ModelArts

Image Management

Issue	01
Date	2024-06-11

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

Address:	Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China
Website:	https://www.huawei.com
Email:	support@huawei.com

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Image Management

Overview

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

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

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

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

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

Application Scenarios of Preset Images

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

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

Application Scenarios of Custom Images

- **Using custom images on notebook instances**

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

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

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

Custom Image Services

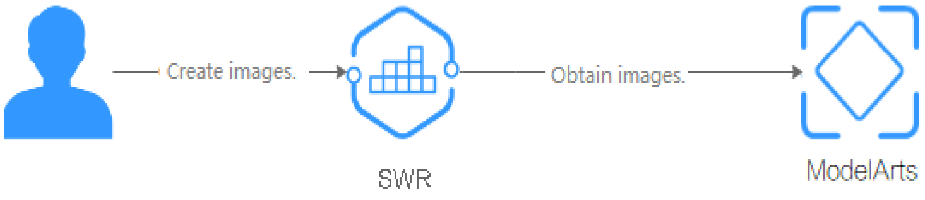
When you use a custom image, the following services may be involved:

- SWR

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

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

  Figure 1-1 Obtaining images

- OBS

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

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

- ECS

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

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

NOTE

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

2 Using Custom Images in Notebook Instances

2.1 Registering an Image in ModelArts

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

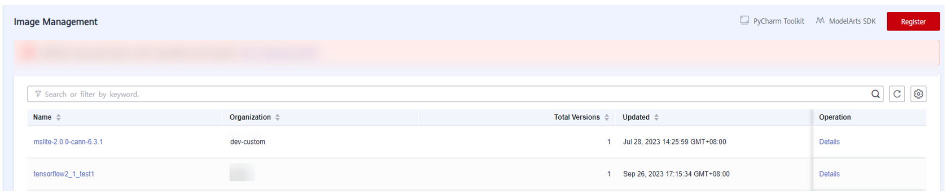
NOTE

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

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

1. Log in to the ModelArts management console and choose **Image Management**. Then, click **Register**.
2. Configure parameters and click **Register**.
   - **SWR Source**: Select a built image as the image source. You can copy the complete SWR address or click to select the target image for registration.
   - **Architecture** and **Type**: Configure them based on the actual framework of the custom image.
3. View the registered image on the **Image Management** page.

Figure 2-1 Image list



[redacted]

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

Creating a Notebook Instance

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

Figure 2-2 Image details page

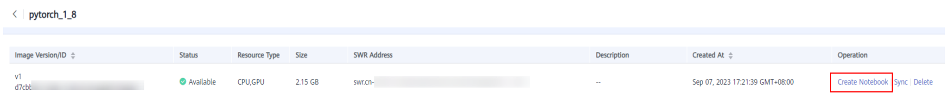

pytorch_1_8

Image Version/ID	Status	Resource Type	Size	SWR Address	Description	Created At	Operation
v1 d7cbt[redacted]	Available	CPU,GPU	2.15 GB	swr.cn-[redacted]	--	Sep 07, 2023 17:21:39 GMT+08:00	Create Notebook Sync Delete

Synchronizing an Image

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

2.2 Saving a Notebook Instance as a Custom Image

2.2.1 Saving a Notebook Environment Image

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

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

NOTE

Images stored in a notebook instance cannot be larger than 25 GB and there cannot be more than 125 image layers. Otherwise, the image cannot be created.

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

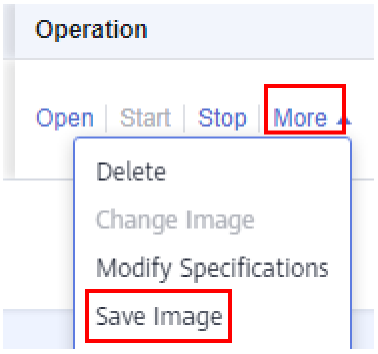
Prerequisites

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

Saving an Image

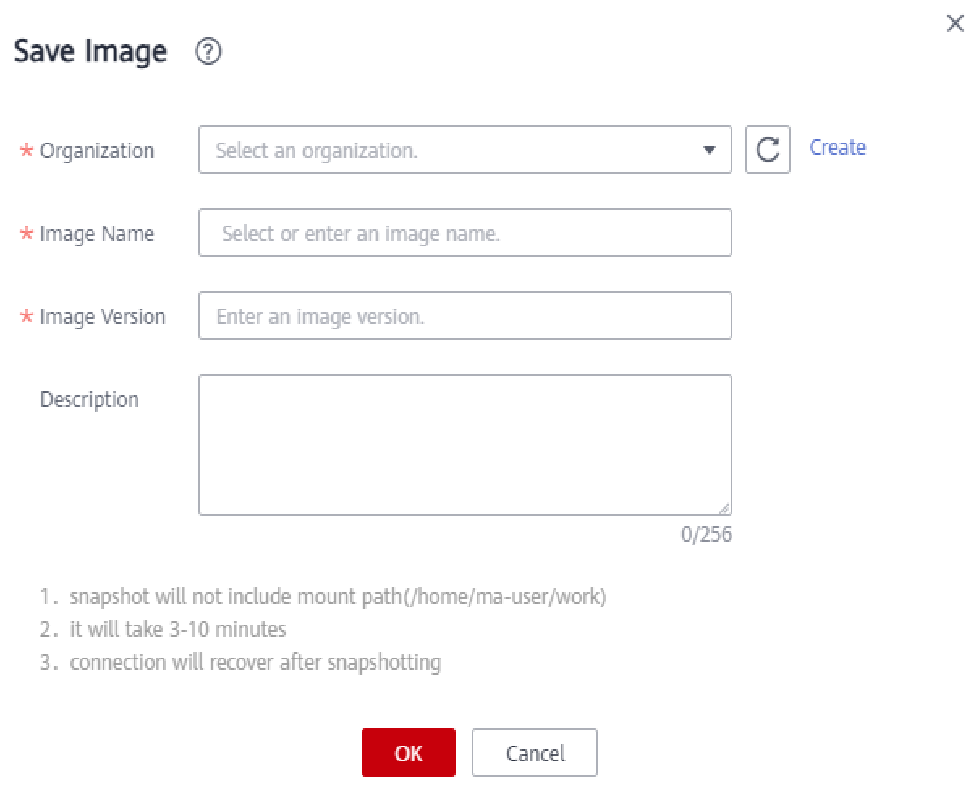
1. Log in to the ModelArts management console and choose **DevEnviron** > **Notebook** in the navigation pane on the left to switch to notebook of the new version.
2. In the notebook instance list, select the target notebook instance and choose **Save Image** from the **More** drop-down list in the **Operation**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

Figure 2-3 Save Image



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

Figure 2-4 Configuring image parameters



* Organization	Select an organization.
* Image Name	Select or enter an image name.
* Image Version	Enter an image version.
Description	0/256

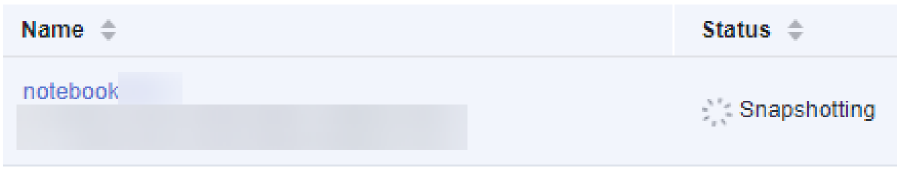
1. snapshot will not include mount path(/home/ma-user/work)
2. it will take 3-10 minutes
3. connection will recover after snapshotting

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

Users in an organization can share all images in the organization.

4. The image will be saved as a snapshot, and it will take about 5 minutes. During this period of time, do not perform any operations on the instance.

Figure 2-5 Saving as a snapshot



Name	Status
notebook	Snapshotting

NOTICE

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

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

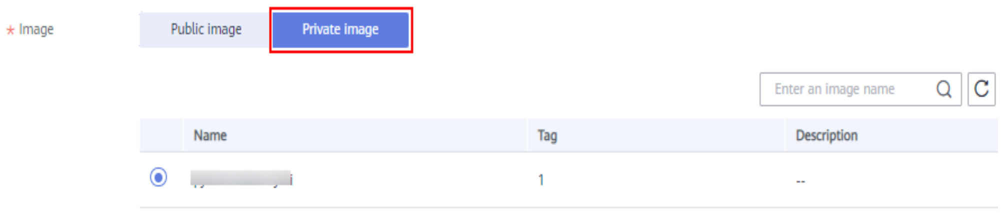
2.2.2 Using a Custom Image to Create a Notebook Instance

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

You can use either of the following methods:

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

Figure 2-6 Selecting a custom image to create a notebook instance



	Name	Tag	Description
●		1	--

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

2.3 Creating and Using a Custom Image on a Notebook Instance

2.3.1 Application Scenarios and Process

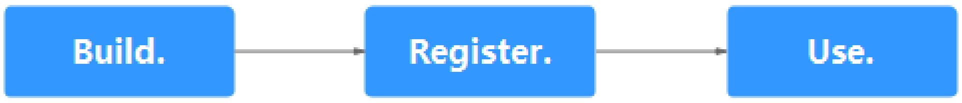
If preset images cannot meet your service requirements, you can create container images based on the preset images for development and training.

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

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

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

Figure 2-7 Creating an image

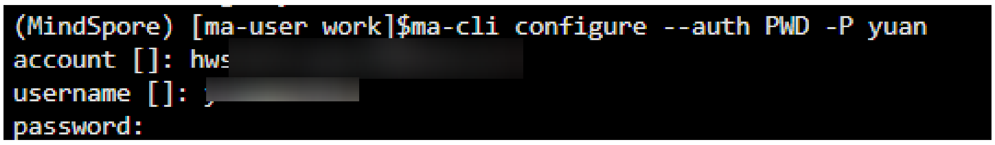


2.3.2 Step 1 Creating a Custom Image

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

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

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



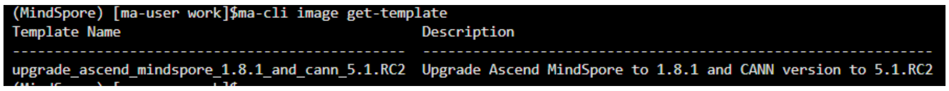
```
(MindSpore) [ma-user work]$ma-cli configure --auth PWD -P yuan
account []: hws[redacted]
username []: [redacted]
password:
```

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

Step 3 Create an image.

1. Obtain the SWR address of the base image.
2. Load an image creation template.

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



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

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

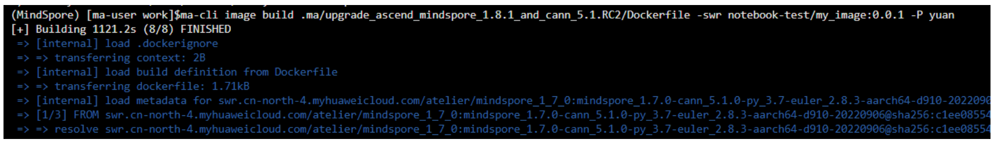
3. Modify a Dockerfile.

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

4. Build an image.

   Run the **ma-cli image build** command to build an image with the Dockerfile. For more information, see .

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



```
(MindSpore) [ma-user work]$ma-cli image build .ma/upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2/Dockerfile -swr notebook-test/my_image:0.0.1 -P yuan
[+] Building 1121.2s (8/8) FINISHED
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.71kB
 => [internal] load metadata for swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-2022090
 => [1/3] FROM swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906@sha256:c1ee08554
 => => resolve swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906@sha256:c1ee08554
```

----End

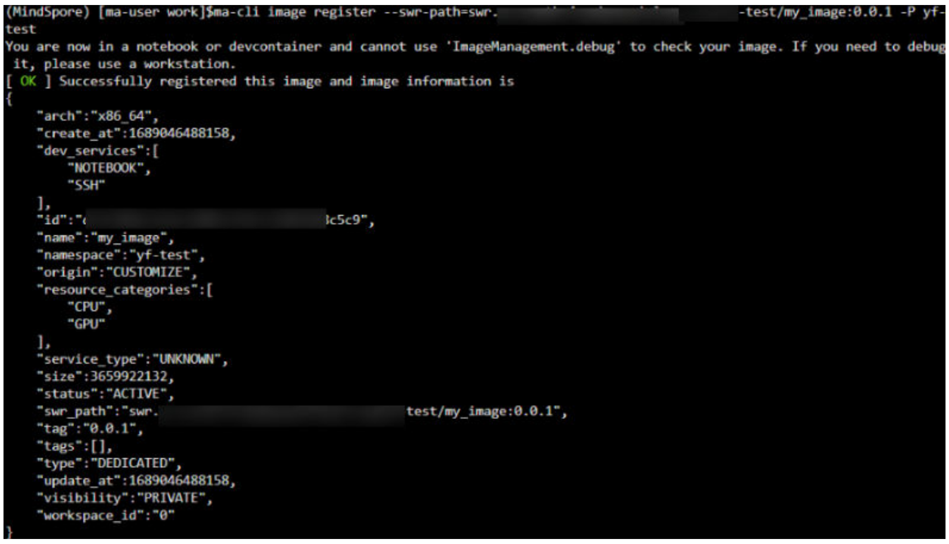
2.3.3 Step 2 Registering a New Image

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

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

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

Figure 2-8 Registered image



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

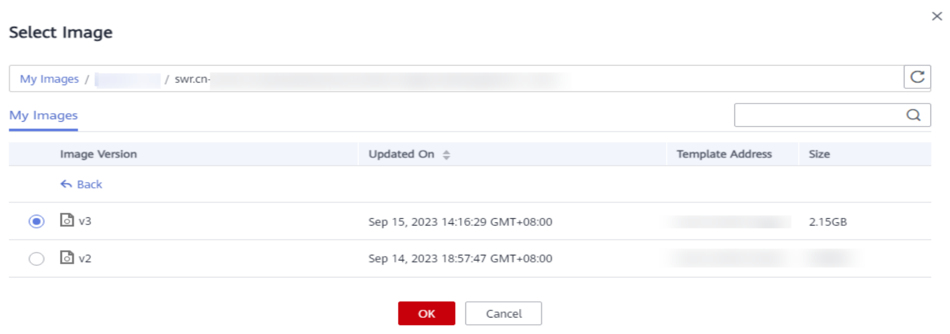
- **Method 2**: Register the image on the ModelArts management console.

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

  Click **Register**. Paste the complete SWR address, or click 🗁 to select a private image from SWR for registration, as shown in **Figure 2-9**.

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

Figure 2-9 Selecting an image


Select Image

My Images / / swr.cn-

My Images

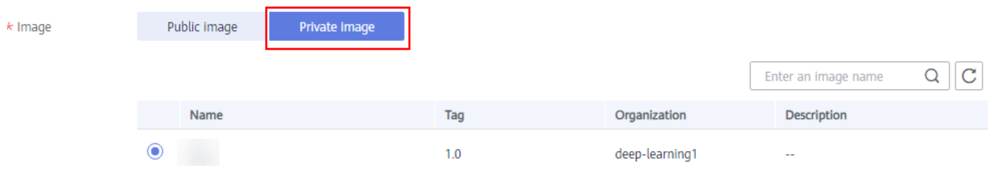
	Image Version	Updated On	Template Address	Size
	Back			
●	v3	Sep 15, 2023 14:16:29 GMT+08:00		2.15GB
○	v2	Sep 14, 2023 18:57:47 GMT+08:00		

2.3.4 Step 3 Using a New Image to Create a Development Environment

Procedure

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

Figure 2-10 Creating a development environment



	Name	Tag	Organization	Description
●		1.0	deep-learning1	--

3 Using a Custom Image to Train Models (New-Version Training)

3.1 Overview

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

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

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

- Using a preset image with customization

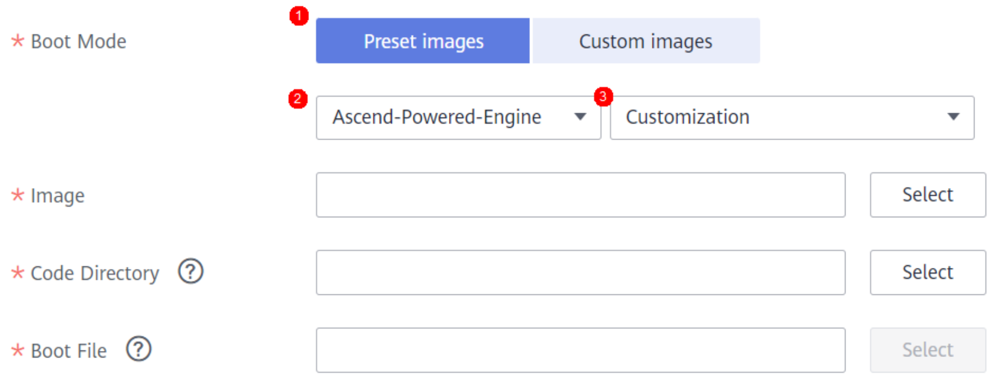
  If you need to modify or add some software dependencies based on the preset image, you can customize the preset image. In this case, select a preset image and choose **Customize** from the version drop-down list.
- Using a custom image

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

Using a Preset Image with Customization

The only difference between this method and creating a training job totally based on a preset image is that you must select an image. You must create a custom image based on a preset image. For details about how to customize a preset image, see **Using a Base Image to Create a Training Image**.

Figure 3-1 Creating an algorithm using a preset image with customization

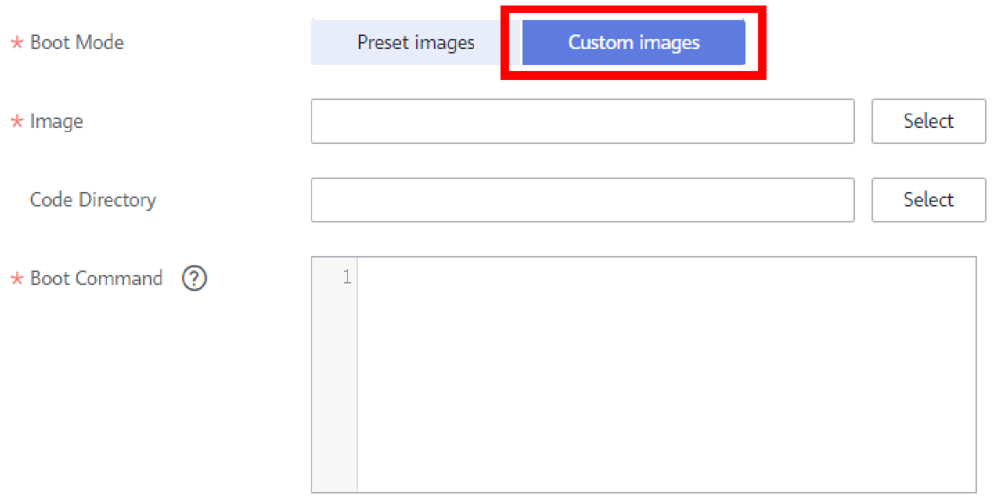


The process of this method is the same as that of creating a training job based on a preset image. For example:

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

Using a Custom Image

Figure 3-2 Creating an algorithm using a custom image



For details about how to use custom images supported by new-version training, see **Specifications for Custom Images for Training Jobs**.

3.2 Preparing a Training Image

3.2.1 Specifications for Custom Images for Training Jobs

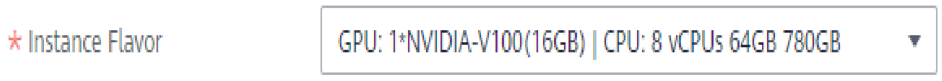
When you use a locally developed model and training script to create a custom image, ensure that the custom image complies with the specifications defined by ModelArts.

NOTE

In both new-version and old-version training management, custom images can be used to create training jobs. This document describes training management of the new version. The old version will be discontinued soon. You are advised to use the new version.

Specifications

- The size of a custom image cannot exceed 30 GB. It is recommended that the size be less than or equal to 15 GB. An oversized image affects the startup of a training job.
- 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.



- ModelArts does not support the download of open source installation packages. Install the dependency packages required by the training job in the custom image.

3.2.2 Migrating an Image to ModelArts Training

To migrate an image to the new-version training management, perform the following operations:

1. Add the default user group **ma-group** (GID = 100) of the new-version training management for the image.

   NOTE

   If the user group whose GID is 100 already exists, the error message "groupadd: GID '100' already exists" may be displayed. You can use the command **cat /etc/group | grep 100** to check whether the user group whose GID is 100 exists.

   If the user group whose GID is 100 already exists, skip this step and delete the command **RUN groupadd ma-group -g 100** from the Dockerfile.

2. Add the default user **ma-user** (UID = 1000) of the new-version training management for the image.

   NOTE

   If the user whose UID is 1000 already exists, the error message "useradd: UID 1000 is not unique" may be displayed. You can use the command **cat /etc/passwd | grep 1000** to check whether the user whose UID is 1000 exists.

   If the user whose UID is 1000 already exists, skip this step and delete the command **RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user** from the Dockerfile.

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

You can modify an image by referring to the following Dockerfile so that the image complies with specifications for custom images of the new-version training management.

```
FROM {An existing image}

USER root

# If the user group whose GID is 100 already exists, delete the groupadd command.
RUN groupadd ma-group -g 100
# If the user whose UID is 1000 already exists, delete the useradd command.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Modify the permissions on image files so that user ma-user whose UID is 1000 can read and write the files.
RUN chown -R ma-user:100 {Path to the Python software package}

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

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

After editing the Dockerfile, run the following command to build a new image:

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

Upload the new image to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**

3.2.3 Using a Base Image to Create a Training Image

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

Procedure

Perform the following operations to create an image using a training base image:

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

   The following uses Linux x86_64 as an example to describe how to obtain the Docker installation package. Run the following command to install Docker:

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

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

   ```
   mkdir -p context
   ```

3. Obtain the **pip.conf** file.

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

4. Create a new image based on a training base image provided by ModelArts. Save the edited Dockerfile in the **context** folder.

   ```
   FROM {Path to the training base image provided by ModelArts}

   # Configure pip.
   RUN mkdir -p /home/ma-user/.pip/
   COPY --chown=ma-user:ma-group pip.conf /home/ma-user/.pip/pip.conf

   # Configure the preset environment variables of the container image.
   # Add the Python interpreter path to the PATH environment variable.
   # Set PYTHONUNBUFFERED to 1 to prevent log loss.
   ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
       PYTHONUNBUFFERED=1

   RUN /home/ma-user/anaconda/bin/pip install --no-cache-dir numpy
   ```

5. Run the following command in the directory where the Dockerfile is stored to create a container image, for example, **training:v1**:

   ```
   docker build . -t training:v1
   ```

6. Upload the new image to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**.

7. Use the custom image to create a training job on ModelArts. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

3.3 Creating an Algorithm Using a Custom Image

Your locally developed algorithms or algorithms developed using other tools can be uploaded to ModelArts for unified management.

Entries for Creating an Algorithm

You can create an algorithm using a custom image on ModelArts in either of the following ways:

- Entry 1: On the ModelArts console, choose **Algorithm Management** > **My algorithms**. Then, create an algorithm and use it in training jobs or publish it to .
- Entry 2: On the ModelArts console, choose **Training Management** > **Training Jobs**, and click **Create Training Job** to create a custom algorithm and submit a training job. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

Parameters for creating an algorithm

Figure 3-3 Creating an algorithm using a custom image

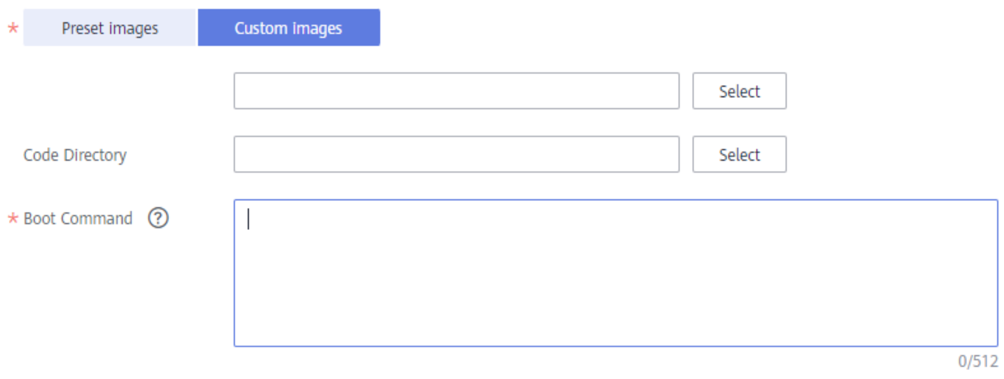


Table 3-1 Parameters for creating an algorithm

Parameter	Description
Boot Mode	Select **Custom images**. This parameter is mandatory.
Image Path	URL of an SWR image. This parameter is mandatory. ● Private images or shared images: Click **Select** on the right to select an SWR image. Ensure that the image has been uploaded to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**. ● Public images: You can also manually enter the image path in the format of "<Organization to which your image belongs>/<Image name>" on SWR. Do not contain the domain name (swr.<region>.example.com) in the path because the system will automatically add the domain name to the path. For example: `modelarts-job-dev-image/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-euleros_2.10.1-x86_64-8.1.1`

Parameter	Description
Code Directory	OBS path for storing the training code. This parameter is optional. Take OBS path **obs://obs-bucket/training-test/demo-code** as an example. The content in the OBS path will be automatically downloaded to **${MA_JOB_DIR}/demo-code** in the training container, and **demo-code** (customizable) is the last-level directory of the OBS path.
Boot Command	Command for booting an image. This parameter is mandatory. The boot command will be automatically executed after the code directory is downloaded. ● If the training boot script is a .py file, **train.py** for example, the boot command can be **python ${MA_JOB_DIR}/demo-code/train.py**. ● If the training boot script is an .sh file, **main.sh** for example, the boot command can be **bash ${MA_JOB_DIR}/demo-code/main.sh**. Semicolons (;) and ampersands (&&) can be used to combine multiple boot commands, but line breaks are not supported. **demo-code** (customizable) in the boot command is the last-level directory of the OBS path.

Configuring Pipelines

A preset image-based algorithm obtains data from an OBS bucket or dataset for model training. The training output is stored in an OBS bucket. The input and output parameters in your algorithm code must be parsed to enable data exchange between ModelArts and OBS. For details about how to develop code for training on ModelArts, see **Developing a Custom Script**.

When you use a preset image to create an algorithm, configure the input and output pipelines.

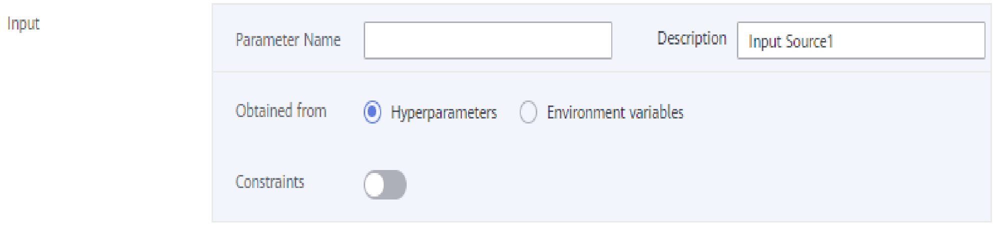
- Input configurations

Table 3-2 Input configurations

Parameter	Description
Parameter Name	If you use **argparse** in the algorithm code to parse **data_url** into the data input, set the data input parameter to **data_url** when creating the algorithm. Set the name based on the data input parameter in your algorithm code. The code path parameter must be the same as the data input parameter parsed in your algorithm code. Otherwise, the algorithm code cannot obtain the input data.

Parameter	Description
Description	Customizable description of the input parameter,
Obtained from	Source of the input parameter. You can select **Hyperparameters** (default) or **Environment variables**.
Constraints	Whether data is obtained from a storage path or ModelArts dataset. If you select the ModelArts dataset as the data source, the following constraints are added: ● **Labeling Type**: For details, see **Creating a Labeling Job**. ● **Data Format**, which can be **Default**, **CarbonData**, or both. **Default** indicates the manifest format. ● **Data Segmentation**: available only for image classification, object detection, text classification, and sound classification datasets. Possible values are **Segmented dataset**, **Dataset not segmented**, and **Unlimited**. For details, see **Publishing a Data Version**.
Yes	Allow multiple data input sources based on the algorithm

Figure 3-4 Input configurations


Input

Parameter Name:
Description: Input Source1
Obtained from: [✓ Hyperparameters] Environment variables
Constraints: off

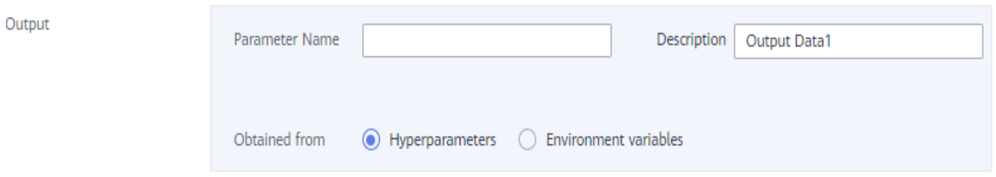
- Output configurations

Table 3-3 Output configurations

Parameter	Description
Parameter Name	If you use **argparse** in the algorithm code to parse **train_url** into the data output, set the data output parameter to **train_url** when creating the algorithm. Set the name based on the data output parameter in your algorithm code. The code path parameter must be the same as the data output parameter parsed in your algorithm code. Otherwise, the algorithm code cannot obtain the output path.
Description	Customizable description of the output parameter,

Parameter	Description
Obtained from	Source of the output parameter. You can select **Hyperparameters** (default) or **Environment variables**.
Yes	Allow multiple data output paths based on the algorithm

Figure 3-5 Output configurations



Output

Parameter Name

Description: Output Data1

Obtained from: [✓ Hyperparameters] Environment variables

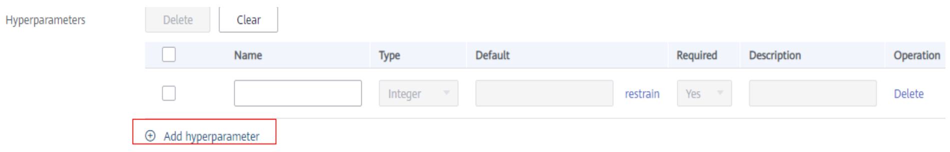
Defining Hyperparameters

When you use a preset image to create an algorithm, ModelArts allows you to customize hyperparameters so you can view or modify them anytime. After the hyperparameters are defined, they are displayed in the startup command and transferred to your boot file as CLI parameters.

1. Import hyperparameters.

   You can click **Add hyperparameter** to manually add hyperparameters.

   Figure 3-6 Adding hyperparameters

2. Edit hyperparameters. For details, see **Table 3-4**.

Table 3-4 Hyperparameters

Parameter	Description
Name	Hyperparameter name Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Type	Type of the hyperparameter, which can be **String**, **Integer**, **Float**, or **Boolean**
Default	Default value of the hyperparameter, which is used for training jobs by default

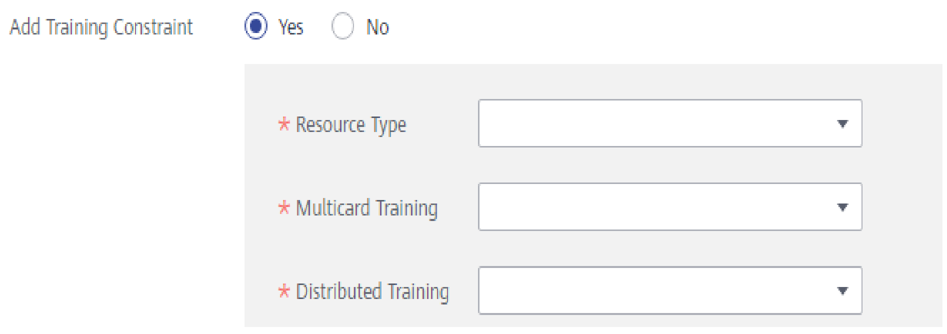
Parameter	Description
Constraints	Click **restrain**. Then, set the range of the default value or enumerated value in the dialog box displayed.
Required	Whether the parameter is mandatory. The value can be **Yes** or **No**. If you select **No**, you can delete the hyperparameter on the training job creation page when using this algorithm to create a training job. If you select **Yes**, the hyperparameter cannot be deleted.
Description	Description of the hyperparameter Only letters, digits, spaces, hyphens (-), underscores (_), commas (,), and periods (.) are allowed.

Adding Training Constraints

You can add training constraints of the algorithm based on your needs.

- **Resource Type**: The options are **CPU** and **GPU**. You can select multiple options.
- **Multicard Training**: Select **Supported** or **Not supported**.
- **Distributed Training**: Select **Supported** or **Not supported**.

Figure 3-7 Training constraints


Add Training Constraint [✓ Yes] No

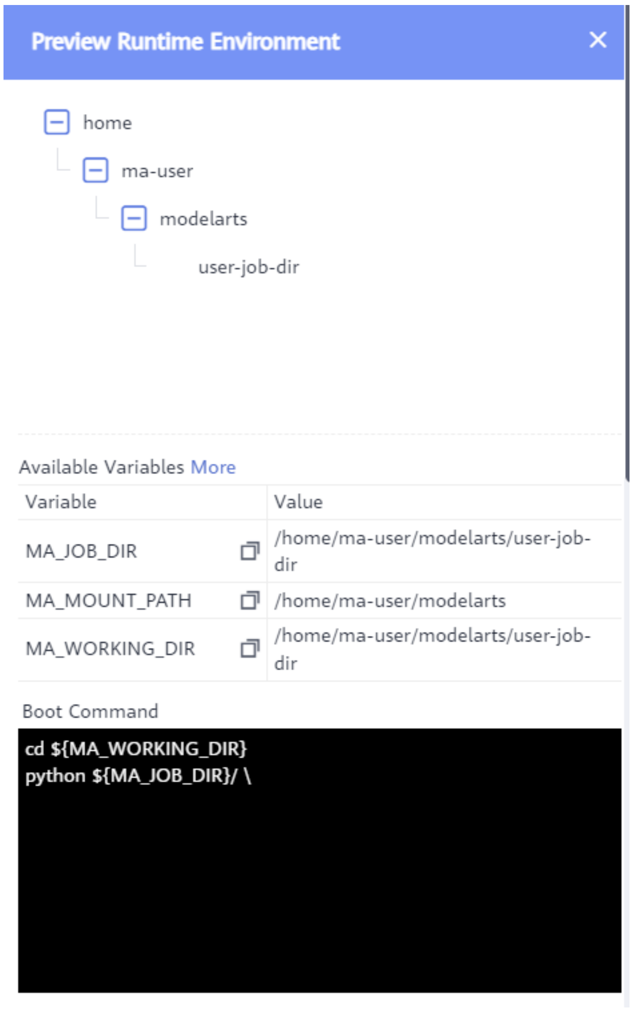
* Resource Type	
* Multicard Training	
* Distributed Training	

Runtime Environment Preview

When creating an algorithm, click the arrow on

in the lower right corner of the page to know the path of the code directory, boot file, and input and output data in the training container.

Figure 3-8 Runtime environment preview



Available Variables More

Variable	Value
MA_JOB_DIR	/home/ma-user/modelarts/user-job-dir
MA_MOUNT_PATH	/home/ma-user/modelarts
MA_WORKING_DIR	/home/ma-user/modelarts/user-job-dir

Boot Command

```
cd ${MA_WORKING_DIR}
python ${MA_JOB_DIR}/ \
```

Follow-Up Procedure

After an algorithm is created, use it to create a training job. For details, see **Using a Custom Image to Create a CPU- or GPU-based Training Job**.

3.4 Using a Custom Image to Create a CPU- or GPU-based Training Job

Model training is an iterative optimization process. Through unified training management, you can flexibly select algorithms, data, and hyperparameters to obtain the optimal input configuration and model. After comparing metrics between job versions, you can determine the most satisfactory training job.

Prerequisites

- The data to be trained has been uploaded to an OBS directory.
- At least one empty folder for storing the training output has been created in OBS.

- A custom image has been created based on ModelArts specifications. For details about the custom image specifications, see **Specifications for Custom Images for Training Jobs**.
- The custom image has been uploaded to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**.

Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management** > **Training Jobs**.
2. Click **Create Training Job** and set parameters. **Table 3-5** lists the parameters.

Table 3-5 Job parameters

Parameter	Description
Created By	Select **Custom algorithms**. This parameter is mandatory. If you have created an algorithm based on a custom image in **Algorithm Management**, choose the created algorithm from **My algorithms**.
Boot Mode	Select **Custom images**. This parameter is mandatory.
Image Path	URL of an SWR image. This parameter is mandatory. ● Private images or shared images: Click **Select** on the right to select an SWR image. Ensure that the image has been uploaded to SWR. ● Public images: You can also manually enter the image path in the format of "<Organization to which your image belongs>/<Image name>" on SWR. Do not contain the domain name (swr.<region>.example.com) in the path because the system will automatically add the domain name to the path. For example: `modelarts-job-dev-image/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-euleros_2.10.1-x86_64-8.1.1`
Code Directory	OBS path for storing the training code. This parameter is optional. Take OBS path **obs://obs-bucket/training-test/demo-code** as an example. The training code in this path will be automatically downloaded to **${MA_JOB_DIR}/demo-code** in the training container, where **demo-code** is the last-level directory of the OBS path and can be customized.

Parameter	Description
Boot Command	Command for booting an image. This parameter is mandatory. The boot command will be automatically executed after the code directory is downloaded. ● If the training startup script is a .py file, **train.py** for example, the boot command can be **python ${MA_JOB_DIR}/demo-code/train.py**. ● If the training startup script is a .sh file, **main.sh** for example, the boot command can be **bash ${MA_JOB_DIR}/demo-code/main.sh**. In the preceding examples, **demo-code** is the last-level OBS directory for storing code and can be customized.
Local Code Directory	You can specify the local directory of a training container. When a training job starts, the system automatically downloads the code directory to this directory. The default local code directory is **/home/ma-user/modelarts/user-job-dir**. This parameter is optional.
Work Directory	Directory where the boot file in the training container is located. When a training job starts, the system automatically runs the **cd** command to change the work directory to the specified directory.

Training Input - Parameter Name	The recommended value is **data_url**, which must be the same as the parameter for parsing the input data in the training code. You can set multiple training input parameters. The name of each training input parameter must be unique, for example, **car_data_url**, **dog_data_url**, and **cat_data_url**.
	For example, if you use **argparse** in the training code to parse **data_url** into the data input, set the parameter name of the training input to **data_url**.
	`import argparse`
	`# Create a parsing task.`
	`parser = argparse.ArgumentParser(description="train mnist",`
	`formatter_class=argparse.ArgumentDefaultsHelpFormatter)`
	`# Add parameters.`
	`parser.add_argument('--train_url', type=str, help='the path model saved')`
	`parser.add_argument('--data_url', type=str, help='the training data')`
	`# Parse the parameters.`
	`args, unknown = parser.parse_known_args()`

Parameter	Description
Training Input - Data Path	Select **Dataset** or **Data path** as the training input. If you select **Data path**, set an OBS path as the training input. When the training starts, data in the specified path will be automatically downloaded to the training container. Take OBS path **obs://obs-bucket/training-test/data** as an example. The data will be automatically downloaded to **${MA_MOUNT_PATH}/inputs/${data_url}_N** of the training container. The value of **N** is the number of training input parameters minus 1. For example: ● If there is only one training input parameter **data_url**, the data will be automatically downloaded to **${MA_MOUNT_PATH}/inputs/data_url_0/** of the training container. ● If there are multiple training input parameters **car_data_url**, **dog_data_url**, and **cat_data_url**, the training data will be automatically downloaded to **${MA_MOUNT_PATH}/inputs/car_data_url_0/**, **${MA_MOUNT_PATH}/inputs/dog_data_url_1/**, and **${MA_MOUNT_PATH}/inputs/cat_data_url_2/** of the container, respectively.
Training Output - Parameter Name	The recommended value is **train_url**, which must be the same as the parameter for parsing the output data in the training code. You can set multiple training output parameters. The name of each training output parameter must be unique.
Training Output - Data Path	Select an OBS path as the training output. To minimize errors, select an empty directory. The training result file in the training container **${MA_MOUNT_PATH}/outputs/${train_url}_N/** will be automatically uploaded to **obs://obs-bucket/training-test/output**. The value of **N** is the number of training output parameters minus 1. For example: ● If there is only one training output parameter **train_url**, the container directory of the training output is **${MA_MOUNT_PATH}/outputs/data_url_0/**. ● If there are multiple training output parameters, for example, **car_train_url**, **dog_train_url**, and **cat_train_url**, the container directories of the training output are **${MA_MOUNT_PATH}/outputs/car_train_url_0/**, **${MA_MOUNT_PATH}/outputs/dog_train_url_1/**, and **${MA_MOUNT_PATH}/outputs/cat_train_url_2/**, respectively.

Parameter	Description

Training Output - Obtained from	The following uses the training output **train_url** as an example.
	Obtain the training output from hyperparameters by using the following code:
	`import argparse`
	`parser = argparse.ArgumentParser()`
	`parser.add_argument('--train_url')`
	`args, unknown = parser.parse_known_args()`
	`train_url = args.train_url`
	Obtain the training output from environment variables by using the following code:
	`import os`
	`train_url = os.getenv("train_url", "")`

Training Output - Predownload	If you set **Predownload** to **Yes**, the system automatically downloads the files in the training output data path to the local directory of the training container when the training job is started. . Select **Yes** for **resumable training and incremental training**.
Hyperparameters	Used for training tuning. This parameter is optional.
Environment Variable	After the container is started, the system loads the default environment variables and the environment variables customized here. **Table 3-6** lists the default environment variables.
Auto Restart	After this function is enabled, you can set the number of restart times for a training failure. This parameter is optional.

Table 3-6 Default environment variables

Environment Variable	Description
MA_JOB_DIR	Parent directory of the code directory.
MA_MOUNT_PATH	Parent directory of the training input and output directories.
VC_TASK_INDEX	Container index, starting from **0**. This parameter is meaningless for single-node training. In multi-node training jobs, you can use this parameter to determine the algorithm logic of the container.

Environment Variable	Description
VC_WORKER_HOSTS	Node communication domain names. Multiple node domain names are separated by commas (,). For example: ● Single node: **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}** ● Two nodes: **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}**,**${MA_VJ_NAME}-${MA_TASK_NAME}-1.${MA_VJ_NAME}**
MA_NUM_HOSTS	Number of compute nodes, which is automatically obtained from **Compute Nodes**.
MA_NUM_GPUS	Number of GPUs on a node
${MA_VJ_NAME}-${MA_TASK_NAME}-N.${MA_VJ_NAME}	Communication domain name of a node. For example, the communication domain name of node 0 is **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}**. **N** indicates the number of compute nodes. For example, if the number of compute nodes is **4**, the environment variables are as follows: **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-1.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-2.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-3.${MA_VJ_NAME}**

3. Select an instance flavor. The value range of the training parameters is consistent with the constraints of existing custom images.

Table 3-7 Resource parameters

Parameter	Description
Resource Pool	Select a resource pool for the job. Public and dedicated resource pools are available for you to select. If you select a dedicated resource pool, you can view details about the pool. If the number of available cards of this pool is insufficient, jobs may need to be queued. In this case, use another resource pool or reduce the number of cards required.
Resource Type	Select CPU or GPU as needed. Set this parameter based on the resource type specified in your training code.

Parameter	Description
Instance Flavor	Select a resource flavor based on the resource type. If the type of resources to be used has been specified in your training code, only the options that comply with the constraints of the selected algorithm are available for you to choose. For example, if **GPU** is selected in the training code but you select **CPU** here, the training may fail. During training, ModelArts will mount NVME SSDs to the **/cache** directory. You can use this directory to store temporary files. The data disk size varies depending on the resource type. To prevent insufficient memory during training, click **Check Input Size** and check the disk size of selected instance flavor.
Compute Nodes	Set the number of compute nodes. The default value is **1**.
Job Priority	When using a new-version dedicated resource pool, you can set the priority of a training job. The value ranges from 1 to 3. The default priority is **1**, and the highest priority is **3**. You can change the priority of a pending job.
SFS Turbo	When using a dedicated resource pool, the training job can be mounted with multiple cloud storage disks (NAS). A disk can be mounted only once and to only one mounting path. Each mounting path must be unique. A maximum of 8 disks can be mounted to a training job.
Persistent Log Saving	If you select CPU or GPU flavors, **Persistent Log Saving** is available for you to set. This function is disabled by default. ModelArts automatically stores the logs for 30 days. You can download all logs on the job details page. After enabling this function, you can store training logs in a specified OBS directory. Set **Job Log Path** and select an empty OBS directory to store the log files generated during training and ensure you have the reading and writing permissions of the directory.
Job Log Path	If you select Ascend resources, select an empty OBS path for storing training logs. Ensure that you have read and write permissions to the selected OBS directory.

Parameter	Description
Event Notification	Whether to subscribe to event notifications. After this function is enabled, you will be notified of specific events, such as job status changes or suspected suspensions, via an SMS or email. If you enable this function, set the following parameters: ● **Topic**: topic of event notifications. You can create a topic on the SMN console. ● **Event**: type of events you want to subscribe to. Options: **JobStarted**, **JobCompleted**, **JobFailed**, **JobTerminated**, and **JobHanged**. **NOTE** ● After you create a topic on the SMN console, add a subscription to the topic, and confirm the subscription. Then, you will be notified of events. ● Currently, only training jobs using GPUs support **JobHanged** events.
Auto Stop	● After this parameter is enabled and the auto stop time is set, a training job automatically stops at the specified time. ● If this function is disabled, a training job will continue to run. ● The options are **1hour**, **2hours**, **4hours**, **6hours**, and **Customization** (1 hour to 72 hours).

4. Click **Submit** to create the training job.

   It takes a period of time to create a training job.

   To view the real-time status of a training job, go to the training job list and click the name of the training job. On the training job details page that is displayed, view the basic information of the training job. For details, see **Viewing Training Job Details**.

4 Using a Custom Image to Create AI applications for Inference Deployment

4.1 Custom Image Specifications for Creating AI Applications

When building a custom image using a locally developed model, ensure that the image complies with ModelArts specifications.

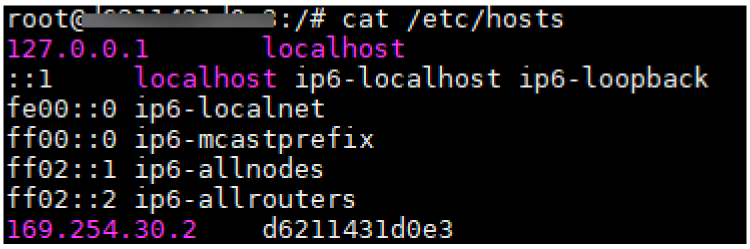
- No malicious code is allowed.
- The size of a custom image cannot exceed 30 GB.
- **External APIs**

  Set the external service API for a custom image. The inference API must be the same as the URL defined by **apis** in **config.json**. Then, the external service API can be directly accessed when the image is started. The following is an example of accessing an MNIST image. The image contains a model trained using an MNIST dataset and can identify handwritten digits. **listen_ip** indicates the container IP address. You can start a custom image to obtain the container IP address from the container.

  – Sample request

    ```
    curl -X POST \ http://{Listening IP address}:8080/ \ -F images=@seven.jpg
    ```

    Figure 4-1 Example of obtaining **listen_ip**

    ```
    root@[redacted]:/# cat /etc/hosts
    127.0.0.1       localhost
    ::1     localhost ip6-localhost ip6-loopback
    fe00::0 ip6-localnet
    ff00::0 ip6-mcastprefix
    ff02::1 ip6-allnodes
    ff02::2 ip6-allrouters
    169.254.30.2    d6211431d0e3
    ```

  – Sample response

    ```
    {"mnist_result": 7}
    ```

- **(Optional) Health check API**

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

NOTICE

The health check API must be configured for a hitless rolling upgrade.

The following shows a sample health check API:

- URI

  ```
  GET /health
  ```

- Sample request: curl -X GET \ http://*{Listening IP address}*:8080/health
- Sample response

  ```
  {"health": "true"}
  ```

- Status code

  Table 4-1 Status code

Status Code	Message	Description
200	OK	Request sent

- **Log file output**

  Configure standard output so that logs can be properly displayed.

- **Image boot file**

  To deploy a batch service, set the boot file of an image to **/home/run.sh** and use CMD to set the default boot path. The following is a sample Dockerfile:

  CMD ["sh", "/home/run.sh"]

- **Image dependencies**

  To deploy a batch service, install dependency packages such as Python, JRE/JDK, and ZIP in the image.

- **(Optional) Hitless rolling upgrade**

  To ensure that services are not interrupted during a rolling upgrade, set HTTP **keep-alive** to **200**. For example, Gunicorn does not support keep-alive by default. To ensure a hitless rolling upgrade, install Gevent and configure **--keep-alive 200 -k gevent** in the image. The parameter settings vary depending on the service framework. Set the parameters as required.

- **(Optional) Gracefully exiting a container**

  To ensure that services are not interrupted during a rolling upgrade, the system must capture SIGTERM signals in the container and wait for 60s before gracefully exiting the container. If the duration is less than 60s before the graceful exiting, services may be interrupted during the rolling upgrade. To ensure uninterrupted service running, the system exits the container after the system receives SIGTERM signals and processes all received requests. The whole duration is not longer than 90s. The following shows example **run.sh**:

  ```
  #!/bin/bash
  gunicorn_pid=""
  ```

```
handle_sigterm() {
  echo "Received SIGTERM, send SIGTERM to $gunicorn_pid"
  if [ $gunicorn_pid != "" ]; then
      sleep 60
      kill -15 $gunicorn_pid  # Transfer SIGTERM signals to the Gunicorn process.
      wait $gunicorn_pid          # Wait until the Gunicorn process stops.
  fi
}

trap handle_sigterm TERM
```

4.2 Creating a Custom Image and Using It to Create an AI Application

If you want to use an AI engine that is not supported by ModelArts, create a custom image for the engine, import the image to ModelArts, and use the image to create AI applications. This section describes how to use a custom image to create an AI application and deploy the application as a real-time service.

The process is as follows:

1. **Building an Image Locally**: Create a custom image package locally. For details, see **Custom Image Specifications for Creating AI Applications**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Using the Custom Image to Create an AI Application**: Import the image to ModelArts AI application management.
4. **Deploying the AI Application as a Real-Time Service**: Deploy the model as a real-time service.

Building an Image Locally

This section uses a Linux x86_x64 host as an example. You can purchase an ECS of the same specifications or use an existing local host to create a custom image.

1. After logging in to the host, install Docker. For details, see **Docker official documents**. Alternatively, run the following commands to install Docker:
```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```
2. Obtain the base image. Ubuntu 18.04 is used in this example.
```
docker pull ubuntu:18.04
```
3. Create the **self-define-images** folder, and edit **Dockerfile** and **test_app.py** in the folder for the custom image. In the sample code, the application code runs on the Flask framework.

   The file structure is as follows:
```
self-define-images/
    --Dockerfile
    --test_app.py
```
   – **Dockerfile**
```
From ubuntu:18.04
# Configure the 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.example.com@g" /etc/apt/sources.list && \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.example.com@g" /etc/apt/sources.list && \
  apt-get update && \
  apt-get install -y python3 python3-pip && \
  pip3 install  --trusted-host https://repo.example.com -i https://repo.example.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 .
```

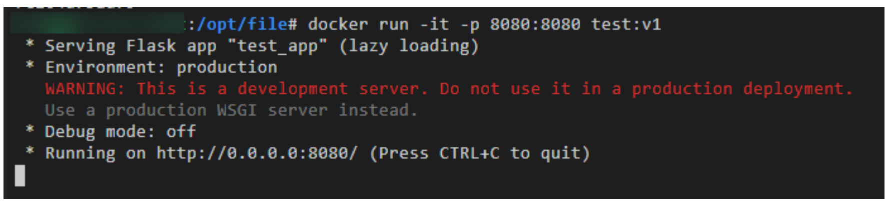
5. Run **docker images** to view the custom image you have created.

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 4-2 Starting a custom image



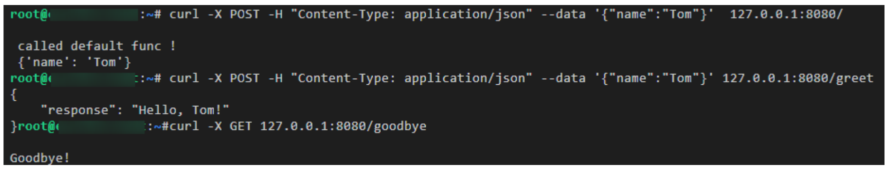
```
[redacted]:/opt/file# docker run -it -p 8080:8080 test:v1
 * Serving Flask app "test_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

2. Open another terminal and run the following commands to test the functions of the three APIs of the custom image:

```
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye
```

If information similar to the following is displayed, the function verification is successful.

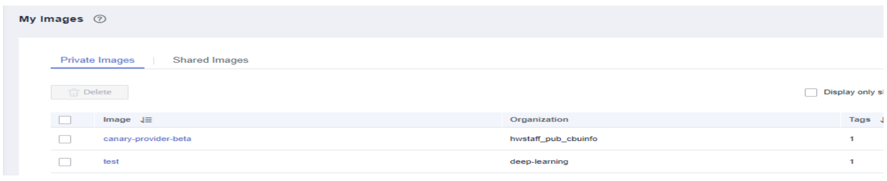
Figure 4-3 Testing API functions



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

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

Goodbye!
```

3. Upload the custom image to SWR. For details, see **How Can I Log In to SWR and Upload Images to It?**
4. View the uploaded image on the **My Images** > **Private Images** page of the SWR console.

Figure 4-4 Uploaded images



My Images

Private Images | Shared Images

Delete

Display only s

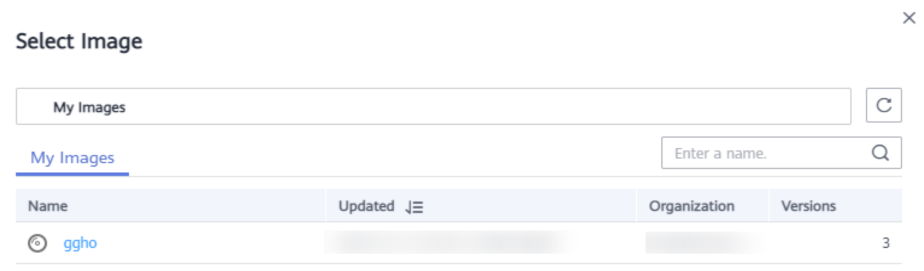
	Image	Organization	Tags
☐	canary-provider-beta	hwstaff_pub_cbuinfo	1
☐	test	deep-learning	1

Using the Custom Image to Create an AI Application

Import a meta model. For details, see **Creating and Importing a Model Image**. Key parameters are as follows:

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

Figure 4-5 Created private image



Select Image

My Images

My Images

Enter a name.

Name	Updated	Organization	Versions
ggho	[redacted]	[redacted]	3

  – **Container API**: Protocol and port number for starting a model. Ensure that the protocol and port number are the same as those provided in the custom image.
  – **Image Replication**: indicates whether to copy the model image in the container image to ModelArts. This parameter is optional.
  – **Health Check**: checks health status of a model. This parameter is optional. This parameter is configurable only when the health check API is configured in the custom image. Otherwise, creating the AI application will fail.
- **APIs**: APIs of a custom image. This parameter is optional. The model APIs must comply with ModelArts specifications. For details, see **Specifications for Editing a Model Configuration File**.

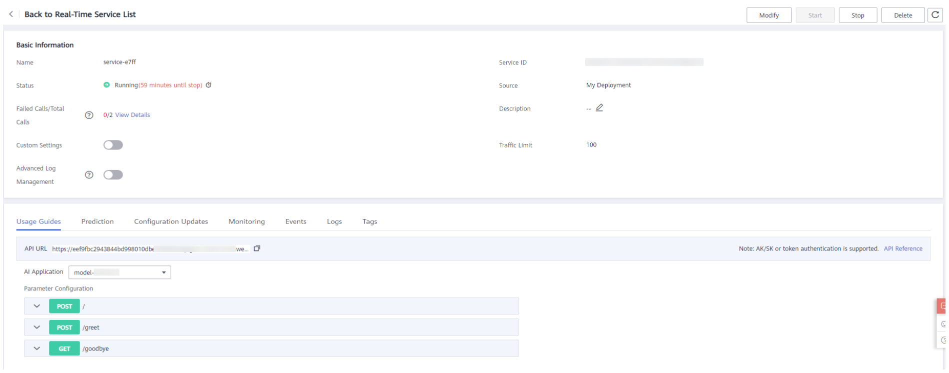
The configuration file is as follows:

```
[{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    },
{
        "url": "/greet",
        "method": "post",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    },
{
        "url": "/goodbye",
        "method": "get",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    }
]
```

Deploying the AI Application as a Real-Time Service

1. Deploy the AI application as a real-time service. For details, see **Deploying as a Real-Time Service**.
2. View the details about the real-time service.

Figure 4-6 Usage Guides



Basic Information

Name	service-e7ff	Service ID	[redacted]
Status	Running(59 minutes until stop)	Source	My Deployment
Failed Calls/Total Calls	0/2 View Details	Description	--
Custom Settings		Traffic Limit	100
Advanced Log Management			

API URL https://eef9fbc2943844bd998010dbe[redacted]we...
Note: AK/SK or token authentication is supported. API Reference

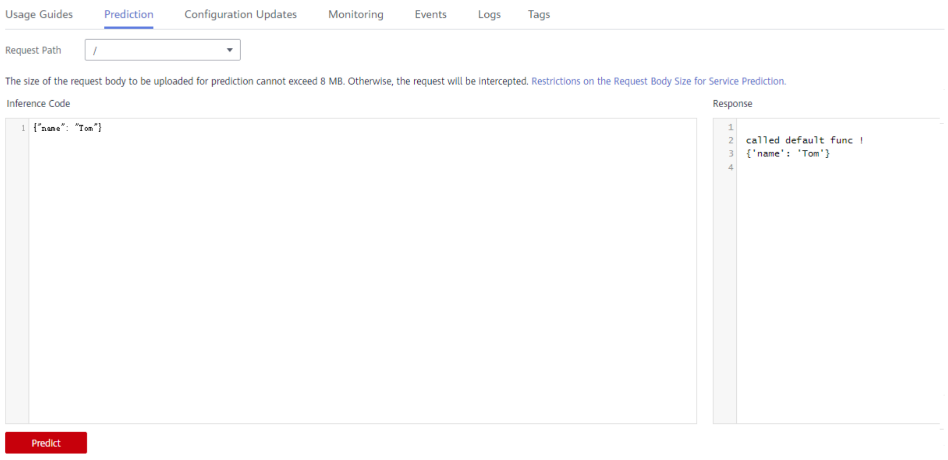
AI Application model-[redacted]

Parameter Configuration

POST /
POST /greet
GET /goodbye

3. Access the real-time service on the **Prediction** tab page.

Figure 4-7 Accessing a real-time service



The size of the request body to be uploaded for prediction cannot exceed 8 MB. Otherwise, the request will be intercepted. Restrictions on the Request Body Size for Service Prediction.

```
{"name": "Tom"}
```

```

called default func !
{'name': 'Tom'}

```

5 FAQs

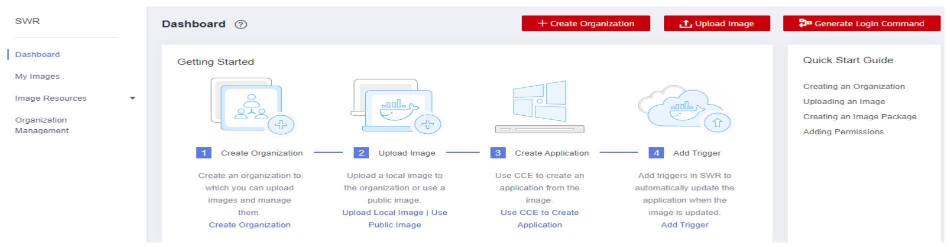
5.1 How Can I Log In to SWR and Upload Images to It?

This section describes how to log in to SWR and upload images to it.

Step 1 Log In to SWR

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

Figure 5-1 SWR console



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

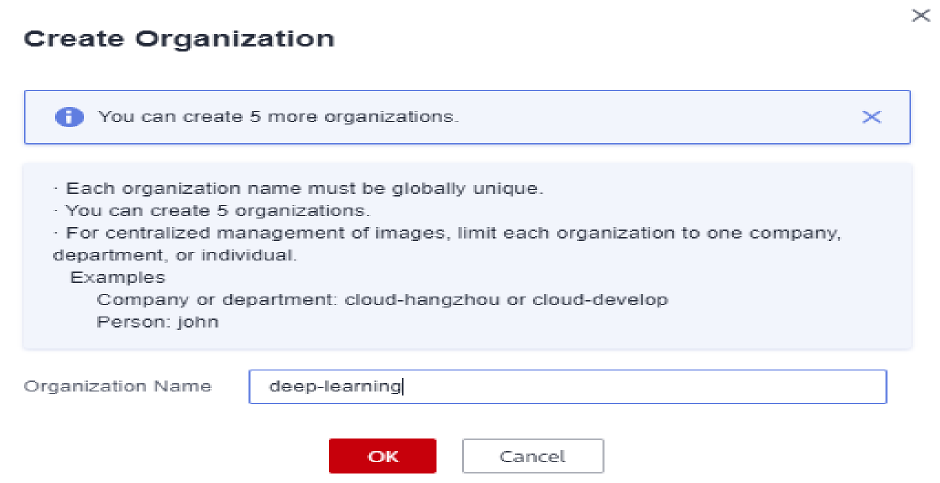
Upload a local image to the organization or use a public image.

Use CCE to create an application from the image.

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

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

Figure 5-2 Creating an organization



Create Organization

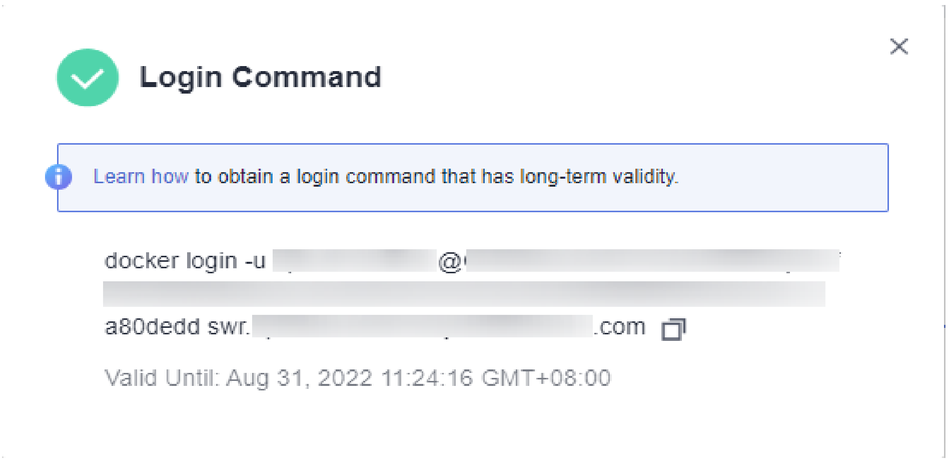
You can create 5 more organizations.

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

Organization Name: deep-learning

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

Figure 5-3 Login Command



Login Command

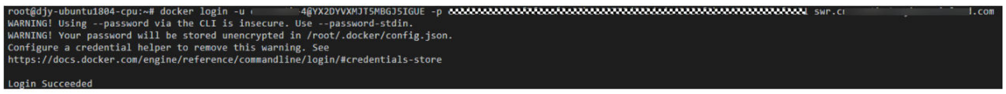
Learn how to obtain a login command that has long-term validity.

```
docker login -u [redacted]@[redacted]
[redacted]
a80dedd swr.[redacted].com
```

Valid Until: Aug 31, 2022 11:24:16 GMT+08:00

4. Log in to the ECS as user **root** and enter the login command.

Figure 5-4 Login command executed on the ECS



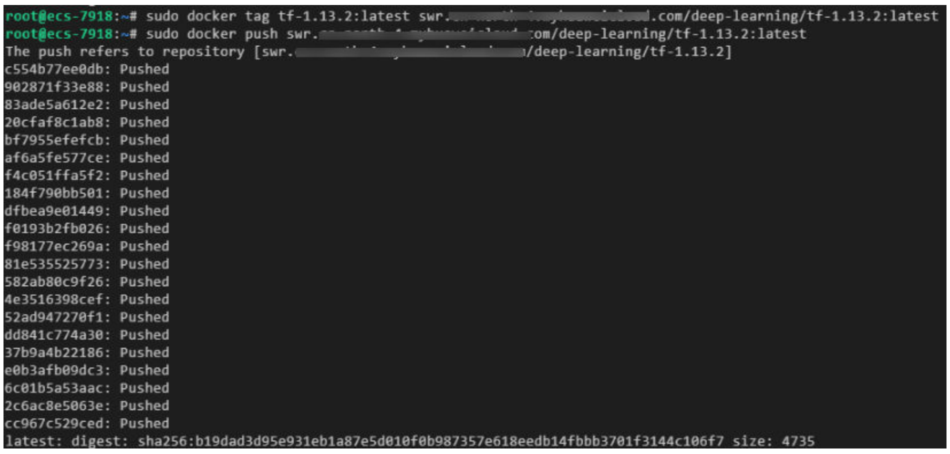
```
root@djy-ubuntu1804-cpu:~# docker login -u [redacted]4@YX2DYVXMJT5MBGJ5IGUE -p [redacted] swr.c[redacted].com
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Step 2 Upload Images to SWR

This section describes how to upload an image to SWR.

1. Log in to SWR and tag the image to be uploaded. Replace the organization name **deep-learning** in the following command with the actual organization name obtained in step 1.
   ```
   sudo docker tag tf-1.13.2:latest swr.example.com/deep-learning/tf-1.13.2:latest
   ```
2. Run the following command to upload the image:
   ```
   sudo docker push swr.example.com/deep-learning/tf-1.13.2:latest
   ```

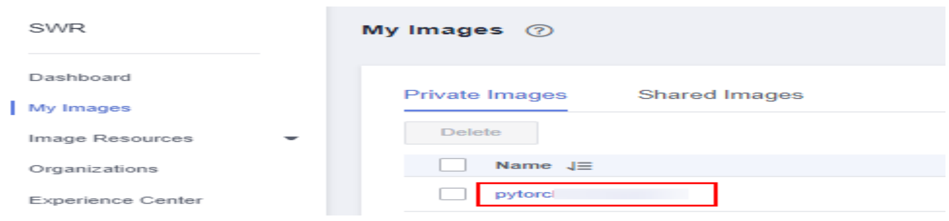
Figure 5-5 Uploading an image



```
root@ecs-7918:~# sudo docker tag tf-1.13.2:latest swr.[redacted].com/deep-learning/tf-1.13.2:latest
root@ecs-7918:~# sudo docker push swr.[redacted]com/deep-learning/tf-1.13.2:latest
The push refers to repository [swr.[redacted]/deep-learning/tf-1.13.2]
c554b77ee0db: Pushed
902871f33e88: Pushed
83ade5a612e2: Pushed
20cfaf8c1ab8: Pushed
bf7955efefcb: Pushed
af6a5fe577ce: Pushed
f4c051ffa5f2: Pushed
184f790bb501: Pushed
dfbea9e01449: Pushed
f0193b2fb026: Pushed
f98177ec269a: Pushed
81e535525773: Pushed
582ab80c9f26: Pushed
4e3516398cef: Pushed
52ad947270f1: Pushed
dd841c774a30: Pushed
37b9a4b22186: Pushed
e0b3afb09dc3: Pushed
6c01b5a53aac: Pushed
2c6ac8e5063e: Pushed
cc967c529ced: Pushed
latest: digest: sha256:b19dad3d95e931eb1a87e5d010f0b987357e618eedb14fbbb3701f3144c106f7 size: 4735
```

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

Figure 5-6 Uploaded custom image



swr.example.com/deep-learning/tf-1.13.2:latest is the SWR URL of the custom image.

5.2 How Do I Configure Environment Variables for an Image?

In a Dockerfile, use the ENV instruction to configure environment variables. For details, see **Dockerfile reference**.

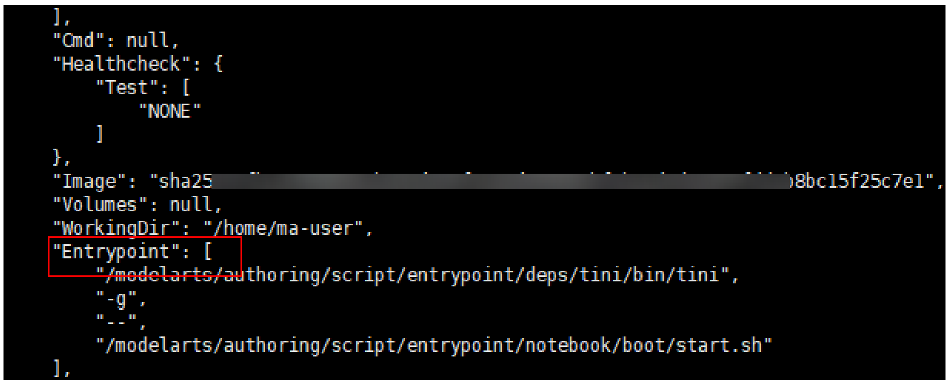
5.3 How Do I Use Docker to Start an Image Saved Using a Notebook Instance?

An image saved using a notebook instance contains the **Entrypoint** parameter, as shown in **Entrypoint**. The executable file or command specified in the **Entrypoint** parameter overwrites the default boot command of the image. The command input in the **Entrypoint** parameter is not preset in the image. When you run **docker run** in the local environment to start the image, an error message is displayed, indicating that the container creation task fails because the boot file or directory is not found, as shown in **Figure 5-8**.

To avoid this error, configure the **--entrypoint** parameter to overwrite the program specified in **Entrypoint**. Use the boot file or command specified by the **--entrypoint** parameter to start the image. Example:

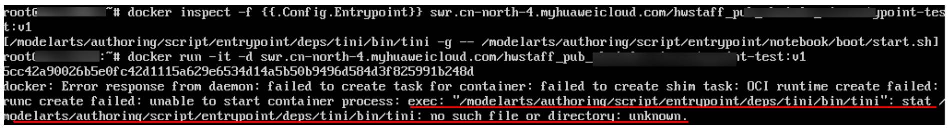
```
docker run -it -d --entrypoint /bin/bash image:tag
```

Figure 5-7 Entrypoint



```
    ],
    "Cmd": null,
    "Healthcheck": {
        "Test": [
            "NONE"
        ]
    },
    "Image": "sha25[redacted]8bc15f25c7e1",
    "Volumes": null,
    "WorkingDir": "/home/ma-user",
    "Entrypoint": [
        "/modelarts/authoring/script/entrypoint/deps/tini/bin/tini",
        "-g",
        "--",
        "/modelarts/authoring/script/entrypoint/notebook/boot/start.sh"
    ],
```

Figure 5-8 Error reported when an image is being started



```
root@[redacted]~# docker inspect -f {{.Config.Entrypoint}} swr.cn-north-4.myhuaweicloud.com/hwstaff_pu[redacted]ypoint-tes
t:v1
[/modelarts/authoring/script/entrypoint/deps/tini/bin/tini -g -- /modelarts/authoring/script/entrypoint/notebook/boot/start.sh]
root([redacted]:~# docker run -it -d swr.cn-north-4.myhuaweicloud.com/hwstaff_pub_[redacted]nt-test:v1
5cc42a90026b5e0fc42d1115a629e6534d14a5b50b9496d584d3f825991b248d
docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed:
runc create failed: unable to start container process: exec: "/modelarts/authoring/script/entrypoint/deps/tini/bin/tini": stat /
modelarts/authoring/script/entrypoint/deps/tini/bin/tini: no such file or directory: unknown.
```

5.4 How Do I Configure a Conda Source in a Notebook Development Environment?

You can install the development dependencies in Notebook as you need. Package management tools pip and Conda can be used to install regular dependencies. The pip source has been configured and can be used for installation, while the Conda source requires further configuration.

This section describes how to configure the Conda source on a notebook instance.

Configuring the Conda Source

The Conda software has been preset in images.

Common Conda Commands

For details about all Conda commands, see **Conda official documents**. The following table lists only common commands.

Table 5-1 Common Conda commands

Description	Command

Obtain online help.	`conda --help`
	`conda update --help # Obtain help for a command, for example, update.`

Description	Command
View the Conda version.	conda -V
Update Conda.	conda update conda # Update Conda.
	conda update anaconda # Update Anaconda.
Manage environments.	conda env list # Show all virtual environments.
	conda info -e # Show all virtual environments.
	conda create -n myenv python=3.7 # Create an environment named **myenv** with Python version **3.7**.
	conda activate myenv # Activate the **myenv** environment.
	conda deactivate # Disable the current environment.
	conda remove -n myenv --all # Delete the **myenv** environment.
	conda create -n newname --clone oldname # Clone the old environment to the new environment.
Manage packages.	conda list # Check the packages that have been installed in the current environment.
	conda list -n myenv # Specify the packages installed in the **myenv** environment.
	conda search numpy # Obtain all information of the **numpy** package.
	conda search numpy=1.12.0 --info # View the information of NumPy 1.12.0.
	conda install numpy pandas # Concurrently install the NumPy and Pandas packages.
	conda install numpy=1.12.0 # Install NumPy of a specified version.
	# The **install**, **update**, and **remove** commands use **-n** to specify an environment, and the **install** and **update** commands use **-c** to specify a source address.
	conda install -n myenv numpy # Install the **numpy** package in the **myenv** environment.
	conda install -c https://conda.anaconda.org/anaconda numpy # Install NumPy using https://conda.anaconda.org/anaconda.
	conda update numpy pandas # Concurrently update the NumPy and Pandas packages.
	conda remove numpy pandas # Concurrently uninstall the NumPy and Pandas packages.
	conda update –-all # Update all packages in the current environment.
Clear Conda.	conda clean -p # Delete useless packages.
	conda clean -t # Delete compressed packages.
	conda clean -y --all # Delete all installation packages and clear caches.

Saving as an Image

After installing the external libraries, save the environment using the image saving function provided by ModelArts notebook of the new version. You can save a running notebook instance as a custom image with one click for future use. After the dependency packages are installed on a notebook instance, it is a good practice to save the instance as an image to prevent the dependency packages from being lost. For details, see **Saving a Notebook Environment Image**.