ModelArts

Image Management

Issue	01
Date	2026-06-08

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

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 Functions in ModelArts

During AI service development, developers often face challenges such as complex environment dependencies, unstable development environments, and the difficulty of switching between multiple runtime environments. These issues significantly hinder development efficiency and slow down model iteration. To address these problems and ensure a stable and portable development environment, ModelArts offers a solution using container images. By encapsulating dependencies and runtime configurations into containers, developers can maintain consistent environments, manage dependencies more effectively, and switch between different runtime environments with ease. The container resources provided by ModelArts enable quick and efficient AI development and model experiment iteration.

This section describes the concepts related to images, application scenarios of preset images and custom images, and how to create custom images.

Concepts

Preset images: ModelArts provides preset images based on different frameworks and CUDA versions. When creating a notebook instance, training job, or real-time inference service, you can directly select a preset image provided by ModelArts. For details, see **ModelArts Preset Images**.

Unified images: are applicable to notebook creation, model training, and real-time inference. These images can be pulled using the **docker pull** command. For details, see **ModelArts Unified Images**.

Custom image: created by user by following the ModelArts image creation specifications. For details, see **Creating a Custom Image for a Notebook Instance**, **Creating a Custom Image for Model Training**, and **Creating a Custom Image for Inference**.

Base image: used for image creation. A base image can be a ModelArts preset image or a third-party image.

Applications of Preset Images

You can use ModelArts preset images to create notebook instances, training jobs, and real-time inference services. 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.

The preset images provided by ModelArts 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.

Applications of Custom Images

Preset images cannot meet the requirements for deep learning engines and development library. To resolve this issue, ModelArts allows image customization so that you can customize runtime engines. For details about how to create a custom image, see **Creating a Custom Image for a Notebook Instance**, **Creating a Custom Image for Model Training**, and **Creating a Custom Image for Inference**.

ModelArts runs in containers. You can customize container images to run on ModelArts. Custom images support CLI parameters and environment variables in free-text format, featuring high flexibility for a wide range of compute engines.

When you create a custom image, use the ModelArts preset image as the base image, that is, obtain the preset image using the SWR address in Dockerfile for image creation. You can obtain the SWR address of the image from the preset image list of ModelArts. For details, see **Viewing the Preset Image Version List/ Image Address**.

The rules for creating a custom image vary according to the application scenario. The details are as follows:

- General rule: SWR images can be shared with others only when the image type is **Private**. This rule applies to development environments, training jobs, and real-time inference services.
- Development environment: Other users can register and use SWR images on the ModelArts image management page only when the SWR image type is **Public**.
- Training job: When the SWR image type is **Public**, you can select the image source address as required when using a custom image to create a training job.

Custom Image Services

- 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 development environments, training jobs, and real-time inference 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 as 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 dependency services, such as SWR and OBS. The custom image can only be used after the access is authorized. It is a good practice to use an agency for authorization. After an 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 Agency Authorization for ModelArts with One Click**.

Using an Image Through a Command Line Tool

Based on the preset images provided by notebook or third-party images, you can use the ModelArts command line tool (**ma-cli image Commands for Building Images**) to create and register images, and build a custom image for AI development.

The **ma-cli image** command can be used to obtain registered images, obtain or load image creation templates, create images using Dockerfiles, obtain or clear image creation caches, register or deregister images, and debug whether images can be used in notebook instances. For details, run the **ma-cli image -h** command.

2 ModelArts Preset Images

When using ModelArts for machine learning development, you often need to configure the environment based on different frameworks and CUDA versions, which may increase configuration complexity and cause usage issues. To address these challenges, ModelArts provides preset images based on different frameworks and CUDA versions. You can directly select a preset image provided by ModelArts when creating a notebook instance, training a model, or performing real-time inference. By using preset images, you can quickly start the development environment, reduce configuration time, and focus on model development and training, thereby improving development efficiency.

Constraints

Container images based on the HCE system have the following restrictions:

The Docker version of the resource pool cluster must meet the following requirements: EulerOS 2.9 ≥ h59, EulerOS 2.10 ≥ h53, and EulerOS 2.11 ≥ h2. Otherwise, system compatibility problems may occur.

Log in to the resource pool cluster node to query the Docker version.

```
docker --version
```

Naming Conventions for Preset Images

Preset images in ModelArts comply with certain naming conventions. You can learn the basic information about an image based on its name. Generally, an image name contains the following fixed fields. You are advised to use unified naming conventions when adding custom images.

Table 2-1 Naming conventions for preset images

Framework	Example Name of a Preset Image	Image Name Interpretation
VeOmni	veomni_v0.1.6-pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b	• veomni_v0.1.6: indicates that the Veomni framework version is 0.1.6. • pytorch_2.7.1: indicates that the PyTorch framework version is 2.7.1. • cann_8.3.rc1: indicates that the version of the Compute Architecture for Neural Networks (CANN) software stack is 8.3 Release Candidate 1. • py_3.11: indicates that the Python version is 3.11. • hce_2.0.2509: indicates that the Huawei Cloud EulerOS version is 2.0.2509. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.
MindSpeed-LLM	mindspeed_llm_2.2.0-pytorch_2.7.1-cann_8.2.rc2-py_3.11-hce_2.0.2509-aarch64-snt9b	• mindspeed_llm_2.2.0: indicates that the MindSpeed Large Language Model (LLM) framework version is 2.2.0. • pytorch_2.7.1: indicates that the PyTorch framework version is 2.7.1. • cann_8.2.rc2: indicates that the version of the CANN software stack is 8.2 Release Candidate 2. • py_3.11: indicates that the Python version is 3.11. • hce_2.0.2509: indicates that the Huawei Cloud EulerOS version is 2.0.2509. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.

Framework	Example Name of a Preset Image	Image Name Interpretation
VeRL	verl_0.7.0-pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b	• verl_0.7.0: indicates that the VeRL framework version is 0.7.0. • pytorch_2.7.1: indicates that the PyTorch framework version is 2.7.1. • cann_8.3.rc1: indicates that the version of the CANN software stack is 8.3 Release Candidate 1. • py_3.11: indicates that the Python version is 3.11. • hce_2.0.2509: indicates that the Huawei Cloud EulerOS version is 2.0.2509. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.
AReaL	areal_0.5.1-pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b	• areal_0.5.1: indicates that the AReaL framework version is 0.5.1. • pytorch_2.7.1: indicates that the PyTorch framework version is 2.7.1. • cann_8.3.rc1: indicates that the version of the CANN software stack is 8.3 Release Candidate 1. • py_3.11: indicates that the Python version is 3.11. • hce_2.0.2509: indicates that the Huawei Cloud EulerOS version is 2.0.2509. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.

Framework	Example Name of a Preset Image	Image Name Interpretation
MindSpore	mindspore_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b	• mindspore_2.7.1: indicates that the MindSpore framework version is 2.7.1. • cann_8.3.rc1: indicates that the version of the CANN software stack is 8.3 Release Candidate 1. • py_3.11: indicates that the Python version is 3.11. • euler_2.10.11: indicates that the EulerOS version is 2.10.11. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	• tensorflow_2.1.0: indicates that the TensorFlow framework version is 2.1.0. • cuda_10.1: indicates that the CUDA version is 10.1. • py_3.7: indicates that the Python version is 3.7. • ubuntu_18.04: indicates that the OS is Ubuntu 18.04. • x86_64: indicates that the image is compiled for x86_64.
PyTorch	pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b	• pytorch_2.7.1: indicates that the PyTorch framework version is 2.7.1. • cann_8.3.rc1: indicates that the version of the CANN software stack is 8.3 Release Candidate 1. • py_3.11: indicates that the Python version is 3.11. • hce_2.0.2509: indicates that the Huawei Cloud EulerOS version is 2.0.2509. • aarch64: indicates that the image is compiled for the 64-bit Arm architecture. • snt9b: indicates that the image is adapted to the Ascend snt9b chip.

Capabilities of Preset Images

ModelArts provides official images based on different machine learning frameworks. The following describes the official images based on mainstream frameworks such as VeOmni, MindSpeed-LLM, VeRL, AReaL, and Conda.

VeOmni

VeOmni is an open-source full-modal distributed training framework developed by the seed team and built on PyTorch. With models as the core, the framework decouples distributed parallel logic from the model computation process, supports flexible combination of multiple parallel policies (such as FSDP, SP, and EP), and can be efficiently extended to training scenarios of ultra-long sequences and large-scale MoE models. VeOmni provides lightweight, full-modal APIs to streamline access to multimodal codecs. It incorporates system-level optimizations, including dynamic batch processing and efficient operators, to boost both training efficiency and stability.

Highlights

- Decoupling between model computation and parallelism: Based on PyTorch's native distributed capabilities, the framework decouples parallelism logic from model computation, supporting flexible combinations of Fully Sharded Data Parallel (FSDP), Sequence Parallelism (SP), and Expert Parallelism (EP).
- Full-modal access: Lightweight full-modal APIs are provided to simplify the access process of multimodal (such as visual and voice) codecs and support complex multimodal model training.
- Ascend operator-level optimization: The framework is deeply optimized for PyTorch NPU (torch_npu 2.7.1), integrating dynamic batching and high-performance fused operators to enhance computing throughput across full-modal scenarios.
- Ultra-large-scale scalability: The framework enables efficient training of ultra-long sequences and is optimized for large-scale MoE architectures to ensure training stability.

MindSpeed-LLM

MindSpeed-LLM is a distributed training suite for LLMs built on the Ascend computing ecosystem. It deeply integrates the MindSpeed acceleration library and Megatron-LM core architecture to provide end-to-end large model training solutions for partners. This image integrates distributed pre-training, instruction fine-tuning, and full-process toolchain (data preprocessing, weight conversion, operator acceleration, and baseline evaluation) to implement an out-of-the-box Huawei Ascend NPU development environment.

Core Component Version

The following core components have been pre-installed and adapted to the environment in this image:

- MindSpeed-LLM: **GitCode (Ascend)**
- MindSpeed: **GitHub (NVIDIA)**

Highlights

- Ascend native acceleration: MindSpeed is deeply integrated to optimize the instruction set for torch_npu and support efficient mixed precision training and operator fusion.
- Distributed training: Based on the Megatron-LM core, multi-dimensional parallelism policies such as tensor parallelism (TP) and pipeline parallelism (PP) are supported.
- Full-stack toolchain: It provides built-in weight conversion tools (supporting format conversion of mainstream open-source models to NPU-compatible formats), distributed data preprocessing, and Ray-based resource scheduling.
- Ecosystem compatibility: Mainstream LLM architectures are supported, and the PEFT and Transformers libraries are integrated to facilitate the migration of open-source community models.

VeRL

VeRL is an easy-to-use and high-performance open-source framework in the reinforcement learning field. With its modular algorithm library and native distributed training features, VeRL has become a preferred tool for implementing reinforcement learning solutions in scientific research and industry. This image deeply integrates the VeRL core framework, covering the entire process of reinforcement learning algorithm R&D, policy training, and simulation verification.

Application Scenarios

- Reinforcement learning algorithm development and experiment: This image can be used to quickly design and verify reinforcement learning algorithms in Notebook.
- Training and tuning: A single device or multiple devices are used to perform large-scale reinforcement learning training in a training job.
- Education and training: A stable and consistent lab environment is provided to facilitate course teaching and skill training.

AReaL

As one of the mainstream open-source frameworks in the reinforcement learning field, AReaL is widely favored due to its fully asynchronous training and inference and active community ecosystem. This image integrates AReaL and its ecosystem components (vLLM and vLLM-Ascend) and provides multiple version combinations to meet the requirements of different development, training, and deployment scenarios.

ModelArts provides flexible combinations of AReaL, Python, and CANN versions (applicable to Huawei Ascend NPUs). You can select a proper image version to implement an out-of-the-box reinforcement learning environment.

Highlights

- Pre-installed dependencies
  - Common scientific computing libraries: NumPy and Pandas
  - Inference frameworks: vLLM and vLLM-Ascend
  - Model deployment tool: TensorBoard

- Out-of-the-box availability: No additional environment configuration is required. The best practice script is directly started to automatically start the cluster and training.

Application Scenarios

This framework applies to most scenarios of reinforcement learning.

MindSpore

MindSpore is a deep learning framework in all scenarios, aiming to achieve easy development, efficient execution, and all-scenario unified deployment. MindSpore preset images have MindSpore pre-installed and provide multiple combinations of Python, CANN, EulerOS, and Ubuntu versions. They support Ascend accelerator card scenarios.

TensorFlow

TensorFlow is an end-to-end platform that enables researchers to advance cutting-edge technologies in the field of machine learning and allows developers to easily build and deploy machine learning-driven applications. TensorFlow preset images have TensorFlow pre-installed and provide multiple combinations of Python, CUDA, CANN, EulerOS, and Ubuntu versions. They support GPU and Ascend accelerator card scenarios.

PyTorch

As one of the mainstream open-source frameworks in the deep learning field, PyTorch is widely favored for its dynamic graph mechanism, Python-first design, and active community ecosystem. PyTorch preset images integrate PyTorch and its ecosystem components (Torchvision and Torchaudio) and provide multiple version combinations to meet the requirements of different development, training, and deployment scenarios.

ModelArts provides flexible combinations of PyTorch, Python, CUDA (for NVIDIA GPUs), and CANN (for Huawei Ascend NPUs), and is compatible with multiple OSs, such as Ubuntu and EulerOS. You can select a proper image version based on your hardware platform (NVIDIA GPU or Huawei Ascend NPU) and project requirements to implement an out-of-the-box deep learning environment.

Highlights

- Multi-hardware support
  - NVIDIA GPU: supports CUDA acceleration and provides optimization libraries such as cuDNN and NCCL.
  - Huawei Ascend NPU: integrates the torch_npu plug-in to support mixed precision training and distributed training.
- Pre-installed dependencies
  - Common scientific computing libraries: NumPy, SciPy, and Pandas
  - Visual processing libraries: OpenCV and Pillow
  - Model deployment tools: ONNX Runtime and TensorBoard
  - Development tools: JupyterLab, IPython, tqdm, and more

- Out-of-the-box availability: No additional environment configuration is required. You can directly start notebook instances, real-time services, or training jobs.

Application Scenarios

- AI development and experiment: This image can be used to quickly design, debug, and visualize model prototypes in Notebook.
- Training and tuning: A single device or multiple devices are used to perform large-scale data training in a training job.
- Deployment and servitization: You can deploy a trained model as a RESTful API through a real-time service.
- Education and training: A stable and consistent lab environment is provided to facilitate course teaching and skill training.

Core Images

The following lists the core preset images based on the **PyTorch**, **MindSpore**, and **TensorFlow** frameworks. The images available in each region may vary.

- For details about the image versions and image path supported by ModelArts, see **Viewing the Preset Image Version List/Image Address**.
- For details about how to view the details of an image component, see **Viewing Details About an Image Component**.

Table 2-2 PyTorch preset images

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b23	Ascend Snt9b2x (such as Snt9b23)	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.12, and Huawei Cloud EulerOS to 2.0.2512.	Viewing the image Address

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.12, and Huawei Cloud EulerOS to 2.0.2512.	
pytorch_2.7.1-cann_8.5.2-py_3.12-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.12, and EulerOS to 2.10.11.	
pytorch_2.6.0-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated PyTorch to 2.6.0, CANN to 8.2.RC1, and the driver to Ascend HDK 25.2.0.	

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_2.1.0-cann_8.1.rc1-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.1.RC1.B150 and the driver to Ascend HDK 25.0.RC1.	
pytorch_2.1.0-cann_8.0.rc3-py_3.9-euler_2.10.10-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc3 and the driver to Ascend HDK 24.1.RC3.	
pytorch_2.1.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc2 and the driver to Ascend HDK 24.1.RC2.	

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_2.1.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc1.	
pytorch_2.1.0-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.0.beta1 and the driver to Ascend HDK 24.1.0.	
pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc2 and the driver to Ascend HDK 24.1.RC2.	

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_1.11.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc1.	
pytorch_2.7.1-cann_8.3.rc1-py_3.11-euler2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2025-12-08	Upgraded Python to 3.11, upgraded third-party software, and more.	
pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-d910b	Ascend snt9b	Notebook, training, and inference deployment	2023/09/14	Updated CANN to 6.3.2 and Python to 3.7.	

Image	Supported Chip	Applicable Scope	Created	Description	Image Address
pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2025-11-11	Upgraded Python to 3.11 and OS to HCE 2.0.2509, and updated AReaL.	

Table 2-3 MindSpore preset images

Preset Image	Supported Chip	Applicable Scope	Created	Description	Image Address
mindspore_2.7.2-cann_8.5.2-py_3.11-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.11, and EulerOS to 2.10.11.	**Viewing the image Address**

Preset Image	Supported Chip	Applicable Scope	Created	Description	Image Address
mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.11, and Huawei Cloud EulerOS to 2.0.2512.	
mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b23	Ascend Snt9b2x (such as Snt9b23)	Notebook, training, and inference deployment	2026/04/29	Updated CANN to 8.5.2, Python to 3.11, and Huawei Cloud EulerOS to 2.0.2512.	
mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated MindSpore to 2.7.0rc1, CANN to 8.2.RC1, and the driver to Ascend HDK 25.2.0.	
mindspore_2.6.0rc1-cann_8.1.rc1-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated MindSpore to mindspore_2.6.0rc1, CANN to 8.1.RC1.B150, and the driver to Ascend HDK 25.0.RC1.	

Preset Image	Supported Chip	Applicable Scope	Created	Description	Image Address
mindspore_2.4.10-cann_8.0.0-py_3.10-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated MindSpore to 2.4.10, CANN to 8.0.0.beta1, and the driver to Ascend HDK 24.1.0.	
mindspore_2.4.0-cann_8.0.rc3-py_3.9-euler_2.10.10-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated CANN to 8.0.rc3 and the driver to Ascend HDK 24.1.RC3.	
mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated MindSpore to 2.3.0, CANN to 8.0.rc2, and the driver to Ascend HDK 24.1.RC2.	
mindspore_2.3.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2026/02/04	Updated MindSpore to 2.3.0-rc4 and CANN to 8.0.rc1. Taken ma-cau1.1.6 and ma-cau-adapter1.1.3 offline.	

Preset Image	Supported Chip	Applicable Scope	Created	Description	Image Address
mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2023/11/17	Updated CANN to 6.3.2 and Python to 3.7.	
mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2023/08/11	Updated CANN to 7.0.1 and Python to 3.9.	
mindspore_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment	2025-12-05	Upgraded Python to 3.11, upgraded third-party software, and more.	

Table 2-4 TensorFlow preset images

Preset Image	Supported Chip	Applicable Scope	Updated	Description	Image Address
tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	GPU (CUDA 10.1)	Notebook, training, and inference deployment	2022-09-26	Updated CUDA to 10.1 and Python to 3.7.	**Viewing the image Address**

Viewing the Preset Image Version List/Image Address

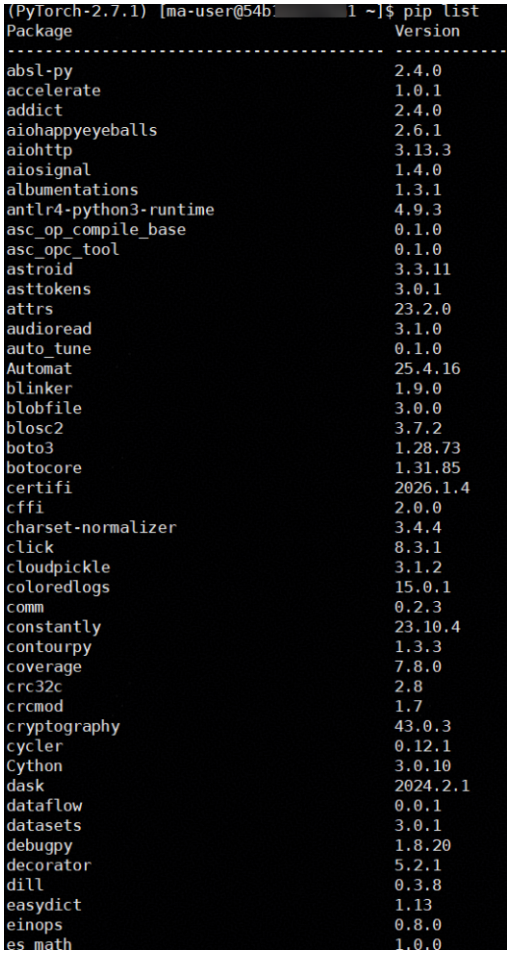
1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Asset Management** > **Image Management**.
2. On the page that appears, view the image name, organization, and total number of versions.
3. Click the image name to view the preset image version list, including the image version, status, resource type, image size, and SWR address.

Viewing Details About an Image Component

You can use either of the following methods to view details about an image component:

- Method 1: View the details in a container.
  a. Run the following command to start the container:
  ```
  docker run -it image_name bash
  ```
  Replace **image_name** with the actual image path. For details about how to obtain the image name, see **Viewing the Preset Image Version List/Image Address**.
  b. After accessing the container, run the following command to view the version information:
  ```
  pip list
  ```
- Method 2: View the details in a notebook instance.
  – Use a preset image to create a notebook instance and access the instance in JupyterLab mode. For details, see **Using JupyterLab to Develop and Debug Code Online**.
  – Open a Terminal interface, and then run the following command to view the version information:
  ```
  pip list
  ```


Figure 2-1 Viewing image component details



```
(PyTorch-2.7.1) [ma-user@54b      1 ~]$ pip list
Package                                  Version
---------------------------------------- ------------
absl-py                                  2.4.0
accelerate                               1.0.1
addict                                   2.4.0
aiohappyeyeballs                         2.6.1
aiohttp                                  3.13.3
aiosignal                                1.4.0
albumentations                           1.3.1
antlr4-python3-runtime                   4.9.3
asc_op_compile_base                      0.1.0
asc_opc_tool                             0.1.0
astroid                                  3.3.11
asttokens                                3.0.1
attrs                                    23.2.0
audioread                                3.1.0
auto_tune                                0.1.0
Automat                                  25.4.16
blinker                                  1.9.0
blobfile                                 3.0.0
blosc2                                   3.7.2
boto3                                    1.28.73
botocore                                 1.31.85
certifi                                  2026.1.4
cffi                                     2.0.0
charset-normalizer                       3.4.4
click                                    8.3.1
cloudpickle                              3.1.2
coloredlogs                              15.0.1
comm                                     0.2.3
constantly                               23.10.4
contourpy                                1.3.3
coverage                                 7.8.0
crc32c                                   2.8
crcmod                                   1.7
cryptography                             43.0.3
cycler                                   0.12.1
Cython                                   3.0.10
dask                                     2024.2.1
dataflow                                 0.0.1
datasets                                 3.0.1
debugpy                                  1.8.20
decorator                                5.2.1
dill                                     0.3.8
easydict                                 1.13
einops                                   0.8.0
es_math                                  1.0.0
```


3 ModelArts Unified Images

When using ModelArts for machine learning development, you often need to configure the environment based on different frameworks, which may increase configuration complexity and cause usage issues. How to reduce configuration complexity and usage problems? ModelArts provides unified images based on different frameworks. When creating a notebook instance, training a model, or performing online inference, you can pull the images in docker pull mode to quickly start the development environment. This reduces the configuration time and enables you to focus on model development and training, improving the development efficiency.

For details about how to pull a unified image using the **docker pull** command, see **Creating a Custom Image for a Notebook Instance**, **Creating a Custom Image for Model Training**, and **Creating a Custom Image for Inference**.

Unified PyTorch Images

Table 3-1 Unified PyTorch images

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-euler_2.10.11-aarch64-snt9b-20260417112518-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-euler_2.10.11-aarch64-snt9b-20260417112518-aabfd52	MA+A2+EULER	Updated CANN to 8.5.2, Python to 3.12, and EulerOS to 2.10.11.
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b-20260417112518-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b-20260417112518-aabfd52	MA+A2+HCE2.0	Updated CANN to 8.5.2, Python to 3.12, and Huawei Cloud EulerOS to 2.0.2512.

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b23-20260417112518-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.5.2-py_3.12-hce_2.0.2512-aarch64-snt9b23-20260417112518-aabfd52	MA+A3+HCE2.0	Updated CANN to 8.5.2, Python to 3.12, and Huawei Cloud EulerOS to 2.0.2512.
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b-20251205091605-e41006e Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b-20251205091605-e41006e	MA+A2+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b23-20251205091605-e41006e Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b23-20251205091605-e41006e	MA+A3+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt3p-20251205091605-e41006e Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt3p-20251205091605-e41006e	MA+310P	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20251208183724-e1cc47c Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20251208183724-e1cc47c	MA+A2+EULER	Upgraded Python to 3.11, upgraded third-party software, and more.
pytorch_cuda	swr.<region>.myhuaweicloud.com/atelier/pytorch_cuda:pytorch_2.7.0-cuda_12.8-py_3.11.10-ubuntu_22.04-x86_64-20251215163925-4e5422a Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_cuda:pytorch_2.7.0-cuda_12.8-py_3.11.10-ubuntu_22.04-x86_64-20251215163925-4e5422a	MA+Ubuntu22.04	Upgraded CUDA to 12.8, upgraded Python to 3.11, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20250908155148-2c423b4 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20250908155148-2c423b4	MA +A2+EULER	Upgraded Python to 3.11, upgraded third-party software, and more.
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt3p-20250908091038-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt3p-20250908091038-4a84562	MA+310P	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b-20250908091038-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b-20250908091038-4a84562	MA +A2+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.
pytorch_ascend	swr.<region>.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b23-20250908091038-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.6.0-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b23-20250908091038-4a84562	MA +A3+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.

Unified MindSpore Images

Table 3-2 Unified MindSpore images

Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-euler_2.10.11-aarch64-snt9b-20260417155905-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-euler_2.10.11-aarch64-snt9b-20260417155905-aabfd52	MA+A2+EULER	Updated CANN to 8.5.2, Python to 3.11, and EulerOS to 2.10.11.
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b-20260417155905-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b-20260417155905-aabfd52	MA+A2+HCE2.0	Updated CANN to 8.5.2, Python to 3.11, and Huawei Cloud EulerOS to 2.0.2512.

Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b23-20260417155905-aabfd52 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.2-cann_8.5.2-py_3.11-hce_2.0.2512-aarch64-snt9b23-20260417155905-aabfd52	MA+A3+HCE2.0	Updated CANN to 8.5.2, Python to 3.11, and Huawei Cloud EulerOS to 2.0.2512.
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b-20251204204157-4708bb0 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b-20251204204157-4708bb0	MA+A2+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b23-20251204204157-4708bb0 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt9b23-20251204204157-4708bb0	MA+A3+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt3p-20251205162951-b29ea96 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-hce_2.0.2509-aarch64-snt3p-20251205162951-b29ea96	MA+310P	Upgraded Python to 3.11 and OS to HCE.2509, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20251205162951-b29ea96 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.1-cann_8.3.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20251205162951-b29ea96	MA+A2+EULER	Upgraded Python to 3.11, upgraded third-party software, and more.
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20250908143531-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-euler_2.10.11-aarch64-snt9b-20250908143531-4a84562	MA +A2+EULER	Upgraded Python to 3.11, upgraded third-party software, and more.

Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt3p-20250908143531-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt3p-20250908143531-4a84562	MA+310P	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b-20250908143531-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b-20250908143531-4a84562	MA +A2+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.

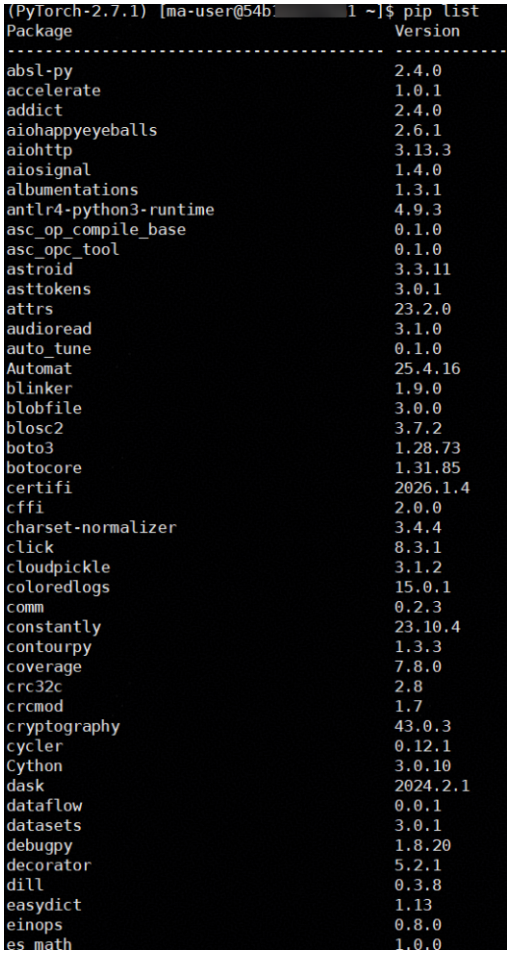
Image Name	Image Link	Image Type	Description
mindspore_ascend	swr.<region>.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b23-20250908143531-4a84562 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_ascend:mindspore_2.7.0rc1-cann_8.2.rc1-py_3.11-hce_2.0.2506-aarch64-snt9b23-20250908143531-4a84562	MA +A3+HCE2.0	Upgraded Python to 3.11 and OS to HCE.2506, upgraded third-party software, and more.

Viewing Details About an Image Component

You can use either of the following methods to view details about an image component:

- Method 1: View the details in a container.
  a. Run the following command to start the container:
     ```
     docker run -it image_name bash
     ```
     Replace **image_name** with the actual image path. For details about image path, see the preceding description.
  b. After accessing the container, run the following command to view the version information:
     ```
     pip list
     ```
- Method 2: View the details in a notebook instance.
  - Use a preset image to create a notebook instance and access the instance in JupyterLab mode. For details, see **Using JupyterLab to Develop and Debug Code Online**.
  - Open a terminal interface, and then run the following command to view the version information:
    ```
    pip list
    ```


Figure 3-1 Viewing image component details



```
(PyTorch-2.7.1) [ma-user@54b      1 ~]$ pip list
Package                                  Version
---------------------------------------- ------------
absl-py                                  2.4.0
accelerate                               1.0.1
addict                                   2.4.0
aiohappyeyeballs                         2.6.1
aiohttp                                  3.13.3
aiosignal                                1.4.0
albumentations                           1.3.1
antlr4-python3-runtime                   4.9.3
asc_op_compile_base                      0.1.0
asc_opc_tool                             0.1.0
astroid                                  3.3.11
asttokens                                3.0.1
attrs                                    23.2.0
audioread                                3.1.0
auto_tune                                0.1.0
Automat                                  25.4.16
blinker                                  1.9.0
blobfile                                 3.0.0
blosc2                                   3.7.2
boto3                                    1.28.73
botocore                                 1.31.85
certifi                                  2026.1.4
cffi                                     2.0.0
charset-normalizer                       3.4.4
click                                    8.3.1
cloudpickle                              3.1.2
coloredlogs                              15.0.1
comm                                     0.2.3
constantly                               23.10.4
contourpy                                1.3.3
coverage                                 7.8.0
crc32c                                   2.8
crcmod                                   1.7
cryptography                             43.0.3
cycler                                   0.12.1
Cython                                   3.0.10
dask                                     2024.2.1
dataflow                                 0.0.1
datasets                                 3.0.1
debugpy                                  1.8.20
decorator                                5.2.1
dill                                     0.3.8
easydict                                 1.13
einops                                   0.8.0
es_math                                  1.0.0
```


4 Image Registration and Management

ModelArts allows you to create and run container images tailored to your requirements. This section describes how to register an image and view and delete the registered image.

Registering an Image

1. Log in to the **ModelArts console**. In the navigation pane, choose **Asset Management** > **Image Management**. Click **Register**.
2. On the **Register Image** page, set the following parameters and click **Register Now**.

Table 4-1 Parameters

Parameter	Description
SWR Source	Use one of the following methods to configure the image source: • Method 1: Enter the image path. Example path: *<swr-domain-name>*/*<namespace>*/*<repository>*:*<tag>* • Method 2: Click , select an image, and click **OK**.
Description	Enter a description for the image version. It can contain up to 256 characters and cannot include the following special characters: &<>"'/
Architecture	Select **ARM** or **X86_64** based on your requirements. The architecture must match that of the image source.
Type	Select a type based on your requirements. The type must match that of the image source.
Specifications	Select the target specifications for the image.

Creating a Notebook Instance Using a Registered Image

1. On the **Image Management** page, click the name of the target image.
2. In the **Operation** column of the target image version, click **Create Notebook**.
3. On the **Create Notebook** page, configure the required information and click **Next**.

   For details about the parameters, see **Creating a Notebook Instance (New Page)**.

Synchronizing an Image

Synchronize the image status from SWR.

1. On the **Image Management** page, click the name of the target image.
2. In the **Operation** column of the target image version, click **Sync**.

   If the message **Synchronization status command set successfully** is displayed, the image synchronization is successful.

Deleting a Registered Image

Deleted images cannot be restored. After an image is deleted, all instances using this image will fail to be restarted. Exercise caution when performing this operation.

1. On the **Image Management** page, click the name of the target image.
2. In the **Operation** column of the target image version, click **Delete**.
3. In the **Delete Image** dialog box, enter **DELETE** and click **OK**.

5 Creating a Custom Image for a Notebook Instance

5.1 Creating a Custom Image

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.

Process for Creating a Custom Image

Scenario 1: Create a notebook instance using a preset image, install custom software and dependencies on the preset image, and save the running instance environment as a container image. For details, see **Creating a Custom Image Using the Image Saving Function**.

Scenario 2: Build and register an image based on the preset image or third-party image in a notebook instance, configure the Docker environment on the server, and compile the Dockerfile. For details, see **Creating a Custom Image on ECS and Using It**.

Specifications for Custom Images

The base image for creating a custom image must meet either of the following conditions:

- It is an open-source image from the official website of Ascend or Docker Hub and it meets the following OS constraints:

  x86: Ubuntu 18.04 or Ubuntu 20.04

  Arm: Euler 2.8.3 or Euler 2.10.7

> **NOTE**
>
> There may be a compatibility issue for Ubuntu 20.04.6. Use an earlier version.

- If an image error occurs due to unmet requirements, check the image specifications and rectify the fault by referring to **Troubleshooting for Custom Images in Notebook Instances**. If the fault persists, contact technical support.

Registering a Created Image

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

> **NOTE**
>
> Only the IAM users of the account can register and use the SWR images when the image type is **Private**.
>
> Other users can register and use SWR images when the image type is **Public**.

1. Log in to the **ModelArts console**. In the navigation pane, choose **Asset Management** > **Image Management**. Click **Register**.
2. Configure parameters and click **Register**.
   - **SWR Source**: Select a built image as the image source.

     Figure 5-1 Selecting an image source

   - **Architecture**, **Type**, and **Specifications**: Configure them based on the actual framework of the custom image.
3. The registered image is displayed on the **Image Management** page.

5.2 Creating a Custom Image Using the Image Saving Function

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. After the image is saved, the default working directory is the **/** path in the root directory.

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.

The image saving function implements version iterations by stacking a new layer onto the original image. Due to the immutability of image layers, the size of the newly generated image will always exceed that of the original, regardless of whether addition or deletion is performed.

NOTE

- If the image fails to be saved, view the event on the notebook instance details page. For details, see **Viewing Notebook Events**.
- The image to be saved should not be larger than 35 GB and there should be no more than 125 layers. Otherwise, the image may fail to be saved. For details, see **Space Allocation for Container Engines**.
  - If a dedicated resource pool is used, log in to the ModelArts console. On the dedicated resource pool scaling page, configure the container engine size as needed. For details, see **Resizing a Dedicated Resource Pool**.
  - If the fault persists, contact technical support.
- When you use the containerd image saving function, pay attention to the following:
  - Do not download files and save images at the same in an instance to prevent performance deterioration caused by resource competition.
  - If there is a file larger than 10 GB in an instance system folder (for example, **/opt/**), you should not use the image saving function, in case the image size is too large or the saving duration is too long.

Prerequisites

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

Saving an Image

1. Locate the target notebook instance in the list and click **Save Image** in the **Operation** column.
2. On the displayed page, set the parameters, select **I understand the risks caused by the change and agree to save the image**, and click **OK**.

   You can save the image as a new version of an existing image or create a new image.

   - **Save as new version**: Save the image as a new version of a registered image, that is, in an existing image folder in the **Registered Images** tab.
   - **Create new image**: The image will be saved to Software Repository for Container (SWR) and automatically registered with ModelArts upon completion.

Table 5-1 Parameters

Save Options	Parameter	Description
Save as new version	Image	Click the image selection box. In the **Image** panel, select an image from the basic or enterprise edition as needed, and then click **OK**.
	Image Version	Enter a version for the custom image. It can contain letters, digits, underscores (_), hyphens (-), and periods (.). The maximum length is 64 characters.

Save Options	Parameter	Description
	Description	Enter a description for the custom image. It cannot contain the characters & < > " ' /. The length must be between 0 and 256 characters.
Create new image	Image Repository	Select **SWR Basic Edition** or **SWR Enterprise Edition** as needed. By default, **SWR Basic Edition** is used. If you want to use an enterprise repository, go to the **SWR Enterprise Edition console** to register one. The repository in the Enterprise Edition is identified by its registry name. For details about the differences between the basic and enterprise editions, see **Differences Between SWR Basic and Enterprise Editions**.
	Domain Name	This parameter is mandatory only if **Image Repository** is set to **SWR Enterprise Edition**. It specifies the access address of the SWR Enterprise Edition instance.
	Namespace	This parameter is mandatory only if **Image Repository** is set to **SWR Enterprise Edition**. Namespaces are used to isolate image repositories in SWR Enterprise Edition.
	Organization	This parameter is mandatory only if **Image Repository** is set to **SWR Basic Edition**. Organizations are used in SWR to isolate images. An organization can represent a company or department, and its images are managed centrally under that organization. Different organizations may have images with identical names. An IAM user can join different organizations. Users in an organization can share all images in the organization. If no organization is available, click **Go to SWR to create one**. For details, see **Organization Management**.
	Image Name	Enter a name for the custom image. It can only contain lowercase letters, digits, and special characters (_/.-). It must start and end with a lowercase letter or digit and cannot contain more than two consecutive underscores. The maximum length is 128 characters.
	Image Version	Enter a version for the custom image. It can contain letters, digits, underscores (_), hyphens (-), and periods (.). The maximum length is 64 characters.

Save Options	Parameter	Description
	Description	Enter a description for the image. It cannot contain the characters & < > " ' /. The length must be between 0 and 256 characters.

3. Wait until the image is saved. The image will be saved as a snapshot. The process takes approximately 3 to 10 minutes, during which the instance status will be **Snapshotting**. During this period of time, do not perform any operations on the instance. After the image is saved, the instance status changes to **Running**.

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.
- The connection may temporarily drop during the saving process; it will recover once the operation is complete.
- The saved image does not contain files and data in the mount directory (**/home/ma-user/work**) used for persistent storage.

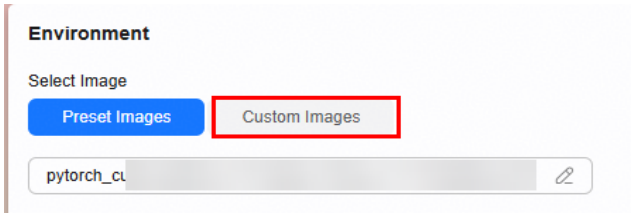
4. View image details.
   a. In the navigation pane on the left, choose **Asset Management** > **Image Management**.
   b. In the **Registered Images** tab, click the name of the image to go to the image details page and view the image details.

Using a Custom Image to Create a Notebook Instance

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

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

Figure 5-2 Selecting a custom image to create a notebook instance



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

Which Data Can Be Saved When I Save an Image?

- Data that can be saved: Files and directories that are statically added to images during container building.

  For example, dependencies and the **/home/ma-user** directory are saved in the image environment.

- Data that cannot be saved: Mounting directories or data volumes that are dynamically connected to the host during container startup. You can run the **df -h** command to view the mounted dynamic directories. Data that is not in the **/** path will not be saved.

  For example, data that is persistently stored in **home/ma-user/work** and data that is dynamically mounted to **/data** is not saved.

5.3 Creating a Custom Image on ECS and Using It

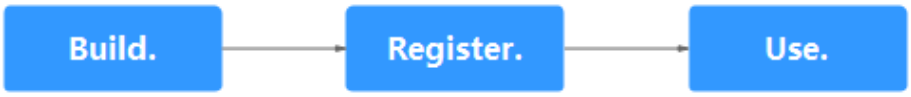
Application Scenarios and Process

You can write a Dockerfile based on a preset base image or third-party image to customize your image on ECS. Then, 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 5-3 Creating and debugging an image



Specifications for Custom Images

The base image for creating a custom image must meet either of the following conditions:

- It is an open-source image from the official website of Ascend or Docker Hub and it meets the following OS constraints:

  x86: Ubuntu 18.04 or Ubuntu 20.04

  Arm: Euler 2.8.3 or Euler 2.10.7

  NOTE

  There may be a compatibility issue for Ubuntu 20.04.6. Use an earlier version.

- If an image error occurs due to unmet requirements, check the image specifications and rectify the fault by referring to **Troubleshooting for Custom Images in Notebook Instances**. If the fault persists, contact technical support.

Procedure

1. Prepare a Linux environment. The following uses ECS as an example.
2. Create an image on ECS. The Dockerfile sample file is provided.
3. Upload the created image to SWR.
4. Register an SWR image on ModelArts.
5. Create a notebook instance and verify the new image.

Preparing a Docker Server and Configuring the 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.

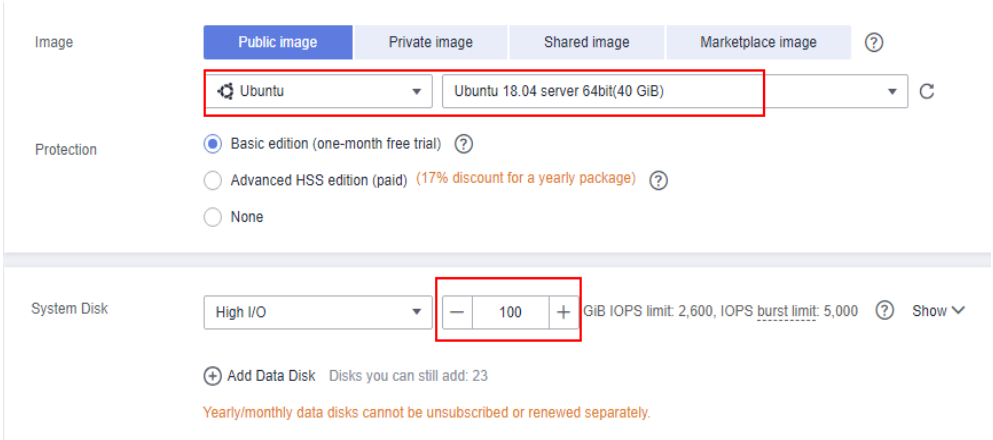
ModelArts provides Ubuntu scripts for you to install Docker easier.

NOTE

Operations on a local Linux server are identical to those on an ECS server. For details, see this case.

1. 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 details, see **Purchasing and Logging In to a Linux ECS**.

   Figure 5-4 Selecting an image and a disk

2. Purchase an EIP and bind it to the ECS. For details, see **Configure Network**.
3. Configure the VM environment.
   a. 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.

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

```
bash install_on_ubuntu1804.sh
```

Figure 5-5 Configuration succeeded


```
source /etc/profile
```

The installation script is executed to:

i. Install Docker.

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

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 the Dockerfile, see **Dockerfile reference**.

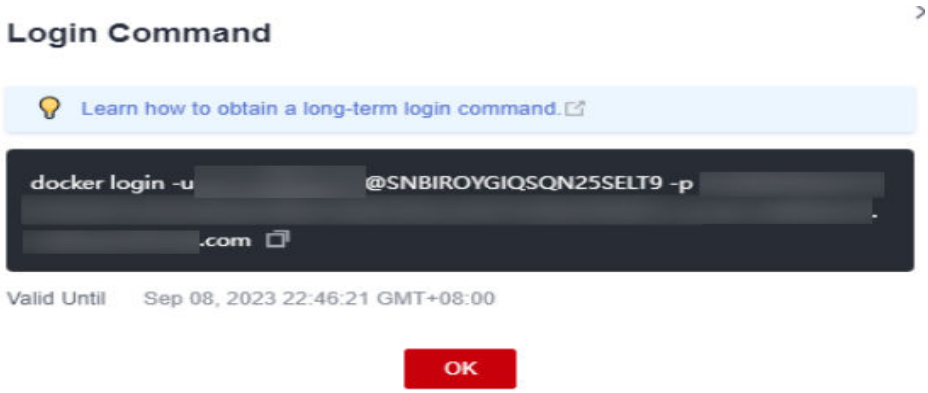
1. Querying Base Images (Skip This Step for Third-Party Images)

   For details about ModelArts base images, see **ModelArts Unified Images**. Check the image URL in the corresponding section based on the engine type of the preset image.

2. Access SWR.

   a. Log in to the SWR console. 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 5-6 Obtaining the login command



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.

b. Run the login command on the machine where the container engine is installed. The message "Login Succeeded" will be displayed upon a successful login.

3. 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
```

4. 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.

5. Build an image.

   Run the **docker build** command to build a new image from the Dockerfile. The descriptions 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.
   - The period (**.**) 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 5-7 Image created

Registering an Image

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

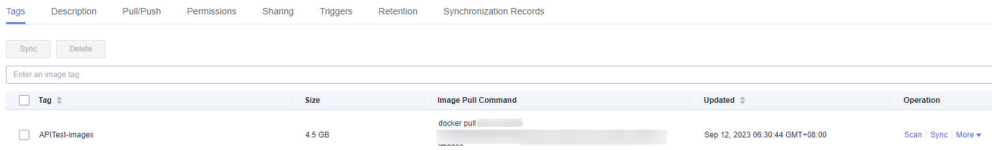
1. Upload the image to SWR.

   Log in to SWR first. For details, see **Logging 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 5-8 Upload the image to SWR.



2. Register an Image.

   Registering an image on the ModelArts console

   a. Log in to the **ModelArts console**. In the navigation pane, choose **Asset Management** > **Image Management**. Click **Register**.

   b. Set **SWR Source** to the image pushed to SWR in **step 1**.

   c. Set **Architecture**, **Type**, and **Specifications** based on site requirements. The values must be those of the image source.

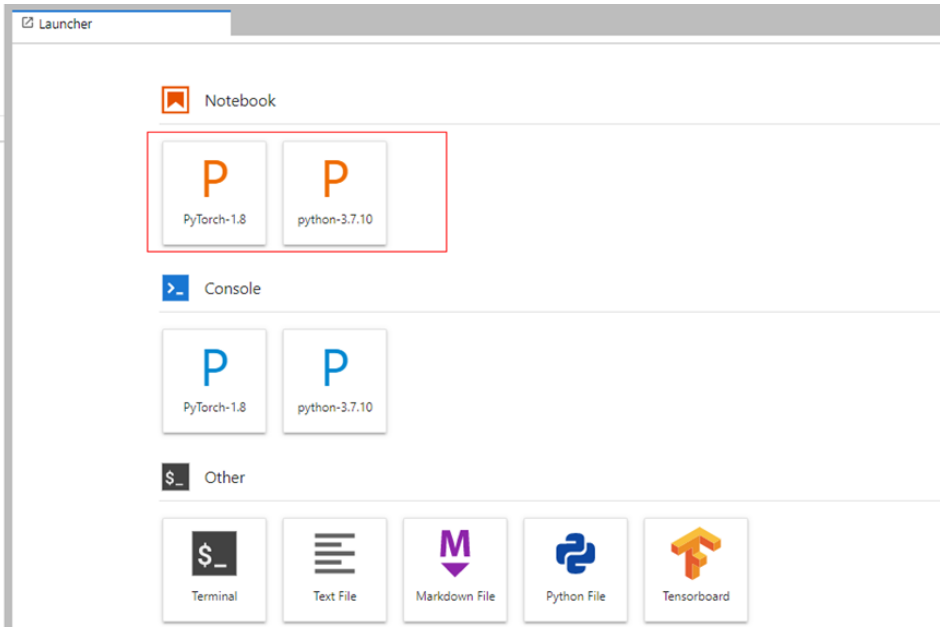
   NOTE

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

Using a New Image to Create a Development Environment

1. After an image is registered in **2**, use it to create a development environment on the **Notebook** page of the **ModelArts console**.
2. After the notebook instance is created, go to the **Notebook** page. Locate the target notebook instance and click **Access Environment** in the **Operation** column. Then, click **Access** next to **JupyterLab Access**.

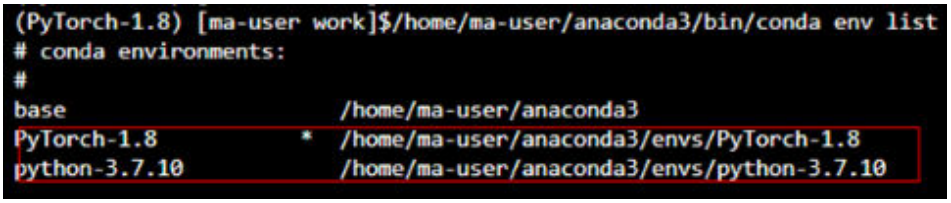
   Figure 5-9 Accessing a development environment



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 5-10 Checking the conda environment



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 following provides two Dockerfile file examples for your reference.

- Example 1: Basic configuration, that is, adding a specified user to a user group.

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

- Example 2: Basic configuration + configuration of related tools and environments.
  - Basic configuration: adding a specified user to a user group
  - (Optional) Example of configuring related tools and environments:
    - Configure the APT source to accelerate software package download.
    - Install Miniconda and configure the Conda and pip sources.
    - Create a Conda environment that contains specific Python packages (such as PyTorch and TorchVision).
    - Install third-party packages, for example, FFmpeg and GCC 8, to support multimedia processing and compilation tasks.

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

#Configure the APT source and install the ZIP and Wget tools (required for installing conda).
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb http://
repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://repo.huaweicloud.com/
```



```
ubuntu/ bionic universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates universe\ndeb
http://repo.huaweicloud.com/ubuntu/ bionic multiverse\ndeb http://repo.huaweicloud.com/ubuntu/
bionic-updates multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main
restricted universe multiverse\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main
restricted\ndeb http://repo.huaweicloud.com/ubuntu bionic-security universe\ndeb http://
repo.huaweicloud.com/ubuntu bionic-security multivers e" > /etc/apt/sources.list && \
apt-get update && \
apt-get install -y zip wget

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

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

#Configure the conda and pip sources
RUN mkdir -p /home/ma-user/.pip && \
    echo -e "channels:\n  - defaults\nshow_channel_urls: true\ndefault_channels:\n  - https://
mirrors.tuna.tsinghua.edu.cn/anaconda/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
```

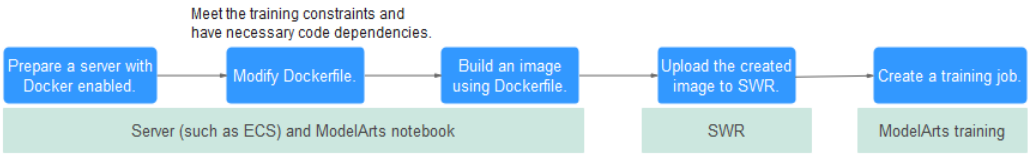
6 Creating a Custom Image for Model Training

6.1 Creating a Custom Training Image

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.

Creating a Custom Training Image

Figure 6-1 Creating a Custom Training Image



Scenario 1: If the preset images meet ModelArts training constraints but lack necessary code dependencies, install additional software packages.

For details, see **Creating a Custom Training Image Using a Preset Image**.

Scenario 2: If the local images meet code dependency requirements but not ModelArts training constraints, adapt them to ModelArts.

For details, see **Migrating Existing Images to ModelArts**.

Scenario 3: If neither the preset nor local images meet your needs, create an image that meets both code dependency and ModelArts training constraints. For details, see the following cases:

Creating a Custom Training Image (PyTorch + Ascend)

Creating a Custom Image for Training (PyTorch + CPU)

Creating a Custom Image for Training (MPI + CPU)

Creating a Custom Training Image (Tensorflow + GPU)

Constraints on Custom Images of the Training Framework

- When packaging an image, ensure that the image version is compatible with the Docker version in the environment. 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 a ModelArts public resource pool is 50 GB. By default, the container engine space of a 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.
- The ModelArts backend does not support the download of open source installation packages. Install the dependency packages required for training in the custom image.
- Custom images can be used to train models in ModelArts only after they are uploaded to Software Repository for Container (SWR).

6.2 Creating a Custom Training Image Using a Preset Image

Principles

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 create a custom image. In this case, on the training job creation page, select a preset image and choose **Customize** from the framework version drop-down list.

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, as shown below:
  - 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.

Creating a Training Image Using a Preset 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.

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 commands 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 an 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 **ModelArts Unified 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. Create an image. Run the following command in the directory where the Dockerfile is stored to build the container image **training:v1**:
   ```
   docker build . -t training:v1
   ```
6. Upload the new image to SWR.
   a. Log in to the SWR console and select the target region.
   b. Click **Create Organization** in the upper right corner and enter an organization name. In this case, **deep-learning** is used as an example. Replace it in subsequent commands with the actual organization name.
   c. Click **Generate Login Command** in the upper right corner to obtain the login command. Log in to the ECS environment as the **root** user and enter the login command.

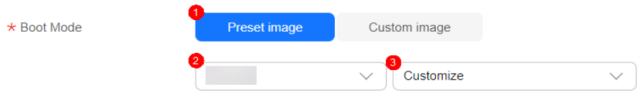
      Figure 6-2 Login command executed on the ECS

   d. Log in to SWR and run the **docker tag** command to add tags to the image to be uploaded. Replace the organization name **deep-learning** in the following command with the actual organization name obtained in **previous operations**.
      ```
      sudo docker tag tf-1.13.2:latest swr.<actual-domain-name>.com/deep-learning/tf-1.13.2:latest
      ```
   e. Run the **docker push** command to upload the image.
      ```
      sudo docker push swr.<actual-domain-name>.com/deep-learning/tf-1.13.2:latest
      ```
   f. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

      SWR URL of the custom image: swr.<*region*>.myhuaweicloud.com/deep-learning/tf-1.13.2:latest
7. Create a training job on ModelArts.
   a. Log in to the **ModelArts console**. In the navigation pane, choose **Model Training** > **Training Jobs**.
   b. Click **Create Training Job**. On the displayed page, configure job information by referring to **Table 6-1**. For details about other parameters, see **Creating a Production Training Job**.

      Table 6-1 Creating a training job

Parameter	Description
Algorithm Type	Select **Custom algorithm**. This parameter is mandatory.
Boot Mode	Mandatory. Select **Preset image** and choose the required framework and engine version. In this case, choose **Customize** for the engine version drop-down list. 

Parameter	Description
Image	Select **the image uploaded to SWR** for container image.
Code Directory	Mandatory. Select the OBS directory where the training code file is stored. • 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. • 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.
Boot File	Mandatory. Select the Python boot script of the training job in the code directory. ModelArts supports only the boot file written in Python. Therefore, the boot file must end with .py.

6.3 Migrating Existing Images to ModelArts

Scenario

An image is available on the local host and needs to be adapted on the cloud for ModelArts model training.

Procedure

1. Modify an existing image by referring to the following Dockerfile so that the image complies with specifications for custom images of the model training.

```
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
```

Note:

a. Add the default user group **ma-group** (**gid = 100**) of the model training for the image.

- If the user group whose **gid** is **100** already exists, the error message "groupadd: GID '100' already exists" may be displayed. You can run **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 **RUN groupadd ma-group -g 100** from the Dockerfile.

b. Add the default user **ma-user** (**uid = 1000**) of the model training for the image.

- If the user whose **uid** is **1000** already exists, the error message "useradd: UID 1000 is not unique" may be displayed. You can run **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 **RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user** from the Dockerfile.

c. Modify the permissions on files in the image to allow **ma-user** whose **uid** is **1000** to read and write the files.

2. After editing the Dockerfile, run the following command to build an image:

```
docker build -f Dockerfile . -t {New image}
```

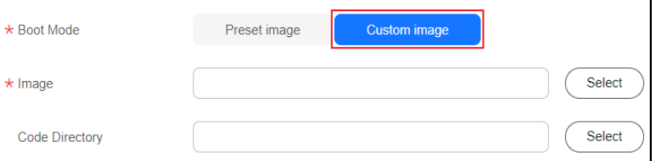
3. Upload the new image to SWR. For details, see **6**.
4. Create a training job on ModelArts.

a. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Model Training** > **Training Management** > **Training Jobs**.

b. Click **Create Training Job**. On the displayed page, configure job information by referring to **Table 6-1**. For details about other parameters, see **Creating a Production Training Job**.

Table 6-2 Creating a training job using a custom image

Parameter	Description
Algorithm Type	Select **Custom algorithm**. This parameter is mandatory.

Parameter	Description
Boot Mode	Mandatory. Select **Custom image**. 
Image	Mandatory. Select **the image uploaded to SWR** for container image.
Code Directory	OBS directory where the training code file is stored. Configure this parameter only if your custom image does not contain training code. • 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. • 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. 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: • Public resource pool: **1000** to **65535** • Dedicated resource pool: **0** to **65535**

Parameter	Description

Boot Command	Mandatory. Command for booting an image.
	When a training job is running, the boot command is automatically executed after the code directory is downloaded.
	• If the training boot script is a .py file, **train.py** for example, the boot command is as follows:
	`python ${MA_JOB_DIR}/demo-code/train.py`
	• If the training boot script is an .sh file, **main.sh** for example, the boot command is as follows:
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	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 the training starts, the system automatically downloads the code directory to this directory. (Optional) The default local code directory is **/home/ma-user/modelarts/user-job-dir**.
Work Directory	During training, the system automatically runs the **cd** command to execute the boot file in this directory.

6.4 Creating a Custom Training Image (PyTorch + Ascend)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is PyTorch, and the resources used for training are powered by Ascend in a dedicated resource pool.

Prerequisites

A Linux virtual or physical server with Docker 18.09.7 or later versions installed is available. The server can access the Internet and function as an image creation node.

Run the **docker pull**, **apt-get update/upgrade**, and **pip install** commands to check whether the node can access an external open-source software repository. If so, the node can access the Internet.

NOTICE

- The preceding servers must be Arm64-powered.
- It is a good practice to install Ubuntu 18.04 on the image creation node.
- In this section, the **/opt** directory is used for the image creation task. Ensure that the available storage of this directory is greater than 30 GB.
- For details about how to install Docker, see **Install Docker Engine on Ubuntu**. Miniconda and TFLite installation packages are provided by third parties. ModelArts is not responsible for their security issues. If you have security requirements, harden the security of these packages, release them as files with the same names, and upload them to the image creation node.

Creating a Custom Image

Step 1 Check the Docker engine version. Run the following command:

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

The command output is as follows:

```
Engine:
 Version: 18.09.0
```

NOTE

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

Step 2 Create a folder named **context**.

```
mkdir -p context
```

Step 3 Obtain the **pip.conf** file. In this example, the pip source provided at Huawei Mirrors is used. Its content is as follows:

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

Step 4 Obtain the APT source file **Ubuntu-Ports-bionic.list**. In this example, the APT source provided at Huawei Mirrors is used. Run the following command to obtain the APT source file:

```
wget -O Ubuntu-Ports-bionic.list --no-check-certificate
https://repo.huaweicloud.com/repository/conf/Ubuntu-Ports-bionic.list
```

Step 5 Download the **Ascend-cann-nnae_7.0.0_linux-aarch64.run**, **torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl**, and **torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl** installation files.

- Download the **Ascend-cann-nnae_7.0.0_linux-aarch64.run** file. Click the following link based on your user type. Search for **CANN7** for version and click the **CANN7.0.0** link. On the displayed page, search for **Ascend-cann-nnae_7.0.0_linux-aarch64.run** and download it.
  - For enterprise users, click **here** for download.
  - For carrier users, click **here** for download.

- Click **here** to download the **torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl** file.
- Click **here** to download the **torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl** file.

NOTE

ModelArts supports only the commercial CANN edition.

Step 6 Download the Miniconda3 installation file.

Click **here** to download the **Miniconda3-py39_24.5.0-0** installation file (for Python 3.9).

NOTE

Download Python of a different version from **Miniconda3 File List**. The MindSpore version must correspond to the Python version.

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

```
context
├── Ascend-cann-nnae_7.0.0_linux-aarch64.run
├── torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── Miniconda3-py39_24.5.0-0-Linux-aarch64.sh
├── torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

Step 8 Write the Dockerfile of the container image.

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.
FROM ubuntu:18.04 AS builder

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

# Install OS dependencies.
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 7.0.0
    gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils net-tools
libblas-dev gfortran libblas3 libopenblas-dev \
    # MindSpore 2.2.0
    libgmp-dev && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    # Grant the write permission of the parent directory of the CANN 7.0.0 installation directory to ma-user.
    chmod o+w /usr/local

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

# Use the PyPI configuration obtained at the open-source image website.
```

```
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:100 pip.conf /home/ma-user/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
COPY --chown=ma-user:100 Miniconda3-py39_24.5.0-0-Linux-aarch64.sh /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-py39_24.5.0-0-Linux-aarch64.sh -b -p /home/ma-user/miniconda3



ENV PATH=$PATH:/home/ma-user/miniconda3/bin

# Install the CANN 7.0.0 Python dependency package.
RUN pip install numpy~=1.19.2 decorator~=4.4.0 sympy~=1.5.1 cffi~=1.12.3 protobuf~=3.13.0 \
    attrs pyyaml pathlib2 scipy requests psutil absl-py

# Install CANN 7.0.0 in /usr/local/Ascend.
COPY --chown=ma-user:100 Ascend-cann-nnae_7.0.0_linux-aarch64.run /tmp
RUN chmod +x /tmp/Ascend-cann-nnae_7.0.0_linux-aarch64.run && \
    echo Y|/tmp/Ascend-cann-nnae_7.0.0_linux-aarch64.run --install --install-path=/usr/local/Ascend

# Install PyTorch 2.1.0.
COPY --chown=ma-user:100 torch-2.1.0-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl /tmp
RUN chmod +x /tmp/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl && \
    pip install /tmp/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl

# Install touch-npu.
COPY --chown=ma-user:100 torch_npu-2.1.0.post7-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl /tmp
RUN chmod +x /tmp/torch_npu-2.1.0.post7-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl && \
    pip install /tmp/torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl

# Create the container image.
FROM ubuntu:18.04

# Install OS dependencies.
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 7.0.RC1
    gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils net-tools
libblas-dev gfortran libblas3 libopenblas-dev \
    # MindSpore 2.2.0
    libgmp-dev && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

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



```
ENV PATH=$PATH:/usr/local/Ascend/nnae/latest/bin:/usr/local/Ascend/nnae/latest/compiler/ccec_compiler/bin:/home/ma-user/miniconda3/bin \
    LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Ascend/driver/lib64:/usr/local/Ascend/driver/lib64/common:/usr/local/Ascend/driver/lib64/driver:/usr/local/Ascend/nnae/latest/lib64:/usr/local/Ascend/nnae/latest/lib64/plugin/opskernel:/usr/local/Ascend/nnae/latest/lib64/plugin/nnengine \
    PYTHONPATH=$PYTHONPATH:/usr/local/Ascend/nnae/latest/python/site-packages:/usr/local/Ascend/nnae/latest/opp/op_impl/built-in/ai_core/tbe \
    ASCEND_AICPU_PATH=$ASCEND_AICPU_PATH:/usr/local/Ascend/nnae/latest \
    ASCEND_OPP_PATH=$ASCEND_OPP_PATH:/usr/local/Ascend/nnae/latest/opp \
    ASCEND_HOME_PATH=$ASCEND_HOME_PATH:/usr/local/Ascend/nnae/latest \
    PYTHONUNBUFFERED=1

# Configure 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**.

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

```
context
├── Ascend-cann-nnae_7.0.0_linux-aarch64.run
├── Dockerfile
├── torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── Miniconda3-py39_24.5.0-0-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

Step 10 Create the container image. Run the following command in the directory where the Dockerfile is stored to create a container image:

```
docker build . -t pytorch:2.1.0-cann7.0.0
```

NOTE

If "connection refused" or "Client.Timeout exceeded" is reported when you access https://registry-1.docker.io/v2/ during image creation, configure the Docker proxy.

vi /etc/docker/daemon.json

Add the following content to the file and save the file:

```
{
"registry-mirrors":[
"https://docker.m.daocloud.io",
"https://docker.jianmuhub.com",
"https://huecker.io",
"https://dockerhub.timeweb.cloud",
"https://dockerhub1.beget.com",
"https://noohub.ru"]
}
```

Run the **systemctl daemon-reload** and **systemctl restart docker** commands in sequence.

Recreate the image.

The following log shows that the image has been created.

```
Successfully tagged pytorch:2.1.0-cann7.0.0
```

----End

Uploading an Image to SWR

1. Log in to the SWR console and select a region. It must share the same region with ModelArts. Otherwise, the image cannot be selected.

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.
3. Click **Generate Login Command** in the upper right corner to obtain the login command. In this example, the temporary login command is copied.
4. Log in to the local environment as user **root** and enter the copied temporary 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 pytorch:2.1.0-cann7.0.0 swr.{region-id}.{domain}/deep-learning/pytorch:2.1.0-cann7.0.0
   ```
   b. 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/pytorch:2.1.0-cann7.0.0
   ```
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.

Creating a Training Job on ModelArts

1. Log in to the **ModelArts console** and check whether access authorization has been configured for your account. If not, configure the authorization by referring to **Configuring Agency Authorization for ModelArts with One Click**. For users who used AKs for authorization, you are advised to clear the existing authorization and use an agency.
2. In the navigation pane, choose **Model Training** > **Training Jobs**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - **Image**: **swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:2.1.0-cann7.0.0**
   - **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**: **Dedicated resource pool**
   - **Resource Type**: Ascend with the required driver and firmware version
   - **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, depending on the service logic and selected resources. You can view log information on the job details page.

6.5 Creating a Custom Training Image (PyTorch + CPU/GPU)

This section describes how to create an image and use it for training on ModelArts. 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.

Scenario

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

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training jobs 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, 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 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 the folders in OBS for storing the sample dataset and training code. **Table 6-3** lists the folders to be created. Replace the bucket name **test-modelarts** 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 6-3 Required OBS folders

Folder	Usage
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**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

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 details about how to install Docker, see **official Docker 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. Run the following command to check the Docker Engine version:

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

   The command output is as follows:

   ```
   ...
   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 open-source image site is used. The content of **pip.conf** is as follows:

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

   NOTE

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

5. Download the **torch*.whl** file.

   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 plus sign (+) must be URL-encoded to **%2B**. When searching for the target file name on the preceding website, replace the plus sign (+) in the original file name with **%2B**.

   Example: **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**

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. 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 Dockerfile of the container image.

   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-runtime-ubuntu18.04 AS builder

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

# Use the PyPI configuration provided by the open-source image site.
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 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 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 container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim and cURL in the open-source image site.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweiexamplecloud.com@g" /etc/apt/
sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweiexamplecloud.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 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 preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
```



```
    PYTHONUNBUFFERED=1

# Configure 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. Ensure 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 shows 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 a region. It must share the same region with ModelArts. Otherwise, the image cannot be selected.
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.
3. Click **Generate Login Command** in the upper right corner to obtain the login command. In this example, the temporary login command is copied.
4. Log in to the local environment as user **root** and enter the copied temporary 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 pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

   b. 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/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 console** and check whether access authorization has been configured for your account. If not, configure the authorization by referring to **Configuring Agency Authorization for ModelArts with One Click**. For users who used AKs for authorization, you are advised to clear the existing authorization and use an agency.

2. In the navigation pane, choose **Model Training** > **Training Jobs**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - 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 **GPU** or **CPU**.
   - **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, depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

   Figure 6-3 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')

```


6.6 Creating a Custom Training Image (MPI + CPU/GPU)

This section describes how to create an image and use it for training on ModelArts. 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.

Scenario

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

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training jobs 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 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 the folders in OBS for storing the sample dataset and training code. **Table 6-4** lists the folders to be created. Replace the bucket name **test-modelarts** and folder name in the example with the 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 6-4 Required OBS folders

Folder	Usage
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 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**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

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
- 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 the Docker installation package. For details about how to install Docker, see **official Docker 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 command output is as follows:


```
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. 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 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-runtime-ubuntu18.04 AS builder

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

# 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
```

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

8. Ensure 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 shows 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 a region. It must share the same region with ModelArts. Otherwise, the image cannot be selected.

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.
3. Click **Generate Login Command** in the upper right corner to obtain the login command. In this example, the temporary login command is copied.
4. Log in to the local environment as user **root** and enter the copied temporary 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 mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. Upload the image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   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** in navigation pane on the left 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 console** and check whether access authorization has been configured for your account. If not, configure the authorization by referring to **Configuring Agency Authorization for ModelArts with One Click**. For users who used AKs for authorization, you are advised to clear the existing authorization and use an agency.
2. In the navigation pane, choose **Model Training** > **Training Jobs**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - 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**
   - **Compute Nodes**: Enter **1** or **2**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mpi/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, depending on the service logic and selected resources. **Figure 6-4** shows the log information after the training job is executed.

Figure 6-4 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 6-5** and **Figure 6-6** show the log information.

Figure 6-5 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 6-6 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/[illegible]e/run_mpi.sh: line 109:    66 Terminated              sleep 365d
```


6.7 Creating a Custom Training Image (Tensorflow + GPU)

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.

Objective: Build and install container images of the following software and use the images and GPUs for training jobs 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, 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 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 the folders in OBS for storing the sample dataset and training code. **Table 6-5** lists the folders to be created. Replace the bucket name **test-modelarts** and folder name in the example with the 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 6-5 Required OBS folders

Folder	Usage
obs://test-modelarts/tensorflow/code/	Stores the training script.
obs://test-modelarts/tensorflow/data/	Stores dataset files.
obs://test-modelarts/tensorflow/log/	Stores 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**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

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

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 the Docker installation package. For details about how to install Docker, see **official Docker 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 command output is as follows:

   ```
   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 at Huawei Mirrors is used. Its content 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**, go to Huawei Mirrors at 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 Dockerfile of the container image.

   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 **Linux Drivers**. In the **Download** tab, set **Version** to **5.4-3.5.8.0-LTS**, **OS Distribution Version** to **Ubuntu 18.04**, **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 a region. It must share the same region with ModelArts. Otherwise, the image cannot be selected.
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.
3. Click **Generate Login Command** in the upper right corner to obtain the login command. In this example, the temporary login command is copied.
4. Log in to the local environment as user **root** and enter the copied temporary 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. 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 console** and check whether access authorization has been configured for your account. If not, configure the authorization by referring to **Configuring Agency Authorization for ModelArts with One Click**. For users who used AKs for authorization, you are advised to clear the existing authorization and use an agency.
2. In the navigation pane, choose **Model Training** > **Training Jobs**.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - 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, depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

Figure 6-7 Run logs of training jobs with GPU specifications

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


7 Creating a Custom Image for Inference

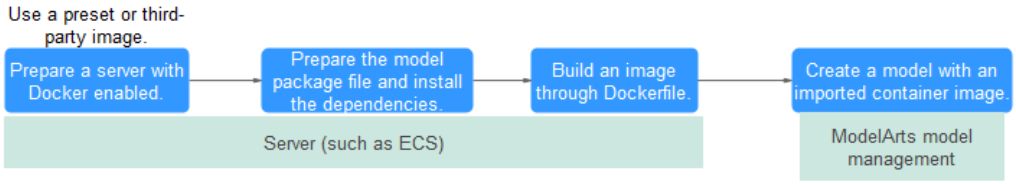
7.1 Creating a Custom Image for a Model

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

Procedure

The preset image does not meet the software environment requirements. You need to import a model package. The new image is larger than 35 GB and needs to be created on a server such as ECS. For details, see **Creating a Custom Image on ECS**.

Figure 7-1 Creating a custom image for a model



Constraints

- No malicious code is allowed.
- The image for creating a model cannot be larger than 50 GB.
- For models in synchronous request mode, if the prediction request latency exceeds 60 seconds, the request will fail, and there is a possibility that the service may be interrupted. Therefore, in this case, create a model in asynchronous mode.

Specifications for Custom Images

- **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://{listen_ip}:8080/ \ -F images=@seven.jpg
  ```

  Figure 7-2 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 APIs**

  If services must not be interrupted during a rolling upgrade, the health check APIs must be configured in **config.json** for ModelArts. The health check APIs return the health status for a service when the service is running properly or an error when the service becomes faulty.

  NOTICE

  - The health check APIs must be configured for a hitless rolling upgrade.
  - If you need to use OBS external storage mounting for custom images in real-time services, create a new directory for OBS data, for example, **/obs-mount/**. Otherwise, the existing files will be overwritten. You can add, view, and modify files in the OBS mount directory. To delete the files, delete them in the OBS parallel file system.

  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 7-1 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 dependencies such as Python, JRE/JDK, and ZIP in the image.

- **(Optional) Maintaining HTTP persistent connections for hitless rolling upgrade**

  If you need to ensure that services are not interrupted during a rolling upgrade, set the HTTP **keep-alive** parameter of the service to 200 seconds. For example, Gunicorn does not support keep-alive by default. To ensure 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) Processing SIGTERM signals and 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 exit, 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  # Pass SIGTERM signals to the Gunicorn process.
        wait $gunicorn_pid          # Wait until the Gunicorn process stops.
    fi
  }

  trap handle_sigterm TERM
  ```

7.2 Creating a Custom Image on ECS

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 models. This section describes how to use a custom image to create a model and deploy it as a real-time service.

The procedure is as follows:

1. **Building an Image Locally**: Create a custom image package locally. For details, see **Creating a Custom Image for a Model**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.

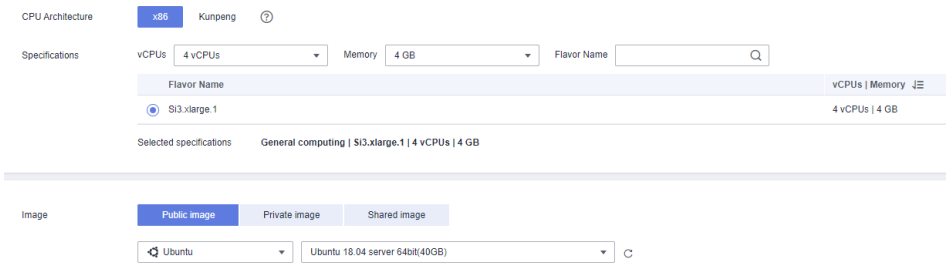
3. **Creating a Model Using a Custom Image**: Import the image to ModelArts AI application management.
4. **Deploying the Model as a Real-Time Service**: Deploy the model as a real-time service.

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 7-3 Creating an ECS using an x86 public image



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 base images. 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 7-4 Starting a custom image

   ```
   :/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 7-5 Testing API functions

   ```
   root@:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/

    called default func !
    {'name': 'Tom'}
   root@:~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
   {
       "response": "Hello, Tom!"
   }root@:~#curl -X GET 127.0.0.1:8080/goodbye

   Goodbye!
   ```

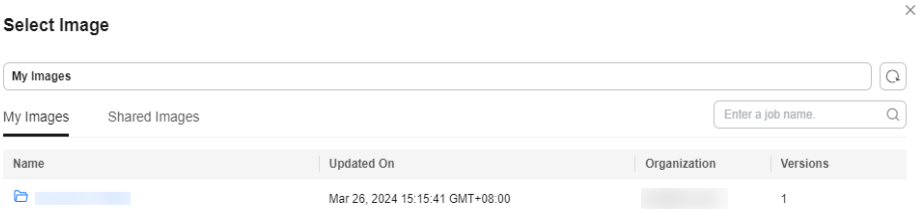

3. Upload the custom image to SWR.
4. After the custom image is uploaded, view the uploaded image on the **My Images** > **Private Images** page of the SWR console.

Creating a Model Using a Custom Image

Import the meta model by referring to **Importing a Meta Model from a Container Image**. Pay special attention to the following parameters:

- **Meta Model Source**: Select **Container image**.
  - **Container Image Path**: Select the created private image.

    Figure 7-6 Created private image

  - **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 model will fail.
- **APIs**: APIs of a custom image. This parameter is optional. Ensure that the APIs 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 Model as a Real-Time Service

1. **Deploy the model as a real-time service.**
2. View the details about the real-time service.
3. Access the real-time service on the **Prediction** tab.

