatter_class=argparse.ArgumentDefaultsHelpFormatter)`<br>`# Add parameters.`<br>`parser.add_argument('--train_url', type=str, help='the path model`<br>`saved')`<br>`parser.add_argument('--data_url', type=str, help='the training data')`<br>`# Parse the parameters.`<br>`args, unknown = parser.parse_known_args()` |
| | Dataset | Click **Dataset** and select the target dataset and its version in the ModelArts dataset list.<br>When the training job is started, ModelArts automatically downloads the data in the input path to the training container.<br>**NOTE**<br>ModelArts data management is being upgraded and is invisible to users who have not used data management. It is recommended that new users store their training data in OBS buckets. |
| | Data path | Click **Data path** and select the storage path to the training input data from an OBS bucket.<br>When the training job is started, ModelArts automatically downloads the data in the input path to the training container. |

| Parameter | Sub-Parameter | Description |
|---|---|---|
| | How to Obtain | The following uses training input **data_path** as an example.<br>● If you select **Hyperparameters**, use this code to obtain the data:<br>`import argparse`<br>`parser = argparse.ArgumentParser()`<br>`parser.add_argument('--data_path')`<br>`args, unknown = parser.parse_known_args()`<br>`data_path = args.data_path`<br>● If you select **Environment variables**, use this code to obtain the data:<br>`import os`<br>`data_path = os.getenv("data_path", "")` |
| Output | Parameter | The algorithm code reads the training output data based on the output parameter name.<br>Set this parameter to **train_url**, which is the same as the parameter for parsing the output data in the training code. You can set multiple training output parameters. The name of each training output parameter must be unique. |
| | Data path | Click **Data path** and select the storage path to the training output data from an OBS bucket. During training, the system automatically synchronizes files from the local code directory of the training container to the data path.<br>**NOTE**<br>The data path can only be an OBS path. To prevent any issues with data storage, choose an empty directory as the data path. |
| | How to Obtain | The following uses the training output **train_url** as an example.<br>● If you select **Hyperparameters**, use this code to obtain the data:<br>`import argparse`<br>`parser = argparse.ArgumentParser()`<br>`parser.add_argument('--train_url')`<br>`args, unknown = parser.parse_known_args()`<br>`train_url = args.train_url`<br>● If you select **Environment variables**, use this code to obtain the data:<br>`import os`<br>`train_url = os.getenv("train_url", "")` |

| Parameter | Sub-Parameter | Description |
|---|---|---|
| | Predownload | Indicates whether to pre-download the files in the output directory to a local directory.<br>● If you set **Predownload** to **No**, the system does not download the files in the training output data path to a local directory of the training container when the training job is started.<br>● If you set **Predownload** to **Yes**, the system automatically downloads the files in the training output data path to a local directory of the training container when the training job is started. The larger the file size, the longer the download time. To avoid excessive training time, remove any unneeded files from the local code directory of the training container as soon as possible. To use **resumable training and incremental training**, **Download** must be selected. |
| Hyperparameters | - | Used for training tuning. This parameter is optional. |
| Environment Variable | - | Add environment variables based on service requirements. For details about preset environment variables in the training container, see **Viewing Environment Variables of a Training Container**. |
| Auto Restart | - | Number of retries for a failed training job. If this parameter is enabled, a failed training job will be automatically re-delivered and run. On the training job details page, you can view the number of retries for a failed training job.<br>● This function is disabled by default.<br>● If you enable this function, set the number of retries. The value ranges from **1** to **3** and cannot be changed. |

3. Select an instance flavor. The value range of the training parameters is consistent with the constraints of existing custom images. Select a public resource pool or dedicated resource pool as required. For details about the parameters, see **Creating a Training Job**.

4. Click **Submit** to create the training job.

   It takes a period of time to create a training job.

   To view the real-time status of a training job, go to the training job list and click the name of the training job. On the training job details page that is displayed, view the basic information of the training job. For details, see **Viewing Training Job Details**.

# 4.6 Troubleshooting Process

## Symptom

A training job using a custom image failed.

## Locating Method

1. Determine the image source.
   - Check whether the base image of the custom image is from ModelArts. Use a base image provided by ModelArts to create a custom image. For details, see **Using a Base Image to Create a Training Image**.
   - If the image is from a third party, check with the creator of the custom image for how to use this image.
2. Determine the size of the custom image.

   Do not use a custom image larger than 15 GB. The size should not exceed half of the container engine space of the resource pool. Otherwise, the start time of the training job is affected.

   The container engine space of ModelArts public resource pool is 50 GB. By default, the container engine space of the dedicated resource pool is also 50 GB. You can customize the container engine space when creating a dedicated resource pool.
3. Determine the error type.
   - If an error message is displayed indicating that a file could not be found, see **Error Message "No such file or directory" Displayed in Training Job Logs**.
   - If an error message is displayed indicating that a package could not be found, see **Error Message "No module named .*" Displayed in Training Job Logs**.
   - An error occurred in the Ascend startup script or initialization script.

     Check whether the script is obtained from the official website and whether the script is used strictly following the instructions provided in official documents. For example, check whether the script name and path are correct.
   - The driver version is incompatible with the underlying driver.

     Before upgrading the driver of a custom image, check whether the upgraded version is supported by the underlying driver. **Obtain the supported driver versions**.
   - You are not allowed to access a file.

     The possible cause is that the user of the custom image is different from that of the job container. In this case, modify the Dockerfile.

     ```
     RUN if id -u ma-user > /dev/null 2>&1 ; \
     then echo 'The ModelArts user already exists.' ; \
     else echo 'The ModelArts user does not exist.' && \
     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
```

– For other issues, search for solutions in **training failure cases**.

## Summary and Suggestions

Before using a custom image for training jobs, create the image by following the **custom image specifications**. which also provides end-to-end examples for your reference.

# 5 Using a Custom Image to Create AI applications for Inference Deployment

## 5.1 Custom Image Specifications for Creating AI Applications

When building a custom image using a locally developed model, ensure that the image complies with ModelArts specifications.

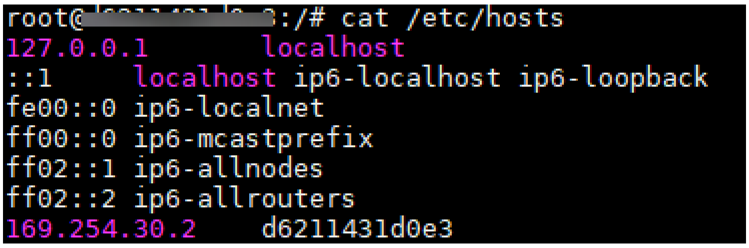
- No malicious code is allowed.
- A custom image cannot be larger than 50 GB.
- **External APIs**

  Set the external service API for a custom image. The inference API must be the same as the URL defined by **apis** in **config.json**. Then, the external service API can be directly accessed when the image is started. The following is an example of accessing an MNIST image. The image contains a model trained using an MNIST dataset and can identify handwritten digits. **listen_ip** indicates the container IP address. You can start a custom image to obtain the container IP address from the container.

  – Sample request

```
curl -X POST \ http://{Listening IP address}:8080/ \ -F images=@seven.jpg
```

**Figure 5-1** Example of obtaining **listen_ip**



```
root@[redacted]:/# 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
```

  – Sample response

```
{"mnist_result": 7}
```

- **(Optional) Health check API**

If services must not be interrupted during a rolling upgrade, the health check API must be configured in **config.json** for ModelArts. The health check API returns the healthy state for a service when the service is running properly or an error when the service becomes faulty.

NOTICE

The health check API must be configured for a hitless rolling upgrade.

The following shows a sample health check API:

- URI
```
GET /health
```
- Sample request: curl -X GET \ http://*{Listening IP address}*:8080/health
- Sample response
```
{"health": "true"}
```
- Status code

**Table 5-1** Status code

| Status Code | Message | Description |
|---|---|---|
| 200 | OK | Request sent |

- **Log file output**

  Configure standard output so that logs can be properly displayed.
- **Image boot file**

  To deploy a batch service, set the boot file of an image to **/home/run.sh** and use CMD to set the default boot path. The following is a sample Dockerfile:

  **CMD ["sh", "/home/run.sh"]**
- **Image dependencies**

  To deploy a batch service, install dependency packages such as Python, JRE/JDK, and ZIP in the image.
- **(Optional) Hitless rolling upgrade**

  To ensure that services are not interrupted during a rolling upgrade, set HTTP **keep-alive** to **200**. For example, Gunicorn does not support keep-alive by default. To ensure a hitless rolling upgrade, install Gevent and configure **--keep-alive 200 -k gevent** in the image. The parameter settings vary depending on the service framework. Set the parameters as required.
- **(Optional) Gracefully exiting a container**

  To ensure that services are not interrupted during a rolling upgrade, the system must capture SIGTERM signals in the container and wait for 60s before gracefully exiting the container. If the duration is less than 60s before the graceful exiting, services may be interrupted during the rolling upgrade. To ensure uninterrupted service running, the system exits the container after the system receives SIGTERM signals and processes all received requests. The whole duration is not longer than 90s. The following shows example **run.sh**:

```
#!/bin/bash
gunicorn_pid=""
```

```
handle_sigterm() {
  echo "Received SIGTERM, send SIGTERM to $gunicorn_pid"
  if [ $gunicorn_pid != "" ]; then
      sleep 60
      kill -15 $gunicorn_pid  # Transfer SIGTERM signals to the Gunicorn process.
      wait $gunicorn_pid           # Wait until the Gunicorn process stops.
  fi
}

trap handle_sigterm TERM
```

# 5.2 Creating a Custom Image and Using It to Create an AI Application

If you want to use an AI engine that is not supported by ModelArts, create a custom image for the engine, import the image to ModelArts, and use the image to create AI applications. This section describes how to use a custom image to create an AI application and deploy the application as a real-time service.

The process is as follows:

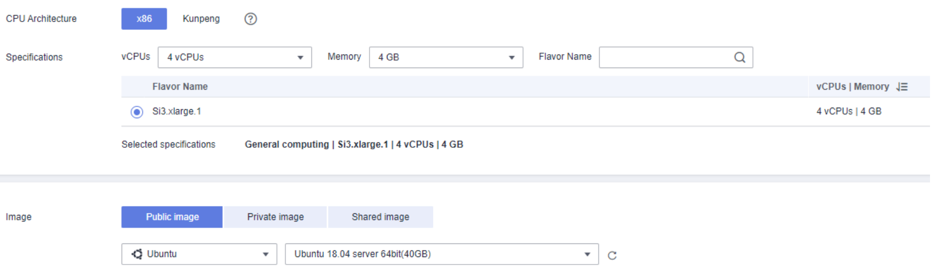
1. **Building an Image Locally**: Create a custom image package locally. For details, see **Custom Image Specifications for Creating AI Applications**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Using the Custom Image to Create an AI Application**: Import the image to ModelArts AI application management.
4. **Deploying the AI Application as a Real-Time Service**: Deploy the model as a real-time service.

## Building an Image Locally

This section uses a Linux x86_x64 host as an example. You can purchase an ECS of the same specifications or use an existing local host to create a custom image.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. When creating the ECS, select an Ubuntu 18.04 public image.

**Figure 5-2** Creating an ECS using an x86 public image



CPU Architecture: x86 | Kunpeng

Specifications: vCPUs 4 vCPUs | Memory 4 GB | Flavor Name

| | Flavor Name | vCPUs \| Memory |
|---|---|---|
| ● | Si3.xlarge.1 | 4 vCPUs \| 4 GB |

Selected specifications: General computing | Si3.xlarge.1 | 4 vCPUs | 4 GB

Image: Public image | Private image | Shared image

Ubuntu | Ubuntu 18.04 server 64bit(40GB)

1. After logging in to the host, install Docker. For details, see **Docker official documents**. Alternatively, run the following commands to install Docker:

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

2. Obtain the base image. Ubuntu 18.04 is used in this example.

   ```
   docker pull ubuntu:18.04
   ```

3. Create the **self-define-images** folder, and edit **Dockerfile** and **test_app.py** in the folder for the custom image. In the sample code, the application code runs on the Flask framework.

   The file structure is as follows:

   ```
   self-define-images/
       --Dockerfile
       --test_app.py
   ```

   – **Dockerfile**

     ```
     From ubuntu:18.04
     # Configure the HUAWEI CLOUD source and install Python, Python3-PIP, and 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

     # Copy the application code to the image.
     COPY test_app.py /opt/test_app.py

     # Specify the boot command of the image.
     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. Switch to the **self-define-images** folder and run the following command to create custom image **test:v1**:

   ```
   docker build -t test:v1 .
   ```

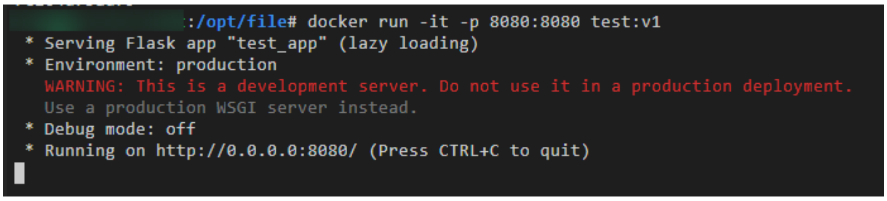
5. Run **docker images** to view the custom image you have created.

## Verifying the Image Locally and Uploading It to SWR

1. Run the following command in the local environment to start the custom image:

```
docker run -it -p 8080:8080 test:v1
```

**Figure 5-3** Starting a custom image



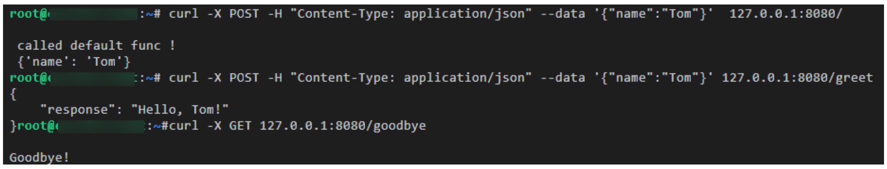
```
[redacted]:/opt/file# docker run -it -p 8080:8080 test:v1
 * Serving Flask app "test_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

2. Open another terminal and run the following commands to test the functions of the three APIs of the custom image:

```
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
```

If information similar to the following is displayed, the function verification is successful.

**Figure 5-4** Testing API functions



```
root@[redacted]:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/

 called default func !
 {'name': 'Tom'}
root@[redacted]:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
{
    "response": "Hello, Tom!"
}root@[redacted]:~#curl -X GET 127.0.0.1:8080/goodbye

Goodbye!
```

3. Upload the custom image to SWR. For details, see **How Can I Upload Images to SWR?**

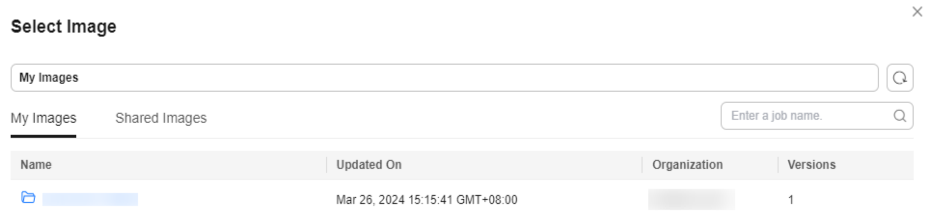
4. View the uploaded image on the **My Images** > **Private Images** page of the SWR console.

## Using the Custom Image to Create an AI Application

Import a meta model. For details, see **Creating and Importing a Model Image**. Key parameters are as follows:

- **Meta Model Source**: Select **Container image**.
  - **Container Image Path**: Select the created private image.

    **Figure 5-5** Created private image

    Select Image

    My Images

    | Name | Updated On | Organization | Versions |
    |---|---|---|---|
    | [redacted] | Mar 26, 2024 15:15:41 GMT+08:00 | [redacted] | 1 |

  - **Container API**: Protocol and port number for starting a model. Ensure that the protocol and port number are the same as those provided in the custom image.

  - **Image Replication**: indicates whether to copy the model image in the container image to ModelArts. This parameter is optional.
  - **Health Check**: checks health status of a model. This parameter is optional. This parameter is configurable only when the health check API is configured in the custom image. Otherwise, creating the AI application will fail.

- **APIs**: APIs of a custom image. This parameter is optional. The model APIs must comply with ModelArts specifications. For details, see **Specifications for Editing a Model Configuration File**.

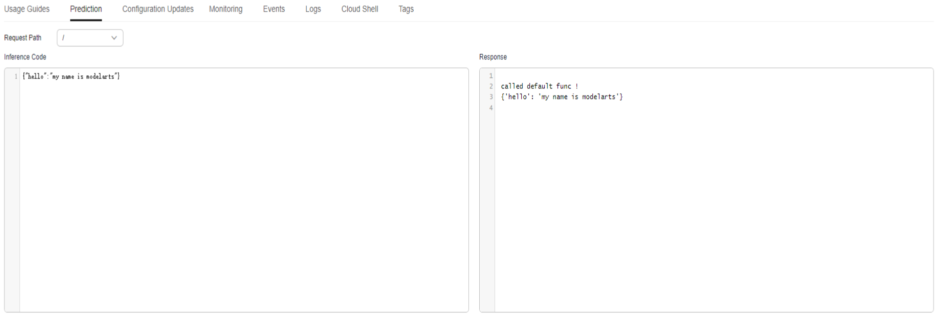
  The configuration file is as follows:

```
[{
      "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"
      }
   }
]
```

## Deploying the AI Application as a Real-Time Service

1. Deploy the AI application as a real-time service. For details, see **Deploying as a Real-Time Service**.
2. View the details about the real-time service.
3. Access the real-time service on the **Prediction** tab page.

**Figure 5-6** Accessing a real-time service



Request Path: /

Inference Code

```
{"hello":"my name is modelarts"}
```

Response

```

called default func !
{'hello': 'my name is modelarts'}

```

# 6 FAQs

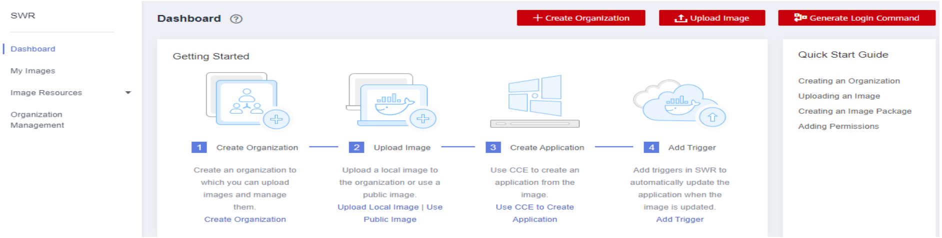
## 6.1 How Can I Log In to SWR and Upload Images to It?

This section describes how to log in to SWR and upload images to it.

### Step 1 Log In to SWR

1. Log in to the SWR console and select the target region.

**Figure 6-1** SWR console



Create an organization to which you can upload images and manage them. Create Organization

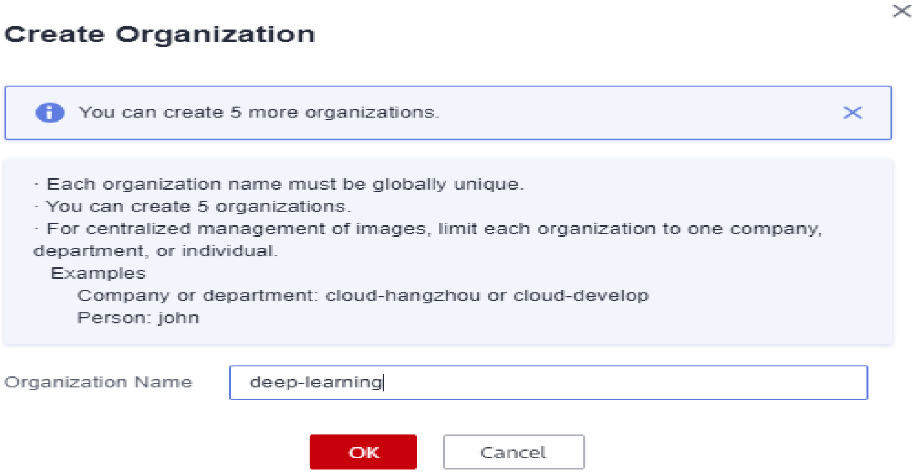
Upload a local image to the organization or use a public image. Upload Local Image | Use Public Image

Use CCE to create an application from the image. Use CCE to Create Application

Add triggers in SWR to automatically update the application when the image is updated. Add Trigger

2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. **deep-learning** is used as an example. Replace it in subsequent commands with the actual organization name.

**Figure 6-2** Creating an organization



Create Organization

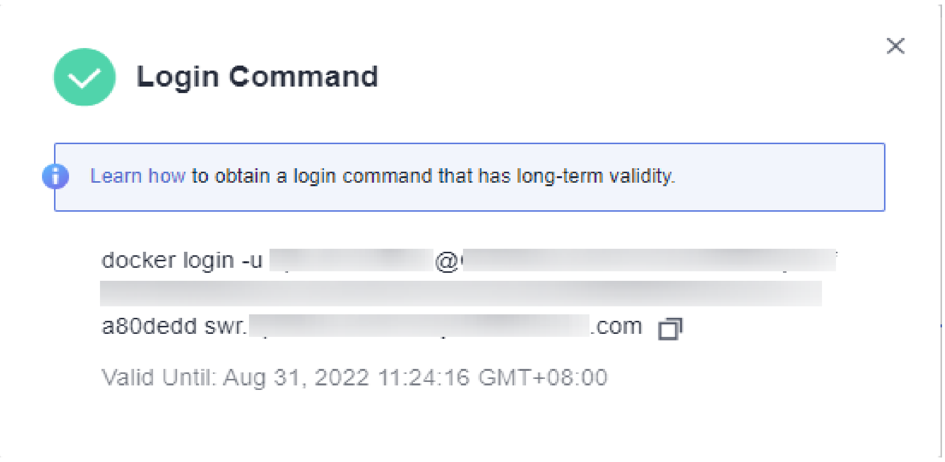
You can create 5 more organizations.

- Each organization name must be globally unique.
- You can create 5 organizations.
- For centralized management of images, limit each organization to one company, department, or individual.
  Examples
  Company or department: cloud-hangzhou or cloud-develop
  Person: john

Organization Name: deep-learning

3. Click **Generate Login Command** in the upper right corner to obtain a login command.

**Figure 6-3** Login Command



Login Command

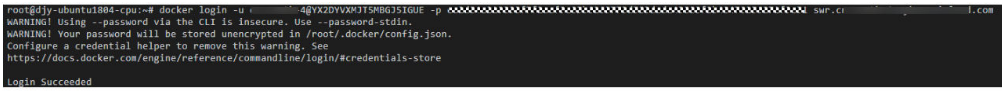
Learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted]@[redacted]
[redacted]
a80dedd swr.[redacted].com
```

Valid Until: Aug 31, 2022 11:24:16 GMT+08:00

4. Log in to the ECS as user **root** and enter the login command.

**Figure 6-4** Login command executed on the ECS



```
root@djy-ubuntu1804-cpu:~# docker login -u [redacted]4@YX2DYVXMJT5MBGJ5IGUE -p [redacted] swr.c[redacted].com
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
```

## Step 2 Upload Images to SWR

This section describes how to upload an image to SWR.

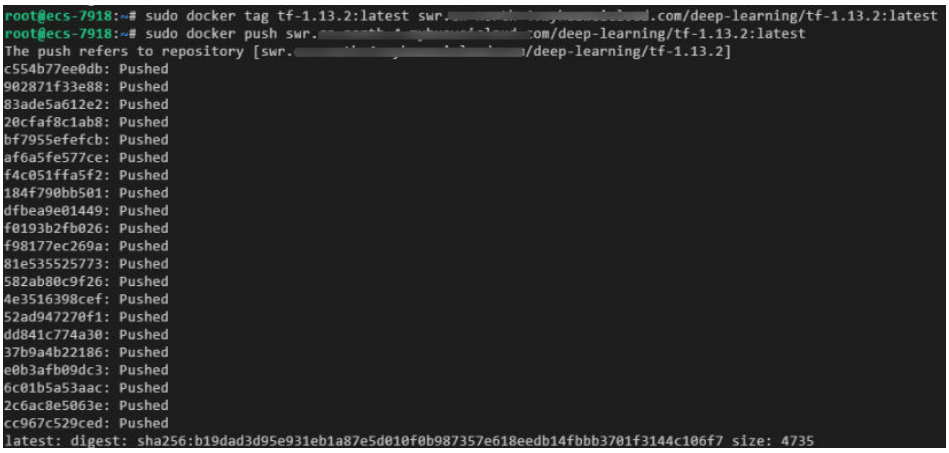
1. Log in to SWR and tag the image to be uploaded. Replace the organization name **deep-learning** in the following command with the actual organization name obtained in step 1.

```
sudo docker tag tf-1.13.2:latest swr.xxx.com/deep-learning/tf-1.13.2:latest
```

2. Run the following command to upload the image:

```
sudo docker push swr.xxx.com/deep-learning/tf-1.13.2:latest
```

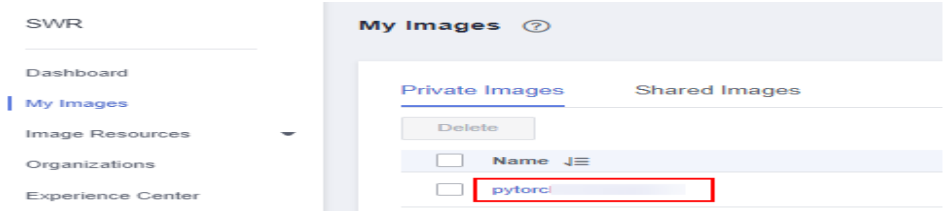
**Figure 6-5** Uploading an image



```
root@ecs-7918:~# sudo docker tag tf-1.13.2:latest swr.[redacted].com/deep-learning/tf-1.13.2:latest
root@ecs-7918:~# sudo docker push swr.[redacted]om/deep-learning/tf-1.13.2:latest
The push refers to repository [swr.[redacted]/deep-learning/tf-1.13.2]
c554b77ee0db: 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
```

3. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

**Figure 6-6** Uploaded custom image



**swr.xxx.com/deep-learning/tf-1.13.2:latest** is the SWR URL of the custom image.

## 6.2 How Do I Configure Environment Variables for an Image?

In a Dockerfile, use the ENV instruction to configure environment variables. For details, see **Dockerfile reference**.

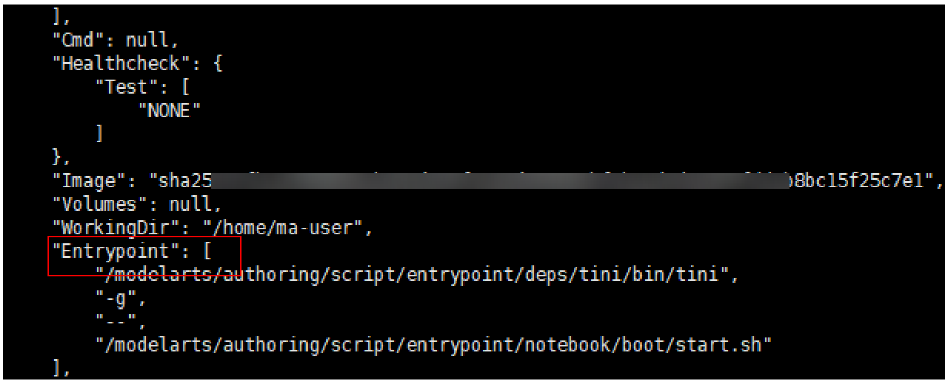
## 6.3 How Do I Use Docker to Start an Image Saved Using a Notebook Instance?

An image saved using a notebook instance contains the **Entrypoint** parameter, as shown in **Entrypoint**. The executable file or command specified in the **Entrypoint** parameter overwrites the default boot command of the image. The command input in the **Entrypoint** parameter is not preset in the image. When you run **docker run** in the local environment to start the image, an error message is displayed, indicating that the container creation task fails because the boot file or directory is not found, as shown in **Figure 6-8**.

To avoid this error, configure the **--entrypoint** parameter to overwrite the program specified in **Entrypoint**. Use the boot file or command specified by the **--entrypoint** parameter to start the image. Example:

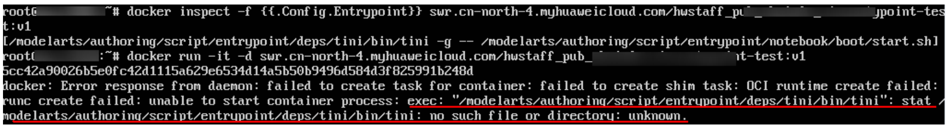
```
docker run -it -d --entrypoint /bin/bash image:tag
```

**Figure 6-7** Entrypoint



```
    ],
    "Cmd": null,
    "Healthcheck": {
        "Test": [
            "NONE"
        ]
    },
    "Image": "sha25[redacted]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"
    ],
```

**Figure 6-8** Error reported when an image is being started



```
root@[redacted]~# docker inspect -f {{.Config.Entrypoint}} swr.cn-north-4.myhuaweicloud.com/hwstaff_pu[redacted]ypoint-tes
t:v1
[/modelarts/authoring/script/entrypoint/deps/tini/bin/tini -g -- /modelarts/authoring/script/entrypoint/notebook/boot/start.sh]
root([redacted]:~# docker run -it -d swr.cn-north-4.myhuaweicloud.com/hwstaff_pub_[redacted]nt-test:v1
5cc42a90026b5e0fc42d1115a629e6534d14a5b50b9496d584d3f825991b248d
docker: Error response from daemon: failed to create task for container: 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 How Do I Configure a Conda Source in a Notebook Development Environment?

You can install the development dependencies in Notebook as you need. Package management tools pip and Conda can be used to install regular dependencies. The pip source has been configured and can be used for installation, while the Conda source requires further configuration.

This section describes how to configure the Conda source on a notebook instance.

## Configuring the Conda Source

The Conda software has been preset in images.

## Common Conda Commands

For details about all Conda commands, see **Conda official documents**. The following table lists only common commands.

**Table 6-1** Common Conda commands

| Description | Command |
|---|---|
| Obtain online help. | conda --help<br>conda update --help # Obtain help for a command, for example, **update**. |

| Description | Command |
|---|---|
| View the Conda version. | conda -V |
| Update Conda. | conda update conda  # Update Conda.<br>conda update anaconda # Update Anaconda. |
| Manage environments. | conda env list  # Show all virtual environments.<br>conda info -e # Show all virtual environments.<br>conda create -n myenv python=3.7 # Create an environment named **myenv** with Python version **3.7**.<br>conda activate myenv  # Activate the **myenv** environment.<br>conda deactivate  # Disable the current environment.<br>conda remove -n myenv --all # Delete the **myenv** environment.<br>conda create -n newname --clone oldname # Clone the old environment to the new environment. |
| Manage packages. | conda list  # Check the packages that have been installed in the current environment.<br>conda list  -n myenv  # Specify the packages installed in the **myenv** environment.<br>conda search numpy # Obtain all information of the **numpy** package.<br>conda search numpy=1.12.0 --info  # View the information of NumPy 1.12.0.<br>conda install numpy pandas  # Concurrently install the NumPy and Pandas packages.<br>conda install numpy=1.12.0  # Install NumPy of a specified version.<br># The **install**, **update**, and **remove** commands use **-n** to specify an environment, and the **install** and **update** commands use **-c** to specify a source address.<br>conda install -n myenv numpy  # Install the **numpy** package in the **myenv** environment.<br>conda install -c https://conda.anaconda.org/anaconda numpy  # Install NumPy using https://conda.anaconda.org/anaconda.<br>conda update numpy pandas   # Concurrently update the NumPy and Pandas packages.<br>conda remove numpy pandas   # Concurrently uninstall the NumPy and Pandas packages.<br>conda update –-all # Update all packages in the current environment. |
| Clear Conda. | conda clean -p      # Delete useless packages.<br>conda clean -t      # Delete compressed packages.<br>conda clean -y --all # Delete all installation packages and clear caches. |

### Saving as an Image

After installing the external libraries, save the environment using the image saving function provided by ModelArts notebook of the new version. You can save a running notebook instance as a custom image with one click for future use. After the dependency packages are installed on a notebook instance, it is a good practice to save the instance as an image to prevent the dependency packages from being lost. For details, see **Saving a Notebook Environment Image**.

## 6.5 What Are Supported Software Versions for a Custom Image?

If your custom image uses software libraries such as NCCL, CUDA, and OFED, ensure that the software libraries meet the following version requirements:

- NCCL 2.7.8 or later
- OFED MLNX_OFED_LINUX-5.4-3.1.0.0 or later

- The CUDA version needs to be adapted to the GPU driver version of the dedicated resource pool. To obtain the GPU driver version, go to the dedicated resource pool details page.

# 7 Modification History

| Released Date | Description |
|---|---|
| 2023-10-01 | Added the following content:<br>- Added **Starting Training with a Preset Image**. |
| 2023-09-07 | Added the following content:<br>- Added **What Are Supported Software Versions for a Custom Image?**.<br>Modified the following content:<br>- Changed the manual name from "Using Custom Images" to "Image Management".<br>- Adjusted content in "Preset Images". Combined the development environment and training and inference base images into **Using a Preset Image**. |