

ModelArts

User Guide

Issue	01
Date	2025-09-08



Copyright © Huawei Technologies Co., Ltd. 2025. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks 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 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 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 ModelArts Standard Preparations..... 1

1.1 Configuring Access Authorization for ModelArts Standard..... 1

1.1.1 Configuring Agency Authorization for ModelArts with One Click..... 1

1.1.2 Creating an IAM User and Granting ModelArts Permissions..... 8

1.2 Creating an OBS Bucket for ModelArts to Store Data..... 15

2 ModelArts Standard Resource Management..... 18

2.1 About ModelArts Standard Resource Pools..... 18

2.2 Creating a Standard Dedicated Resource Pool..... 19

2.3 Managing Standard Dedicated Resource Pools.....25

2.3.1 Viewing Details About a Standard Dedicated Resource Pool..... 25

2.3.2 Resizing a Standard Dedicated Resource Pool..... 28

2.3.3 Upgrading the Standard Dedicated Resource Pool Driver..... 29

2.3.4 Rectifying a Faulty Node in a Standard Dedicated Resource Pool.....31

2.3.5 Modifying the Job Types Supported by a Standard Dedicated Resource Pool.....36

2.3.6 Configuring the Standard Dedicated Resource Pool to Access the Internet.....36

2.3.7 Releasing Standard Dedicated Resource Pools and Deleting the Network..... 38

3 Development Environments..... 39

3.1 Application Scenarios..... 39

3.2 Creating a Notebook Instance.....40

3.3 Using a Notebook Instance for AI Development Through JupyterLab..... 50

3.3.1 Using JupyterLab to Develop and Debug Code Online.....50

3.3.2 Common Functions of JupyterLab..... 51

3.3.3 Using Git to Clone the Code Repository in JupyterLab..... 60

3.3.4 Creating a Scheduled Job in JupyterLab.....66

3.3.5 Uploading Files to JupyterLab.....70

3.3.5.1 Uploading Files from a Local Path to JupyterLab.....70

3.3.5.2 Cloning GitHub Open-Source Repository Files to JupyterLab..... 77

3.3.5.3 Uploading OBS Files to JupyterLab.....78

3.3.5.4 Uploading Remote Files to JupyterLab.....81

3.3.6 Downloading a File from JupyterLab to a Local PC..... 82

3.3.7 Using MindInsight Visualization Jobs in JupyterLab..... 85

3.3.8 Using TensorBoard Visualization Jobs in JupyterLab..... 87

Issue 01 (2025-09-08)

Copyright © Huawei Technologies Co., Ltd.

iii

3.4 Using Notebook Instances Remotely Through PyCharm.....	90
3.4.1 Connecting to a Notebook Instance Through PyCharm Toolkit.....	90
3.4.2 Manually Connecting to a Notebook Instance Through PyCharm.....	97
3.4.3 Uploading Data to a Notebook Instance Through PyCharm.....	103
3.5 Using Notebook Instances Remotely Through VS Code.....	104
3.5.1 Connecting to a Notebook Instance Through VS Code.....	104
3.5.2 Installing VS Code.....	105
3.5.3 Connecting to a Notebook Instance Through VS Code Toolkit.....	106
3.5.4 Manually Connecting to a Notebook Instance Through VS Code.....	114
3.5.5 Uploading and Downloading Files in VS Code.....	119
3.6 Using a Notebook Instance Remotely with SSH	121
3.7 Managing Notebook Instances.....	127
3.7.1 Searching for a Notebook Instance.....	127
3.7.2 Updating a Notebook Instance.....	128
3.7.3 Starting, Stopping, or Deleting a Notebook Instance.....	129
3.7.4 Saving a Notebook Instance.....	130
3.7.5 Dynamically Expanding EVS Disk Capacity.....	132
3.7.6 Dynamically Mounting an OBS Parallel File System.....	133
3.8 ModelArts CLI Command Reference.....	134
3.8.1 (Optional) Installing ma-cli Locally.....	134
3.8.2 ma-cli Authentication.....	135
3.9 Using Moxing Commands in a Notebook Instance.....	137
3.9.1 MoXing Framework Functions.....	137
3.9.2 Using MoXing in Notebook.....	139
3.9.3 Mapping Between mox.file and Local APIs and Switchover.....	141
3.9.4 Sample Code for Common Operations.....	142
3.9.5 Sample Code for Advanced MoXing Usage.....	146
4 Model Training.....	149
4.1 Model Training Process.....	149
4.2 Preparing Model Training Code.....	152
4.2.1 Starting a Preset Image's Boot File.....	152
4.2.2 Developing Code for Training Using a Preset Image.....	159
4.2.3 Developing Code for Training Using a Custom Image.....	161
4.2.4 Configuring Password-free SSH Mutual Trust Between Nodes for a Training Job Created Using a Custom Image.....	164
4.3 Preparing a Model Training Image.....	165
4.4 Creating a Debug Training Job.....	166
4.4.1 Using PyCharm Toolkit to Create and Debug a Training Job.....	166
4.5 Creating an Algorithm.....	171
4.6 Creating a Training Job.....	179
4.7 Distributed Model Training.....	194
4.7.1 Overview.....	194

4.7.2 Creating a Single-Node Multi-Card Distributed Training Job (DataParallel).....	195
4.7.3 Creating a Multiple-Node Multi-Card Distributed Training Job (DistributedDataParallel).....	196
4.8 Incremental Model Training.....	206
4.9 High Model Training Reliability.....	207
4.9.1 Training Job Fault Tolerance Check.....	208
4.9.2 Training Log Failure Analysis.....	212
4.9.3 Training Job Rescheduling.....	213
4.9.4 Resumable Training.....	213
4.9.5 Enabling Unconditional Auto Restart.....	214
4.10 Managing Model Training Jobs.....	215
4.10.1 Viewing Training Job Details.....	215
4.10.2 Viewing the Resource Usage of a Training Job.....	217
4.10.3 Viewing the Model Evaluation Result.....	218
4.10.4 Viewing Training Job Events.....	223
4.10.5 Viewing Training Job Logs.....	224
4.10.6 Priority of a Training Job.....	231
4.10.7 Using Cloud Shell to Debug a Production Training Job.....	233
4.10.8 Rebuilding, Stopping, or Deleting a Training Job.....	235
4.10.9 Managing Environment Variables of a Training Container.....	237
5 Image Management.....	243
5.1 Application Scenarios of Custom Images.....	243
5.2 Creating a Custom Image for a Notebook Instance.....	245
5.2.1 Creating a Custom Image.....	245
5.2.2 Creating a Custom Image Using the Image Saving Function.....	247
5.3 Creating a Custom Image for Model Training.....	248
5.3.1 Creating a Custom Training Image.....	248
5.3.2 Creating a Custom Training Image Using a Preset Image.....	250
5.3.3 Migrating Existing Images to ModelArts.....	252
5.3.4 Creating a Custom Training Image (PyTorch + Ascend).....	255
5.3.5 Creating a Custom Training Image (MindSpore + Ascend).....	261
5.4 Creating a Custom Image for Inference(To go offline).....	277
5.4.1 Creating a Custom Image for a Model(To go offline).....	277
5.4.2 Creating a Custom Image on ECS(To go offline).....	279
6 Resource Monitoring.....	283
6.1 Overview.....	283
6.2 Viewing Monitoring Metrics on the ModelArts Console.....	284
6.3 Viewing All ModelArts Monitoring Metrics on the AOM Console.....	284
6.4 Using Grafana to View AOM Monitoring Metrics.....	341
6.4.1 Installing and Configuring Grafana.....	341
6.4.1.1 Installing and Configuring Grafana on Windows.....	341
6.4.1.2 Installing and Configuring Grafana on Linux.....	342
6.4.1.3 Installing and Configuring Grafana on a Notebook Instance.....	345

6.4.2 Configuring a Grafana Data Source.....	348
6.4.3 Configuring a Dashboard to View Metric Data.....	352
7 History.....	358
7.1 Inference Deployment(To go offline).....	358
7.1.1 Overview.....	358
7.1.2 Creating a Model.....	359
7.1.2.1 Creation Methods.....	359
7.1.2.2 Importing a Meta Model from a Training Job.....	360
7.1.2.3 Importing a Meta Model from OBS.....	362
7.1.2.4 Importing a Meta Model from a Container Image.....	365
7.1.3 Model Creation Specifications.....	369
7.1.3.1 Model Package Structure.....	369
7.1.3.2 Specifications for Editing a Model Configuration File.....	371
7.1.3.3 Specifications for Writing a Model Inference Code File.....	386
7.1.3.4 Specifications for Using a Custom Engine to Create a Model.....	391
7.1.3.5 Examples of Custom Scripts.....	394
7.1.4 Deploying a Model as Real-Time Inference Jobs.....	404
7.1.4.1 Deploying and Using Real-Time Inference.....	404
7.1.4.2 Deploying a Model as a Real-Time Service.....	405
7.1.4.3 Authentication Methods for Accessing Real-time Services.....	410
7.1.4.3.1 Accessing a Real-Time Service Through Token-based Authentication.....	410
7.1.4.3.2 Accessing a Real-Time Service Through App Authentication.....	414
7.1.4.4 Accessing a Real-Time Service Through Different Channels.....	424
7.1.4.4.1 Accessing a Real-Time Service Through a Public Network.....	424
7.1.4.4.2 Accessing a Real-Time Service Through a VPC High-Speed Channel.....	425
7.1.4.5 Accessing a Real-Time Service Using Different Protocols.....	429
7.1.4.5.1 Accessing a Real-Time Service Using Server-Sent Events.....	429
7.1.5 Deploying a Model as a Batch Inference Service.....	430
7.1.6 Managing ModelArts Models.....	436
7.1.6.1 Viewing ModelArts Model Details.....	437
7.1.6.2 Viewing ModelArts Model Events.....	441
7.1.6.3 Managing ModelArts Model Versions.....	445
7.1.7 Managing a Synchronous Real-Time Service.....	446
7.1.7.1 Viewing Details About a Real-Time Service.....	446
7.1.7.2 Viewing Events of a Real-Time Service.....	451
7.1.7.3 Managing the Lifecycle of a Real-Time Service.....	454
7.1.7.4 Modifying a Real-Time Service.....	455
7.1.7.5 Viewing Performance Metrics of a Real-Time Service on Cloud Eye.....	456
7.1.7.6 Configuring Auto Restart upon a Real-Time Service Fault.....	462
7.1.8 Managing Batch Inference Jobs.....	462
7.1.8.1 Viewing Details About a Batch Service.....	462
7.1.8.2 Viewing Events of a Batch Service.....	464

7.1.8.3 Managing the Lifecycle of a Batch Service.....467

7.1.8.4 Modifying a Batch Service.....468

1 ModelArts Standard Preparations

1.1 Configuring Access Authorization for ModelArts Standard

1.1.1 Configuring Agency Authorization for ModelArts with One Click

Scenarios

ModelArts, the AI platform, must access other services when executing tasks. For example, ModelArts must access OBS to read your data for training. In such cases, ModelArts accesses other cloud services on behalf of you. To ensure security, ModelArts requires your authorization before accessing any cloud services, which is the agency process. Then, you can perform AI computing tasks on ModelArts.

ModelArts provides one-click automatic authorization. You can quickly configure agency authorization on the **Permission Management** page of ModelArts. Then, ModelArts will automatically create an agency for you and configure it in ModelArts.

In this method, the authorization scope is specified based on the preset system policies of dependent services to ensure sufficient permissions for using services. The created agency has almost all permissions of dependent services. If you want to precisely control the scope of permissions granted to an agency, use custom authorization. For more about permissions management, see [Permissions Management](#).

This section introduces one-click automatic authorization. It allows you to grant permissions to IAM users, federated users (virtual IAM users), agencies, or all users with one click.

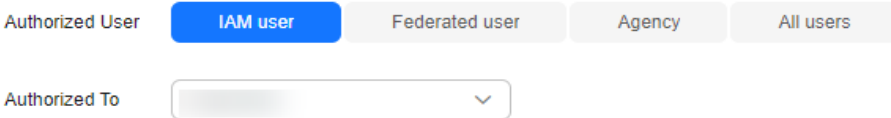
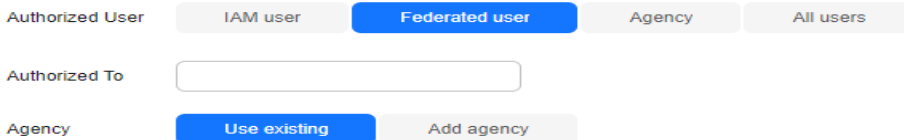
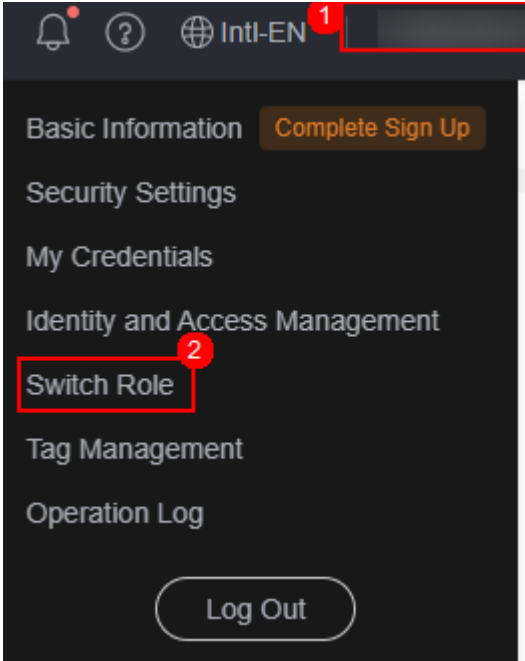
Adding Authorization

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Settings** page is displayed.

2. Click **Add Authorization**. On the **Add Authorization** page that is displayed, set parameters.

Table 1-1 Parameters

Parameter	Description
Authorized User	The options are IAM user, Federated user, Agency, and All users. • IAM user: You can use a tenant account to create IAM users and assign permissions for specific resources. Each IAM user has their own identity credentials (password and access keys) and uses cloud resources based on the assigned permissions. • Federated user: A federated user is also called a virtual enterprise user. • Agency: You can create agencies in IAM. For details about how to create an agency, see Creating an Agency. • All users: If you select this option, the agency permissions will be granted to all IAM users under the account, including those created in the future. For individual users, select All users.

Parameter	Description
Authorized To	This parameter is not displayed when Authorized User is set to All users. • IAM user: Select an IAM user and configure an agency for the IAM user. Figure 1-1 Selecting an IAM user  • Federated user: Enter the username or user ID of the target federated user. Figure 1-2 Entering a federated user  • Agency: Select an agency name. You can create an agency under account A and grant the agency permissions to account B. When using account B, you can switch the role in the upper right corner of the console to account A and use the agency permissions of account A. Figure 1-3 Switch Role 

Parameter	Description
Agency	● Use existing: If there are agencies in the list, select an available one to authorize the selected user. Click the drop-down arrow next to an agency name to view its permission details. ● Add agency: If there is no available agency, create one. If you use ModelArts for the first time, select Add agency.
Add agency > Agency Name	ModelArts automatically creates an agency name for you, but it is editable.
Add agency > Authorization Method	● Role-based: A coarse-grained IAM authorization strategy to assign permissions based on user responsibilities. Only a limited number of service-level roles are available. When using roles to grant permissions, assign other roles on which the permissions depend to take effect. However, roles are not the best choice for fine-grained authorization and secure access control. ● Policy-based: A fine-grained authorization tool that defines permissions for operations on specific cloud resources under certain conditions. This type of authorization is more flexible and ideal for secure access control. For details about roles and policies, see Basic Concepts.
Add agency > Permissions > Common User	You can use basic ModelArts functions, for example, accessing data and creating and managing training jobs. Select this option generally. Click View permissions to view common user permissions.
Add agency > Permissions > Custom	You can flexibly assign permissions to the created agency. Select this option for refined permission management. You can select the required permission from the list.

3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

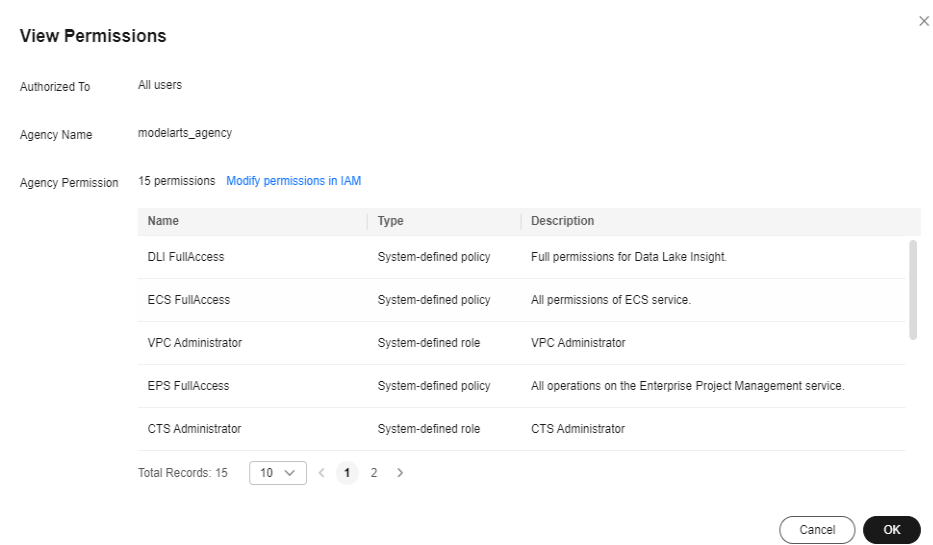
Viewing Authorized Permissions

You can view the configured authorizations on the **Permission Management** page. Locate an authorization and click **View Permissions** in the **Operation** column to view the permission details.

Figure 1-4 View Permissions

Authorized To	Authori...	Authori...	Authorization Content	Creation Time	Operation
All users	All users	Agency	modelarts_agency	T=0...	View Permissions Delete

Figure 1-5 Common user permissions



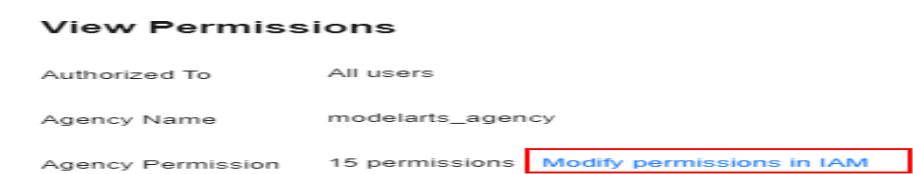
Modifying the Authorization Scope

NOTE

Modifying the authorization scope will cause running jobs to fail due to agency absence. Exercise caution when performing this operation.

1. To modify the authorization scope, click **Modify permissions in IAM** in the **View Permissions** dialog box.

Figure 1-6 Modify permissions in IAM



2. On the **Agencies** page of the IAM console, locate the target agency and modify the agency information on the **Basic Information** tab. Set **Validity Period** as needed. The value can be **Unlimited**, **1 day**, or **Custom**. If you select **Custom**, you can then enter the number of days you want the agency to stay valid for, for example, **30**.

Figure 1-7 Agency information

The screenshot shows a dialog box titled 'Agency Information' with two tabs: 'Basic Information' and 'Permissions'. The 'Basic Information' tab is selected. The form contains the following fields:

- Agency Name:** A text input field containing 'modelarts_agency'.
- URN:** A text input field containing 'iam::...'.
- ★ Agency Type:** A dropdown menu with 'Cloud service' selected.
- ★ Cloud Service:** A dropdown menu with 'ModelArts' selected.
- ★ Validity Period:** A dropdown menu with 'Unlimited' selected.
- Description:** A text area containing 'Created by ModelArts service.' with a character count '29/255' and a refresh icon.

At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

3. On the **Permissions** page, click **Authorize**, select policies or roles, and click **Next**. Select the scope for minimum authorization and click **OK**.

When setting the minimum authorization scope, you can select specified projects or all resources. If you select **All resources**, the selected permissions will be applied to all resources.

Deleting an Authorization

To better manage your authorization, you can delete the authorization of an IAM user or delete the authorizations of all users in batches.

NOTE

Deleting an authorization will cause running jobs to fail due to agency absence. Exercise caution when performing this operation.

- **Deleting the authorization of a user**

The authorizations configured for the IAM user of the current account are displayed on the **Permission Management** page. You can locate an authorization, click **Delete** in the **Operation** column, enter **DELETE** in the text box, and click **OK**. After it is deleted, the user cannot use ModelArts functions.

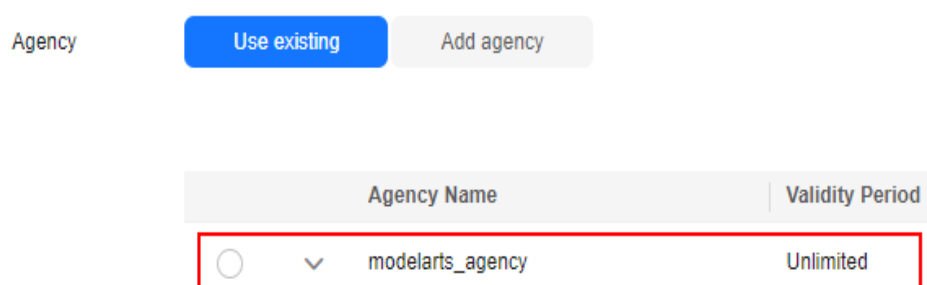
- **Deleting authorizations in batches**

On the **Permission Management** page, click **Clear Authorization** above the authorization list. Enter **DELETE** in the text box and click **OK**. After the deletion, the account and all its IAM users cannot use ModelArts functions.

FAQs

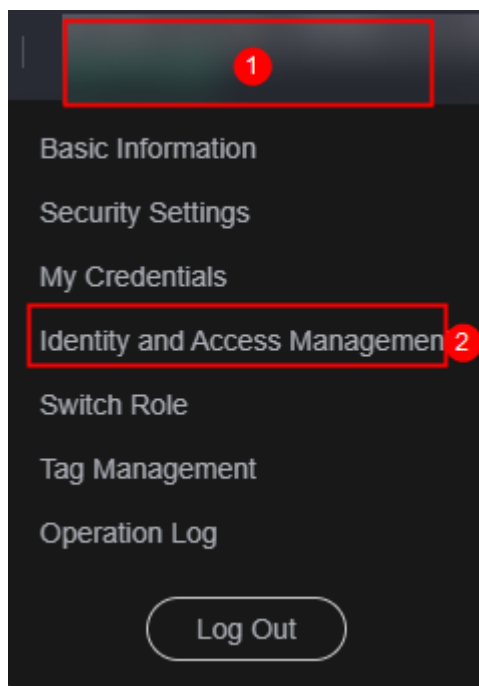
1. How do I configure authorization when I use ModelArts for the first time?
On the **Add Authorization** page, set **Agency** to **Add agency** and select **Common User**, which provides the permissions to use all basic ModelArts functions. For example, you can access data, and create and manage training jobs. Select this option generally.
2. How do I obtain access keys (AK/SK)?
You will need to obtain an access key if you are using access key authentication to access certain functions like using real-time services. For details, see [How Do I Obtain an Access Key?](#)
3. How do I delete an existing agency from the agency list?

Figure 1-8 Use existing

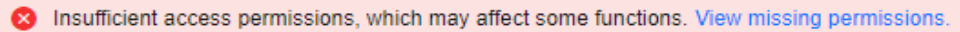


Go to the IAM console, choose **Agencies** in the navigation pane, and delete the target agency.

Figure 1-9 Identity and Access Management



4. Why is a message indicating insufficient permissions displayed when I access a page on the ModelArts console?

Figure 1-10 Insufficient permissionsA red rectangular error message box with a red 'x' icon on the left. The text inside reads: "Insufficient access permissions, which may affect some functions. [View missing permissions.](#)"

The permissions configured for the user agency are insufficient or the module has been upgraded. The authorization information needs to be updated. In this case, you can add authorization as prompted.

1.1.2 Creating an IAM User and Granting ModelArts Permissions

The agency created in [Configuring Agency Authorization for ModelArts with One Click](#) almost has all permissions of dependent services. If your Huawei Cloud account meets your permissions requirements, you can skip this section.

ModelArts allows you to configure fine-grained permissions for refined management of resources and permissions. This type of feature is commonly used in scenarios with large-scale enterprise users. If you want to precisely control the scope of permissions granted to an agency, use custom authorization.

This section describes how to configure fine-grained permissions for IAM users.

Using ModelArts requires OBS authorization. ModelArts users require OBS system permissions.

- For details about how to grant operation permissions of ModelArts and dependent services to a user, see [Configuring Common Operations Permissions for ModelArts Standard](#).
- For details about how to manage user permissions on ModelArts and dependent services in a refined manner and how to configure custom policies, see [Creating a Custom Policy for ModelArts Standard](#).

Prerequisites

You have learned about the permissions that can be added to user groups for using ModelArts and dependent services so that you can select them based on your requirements. For details about the permissions supported by ModelArts, see [Table 1-2](#).

Table 1-2 Service authorizations

Target Service	Authorization Description	IAM Permission	Mandatory
ModelArts	This permission allows sub-users to use ModelArts. The sub-users with the ModelArts CommonOperations permission can only use resources, but cannot create, update, or delete any dedicated resource pool. You are advised to assign this permission to sub-users.	ModelArts CommonOperations	Yes
	The sub-users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when assigning this permission.	ModelArts FullAccess	No Select either ModelArts FullAccess or ModelArts CommonOperations .
OBS	This permission allows sub-users to use OBS. ModelArts datasets, development environments, training jobs, and model inference and deployment require OBS for forwarding data .	OBS OperateAccess	Yes
SWR	This permission allows sub-users to use SWR. ModelArts custom images require the SWR FullAccess permission.	SWR OperateAccess	Yes
KMS	This permission allows sub-users to use remote SSH of ModelArts notebook.	KMS CMKFullAccess	No
IEF	This permission allows sub-users to use IEF. It is required if you use ModelArts edge services that depend on IEF.	Tenant Administrator	No

Target Service	Authorization Description	IAM Permission	Mandatory
Cloud Eye	This permission allows sub-users to use Cloud Eye. Using Cloud Eye, you can view the running statuses of ModelArts real-time services and model loads, and set monitoring alarms.	CES FullAccess	No
SMN	This permission allows sub-users to use SMN. SMN is used with Cloud Eye.	SMN FullAccess	No
VPC	When creating a ModelArts dedicated resource pool, sub-users require VPC permissions so that they can customize networks.	VPC FullAccess	No
SFS	This permission allows sub-users to use SFS. SFS file systems can be mounted to ModelArts dedicated resource pools to serve as storage for development environments or training jobs.	SFS Turbo FullAccess SFS FullAccess	No

Configuring Common Operations Permissions for ModelArts Standard

Step 1 Create a user group.

. Choose **User Groups** and click **Create User Group**. Enter a user group name and click **OK**.

Step 2 Configure permissions for the user group.

In the user group list, locate the created user group, click **Authorize** in the **Operation** column, and perform the following operations.

1. Assign permissions for using ModelArts. Search for **ModelArts** in the search box. Select either **ModelArts FullAccess** or **ModelArts CommonOperations**.

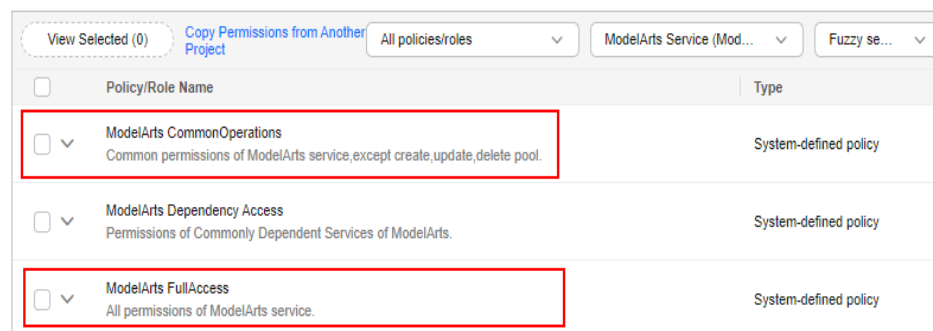
The differences between the options are as follows:

- The sub-users with the **ModelArts CommonOperations** permission can only use resources, but cannot create, update, or delete any dedicated resource pool. You are advised to assign this permission to sub-users.

- The sub-users with the **ModelArts FullAccess** permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when assigning this permission.

Figure 1-11 Assigning permissions for using ModelArts

Assign selected permissions to user_group. ?



2. Configure permissions for using other dependent services. For example, to use OBS, search for **OBS** and select **OBS OperateAccess**. ModelArts training jobs use OBS to forward data. Therefore, the permissions for using OBS are necessary.
For permissions of more cloud services, such as SWR, see [Table 1-2](#).
3. Click **Next** and set the minimum authorization scope. Select **Region-specific projects**, select the region to be authorized, and click **OK**.
4. A message is displayed, indicating that the authorization is successful. View the authorization information and click **Finish**. It takes 15 to 30 minutes for the authorization to take effect.

Step 3 Create a sub-user. In the navigation pane on the left of the IAM console, choose **Users**. In the right pane, click **Create User** in the upper right corner. On the **Create User** page, add multiple users. Set parameters as prompted and click **Next**.

Step 4 Add the sub-user to the user group. On the **Add User to Group** page, select a user group and click **Create**. The system adds the created users to the user group.

Step 5 Log in to the console as the created user, switch to the authorized region, and verify the permissions.

- Choose **Service List > ModelArts**. In the navigation pane of the ModelArts console, choose your desired type of AI dedicated resource pools and create one. You should not be able to create a new resource pool if the **ModelArts CommonOperations** permission has taken effect.
- Choose any other service in **Service List**. (Assume that the current policy contains only **ModelArts CommonOperations**.) If a message appears indicating that you have insufficient permissions to access the service, the **ModelArts CommonOperations** permission has taken effect.
- Choose **Service List > ModelArts**. In the navigation pane of the ModelArts console, choose **Asset Management > Datasets** and click **Create**. You should be able to access the corresponding OBS path if the OBS permission has taken effect.

- Verify other optional permissions according to [Table 1-2](#).

----End

Creating a Custom Policy for ModelArts Standard

In addition to the default system permissions of ModelArts, you can create custom policies for refined user permission management, for example, managing OBS operation permissions.

You can create custom policies using either the visual editor or JSON views on IAM. This section describes how to use a JSON view to create a custom policy to grant permissions required to use development environments and the minimum permissions required by ModelArts to access OBS.

For details about how to create custom policies and related parameters, see [Creating a Custom Policy](#).

NOTE

A custom policy can contain actions of multiple services that are globally accessible or accessible through region-specific projects.

ModelArts is accessible through region-specific projects, but OBS is globally accessible, so you need to create separate policies for the two services and then apply these policies to the users.

1. Create a custom policy with minimum OBS permissions for ModelArts.
Log in to the IAM console, choose **Permissions > Policies/Roles**, and click **Create Custom Policy**. Set the following parameters:
 - **Policy Name:** Enter a policy name.
 - **Policy View:** Select **JSON**.
 - **Policy Content:** Configure the policy by referring to [Example Custom Policy with OBS Permissions for ModelArts](#).

Figure 1-12 Minimum OBS permissions

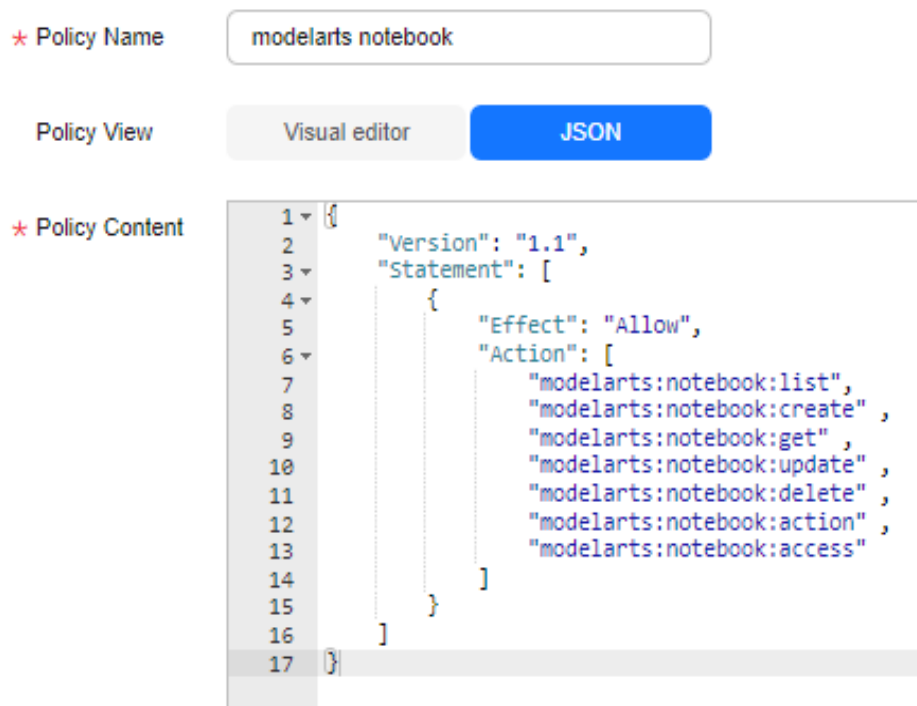
★ Policy Name

Policy View Visual editor JSON

★ Policy Content

```
1 {  
2   "Version": "1.1",  
3   "Statement": [  
4     {  
5       "Action": [  
6         "obs:bucket:ListAllMybuckets",  
7         "obs:bucket:HeadBucket",  
8         "obs:bucket:ListBucket",  
9         "obs:bucket:GetBucketLocation",  
10        "obs:object:GetObject",  
11        "obs:object:GetObjectVersion",  
12        "obs:object:PutObject",  
13        "obs:object:DeleteObject",  
14        "obs:object:DeleteObjectVersion",  
15        "obs:object:ListMultipartUploadParts",  
16        "obs:object:AbortMultipartUpload",  
17        "obs:object:GetObjectAcl",  
18        "obs:object:GetObjectVersionAcl",  
19        "obs:bucket:PutBucketAcl",  
20        "obs:object:PutObjectAcl"  
21      ],  
22      "Effect": "Allow"  
23    }  
24  ]  
25 }
```

2. Create a custom policy for the permission to use the ModelArts development environment, as shown in [Figure 1-13](#). Set the following parameters:
 - **Policy Name:** Enter a policy name.
 - **Policy View:** Select **JSON**.
 - **Policy Content:** Configure the policy by referring to [Example Custom Policy with Permissions for Using the ModelArts Development Environment](#).

Figure 1-13 Permission to use the development environment

- To authorize the services other than ModelArts and OBS, see [ModelArts Standard Role/Policy-based Authorization](#).
- 3. After creating a user group on the IAM console, grant the custom policy created in **1** to the user group.
- 4. Create a user on the IAM console and add the user to the group created in **3**.
- 5. Log in to the console as the created user, switch to the authorized region, and verify the permissions.
 - Choose **Service List > ModelArts**. In the navigation pane of the ModelArts console, choose **Asset Management > Datasets**. If you cannot create a dataset, the permission granted only for using the development environment has taken effect.
 - Choose **Service List > ModelArts**. In the navigation pane of the ModelArts console, choose **Development & Production > DevEnviron > Notebook** and click **Create Notebook**. If you can access the OBS path specified in **Storage**, the OBS permission has taken effect.

Example Custom Policy with OBS Permissions for ModelArts

In this example, the minimum permissions required by ModelArts to access OBS are assigned, including the permissions for OBS buckets and objects. With these permissions, you can access OBS from ModelArts without restrictions.

```
{  
  "Version": "1.1",  
  "Statement": [  
    {  
      "Action": [  
        "obs:bucket:ListAllMybuckets",  
        "obs:bucket:HeadBucket",  
        "obs:object:PutObject",  
        "obs:object:DeleteObject",  
        "obs:object:HeadObject",  
        "obs:object:CopyObject",  
        "obs:object:MoveObject",  
        "obs:object:RestoreObject",  
        "obs:object:ListMyobjects",  
        "obs:object:DownloadObject",  
        "obs:object:UploadObject",  
        "obs:object:CopyObject",  
        "obs:object:MoveObject",  
        "obs:object:RestoreObject",  
        "obs:object:ListMyobjects",  
        "obs:object:DownloadObject",  
        "obs:object:UploadObject"  
      ]  
    }  
  ]  
}
```

```
        "obs:bucket:ListBucket",
        "obs:bucket:GetBucketLocation",
        "obs:object:GetObject",
        "obs:object:GetObjectVersion",
        "obs:object:PutObject",
        "obs:object:DeleteObject",
        "obs:object:DeleteObjectVersion",
        "obs:object:ListMultipartUploadParts",
        "obs:object:AbortMultipartUpload",
        "obs:object:GetObjectAcl",
        "obs:object:GetObjectVersionAcl",
        "obs:bucket:PutBucketAcl",
        "obs:object:PutObjectAcl"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Example Custom Policy with Permissions for Using the ModelArts Development Environment

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:notebook:list",
        "modelarts:notebook:create",
        "modelarts:notebook:get",
        "modelarts:notebook:update",
        "modelarts:notebook:delete",
        "modelarts:notebook:action",
        "modelarts:notebook:access"
      ]
    }
  ]
}
```

1.2 Creating an OBS Bucket for ModelArts to Store Data

ModelArts does not provide data storage. Instead, it uses Object Storage Service (OBS) to store data, and backs up and takes snapshots for models, achieving secure, reliable, and cost-effective storage.

All the input data, output data, and cache data during AI development can be stored in OBS buckets for reading. Before using ModelArts, create an OBS bucket and folders for storing data.

Figure 1-14 Interaction between ModelArts and OBS

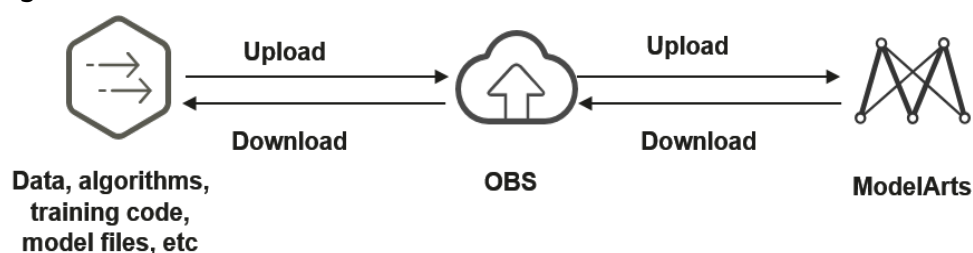
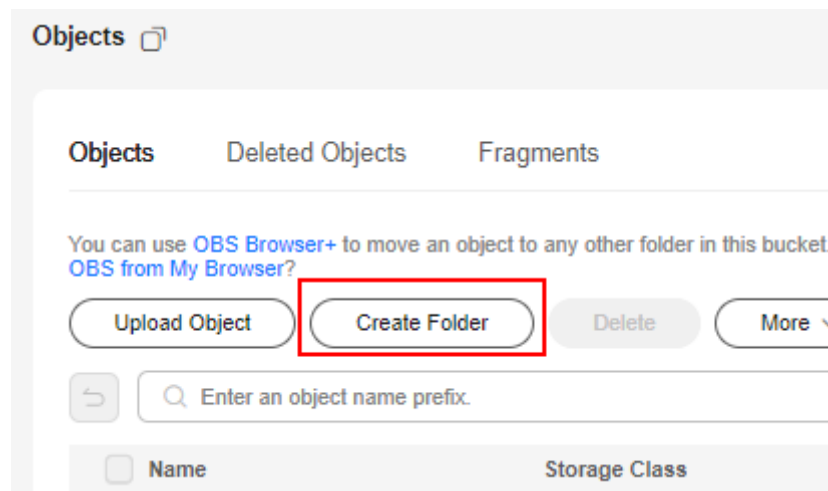


Table 1-3 Relationship between ModelArts and OBS

Function	Subtask	Relationship
Standard workflow of ModelArts Standard ExeML	Data labeling	The data labeled on ModelArts is stored in OBS.
	Auto training	After a training job is complete, the generated model is stored in OBS.
	Model deployment	ModelArts deploys models stored in OBS as real-time services.
ModelArts Standard AI development lifecycle	Data management	• Datasets are stored in OBS. • The dataset labeling information is stored in OBS. • Data can be imported from OBS.
	Development environment	Data or code files in a notebook instance can be stored in OBS.
	Model training	Datasets, algorithms, running scripts, training outputs, and training logs used by training jobs can be stored in OBS.
	Inference deployment	After a training job is complete, the generated model can be stored in OBS. You can import the model from OBS when creating a model.

Procedure for Creating an OBS Bucket

1. Log in to the [OBS console](#) and click **Create Bucket** in the upper right corner of the page to create an OBS bucket.
2. In the bucket list, click the bucket name to access its details page.
3. In the navigation pane on the left, choose **Objects**. On the **Objects** page, click **Create Folder** to create an OBS folder. For example, create a folder named **flowers** in the created **c-flowers** OBS bucket.

Figure 1-15 Creating a folder

After creating a folder in the OBS bucket, you can upload files. For details about how to upload files, see [Upload Overview](#).

FAQs

- Why can't I find my created OBS bucket after I select an OBS path in ModelArts?
- How do I check whether ModelArts and an OBS bucket are in the same region?
- What should I do if "Error: stat:403" is reported when I perform operations on an OBS bucket?

To resolve the preceding problems or other OBS path exceptions, see [Incorrect OBS Path on ModelArts](#).

2 ModelArts Standard Resource Management

2.1 About ModelArts Standard Resource Pools

This section describes the required compute resources when you use ModelArts for AI development, including ExeML, creating a workflow, creating a notebook instance, creating a training job, and creating an inference service. You can purchase a standard resource pool as needed.

ModelArts Standard Resource Pools

When using ModelArts for AI development, you can use either of the following resource pools:

- **Dedicated resource pool:** It delivers more controllable resources and cannot be shared with other users. Create a dedicated resource pool and select it during AI development.
- **Public Resource Pool:** provides large-scale public computing clusters, which are allocated based on job parameter settings. Resources are isolated by job. You can use ModelArts public resource pools to deliver training jobs, deploy models, or run DevEnviron instances and will be billed on a pay-per-use basis.

Differences between **dedicated resource pools** and **public resource pools**:

- Dedicated resource pools provide dedicated computing clusters and network resources for users. The dedicated resource pools of different users are physically isolated, while public resource pools are only logically isolated. Compared with public resource pools, dedicated resource pools feature better performance in isolation and security.
- When a dedicated resource pool is used for creating jobs and the resources are sufficient, the jobs will not be queued. When a public resource pool is used for creating jobs, there is a high probability that the jobs will be queued.
- A dedicated resource pool is accessible to your network. All running jobs in the pool can access storage and resources in your network. For example, if you select a dedicated resource pool with an accessible network when

creating a training job, you can access SFS data after the training job is created.

- Dedicated resource pools allow you to customize the runtime environment of physical nodes, for example, you can upgrade GPU or Ascend drivers. This function is not supported by public resource pools.

Instructions of Dedicated Resource Pools

- If you are using dedicated resource pools for the first time, get started by reading this section.
- Create a dedicated resource pool by referring to [Creating a Standard Dedicated Resource Pool](#).
- View the details about a created dedicated resource pool by referring to [Viewing Details About a Standard Dedicated Resource Pool](#).
- If the specifications of a dedicated resource pool do not meet your service requirements, adjust the specifications by referring to [Resizing a Standard Dedicated Resource Pool](#).
- On the dedicated resource pool list page, you can select an accelerator card driver and perform changes upon submission or smooth upgrade of the driver based on service requirements. Upgrade the GPU/Ascend driver of your dedicated resource pools by referring to [Upgrading the Standard Dedicated Resource Pool Driver](#).
- Set or change job types supported by a dedicated resource pool by referring to [Modifying the Job Types Supported by a Standard Dedicated Resource Pool](#).
- If a dedicated resource pool is no longer needed, delete it by referring to [Releasing Standard Dedicated Resource Pools and Deleting the Network](#).

2.2 Creating a Standard Dedicated Resource Pool

This section describes how to create a standard dedicated resource pool.

Prerequisites

- A VPC is available.
- A subnet is available.

Step 1: Create a Network

ModelArts networks are backed by VPCs and used for interconnecting nodes in a ModelArts resource pool. You can only configure the name and CIDR block for a network. To ensure that there is no IP address segment in the CIDR block overlapped with that of the VPC to be accessed, multiple CIDR blocks are available for you to select. A VPC provides a logically isolated virtual network for your instances. You can configure and manage the network as required. VPC provides logically isolated, configurable, and manageable virtual networks for cloud servers, cloud containers, and cloud databases. It helps you improve cloud service security and simplify network deployment.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.

2. Click the **Network** tab and click **Create**.
3. In the **Create Network** dialog box, set parameters.
 - **Network Name**: customizable name
 - **CIDR Block**: You can select **Preset** or **Custom**. Recommended CIDR blocks for a custom network: 10.0.0.0/8-24, 172.16.0.0/12-24, and 192.168.0.0/16-24. The subnet mask ranges from 8 to 28.

Figure 2-1 Creating a network

Create Network

Network Name

Network Segment Type **Preset** Customization

?

The VPC network segment, container network segment (fixed at 172.16.0.0/16), service network segment (fixed at 10.247.0.0/16), and created VPC network segment can be displayed only in the same workspace.

Cancel OK

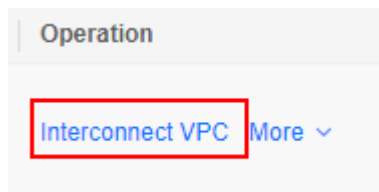
NOTE

- Each user can create a maximum of 15 networks.
 - Ensure there is no IP address segment in the CIDR block overlaps that of the VPC to be accessed. The CIDR block cannot be changed after the network is created. Possible conflict CIDR blocks are as follows:
 - Your VPC CIDR block
 - Container CIDR block (consistently to be 172.16.0.0/16)
 - Service CIDR block (consistently to be 10.247.0.0/16)
4. Confirm the settings and click **OK**.

(Optional) Step 2: Interconnect with a VPC

VPC interconnection allows you to use resources across VPCs, improving resource utilization.

1. On the **Network** page, click **Interconnect VPC** in the **Operation** column of the target network.

Figure 2-2 Interconnecting the VPC

2. In the displayed dialog box, click the button on the right of **Interconnect VPC**, and select an available VPC and subnet from the drop-down lists.

NOTE

The peer network to be interconnected cannot overlap with the current CIDR block.

Figure 2-3 Parameters for interconnecting a VPC with a network

Interconnect VPC - [redacted] X

Interconnect VPC ☒

VPC [Create VPC](#)

Subnet [Create](#)

default route ☐

- If no VPC is available, click **Create VPC** on the right to create a VPC.
- If no subnet is available, click **Create Subnet** on the right to create a subnet.
- A VPC can interconnect with at most 10 subnets. To add a subnet, click the plus sign (+).
- To enable a dedicated resource pool to access the public network through a VPC, create a SNAT in the VPC, as the public network address is unknown. After the VPC is interconnected, by default, the public address cannot be forwarded to the SNAT of your VPC. To add a default route, contact technical support. Then, when you interconnect with a VPC, ModelArts **0.0.0.0/0** is used as the default route. In this case, you do not need to submit a service ticket. Add the default route for network configuration.

Step 3: Create a Standard Dedicated Resource Pool

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.
2. In the **Standard Resource Pool** tab, click **Buy AI Dedicated Cluster**. On the displayed page, configure the parameters as follows.

Table 2-1 Parameters

Parameter	Sub-Parameter	Description
Name	-	Enter a name. Only lowercase letters, digits, and hyphens (-) are allowed. The value must start with a lowercase letter and cannot end with a hyphen (-).
Description	-	Enter a brief description of the dedicated resource pool.
Billing Modes	-	You can select Pay-per-use .
Job Type	-	Select job types supported by the resource pool based on service requirements. DevEnviron , Training Job , and Inference Service are supported.
Network	-	Network in which the target service instance is deployed. The instance can exchange data with other cloud service resources in the same network. Select a network from the drop-down list box. If no network is available, click Create on the right to create a network. For details about how to create a network, see Step 1: Create a Network .
Instance Specification Management	Instance Specifications Type	Select required specifications. Due to system loss, the available resources are fewer than specified. After a dedicated resource pool is created, view the available resources in the Nodes tab on the details page. Contact your account manager to request resource specifications (such as Ascend) in advance. They will enable the specifications within one to three working days. If there is no account manager, submit a service ticket.
	AZ	You can select Automatically allocated or Specifies AZ . An AZ is a physical region where resources use independent power supplies and networks. AZs are physically isolated but interconnected over an intranet. • Automatically allocated: AZs are automatically allocated. • Specifies AZ: Specify AZs for resource pool nodes. To ensure system disaster recovery, deploy all nodes in the same AZ. You can set the number of instances in an AZ.

Parameter	Sub-Parameter	Description
	Instances	Select the number of instances in a dedicated resource pool. More instances mean higher computing performance. If AZ Allocation is set to Manual, you do not need to configure Instances. NOTE It is a good practice to create no more than 30 instances at a time. Otherwise, the creation may fail due to traffic limiting. For certain specifications, you can purchase instances by rack. The instances you purchase is the number of racks multiplied by rack(6 node). Purchasing a full rack allows you to isolate tasks physically, preventing communication conflicts and maintaining linear computing performance as task scale increases. All nodes in a rack must be created or deleted together.
	Advanced Configuration	After this function is enabled, you can set the container engine space size and OS. You must enter an integer for the container engine space. It cannot be less than 50 GB, which is the default and minimum value. The maximum value depends on the specifications. To see the valid values, check the console prompt. Customizing the container engine space does not increase costs.
Custom Driver	-	Disabled by default. Some GPU and Ascend resource pools allow custom driver installation. The driver is automatically installed in the cluster by default. Enable this function if you need to specify the driver version.
GPU / Ascend Driver	-	This parameter is displayed if Custom Driver is enabled. You can select a GPU or Ascend driver. The value depends on the driver you choose.
Advanced Options	-	Select Configure Now to set the tag information, CIDR block, cluster specifications, and controller node distribution.

Parameter	Sub-Parameter	Description
CIDR block	-	You can select Default or Custom. • Default: The system randomly allocates an available CIDR block to you, which cannot be modified after the resource pool is created. For commercial use, customize your CIDR block. • Custom: You need to customize Kubernetes container and Kubernetes service CIDR blocks. – K8S Container Network: used by the container in a cluster, which determines how many containers there can be in a cluster. The value cannot be changed after the resource pool is created. – Kubernetes Service CIDR Block: CIDR block for services used by containers in the same cluster to access each other. The value determines the maximum number of Services you can create. The value cannot be changed after the cluster is created.
Cluster Specifications	-	Cluster Scale: maximum number of nodes that can be managed by the cluster. After the creation, the cluster can be scaled out but cannot be scaled in. You can select Default or Custom.
Master Distribution	-	Distribution locations of controller nodes. You can select Random or Custom. • Random: AZs of controller nodes are randomly allocated to improve DR capabilities. If the number of available AZs is less than the number of nodes to be created, the nodes will be created in the AZs with sufficient resources to preferentially ensure cluster creation. In this case, AZ-level DR may not be ensured. • Custom: Select AZs for controller nodes. Distribute controller nodes in different AZs for disaster recovery.

3. Click **Submit**.

- After a resource pool is created, its status changes to **Running**. Only when the number of available nodes is greater than 0, tasks can be delivered to this resource pool.
- Hover over **Creating** to view the details about the creation process. Click **View Details** to go the operation record page.
- You can view the task records of the resource pool by clicking **Records** in the upper left corner of the resource pool list.

FAQs

What if I choose a flavor for a dedicated resource pool, but get an error message saying no resource is available?

The flavors of dedicated resources change based on real-time availability. Sometimes, you might choose a flavor on the purchase page, but it is sold out before you pay and create the resource pool. This causes the resource pool creation to fail.

You can try a different flavor on the creation page and create the resource pool again.

Q: Why cannot I use all the CPU resources on a node in a resource pool?

Resource pool nodes have systems and plug-ins installed on them. These take up some CPU resources. For example, if a node has 8 vCPUs, but some of them are used by system components, the available resources will be fewer than 8 vCPUs.

You can check the available CPU resources by clicking the **Nodes** tab on the resource pool details page, before you start a task.

2.3 Managing Standard Dedicated Resource Pools

2.3.1 Viewing Details About a Standard Dedicated Resource Pool

Resource Pool Details Page

- Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.
- You can search for the resource pools by name, ID, resource pool status, node status, resource pool type, and creation time.
- In the resource pool list, click a resource pool to go to its details page and view its information.
 - If there are multiple ModelArts Standard resource pools, click ▼ in the upper left corner of the details page of one resource pool to switch between resource pools.
 - For a pay-per-use standard resource pool: On the details page, click **More** in the upper right corner to scale in/out, delete a resource pool, and perform other operations. Operations that can be performed vary depending on resource pools.
 - In the **Network** area of **Basic Information**, you can click the number of resource pools associated to view associated resource pools. You can view the number of available IP addresses on the network.
 - In the extended information area, you can view the monitoring information, jobs, nodes, specifications, tags, and events. For details, see the following section.

Viewing Jobs in a Resource Pool

On the resource pool details page, click **Jobs**. You can view all jobs running in the resource pool. If a job is queuing, you can view its queuing position.

NOTE

Only training jobs can be viewed.

Viewing Resource Pool Events

On the resource pool details page, click **Events**. You can view all events of the resource pool. The cause of an event is **PoolStatusChange** or **PoolResourcesStatusChange**.

In the event list, click  on the right of **Event Type** to filter events.

- When a resource pool starts to be created or becomes abnormal, the resource pool status changes and the change will be recorded as an event.
- When the number of nodes that are available or abnormal or in the process of being created or deleted changes, the resource pool node status changes and the change will be recorded as an event.

Viewing Resource Pool Nodes

On the resource pool details page, click **Nodes**. You can view all nodes in the resource pool.

Some resources are reserved for cluster components. Therefore, **CPUs (Available/Total)** does not indicate the number of physical resources on the node. It only displays the number of resources that can be used by services. CPU cores are metered in milicores, and 1,000 milicores equal 1 physical core.

In the node search box, you can search for nodes by node name, node status, HA redundancy, batch, driver version, driver status, IP address, and resource tag.

You can export the node information of a standard resource pool to an Excel file. Select the node names, click **Export > Export All Data to XLSX** or **Export >**

Export Part Data to XLSX above the node list, and click  in the browser to view the exported Excel files.

On the node list page, click  to customize the information displayed in the node list.

Viewing Resource Pool Specifications

On the resource pool details page, click **Specifications**. You can view the resource specifications used by the resource pool and the number of resources corresponding to the specifications, and adjust the container engine space.

Viewing Resource Pool Monitoring Information

On the resource pool details page, click **Monitoring**. The resource usage including used CPUs, memory usage, and available disk capacity of the resource pool is

displayed. If AI accelerators are used in the resource pool, the GPU and NPU monitoring information is also displayed.

Table 2-2 Monitoring metrics

Parameter	Description	Unit	Value Range
CPU usage	CPU usage of a measured object	%	0%–100%
Memory usage	Percentage of the used physical memory to the total physical memory	%	0%–100%
Used GPUs	Percentage of the used GPU memory to the total GPU memory	%	0%–100%
Used GPU memory	Percentage of the used GPU memory to the total GPU memory	%	0%–100%
Used NPUs	Percentage of the used NPU memory to the total NPU memory	%	0%–100%
Used NPU Memory	Percentage of the used GPU memory to the total GPU memory	%	0%–100%
Available Disk Capacity (GB)	Available disk capacity of a measured object	MB	≥0
Disk Capacity (GB)	Total disk capacity of a monitored object	MB	≥0
Disk Usage	Disk usage of the monitored object	%	0%–100%
GPU/NPU Fragments	Fragments are generated during resource scheduling. As a result, some cards are idle but cannot be used by multi-card tasks. For tasks with different numbers of cards, fragments vary according to the distribution of occupied cards and vary with time. The table lists only the status at the current time.	/	/

2.3.2 Resizing a Standard Dedicated Resource Pool

Description

The demand for resources in a dedicated resource pool may change due to the changes of AI development services. In this case, you can resize your dedicated resource pools in ModelArts Standard.

- You can add instances for existing flavors in the resource pool or add flavors to scale out this resource pool.
- You can delete instances for existing flavors in the resource pool.

NOTE

Before scaling in a resource pool, ensure that there are no services running in the pool. Alternatively, go to the resource pool details page, delete the nodes where no services are running to scale in the pool.

Constraints

- Only dedicated resource pools in the **Running** status can be resized.
- When resizing a dedicated resource pool, you cannot add nodes for a flavor and delete nodes for another flavor at the same time.
- When scaling in a dedicated resource pool, the number of flavors or nodes of a flavor cannot be decreased to 0.

Resizing a Dedicated Resource Pool

You can resize a resource pool in any of the following ways:

- Increase or decrease the total number of instances in existing specifications.
 - Resizing the container engine space
1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.

NOTE

A resource pool is suspended when it is migrated from the old version to the new version. You cannot adjust the capacity of such a resource pool.

2. Add or delete nodes.

Click **Adjust Capacity** in the **Operation** column of the target resource pool.

In the **Resource Configurations** area, set **AZ Allocation** to **Automatic** or **Manual**. Click **Submit** and then **OK** to save the changes.

- If **AZ** is set to **Automatic**, you can increase or decrease the number of nodes to scale out or in the resource pool. After the scaling, nodes are automatically allocated to AZs.
- If you select **Specifies AZ**, you can allocate nodes to different AZs.

3. On the dedicated resource pool scaling page, set **Target Instances** as needed. Scale-out is the process of increasing the number of target instances, while scale-in is the process of decreasing the number of target instances.

Figure 2-4 Configuring resources by rack

The screenshot displays the 'Resource Configurations' panel. On the left, under 'Instance Specifications', the CPU is '8 vCPUs 32GiB (modelarts.vm.cpu.8ud)'. The 'AZ Allocation' is set to 'Automatic' (highlighted in blue) with a 'Manual' option available. The 'Container Engine' section shows a space size of '50 GiB' with minus and plus buttons for adjustment. The 'Operating system' is shown as a dropdown menu. On the right, 'Specifications' are listed as '8vCPUs | 32GiB | 500GiB'. Below this, 'Instances in Total' is '1', and 'Target Instances' is a control with minus, 1, and plus buttons.

4. In the **Resource Configurations** area, set **AZ** to **Automatically allocated** or **Specifies AZ**.
 - After the scaling, nodes are randomly allocated to AZs.
 - If you select **Specifies AZ**, you can allocate nodes to different AZs.
5. Resizing the container engine space
When scaling out a resource pool, you can set the container engine space size of the created node. This operation will cause inconsistency of dockerBaseSize of nodes in the resource pool. Some tasks may run differently on different nodes. Exercise caution when performing this operation. The container engine space size cannot be changed for existing nodes.
6. Modify the OS. Select an OS version from the **OS** drop-down list.
7. Configure the node billing mode. When adding nodes, you can enable **Node Billing Mode** to set the billing mode and required duration for the new nodes, and enable the auto-renewal function. For example, you can create pay-per-use nodes in a yearly/monthly resource pool. If the billing mode is not specified, it will be the mode for charging resource pools.
8. Click **Submit** and then **OK**.

2.3.3 Upgrading the Standard Dedicated Resource Pool Driver

Description

If GPUs or Ascend resources are used in a dedicated resource pool, you may need to customize GPU or Ascend drivers. ModelArts allows you to upgrade GPU or Ascend drivers of your dedicated resource pools.

There are two driver upgrade modes: secure upgrade and forcible upgrade.

NOTE

- Secure upgrade: Running services are not affected. After the upgrade starts, the nodes are isolated (new jobs cannot be delivered). After the existing jobs on the nodes are complete, the upgrade is performed. The secure upgrade may take a long time because the jobs must be completed first.
- Forcible upgrade: The drivers are directly upgraded, regardless of whether there are running jobs.

Constraints

- The target dedicated resource pool must be running, and the resource pool contains GPU or Ascend resources.
- For a logical resource pool, the driver can be upgraded only after node binding is enabled. To enable node binding, submit a service ticket to contact engineers.

Upgrading the Driver

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.
2. Locate the target resource pool in the list and choose **More > Upgrade Driver** in the **Operation** column.
3. In the displayed dialog box, you can view the driver type, number of instances, current version, target version, upgrade mode, upgrade scope, and rolling switch of the dedicated resource pool.
 - **Target Version:** Select a target driver version from the drop-down list. The driver of the added nodes may not be that of the existing nodes. Select the current driver version for **Target Version**. After the upgrade, all nodes will be upgraded to the same version
 - Upgrade mode: You can select secure upgrade or forcible upgrade.
 - Secure upgrade: Perform the upgrade when no job is running on the node. The upgrade may take a long time.
 - Forcible upgrade: Ignore the running jobs and perform the upgrade directly. This may cause the running jobs to fail.
 - **Rolling Mode:** Once enabled, you can upgrade the driver in rolling mode. Currently, **By node percentage** and **By instance quantity** are supported.
 - **By node percentage:** The number of instances to be upgraded is the percentage multiplied by the total number of instances in the resource pool.
 - **By instance quantity:** The number of instances to be upgraded is the value of this parameter.

For different upgrade modes, the policies for upgrading nodes are different.

- If **Secure upgrade** is selected, the instances without services are upgraded.
- If **Forcible upgrade** is selected, random instances are upgraded.

NOTE

- To check whether a node has any service, go to the resource pool details page. In the **Nodes** tab, check whether all GPUs and Ascend chips are available. If yes, the node has no services.
 - During the rolling upgrade, the nodes with abnormal drivers do not affect the upgrade and will also be upgraded.
4. Click **OK** to start the driver upgrade.

2.3.4 Rectifying a Faulty Node in a Standard Dedicated Resource Pool

This section describes how to rectify the faulty nodes. Currently, you can replace and restart a node. During fault locating and performance diagnosis, certain O&M operations need to be authorized by users.

Handling Faulty Nodes

- Replacing a node: The node name will be changed and the original node will be deleted. On the resource pool details page, click the **Nodes** tab, you can replace a single node. To do so, locate the target node in the list and click **Replace** in the **Operation** column. No fee is charged for this operation.

Check the node replacement records on the **Records** page. **Running** indicates that the node is being replaced. After the replacement, you can check the new node in the node list.

The replacement can last no longer than 24 hours. If no suitable resource is found after the replacement times out, the status changes to **Failed**. Hover over  to check the failure cause.

NOTE

- The number of replacements per day cannot exceed 20% of the total nodes in the resource pool. The number of nodes to be replaced cannot exceed 5% of the total nodes in the resource pool.
 - Ensure that there are idle node resources. Otherwise, the replacement may fail.
 - If there are any nodes in the **Resetting** state in the operation records, nodes in the resource pool cannot be replaced.
- HA redundant node

An HA redundant node is the backup node in the dedicated resource pool. It can automatically replace a faulty common node to improve the Service Level Agreement (SLA) of the resource pool and prevent service loss. You can set the number of HA nodes based your requirements.

NOTICE

HA redundant nodes cannot be used for service running, which will affect the number of actual available nodes in the resource pool. When a resource pool delivers a task, select the number of actual available nodes with caution. If the HA redundant nodes are included, the task will keep waiting.

The running mechanism of HA redundant nodes are as follows:

- HA redundant nodes are isolated and cannot be scheduled by default. Workloads cannot be scheduled to the nodes.
- HA redundant nodes function as standby nodes and work with node fault detection to ensure that the faulty nodes are automatically replaced by normal nodes in a resource pool. The replacement normally takes only a few minutes. After the replacement, the original HA redundant nodes will be de-isolated and become normal nodes, and the faulty nodes will be labeled as HA redundant nodes. Rectify the fault for future automatic

switchover. After the faulty nodes are rectified, they become the new HA redundant nodes.

HA redundant nodes can free you from paying attention to node status and reduce O&M costs. However, you need to purchase backup nodes as HA redundant nodes. The resource costs are higher.

Setting HA nodes: You can set multiple HA redundant nodes in batches for a resource pool or set a single node as an HA redundant node.

- Configure a single node as HA redundant node.
 - Enabling HA redundancy
Select a node without any service as the HA redundant node. On the resource pool details page, click the **Nodes** tab, locate the target node, and click **More** > **Enable HA Redundancy** in the **Operation** column.
If you need to enable HA redundancy for nodes in batches, select the target nodes, and click **Enable HA Redundancy** above the list.

Figure 2-5 Enabling HA redundancy

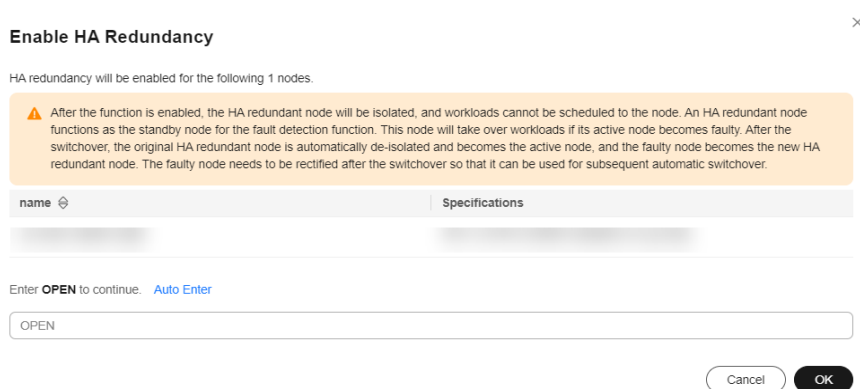


Figure 2-6 HA redundant node

HA Redundancy

✔ Enable

NOTE

- 5% of the total nodes in a resource pool should be HA redundant nodes. For example, there are 20 nodes in a resource pool, then there should be one HA redundant node.
- To check whether a node has any service, go to the resource pool details page. In the **Nodes** tab, check whether all GPUs and Ascend chips are available. If yes, the node has no services.

- Disabling HA redundancy
On the resource pool details page, click the **Nodes** tab. Locate the target node and click **More** > **Disable HA Redundancy** in the **Operation** column.

Once disabled, the nodes will be de-isolated and no longer used as standby nodes. Workloads can be scheduled to the nodes properly.

If you need to disable HA redundancy for nodes in batches, select the target nodes, and click **Disable HA Redundancy** above the list.

Figure 2-7 Disabling HA redundancy

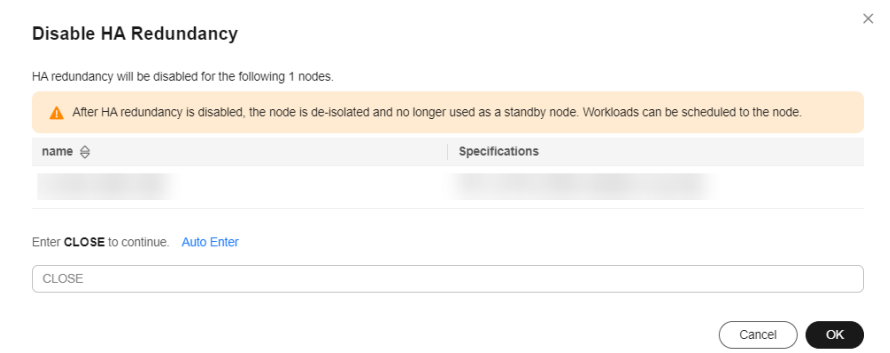


Figure 2-8 Non-HA redundancy



- Resetting a node: Perform this operation to upgrade the OS of the node. If an error occurs during node configuration update, reset the node to rectify the fault.

In the **Nodes** tab, locate the node you want to reset. Click **Reset** in the **Operation** column to reset a node. You can also select multiple nodes, and click **Reset** to reset multiple nodes.

Configure the parameters described in the table below.

Table 2-3 Parameters

Parameter	Description
Operating system	Select an OS from the drop-down list box.
Configuration Mode	Select a configuration mode for resetting the node. By node percentage: the maximum ratio of nodes that can be reset if there are multiple nodes in the reset task By node quantity: the maximum number of nodes that can be reset if there are multiple nodes in the reset task

Check the node reset records on the **Records** page. If the node is being reset, its status is **Resetting**. After the reset is complete, the node status changes to **Available**. Resetting a node will not be charged.

Figure 2-9 Resetting a node

Reset Node

⚠️ Resetting a node will affect the running of related services. Exercise caution when performing this operation.

You are about to reset the following 1 nodes.

Name	Status	Operating system
	✔️ Available	EulerOS 2.9

Operating system

EulerOS 2.9

Configuration Mode

By node percentage

By node quantity

Node Percentage

-

25

+

%

If you really want to reset 1 node, please enter **RESET**

RESET

Cancel

OK

NOTE

- Resetting a node will affect running services.
- Only nodes in the **Available** state can be reset.
- A single node can be in only one reset task at a time. Multiple reset tasks cannot be delivered to the same node at a time.
- If there are any nodes in the **Replacing** state in the operation records, nodes in the resource pool cannot be reset.
- When the driver of a resource pool is being upgraded, nodes in this resource pool cannot be reset.
- For GPU and NPU specifications, after the node is reset, the driver of the node may be upgraded. Wait patiently.

Figure 2-10 Operation records

Records

You can view task operation records below. Each record can be retained for a maximum of 90 days.

Apr 23, 2024 08:53:21 — Jul 23, 2024 08:53:21

Name/ID	Type	Status	Pool Name	Node Name	Description	Started	Ended
No data available.							

Close

- Restarting a node
On the resource pool details page, click the **Nodes** tab, locate the target node and choose **More > Reboot** in the **Operation** column. You can also select multiple nodes and click **Reboot** above the list to reboot multiple nodes.
When delivering a node restart task, select the corresponding nodes.
Restarting a node will affect running services. Exercise caution.

Check the node operation records in the **Records** dialog box. If the node is being restarted, its status is **Rebooting**. After it is restarted, the status changes to **Available**. Restarting a node will not be charged.

 NOTE

- Restarting a node will affect running services.
- Only nodes in the **Available** or **Unavailable** state can be restarted.
- A single node can be in only one restart task at a time. Multiple restart tasks cannot be delivered to the same node at a time.
- If the node is being replaced, reset, or deleted, it cannot be restarted.
- When the driver of a resource pool is being upgraded, nodes in this resource pool cannot be restarted.
- After the node is restarted, it will be unavailable for a short time. The service is being started and the health status is being checked. Wait patiently.
- Deleting or unsubscribing from a node
 - Click **Delete** in the **Operation** column.
To delete nodes in batches, select the check boxes next to the node names, and click **Delete**.
 - For a yearly/monthly resource pool whose resources are not expired, click **Unsubscribe** in the **Operation** column.
 - For a yearly/monthly resource pool whose resources are expired (in the grace period), click **Release** in the **Operation** column.

If the delete button is available for a yearly/monthly node, click the button to delete the node.

 NOTE

- Before deleting, unsubscribing from, or releasing a node, ensure that there are no running jobs on this node. Otherwise, the jobs will be interrupted.
- Delete, unsubscribe from, or release abnormal nodes in a resource pool and add new ones for substitution.
- If there is only one node, it cannot be deleted, unsubscribed from, or released.

Authorizing Technical Support to Locate Faults

During fault locating and performance diagnosis performed by Huawei technical support, you need to authorize certain O&M operations. To do so, go to the resource pool details page, click the **Nodes** tab, locate the target node, and click **More > Authorize** in the **Operation** column. In the displayed dialog box, click **OK**.

 NOTE

Normally, the **Authorize** button is unavailable. It will become available after the Huawei technical support applies for O&M.

After the O&M, Huawei technical support will disable the authorization. No further operations are required.

2.3.5 Modifying the Job Types Supported by a Standard Dedicated Resource Pool

Description

ModelArts supports many types of jobs. Some of them can run in dedicated resource pools, including training jobs, inference services, and notebook development environments.

You can change job types supported by a dedicated resource pool. Available options for **Job Type** are **Training Job**, **Inference Service**, and **DevEnviron**.

Only selected types of jobs can be delivered to the corresponding dedicated resource pool.

CAUTION

To support different job types, different operations are performed in the backend, such as installing plug-ins and setting the network environment. Some operations use resources in the resource pool. As a result, available resources for you decrease. Therefore, select only the job types you need to avoid resource waste.

Constraints

The target dedicated resource pool must be running.

Procedure

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.
2. Locate the target resource pool in the list and choose **More > Set Job Type** in the **Operation** column.
3. In the **Set Job Type** dialog box, select job types.
4. Click **OK**.

2.3.6 Configuring the Standard Dedicated Resource Pool to Access the Internet

Description

When you use a dedicated resource pool to create a job, for example, a training job, you can interconnect with VPC for the dedicated resource pool to access the Internet. In this case, the dedicated resource pool and the ECS with bound EIP are in the same VPC.

Prerequisites

- You have obtained the ECS where the SNAT function is to be deployed.
- The ECS where the SNAT function is to be deployed runs the Linux OS.

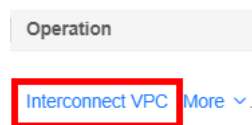
- The ECS where the SNAT function is to be deployed has only one network interface card (NIC) configured.

Step 1: Interconnect with the VPC

VPC interconnection allows you to use resources across VPCs, improving resource utilization.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**. Click the **Network** tab, locate the target network in the list and click **Interconnect VPC** in the **Operation** column.

Figure 2-11 Interconnect VPC



2. In the displayed dialog box, click the button on the right of **Interconnect VPC**, and select an available VPC and subnet from the drop-down lists.

NOTE

- The peer network to be interconnected cannot overlap with the current CIDR block.
- If no VPC is available, click **Create VPC** on the right to create a VPC.
 - If no subnet is available, click **Create Subnet** on the right to create a subnet.
 - To enable a dedicated resource pool to access the public network through a VPC, create a SNAT in the VPC, as the public network address is unknown. After the VPC is interconnected, by default, the public address cannot be forwarded to the SNAT of your VPC. Submit a service ticket and contact technical support to add a default route. Then, when you interconnect with a VPC, ModelArts **0.0.0.0/0** is used as the default route. In this case, you do not need to submit a service ticket. Add the default route for network configuration.

Step 2: Configure SNAT

Configure and verify SNAT by referring to [Using a Public NAT Gateway to Enable Servers to Share One or More EIPs to Access the Internet](#).

(Optional) Checking the Number of Available IP Addresses

Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**. Click the **Network** tab, locate the target network in the list and choose **More > View Available IP Addresses** in the **Operation** column.

On the resource pool details page, you can view the number of available IP addresses bound to the network.

2.3.7 Releasing Standard Dedicated Resource Pools and Deleting the Network

Deleting a Resource Pool

If a dedicated resource pool is no longer needed for AI service development, you can delete the resource pool to release resources.

NOTE

After a dedicated resource pool is deleted, the development environments, training jobs, and inference services that depend on the resource pool are unavailable. A dedicated resource pool cannot be restored after being deleted.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Dedicated Resource Pools > Elastic Clusters**.
2. Locate the target resource pool in the list and choose **More > Delete** in the **Operation** column.
3. In the **Delete Dedicated Resource Pool** dialog box, enter **DELETE** in the text box and click **OK**.

You can switch between tabs on the details page to view the training jobs and notebook instances created using the resource pool, and inference services deployed in the resource pool.

Deleting a Network

If a network is no longer needed for AI service development, you can delete the network.

NOTE

By deleting a network, the resource pool that uses the network will become unavailable. Exercise caution when performing this operation.

1. Go to the **Network** tab, locate the target network in the list, and choose **More > Delete** in the **Operation** column.
2. Confirm the information and click **OK**.

3 Development Environments

3.1 Application Scenarios

ModelArts provides flexible, open development environments. Select a development environment based on site requirements.

- In-cloud notebook, which is out of the box, relieving you from concerning environment installation or configuration. For details, see [Creating a Notebook Instance](#).
- ModelArts Notebook supports the following methods to develop AI models based on engines such as PyTorch, TensorFlow, and MindSpore:
 - Use JupyterLab to open a notebook instance. For details, see [Using a Notebook Instance for AI Development Through JupyterLab](#).
 - Local IDE for model development. After enabling remote SSH, you can remotely access the ModelArts notebook development environment to debug and run code from a local IDE. The local IDE allows you to use the in-cloud notebook development environment while with local coding habits unchanged.

A local IDE supports Visual Studio (VS) Code, PyCharm, and SSH. Additionally, PyCharm Toolkit and VS Code Toolkit are provided for convenient remote access. For details, see [Using Notebook Instances Remotely Through PyCharm](#), [Using Notebook Instances Remotely Through VS Code](#), and [Using a Notebook Instance Remotely with SSH](#).

- Easy and fast file uploading is a common requirement in AI development. ModelArts allows you to upload files in multiple ways. You can view the upload progress and speed during the uploading.
 - [Upload local files](#).
 - [Upload GitHub open-source repository files](#).
 - [Upload OBS files](#).
 - [Upload remote files such as open-source datasets](#).
- When using a notebook instance, you can quickly obtain target instances and switch images in the same instance. You can flexibly adjust the specifications by adjusting the AI engine and switching node runtime specifications. For

users who just start to use ModelArts notebook, choose a rather small storage at first. After the notebook instance is created, scale out EVS as needed. You can also view events to locate faults when a notebook instance is faulty. For details, see [Managing Notebook Instances](#).

- ModelArts CLI (ma-cli) is integrated in the notebook instance to interconnect with ModelArts and run management commands on ModelArts resources. ma-cli allows you to interact with cloud services in ModelArts notebook instances and offline VMs. You can run ma-cli commands to implement automatic command completion, authentication, image building, ModelArts training job submission, DLI Spark job submission, and OBS data replication. For details, see [ModelArts CLI Command Reference](#).
- ModelArts Notebook has a built-in MoXing Framework module. ModelArts mox.file provides a set of APIs for accessing OBS more conveniently, allowing users to operate OBS files by simulating a series of APIs for operating the local file system. For details, see [Using Moxing Commands in a Notebook Instance](#).

3.2 Creating a Notebook Instance

Before developing a model, create a notebook instance and access it for coding.

Procedure

1. Log in to the ModelArts management console. In the navigation pane, choose **Settings** and check whether the access authorization has been configured. If not, configure access authorization. For details, see [Configuring Agency Authorization for ModelArts with One Click](#).
2. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
3. Click **Create Notebook** in the upper right corner. On the displayed page, configure the parameters.
 - a. Configure the basic information of the notebook instance, including its name, description, and auto stop status. For details, see [Table 3-1](#).

Table 3-1 Basic parameters

Parameter	Description
Name	Name of the notebook instance, which is automatically generated by the system. You can rename it based on service requirements. A name consists of a maximum of 128 characters and cannot be empty. It can contain only digits, letters, underscores (_), and hyphens (-).
Description	Brief description of the notebook instance

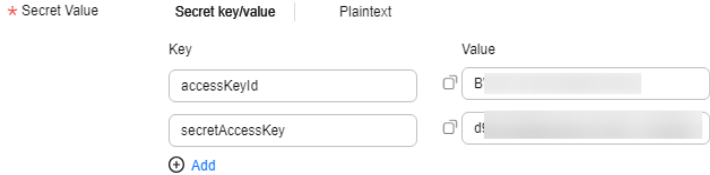
Parameter	Description
Auto Stop	Automatically stops the notebook instance at a specified time. This function is enabled by default. The default value is 1 hour, indicating that the notebook instance automatically stops after running for 1 hour. The options are 1 hour, 2 hours, 4 hours, 6 hours, and Custom. You can select Custom to specify any integer from 1 to 72 hours. • Stop as scheduled: If this option is enabled, the notebook instance automatically stops when the running duration exceeds the specified duration. NOTE To protect in-progress jobs, a notebook instance does not automatically stop immediately at the auto stop time. Instead, there is a period of 2 to 5 minutes (during which you will still be charged) provided for you to renew the auto stop time.

- b. Configure notebook parameters, such as the image and instance flavor. For details, see [Table 3-2](#).

Table 3-2 Notebook instance parameters

Parameter	Description
Image	Public and private images are supported. • Public images are the AI engines built in ModelArts. • Private images can be created using an instance that is created using a public image. For details, see Saving a Notebook Instance. You can also create a custom image using preset or third-party images. For details, see Creating a Custom Image. An image corresponds to an AI engine. When you select an image during instance creation, the AI engine is specified accordingly. Select an image as required. Enter a keyword of the image name in the search box on the right to quickly search for the image. You can change an image on a stopped notebook instance.
Resource Type	Public and dedicated resource pools are available for you to select. NOTE If the dedicated resource pool you purchased is a single-node Tnt004 pool whose specification is GPU: 1*tnt004 CPU: 8 vCPUs and 32 GiB (modelarts.vm.gpu._tnt004u8), when you use the cluster to create a notebook instance, the Tnt004 card is idle but is displayed as sold out or the creation fails due to insufficient resources, contact technical support.

Parameter	Description
Type	Processor type, which can be CPU or GPU . The chips vary depending on the selected image.
Instance Specifications	The flavor of your notebook instance. Select a flavor based on your needs.

Parameter	Description
Storage	NOTE OBS and PFS are whitelist functions. If you have trial requirements, submit a service ticket to apply for permissions. • EVS Set a disk size based on service requirements. The default value is 5 GB. The maximum disk size is displayed on the GUI. The EVS disk space is charged by GB from the time the notebook instance is created to the time the notebook instance is deleted. • SFS Select this type only for a dedicated resource pool. SFS takes effect only after a dedicated resource pool can communicate with your VPC. For details, see Creating a Network. – Scalable File Service: Select a created SFS Turbo file system. To create an SFS Turbo file system, log in to . – Cloud Mount Path: Retain the default value /home/ma-user/work/. – Mounted Subdirectory: Select the storage path on SFS Turbo. – Mount Method: This parameter is displayed when the folder control permission is granted for the user. The read/write or read-only permission is displayed based on the storage path on SFS Turbo. • The value can be OBS or PFS. Storage Path: Set the OBS path for storing notebook data. If you want to use existing files or data, upload them to the specified OBS path. Storage Path must be set to a specific directory in an OBS bucket rather than the root directory of the OBS bucket. Secret: Select an existing secret or click Create on the right to create one. On the displayed DEW console, . Enter accessKeyId and secretAccessKey under Key, and enter the AKs/SKs obtained from My Credentials > Access Keys under Value. Figure 3-1 Configuring the secret values  EVS and SFS are all mounted to the /home/ma-user/work directory.

Parameter	Description
	The data is retained in <code>/home/ma-user/work</code>, even if the notebook instance is stopped or restarted. When a notebook instance is deleted, the EVS storage is released and the stored data is not retained. SFS can be mounted to a new notebook instance and data can be retained.
Remote SSH	- After you enable this function, you can remotely access the development environment of the notebook instance from your local development environment. - When a notebook instance is stopped, you can update the SSH configuration on the instance details page. NOTE The notebook instances with remote SSH enabled have VS Code plug-ins (such as Python and Jupyter) and the VS Code server package pre-installed, which occupy about 1 GB persistent storage space.
Key Pair	Set a key pair after remote SSH is enabled. Select an existing key pair. Alternatively, click Create on the right of the text box to create one on the DEW console. On the displayed Create Account Key Pair page, configure the parameters. After a notebook instance is created, you can change the key pair on the instance details page. CAUTION Download the created key pair and properly keep it. When you use a local IDE to remotely access the notebook development environment, the key pair is required for authentication.

4. Click **Next**.

5. After confirming the parameter settings, click **Submit**.

Switch to the notebook instance list. The notebook instance is being created. It will take several minutes before its status changes to **Running**. Then, the notebook instance is created.

6. In the notebook instance list, click the instance name. On the instance details page that is displayed, view the instance configuration.

If **Remote SSH** is enabled, you can click the modification icon on the right of the whitelist to modify it. You can click the modification icon on the right of **Authentication** to update the key pair of a stopped notebook instance.

If an EVS disk is used, click **Expansion** on the right of **Storage Capacity** to dynamically expand the EVS disk capacity. For details, see [Dynamically Expanding EVS Disk Capacity](#).

Accessing a Notebook Instance

Access a notebook instance in the **Running** state for coding.

- Online access: Use JupyterLab. For details, see [Using a Notebook Instance for AI Development Through JupyterLab](#).
- Remotely accessed from a local IDE through PyCharm. For details, see [Using Notebook Instances Remotely Through PyCharm](#).
- Remotely accessed from a local IDE through VS Code. For details, see [Using Notebook Instances Remotely Through VS Code](#).
- Remotely accessed from a local IDE through SSH. For details, see [Using a Notebook Instance Remotely with SSH](#).

A ModelArts notebook instance is started as user **ma-user**. The default working directory of the instance is **/home/ma-user**.

```
sh-4.4$pwd
/home/ma-user
sh-4.4$
```

Mounting Directories of Notebook Containers

When you use EVS storage when creating a notebook instance, the **/home/ma-user/work** directory is used as the workspace for persistent storage.

The data stored in only the **work** directory is retained after the instance is stopped or restarted. When you use a development environment, store the data for persistence in **/home/ma-user/work**.

For details about directory mounting of a notebook instance, see [Table 3-3](#). The following mounting points are not saved when images are saved.

Table 3-3 Mounting directories

Mount Point	Read Only	Remarks
/home/ma-user/work/	No	Persistent directory of your data
/data	No	Mount directory of your PFS
/cache	No	Used to mount the hard disk of the host NVMe (supported by bare metal specifications)
/train-worker1-log	No	Compatible with training job debugging
/dev/shm	No	Used for PyTorch engine acceleration

Selecting Storage for a Notebook Instance

Storage varies depending on performance, usability, and cost. No storage media can cover all scenarios. Learn about in-cloud storage application scenarios for better usage.

Table 3-4 In-cloud storage application scenarios

Storage	Application Scenario	Advantage	Disadvantage
EVS	Data and algorithm exploration only in the development environment.	Block storage SSDs feature better overall I/O performance than NFS. The storage capacity can be dynamically expanded to up to 4096 GB. As persistent storage, EVS disks are mounted to /home/ma-user/work. The data in this directory is retained after the instance is stopped. The storage capacity can be expanded online based on demand.	This type of storage can only be used in a single development environment.

Storage	Application Scenario	Advantage	Disadvantage
Parallel File System (PFS)	NOTE - PFS is a whitelist function. To use this function, contact technical support. - Only OBS PFS in the same region can be mounted. PFS buckets mounted as persistent storage for AI development and exploration. - Storage for datasets. Datasets stored in the PFS buckets are directly mounted to notebooks for browsing and data processing and can be directly used during training. Select PFS when creating a notebook instance. After the instance is running, the OBS parallel file system that carries the datasets is dynamically mounted to the notebook instances. For details, see Dynamically Mounting an OBS Parallel File System. - Storage for code. After debugging on a notebook instance, specify the OBS path as the code path for starting training, facilitating temporary modification.	PFS is an optimized high-performance object storage file system with low storage costs and large throughput. It can quickly process high-performance computing (HPC) workloads. PFS mounting is recommended if OBS is used. NOTE Package or split the data to be uploaded by 128 MB or 64 MB. Download and decompress the data in local storage for better I/O and throughput performance.	Due to average performance in frequent read and write of small files, PFS storage is not suitable for large model training or file decompression. NOTE Before mounting PFS storage to a notebook instance, grant ModelArts with full read and write permissions on the PFS bucket. The policy will be retained even after the notebook instance is deleted.

Storage	Application Scenario	Advantage	Disadvantage
	3. Storage for checking training. Mount storage to the training output path such as the path to training logs. In this way, view and check training on the notebook instance in real time. This is especially suitable for analyzing the output of jobs trained using TensorBoard .		
OBS	NOTE • OBS is a whitelist function. To use this function, contact technical support. • Only OBS objects in the same region can be mounted. When uploading or downloading a large amount of data in the development environment, you can use OBS buckets to transfer data.	Low storage cost and high throughput, but average performance in reading and writing small files. It is a good practice to package or split the file by 128 MB or 64 MB. In this way, you can download the packages, decompress them, and use them locally.	The object storage semantics is different from the Posix semantics and needs to be further understood.

Storage	Application Scenario	Advantage	Disadvantage
Scalable File Service (SFS)	Available only in dedicated resource pools. Use SFS storage in informal production scenarios such as exploration and experiments. One SFS device can be mounted to both a development environment and a training environment. In this way, you do not need to download data each time your training job starts. This type of storage is not suitable for heavy I/O training on more than 32 cards.	SFS is implemented as NFS and can be shared between multiple development environments and between development and training environments. This type of storage is preferred for non-heavy-duty distributed training jobs, especially for the ones not requiring to download data additionally when the training jobs start.	The performance of the SFS storage is not as good as that of the EVS storage.
OceanStor Pacific storage (SFS capacity-oriented 2.0)	Currently, it can be used only in Tiangong resource pools. It is suitable for the training jobs that use the file systems provided by SFS capacity-oriented 2.0 for AI model training and exploration. In addition, OBS APIs are provided to import training data from outside the cloud.	It provides a high-performance file client to meet the high storage bandwidth requirements of heavy-load training jobs. It also supports access to OBS. After the training data is imported to the storage via OBS APIs, it can be used to train the model directly without any conversion.	It provides the object storage semantics, which is different from the Posix semantics and needs to be further understood.

Storage	Application Scenario	Advantage	Disadvantage
Local storage	First choice for heavy-duty training jobs.	High-performance SSDs for the used VM or BMS, featuring high file I/O throughput. For heavy-duty training jobs, store data in the target directory and then start training. By default, the storage is mounted to the /cache directory. For details about the available space of the /cache directory, see What Are Sizes of the /cache Directories for Different Notebook Specifications in DevEnviron?	The storage lifecycle is associated with the container lifecycle. Data needs to be downloaded each time the training job starts.

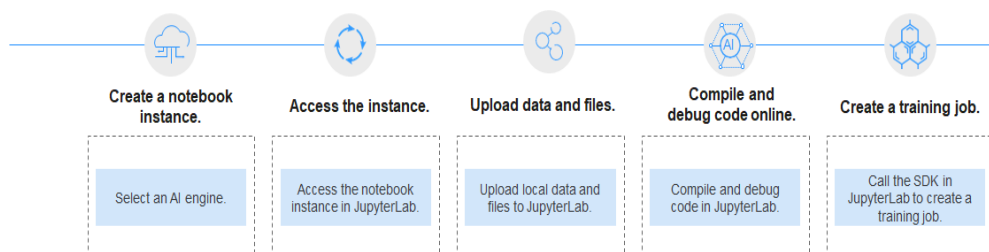
3.3 Using a Notebook Instance for AI Development Through JupyterLab

3.3.1 Using JupyterLab to Develop and Debug Code Online

JupyterLab is an interactive development environment, enabling you to compile notebooks, operate terminals, edit Markdown text, enable interaction, and view CSV files and images. JupyterLab is the future mainstream development environment for developers.

ModelArts allows you to access notebook instances online using JupyterLab and develop AI models based on the PyTorch, TensorFlow, or MindSpore engines. [Figure 3-2](#) shows the operation process.

Figure 3-2 Using JupyterLab to develop and debug code online

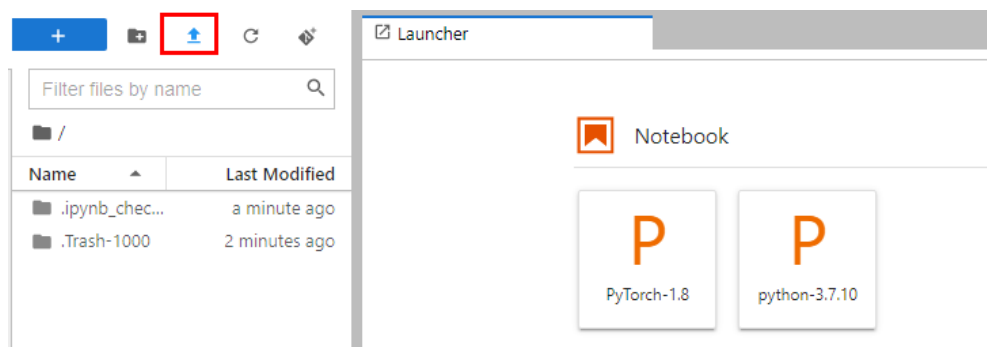


Procedure

1. Create a notebook instance.
On the ModelArts management console, create a notebook instance with a proper AI engine. For details, see [Creating a Notebook Instance](#).

2. After the notebook instance is created, it is in the **Running** state. Locate it in the list and click **Open** in the **Operation** column to access JupyterLab.
3. The **Launcher** page is automatically displayed. Perform required operations. For details, see [JupyterLab Documentation](#).
4. Upload training data and code files to JupyterLab. For details, see [Uploading Files from a Local Path to JupyterLab](#).

Figure 3-3 Button for uploading a file

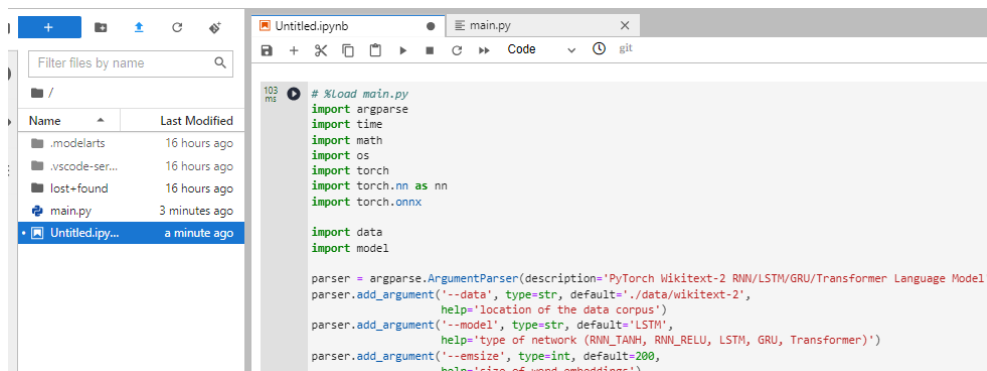


5. In the navigation pane on the left, double-click the uploaded code file, compile the file in JupyterLab, and debug it. For details about how to use JupyterLab, see [Common Functions of JupyterLab](#).

NOTE

If your code file is in .py format, open a new .ipynb file and run the `%load main.py` command to load the content of the .py file to the .ipynb file for encoding and debugging.

Figure 3-4 Opening a code file



6. In JupyterLab, call the ModelArts SDK to create a training job for in-cloud training.
For details, see [Creating a Training Job](#).

3.3.2 Common Functions of JupyterLab

Introduction

To access JupyterLab from a running notebook instance, perform the following operations:

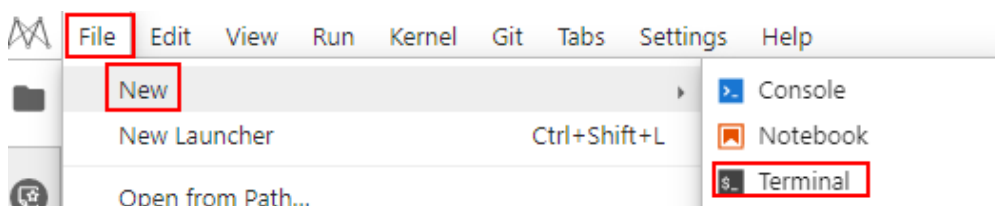
1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. Click **Open** in the **Operation** column of a running notebook instance to access JupyterLab.
3. The **Launcher** page is automatically displayed. Perform required operations. For details, see [JupyterLab Documentation](#).
 - **Notebook**: Select a kernel for running notebook, for example, TensorFlow or Python.
 - **Console**: Call the terminal for command control.
 - **Other**: Edit other files.

Creating a Terminal in JupyterLab

You can run Python commands on the terminal to operate the terminal. The following describes how to open the terminal of JupyterLab.

1. Create a notebook instance. When the instance is running, click **Open** in the **Operation** column. The **JupyterLab** page is displayed.
2. Choose **File > New > Terminal**. The **Terminal** page is displayed.

Figure 3-5 Terminal



3. You can use **pip** to install external libraries in the **TensorFlow-1.8** environment on the **Terminal** page. For example, to install Shapely:
Enter the following commands in the code input box to obtain the kernel of the current environment and activate the Python environment on which the installation depends:

```
cat /home/ma-user/README
```

```
source /home/ma-user/anaconda3/bin/activate TensorFlow-1.8
```

NOTE

To install TensorFlow in another Python environment, replace **TensorFlow-1.8** in the command with the target engine.

Figure 3-6 Activating the environment

```
sh-4.3$cat /home/ma-user/README
Please use one of following command to start the specified framework environment.

for Conda-python3 ----- source /home/ma-user/anaconda3/bin/activate base
for MXNet-1.2.1 ----- source /home/ma-user/anaconda3/bin/activate MXNet-1.2.1
for PySpark-2.3.2 ----- source /home/ma-user/anaconda3/bin/activate PySpark-2.3.2
for Pytorch-1.0.0 ----- source /home/ma-user/anaconda3/bin/activate Pytorch-1.0.0
for TensorFlow-1.13.1 ----- source /home/ma-user/anaconda3/bin/activate TensorFlow-1.13.1
for TensorFlow-1.8 ----- source /home/ma-user/anaconda3/bin/activate TensorFlow-1.8
for XGBoost-Sklearn ----- source /home/ma-user/anaconda3/bin/activate XGBoost-Sklearn
```

Run the following command in the code input box to install Shapely:

```
pip install Shapely
```

Creating an IPYNB File in JupyterLab

On the JupyterLab homepage, click a proper AI engine in the **Notebook** area to create an IPYNB file.

The AI engines supported by each notebook instance vary depending on the runtime environment. The following figure is only an example. Select an AI engine based on site requirements.

The created IPYNB file is displayed in the navigation pane on the left.

Creating a Notebook File and Accessing the Console

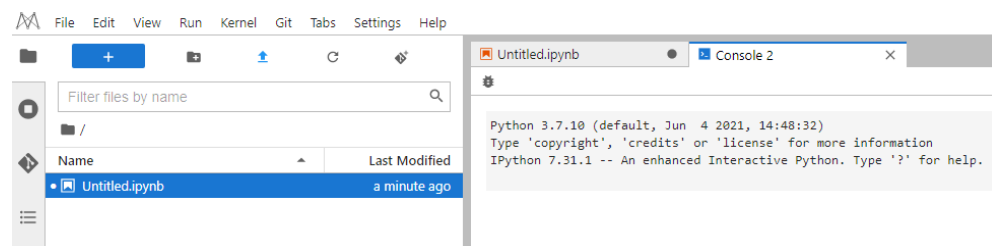
A console is a Python terminal, which is similar to the native IDE of Python, displaying the output after a statement is entered.

On the JupyterLab homepage, click a proper AI engine in the **Console** area to create a notebook file.

The AI engines supported by each notebook instance vary depending on the runtime environment. The following figure is only an example. Select an AI engine based on site requirements.

After the file is created, the console page is displayed.

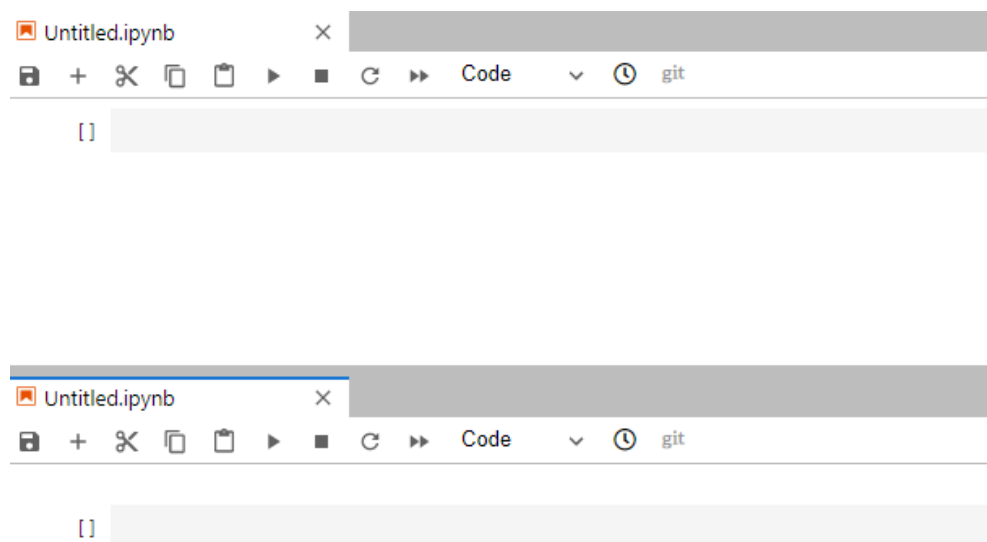
Figure 3-7 Creating a notebook file (console)



Editing a File in JupyterLab

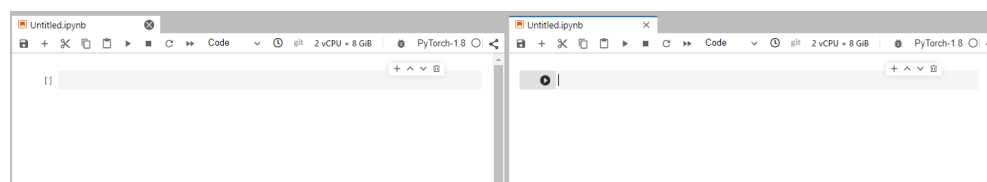
JupyterLab allows you to open multiple notebook instances or files (such as HTML, TXT, and Markdown files) in one window and displays them on different tab pages.

In JupyterLab, you can customize the display of multiple files. In the file display area on the right, you can drag a file to adjust its position. Multiple files can be concurrently displayed.

Figure 3-8 Customized display of multiple files

When writing code in a notebook instance, you can create multiple views of a file to synchronously edit the file and view execution results in real time.

To open multiple views, open an IPYNB file and choose **File > New View for Notebook**.

Figure 3-9 Multiple views of a file

Before coding in the code area of an IPYNB file in JupyterLab, add an exclamation mark (!) before the code.

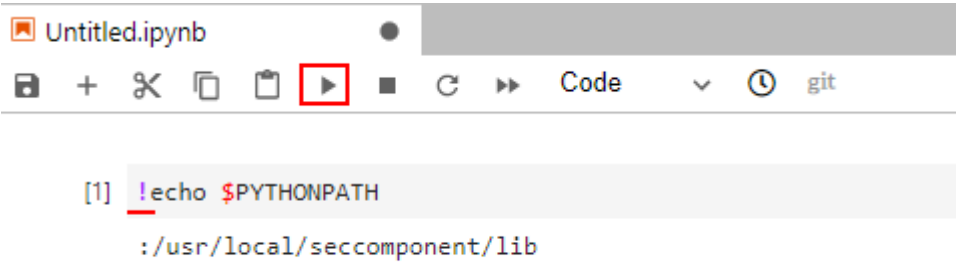
For example, install an external library Shapely.

```
!pip install Shapely
```

For example, obtain PythonPath.

```
!echo $PYTHONPATH
```

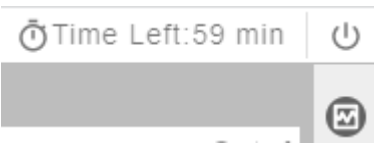
Figure 3-10 Running code



Renewing or Automatically Stopping a Notebook Instance

If you enable auto stop when you created or started a notebook instance, the remaining duration for stopping the instance is displayed in the upper right corner of JupyterLab. You can click the time for renewal.

Figure 3-11 Remaining duration



Common JupyterLab Buttons and Plug-ins

Figure 3-12 Common JupyterLab buttons and plug-ins

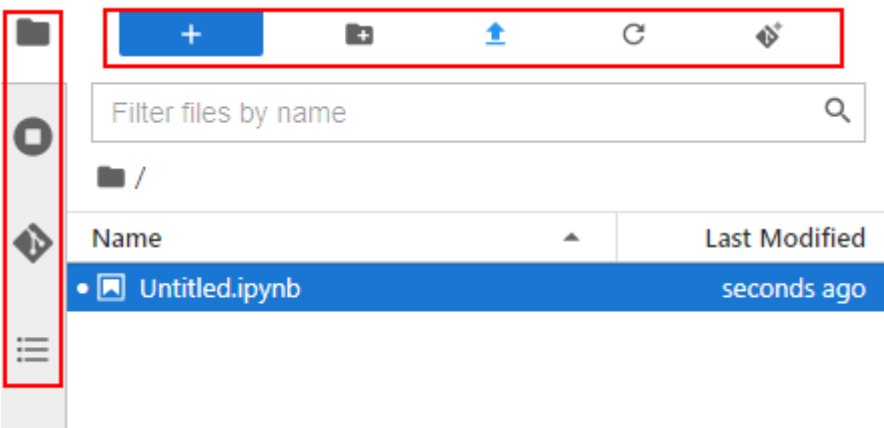


Table 3-5 JupyterLab buttons

Button	Description
	Quickly open notebook instances and terminals. Open the Launcher page, on which you can quickly create notebook instances, consoles, or other files.
	Create a folder.

Button	Description
	Upload files.
	Refresh the file directory.
	Git plug-in, which can be used to access the GitHub code library associated with the notebook instance.

Table 3-6 JupyterLab plug-ins

Plug-in	Description
	List files. Click this button to show all files in the notebook instance.
	Display the terminals and kernels that are running in the current instance.
	Git plug-in, which can be used to quickly access the GitHub code library.
	Property inspector.
	Show the document organization.

Figure 3-13 Buttons in the navigation bar**Table 3-7** Buttons in the navigation bar

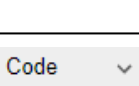
Button	Description
File	Actions related to files and directories, such as creating, closing, or saving notebooks.
Edit	Actions related to editing documents and other activities in the IPYNB file, such as undoing, redoing, or cutting cells.
View	Actions that alter the appearance of JupyterLab, such as showing the bar or expanding code.
Run	Actions for running code in different activities such as notebooks and code consoles.

Button	Description
Kernel	Actions for managing kernels, such as interrupting, restarting, or shutting down a kernel.
Git	Actions on the Git plug-in, which can be used to quickly access the GitHub code library.
Tabs	A list of the open documents and activities in the dock panel.
Settings	Common settings and an advanced settings editor.
Help	A list of JupyterLab and kernel help links.

Figure 3-14 Buttons in the menu bar of an IPYNB file



Table 3-8 Buttons in the menu bar of an IPYNB file

Button	Description
	Save a file.
	Add a new cell.
	Cut the selected cell.
	Copy the selected cell.
	Paste the selected cell.
	Execute the selected cell.
	Terminate a kernel.
	Restart a kernel.
	Restart a kernel and run all code of the current notebook again.
	There are four options in the drop-down list: Code (Python code), Markdown (Markdown code, typically used for comments), Raw (a conversion tool), and - (not modified)

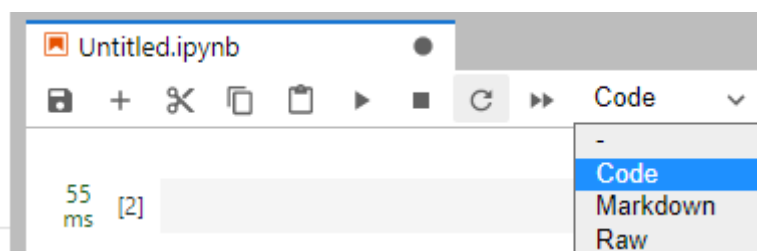
Button	Description
	View historical code versions.
	Git plug-in. The gray button indicates that the plug-in is unavailable in the current region.
2 vCPU + 4 GiB	Instance flavor.
PyTorch-1.4	Kernel for you to select.
	Code running status.  indicates the code is being executed.

Using Code-based Plug-ins

The code parametrization plug-in simplifies notebook cases. You can quickly adjust parameters and train models based on notebook cases without complex code. This plug-in can be used to customize notebook cases for competitions and learning.

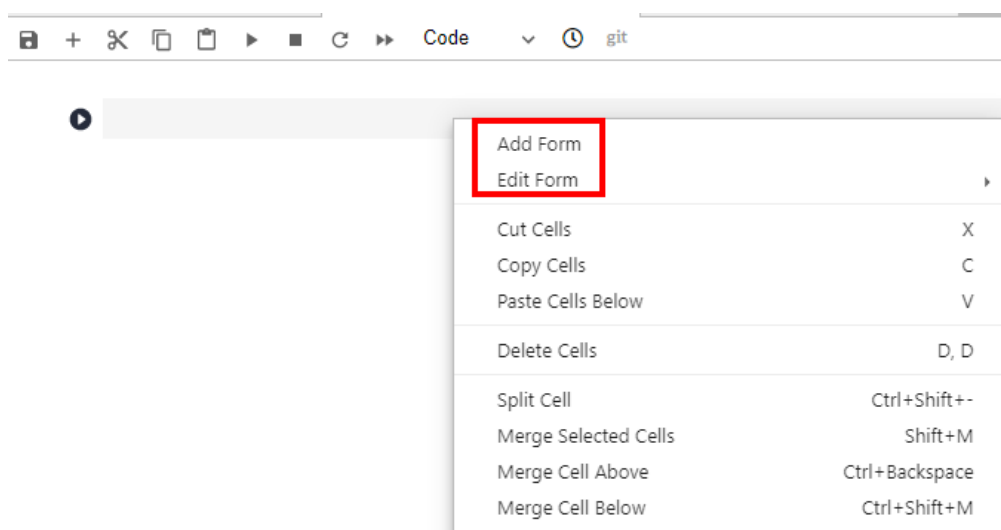
- The **Add Form** and **Edit Form** buttons are available only to the shortcut menu of code cells, as shown below.

Figure 3-15 Viewing a code cell



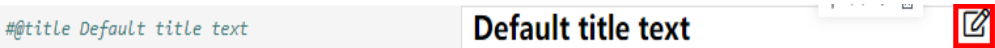
- After opening new code, add a form before editing it.

Figure 3-16 Shortcut menu of code cells



- If you click **Add Form**, a code cell will be split into the code and form edit area. Click **Edit** on the right of the form to change the default title.

Figure 3-17 Two edit areas



- If you click **Edit Form**, four sub-options will be displayed: **Add new form field**, **Hide code**, **Hide form**, and **Show All**.

Table 3-9 Edit Form sub-options

Edit-Form Sub-option	Description
Add new form field	• The form field types include dropdown, input, and slider, as shown in Figure 3-18. Each time a field is added, the corresponding variable is added to the code and form areas. If a value in the form area is changed, the corresponding variable in the code area is also changed. NOTE When creating a dropdown form, click ADD Item and add at least two items, as shown in Figure 3-19. • If the form field type is set to dropdown, the supported variable types are raw and string. • If the form field type is set to input, the supported variable types are boolean, date, integer, number, raw, and string. • If the form field type is set to slider, the minimum value, maximum value, and step can be set.
Hide code	Hide the code.
Hide form	Hide the forms.
Show all	Display both code and forms.

Figure 3-18 Form field types

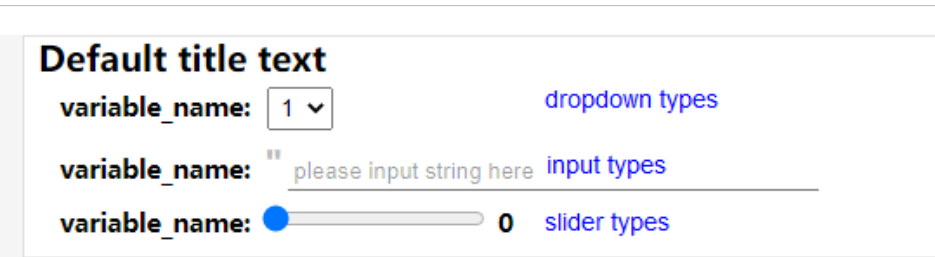


Figure 3-19 Creating a dropdown form

Add new form field

Form field type: dropdown

Variable type: string

Variable name: variable_name

Add Item

option1 input a value

option2 input a value

Cancel Save

Figure 3-20 Deleting a form

Default title text

variable_name1: bb

variable_name2: [1, 2, 3]

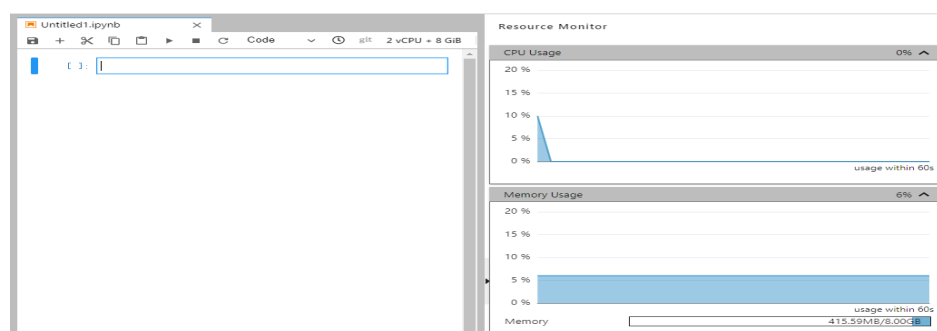
variable_name3: {'1': 'a', 'b': 2}

variable_name4: (1, 2, 3)

Monitoring Resources

To obtain resource usage, select **Resource Monitor** in the right pane. The CPU usage and memory usage can be viewed.

Figure 3-21 Resource usage



3.3.3 Using Git to Clone the Code Repository in JupyterLab

In JupyterLab, you can use the Git plug-in to clone the GitHub open-source code repository, quickly view and edit data, and submit the modified data.

Prerequisites

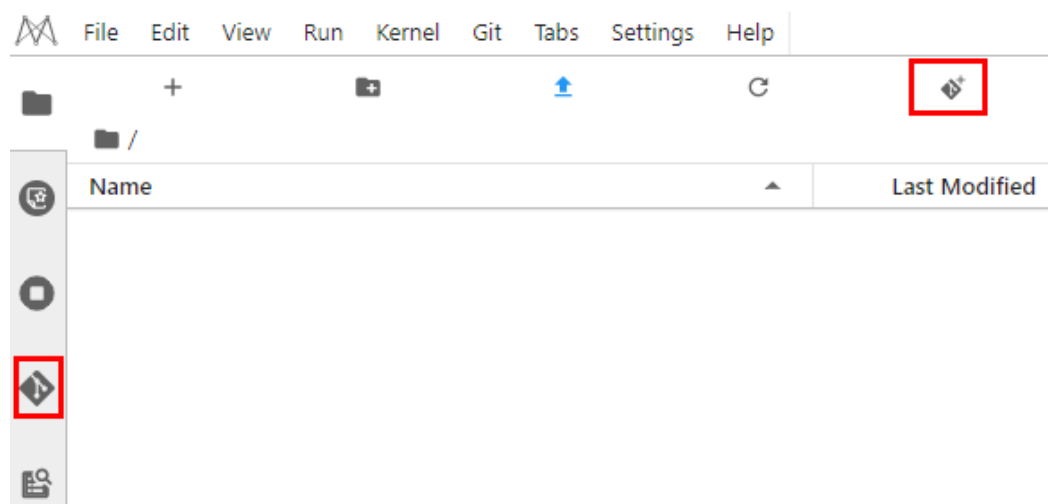
The notebook instance is running.

Starting the Git Plug-in of JupyterLab

In the notebook instance list, locate the target instance and click **Open** in the **Operation** column to go to the JupyterLab page.

Figure 3-22 shows the Git plug-in of JupyterLab.

Figure 3-22 Git plug-in



Cloning a GitHub Open-Source Code Repository

Access a GitHub open-source code repository at <https://github.com/jupyterlab/extension-examples>. Click , enter the repository address, and click **OK** to start cloning. After the cloning is complete, the code library folder is displayed in the navigation pane of JupyterLab.

Cloning a GitHub Private Code Repository

1. When you clone a GitHub private code repository, a dialog box will be displayed, asking you to enter your personal credentials. In this case, enter the personal access token in GitHub.

Git credentials required

Enter credentials for remote repository

2. To obtain a personal access token, perform the following operations:
 - a. Log in to [GitHub](#) and open the configuration page.
 - b. Click **Developer settings**.
 - c. Choose **Personal access tokens > Generate new token**.
 - d. Verify the login account.
 - e. Describe the token, select permissions to access the private repository, and click **Generate token** to generate a token.
 - f. Copy the generated token to CloudBuild.

NOTICE

- Save the token securely once it is generated. It will be unavailable after you refresh the page. If it is not obtained, generate a new token.
- Enter a valid token description so that it can be easily identified. If the token is deleted by mistake, the building will fail.
- Delete the token when it is no longer used to prevent information leakage.

Figure 3-23 Cloning a GitHub private code repository (only authorization using a personal access token is supported)

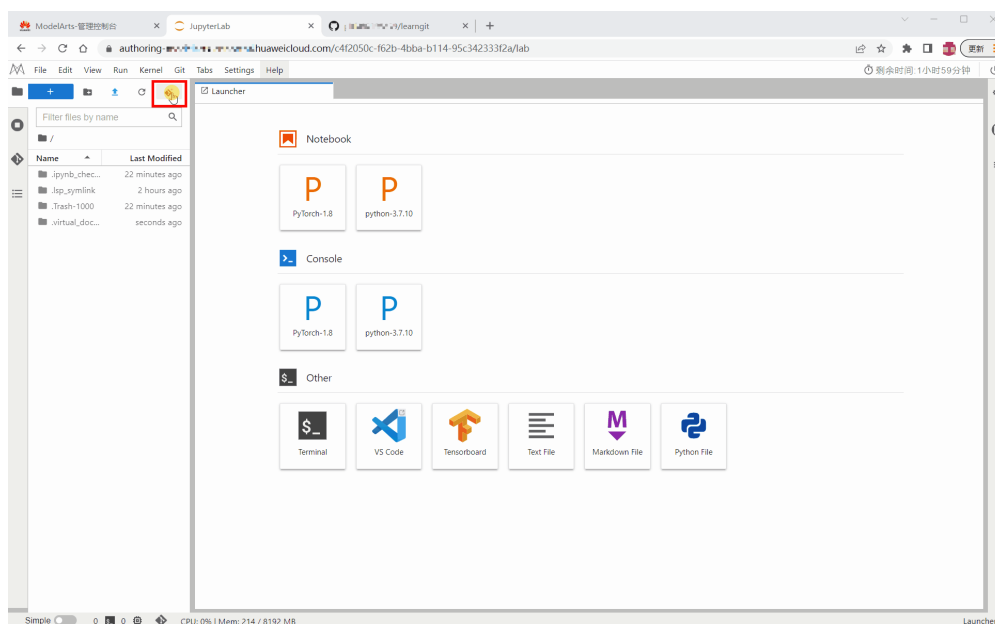
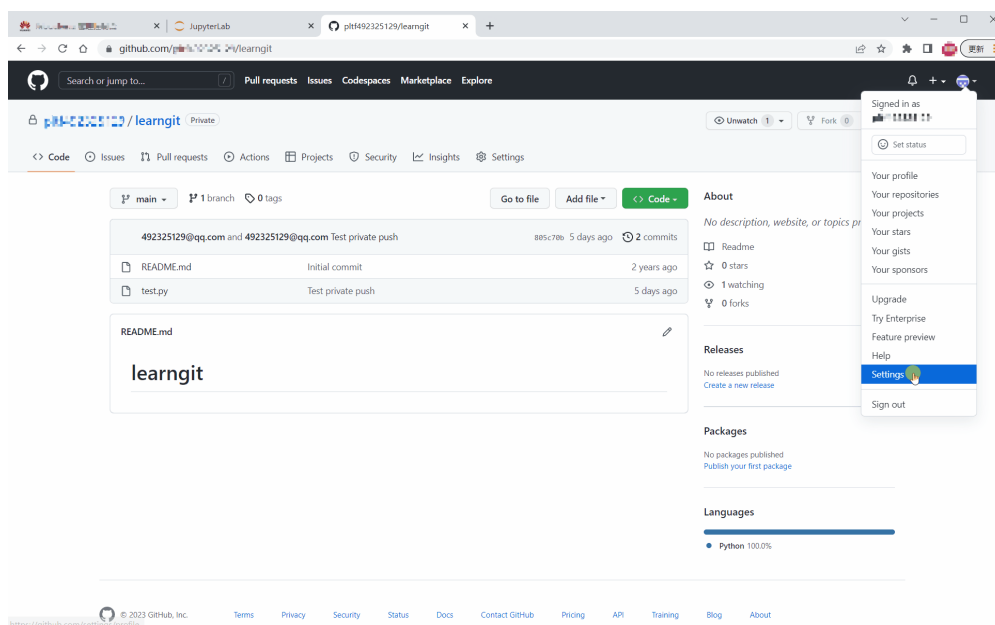
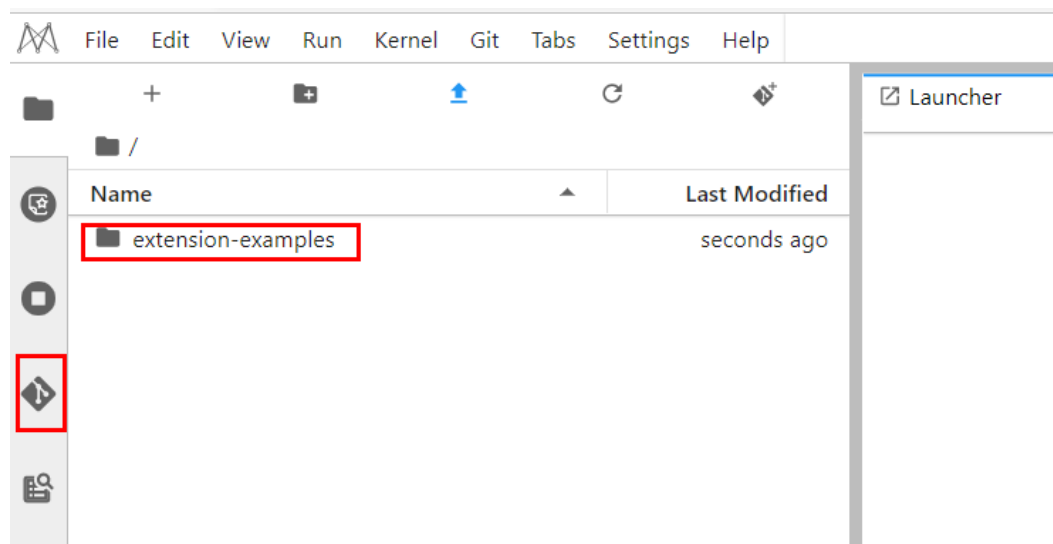


Figure 3-24 Obtaining a personal access token

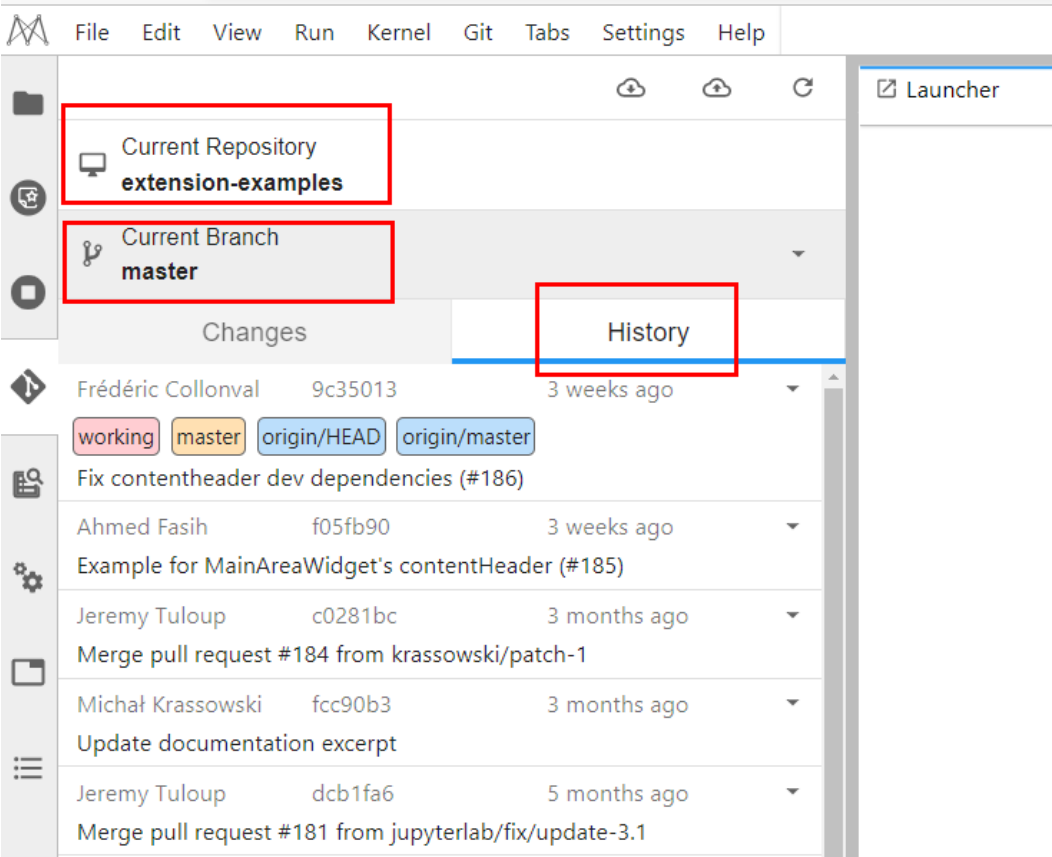
Viewing a Code Repository

In the list under **Name**, double-click the folder you want to use and click the Git plug-in icon on the left to access the code repository corresponding to the folder.

Figure 3-25 Opening the folder and starting the Git plug-in

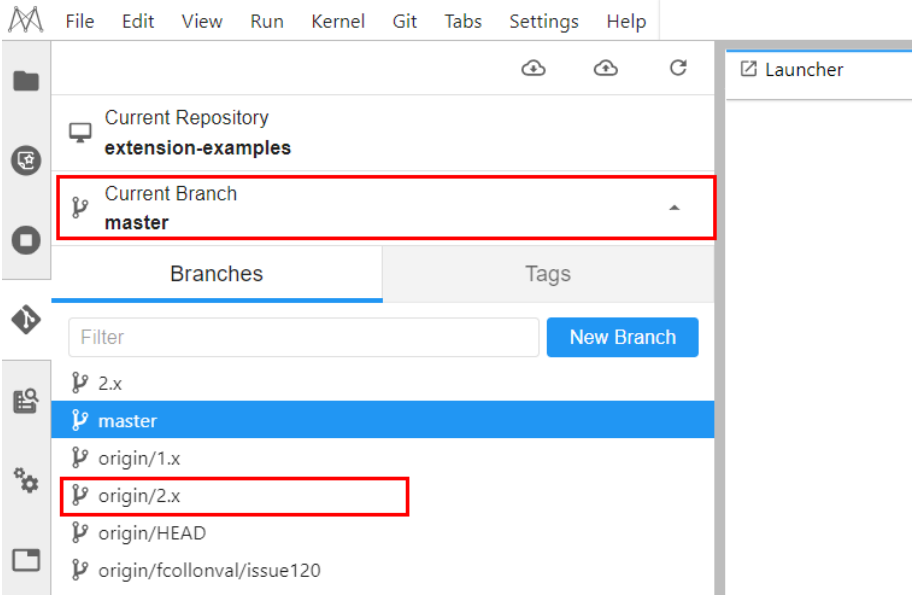
You can view the information current code repository, such as the repository name, branch, and historical submission records.

Figure 3-26 Viewing a code repository



NOTE

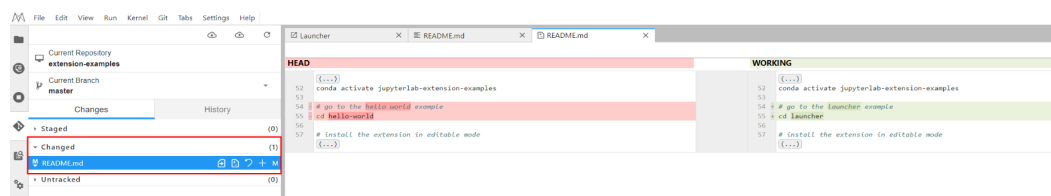
By default, the Git plug-in clones the master branch. To switch another branch, click **Current Branch** to expand all branches and click the target branch name.



Viewing Modifications

If a file in the code repository has been modified, you can view the modified file under **Changed** on the **Changes** tab page. Click **Diff this file** on the right of the file name to view the modifications.

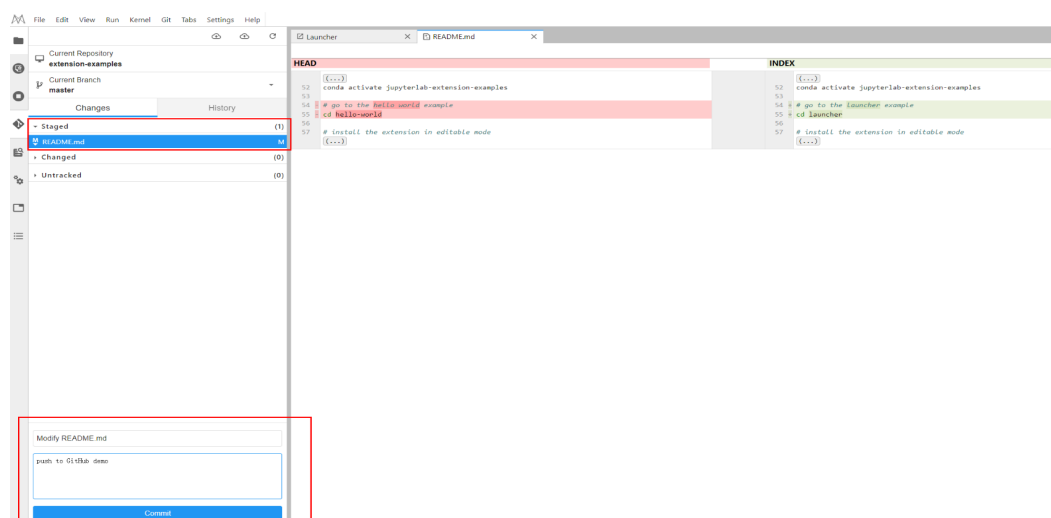
Figure 3-27 Viewing modifications



Committing Modifications

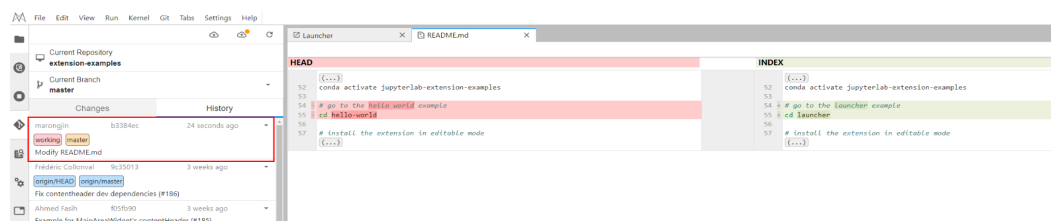
After confirming that the modifications are correct, click **Stage this change** on the right of the file name, which is equivalent to running the **git add** command. The file enters the **Staged** state. Enter the message to be committed in the lower left corner and click **Commit** that is equivalent to running the **git commit** command.

Figure 3-28 Committing modifications



On the **History** tab page, view the committing status.

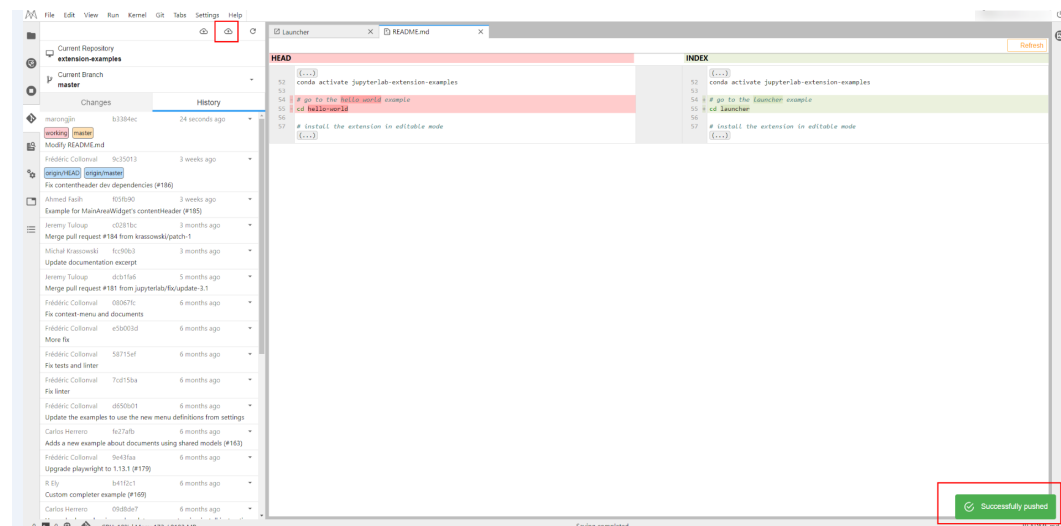
Figure 3-29 Checking whether the committing is successful



Click the **push** icon, which is equivalent to running the **git push** command, to push the code to the GitHub repository. After the pushing is successful, the

message "Successfully completed" is displayed. If the token used for OAuth authentication has expired, a dialog box is displayed asking you to enter the user token or account information. Enter the information as prompted. This section describes the authorization using a personal access token. If you use a password for authorization but the password becomes unavailable, perform the operations described in

Figure 3-30 Pushing code to the GitHub repository



After the preceding operations are complete, on the **History** tab page of the JupyterLab Git plug-in page, you can see that **origin/HEAD** and **origin/master** point to the latest push. In addition, you can find the corresponding information in the committing records of the GitHub repository.

3.3.4 Creating a Scheduled Job in JupyterLab

You can create a scheduled job in a ModelArts notebook instance. This section describes how to create a scheduled job and run a notebook file with one click to improve efficiency.

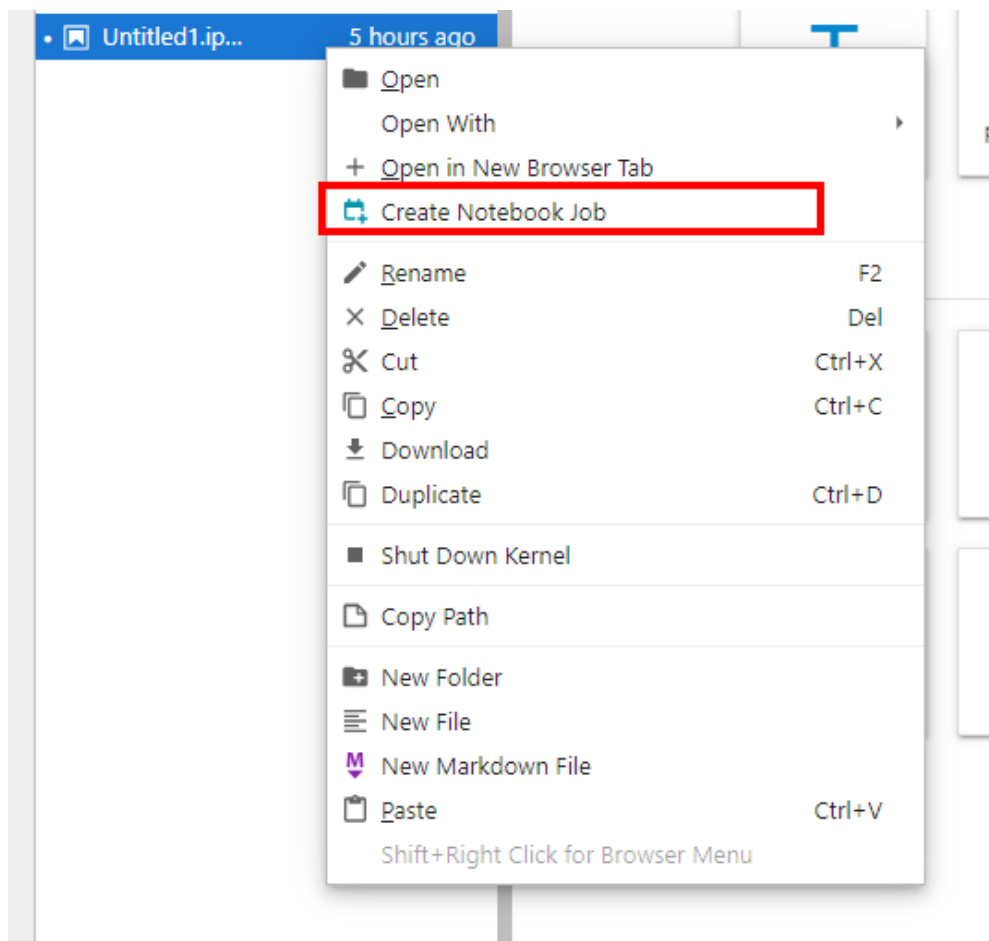
Highlights

- **One-click running:** You can run a notebook file with one click.
- **Scheduled task:** You can set the time and frequency for executing code blocks. The time can be set by second, minute, hour, day, week, or month.
- **Parameter-based execution:** You can transfer parameters to a notebook job during its runtime, so that the notebook job can adjust its behavior as required.
- **Task management GUI:** A user-friendly GUI is provided for viewing, adding, and deleting scheduled tasks.
- **Task execution record:** The status and output of each job are recorded for future query and debugging.

Procedure

1. Open ModelArts Notebook.
2. Choose a notebook file (IPYNV file) to create a scheduled job.

Figure 3-31 Opening notebook jobs



3. On the **Create Job** page, configure the parameters and click **Create**.

Figure 3-32 Parameters for creating a scheduled job

The screenshot displays the 'Create Job' configuration window. At the top, there are three tabs: 'Terminal 3', 'Notebook Jobs', and 'Launcher'. The 'Notebook Jobs' tab is active. The main content area is titled 'Create Job' and contains several sections:

- Job name:** A text field containing 'Untitled'.
- Input file:** A file selector showing a folder icon and the path '/ Untitled.ipynb'.
- Environment:** A dropdown menu with 'anaconda3' selected.
- Output formats:** Two checkboxes, 'Notebook' and 'HTML', both of which are checked.
- Parameters:** A section with a plus sign (+) to add parameters.
- Additional options:** A dropdown menu.
- Schedule:** Two radio buttons: 'Run now' (unselected) and 'Run on a schedule' (selected).
- Interval:** A dropdown menu with 'Custom schedule' selected.
- cron expression:** A text field containing '00 ** 1'.
- Time zone:** A dropdown menu with 'Asia/Shanghai' selected.
- Buttons:** 'Cancel' and 'Create' buttons at the bottom right.

- **Job name:** Enter the scheduled job name.
- **Environment:** Choose a Python environment where the notebook job is to be run.
- **Output formats:** Select the output file type of the execution result
- **Parameters:** Click the plus sign (+) to set the Python variables for running the job.

- **Schedule:** Choose **Run now** or **Run on a schedule**. The cron expression is supported.

 NOTE

- The cron expression must be in the format supported by Linux OS. Otherwise, an error will be reported. If the expression contains a question mark (?), replace it with an asterisk (*).
 - After a scheduled job is configured, modify the file name and content. The created jobs are not affected.
4. Run the job. You can view the job running records in the **Notebook Jobs** tab. Click **Reload** in the upper right corner to get the latest records.

Figure 3-33 Viewing the running records of scheduled jobs

Notebook Jobs Notebook Job Definitions

 Your job "Untitled1" has been created. If you do not see it in the list below, please reload the list in a few seconds. 

Notebook Jobs 

Job name	Input filename	Output files	Created at	Status	Actions
Untitled1	Untitled1.ipynb		2024/9/20 15:30:52	Stopped	

1-1 of 1 < >

5. After the job is executed, you can download the output files. Click the file name to view the execution result.

Figure 3-34 Viewing the execution result of a scheduled job

Job name	Input filename	Output files	Created at	Status	Actions
Untitled1	Untitled1.ipynb	Notebook HTML	2024/9/20 15:30:52	Completed	

6. In the **Notebook Job Definitions** tab, view all jobs. Click a job name to access the **Job Definition** page. You can start, stop, or delete a scheduled job. You can also click **Edit Job Definition** to update the scheduling information or view the running history of a scheduled job.

Figure 3-35 Notebook Job Definitions tab

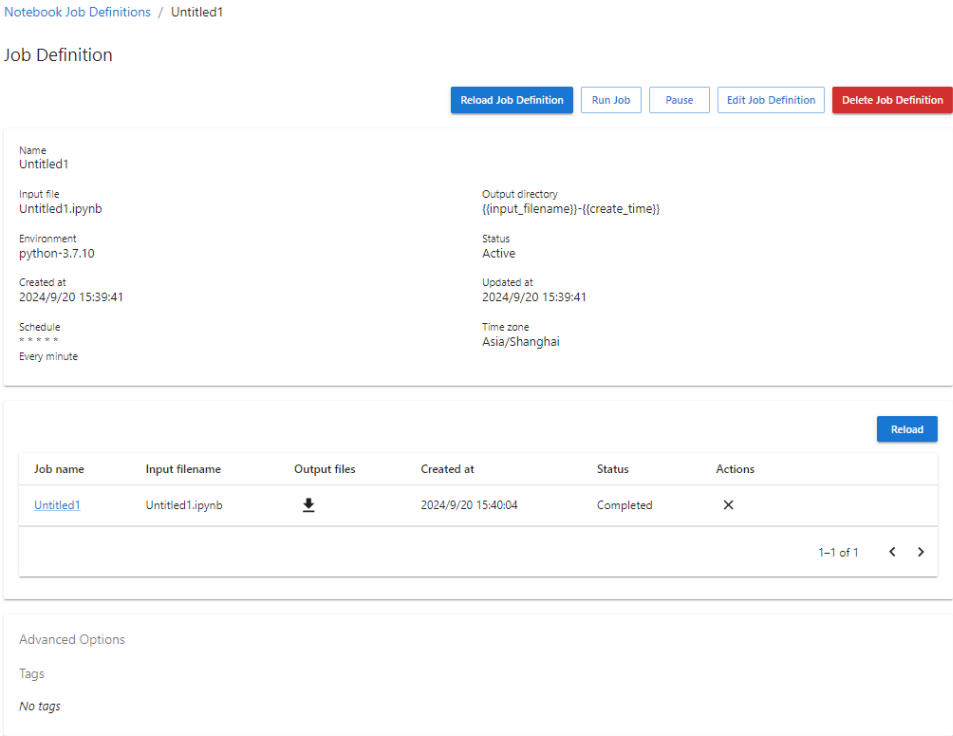
Notebook Jobs Notebook Job Definitions

 Your job definition "Untitled1" has been created. If you do not see it in the list below, please reload the list in a few seconds. 

Notebook Job Definitions 

Job definition name	Input filename	Created at	Schedule	Status	Actions
Untitled1	Untitled1.ipynb	2024/9/20 15:36:04	At 12:00 AM, only on Monday	Active	 

Figure 3-36 Configuring a scheduled job



3.3.5 Uploading Files to JupyterLab

3.3.5.1 Uploading Files from a Local Path to JupyterLab

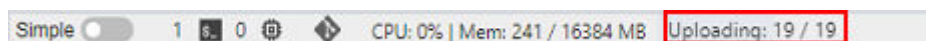
JupyterLab provides multiple methods for uploading files.

Methods for Uploading a File

- For a file not exceeding 100 MB, directly upload it to the target notebook instance. Detailed information, such as the file size, upload progress, and upload speed are displayed.
- For a file that exceeds 100 MB but does not exceed 50 GB, upload the file to OBS (an object bucket or a parallel file system), and then download the file from OBS to a notebook instance. After the download is complete, the file is deleted from OBS.
- For a file that exceeds 50 GB, upload it by calling ModelArts SDK or MoXing.
- For a file that shares the same name with an existing file in the current directory of a notebook instance, overwrite the existing file or cancel the upload.
- A maximum of 10 files can be uploaded at a time. The other files are in awaiting upload state. No folders can be uploaded. If a folder is required, compress it into a package, upload the package to notebook, and decompress the package in Terminal.
unzip xxx.zip # Directly decompress the package in the path where the package is stored.

For more details, search for the decompression command in mainstream search engines.

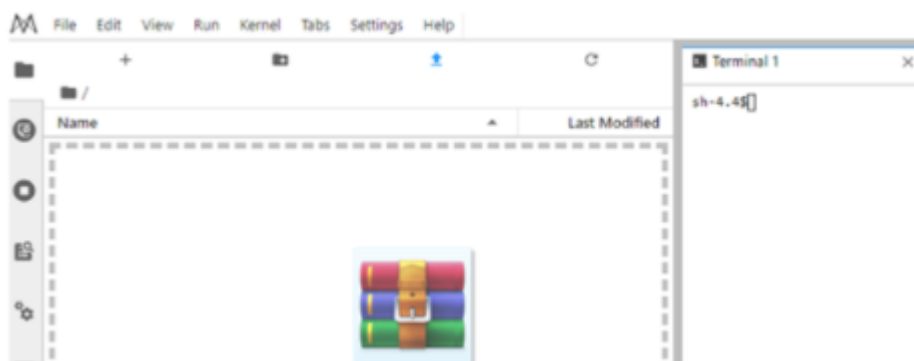
- When multiple files are uploaded in a batch, the total number of files to be uploaded and the number of files that have been uploaded are displayed at the bottom of the JupyterLab window.



Uploading File

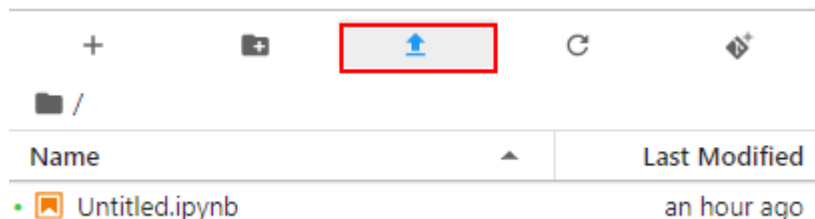
Method 1: Use JupyterLab to open a running notebook environment.

Figure 3-37 Dragging the file to JupyterLab



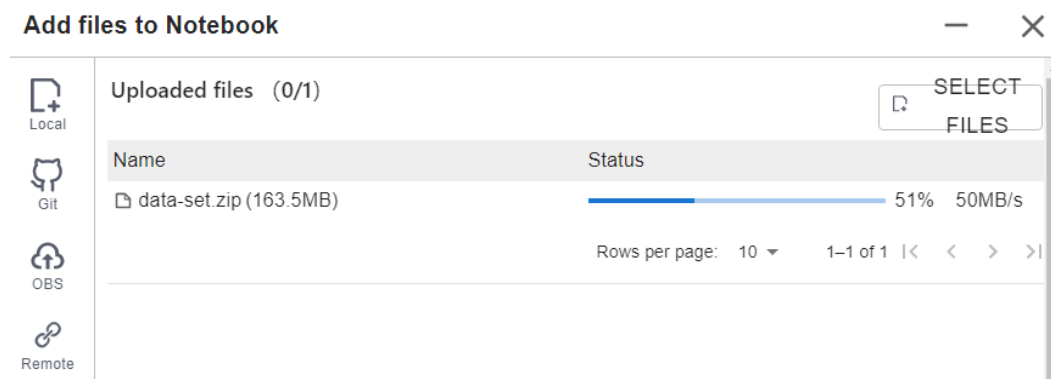
Method 2: Click  in the navigation bar on the top of the window. In the displayed dialog box, drag or select a local file and upload it.

Figure 3-38 Clicking the upload button

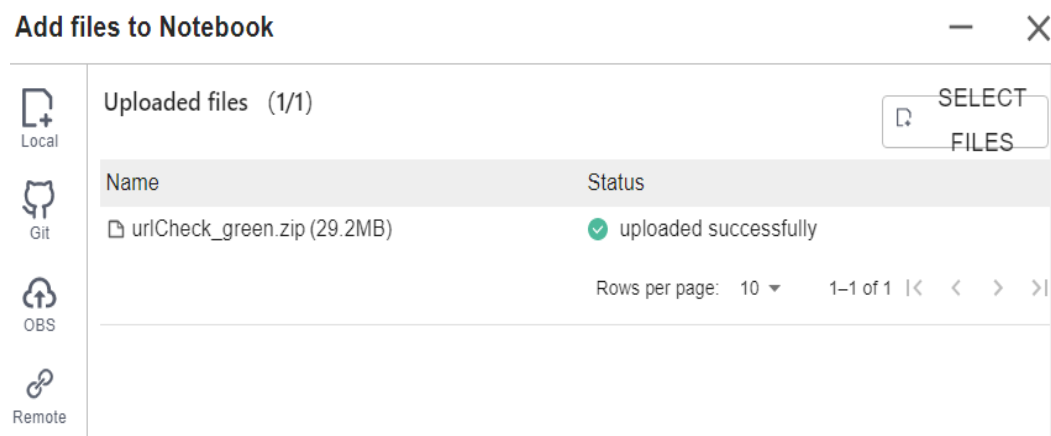


Uploading a Local File Smaller Than 100 MB to JupyterLab

For a file not exceeding 100 MB, directly upload it to the target notebook instance. Detailed information, such as the file size, upload progress, and upload speed are displayed.

Figure 3-39 Uploading a file less than 100 MB

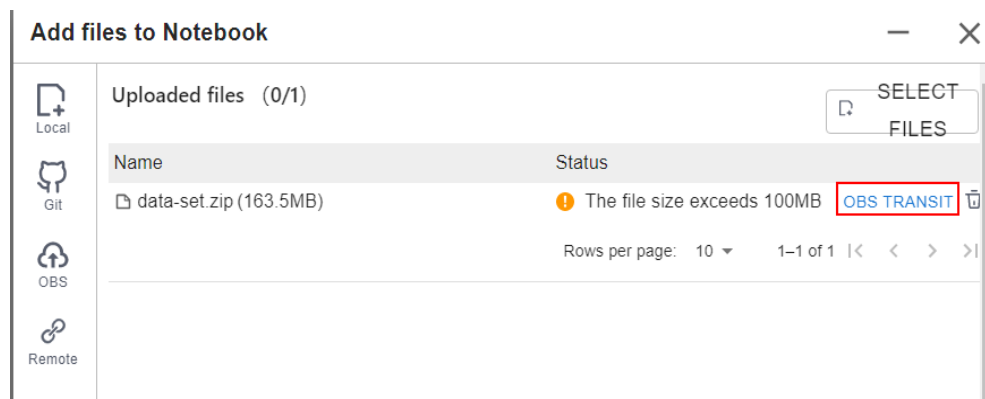
A message is displayed after the file is uploaded.

Figure 3-40 Uploaded

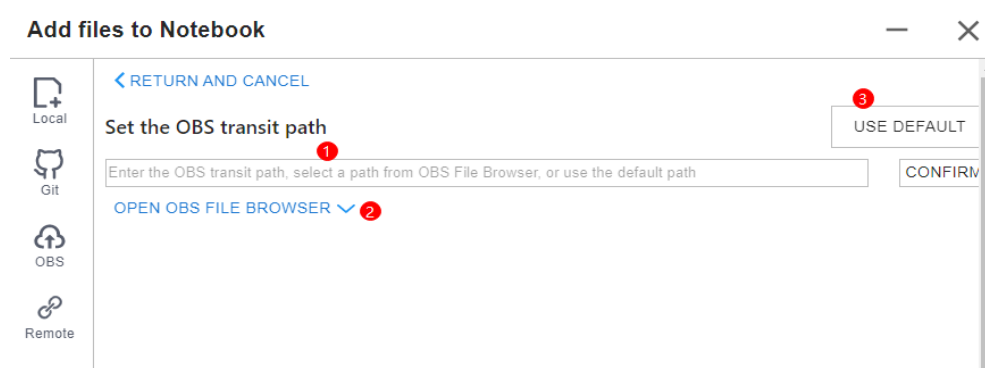
Uploading a Local File that Is 100 MB to 50 GB to JupyterLab

For a file that exceeds 100 MB but does not exceed 50 GB, upload the file to OBS (an object bucket or a parallel file system), and then download the file from OBS to the target notebook instance. After the download is complete, the file is automatically deleted from OBS.

For example, in the scenario shown in the following figure, upload the file through OBS.

Figure 3-41 Uploading a large file through OBS

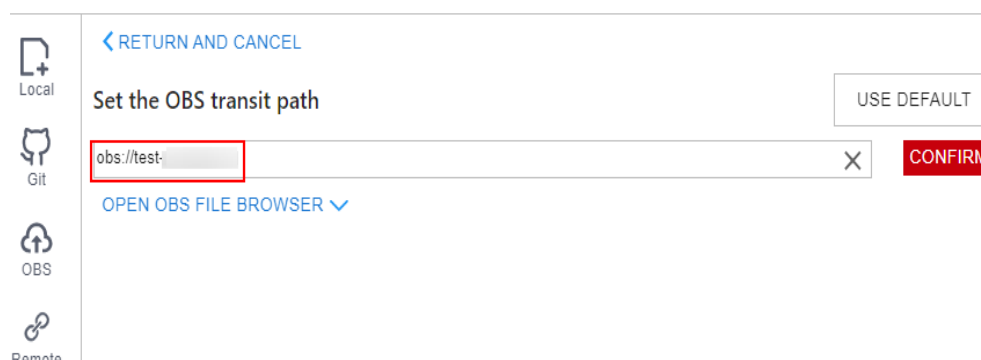
To upload a large file through OBS, set an OBS path.

Figure 3-42 Uploading a file through OBS

NOTE

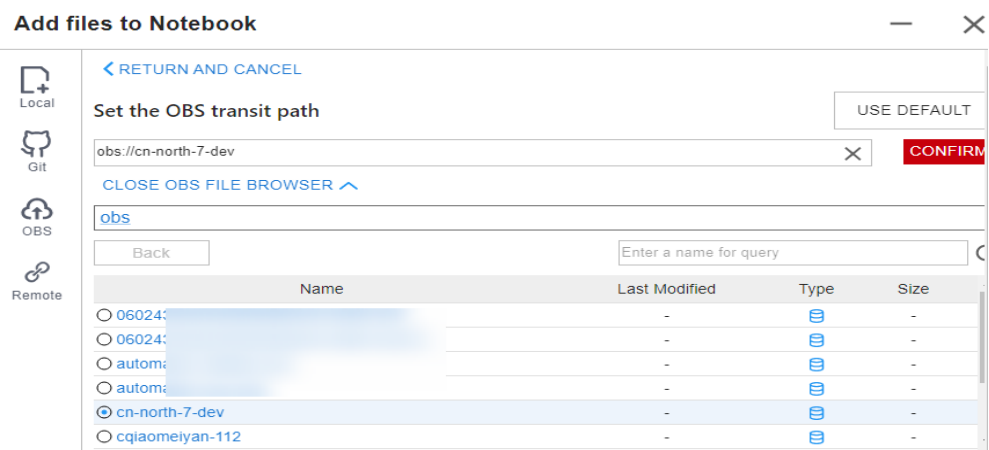
Set an OBS path for uploading local files to JupyterLab. After the setting, this path is used by default in follow-up operations. To change the path, click  in the lower left corner of the file upload window, as shown in [Figure 3-46](#).

- Method 1: Enter a valid OBS path in the text box and click **OK**.

Figure 3-43 Configuring an OBS path

- Method 2: Select an OBS path in **OBS File Browser** and click **OK**.

Figure 3-44 OBS File Browser



- Method 3: Use the default path.

Figure 3-45 Using the default path to upload a file

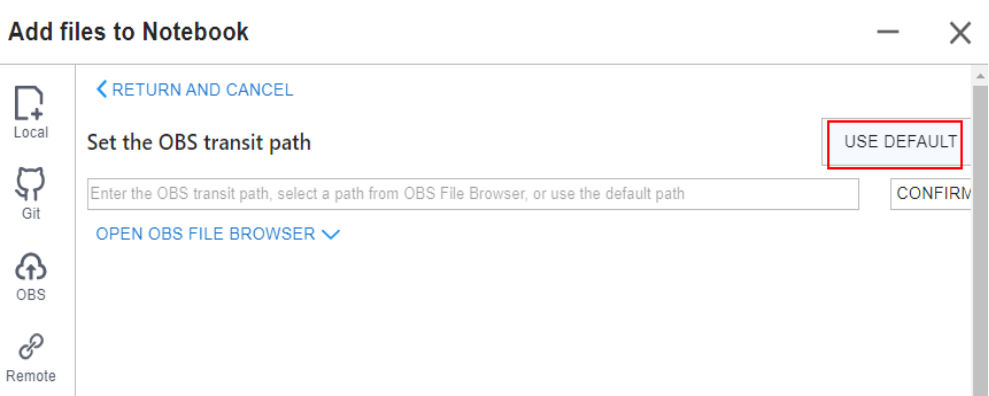
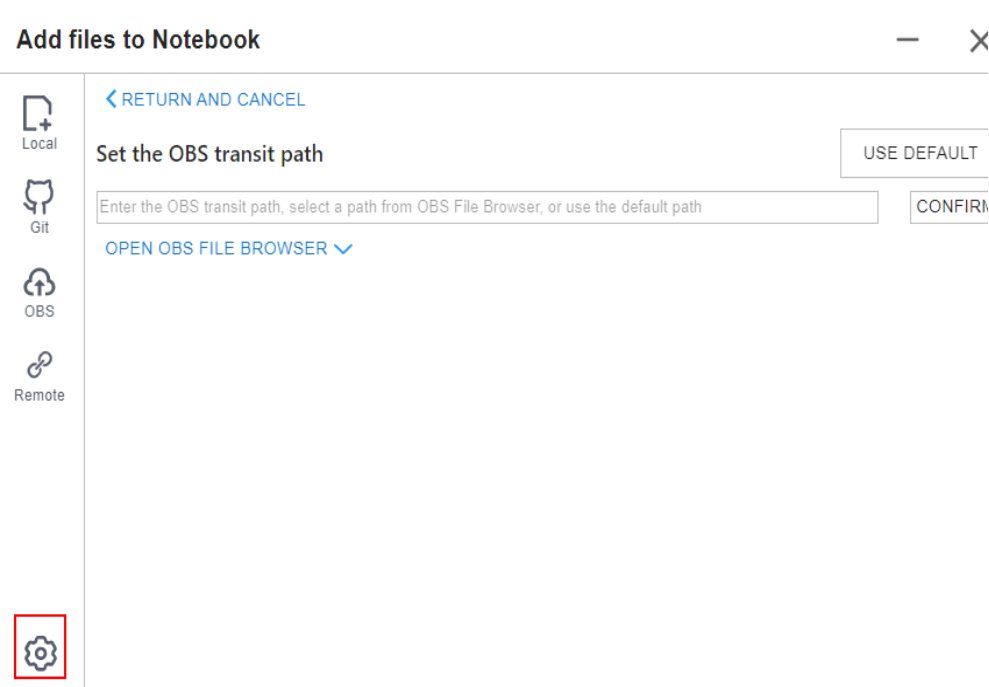
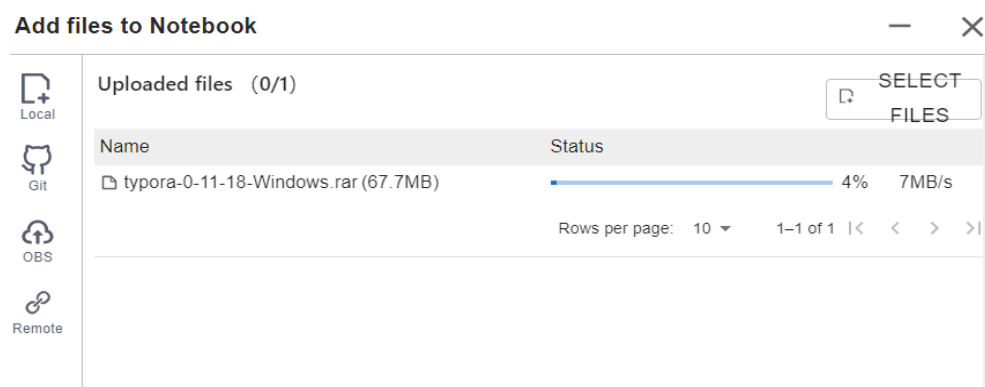


Figure 3-46 Setting an OBS path to upload a local file



After the OBS path is set, upload a file.

Figure 3-47 Uploading a file



Decompressing a package

After a large file is uploaded to Notebook JupyterLab as a compressed package, you can decompress the package in Terminal.

```
unzip xxx.zip # Directly decompress the package in the path where the package is stored.
```

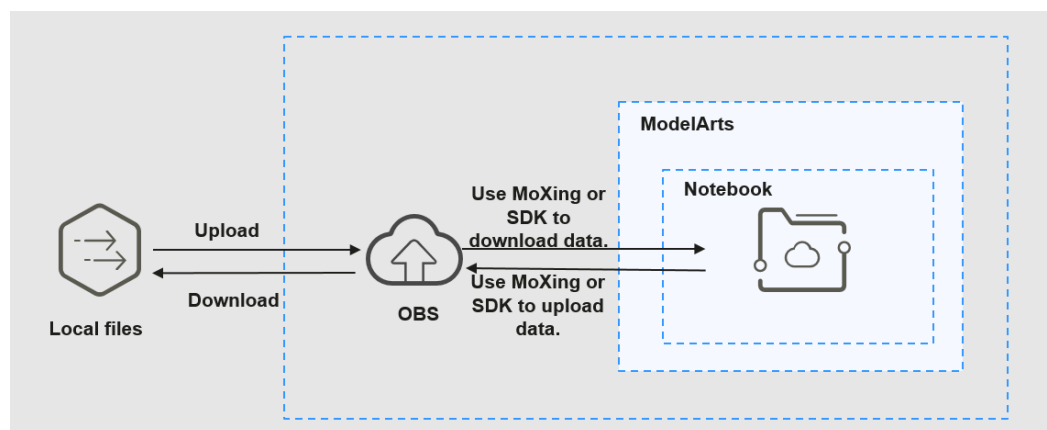
For more details, search for the decompression command in mainstream search engines.

Uploading a Local File Larger Than 50 GB to JupyterLab

A file exceeding 50 GB cannot be directly uploaded to JupyterLab.

To upload files exceeding 50 GB, upload them to OBS. Then, call the ModelArts MoXing or SDK API in the target notebook instance to read and write the files in OBS.

Figure 3-48 Uploading and downloading large files in a notebook instance



The procedure is as follows:

1. Upload the file from a local path to OBS.

2. Download the file from OBS to the notebook instance by calling the ModelArts SDK or MoXing API.

- Method 1: Call the ModelArts SDK to download a file from OBS.

Example code:

```
from modelarts.session import Session
session = Session()
session.obs.copy("obs://bucket-name/obs_file.txt", "/home/ma-user/work/")
```

- Method 2:

```
import moxing as mox
```

```
# Download the OBS folder sub_dir_0 from OBS to a notebook instance.
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/home/ma-user/work/sub_dir_0')
# Download the OBS file obs_file.txt from OBS to a notebook instance.
mox.file.copy('obs://bucket_name/obs_file.txt', '/home/ma-user/work/obs_file.txt')
```

If a .zip file is downloaded, run the following command on the terminal to decompress the package:

```
unzip xxx.zip # Directly decompress the package in the path where the package is stored.
```

After the code is executed, open the terminal shown in [Figure 3-49](#) and run the **ls /home/ma-user/work** command to view the file downloaded to the notebook instance. Alternatively, go to JupyterLab. In the navigation pane on the left, view the downloaded file. If the file is not displayed, refresh the page, as shown in [Figure 3-50](#).

Figure 3-49 Terminal

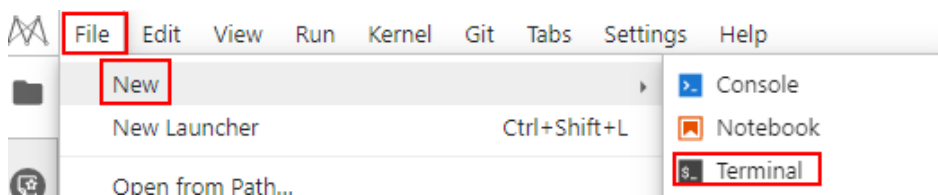
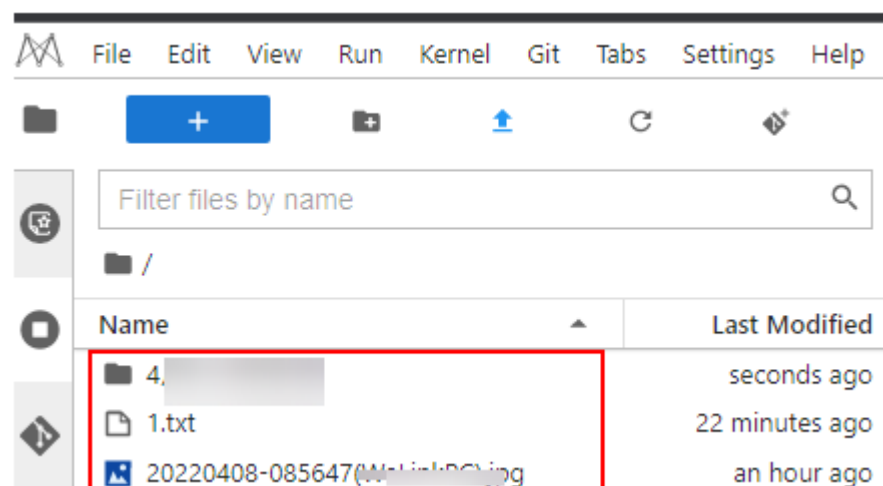


Figure 3-50 File downloaded to a notebook instance



Error Handling

If you download a file from OBS to your notebook instance and the system displays error message "Permission denied", perform the following operations for troubleshooting:

- Ensure that the target OBS bucket and notebook instance are in the same region. If they are in different regions, the access to OBS will be denied.
- In this case, ensure that the notebook account has the permission to read data in the OBS bucket.

3.3.5.2 Cloning GitHub Open-Source Repository Files to JupyterLab

Files can be cloned from a GitHub open-source repository to JupyterLab.

1. Use JupyterLab to open a running notebook instance.
2. Click  in the navigation bar on the top of the JupyterLab window. In the

displayed dialog box, click  on the left to go to the page for cloning files from a GitHub open-source repository.

Figure 3-51 File upload icon

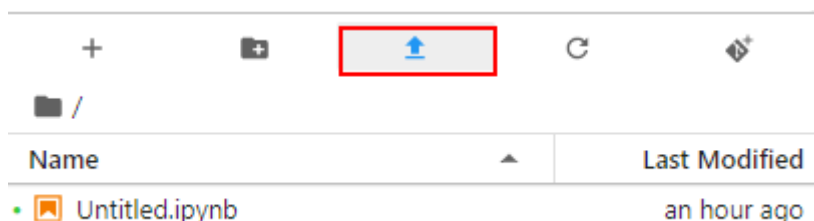
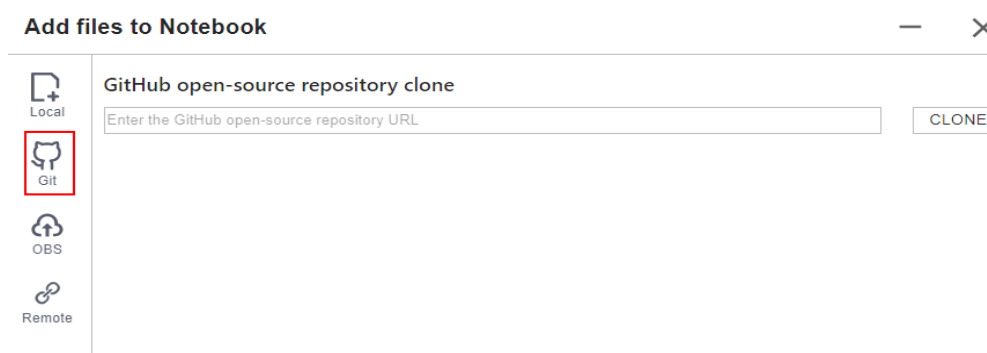
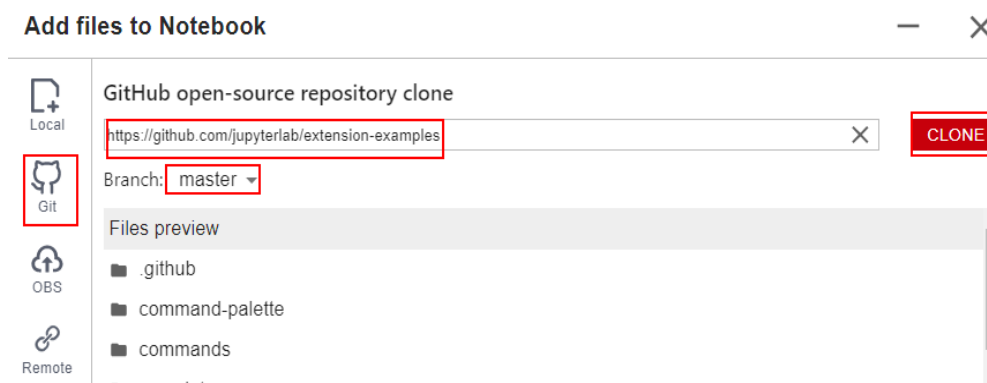


Figure 3-52 Page for cloning files from a GitHub open-source repository

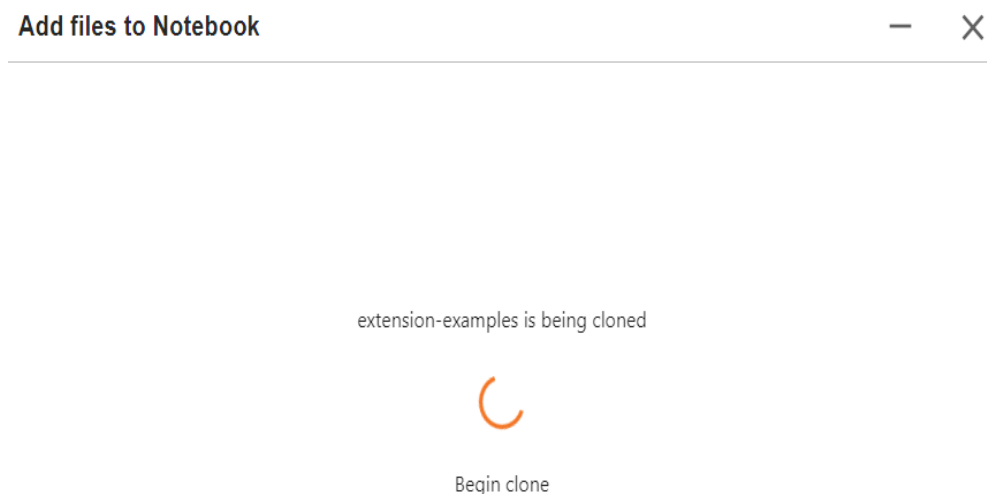


3. Enter a valid address of a GitHub open-source repository, select files from the displayed files and folders, and click **Clone**.

GitHub open-source repository address: <https://github.com/jupyterlab/extension-examples>

Figure 3-53 Entering a valid address of a GitHub open-source repository

4. View the clone process.

Figure 3-54 Process of cloning a repository

5. Complete the clone.

Error Handling

- Failing to clone the repository may be caused by network issues. In this case, run the **git clone https://github.com/jupyterlab/extension-examples.git** command on the **Terminal** page to test the network connectivity.
- If the repository already exists in the current directory of the notebook instance, the system displays a message indicating that the repository name already exists. In this case, you can overwrite the existing repository or click



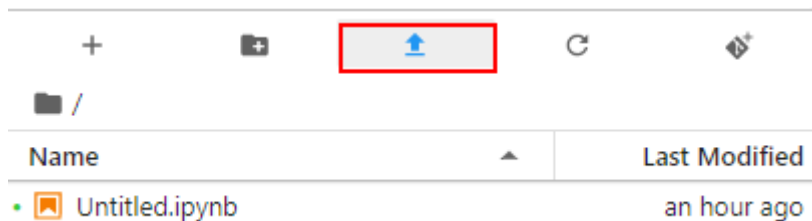
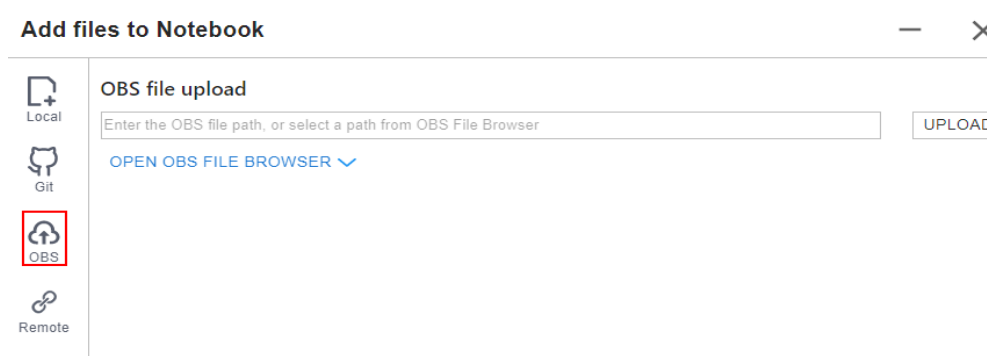
to cancel the cloning.

3.3.5.3 Uploading OBS Files to JupyterLab

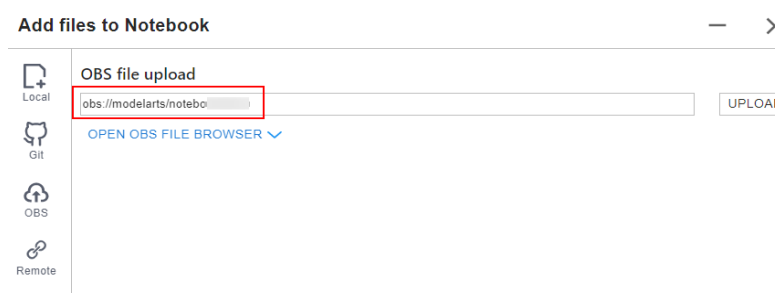
In JupyterLab, you can download files from OBS to a notebook instance. Ensure that the file is not larger than 10 GB. Otherwise, the upload will fail.

1. Use JupyterLab to open a running notebook instance.

- Click  in the navigation bar on the top of the JupyterLab window. In the displayed window, click  on the left to go to the OBS file upload page.

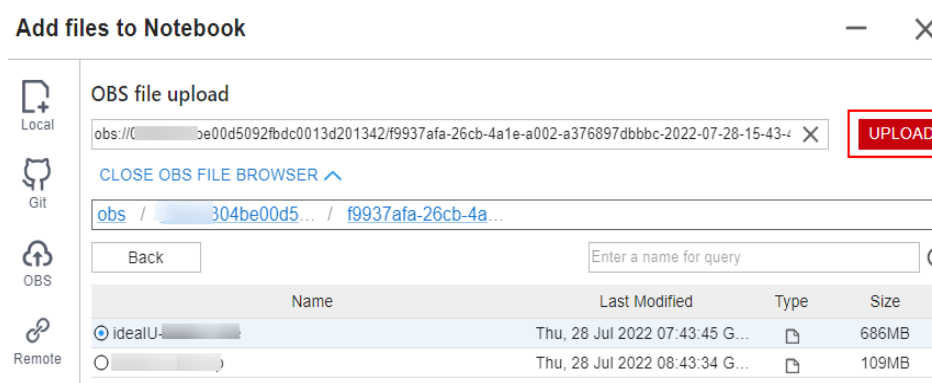
Figure 3-55 File upload icon**Figure 3-56** OBS file upload

- Set an OBS file path in either of the following ways:
 - Method 1: Enter a valid OBS file path in the text box and click **Upload**.

Figure 3-57 Entering a valid OBS file path**NOTE**

Enter an OBS file path instead of a folder path. Otherwise, the upload fails.

- Method 2: Open **OBS File Browser**, select an OBS file path, and click **Upload**.

Figure 3-58 Uploading an OBS File

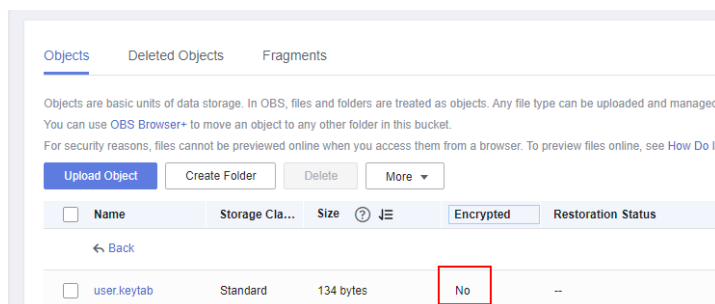
Error Handling

There are three typical scenarios in which uploading a file failed.

- **Scenario 1**

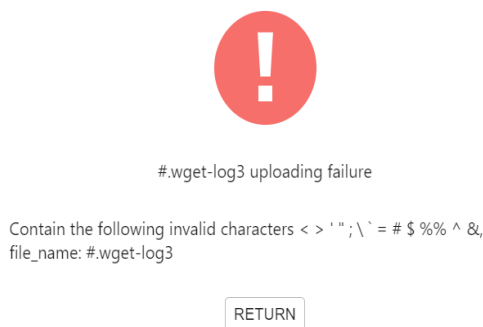
Possible causes:

- The OBS path is set to a folder instead of a file path.
- The file in OBS is encrypted. In this case, go to the OBS console and ensure that the file is encrypted.



- The OBS bucket and notebook instance are not in the same region. Ensure that the OBS bucket to be read is in the same region as the notebook instance. You cannot access an OBS bucket in another region.
- The account does not have the permission to access the OBS bucket. In this case, ensure that the notebook account has the permission to read data in the OBS bucket.
- The OBS file has been deleted. In this case, make sure that the OBS file to be uploaded is available.

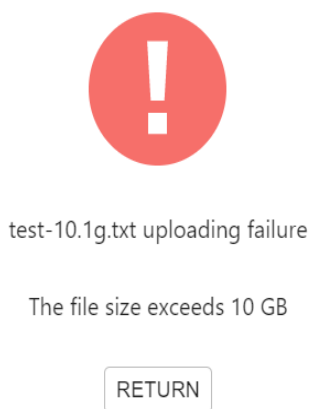
- **Scenario 2**

Figure 3-59 File uploading failure

Possible causes:

The file name contains special characters such as <>"';\`=#\$%^&.

- **Scenario 3**

Figure 3-60 File uploading failure

Possible causes:

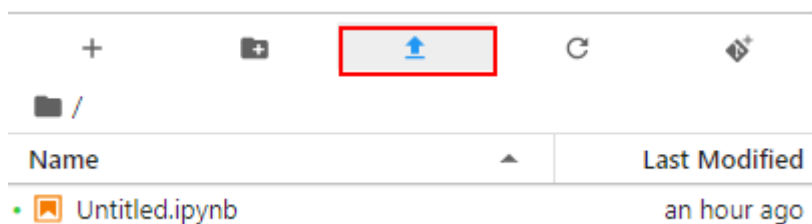
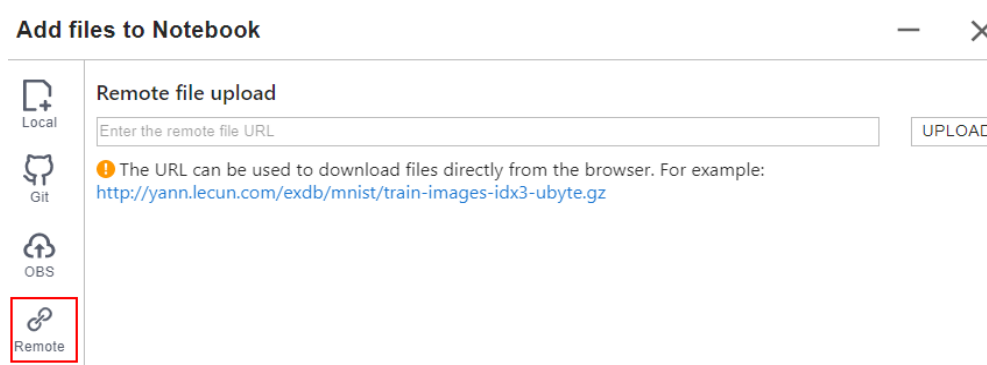
The uploaded file exceeded 50 GB.

3.3.5.4 Uploading Remote Files to JupyterLab

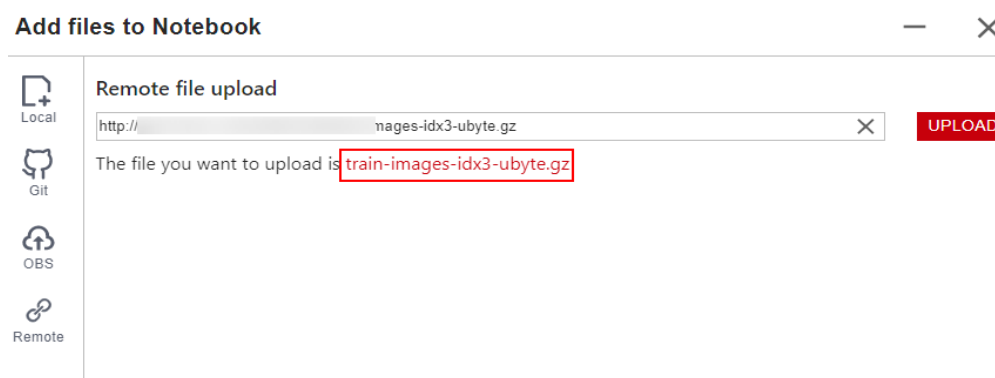
Files can be downloaded through remote file addresses to JupyterLab.

Method: Enter the URL of a remote file in the text box of a browser, and the file is directly downloaded.

1. Use JupyterLab to open a running notebook instance.
2. Click  in the navigation bar on the top of the JupyterLab window. In the displayed window, click  on the left to go to the remote file upload page.

Figure 3-61 File upload icon**Figure 3-62** Remote file upload page

3. Enter a valid remote file URL, and the system automatically identifies the file name. Then, click **Upload**.

Figure 3-63 Entering a valid remote file URL

Error Handling

Failing to upload the remote file may be caused by network issues. In this case, enter the URL of the remote file in the text box of a browser to check whether the file can be downloaded.

3.3.6 Downloading a File from JupyterLab to a Local PC

Files created in JupyterLab can be downloaded to a local path. For details about how to upload files to JupyterLab, see [Uploading Files to JupyterLab](#).

- If a file is less than or equal to 100 MB, directly download it from JupyterLab. For details, see [Downloading a File Less Than or Equal to 100 MB](#).

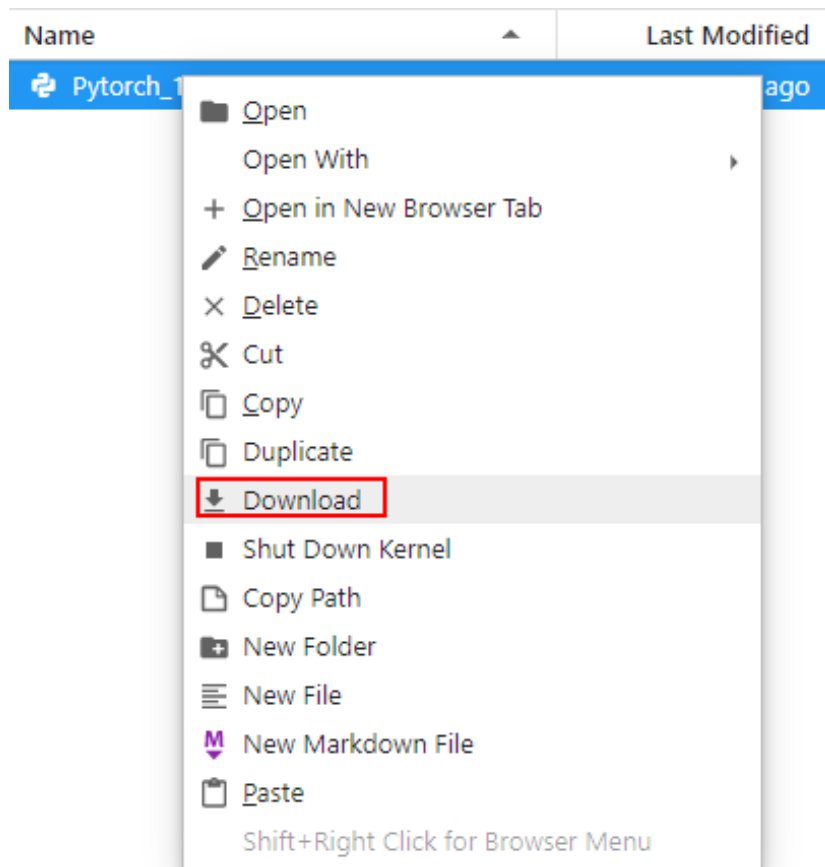
- If a file is larger than 100 MB, use OBS to transfer it to your local path. For details, see [Downloading a File Larger Than 100 MB](#).

Downloading a File Less Than or Equal to 100 MB

In the JupyterLab file list, right-click the file to be downloaded and choose **Download** from the shortcut menu.

The file is downloaded to your browser's downloads folder.

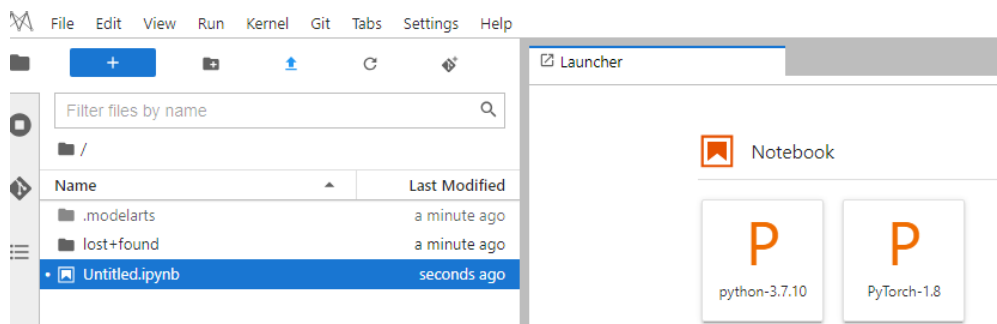
Figure 3-64 Downloading a file



Downloading a File Larger Than 100 MB

Use OBS to transfer the file from the target notebook instance to the local path. To do so, perform the following operations:

1. Open the Python runtime environment.
In the **Launcher** tab, click **python-3.7.10** under **Notebook**, as shown in the following figure.

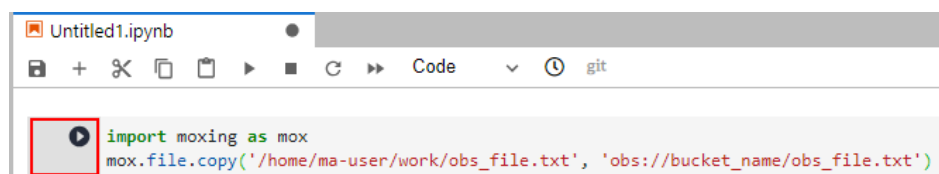
Figure 3-65 Opening the Python runtime environment

2. Use MoXing to upload the target file from the notebook instance to OBS.
The following shows the sample Python code for uploading the TXT file and compressed folder. **/home/ma-user/work/xxx** indicates the path for storing the file in the notebook instance. **obs://bucket_name/xxx** indicates the path for storing the file on OBS. Replace them with the actual paths.

- Upload the **obs_file.txt** file from the notebook instance to OBS.

Enter the following code, modify the path as required, and click  to run the code. If the TXT file exists in the OBS bucket, the upload is successful.

```
import moxing as mox
mox.file.copy('/home/ma-user/work/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

Figure 3-66 Running code example

- Upload the compressed folder **sub_dir_0** from the notebook instance to OBS.

Enter the following code, modify the path as required, and click  to run the code. If the folder exists in the OBS bucket, the upload is successful.

```
import moxing as mox
mox.file.copy_parallel('/home/ma-user/work/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```

3. Use OBS or ModelArts SDK to download the file from OBS to the local path.
 - Method 1: Use OBS to download the file.
Download **obs_file.txt** from OBS to the local path. If a large amount of data is to be downloaded, use OBS Browser+ to download. For details, see [Downloading an Object](#).
 - Method 2: Use ModelArts SDK to download the file.
 - i. [Download and install the SDK locally](#).
 - ii. [Authenticate sessions](#).
 - iii. [Download the file from OBS to the local path](#). Example code is as follows:

```
from modelarts.session import Session
```

```
# Hardcoded or plaintext AK/SK is risky. For security, encrypt your AK/SK and store them
# in the configuration file or environment variables.
# In this example, the AK and SK are stored in environment variables for identity
# authentication. Before running this example, set environment variables SDK_AK and
# SDK_SK.
__AK = os.environ["SDK_AK"]
__SK = os.environ["SDK_SK"]
# Decrypt the password if it is encrypted.
session = Session(access_key=__AK,secret_key=__SK, project_id='****', region_name='****')

session.download_data(bucket_path="/bucket_name/obs_file.txt",path="/home/user/
obs_file.txt")
```

3.3.7 Using MindInsight Visualization Jobs in JupyterLab

ModelArts notebook supports MindInsight visualization jobs. In a development environment, use a small dataset to train and debug an algorithm. This is used to check algorithm convergence and detect training issues, facilitating debugging.

MindInsight visualizes information such as scalars, images, computational graphs, and model hyperparameters during training. It also provides functions such as training dashboard, model lineage, data lineage, and performance debugging, helping you train and debug models efficiently. MindInsight supports MindSpore training jobs. For more information about MindInsight, see [MindSpore official website](#).

MindSpore allows you to save data into the summary log file and obtain the data on the MindInsight GUI.

Prerequisites

When using MindSpore to edit a training script, add the code for collecting the summary record to the script to ensure that the summary file is generated in the training result.

For details, see [Collecting Summary Record](#).

Note

- To run a MindInsight training job in a development environment, start MindInsight and then the training process.
- Only one-card single-node training is supported.

Creating a MindInsight Visualization Job in a Development Environment

[Step 1 Create a Development Environment and Access It Online](#)

[Step 2 Upload the Summary Data](#)

[Step 3 Start MindInsight](#)

[Step 4 View Visualized Data on the Training Dashboard](#)

Step 1 Create a Development Environment and Access It Online

Log in to ModelArts management console. In the navigation pane on the left, choose **DevEnviron** > **Notebook**, and create a development environment instance

using the MindSpore engine. After the instance is created, click **Open** in the **Operation** column of the instance to access it online.

Step 2 Upload the Summary Data

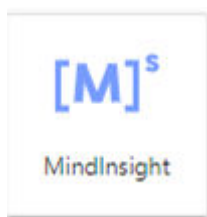
Summary data is required for MindInsight visualization in a development environment.

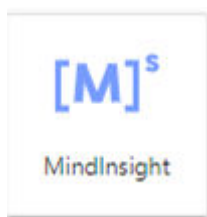
Upload the summary data to the `/home/ma-user/work/` directory in a development environment or store it in an OBS parallel file system.

- For details about how to upload the summary data to the notebook path `/home/ma-user/work/`, see [Uploading Files from a Local Path to JupyterLab](#).
- To store the summary data in an OBS parallel file system that is mounted to a notebook instance, upload the summary file generated during model training to the OBS parallel file system. When MindInsight is started in a notebook instance, the notebook instance automatically reads the summary data from the mounted OBS parallel file system.

Step 3 Start MindInsight

Open MindInsight in JupyterLab.

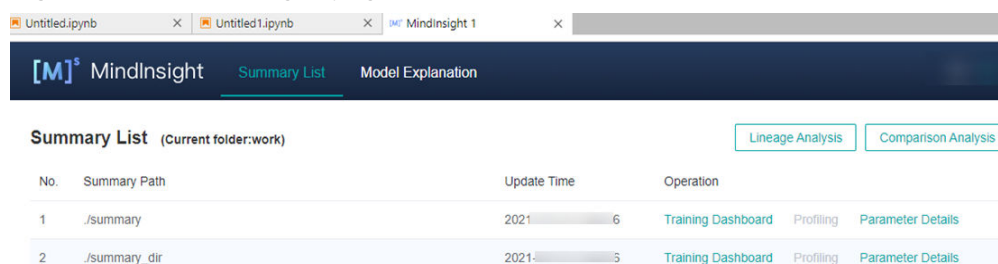


Click  to go to the MindInsight page.

Data is read from `/home/ma-user/work/` by default.

If there are two projects or more, select the target project to view its logs.

Figure 3-67 MindInsight page (2)



Step 4 View Visualized Data on the Training Dashboard

The training dashboard is important for MindInsight visualization. It allows visualization for scalars, parameter distribution, computational graphs, dataset graphs, images, and tensors.

For more information, see [Viewing Training Dashboard](#) on the MindSpore official website.

Disabling MindInsight

Click . The MindInsight instance management page is displayed, which shows all started MindInsight instances. Click **SHUT DOWN** next to the target instance to stop it.

Figure 3-68 Stopping an instance



3.3.8 Using TensorBoard Visualization Jobs in JupyterLab

ModelArts supports TensorBoard for visualizing training jobs. TensorBoard is a visualization tool package of TensorFlow. It provides visualization functions and tools required for machine learning experiments.

TensorBoard effectively displays the computational graph of TensorFlow in the running process, the trend of all metrics in time, and the data used in the training. For more details about TensorBoard, see [TensorBoard official website](#).

TensorBoard visualization training jobs support only CPU and GPU flavors based on TensorFlow and PyTorch images. Select images and flavors based on the site requirements.

Prerequisites

When you write a training script, add the code for collecting the summary record to the script to ensure that the summary file is generated in the training result.

For details about how to add the code for collecting the summary record to a TensorFlow-powered training script, see [TensorFlow official website](#).

Process of Creating a TensorBoard Visualization Job in a Development Environment

[Step 1 Create a Development Environment and Access It Online](#)

[Step 2 Upload the Summary Data](#)

[Step 3 Start TensorBoard](#)

[Step 4 View Visualized Data on the Training Dashboard](#)

Step 1 Create a Development Environment and Access It Online

Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**. Create an instance using a TensorFlow or

PyTorch image. After the instance is created, click **Open** in the **Operation** column of the instance to access it online.

TensorBoard visualization training jobs support only CPU and GPU flavors based on TensorFlow and PyTorch images. Select images and flavors based on the site requirements.

Step 2 Upload the Summary Data

Summary data is required for using TensorBoard visualization functions in DevEnviron.

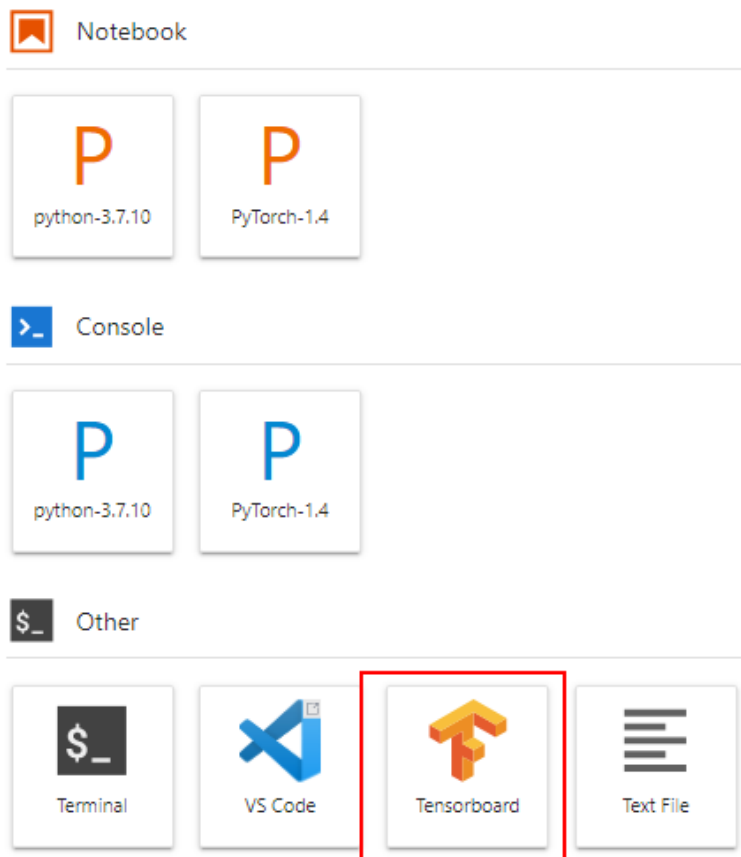
You can upload the summary data to the `/home/ma-user/work/` directory in the development environment or store it in the OBS parallel file system.

- For details about how to upload the summary data to the notebook path `/home/ma-user/work/`, see [Uploading Files from a Local Path to JupyterLab](#).
- To store the summary data in an OBS parallel file system that is mounted to a notebook instance, upload the summary file generated during model training to the OBS parallel file system. When TensorBoard is started in a notebook instance, the notebook instance automatically mounts the OBS parallel file system directory and reads the summary data.

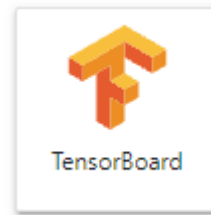
Step 3 Start TensorBoard

Choose a way you like to start TensorBoard in JupyterLab.

Figure 3-69 Starting TensorBoard in JupyterLab

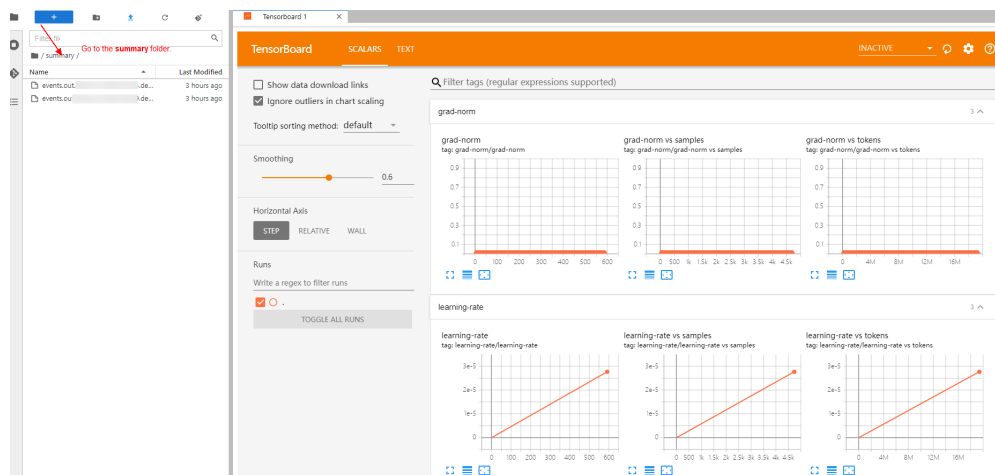


1. Open JupyterLab, in the navigation pane on the left, create the **summary** folder, and upload data to **/home/ma-user/work/summary**. The folder name must be **summary**.



2. Go to the **summary** folder and click TensorBoard page. See [Figure 3-70](#).

Figure 3-70 TensorBoard page (1)



Step 4 View Visualized Data on the Training Dashboard

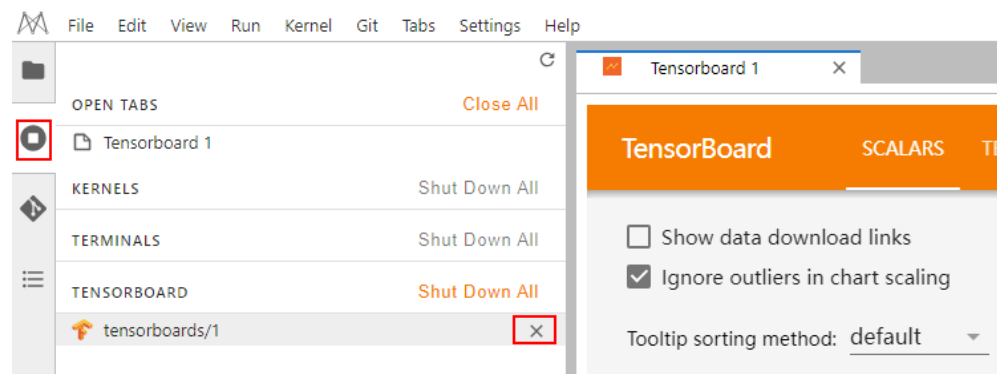
For TensorBoard visualization, you need the training dashboard. It lets you visualize scalars, images, and computational graphs.

For more functions, see [Get started with TensorBoard](#).

Disabling TensorBoard

To stop a TensorBoard instance, use any of the following methods:

Click . The TensorBoard instance management page is displayed, which shows all started TensorBoard instances. Click **SHUT DOWN** next to an instance.

Figure 3-71 Clicking SHUT DOWN to stop an instance

3.4 Using Notebook Instances Remotely Through PyCharm

3.4.1 Connecting to a Notebook Instance Through PyCharm Toolkit

AI developers use PyCharm to develop algorithms or models. ModelArts provides the PyCharm Toolkit plug-in to help AI developers quickly submit locally developed code to the ModelArts training environment. With PyCharm Toolkit, developers can quickly remotely access notebook instances, upload code, submit training jobs, and obtain training logs for local display so that they can better focus on local code development.

This section introduces how to connect to a notebook instance through PyCharm Toolkit.

Constraints

- Currently, only versions between 2019.2 and 2023.2 (including 2019.2 and 2023.2) are supported, including the community edition and professional edition.
- PyCharm of only the professional edition can be used to access notebook instances.
- You can use a community or professional edition of PyCharm Toolkit to submit training jobs. The latest version of PyCharm Toolkit can be used only to submit training jobs of the new version.
- PyCharm Toolkit supports only PyCharm of the Windows version.

Table 3-10 Toolkit functions of the latest version

Supported Function	Description	Reference
Remote SSH connection	The notebook development environment can be accessed through remote SSH.	Connecting to a Notebook Instance Through PyCharm Toolkit
Model training	Code developed locally can be quickly submitted to ModelArts and a training job of the new version is automatically created. During the running of the training job, training logs can be obtained and displayed on a local host.	Using PyCharm Toolkit to Create and Debug a Training Job
OBS-based upload and download	Local files or folders can be uploaded to OBS and files or folders can be downloaded from OBS to a local directory.	Uploading Data to a Notebook Instance Through PyCharm

Prerequisites

PyCharm professional edition of a version between 2019.2 and 2023.2 (including 2019.2 and 2023.2) has been installed on the local PC. Remote SSH applies only to the PyCharm professional edition. [Download PyCharm](#) and install it.

Step 1 Downloading and Installing PyCharm Toolkit

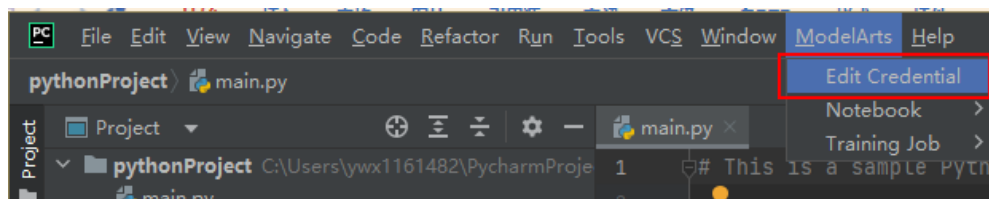
In PyCharm, choose **File > Settings > Plugins**, search for **ModelArts** in Marketplace, and click **Install**.

Step2 Creating a Notebook Instance

Create a notebook instance with remote SSH enabled and whitelist configured. Ensure that the instance is running. For details, see [Creating a Notebook Instance](#).

To use the AK/SK pair for login authentication, perform the following steps:

1. Open PyCharm with Toolkit installed. Choose **ModelArts > Edit Credential** from the menu bar.

Figure 3-72 Edit Credential

NOTE

If **ModelArts > Edit Credential** is not displayed on the menu bar, the PyCharm version may be too high. PyCharm Toolkit is not adapted to PyCharm versions later than 2023.2. Download the PyCharm professional edition of a version between 2019.2 and 2023.2 (including 2019.2 and 2023.2).

- In the displayed dialog box, select the region where ModelArts is located, enter the AK and SK, and click **OK**.
 - Region:** Select a region from the drop-down list. It must be the same as the region of the ModelArts console.
 - Project:** After the region is selected, the project is automatically filled.
 - Access Key ID:** Enter the AK.
 - Secret Access Key:** Enter the SK.
- View the verification result.

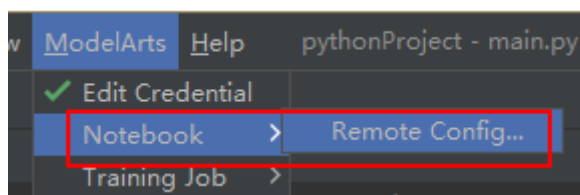
In the **Event Log** area, if information similar to the following is displayed, the access key has been successfully added:

16:01Validate Credential Success: The credential is valid.

Step 4 Automatically Configuring PyCharm Toolkit

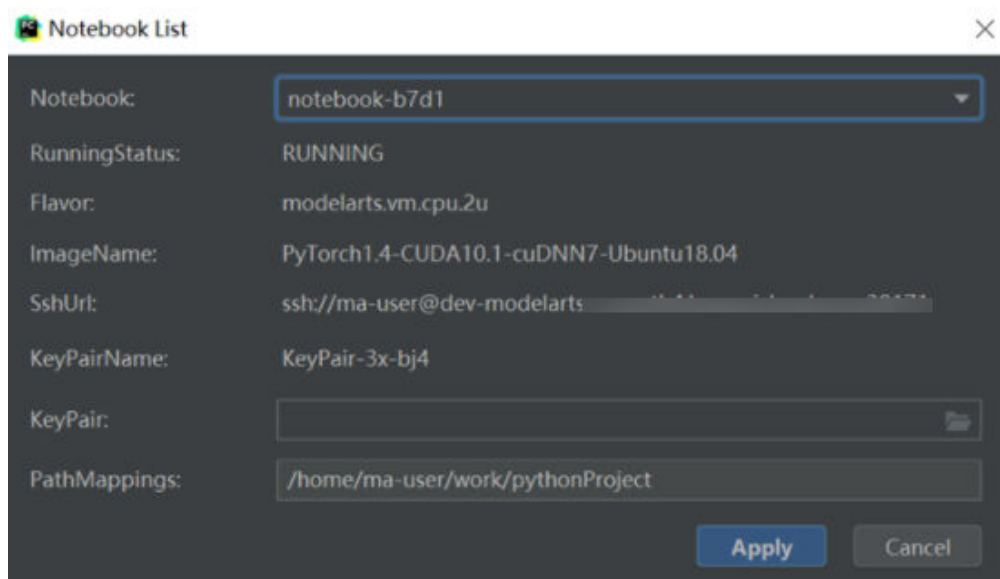
- In the local PyCharm development environment, choose **ModelArts > Notebook > Remote Config...**, and configure PyCharm Toolkit.

Figure 3-73 Remotely connecting to PyCharm Toolkit



- Choose the target instance from the drop-down list, where all notebook instances with remote SSH enabled under the account are displayed.

Figure 3-74 Notebook list

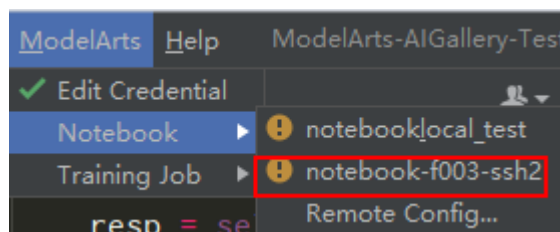


- **KeyPair:** Select the locally stored key pair of the notebook instance for authentication. The key pair created during the notebook instance creation is saved in your browser's default downloads folder.
 - **PathMappings:** Synchronization directory for the local IDE project and notebook, which defaults to `/home/ma-user/work/Project name` and is adjustable.
3. Click **Apply**. After the configuration is complete, restart the IDE for the configuration to take effect.
- After the restart, it takes about 20 minutes to update the Python interpreter for the first time.

Step 5 Accessing a Notebook Instance Through PyCharm Toolkit

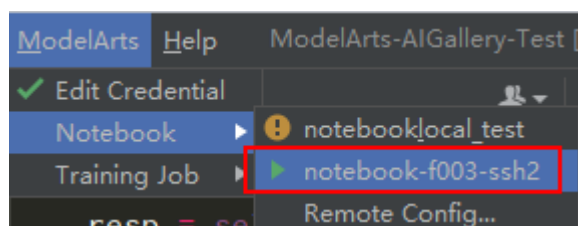
Click the notebook instance name and connect it to the local IDE as prompted. The connection is kept for 4 hours by default.

Figure 3-75 Starting the connection



To interrupt the connection, click the notebook name and disconnect it from the local IDE as prompted.

Figure 3-76 Interrupting the connection

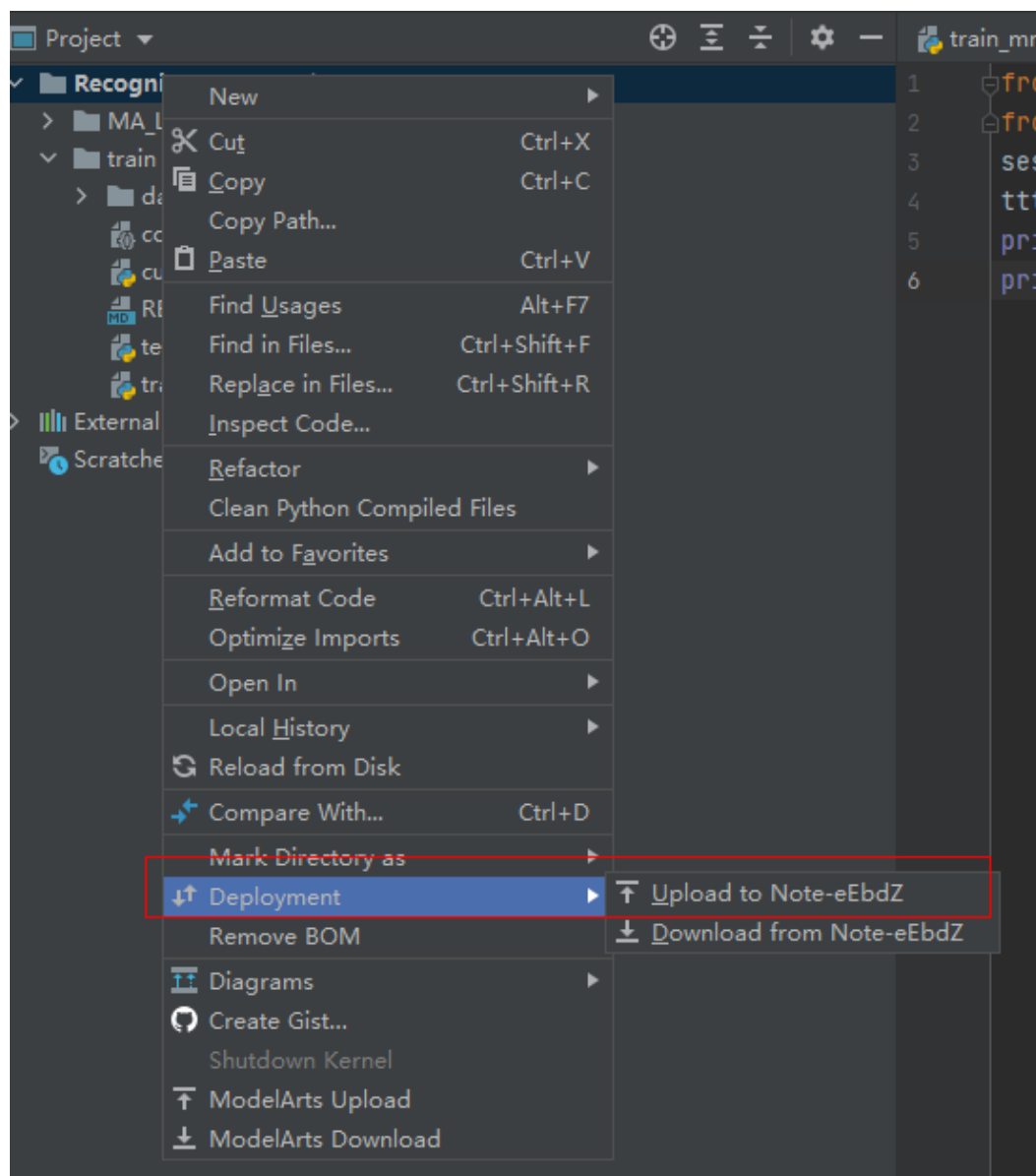


Step 6 Uploading Local Files to a Notebook Instance

Code in a local file can be copied to the local IDE, which will automatically synchronize the code to the in-cloud development environment.

Initial synchronization

In the **Project** directory of the local IDE, right-click **Deployment** and choose **Upload to Notebook name** from the shortcut menu to upload the local project file to the specified notebook instance.

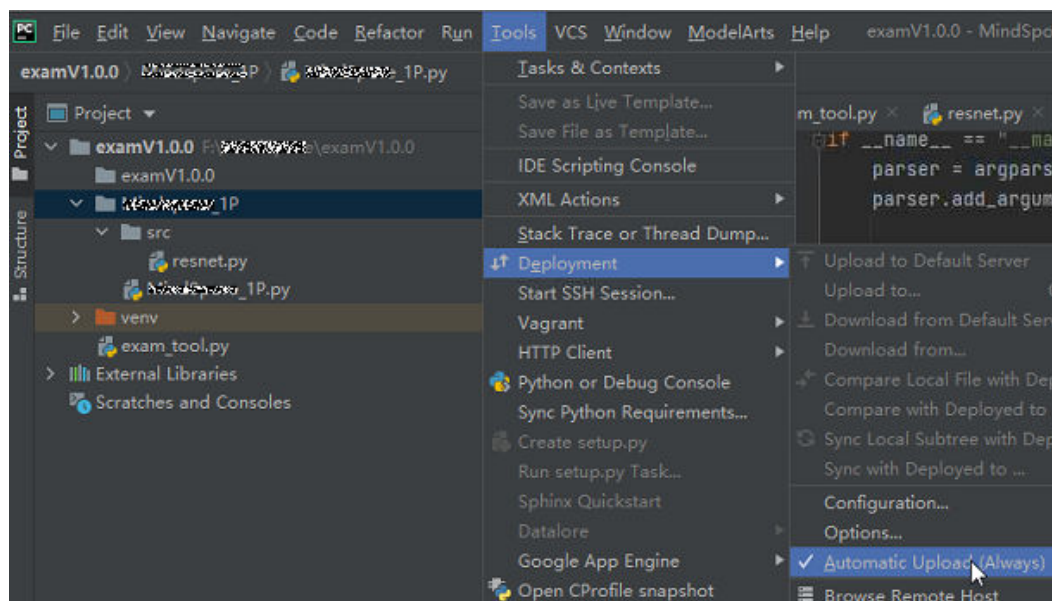
Figure 3-77 Synchronizing local data to a notebook instance

Follow-up synchronization

After modifying the code, press **Ctrl+S** to save it. The local IDE will automatically synchronize the modification to the specified notebook instance.

After PyCharm Toolkit is installed, **Automatic Upload** is automatically enabled in the local IDE for automatically uploading the files in the local directory to the target notebook instance. If **Automatic Upload** is not enabled, enable it by referring to the following figure.

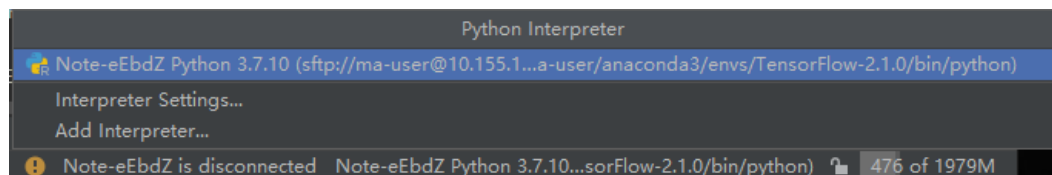
Figure 3-78 Enabling Automatic Upload



Step 7 Remotely Debugging the Code

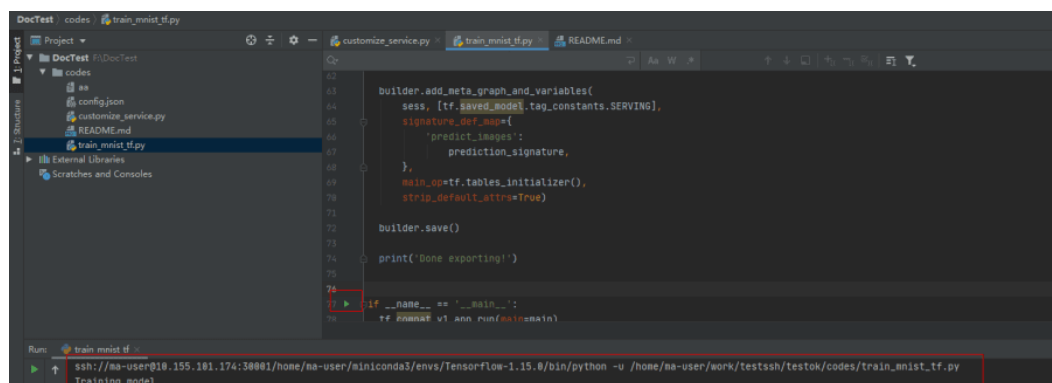
Click **Interpreter** in the lower right corner of the local IDE and select a notebook Python interpreter.

Figure 3-79 Selecting a Python interpreter

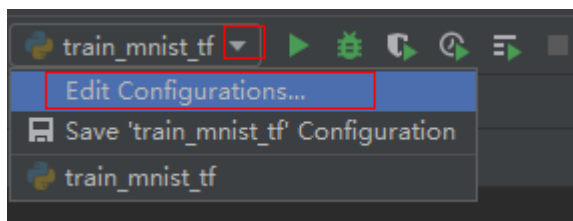


Run the code in the notebook instance. The logs are displayed locally.

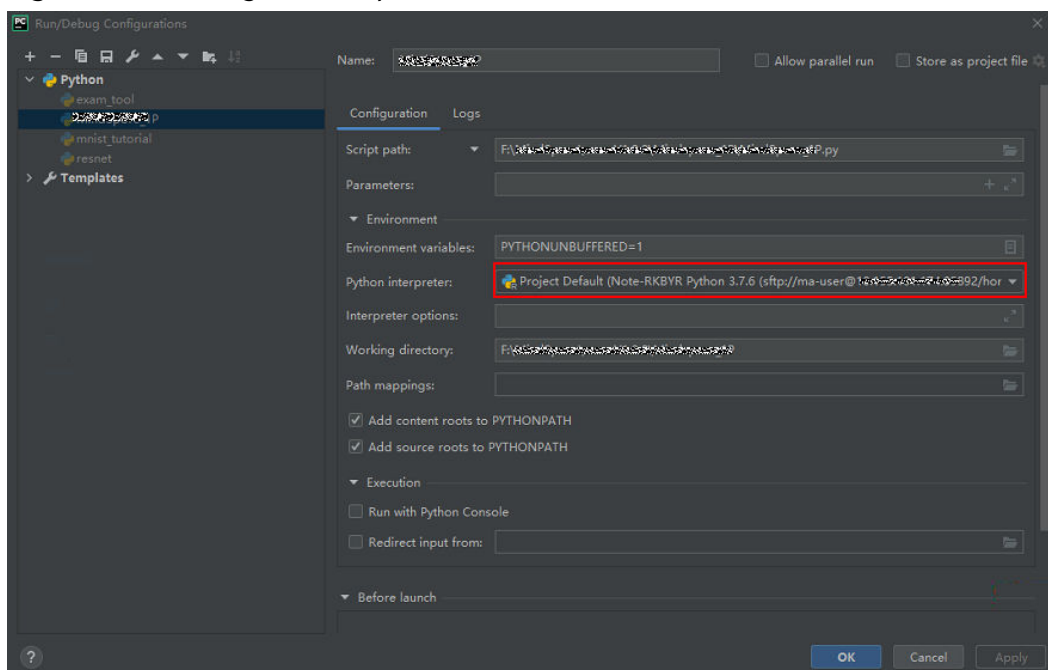
Figure 3-80 Runtime logs



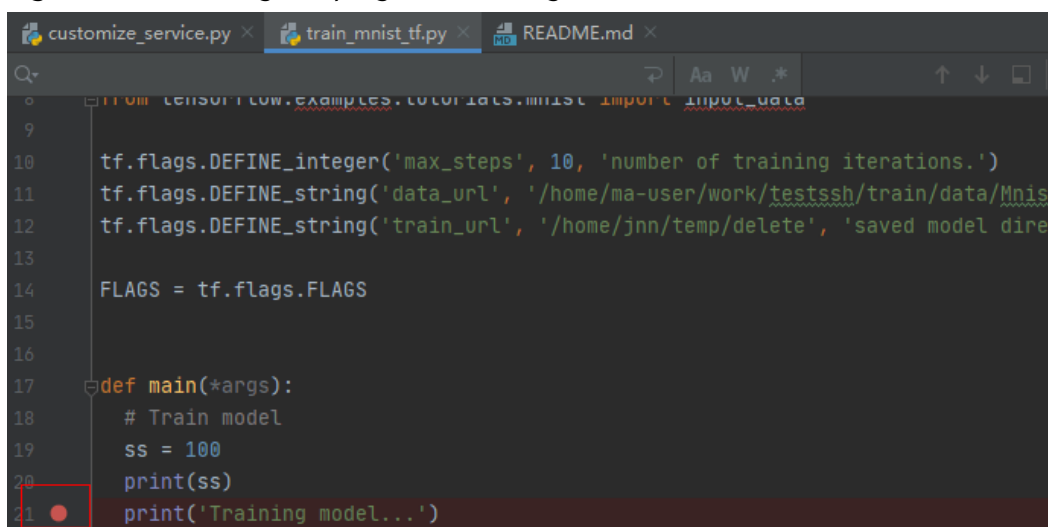
Click **Run/Debug Configurations** in the upper right corner of the local IDE to set runtime parameters.

Figure 3-81 Setting runtime parameters (1)

Select the Python interpreter that remotely connects to the target notebook instance.

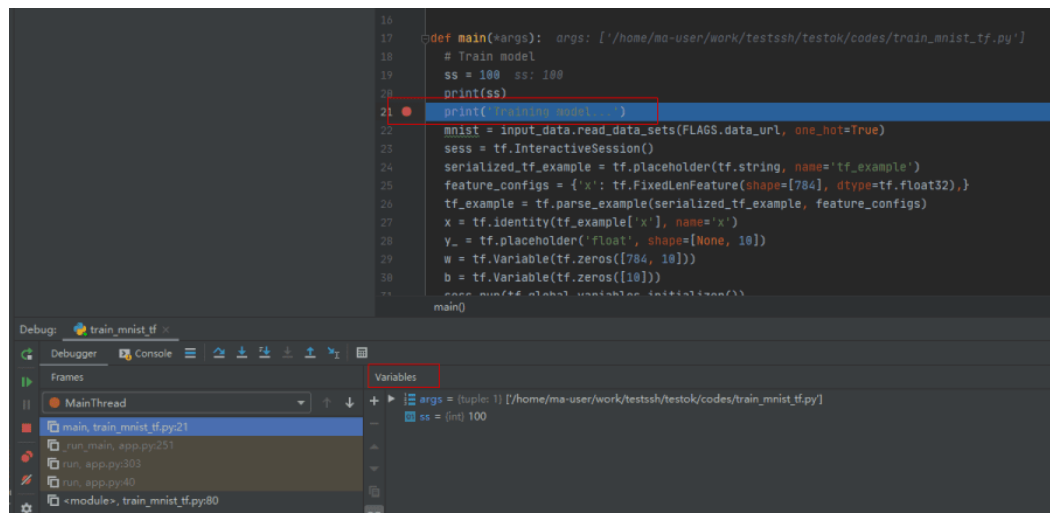
Figure 3-82 Setting runtime parameters (2)

To debug code, set breakpoints and run the program in debug mode.

Figure 3-83 Running the program in debug mode

In debug mode, the code execution is suspended in the specified line, and you can obtain variable values.

Figure 3-84 Viewing variable values in debug mode



3.4.2 Manually Connecting to a Notebook Instance Through PyCharm

A local IDE supports PyCharm and VS Code. You can use PyCharm or VS Code to remotely connect the local IDE to the target notebook instance on ModelArts for running and debugging code.

This section describes how to use PyCharm to access a notebook instance.

Prerequisites

- PyCharm professional 2019.2 or later has been installed locally. The PyCharm professional edition is available because remote SSH applies only to the professional edition.
- A notebook instance has been created with remote SSH enabled. Ensure that the instance is running. For details, see [Creating a Notebook Instance](#).
- The address and port number of the development environment are available. To obtain this information, go to the notebook instance details page.

Figure 3-85 Instance details page

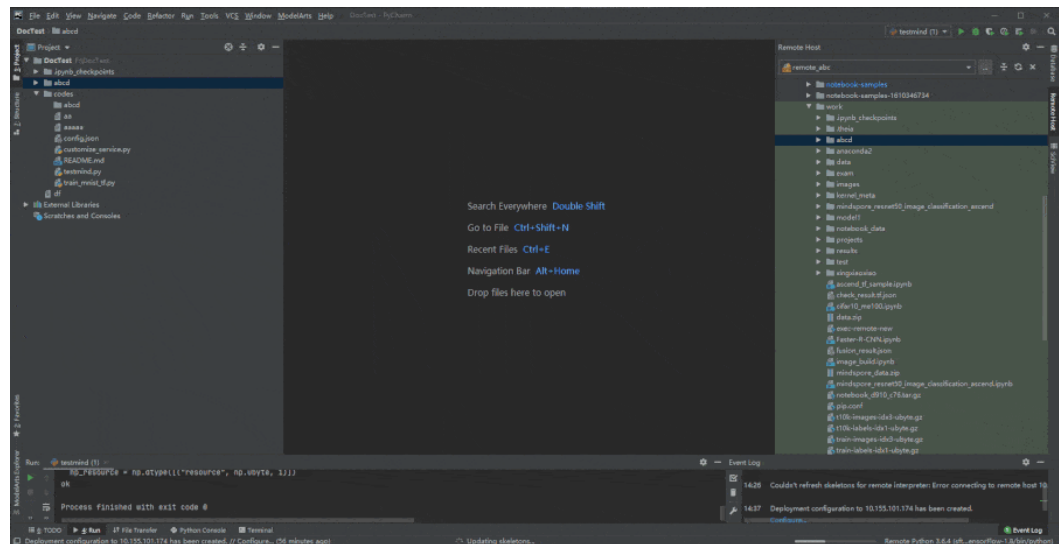
Address	ssh://ma-user@dev-modelarts- [redacted] .com:32651
Authentication	KeyPair-9a64  Access address of the development environment Port number

- The key pair is available.
A key pair is automatically downloaded after you create it. Securely store your key pair. If an existing key pair is lost, create a new one.

Step 1 Configure SSH

1. In your local PyCharm development environment, choose **File > Settings > Tools > SSH Configurations** and click **+** to add an SSH configuration.
 - **Host:** address for accessing the cloud development environment. Obtain the address on the page providing detailed information of the target notebook instance .
 - **Port:** port number for accessing the cloud development environment. Obtain the port number on the page providing detailed information of the target notebook instance.
 - **User name:** consistently set to **ma-user**.
 - **Authentication type:** key pair
 - **Private key file:** locally stored private key file of the cloud development environment. It is the key pair file automatically downloaded when you created the notebook instance.
2. Click  to rename the connection. Then, click **OK**.
3. After the configuration is complete, click **Test Connection** to test the connectivity.
4. Select **Yes**. If "Successfully connected" is displayed, the network is accessible. Then, click **OK**.
5. Click **OK** at the bottom to save the configuration.

Figure 3-86 Configuring SSH



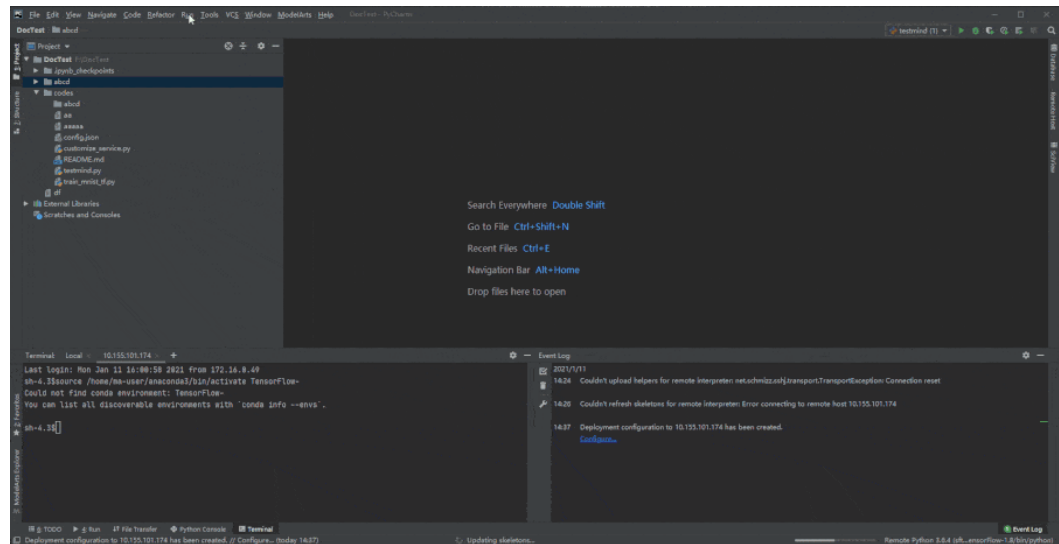
Step 2 Obtain the Path to the Virtual Environment Built in the Development Environment

1. Choose **Tools > Start SSH Session** to access the cloud development environment.
2. Run the following command to view the Python virtual environments built in the current environment in the **README** file in **/home/ma-user/**:

```
cat /home/ma-user/README
```

3. Run the **source** command to switch to a specific Python environment.
4. Run **which python** to obtain the Python path and copy it for configuring the Python interpreter on the cloud.

Figure 3-87 Obtaining the path to the virtual environment built in the development environment



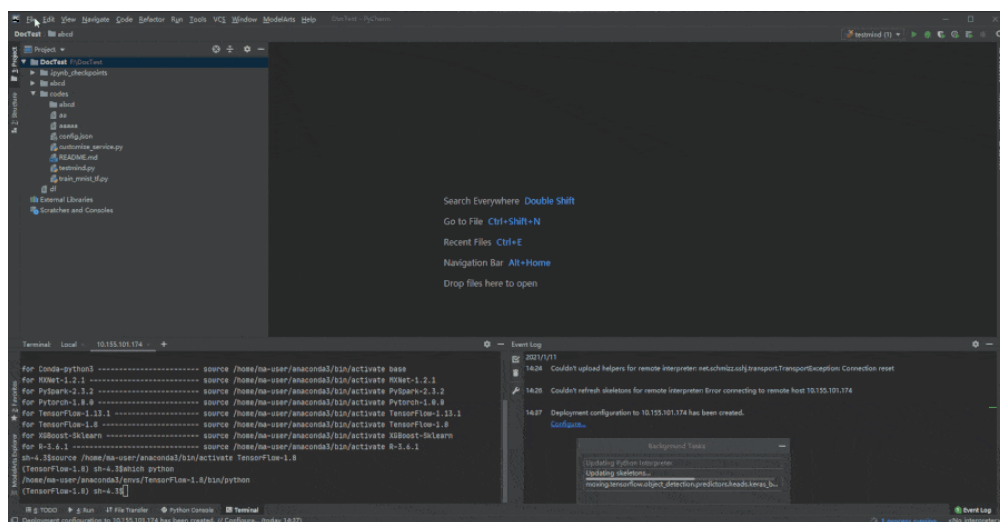
Step 3 Configure a Python Interpreter

1. Choose **File > Settings > Project: Python project > Python Interpreter**. Then, click  and **Add** to add an interpreter.
2. Select **Existing server configuration**, choose the SSH configuration from the drop-down list, and click **Next**.
3. Configure the Python interpreter.
 - **Interpreter:** Enter the Python path copied in step 1, for example, **/home/ma-user/anaconda3/envs/Pytorch-1.0.0/bin/python**.
If the path is **~/anaconda3/envs/Pytorch-1.0.0/bin/python**, replace **~** with **/home/ma-user**.
 - **Sync folders:** Set this parameter to a directory in the cloud development environment for synchronizing local project directory files. A directory in **/home/ma-user** is recommended, for example, **/home/ma-user/work/projects**, because other directories may be prohibited from accessing.
4. Click **!** on the right and select **Automatically upload** so that the locally modified file can be automatically uploaded to the container.
5. Click **Finish**.

The local project file has been automatically uploaded to the cloud environment. Each time a local file is modified, the modification is automatically synchronized to the cloud environment.

In the lower right corner, the current interpreter is displayed as a remote interpreter.

Figure 3-88 Configuring a Python interpreter



Step 4 Install the Dependent Library for the Cloud Environment

After accessing the development environment, you can use different virtual environments, such as TensorFlow and PyTorch. However, in actual development, you need to install dependency packages. Then, you can access the environment through the terminal to perform operations.

Choose **Tools > Start SSH Session** and select the configured development environment. Run the **pip install** command to install the required dependency packages.

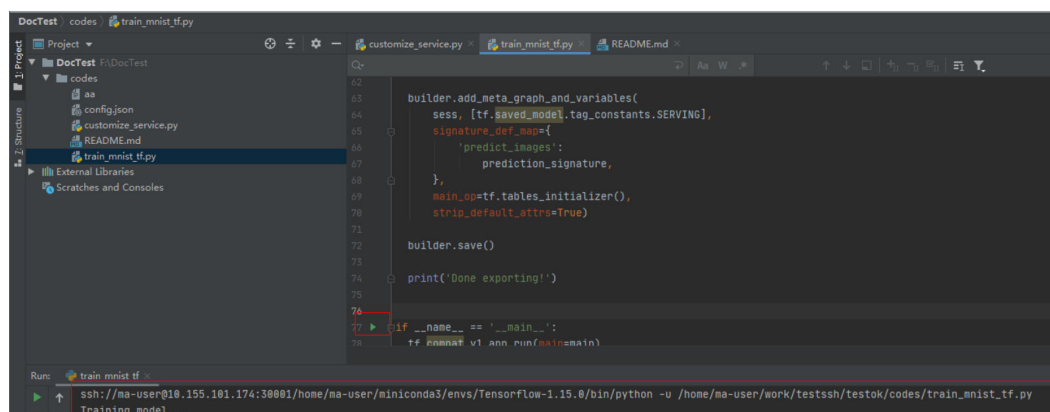
```
Terminal: Local x 10.155.101.174 x +
Last login: Wed Dec 30 12:46:18 2020 from 10.155.101.174
sh-4.4$cat /home/ma-user/README
Please use one of following command to start the specified framework environment.

for Mindspore-1.0.1-python3.7-aarch64 ----- source /home/ma-user/miniconda3/bin/activate Mindspore-1.0.1-python3.7-aarch64
for TensorFlow-1.15.0 ----- source /home/ma-user/miniconda3/bin/activate TensorFlow-1.15.0
sh-4.4$source /home/ma-user/miniconda3/bin/activate TensorFlow-1.15.0
(TensorFlow-1.15.0) sh-4.4$which python
~/miniconda3/envs/TensorFlow-1.15.0/bin/python
(TensorFlow-1.15.0) sh-4.4$pwd
/home/ma-user
(TensorFlow-1.15.0) sh-4.4$pip install spacy
```

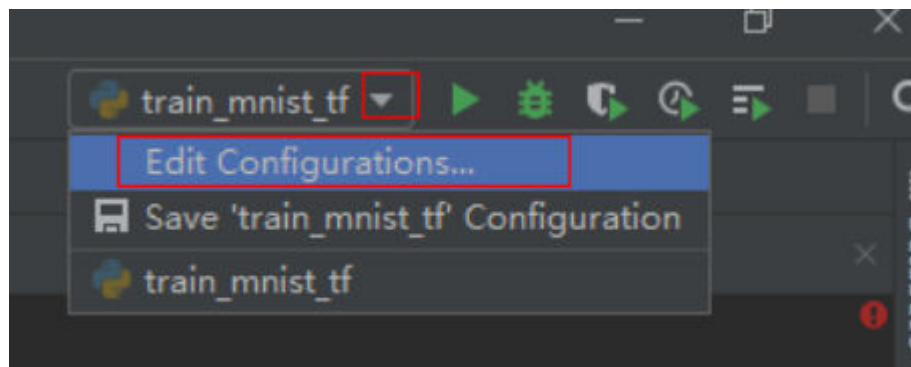
Step 5 Debug Code in the Development Environment

You have accessed the cloud development environment. Then, you can write, debug, and run the code in the local PyCharm. The code is actually executed in the cloud development environment, and the Ascend AI resources on the cloud are used. In this way, you compile and modify code locally and run the code in the cloud.

Run the code in the local IDE. The logs can be displayed locally.

Figure 3-89 Debugging code

Click **Run/Debug Configurations** in the upper right corner of the local IDE to set runtime parameters.

Figure 3-90 Setting runtime parameters

To debug code, set breakpoints and run the program in debug mode.

Figure 3-91 Code breakpoint

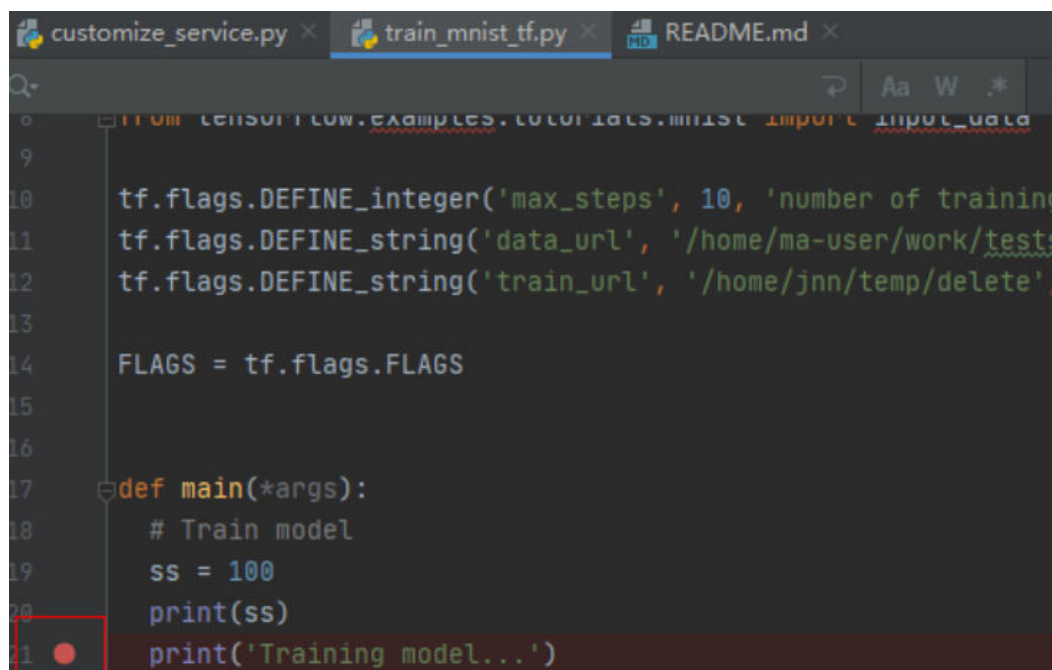
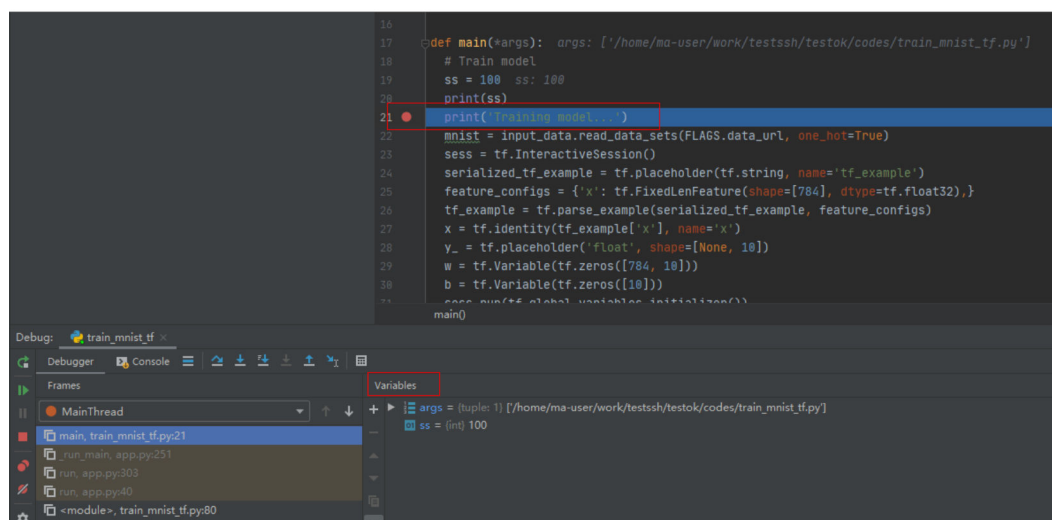


Figure 3-92 Debugging in debug mode



In debug mode, the code execution is suspended in the specified line, and you can obtain variable values.

Figure 3-93 Debug mode



Before debugging code in debug mode, ensure that the local code is the same as the cloud code. If they are different, the line where a breakpoint is added locally may be different from the line of the cloud code, leading to errors.

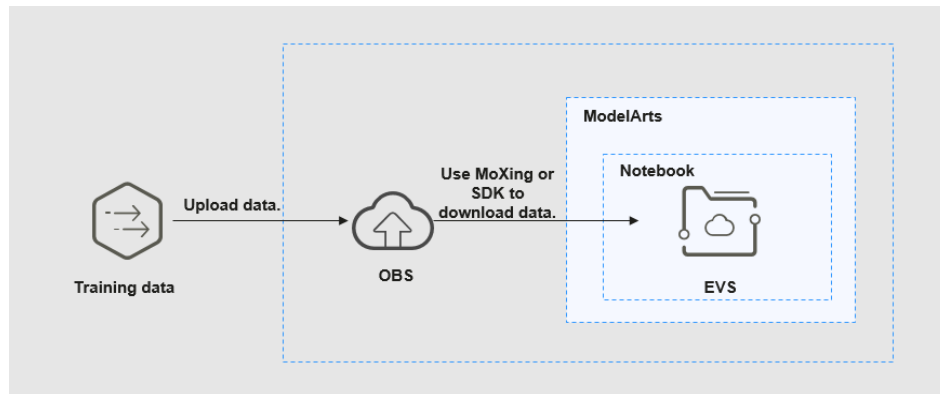
When configuring a Python interpreter in the cloud development environment, select **Automatically upload** so that any local file modification can be automatically uploaded to the cloud. If you do not select **Automatically upload**, manually upload the directory or code after you modify the local code. For details, see [Step 6 Uploading Local Files to a Notebook Instance](#).

3.4.3 Uploading Data to a Notebook Instance Through PyCharm

If the data is less than or equal to 500 MB, directly copy the data to the local IDE.

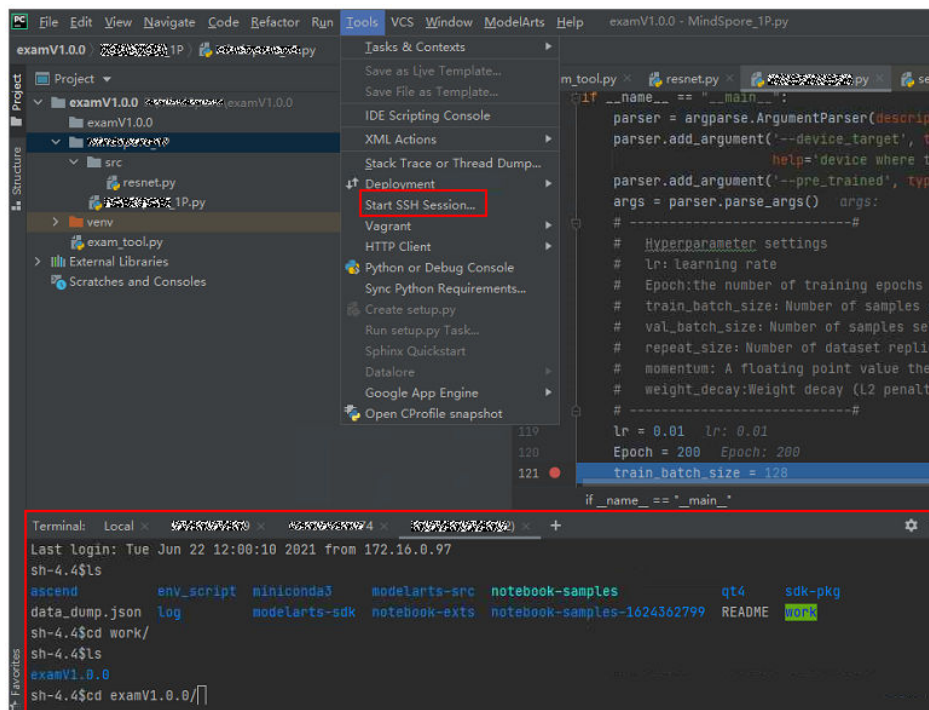
If the data is larger than 500 MB, upload it to OBS, and then download it to the notebook instance.

Figure 3-94 Uploading data to a notebook Instance through OBS



1. Upload data to OBS.
2. Call the **mox.file.copy_parallel** MoXing API provided by ModelArts in the terminal of the local IDE to transfer data from OBS to the notebook instance.
 - a. Open terminal in PyCharm. The operations in Visual Studio Code (VS Code) are similar.

Figure 3-95 Enabling the terminal in PyCharm



- b. The following shows how to use MoXing in the terminal of the local IDE to upload files from OBS to a notebook instance:

```
# Manually access the development environment.
cat /home/ma-user/README
# Select the source environment.
source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
# Enter python and press Enter to enter the Python environment.
python
# Use MoXing for access.
import moxing as mox
# Download a folder from OBS to EVS.
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```

3.5 Using Notebook Instances Remotely Through VS Code

3.5.1 Connecting to a Notebook Instance Through VS Code

VS Code is a typical code editor that supports multiple programming languages and development environments. You can connect to and use Jupyter Notebook through VS Code.

After creating a notebook instance with remote SSH enabled, you can use VS Code to access the development environment in any of the following ways:

- **Connecting to a Notebook Instance Through VS Code Toolkit**
In this mode, log in to the ModelArts VS Code Toolkit plug-in and use it to connect to an instance.
- **Manually Connecting to a Notebook Instance Through VS Code**

In this mode, use the VS Code Remote-SSH plug-in to configure connection information and connect to an instance.

Installing VS Code

Install VS Code first.

- **Download URL:**

Download address: https://code.visualstudio.com/updates/v1_85

Figure 3-96 VS Code download URL

November 2023 (version 1.85)

Update 1.85.1: The update addresses these [issues](#).

Update 1.85.2: The update addresses these [issues](#).

Downloads: Windows: [x64](#) [Arm64](#) | Mac: [Universal Intel silicon](#) | Linux: [deb](#) [rpm](#) [tarball](#) [Arm snap](#)

- **VS Code version requirements:**

You are advised to use VS Code 1.85.2 or the latest version for remote connection.

- **VS Code installation guide:**

In Linux, run the command `sudo dpkg -i code_1.85.2-1705561292_amd64.deb` to install VS Code.

 NOTE

Linux system users must install VS Code as a non-root user.

3.5.2 Installing VS Code

Download URL:

Download address: https://code.visualstudio.com/updates/v1_85

Figure 3-97 VS Code download URL

November 2023 (version 1.85)

Update 1.85.1: The update addresses these [issues](#).

Update 1.85.2: The update addresses these [issues](#).

Downloads: Windows: [x64](#) [Arm64](#) | Mac: [Universal Intel silicon](#) | Linux: [deb](#) [rpm](#) [tarball](#) [Arm snap](#)

VS Code version requirements:

Use VS Code 1.85.2 for remote connection.

VS Code installation guide:

In Linux, run the command `sudo dpkg -i code_1.85.2-1705561292_amd64.deb` to install VS Code.

 NOTE

Linux system users must install VS Code as a non-root user.

3.5.3 Connecting to a Notebook Instance Through VS Code Toolkit

This section describes how to use the ModelArts VS Code Toolkit plug-in to remotely connect to a notebook instance.

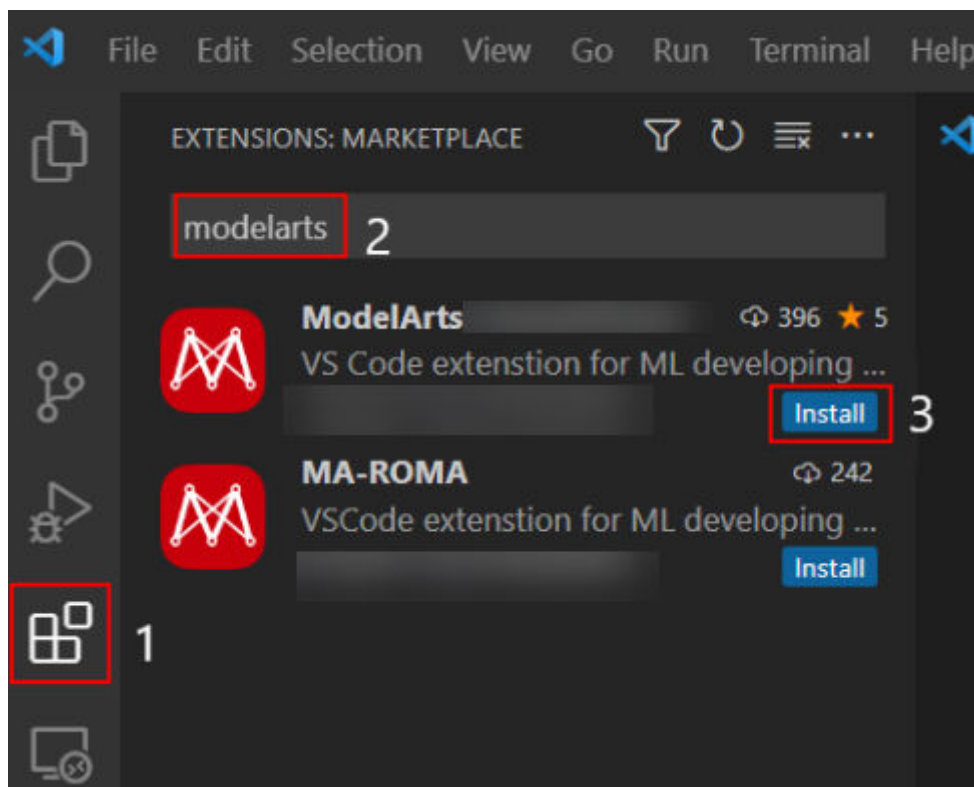
Prerequisites

You have downloaded and installed VS Code. For details, see [Installing VS Code](#).

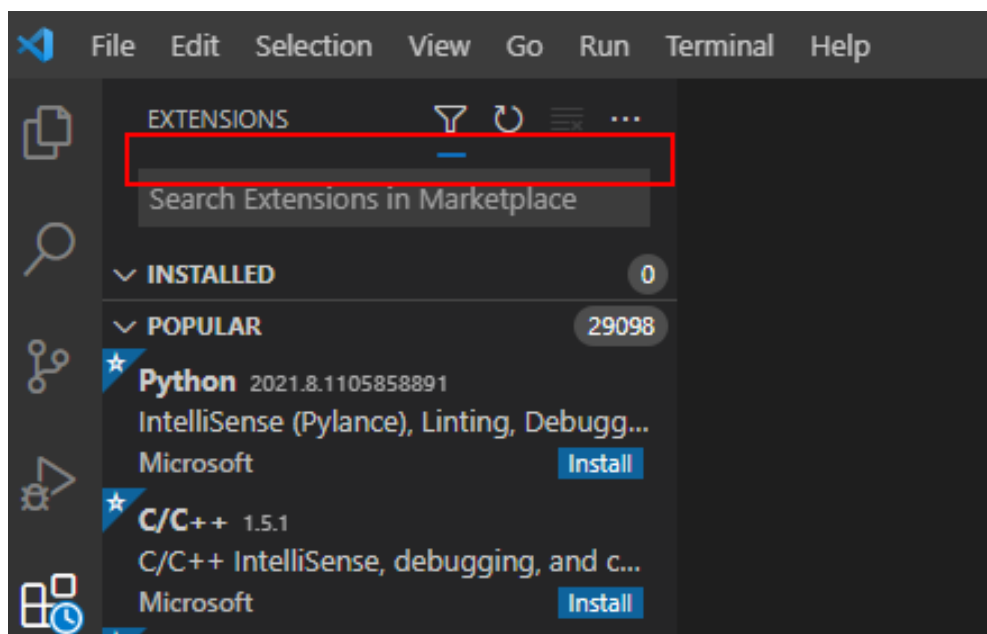
Step 1 Install the VS Code Plug-in

1. Search for **ModelArts** in the **EXTENSIONS** text box and click **Install**.

Figure 3-98 Installing the VS Code plug-in



2. Wait for about 1 to 2 minutes.

Figure 3-99 Installation process

3. After the installation is complete, check the message displayed in the lower right corner. If the ModelArts icon  and remote SSH icon  are displayed in the navigation pane on the left, the VS Code plug-in is installed.

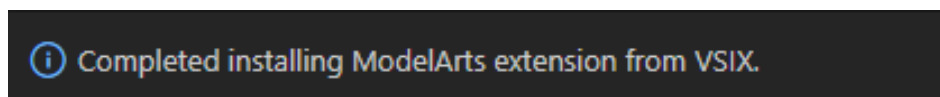
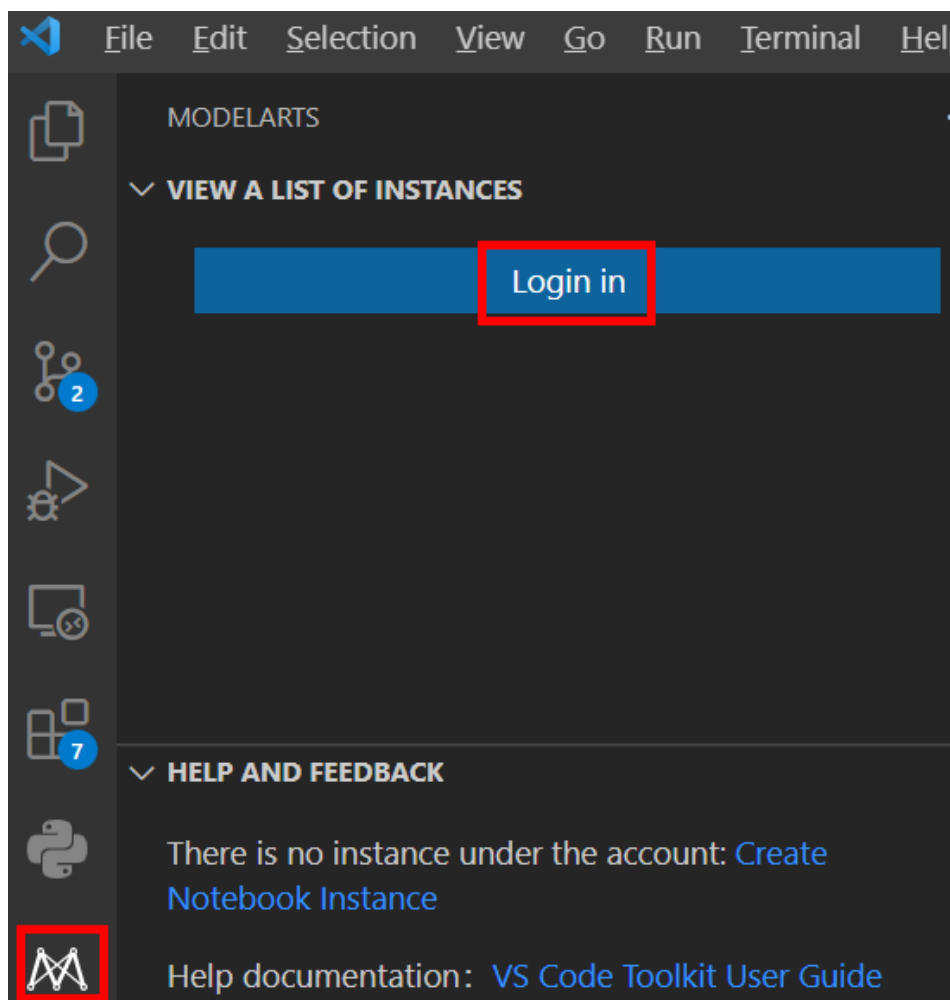
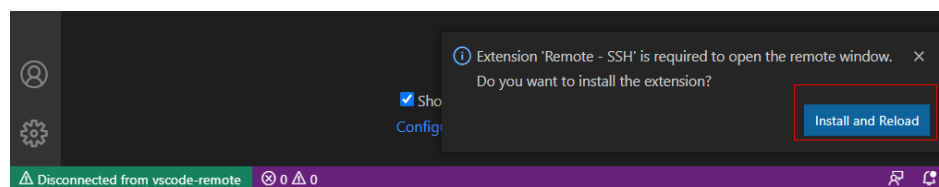
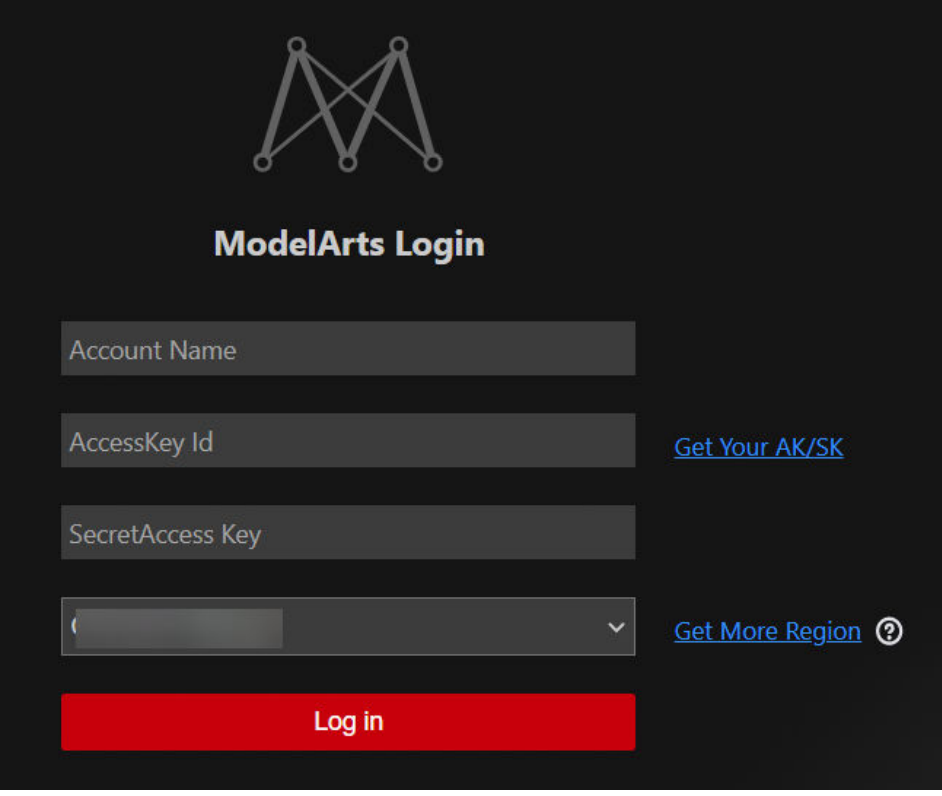
Figure 3-100 Installation completion message

Figure 3-101 Installation completed

Network issues may cause an installation failure. If this occurs, proceed with follow-up operations. After **1** in **Step 4 Access the Notebook Instance** is performed, the system will automatically display a dialog box shown in the following figure. In this case, click **Install and Reload**.

Figure 3-102 Reconnecting remote SSH

1. In the local VS Code development environment, click  and **User Settings**, and configure the login information.

Figure 3-103 Logging in to the plug-inThe image shows the ModelArts Login interface. At the top, there is a logo consisting of a stylized 'M' made of lines and dots. Below the logo, the text 'ModelArts Login' is displayed. The login form includes four input fields: 'Account Name', 'AccessKey Id', 'SecretAccess Key', and a region dropdown menu. To the right of the 'AccessKey Id' field is a link 'Get Your AK/SK'. To the right of the region dropdown is a link 'Get More Region' with a question mark icon. At the bottom of the form is a red 'Log in' button.

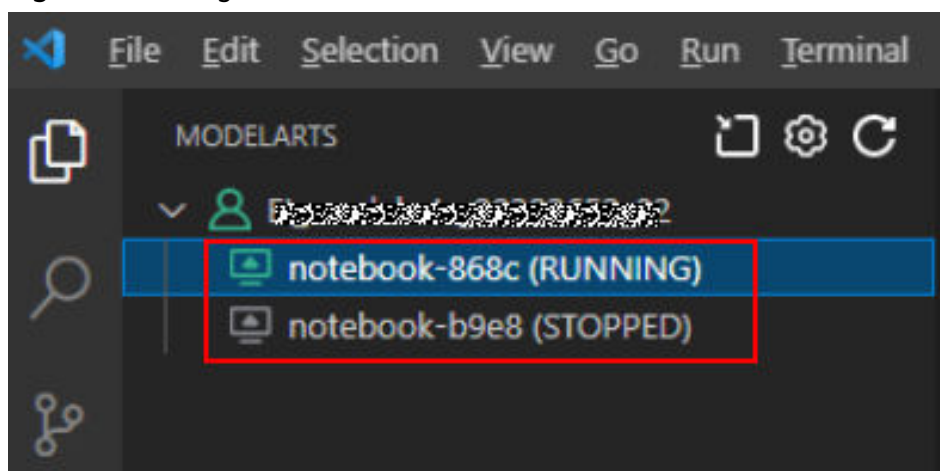
Enter the login information and click **Log in**.

- **Name:** Custom username, which is displayed only on the VS Code page and is not associated with any account.
- **AK and SK:** Access key pair. To create a key pair, choose **My Credentials > API Credentials > Access Keys**, and click **Create Access Key**.
- **Region:** must be the same as that of the notebook instance to be remotely connected. Otherwise, the connection will fail.

2. After the login, check the notebook instance list.

 NOTE

The list displays only notebook instances in the default workspace on the ModelArts console.

Figure 3-104 Login succeeded**CAUTION**

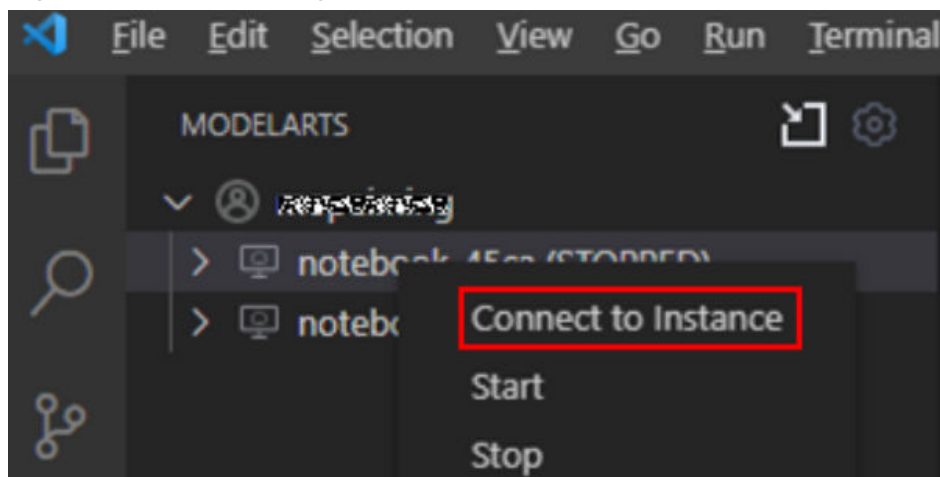
- Create a notebook instance with remote SSH enabled, and download the key file to either of the following directories based on your OS:
Windows: `C:\Users\{{user}}`
macOS/Linux: `Users/{{user}}`
- A key pair is automatically downloaded after you create it. Securely store your key pair. If an existing key pair is lost, create a new one.

Create a notebook instance with remote SSH enabled. For details, see [Creating a Notebook Instance](#).

Step 4 Access the Notebook Instance

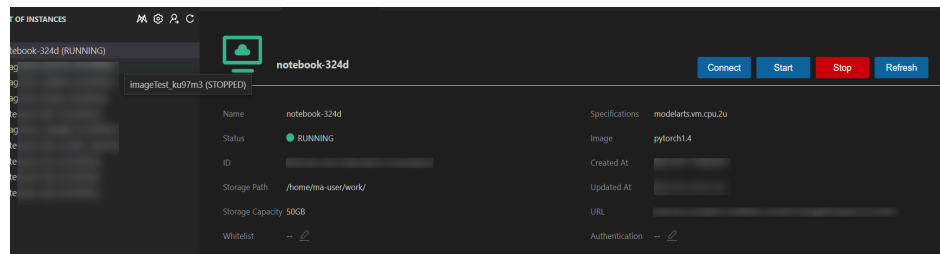
1. In the local VS Code development environment, right-click the instance name and choose **Connect to Instance** from the shortcut menu to start and connect to the notebook instance.

The notebook instance can either be running or stopped. If it is stopped, the VS Code plug-in starts the instance and then connects to it.

Figure 3-105 Connecting to a notebook instance

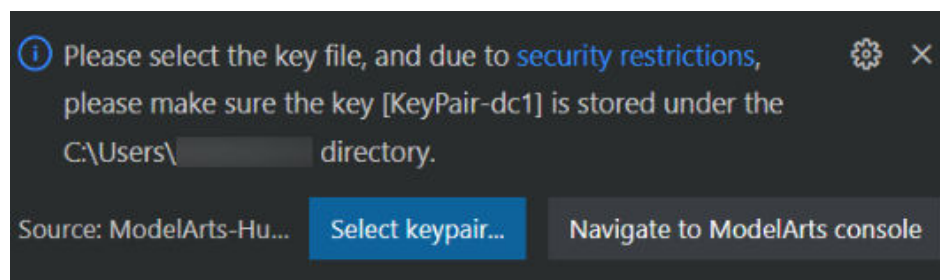
Alternatively, click the instance name. On the instance details page, click **Connect**. Then, the system automatically starts and connects to the notebook instance.

Figure 3-106 Viewing details about a notebook instance



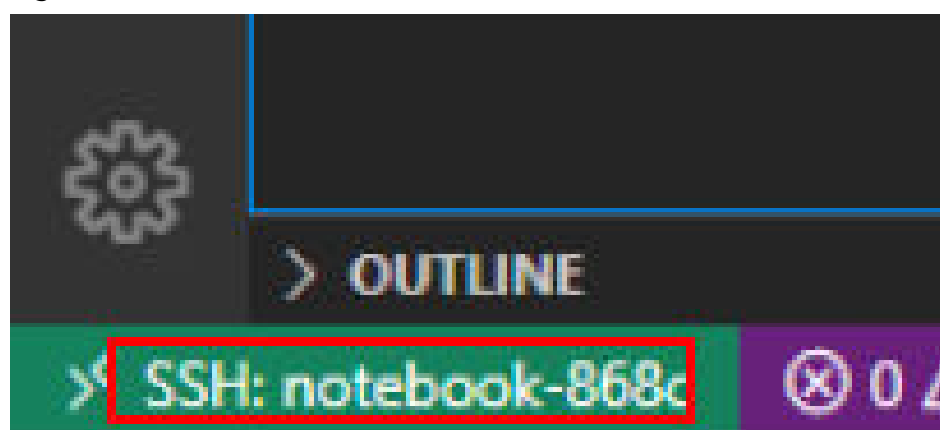
2. When you connect to a notebook instance for the first time, the system prompts you in the lower right corner to configure the key file. In this case, select the local .pem key file and click **OK**.

Figure 3-107 Configuring the key file



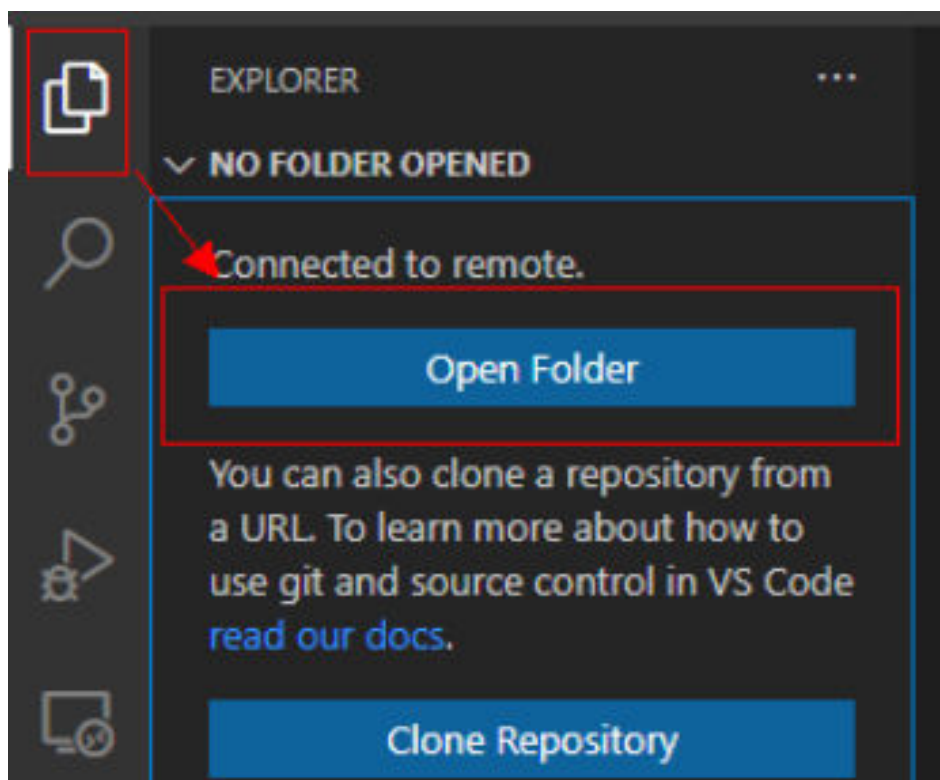
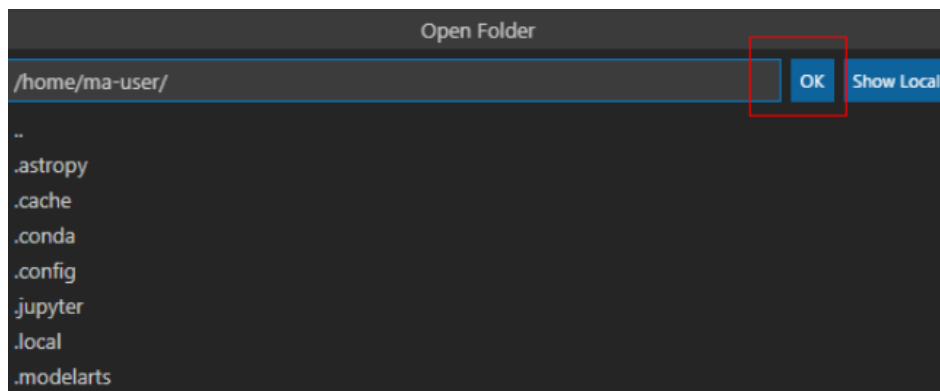
3. Wait for about 1 to 2 minutes until the notebook instance is accessed. After information similar to the following is displayed in the lower left corner of the VS Code environment, the connection is succeeded.

Figure 3-108 Connection succeeded

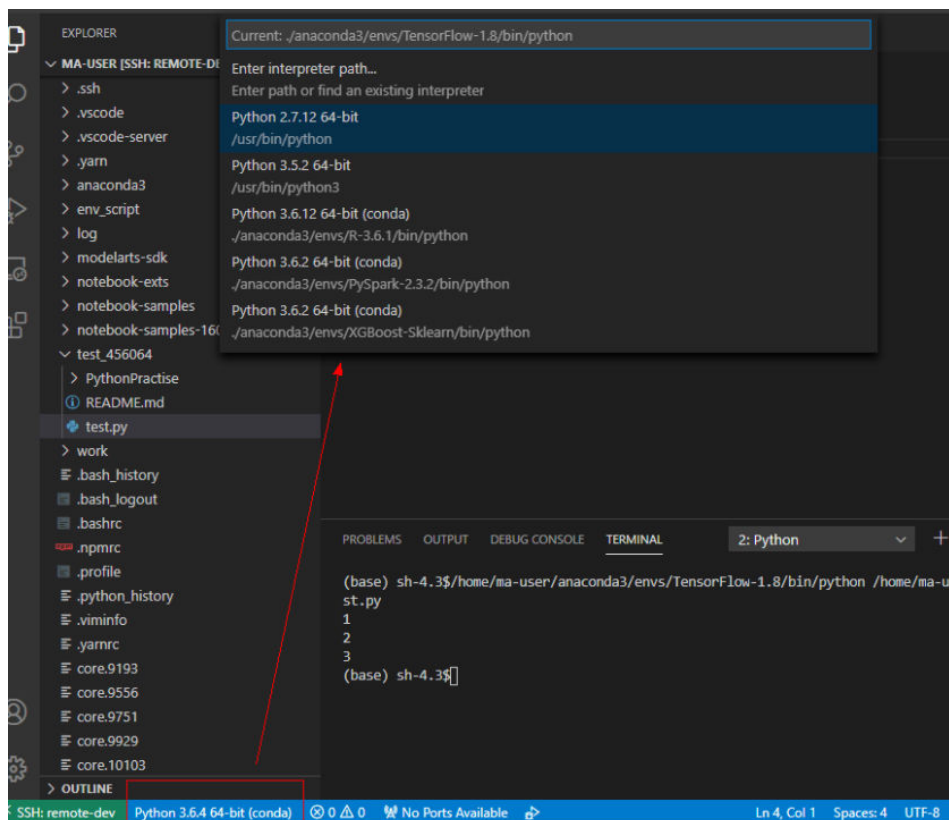


Debugging Code Remotely

1. On the VS Code page, upload local code to the cloud development environment.
 - a. Choose **File > OpenFolder**, select the path to be opened, and click **OK**.

Figure 3-109 Open Folder**Figure 3-110** Selecting a file path

- b. In the displayed directory structure on the left of the IDE, drag the code and files you want to upload to the corresponding folders. Then, the code is uploaded to the cloud development environment.
- c. Open the code file to be debugged in VS Code. Before running the code, click the default Python version in the lower left part and select a version as required.

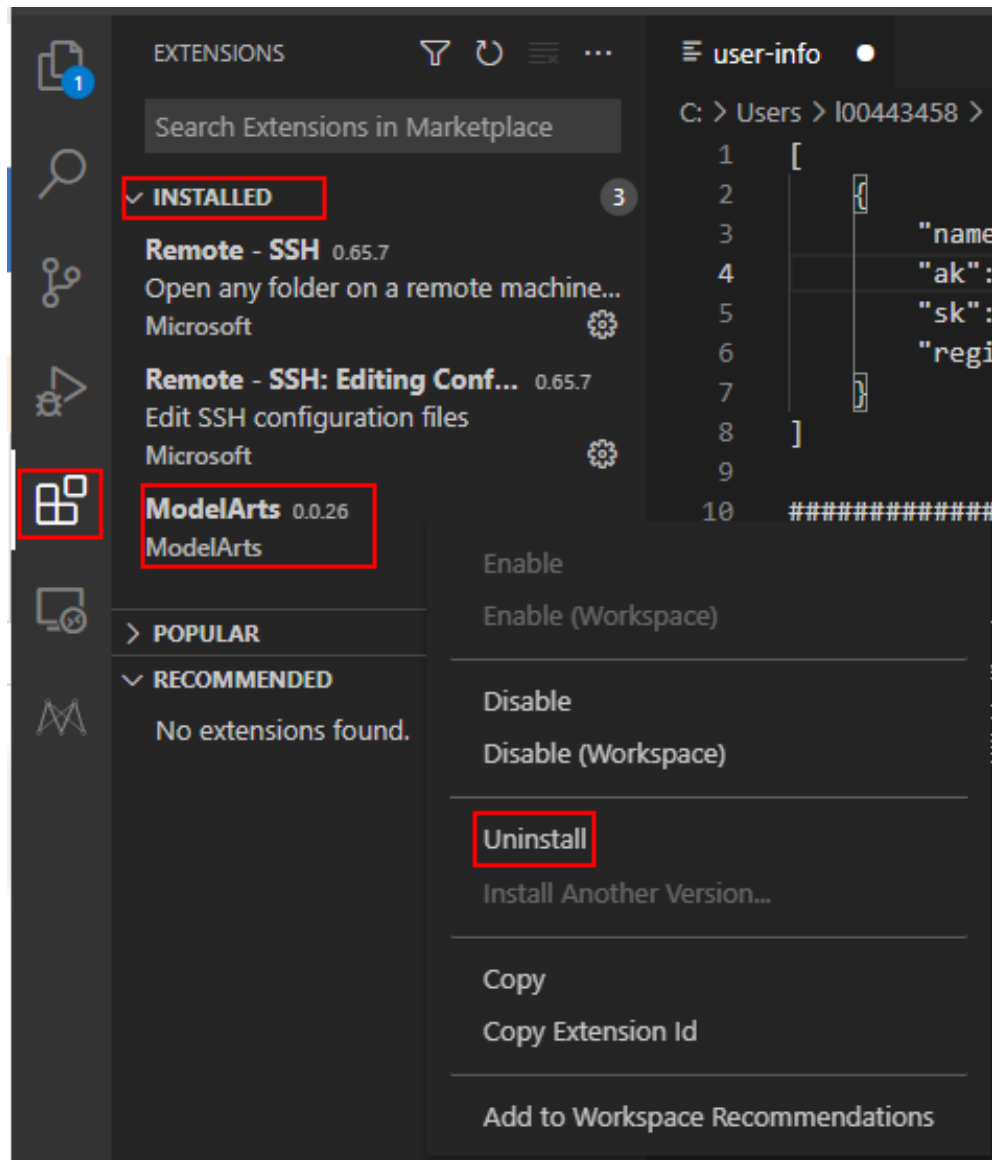
Figure 3-111 Selecting a Python version

2. Click the execution button to run the code. The code output is shown in the **TERMINAL** tab.
 - If a training job takes a long time to execute, run the job at the backend through the nohup command. This prevents the disconnection of an SSH session or a network failure from affecting job execution. The following shows an example nohup command:

```
nohup your_train_job.sh > output.log 2>&1 & tail -f output.log
```
 - To debug the code, perform the following operations:
 - i. Choose **Run > Run and Debug** on the left.
 - ii. Select the default Python code file.
 - iii. Click on the left of the code to set breakpoints.
 - iv. Debug the code according to the debug procedure which is displayed above the code, and the debug information is displayed on the left of the page.

Related Operations

For details about uninstalling the VS Code plug-in, see [Figure 3-112](#).

Figure 3-112 Uninstalling the VS Code plug-in

3.5.4 Manually Connecting to a Notebook Instance Through VS Code

A local IDE supports PyCharm and VS Code. You can use PyCharm or VS Code to remotely connect the local IDE to the target notebook instance on ModelArts for running and debugging code.

This section describes how to use VS Code to access a notebook instance.

Prerequisites

- You have downloaded and installed VS Code. For details, see [Installing VS Code](#).
- Python has been installed on your local PC or server. For details, see [VS Code official documentation](#).

- A notebook instance has been created with remote SSH enabled. Ensure that the instance is running. For details, see [Creating a Notebook Instance](#).
- The address and port number of the development environment are available. To obtain the information, go to the notebook instance details page.

Figure 3-113 Instance details page

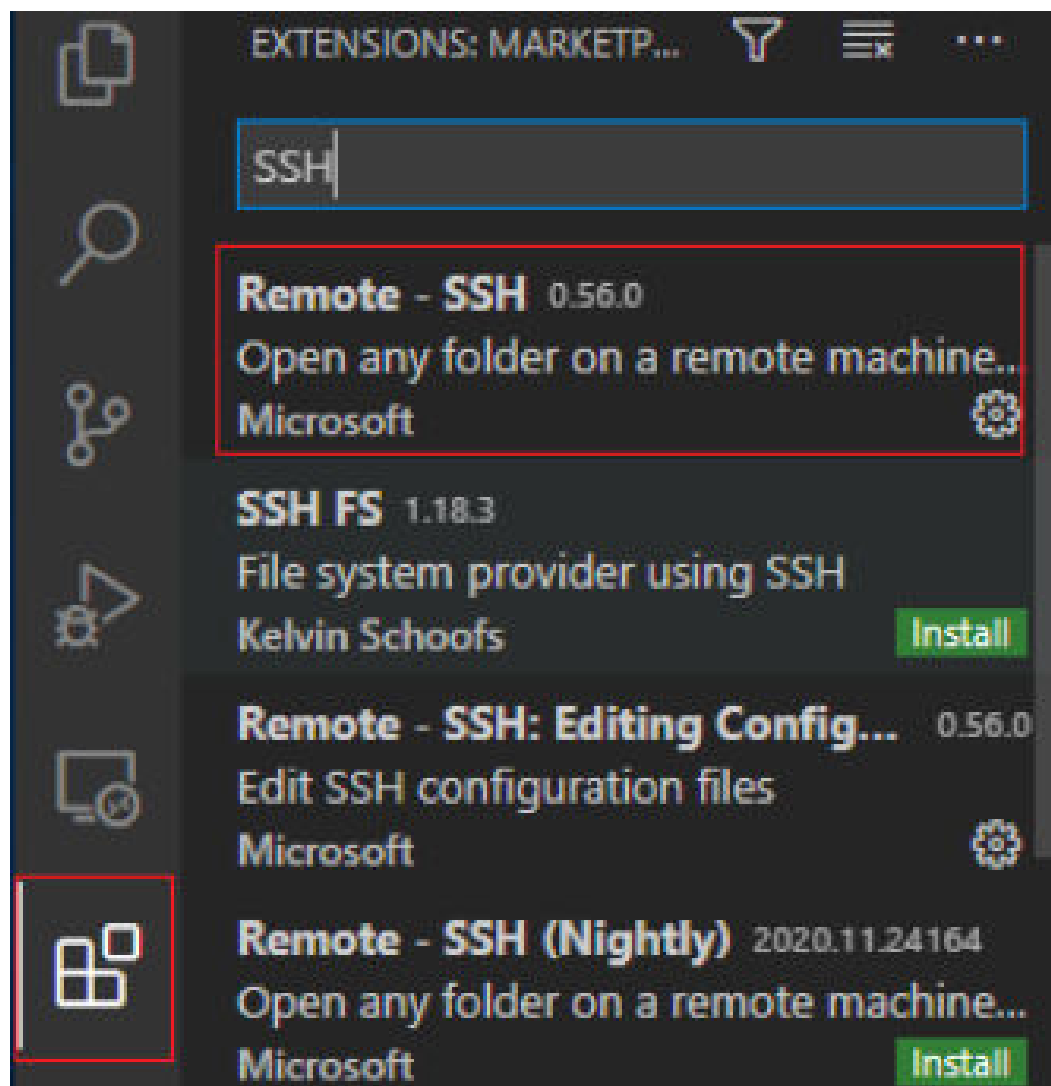


The screenshot shows the 'Instance details' page. It displays the 'Address' field with the value 'ssh://ma-user@dev-modelarts-' followed by a red box containing '30581'. Below the 'Address' field, the 'Authentication' field shows 'KeyPair-e744'. Red labels 'Address' and 'Port number' are positioned below the respective fields.

- The key pair is available.
A key pair is automatically downloaded after you create it. Securely store your key pair. If an existing key pair is lost, create a new one.

Step 1 Add the Remote-SSH Plug-in

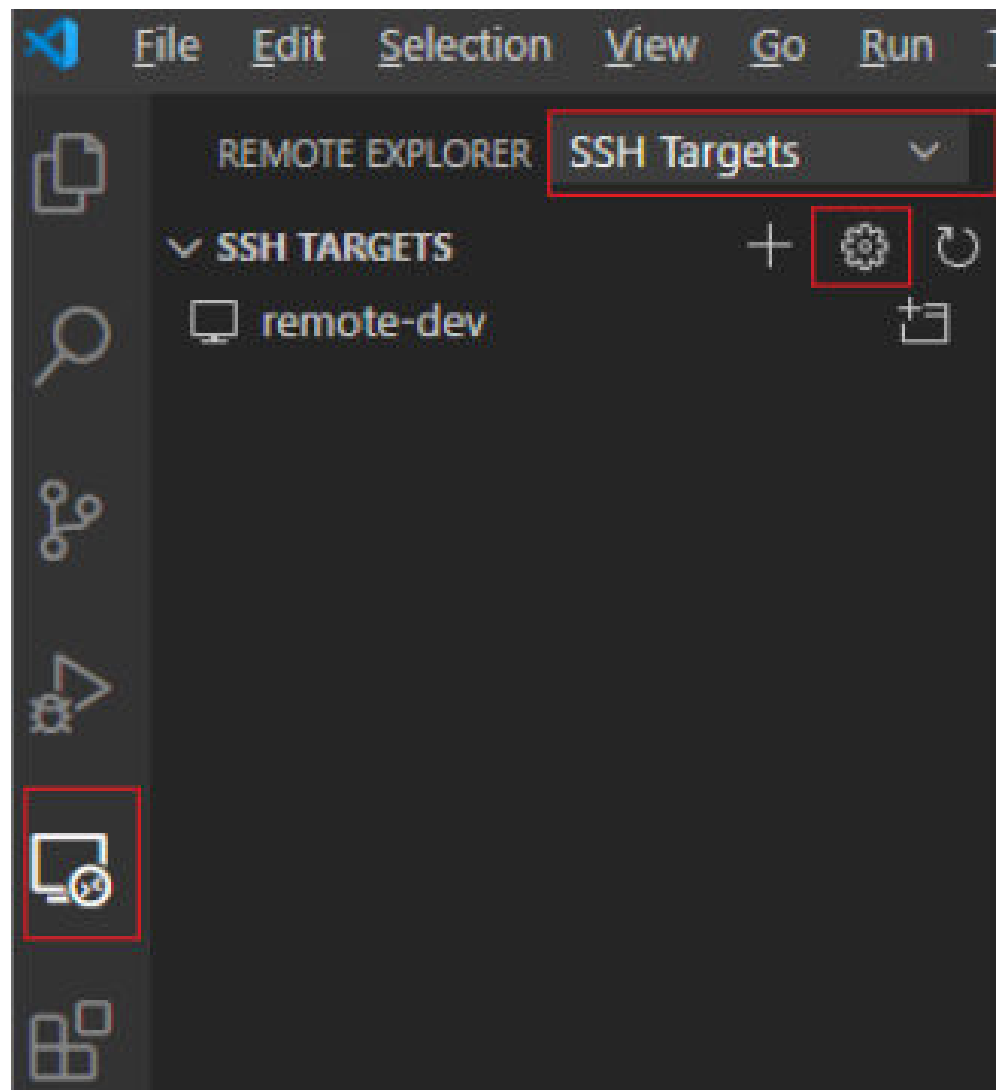
In the local VS Code development environment, click , enter **SSH** in the search box, and click **install** of the Remote-SSH plug-in to install the plug-in.

Figure 3-114 Adding the Remote-SSH plug-in

Step 2 Configure SSH

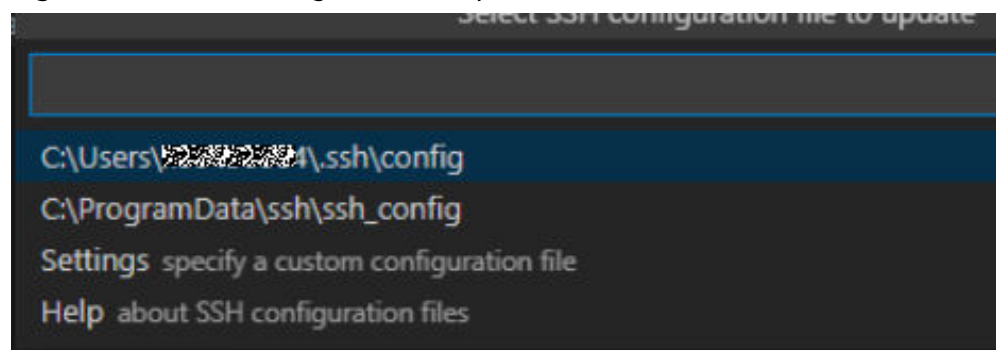
1. In the local VS Code development environment, click  on the left, select **SSH Targets** from the drop-down list box, and click . The SSH configuration file path is displayed.

Figure 3-115 Configuring SSH Targets



2. Click the SSH configuration path and configure SSH.

Figure 3-116 SSH configuration file path

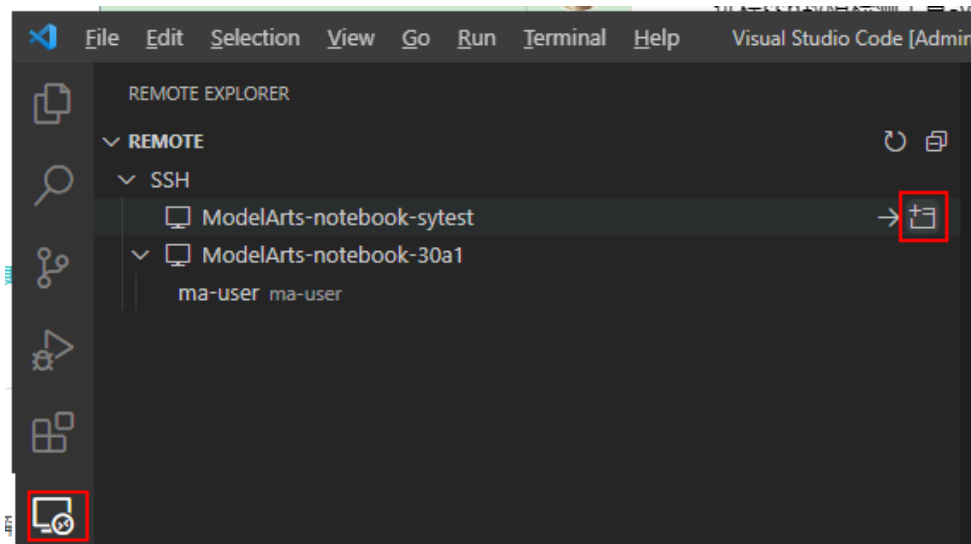


```
HOST remote-dev
  hostname <Instance connection host>
  port <Instance connection port>
  user ma-user
  IdentityFile ~/.ssh/test.pem
```

```
UserKnownHostsFile=/dev/null  
StrictHostKeyChecking no
```

- **HOST:** name of the cloud development environment
 - **HostName:** address for accessing the cloud development environment. Obtain the address on the page providing detailed information of the target notebook instance.
 - **Port:** port number for accessing the cloud development environment. Obtain the port number on the page providing detailed information of the target notebook instance.
 - **user:** **ma-user**
 - **IdentityFile:** locally stored private key file of the cloud development environment. It is the key pair file in [Prerequisites](#).
3. Return to the **SSH Targets** page, locate the target remote development environment, and click  on the right to connect to host in a new window.

Figure 3-117 Opening the development environment



On the displayed page, click **Linux**.

Figure 3-118 Selecting a notebook running environment

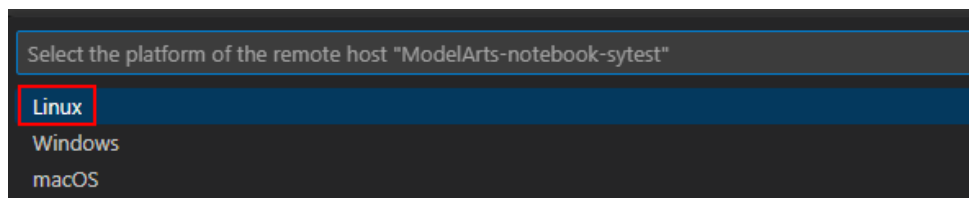
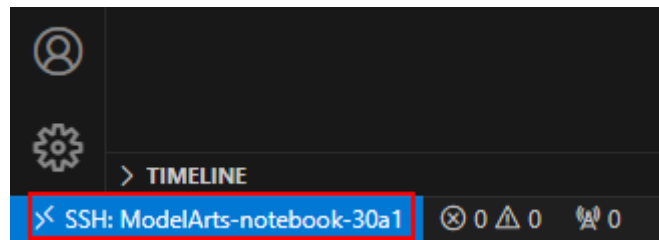
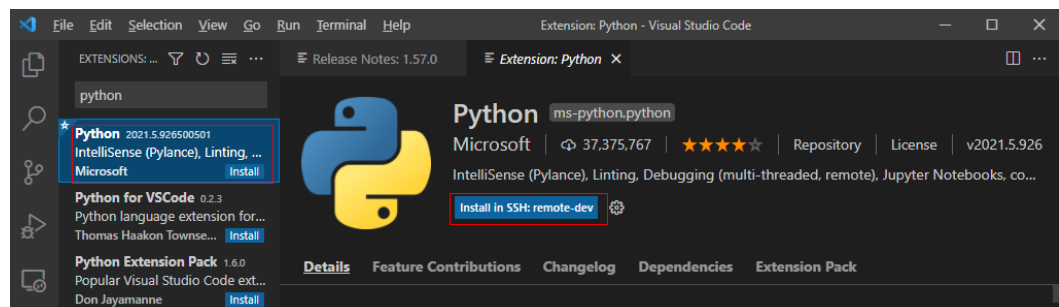


Figure 3-119 Remote connection succeeded

Step 3 Install the Python Plug-in in the Cloud Development Environment

On the displayed VS Code page, click  on the left, enter **Python** in the search box, and click **Install**.

Figure 3-120 Installing the Python plug-in in the cloud development environment

If the Python plug-in fails to be installed on the cloud, install it using an offline package.

Step 4 Install the Dependent Library for the Cloud Environment

After accessing the container environment, you can use different virtual environments, such as TensorFlow and PyTorch. However, in actual development, you need to install dependency packages. Then, you can access the environment through the terminal to perform operations.

1. In VS Code, press **Ctrl+Shift+P**.
2. Search for **Python: Select Interpreter** and select the target Python.
3. Choose **Terminal > New Terminal**. The CLI of the remote container is displayed.
4. Run the following command to install the dependency package:

```
pip install spacy
```

3.5.5 Uploading and Downloading Files in VS Code

Uploading Data to a Notebook Instance Using VS Code

If the data is less than or equal to 500 MB, directly copy the data to the local IDE.

If the data is larger than 500 MB, upload it to OBS and then to the notebook instance.

Procedure

1. Upload data to OBS.

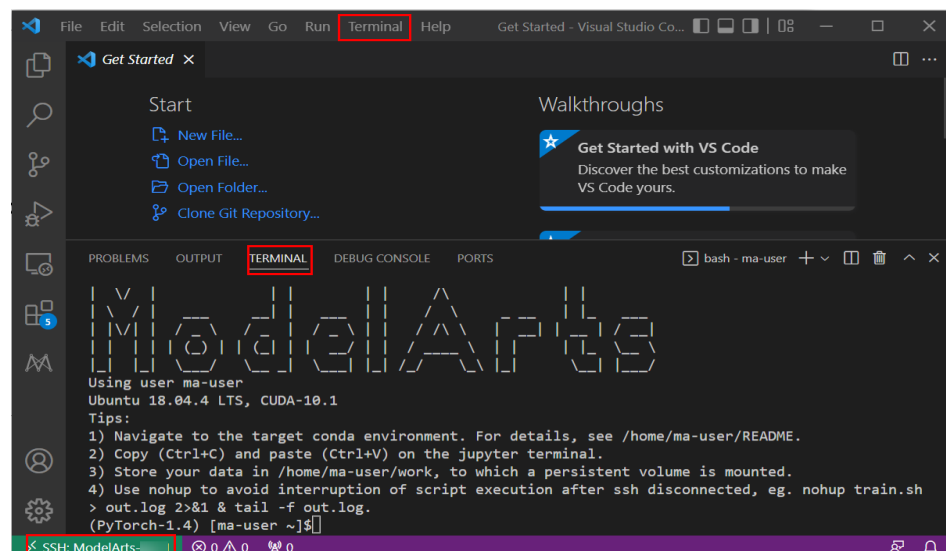
Alternatively, use ModelArts SDK in the terminal of local VS Code to upload data to OBS. To do so, click **Terminal** on the top menu bar. Enter **python** and press **Enter** in terminal to access the Python environment.

```
python
```

In the terminal of the local VS Code, use ModelArts SDK to upload the target local file to OBS. For details, .

2. Upload the OBS file to the notebook instance. Use ModelArts SDK in the terminal of the remote VS Code environment to upload the file from OBS to a notebook instance.

Figure 3-121 Opening the terminal in the remote VS Code environment

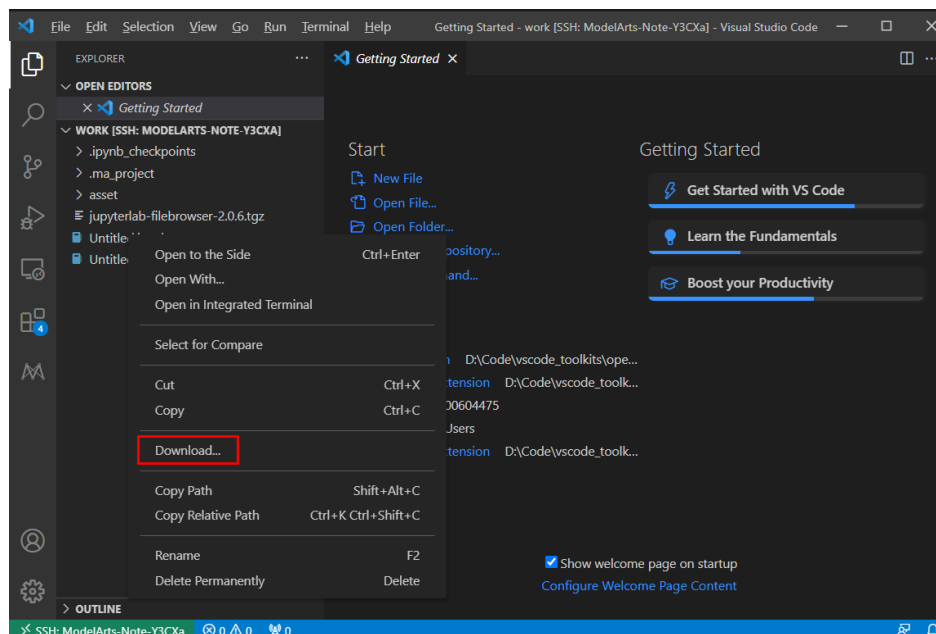


```
# Manually access the development environment using the source command.
cat /home/ma-user/README
# Select the target environment.
source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
# Enter python and press Enter to access the Python environment.
python
```

Then, perform OBS transfer operations by referring to .

Downloading Files from a Notebook Instance to a Local Directory

Files created in Notebook can be downloaded to a local path. In the **Project** directory of the local IDE, right-click the **Notebook2.0** project and choose **Download** from the shortcut menu to download the project file to the local PC.

Figure 3-122 Downloading files from a notebook instance to a local directory in VS Code

3.6 Using a Notebook Instance Remotely with SSH

This section describes how to use PuTTY to remotely log in to a notebook instance on the cloud in the Windows environment.

Prerequisites

- You have created a notebook instance with remote SSH enabled and whitelist configured. Ensure that the instance is running. For details, see [Creating a Notebook Instance](#).
- The address and port number of the development environment are available. To obtain this information, go to the notebook instance details page.

Figure 3-123 Instance details page

Address	ssh://ma-user@dev-modelarts- .com 32651
Authentication	KeyPair-9a64 Access address of the development environment Port number

- The key pair is available.
A key pair is automatically downloaded after you create it. Securely store your key pair. If an existing key pair is lost, create a new one.

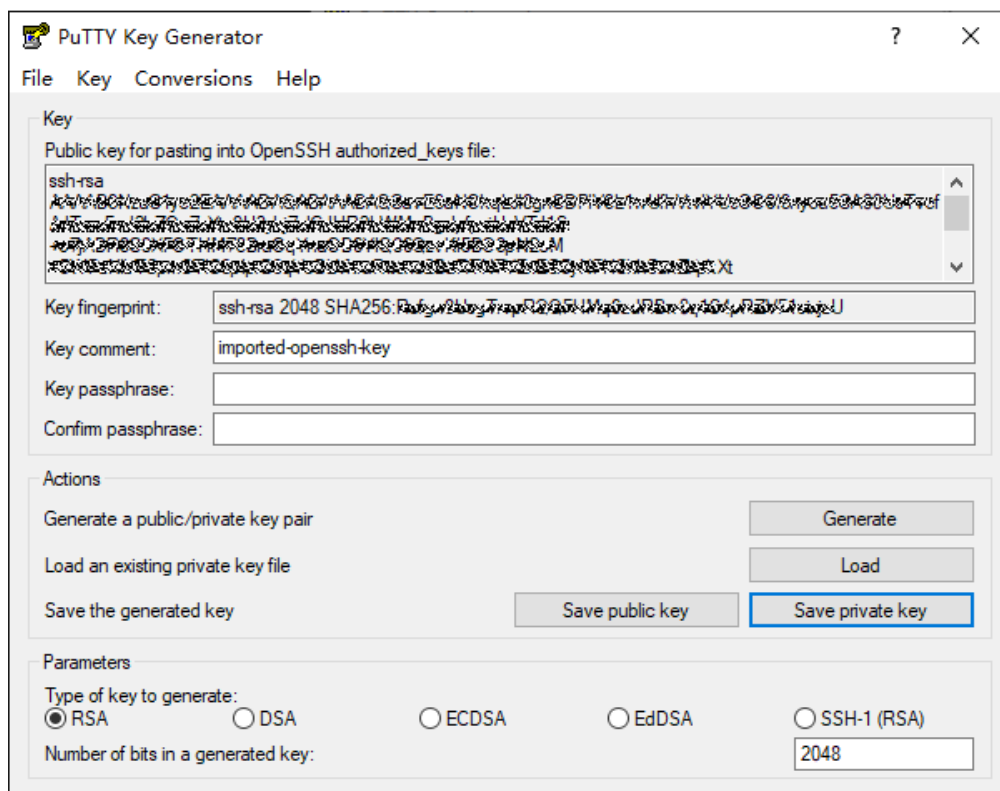
Step 1 Install the SSH Tool

[Download](#) and install the SSH remote access tool, for example, PuTTY.

Step 2 Use PuTTYgen to Convert the .pem Key Pair File to a .ppk Key Pair File

1. [Download PuTTYgen](#) and double-click it to run it.
2. Click **Load** to load the .pem key file created and saved during notebook instance creation.
3. Click **Save private key** to save the generated .ppk file. The file name can be customized, for example, **key.ppk**.

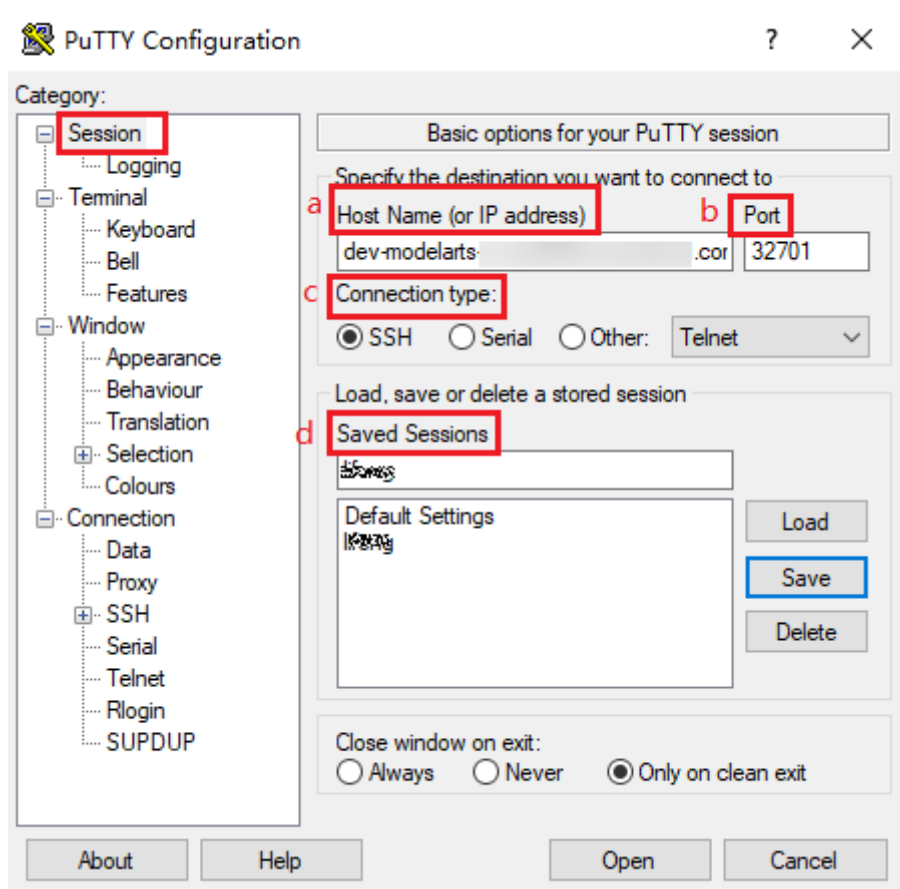
Figure 3-124 Converting the .pem key pair file to a .ppk key pair file



Step 3 Use SSH to Connect to a Notebook Instance

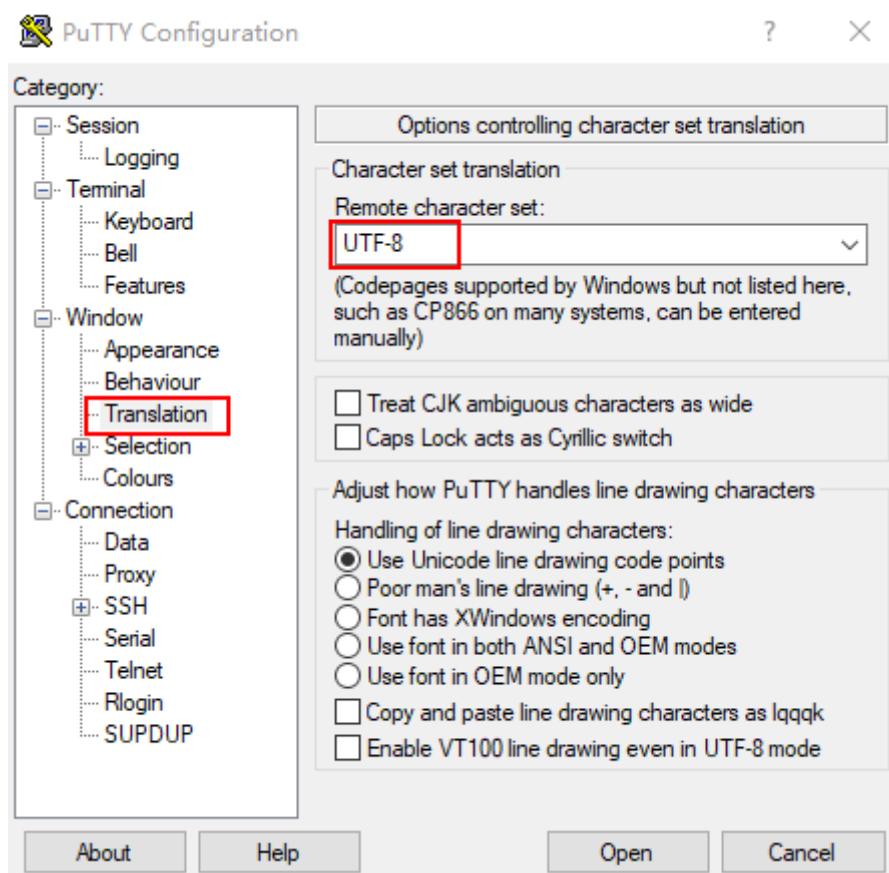
1. Run PuTTY.
2. Click **Session** and set the following parameters:
 - a. **Host Name (or IP address)**: address for accessing the in-cloud notebook instance. Obtain the address on the page providing detailed information of the target notebook instance .
 - b. **Port**: port number for accessing the in-cloud notebook instance. Obtain the port number on the page providing detailed information of the target notebook instance, for example, **32701**.
 - c. **Connection type**: Choose **SSH**.
 - d. **Saved Sessions**: task name, which can be clicked for remote access when you use PuTTY next time

Figure 3-125 Configuring Session



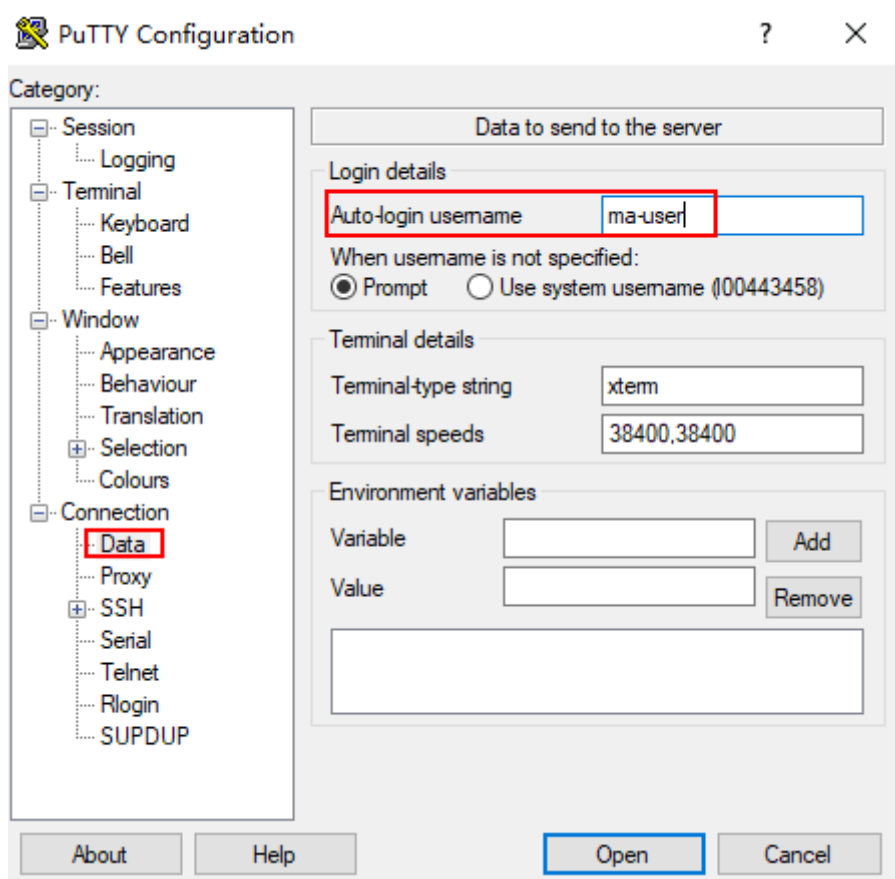
3. Choose **Window > Translation** and select **UTF-8** from the drop-down list box in the **Remote character set** area.

Figure 3-126 Setting the character format

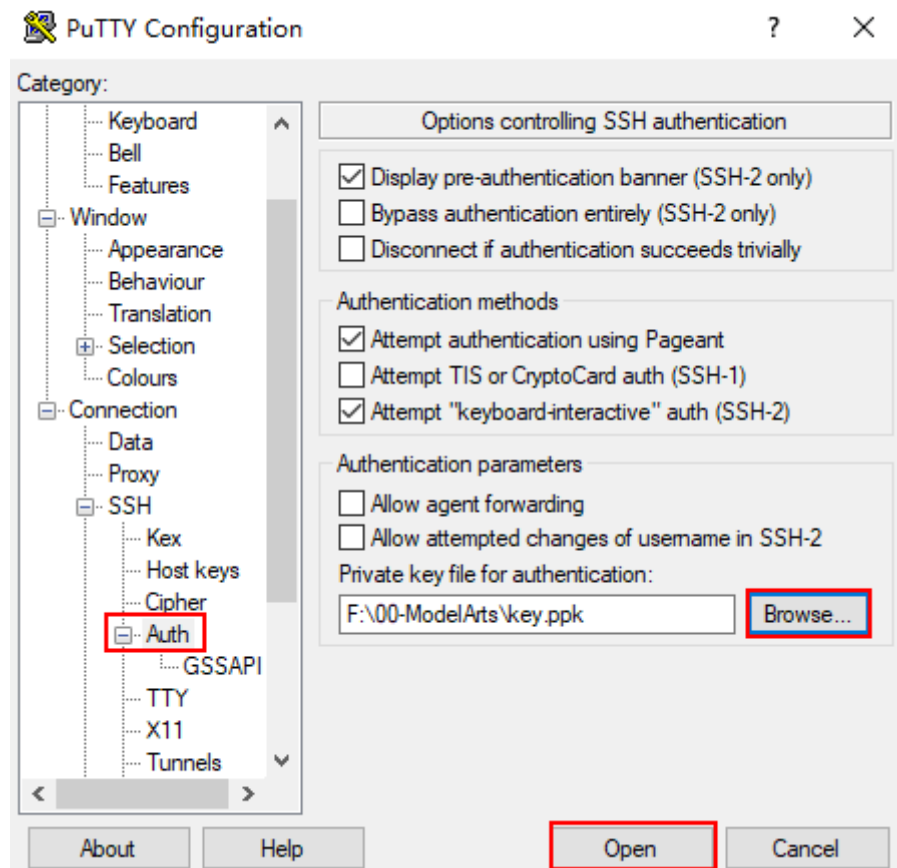


4. Choose **Connection** > **Data** and enter **ma-user** for **Auto-login username**.

Figure 3-127 Entering a username

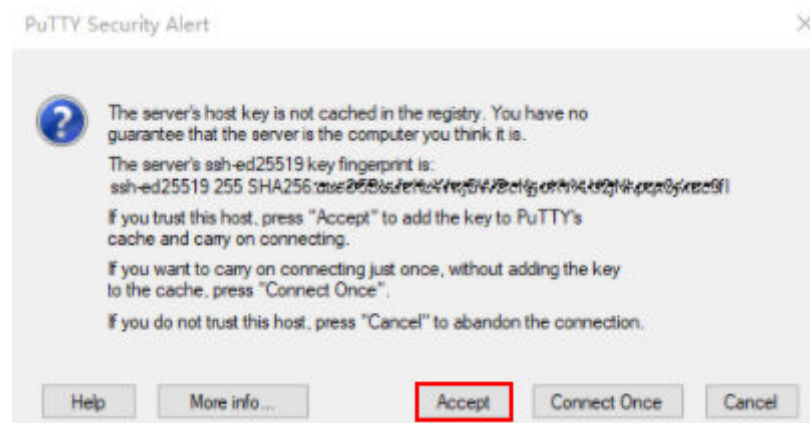


5. Choose **Connection** > **SSH** > **Auth**, click **Browse**, and select the .ppk file generated in [step 2](#).



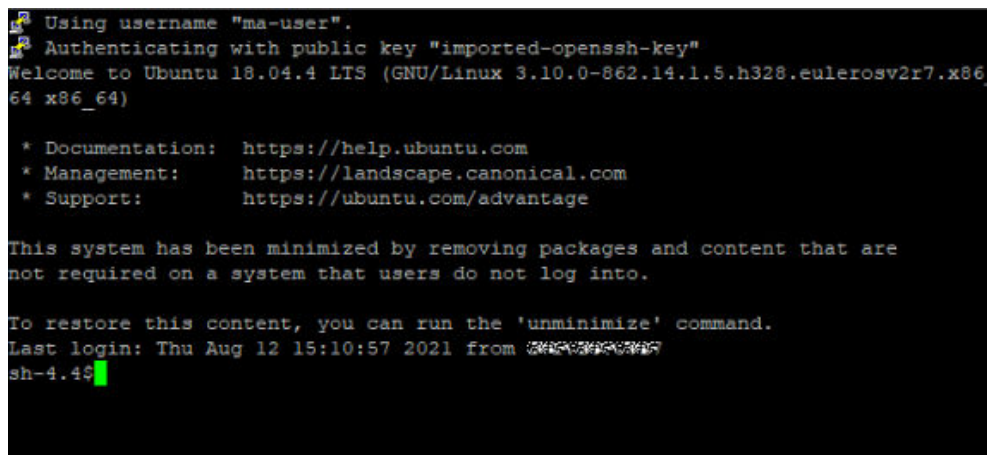
- Click **Open**. If you are logging in to the instance for the first time, PuTTY displays a security warning dialog box, asking if you want to accept the instance security certificate. Click **Accept** to save the certificate to your local registry.

Figure 3-128 Asking if you want to accept the instance security certificate



7. Connect to the notebook instance.

Figure 3-129 Connecting to a notebook instance

A terminal window showing an SSH connection. The prompt is 'ma-user@ubuntu:~\$'. The output shows the user is authenticated with a public key, the system is Ubuntu 18.04.4 LTS, and it provides links for documentation, management, and support. It also mentions that the system has been minimized and provides a command to restore content. The prompt changes to 'sh-4.4\$' at the end.

```
ma-user@ubuntu:~$ ssh -i /home/ma-user/.ssh/id_rsa.pub ma-user@192.168.1.100
Using username "ma-user".
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 3.10.0-862.14.1.el7.x86_64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Thu Aug 12 15:10:57 2021 from 192.168.1.100
sh-4.4$
```

3.7 Managing Notebook Instances

3.7.1 Searching for a Notebook Instance

Searching for an Instance

All created instances are displayed on the notebook page. To display a specific instance, search for it based on filter criteria.

- **Grant the permission to the IAM user for viewing all notebook instances.** Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**. On the displayed page, enable **View all**.
- Set search criteria, such as name, ID, status, image, flavor, description, and creation time.

Assigning the Required Permissions

Any IAM user granted with the **listAllNotebooks** and **listUsers** permissions can click **View all** on the notebook page to view the instances of all IAM users in the current IAM project. After the permission is granted, you can access OBS and SWR of IAM users in a notebook instance.

1. Log in to the ModelArts management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions > Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and create two policies.

Policy 1: Create a policy that allows users to view all notebook instances of an IAM project.

- **Policy Name:** Enter a custom policy name, for example, **Viewing all notebook instances**.

- **Policy View:** Select **Visual editor**.
- **Policy Content:** Select **Allow, ModelArts Service, modelarts:notebook:listAllNotebooks**, and default resources.

Policy 2: Create a policy that allows users to view all users of an IAM project.

- **Policy Name:** Enter a custom policy name, for example, **Viewing all users of the current IAM project**.
- **Policy View:** Select **Visual editor**.
- **Policy Content:** Select **Allow, Identity and Access Management, iam:users:listUsers**, and default resources.

3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in 2, and click **Next**. Then, select the scope and click **OK**.

After the configuration, all users in the user group have the permission to view all notebook instances created by users in the user group.

If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

Starting Notebook Instances of Other IAM Users

If an IAM user wants to access another IAM user's notebook instance through remote SSH, they need to update the SSH key pair to their own. Otherwise, error **ModelArts.6786** will be reported. For details about how to update a key pair, see [Modifying the SSH Configuration for a Notebook Instance](#). ModelArts.6789: Failed to find SSH key pair KeyPair-xxx on the ECS key pair page. Update the key pair and try again later.

3.7.2 Updating a Notebook Instance

Changing an Image

ModelArts allows you to change images on a notebook instance to flexibly adjust its AI engine. Images can be changed for only stopped notebook instances.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. In the notebook list, click **More** in the **Operation** column of the target notebook instance and select **Change Image**.
3. In the **Change Image** dialog box, select a new image and click **OK**. After the modification, you can view the new image on the notebook list page.

Changing the Specifications of a Notebook Instance

ModelArts allows you to change the node specifications for a notebook instance. Specifications of a notebook instance can be modified only when the notebook instance is in the **Stopped**, **Running**, or **Startup failed** state.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.

2. In the notebook instance list, locate the target notebook instance and choose **More > Modify Specifications** in the **Operation** column. In the displayed **Modify Specifications** dialog box, select the required specifications.

Modifying the SSH Configuration for a Notebook Instance

ModelArts allows you to modify the SSH configuration only when the notebook instance is stopped.

If remote SSH connection is not configured when you create the notebook instance and you want to enable it after the instance is created, go to the notebook instance details page, and enable SSH configuration. If a whitelist is configured for remote connection and you need to use another IP address, modify the IP addresses in the whitelist on the instance details page. If an IAM user needs to start a notebook instance of another user's, change the key pair.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. Click the target notebook instance. Enable remote SSH and change the key pair and whitelist.

NOTE

After the SSH configuration is updated, the remote SSH function cannot be disabled. If SSH remote development must be configured for the selected image, configure

Whitelist or click  next to **Authentication**.

- Click  and choose an existing key pair, or click **Create** to create one.
- After you change the IP addresses, the existing links are still valid. After the links are released, the new links only from the changed IP addresses can be set up.
- Ensure that public IP addresses are set. If your source device and the ModelArts are isolated from each other in network, obtain the public IP address of your source device using a mainstream search engine, for example, by entering "IP address lookup", but not by running **ipconfig** or **ifconfig/ip** locally.

3.7.3 Starting, Stopping, or Deleting a Notebook Instance

Starting or Stopping an Instance

Stop the notebook instances that are not needed. You can also restart a stopped instance.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. Start or stop the target notebook instance.
 - To start a notebook instance, click **Start** in the **Operation** column of the target notebook instance. Only stopped notebook instances can be started.

- To stop a notebook instance, click **Stop** in the **Operation** column of the target notebook instance. Only running notebook instances can be stopped.

CAUTION

After a notebook instance is stopped:

- Data in the **/home/ma-user/work** directory and directories dynamically mounted to **/data** is saved. Data in other directories will be deleted. For example, the external dependency packages installed in other directories in the development environment will be deleted. Save your development environment settings as an image. For details, see [Saving a Notebook Instance](#).
-

Deleting an Instance

Delete the notebook instances that are not needed.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. In the notebook list, locate the target notebook instance, and click **Delete** in the **Operation** column. In the displayed dialog box, confirm the information, enter **DELETE** in the text box, and click **OK**.

CAUTION

Deleted notebook instances cannot be recovered. After a notebook instance is deleted, the data stored in the mounted directory will be deleted.

3.7.4 Saving a Notebook Instance

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.

 NOTE

If the image fails to be saved, view the event on the notebook instance details page.

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. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**. On the displayed page, configure the container engine size as needed. For details, see [Resizing a Standard Dedicated Resource Pool](#).
- If the fault persists, contact technical support.

Prerequisites

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

Saving an Image

1. 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.
2. In the **Save Image** dialog box, configure parameters. Click **OK** to save the image.
Choose an organization from the **Organization** drop-down list. If no organization is available, click **Create** on the right to create one.
Users in an organization can share all images in the organization.
3. The image will be saved as a snapshot, and it will take about 5 minutes. During this period of time, do not perform any operations on the instance.

Figure 3-130 Saving as a snapshot



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.

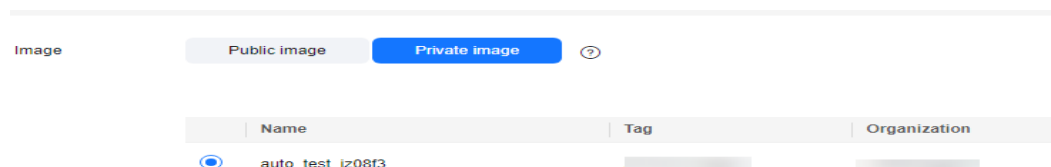
4. After the image is saved, the instance status changes to **Running**. View the image on the **Image Management** page.
5. Click the name of the image to view its details.

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.

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

Figure 3-131 Selecting a custom image to create a notebook instance



Method 2: On the **Image Management** 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.

3.7.5 Dynamically Expanding EVS Disk Capacity

Overview

If a notebook instance uses an EVS disk for storage, the disk is mounted to **/home/ma-user/work/** of the notebook container and the disk capacity can be expanded by up to 100 GB at a time when the instance is running.

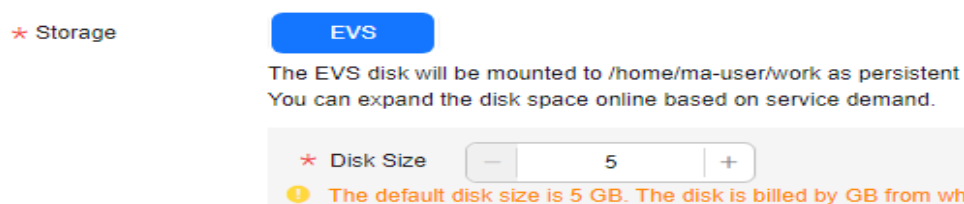
Application Scenarios

During notebook development, select a small EVS disk capacity, for example, 5 GB, when creating a notebook instance because the storage requirements are low at the initial stage. After the development, a large volume of data must be trained. Then, expand the disk capacity to cost-effectively meet your service needs.

Restrictions

- The target notebook instance must use EVS for storage.

Figure 3-132 Selecting EVS when creating a notebook instance



- Up to 100 GB can be expanded at a time. Additionally, the total capacity after expansion cannot exceed 4,096 GB.
- If the original capacity of an EVS disk is 4,096 GB, the disk capacity cannot be expanded.
- After the instance is stopped, the expanded capacity still takes effect.

Procedure

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.
2. Click the name of a running notebook instance. On the instance details page, click **Expansion**.
3. Set the capacity to be expanded and click **OK**. **Expanding** shows that the capacity expansion is in progress. After the expansion, the displayed storage capacity is the expanded capacity.

3.7.6 Dynamically Mounting an OBS Parallel File System

Overview

Parallel File System (Parallel File System) is an optimized high-performance file system provided by Object Storage Service (OBS).

Dynamic OBS mounting uses a mounting tool to convert the object storage protocol into the POSIX file protocol. OBS storage is simulated as a local file system and dynamically mounted to a running notebook container in ModelArts. After the mounting, you can perform application operations on the OBS objects in the notebook container.

Application Scenarios

Scenario 1: After you mount the OBS storage in which the target dataset is stored to your notebook instance, you can preview and perform operations in the dataset like operating a local file system.

Scenario 2: When training data in a notebook instance, you can use the dataset mounted to a notebook container.

Restrictions

OBS provides object buckets and PFS for storage.

ModelArts notebook supports only the mounting of an OBS parallel file system to **/data/** of a notebook container.

Procedure

Method 1: Through the ModelArts management console

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **DevEnviron > Notebook**.

2. Select a running notebook instance and click its name. On the notebook instance details page, click the **Storage** tab. From there, click **Mount Storage** and configure mounting parameters.
 - a. Set a local mounting directory. Enter a folder name in **/data/**, for example, **demo**. The system will automatically create the folder in **/data/** of the notebook container to mount the OBS file system.
 - b. Select the folder for storing the OBS parallel file system and click **OK**.
3. View the mounting result on the notebook instance details page.

3.8 ModelArts CLI Command Reference

3.8.1 (Optional) Installing ma-cli Locally

Application Scenarios

This document describes how to install ma-cli on Windows.

Step 1: Install ModelArts SDKs

Install ModelArts SDKs by referring to .

Step 2: Download ma-cli

1. .
2. Verify the software package signature.
 - a. .
 - b. Install OpenSSL and run the following command to verify the signature:

```
openssl cms -verify -binary -in D:\ma_cli-latest-py3-none-any.whl.cms -inform DER -content D:\ma_cli-latest-py3-none-any.whl -noverify > ./test
```

NOTE

In this example, the software package is stored in **D:**. Replace it with the actual path.

```
$openssl cms -verify -binary -in package.tar.gz.cms -signer "root" -inform DER -content package.tar.gz -noverify > ./test
CMS Verification successful
```

Step 3: Install ma-cli

1. Run **python --version** in the command prompt of your local environment to check whether Python has been installed. The Python version must be later than 3.7.x and earlier than 3.10.x. Version 3.7.x is recommended.

```
C:\Users\xxx>python --version
Python 3.7.4
```
2. Run **pip --version** to check whether the general package management tool pip is available.

```
C:\Users\xxx>pip --version
pip 20.0.2 from c:\users\xxx\appdata\local\programs\python\python37\lib\site-packages\pip (python 3.7.4)
```
3. Install ma-cli.
pip install {Path to the ma-cli software package}\ma_cli-latest-py3-none-any.whl

```
C:\Users\xxx>pip install C:\Users\xxx\Downloads\ma_cli-latest-py3-none-any.whl
.....
Successfully installed ma_cli.*.*
```

When ma-cli is installed, dependency packages are installed by default. If message "Successfully installed" is displayed, ma-cli has been installed.

NOTE

If an error message is displayed during the installation, indicating that a dependency package is missing, run the following command to install the dependency package as prompted:

pip install xxxx

xxxx is the name of the dependency package.

3.8.2 ma-cli Authentication

Overview

- VMs and personal computers require the configuration of authentication. Both a username and password (default) and an AK/SK can be used for authentication.
- When using an account for authentication, specify a username and password. When using an IAM account for authentication, specify an account, username, and password.
- In ModelArts notebook, you do not need to manually configure authentication because an agency is used for authentication by default.
- If you have configured authentication in ModelArts notebook, the specified authentication is preferentially used.

NOTE

To ensure the security of your sensitive information, it is important to prevent any potential leakage during authentication.

CLI Parameters

```
$ ma-cli configure -h
Usage: ma-cli configure [OPTIONS]

Options:
  -auth, --auth [PWD|AKSK|ROMA]  Authentication type.
  -rp, --region-profile PATH      ModelArts region file path.
  -a, --account TEXT              Account of an IAM user.
  -u, --username TEXT             Username of an IAM user.
  -p, --password TEXT             Password of an IAM user.
  -ak, --access-key TEXT          User access key.
  -sk, --secret-key TEXT          User secret key.
  -r, --region TEXT               The region you want to visit.
  -pi, --project-id TEXT          User project id.
  -C, --config-file TEXT          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

Table 3-11 Authentication CLI parameters

Parameter	Type	Man dator y	Description
-auth / --auth	String	No	Authentication mode, which can be PWD (username and password) or AKSK (AK/SK). The default value is PWD .
-rp / --region-profile	String	No	ModelArts region configuration file
-a / --account	String	No	IAM tenant account, which needs to be specified when authentication using an IAM account is used. It is required in authentication using a username and password.
-u / --username	String	No	Username, which is a username or an IAM username for authentication using an account or an IAM account. It is required in authentication using a username and password.
-p / --password	String	No	Password, which is required in authentication using a username and password
-ak / --access-key	String	No	Access key, which is required in authentication using an AK/SK
-sk / --secret-key	String	No	Secret key, which is required in authentication using an AK/SK
-r / --region	String	No	Region name. If this parameter is left blank, the value of the REGION_NAME environment variable will be used by default.
-pi / --project-id	String	No	Project ID. If this parameter is left blank, the region value (default) or the value of the PROJECT_ID environment variable will be used.
-P / --profile	String	No	Authentication configuration, which defaults to DEFAULT
-C / --config-file	String	No	Local path to the configuration file, which defaults to ~/.modelarts/ma-cli-profile.yaml

Authentication Using Username and Password

The following describes how to use the **ma-cli configure** command on a VM to configure authentication using the user name and password.

 NOTE

In the following example, any string with `${}` is a variable. You can specify a value.

For example, `${your_password}` indicates that you need to type your password.

```
# The DEFAULT authentication configuration is used by default. You need to type the account, username,
and password one by one. If the account and username are not required, press Enter to skip them.
$ ma-cli configure --auth PWD --region ${your_region}
account: ${your_account}
username: ${your_username}
password: ${your_password} # The input is not displayed on the console.
```

Authentication Using an AK/SK

This command uses an AK/SK for authentication, which means you have to enter them interactively. Your AK/SK will not be visible on the console.

 CAUTION

In the following example, any string with `${}` is a variable. You can specify a value.
For example, you need to replace `${access key}` with your access key.

```
ma-cli configure --auth AKSK
access key [***]: ${access key}
secret key [***]: ${secret key}
```

After the authentication command is executed, the authentication information will be saved in the `~/.modelarts/ma-cli-profile.yaml` configuration file.

3.9 Using Moxing Commands in a Notebook Instance

3.9.1 MoXing Framework Functions

MoXing Framework provides basic common components for MoXing. For example, it facilitates access to OBS. Importantly, MoXing Framework is decoupled from specific AI engines and can be seamlessly integrated with all major AI engines (including TensorFlow, MXNet, PyTorch, and MindSpore) supported by ModelArts. MoXing Framework allows you to interact with OBS components using the `mox.file` APIs described in this section.

 NOTE

MoXing primarily serves to streamline the process of reading and downloading data from OBS buckets. However, it is not suitable for OBS parallel file systems. You are advised to call the OBS Python SDK to develop production service code. For details, see [API Overview of OBS SDK for Python](#).

Why `mox.file`

Use Python to open a local file.

```
with open('/tmp/a.txt', 'r') as f:
    print(f.read())
```

An OBS directory starts with **obs://**, for example, **obs://bucket/XXX.txt**. You cannot directly use the **open** function to open an OBS file. The preceding code for opening a local file will report an error.

With OBS, you can access various tools like SDK, API, OBS console, and OBS Browser. ModelArts `mox.file` offers a set of APIs that mimic a local file system, enabling easy management of OBS files. For example, you can use the following code to open a file on OBS:

```
import moxing as mox
with mox.file.File('obs://bucket_name/a.txt', 'r') as f:
    print(f.read())
```

The following Python code lists a local path:

```
import os
os.listdir('/tmp/my_dir/')
```

To list an OBS path, add the following code in `mox.file`:

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/my_dir/')
```

Importing MoXing Framework

To use the MoXing Framework, first add the MoXing Framework module to the beginning of your code.

Import the MoXing Framework module:

```
import moxing as mox
```

Related Notes

After the MoXing module is introduced, the standard logging module of Python is set to the INFO level, and the version number is printed. You can use the following API to reset the **logging** level:

```
import logging

from moxing.framework.util import runtime
runtime.reset_logger(level=logging.WARNING)
```

Before introducing MoXing, you can set the **MOX_SILENT_MODE** environment variable to **1** to prevent MoXing from printing the version number. Before importing MoXing, set the environment variables using the following Python code.

```
import os
os.environ['MOX_SILENT_MODE'] = '1'
import moxing as mox
```

Data Downloading Acceleration

You can use MoXing Framework to accelerate data downloading for training jobs created using ModelArts preset images. This is suitable when the number of files ranges from 1 million to 10 million, a single large file, or the file size is greater than 20 GB.

1. Log in to the ModelArts console and choose **Model Training > Training Jobs** in the navigation pane on the left.

2. In the upper right corner of the page, click **Create Training Job**, and set **MA_MOXING_FWVER=2.2.8.0aa484aa** in **Environment Variable** to install the latest MoXing Framework. For details about other parameters, see [Creating a Training Job](#). Then, you can use **moxing.file.copy_parallel** in the training job script to accelerate data downloading.
3. Set **MOX_C_ACCELERATE=0** in **Environment Variable** to disable data downloading acceleration if it is not needed.

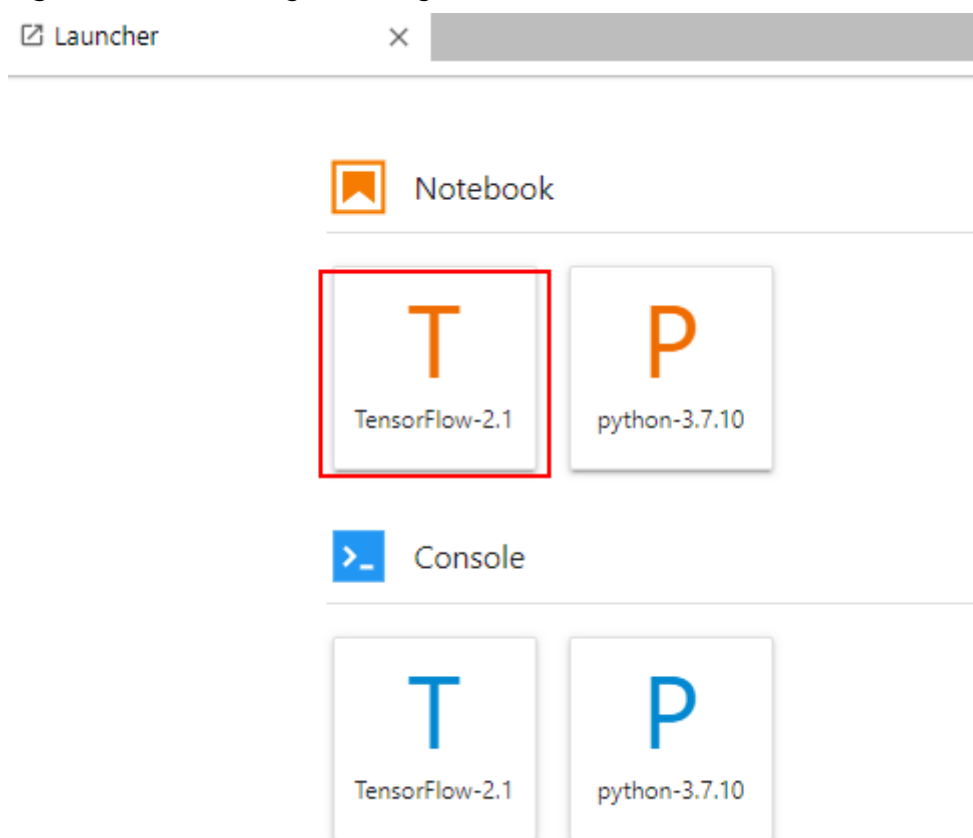
3.9.2 Using MoXing in Notebook

This document describes how to call MoXing Framework APIs in ModelArts.

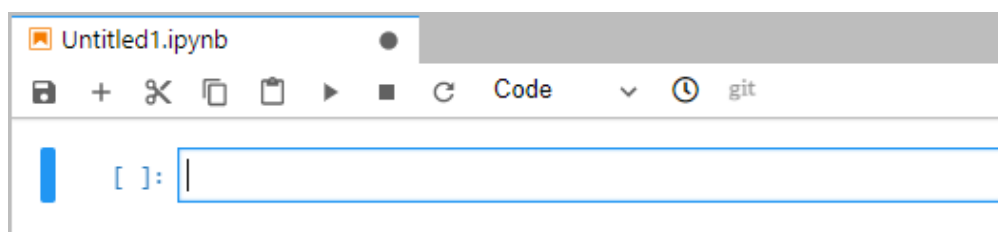
Logging In to ModelArts and Creating a Notebook Instance

1. Log in to the ModelArts console. In the navigation pane, choose **Development Workspace > Notebook** to access the **Notebook** page.
2. Click **Create**. On the **Create Notebook** page that is displayed, create a notebook instance by referring to .
3. After a notebook instance is created and enters the **Running** status, click **Open** in the **Operation** column to go to the **JupyterLab Notebook** page.
4. On the **Launcher** page of JupyterLab, for example, click **TensorFlow** to create a file for encoding.

Figure 3-133 Selecting an AI engine



After the file is created, the **JupyterLab** page is displayed by default.

Figure 3-134 Encoding page

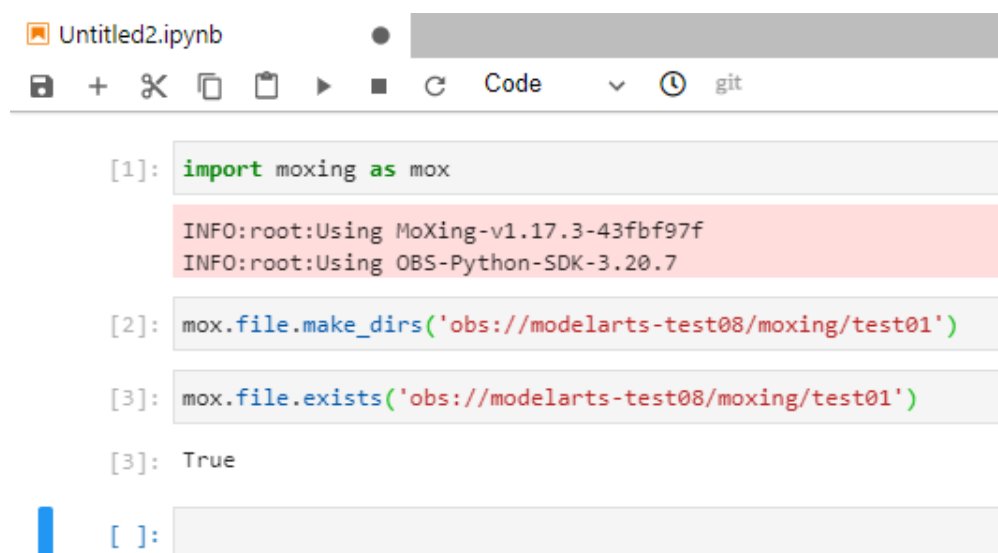
Calling `mox.file`.

Enter the following code to implement the following simple functions:

1. Introduce MoXing Framework.
2. Create the **test01** folder in the existing **modelarts-test08/moxing** directory.
3. Check whether the **test01** folder exists. If the folder exists, the preceding operation is successful.

```
import moxing as mox  
  
mox.file.make_dirs('obs://modelarts-test08/moxing/test01')  
mox.file.exists('obs://modelarts-test08/moxing/test01')
```

Figure 3-135 shows the result. Note that each time you enter a line of code, click **Run**. You can also go to OBS Console and check whether the **test01** folder has been created in the **modelarts-test08/moxing** directory. For more common MoXing operations, see [Sample Code for Common Operations](#).

Figure 3-135 Example

Copying Data to OBS

On the Notebook JupyterLab page, copy the **yolov8_train_ascend.zip** file to an OBS bucket. The sample code is as follows:

```
import os  
import zipfile
```

```
import moxing as mox
mox.file.copy('yolov8_train_ascend.zip','obs://pcb-data-me/pcb.zip')
```

3.9.3 Mapping Between mox.file and Local APIs and Switchover

API Mapping

- Python: local file operation APIs of Python. The APIs can be shifted to the corresponding MoXing file operation APIs (mox.file) by one click.
- mox.file: file operation APIs of MoXing Framework. The APIs correspond to the Python APIs.
- tf.gfile: TensorFlow APIs with the same functions as MoXing file operation APIs. In MoXing, file operation APIs cannot be automatically switched to TensorFlow APIs. The following table lists only the APIs with similar functions.

Table 3-12 API mapping

Python (Local File Operation API)	mox.file (MoXing File Operation API)	tf.gfile (TensorFlow File Operation API)
glob.glob	mox.file.glob	tf.gfile.Glob
os.listdir	mox.file.list_directory(..., recursive=False)	tf.gfile.ListDirectory
os.makedirs	mox.file.make_dirs	tf.gfile.MakeDirs
os.mkdir	mox.file.mk_dir	tf.gfile.MkDir
os.path.exists	mox.file.exists	tf.gfile.Exists
os.path.getsize	mox.file.get_size	-
os.path.isdir	mox.file.is_directory	tf.gfile.IsDirectory
os.remove	mox.file.remove(..., recursive=False)	tf.gfile.Remove
os.rename	mox.file.rename	tf.gfile.Rename
os.scandir	mox.file.scan_dir	-
os.stat	mox.file.stat	tf.gfile.Stat
os.walk	mox.file.walk	tf.gfile.Walk
open	mox.file.File	tf.gfile.FastGFile(tf.gfile.Gfile)
shutil.copyfile	mox.file.copy	tf.gfile.Copy
shutil.copytree	mox.file.copy_parallel	-
shutil.rmtree	mox.file.remove(..., recursive=True)	tf.gfile.DeleteRecursively

3.9.4 Sample Code for Common Operations

Data Reads and Writes

- Read an OBS file.

For example, if you read the **obs://bucket_name/obs_file.txt** file, the content is returned as strings.

```
import moxing as mox
file_str = mox.file.read('obs://bucket_name/obs_file.txt')
```

You can also open the file object and read data from it. Both methods are the same.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_str = f.read()
```

- Read a line from a file. A string that ends with a newline character is returned. You can also open the file object in OBS.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_line = f.readline()
```

- Read all lines from a file. A list is returned, in which each element is a line and ends with a newline character.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_line_list = f.readlines()
```

- Read an OBS file in binary mode.

For example, if you read the **obs://bucket_name/obs_file.bin** file, the content is returned as bytes.

```
import moxing as mox
file_bytes = mox.file.read('obs://bucket_name/obs_file.bin', binary=True)
```

You can also open the file object and read data from it. Both methods are the same.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.bin', 'rb') as f:
    file_bytes = f.read()
```

One or all lines in a file opened in binary mode can be read with the same method.

- Write a string to a file.

For example, write **Hello World!** into the **obs://bucket_name/obs_file.txt** file.

```
import moxing as mox
mox.file.write('obs://bucket_name/obs_file.txt', 'Hello World!')
```

You can also open the file object and write data into it. Both methods are the same.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'w') as f:
    f.write('Hello World!')
```

NOTE

When you open a file in write mode or call **mox.file.write**, if the file to be written does not exist, the file will be created. If the file to be written already exists, the file is overwritten.

- Append content to an OBS file.

For example, append **Hello World!** to the **obs://bucket_name/obs_file.txt** file.

```
import moxing as mox
mox.file.append('obs://bucket_name/obs_file.txt', 'Hello World!')
```

You can also open the file object and append content to it. Both methods are the same.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'a') as f:
    f.write('Hello World!')
```

When you open a file in append mode or call **mox.file.append**, if the file to be appended does not exist, the file will be created. If the file to be appended already exists, the content is directly appended.

If the size of the source file to be appended is large, for example, the **obs://bucket_name/obs_file.txt** file exceeds 5 MB, the append performance is low.

NOTE

If the file object is opened in write or append mode, when the **write** function is called, the content to be written is temporarily stored in the cache until the file object is closed (the file object is automatically closed when the **with** statement exits). Alternatively, you can call the **close()** or **flush()** function of the file object to write the file content.

List

- List an OBS directory. Only the top-level result (relative path) is returned. Recursive listing is not performed.

For example, if you list **obs://bucket_name/object_dir**, all files and folders in the directory are returned, but recursive queries are not performed.

Assume that **obs://bucket_name/object_dir** is in the following structure:

```
bucket_name
|- object_dir
   |- dir0
   |- file00
   |- file1
```

Call the following code:

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir')
```

The following list is returned:

```
['dir0', 'file1']
```

- Recursively list an OBS directory. All files and folders (relative paths) in the directory are returned, and recursive queries are performed.

Assume that **obs://bucket_name/object_dir** is in the following structure:

```
bucket_name
|- object_dir
   |- dir0
   |- file00
   |- file1
```

Call the following code:

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir', recursive=True)
```

The following list is returned:

```
['dir0', 'dir0/file00', 'file1']
```

Create a Folder

Create an OBS directory, that is, an OBS folder. Recursive creation is supported. That is, if the **sub_dir_0** folder does not exist, it is automatically created. If the **sub_dir_0** folder exists, no folder will be created.

```
import moxing as mox
mox.file.make_dirs('obs://bucket_name/sub_dir_0/sub_dir_1')
```

Query

- Check whether an OBS file exists. If the file exists, **True** is returned. If the file does not exist, **False** is returned.

```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/file.txt')
```

- Check whether an OBS folder exists. If the folder exists, **True** is returned. If the folder does not exist, **False** is returned.

```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/sub_dir_1')
```

NOTE

OBS allows files and folders with the same name exist (not allowed in UNIX). If a file or folder with the same name exists, for example, **obs://bucket_name/sub_dir_0/abc**, when **mox.file.exists** is called, **True** is returned regardless of whether **abc** is a file or folder.

- Check whether an OBS path is a folder. If it is a folder, **True** is returned. If it is not a folder, **False** is returned.

```
import moxing as mox
mox.file.is_directory('obs://bucket_name/sub_dir_0/sub_dir_1')
```

NOTE

OBS allows files and folders with the same name exist (not allowed in UNIX). If a file or folder with the same name exists, for example, **obs://bucket_name/sub_dir_0/abc**, when **mox.file.is_directory** is called, **True** is returned.

- Obtain the size of an OBS file, in bytes.

For example, obtain the size of **obs://bucket_name/obs_file.txt**.

```
import moxing as mox
mox.file.get_size('obs://bucket_name/obs_file.txt')
```

- Recursively obtain the size of all files in an OBS folder, in bytes.

For example, obtain the total size of all files in the **obs://bucket_name/object_dir** directory.

```
import moxing as mox
mox.file.get_size('obs://bucket_name/object_dir', recursive=True)
```

- Obtain the **stat** information about an OBS file or folder. The **stat** information contains the following:

- **length**: file size
- **mtime_nsec**: creation timestamp
- **is_directory**: whether the path is a folder

For example, if you want to query the OBS file **obs://bucket_name/obs_file.txt**, you can replace the file path with a folder path.

```
import moxing as mox
stat = mox.file.stat('obs://bucket_name/obs_file.txt')
print(stat.length)
print(stat.mtime_nsec)
print(stat.is_directory)
```


Delete

- Delete an OBS file.

For example, delete **obs://bucket_name/obs_file.txt**.

```
import moxing as mox
mox.file.remove('obs://bucket_name/obs_file.txt')
```

- Delete an OBS folder and recursively delete all content in the folder. If the folder does not exist, an error is reported.

For example, delete all content in **obs://bucket_name/sub_dir_0**.

```
import moxing as mox
mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
```

Move and Copy

- Move an OBS file or folder. The move operation is implemented by copying and deleting data.

- Move an OBS file to another OBS file. For example, move **obs://bucket_name/obs_file.txt** to **obs://bucket_name/obs_file_2.txt**.

```
import moxing as mox
mox.file.rename('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
```

NOTE

The move and copy operation must be performed in the same bucket.

- Move an OBS file to a local file. For example, move **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.

```
import moxing as mox
mox.file.rename('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
```

- Move a local file to an OBS file. For example, move **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

- Move a local file to another local file. For example, move **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**. This operation is equivalent to **os.rename**.

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
```

You can move folders in the same way. If you move a folder, all content in the folder is moved recursively.

- Copy a file. **mox.file.copy** can be used to perform operations only on files. To perform operations on folders, use **mox.file.copy_parallel**.

- Copy an OBS file to another OBS path. For example, copy **obs://bucket_name/obs_file.txt** to **obs://bucket_name/obs_file_2.txt**.

```
import moxing as mox
mox.file.copy('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
```

- Copy an OBS file to a local path, that is, download an OBS file. For example, download **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.

```
import moxing as mox
mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
```

- Copy a local file to OBS, that is, upload an OBS file. For example, upload **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.

```
import moxing as mox
mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

- Copy a local file to another local path. This operation is equivalent to **shutil.copyfile**. For example, copy **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**.

```
import moxing as mox
mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
```
- Copy a folder. **mox.file.copy_parallel** can be used to perform operations only on folders. To perform operations on files, use **mox.file.copy**.
 - Copy an OBS file to another OBS path. For example, copy **obs://bucket_name/sub_dir_0** to **obs://bucket_name/sub_dir_1**.

```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', 'obs://bucket_name/sub_dir_1')
```
 - Copy an OBS folder to a local path, that is, download an OBS folder. For example, download **obs://bucket_name/sub_dir_0** to **/tmp/sub_dir_0**.

```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```
 - Copy a local folder to OBS, that is, upload an OBS folder. For example, upload **/tmp/sub_dir_0** to **obs://bucket_name/sub_dir_0**.

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```
 - Copy a local folder to another local path. This operation is equivalent to **shutil.copytree**. For example, copy **/tmp/sub_dir_0** to **/tmp/sub_dir_1**.

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', '/tmp/sub_dir_1')
```

3.9.5 Sample Code for Advanced MoXing Usage

If you are familiar with common operations, the MoXing Framework API document, and common Python code, you can refer to this section to use advanced MoXing Framework functions.

Closing a File After File Reading Is Completed

When you read an OBS file, you are establishing an HTTP connection to access the network stream. Once done, close the file immediately. To prevent you from forgetting to close a file, you are advised to use the **with** statement. When the **with** statement exits, the **close()** function of the **mox.file.File** object is automatically called.

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    data = f.readlines()
```

Reading or Writing an OBS File Using pandas

- Use **pandas** to read an OBS file.

```
import pandas as pd
import moxing as mox
with mox.file.File("obs://bucket_name/b.txt", "r") as f:
    csv = pd.read_csv(f)
```
- Use **pandas** to write an OBS file.

```
import pandas as pd
import moxing as mox
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
with mox.file.File("obs://bucket_name/b.txt", "w") as f:
    df.to_csv(f)
```

Reading an Image Using a File Object

When OpenCV is used to open an image, the OBS path cannot be passed and the image must be read using a file object. The following code cannot read the image:

```
import cv2
cv2.imread('obs://bucket_name/xxx.jpg', cv2.IMREAD_COLOR)
```

Modify the code as follows:

```
import cv2
import numpy as np
import moxing as mox
img = cv2.imdecode(np.fromstring(mox.file.read('obs://bucket_name/xxx.jpg', binary=True), np.uint8),
cv2.IMREAD_COLOR)
```

Reconstructing an API That Does Not Support OBS Paths to One That Does

In **pandas**, **to_hdf** and **read_hdf** used to read and write H5 files do not support OBS paths, nor do they support file objects to be entered. The following code may cause errors:

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['a', 'b', 'c'])
df.to_hdf('obs://wolfros-net/hdftest.h5', key='df', mode='w')
pd.read_hdf('obs://wolfros-net/hdftest.h5')
```

The API compiled using the pandas source code is rewritten to support OBS paths.

- Write H5 to OBS = Write H5 to the local cache + Upload the local cache to OBS + Delete the local cache
- Read H5 from OBS = Download H5 to the local cache + Read the local cache + Delete the local cache

That is, write the following code at the beginning of the script to enable **to_hdf** and **read_hdf** to support OBS paths:

```
import os
import moxing as mox
import pandas as pd
from pandas.io import pytables
from pandas.core.generic import NDFrame

to_hdf_origin = getattr(NDFrame, 'to_hdf')
read_hdf_origin = getattr(pytables, 'read_hdf')

def to_hdf_override(self, path_or_buf, key, **kwargs):
    tmp_dir = '/cache/hdf_tmp'
    file_name = os.path.basename(path_or_buf)
    mox.file.make_dirs(tmp_dir)
    local_file = os.path.join(tmp_dir, file_name)
    to_hdf_origin(self, local_file, key, **kwargs)
    mox.file.copy(local_file, path_or_buf)
    mox.file.remove(local_file)

def read_hdf_override(path_or_buf, key=None, mode='r', **kwargs):
    tmp_dir = '/cache/hdf_tmp'
    file_name = os.path.basename(path_or_buf)
    mox.file.make_dirs(tmp_dir)
    local_file = os.path.join(tmp_dir, file_name)
    mox.file.copy(path_or_buf, local_file)
    result = read_hdf_origin(local_file, key, mode, **kwargs)
    mox.file.remove(local_file)
```

```
return result

setattr(NDFrame, 'to_hdf', to_hdf_override)
setattr(pytables, 'read_hdf', read_hdf_override)
setattr(pd, 'read_hdf', read_hdf_override)
```

Use MoXing to Enable h5py.File to Support OBS

```
import os
import h5py
import numpy as np
import moxing as mox

h5py_File_class = h5py.File

class OBSFile(h5py_File_class):
    def __init__(self, name, *args, **kwargs):
        self._tmp_name = None
        self._target_name = name
        if name.startswith('obs://'):
            self._tmp_name = name.replace('/', '_')
            if mox.file.exists(name):
                mox.file.copy(name, os.path.join('cache', 'h5py_tmp', self._tmp_name))
            name = self._tmp_name

        super(OBSFile, self).__init__(name, *args, **kwargs)

    def close(self):
        if self._tmp_name:
            mox.file.copy(self._tmp_name, self._target_name)

        super(OBSFile, self).close()

setattr(h5py, 'File', OBSFile)

arr = np.random.randn(1000)
with h5py.File('obs://bucket/random.hdf5', 'r') as f:
    f.create_dataset("default", data=arr)

with h5py.File('obs://bucket/random.hdf5', 'r') as f:
    print(f.require_dataset("default", dtype=np.float32, shape=(1000,)))
```

4 Model Training

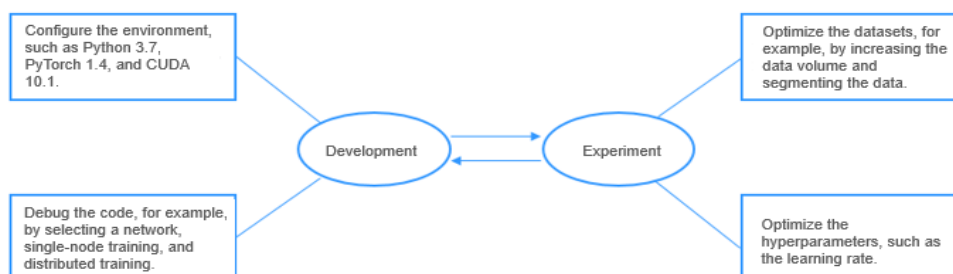
4.1 Model Training Process

AI modeling involves two stages:

- Development: Prepare and configure the environment, and debug code for training based on deep learning. ModelArts DevEnviron is recommended for code debugging.
- Experiment: Optimize the datasets and hyperparameters, and obtain an ideal model through multiple rounds of experiments. The ModelArts training platform is recommended for training.

In the two stages, code is designed, developed and tested in repeated cycles. In the development stage, when the code becomes stable, the modeling process enters the experiment stage, during which hyperparameters are continuously optimized to iterate the model. In the experiment stage, when the training performance can be optimized, the modeling process returns to the development stage for optimizing code.

Figure 4-1 Model development process



ModelArts provides model training, which allows you to view training results and tune model parameters based on the training results. You can select resource pools with different instance flavors for model training.

To train a model on ModelArts Standard, follow these steps:

Figure 4-2 ModelArts Standard model training process

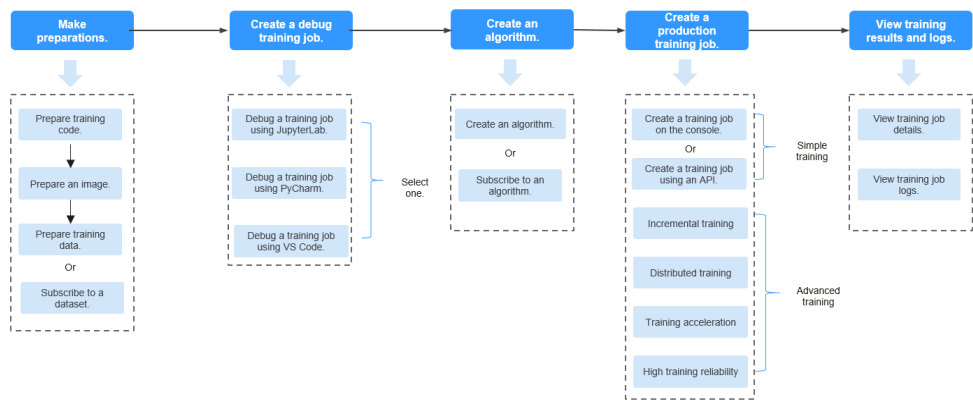


Table 4-1 ModelArts Standard model training process

Task	Subtask	Description
Making preparations	Preparing training code	Model training includes training code, training framework, and training data. Training code contains the boot file or command and dependency package of a training job. - To use a preset image to create a training job, develop training code by referring to Developing Code for Training Using a Preset Image. - To use a custom image to create a training job, develop training code by referring to Developing Code for Training Using a Custom Image.
	Preparing a training image	There are multiple training image sources. For details, see Preparing a Model Training Image . - ModelArts Standard offers mainstream preset images for model training, ready for immediate use. - If the preset images do not meet your needs, create custom images.
	Preparing training data	Before training, prepare necessary data, which can be datasets or predictive models. - Upload your training data to OBS if it does not need further processing. To create a training job, enter the OBS bucket path directly as the input parameter path. - Import your unlabeled or unprocessed training dataset to ModelArts data management for processing. To create a training job, choose your dataset in data management as the input parameter.

Task	Subtask	Description
Creating a debug training job	Creating a debug training job	Before model training, debug your code. ModelArts offers multiple methods for creating a debug training job. • With ModelArts, you can easily access JupyterLab in the cloud without worrying about environment installation or configuration. • After enabling remote SSH, you can remotely access a training job for debugging from a local IDE. This method does not affect your coding habits. Debugged code can be used for production training at zero cost. ModelArts supports local IDE PyCharm. For details, see Using PyCharm Toolkit to Create and Debug a Training Job.
Creating an algorithm	Creating an algorithm	Before creating a production training job, create an algorithm.
Creating a production training job	Using basic training features	• You can create a training job on the ModelArts Standard console. There are multiple algorithm types and training frameworks for you to select. For details, see Table 4-2. • ModelArts Standard also allows you to create training jobs using APIs. For details, see Using PyTorch to Create a Training Job (New-Version Training).
	Using advanced training features	ModelArts Standard supports the following advanced training features: • Incremental learning • Distributed training • Training acceleration • High training reliability
Viewing training results and logs	Viewing training job details	You can view a training job's parameter settings and events on the job details page at any time, whether the job is running or has completed.
	Viewing training job logs	Training logs track the execution and any errors that occur during training job runs. You can view these logs to identify and troubleshoot issues that cause jobs to fail.

Table 4-2 Methods of creating a training job

Creation Method	Description
Using a preset image to create a training job	If you have used some mainstream images to develop algorithms locally, you can select a mainstream image and create a training job to build a model.
Using a custom image to create a training job	To use a non-mainstream image, create a custom algorithm image and then use it to create a training job.
Using an existing algorithm to create a training job	In Algorithm Management, you can manage your created algorithms. This allows you to quickly create training jobs and build models using these algorithms.

4.2 Preparing Model Training Code

4.2.1 Starting a Preset Image's Boot File

ModelArts Standard offers multiple AI images for model training, which can be adapted by modifying their boot commands.

This section describes how to modify the boot file when creating a training job using different preset images.

Ascend-Powered-Engine Boot Principles

Ascend-Powered-Engine is a unique engine that combines multiple AI frameworks, runtime environments, and boot modes tailored to Ascend accelerator chips.

Major Snt9 Ascend accelerators run on Arm-backed CPU servers, which means the upper-layer Docker images are Arm images. The NVIDIA CUDA (unified computing architecture) compute library is installed in images for GPU scenarios. The CANN (heterogeneous computing architecture) compute library is installed in images powered by the Ascend-Powered-Engine, adapting to the Ascend driver.

After a training job is submitted, ModelArts Standard automatically runs the boot file.

The default boot mode of the boot file of the Ascend-Powered-Engine framework is as follows:

The number of times the boot file runs for each training job depends on the number of cards used. When a job is running, the boot file is executed once for each card. For example, in a single-node job with one card, the boot file runs once. In a single-node job with eight cards, it runs eight times. So, do not listen on ports in the boot file.

The following environment variables are automatically configured in the boot file:

- **RANK_TABLE_FILE**: path of the rank table file.
- **ASCEND_DEVICE_ID**: logical device ID. For example, for single-card training, the value is always **0**.
- **RANK_ID**: logical (sequential) number of a device in a training job.
- **RANK_SIZE**: Set this parameter based on the number of devices in the rank table file. For example, the value is **4** for 4 snt9b devices.

To ensure the boot file runs only once, check the **ASCEND_DEVICE_ID** value. If it is **0**, execute the logic; otherwise, exit directly.

For details about the example code file **mindspore-verification.py** of the Ascend-Powered-Engine framework, see [Training the mindspore-verification.py File](#).

The command for starting Ascend-Powered-Engine in standalone mode is the same as that in distributed mode.

The Ascend-Powered-Engine framework offers multiple boot modes. By default, the boot file is executed based on **RANK_TABLE_FILE**. You can also configure the **MA_RUN_METHOD** environment variable to run the boot file using alternative methods. The **MA_RUN_METHOD** environment variable can be set to **torchrun** and **msrun**.

- If **MA_RUN_METHOD** is set **torchrun**, ModelArts Standard uses the **torchrun** command to run the boot file.

NOTE

The PyTorch version must be 1.11.0 or later.

- For single-node jobs, ModelArts Standard uses these commands to start the boot file:

```
torchrun --standalone --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} $  
{MA_EXTRA_TORCHRUN_PARAMS} "Boot file" {arg1} {arg2} ...
```

- For multi-node jobs, ModelArts Standard uses these commands to start the boot file:

```
torchrun --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} --node_rank=$  
{VC_TASK_INDEX} --master_addr={master_addr} --master_port=$  
{MA_TORCHRUN_MASTER_PORT} --rdzv_id={ma_job_name} --rdzv_backend=static $  
{MA_EXTRA_TORCHRUN_PARAMS} "Boot file" {arg1} {arg2} ...
```

Parameters:

- **standalone**: identifier of a single-node job.
- **nnodes**: number of task nodes.
- **nproc_per_node**: number of main processes started by each task node. Set this parameter to the number of NPUs allocated to the task.
- **node_rank**: task rank, which is used for multi-task distributed training.
- **master_addr**: address of the main task (rank 0). Set it to the communication domain name of **worker-0**.
- **master_port**: port used for communication during distributed training on the main task (rank 0). The default value is **18888**. When a **master_port** conflict occurs, you can modify the port configuration by configuring the **MA_TORCHRUN_MASTER_PORT** environment variable.
- **rdzv_id**: Rendezvous ID. Set it to a value with the training job ID.
- **rdzv_backend**: Rendezvous backend, which is fixed at **static**. That is, **master_addr** and **master_port** are used instead of Rendezvous.

Additionally, you can configure the **MA_EXTRA_TORCHRUN_PARAMS** environment variable to add additional **torchrun** command parameters or overwrite the preset **torchrun** command parameters. The following is an example of configuring the **rdzv_conf** parameter in the **torchrun** command:

```
"environments": {  
  "MA_RUN_METHOD": "torchrun",  
  "MA_EXTRA_TORCHRUN_PARAMS": "--rdzv_conf=timeout=7200"  
}
```

NOTE

If the **RuntimeError: Socket Timeout** error occurs during the distributed process group initialization using **torchrun**, you can add the following environment variables to create a training job again to view initialization details and further locate the fault.

- LOGLEVEL=INFO
- TORCH_CPP_LOG_LEVEL=INFO
- TORCH_DISTRIBUTED_DEBUG=DETAIL

The **RuntimeError: Socket Timeout** error is caused by a significant time discrepancy between tasks when running the **torchrun** command. The time discrepancy is usually caused by initialization tasks, like downloading the training data and checkpoint read/write, which happen before the **torchrun** command is run. If the time taken to complete these initialization tasks varies significantly, a Socket Timeout error may occur. When this error happens, check the time difference between the **torchrun** execution points for each task. If the time difference is too large, optimize the initialization process before running the **torchrun** command to ensure a reasonable time gap.

- If **MA_RUN_METHOD** is set **msrun**, ModelArts Standard uses the **msrun** command to run the boot file.

NOTE

The MindSpore version must be 2.3.0 or later.

This solution supports dynamic networking and rank table file-based networking. If you set the environment variable **MS_RANKTABLE_ENABLE** to **True**, **msrun** reads the rank table file for networking. Otherwise, dynamic networking is used by default.

msrun uses these commands to start the boot file:

```
msrun --worker_num=${msrun_worker_num} --local_worker_num=${MA_NUM_GPUS} --master_addr=${msrun_master_addr} --node_rank=${VC_TASK_INDEX} --master_port=${msrun_master_port} --log_dir=${msrun_log_dir} --join=True --cluster_time_out=${MSRUN_CLUSTER_TIME_OUT} --rank_table_file=${msrun_rank_table_file} "Boot file" {arg1} {arg2} ...
```

Parameters:

- **worker_num**: total number of processes, which is also equivalent to the number of cards involved, as each card initiates a process.
- **local_worker_num**: number of processes on the current node, which is also the number of cards used by the current node.
- **master_addr**: IP address of the node where the **msrun** scheduling process is located. This parameter does not need to be configured for single-node jobs.
- **master_port**: port number of the **msrun** scheduling process.
- **node_rank**: ID of the current node.
- **log_dir**: log output directory of **msrun** and all processes.

- **join**: whether the **msrun** process still exists after the training process is started. The default value is **True**, indicating that the **msrun** process exits after all processes exit.
- **cluster_time_out**: timeout interval of the cluster networking. The default value is **600s**. The value can be controlled by the **MSRUN_CLUSTER_TIME_OUT** environment variable.
- **rank_table_file**: address of the rank table file. If the environment variable **MS_RANKTABLE_ENABLE** is set to **True**, this parameter is added during startup.

PyTorch-GPU Boot Principles

For single-node multi-card scenarios, the platform adds the **--init_method "tcp://<ip>:<port>"** parameter to the boot file.

For multi-node multi-card scenarios, the platform adds the **--init_method "tcp://<ip>:<port>" --rank <rank_id> --world_size <node_num>** parameter to the boot file.

The preceding parameters must be parsed in the boot file.

TensorFlow-GPU Boot Principles

For a single-node job, ModelArts starts a training container that exclusively uses the resources on the node.

For a multi-node job, ModelArts starts a parameter server and a worker on the same node. It allocates parameter server and worker tasks in a 1:1 ratio. For example, in a two-node job, two parameter servers and two workers are allocated. ModelArts also injects the following parameters into the boot file:

```
--task_index <VC_TASK_INDEX> --ps_hosts <TF_PS_HOSTS> --worker_hosts <TF_WORKER_HOSTS> --  
job_name <MA_TASK_NAME>
```

The following parameters must be parsed in the boot file.

- **VC_TASK_INDEX**: task serial number, for example, **0**, **1**, or **2**.
- **TF_PS_HOSTS**: addresses of parameter server nodes, for example, [**xx-ps-0.xx.TCP_PORT,xx-ps-1.xx.TCP_PORT**]. The value of **TCP_PORT** is a random port ranging from 5,000 to 10,000.
- **TF_WORKER_HOSTS**: addresses of worker nodes, for example, [**xx-worker-0.xx.TCP_PORT,xx-worker-1.xx.TCP_PORT**]. The value of **TCP_PORT** is a random port ranging from 5,000 to 10,000.
- **MA_TASK_NAME**: task name, which can be **ps** or **worker**.

Horovod/MPI/MindSpore-GPU

ModelArts uses **mpirun** to run boot files for Horovod, MPI, or MindSpore-GPU. To use a preset engine in ModelArts Standard, simply edit the boot file (training script). ModelArts Standard automatically builds the **mpirun** command and training job cluster. The platform does not add extra parameters to the boot file.

Example of **pytorch_synthetic_benchmark.py**:

```
import argparse  
import torch.backends.cudnn as cudnn
```

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

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

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

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

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

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

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

hvd.init()

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

cudnn.benchmark = True

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

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

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

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

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

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

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

```
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

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

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

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

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

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

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

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

run_mpi.sh is as follows:

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

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

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

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

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

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

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

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

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

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

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

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

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

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

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

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

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

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

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

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

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

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

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

```
"$@"

RET_CODE=$?

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

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

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

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

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

exit $RET_CODE
```

4.2.2 Developing Code for Training Using a Preset Image

Before you use a preset image in ModelArts Standard to create an algorithm, develop the algorithm code. This section describes how to modify local code for model training on ModelArts.

When creating an algorithm, set the code directory, boot file, input path, and output path. These settings enable the interaction between your codes and ModelArts Standard.

- Code directory

Specify the code directory in the OBS bucket and upload training data such as training code, dependency installation packages, or pre-generated model to the directory. After you create the training job, ModelArts downloads the code directory and its subdirectories to the container.

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.

Do not store training data in the code directory. When the training job starts, the data stored in the code directory will be downloaded to the backend. A large amount of training data may lead to a download failure. It is recommended that the size of the code directory does not exceed 50 MB.

- Boot file

The boot file in the code directory is used to start the training. Only Python boot files are supported. For details about the boot process of the boot file of a preset image, see [Starting a Preset Image's Boot File](#).

- Input path

The training data must be uploaded to an OBS bucket. In the training code, [the input path](#) must be parsed. ModelArts automatically downloads the data in the input path to the local container directory for training. Ensure that you have the read permission to the OBS bucket. After the training job is started, ModelArts mounts a disk to the `/cache` directory. You can use this directory to store temporary files. For details about the size of the `/cache` directory, see [What Are Sizes of the /cache Directories for Different Resource Specifications in the Training Environment?](#)

- Output path

You are advised to set an empty directory as the training output path. In the training code, [the output path](#) must be parsed. ModelArts automatically uploads the training output to the output path. Ensure that you have the write and read permissions to the OBS bucket.

The following section describes how to develop training code in ModelArts.

(Optional) Introducing Dependencies

1. If your model references other dependencies, place the required file or installation package in **Code Directory** you set during algorithm creation.

Parsing Input and Output Paths

When a ModelArts Standard training job reads data stored in OBS or outputs training results to a specified OBS path, perform the following operations to configure the input and output data:

1. Parse the input and output paths in the training code. The following method is recommended:

```
import argparse
# Create a parsing task.
parser = argparse.ArgumentParser(description='train mnist')

# Add parameters.
parser.add_argument('--data_url', type=str, default="./Data/mnist.npz", help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='path where the model is saved')

# Parse the parameters.
args = parser.parse_args()
```

After the parameters are parsed, use **data_url** and **train_url** to replace the paths to the data source and the data output, respectively.

2. When creating a training job, set the input and output paths.
Select the OBS path or dataset path as the training input, and the OBS path as the output.

Complete Training Code Example

The training code is closely related to the AI engine you use. The following uses the TensorFlow framework as an example. Before using this case, you need to

download the **mnist.npz** file and upload it to the OBS bucket. The training input is the OBS path where the **mnist.npz** file is stored.

The following training code example contains the code for saving the model.

```
import os
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='train mnist')
parser.add_argument('--data_url', type=str, default="/Data/mnist.npz", help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="/Model", help='path where the model 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)

model.save(os.path.join(args.train_url, 'model'))
```

4.2.3 Developing Code for Training Using a Custom Image

If the preset images offered by ModelArts Standard do not meet your needs, create custom images for model training.

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 Standard only after they are uploaded to the Software Repository for Container (SWR).

Boot Command Specifications for Custom Images

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

NOTE

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

conda env starts training jobs created using custom images. Training jobs do not run in a shell. Therefore, you are not allowed to run the **conda activate** command to activate a specified Conda environment. In this case, use other methods to start training. For example, Conda in your custom image is installed in the **/home/ma-user/anaconda3** directory, the Conda environment is **python-3.7.10**, and the training script is stored in **/home/ma-user/modelarts/user-job-dir/code/train.py**. Use a specified Conda environment to start training in one of the following ways:

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

```
ANACONDA_DIR=/home/ma-user/anaconda3  
DEFAULT_CONDA_ENV_NAME=python-3.7.10
```

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

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

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

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

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

- Method 3: Configure the **PATH** environment variable.

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

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

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

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

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

NOTE

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

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

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

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

Training Code Adaptation Specifications for Training Using an Ascend-powered Custom Image

When creating a training job that uses NPU resources, the system automatically generates the **Ascend HCCL RANK_TABLE_FILE** file in the training container. When using a preset image, **Ascend HCCL RANK_TABLE_FILE** is automatically parsed during training. When using a custom image, the training code must be modified to read and parse **Ascend HCCL RANK_TABLE_FILE**.

Ascend HCCL RANK_TABLE_FILE file description

Ascend HCCL RANK_TABLE_FILE provides the cluster used by Ascend distributed training jobs. It is used for distributed communication between Ascend chips and can be parsed by Huawei Collective Communication Library (HCCL). The file has two format versions: template 1 and template 2.

- ModelArts provides the template 2 format. The **Ascend HCCL RANK_TABLE_FILE** file in the ModelArts training environment is named **jobstart_hccl.json**. You can access this file using the preset **RANK_TABLE_FILE** environment variable.

Table 4-3 RANK_TABLE_FILE environment variables

Environment Variable	Description
RANK_TABLE_FILE	Directory of Ascend HCCL RANK_TABLE_FILE , which is /user/config . Obtain the file using \${RANK_TABLE_FILE}/jobstart_hccl.json .

Example of the **jobstart_hccl.json** file content in the ModelArts training environment (template 2):

```
{
  "group_count": "1",
  "group_list": [{
    "device_count": "1",
    "group_name": "job-trainjob",
    "instance_count": "1",
    "instance_list": [{
      "devices": [{
        "device_id": "4",
        "device_ip": "192.1.10.254"
      }],
      "pod_name": "jobxxxxxxx-job-trainjob-0",
      "server_id": "192.168.0.25"
    }],
    "status": "completed"
  }
}
```

In **jobstart_hccl.json**, the **status** value may not be **completed** when the training script is started. In this case, wait until the **status** value changes to **completed** and read the remaining content of the file.

- After the **status** field is **completed**, use the training script to convert the **jobstart_hccl.json** file from template 2 to template 1 format.

Format of the **jobstart_hccl.json** file after format conversion (template 1):

```
{
  "server_count": "1",
  "server_list": [{
    "device": [{
      "device_id": "4",
      "device_ip": "192.1.10.254",
      "rank_id": "0"
    }],
    "server_id": "192.168.0.25"
  }],
  "status": "completed",
  "version": "1.0"
}
```

Mount Points of a Training Job in a Container

When training a model with a custom image, the mount points in the container are shown in [Table 4-4](#).

Table 4-4 Training job mount points

Mount Point	Read Only	Remarks
/xxx	No	Directory where a dedicated resource pool mounts an SFS disk. You can specify this directory.
/home/ma-user/modelarts	No	This folder is empty. You should use it as the main directory.
/cache	No	Used to mount the hard disk of the host NVMe (supported by bare metal specifications).
/dev/shm	No	Used for PyTorch engine acceleration
/usr/local/nvidia	Yes	NVIDIA library of the host machine.

4.2.4 Configuring Password-free SSH Mutual Trust Between Nodes for a Training Job Created Using a Custom Image

If you use a custom image based on the MPI or Horovod framework for distributed training, you must configure password-free SSH mutual trust between training job nodes. Otherwise, the training will fail.

This involves code adaptation and training job parameter configuration.

1. Create a custom image with OpenSSH pre-installed. The training framework should be MPI or Horovod.
2. Create a boot script file **start_sshd.sh**.

```
MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}
mkdir -p /home/ma-user/etc
ssh-keygen -f /home/ma-user/etc/ssh_host_rsa_key0 -N "" -t rsa > /dev/null
/usr/sbin/sshd -p $MY_SSHD_PORT -h /home/ma-user/etc/ssh_host_rsa_key0
```
3. Upload the sshd startup script file to the training code directory in OBS.
4. Create a training job using the custom image.
 - **Code Directory:** Select the OBS path where the sshd boot script file is stored.
 - **Boot Command:** Adapt the boot command to the sshd boot script.

```
bash ${MA_JOB_DIR}/demo-code/start_sshd.sh && your custom command
```

In the command, **your custom command** indicates custom commands you want to execute in the training job.
 - **Environment Variable:** Add **MY_SSHD_PORT = 38888**.
 - **training_ssh_configure_nodes:** Enable it and configure the SSH key directory. Retain default settings unless you have specific needs. After a training job is delivered, the SSH key file and configuration file **authorized_keys config id_rsa id_rsa.pub** are automatically generated in the **/home/ma-user/.ssh** directory of the training container.
5. After a training job is created, its nodes can establish an SSH connection with each other by using the domain name and port number throughout the training process. The sample code is as follows:

```
ssh modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1 -p $MY_SSHD_PORT
```

4.3 Preparing a Model 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.

Preset Training Images

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

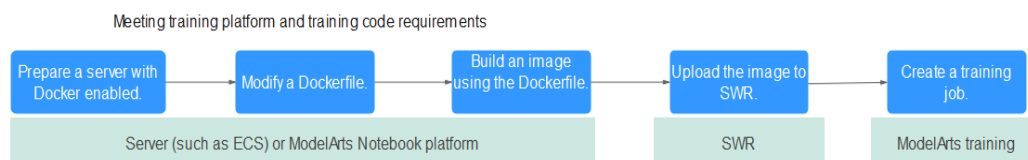
Table 4-5 ModelArts training base images

Engine	Version
PyTorch	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
Horovod	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

Creating a Custom Training Image

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. [Figure 4-3](#) shows the process of creating an image.

Figure 4-3 Process of 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 Image Using a Preset Image for Model Training](#).

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 for Model Training](#).

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

4.4 Creating a Debug Training Job

4.4.1 Using PyCharm Toolkit to Create and Debug a Training Job

AI developers use PyCharm to develop algorithms or models. ModelArts provides the PyCharm Toolkit plug-in to help AI developers quickly submit locally developed code to the ModelArts training environment. With PyCharm Toolkit, developers can quickly remotely access notebook instances through SSH, upload code, submit training jobs, and obtain training logs for local display so that they can better focus on local code development.

This section describes how to use PyCharm Toolkit to create and debug a training job.

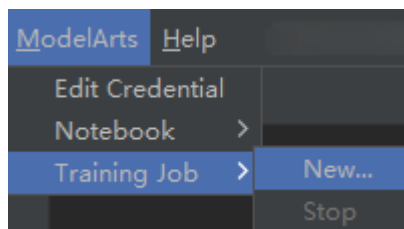
Prerequisites

- [Step 1 Download and Install PyCharm Toolkit](#)
- A training code project exists in the local PyCharm.
- You have created a bucket and folders in OBS for storing datasets and trained models. For example, create a bucket named **test-modelarts2** and folders **dataset-mnist** and **mnist-output**. Data used by the training job has been uploaded to OBS. OBS and ModelArts are in the same region.

Configuring Training Job Parameters

1. In PyCharm, open the training code project and training boot file, and choose **ModelArts > Training Job > New...** on the menu bar.

Figure 4-4 Configuring job parameters



2. In the displayed dialog box, configure the training job parameters. For details, see [Table 4-6](#).

Table 4-6 Training job parameters

Parameter	Description
Job Name	Name of the training job. The system automatically generates a name. You can rename it based on the following naming rules: • The name contains 1 to 64 characters. • Letters, digits, hyphens (-), and underscores (_) are allowed.
Job Description	Brief description of the training job.
Algorithm Source	Source of the training algorithm. The options are Frequently-used and Custom . Frequently-used refers to the frequently-used AI engines supported by ModelArts Training Management. If the AI engine you use is not in the supported list, you are advised to create a training job using a custom image.
AI Engine	Select the AI engine and the version used in code.
Boot File Path	Training boot file. The selected boot file must be a file in the current PyCharm training project. This parameter is displayed if Algorithm source is set to Frequently-used .
Code Directory	Training code directory. The system automatically sets this parameter to the directory where the training boot file is located. You can change the parameter value to a directory that is in the current project and contains the boot file. If the algorithm source is a custom image and the training code has been built in the image, this parameter can be left blank.
Image Path(optional)	URL of the SWR image
Boot Command	Command for starting the training job, for example, bash /home/work/run_train.sh python {Python boot file and parameters} . This parameter is displayed if Algorithm source is set to Custom . If the command does not contain the --data_url or --train_url parameter, the tool automatically adds the two parameters to the end of the command when submitting the training job. The two parameters correspond to the OBS path for storing training data and the OBS path for storing training output, respectively.
Data OBS Path	OBS path for storing training data, for example, /test-modelarts2/mnist/dataset-mnist/ , in which test-modelarts2 indicates a bucket name.

Parameter	Description
Training OBS Path	OBS path. A directory is automatically created in the path for storing a trained model and training logs.
Running Parameters	Running parameters. Add running parameters to your code based on your needs. Separate multiple running parameters with semicolons (;), for example, key1=value1;key2=value2 . This parameter can be left blank.
Specifications	Type of resources used for training. Currently, public resource pools and dedicated resource pools are supported. Dedicated resource pool specifications are identified by Dedicated Resource Pool .
Compute Nodes	Number of compute nodes. If this parameter is set to 1 , the system runs in standalone mode. If this parameter is set to a value greater than 1, the distributed computing mode is used at the background.
Available/Total Nodes	When Specifications is set to a dedicated resource pool, the number of available instances and the total number of instances are displayed. The value of Compute Nodes cannot exceed the number of available instances.

Figure 4-5 Configuring training job parameter (public resource pool)

Edit Training Job Configurations

JobName: MA-new-models-07-04-15-06-7

Job Description:

Frequently-used Custom

AI Engine: TensorFlow TF-1.13.1-python3.6

Algorithm Source: Boot File Path: D:\models\official\cv\deeplabv3\train.py Code Directory: D:\models\official Image Path(optional): For example: swr-info/image-org/image-name:version

Data OBS Path: obs://

Training OBS Path: obs://

Running Parameters:

Specifications: CPU:2*vCPUs 16GB

Compute Nodes: 1

Apply and Run Cancel Apply

Figure 4-6 Configuring training job parameters (dedicated resource pool)

Edit Training Job Configurations

JobName: MA-new-models-07-04-15-12-606

Job Description:

Frequently-used Custom

AI Engine: TensorFlow TF-1.13.1-python3.6

Algorithm Source: Boot File Path: D:\...\train.py Code Directory: D:\...\models\official

Image Path(optional):

Data OBS Path: obs://

Training OBS Path: obs://

Running Parameters:

Specifications: pool-maostest-notebook: CPU:8vCPUs 32Gi

Compute Nodes: 1

Apply and Run Cancel Apply

Figure 4-7 Configuring training job parameters (custom image)

Edit Training Job Configurations

JobName: MA-new-models-07-04-15-14-100

Job Description:

Frequently-used Custom

Image Path: swr.

Algorithm Source: Boot Command: python run.py Code Directory:

Data OBS Path: obs://

Training OBS Path: obs://

Running Parameters:

Specifications: CPU:2*vCPUs 16GB

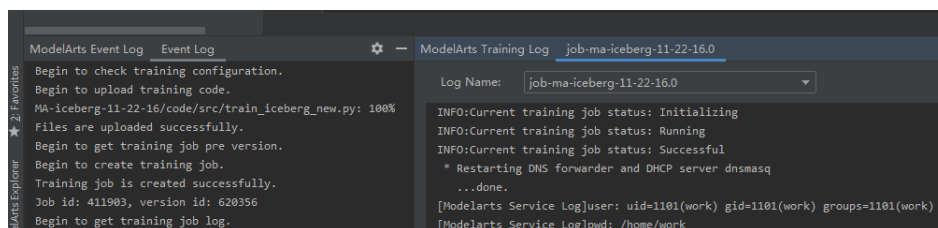
Compute Nodes: 1

Apply and Run Cancel Apply

- Click **Apply and Run**. Then, local code is automatically uploaded to the cloud and training is started. The training job status is displayed in the **Training Log** area in real time. If information similar to **Current training job status: Successful** is displayed in the training logs, the training job has been successfully executed.

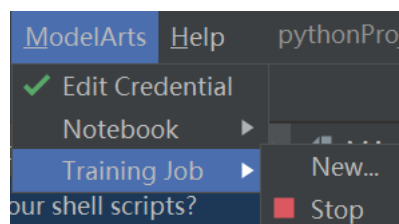
 NOTE

- After you click **Apply and Run**, the system automatically executes the training job. To stop the training job, choose **ModelArts > Training Job > Stop** on the menu bar.
- If you click **Apply**, the job is not started directly, and the training job settings are saved instead. To start the job, click **Apply and Run**.

Figure 4-8 Training log example

Stopping a Job

When a training job is running, choose **ModelArts > Training Job > Stop** on the PyCharm menu bar to stop the training job.

Figure 4-9 Stopping a job

Viewing Training Logs

You can view training logs in OBS or PyCharm Toolkit.

- **Viewing training logs in OBS**

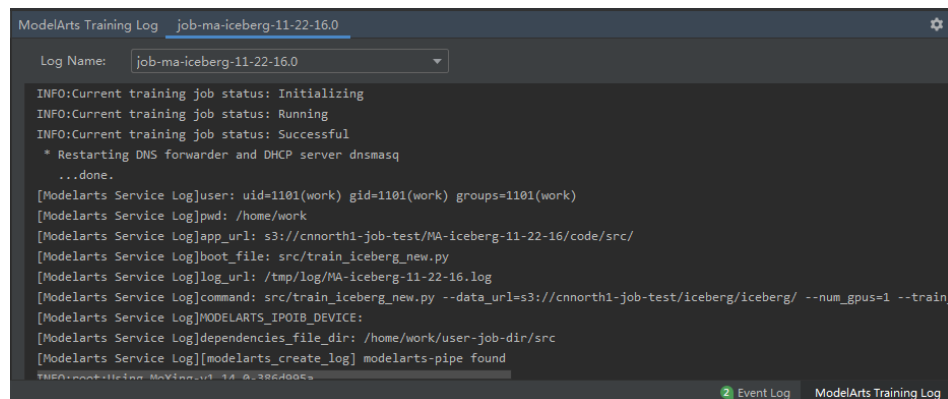
When you submit a training job, the system automatically creates a folder with the same name as the training job in the configured OBS path to store the model, logs, and code outputted during training.

For example, when the **train-job-01** job is submitted, a folder named **train-job-01** is created in the **test-modelarts2** bucket. In this folder, three sub-folders (**output**, **log**, and **code**) are created to store the outputted model, logs, and training code, respectively. Sub-folders will be created in the **output** folder based on your training job version. The following is an example of the folder structure:

```
test-modelarts2
|--train-job-01
|   |--output
|   |--log
|   |--code
```

- **Viewing training logs in PyCharm Toolkit**

In PyCharm Toolkit, click **ModelArts Training Log** in the lower right corner of the page to view the training logs.

Figure 4-10 Viewing training logs

4.5 Creating an Algorithm

Machine learning explores general rules from limited volume of data and uses these rules to predict unknown data. To obtain more accurate prediction results, select a proper algorithm to train your model. ModelArts provides a large number of algorithm samples for different scenarios. This section describes algorithm sources and learning modes.

Algorithm Sources

ModelArts provides the following algorithm sources for model training:

- Using a preset image

To use a custom algorithm, use a framework built in ModelArts. ModelArts supports most mainstream AI engines. For details, see [Starting a Preset Image's Boot File](#). These built-in engines pre-load some extra Python packages, such as NumPy. You can also use the **requirements.txt** file in the code directory to install dependency packages. For details about how to create a training job using a preset image, see [Developing Code for Training Using a Preset Image](#).
- Using a preset image with customization

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

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

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

Custom images can be used to train models in ModelArts only after they are uploaded to Software Repository for Container (SWR). For details, see

Creating a Custom Image for Model Training. Customizing an image requires a deep understanding of containers. Use this method only if the subscribed algorithms and custom scripts cannot meet your requirements.

 NOTE

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

Creating an Algorithm

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

1. Make preparations.
 - Create a dataset in ModelArts or upload a training dataset to an OBS directory.
 - Your training script has been uploaded to an OBS directory. For details about how to develop a training script, see [Developing Code for Training Using a Preset Image](#) or [Developing Code for Training Using a Custom Image](#).
 - Create at least one empty folder in OBS for storing training outputs.
 - Make sure your OBS directory and ModelArts are in the same region.
2. Access the algorithm creation page.
 - a. Log in to the ModelArts console. In the navigation pane, choose **Asset Management > Algorithm Management**.
 - b. In the **My algorithm** tab, click **Create Algorithm**. Enter the basic algorithm information, including **Name** and **Description**.
3. Set the algorithm boot mode. The options are as follows:
 - Using a preset image
Set **Code Directory** and **Boot File** based on the algorithm code. Ensure that the preset image you select is the same as the one you use for editing algorithm code. For example, if TensorFlow is used for writing algorithm code, select TensorFlow when you create an algorithm.

Table 4-7 Parameters

Parameter	Description
Boot Mode	Select Preset image . Select a preset image and its version used by the algorithm.

Parameter	Description
Code Directory	Select an OBS path for storing the algorithm code. The files required for training, such as the training code, dependency installation packages, and pre-generated models, are uploaded to the code directory. Do not store training data in the code directory. When the training job starts, the data stored in the code directory will be downloaded to the backend. A large amount of training data may lead to a download failure. After you create the training job, ModelArts downloads the code directory and its subdirectories to the training container. 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. NOTE - Any programming language is supported. - The total number of both files and folders cannot exceed 1,000. - The total size of files cannot exceed 5 GB.
Boot File	The file must be stored in the code directory and end with .py. ModelArts supports boot files edited only in Python. The boot file in the code directory is used to start a training job.

- Using a preset image with customization

Set **Image**, **Code Directory**, and **Boot File** based on the algorithm code. Ensure that the preset image you select is the same as the one you use for editing algorithm code. For example, if TensorFlow is used for writing algorithm code, select TensorFlow when you create an algorithm.

Table 4-8 Parameters

Parameter	Description
Boot Mode	Select Preset image . Select Customize for the engine version.
Image	Select your image uploaded to SWR. For details about how to create an image, see Creating a Custom Training Image .

Parameter	Description
Code Directory	Select an OBS path for storing the algorithm code. The files required for training, such as the training code, dependency installation packages, and pre-generated models, are uploaded to the code directory. Do not store training data in the code directory. When the training job starts, the data stored in the code directory will be downloaded to the backend. A large amount of training data may lead to a download failure. When the training job starts, ModelArts downloads the training code directory and its subdirectories to the training container. 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. NOTE - Any programming language is supported for training code. The training boot file must be a Python file. - The total number of both files and folders cannot exceed 1,000. - The total size of files cannot exceed 5 GB. - The file depth cannot exceed 32.
Boot File	The file must be stored in the code directory and end with .py. ModelArts supports boot files edited only in Python. The boot file in the code directory is used to start a training job.

Selecting a preset image with customization results in the same background behavior as running a training job directly with that image. For example:

- The system automatically injects environment variables.

```
PATH=${MA_HOME}/anaconda/bin:${PATH}
LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
```
- The selected boot file will be automatically started using Python commands. Ensure that the Python environment is correct. The **PATH** environment variable is automatically injected. Run the following commands to check the Python version for the training job:

```
export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
docker run --rm {image} $(which python) -V
```

- The system automatically adds hyperparameters associated with the preset image.
- Using a custom image

Table 4-9 Parameters

Parameter	Description
Boot Mode	Select Custom image .
Image	Select your image uploaded to SWR. For details about how to create an image, see Creating a Custom Training Image .
Code Directory	Select an OBS path for storing the algorithm code. The files required for training, such as the training code, dependency installation packages, and pre-generated models, are uploaded to the code directory. Configure this parameter only if your custom image does not contain training code. Do not store training data in the code directory. When the training job starts, the data stored in the code directory will be downloaded to the backend. A large amount of training data may lead to a download failure. When the training job starts, ModelArts downloads the training code directory and its subdirectories to the training container. 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. NOTE • Any programming language is supported for training code. The training boot file must be a Python file. • The total number of both files and folders cannot exceed 1,000. • The total size of files cannot exceed 5 GB. • The file depth cannot exceed 32.

Parameter	Description
Boot Command	Command for booting an image. This parameter is mandatory. 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. <code>python \${MA_JOB_DIR}/demo-code/train.py</code> • If the training boot script is a .sh file, main.sh for example, the boot command is as follows: <code>bash \${MA_JOB_DIR}/demo-code/main.sh</code> 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.

For details about how to use custom images supported by training, see [Boot Command Specifications for Custom Images](#).

4. Configure pipelines.

An 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 [Preparing Model Training Code](#).

- Input configurations

Table 4-10 Input configurations

Parameter	Description
Parameter Name	Set this parameter based on the data input parameter in your algorithm code. The code path parameter must be the same as the training input parameter parsed in your algorithm code. Otherwise, the algorithm code cannot obtain the input data. For example, If you use argparse in the algorithm code to parse data_url into the data input, set the data input parameter to data_url when creating the algorithm.
Description	Customize the description of the input parameter.
Obtained from	Select a source of the input parameter, Hyperparameters (default) or Environment variables .

Parameter	Description
Constraints	Enable this parameter to specify the input source. You can select a storage path or ModelArts dataset. This parameter is optional.
Add	Add multiple input data sources based on your algorithm.

- Output configurations

Table 4-11 Output configurations

Parameter	Description
Parameter Name	Set this parameter 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. For example, if you use argparse in the algorithm code to parse train_url into the data output, set the data output parameter to train_url when creating the algorithm.
Description	Customize the description of the output parameter.
Obtained from	Select a source of the output parameter, Hyperparameters (default) or Environment variables .
Add	Add multiple output data paths based on your algorithm.

5. Define hyperparameters.

When you create an algorithm, ModelArts allows you to customize hyperparameters so you can view or modify them anytime. Defined hyperparameters are displayed in the boot command and passed to your boot file as CLI parameters.

- Click **Add hyperparameter** to manually add hyperparameters.
- Edit hyperparameters.

 NOTE

To ensure data security, do not enter sensitive information, such as plaintext passwords.

Table 4-12 Hyperparameter parameters

Parameter	Description
Name	Enter the hyperparameter name. Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Type	Select the data type of the hyperparameter. The value can be String , Integer , Float , or Boolean
Default	Set the default value of the hyperparameter. This value will be used for training jobs by default.
Restrained	Click Restrained . Then, set the range of the default value or enumerated value in the dialog box displayed.
Required	Select 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, you cannot delete the hyperparameter on the training job creation page when using this algorithm to create a training job.
Description	Enter the description of the hyperparameter. Only letters, digits, spaces, hyphens (-), underscores (_), commas (,), and periods (.) are allowed.

6. Configure supported policies.

Auto search on ModelArts automatically finds the optimal hyperparameters without any code modification.

Only the **tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64** and **pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64** images are available for auto search.

7. Add training constraints.

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

- **Resource Type**: Select the required resource types.
- **Multicard Training**: Choose whether to support multi-card training.
- **Distributed Training**: Choose whether to support distributed training.

8. Click **Submit**.

In the algorithm list, click the algorithm to access its details page and view the algorithm details.

- In the **Basic Information** tab, you can view the algorithm information.
In the **Basic Information** tab, click **Edit** to modify algorithm information except the name and ID. After the modification, click **Save**.
- In the **Training** tab, you can view the information about the training jobs that use the algorithm, such as the training job name and status.

Previewing the Runtime Environment



When creating an algorithm, click the arrow on the right corner of the page to know the paths of the code directory, boot file, and input and output data in the training container.

Deleting an Algorithm

NOTICE

Deleted algorithm assets cannot be restored.

To delete your algorithm, choose **Asset Management > Algorithm Management**. Click **Delete** in the **Operation** column. In the displayed dialog box, click **OK** to confirm the deletion.

4.6 Creating a Training Job

Model training continuously iterates and optimizes model weights. ModelArts training management allows you to create training jobs, view training status, and manage training versions. Through model training, you can test various combinations of model structures, data, and hyperparameters to obtain the optimal model structure and weight.

Create a production training job in either of the following ways:

- Use the ModelArts Standard console. For details, see the following sections.
- Use the ModelArts API to create a production training job. For details, see [Using PyTorch to Create a Training Job \(New-Version Training\)](#).

Prerequisites

- Data for training uploaded to an OBS directory.
- At least one empty folder in OBS for storing training output.

NOTE

ModelArts does not support encrypted OBS buckets. When creating an OBS bucket, do not enable bucket encryption.

- Access authorization configured. If it is not configured, configure it by referring to [Configuring Access Authorization for ModelArts Standard](#).
- Training algorithm. For details, see [Creating an Algorithm](#).

Procedure

To create a training job, follow these steps:

Step 1 Follow the steps in [Accessing the Page for Creating a Training Job](#).

Step 2 Follow the steps in [Configuring Basic Information](#).

Step 3 Select an algorithm type.

- Use an existing algorithm to create a training job by referring to [Choosing an Algorithm Type \(My Algorithm\)](#).
- Use a preset image to create a training job by referring to [Choosing an Algorithm Type \(Custom Algorithm\)](#).
- Use a custom image to create a training job by referring to [Choosing a Boot Mode \(Custom Image\)](#).

Step 4 Configure training parameters, including the input, output, hyperparameters, and environment variables. For details, see [Configuring Training Parameters](#).

Step 5 Select a resource pool as needed. A dedicated resource pool is recommended for optimal performance. For details about the differences between dedicated and public resource pools, see [Differences Between Dedicated Resource Pools and Public Resource Pools](#).

- [Configuring a Public Resource Pool](#)
- [Configuring a Dedicated Resource Pool](#)

Step 6 Perform follow-up procedure. For details, see [Follow-Up Operations](#).

----End

Accessing the Page for Creating a Training Job

1. Log in to the ModelArts console.
2. In the navigation pane, choose .
3. Click **Create Training Job**.

Configuring Basic Information

On the **Create Training Job** page, configure parameters.

Table 4-13 Basic information

Parameter	Description
Name	Job name, which is mandatory. The system automatically generates a name, which you can then rename according to the following rules. • The name contains 1 to 64 characters. • Letters, digits, hyphens (-), and underscores (_) are allowed.
Description	Job description, which helps you learn about the job information in the training job list.

Parameter	Description
Experiment	Experiment for classifying and managing the job. • If you select Create new, enter the experiment name and description. • If you select Use existing, select an experiment name. • If you select Not required, this job will not be managed in any experiment.

Choosing an Algorithm Type (My Algorithm)

Set **Algorithm Type** to **My algorithm** and select an algorithm from the algorithm list. If no algorithm meets the requirements, you can create an algorithm. For details, see [Creating an Algorithm](#).

Choosing an Algorithm Type (Custom Algorithm)

If an algorithm is available in algorithm management, choose **My algorithm**. If no algorithm is available, choose **Custom algorithm**. If you use a custom algorithm to create a training job, select a boot mode by referring to [Table 4-14](#).

Table 4-14 Creating a training job using a custom algorithm

Parameter	Description
Algorithm Type	Select Custom algorithm . This parameter is mandatory.
Boot Mode	Select Preset image and select the preset image engine and engine version to be used by the training job. If you select Customize for the engine version, select a custom image from Image .
Image	This parameter is displayed and mandatory only when the preset image version is set to Customize . You can set the container image path in either of the following ways: • To select your image or an image shared by others, click Select on the right and select a container image for training. The required image must be uploaded to SWR beforehand. • To select a public image, enter the address of the public image in SWR. Enter the image path in the format of "Organization name/Image name:Version name". Do not contain the domain name in the path because the system will automatically add the domain name to the path.

Parameter	Description
Code Source	Select a training code source. • OBS: Select OBS if the training code is stored in an OBS bucket. • SFS: Select SFS if the training code is stored in an SFS file system.
Code Directory	This parameter is available only when Code Source is set to OBS. Select the OBS directory where the training code file is stored. This parameter is mandatory. • 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	Select the Python boot script of the training job in the code directory. This parameter is mandatory. ModelArts supports only the boot file written in Python. Therefore, the boot file must end with .py.
Local Code Directory	This parameter is available only when Code Source is set to OBS. Specify the local directory of a training container. When a training 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	During training, the system automatically runs the cd command to execute the boot file in this directory.

Selecting a preset image with customization results in the same background behavior as running a training job directly with that image. For example:

- The system automatically injects environment variables.

```
PATH=${MA_HOME}/anaconda/bin:${PATH}
LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
```
- The selected boot file will be automatically started using Python commands. Ensure that the Python environment is correct. The **PATH** environment

variable is automatically injected. Run the following commands to check the Python version for the training job:

```
export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
docker run --rm {image} $(which python) -V
```

- The system automatically adds hyperparameters associated with the preset image.

Choosing a Boot Mode (Custom Image)

If you use a custom image to create a training job, select a boot mode by referring to [Table 4-15](#).

Table 4-15 Creating a training job using a custom image

Parameter	Description
Algorithm Type	Select Custom algorithm . This parameter is mandatory.
Boot Mode	Select Custom image . This parameter is mandatory.
Image	Container image path. This parameter is mandatory. You can set the container image path in either of the following ways: • To select your image or an image shared by others, click Select on the right and select a container image for training. The required image must be uploaded to SWR beforehand. • To select a public image, enter the address of the public image in SWR. Enter the image path in the format of "Organization name/Image name:Version name". Do not contain the domain name in the path because the system will automatically add the domain name to the path.
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.

Parameter	Description
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
Boot Command	Command for booting an image. This parameter is mandatory. 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. <code>python \${MA_JOB_DIR}/demo-code/train.py</code> • If the training boot script is a .sh file, main.sh for example, the boot command is as follows: <code>bash \${MA_JOB_DIR}/demo-code/main.sh</code> 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. NOTE To ensure data security, do not enter sensitive information, such as plaintext passwords.
Local Code Directory	This parameter is available only when Code Source is set to OBS. Specify the local directory of a training container. When a training 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	During training, the system automatically runs the cd command to execute the boot file in this directory.

For details about how to use custom images supported by training, see [Boot Command Specifications for Custom Images](#).

Configuring Training Parameters

Data is obtained from an OBS bucket or dataset for model training. The training output can also be stored in an OBS bucket. When creating a training job, you can configure parameters such as input, output, hyperparameters, and environment variables by referring to [Table 4-16](#).

 NOTE

The input, output, and hyperparameter parameters of a training job vary depending on the algorithm type selected during training job creation. If a parameter value is dimmed, the parameter has been configured in the algorithm code and cannot be modified.

Table 4-16 Configuring training parameters

Parameter	Sub-Parameter	Description
Input	Parameter name	The algorithm code reads the training input data based on the input parameter name. The recommended value is data_url. The training input parameters must match the input parameters of the selected algorithm. For details, see Table 4-10.
	Dataset	Click Dataset and select the target dataset and its version in the ModelArts dataset list. When the training job is started, ModelArts automatically downloads the data in the input path to the training container. NOTE ModelArts data management is being upgraded and is invisible to users who have not used data management. It is recommended that new users store their training data in OBS buckets.
	Data path	Click Data path and select the storage path to the training input data from an OBS bucket. Files must not exceed 10 GB in total size, 1,000 in number, or 1 GB per file. When the training job is started, ModelArts automatically downloads the data in the input path to the training container.
	Obtained from	The following uses training input data_path as an example. - If you select Hyperparameters, use this code to obtain the data:<pre>import argparse parser = argparse.ArgumentParser() parser.add_argument('--data_path') args, unknown = parser.parse_known_args() data_path = args.data_path</pre> - If you select Environment variables, use this code to obtain the data:<pre>import os data_path = os.getenv("data_path", "")</pre>

Parameter	Sub-Parameter	Description
Output	Parameter name	The algorithm code reads the training output data based on the output parameter name. The recommended value is train_url. The training output parameters must match the output parameters of the selected algorithm. For details, see Table 4-11.
	Data path	Click Data path and select the storage path for the training output data from an OBS bucket. Files must not exceed 1 GB in total size, 128 in number, or 128 MB per file. During training, the system automatically synchronizes files from the local code directory of the training container to the data path. NOTE The data path can only be an OBS path. To prevent any issues with data storage, choose an empty directory as the data path.
	Obtained from	The following uses the training output train_url as an example. • If you select Hyperparameters, use this code to obtain the data:<pre>import argparse parser = argparse.ArgumentParser() parser.add_argument('--train_url') args, unknown = parser.parse_known_args() train_url = args.train_url</pre> • If you select Environment variables, use this code to obtain the data:<pre>import os train_url = os.getenv("train_url", "")</pre>
	Predownload	Indicates whether to pre-download the files in the output directory to a local directory. • If you set Predownload to No, the system does not download the files in the training output data path to a local directory of the training container when the training job is started. • If you set Predownload to Yes, the system automatically downloads the files in the training output data path to a local directory of the training container when the training job is started. The larger the file size, the longer the download time. To avoid excessive training time, remove any unneeded files from the local code directory of the training container as soon as possible. To use Resumable Training, select Yes.

Parameter	Sub-Parameter	Description
Hyperparameter	N/A	Used for training tuning. This parameter is determined by the selected algorithm. If hyperparameters have been defined in the algorithm, all hyperparameters in the algorithm are displayed. Hyperparameters can be modified and deleted. The status depends on the hyperparameter constraint settings in the algorithm. For details, see Table 4-12. To import hyperparameters in batches, click Upload. You will need to fill in the hyperparameters based on the provided template. The total number of hyperparameters should not exceed 100, or the import will fail. NOTE To ensure data security, do not enter sensitive information, such as plaintext passwords.
Environment Variable	N/A	Add environment variables based on service requirements. For details about the environment variables preset in the training container, see Managing Environment Variables of a Training Container. To import environment variables in batches, click Upload. You will need to fill in the environment variables based on the provided template. The total number of environment variables should not exceed 100, or the import will fail. NOTE To ensure data security, do not enter sensitive information, such as plaintext passwords.

Parameter	Sub-Parameter	Description
Auto Restart	N/A	Once this feature is enabled, you can set the number of restarts and whether to enable Unconditional auto restart. After you enable auto restart, ModelArts will handle any exceptions caused by environmental issues during a training job. It will either automatically handle the exception or isolate the faulty node and then restart the job, which helps to increase the success rate of the training. To avoid losing training progress and make full use of compute power, ensure that your code logic supports resumable training before enabling this function. For details, see Resumable Training. The value ranges from 1 to 128. The default value is 3. The value cannot be changed once the training job is created. Set this parameter based on your needs. If Unconditional auto restart is selected, the training job will be restarted unconditionally once the system detects a training exception. To prevent invalid restarts, it supports a maximum of three consecutive unconditional restarts. ModelArts continuously monitors job processes to detect suspension and optimize resource usage. When Restart Upon Suspension is enabled, suspended jobs can be automatically restarted at the process level. However, ModelArts does not verify code logic, and suspension detection is periodic, which may result in false reports. By enabling this feature, you acknowledge the possibility of false positives. To prevent unnecessary restarts, ModelArts limits consecutive restarts to three. If auto restart is triggered during training, the system records the restart information. You can check the fault recovery details on the training job details page. For details, see Training Job Rescheduling.

Configuring a Public Resource Pool

To configure a public resource pool, refer to [Table 4-17](#).

Table 4-17 Configuring a public resource pool for a training job

Parameter	Description
Resource Pool	Select Public resource pool .

Parameter	Description
Resource Type	Select the resource type required for training. This parameter is mandatory. If a resource type has been defined in the training code, select a proper resource type based on algorithm constraints. For example, if the resource type defined in the training code is CPU and you select other types, the training fails. If some resource types are invisible or unavailable for selection, they are not supported.
Specifications	Select the required resource specifications based on the resource type. If Data path is selected for Input, you can click Check Input Size on the right to ensure the storage is larger than the input data size.  ★ Specifications NOTICE The resource flavor GPU:n*tnt004 (<i>n</i> indicates a specific number) does not support multi-process training.
Compute Nodes	Select the number of instances as required. The default value is 1. • If only one instance is used, a single-node training job is created. ModelArts starts one training container on this node. The training container exclusively uses the compute resources of the selected flavor. • If more than one instance is used, a distributed training job is created. For more information about distributed training configurations, see Overview.
Persistent Log Saving	If you select CPU or GPU flavors, Persistent Log Saving is available for you to configure. • After this feature is enabled (default), configure Job Log Path. The system permanently stores training logs to the specified OBS path. • After this feature is disabled, ModelArts automatically stores the logs for 30 days. You can download all logs on the job details page to a local path.
Job Log Path	When enabling Persistent Log Saving, select an empty OBS directory for Job Log Path to store log files generated by the training job. Ensure that you have read and write permissions to the selected OBS directory.

Parameter	Description
Event Notification	Indicates whether to enable event notification. • This feature is disabled by default, which means SMN is disabled. • After this feature is enabled, you will be notified of specific events, such as job status changes or suspected suspensions, via an SMS or email. Notifications will be billed based on SMN pricing. In this case, you must configure the topic name and events. – Topic: topic of event notifications. Click Create Topic to create a topic on the SMN console. – Event: events you want to subscribe to. Examples: 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. • Only training jobs using GPUs or NPUs support JobHanged events.
Auto Stop	• This function is disabled by default, the training job keeps running until the training is completed. • If this function is enabled, configure the auto stop time. The value can be 1 hour, 2 hours, 4 hours, 6 hours, or Customize. The customized time must range from 1 hour to 720 hours. When you enable this feature, the training stops automatically when the time limit is reached. The time limit does not count down when the training is paused.
training_ssh_configure_nodes	Whether to enable password-free SSH mutual trust between nodes. • This feature is disabled by default. • Enabling this feature requires you to configure the SSH key directory. This is where the automatically generated SSH key file will be stored within the training container. The default value is /home/ma-user/.ssh.

Configuring a Dedicated Resource Pool

To configure a dedicated resource pool, refer to [Table 4-18](#).

Table 4-18 Configuring a dedicated resource pool for a training job

Parameter	Description
Resource Pool	Select a dedicated resource pool. If you select a dedicated resource pool, you can view the status, node specifications, number of idle/fragmented nodes, number of available/total nodes, and number of cards of the resource pool. Hover over View in the Idle/Fragmented Nodes column to check fragment details and check whether the resource pool meets the training requirements.
Specifications	Select the required resource specifications based on the resource type. If Data path is selected for Input, you can click Check Input Size on the right to ensure the storage is larger than the input data size. ★ Specifications 32GB Data disk size Check Input Size Ensure the storage is larger than the input data size. NOTICE The resource flavor GPU:n*tnt004 (<i>n</i> indicates a specific number) does not support multi-process training.
Compute Nodes	Select the number of instances as required. The default value is 1. • If only one instance is used, a single-node training job is created. ModelArts starts one training container on this node. The training container exclusively uses the compute resources of the selected flavor. • If more than one instance is used, a distributed training job is created. For more information about distributed training configurations, see Overview.
Job Priority	When using a dedicated resource pool, you can set the priority of the training job. The value ranges from 1 to 3. The default priority is 1, and the highest priority is 3. • By default, the job priority can be set to 1 or 2. After the permission to set the highest job priority is configured, the priority can be set to 1 to 3. • If a training job is in the Pending state for a long time, you can change the job priority to reduce the queuing duration. For details, see Priority of a Training Job.

Parameter	Description
SFS Turbo	When ModelArts and SFS Turbo are directly connected, multiple SFS Turbo file systems can be mounted to a training job to store training data. Click Add Mount Configuration and set the following parameters: • Name: Select an SFS Turbo file system. • Mount Path: Enter the SFS Turbo mounting path in the training container. • Directory: Specify the SFS Turbo storage location. If you have configured the folder control permission, select a storage location. If you have not configured the folder control permission, retain the default value / or customize a location. • Mounting Mode: Permission on the mounted SFS Turbo file system. This parameter is displayed as Read/Write or Read-only based on the permission of the SFS Turbo storage location. If you have not configured the folder control permission, this parameter is unavailable. • Mount Options: Configure SFS mount parameters to accelerate and optimize training. Alternatively, retain the default settings below: mountOptions: - vers=3 - timeo=600 - nolock - hard NOTE • You can mount a file system multiple times, but each mount path must be distinct. A maximum of five disks can be mounted to a training job. • The mounting path cannot be a / directory or a default mounting path, such as /cache and /home/ma-user/modelarts.
Persistent Log Saving	If you select CPU or GPU flavors, Persistent Log Saving is available for you to configure. • After this feature is enabled (default), configure Job Log Path. The system permanently stores training logs to the specified OBS path. • After this feature is disabled, ModelArts automatically stores the logs for 30 days. You can download all logs on the job details page to a local path.
Job Log Path	When enabling Persistent Log Saving, select an empty OBS directory for Job Log Path to store log files generated by the training job. Ensure that you have read and write permissions to the selected OBS directory.

Parameter	Description
Event Notification	Indicates whether to enable event notification. • This feature is disabled by default, which means SMN is disabled. • After this feature is enabled, you will be notified of specific events, such as job status changes or suspected suspensions, via an SMS or email. Notifications will be billed based on SMN pricing. In this case, you must configure the topic name and events. – Topic: topic of event notifications. Click Create Topic to create a topic on the SMN console. – Event: events you want to subscribe to. Examples: 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. • Only training jobs using GPUs or NPUs support JobHanged events.
Auto Stop	• This feature is disabled by default, the training job keeps running until the training is completed. • If this feature is enabled, configure the auto stop time. The value can be 1 hour, 2 hours, 4 hours, 6 hours, or Customize. The customized time must range from 1 hour to 720 hours. When you enable this feature, the training stops automatically when the time limit is reached. The time limit does not count down when the training is paused.
training_ssh_configure_nodes	Whether to enable password-free SSH mutual trust between nodes. • This feature is disabled by default. • Enabling this feature requires you to configure the SSH key directory. This is where the automatically generated SSH key file will be stored within the training container. The default value is /home/ma-user/.ssh.

Follow-Up Operations

After parameter setting for creating a training job, click **Submit**. On the **Confirm** dialog box, click **OK**.

A training job runs for a period of time. You can go to the training job list to view the basic information about the training job.

- In the training job list, **Status** of a newly created training job is **Pending**.
- When the status of a training job changes to **Completed**, the training job is finished, and the generated model is stored in the corresponding output path.
- If the status is **Failed** or **Abnormal**, click the job name to go to the job details page and view logs for troubleshooting.

4.7 Distributed Model Training

4.7.1 Overview

ModelArts provides the following capabilities:

- Extensive built-in images, meeting your requirements
- Custom development environments set up using built-in images
- Extensive tutorials, helping you quickly understand distributed training
- Distributed training debugging in development tools such as PyCharm, VS Code, and JupyterLab

Constraints

- If the instance flavors are changed, you can only perform single-node debugging. You cannot perform distributed debugging or submit remote training jobs.
- Only the PyTorch and MindSpore AI frameworks can be used for multi-node distributed debugging. If you want to use MindSpore, each node must be equipped with eight cards.
- The OBS paths in the debugging code should be replaced with your OBS paths.
- PyTorch is used to write debugging code in this document. The process is the same for different AI frameworks. You only need to modify some parameters.

Advantages and Disadvantages of Single-Node Multi-Card Training Using DataParallel

- Straightforward coding: Only one line of code needs to be modified.
- Bottlenecks in communication: The master GPU is used to update and distribute parameter settings, which causes high communication costs.
- Unbalanced GPU loading: The master GPU is used to summarize outputs, calculate loss, and update weights. Therefore, the GPU memory and usage are higher than those of other GPUs.

Advantages of Multi-Node Multi-Card Training Using DistributedDataParallel

- Fast communication
- Balanced load
- Fast running speed

Related Chapters

- [Creating a Single-Node Multi-Card Distributed Training Job \(DataParallel\)](#): describes single-node multi-card training using DataParallel, and corresponding code modifications.
- [Creating a Multiple-Node Multi-Card Distributed Training Job \(DistributedDataParallel\)](#): describes multi-node multi-card training using DistributedDataParallel, and corresponding code modifications.

4.7.2 Creating a Single-Node Multi-Card Distributed Training Job (DataParallel)

This section describes how to perform single-node multi-card parallel training based on the PyTorch engine.

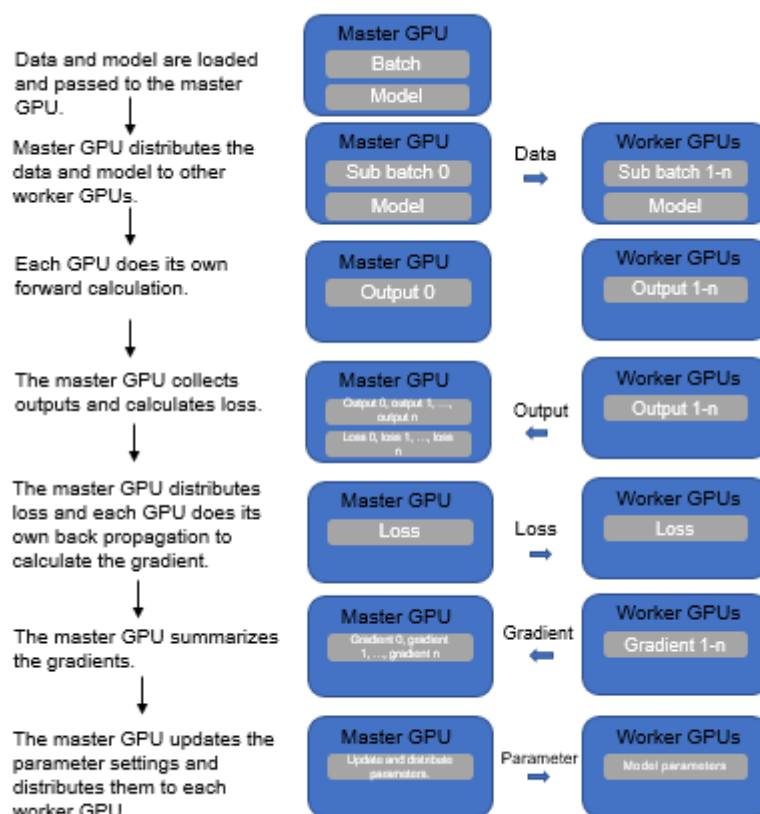
For details about the distributed training using the MindSpore engine, see [the MindSpore official website](#).

Training Process

The process of single-node multi-card parallel training is as follows:

1. A model is copied to multiple GPUs.
2. Data of each batch is distributed evenly to each worker GPU.
3. Each GPU does its own forward propagation and an output is obtained.
4. The master GPU with device ID 0 collects the output of each GPU and calculates the loss.
5. The master GPU distributes the loss to each worker GPU. Each GPU does its own backward propagation and calculates the gradient.
6. The master GPU collects gradients, updates parameter settings, and distributes the settings to each worker GPU.

The detailed flowchart is as follows.

Figure 4-11 Single-node multi-card parallel training

Code Modifications

Model distribution: `DataParallel(model)`

The code is slightly changed and the following is a simple example:

```
import torch
class Net(torch.nn.Module):
    pass

model = Net().cuda()

### DataParallel Begin ###
model = torch.nn.DataParallel(Net().cuda())
### DataParallel End ###
```

4.7.3 Creating a Multiple-Node Multi-Card Distributed Training Job (DistributedDataParallel)

To perform multi-node multi-card parallel training with PyTorch, follow the steps in this section and refer to the code example below. This section also includes a complete code sample for distributed parallel training on the CIFAR-10 dataset with ResNet18 for classification tasks.

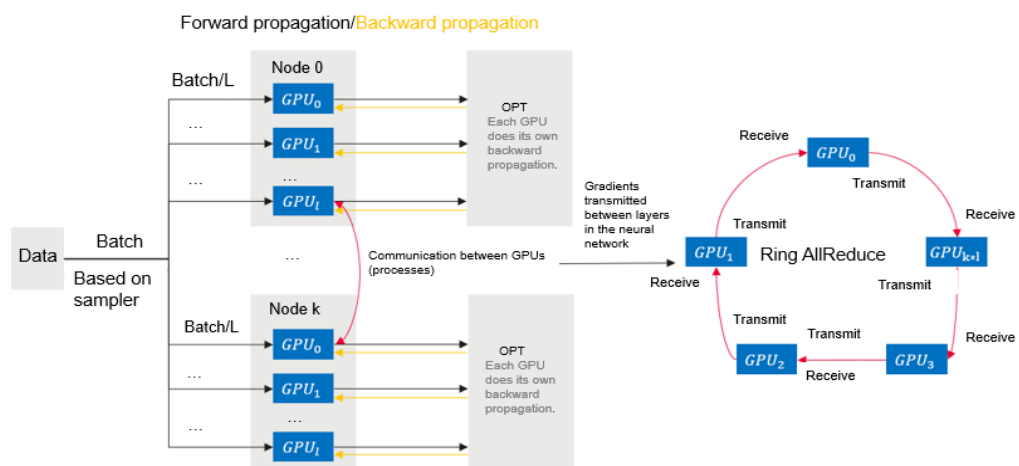
Training Process

Compared with DataParallel, DistributedDataParallel can start multiple processes for computing, greatly improving compute resource usage. Based on **torch.distributed**, DistributedDataParallel has obvious advantages over DataParallel in the distributed computing case. The process is as follows:

1. Initializes the process group.
2. Creates a distributed parallel model. Each process has the same model and parameters.
3. Creates a distributed sampler for data distribution to enable each process to load a unique subset of the original dataset in a mini batch.
4. Parameters are organized into buckets based on their shapes or sizes, which are generally determined by each layer of the network that requires parameter update in a neural network model.
5. Each process does its own forward propagation and computes its gradient.
6. After all parameter gradients at a bucket are obtained, communication is performed for gradient averaging.
7. Each GPU updates model parameters.

The detailed flowchart is as follows.

Figure 4-12 Multi-node multi-card parallel training



Code Modifications

- Multi-process startup
- New variables such as rank ID and world_size are used along with the TCP protocol.
- Sampler for data distribution to avoid duplicate data between different processes
- Model distribution: DistributedDataParallel(model)
- Model saved in GPU 0

```
import torch
class Net(torch.nn.Module):
```

```
pass

model = Net().cuda()

### DistributedDataParallel Begin ###
model = torch.nn.parallel.DistributedDataParallel(Net().cuda())
### DistributedDataParallel End ###
```

Multi-Node Distributed Debugging Adaptation and Code Example

In DistributedDataParallel, each process loads a subset of the original dataset in a batch, and finally the gradients of all processes are averaged as the final gradient. Due to a large number of samples, a calculated gradient is more reliable, and a learning rate can be increased.

This section describes the code of single-node training and distributed parallel training for the classification job of ResNet18 on the CIFAR-10 dataset. Directly execute the code to perform multi-node distributed training with CPUs or GPUs; comment out the distributed training settings in the code to perform single-node single-card training.

The training code contains three input parameters: basic training parameters, distributed parameters, and data parameters. The distributed parameters are automatically input by the platform. **custom_data** indicates whether to use custom data for training. If this parameter is set to **true**, torch-based random data is used for training and validation.

CIFAR-10 dataset

In notebook instances, torchvision of the default version cannot be used to obtain datasets. Therefore, the sample code provides three training data loading methods.

Click **CIFAR-10 python version** on the [download page](#) to download the CIFAR-10 dataset.

- Download the CIFAR-10 dataset using torchvision.
- Download the CIFAR-10 dataset based on the URL and decompress the dataset in a specified directory. The sizes of the training set and test set are (50000, 3, 32, 32) and (10000, 3, 32, 32), respectively.
- Use Torch to obtain a random dataset similar to CIFAR-10. The sizes of the training set and test set are (5000, 3, 32, 32) and (1000, 3, 32, 32), respectively. The labels are still of 10 types. Set **custom_data** to **true**, and the training task can be directly executed without loading data.

Training code

In the following code, those commented with **### Settings for distributed training ... ###** are code modifications for multi-node distributed training.

Do not modify the sample code. After the data path is changed to your path, multi-node distributed training can be executed on ModelArts.

After the distributed code modifications are commented out, the single-node single-card training can be executed. For details about the complete code, see [Code Example of Distributed Training](#).

- **Importing dependency packages**

```
import datetime
import inspect
import os
import pickle
import random

import argparse
import numpy as np
import torch
import torch.distributed as dist
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler
from sklearn.metrics import accuracy_score
```

- **Defining the method and random number for loading data** (The code for loading data is not described here due to its large amount.)

```
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

def get_data(path):
    pass
```

- **Defining a network structure**

```
class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)

class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
```

```
for stride in strides:
    layers.append(block(in_channels, out_channels, stride))
    in_channels = out_channels
return nn.Sequential(*layers)

def forward(self, x):
    out = self.conv1(x)
    out = self.conv2(out)
    out = self.conv3(out)
    out = self.conv4(out)
    out = self.conv5(out)
    out = self.avg_pool(out)
    out = out.view(out.size(0), -1)
    out = self.dense_layer(out)
    return out
```

- **Training and validation**

```
def main():
    file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)

    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        print('[warning] you are training on custom random dataset, '
              'validation accuracy may range from 0.4 to 0.6.')

    ### Settings for distributed training. Initialize DistributedDataParallel process. The init_method,
rank, and world_size parameters are automatically input by the platform. ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
                           rank=args.rank)
    ### Settings for distributed training. Initialize DistributedDataParallel process. The init_method,
rank, and world_size parameters are automatically input by the platform. ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### Settings for distributed training. Create a sampler for data distribution to ensure that different
    processes load different data. ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
    ### Settings for distributed training. Create a sampler for data distribution to ensure that different
    processes load different data. ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)
```



```
lr = args.lr * gpus_per_node
max_epoch = args.epochs
model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

### Settings for distributed training. Build a DistributedDataParallel model. ###
model = nn.parallel.DistributedDataParallel(model)
### Settings for distributed training. Build a DistributedDataParallel model. ###

optimizer = optim.Adam(model.parameters(), lr=lr)
loss_func = torch.nn.CrossEntropyLoss()

os.makedirs(args.output_dir, exist_ok=True)

for epoch in range(1, max_epoch + 1):
    model.train()
    train_loss = 0

    ### Settings for distributed training. DistributedDataParallel sampler. Random numbers are set
    for the DistributedDataParallel sampler based on the current epoch number to avoid loading
    duplicate data. ###
    tr_sampler.set_epoch(epoch)
    ### Settings for distributed training. DistributedDataParallel sampler. Random numbers are set
    for the DistributedDataParallel sampler based on the current epoch number to avoid loading
    duplicate data. ###

    for step, (tr_x, tr_y) in enumerate(tr_loader):
        if args.enable_gpu:
            tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
        out = model(tr_x)
        loss = loss_func(out, tr_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

    val_loss = 0
    pred_record = []
    real_record = []
    model.eval()
    with torch.no_grad():
        for step, (val_x, val_y) in enumerate(val_loader):
            if args.enable_gpu:
                val_x, val_y = val_x.cuda(), val_y.cuda()
            out = model(val_x)
            pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
            real_record += list(val_y.cpu().numpy())
            val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
    print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu),
          '\n')

    if args.rank == 0:
        # save ckpt every epoch
        torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))

if __name__ == '__main__':
    main()
```

- **Result comparison**

100-epoch **cifar-10** dataset training is completed using two resource types respectively: single-node single-card and two-node 16-card. The training duration and test set accuracy are as follows.

Table 4-19 Training result comparison

Resource Type	Single-Node Single-Card	Two-Node 16-Card
Duration	60 minutes	20 minutes
Accuracy	80+	80+

Code Example of Distributed Training

The following provides a complete code sample of distributed parallel training for the classification task of ResNet18 on the CIFAR-10 dataset.

The content of the training boot file **main.py** is as follows (if you need to execute a single-node and single-card training job, delete the code for distributed reconstruction):

```
import datetime
import inspect
import os
import pickle
import random
import logging

import argparse
import numpy as np
from sklearn.metrics import accuracy_score
import torch
from torch import nn, optim
import torch.distributed as dist
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler

file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)

def load_pickle_data(path):
    with open(path, 'rb') as file:
        data = pickle.load(file, encoding='bytes')
    return data

def _load_data(file_path):
    raw_data = load_pickle_data(file_path)
    labels = raw_data[b'labels']
    data = raw_data[b'data']
    filenames = raw_data[b'filenames']

    data = data.reshape(10000, 3, 32, 32) / 255
    return data, labels, filenames

def load_cifar_data(root_path):
    train_root_path = os.path.join(root_path, 'cifar-10-batches-py/data_batch_')
    train_data_record = []
    train_labels = []
    train_filenames = []
    for i in range(1, 6):
        train_file_path = train_root_path + str(i)
        data, labels, filenames = _load_data(train_file_path)
        train_data_record.append(data)
        train_labels += labels
        train_filenames += filenames
```

```
train_data = np.concatenate(train_data_record, axis=0)
train_labels = np.array(train_labels)

val_file_path = os.path.join(root_path, 'cifar-10-batches-py/test_batch')
val_data, val_labels, val_filenames = _load_data(val_file_path)
val_labels = np.array(val_labels)

tr_data = torch.from_numpy(train_data).float()
tr_labels = torch.from_numpy(train_labels).long()
val_data = torch.from_numpy(val_data).float()
val_labels = torch.from_numpy(val_labels).long()
return tr_data, tr_labels, val_data, val_labels

def get_data(root_path, custom_data=False):
    if custom_data:
        train_samples, test_samples, img_size = 5000, 1000, 32
        tr_label = [1] * int(train_samples / 2) + [0] * int(train_samples / 2)
        val_label = [1] * int(test_samples / 2) + [0] * int(test_samples / 2)
        random.seed(2021)
        random.shuffle(tr_label)
        random.shuffle(val_label)
        tr_data, tr_labels = torch.randn((train_samples, 3, img_size, img_size)).float(),
        torch.tensor(tr_label).long()
        val_data, val_labels = torch.randn((test_samples, 3, img_size, img_size)).float(), torch.tensor(
            val_label).long()
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    elif os.path.exists(os.path.join(root_path, 'cifar-10-batches-py')):
        tr_data, tr_labels, val_data, val_labels = load_cifar_data(root_path)
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    else:
        try:
            import torchvision
            from torchvision import transforms
            tr_set = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True, transform=transforms)
            val_set = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                  download=True, transform=transforms)
            return tr_set, val_set
        except Exception as e:
            raise Exception(
                f"{e}, you can download and unzip cifar-10 dataset manually, "
                "the data url is http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz")

class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
```

```
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)

class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out

    def setup_seed(seed):
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        np.random.seed(seed)
        random.seed(seed)
        torch.backends.cudnn.deterministic = True

def obs_transfer(src_path, dst_path):
    import moxing as mox
    mox.file.copy_parallel(src_path, dst_path)
    logging.info(f"end copy data from {src_path} to {dst_path}")

def main():
    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
```

```
parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
args, unknown = parser.parse_known_args()

args.enable_gpu = args.enable_gpu == 'true'
args.custom_data = args.custom_data == 'true'
args.lr = float(args.lr)
args.epochs = int(args.epochs)

if args.custom_data:
    logging.warning('you are training on custom random dataset, '
        'validation accuracy may range from 0.4 to 0.6.')

    ### Settings for distributed training. Initialize DistributedDataParallel process. The init_method, rank,
    and world_size parameters are automatically input by the platform. ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
        rank=args.rank)
    ### Settings for distributed training. Initialize DistributedDataParallel process. The init_method, rank,
    and world_size parameters are automatically input by the platform. ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### Settings for distributed training. Create a sampler for data distribution to ensure that different
    processes load different data. ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
    ### Settings for distributed training. Create a sampler for data distribution to ensure that different
    processes load different data. ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)

    lr = args.lr * gpus_per_node * args.world_size
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### Settings for distributed training. Build a DistributedDataParallel model. ###
    model = nn.parallel.DistributedDataParallel(model)
    ### Settings for distributed training. Build a DistributedDataParallel model. ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### Settings for distributed training. DistributedDataParallel sampler. Random numbers are set for
        the DistributedDataParallel sampler based on the current epoch number to avoid loading duplicate data.
        ###
        tr_sampler.set_epoch(epoch)
        ### Settings for distributed training. DistributedDataParallel sampler. Random numbers are set for
        the DistributedDataParallel sampler based on the current epoch number to avoid loading duplicate data.
        ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
train_loss += loss.item()
print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

val_loss = 0
pred_record = []
real_record = []
model.eval()
with torch.no_grad():
    for step, (val_x, val_y) in enumerate(val_loader):
        if args.enable_gpu:
            val_x, val_y = val_x.cuda(), val_y.cuda()
        out = model(val_x)
        pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
        real_record += list(val_y.cpu().numpy())
        val_loss += loss_func(out, val_y).item()
val_accu = accuracy_score(real_record, pred_record)
print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu), '\n')

if args.rank == 0:
    # save ckpt every epoch
    torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))

if __name__ == '__main__':
    main()
```

FAQs

1. How Do I Use Different Datasets in the Sample Code?

- To use the CIFAR-10 dataset in the preceding code, download and decompress the dataset and upload it to the OBS bucket. The file directory structure is as follows:

```
DDP
|-- main.py
|-- input_dir
|----- cifar-10-batches-py
|----- data_batch_1
|----- data_batch_2
|----- ...
```

DDP is the code directory specified during training job creation, **main.py** is the preceding code example (the boot file specified during training job creation), and **cifar-10-batches-py** is the unzipped dataset folder (stored in **input_dir**).

- To use user-defined random data, change the value of **custom_data** in the code example to **true**.

```
parser.add_argument('--custom_data', default='true')
```

Then, run **main.py**. The parameters for creating a training job are the same as those shown in the preceding figure.

2. Why Can I Leave the IP Address of the Master Node Blank for DDP?

The **init method** parameter in **parser.add_argument('--init_method', default=None, help='tcp_port')** contains the IP address and port number of the master node, which are automatically input by the platform.

4.8 Incremental Model Training

What Is Incremental Training?

Incremental learning is a machine learning method that enables AI models to learn from new data without restarting the training process. It builds on existing

knowledge, allowing the model to expand its capabilities and improve its performance over time.

Incremental learning allows for training on data in smaller chunks, reducing storage needs and alleviating resource constraints. It also conserves computing power and time, and lowers retraining costs.

Incremental training is ideal for these scenarios:

- Continuous data updates: It allows models to adapt to new data without retraining.
- Resource constraints: It is a more economical choice when retraining a model is too costly.
- Avoiding knowledge loss: It retains old knowledge while learning new information, preventing the model from forgetting what it has learned.

Incremental training is used in various fields, including natural language processing, computer vision, and recommendation systems. It makes AI systems more flexible and adaptable, allowing them to handle changing data in real-world environments.

Implementing Incremental Training in ModelArts Standard

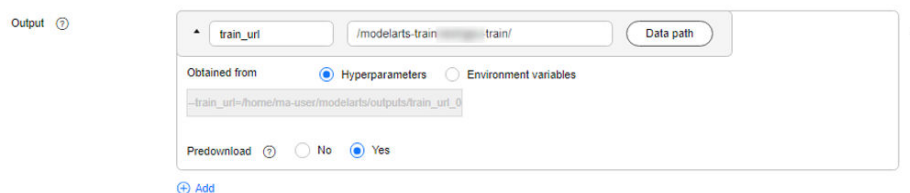
The checkpoint mechanism enables incremental training.

During model training, training results (including but not limited to epochs, model weights, optimizer status, and scheduler status) are continuously saved. To add data and resume a training job, load a checkpoint and use the checkpoint information to initialize the training status. To do so, add **reload ckpt** to the code.

To incrementally train a model in ModelArts, configure the training output.

When creating a training job, set the data path to the training output, save checkpoints in this data path, and set **Predownload** to **Yes**. If you set **Predownload** to **Yes**, the system automatically downloads the **checkpoint** file in the training output data path to a local directory of the training container before the training job is started.

Figure 4-13 Configuring training output



4.9 High Model Training Reliability

4.9.1 Training Job Fault Tolerance Check

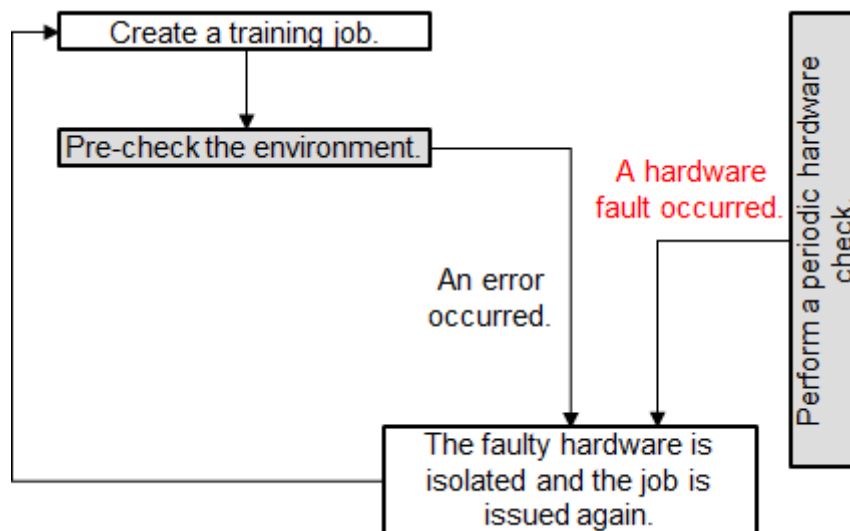
During model training, a training failure may occur due to a hardware fault. For hardware faults, ModelArts provides fault tolerance check to isolate faulty nodes to improve user experience in training.

The fault tolerance check involves environment pre-check and periodic hardware check. If any fault is detected during either of the checks, ModelArts automatically isolates the faulty hardware and issues the training job again. In distributed training, the fault tolerance check will be performed on all compute nodes used by the training job.

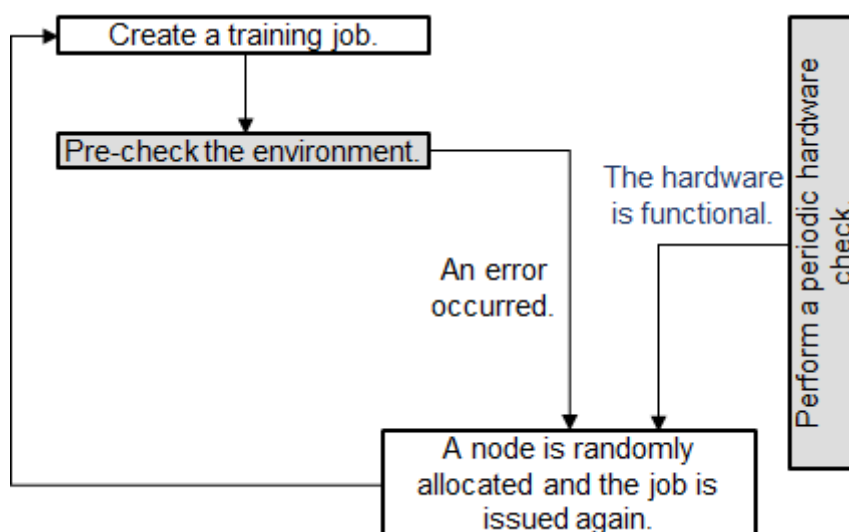
The following shows four failure scenarios, among which the failure in scenario 4 is not caused by a hardware fault. You can enable fault tolerance in the other three scenarios to automatically resume the training job.

- Scenario 1: The environment pre-check fails, and the hardware is faulty. Then, ModelArts automatically isolates all faulty nodes and issues the training job again.

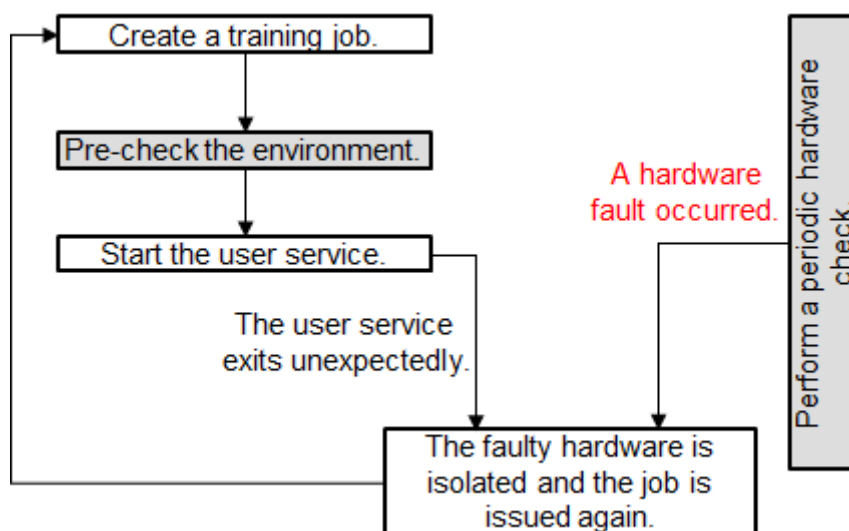
Figure 4-14 Pre-check failure and hardware fault



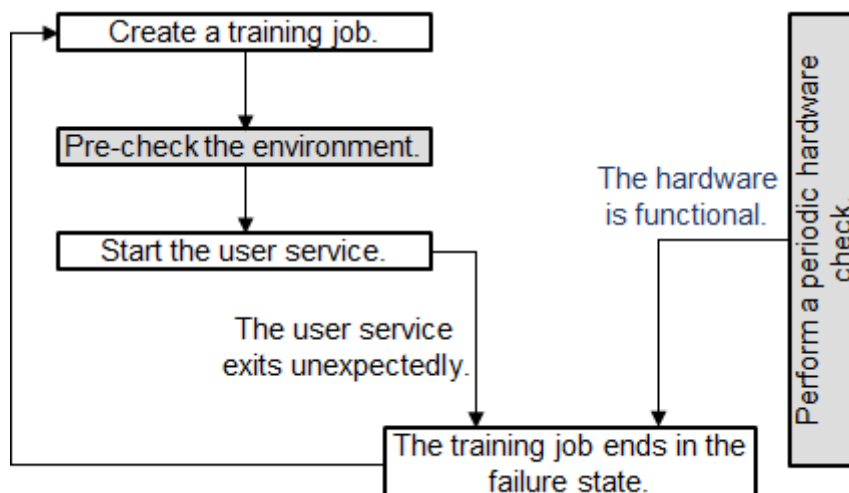
- Scenario 2: The environment pre-check fails but the hardware is functional. Then, ModelArts randomly allocates nodes and issues the training job again.

Figure 4-15 Pre-check failure but functional hardware

- Scenario 3: The environment pre-check is successful and the user service starts. A hardware fault occurs and the user service exits unexpectedly. Then, ModelArts automatically isolates all faulty nodes and issues the training job again.

Figure 4-16 Service failure and hardware fault

- Scenario 4: The environment pre-check is successful and the user service starts. The hardware is functional. A fault occurs in the user service, the training job ends in the failure state.

Figure 4-17 Service failure and functional hardware

After the faulty node is isolated, ModelArts creates a training job on new compute nodes. If the resources provided by the resource pool are limited, the re-issued training job will be queued with the highest priority. If the waiting time exceeds 30 minutes, the training job will automatically exit. This indicates that the resources are so limited that the training job cannot start. In this case, buy a dedicated resource pool to obtain dedicated resources.

If you use a dedicated resource pool to create a training job, the faulty nodes identified during the fault tolerance check will be removed. The system automatically adds healthy compute nodes to the dedicated resource pool. (This function is coming soon.)

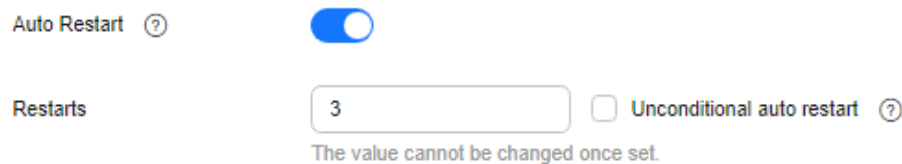
More details of a fault tolerance check:

1. [Enabling Fault Tolerance Check](#)
2. [Check Items and Conditions](#)
3. [Effect of a Fault Tolerance Check](#)
4. After the environment pre-check is successful, any hardware fault will interrupt the user service. Add the reload ckpt code logic to the training so that the pre-trained model saved before the training is interrupted can be obtained. For details, see [Resumable Training](#).

Enabling Fault Tolerance Check

To enable fault tolerance check, enable auto restart when creating a training job.

- Configure fault tolerance check on the ModelArts Standard console:
Enable **Auto Restart** on the ModelArts management console. **Auto Restart** is disabled by default, indicating that the job will not be re-issued and the environment pre-check will not be enabled. After **Auto Restart** is enabled, the number of restart retries ranges from 1 to 128.

Figure 4-18 Auto Restart

- Configure fault tolerance check using an API:

Enable auto restart upon a fault using an API. When creating a training job, configure the **fault-tolerance/job-retry-num** field in **annotations** of the **metadata** field.

If the **fault-tolerance/job-retry-num** field is added, auto restart is enabled. The value can be an integer ranging from **1** to **128**, specifying the maximum number of times that a job can be re-issued. If this hyperparameter is not specified, the default value **0** is used, indicating that the job will not be re-issued and the environment pre-check will not be enabled.

Figure 4-19 Setting the API

```
{
  "kind": "job",
  "metadata": {
    "annotations": {
      "fault-tolerance/job-retry-num": "3"
    }
  }
}
```

Check Items and Conditions

Check Item	Item (Log Keyword)	Execution Condition	Requirements for a Check
Domain name detection	dns	None	The domain names of the volcano containers in the .host file in /etc/volcano are successfully resolved.
Disk size - Container root directory	disk-size root	None	The directory is greater than 32 GB.
Disk size - /dev/shm	disk-size shm	None	The directory is greater than 1 GB.
Disk size - / cache	disk-size cache	None	The directory is greater than 32 GB.

Check Item	Item (Log Keyword)	Execution Condition	Requirements for a Check
ulimit check	ulimit	An IB network is used.	- Maximum locked memory > 16000 - Open files > 1000000 - Stack size > 8000 - Maximum user processes > 1000000
GPU check	gpu-check	GPU and the v2 training engine are used.	GPUs are detected.

Effect of a Fault Tolerance Check

- If the fault tolerance check is passed, the logs of the check items will be recorded, indicating that the check items are successful. You can search for the keyword **item** in the log file. A fault tolerance check minimizes reported runtime faults.
- If a fault tolerance check fails, check failure logs will be recorded. You can search for the keyword **item** in the log file to view the failure information.

If the number of job restarts does not reach the specified time, the job will be automatically issued again. You can search for keywords **error,exiting** to obtain the logs recording a restarted job that ends with a failure.

Using reload ckpt to Resume an Interrupted Training

With fault tolerance enabled, if a training job is restarted due to a hardware fault, you can obtain the pre-trained model in the code to restore the training to the state before the restart. To do so, add reload ckpt to the code. For details, see [Resumable Training](#).

4.9.2 Training Log Failure Analysis

If you encounter an issue during the execution of a ModelArts Standard training job, view logs first. In most scenarios, you can locate the issue based on the error information reported in logs.

If a training job fails, ModelArts Standard automatically identifies the failure cause and displays a message on the log page. The message consists of possible causes, recommended solutions, and error logs (marked in red).

ModelArts Standard provides possible causes (for reference only) and solutions for some common training faults. Not all faults can be identified. For a distributed job, only the analysis result of the current node is displayed. To obtain the failure cause of a training job, check the analysis results of all nodes used by the training job.

To rectify common training faults, perform the following steps:

1. Rectify the fault based on the analysis and suggestions provided on the log page.
 - Solution 1: A troubleshooting document is provided for you to follow.
 - Solution 2: Rebuild the training job and run it again.
2. If the fault persists, analyze the error information in the logs to locate and rectify the fault.
3. If the provided solutions cannot rectify your fault, you can submit a service ticket for technical support.

4.9.3 Training Job Rescheduling

When a training job fault occurs (such as process-level recovery, POD-level rescheduling, and job-level rescheduling), the **Fault Recovery Details** tab appears on the job details page, recording the start and stop details of the training job.

1. On the ModelArts console, choose **Model Training > Training Jobs** from the navigation pane.
2. In the training job list, click the name of the target job to go to the training job details page.
3. On the training job details page, click the **Fault Recovery Details** tab to view the fault recovery information.

4.9.4 Resumable Training

Overview

Resumable training indicates that an interrupted training job can be automatically resumed from the checkpoint where the previous training was interrupted. This method is applicable to model training that takes a long time.

The checkpoint mechanism enables resumable training.

During model training, training results (including but not limited to epochs, model weights, optimizer status, and scheduler status) are continuously saved. In this way, an interrupted training job can be automatically resumed from the checkpoint where the previous training was interrupted.

To resume a training job, load a checkpoint and use the checkpoint information to initialize the training status. To do so, add `reload ckpt` to the code.

Implementing Resumable Training in ModelArts Standard

To resume model training or incrementally train a model in ModelArts Standard, configure training output.

When creating a training job, set the data path to the training output, save checkpoints in this data path, and set **Predownload** to **Yes**. If you set **Predownload** to **Yes**, the system automatically downloads the **checkpoint** file in the training output data path to a local directory of the training container before the training job is started.

Enable fault tolerance check (auto restart) for resumable training. On the training job creation page, enable **Auto Restart**. If the environment pre-check fails, the hardware is not functional, or the training job fails, ModelArts will automatically issue the training job again.

4.9.5 Enabling Unconditional Auto Restart

To prevent training failures and delays, use unconditional auto restart. This feature automatically restarts a failed job, regardless of the cause, improving training success rates and job stability. To prevent invalid restarts, the system limits unconditional restarts to three consecutive attempts.

To avoid losing training progress, ensure your code can resume training from where it is interrupted, and then enable unconditional auto restart to optimize compute usage. For details, see [Resumable Training](#).

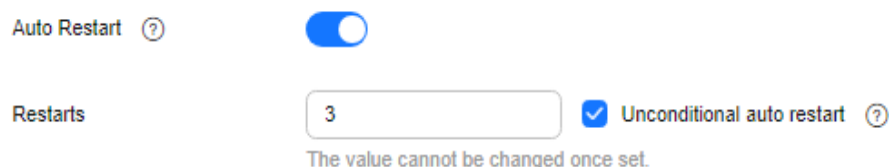
If auto restart is triggered during training, the system records the restart information. You can check the fault recovery details on the training job details page. For details, see [Training Job Rescheduling](#).

Procedure

You can enable unconditional auto restart either on the console or through an API.

- Using the console
On the training job creation page, enable **Auto Restart** and select **Unconditional auto restart**. If **Unconditional auto restart** is enabled, the training job will be restarted unconditionally once the system detects a training exception. If you enable auto restart but do not select **Unconditional auto restart**, the training job will only automatically restart if it encounters environmental issues. In case of any other problems, the status of the training job will become **Failed**.

Figure 4-20 Enabling unconditional auto restart



- Using an API
When creating a training job through an API, input the **fault-tolerance/job-retry-num** and **fault-tolerance/job-unconditional-retry** fields in **annotations** of the **metadata** field. To enable auto restart and set the number of restarts, set **fault-tolerance/job-retry-num** to an integer between 1 and 128. For unconditional auto restart, set **fault-tolerance/job-unconditional-retry** to **true**.

```
{
  "kind": "job",
  "metadata": {
    "annotations": {
      "fault-tolerance/job-retry-num": "8",
      "fault-tolerance/job-unconditional-retry": "true"
    }
  }
}
```

```
}  
}  
}
```

4.10 Managing Model Training Jobs

4.10.1 Viewing Training Job Details

1. Log in to the ModelArts console.
2. In the navigation pane, choose **Model Training > Training Jobs**.
In the job list, click **Export** to export training job details in a certain time range as an Excel file. A maximum of 200 rows of data can be exported.
3. In the training job list, click the target job name to switch to the training job details page.
4. On the left of the training job details page, view basic job settings and algorithm parameters.
 - **Basic job settings**

Table 4-20 Basic job settings

Parameter	Description
Job ID	Unique ID of the training job.
Status	Status of the training job.
Created	Time when the training job is created.
Duration	Duration of a training job, which is the total duration of Kubernetes resources in the entire lifecycle of a training job.
Retries	Number of times that the training job automatically restarts upon a fault during training. This parameter is available only when Auto Restart is enabled during training job creation.
Description	Description of the training job. You can click the edit icon to update the description of a training job.
Job Priority	Priority of the training job.

- **Algorithm parameters**

Table 4-21 Algorithm parameters

Parameter	Description
Algorithm Name	Algorithm used in the training job. You can click the algorithm name to go to the algorithm details page.

Parameter	Description
Preset images	Preset image used by the training job. This parameter is available only for training jobs created using a preset image.
Custom image	Custom image used by the training job. This parameter is available only for training jobs created using a custom image.
Code Directory	OBS path to the code directory of the training job. You can click Edit Code on the right to edit the training script code in OBS Online Editor. OBS Online Editor is not available for a training job in the Pending, Creating, or Running status. NOTE This parameter is not supported when you use a subscribed algorithm to create a training job.
Boot File	Location where the training boot file is stored. NOTE This parameter is not supported when you use a subscribed algorithm to create a training job.
User ID	ID of the user who runs the container.
Local Code Directory	Path to the training code in the training container.
Work Directory	Path to the training boot file in the training container.
Compute Nodes	Number of instances for the training job.
Dedicated resource pool	Dedicated resource pool information. This parameter is available only when a training job uses a dedicated resource pool.
Specifications	Training specifications used by the training job.
Input > Input Path	OBS path where the input data is stored.
Input > Parameter Name	Input path parameter specified in the algorithm code.
Input > Obtained from	Method of obtaining the training job input.
Input > Local Path (Training Parameter Value)	Path for storing the input data in the ModelArts backend container. After the training is started, ModelArts downloads the data stored in OBS to the backend container.

Parameter	Description
Output > Output Path	OBS path where the output data is stored.
Output > Parameter Name	Output path parameter specified in the algorithm code.
Output > Obtained from	Method of obtaining the training job output.
Output > Local Path (Training Parameter Value)	Path for storing the output data in the ModelArts backend container.
Hyperparameter	Hyperparameters used in the training job.
Environment Variable	Environment variables for the training job.

4.10.2 Viewing the Resource Usage of a Training Job

Notes and Constraints

Usage data of training job resources is stored for 30 days before being automatically deleted.

Operations

1. On the ModelArts console, choose **Model Training > Training Jobs** from the navigation pane.
2. In the training job list, click the name of the target job to go to the training job details page.
3. On the training job details page, click the **Resource Usages** tab to view the resource usage of the compute nodes. The data of at most the last three days can be displayed. When the resource usage window is opened, the data is loading and refreshed periodically.

Operation 1: If a training job uses multiple compute nodes, choose a node from the drop-down list box to view its metrics.

Operation 2: Click **cpuUsage**, **gpuMemUsage**, **gpuUtil**, **memUsage**, **npuMemUsage**, or **npuUtil** to show or hide the usage chart of the parameter.

Operation 3: Hover the cursor on the graph to view the usage at the specific time.

Table 4-22 Parameters

Parameter	Description
cpuUsage	CPU usage
gpuMemUsage	GPU memory usage
gpuUtil	GPU usage
memUsage	Memory usage
npuMemUsage	NPU memory usage
npuUtil	NPU usage

Alarms of Job Resource Usage

You can view the job resource usage on the training job list page. If the average GPU/NPU usage of the job's worker-0 instance is lower than 50%, an alarm is displayed in the training job list.

The job resource usage here involves only GPU and NPU resources. The method of calculating the average GPU/NPU usage of a job's worker-0 instance is:
Summarize the usage of each GPU/NPU accelerator card at each time point of the job's worker-0 instance and calculate the average value.

Improving Job Resource Utilization

- Increasing the value of **batch_size** increases GPU and NPU usage. You must decide the batch size that will not cause a memory overflow.
- If the time for reading data in a batch is longer than the time for GPUs or NPUs to calculate data in a batch, GPU or NPU usage may fluctuate. In this case, optimize the performance of data reading and data augmentation. For example, read data in parallel or use tools such as NVIDIA Data Loading Library (DALI) to improve the data augmentation speed.
- If a model is large and frequently saved, GPU or NPU usage is affected. In this case, do not save models frequently. Similarly, make sure that other non-GPU/NPU operations, such as log printing and training metric saving, do not affect the training process for too much time.

4.10.3 Viewing the Model Evaluation Result

After a training job has been executed, ModelArts evaluates your model and provides optimization diagnosis and suggestions.

- When you use a built-in algorithm to create a training job, you can view the evaluation result without any configurations. The system automatically provides optimization suggestions based on your model metrics. Read the suggestions and guidance on the page carefully to further optimize your model.
- For a training job created by writing a training script or using a custom image, you need to add the evaluation code to the training code so that you can

view the evaluation result and diagnosis suggestions after the training job is complete.

NOTE

- Only validation sets of the image type are supported.
- You can add the evaluation code only when the training scripts of the following frequently-used frameworks are used:
 - TF-1.13.1-python3.6
 - TF-2.1.0-python3.6
 - PyTorch-1.4.0-python3.6

This section describes how to use the evaluation code in a training job. To adapt and modify the training code, three steps are involved, [Adding the Output Path](#), [Copying the Dataset to the Local Host](#), and [Mapping the Dataset Path to OBS](#).

Adding the Output Path

The code for adding the output path is simple. That is, add a path for storing the evaluation result file to the code, which is called **train_url**, that is, the training output path on the console. Add **train_url** to the analysis function and use **save_path** to obtain **train_url**. The sample code is as follows:

```
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method": "FGSM", "eps": 40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)

...

# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
    label_map_dict=label_dict,
    save_path=FLAGS.train_url)
```

Copying the Dataset to the Local Host

Copying a dataset to the local host is to prevent the OBS connection from being interrupted due to long-time access. Therefore, copy the dataset to the local host before performing operations.

There are two methods for copying datasets. The recommended method is to use the OBS path.

- OBS path (recommended)
Call the `copy_parallel` API of MoXing to copy the corresponding OBS path.
- Dataset in ModelArts data management (manifest file format)
Call the `copy_manifest` API of MoXing to copy the file to the local host and obtain the path of the new manifest file. Then, use SDK to parse the new manifest file.

 NOTE

ModelArts data management is being upgraded and is invisible to users who have not used data management. It is recommended that new users store their training data in OBS buckets.

```
if data_path.startswith('obs://'):
    if 'manifest' in data_path:
        new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
        data_path = new_manifest_path
    else:
        mox.file.copy_parallel(data_path, '/cache/data/')
        data_path = '/cache/data/'
print('----- download dataset success -----')
```

Mapping the Dataset Path to OBS

The actual path of the image file, that is, the OBS path, needs to be entered in the JSON body. Therefore, after analysis and evaluation are performed on the local host, the original local dataset path needs to be mapped to the OBS path, and the new list needs to be sent to the analysis API.

If the OBS path is used as the input of **data_url**, you only need to replace the string of the local path.

```
if FLAGS.data_url.startswith('obs://'):
    for idx, item in enumerate(file_name_list):
        file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
```

If the manifest file is used, the original manifest file needs to be parsed again to obtain the list and then the list is sent to the analysis API.

```
if or FLAGS.data_url.startswith('obs://'):
    if 'manifest' in FLAGS.data_url:
        file_name_list = []
        manifest, _ = get_sample_list(
            manifest_path=FLAGS.data_url, task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                file_name_list.append(item[0])
```

An example code for image classification that can be used to create training jobs is as follows:

```
import json
import logging
import os
import sys
import tempfile

import h5py
import numpy as np
from PIL import Image

import moxing as mox
import tensorflow as tf
from deep_moxing.framework.manifest_api.manifest_api import get_sample_list
from deep_moxing.model_analysis.api import analyse, tmp_save
from deep_moxing.model_analysis.common.constant import TMP_FILE_NAME

logging.basicConfig(level=logging.DEBUG)

FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
```

```
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method":"FGSM","eps":40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)

def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img

def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x, 1, x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x

def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
        manifest, _ = get_sample_list(
            manifest_path=data_path, task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                label_list.append(label_map_dict.get(item[1][0]))
                img_name_list.append(item[0])
            else:
                continue
    else:
        label_name_list = os.listdir(data_path)
        label_dict = {}
        for idx, item in enumerate(label_name_list):
            label_dict[str(idx)] = item
            sub_img_list = os.listdir(os.path.join(data_path, item))
            img_name_list += [
                os.path.join(data_path, item, img_name) for img_name in sub_img_list
            ]
            label_list += [label_map_dict.get(item)] * len(sub_img_list)
    return img_name_list, label_list

def deal_ckpt_and_data_with_obs():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url

    if pb_dir.startswith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/')
        pb_dir = '/cache/ckpt'
        print('----- download success -----')
    if data_path.startswith('obs://'):
```

```
if 'manifest' in data_path:
    new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
    data_path = new_manifest_path
else:
    mox.file.copy_parallel(data_path, '/cache/data/')
    data_path = '/cache/data/'
print('----- download dataset success -----')
assert os.path.isdir(pb_dir), 'Error, pb_dir must be a directory'
return pb_dir, data_path

def evalution():
    pb_dir, data_path = deal_ckpt_and_data_with_obs()
    index_file = os.path.join(pb_dir, 'index')
    try:
        label_file = h5py.File(index_file, 'r')
        label_array = label_file['labels_list'][:].tolist()
        label_array = [item.decode('utf-8') for item in label_array]
    except Exception as e:
        logging.warning(e)
        logging.warning('index file is not a h5 file, try json.')
        with open(index_file, 'r') as load_f:
            label_file = json.load(load_f)
            label_array = label_file['labels_list'][:].tolist()
    label_map_dict = {}
    label_dict = {}
    for idx, item in enumerate(label_array):
        label_map_dict[item] = idx
        label_dict[idx] = item
    print(label_map_dict)
    print(label_dict)

    data_file_list, label_list = get_dataset(data_path, label_map_dict)

    assert len(label_list) > 0, 'missing valid data'
    assert None not in label_list, 'dataset and model not match'

    pred_list = []
    file_name_list = []
    img_list = []

    for img_path in data_file_list:
        img = _preprocess(img_path)
        img_list.append(img)
        file_name_list.append(img_path)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.visible_device_list = '0'
    with tf.Session(graph=tf.Graph(), config=config) as sess:
        meta_graph_def = tf.saved_model.loader.load(
            sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
        signature = meta_graph_def.signature_def
        signature_key = 'predict_object'
        input_key = 'images'
        output_key = 'logits'
        x_tensor_name = signature[signature_key].inputs[input_key].name
        y_tensor_name = signature[signature_key].outputs[output_key].name
        x = sess.graph.get_tensor_by_name(x_tensor_name)
        y = sess.graph.get_tensor_by_name(y_tensor_name)
        for img in img_list:
            pred_output = sess.run([y], {x: img})
            pred_output = softmax(pred_output[0])
            pred_list.append(pred_output[0].tolist())

    label_dict = json.dumps(label_dict)
    task_type = 'image_classification'

    if FLAGS.data_url.startswith('obs://'):
```

```
if 'manifest' in FLAGS.data_url:
    file_name_list = []
    manifest, _ = get_sample_list(
        manifest_path=FLAGS.data_url, task_type='image_classification')
    for item in manifest:
        if len(item[1]) != 0:
            file_name_list.append(item[0])
    for idx, item in enumerate(file_name_list):
        file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
    label_map_dict=label_dict,
    save_path=FLAGS.train_url)

if __name__ == "__main__":
    evaluation()
```

4.10.4 Viewing Training Job Events

Any key event of a training job will be recorded at the backend after the training job is displayed for you. You can check events on the training job details page.

This helps you better understand the running process of a training job and locate faults more accurately when a task exception occurs. The following job events are supported:

- Training job created.
- Training job failures:
- Preparations timed out. The possible cause is that the cross-region algorithm synchronization or creating shared storage timed out.
- The training job is queuing and awaiting resource allocation.
- Failed to be queued.
- The training job starts to run.
- Training job executed.
- Failed to run the training job.
- The training job is preempted.
- The system detects that your training job may be suspended. Go to the job details page to view the cause and handle the issue.
- The training job has been restarted.
- The training job has been manually stopped.
- The training job has been stopped. (Maximum running duration: x hours)
- The training job has been manually deleted.
- Billing information synchronized.
- [worker-0] The training environment is being pre-checked.
- [worker-0] [Duration: second] Pre-check completed.
- [worker-0] [Duration: second] Pre-check failed. Error: xxx
- [worker-0] [Duration: second] Pre-check failed. Error: xxx
- [worker-0] The training code is being downloaded.

- [worker-0] [Duration: second] Training code downloaded.
- [worker-0] [Duration: second] Failed to download the training code. Failure cause:
- [worker-0] The training input is being downloaded.
- [worker-0] [Duration: second] Training input (parameter: xxx) downloaded.
- [worker-0] [Duration: second] Failed to download the training input (parameter: xxx). Failure cause:
- [worker-0] Python dependency packages are being installed. Import the following files:
- [worker-0] [Duration: second] Python dependency packages installed. Import the following files:
- [worker-0] The training job starts to run.
- [worker-0] Training job executed.
- [worker-0] The training input is being uploaded.
- [worker-0] [Duration: second] Training output (parameter: xxx) uploaded.

During the training process, key events can be manually or automatically refreshed.

Notes and Constraints

The system automatically stores training job events for 30 days, and any expired events will be deleted.

Procedure

1. On the ModelArts console, choose **Model Training > Training Jobs** from the navigation pane.
2. In the training job list, click the name of the target job to go to the training job details page.
3. Click **Events** to view events.

4.10.5 Viewing Training Job Logs

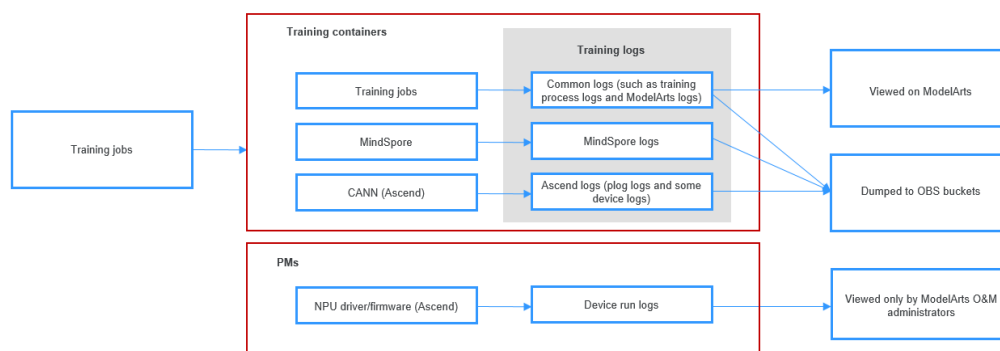
Overview

Training logs record the runtime process and exception information of training jobs and provide useful details for fault location. The standard output and standard error information in your code are displayed in training logs. If you encounter an issue during the execution of a ModelArts training job, view logs first. In most scenarios, you can locate the issue based on the error information reported in logs.

Training logs include common training logs and Ascend logs.

- **Common Logs:** When resources other than Ascend are used for training, only common training logs are generated. Common logs include the logs for **pip-requirement.txt**, training process, and ModelArts.

- **Ascend Logs:** When Ascend resources are used for training, device logs, plog logs, proc log for single-card training logs, MindSpore logs, and common logs are generated.

Figure 4-21 ModelArts training logs**NOTE**

Separate MindSpore logs are generated only in the MindSpore+Ascend training scenario. Logs of other AI engines are contained in common logs.

Retention Period

Logs are classified into the following types based on the retention period:

- Real-time logs: generated during training job running and can be viewed on the ModelArts training job details page.
- Historical logs: After a training job is completed, you can view its historical logs on the ModelArts training job details page. ModelArts automatically stores the logs for 30 days.
- Permanent logs: These logs are dumped to your OBS bucket. When creating a training job, you can enable persistent log saving and set a job log path for dumping.

Real-time logs and historical logs have no difference in content. In the Ascend training scenario, permanent logs contain Ascend logs, which are not displayed on ModelArts.

Common Logs

Common logs include the logs for **pip-requirement.txt**, training process, and ModelArts Standard.

Table 4-23 Common log types

Type	Description
Training process log	Standard output of your training code.

Type	Description
Installation logs for pip-requirement.txt	If pip-requirement.txt is defined in training code, pip package installation logs are generated.
ModelArts logs	ModelArts logs are used by O&M personnel to locate service faults.

The format of a common log file is as follows. **task id** is the node ID of a training job.

Unified log format: modelarts-job-[job id]-[task id].log

Example: log/modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log

- Single-node training jobs generate a log file, and **task id** defaults to **worker-0**.
- Distributed training generates multiple node log files, which are distinguished by **task id**, such as **worker-0** and **worker-1**.

Common logs include the logs for **pip-requirement.txt**, training process, and ModelArts.

ModelArts logs can be filtered in the common log file **modelarts-job-[job id]-[task id].log** using the following keywords: **[ModelArts Service Log]** or **Platform=ModelArts-Service**.

- Type 1: [ModelArts Service Log] xxx
[ModelArts Service Log][init] download code_url: s3://dgg-test-user/snt9-test-cases/mindspore/lenet/
- Type 2: time="xxx" level="xxx" msg="xxx" file="xxx" Command=xxx
Component=xxx Platform=xxx
time="2021-07-26T19:24:11+08:00" level=info msg="start the periodic upload task, upload period = 5 seconds " file="upload.go:46" Command=obs/upload Component=ma-training-toolkit
Platform=ModelArts-Service

Ascend Logs

Ascend logs are generated when Ascend resources are used to for training. When Ascend resources are used for training, device logs, plog logs, proc logs for single-card training logs, MindSpore logs, and common logs are generated.

Common logs in the Ascend training scenario include the logs for **pip-requirement.txt**, **ma-pre-start**, **davincirun**, training process, and ModelArts.

The following is an example of the Ascend log structure:

```
obs://dgg-test-user/snt9-test-cases/log-out/                                     # Job log path
├── modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e
│   ├── ascend
│   │   ├── process_log
│   │   │   ├── rank_0
│   │   │   └── plog                                     # Plog logs
│   │   └── device-0 ...                                # Device logs
│   └── mindspore ...                                    # MindSpore logs
├── modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log # Common logs
└── modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-proc-rank-0-device-0.txt # proc log for
single-card training logs
```

Table 4-24 Ascend log description

Type	Description	Name
Device logs	User process AICPU and HCCP logs generated on the device and sent back to the host (training container). If any of the following situations occur, device logs cannot be obtained: • The compute node restarts unexpectedly. • The compute node stops expectedly. After the training process ends, the log is generated in the training container. The device logs for training using the preset MindSpore image are automatically uploaded to OBS. To automatically upload device logs for training using other preset images or custom images to OBS, specify ASCEND_PROCESS_LOG_PATH in the code. For details, see this sample code. <pre># set npu plog env ma_vj_name='echo \${MA_VJ_NAME} sed 's:ma-job:modelarts-job:g'` task_name="worker-\${ VC_TASK_INDEX}" task_plog_path=\${MA_LOG_DIR}/\${ ma_vj_name}/\${task_name} mkdir -p \${task_plog_path} export ASCEND_PROCESS_LOG_PATH=\${ task_plog_path}</pre>	~/ascend/log/device-{device-id}/device-{pid}_{timestamp}.log In the preceding command, pid indicates the user process ID on the host. Example: device-166_20220718191853764.log

Type	Description	Name
Plog logs	User process logs, for example, ACL/GE. Plog logs are generated in the training container. The plog logs for training using the preset MindSpore image are automatically uploaded to OBS. To automatically upload plog logs for training using custom images to OBS, specify ASCEND_PROCESS_LOG_PATH in the code. For details, see this sample code. <pre># set npu plog env ma_vj_name='echo \${MA_VJ_NAME} sed 's:ma-job:modelarts-job:g'` task_name="worker-\${ VC_TASK_INDEX}" task_plog_path=\${MA_LOG_DIR}/\${ ma_vj_name}/\${task_name} mkdir -p \${task_plog_path} export ASCEND_PROCESS_LOG_PATH=\${ task_plog_path}</pre>	~/ascend/log/plog/plog-{pid}_{timestamp}.log In the preceding command, pid indicates the user process ID on the host. Example: plog-166_20220718191843620.log
proc log	proc log is a redirection file of single-node training logs, helping you quickly obtain logs of a compute node. Training jobs using custom images do not involve proc log. proc log for training using a preset image is generated in the training container and automatically saved in OBS.	[modelarts-job-uuid]-proc-rank-[rank id]-device-[device logic id].txt • device id indicates the ID of the NPU used in the training job. The value is 0 for a single NPU and 0 to 7 for eight NPUs. For example, if the Ascend specification is 8*Snt9, the value of device id ranges from 0 to 7. If the Ascend specification is 1*Snt9, the value of device id is 0. • rank id indicates the global NPU ID of the training job. The value ranges from 0 to the number of instances multiplied by the number of NPUs minus 1. If a single instance is used, the value of rank id is the same as that of device id. Example: modelarts- job-95f661bd-1527-41b8-971c- eca55e513254-proc-rank-0- device-0.txt

Type	Description	Name
MindSpore logs	Separate MindSpore logs are generated in the MindSpore +Ascend training scenario. MindSpore logs are generated in the training container. The plog logs for training using the preset MindSpore image are automatically uploaded to OBS. To automatically upload plog logs for training using custom images to OBS, specify ASCEND_PROCESS_LOG_PATH in the code. For details, see this sample code. <pre># set npu plog env ma_vj_name=`echo \${MA_VJ_NAME}   sed 's:ma-job:modelarts-job:g` task_name="worker-\${ VC_TASK_INDEX}" task_plog_path=\${MA_LOG_DIR}/\${ ma_vj_name}/\${task_name} mkdir -p \${task_plog_path} export ASCEND_PROCESS_LOG_PATH=\${ task_plog_path}</pre>	For details about MindSpore logs, visit the MindSpore official website .

Type	Description	Name
Common training logs	Common training logs are generated in the /home/ma-user/modelarts/log directory of the training container and automatically uploaded to OBS. The common training logs include these types: • Logs for ma-pre-start (specific to Ascend training): If the ma-pre-start script is defined, the script execution log is generated. • Logs for davincirun (specific to Ascend training): log generated when the Ascend training process is started using the davincirun.py file • Training process logs: standard output of user training code • Logs for pip-requirement.txt: If pip-requirement.txt is defined in training code, pip package installation logs are generated. • ModelArts logs: used by O&M personnel to locate service faults.	Contained in the modelarts-job-[job id]-[task id].log file. task id indicates the instance ID. If a single node is used, the value is worker-0. If multiple nodes are used, the value is worker-0, worker-1, ..., or worker-{n-1}. n indicates the number of instances. Example: modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log

In the Ascend training scenario, after the training process exits, ModelArts uploads the log files in the training container to the OBS directory specified by **Job Log Path**. On the job details page, you can obtain the job log path and click the OBS address to go to the OBS console to check logs.

You can run the **ma-pre-start** script to modify the default environment variable configurations.

```
ASCEND_GLOBAL_LOG_LEVEL=3    # Log level, 0 for debug, 1 for info, 2 for warning, and 3 for error.
ASCEND_SLOG_PRINT_TO_STDOUT=1 # Whether to display plog logs. The value 1 indicates that plog logs
                              # are displayed by default.
ASCEND_GLOBAL_EVENT_ENABLE=1  # Event log level, 0 for disabling event logging and 1 for enabling
                              # event logging.
```

Place the **ma-pre-start.sh** or **ma-pre-start.py** script in the directory at the same level as the training boot file.

Before the training boot file is executed, the system executes the **ma-pre-start** script in **/home/work/user-job-dir/**. This method can be used to update the

Ascend RUN package installed in the container image or set some additional global environment variables required for training.

Viewing Training Job Logs

On the training job details page, you can preview logs, download logs, search for logs by keyword, and filter system logs in the log pane.

- Previewing logs

You can preview training logs on the system log pane. If multiple compute nodes are used, you can choose the target node from the drop-down list on the right.

If a log file is oversized, the system displays only the latest logs in the log pane. To view all logs, click the link in the upper part of the log pane, which will direct you to a new page. Then you will be redirected to a new page.

NOTE

- If the total size of all logs exceeds 500 MB, the log page may be frozen. In this case, download the logs to view them locally.
 - A log preview link can be accessed by anyone within one hour after it is generated. You can share the link with others.
 - **Ensure that no privacy information is contained in the logs. Otherwise, information leakage may occur.**
- Downloading logs
Training logs are retained for only 30 days. To permanently store logs, click the download icon in the upper right corner of the log pane. You can download the logs of multiple compute nodes in a batch. You can also enable **Persistent Log Saving** and set a log path when you create a training job. In this way, the logs will be automatically stored in the specified OBS path.
If a training job is created on Ascend compute nodes, certain system logs cannot be downloaded in the training log pane. To obtain these logs, go to the **Job Log Path** you set when you created the training job.
 - Searching for logs by keyword
In the upper right corner of the log pane, enter a keyword in the search box to search for logs.
The system will highlight the keyword and redirect you between search results. Only the logs loaded in the log pane can be searched for. If the logs are not fully displayed (see the message displayed on the page), obtain all the logs by downloading them or clicking the full log link and then search for the logs. On the page redirected by the full log link, press **Ctrl+F** to search for logs.
 - Filtering system logs
If **System logs** is selected, system logs and user logs are displayed. If **System logs** is deselected, only user logs are displayed.

4.10.6 Priority of a Training Job

When using a dedicated resource pool for training jobs, you can set the job priority when creating a training job or adjust the priority when a job is in the **Pending** state for a long time. By adjusting the job priority, you can reduce the job queuing duration.

Overview

Some training jobs, such as unimportant tests or experiments, are of low priority. In this case, you need to prioritize training tasks (jobs). A task with a higher priority is queued earlier than a task with a lower priority.

You can adjust the job execution sequence by configuring the priority of training jobs to ensure normal running of important services at peak hours.

Constraints

- You can set the priority of a training job only if it is created using a new-version dedicated resource pool.
- The value ranges from 1 to 3. The default priority is **1**, and the highest priority is **3**. You can set the job priority to **1** or **2** by default. Once permission to set the highest priority is granted, you can set it to **1**, **2**, or **3**.

Configuring the Priority

Set the priority when you create a training job. The value ranges from 1 to 3. The default priority is **1**, and the highest priority is **3**.

Changing the Priority

On the **Training Jobs** page, locate a training job in the **Pending** state and click  in the **Job Priority** column. In the dialog box that appears, change the priority and click **OK**.

Assigning the Permission to Set the Highest Job Priority to an IAM User

You can set the job priority to **1** or **2** by default. Once permission to set the highest priority is granted, you can set it to **1**, **2**, or **3**.

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions > Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and configure the following parameters.
 - **Policy Name:** Enter a custom policy name, for example, **Allowing Users to Set the Highest Job Priority**.
 - **Policy View:** Select **Visual editor**.
 - **Policy Content:** Select **Allow**, **ModelArts Service**, **modelarts:trainJob:setHighPriority**, and default resources.
3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in [2](#), and click **Next**. Then, select the scope and click **OK**.

After the configuration, all users in the user group have the permission to use Cloud Shell to log in to a running training container.

If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

4.10.7 Using Cloud Shell to Debug a Production Training Job

ModelArts Standard provides Cloud Shell, which allows you to log in to a running container to debug training jobs in the production environment.

Constraints

Only dedicated resource pools allow logging in to training containers using Cloud Shell. The training job must be running.

Preparation: Assigning the Cloud Shell Permission to an IAM User

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions > Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and configure the following parameters.
 - **Policy Name:** Enter a custom policy name, for example, **Using Cloud Shell to access a running job**.
 - **Policy View:** Select **Visual editor**.
 - **Policy Content:** Select **Allow**, **ModelArts Service**, **modelarts:trainJob:exec**, and default resources.
3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in [2](#), and click **Next**. Then, select the scope and click **OK**.

After the configuration, all users in the user group have the permission to use Cloud Shell to log in to a running training container.

If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

Using Cloud Shell

1. Configure parameters based on [Preparation: Assigning the Cloud Shell Permission to an IAM User](#).
2. Log in to the ModelArts console. In the navigation pane, choose **Model Training > Training Jobs**.
3. In the training job list, click the name of the target job to go to the training job details page.
4. On the training job details page, click the **Cloud Shell** tab and log in to the training container.

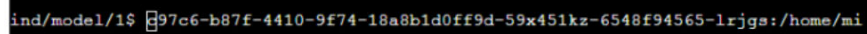
Verify that the login is successful, as shown in the following figure.

If the job is not running or the permission is insufficient, Cloud Shell cannot be used. In this case, locate the fault as prompted.

NOTE

If you encounter a path display issue when logging in to Cloud Shell, press **Enter** to resolve the problem.

Figure 4-22 Path display issue



Keeping a Training Job Running

You can only log in to Cloud Shell when the training job is in **Running** state. This section describes how to log in to a running training container through Cloud Shell.

Using the sleep Command

- For training jobs using a preset image

When creating a training job, set **Algorithm Type** to **Custom algorithm** and **Boot Mode** to **Preset image**, add **sleep.py** to the code directory, and use the script as the boot file. The training job keeps running for 60 minutes. You can access the container through Cloud Shell for debugging.

Example of **sleep.py**

```
import os
os.system('sleep 60m')
```

- For training jobs using a custom image

When creating a training job, set **Algorithm Type** to **Custom algorithm** and **Boot Mode** to **Custom image**, and enter **sleep 60m** in **Boot Command**. The training job keeps running for 60 minutes. You can access the container through Cloud Shell for debugging.

Keeping a Failed Job Running

When creating a training job, add **|| sleep 5h** at the end of the boot command and start the training job. For example:

```
cmd || sleep 5h
```

If the training fails, the **sleep** command is executed. In this case, you can log in to the container image through Cloud Shell for debugging.

NOTE

To debug a multi-node training job in Cloud Shell, you need to switch between worker-0 and worker-1 in Cloud Shell and run the boot command on each node. Otherwise, the task will wait for other nodes to join.

Preventing Cloud Shell Session from Disconnection

To run a job for a long time, you can use the **screen** command to run the job in a remote terminal that stays active even if you disconnect. This prevents the job from failing due to disconnection.

1. If **screen** is not installed in the image, run **apt-get install screen** to install it.
2. Create a screen terminal.
Use **-S** to create a screen terminal named **name**.
`screen -S name`
3. View the created screen terminals.
`screen -ls`
There are screens on:
2433.pts-3.linux (2013-10-20 16:48:59) (Detached)
2428.pts-3.linux (2013-10-20 16:48:05) (Detached)
2284.pts-3.linux (2013-10-20 16:14:55) (Detached)
2276.pts-3.linux (2013-10-20 16:13:18) (Detached)
4 Sockets in /var/run/screen/S-root.
4. Connect to the screen terminal whose **screen_id** is **2276**.
`screen -r 2276`
5. Press **Ctrl+A+D** to exit the screen terminal. After the exit, the screen session is still active and can be reconnected at any time.

For details about how to use screens, see [Screen User's Manual](#).

Analyzing the Call Stack of the Suspended Process Using the py-spy Tool

Use py-spy to analyze the call stack of a suspended process and identify the issue.

Step 1 On the ModelArts Standard console, choose .

Step 2 Click the target training job to go to its details page. On the page that appears, click the **Cloud Shell** tab and log in to the training container (the training job must be in the **Running** state).

Step 3 Install the py-spy tool.

```
# Use the utils.sh script to automatically configure the Python environment.
source /home/ma-user/modelarts/run/utils.sh

# Install py-spy.
pip install py-spy

# If the message "connection broken by 'ProxyError('Cannot connect to proxy.')" is displayed, disable the
proxy.
export no_proxy=$no_proxy,repo..com (Replace it with the pip source address of the corresponding region.)
pip install py-spy
```

Step 4 View the stack. For details about how to use the py-spy tool, see the [py-spy official document](#).

```
# Find the PID of the training process.
ps -ef

# Check the process stack of process 12345.
# For a training job using eight cards, run the following command to check the stacks of the eight
processes started by the main process in sequence.
py-spy dump --pid 12345
```

----End

4.10.8 Rebuilding, Stopping, or Deleting a Training Job

Saving As an Algorithm

To modify the algorithm of a training job, click **Save As Algorithm** in the upper right corner of the training job details page.

On the **Algorithms** page, the algorithm parameters for the last training job are automatically set. You can modify the settings.

 NOTE

A subscribed algorithm cannot be saved as a new algorithm.

Rebuilding a Training Job

If you are not satisfied with a created training job, click **Rebuild** in the **Operation** column to rebuild it. The page for creating a training job is displayed. On this page, the parameter settings for the previous training job are automatically retained. You only need to modify certain parameter settings.

Stopping a Training Job

In the training job list, click **Stop** in the **Operation** column of a training job that is in creating, pending, or running state to stop the job.

A training job in completed, failed, terminated, or abnormal state cannot be stopped.

Deleting a Training Job

Release resources of a training job when not in use.

 NOTE

Deleted training jobs cannot be restored.

- On the **Training Jobs** page, click **Delete** in the **Operation** column. In the displayed dialog box, click **OK** to delete the training job.
- Go to OBS and delete the OBS bucket and files used by the training job.

Searching for a Training Job

If you log in to ModelArts using an IAM account, all training jobs under this account are displayed in the training job list. To quickly search for a training job, use the following methods:

Method 1: Click **Only my jobs**. Then, only jobs created under the current IAM user account are displayed in the training job list.

Method 2: Search for jobs by name, ID, job type, status, creation time, algorithm, and resource pool.

Method 3: Click the refresh button in the upper right corner of the job list to refresh it.

Method 4: Configure the custom columns and other basic settings.

4.10.9 Managing Environment Variables of a Training Container

What Is an Environment Variable

This section describes environment variables preset in a training container. The environment variables include:

- Path environment variables
- Environment variables of a distributed training job
- Nvidia Collective multi-GPU Communication Library (NCCL) environment variables
- OBS environment variables
- Environment variables of the pip source
- Environment variables of the API Gateway address
- Environment variables of job metadata

Notes and Constraints

When defining custom environment variables, avoid using names that start with **MA_** to prevent conflicts with system environment variables.

Configuring Environment Variables

When you create a training job, you can add environment variables or modify environment variables preset in the training container.

NOTE

To ensure data security, do not enter sensitive information, such as plaintext passwords.

Environment Variables Preset in a Training Container

[Table 4-25](#), [Table 4-26](#), [Table 4-27](#), [Table 4-28](#), [Table 4-29](#), [Table 4-30](#), and [Table 4-31](#) list environment variables preset in a training container.

The environment variable values are examples only.

Table 4-25 Path environment variables

Variable	Description	Example
PATH	Executable file paths	<code>PATH=/usr/local/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin</code>

Variable	Description	Example
LD_LIBRARY_PATH	Dynamic load library paths	LD_LIBRARY_PATH=/usr/local/seccomponent/lib:/usr/local/cuda/lib64:/usr/local/cuda/compat:/root/miniconda3/lib:/usr/local/lib:/usr/local/nvidia/lib64
LIBRARY_PATH	Static library paths	LIBRARY_PATH=/usr/local/cuda/lib64/stubs
MA_HOME	Main directory of a training job	MA_HOME=/home/ma-user
MA_JOB_DIR	Parent directory of the training algorithm folder	MA_JOB_DIR=/home/ma-user/modelarts/user-job-dir
MA_MOUNT_PATH	Path mounted to a ModelArts training container, which is used to temporarily store training algorithms, algorithm input, algorithm output, and logs	MA_MOUNT_PATH=/home/ma-user/modelarts
MA_LOG_DIR	Training log directory	MA_LOG_DIR=/home/ma-user/modelarts/log
MA_SCRIPT_INTERPRETER	Training script interpreter	MA_SCRIPT_INTERPRETER=
WORKSPACE	Training algorithm directory	WORKSPACE=/home/ma-user/modelarts/user-job-dir/code

Table 4-26 Environment variables of a distributed training job

Variable	Description	Example
MA_CURRENT_IP	IP address of a job container.	MA_CURRENT_IP=192.168.23.38
MA_NUM_GPUS	Number of accelerator cards in a job container.	MA_NUM_GPUS=8

Variable	Description	Example
MA_TASK_NAME	Name of a job container, for example: • worker in MindSpore and PyTorch • learner or worker in reinforcement learning engines • ps or worker in TensorFlow	MA_TASK_NAME=worker
MA_NUM_HOSTS	Number of instances which is automatically obtained from Compute Nodes .	MA_NUM_HOSTS=4
VC_TASK_INDEX	Container index, starting from 0 . This parameter is invalid for single-node training. In multi-node training jobs, you can use this parameter to determine the algorithm logic of the container.	VC_TASK_INDEX=0
VC_WORKER_NUM	Instances required for a training job.	VC_WORKER_NUM=4
VC_WORKER_HOSTS	Domain name of each node for multi-node training. Use commas (,) to separate the domain names in sequence. You can obtain the IP address through domain name resolution.	VC_WORKER_HOSTS=modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-0.modelarts-job-a0978141-1712-4f9b-8a83-000000000000,modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1.ob-a0978141-1712-4f9b-8a83-000000000000,modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-2.modelarts-job-a0978141-1712-4f9b-8a83-000000000000,ob-a0978141-1712-4f9b-8a83-000000000000-worker-3.modelarts-job-a0978141-1712-4f9b-8a83-000000000000

Variable	Description	Example
<code>\$</code> <code>{MA_VJ_NAME}</code> <code>-\$</code> <code>{MA_TASK_NAME}-N.\$</code> <code>{MA_VJ_NAME}</code>	Communication domain name of a node. For example, the communication domain name of node 0 is <code>{MA_VJ_NAME}-\$ {MA_TASK_NAME}-0.\$ {MA_VJ_NAME}</code> . N indicates the number of instances.	For example, if there are four instances, the environment variables are as follows: <code>{MA_VJ_NAME}-\$ {MA_TASK_NAME}-0.\$ {MA_VJ_NAME}</code> <code>{MA_VJ_NAME}-\$ {MA_TASK_NAME}-1.\$ {MA_VJ_NAME}</code> <code>{MA_VJ_NAME}-\$ {MA_TASK_NAME}-2.\$ {MA_VJ_NAME}</code> <code>{MA_VJ_NAME}-\$ {MA_TASK_NAME}-3.\$ {MA_VJ_NAME}</code>

Table 4-27 NCCL environment variables

Variable	Description	Example
<code>NCCL_VERSION</code>	NCCL version	<code>NCCL_VERSION=2.7.8</code>
<code>NCCL_DEBUG</code>	NCCL log level	<code>NCCL_DEBUG=INFO</code>
<code>NCCL_IB_HCA</code>	InfiniBand NIC to use for communication	<code>NCCL_IB_HCA=^mlx5_bond_0</code>
<code>NCCL_SOCKET_IFNAME</code>	IP interface to use for communication	<code>NCCL_SOCKET_IFNAME=bond0, eth0</code>

Table 4-28 OBS environment variables

Variable	Description	Example
<code>S3_ENDPOINT</code>	OBS endpoint	N/A
<code>S3_VERIFY_SSL</code>	Whether to use SSL to access OBS	<code>S3_VERIFY_SSL=0</code>
<code>S3_USE_HTTPS</code>	Whether to use HTTPS to access OBS	<code>S3_USE_HTTPS=1</code>

Table 4-29 Environment variables of the pip source and API Gateway address

Variable	Description	Example
MA_PIP_HOST	Domain name of the pip source	MA_PIP_HOST=repo.example.com
MA_PIP_URL	Address of the pip source	MA_PIP_URL=http://repo.example.com/repository/pypi/simple/
MA_APIGW_ENDPOINT	ModelArts API Gateway address	MA_APIGW_ENDPOINT=https://modelarts.region..example.com

Table 4-30 Environment variables of job metadata

Variable	Description	Example
MA_CURRENT_INSTANCE_NAME	Name of the current node for multi-node training	MA_CURRENT_INSTANCE_NAME=modelarts-job-a0978141-1712-4f9b-8a83-0000000000-worker-1

Table 4-31 Precheck environment variables

Variable	Description	Example
MA_SKIP_IMAGE_DETECT	Whether to enable ModelArts precheck. The default value is 1 , which indicates that the pre-check is enabled; the value 0 indicates that the pre-check is disabled. It is good practice to enable precheck to detect node and driver faults before they affect services.	1

Table 4-32 Suspension detection environment variables

Variable	Description	Example
MA_HANG_DETECT_TIME	Suspension detection time. The job is considered suspended if its process I/O does not change for this time. Value range: 10 to 720 Unit: minute Default value: 30	30

How Do I View Training Environment Variables?

When creating a training job, set the boot command to **env** and retain default settings of other parameters.

After the training job is complete, view the **Logs** tab on the training job details page. The logs contain information about all environment variables.

Figure 4-23 Viewing logs

```

1 NV_LIBCUBLAS_DEV_VERSION=11.3.1.68-1
2 NV_CUDA_COMPAT_PACKAGE=cuda-compat-11-2
3 NV_CUDNN_PACKAGE_DEV=libcudnn8-dev=8.1.1.33-1+cuda11.2
4 LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64
5 NV_LIBNCCL_DEV_PACKAGE=libnccl-dev=2.8.4-1+cuda11.2
6 MA_ENGINE_VERSION=
7 MA_NUM_HOSTS=1
8 VC_WORKER_HOSTS=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0.modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
9 VK_TASK_INDEX=0
10 _=/usr/bin/env
11 MA_SCRIPT_INTERPRETER=
12 NV_LIBNPP_DEV_PACKAGE=libnpp-dev-11-2=11.2.1.68-1
13 MA_MAX_BACKOFF=0
14 HOSTNAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0
15 MA_IAM_USER_ID=79098163dd814fd986eca7ef0325d086
16 MA_CURRENT_IP=10.0.0.62
17 NV_LIBNPP_VERSION=11.2.1.68-1
18 NV_NVPROF_DEV_PACKAGE=cuda-nvprof-11-2=11.2.67-1
19 MA_MOUNT_PATH=/home/ma-user/modelarts
20 NVIDIA_VISIBLE_DEVICES=all
21 MA_ENGINE_TYPE=
22 NV_NVPROF_VERSION=11.2.67-1
23 NV_LIBCUSPARSE_VERSION=11.3.1.68-1
24 MODELARTS_SCC_SERVICE_PORT=60687
25 MA_HOME=/home/ma-user
26 KUBERNETES_PORT_443_TCP_PROTO=tcp
27 KUBERNETES_PORT_443_TCP_ADDR=10.247.0.1
28 NV_LIBCUBLAS_DEV_PACKAGE=libcublas-dev-11-2=11.3.1.68-1
29 MA_V1_NAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
30 MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/
31 NCCL_VERSION=2.8.4-1
32 KUBERNETES_PORT=tcp://10.247.0.1:443
33 PWD=/
34 NVARCH=x86_64
35 HOME=/home/ma-user

```

5 Image Management

5.1 Application Scenarios of Custom Images

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.

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

Concepts

Preset image: provided by ModelArts. You can select a ModelArts image when creating a notebook instance, training job, or model.

There are two types of preset images:

- **Unified image:** Use a unified image when you create a notebook instance, training job, or model. All images released later are unified images.
- **Unique image:** Use a unique image for a single module. For example, the preset image for training can only be used to create a training job. These images will be brought offline.

Custom image: created by user by following the ModelArts image creation specifications.

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

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 for training or creating models are obtained from the SWR service management list.

Figure 5-1 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 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 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](#).

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 models.

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 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.

Application Scenarios 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.

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.

- **Creating a Custom Image for a Notebook Instance**

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. You can also create a custom image based on existing images in notebook instances.

- **Creating a Custom Image for Model Training**

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.

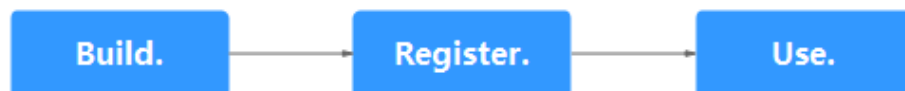
5.2 Creating a Custom Image for a Notebook Instance

5.2.1 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

Figure 5-2 Creating a custom image



Scenario: 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](#).

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 Huawei 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**.

- Log in to the ModelArts management console and choose **Image Management**. Then, click **Register**.
- 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.

5.2.2 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.

NOTE

If the image fails to be saved, view the event on the notebook instance details page.

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. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**. On the displayed page, configure the container engine size as needed. For details, see [Resizing a Dedicated Resource Pool](#).
- If the fault persists, contact technical support.

Prerequisites

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

Saving an Image

1. 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.
2. In the **Save Image** dialog box, configure parameters. Click **OK** to save the image.

Choose an organization from the **Organization** drop-down list. If no organization is available, click **Create** on the right to create one.
Users in an organization can share all images in the organization.
3. 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 5-3 Saving as a snapshot



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.

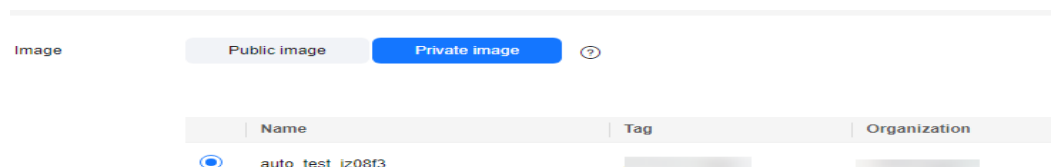
4. After the image is saved, the instance status changes to **Running**. View the image on the **Image Management** page.
5. Click the name of the image to view its details.

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.

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

Figure 5-4 Selecting a custom image to create a notebook instance



Method 2: On the **Image Management** 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.

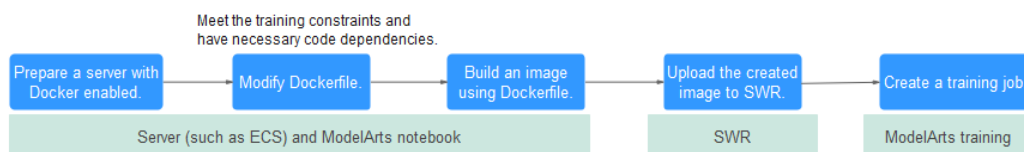
5.3 Creating a Custom Image for Model Training

5.3.1 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.

Procedure

Figure 5-5 Creating a custom image for a training job



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 has necessary code dependencies and meet ModelArts constraints. For details, see the following cases:

Constraints on Custom Images of the Training Framework

- Use Ubuntu 18.04 for custom images to in case versions are not compatible.
- Do not use a custom image larger than 15 GB. The size should not exceed half of the container engine space of the resource pool. Otherwise, the start time of the training job is affected.

The container engine space of ModelArts public resource pool is 50 GB. By default, the container engine space of the dedicated resource pool is also 50 GB. You can customize the container engine space when creating a dedicated resource pool.

- The **uid** of the default user of a custom image must be **1000**.
- The GPU or Ascend driver cannot be installed in a custom image. When you select GPU resources to run training jobs, ModelArts automatically places the GPU driver in the **/usr/local/nvidia** directory in the training environment. When you select Ascend resources to run training jobs, ModelArts automatically places the Ascend driver in the **/usr/local/Ascend/driver** directory.
- x86- or Arm-based custom images can run only with specifications corresponding to their architecture.

Run the following command to check the CPU architecture of a custom image:

```
docker inspect {Custom image path} | grep Architecture
```

The following is the command output for an Arm-based custom image:

```
"Architecture": "arm64"
```

- If the name of a specification contains **Arm**, this specification is an Arm-based CPU architecture.
- If the name of a specification does not contain **Arm**, this specification is an x86-based CPU architecture.

- 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).

5.3.2 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 box.

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 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.com/repository/pypi/simple
trusted-host = repo.com
timeout = 120
```

4. Create an 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. 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.

- Log in to the SWR console and select the target region.
- 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.
- Click **Generate Login Command** in the upper right corner to obtain a login command. Log in to ECS as user **root** and enter the login command.

Figure 5-6 Login command executed on ECS

```
root@bj-ununtu1804-cpu-0:~# docker login -u myusername --password-store=stdcli
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
```

- d. Log in to SWR and run the **docker tag** command to add tags to the image to be uploaded. In this case, **deep-learning** is used as an example. Replace it with the information configured in [a](#) for subsequent commands.

```
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.

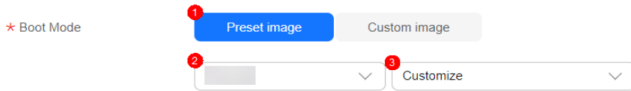
```
sudo docker push swr.Actual domain name.com/deep-learning/tf-1.13.2:latest
```

- 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.

7. Create a training job on ModelArts.

- Log in to the ModelArts console.
- In the navigation pane on the left, choose **Model Training** > **Training Jobs**.
- Click **Create Training Job**. On the displayed page, configure the parameters by referring to [Table 5-1](#). For details about the parameters, see [Creating a Production Training Job](#).

Table 5-1 Creating a training job

Parameter	Description
Algorithm Type	Mandatory. Select Custom algorithm .
Boot Mode	Mandatory. Select Preset image and choose the required framework and engine version. In this case, choose Customize for the engine version. 
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 1,000, and the folder depth cannot exceed 32. • The training code file is automatically downloaded to the <code>\${MA_JOB_DIR}/demo-code</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 <code>/test/code</code>, the training code file is downloaded to the <code>\${MA_JOB_DIR}/code</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.

5.3.3 Migrating Existing Images to ModelArts

Description

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.

 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 **cat /etc/group | grep 100** command 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.

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

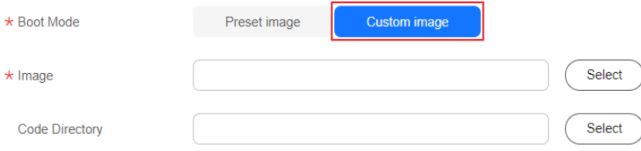
 NOTE

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 **cat /etc/passwd | grep 1000** command 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.

- 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.
 - b. In the navigation pane on the left, choose **Model Training > Training Jobs**.
 - c. Click **Create Training Job**. On the displayed page, configure the parameters by referring to [Table 5-1](#). For details about the parameters, see [Creating a Production Training Job](#).

Table 5-2 Creating a training job using a custom image

Parameter	Description
Algorithm Type	Mandatory. Select Custom algorithm .
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 1,000, and the folder depth cannot exceed 32. • The training code file is automatically downloaded to the <code>\${MA_JOB_DIR}/demo-code</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 <code>/test/code</code>, the training code file is downloaded to the <code>\${MA_JOB_DIR}/code</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: <code>python \${MA_JOB_DIR}/demo-code/train.py</code> • If the training boot script is an .sh file, main.sh for example, the boot command is as follows. <code>bash \${MA_JOB_DIR}/demo-code/main.sh</code> You can use semicolons (;) and ampersands (&&) to combine multiple commands. demo-code in the command is the last-level OBS directory where the code is stored. Replace it with the actual one.
Local Code Directory	Specify the local directory of a training container. When a training starts, the system automatically downloads the code directory to this directory. (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.

5.3.4 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.

```
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 by Huawei Mirrors is used, which is as follows:

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

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. 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 torch-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=$PATH:/usr/local/Ascend/nnae/latest/bin:/usr/local/Ascend/nnae/latest/compiler/cccec_compiler/
bin:/home/ma-user/miniconda3/bin \
    LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Ascend/driver/lib64:/usr/local/Ascend/driver/lib64/
    common:/usr/local/Ascend/driver/lib64/driver:/usr/local/Ascend/nnae/latest/lib64:/usr/local/Ascend/nnae/
    latest/lib64/plugin/opskernel:/usr/local/Ascend/nnae/latest/lib64/plugin/nnengine \
    PYTHONPATH=$PYTHONPATH:/usr/local/Ascend/nnae/latest/python/site-packages:/usr/local/Ascend/
    nnae/latest/opp/op_impl/built-in/ai_core/tbe \
    ASCEND_AICPU_PATH=$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 Verify 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 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. Run the **docker tag** command to add tags to 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. Run the following command to upload the image:
Replace the region, domain, as well as organization name **deep-learning** with the actual values.

```
sudo docker push swr:{region-id}.{domain}/deep-learning/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 management console, check whether access authorization has been configured for your account. For details, see [Configuring Agency Authorization for ModelArts with One Click](#). If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane on the left, choose **Model Training > Training Jobs**. Then, click **Create Training Job**.
3. On the displayed 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.

5.3.5 Creating a Custom Training Image (MindSpore + Ascend)

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

Description

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

- ubuntu-18.04
- CANN 6.3.RC2 (commercial edition)
- python-3.7.13
- mindspore-2.1.1

NOTE

- CANN 6.3.RC2 and MindSpore 2.1.1 are used in the following examples.
- These examples show how to create an Ascend container image and run the image in a dedicated resource pool with the required Ascend driver or firmware installed.

Procedure

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

1. [Step 1 Creating an OBS Bucket and Folder](#)
2. [Step 2 Preparing Script Files and Uploading Them to OBS](#)
3. [Step 3 Creating a Custom Image](#)
4. [Step 4 Uploading the Image to SWR](#)
5. [Step 5 Creating and Debugging a Notebook Instance on ModelArts](#)
6. [Step 6 Creating a Training Job on ModelArts](#)

Constraints

- This example requires the CANN commercial edition. If you do not have permission to download the CANN commercial edition, see other examples for creating a custom image.
- Pay attention to the version mapping between MindSpore and CANN, and between CANN and Ascend driver or firmware. Unmatched versions will lead to a training failure.

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. In this example, create a bucket named **test-modelarts** and folders listed in [Table 5-3](#).

Table 5-3 Required OBS folders

Folder	Description
obs://test-modelarts/ascend/demo-code/	Store the Ascend training script.
obs://test-modelarts/ascend/demo-code/run_ascend/	Store the startup scripts of the Ascend training script.
obs://test-modelarts/ascend/log/	Store training log files.

Step 2 Preparing Script Files and Uploading Them to OBS

1. Prepare the training script **mindspore-verification.py** and Ascend startup scripts (five in total) required in this example.
 - For details about the training script, see [Training the mindspore-verification.py File](#).
 - For details about the following Ascend startup scripts, see [Ascend Startup Scripts](#).
 - i. run_ascend.py
 - ii. common.py
 - iii. rank_table.py
 - iv. manager.py
 - v. fmk.py

NOTE

The **mindspore-verification.py** and **run_ascend.py** scripts are invoked by the **Boot Command** parameter during training job creation. For details, see [Boot Command](#).

The **common.py**, **rank_table.py**, **manager.py**, and **fmk.py** scripts are invoked when the **run_ascend.py** script is running.

2. Upload the training script **mindspore-verification.py** to **obs://test-modelarts/ascend/demo-code/** in the OBS bucket.
3. Upload the five Ascend startup scripts to the **obs://test-modelarts/ascend/demo-code/run_ascend/** folder in the OBS bucket.

Step 3 Creating a Custom Image

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

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

- ubuntu-18.04
- CANN 6.3.RC2 (commercial edition)
- python-3.7.13
- mindspore-2.1.1

 NOTE

Pay attention to the version mapping between MindSpore and CANN, and between CANN and Ascend driver or firmware. Unmatched versions will lead to a training failure.

These examples show how to create an Ascend container image and run the image in a dedicated resource pool with the required Ascend driver or firmware installed.

1. Obtain a Linux AArch64 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.

2. Install Docker.

The following uses Linux AArch64 as an example to describe how to obtain a Docker installation package. For more details, 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.

Start Docker.

```
systemctl start docker
```

3. 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.

4. Create a folder named **context**.

```
mkdir -p context
```

5. Obtain the **pip.conf** file.

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

6. Obtain the APT source file **Ubuntu-Ports-bionic.list**.

```
wget -O Ubuntu-Ports-bionic.list https://repo..com/repository/conf/Ubuntu-Ports-bionic.list
```

7. Download the **CANN 6.3.RC2-linux aarch64** and **mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl** installation files.

- Download the **Ascend-cann-nnae_6.3.RC2_linux-aarch64.run** file by referring to [CANN 6.3.RC2](#).
- Download the [mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl](#) file.

 NOTE

ModelArts supports only the commercial CANN edition.

8. Download the Miniconda3 installation file.

Download **Miniconda3-py37-4.10.3** (Python 3.7.10) at https://repo.anaconda.com/miniconda/Miniconda3-py37_4.10.3-Linux-aarch64.sh.

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

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

10. 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 arm64v8/ubuntu:18.04 AS builder

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

# Install OS dependencies obtained from Mirrors.
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 6.3.RC2
    gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
net-tools libblas-dev gfortran libblas3 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    # Grant the write permission of the parent directory of the CANN 6.3.RC2 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 provided by Mirrors.
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-py37_4.10.3-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-py37_4.10.3-Linux-aarch64.sh -b -p /home/ma-user/miniconda3

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

# Install the CANN 6.3.RC2 Python dependency package.
RUN pip install numpy~=1.14.3 decorator~=4.4.0 sympy~=1.4 cffi~=1.12.3 protobuf~=3.11.3 \
    attrs pyyaml pathlib2 scipy requests psutil absl-py

# Install CANN 6.3.RC2 in /usr/local/Ascend.
COPY --chown=ma-user:100 Ascend-cann-nnae_6.3.RC2_linux-aarch64.run /tmp
RUN chmod +x /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run && \
    /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run --install --install-path=/usr/local/Ascend

# Install MindSpore 2.1.1.
COPY --chown=ma-user:100 mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl /tmp
RUN chmod +x /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl && \
    pip install /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl

# Create the container image.
FROM arm64v8/ubuntu:18.04
```



```
# Install OS dependencies obtained from Mirrors.
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
  echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
  apt-get update && \
  apt-get install -y \
  # utils
  ca-certificates vim curl \
  # CANN 6.3.RC2
  gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
  net-tools libblas-dev gfortran libblas3 && \
  apt-get clean && \
  mv /etc/apt/sources.list.bak /etc/apt/sources.list

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

# 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/built-in/op_impl/ai_core/tbe \
  ASCEND_AICPU_PATH=/usr/local/Ascend/nnae/latest \
  ASCEND_OPP_PATH=/usr/local/Ascend/nnae/latest/opp \
  ASCEND_HOME_PATH=/usr/local/Ascend/nnae/latest \
  PYTHONUNBUFFERED=1

# 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](#).

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

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── Dockerfile
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

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

```
docker build . -t mindspore:2.1.1-cann6.3.RC2
```

The following log shows that the image has been created.

```
Successfully tagged mindspore:2.1.1-cann6.3.RC2
```

13. Upload the created image to SWR. For details, see [Step 4 Uploading the Image to SWR](#).

Step 4 Uploading the Image to SWR

Upload the created image to SWR so that it can be used to create training jobs on ModelArts.

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. Run the **docker tag** command to add tags to the uploaded image:
Replace the region, domain, as well as organization name **deep-learning** with the actual values.

```
sudo docker tag mindspore:2.1.1-cann6.3.RC2 swr.{region}-{domain}/deep-learning/  
mindspore:2.1.1-cann6.3.RC2
```
 - b. Run the following command to upload the image:
Replace the region, domain, as well as organization name **deep-learning** with the actual values.

```
sudo docker push swr.{region}-{domain}/deep-learning/mindspore:2.1.1-cann6.3.RC2
```
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 5 Creating and Debugging a Notebook Instance on ModelArts

1. Register the image uploaded to SWR with ModelArts Image Management.
Log in to the ModelArts management console. In the navigation pane on the left, choose **Image Management**. Click **Register** and register the image. The registered image can be used to create notebook instances.
2. Use the custom image to create a notebook instance and debug it. After the debugging is successful, save the image.
 - a. [Create a custom image using a custom image.](#)
 - b. [Save the image.](#)
3. After the image is debugged, [create a training job on ModelArts](#).

Step 6 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console. In the navigation pane on the left, choose > **Training Jobs**. The training job list is displayed by default.
2. On the **Create Training Job** page, configure parameters and click **Submit**.
 - **Algorithm Type:** Custom algorithm
 - **Boot Mode:** Custom image
 - **Image Path:**
 - **Code Directory:** OBS path to startup scripts, for example, **obs://test-modelarts/ascend/demo-code/**

- **Boot Command:** `python ${MA_JOB_DIR}/demo-code/run_ascend/run_ascend.py python ${MA_JOB_DIR}/demo-code/mindspore-verification.py`
 - **Resource Pool:** Dedicated resource pools
 - **Resource Type:** Ascend with the required driver and firmware version
 - **Job Log Path:** OBS path to stored training logs, for example, `obs://test-modelarts/ascend/log/`
3. Confirm the configurations of the training job and click **Submit**.
 4. 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, logs are displayed.

Figure 5-7 Runtime logs of a training job powered by Ascend resources in a dedicated resource pool

```
75 Ascend Envs
76 -----
77 JOB_ID: modelarts-job-2436291a-8543-4ab8-84ad-2dda8f1e4f5c
78 RANK_TABLE_FILE: /home/ma-user/modelarts/rank_table/jobstart_hcc1.json
79 RANK_SIZE: 1
80 ASCEND_DEVICE_ID: 0
81 DEVICE_ID: 0
82 RANK_ID: 0
83 -----
84 [2. 2. 2. 2.]
85 [2. 2. 2. 2.]
86
87 [[2. 2. 2. 2.]
88 [2. 2. 2. 2.]
89 [2. 2. 2. 2.]]
90
91 [[2. 2. 2. 2.]
92 [2. 2. 2. 2.]
93 [2. 2. 2. 2.]]]
```

Training the mindspore-verification.py File

The `mindspore-verification.py` file contains the following information:

```
import os
import numpy as np
from mindspore import Tensor
import mindspore.ops as ops
import mindspore.context as context

print('Ascend Envs')
print('-----')
print('JOB_ID: ', os.environ['JOB_ID'])
print('RANK_TABLE_FILE: ', os.environ['RANK_TABLE_FILE'])
print('RANK_SIZE: ', os.environ['RANK_SIZE'])
print('ASCEND_DEVICE_ID: ', os.environ['ASCEND_DEVICE_ID'])
print('DEVICE_ID: ', os.environ['DEVICE_ID'])
print('RANK_ID: ', os.environ['RANK_ID'])
```

```
print('-----')

context.set_context(device_target="Ascend")
x = Tensor(np.ones([1,3,3,4]).astype(np.float32))
y = Tensor(np.ones([1,3,3,4]).astype(np.float32))

print(ops.add(x, y))
```

Ascend Startup Scripts

- **run_ascend.py**

```
import sys
import os

from common import RunAscendLog
from common import RankTableEnv

from rank_table import RankTable, RankTableTemplate1, RankTableTemplate2

from manager import FMKManager

if __name__ == '__main__':
    log = RunAscendLog.setup_run_ascend_logger()

    if len(sys.argv) <= 1:
        log.error('there are not enough args')
        sys.exit(1)

    train_command = sys.argv[1:]
    log.info('training command')
    log.info(train_command)

    if os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1) is not None:
        # new format rank table file
        rank_table_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1)
        RankTable.wait_for_available(rank_table_path)
        rank_table = RankTableTemplate1(rank_table_path)
    else:
        # old format rank table file
        rank_table_path_origin = RankTableEnv.get_rank_table_template2_file_path()
        RankTable.wait_for_available(rank_table_path_origin)
        rank_table = RankTableTemplate2(rank_table_path_origin)

    if rank_table.get_device_num() >= 1:
        log.info('set rank table %s env to %s' % (RankTableEnv.RANK_TABLE_FILE,
rank_table.get_rank_table_path()))
        RankTableEnv.set_rank_table_env(rank_table.get_rank_table_path())
    else:
        log.info('device num < 1, unset rank table %s env' % RankTableEnv.RANK_TABLE_FILE)
        RankTableEnv.unset_rank_table_env()

    instance = rank_table.get_current_instance()
    server = rank_table.get_server(instance.server_id)
    current_instance = RankTable.convert_server_to_instance(server)

    fmk_manager = FMKManager(current_instance)
    fmk_manager.run(rank_table.get_device_num(), train_command)
    return_code = fmk_manager.monitor()

    fmk_manager.destroy()

    sys.exit(return_code)
```

- **common.py**

```
import logging
import os

logo = 'Training'
```

```
# Rank Table Constants
class RankTableEnv:
    RANK_TABLE_FILE = 'RANK_TABLE_FILE'

    RANK_TABLE_FILE_V1 = 'RANK_TABLE_FILE_V1_0'

    HCCL_CONNECT_TIMEOUT = 'HCCL_CONNECT_TIMEOUT'

    # jobstart_hccl.json is provided by the volcano controller of Cloud-Container-Engine(CCE)
    HCCL_JSON_FILE_NAME = 'jobstart_hccl.json'

    RANK_TABLE_FILE_DEFAULT_VALUE = '/user/config/%s' % HCCL_JSON_FILE_NAME

    @staticmethod
    def get_rank_table_template1_file_dir():
        parent_dir = os.environ[ModelArts.MA_MOUNT_PATH_ENV]
        return os.path.join(parent_dir, 'rank_table')

    @staticmethod
    def get_rank_table_template2_file_path():
        rank_table_file_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE)
        if rank_table_file_path is None:
            return RankTableEnv.RANK_TABLE_FILE_DEFAULT_VALUE

        return os.path.join(os.path.normpath(rank_table_file_path),
RankTableEnv.HCCL_JSON_FILE_NAME)

    @staticmethod
    def set_rank_table_env(path):
        os.environ[RankTableEnv.RANK_TABLE_FILE] = path

    @staticmethod
    def unset_rank_table_env():
        del os.environ[RankTableEnv.RANK_TABLE_FILE]

class ModelArts:
    MA_MOUNT_PATH_ENV = 'MA_MOUNT_PATH'
    MA_CURRENT_INSTANCE_NAME_ENV = 'MA_CURRENT_INSTANCE_NAME'
    MA_VJ_NAME = 'MA_VJ_NAME'

    MA_CURRENT_HOST_IP = 'MA_CURRENT_HOST_IP'

    CACHE_DIR = '/cache'

    TMP_LOG_DIR = '/tmp/log/'

    FMK_WORKSPACE = 'workspace'

    @staticmethod
    def get_current_instance_name():
        return os.environ[ModelArts.MA_CURRENT_INSTANCE_NAME_ENV]

    @staticmethod
    def get_current_host_ip():
        return os.environ.get(ModelArts.MA_CURRENT_HOST_IP)

    @staticmethod
    def get_job_id():
        ma_vj_name = os.environ[ModelArts.MA_VJ_NAME]
        return ma_vj_name.replace('ma-job', 'modelarts-job', 1)

    @staticmethod
    def get_parent_working_dir():
        if ModelArts.MA_MOUNT_PATH_ENV in os.environ:
            return os.path.join(os.environ.get(ModelArts.MA_MOUNT_PATH_ENV),
ModelArts.FMK_WORKSPACE)
```

```
        return ModelArts.CACHE_DIR

class RunAscendLog:

    @staticmethod
    def setup_run_ascend_logger():
        name = logo
        formatter = logging.Formatter(fmt='[run ascend] %(asctime)s - %(levelname)s - %(message)s')

        handler = logging.StreamHandler()
        handler.setFormatter(formatter)

        logger = logging.getLogger(name)
        logger.setLevel(logging.INFO)
        logger.addHandler(handler)
        logger.propagate = False
        return logger

    @staticmethod
    def get_run_ascend_logger():
        return logging.getLogger(logo)
```

- **rank_table.py**

```
import json
import time
import os

from common import ModelArts
from common import RunAscendLog
from common import RankTableEnv

log = RunAscendLog.get_run_ascend_logger()

class Device:
    def __init__(self, device_id, device_ip, rank_id):
        self.device_id = device_id
        self.device_ip = device_ip
        self.rank_id = rank_id

class Instance:
    def __init__(self, pod_name, server_id, devices):
        self.pod_name = pod_name
        self.server_id = server_id
        self.devices = self.parse_devices(devices)

    @staticmethod
    def parse_devices(devices):
        if devices is None:
            return []
        device_object_list = []
        for device in devices:
            device_object_list.append(Device(device['device_id'], device['device_ip'], ""))

        return device_object_list

    def set_devices(self, devices):
        self.devices = devices

class Group:
    def __init__(self, group_name, device_count, instance_count, instance_list):
        self.group_name = group_name
        self.device_count = int(device_count)
        self.instance_count = int(instance_count)
        self.instance_list = self.parse_instance_list(instance_list)

    @staticmethod
```

```
def parse_instance_list(instance_list):
    instance_object_list = []
    for instance in instance_list:
        instance_object_list.append(
            Instance(instance['pod_name'], instance['server_id'], instance['devices']))

    return instance_object_list

class RankTable:
    STATUS_FIELD = 'status'
    COMPLETED_STATUS = 'completed'

    def __init__(self):
        self.rank_table_path = ""
        self.rank_table = {}

    @staticmethod
    def read_from_file(file_path):
        with open(file_path) as json_file:
            return json.load(json_file)

    @staticmethod
    def wait_for_available(rank_table_file, period=1):
        log.info('Wait for Rank table file at %s ready' % rank_table_file)
        complete_flag = False
        while not complete_flag:
            with open(rank_table_file) as json_file:
                data = json.load(json_file)
                if data[RankTable.STATUS_FIELD] == RankTable.COMPLETED_STATUS:
                    log.info('Rank table file is ready for read')
                    log.info('\n' + json.dumps(data, indent=4))
                    return True

            time.sleep(period)

        return False

    @staticmethod
    def convert_server_to_instance(server):
        device_list = []
        for device in server['device']:
            device_list.append(
                Device(device_id=device['device_id'], device_ip=device['device_ip'],
rank_id=device['rank_id']))

        ins = Instance(pod_name="", server_id=server['server_id'], devices=[])
        ins.set_devices(device_list)
        return ins

    def get_rank_table_path(self):
        return self.rank_table_path

    def get_server(self, server_id):
        for server in self.rank_table['server_list']:
            if server['server_id'] == server_id:
                log.info('Current server')
                log.info('\n' + json.dumps(server, indent=4))
                return server

        log.error('server [%s] is not found' % server_id)
        return None

class RankTableTemplate2(RankTable):

    def __init__(self, rank_table_template2_path):
        super().__init__()
```

```
json_data = self.read_from_file(file_path=rank_table_template2_path)

self.status = json_data[RankTableTemplate2.STATUS_FIELD]
if self.status != RankTableTemplate2.COMPLETED_STATUS:
    return

# sorted instance list by the index of instance
# assert there is only one group
json_data["group_list"][0]["instance_list"] = sorted(json_data["group_list"][0]["instance_list"],
                                                    key=RankTableTemplate2.get_index)

self.group_count = int(json_data['group_count'])
self.group_list = self.parse_group_list(json_data['group_list'])

self.rank_table_path, self.rank_table = self.convert_template2_to_template1_format_file()

@staticmethod
def parse_group_list(group_list):
    group_object_list = []
    for group in group_list:
        group_object_list.append(
            Group(group['group_name'], group['device_count'], group['instance_count'],
group['instance_list']))

    return group_object_list

@staticmethod
def get_index(instance):
    # pod_name example: job94dc1dbf-job-bj4-yolov4-15
    pod_name = instance["pod_name"]
    return int(pod_name[pod_name.rfind("-") + 1:])

def get_current_instance(self):
    """
    get instance by pod name
    specially, return the first instance when the pod name is None
    :return:
    """
    pod_name = ModelArts.get_current_instance_name()
    if pod_name is None:
        if len(self.group_list) > 0:
            if len(self.group_list[0].instance_list) > 0:
                return self.group_list[0].instance_list[0]

    return None

    for group in self.group_list:
        for instance in group.instance_list:
            if instance.pod_name == pod_name:
                return instance
    return None

def convert_template2_to_template1_format_file(self):
    rank_table_template1_file = {
        'status': 'completed',
        'version': '1.0',
        'server_count': '0',
        'server_list': []
    }

    logic_index = 0
    server_map = {}
    # collect all devices in all groups
    for group in self.group_list:
        if group.device_count == 0:
            continue
        for instance in group.instance_list:
            if instance.server_id not in server_map:
                server_map[instance.server_id] = []
```



```
        for device in instance.devices:
            template1_device = {
                'device_id': device.device_id,
                'device_ip': device.device_ip,
                'rank_id': str(logic_index)
            }
            logic_index += 1
            server_map[instance.server_id].append(template1_device)

    server_count = 0
    for server_id in server_map:
        rank_table_template1_file['server_list'].append({
            'server_id': server_id,
            'device': server_map[server_id]
        })
    server_count += 1

    rank_table_template1_file['server_count'] = str(server_count)

    log.info('Rank table file (Template1)')
    log.info('\n' + json.dumps(rank_table_template1_file, indent=4))

    if not os.path.exists(RankTableEnv.get_rank_table_template1_file_dir()):
        os.makedirs(RankTableEnv.get_rank_table_template1_file_dir())

    path = os.path.join(RankTableEnv.get_rank_table_template1_file_dir(),
RankTableEnv.HCCL_JSON_FILE_NAME)
    with open(path, 'w') as f:
        f.write(json.dumps(rank_table_template1_file))
    log.info('Rank table file (Template1) is generated at %s', path)

    return path, rank_table_template1_file

def get_device_num(self):
    total_device_num = 0
    for group in self.group_list:
        total_device_num += group.device_count
    return total_device_num

class RankTableTemplate1(RankTable):
    def __init__(self, rank_table_template1_path):
        super().__init__()
        self.rank_table_path = rank_table_template1_path
        self.rank_table = self.read_from_file(file_path=rank_table_template1_path)

    def get_current_instance(self):
        current_server = None
        server_list = self.rank_table['server_list']
        if len(server_list) == 1:
            current_server = server_list[0]
        elif len(server_list) > 1:
            host_ip = ModelArts.get_current_host_ip()
            if host_ip is not None:
                for server in server_list:
                    if server['server_id'] == host_ip:
                        current_server = server
                        break
            else:
                current_server = server_list[0]

        if current_server is None:
            log.error('server is not found')
            return None
        return self.convert_server_to_instance(current_server)

    def get_device_num(self):
        server_list = self.rank_table['server_list']
```

```
device_num = 0
for server in server_list:
    device_num += len(server['device'])
return device_num
```

- **manager.py**

```
import time
import os
import os.path
import signal

from common import RunAscendLog
from fmk import FMK

log = RunAscendLog.get_run_ascend_logger()

class FMKManager:
    # max destroy time: ~20 (15 + 5)
    # ~ 15 (1 + 2 + 4 + 8)
    MAX_TEST_PROC_CNT = 4

    def __init__(self, instance):
        self.instance = instance
        self.fmk = []
        self.fmk_processes = []
        self.get_sigterm = False
        self.max_test_proc_cnt = FMKManager.MAX_TEST_PROC_CNT

    # break the monitor and destroy processes when get terminate signal
    def term_handle(func):
        def receive_term(signum, stack):
            log.info('Received terminate signal %d, try to destroyed all processes' % signum)
            stack.f_locals['self'].get_sigterm = True

        def handle_func(self, *args, **kwargs):
            origin_handle = signal.getsignal(signal.SIGTERM)
            signal.signal(signal.SIGTERM, receive_term)
            res = func(self, *args, **kwargs)
            signal.signal(signal.SIGTERM, origin_handle)
            return res

        return handle_func

    def run(self, rank_size, command):
        for index, device in enumerate(self.instance.devices):
            fmk_instance = FMK(index, device)
            self.fmk.append(fmk_instance)

            self.fmk_processes.append(fmk_instance.run(rank_size, command))

    @term_handle
    def monitor(self, period=1):
        # busy waiting for all fmk processes exit by zero
        # or there is one process exit by non-zero

        fmk_cnt = len(self.fmk_processes)
        zero_ret_cnt = 0
        while zero_ret_cnt != fmk_cnt:
            zero_ret_cnt = 0
            for index in range(fmk_cnt):
                fmk = self.fmk[index]
                fmk_process = self.fmk_processes[index]
                if fmk_process.poll() is not None:
                    if fmk_process.returncode != 0:
                        log.error('proc-rank-%s-device-%s (pid: %d) has exited with non-zero code: %d'
                                % (fmk.rank_id, fmk.device_id, fmk_process.pid, fmk_process.returncode))
            return fmk_process.returncode
```

```
        zero_ret_cnt += 1
    if self.get_sigterm:
        break
    time.sleep(period)

    return 0

def destroy(self, base_period=1):
    log.info('Begin destroy training processes')
    self.send_sigterm_to_fm_process()
    self.wait_fm_process_end(base_period)
    log.info('End destroy training processes')

def send_sigterm_to_fm_process(self):
    # send SIGTERM to fm processes (and process group)
    for r_index in range(len(self.fm_processes) - 1, -1, -1):
        fm = self.fm[r_index]
        fm_process = self.fm_processes[r_index]
        if fm_process.poll() is not None:
            log.info('proc-rank-%s-device-%s (pid: %d) has exited before receiving the term signal',
                    fm.rank_id, fm.device_id, fm_process.pid)
            del self.fm_processes[r_index]
            del self.fm[r_index]

    try:
        os.killpg(fm_process.pid, signal.SIGTERM)
    except ProcessLookupError:
        pass

def wait_fm_process_end(self, base_period):
    test_cnt = 0
    period = base_period
    while len(self.fm_processes) > 0 and test_cnt < self.max_test_proc_cnt:
        for r_index in range(len(self.fm_processes) - 1, -1, -1):
            fm = self.fm[r_index]
            fm_process = self.fm_processes[r_index]
            if fm_process.poll() is not None:
                log.info('proc-rank-%s-device-%s (pid: %d) has exited',
                        fm.rank_id, fm.device_id, fm_process.pid)
                del self.fm_processes[r_index]
                del self.fm[r_index]
        if not self.fm_processes:
            break

    time.sleep(period)
    period *= 2
    test_cnt += 1

    if len(self.fm_processes) > 0:
        for r_index in range(len(self.fm_processes) - 1, -1, -1):
            fm = self.fm[r_index]
            fm_process = self.fm_processes[r_index]
            if fm_process.poll() is None:
                log.warn('proc-rank-%s-device-%s (pid: %d) has not exited within the max waiting time,
                        'send kill signal',
                        fm.rank_id, fm.device_id, fm_process.pid)
                os.killpg(fm_process.pid, signal.SIGKILL)
```

- **fm.py**

```
import os
import subprocess
import pathlib
from contextlib import contextmanager

from common import RunAscendLog
from common import RankTableEnv
from common import ModelArts

log = RunAscendLog.get_run_ascend_logger()
```

```
class FMK:

    def __init__(self, index, device):
        self.job_id = ModelArts.get_job_id()
        self.rank_id = device.rank_id
        self.device_id = str(index)

    def gen_env_for_fmk(self, rank_size):
        current_envs = os.environ.copy()

        current_envs['JOB_ID'] = self.job_id

        current_envs['ASCEND_DEVICE_ID'] = self.device_id
        current_envs['DEVICE_ID'] = self.device_id

        current_envs['RANK_ID'] = self.rank_id
        current_envs['RANK_SIZE'] = str(rank_size)

        FMK.set_env_if_not_exist(current_envs, RankTableEnv.HCCL_CONNECT_TIMEOUT, str(1800))

        log_dir = FMK.get_log_dir()
        process_log_path = os.path.join(log_dir, self.job_id, 'ascend', 'process_log', 'rank_' + self.rank_id)
        FMK.set_env_if_not_exist(current_envs, 'ASCEND_PROCESS_LOG_PATH', process_log_path)
        pathlib.Path(current_envs['ASCEND_PROCESS_LOG_PATH']).mkdir(parents=True, exist_ok=True)

        return current_envs

    @contextmanager
    def switch_directory(self, directory):
        owd = os.getcwd()
        try:
            os.chdir(directory)
            yield directory
        finally:
            os.chdir(owd)

    def get_working_dir(self):
        fmk_workspace_prefix = ModelArts.get_parent_working_dir()
        return os.path.join(os.path.normpath(fmk_workspace_prefix), 'device%s' % self.device_id)

    @staticmethod
    def get_log_dir():
        parent_path = os.getenv(ModelArts.MA_MOUNT_PATH_ENV)
        if parent_path:
            log_path = os.path.join(parent_path, 'log')
            if os.path.exists(log_path):
                return log_path

        return ModelArts.TMP_LOG_DIR

    @staticmethod
    def set_env_if_not_exist(envs, env_name, env_value):
        if env_name in os.environ:
            log.info('env already exists. env_name: %s, env_value: %s ' % (env_name, env_value))
            return

        envs[env_name] = env_value

    def run(self, rank_size, command):
        envs = self.gen_env_for_fmk(rank_size)
        log.info('bootstrap proc-rank-%s-device-%s' % (self.rank_id, self.device_id))

        log_dir = FMK.get_log_dir()
        if not os.path.exists(log_dir):
            os.makedirs(log_dir)

        log_file = '%s-proc-rank-%s-device-%s.txt' % (self.job_id, self.rank_id, self.device_id)
```

```
log_file_path = os.path.join(log_dir, log_file)

working_dir = self.get_working_dir()
if not os.path.exists(working_dir):
    os.makedirs(working_dir)

with self.switch_directory(working_dir):
    # os.setsid: change the process(forked) group id to itself
    training_proc = subprocess.Popen(command, env=envs, preexec_fn=os.setsid,
                                     stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

    log.info('proc-rank-%s-device-%s (pid: %d)', self.rank_id, self.device_id, training_proc.pid)

    # https://docs.python.org/3/library/subprocess.html#subprocess.Popen.wait
    subprocess.Popen(['tee', log_file_path], stdin=training_proc.stdout)

return training_proc
```

5.4 Creating a Custom Image for Inference(To go offline)

5.4.1 Creating a Custom Image for a Model(To go offline)

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.

Constraints

- No malicious code is allowed.
- 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 5-8 Example of obtaining `listen_ip`

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

- 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 5-4 Status code

Status Code	Message	Description
200	OK	Request sent.

- **Log file output**
To ensure that logs can be properly displayed, the logs must be standard output.
- **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**

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 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
```

5.4.2 Creating a Custom Image on ECS(To go offline)

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 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..com@g" /etc/apt/sources.list && \
sed -i "s@http://.*archive.ubuntu.com@http://repo..com@g" /etc/apt/sources.list && \
apt-get update && \
apt-get install -y python3 python3-pip && \
pip3 install --trusted-host https://repo..com -i https://repo..com/repository/pypi/simple Flask

# Copy the application code to the image.
COPY test_app.py /opt/test_app.py

# Specify the boot command of the image.
CMD python3 /opt/test_app.py
```

– **test_app.py**

```
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----- in hello func -----")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----- in goodbye func -----")
    return '\nGoodbye!\n'

@app.route('/', methods=['POST'])
def default_func():
    print("----- in default func -----")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. Switch to the **self-define-images** folder and run the following command to create custom image **test:v1**:

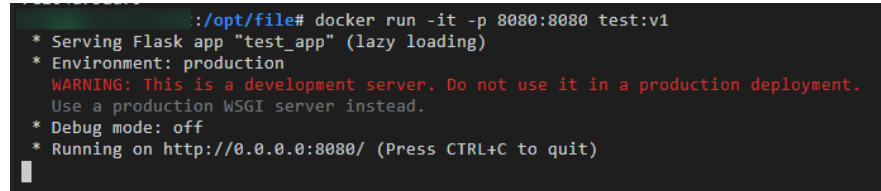
```
docker build -t test:v1 .
```
5. Run **docker images** to view the custom image you have created.

Verifying the Image Locally and Uploading It to SWR

1. Run the following command in the local environment to start the custom image:

```
docker run -it -p 8080:8080 test:v1
```

Figure 5-9 Starting a custom image



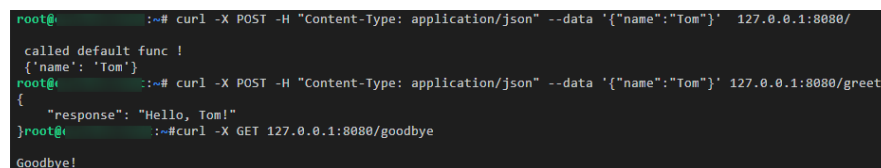
```
:/opt/file# docker run -it -p 8080:8080 test:v1
* Serving Flask app "test_app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press 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 5-10 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

When you import a meta model by referring to [Importing a Meta Model from a Container Image](#), pay attention to the following parameters:

- **Meta Model Source:** Select **Container image**.
 - **Container Image Path:** Select the 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 a model as a real-time service.](#)
2. View the details about the real-time service.
3. Access the real-time service in the **Prediction** tab.

6 Resource Monitoring

6.1 Overview

ModelArts Standard provides multiple methods for you to view metrics.

- Method 1: View the metrics on the ModelArts Standard console.
You can view the metrics on the ModelArts **Overview** page or the monitoring tab of each module.
 - You can view metrics on the ModelArts **Overview** page. For details, see [Viewing Monitoring Metrics on the ModelArts Console](#).
 - Standard training jobs: When a training job is running, you can view the usage of resources such as CPUs, GPUs, and NPUs. For details, see [Viewing the Resource Usage of a Training Job](#).
- Method 2: View all metrics on AOM.
All metrics reported by ModelArts Standard are stored in AOM, which enables you to consume metrics. You can set metric threshold alarms and report alarms on the AOM console. For details, see [Viewing All ModelArts Monitoring Metrics on the AOM Console](#).
- Method 3: View all metrics on Grafana.
If AOM's monitoring templates cannot meet your requirements, you can use Grafana to view and analyze metrics. Grafana provides different views and templates for monitoring, which allow you to see the real-time resource usage on dashboards clearly.
After adding the data source in Grafana, you can view all ModelArts Standard monitoring metrics stored in AOM using Grafana. For details, see [Using Grafana to View AOM Monitoring Metrics](#).
To view AOM monitoring metrics using Grafana plugins, follow these steps:
 - a. Install and configure Grafana.
You can install and configure Grafana using any of the following ways: [Installing and Configuring Grafana on Windows](#), [Installing and Configuring Grafana on Linux](#), and [Installing and Configuring Grafana on a Notebook Instance](#).
 - b. [Configuring a Grafana Data Source](#)

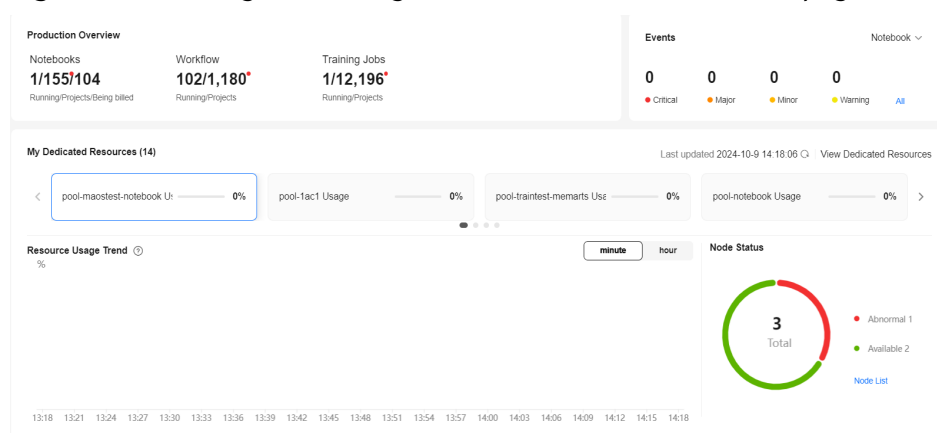
c. [Configuring a Dashboard to View Metric Data](#)

6.2 Viewing Monitoring Metrics on the ModelArts Console

Viewing ModelArts Monitoring Metrics on the Overview Page

On the ModelArts overview page, you can view various information, including production overview, resource usage, and resource usage of training jobs. Click the links to view more details.

Figure 6-1 Viewing monitoring information on the **Overview** page



NOTE

If there is a discrepancy between the total number of events displayed at the top and bottom of the **Overview** page, refresh the page.

Viewing ModelArts Monitoring Metrics of Each Module

- Training jobs: When a training job is running, you can view the usage of resources such as CPUs, GPUs, and NPUs. For details, see [Viewing the Resource Usage of a Training Job](#).

6.3 Viewing All ModelArts Monitoring Metrics on the AOM Console

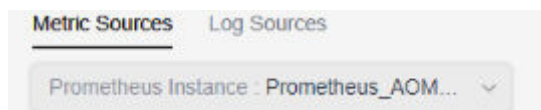
ModelArts periodically collects the usage of key metrics (such as GPUs, NPUs, CPUs, and memory) of each node in a resource pool as well as the usage of key metrics of development environments, training jobs, and inference services, and reports the data to AOM. You can view the information on AOM.

Viewing Monitoring Metrics on the AOM Console

1. Log in to the console and search for **AOM** to go to the AOM console.

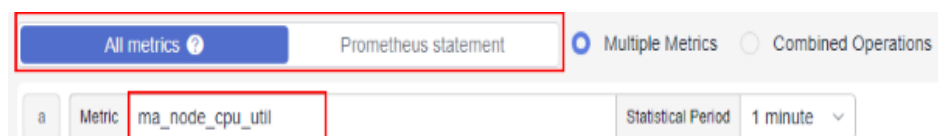
2. In the navigation pane on the left, choose **Metric Browsing**.
3. Select the **Prometheus_AOM_Default** instance from the drop-down list.

Figure 6-2 Specifying the metric source



4. Select one or more metrics from **All metrics** or **Prometheus statement**.

Figure 6-3 Adding a metric



For details about how to view metrics, see in the Help Center.

Container-level Metrics

Table 6-1 Container metrics

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
CPU	CPU Usage	ma_container_cpu_util	CPU usage of a measured object	%	0%–100%	Raw data > 95% for two consecutive periods	Suggestion	Check whether the service resource usage meets the expectation. If the service is normal, no action is required.
	Used CPU Cores	ma_container_cpu_used_core	Number of CPU cores used by a measured object	Cores	≥ 0	N/A	N/A	N/A
	Total CPU Cores	ma_container_cpu_limit_core	Total number of CPU cores that have been applied for a measured object	Cores	≥ 1	N/A	N/A	N/A

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
	CPU Memory Usage	ma_container_gpu_mem_util	Percentage of the used GPU memory to the total GPU memory	%	0%–100%	Raw data > 95% for two consecutive periods	Suggestion	Check whether the service resource usage meets the expectation. If the service is normal, no action is required.
Memory	Total Physical Memory	ma_container_memory_capacity_megabytes	Total physical memory that has been requested for a measured object	MB	≥ 0	N/A	N/A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	Physical Memory Usage	ma_cont ainer_me memory_util	Percentage of the used physical memory to the total physical memory	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	Used Physical Memory	ma_cont ainer_me mory_us ed_mega bytes	Physical memory that has been used by a measured object (container_memory_working_set_bytes in the current working set) (Memory usage in a working set = Active anonymous page and cache, and file-baked page ≤ container_memory_usage_bytes)	MB	≥ 0	N/A	N/A	N/A
Stor age	Disk Read Rate	ma_cont ainer_dis k_read_k ilobytes	Volume of data read from a disk per second	KB/s	≥ 0	N/A	N/A	N/A
	Disk Write Rate	ma_cont ainer_dis k_write_ kilobytes	Volume of data written into a disk per second	KB/s	≥ 0	N/A	N/A	N/A
GPU me mor y	Total GPU Memory	ma_cont ainer_gp u_mem_t otal_me gabytes	Total GPU memory of a training job	MB	> 0	N/A	N/A	N/A
	GPU Memory Usage	ma_cont ainer_gp u_mem_ util	Percentage of the used GPU memory to the total GPU memory	%	0%–100 %	N/A	N/A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	Used GPU Memory	ma_cont ainer_gp u_mem_ used_me gabytes	GPU memory used by a measured object	MB	≥ 0	N/A	N/A	N/A
	Idle GPU Memory	ma_cont ainer_gp u_mem_f ree_meg abytes	Idle GPU memory of a measured object	MB	≥ 0	N/A	N/A	N/A
GPU	GPU usage	ma_cont ainer_gp u_util	GPU usage of a measured object	%	0%– 100 %	Raw data > 95% for two consecutive periods	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	GPU Memory Bandwid th Usage	ma_cont ainer_gp u_mem_ copy_util	GPU memory bandwidth usage of a measured object For example, the maximum memory bandwidth of GP Vnt1 is 900 GB/s. If the current memory bandwidth is 450 GB/s, the memory bandwidth usage is 50%.	%	0%–100 %	N/A	N/A	N/A
	GPU Encoder Usage	ma_cont ainer_gp u_enc_util	GPU encoder usage of a measured object	%	%	N/A	N/A	N/A
	GPU Decoder Usage	ma_cont ainer_gp u_dec_util	GPU decoder usage of a measured object	%	%	N/A	N/A	N/A
	GPU Tempera ture	DCGM_F I_DEV_G PU_TEMP	GPU temperature	°C	Nat ural num ber	N/A	N/A	N/A
	GPU Power	DCGM_F I_DEV_P OWER_U SAGE	GPU power	Watt (W)	> 0	N/A	N/A	N/A
	GPU Memory Tempera ture	DCGM_F I_DEV_M EMORY_ TEMP	GPU memory temperature	°C	Nat ural num ber	N/A	N/A	N/A

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
Network I/O	Downlink Rate	ma_container_network_receive_bytes	Inbound traffic rate of a measured object	Bytes/s	≥ 0	N/A	N/A	N/A
	Packet RX Rate	ma_container_network_receive_packets	Number of data packets received by an NIC per second	Packets/s	≥ 0	N/A	N/A	N/A
	Downlink Error Rate	ma_container_network_receive_error_packets	Number of error packets received by a NIC per second	Packets/s	≥ 0	Raw data > 1 for two consecutive periods	Critical	Packet loss on the network. Submit a service ticket and contact the O&M support to locate the fault.
	Uplink Rate	ma_container_network_transmit_bytes	Outbound traffic rate of a measured object	Bytes/s	≥ 0	N/A	N/A	N/A

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
	Uplink Error Rate	ma_container_network_transmit_error_packets	Number of error packets sent by a NIC per second	Packets/s	≥ 0	Raw data > 1 for two consecutive periods	Critical	Packet loss on the network. Submit a service ticket and contact the O&M support to locate the fault.
	Packet TX Rate	ma_container_network_transmit_packets	Number of data packets sent by a NIC per second	Packets/s	≥ 0	N/A	N/A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
NPU	NPU Usage	ma_cont ainer_np u_util	NPU usage of a measured object (To be replaced by ma_container _npu_ai_core _util)	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	NPU Memory Usage	ma_cont ainer_np u_memo ry_util	Percentage of the used NPU memory to the total NPU memory (To be replaced by ma_container_npu_ddr_memory_util for Snt3 series, and ma_container_npu_hbm_util for Snt9 series)	%	0%–100 %	Raw data > 98% for two consecutive periods	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.
	Used NPU Memory	ma_cont ainer_np u_memo ry_used_ megabyt es	NPU memory used by a measured object (To be replaced by ma_container_npu_ddr_memory_usage_bytes for Snt3 series, and ma_container_npu_hbm_usage_bytes for Snt9 series)	≥ 0	MB	N/A	N/ A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	Total NPU Memory	ma_cont ainer_np u_memo ry_total_ megabyt es	Total NPU memory of a measured object (To be replaced by ma_container _npu_ddr_me mory_bytes for Snt3 series, and ma_container _npu_hbm_by tes for Snt9 series)	> 0	MB	N/A	N/ A	N/A
	AI Processo r Error Codes	ma_cont ainer_np u_ai_core_ error_co de	Error codes of Ascend AI processors	-	-	Raw data > 0 for three conse cutiv e perio ds	Cri tic al	Abno rmal card. Sub mit a servi ce ticket and cont act the O&M supp ort.

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
	AI Processor Health Status	ma_container_npu_ai_core_health_status	Health status of Ascend AI processors	-	1 : healthy 0 : unhealthy	Raw data > 0 for two consecutive periods	Critical	Abnormal card. Submit a service ticket and contact the O&M support.
	AI Processor Power Consumption	ma_container_npu_ai_core_power_usage_watts	Power consumption of Ascend AI processors	Watt (W)	> 0	N/A	N/A	N/A
	AI Processor Temperature	ma_container_npu_ai_core_temperature_celsius	Temperature of Ascend AI processors	°C	Natural number	N/A	N/A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI Core Usage	ma_cont ainer_np u_ai_core _util	AI core usage of Ascend AI processors	%	0%–100 %	Raw data > 95% for two consecutive periods	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.
	Overall NPU Usage	ma_cont ainer_np u_genera l_util	NPU usage of Ascend AI processors (supported by driver version 24.1.RC2 and later)	%	0%–100 %	N/A	N/ A	N/A
	AI Core Clock Frequency	ma_cont ainer_np u_ai_core _frequen cy_hertz	AI core clock frequency of Ascend AI processors	Hertz (Hz)	> 0	N/A	N/ A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI Processo r Voltage	ma_cont ainer_np u_ai_core _voltage _volts	Voltage of Ascend AI processors	Volt (V)	Nat ural num ber	N/A	N/ A	N/A
	AI Processo r DDR Memory	ma_cont ainer_np u_ddr_m emory_b ytes	Total DDR memory capacity of Ascend AI processors	Byte	> 0	N/A	N/ A	N/A
	AI Processo r DDR Usage	ma_cont ainer_np u_ddr_m emory_u sage_byt es	DDR memory usage of Ascend AI processors	Byte	> 0	N/A	N/ A	N/A
	AI Processo r DDR Memory Utilizatio n	ma_cont ainer_np u_ddr_m emory_u til	DDR memory utilization of Ascend AI processors Invalid metric for Snt9C.	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI Processo r HBM Memory	ma_cont ainer_np u_hbm_b ytes	Total HBM memory of Ascend AI processors (dedicated for Snt9 processors)	Byte	> 0	N/A	N/ A	N/A
	AI Processo r HBM Memory Usage	ma_cont ainer_np u_hbm_u sage_byt es	HBM memory usage of Ascend AI processors (dedicated for Snt9 processors)	Byte	> 0	N/A	N/ A	N/A
	AI Processo r HBM Memory Utilizatio n	ma_cont ainer_np u_hbm_u til	HBM memory utilization of Ascend AI processors (dedicated for Snt9 processors)	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI Processo r HBM Memory Bandwid th Utilizatio n	ma_cont ainer_np u_hbm_b andwidt h_util	HBM memory bandwidth utilization of Ascend AI processors (dedicated for Snt9 processors)	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.
	AI Processo r HBM Memory Clock Frequenc y	ma_cont ainer_np u_hbm_f requency _hertz	HBM memory clock frequency of Ascend AI processors (dedicated for Snt9 processors)	Hertz (Hz)	> 0	N/A	N/ A	N/A
	AI Processo r HBM Memory Tempera ture	ma_cont ainer_np u_hbm_t emperat ure_celsi us	HBM memory temperature of Ascend AI processors (dedicated for Snt9 processors)	°C	Nat ural num ber	N/A	N/ A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI CPU Utilization	ma_cont ainer_np u_ai_cpu _util	AI CPU utilization of Ascend AI processors	%	0%–100 %	N/A	N/A	N/A
	AI Processor Control CPU Utilization	ma_cont ainer_np u_ctrl_cp u_util	Control CPU utilization of Ascend AI processors	%	0%–100 %	N/A	N/A	N/A
	AI Processor Control CPU Frequency	ma_node _npu_ctrl _cpu_fre quency_ hertz	Control CPU frequency of Ascend AI processors	Hertz (Hz)	> 0 Syst em mod e (ded icat ed reso urce pool user mod e)	N/A	N/A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
	AI Vector Core Usage	ma_cont ainer_np u_vector _core_uti l	AI vector core usage of Ascend AI processors	%	0%– 100 %	Raw data > 95% for two conse cutiv e perio ds	Su gg est ion	Chec k whet her the servi ce resou rce usag e meet s the expe ctati on. If the servi ce is norm al, no actio n is requi red.
NPU RoC E net wor k	NPU RoCE Network Uplink Rate	ma_cont ainer_np u_roce_t x_rate_b ytes_per_ second	Uplink rate of the NPU network module used by the container	Bytes /s	≥ 0	N/A	N/ A	N/A
	NPU RoCE Network Downlin k Rate	ma_cont ainer_np u_roce_r x_rate_b ytes_per_ second	Downlink rate of the NPU network module used by the container	Bytes /s	≥ 0	N/A	N/ A	N/A

Cate gory	Name	Metric	Description	Unit	Val ue Ran ge	Alar m Thres hold	Al ar m Se ve rit y	Solut ion
Not ebo ok servi ce metr ics	Noteboo k Cache Directory Size	ma_cont ainer_no tebook_c ache_dir _size_byt es	A high-speed local disk is attached to the /cache directory for GPU and NPU notebook instances. This metric indicates the total size of the directory.	Bytes	≥ 0	N/A	N/ A	N/A
	Noteboo k Cache Directory Utilizatio n	ma_cont ainer_no tebook_c ache_dir _util	A high-speed local disk is attached to the /cache directory for GPU and NPU notebook instances. This metric indicates the utilization of the directory.	%	0%– 100 %	Raw data > 90% for two conse cutiv e perio ds	Ma jor	If the disk usag e is too high, the note book insta nce will be resta rted.

Node-level Metrics

Table 6-2 Node metrics (collected only in dedicated resource pools)

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
CPU	Total CPU Cores	ma_node_cpu_limit_core	Total number of CPU cores that have been requested for a measured object	Cores	≥ 1	N/A	N/A	N/A
	Used CPU Cores	ma_node_cpu_used_core	Number of CPU cores used by a measured object	Cores	≥ 0	N/A	N/A	N/A
	CPU Usage	ma_node_cpu_util	CPU usage of a measured object	%	0% – 100 %	Raw data > 95% for two consecutive periods	Major	Check whether the service resource usage meets the expectation . If the service is normal, no action is required.
	CPU I/O Wait Time	ma_node_cpu_io_wait_counter	Disk I/O wait time accumulated since system startup	jiffies	≥ 0	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
Me mor y	Physica l Mem ory Usage	ma_nod e_memo ry_util	Percentage of the used physical memory to the total physical memory	%	0% – 100 %	Raw data > 95% for two consecu tive periods	M ajo r	Check wheth er the servic e resour ce usage meets the expectation . If the servic e is norm al, no action is requir ed.
	Total Physica l Mem ory	ma_nod e_memo ry_total_ megabyt es	Total physical memory that has been applied for a measured object	MB	≥ 0	N/A	N/ A	N/A
Net wor k I/O	Downli nk Rate (BPS)	ma_nod e_netwo rk_receiv e_rate_b ytes_sec onds	Inbound traffic rate of a measured object	Bytes /s	≥ 0	N/A	N/ A	N/A
	Uplink Rate (BPS)	ma_nod e_netwo rk_trans mit_rate_ bytes_s econds	Outbound traffic rate of a measured object	Bytes /s	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
Stor age	Disk Read Rate	ma_nod e_disk_r ead_rate _kilobyte s_second s	Volume of data read from a disk per second (Only data disks used by containers are collected.)	KB/s	≥ 0	N/A	N/ A	N/A
	Disk Write Rate	ma_nod e_disk_w rite_rate _kilobyte s_second s	Volume of data written into a disk per second (Only data disks used by containers are collected.)	KB/s	≥ 0	N/A	N/ A	N/A
	Total Cache	ma_nod e_cache_ space_ca pacity_m egabytes	Total cache of the Kubernetes space	MB	≥ 0	N/A	N/ A	N/A
	Used Cache	ma_nod e_cache_ space_us ed_capa city_meg abytes	Used cache of the Kubernetes space	MB	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	Cache Usage	ma_node_cache_space_used_percent	Cache usage of the Kubernetes space	%	≥ 0	Raw data > 90% for two consecutive periods	Critical	Check the disk in a timely manner to avoid affecting services. Clear invalid data on compute nodes.
	Total Container Space	ma_node_container_space_capacity_megabytes	Total container space	MB	≥ 0	N/A	N/A	N/A
	Used Container Space	ma_node_container_space_used_capacity_megabytes	Used container space	MB	≥ 0	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	Contai ner Space Usage	ma_nod e_contai ner_spac e_used_ percent	Space usage of a container	%	≥ 0	Raw data > 90% for two consecu tive periods	Cri tic al	Check the disk in a timely mann er to avoid affect ing servic es. Clear invali d data on comp ute nodes .
	Disk Inform ation	ma_nod e_disk_in fo	Basic disk information	-	≥ 0	N/A	N/ A	N/A
	Total Reads	ma_nod e_disk_r eads_co mpleted _total	Total number of successful reads	-	≥ 0	N/A	N/ A	N/A
	Merge d Reads	ma_nod e_disk_r eads_me rged_tot al	Number of merged reads	-	≥ 0	N/A	N/ A	N/A
	Bytes Read	ma_nod e_disk_r ead_byte s_total	Total number of bytes that are successfully read	Bytes	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	Read Time Spent	ma_nod e_disk_r ead_tim e_secon ds_total	Time spent on all reads	Secon ds	≥ 0	N/A	N/ A	N/A
	Total Writes	ma_nod e_disk_w rites_co mpleted _total	Total number of successful writes	-	≥ 0	N/A	N/ A	N/A
	Merge d Writes	ma_nod e_disk_w rites_me rged_tot al	Number of merged writes	-	≥ 0	N/A	N/ A	N/A
	Writte n Bytes	ma_nod e_disk_w ritten_by tes_total	Total number of bytes that are successfully written	Bytes	≥ 0	N/A	N/ A	N/A
	Write Time Spent	ma_nod e_disk_w rite_time _seconds _total	Time spent on all write operations	Secon ds	≥ 0	N/A	N/ A	N/A
	Ongoin g I/Os	ma_nod e_disk_io _now	Number of ongoing I/Os	-	≥ 0	N/A	N/ A	N/A
	I/O Executi on Durati on	ma_nod e_disk_io _time_se conds_to tal	Time spent on executing I/Os	Secon ds	≥ 0	N/A	N/ A	N/A
	I/O Executi on Weight ed Time	ma_nod e_disk_io _time_w eighted_ seconds_ tota	Weighted time spent on executing I/Os	Secon ds	≥ 0	N/A	N/ A	N/A

Category	Name	Metric	Description	Unit	Value Range	Alarm Threshold	Alarm Severity	Solution
GPU	GPU Usage	ma_node_gpu_util	GPU usage of a measured object	%	0% – 100%	N/A	N/A	N/A
	Total GPU Memory	ma_node_gpu_mem_total_megabytes	Total GPU memory of a measured object	MB	> 0	N/A	N/A	N/A
	GPU Memory Usage	ma_node_gpu_mem_util	Percentage of the used GPU memory to the total GPU memory	%	0% – 100%	Raw data > 97% for two consecutive periods	Suggestion	Check whether the service resource usage meets the expectation. If the service is normal, no action is required.
	Used GPU Memory	ma_node_gpu_mem_used_megabytes	GPU memory used by a measured object	MB	≥ 0	N/A	N/A	N/A
	Idle GPU Memory	ma_node_gpu_mem_free_megabytes	Idle GPU memory of a measured object	MB	> 0	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	Tasks on a Shared GPU	node_gpu_share_job_count	Number of tasks running on a shared GPU	Num ber	≥ 0	N/A	N/A	N/A
	GPU Temperature	DCGM_FI_DEV_GPU_TEMP	GPU temperature	°C	Nat ural nu mb er	N/A	N/A	N/A
	GPU Power	DCGM_FI_DEV_POWER_USAGE	GPU power	Watt (W)	> 0	N/A	N/A	N/A
	GPU Memory Temperature	DCGM_FI_DEV_MEMORY_TEMP	GPU memory temperature	°C	Nat ural nu mb er	N/A	N/A	N/A
NPU	NPU Usage	ma_node_npu_util	NPU usage of a measured object (To be replaced by ma_node_npu_ai_core_util)	%	0% – 100 %	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NPU Memor y Usage	ma_nod e_npu_ memory _util	Percentage of the used NPU memory to the total NPU memory (To be replaced by ma_node_np u_ddr_memo ry_util for Snt3 series, and ma_node_np u_hbm_util for Snt9 series)	%	0% – 100 %	Raw data > 97% for two consecu tive periods	Su gg est ion	Check wheth er the servic e resour ce usage meets the expectation . If the servic e is norm al, no action is requir ed.
	Used NPU Memor y	ma_nod e_npu_ memory _used_m egabytes	NPU memory used by a measured object (To be replaced by ma_node_np u_ddr_memo ry_usage_byt es for Snt3 series, and ma_node_np u_hbm_usag e_bytes for Snt9 series)	≥ 0	MB	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	Total NPU Memory	ma_node_npu_memory_total_megabytes	Total NPU memory of a measured object (To be replaced by ma_node_npu_ddr_memory_bytes for Snt3 series, and ma_node_npu_hbm_bytes for Snt9 series)	> 0	MB	N/A	N/A	N/A
	AI Processor Error Codes	ma_node_npu_ai_core_error_code	Error codes of Ascend AI processors	-	-	N/A	N/A	N/A
	AI Processor Health Status	ma_node_npu_ai_core_health_status	Health status of Ascend AI processors	-	• 1 : healthy • 0 : unhealthy	The value is 0 for two consecutive periods.	Critical	Submit a service ticket.

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	AI Process or Power Consumption	ma_node_npu_ai_core_power_usage_watts	Power consumption of Ascend AI processors	Watt (W)	> 0	N/A	N/A	N/A
	AI Process or Temperature	ma_node_npu_ai_core_temperature_celsius	Temperature of Ascend AI processors	°C	Natural number	N/A	N/A	N/A
	AI Core Usage	ma_node_npu_ai_core_util	AI core usage of Ascend AI processors	%	0% – 100%	N/A	N/A	N/A
	Overall NPU Usage	ma_node_npu_general_util	NPU usage of Ascend AI processors (supported by driver version 24.1.RC2 and later)	%	0% – 100%	N/A	N/A	N/A
	AI Core Clock Frequency	ma_node_npu_ai_core_frequency_hertz	AI core clock frequency of Ascend AI processors	Hertz (Hz)	> 0	N/A	N/A	N/A
	AI Process or Voltage	ma_node_npu_ai_core_voltage_volts	Voltage of Ascend AI processors	Volt (V)	Natural number	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	AI Process or DDR Memory	ma_node_npu_ddr_memory_bytes	Total DDR memory capacity of Ascend AI processors Invalid metric for Snt9C.	Byte	> 0	N/A	N/A	N/A
	AI Process or DDR Usage	ma_node_npu_ddr_memory_usage_bytes	DDR memory usage of Ascend AI processors	Byte	> 0	N/A	N/A	N/A
	AI Process or DDR Memory Utilization	ma_node_npu_ddr_memory_util	DDR memory utilization of Ascend AI processors	%	0% – 100%	Raw data > 90% for two consecutive periods	Suggestion	Check whether the service resource usage meets the expectation. If the service is normal, no action is required.
	AI Process or HBM Memory	ma_node_npu_hbm_bytes	Total HBM memory of Ascend AI processors (dedicated for Snt9 processors)	Byte	> 0	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	AI Process or HBM Memor y Usage	ma_nod e_npu_h bm_usag e_bytes	HBM memory usage of Ascend AI processors (dedicated for Snt9 processors)	Byte	> 0	N/A	N/ A	N/A
	AI Process or HBM Memor y Utilizat ion	ma_nod e_npu_h bm_util	HBM memory utilization of Ascend AI processors (dedicated for Snt9 processors)	%	0% – 100 %	Raw data > 97% for two consecu tive periods	Su gg est ion	Check wheth er the servic e resour ce usage meets the expecta tion . If the servic e is norma l, no action is requir ed.
	AI Process or HBM Memor y Bandwi dth Utilizat ion	ma_nod e_npu_h bm_ban dwidth_ util	HBM memory bandwidth utilization of Ascend AI processors (dedicated for Snt9 processors)	%	0% – 100 %	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	AI Process or HBM Memory Clock Freque ncy	ma_nod e_npu_h bm_freq uency_h ertz	HBM memory clock frequency of Ascend AI processors (dedicated for Snt9 processors)	Hertz (Hz)	> 0	N/A	N/ A	N/A
	AI Process or HBM Memory Tempe rature	ma_nod e_npu_h bm_tem perature _celsius	HBM memory temperature of Ascend AI processors (dedicated for Snt9 processors)	°C	Nat ural nu mb er	N/A	N/ A	N/A
	AI CPU Utilizat ion	ma_nod e_npu_ai _cpu_util	AI CPU utilization of Ascend AI processors	%	0% – 100 %	N/A	N/ A	N/A
	AI Process or Control CPU Utilizat ion	ma_nod e_npu_ct rl_cpu_u til	Control CPU utilization of Ascend AI processors	%	0% – 100 %	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	AI Process or Control CPU Freque ncy	ma_nod e_npu_ct rl_cpu_fr equency _hertz	Control CPU frequency of Ascend AI processors	Hertz (Hz)	> 0 Syst em mo de (av aila ble for ded icat ed reso urc e poo l use rs)	N/A	N/ A	N/A
	HBM ECC Detecti on Switch	ma_nod e_npu_h bm_ecc_ enable	0 indicates that ECC detection is disabled. 1 indicates that ECC detection is enabled.	-	• 1 : e n a b l e d • 0 : d i s a b l e d	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	Current HBM Single-bit Errors	ma_node_npu_hbm_single_bit_error_total	Current number of HBM single-bit errors	Number	≥ 0	N/A	N/A	N/A
	Current HBM Multi-bit Errors	ma_node_npu_hbm_double_bit_error_total	Current number of HBM multi-bit errors	Number	≥ 0	N/A	N/A	N/A
	Total Single-bit Errors in the HBM Life Cycle	ma_node_npu_hbm_total_single_bit_error_total	Total number of single-bit errors in the HBM life cycle	Number	≥ 0	N/A	N/A	N/A
	Total Multi-bit Errors in the HBM Life Cycle	ma_node_npu_hbm_total_double_bit_error_total	Total number of multi-bit errors in the HBM life cycle	Number	≥ 0	N/A	N/A	N/A
	Isolated NPU Memory Pages with HBM Single-bit Errors	ma_node_npu_hbm_single_bit_isolated_pages_total	Number of isolated NPU memory pages with HBM single-bit errors	Number	≥ 0	N/A	N/A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	Isolate d NPU Memor y Pages with HBM Multi- bit Errors	ma_nod e_npu_h bm_dou ble_bit_i solated_ pages_to tal	Number of isolated NPU memory pages with HBM multi- bit errors Note: If there are more than 64 pages, change the NPU.	Num ber	≥ 0	Raw data ≥ 64 for two consecu tive periods	Cri tical	If there are more than 64 pages , submi t a servic e ticket, and switc h the NPU server .
	AI Vector Core Usage	ma_nod e_npu_v ector_co re_util	AI vector core usage of Ascend AI processors	%	0% – 100 %	N/A	N/ A	N/A
NPU RoCE net wor k	NPU RoCE Netwo rk Uplink Rate	ma_nod e_npu_r oce_tx_r ate_byte s_per_se cond	NPU RoCE network uplink rate	Bytes /s	≥ 0	N/A	N/ A	N/A
	NPU RoCE Netwo rk Downli nk Rate	ma_nod e_npu_r oce_rx_r ate_byte s_per_se cond	NPU RoCE network downlink rate	Bytes /s	≥ 0	N/A	N/ A	N/A
	MAC Uplink Pause Frames	ma_nod e_npu_r oce_mac _tx_paus e_packet s_total	Total number of pause frame packets sent by NPU RoCE network MAC	Num ber	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	MAC Downlink Pause Frames	ma_node_npu_roce_mac_rx_pause_packets_total	Total number of pause frame packets received by NPU RoCE network MAC	Number	≥ 0	N/A	N/A	N/A
	MAC Uplink PFC Frames	ma_node_npu_roce_mac_tx_pfc_packets_total	Total number of PFC frame packets sent by NPU RoCE network MAC	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_tx\_pause\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	MAC Downlink PFC Frames	ma_node_npu_roce_mac_rx_pfc_packets_total	Total number of PFC frame packets received by NPU RoCE network MAC	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_rx\_pause\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	MAC Uplink Bad Packets	ma_node_npu_roce_mac_tx_bad_packets_total	Total number of bad packets sent by NPU RoCE network MAC	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_tx\_pfc\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	MAC Downlink Bad Packets	ma_node_npu_roce_mac_rx_bad_packets_total	Total number of bad packets received by NPU RoCE network MAC	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_rx\_pfc\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	RoCE Uplink Bad Packets	ma_node_npu_roce_tx_err_packets_total	Total number of bad packets sent by NPU RoCE	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_tx\_bad\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	RoCE Downlink Bad Packets	ma_node_npu_roce_rx_err_packets_total	Total number of bad packets received by NPU RoCE	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_mac\_rx\_bad\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	RoCE Uplink Packets	ma_node_npu_roce_tx_all_packets_total	Total number of packets sent by NPU RoCE	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_tx\_err\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.
	RoCE Downlink Packets	ma_node_npu_roce_rx_all_packets_total	Total number of packets received by NPU RoCE	Number	≥ 0	$\text{delta}(\text{ma\_node\_npu\_roce\_rx\_err\_packets\_total}[1\text{m}]) > 0$	Major	Submit a service ticket.

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
NPU opti cal mod ule (Thi s met ric is avai lable for Snt9 B/C air- cool ed net wor king .)	Optical Modul e Tempe rature	ma_nod e_npu_o ptical_te mperatu re	Optical module temperature	°C	≥ 0	N/A	N/ A	N/A
	Optical Modul e Power Voltag e	ma_nod e_npu_o ptical_vc c	Power voltage of the optical module	Milliv olt (mV)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Transm it Power 0	ma_nod e_npu_o ptical_tx _power0	Transmit power 0 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Transm it Power 1	ma_nod e_npu_o ptical_tx _power1	Transmit power 1 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Transm it Power 2	ma_nod e_npu_o ptical_tx _power2	Transmit power 2 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Transm it Power 3	ma_nod e_npu_o ptical_tx _power3	Transmit power 3 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	Optical Modul e Receiv e Power 0	ma_nod e_npu_o ptical_rx _power0	Receive power 0 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Receiv e Power 1	ma_nod e_npu_o ptical_rx _power1	Receive power 1 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Receiv e Power 2	ma_nod e_npu_o ptical_rx _power2	Receive power 2 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
	Optical Modul e Receiv e Power 3	ma_nod e_npu_o ptical_rx _power3	Receive power 3 of the optical module	Milli watt (mW)	≥ 0	N/A	N/ A	N/A
Infin iBan d or RoC E net wor k	Total Amoun t of Data Receiv ed by a NIC	ma_nod e_infinib and_port _receive d_data_ bytes_to tal	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLS from the port.	(coun ting in doubl e words , 32 bits	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	Total Amoun t of Data Sent by a NIC	ma_nod e_infinib and_port _transmi tted_dat a_bytes_ total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLS from the port.	(coun ting in doubl e words , 32 bits	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
NFS mou ntin g stat us	NFS Getattr Conges tion Time	ma_nod e_mount stats_get attr_bac klog_wai t	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	NFS Getattr Round Trip Time	ma_nod e_mount stats_get attr_rtt	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply ³⁴ . RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al arm Se ve rit y	Soluti on
	NFS Access Conges tion Time	ma_nod e_mount stats_acc ess_back log_wait	Access is an NFS operation that checks the access permissions of a file or directory for a given user. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Access Round Trip Time	ma_nod e_mount stats_acc ess_rtt	Access is an NFS operation that checks the access permissions of a file or directory for a given user. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply ³⁴ . RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Lookup Conges tion Time	ma_nod e_mount stats_loo kup_bac klog_wai t	Lookup is an NFS operation that resolves a file name in a directory to a file handle. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Lookup Round Trip Time	ma_nod e_mount stats_loo kup_rtt	Lookup is an NFS operation that resolves a file name in a directory to a file handle. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply ³⁴ . RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Read Conges tion Time	ma_nod e_mount stats_rea d_backlo g_wait	Read is an NFS operation that reads data from a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Read Round Trip Time	ma_nod e_mount stats_rea d_rtt	Read is an NFS operation that reads data from a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply ³⁴ . RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Write Conges tion Time	ma_nod e_mount stats_wri te_backl og_wait	Write is an NFS operation that writes data to a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥ 0	N/A	N/ A	N/A

Cat ego ry	Name	Metric	Description	Unit	Val ue Ran ge	Alarm Thresh old	Al ar m Se ve rit y	Soluti on
	NFS Write Round Trip Time	ma_nod e_mount stats_wri te_rtt	Write is an NFS operation that writes data to a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply ³⁴ . RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥ 0	N/A	N/ A	N/A

Networking Metrics

Table 6-3 Diagnosis (InfiniBand, collected only in dedicated resource pools)

Category	Name	Metric	Description	Unit	Value Range
InfiniB and or RoCE network	PortXmitData	infiniband_port_xmit_data_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.	Total count	Natural number
	PortRcvData	infiniband_port_rcv_data_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.	Total count	Natural number
	SymbolErrorCounter	infiniband_symbol_error_counter_total	Total number of minor link errors detected on one or more physical lanes.	Total count	Natural number
	LinkErrorRecoveryCounter	infiniband_link_error_recovery_counter_total	Total number of times the Port Training state machine has successfully completed the link error recovery process.	Total count	Natural number

Category	Name	Metric	Description	Unit	Value Range
	PortRcvErrors	infiniband_port_rcv_errors_total	Total number of packets containing errors that were received on the port including: Local physical errors (ICRC, VCRC, LPCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine) Malformed data packet errors (LVer, length, VL) Malformed link packet errors (operand, length, VL) Packets discarded due to buffer overrun (overflow)	Total count	Natural number
	LocalLinkIntegrityErrors	infiniband_local_link_integrity_errors_total	This counter indicates the number of retries initiated by a link transfer layer receiver.	Total count	Natural number
	PortRcvRemotePhysicalErrors	infiniband_port_rcv_remote_physical_errors_total	Total number of packets marked with the EBP delimiter received on the port.	Total count	Natural number
	PortRcvSwitchRelayErrors	infiniband_port_rcv_switch_relay_errors_total	Total number of packets received on the port that were discarded when they could not be forwarded by the switch relay for the following reasons: DLID mapping VL mapping Looping (output port = input port)	Total count	Natural number

Category	Name	Metric	Description	Unit	Value Range
	PortXmitWait	infiniband_port_transmit_wait_total	The number of ticks during which the port had data to transmit but no data was sent during the entire tick (either because of insufficient credits or because of lack of arbitration).	Total count	Natural number
	PortXmitDiscards	infiniband_port_xmit_discards_total	Total number of outbound packets discarded by the port because the port is down or congested.	Total count	Natural number

Label Metrics

Table 6-4 Metric labels

Classification	Label	Description
Container metrics	modelarts_service	Service to which a container belongs, which can be notebook , train , or infer
	instance_name	Name of the pod to which the container belongs
	service_id	Instance or job ID displayed on the page, for example, cf55829e-9bd3-48fa-8071-7ae870dae93a for a development environment 9f322d5a-b1d2-4370-94df-5a87de27d36e for a training job
	node_ip	IP address of the node to which the container belongs
	container_id	Container ID
	cid	Cluster ID
	container_name	Container name
	project_id	Project ID of the account to which the user belongs

Classification	Label	Description
	user_id	User ID of the account to which the user who submits the job belongs
	pool_id	ID of a resource pool corresponding to a physical dedicated resource pool
	pool_name	Name of a resource pool corresponding to a physical dedicated resource pool
	logical_pool_id	ID of a logical subpool
	logical_pool_name	Name of a logical subpool
	gpu_uuid	UUID of the GPU used by the container
	gpu_index	Index of the GPU used by the container
	gpu_type	Type of the GPU used by the container
	account_name	Account name of the creator of a training, inference, or development environment task
	user_name	Username of the creator of a training, inference, or development environment task
	task_creation_time	Time when a training, inference, or development environment task is created
	task_name	Name of a training, inference, or development environment task
	task_spec_code	Specifications of a training, inference, or development environment task
	cluster_name	CCE cluster name
Node metrics	cid	ID of the CCE cluster to which the node belongs
	node_ip	IP address of the node
	host_name	Hostname of a node
	pool_id	ID of a resource pool corresponding to a physical dedicated resource pool
	project_id	Project ID of the user in a physical dedicated resource pool
	gpu_uuid	UUID of a node GPU
	gpu_index	Index of a node GPU
	gpu_type	Type of a node GPU

Classification	Label	Description
	device_name	Device name of an InfiniBand or RoCE network NIC
	port	Port number of the InfiniBand NIC
	physical_state	Status of each port on the InfiniBand NIC
	firmware_version	Firmware version of the IB NIC
	filesystem	NFS-mounted file system
	mount_point	NFS mount point
Diagnos	cid	ID of the CCE cluster to which the node where the GPU resides belongs
	node_ip	IP address of the node where the GPU resides
	pool_id	ID of a resource pool corresponding to a physical dedicated resource pool
	project_id	Project ID of the user in a physical dedicated resource pool
	gpu_uuid	GPU UUID
	gpu_index	Index of a node GPU
	gpu_type	Type of a node GPU
	device_name	Name of a network device or disk device
	port	Port number of the InfiniBand NIC
	physical_state	Status of each port on the InfiniBand NIC
	firmware_version	Firmware version of the InfiniBand NIC

6.4 Using Grafana to View AOM Monitoring Metrics

6.4.1 Installing and Configuring Grafana

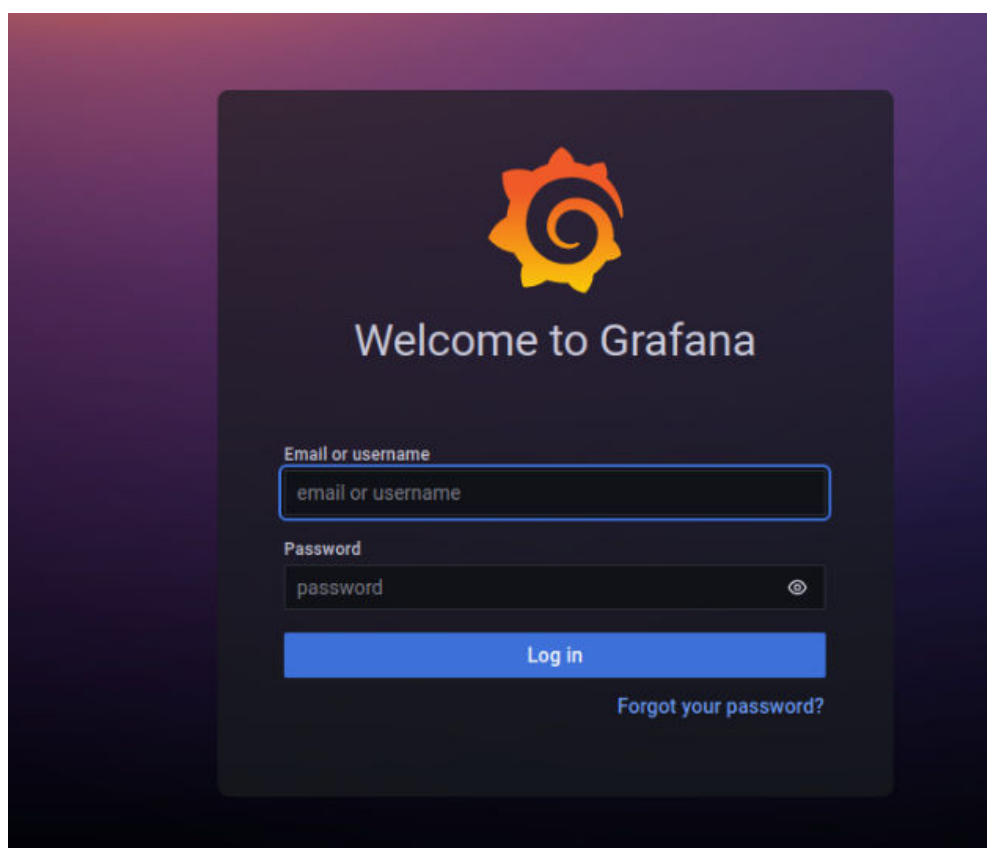
6.4.1.1 Installing and Configuring Grafana on Windows

Description

This section describes how to install and configure Grafana on a PC running Windows.

Procedure

1. Download the Grafana installation package.
Go to the [download link](#), click **Download the installer**, and wait until the download is successful.
2. Install Grafana.
Double-click the installation package and install Grafana as instructed.
3. In Windows Services Manager, enable Grafana.
4. Log in to Grafana.
Grafana runs on port 3000 by default. After you open <http://localhost:3000>, the Grafana login page is displayed. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.



6.4.1.2 Installing and Configuring Grafana on Linux

Description

This section describes how to install and configure Grafana on a PC running Linux.

Prerequisites

- An Ubuntu server that is accessible to the Internet is available. If no, the following conditions must be met:

- You have obtained an ECS. (You are advised to select 8 vCPUs or higher, Ubuntu image of 22.04 version, and 100 GB local storage.) For details, see [Purchasing a Custom ECS](#).
- You have purchased an EIP and bound it to the ECS. For details, see [Assigning an EIP and Binding It to an ECS](#).

Procedure

1. Log in to the ECS. Select a login method. For details, see [Logging in to an ECS](#).

2. Install libfontconfig1:

```
sudo apt-get install -y adduser libfontconfig1
```

The operation is successful if the following information is displayed.

```
root@ecs-9ec3:~# sudo apt-get install -y adduser libfontconfig1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
adduser is already the newest version (3.118ubuntu5).
adduser set to manually installed.
libfontconfig1 is already the newest version (2.13.1-4.2ubuntu5).
libfontconfig1 set to manually installed.
The following packages were automatically installed and are no longer required:
  eatmydata libeatmydata1 libflashrom1 libftdi1-2 python-babel-localedata python3-babel python3-certifi python3-jinja2
  python3-json-pointer python3-jsonpatch python3-jsonschema python3-markupsafe python3-pyrsistent python3-requests python3-tz
  python3-urllib3
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
```

3. Download the Grafana installation package:

```
wget https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb --no-check-certificate
```

Download completed

```
root@ecs-9ec3:~# wget https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb --no-check-certificate
--2023-03-07 10:22:12-- https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb
Resolving dl.grafana.com (dl.grafana.com)... 151.101.42.217
Connecting to dl.grafana.com (dl.grafana.com)|151.101.42.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 89252050 (85M) [application/octet-stream]
Saving to: 'grafana_9.3.6_amd64.deb'

grafana_9.3.6_amd64.deb 100%[=====] 85.12M 379KB/s in 2m 21s
2023-03-07 10:24:36 (617 KB/s) - 'grafana_9.3.6_amd64.deb' saved [89252050/89252050]
```

4. Install Grafana:

```
sudo dpkg -i grafana_9.3.6_amd64.deb
```

```
root@ecs-9ec3:~# sudo dpkg -i grafana_9.3.6_amd64.deb
Selecting previously unselected package grafana.
(Reading database ... 80788 files and directories currently installed.)
Preparing to unpack grafana_9.3.6_amd64.deb ...
Unpacking grafana (9.3.6) ...
Setting up grafana (9.3.6) ...
Adding system user 'grafana' (UID 116) ...
Adding new user 'grafana' (UID 116) with group 'grafana' ...
Not creating home directory '/usr/share/grafana'.
### NOT starting on installation, please execute the following statements to configure grafana to start automatically using syst
emd
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable grafana-server
### You can start grafana-server by executing
sudo /bin/systemctl start grafana-server
```

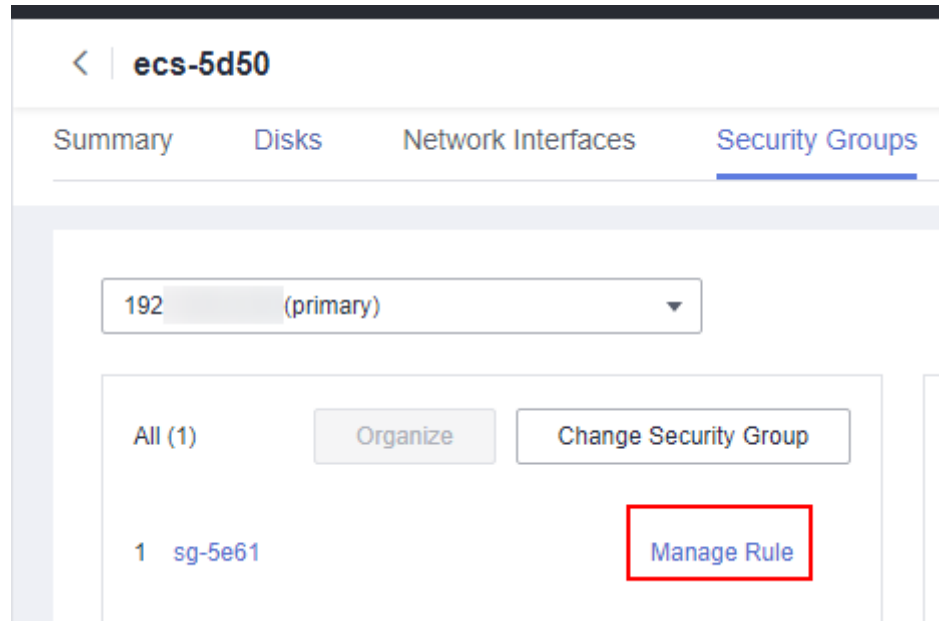
5. Start Grafana:

```
sudo /bin/systemctl start grafana-server
```

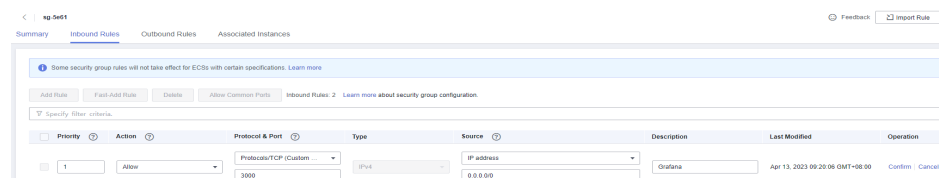
6. Access Grafana configurations on your local PC.

Ensure that an EIP has been bound to the ECS and the [security group](#) configuration is correct (the inbound traffic from TCP port 3000 and all outbound traffic are allowed). Configuration process:

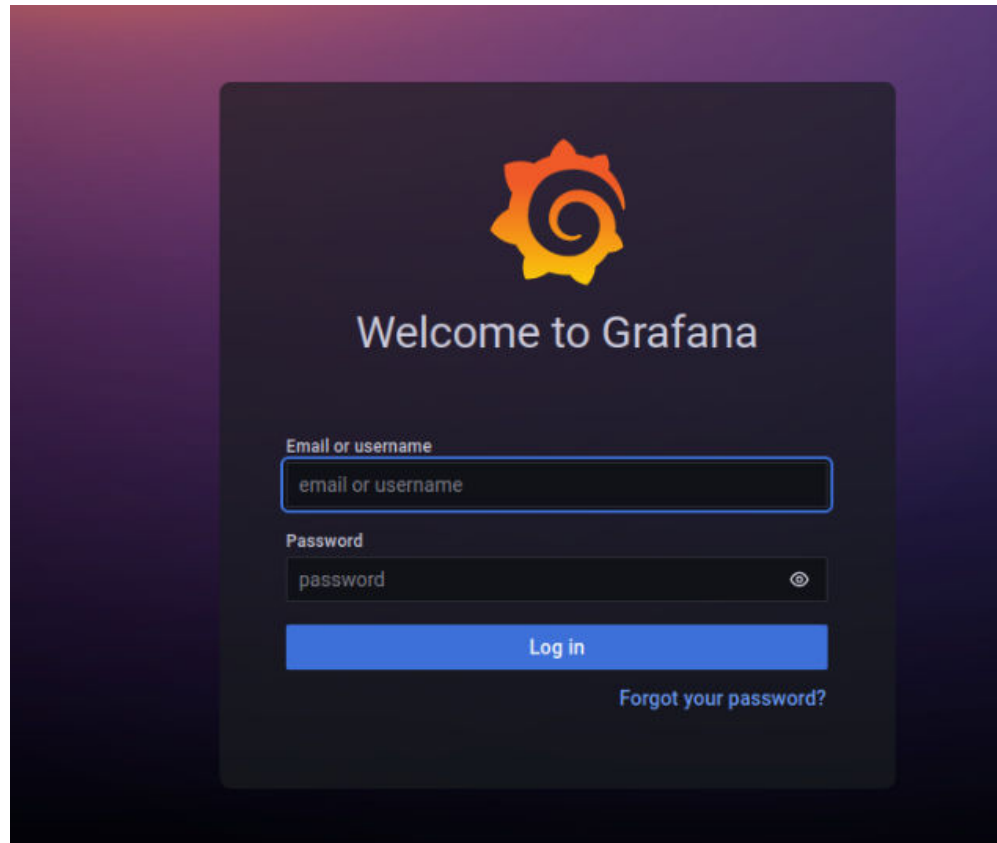
- a. Click the ECS name to go to the ECS details page. Then, click the **Security Groups** tab, and click **Manage Rule**.



- b. Click **Inbound Rules** and allow inbound traffic from TCP port 3000. By default, all outbound traffic is allowed.



7. Access **http://{E/P}:3000** in a browser. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.



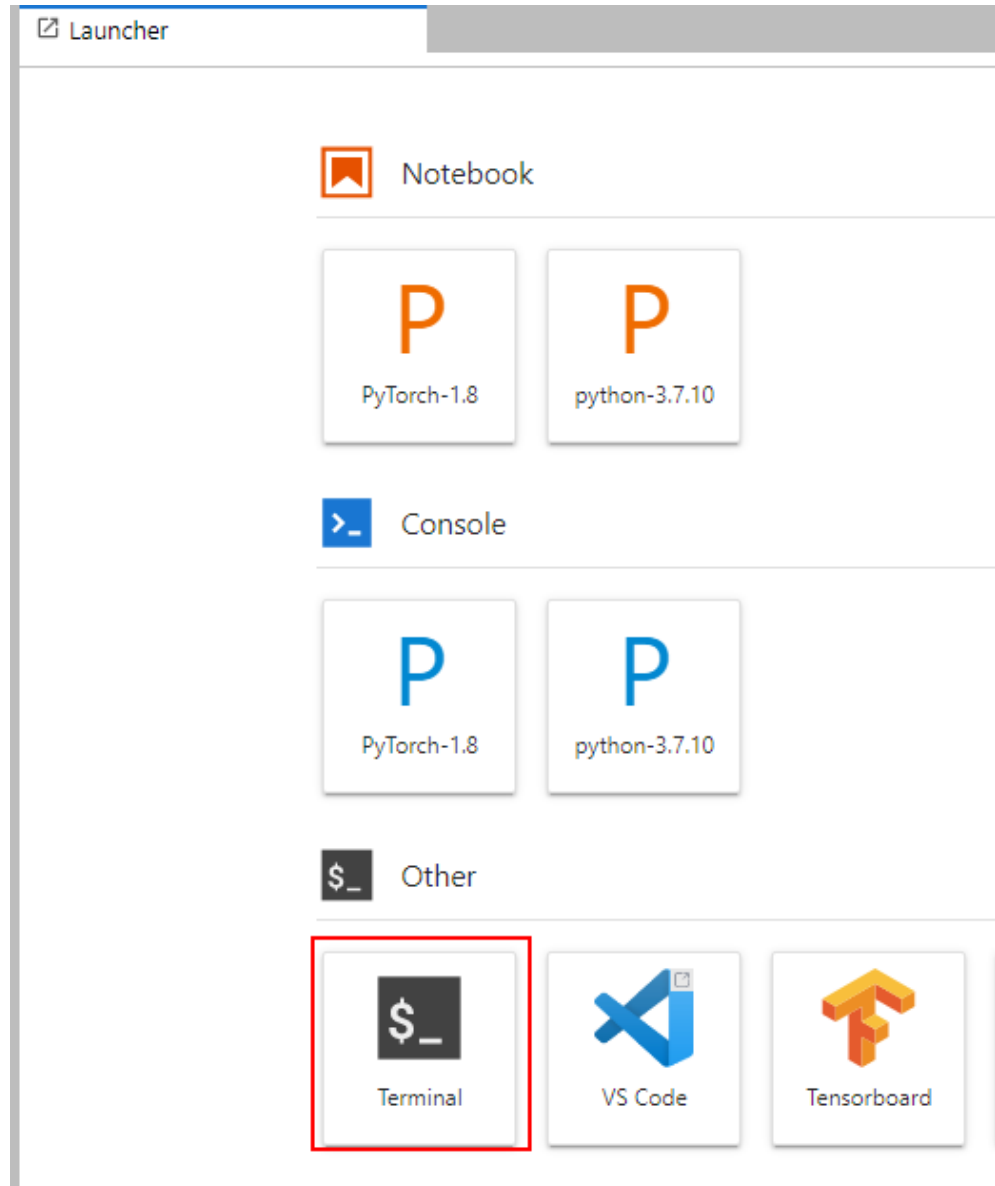
6.4.1.3 Installing and Configuring Grafana on a Notebook Instance

Description

This section describes how to install and configure Grafana in ModelArts Standard notebook instances.

Prerequisites

- A running CPU- or GPU-based notebook instance is available.
- A terminal is opened.



Procedure

1. Run the following commands in sequence in your terminal to download and install Grafana:

```
mkdir -p /home/ma-user/work/grf
cd /home/ma-user/work/grf
wget https://dl.grafana.com/oss/release/grafana-9.1.6.linux-amd64.tar.gz
tar -zxvf grafana-9.1.6.linux-amd64.tar.gz
```

```
(PyTorch-1.8) [ma-user work]$ mkdir -p /home/ma-user/work/grf
(PyTorch-1.8) [ma-user work]$ cd /home/ma-user/work/grf
(PyTorch-1.8) [ma-user grf]$ wget https://dl.grafana.com/oss/release/grafana-9.1.6.linux-amd64.tar.gz
--2023-01-08 15:53:41-- https://dl.grafana.com/oss/release/grafana-9.1.6.linux-amd64.tar.gz
Resolving proxy.modelarts.com (proxy.modelarts.com)... 192.168.6.3
Connecting to proxy.modelarts.com (proxy.modelarts.com)|192.168.6.3|:80... connected.
Proxy request sent, awaiting response... 200 OK
Length: 81857482 (77M) [application/x-tar]
Saving to: 'grafana-9.1.6.linux-amd64.tar.gz.1'
grafana-9.1.6.linux-amd64.tar.gz.1 50K[====>] 4.41M 57.0KB/s eta 8m 19s
```

2. Register Grafana with jupyter-server-proxy.
 - a. Run the following commands in your terminal:

```
mkdir -p /home/ma-user/.local/etc/jupyter
vi /home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py
```

```
(PyTorch-1.8) [ma-user grf]$mkdir -p /home/ma-user/.local/etc/jupyter
(PyTorch-1.8) [ma-user grf]$vi /home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py
```

- b. In `jupyter_notebook_config.py`, add the following code, press **Esc** to exit, and type **:wq** to save the changes:

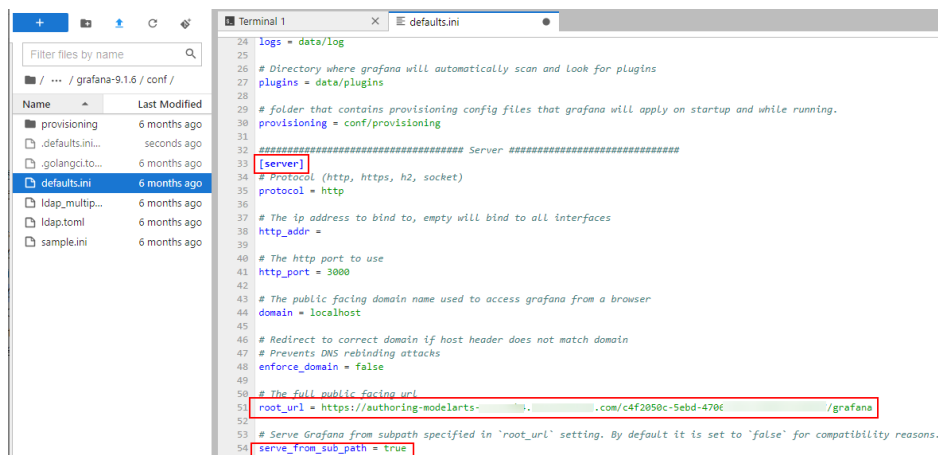
```
c.ServerProxy.servers = {
    'grafana': {
        'command': ['/home/ma-user/work/grf/grafana-9.1.6/bin/grafana-server', '--
homepath', '/home/ma-user/work/grf/grafana-9.1.6', 'web'],
        'timeout': 1800,
        'port': 3000
    }
}
```

NOTE

If `jupyter_notebook_config.py` (path: `/home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py`) contains the `c.ServerProxy.servers` field, add the corresponding key-value pair.

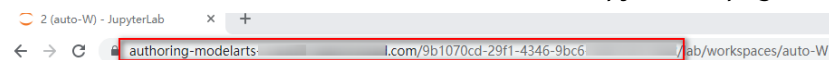
3. Modify the URL for accessing Grafana in JupyterLab.
- a. In the navigation pane on the left, open the `vi /home/ma-user/work/grf/grafana-9.1.6/conf/defaults.ini` file.
- b. Change the `root_url` and `serve_from_sub_path` fields in `[server]`.

Figure 6-4 Modifying the `defaults.ini` file



In the file:

- The value of `root_url` is in the format of `https://{JupyterLab domain name}/{Instance ID}/grafana`. You can obtain the domain name and instance ID from the address box of the JupyterLab page.

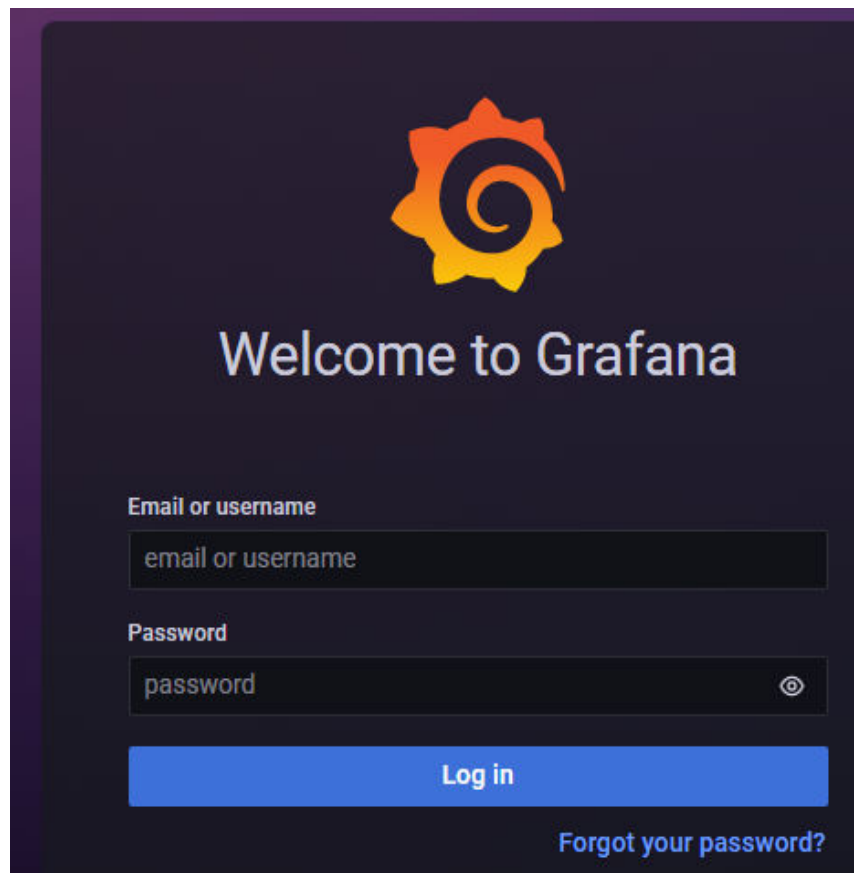


- Set `Serve_from_sub_path` to `true`.

4. Save the image of the notebook instance.
- a. Log in to the ModelArts console and choose **DevEnviron > Notebook**. In the notebook instance list, locate the target instance, and choose **More > Save Image** in the **Operation** column.

- b. In the **Save Image** dialog box, configure parameters. Click **OK** to save the image.
 - c. 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.
 - d. After the image is saved, the instance status changes to **Running**. Then, restart the notebook instance.
5. Open the Grafana page.

Open a browser window and type the value of **root_url** configured in **3** in the address box. If the Grafana login page is displayed, Grafana is installed and configured in the notebook instance. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.



6.4.2 Configuring a Grafana Data Source

Before viewing ModelArts monitoring data on Grafana, configure the data source.

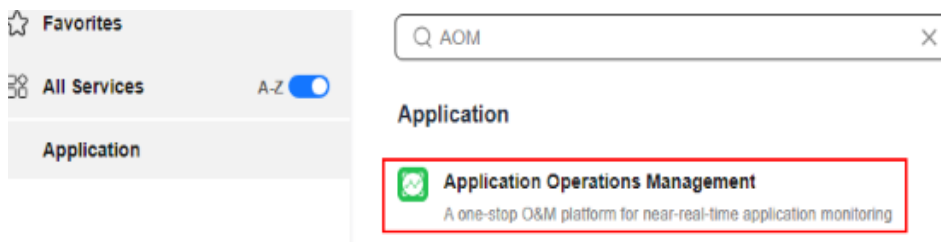
Prerequisites

Grafana has been installed.

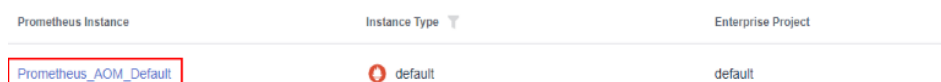
Procedure

1. Obtain the Grafana data source configuration code.

- a. Log in to the AOM console.

Figure 6-5 AOM console

- b. In the navigation pane on the left, choose **Prometheus Monitoring** > **Instances**. Then, click the **Prometheus_AOM_Default** instance.

Figure 6-6 Prometheus_AOM_Default

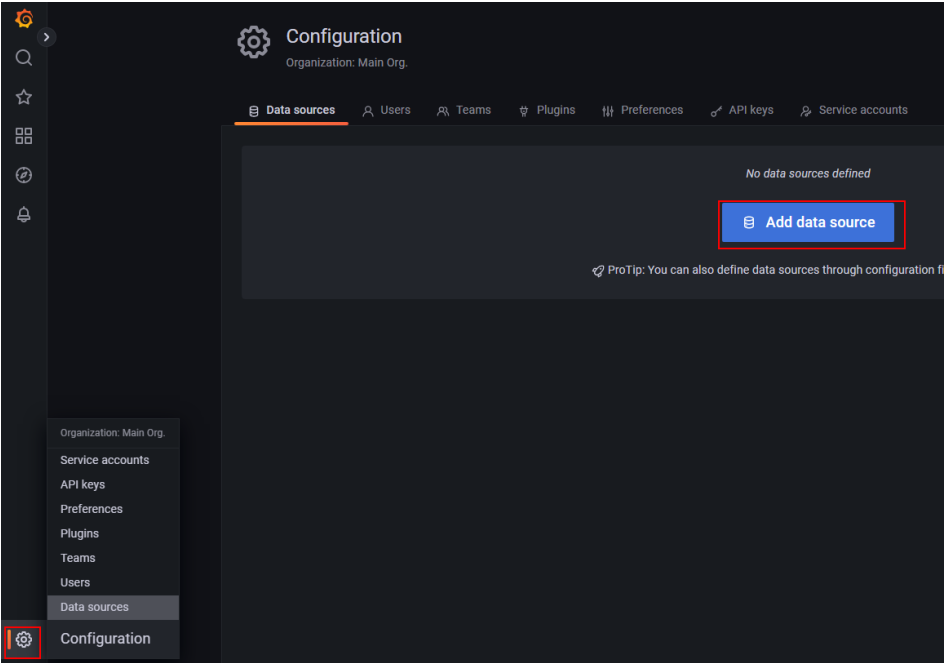
- c. In the **Settings** tab, obtain the Grafana data source configuration code of the instance in the **Grafana Data Source Info** area.

Grafana Data Source InfoIntranet Public Network

HTTP URL	https://aom.ap-
Username	a8e-
Password	

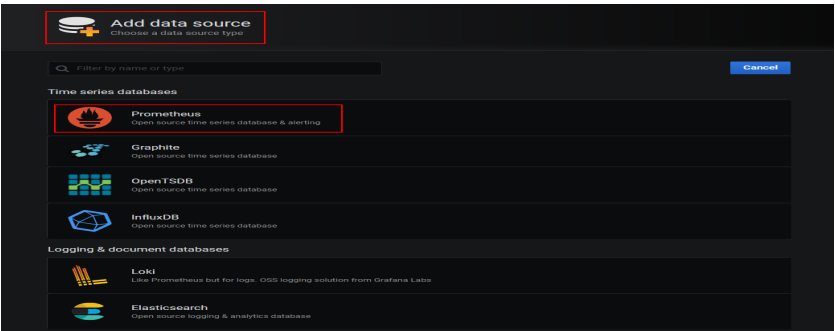
2. Add a data source to Grafana.
 - a. Log in to Grafana. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.
 - b. In the navigation pane, choose **Configuration** > **Data Sources**. Then, click **Add data source**.

Figure 6-7 Configuring Grafana



- c. Click **Prometheus** to access the configuration page.

Figure 6-8 Prometheus configuration page



- d. Configure parameters as shown in the following figure.

Figure 6-9 Configuring a Grafana data source

Name ⓘ PrometheusDefault ☐

HTTP

URL ⓘhttps://

AccessServer (default) ▾Help >

Allowed cookies ⓘNew tag (enter key to add)

Timeout ⓘTimeout in seconds

Auth

Basic auth ☒With Credentials ⓘ ☐

TLS Client Auth ☐With CA Cert ⓘ ☐

Skip TLS Verify ☒

Forward OAuth Identity ⓘ ☐

Basic Auth Details

Useruser

PasswordconfiguredReset

NOTE

The actual Grafana version varies depending on the installation method. [Figure 6-9](#) is only an example.

Table 6-5 Parameters

Parameter	Description
Name	Enter a name.
URL	Set this parameter to the HTTP URL obtained in step 1.
Basic auth	Enable it.
Skip TLS Verify	Enable it.
User	Set this parameter to the username obtained in step a.
Password	Set this parameter to the password obtained in step a.

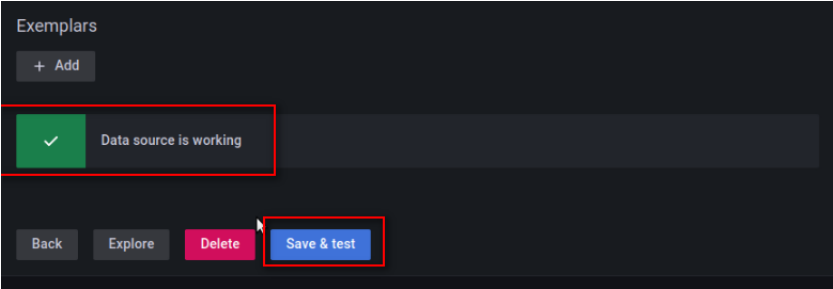
Issue 01 (2025-09-08)

Copyright © Huawei Technologies Co., Ltd.

351

- e. After the configuration, click **Save & test**. If the message **Data source is working** is displayed, the data source is configured.

Figure 6-10 Data source added



6.4.3 Configuring a Dashboard to View Metric Data

In Grafana, you can customize dashboards for various views. ModelArts also provides configuration templates for clusters. This section describes how to configure a dashboard by using a ModelArts template or creating a dashboard. For more usage, see [Grafana tutorials](#).

Preparations

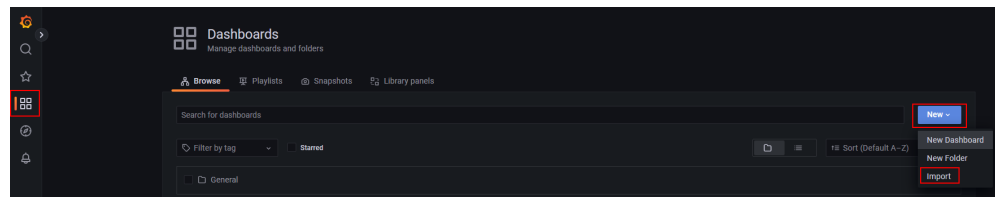
ModelArts provides templates for cluster view, node view, user view, task view, and task details view. These templates can be downloaded from Grafana official documents. You can import and use them on Dashboards.

Table 6-6 Template download URLs

Template	Download URL
Cluster view	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Cluster-View.json
Node view	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Node-View.json
User view	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-User-View.json
Task view	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Task-View.json
Task details view	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Task-Detail-View.json

Using a ModelArts Template to View Metrics

1. Open **Dashboards** and choose **New > Import**.



2. Import the dashboard template.

Copy the template download URL provided in [Preparations](#) to a web browser and copy the content of the JSON file. Paste the content to the dashboard template and click **Load**.

Figure 6-11 Copying the content of the JSON file

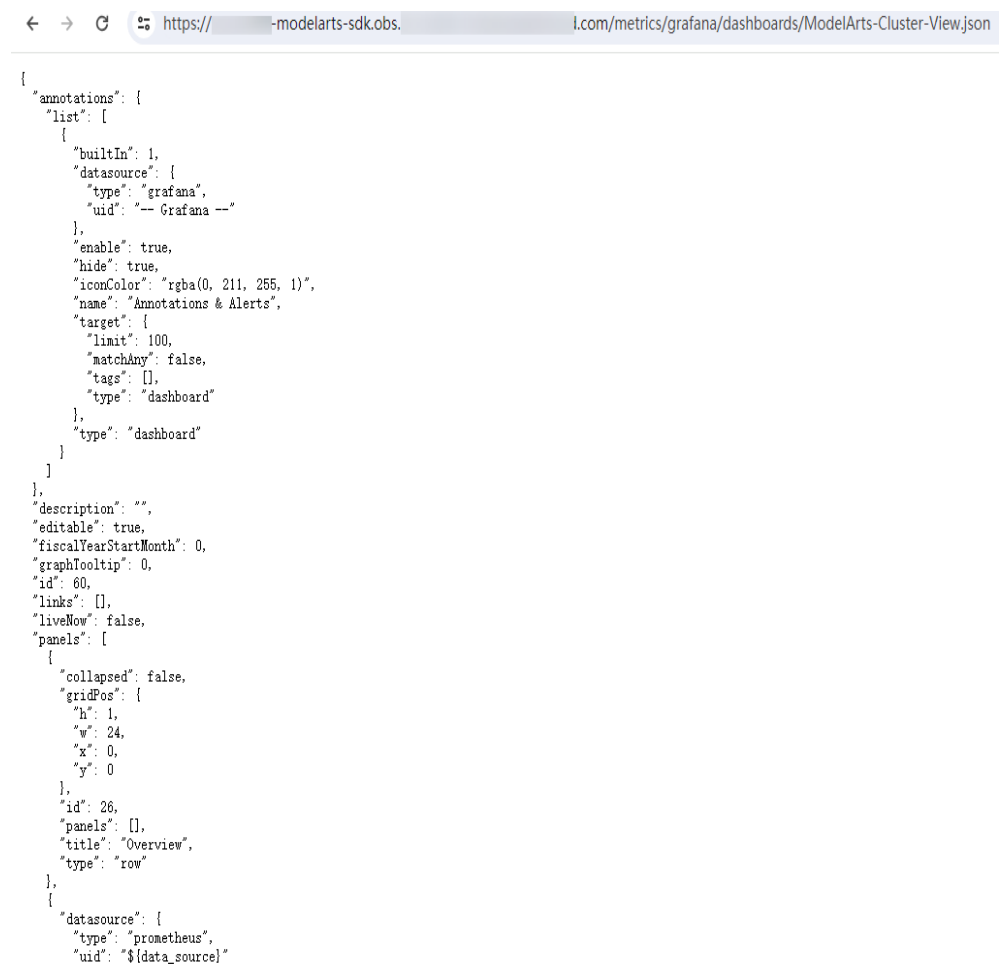
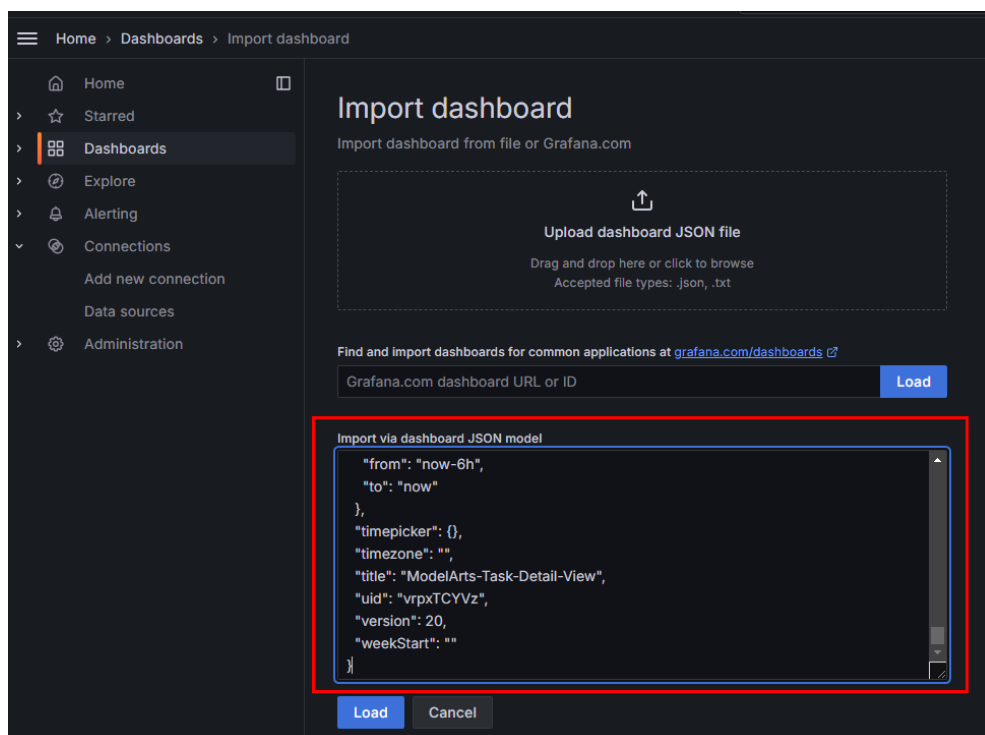
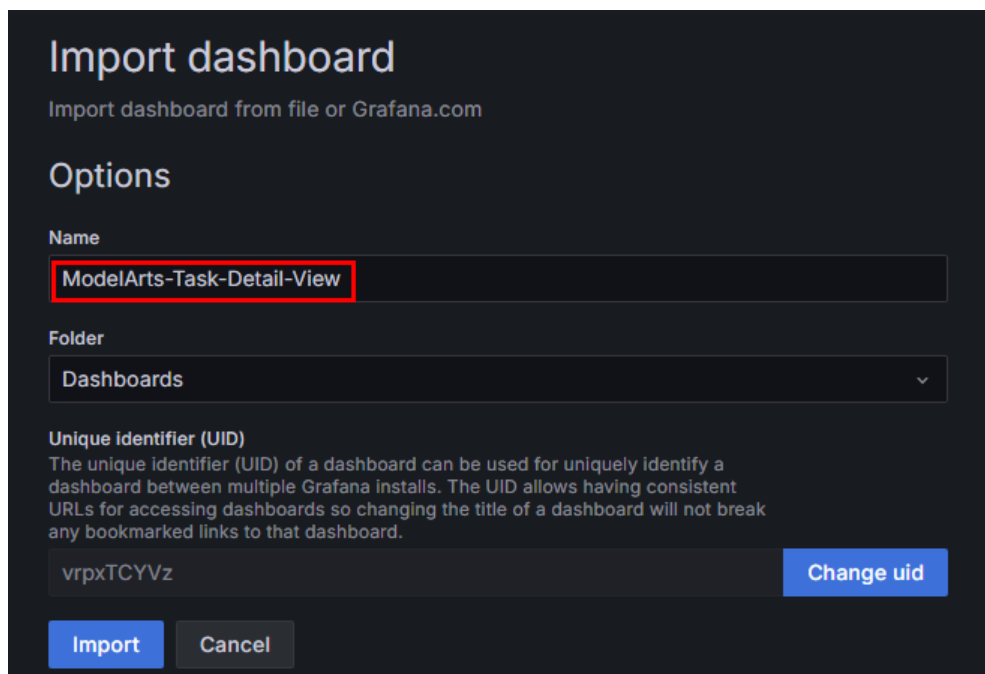
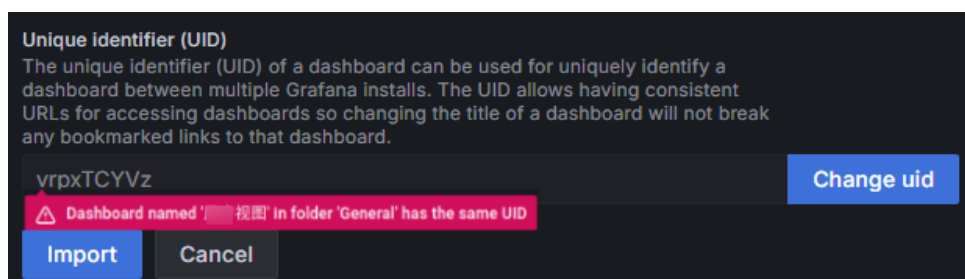


Figure 6-12 Pasting the content of the JSON file to the dashboard template

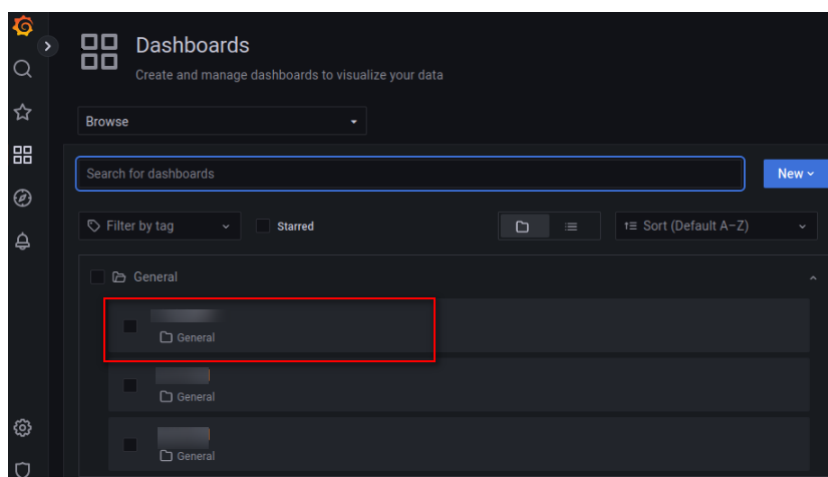
3. Change the view name and click **Import**.

Figure 6-13 Changing the view name

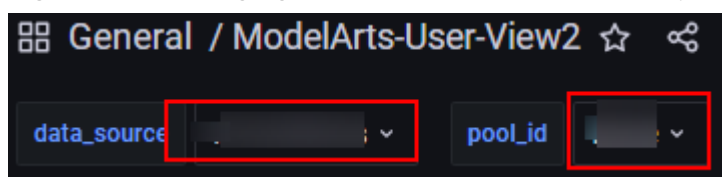
Note: If a message is displayed, indicating that the UID is duplicate, click **Change uid**, change the UID in the JSON file, and click **Import**.

Figure 6-14 Changing the UID

4. After the import, view the imported views in **Dashboards**. Then, click a view to open the monitoring page.



5. Use the template.
After the import is successful, you can click the template to view its details. This section introduces some commonly-used functions.
 - Changing the data source and resource pool

Figure 6-15 Changing the data source and resource pool

Click the area marked by the red box. A drop-down list will appear. From there, you can change the data source and the resource pool.

- Refreshing data



Click the refresh button in the upper right corner to refresh all data on the dashboard. The data on each panel is also updated.

- Changing the automatic refresh time

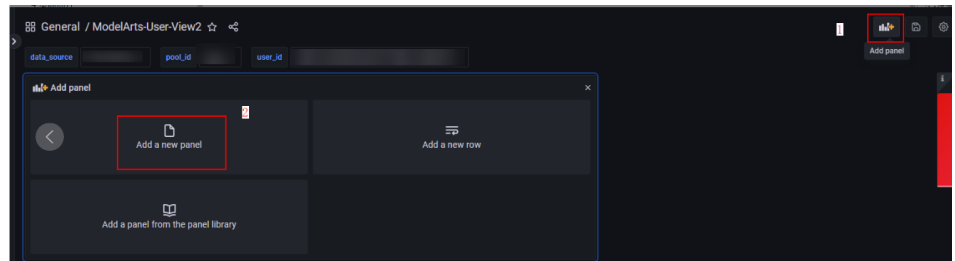
The default refresh interval of a template is 15 minutes. If you need to update the interval, change the value from the drop-down list box in the upper right corner.

- Changing the time range for obtaining dashboard data

Figure 6-16 Changing the time range for obtaining data

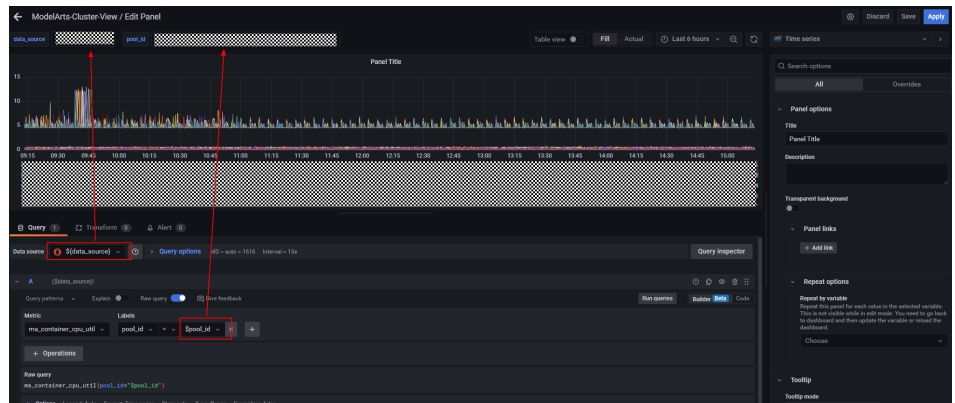
Click the button in the upper right corner to change time range for obtaining data. This time range affects all panels except those with a fixed time.

- Adding a panel

Figure 6-17 Adding a panel

Click the + icon in the upper right corner to add a panel.

After a panel is added, you can obtain the data in the panel. Configure the data source and resource pool as follows to use the current dashboard settings.

Figure 6-18 Using the current dashboard settings

Creating a Dashboard to View Metrics

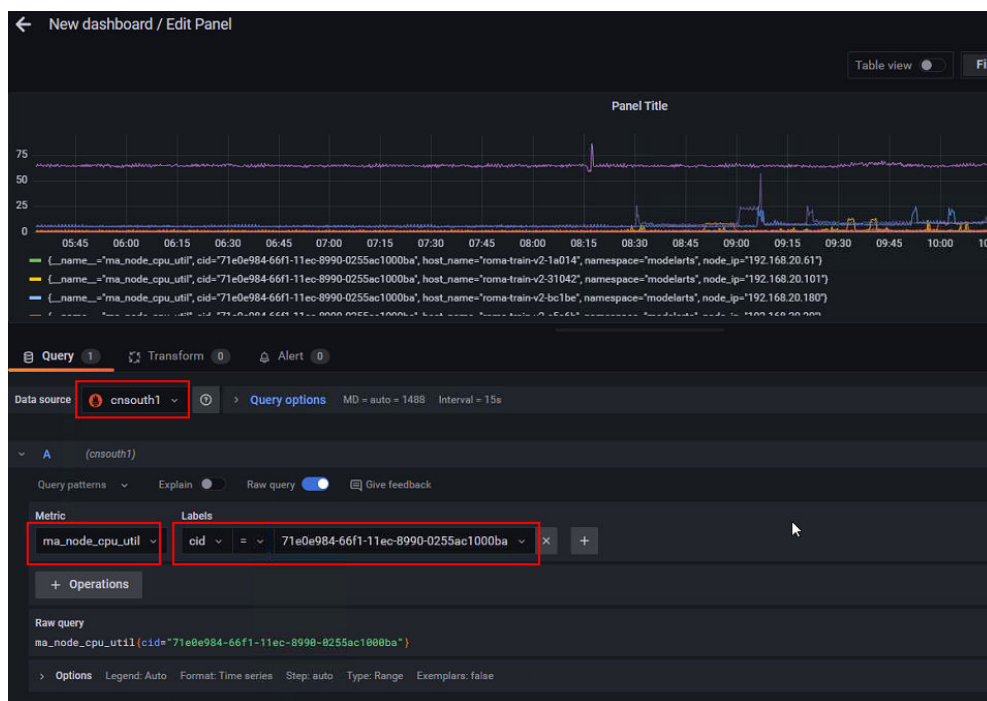
1. Open **Dashboards**, click **New**, and choose **New Dashboard**.
2. Click **Add a new panel**.
3. On the **New dashboard / Edit Panel** page, set the following parameters:

Data source: [Configured Grafana data source](#)

Metric: Metric name. You can obtain the metric to be queried by referring to [Table 6-1](#), [Table 6-2](#), and [Table 6-3](#).

Labels: Used for filtering the metric. For details, see [and](#) .

Figure 6-19 Creating a dashboard to view metrics



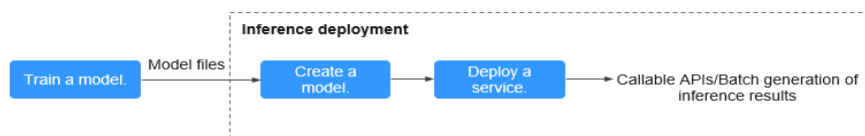
7 History

7.1 Inference Deployment(To go offline)

7.1.1 Overview

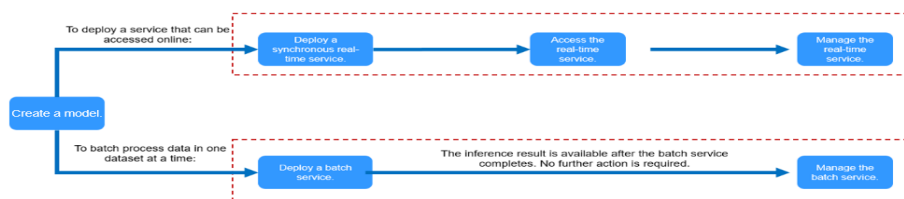
You can import and deploy AI models as inference services. These services can be integrated into your IT platform by calling APIs or generate batch results.

Figure 7-1 Introduction to inference



1. Train a model: Models can be trained in ModelArts or your local development environment. A locally developed model must be uploaded to OBS.
2. Create a model: Import the model file and inference file to the ModelArts model repository and manage them by version. Use these files to build an executable model.
3. Deploy a service: Deploy the model as a service type based on your needs.
 - **Deploying a Model as Real-Time Inference Jobs**
Deploy a model as a web service with real-time UI and monitoring. This service provides you a callable API.
 - **Deploying a Model as a Batch Inference Service**
Deploy an AI application as a batch service that performs inference on batch data and automatically stops after data processing is complete.

Figure 7-2 Different inference scenarios



7.1.2 Creating a Model

7.1.2.1 Creation Methods

AI development and optimization require frequent iterations and debugging. Modifications in datasets, training code, or parameters affect the quality of models. If the metadata of the development process cannot be centrally managed, the optimal model may fail to be reproduced.

With ModelArts, you can create models using meta models from training jobs, OBS, or container images, and centrally manage all iterated and debugged models.

Meta Model Sources

- **Importing a Meta Model from a Training Job:** Create a training job in ModelArts to train a model. After obtaining a desired model, use it to create a model for service deployment.
- **Importing a Meta Model from OBS:** If you use a mainstream framework to develop and train a model locally, you can upload the model to an OBS bucket based on the model package specifications, import the model from OBS to ModelArts, and use the model for service deployment.
- **Importing a Meta Model from a Container Image:** If an AI engine is not supported by ModelArts, you can use it to build a model, import the model to ModelArts as a custom image, and use the image to create a model for service deployment.

Supported AI Engines for ModelArts Inference

If you import a model from OBS to ModelArts, the following AI engines and versions are supported.

NOTE

- A runtime environment of a unified image is named in the following format: *<AI engine and version> - <Hardware and version: CPU, CUDA, or CANN> - <Python version> - <OS version> - <CPU architecture>*
- Each preset AI engine has its default model start command. Do not modify it unless necessary.

Table 7-1 Supported AI engines, their runtime environments, and default start commands

Engine	Runtime Environment	Note
TensorFlow	tf1.13-python3.7-cpu tf1.13-python3.7-gpu tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	• The default runtime environment is tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64. • Default start command: sh / home/mind/run.sh
Spark_MLlib	python2.7 python3.6	• Spark_MLlib 2.3.2 is used in python2.7 and python3.6. • python2.7 and python3.6 can only be used to run models on CPUs. • Default start command: bash / home/work/predict/bin/run.sh
Scikit_Learn	python2.7 python3.6	• Scikit_Learn 0.18.1 is used in python2.7 and python3.6. • python2.7 and python3.6 can only be used to run models on CPUs. • Default start command: bash / home/work/predict/bin/run.sh
XGBoost	python2.7 python3.6	• XGBoost 0.80 is used in python2.7 and python3.6. • python2.7 and python3.6 can only be used to run models on CPUs. • Default start command: bash / home/work/predict/bin/run.sh
PyTorch	python3.7 pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64	• The default runtime environment is pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64. • Default start command: sh / home/mind/run.sh

7.1.2.2 Importing a Meta Model from a Training Job

Create a training job in ModelArts to obtain a satisfactory model. The model can then be imported to create an AI application for centralized management. The application can be quickly deployed as a service.

Constraints

- You can directly import a model generated from a training job that uses a subscribed algorithm to ModelArts, without needing to use the inference code or configuration file.
- If the meta model is from a container image, ensure the size of the meta model complies with [Restrictions on the Size of an Image for Importing a Model](#).

Prerequisites

- The training job has been executed, and the model has been stored in the OBS directory where the training output is stored (the input parameter is **train_url**).
- If the training job uses a mainstream framework or custom image, upload the inference code and configuration file to the model storage directory by referring to [Model Package Structure](#).
- The OBS directory you use must be in the same region as ModelArts.

Procedure

1. Log in to the ModelArts console and choose **Model Management** in the navigation pane on the left.
2. Click **Create Model**.
3. Configure parameters.
 - a. Set basic information about the model. For details about the parameters, see [Table 7-2](#).

Table 7-2 Basic information

Parameter	Description
Name	Model name. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Version	Model version. The default value is 0.0.1 for the first import. NOTE After a model is created, you can create new versions using different meta models for optimization.
Description	Brief description of the model.

- b. Set **Meta Model Source** to **Training job**. For details, see [Table 7-3](#).

Table 7-3 Meta model source parameters

Parameter	Description
Meta Model Source	Select Training job . - Choose a training job from the Training Job drop-down list. Dynamic loading: You can enable it for quick service deployment and update. When it is enabled, model files and runtime dependencies are only pulled during an actual deployment. Enable this feature if a single model file is larger than 5 GB.
AI Engine	Inference engine used by the meta model, which is automatically matched based on the training job you select.
Inference Code	Inference code customizing the inference logic of the model. You can directly copy the inference code URL for use.
Runtime Dependency	Dependencies that the selected model has on the environment.
Model Description	Model descriptions to help other developers better understand and use your model. Click Add Model Description and set the document name and URL. You can add up to three model descriptions.
Deployment Type	Choose the service types for Service Deployment. The service types you select will be the only options available for deployment. For example, selecting Real-Time Services means the model can only be deployed as real-time services.

- c. Confirm the configurations and click **Create now**.

In the model list, you can view the created model and its version. When the status changes to **Normal**, the model is created. On this page, you can perform such operations as creating new versions and quickly deploying services.

Follow-Up Operations

Deploying a service: In the model list, click **Deploy** in the **Operation** column of the target model. Locate the target version, click **Deploy** and choose a service type selected during model creation.

7.1.2.3 Importing a Meta Model from OBS

Import a model trained using a mainstream AI engine to ModelArts to create a model for centralized management.

Constraints

- The imported model, inference code, and configuration file must comply with ModelArts model package specifications. For details, see [Model Package Structure](#), [Specifications for Editing a Model Configuration File](#), and [Specifications for Writing a Model Inference Code File](#).
- If the meta model is from a container image, ensure the size of the meta model complies with [Restrictions on the Size of an Image for Importing a Model](#).

Prerequisites

- The trained model uses an AI engine supported by ModelArts. For details, see [Supported AI Engines for ModelArts Inference](#).
- The model package, inference code, and configuration file have been uploaded to OBS.
- The OBS directory you use and ModelArts are in the same region.

Procedure

1. Log in to the ModelArts console and choose **Model Management** in the navigation pane on the left.
2. Click **Create Model**.
3. Configure parameters.
 - a. Set basic information about the model. For details about the parameters, see [Table 7-4](#).

Table 7-4 Basic information

Parameter	Description
Name	Model name. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Version	Model version. The default value is 0.0.1 for the first import. NOTE After a model is created, you can create new versions using different meta models for optimization.
Description	Brief description of the model.

- b. Set **Meta Model Source** to **OBS**. For details, see [Table 7-5](#).

To import a meta model from OBS, edit the inference code and configuration file by following [model package specifications](#) and place the inference code and configuration file in the **model** folder storing the meta model. If the selected directory does not comply with the model package specifications, the model cannot be created.

Table 7-5 Meta model source parameters

Parameter	Description
Meta Model Source	Select OBS .
Meta Model	OBS path for storing the meta model. The OBS path cannot contain spaces. Otherwise, the model creation will fail.
AI Engine	AI engine, which is automatically set according to the model storage path you select, used by the meta model.
Dynamic loading	You can enable it for quick Service Deployment and update. When it is enabled, model files and runtime dependencies are only pulled during an actual deployment. Enable this feature if a single model file is larger than 5 GB.
Runtime Dependency	Dependencies that the selected model has on the environment.
Model Description	Model descriptions to help other developers better understand and use your model. Click Add Model Description and set the document name and URL. You can add up to three descriptions.
Configuration File	The system associates the configuration file stored in OBS by default. After enabling this feature, you can review and edit the model configuration file. NOTE This feature is to be discontinued. After that, you can modify the model configuration by setting AI Engine , Runtime Dependency , and API Configuration .
Deployment Type	Choose the service types for Service Deployment. The service types you select will be the only options available for deployment. For example, selecting Real-Time Services means the model can only be deployed as real-time services.
Start Command	Customizable start command of the model. This parameter is optional. When using a preset AI engine, the default start command is used if no start command is entered. If a start command is entered, it replaces the default command. NOTE Start commands containing \$, , >, <, `, !, \n, \, ?, -v, --volume, --mount, --tmpfs, --privileged, or --cap-add will be emptied when a model is being published.

Parameter	Description
API Configuration	After enabling this feature, you can edit RESTful APIs to define the input and output formats of a model. The model APIs must comply with ModelArts specifications. For details, see the apis parameter description in Specifications for Editing a Model Configuration File . For details about the code example, see Code Example of apis Parameters .

- c. Confirm the configurations and click **Create now**.

In the model list, you can view the created model and its version. When the status changes to **Normal**, the model is created. On this page, you can perform such operations as creating new versions and quickly deploying services.

Follow-Up Operations

Deploying a service: In the model list, click **Deploy** in the **Operation** column of the target model. Locate the target version, click **Deploy** and choose a service type selected during model creation.

7.1.2.4 Importing a Meta Model from a Container Image

For AI engines that are not supported by ModelArts, you can import the models from custom images.

Constraints

- For details about the specifications and description of custom images, see [Specifications for Custom Images Used for Importing Models](#).
- If the meta model is from a container image, ensure the size of the meta model complies with [restrictions on the size of an image for importing a model](#)"Restrictions on the Size of an Image for Importing a Model" in FAQs.

Prerequisites

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

Procedure

- Log in to the ModelArts console and choose **Model Management** in the navigation pane on the left.
- Click **Create Model**.
- Configure parameters.
 - Set basic information about the model. For details about the parameters, see [Table 7-6](#).

Table 7-6 Basic information

Parameter	Description
Name	Model name. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Version	Model version. The default value is 0.0.1 for the first import. NOTE After a model is created, you can create new versions using different meta models for optimization.
Description	Brief description of the model.

- b. Select the meta model source and configure related parameters. Set **Meta Model Source** to **Container image**. For details, see [Table 7-7](#).

Table 7-7 Meta model source parameters

Parameter	Description
Meta Model Source	Select Container Image
Container Image Path	Click  to import the container image. You do not need to use swr_location in the configuration file to specify the image location. For details about how to create a custom image, see Specifications for Custom Images Used for Importing Models . NOTE The model image you select will be shared with the system administrator, so ensure you have the permission to share the image (images shared by other accounts are not supported). ModelArts will deploy the image as an inference service. Ensure that your image can be properly started and provide an inference API.
Container API	Set the protocol and port number of the inference API defined by the model.

Parameter	Description
Image Replication	Indicates whether to copy the model image in the container image to ModelArts. • After this feature is disabled, the model image is not copied, models can be rapidly created, but modifying or deleting an image in the SWR source directory will affect service deployment. • After this feature is enabled, the model image is copied, models cannot be rapidly created, and modifying or deleting an image in the SWR source directory will not affect service deployment. NOTE You must enable this feature if you want to use images shared by others. Otherwise, models will fail to be created.

Parameter	Description
Health Check	Specifies health check on a model. This parameter is configurable only when a health check API is configured in the custom image. Otherwise, the model creation will fail. The following probes are supported: • Startup Probe: This probe checks if the application instance has started. If a startup probe is provided, all other probes are disabled until it succeeds. If the startup probe fails, the instance is restarted. If no startup probe is provided, the default status is Success. • Readiness Probe: This probe verifies whether the application instance is ready to handle traffic. If the readiness probe fails (meaning the instance is not ready), the instance is taken out of the service load balancing pool. Traffic will not be routed to the instance until the probe succeeds. • Liveness Probe: This probe monitors the application health status. If the liveness probe fails (indicating the application is unhealthy), the instance is automatically restarted. The parameters of the three types of probes are as follows: • Check Mode: Select HTTP request or Command. • Health Check URL: Enter the health check URL, which defaults to /health. This parameter is displayed when Check Mode is set to HTTP request. • Health Check Command: Enter the health check command. This parameter is displayed when Check Mode is set to Command. • Health Check Period (s): Enter an integer ranging from 1 to 2147483647. • Delay (s): Set a delay for the health check to occur after the instance has started. The value should be an integer between 0 and 2147483647. • Timeout (s): Set the timeout interval for each health check. The value should be an integer between 0 and 2147483647. • Maximum Failures: Enter an integer ranging from 1 to 2147483647. If the service fails the specified number of consecutive health checks during startup, it will enter the abnormal state. If the service fails the specified number of

Parameter	Description
	consecutive health checks during operation, it will enter the alarm state. NOTE If health check is enabled for a model, the associated services will stop three minutes after receiving the stop instruction.
Model Description	Model descriptions to help other developers better understand and use your model. Click Add Model Description and set the document name and URL. You can add up to three descriptions.
Deployment Type	Choose the service types for Service Deployment. The service types you select will be the only options available for deployment. For example, selecting Real-Time Services means the model can only be deployed as real-time services.
Start Command	Customizable start command of a model. NOTE Start commands containing \$, , >, <, `, !, \n, \, ?, -v, --volume, --mount, --tmpfs, --privileged, or --cap-add will be emptied when a model is being published.
API Configuration	After enabling this feature, you can edit RESTful APIs to define the input and output formats of a model. The model APIs must comply with ModelArts specifications. For details, see the apis parameter description in Specifications for Editing a Model Configuration File . For details about the code example, see Code Example of apis Parameters .

- c. Confirm the configurations and click **Create now**.

In the model list, you can view the created model and its version. When the status changes to **Normal**, the model is created. On this page, you can perform such operations as creating new versions and quickly deploying services.

Follow-Up Operations

Deploying a service: In the model list, click **Deploy** in the **Operation** column of the target model. Locate the target version, click **Deploy** and choose a service type selected during model creation.

7.1.3 Model Creation Specifications

7.1.3.1 Model Package Structure

When creating a model, make sure that any meta model imported from OBS complies with specifications.

 NOTE

- The model package specifications apply when you import one model. For multiple models, such as multiple files, use custom images instead.
- If you want to use an AI engine that is not supported by ModelArts, use a custom image.
- For details about how to create a custom image, see [Specifications for Custom Images](#) and [Creating a Custom Image on ECS](#).
- For more examples of custom scripts, see [Examples of Custom Scripts](#).

The model package must contain the **model** directory. The **model** directory stores the model file, model configuration file, and model inference code file.

- Model files: The requirements for model files vary according to the model package structure. For details, see [Model Package Example](#).
- Model configuration file: The model configuration file must be available and its name is consistently to be **config.json**. There must be only one model configuration file. For details about how to edit a model configuration file, see [Specifications for Editing a Model Configuration File](#).
- Model inference code file: It is mandatory. The file name is consistently to be **customize_service.py**. There must be only one model inference code file. For details about how to edit model inference code, see [Specifications for Writing a Model Inference Code File](#).
 - The .py file on which **customize_service.py** depends can be directly stored in the **model** directory. Use relative import for the custom package.
 - The other files on which **customize_service.py** depends can be stored in the **model** directory. Use absolute paths to access these files. For details, see [Obtaining an Absolute Path](#).

ModelArts also provides custom script examples of common AI engines. For details, see [Examples of Custom Scripts](#).

Model Package Example

- Structure of the TensorFlow-based model package

When publishing the model, you only need to specify the **ocr** directory.

OBS bucket or directory name

```
|— ocr
|   |— model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
|   |   |— <<Custom Python package>> (Optional) Your Python package, which can be directly
|   |   |   |— saved_model.pb (Mandatory) Protocol buffer file, which contains the graph description of
|   |   |   |   |— variables Name of a fixed sub-directory, which contains the weight and deviation rate of
|   |   |   |   |   |— variables.index Mandatory
|   |   |   |   |   |— variables.data-00000-of-00001 Mandatory
|   |   |   |   |   |— config.json (Mandatory) Model configuration file. The file name is fixed to config.json.
|   |   |   |   |   |   |— customize_service.py (Mandatory) Model inference code. The file name is fixed to
|   |   |   |   |   |   |   |— customize_service.py (Mandatory) Model inference code. The file name is fixed to
|   |   |   |   |   |   |   |   |— customize_service.py (Mandatory) Model inference code. The file name is fixed to
```

Only one model configuration file is allowed.

Only one model inference code file is allowed.

The files on which **customize_service.py** depends can be directly stored in the model directory.

- Structure of a custom model package depends on the AI engine in your custom image. For example, if the AI engine in your custom image is TensorFlow, the model package uses the TensorFlow structure.

7.1.3.2 Specifications for Editing a Model Configuration File

You must edit the **config.json** file when publishing a model. The configuration file describes the model usage, computing framework, accuracy, inference code dependency package, and model API.

Configuration File Format

The configuration file is in JSON format. [Table 7-8](#) describes the parameters.

Table 7-8 Parameters

Parameter	Mandatory	Type	Description
model_algorithm	Yes	String	Model algorithm, which shows the model usage. The value must start with a letter and contain no more than 36 characters. Chinese characters and special characters (&!"'<>=) are not allowed. Major model algorithms include image_classification , object_detection , and predict_analysis .
model_type	Yes	String	Model AI engine, which indicates the computing framework used by a model. Major AI engines and Image are supported. - For details about supported AI engines, see Supported AI Engines for ModelArts Inference. - If model_type is set to Image, a model will be created using a custom image. In this case, the swr_location parameter is mandatory. For details about how to create an image, see Specifications for Custom Images.
runtime	No	String	Model runtime environment. is used by default. The value of runtime depends on model_type . If model_type is set to Image , you do not need to set runtime . If model_type is set to a mainstream framework, select a runtime environment matching the engine. For details about the supported runtime environments, see Supported AI Engines for ModelArts Inference . If your model must run on specified CPUs or GPUs, select the CPUs or GPUs based on the runtime suffix. If the runtime does not contain the CPU or GPU information, check the runtime description in <i>Supported AI Engines for ModelArts Inference</i> .

Parameter	Mandatory	Type	Description
metrics	No	Object	Model precision information, including the F1 score, recall, precision, and accuracy. For details about the metrics object structure, see Table 7-9. The result is displayed in the model precision area on the model details page.
apis	No	API array	Structure data of requests received and returned by a model. It is a RESTful API array provided by a model. For details about the API structure, see Table 7-10. For details about the code example, see Code Example of apis Parameters. • If model_type is set to Image, an AI application will be created using a custom image. APIs with different paths can be declared in apis based on the request path exposed by the image. • If model_type is not Image, only one API whose request path is / can be declared in apis because the preconfigured AI engine exposes only one inference API whose request path is /.
dependencies	No	Dependency array	Package on which the model inference code depends, which is structure data. You must provide the package name, installation method, and version constraints. The dependency package can be installed only using pip. Table 7-13 describes the dependency array. If the model package does not contain the customize_service.py inference code file, you do not need to set dependencies. Dependency packages cannot be installed for custom image models.
health	No	health data structure	Health check API configuration. This parameter is mandatory only when model_type is set to Image. To ensure uninterrupted services during a rolling upgrade, ModelArts requires a health check API. For details about the health data structure, see Table 7-15.

Table 7-9 metrics object

Parameter	Mandatory	Type	Description
f1	No	Number	F1 score. The value is rounded to 17 decimal places.
recall	No	Number	Recall. The value is rounded to 17 decimal places.
precision	No	Number	Precision. The value is rounded to 17 decimal places.
accuracy	No	Number	Accuracy. The value is rounded to 17 decimal places.

Table 7-10 api structure

Parameter	Mandatory	Type	Description
url	No	String	Request path. The default value is a slash (/). For a custom image model (model_type is Image), set this parameter to the actual request path exposed in the image. For a non-custom image model (model_type is not Image), the URL can only be /.
method	No	String	Request method. The default value is POST .
request	No	Object	Request body. For details, see Table 7-11 .
response	No	Object	Response body. For details, see Table 7-12 .

Table 7-11 request structure

Parameter	Mandatory	Type	Description
Content-type	No for real-time services Yes for batch services	String	Data is sent in a specified content format. The default value is application/json . The options are as follows: • application/json: JSON data is uploaded. • multipart/form-data: A file is uploaded. NOTE For machine learning models, only application/json is supported.

Parameter	Mandatory	Type	Description
data	No for real-time services Yes for batch services	String	The request body is described in JSON schema. For details about the parameter description, see the official guide .

Table 7-12 response structure

Parameter	Mandatory	Type	Description
Content-type	No for real-time services Yes for batch services	String	Data is sent in a specified content format. The default value is application/json . NOTE For machine learning models, only application/json is supported.
data	No for real-time services Yes for batch services	String	The response body is described in JSON schema. For details about the parameter description, see the official guide .

Table 7-13 dependency array

Parameter	Mandatory	Type	Description
installer	Yes	String	Installation method. Only pip is supported.
packages	Yes	package array	Dependency package collection. For details about the package array, see Table 7-14 .

Table 7-14 Package array

Parameter	Mandatory	Type	Description
package_name	Yes	String	Dependency package name. Chinese characters and special characters (&!"'<>=) are not allowed.
package_version	No	String	Dependency package version. Leave it blank if it is not required. Chinese characters and special characters (&!"'<>=) are not allowed.
restraint	No	String	Version restriction. This parameter is mandatory only when package_version is configured. Possible values are EXACT, ATLEAST, and ATMOST. • EXACT indicates that a specified version is installed. • ATLEAST indicates that the installed version is not earlier than the specified version. • ATMOST indicates that the installed version is not later than the specified version. NOTE • If there are specific requirements on the version, preferentially use EXACT. If EXACT conflicts with the system installation packages, you can select ATLEAST. • If there is no specific requirement on the version, retain only the package_name parameter and leave restraint and package_version blank.

Table 7-15 Health check data structure

Parameter	Mandatory	Type	Description
check_method	Yes	String	Health check method. The value can be HTTP or EXEC. • HTTP: Use an HTTP request. • EXEC: Execute a command.

Parameter	Mandatory	Type	Description
command	No	String	Health check command. This parameter is mandatory when check_method is set to EXEC .
url	No	String	Request URL of a health check API. This parameter is mandatory when check_method is set to HTTP .
protocol	No	String	Request protocol of a health check API. The default value is http . This parameter is mandatory when check_method is set to HTTP .
initial_delay_seconds	No	String	Delay for initializing the health check.
timeout_seconds	No	String	Health check timeout.
period_seconds	Yes	String	Health check period, in seconds. Enter a positive integer no more than 2147483647.
failure_threshold	Yes	String	Maximum number of health check failures. Enter a positive integer no more than 2147483647.

Code Example of apis Parameters

```
[{
  "url": "/",
  "method": "post",
  "request": {
    "Content-type": "multipart/form-data",
    "data": {
      "type": "object",
      "properties": {
        "images": {
          "type": "file"
        }
      }
    }
  },
  "response": {
    "Content-type": "application/json",
    "data": {
      "type": "object",
      "properties": {
        "mnist_result": {
          "type": "array",
          "item": [
            {
              "type": "string"
            }
          ]
        }
      }
    }
  }
}]
```



```
}  
}  
}  
}  
}]
```

Example of an Object Detection Model Configuration File

The following code uses the TensorFlow engine as an example. You can modify the **model_type** parameter based on the actual engine type.

- Model input

Key: images

Value: image files

- Model output

```
{  
  "detection_classes": [  
    "face",  
    "arm"  
  ],  
  "detection_boxes": [  
    [  
      33.6,  
      42.6,  
      104.5,  
      203.4  
    ],  
    [  
      103.1,  
      92.8,  
      765.6,  
      945.7  
    ]  
  ],  
  "detection_scores": [0.99, 0.73]  
}
```

- Configuration file

```
{  
  "model_type": "TensorFlow",  
  "model_algorithm": "object_detection",  
  "metrics": {  
    "f1": 0.345294,  
    "accuracy": 0.462963,  
    "precision": 0.338977,  
    "recall": 0.351852  
  },  
  "apis": [{  
    "url": "/",  
    "method": "post",  
    "request": {  
      "Content-type": "multipart/form-data",  
      "data": {  
        "type": "object",  
        "properties": {  
          "images": {  
            "type": "file"  
          }  
        }  
      }  
    }  
  }],  
  "response": {  
    "Content-type": "application/json",  
    "data": {  
      "type": "object",  
      "properties": {  
        "detection_classes": [  
          "face",  
          "arm"  
        ],  
        "detection_boxes": [  
          [  
            33.6,  
            42.6,  
            104.5,  
            203.4  
          ],  
          [  
            103.1,  
            92.8,  
            765.6,  
            945.7  
          ]  
        ],  
        "detection_scores": [0.99, 0.73]  
      }  
    }  
  }  
}
```

```
        "detection_classes": {
          "type": "array",
          "items": [{
            "type": "string"
          }]
        },
        "detection_boxes": {
          "type": "array",
          "items": [{
            "type": "array",
            "minItems": 4,
            "maxItems": 4,
            "items": [{
              "type": "number"
            }]
          }]
        },
        "detection_scores": {
          "type": "array",
          "items": [{
            "type": "number"
          }]
        }
      }
    }
  },
  "dependencies": [{
    "installer": "pip",
    "packages": [{
      "restraint": "EXACT",
      "package_version": "1.15.0",
      "package_name": "numpy"
    },
    {
      "restraint": "EXACT",
      "package_version": "5.2.0",
      "package_name": "Pillow"
    }
  ]
}]
}
```

Example of an Image Classification Model Configuration File

The following code uses the TensorFlow engine as an example. You can modify the **model_type** parameter based on the actual engine type.

- Model input

Key: images

Value: image files

- Model output

```
{
  "predicted_label": "flower",
  "scores": [
    ["rose", 0.99],
    ["begonia", 0.01]
  ]
}
```

- Configuration file

```
{
  "model_type": "TensorFlow",
  "model_algorithm": "image_classification",
  "metrics": {
    "f1": 0.345294,
```

```
    "accuracy": 0.462963,  
    "precision": 0.338977,  
    "recall": 0.351852  
  },  
  "apis": [{  
    "url": "/",  
    "method": "post",  
    "request": {  
      "Content-type": "multipart/form-data",  
      "data": {  
        "type": "object",  
        "properties": {  
          "images": {  
            "type": "file"  
          }  
        }  
      }  
    }  
  }],  
  "response": {  
    "Content-type": "application/json",  
    "data": {  
      "type": "object",  
      "properties": {  
        "predicted_label": {  
          "type": "string"  
        },  
        "scores": {  
          "type": "array",  
          "items": [{  
            "type": "array",  
            "minItems": 2,  
            "maxItems": 2,  
            "items": [  
              {  
                "type": "string"  
              },  
              {  
                "type": "number"  
              }  
            ]  
          }  
        ]  
      }  
    }  
  }  
}],  
  "dependencies": [{  
    "installer": "pip",  
    "packages": [{  
      "restraint": "ATLEAST",  
      "package_version": "1.15.0",  
      "package_name": "numpy"  
    },  
    {  
      "restraint": "",  
      "package_version": "",  
      "package_name": "Pillow"  
    }  
  ]  
}]  
}
```

The following code uses the MindSpore engine as an example. You can modify the **model_type** parameter based on the actual engine type.

- Model input
Key: images
Value: image files

- Model output

```
"[[-2.404526 -3.0476532 -1.9888215 0.45013925 -1.7018927 0.40332815\n 11.290332 -1.5861531 5.7887416 ]]"
```

- Configuration file

```
{
  "model_algorithm": "image_classification",
  "model_type": "MindSpore",
  "metrics": {
    "f1": 0.124555,
    "recall": 0.171875,
    "precision": 0.0023493892851938493,
    "accuracy": 0.00746268656716417
  },
  "apis": [{
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "application/json",
      "data": {
        "type": "object",
        "properties": {
          "mnist_result": {
            "type": "array",
            "item": [{
              "type": "string"
            }]
          }
        }
      }
    }
  ]
},
"dependencies": []
}
```

Example of a Predictive Analytics Model Configuration File

The following code uses the TensorFlow engine as an example. You can modify the **model_type** parameter based on the actual engine type.

- Model input

```
{
  "data": {
    "req_data": [
      {
        "buying_price": "high",
        "maint_price": "high",
        "doors": "2",
        "persons": "2",
        "lug_boot": "small",
        "safety": "low",
        "acceptability": "acc"
      },
      {
        "buying_price": "high",
        "maint_price": "high",

```

```
        "doors": "2",
        "persons": "2",
        "lug_boot": "small",
        "safety": "low",
        "acceptability": "acc"
      }
    ]
  }
}
```

- Model output

```
{
  "data": {
    "resp_data": [
      {
        "predict_result": "unacc"
      },
      {
        "predict_result": "unacc"
      }
    ]
  }
}
```

- Configuration file

 NOTE

In the code, the **data** parameter in the request and response structures is described in JSON Schema. The content in **data** and **properties** corresponds to the model input and output.

```
{
  "model_type": "TensorFlow",
  "model_algorithm": "predict_analysis",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [
    {
      "url": "/",
      "method": "post",
      "request": {
        "Content-type": "application/json",
        "data": {
          "type": "object",
          "properties": {
            "data": {
              "type": "object",
              "properties": {
                "req_data": {
                  "items": [
                    {
                      "type": "object",
                      "properties": {}
                    }
                  ],
                  "type": "array"
                }
              }
            }
          }
        }
      },
      "response": {
        "Content-type": "application/json",
        "data": {
          "type": "object",

```

```
"properties": {
  "data": {
    "type": "object",
    "properties": {
      "resp_data": {
        "type": "array",
        "items": [
          {
            "type": "object",
            "properties": {}
          }
        ]
      }
    }
  }
},
"dependencies": [
  {
    "installer": "pip",
    "packages": [
      {
        "restraint": "EXACT",
        "package_version": "1.15.0",
        "package_name": "numpy"
      },
      {
        "restraint": "EXACT",
        "package_version": "5.2.0",
        "package_name": "Pillow"
      }
    ]
  }
]
```

Example of a Custom Image Model Configuration File

The model input and output are similar to those in [Example of an Object Detection Model Configuration File](#).

- Here is an example of how to make a model prediction request for image files.

To upload files, click the file upload button on the inference page.

```
{
  "Content-type": "multipart/form-data",
  "data": {
    "type": "object",
    "properties": {
      "images": {
        "type": "file"
      }
    }
  }
}
```

- Here is an example of how to make a model prediction request for JSON data.

The input parameter is of string type. To enter prediction requests, use the text box on the inference page.

```
{
  "Content-type": "application/json",
```

```
"data": {  
  "type": "object",  
  "properties": {  
    "input": {  
      "type": "string"  
    }  
  }  
}
```

A complete request example is as follows:

```
{  
  "model_algorithm": "image_classification",  
  "model_type": "Image",  
  "metrics": {  
    "f1": 0.345294,  
    "accuracy": 0.462963,  
    "precision": 0.338977,  
    "recall": 0.351852  
  },  
  "apis": [{  
    "url": "/",  
    "method": "post",  
    "request": {  
      "Content-type": "multipart/form-data",  
      "data": {  
        "type": "object",  
        "properties": {  
          "images": {  
            "type": "file"  
          }  
        }  
      }  
    },  
    "response": {  
      "Content-type": "application/json",  
      "data": {  
        "type": "object",  
        "required": [  
          "predicted_label",  
          "scores"  
        ],  
        "properties": {  
          "predicted_label": {  
            "type": "string"  
          },  
          "scores": {  
            "type": "array",  
            "items": [{  
              "type": "array",  
              "minItems": 2,  
              "maxItems": 2,  
              "items": [{  
                "type": "string"  
              },  
              {  
                "type": "number"  
              }  
            ]  
            }  
          }  
        }  
      }  
    }  
  ]  
},  
}
```

Example of a Machine Learning Model Configuration File

The following uses XGBoost as an example:

- Model input

```
{
  "req_data": [
    {
      "sepal_length": 5,
      "sepal_width": 3.3,
      "petal_length": 1.4,
      "petal_width": 0.2
    },
    {
      "sepal_length": 5,
      "sepal_width": 2,
      "petal_length": 3.5,
      "petal_width": 1
    },
    {
      "sepal_length": 6,
      "sepal_width": 2.2,
      "petal_length": 5,
      "petal_width": 1.5
    }
  ]
}
```

- Model output

```
{
  "resp_data": [
    {
      "predict_result": "Iris-setosa"
    },
    {
      "predict_result": "Iris-versicolor"
    }
  ]
}
```

- Configuration file

```
{
  "model_type": "XGBoost",
  "model_algorithm": "xgboost_iris_test",
  "runtime": "python2.7",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [
    {
      "url": "/",
      "method": "post",
      "request": {
        "Content-type": "application/json",
        "data": {
          "type": "object",
          "properties": {
            "req_data": {
              "items": [
                {
                  "type": "object",
                  "properties": {}
                }
              ]
            },
            "type": "array"
          }
        }
      }
    }
  ]
}
```



```
    }
  }
},
"response": {
  "Content-type": "application/json",
  "data": {
    "type": "object",
    "properties": {
      "resp_data": {
        "type": "array",
        "items": [
          {
            "type": "object",
            "properties": {
              "predict_result": {}
            }
          }
        ]
      }
    }
  }
}
}
```

Example of the Model Configuration File Using a Custom Dependency Package

The following example defines the NumPy 1.16.4 dependency environment.

```
{
  "model_algorithm": "image_classification",
  "model_type": "TensorFlow",
  "runtime": "python3.6",
  "apis": [
    {
      "url": "/",
      "method": "post",
      "request": {
        "Content-type": "multipart/form-data",
        "data": {
          "type": "object",
          "properties": {
            "images": {
              "type": "file"
            }
          }
        }
      }
    }
  ],
  "response": {
    "Content-type": "application/json",
    "data": {
      "type": "object",
      "properties": {
        "mnist_result": {
          "type": "array",
          "item": [
            {
              "type": "string"
            }
          ]
        }
      }
    }
  }
}
```

```
    },  
    ],  
    "metrics": {  
        "f1": 0.124555,  
        "recall": 0.171875,  
        "precision": 0.00234938928519385,  
        "accuracy": 0.00746268656716417  
    },  
    "dependencies": [  
        {  
            "installer": "pip",  
            "packages": [  
                {  
                    "restraint": "EXACT",  
                    "package_version": "1.16.4",  
                    "package_name": "numpy"  
                }  
            ]  
        }  
    ]  
}
```

7.1.3.3 Specifications for Writing a Model Inference Code File

This section describes the general method of editing model inference code in ModelArts. This section also provides an inference code example for the TensorFlow engine and an example of customizing the inference logic in the inference script.

Due to the limitation of API Gateway, the duration of a single prediction in ModelArts cannot exceed 40s. The model inference code must be logically clear and concise for satisfactory inference performance.

Specifications for Writing Inference Code

1. In the model inference code file **customize_service.py**, add a child model class. This child model class inherits properties from its parent model class. For details about the import statements of different types of parent model classes, see [Table 7-16](#). The ModelArts environment has already configured the necessary Python packages for import statements, so you do not need to install them separately.

Table 7-16 Parent class and import statement of each model type

Model Type	Parent Class	Import Statement
TensorFlow	TfServingBaseService	from model_service.tf-serving_model_service import TfServingBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService

2. The following methods can be overridden.

Table 7-17 Methods to be overridden

Method	Description
<code>__init__(self, model_name, model_path)</code>	Initialization method, which is suitable for models created based on deep learning frameworks. Models and labels are loaded using this method. To implement model loading logic, override this method for PyTorch and Caffe-based models.
<code>__init__(self, model_path)</code>	Initialization method, which is suitable for models created based on machine learning frameworks. This method initializes the model path (self.model_path). In Spark_MLLib, this method also initializes SparkSession (self.spark).
<code>_preprocess(self, data)</code>	Preprocess method, which is called before an inference request and converts API request data into the model's expected input format.
<code>_inference(self, data)</code>	Inference request method. You are advised not to override this method, as it will replace the built-in inference process in ModelArts with your custom logic.
<code>_postprocess(self, data)</code>	Postprocess method, which is called after an inference request is complete and converts the model output to the API output.

NOTE

- You can override the preprocess and postprocess methods for preprocessing the API input and postprocessing the inference output.
 - Overriding the init method of the parent model class may cause a model to run abnormally.
3. The attribute that can be used is the local path to the model. The attribute name is **self.model_path**. Additionally, PySpark-based models can use **self.spark** to obtain the SparkSession object in **customize_service.py**.

NOTE

The inference code requires an absolute file path for reading files. You can obtain the local path to the model from the **self.model_path** attribute.

- When TensorFlow, Caffe, or MXNet is used, **self.model_path** indicates the path to the model file. The following provides an example:

```
# Reads the label.json file in the model directory.  
with open(os.path.join(self.model_path, 'label.json')) as f:  
    self.label = json.load(f)
```
- When PyTorch, Scikit_Learn, or PySpark is used, **self.model_path** indicates the path to the model file. The following provides an example:

```
# Reads the label.json file in the model directory.  
dir_path = os.path.dirname(os.path.realpath(self.model_path))  
with open(os.path.join(dir_path, 'label.json')) as f:  
    self.label = json.load(f)
```

4. The API accepts data in either **multipart/form-data** or **application/json** format for pre-processing, actual inference, and post-processing.

- **multipart/form-data** request

```
curl -X POST \  
<modelarts-inference-endpoint> \  
-F image1=@cat.jpg \  
-F images2=@horse.jpg
```

The input data is as follows:

```
[  
  {  
    "image1":{  
      "cat.jpg":"<cat.jpg file io>"  
    }  
  },  
  {  
    "image2":{  
      "horse.jpg":"<horse.jpg file io>"  
    }  
  }  
]
```

- **application/json** request

```
curl -X POST \  
<modelarts-inference-endpoint> \  
-d '{  
  "images":"base64 encode image"  
}'
```

The input data is **python dict**.

```
{  
  "images":"base64 encode image"  
}
```

TensorFlow Inference Script Example

The following is an example of TensorFlow MnistService.

- Inference code

```
from PIL import Image  
import numpy as np  
from model_service.tf-serving_model_service import TfServingBaseService  
  
class MnistService(TfServingBaseService):  
  
    def _preprocess(self, data):  
        preprocessed_data = {}  
  
        for k, v in data.items():  
            for file_name, file_content in v.items():  
                image1 = Image.open(file_content)  
                image1 = np.array(image1, dtype=np.float32)  
                image1.resize((1, 784))  
                preprocessed_data[k] = image1  
  
        return preprocessed_data  
  
    def _postprocess(self, data):  
  
        infer_output = {}  
  
        for output_name, result in data.items():  
            infer_output["mnist_result"] = result[0].index(max(result[0]))  
  
        return infer_output
```

- Request

```
curl -X POST \ Real-time service address \ -F images=@test.jpg
```

- Response

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

The preceding sample code resizes images imported to the user's form to adapt to the model input shape. The **32×32** image is read from the Pillow library and resized to **1×784** to match the model input. In subsequent processing, convert the model output into a list for the RESTful API to display.

Inference Script Example of Custom Inference Logic

Customize a dependency package in the configuration file by referring to [Example of the Model Configuration File Using a Custom Dependency Package](#). Then, use the following code example to load the model in **saved_model** format for inference.

NOTE

Python logging used by inference base images allows the display of only warning logs. To query INFO logs, set the log level to INFO in the code.

```
# -*- coding: utf-8 -*-
import json
import os
import threading
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tf-serving_model_service import TfServingBaseService
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model_inputs = {}
        self.model_outputs = {}

        # The label file can be loaded here and used in the post-processing function.
        # Directories for storing the label.txt file on OBS and in the model package

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)

        # Load the model in saved_model format in non-blocking mode to prevent blocking timeout.
        thread = threading.Thread(target=self.get_tf_sess)
        thread.start()

    def get_tf_sess(self):
        # Load the model in saved_model format.
        # The session will be reused. Do not use the with statement.
        sess = tf.Session(graph=tf.Graph())
        meta_graph_def = tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING],
self.model_path)
        signature_defs = meta_graph_def.signature_def
        self.sess = sess
        signature = []

        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)
        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logger.warning("signatures more than one, use serving_default signature")
            model_signature = tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
```

```
logger.info("model signature: %s", model_signature)

for signature_name in meta_graph_def.signature_def[model_signature].inputs:
    tensorinfo = meta_graph_def.signature_def[model_signature].inputs[signature_name]
    name = tensorinfo.name
    op = self.sess.graph.get_tensor_by_name(name)
    self.model_inputs[signature_name] = op

logger.info("model inputs: %s", self.model_inputs)

for signature_name in meta_graph_def.signature_def[model_signature].outputs:
    tensorinfo = meta_graph_def.signature_def[model_signature].outputs[signature_name]
    name = tensorinfo.name
    op = self.sess.graph.get_tensor_by_name(name)
    self.model_outputs[signature_name] = op

logger.info("model outputs: %s", self.model_outputs)

def _preprocess(self, data):
    # Two HTTPS request formats
    # 1. Request in form-data format: data = {"Request key value":{"File name":<File io>}}
    # 2. Request in JSON format: data = json.loads("JSON body passed in the API")
    preprocessed_data = {}

    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((1, 28, 28))
            preprocessed_data[k] = image1

    return preprocessed_data

def _inference(self, data):
    feed_dict = {}
    for k, v in data.items():
        if k not in self.model_inputs.keys():
            logger.error("input key %s is not in model inputs %s", k, list(self.model_inputs.keys()))
            raise Exception("input key %s is not in model inputs %s" % (k, list(self.model_inputs.keys())))
        feed_dict[self.model_inputs[k]] = v

    result = self.sess.run(self.model_outputs, feed_dict=feed_dict)
    logger.info('predict result : ' + str(result))
    return result

def _postprocess(self, data):
    infer_output = {"mnist_result": []}
    for output_name, results in data.items():

        for result in results:
            infer_output["mnist_result"].append(np.argmax(result))

    return infer_output

def __del__(self):
    self.sess.close()
```

 NOTE

To load multiple models or models that are not supported by ModelArts, specify the loading path using the `__init__` method. Example code:

```
# -*- coding: utf-8 -*-
import os
from model_service.tf-serving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        # Obtain the path to the model folder.
        root = os.path.dirname(os.path.abspath(__file__))
        # test.onnx is the name of the model file to be loaded and must be stored in the model folder.
        self.model_path = os.path.join(root, test.onnx)

        # Load multiple models, for example, test2.onnx.
        # self.model_path2 = os.path.join(root, test2.onnx)
```

7.1.3.4 Specifications for Using a Custom Engine to Create a Model

When using a custom engine to create a model, you can select your image stored in SWR as the engine and specify a file directory in OBS as the model package. In this way, bring-your-own images can be used to meet your dedicated requirements.

Before deploying such a model as a service, ModelArts downloads the SWR image to the cluster and starts the image as a container as the user whose UID is 1000 and GID is 100. Then, ModelArts downloads the OBS file to the **/home/mind/model** directory in the container and runs the boot command preset in the SWR image. ModelArts registers an inference API with API Gateway for you to access the service.

Specifications for Using a Custom Engine to Create a Model

To use a custom engine to create a model, ensure the SWR image, OBS model package, and file size comply with the following requirements:

- SWR image specifications
 - A common user named **ma-user** in group **ma-group** must be built in the SWR image. Additionally, the UID and GID of the user must be 1000 and 100, respectively. The following is the dockerfile command for the built-in user:
groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
 - Specify a command for starting the image. In the dockerfile, specify **cmd**. The following shows an example:
CMD sh /home/mind/run.sh

Customize the boot file **run.sh**. The following is an example:

```
#!/bin/bash

# User-defined script content
...

# run.sh calls app.py to start the server. For details about app.py, see "HTTPS Example".
python app.py
```

 NOTE

You can also customize the boot command for starting an image. Enter the customized command during model creation.

- The provided service can use the HTTPS/HTTP protocol and listening container port. Set the protocol and port number based on the inference API defined by the model. For details about the HTTPS protocol, see [HTTPS Example](#).
- (Optional) On port provided by the service for external access, enable health check with URL **/health**. (The health check URL must be **/health**.)
- OBS model package specifications
The name of the model package must be **model**. For details about the model package specifications, see [Model Package Specifications](#).
- File size specifications
When a public resource pool is used, the total size of the downloaded SWR image (not the compressed image displayed on the SWR page) and the OBS model package cannot exceed 30 GB.

HTTPS Example

Use Flask to start HTTPS. The following is an example of the web server code:

```
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))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

Debugging on a Local Computer

Perform the following operations on a local computer with Docker installed to check whether a custom engine complies with specifications:

1. Download the custom image, for example, **custom_engine:v1** to the local computer.
2. Copy the model package folder **model** to the local computer.
3. Run the following command in the same directory as the model package folder to start the service:

```
docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model custom_engine:v1
```

NOTE

This command is used for simulation only because the directory mounted to **-v** is assigned the root permission. In the cloud environment, after the model file is downloaded from OBS to **/home/mind/model**, the file owner will be changed to **ma-user**.

4. Start another terminal on the local computer and run the following command to obtain the expected inference result:

```
curl https://127.0.0.1:8080/${Request path to the inference service}
```

Deployment Example

The following section describes how to use a custom engine to create a model.

1. Create a model and view model details.

Log in to the ModelArts console, choose **Model Management** from the navigation pane and click **Create Model**. Configure the following parameters:

- **Meta Model Source**: Select **OBS**.
- **Meta Model**: Select a model package from OBS.
- **AI Engine**: Select **Custom**.
- **Engine Package**: Select a container image from SWR.

Retain default settings for other parameters.

Click **Create now**. Wait until the model status changes to **Normal**.

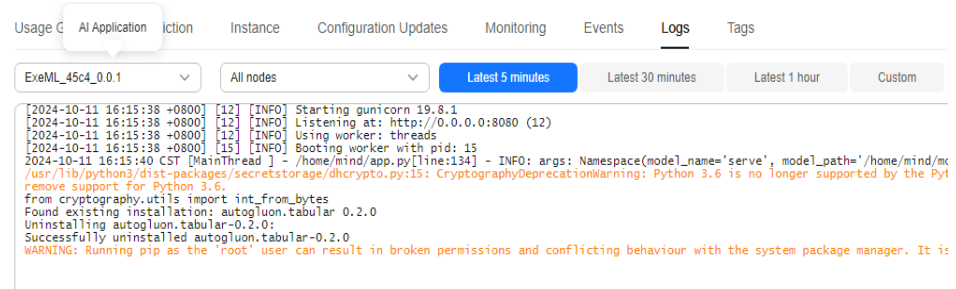
Click the model name to access its details page.

2. Deploy the AI application as a service and view service details.

On the model details page, choose **Deploy > Real-Time Services** in the upper right corner. On the **Deploy** page, select a proper instance flavor (for example, **CPU: 2 vCPUs 8 GB**), retain default settings for other parameters, and click **Next**. When the service status changes to **Running**, the service has been deployed.

Click the service name. On the displayed page, view the service details. Click the **Logs** tab to view the service logs.

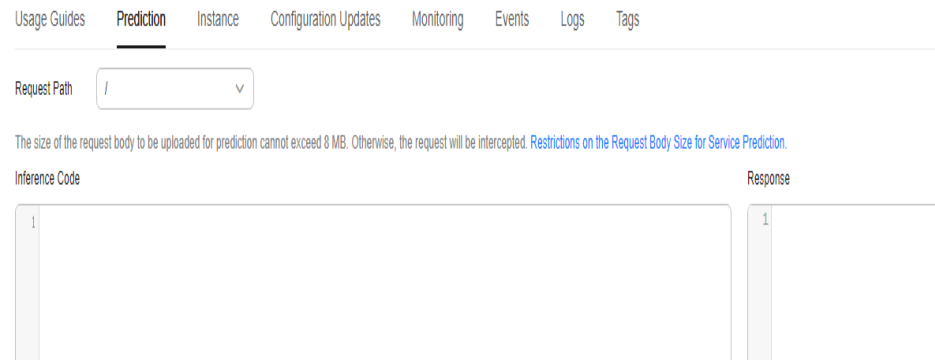
Figure 7-3 Service logs



3. Use the service for prediction.

On the service details page, click the **Prediction** tab to use the service for prediction.

Figure 7-4 Service prediction



7.1.3.5 Examples of Custom Scripts

To create a model in ModelArts by importing a model file from OBS, the model file package needs to comply with the ModelArts model package specifications. Additionally, the inference code and configuration file must also meet the requirements set by ModelArts.

This section provides custom script examples (including inference code examples) for common AI engines. For details about how to write model inference code, see [Specifications for Writing a Model Inference Code File](#).

Tensorflow

There are two types of TensorFlow APIs, Keras and tf. They use different code for training and saving models, but the same code for inference.

Training a Model (Keras API)

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense
import tensorflow as tf

# Import a training dataset.
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

print(x_train.shape)

from keras.layers import Dense
from keras.models import Sequential
import keras
from keras.layers import Dense, Activation, Flatten, Dropout

# Define a model network.
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(units=5120,activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(Dense(units=10, activation='softmax'))

# Define an optimizer and loss functions.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
# Train the model.
model.fit(x_train, y_train, epochs=2)
# Evaluate the model.
model.evaluate(x_test, y_test)
```

Saving a Model (Keras API)

```
from keras import backend as K

# K.get_session().run(tf.global_variables_initializer())

# Define the inputs and outputs of the prediction API.
# The keys of the inputs and outputs dictionaries are used as the index keys for the input and output
tensors of the model.
# The input and output definitions of the model must match the custom inference script.
predict_signature = tf.saved_model.signature_def_utils.predict_signature_def(
    inputs={"images" : model.input},
    outputs={"scores" : model.output}
)

# Define a save path.
builder = tf.saved_model.builder.SavedModelBuilder('./mnist_keras/')

builder.add_meta_graph_and_variables(

    sess = K.get_session(),
    # The tf.saved_model.tag_constants.SERVING tag needs to be defined for inference and deployment.
    tags=[tf.saved_model.tag_constants.SERVING],
    """
signature_def_map: Only one items can exist, or the corresponding key needs to be defined as follows:
    tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
    """
    signature_def_map={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            predict_signature
    }
)
builder.save()
```

Training a Model (tf API)

```
from __future__ import print_function

import gzip
import os
import urllib

import numpy
import tensorflow as tf
from six.moves import urllib

# Training data is obtained from the Yann LeCun official website http://yann.lecun.com/exdb/mnist/.
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
VALIDATION_SIZE = 5000

def maybe_download(filename, work_directory):
```

```
"""Download the data from Yann's website, unless it's already here."""
if not os.path.exists(work_directory):
    os.mkdir(work_directory)
filepath = os.path.join(work_directory, filename)
if not os.path.exists(filepath):
    filepath, _ = urllib.request.urlretrieve(SOURCE_URL + filename, filepath)
    statinfo = os.stat(filepath)
    print('Successfully downloaded %s %d bytes.' % (filename, statinfo.st_size))
return filepath

def _read32(bytestream):
    dt = numpy.dtype(numpy.uint32).newbyteorder('>')
    return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]

def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(
                'Invalid magic number %d in MNIST image file: %s' %
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
        data = data.reshape(num_images, rows, cols, 1)
        return data

def dense_to_one_hot(labels_dense, num_classes=10):
    """Convert class labels from scalars to one-hot vectors."""
    num_labels = labels_dense.shape[0]
    index_offset = numpy.arange(num_labels) * num_classes
    labels_one_hot = numpy.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot

def extract_labels(filename, one_hot=False):
    """Extract the labels into a 1D uint8 numpy array [index]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2049:
            raise ValueError(
                'Invalid magic number %d in MNIST label file: %s' %
                (magic, filename))
        num_items = _read32(bytestream)
        buf = bytestream.read(num_items)
        labels = numpy.frombuffer(buf, dtype=numpy.uint8)
        if one_hot:
            return dense_to_one_hot(labels)
        return labels

class DataSet(object):
    """Class encompassing test, validation and training MNIST data set."""

    def __init__(self, images, labels, fake_data=False, one_hot=False):
        """Construct a DataSet. one_hot arg is used only if fake_data is true."""

        if fake_data:
            self.num_examples = 10000
            self.one_hot = one_hot
```

```
else:
    assert images.shape[0] == labels.shape[0], (
        'images.shape: %s labels.shape: %s' % (images.shape,
                                                labels.shape))
    self._num_examples = images.shape[0]

    # Convert shape from [num examples, rows, columns, depth]
    # to [num examples, rows*columns] (assuming depth == 1)
    assert images.shape[3] == 1
    images = images.reshape(images.shape[0],
                             images.shape[1] * images.shape[2])
    # Convert from [0, 255] -> [0.0, 1.0].
    images = images.astype(numpy.float32)
    images = numpy.multiply(images, 1.0 / 255.0)
    self._images = images
    self._labels = labels
    self._epochs_completed = 0
    self._index_in_epoch = 0

@property
def images(self):
    return self._images

@property
def labels(self):
    return self._labels

@property
def num_examples(self):
    return self._num_examples

@property
def epochs_completed(self):
    return self._epochs_completed

def next_batch(self, batch_size, fake_data=False):
    """Return the next `batch_size` examples from this data set."""
    if fake_data:
        fake_image = [1] * 784
        if self.one_hot:
            fake_label = [1] + [0] * 9
        else:
            fake_label = 0
        return [fake_image for _ in range(batch_size)], [
            fake_label for _ in range(batch_size)
        ]
    start = self._index_in_epoch
    self._index_in_epoch += batch_size
    if self._index_in_epoch > self._num_examples:
        # Finished epoch
        self._epochs_completed += 1
        # Shuffle the data
        perm = numpy.arange(self._num_examples)
        numpy.random.shuffle(perm)
        self._images = self._images[perm]
        self._labels = self._labels[perm]
        # Start next epoch
        start = 0
        self._index_in_epoch = batch_size
        assert batch_size <= self._num_examples
    end = self._index_in_epoch
    return self._images[start:end], self._labels[start:end]

def read_data_sets(train_dir, fake_data=False, one_hot=False):
    """Return training, validation and testing data sets."""

    class DataSets(object):
        pass
```

```
data_sets = DataSets()

if fake_data:
    data_sets.train = DataSet([], [], fake_data=True, one_hot=one_hot)
    data_sets.validation = DataSet([], [], fake_data=True, one_hot=one_hot)
    data_sets.test = DataSet([], [], fake_data=True, one_hot=one_hot)
    return data_sets

local_file = maybe_download(TRAIN_IMAGES, train_dir)
train_images = extract_images(local_file)

local_file = maybe_download(TRAIN_LABELS, train_dir)
train_labels = extract_labels(local_file, one_hot=one_hot)

local_file = maybe_download(TEST_IMAGES, train_dir)
test_images = extract_images(local_file)

local_file = maybe_download(TEST_LABELS, train_dir)
test_labels = extract_labels(local_file, one_hot=one_hot)

validation_images = train_images[:VALIDATION_SIZE]
validation_labels = train_labels[:VALIDATION_SIZE]
train_images = train_images[VALIDATION_SIZE:]
train_labels = train_labels[VALIDATION_SIZE:]

data_sets.train = DataSet(train_images, train_labels)
data_sets.validation = DataSet(validation_images, validation_labels)
data_sets.test = DataSet(test_images, test_labels)
return data_sets

training_iteration = 1000

modelarts_example_path = './modelarts-mnist-train-save-deploy-example'

export_path = modelarts_example_path + '/model/'
data_path = './'

print('Training model...')
mnist = read_data_sets(data_path, one_hot=True)
sess = tf.InteractiveSession()
serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32), }
tf_example = tf.parse_example(serialized_tf_example, feature_configs)
x = tf.identity(tf_example['x'], name='x') # use tf.identity() to assign name
y_ = tf.placeholder('float', shape=[None, 10])
w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
values, indices = tf.nn.top_k(y, 10)
table = tf.contrib.lookup.index_to_string_table_from_tensor(
    tf.constant([str(i) for i in range(10)]))
prediction_classes = table.lookup(tf.to_int64(indices))
for _ in range(training_iteration):
    batch = mnist.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
print('training accuracy %g' % sess.run(
    accuracy, feed_dict={
        x: mnist.train.images,
        y_: mnist.train.labels
    }))
print('Done training!')
```

Saving a Model (tf API)

```
# Export the model.
# The model needs to be saved using the saved_model API.
print('Exporting trained model to', export_path)
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
tensor_info_y = tf.saved_model.utils.build_tensor_info(y)

# Define the inputs and outputs of the prediction API.
# The keys of the inputs and outputs dictionaries are used as the index keys for the input and output
tensors of the model.
# The input and output definitions of the model must match the custom inference script.
prediction_signature = (
    tf.saved_model.signature_def_utils.build_signature_def(
        inputs={'images': tensor_info_x},
        outputs={'scores': tensor_info_y},
        method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
builder.add_meta_graph_and_variables(
    # Set tag to serve/tf.saved_model.tag_constants.SERVING.
    sess, [tf.saved_model.tag_constants.SERVING],
    signature_def_map={
        'predict_images':
            prediction_signature,
    },
    legacy_init_op=legacy_init_op)

builder.save()

print('Done exporting!')
```

Inference Code (Keras and tf APIs)

In the model inference code file **customize_service.py**, add a child model class which inherits properties from its parent model class. For details about the parent class and import statement of each model type, see [Table 7-16](#). This example calls the parent class inference request method **_inference(self, data)**. The method does not need to be overridden in the following code.

```
from PIL import Image
import numpy as np
from model_service.tf-serving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):

    # Match the model input with the user's HTTPS API input during preprocessing.
    # The model input corresponding to the preceding training part is {"images":<array>}.
    def _preprocess(self, data):

        preprocessed_data = {}
        images = []
        # Iterate the input data.
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1,784))
                images.append(image1)
        # Return the numpy array.
        images = np.array(images,dtype=np.float32)
        # Perform batch processing on multiple input samples and ensure that the shape is the same as that
inputted during training.
        images.resize((len(data), 784))
        preprocessed_data['images'] = images
        return preprocessed_data

    # The output corresponding to model saving in the preceding training part is {"scores":<array>}.
```

```
# Postprocess the HTTPS output.
def _postprocess(self, data):
    infer_output = {"mnist_result": []}
    # Iterate the model output.
    for output_name, results in data.items():
        for result in results:
            infer_output["mnist_result"].append(result.index(max(result)))
    return infer_output
```

Tensorflow2.1

Training and Saving a Model

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
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.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # Name the output layer output, which is used to obtain the result during model inference.
    tf.keras.layers.Dense(10, activation='softmax', name="output")
])

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

model.fit(x_train, y_train, epochs=10)

tf.keras.models.save_model(model, "./mnist")
```

Inference Code

In the model inference code file **customize_service.py**, add a child model class which inherits properties from its parent model class. For details about the parent class and import statement of each model type, see [Table 7-16](#).

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tf_serving_model_service import TfServingBaseService

logger = logging.getLogger()
logger.setLevel(logging.INFO)

class MnistService(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

    # The label file can be loaded here and used in the post-processing function.
    # Directories for storing the label.txt file on OBS and in the model package
```



```
# with open(os.path.join(self.model_path, 'label.txt')) as f:
#     self.label = json.load(f)
# Load the model in saved_model format in non-blocking mode to prevent blocking timeout.
thread = threading.Thread(target=self.load_model)
thread.start()

def load_model(self):
    # Load the model in saved_model format.
    self.model = tf.saved_model.load(self.model_path)

    signature_defs = self.model.signatures.keys()

    signature = []
    # only one signature allowed
    for signature_def in signature_defs:
        signature.append(signature_def)

    if len(signature) == 1:
        model_signature = signature[0]
    else:
        logging.warning("signatures more than one, use serving_default signature from %s", signature)
        model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

    self.predict = self.model.signatures[model_signature]

def _preprocess(self, data):
    images = []
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((28, 28, 1))
            images.append(image1)

    images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
    preprocessed_data = images

    return preprocessed_data

def _inference(self, data):
    return self.predict(data)

def _postprocess(self, data):
    return {
        "result": int(data["output"].numpy()[0].argmax())
    }
```

Pytorch

Training a Model

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

# Define a network structure.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # The second dimension of the input must be 784.
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)
```

```
def forward(self, x):
    x = x.view(x.size()[0], -1)
    x = F.relu((self.hidden1(x)))
    x = F.dropout(x, 0.2)
    x = self.output(x)
    return F.log_softmax(x)

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

device = torch.device("cpu")

batch_size=64

kwargs={}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=True, download=True,
        transform=transforms.Compose([
            transforms.ToTensor()
        ])),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=False, transform=transforms.Compose([
        transforms.ToTensor()
    ])),
    batch_size=1000, shuffle=True, **kwargs)

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
optimizer = optim.Adam(model.parameters())

for epoch in range(1, 2 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

Saving a Model

```
# The model must be saved using state_dict and can be deployed remotely.
torch.save(model.state_dict(), "pytorch_mnist/mnist_mlp.pt")
```

Inference Code

In the model inference code file **customize_service.py**, add a child model class which inherits properties from its parent model class. For details about the parent class and import statement of each model type, see [Table 7-16](#).

```
from PIL import Image
import log
from model_service.pytorch_model_service import PTServingBaseService
import torch.nn.functional as F

import torch.nn as nn
import torch
import json

import numpy as np

logger = log.getLogger(__name__)

import torchvision.transforms as transforms

# Define model preprocessing.
infer_transformation = transforms.Compose([
    transforms.Resize((28,28)),
    # Convert data to tensor.
    transforms.ToTensor()
])

import os

class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # Call the constructor of the parent class.
        super(PTVisionService, self).__init__(model_name, model_path)
        # Call the custom function to load the model.
        self.model = Mnist(model_path)
        # Load labels.
        self.label = [0,1,2,3,4,5,6,7,8,9]
        # Labels can also be loaded by label file.
        # Reads the label.json file in the model directory.
        dir_path = os.path.dirname(os.path.realpath(self.model_path))
        with open(os.path.join(dir_path, 'label.json')) as f:
            self.label = json.load(f)

    def _preprocess(self, data):

        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # Gray processing
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    def _postprocess(self, data):
        results = []
```

```
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)

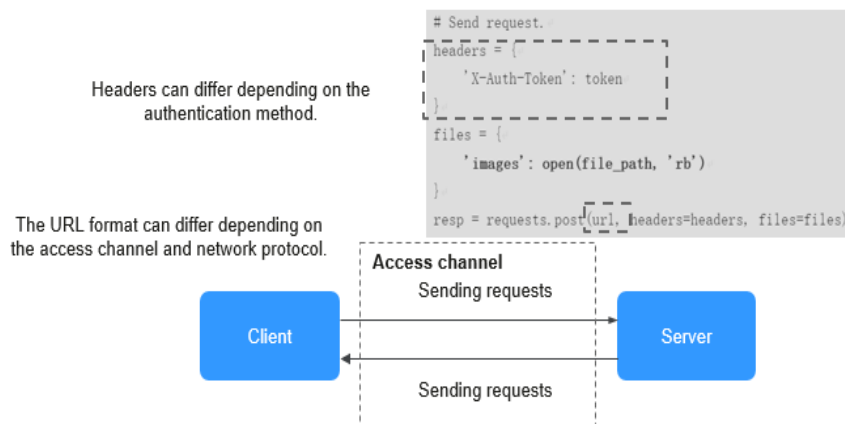
def Mnist(model_path, **kwargs):
    # Generate a network.
    model = Net()
    # Load the model.
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))
    # CPU or GPU mapping
    model.to(device)
    # Set the model to evaluation mode.
    model.eval()

    return model
```

7.1.4 Deploying a Model as Real-Time Inference Jobs

7.1.4.1 Deploying and Using Real-Time Inference

After creating a model, you can deploy it as a real-time service. If a real-time service is in the Running status, it has been deployed. This service provides a standard, callable RESTful API. When accessing a real-time service, you can choose the authentication method, access channel, and transmission protocol that best suit your needs. These three elements make up your access requests and can be mixed and matched without any interference. For example, you can use different authentication methods for different access channels and transmission protocols.

Figure 7-5 Authentication method, access channel, and transmission protocol

ModelArts supports the following authentication methods for accessing real-time services (HTTPS requests are used as examples):

- **Token-based authentication:** The validity period of a token is 24 hours. When using a token for authentication, cache it to prevent frequent calls.

ModelArts allows you to call APIs to access real-time services in the following ways (HTTPS requests are used as examples):

- **Accessing a Real-Time Service Through a Public Network:** By default, ModelArts inference uses the public network to access real-time services. A standard, callable RESTful API is provided after deployment of a real-time service.
- **Accessing a Real-Time Service Through a VPC High-Speed Channel:** When using VPC peering for high-speed access, your service requests are sent directly to instances via VPC peering, bypassing the inference platform. This results in faster service access.

Real-time service APIs are accessed using HTTPS by default. Additionally, the following transmission protocols are also supported:

- **Accessing a Real-Time Service Using Server-Sent Events:** Server-Sent Events (SSE) primarily facilitates unidirectional real-time communication from the server to the client, such as streaming ChatGPT responses. In contrast to WebSockets, which provide bidirectional real-time communication, SSE is designed to be more lightweight and simpler to implement.

7.1.4.2 Deploying a Model as a Real-Time Service

After a model is ready, deploy it as a real-time service. Then, you can call the service for prediction.

Prerequisites

- A ModelArts model in the **Normal** state is available.
- To deploy a service, the image owner must have the **te_admin** permission. Otherwise, the error message "failed to set image shared, please check the agency permission" will be displayed during service deployment.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **Service Deployment > Real-Time Services**.
2. In the real-time service list, click **Deploy** in the upper left corner.
3. Configure parameters.
 - a. Configure basic parameters. For details, see [Table 7-18](#).

Table 7-18 Basic parameters

Parameter	Description
Name	Name of a real-time service.
Description	Brief description for a real-time service.

- b. Enter key information including the resource pool and model configurations. For details, see [Table 7-19](#).

Table 7-19 Parameters

Parameter	Sub-Parameter	Description
Resource Pool	Public Resource Pool	CPU/GPU resource pools are available for you to select.
	Dedicated Resource Pool	Select a dedicated resource pool flavor. The physical pools with logical subpools created are not supported temporarily.
Model and Configuration	Model Source	Choose My Model as needed.
	Model and Version	Select the model and version that are in the Normal state.
	Traffic Ratio (%)	Data proportion of the current AI application version. Service calling requests are allocated to the current version based on this proportion. If you deploy only one version of a model, set this parameter to 100 . If you select multiple versions for gray release, ensure that the sum of the traffic ratios of these versions is 100% .

Parameter	Sub-Parameter	Description
	Instance Flavor	Select available flavors based on the list displayed on the console. The flavors in gray cannot be used in the current environment. If no public resource pool flavors are available, In this case, use a dedicated resource pool. NOTE When deploying the service with the selected flavor, there will be necessary system consumptions. This means that the actual resources required will be greater than the flavor.
	Instances	Number of instances for the current model version. If you set the number of instances to 1, the standalone computing mode is used. If you set the number of instances to a value greater than 1, the distributed computing mode is used. Select a computing mode based on your actual needs.
	Timeout	Timeout of a single model, including both the deployment and startup time. The default value is 20 minutes. The value must range from 3 to 120.
	Add Model and Configuration	If the selected model has multiple versions, you can add multiple versions and configure a traffic ratio. You can use gray release to smoothly upgrade the model version. NOTE Free compute specifications do not support gray release of multiple versions.

Parameter	Sub-Parameter	Description
	Mount Storage	This parameter is displayed when the resource pool is a dedicated resource pool. This feature will mount a storage volume to compute nodes (instances) as a local directory when the service is running. This is a good option to consider when dealing with large input data or models. OBS Bucket • Source Path: Select an OBS bucket path. Cross-region OBS buckets cannot be selected. You can add up to 10 paths. • Mount Path: Enter the container mount path, for example, /obs-mount/. It is good practice to create a directory. Avoid using inventory directories or system directories with strict permissions. OBS parallel file system • Source Path: Select a storage path. A cross-region OBS parallel file system cannot be selected. • Mount Path: Enter the container mount path, for example, /obs-mount/. – Select a new directory. If you select an existing directory, existing files will be overwritten. OBS mounting allows you to add, view, and modify files in the mount directory but does not allow you to delete files in the mount directory. To delete files, manually delete them in the OBS parallel file system. – Mount an empty directory to the container. If the directory is not empty, ensure that the directory does not contain any files that affect container startup. Otherwise, the files will be replaced, and the container cannot start normally. As a result, the workload may not be deployed. – The mount path must start with a slash (/) and can contain a maximum of 1,024 characters, including letters, digits, and the following special characters: _. SFS Turbo • File System Name: Select the target SFS Turbo file system. A cross-region SFS Turbo file system cannot be selected.

Parameter	Sub-Parameter	Description
		• Mount Path: Enter the mount path of the container, for example, <code>/sfs-turbo-mount/</code>. Select a new directory. If you select an existing directory, any existing files within it will be replaced. NOTE • A file system can be mounted only once and to only one path. Each mount path must be unique. A maximum of 8 disks can be mounted to a training job. • Storage mounting is allowed only for services deployed in a dedicated resource pool which has interconnected with a VPC or associated with SFS Turbo. - To interconnect a VPC is to interconnect the VPC where SFS Turbo belongs to a dedicated resource pool network. For details, see Interconnect with a VPC. - You can associate HPC SFS Turbo file systems with dedicated resource pool networks. • If you need to mount multiple file systems, do not use same or similar paths, for example, <code>/obs-mount/</code> and <code>/obs-mount/tmp/</code>. • Once you have chosen SFS Turbo, avoid deleting the interconnected VPC or disassociating SFS Turbo. Otherwise, mounting will not be possible. When you mount the backend OBS storage on the SFS Turbo page, make sure to set the client's umask permission to 777 for normal use.

4. After confirming the entered information, deploy the service as prompted. Deploying a service generally requires a period of time, which may be several minutes or tens of minutes depending on the amount of your data and resources.

 NOTE

Once a real-time service is deployed, it will start immediately.

You can go to the real-time service list to check if the deployment is complete. Once the service status changes from **Deploying** to **Running**, the service is deployed.

Testing Real-Time Service Prediction

After a model is deployed as a real-time service, you can debug code or add files for testing in the **Prediction** tab. You can test the service in two ways, depending on the input request defined by the model – either by using a JSON text or a file.

- **JSON text prediction:** If the input of the deployed model is JSON text, you can enter JSON code in the **Prediction** tab for testing.

- **File Prediction:** If your model uses files as input, you can add images, audios, or videos into the **Prediction** tab to test the service.

 NOTE

- The size of an input image must be less than 8 MB.
- The maximum size of a request body for JSON text prediction is 8 MB.
- Due to the limitation of API Gateway, the duration of a single prediction cannot exceed 40s.
- The following image types are supported: png, psd, jpg, jpeg, bmp, gif, webp, psd, svg, and tiff.
- If you use Ascend flavors for service deployment, you cannot predict transparent .png images because Ascend only supports RGB-3 images.
- This feature is used for commissioning. Use API calling for actual production. You can select [Accessing a Real-Time Service Through Token-based Authentication](#) based on the authentication method.

After a service is deployed, obtain the input parameters of the service in the **Usage Guides** page of the service details page.

The input parameters displayed in the **Usage Guides** tab depend on the model source that you select.

- If you use a custom meta model and your own inference code and configuration file (see [Specifications for Writing the Model Configuration File](#)), the **Usage Guides** tab will only display your configuration file. The following figure shows the mapping between the input parameters in the **Usage Guides** tab and the configuration file.

The prediction methods for different input requests are as follows:

- **JSON Text Prediction**
 - a. Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**.
 - b. Click the name of the target service to access its details page. Enter the inference code in the **Prediction** tab, and click **Predict** to perform prediction.
- **File Prediction**
 - a. Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**.
 - b. Click the name of the target service to access its details page. In the **Prediction** tab, click **Upload** and select a test file. After the file is uploaded, click **Predict** to perform a prediction test.

7.1.4.3 Authentication Methods for Accessing Real-time Services

7.1.4.3.1 Accessing a Real-Time Service Through Token-based Authentication

If a real-time service is in the **Running** state, it has been deployed successfully. This service provides a standard RESTful API for users to call. Before integrating the API to the production environment, commission the API. You can use the following methods to send an inference request to the real-time service:

- **Method 1: Use GUI-based Software for Inference (Postman).** (Postman is recommended for Windows.)
- **Method 2: Run the cURL Command to Send an Inference Request.** (curl commands are recommended for Linux.)
- **Method 3: Use Python to Send an Inference Request.**

Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Prerequisites

You have obtained a user token, local path to the inference file, URL of the real-time service, and input parameters of the real-time service.

- For details about how to obtain a user token, see [Token-based Authentication](#). The real-time service APIs generated by ModelArts do not support tokens whose scope is domain. Therefore, you need to obtain the token whose scope is project.
- The local path to the inference file can be an absolute path (for example, **D:/test.png** for Windows and **/opt/data/test.png** for Linux) or a relative path (for example, **./test.png**).
- You can obtain the service URL and input parameters of a real-time service on the Usage Guides tab page of its service details page.

The API URL is the service URL of the real-time service. If a path is defined for **apis** in the model configuration file, the URL must be followed by the user-defined path, for example, **{URL of the real-time service}/predictions/poetry**.

Method 1: Use GUI-based Software for Inference (Postman)

1. Download Postman and install it, or install the Postman Chrome extension. Alternatively, use other software that can send POST requests. Postman 7.24.0 is recommended.
2. Open Postman.
3. Set parameters on Postman. The following uses image classification as an example.
 - Select a POST task and copy the API URL to the POST text box. On the **Headers** tab page, set **Key** to **X-Auth-Token** and **Value** to the user token.
 - In the **Body** tab, file input and text input are available.
 - **File input**
Select **form-data**. Set **KEY** to the input parameter of the model, which must be the same as the input parameter of the real-time

service. In this example, the **KEY** is **images**. Set **VALUE** to an image to be inferred (only one image can be inferred).

■ Text input

Select **raw** and then **JSON(application/json)**. Enter the request body in the text box below. An example request body is as follows:

```
{
  "meta": {
    "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
  },
  "data": {
    "req_data": [
      {
        "sepal_length": 3,
        "sepal_width": 1,
        "petal_length": 2.2,
        "petal_width": 4
      }
    ]
  }
}
```

meta can carry a universally unique identifier (UUID). When you call an API, the system provides a UUID. When the inference result is returned, the UUID is returned to trace the request. If you do not need this function, leave **meta** blank. **data** contains a **req_data** array for one or multiple pieces of input data. The parameters of each piece of data are determined by the model, such as **sepal_length** and **sepal_width** in this example.

4. After setting the parameters, click **send** to send the request. The result will be displayed in **Response**.
 - Inference result using file input: The field values in the return result vary with the model.
 - Inference result using text input: The request body contains **meta** and **data**. If the request contains **uuid**, **uuid** will be returned in the response. Otherwise, **uuid** is left blank. **data** contains a **resp_data** array for the inference results of one or multiple pieces of input data. The parameters of each result are determined by the model, for example, **sepal_length** and **predictresult** in this example.

Method 2: Run the cURL Command to Send an Inference Request

The command for sending inference requests can be input as a file or text.

- File input

```
curl -kv -F 'images=@Image path' -H 'X-Auth-Token:Token value' -X POST Real-time service URL
```

- **-k** indicates that SSL websites can be accessed without using a security certificate.
- **-F** indicates file input. In this example, the parameter name is **images**, which can be changed as required. The image storage path follows **@**.
- **-H** indicates the header of a POST command. **X-Auth-Token** is the header key, which is fixed. *Token value* indicates the user token.
- **POST** is followed by the API URL of the real-time service.

The following is an example of the cURL command for inference with file input:

```
curl -kv -F 'images=@/home/data/test.png' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
```

- **Text input**

```
curl -kv -d '{"data":{"req_data":  
[{"sepal_length":3,"sepal_width":1,"petal_length":2.2,"petal_width":4}]}}' -H 'X-Auth-  
Token:MIISkAY***80T9wHQ==' -H 'Content-type: application/json' -X POST https://modelarts-  
infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
```

-d indicates the text input of the request body.

Method 3: Use Python to Send an Inference Request

1. Download the Python SDK and configure it in the development tool.
2. Create a request body for inference.

- **File input**

```
# coding=utf-8  
  
import requests  
  
if __name__ == '__main__':  
    # Config url, token and file path.  
    url = "URL of the real-time service"  
    token = "User token"  
    file_path = "Local path to the inference file"  
  
    # Send request.  
    headers = {  
        'X-Auth-Token': token  
    }  
    files = {  
        'images': open(file_path, 'rb')  
    }  
    resp = requests.post(url, headers=headers, files=files)  
  
    # Print result.  
    print(resp.status_code)  
    print(resp.text)
```

The **files** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the file type.

- **Text input (JSON)**

The following is an example of the request body for reading the local inference file and performing Base64 encoding:

```
# coding=utf-8  
  
import base64  
import requests  
  
if __name__ == '__main__':  
    # Config url, token and file path  
    url = "URL of the real-time service"  
    token = "User token"  
    file_path = "Local path to the inference file"  
    with open(file_path, "rb") as file:  
        base64_data = base64.b64encode(file.read()).decode("utf-8")  
  
    # Set body,then send request  
    headers = {  
        'Content-Type': 'application/json',  
        'X-Auth-Token': token  
    }  
    body = {  
        'image': base64_data  
    }
```

```
}  
resp = requests.post(url, headers=headers, json=body)  
  
# Print result  
print(resp.status_code)  
print(resp.text)
```

The **body** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the string type. The value of **base64_data** in **body** is of the string type.

7.1.4.3.2 Accessing a Real-Time Service Through App Authentication

You can enable application authentication when deploying a real-time service. ModelArts registers an API that supports application authentication for the service. After this API is authorized to an application, you can call this API using the AppKey/AppSecret or AppCode of the application.

The process of application authentication for a real-time service is as follows:

1. **Enabling Application Authentication:** Enable application authentication. You can select or create an application.
2. **Managing Authorization for Real-Time Services:** Manage your applications, including viewing, resetting, or deleting them, as well as binding or unbinding real-time services and obtaining your AppKey and AppSecret, or AppCode.
3. **Application Authentication:** To call an API that supports app authentication, you will need to authenticate first. There are two authentication methods: AppKey and AppSecret, or AppCode. You can choose the one that suits you best.
4. Send an inference request.
 - **Method 1: Use Python to Send an Inference request Through AppKey/AppSecret-based Authentication**
 - **Method 2: Use Java to Send an Inference request Through AppKey/AppSecret-based Authentication**
 - **Method 3: Use Python to Send an Inference request Through AppCode-based Authentication**
 - **Method 4: Use Java to Send an Inference request Through AppCode-based Authentication**

Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Prerequisites

- A ModelArts model in the **Normal** state is available.
- The account is not in arrears to ensure available resources for service running.

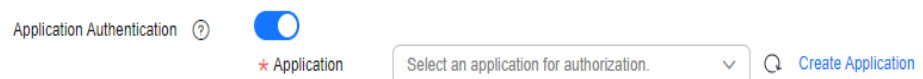
- The local path to the inference file has been obtained. The path can be an absolute path (for example, **D:/test.png** for Windows and **/opt/data/test.png** for Linux) or a relative path (for example, **./test.png**).

Enabling Application Authentication

When deploying a real-time service, you can enable application authentication. You can also modify a deployed real-time service to support application authentication.

1. Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**.
2. Enable application authentication.
 - When deploying a real-time service, enable application authentication on the **Deploy** page.
 - For a deployed real-time service, go to the **Real-Time Services** page, and click **Modify** in the **Operation** column of the service. On the service modification page, enable application authentication.

Figure 7-6 Enabling application authentication



3. Select an application for authorization from the drop-down list. If no application is available, follow these steps to create one:
 - Click **Create Application**, enter the application name and description, and click **OK**. By default, the application name is prefixed with **app_**. You can change this name if needed.
 - On the **Service Deployment > Real-Time Services** page, click **Authorize**. On the **Manage Authorization of Real-Time Services** page, click **Create Application**. For details, see [Managing Authorization for Real-Time Services](#).
4. After enabling application authentication, authorize a service that supports application authentication to the application. Then, you can use the created AppKey/AppSecret or AppCode to call the service's API that supports application authentication.

Managing Authorization for Real-Time Services

If you want to use application authentication, it is good practice to create an application on the authorization management page before deploying a real-time service. In the navigation pane, choose **Service Deployment > Real-Time Services**. On the **Real-Time Services** page, click **Authorize**. From there, you can create, reset, or delete applications, query plaintext, unbind real-time services from applications, and obtain the AppKey/AppSecret.

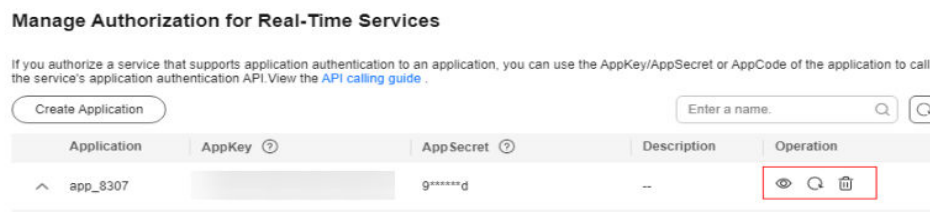
- **Creating an application**

Click **Create Application**, enter the application name and description, and click **OK**. By default, the application name is prefixed with **app_**. You can change this name if needed.

- **Viewing, resetting, or deleting an application**

Query plaintext, reset, or delete an application by clicking the corresponding icon in the **Operation** column of the application. After an application is created, the AppKey and AppSecret are automatically generated for application authentication.

Figure 7-7 Query Plaintext, Reset, or Delete



- **Unbinding a service**

Click  next to the target application name to view the real-time services bound to the application. Click **Unbind** in the **Operation** column to cancel the binding. Then, this API cannot be called.

- **Obtaining the AppKey/AppSecret**

Application authentication is required for API calling. The AppKey and AppSecret are automatically generated during application creation. Click  in the **Operation** column of the application in the **Manage Authorization of Real-Time Services** dialog box to view the complete AppSecret.

Application Authentication

When a real-time service that supports application authentication is in the **Running** state, the service' API can be called. Before calling the API, perform application authentication.

When you use application authentication and enable simplified authentication, you can use your AppKey/AppSecret for signing and verification, or AppCode for simplified authentication. ModelArts uses simplified authentication by default. AppKey/AppSecret-based authentication is recommended because it is more secure than AppCode-based authentication.

- **AppKey/AppSecret-based authentication:** The AppKey and AppSecret are used to encrypt a request, identify the sender, and prevent the request from being modified. When using AppKey/AppSecret-based authentication, use a dedicated signing SDK to sign requests.
 - AppKey: access key ID of the application, which is a unique identifier used together with a secret access key to sign requests cryptographically.
 - AppSecret: application secret access key, used together with the access key ID to encrypt the request, identify the sender, and prevent the request from being tempered.

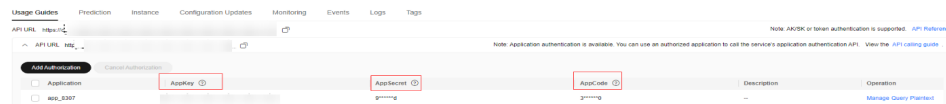
AppKeys can be used for simplified authentication. When an API is called, the **apikey** parameter (value: **AppKey**) is added to the HTTP request header to accelerate authentication.

- **AppCode-based authentication:** Requests are authenticated using AppCodes. In AppCode-based authentication, the **X-Apig-AppCode** parameter (value: **AppCode**) is added to the HTTP request header when an API is called. The

request content does not need to be signed. The API gateway only verifies the AppCode, achieving quick response.

You can obtain the API URL (**url** of the real-time service), AppKey/AppSecret (**app_key** and **app_secret**), and AppCode (**app_code**) from the **Usage Guides** tab on the service details page. Use the API URL for application authentication in the second line of the figure.

Figure 7-8 Obtaining application authentication information



Method 1: Use Python to Send an Inference request Through AppKey/AppSecret-based Authentication

1. Download the Python SDK and configure it in the development tool.
2. Create a request body for inference.

– File input

```
# coding=utf-8

import requests
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
    # f2682*****f42"
    url = "URL of the real-time service"
    # Hardcoded or plaintext app_key and app_secret are risky. For security, encrypt and
    # store them in the configuration file or environment variables.
    # In this example, the app_key and app_secret are stored in environment variables for
    # identity authentication. Before running this example, set environment variables
    # HUAWEICLOUD_APP_KEY and HUAWEICLOUD_APP_SECRET.
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret = os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "Local path to the inference file"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri,
        headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

Request body format of **files**: files = {"Request parameter": ("Load path", File content, "File type")}. For details about parameters of **files**, see [Table 7-20](#).

Table 7-20 Parameters of files

Parameter	Mandatory	Description
Request parameter	Yes	Parameter name of the real-time service.
File path	No	Path for storing the file.
File content	Yes	Content of the file to be uploaded.
File type	No	Type of the file to be uploaded, which can be one of the following options: • txt: text/plain • jpg/jpeg: image/jpeg • png: image/png

– **Text input (JSON)**

The following is an example request body for reading the local inference file and performing Base64 encoding:

```
# coding=utf-8

import base64
import json
import os
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
    # f2682*****f42"
    url = "URL of the real-time service"
    # Hardcoded or plaintext app_key and app_secret are risky. For security, encrypt and
    # store them in the configuration file or environment variables.
    # In this example, the app_key and app_secret are stored in environment variables for
    # identity authentication. Before running this example, set environment variables
    # HUAWEICLOUD_APP_KEY and HUAWEICLOUD_APP_SECRET.
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret = os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "Local path to the inference file"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AppKey&AppSecret to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
```

```
sig.Sign(request)

# Send request
resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri,
headers=request.headers, data=request.body)

# Print result
print(resp.status_code)
print(resp.text)
```

The **body** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the string type. **image** is used as an example. The value of **base64_data** in **body** is of the string type.

Method 2: Use Java to Send an Inference request Through AppKey/AppSecret-based Authentication

1. Download the Java SDK and configure it in the development tool.
2. Create a Java request body for inference.

In the APIG Java SDK, **request.setBody()** can only be a string. Therefore, only text inference requests are supported.

The following is an example of the request body (JSON) for reading the local inference file and performing Base64 encoding:

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
f2682*****f42"
        String url = "URL of the real-time service";
        // Hard-coded or plaintext app_key and app_secret are risky. For security, encrypt and store
them in the configuration file or environment variables.
        // In this example, the app_key and app_secret are stored in environment variables for
identity authentication. Before running this example, set environment variables
HUAWEICLOUD_APP_KEY and HUAWEICLOUD_APP_SECRET.
        String appKey = System.getenv("HUAWEICLOUD_APP_KEY");
        String appSecret = System.getenv("HUAWEICLOUD_APP_SECRET");
        String body = "{}";

        try {
            // Create request
            Request request = new Request();

            // Set the AK/AppSecret to sign and authenticate the request.
            request.setKey(appKey);
            request.setSecret(appSecret);

            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");
```

```
// Set a request URL in the format of https://{Endpoint}/{URI}.
request.setUrl(url);

// Special characters, such as the double quotation mark ("), contained in the body must be
escaped.
request.setBody(body);

// Sign the request.
HttpRequestBase signedRequest = Client.sign(request);

// Send request.
CloseableHttpResponse response = HttpClientUtils.createDefault().execute(signedRequest);

// Print result
System.out.println(response.getStatusLine().getStatusCode());
System.out.println(EntityUtils.toString(response.getEntity()));
} catch (Exception e) {
    e.printStackTrace();
}
}
```

body is determined by the text format. JSON is used as an example.

Method 3: Use Python to Send an Inference request Through AppCode-based Authentication

1. Download the Python SDK and configure it in the development tool.
2. Create a request body for inference.

– **File input**

```
# coding=utf-8

import requests
import os

if __name__ == '__main__':
    # Config url, app code and file path.
    # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
    f2682*****f42"
    url = "URL of the real-time service"
    # Hardcoded or plaintext app_code is risky. For security, encrypt and store it in the
    configuration file or environment variables.
    # In this example, the app_code is stored in environment variables for identity
    authentication. Before running this example, set environment variable
    HUAWEICLOUD_APP_CODE.
    app_code = os.environ["HUAWEICLOUD_APP_CODE"]
    file_path = "Local path to the inference file"

    # Send request.
    headers = {
        'X-Apig-AppCode': app_code
    }
    files = {
        'images': open(file_path, 'rb')
    }
    resp = requests.post(url, headers=headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

The **files** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the file type. In this example, **images** is used.

– Text input (JSON)

The following is an example request body for reading the local inference file and performing Base64 encoding:

```
# coding=utf-8

import base64
import requests
import os

if __name__ == '__main__':
    # Config url, app code and request body.
    # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
    # f2682*****f42"
    url = "URL of the real-time service"
    # Hardcoded or plaintext app_code is risky. For security, encrypt and store it in the
    # configuration file or environment variables.
    # In this example, the app_code is stored in environment variables for identity
    # authentication. Before running this example, set environment variable
    # HUAWEICLOUD_APP_CODE.
    app_code = os.environ["HUAWEICLOUD_APP_CODE"]
    file_path = "Local path to the inference file"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Send request
    headers = {
        'Content-Type': 'application/json',
        'X-Apig-AppCode': app_code
    }
    body = {
        'image': base64_data
    }
    resp = requests.post(url, headers=headers, json=body)

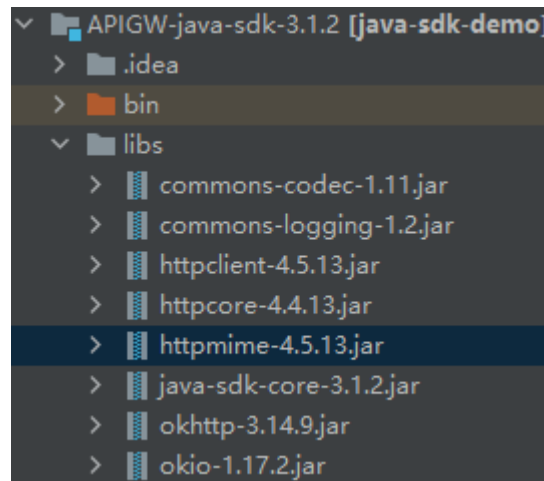
    # Print result
    print(resp.status_code)
    print(resp.text)
```

The **body** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the string type. **image** is used as an example. The value of **base64_data** in **body** is of the string type.

Method 4: Use Java to Send an Inference request Through AppCode-based Authentication

1. Download the Java SDK and configure it in the development tool.
2. (Optional) If the inference request input is in a file format, follow these steps to ensure the Java project includes the httpmime module as a dependency.
 - a. Add **httpmime-x.x.x.jar** to the **libs** folder. **Figure 7-9** shows a complete Java dependency library.

You are advised to use httpmime-x.x.x.jar 4.5 or later. Download httpmime-x.x.x.jar from <https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime>.

Figure 7-9 Java dependency library

- b. After **httpmime-x.x.x.jar** is added, add httpmime information to the **.classpath** file of the Java project as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. Create a Java request body for inference.

– **File input**

A sample Java request body is as follows:

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.File;

public class MyAppCodeFile {

    public static void main(String[] args) {
        # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
f2682*****f42"
        String url = "URL of the real-time service";
        // Hard-coded or plaintext appCode is risky. For security, encrypt and store it in the
configuration file or environment variables.
        // In this example, the appCode is stored in environment variables for identity
authentication. Before running this example, set environment variable
HUAWEICLOUD_APP_CODE.
        String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
        String filePath = "Local path to the inference file";
```

```
try {
    // Create post
    HttpPost httpPost = new HttpPost(url);

    // Add header parameters
    httpPost.setHeader("X-Apig-AppCode", appCode);

    // Special characters, such as the double quotation mark ("), contained in the body
    must be escaped.
    File file = new File(filePath);
    HttpEntity entity = MultipartEntityBuilder.create().addBinaryBody("images",
file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.UTF_8).build();
    httpPost.setEntity(entity);

    // Send post
    CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

    // Print result
    System.out.println(response.getStatusLine().getStatusCode());
    System.out.println(EntityUtils.toString(response.getEntity()));
} catch (Exception e) {
    e.printStackTrace();
}
}
```

The **addBinaryBody** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the file type. In this example, **images** is used.

– Text input (JSON)

The following is an example request body for reading the local inference file and performing Base64 encoding:

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAppCodeTest {

    public static void main(String[] args) {
        # API URL, for example, "https://8e*****5fe.apig.*****.huaweicloudapis.com/v1/infers/
f2682*****f42"
        String url = "URL of the real-time service";
        // Hard-coded or plaintext appCode is risky. For security, encrypt and store it in the
        configuration file or environment variables.
        // In this example, the appCode is stored in environment variables for identity
        authentication. Before running this example, set environment variable
        HUAWEICLOUD_APP_CODE.
        String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            httpPost.setHeader("X-Apig-AppCode", appCode);

            // Special characters, such as the double quotation mark ("), contained in the body
            must be escaped.
```

```
httpPost.setEntity(new StringEntity(body));

// Send post
CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

// Print result
System.out.println(response.getStatusLine().getStatusCode());
System.out.println(EntityUtils.toString(response.getEntity()));
} catch (Exception e) {
    e.printStackTrace();
}
}
```

body is determined by the text format. JSON is used as an example.

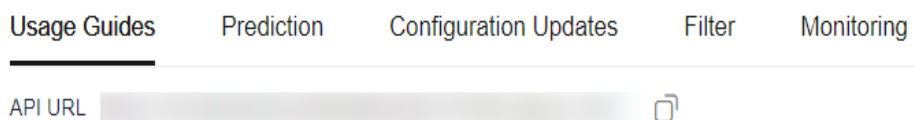
7.1.4.4 Accessing a Real-Time Service Through Different Channels

7.1.4.4.1 Accessing a Real-Time Service Through a Public Network

Context

By default, ModelArts inference uses the public network to access real-time services. After a real-time service is deployed, a standard RESTful API is provided for you to call. You can view the API URL on the **Usage Guides** tab page of the service details page.

Figure 7-10 API URL



Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Accessing a Real-Time Service

The following authentication modes are available for accessing real-time services from a public network:

- [Accessing a Real-Time Service Through Token-based Authentication](#)
- [Accessing a Real-Time Service Through App Authentication](#)

7.1.4.4.2 Accessing a Real-Time Service Through a VPC High-Speed Channel

Context

When accessing a real-time service, you may require:

- High throughput and low latency
- TCP or RPC requests

To meet these requirements, ModelArts enables high-speed access through VPC peering.

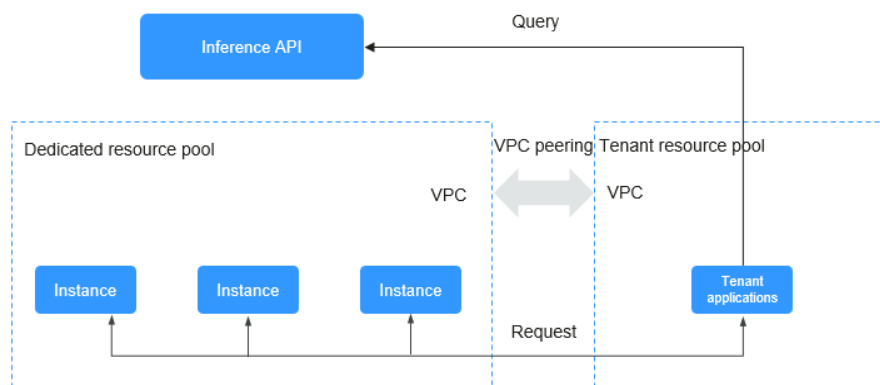
In high-speed access through VPC peering, your service requests are directly sent to instances through VPC peering but not through the inference platform. This accelerates service access.

NOTE

The following features that are available through the inference platform will be unavailable if you use high-speed access:

- Authentication
- Traffic distribution by configuration
- Load balancing
- Alarm, monitoring, and statistics

Figure 7-11 High-speed access through VPC peering



Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Preparations

Deploy a real-time service in a dedicated resource pool and ensure the service is running.

NOTICE

- Only the services deployed in a dedicated resource pool support high-speed access through VPC peering.
- High-speed access through VPC peering is available only for real-time services.
- Due to traffic control, there is a limit on how often you can get the IP address and port number of a real-time service. The number of calls of each tenant account cannot exceed 2000 per minute, and that of each IAM user account cannot exceed 20 per minute.
- High-speed access through VPC peering is available only for the services deployed using the AI applications imported from custom images.

Procedure

To enable high-speed access to a real-time service through VPC peering, perform the following operations:

1. [Interconnect the dedicated resource pool to the VPC.](#)
2. [Create an ECS in the VPC.](#)
3. [Obtain the IP address and port number of the real-time service.](#)
4. [Access the service through the IP address and port number.](#)

Step 1 Interconnect the dedicated resource pool to the VPC.

Log in to the ModelArts console, choose **AI Dedicated Resource Pools > Elastic Clusters**, locate the dedicated resource pool where the service is deployed, and click its name/ID to go to the resource pool details page. Obtain the network configuration. Switch back to the dedicated resource pool list, click the **Network** tab, locate the network associated with the dedicated resource pool, and interconnect it with the VPC. After the VPC is accessed, the VPC will be displayed on the network list and resource pool details pages. Click the VPC to go to the details page.

Figure 7-12 Obtaining the network configuration

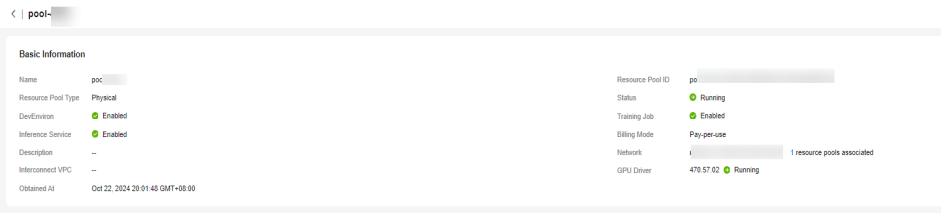


Figure 7-13 Interconnecting the VPC

Resource Pools							
Network							
Nodes							
Create A maximum of 15 network can be created. You can create 0 more.							
Network Name	Status	CIDR Block	Interconnect VPC	Associated sfsturno	IPv6	Obtained At	Operation
network-37f5 network-37f5-04f2	Active	192	vpc- / subnet-11ed	-	Disabled	Apr 28, 2022 16:58:55 GMT+0	Interconnect VPC

Step 2 Create an ECS in the VPC.

Log in to the ECS management console and click **Buy ECS** in the upper right corner. On the **Buy ECS** page, configure basic settings and click **Next: Configure Network**. On the **Configure Network** page, select the VPC connected in **Step 1**, configure other parameters, confirm the settings, and click **Submit**. When the ECS status changes to **Running**, the ECS has been created. Click its name/ID to go to the server details page and view the VPC configuration.

Figure 7-14 Selecting a VPC when purchasing an ECS

Network

VPC ?

Create VPC

Primary NIC

subnet-

Automatically assign IP address

Available private IP addresses: 250

+

Add Extension NIC

NICs you can still add: 1

Source/Destination Check ?

Figure 7-15 Viewing VPC information

< ecs	
Summary	
Disks	
Network Interfaces	
Security Groups	
EIPs	
Monitoring	
ECS Information	
ID	61dfb
Name	ecs 2
Description	--
Region	CN-Hong Kong
AZ	AZ7
Specifications	General computing-plus 2 vCPUs 4 GiB c7n.large.2
Image	Huawei Cloud EulerOS 2.0 Standard 64 bit Public image
VPC	vpc-
Billing Mode	Pay-per-use
Created	Oct 12, 2024 11:23:57 GMT+08:00
Launched	Oct 12, 2024 11:24:05 GMT+08:00
Deletion Time	-- Modify

Step 3 Obtain the IP address and port number of the real-time service.

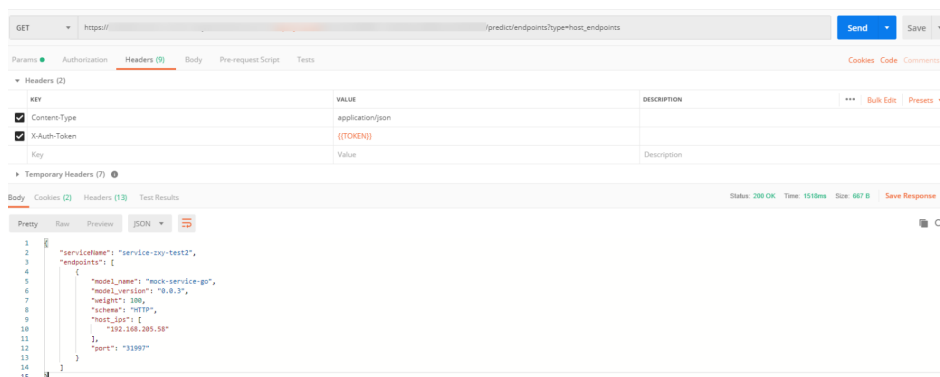
GUI software, for example, Postman can be used to obtain the IP address and port number. Alternatively, log in to the ECS, create a Python environment, and execute code to obtain the service IP address and port number.

API:

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- Method 1: Obtain the IP address and port number using GUI software.

Figure 7-16 Example response



- Method 2: Obtain the IP address and port number using Python.
The following parameters in the Python code below need to be modified:
- **project_id**: your project ID. To obtain it, see [Obtaining a Project ID and Name](#).
- **service_id**: service ID, which can be viewed on the service details page.
- **REGION_ENDPOINT**: service endpoint.

```
def get_app_info(project_id, service_id):
    list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
    headers = {'X-Auth-Token': X_Auth-Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

Step 4 Access the service through the IP address and port number.

Log in to the ECS and access the real-time service either by running Linux commands or by creating a Python environment and executing Python code. Obtain the values of **schema**, **ip**, and **port** from [Step 3](#).

- Run the following command to access the real-time service:

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type: application/json' \
--data-raw '{"a":"a"}'
```

Figure 7-17 Accessing a real-time service

```
[root@ecs-zxy ~]# curl --location --request POST 'http://192.168.205.58:31997' \
> --header 'Content-Type: application/json' \
> --data-raw '{"a":"a"}'
call Post()[root@ecs-zxy ~]# _
```

- Create a Python environment and execute Python code to access the real-time service.

```
def vpc_infer(schema, ip, port, body):  
    infer_url = "{}://{}:{}".format(schema, ip, port)  
    url = infer_url.format(schema, ip, port)  
    response = requests.post(url, data=body)  
    print(response.content)
```

NOTE

High-speed access does not support load balancing. You need to customize load balancing policies when you deploy multiple instances.

----End

7.1.4.5 Accessing a Real-Time Service Using Different Protocols

7.1.4.5.1 Accessing a Real-Time Service Using Server-Sent Events

Context

Server-Sent Events (SSE) is a server push technology enabling a server to push events to a client via an HTTP connection. This technology is usually used to enable a server to push real-time data to a client, for example, a chat application or a real-time news update.

SSE primarily facilitates unidirectional real-time communication from the server to the client, such as streaming ChatGPT responses. In contrast to WebSockets, which provide bidirectional real-time communication, SSE is designed to be more lightweight and simpler to implement.

Prerequisites

The image for importing the model is SSE-compliant.

Constraints

- SSE supports only the deployment of real-time services.
- It supports only real-time services deployed using models imported from custom images.
- When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:
 - The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
 - Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Calling an SSE Real-Time Service

The SSE protocol itself does not introduce new authentication mechanisms; it relies on the same methods as HTTP requests.

The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call an SSE service.

Figure 7-18 Calling an SSE service

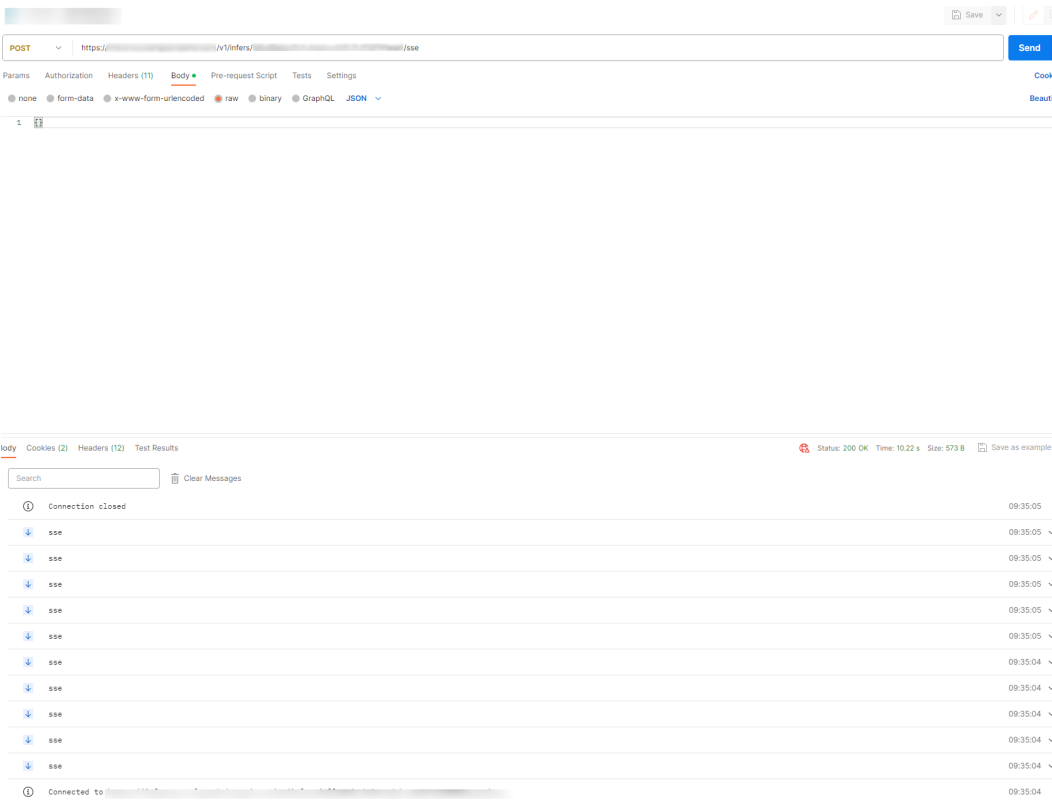


Figure 7-19 Response header Content-Type

Body Cookies Headers Test Results	
KEY	VALUE
Content-Type ⓘ	text/event-stream;charset=UTF-8

 NOTE

In normal cases, the value of **Content-Type** in the response header is **text/event-stream;charset=UTF-8**.

7.1.5 Deploying a Model as a Batch Inference Service

After a model is prepared, you can deploy it as a batch service. The **Service Deployment > Batch Services** page lists all batch services.

Prerequisites

- A ModelArts model in the **Normal** state is available.
- Data to be processed in batches has been uploaded to OBS.
- At least one empty folder has been created in OBS for storing the output.

Context

- You can create up to 1,000 batch services.
- Based on the input request (JSON or file) defined by the model, different parameters are entered. If the model input is a JSON file, a configuration file

is required to generate a mapping file. If the model input is a file, no mapping file is required.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **Service Deployment > Batch Services**.
2. Click **Deploy** in the upper left corner.
3. Configure parameters.
 - a. Enter basic information, including **Name** and **Description**. A name is generated by default, for example, **service-bc0d**, which you can modify.
 - b. Configure other parameters, including model configurations.

Table 7-21 Parameters

Parameter	Description
Resource Pool	Public Resource Pool CPU and GPU public resource pools are available for you to select. To use a public resource pool, contact the administrator to create one.
	Dedicated Resource Pool Select a specification from the resource pool specifications.
Model Source	Choose My Model as needed.
Model and Version	Select the model and version that are in the Normal state.
Input Path	Select the OBS directory where the uploaded data is stored. Select a folder or a .manifest file. For details about the specifications of the .manifest file, see Manifest File Specifications . NOTE • If the input data is an image, ensure that the size of a single image is less than 12 MB. • If the input data is in CSV format, ensure that no Chinese character is included. • If the input data is in CSV format, ensure that the file size does not exceed 12 MB. • If an image or CSV file is larger than 12 MB, an error is reported. In this case, resize the file or contact technical support to adjust the file size limit.
Request Path	URL used for calling the model API in a batch service, and also the request path of the model service. Its value is obtained from the url field of apis in the model configuration file.

Parameter	Description
Mapping Relationship	If the model input is in JSON format, the system automatically generates the mapping based on the configuration file corresponding to the model. If the model input is other file, mapping is not required. The mapping file is generated automatically. Enter the field index corresponding to each parameter in the CSV file. The index starts from 0. Mapping rule: The mapping rule comes from the input parameter (request) in the model configuration file config.json. When type is set to string, number, integer, or boolean, you are required to set the index parameter. For details about the mapping rule, see Mapping Example. The index must be a positive integer starting from 0. If the value of index does not comply with the rule, this parameter is ignored in the request. After the mapping rule is configured, the CSV data must be separated by commas (,).
Output Path	The path for storing the batch prediction results. You can select an empty folder you created.
Instance Flavor	The system provides available compute resources matching your model. Select an available resource from the drop-down list. For example, if the model comes from an ExeML project, the compute resources are automatically associated with the ExeML specifications for use.
Instances	Number of instances for the current model version. If you set the number of nodes to 1 , the standalone computing mode is used. If you set the number of nodes to a value greater than 1, the distributed computing mode is used. Select a computing mode based on your actual needs.
Environment Variable	Set environment variables and inject them to the pod. To ensure data security, do not enter sensitive information in environment variables.
Timeout	Timeout of a single model, including both the deployment and startup time. The default value is 20 minutes. The value must range from 3 to 120.

4. Confirm the configurations and complete service deployment as prompted. Deploying a service generally requires a period of time, which may be several minutes or tens of minutes depending on the amount of your data and resources.

 NOTE

Once a batch service is deployed, it will start immediately. You will be billed for the chosen resources while it is running.

You can go to the batch service list to view the basic information about the batch service. In the batch service list, after the status of the newly deployed service changes from **Deploying** to **Running**, the service is deployed.

Manifest File Specifications

ModelArts batch services support manifest files, which describe data input and output.

Example input manifest file

- File name: **test.manifest**
- File content:

```
{"source": "obs://test/data/1.jpg"}
{"source": "s3://test/data/2.jpg"}
{"source": "https://infern-data.obs.example.com:443/xgboostdata/data.csv?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve
8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
ZaMSxHvl68kKLAY5feYvLDM..."}
```
- Requirements on the file:
 - a. The file name extension must be **.manifest**.
 - b. The file content must be in JSON format. Each line describes a piece of input data, which must be a specific file instead of a folder.
 - c. JSON content requires a **source** field, which must be an OBS file address in either of the following formats:
 - i. Bucket path *<obs path>{{Bucket name}}/{{Object name}}/File name*, which is used to access your OBS data. You can obtain the path by accessing the OBS object. *<obs path>* can be **obs://** or **s3://**.
 - ii. Share link generated by OBS, including signature information. It applies to accessing OBS data of other users. The link has a validity period. Perform operations within the period.

Example output manifest file

A manifest file will be generated in the output directory of the batch service.

- Assume that the output path is **//test-bucket/test/**. The result is stored in the following path:

OBS bucket or directory name

```
test-bucket
├── test
│   ├── infer-result-{{task_id}}.manifest
│   ├── infer-result
│   │   ├── 1.jpg_result.txt
│   │   └── 2.jpg_result.txt
```

- Content of the **infer-result-0.manifest** file:

```
{"source": "obs://obs-data-bucket/test/data/1.jpg", "result": "SUCCESSFUL", "inference-loc": "obs://test-
bucket/test/infer-result/1.jpg_result.txt"}
{"source": "s3://obs-data-bucket/test/data/2.jpg", "result": "FAILED", "error_message": "Download file
failed."}
{"source": "https://infern-data.obs.example.com:443/xgboostdata/2.jpg?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve
8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
ZaMSxHvl68kKLAY5feYvLDMNZWxzhBZ6Q-3HcoZMh9glSwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9jmv
```

```
DGgT9VBC7dC1EyfSjrUcqfB_N0ykCsfrA1Tt_IQYZFDu_HyqVk-  
GunUcTVdDfWlCV3TrYcpmznZjliAnYUO89kAwCYGeRZsCsC0ePu4PHMsBvYV9gWmN9AUZIDn1sfRL4vo  
BpwQnp6tnAgHW49y5a6hP2hCAoQ-95SpUriJ434QlymoeKfTHVMKOEzXZea-  
JxOvevOCGI5CcGehEJaz48sgH81UiH2l21zocNB_hpPfus2jY6KPglEJxMv6Kwmro-  
ZBXWuSJUDOnSYXI-3ciYjg9-  
h10b8W3sW1mOTFCWNGoWsd74it7L_5-7UUholeyPByO_REwkur2FOJsuMpGIRaPyglZxXm_jfdLFXobYtz  
Zhb4yWXga6oxTOKfcwykTOYHONPoPrt5MYGYweOXXxfs3d5w2rd0y7p0QYhyTzlk5Clz7FlWNapFISL  
7zdhs18RfchTqESq94KgkeqatSF_ilvnYMW2r8P8x2k_eb6NJ7U_q5ztMbO9oWEcfr0D2f7n7Bl_nb2HIB_H9tj  
zKvgwngaimYhBbMRPfbvttW86GiwVP8vrC27FOn39Be9z2hSfj_8pHej0yMlyNqZ481FQ5vWT_vFV3JHM-  
7I1ZB0_hldaHfltm-J69cTFHSE0zt7DGaMIES1o7U3w%3D%3D","result":"SUCCESSFUL","inference-loc":  
"obs://test-bucket/test/infer-result/2.jpg_result.txt"]}
```

- File format:
 - a. The file name is **infer-result-{{task_id}}.manifest**, where **task_id** is the batch task ID, which is unique for a batch service.
 - b. If a large number of files need to be processed, multiple manifest files may be generated with the same suffix **.manifest** and are distinguished by suffix, for example, **infer-result-{{task_id}}_1.manifest**.
 - c. The **infer-result-{{task_id}}** directory is created in the manifest directory to store the file processing result.
 - d. The file content is in JSON format. Each line describes the output result of a piece of input data.
 - e. The JSON file contains multiple fields:
 - i. **source**: input data description, which is the same as that of the input manifest file
 - ii. **result**: file processing result, which can be **SUCCESSFUL** or **FAILED**
 - iii. **inference-loc**: output result path. This field is available when result is **SUCCESSFUL**. The format is **obs://{{Bucket name}}/{{Object name}}**.
 - iv. **error_message**: error information. This field is available when the result is **FAILED**.

Mapping Example

The following example shows the relationship between the configuration file, mapping rule, CSV data, and inference request.

The following uses a file for prediction as an example:

```
[  
{  
  "method": "post",  
  "url": "/",  
  "request": {  
    "Content-type": "multipart/form-data",  
    "data": {  
      "type": "object",  
      "properties": {  
        "data": {  
          "type": "object",  
          "properties": {  
            "req_data": {  
              "type": "array",  
              "items": [  
                {  
                  "type": "object",  
                  "properties": {  
                    "input_1": {  
                      "type": "number"  
                    }  
                  }  
                ]  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
}]
```

```
    "input_2": {
      "type": "number"
    },
    "input_3": {
      "type": "number"
    },
    "input_4": {
      "type": "number"
    }
  }
}
]
```

The ModelArts console automatically resolves the mapping relationship from the configuration file as shown below. When calling a ModelArts API, configure the mapping by following the rule.

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "req_data": {
          "type": "array",
          "items": [
            {
              "type": "object",
              "properties": {
                "input_1": {
                  "type": "number",
                  "index": 0
                },
                "input_2": {
                  "type": "number",
                  "index": 1
                },
                "input_3": {
                  "type": "number",
                  "index": 2
                },
                "input_4": {
                  "type": "number",
                  "index": 3
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

The following shows the format of the CSV data for inference. The data must be separated by commas (,).

```
5.1,3.5,1.4,0.2
4.9,3.0,1.4,0.2
4.7,3.2,1.3,0.2
```

Depending on the defined mapping relationship, the inference request is shown below, whose format is similar to that for real-time services.

```
{
  "data": {
    "req_data": [{
      "input_1": 5.1,
      "input_2": 3.5,
      "input_3": 1.4,
      "input_4": 0.2
    }]
  }
}
```

Viewing the Batch Service Prediction Result

When deploying a batch service, you can select the location of the output data directory. You can view the running result of the batch service that is in the **Completed** state.

Step 1 Log in to the ModelArts console and choose **Service Deployment > Batch Services**.

Step 2 Click the name of the target service in the **Completed** status. The service details page is displayed.

- You can view the service name, status, ID, input path, output path, and description.
- You can click  next to **Description** to edit the description.

Step 3 Obtain the detailed OBS path next to **Output Path**, switch to the path and obtain the batch service prediction results, including the prediction result file and the model prediction result.

If the prediction is successful, the directory contains the prediction result file and model prediction result. Otherwise, the directory contains only the prediction result file.

- Prediction result file: The file is in the *xxx.manifest* format and contains the file path and prediction result.
- Model prediction output:
 - If images are input, a result file is generated for each image in the *Image name_result.txt* format, for example, **IMG_20180919_115016.jpg_result.txt**.
 - If audio files are input, a result file is generated for each audio file in the *Audio file name__result.txt* format, for example, **1-36929-A-47.wav_result.txt**.
 - If table data is input, the result file is generated in the *Table name__result.txt* format, for example, **train.csv_result.txt**.

----End

7.1.6 Managing ModelArts Models

7.1.6.1 Viewing ModelArts Model Details

Viewing the Model List

You can view all created models on the model list page. The model list page displays the following information.

Table 7-22 Model list

Parameter	Description
Model Name	Model name.
Latest Version	Latest version of a model.
Status	Model status.
Deployment Type	Types of the services that a model can be deployed as.
Versions	Number of model versions.
Request Mode	Request mode of real-time services. • Synchronous request: one-off inference with results returned synchronously (within 60s). This mode is suitable for images and small video files. • Asynchronous request: one-off inference with results returned asynchronously (longer than 60s). This mode is suitable for real-time video inference and large videos.
Created	Model creation time.
Description	Model description.
Operation	• Deploy: Deploy a model as real-time services or batch services. • Create Version: Create a model version. The settings of the last version are used by default, except for the version. You can change the parameter settings. • Delete: Delete a model. NOTE If a model version has been deployed as a service, you must delete the associated service before deleting the model version. A deleted model cannot be recovered.

Click the number in **Versions** to view the version list.

The version list displays the following information.

Table 7-23 Version list

Parameter	Description
Version	Current version of a model.
Status	Model status.
Deployment Type	Types of the services that a model can be deployed as.
Model Size	Model size.
Model Source	Model source.
Created	Model creation time.
Description	Model description.
Operation	• Deploy: Deploy a model as real-time services or batch services. • Delete: Delete a version of a model.

Viewing Model Details

After a model is created, you can view the model information on the model details page.

1. Log in to the ModelArts console, and choose **Model Management** from the navigation pane.
2. Click the name of the target model to access its details page.

On the model details page, you can view the basic information and precision of the model, and switch tab pages to view more information.

Table 7-24 Basic model information

Parameter	Description
Name	Model name.
Status	Model status.
Version	Current version of a model.
ID	Model ID.
Description	Click the edit button to add the description of a model.
Deployment Type	Types of the services that a model can be deployed as.
Meta Model Source	Source of the meta model, which can be training jobs, OBS, or container images.
Training Name	Associated training job if the meta model comes from a training job. Click the training job name to go to its details page.

Parameter	Description
Training Version	Training job version if the meta model comes from an old-version training job.
Storage path of the meta model	Path to the meta model if the meta model comes from OBS.
Container Image Storage Path	Path to the container image if the meta model comes from a container image.
AI Engine	AI engine if the meta model comes from a training job or OBS.
Engine Package Address	Engine package address if the meta model comes from OBS and AI Engine is Custom .
Runtime Environment	Runtime environment on which the meta model depends if the meta model comes from a training job or OBS and a preset AI engine is used.
Container API	Protocol and port number for starting the model if the meta model comes from OBS (AI Engine is Custom) or a container image.
Inference Code	Path to the inference code if the meta model comes from an old-version training job.
Image Replication	Image replication status for meta models from a container image.
Dynamic loading	Dynamic loading status if the meta model comes from a training job or OBS.
Size	Model size.

Parameter	Description
Health Check	Displays health check status if the meta model comes from OBS or a container image. When health check is enabled, the probe parameter settings are displayed. • Startup Probe: This probe checks if the application instance has started. If a startup probe is provided, all other probes are disabled until it succeeds. If the startup probe fails, the instance is restarted. If no startup probe is provided, the default status is Success. • Readiness Probe: This probe verifies whether the application instance is ready to handle traffic. If the readiness probe fails (meaning the instance is not ready), the instance is taken out of the service load balancing pool. Traffic will not be routed to the instance until the probe succeeds. • Liveness Probe: This probe monitors the application health status. If the liveness probe fails (indicating the application is unhealthy), the instance is automatically restarted. The probe parameters include Check Mode, Health Check URL (displayed when Check Mode is set to HTTP request), Health Check Command (displayed when Check Mode is set to Command), Health Check Period, Delay, Timeout, and Maximum Failures.
Model Description	Description document added during the creation of a model.
Instruction Set Architecture	System architecture.
Inference Accelerator	Type of inference accelerator cards.

Table 7-25 Model details tabs

Parameter	Description
Model Precision	Model recall, precision, accuracy, and F1 score of a model.
Parameter Configuration	API configuration, input parameters, and output parameters of a model.
Runtime Dependency	Model dependency on the environment. If creating a job failed, edit the runtime dependency. After the modification is saved, the system will automatically use the original image to create the job again.

Parameter	Description
Events	The progress of key operations during model creation. Events are stored for three months and will be automatically cleared then. For details about how to view events of a model, see Viewing ModelArts Model Events .
Constraint	Displays the constraints of service deployment, such as the request mode, boot command, and model encryption, based on the settings during model creation. For models in asynchronous request mode, parameters including the input mode, output mode, service startup parameters, and job configuration parameters can be displayed.
Associated Services	The list of services that a model was deployed. Click a service name to go to the service details page.

7.1.6.2 Viewing ModelArts Model Events

During the creation of a model, every key event is automatically recorded. You can view the events on the details page of the model at any time.

The following table lists the events, based on which you can locate faults occurred during model creation. The following table lists the available events.

Type	Event (xxx should be replaced with the actual value.)	Solution
Normal	The model starts to import.	N/A
Abnormal	Failed to create the image.	Locate and rectify the fault based on the error information. FAQs
Abnormal	The custom image does not support specified dependencies.	The runtime dependencies cannot be configured when a custom image is imported. Install the pip dependency package in the Dockerfile that is used to create the image. FAQs

Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	Only custom images support swr_location .	Delete the swr_location field from the model configuration file config.json and try again.
Abnormal	The health check API of a custom image must be xxx.	Modify the health check API of the custom image and try again.
Normal	The image creation task is in the xxx state.	N/A
Abnormal	Label xxx does not exist in image xxx.	Contact technical support.
Abnormal	Invalid parameter value xxx exists in the model configuration file.	Delete invalid parameters from the model configuration file and try again.
Abnormal	Failed to obtain the labels of image xxx.	Contact technical support.
Abnormal	Failed to import data because xxx is larger than xxx GB.	The size of the model or image exceeds the upper limit. Downsize the model or image and import it again. FAQs
Abnormal	User xxx does not have OBS permission obs:object:PutObjectAcl.	The IAM user does not have the obs:object:PutObjectAcl permission on OBS. Add the agency permission for the IAM user. FAQs
Abnormal	Creating the image timed out. The timeout duration is xxx minutes.	There is a timeout limit for image building using ImagePacker. Simplify the code to improve efficiency. FAQs
Normal	Model description updated.	N/A

Type	Event (xxx should be replaced with the actual value.)	Solution
Normal	Model runtime dependencies not updated.	N/A
Normal	Model runtime dependencies updated. Recreating the image.	N/A
Abnormal	SWR traffic control triggered. Try again later.	SWR traffic control triggered. Try again later.
Normal	The system is being upgraded. Try again later.	N/A
Abnormal	Failed to obtain the source image. An error occurred in authentication. The token has expired.	Contact technical support.
Abnormal	Failed to obtain the source image. Check whether the image exists.	Contact technical support.
Normal	Source image size calculated.	N/A
Normal	Source image shared.	N/A
Abnormal	Failed to create the image due to traffic control. Try again later.	Traffic control triggered. Try again later.
Abnormal	Failed to send the image creation request.	Contact technical support.
Abnormal	Failed to share the source image. Check whether the image exists or whether you have the permission to share the image.	Check whether the image exists or whether you have the permission to share the image.
Normal	The model imported.	N/A
Normal	Model file imported.	N/A
Normal	Model size calculated.	N/A
Abnormal	Failed to import the model.	For details about how to locate and rectify the fault, see FAQs .
Abnormal	Failed to copy the model file. Check whether you have the OBS permission.	Check whether you have the OBS permission. FAQs
Abnormal	Failed to schedule the image creation task.	Contact technical support.

Type	Event (<i>xxx</i> should be replaced with the actual value.)	Solution
Abnormal	Failed to start the image creation task.	Contact technical support.
Abnormal	The Roman image has been created but cannot be shared with resource tenants.	Contact technical support.
Normal	Image created.	N/A
Normal	The image creation task started.	N/A
Normal	The environment image creation task started.	N/A
Normal	The request for creating an environment image received.	N/A
Normal	The request for creating an image received.	N/A
Normal	An existing environment image is used.	N/A
Abnormal	Failed to create the image. For details, see image creation logs.	View the build logs to locate and rectify the fault. FAQs
Abnormal	Failed to create the image due to an internal system error. Contact technical support.	Contact technical support.
Abnormal	Failed to import model file <i>xxx</i> because it is larger than 5 GB.	The size of the model file <i>xxx</i> is greater than 5 GB. Downsize the model file and try again, or use dynamic loading to import the model file. FAQs
Abnormal	Failed to create the OBS bucket due to an internal system error. Contact technical support.	Contact technical support.
Abnormal	Failed to calculate the model size. Subpath <i>xxx</i> does not exist in path <i>xxx</i> .	Correct the subpath and try again, or contact technical support.

Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	Failed to calculate the model size. The model of the xxx type does not exist in path xxx.	Check the storage location of the model of the xxx type, correct the path, and try again, or contact technical support.
Warning	Failed to calculate the model size. More than one xxx model file is stored in path xxx.	N/A

During model creation, key events can both be manually and automatically refreshed.

Viewing Events

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Model Management**. In the model list, click the name of the target model to go to its details page.
2. View the events in the **Events** tab.

7.1.6.3 Managing ModelArts Model Versions

For model lineage and tuning, ModelArts provides model versioning.

Prerequisites

You have created a model in ModelArts.

Creating a Version

On the **Model Management** page, click **Create Version** in the **Operation** column of the target model. On the **Create Version** page, configure parameters. For details, see [Creating a Model](#). Click **Create now**.

Deleting a Version

On the **Model Management** page, click the number in **Versions**. Click **Delete** in the **Operation** column of the target version.

NOTE

If a model version has been deployed as a service, you must delete the associated service before deleting the model version. A deleted version cannot be recovered.

Deleting a Model

On the **Model Management** page, click **Delete** in the **Operation** column of the target model to delete it.

NOTE

If a model version has been deployed as a service, you must delete the associated service before deleting the model version. A deleted model cannot be recovered.

7.1.7 Managing a Synchronous Real-Time Service

7.1.7.1 Viewing Details About a Real-Time Service

After a model is deployed as a real-time service, you can access the service page to view its details.

1. Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**.
2. Click the name of the target service to access its details page.

View service information. For details, see [Table 7-26](#).

Table 7-26 real-time service parameters

Parameter	Description
Name	Name of the real-time service.
Status	Status of the real-time service.
Source	AI application source of the real-time service.
Service ID	Real-time service ID.
Description	Service description, which you can click the edit button to modify.
Resource Pool	Resource pool specifications used by the service. If the public resource pool is used for deployment, this parameter is not displayed.
Custom Settings	Customized configurations based on real-time service versions. This allows version-based traffic distribution policies and configurations. Enable this option and click View Settings to customize the settings. For details, see Modifying Customized Settings .

Parameter	Description
Runtime Log Output	This feature is disabled by default. The runtime logs of real-time services are stored only in the ModelArts log system. If this feature is enabled, the runtime logs of real-time services will be exported and stored in Log Tank Service (LTS). LTS automatically creates log groups and log streams and caches run logs generated within seven days by default. NOTE • This cannot be disabled once it is enabled. • Do not print unnecessary audio log files. Otherwise, system logs may fail to be displayed, and the error message "Failed to load audio" may be displayed.
WebSocket	Whether to upgrade to the WebSocket service.

3. Switch between tabs on the details page of a real-time service to view more details. For details, see [Table 7-27](#).

Table 7-27 Details of a real-time service

Parameter	Description
Usage Guides	This page displays the API URL, model information, input parameters, and output parameters. You can click  to copy the API URL to call the service.
Prediction	You can perform real-time prediction on this page. For details, see Testing Real-Time Service Prediction .
Instance	Instance information of the asynchronous real-time service. The number of instances is the same as the number of instances set during service deployment. If the service is modified or the service is abnormal, the number of instances changes. To rebuild an abnormal instance, click Delete . After the instance is deleted, a new instance with the same compute specifications is automatically created.
Configuration Updates	This page displays Current Configurations and Update History. • Current Configurations: model name, version, status, instance flavor, traffic ratio, number of instances, deployment timeout interval, environment variables, and storage mounting. If the service is deployed in a dedicated resource pool, the resource pool information is also displayed. • Update History: historical model information.

Parameter	Description
Monitoring	This page displays resource usage and model calls. • Resource Usage: includes the used and available CPU, memory, GPU, and NPU resources. • AI Application Calls: indicates the number of model calls. The statistics collection starts after the model status changes to Ready. (This parameter is not displayed for WebSocket services.)
Events	This page displays key operations during service use, such as the service deployment progress, detailed causes of deployment exceptions, and time points when a service is started, stopped, or modified. Events are saved for one month and will be automatically cleared then. For details about how to view events of a service, see Viewing Events of a Real-Time Service.
Logs	This page displays the log information about each model in the service. You can view logs generated in the latest 5 minutes, latest 30 minutes, latest 1 hour, and user-defined time segment. You can select the start time and end time when defining a time segment. Log search rules: • Do not enter a string that contains any of the following delimiters: ,";=()[]{}@<>/: \n\t\r. • You can use exact search by keyword. A keyword refers to the word between two adjacent delimiters. • You can use fuzzy search by keyword. For example, you can enter error, er?or, rro*, or er*r. • You can enter a phrase for exact search. For example, Start to refresh. • Before enabling this feature, you can combine keywords with && or . For example, query logs&&erro* or query logs erro*. After enabling this feature, you can combine keywords with AND or OR. For example, query logs AND erro* or query logs OR erro*.

Modifying Customized Settings

A customized configuration rule consists of the configuration condition (**Setting**), access version (**Version**), and customized running parameters (including **Setting Name** and **Setting Value**).

You can configure different settings with customized running parameters for different versions of a real-time service.

The priorities of customized configuration rules are in descending order. You can change the priorities by dragging the sequence of customized configuration rules.

After a rule is matched, the system will no longer match subsequent rules. A maximum of 10 configuration rules can be configured.

Table 7-28 Parameters for Custom Settings

Parameter	Man dator y	Description
Setting	Yes	Expression of the Spring Expression Language (SpEL) rule. Only the equal, matches, and hashCode expressions of the character type are supported.
Version	Yes	Access version for a customized service configuration rule. When a rule is matched, the real-time service of the version is requested.
Setting Name	No	Key of a customized running parameter, consisting of a maximum of 128 characters. Configure this parameter if the HTTP message header is used to carry customized running parameters to a real-time service.
Setting Value	No	Value of a customized running parameter, consisting of a maximum of 256 characters. Configure this parameter if the HTTP message header is used to carry customized running parameters to a real-time service.

Customized settings can be used in the following scenarios:

- If multiple versions of a real-time service are deployed for gray release, customized settings can be used to distribute traffic by user.

Table 7-29 Built-in variables

Built-in Variable	Description
DOMAIN_NAME	Account name used to call an inference request
DOMAIN_ID	Account ID used to call an inference request
PROJECT_NAME	Project name used to call an inference request
PROJECT_ID	Project ID used to call an inference request
USER_NAME	Username used to call an inference request
USER_ID	User ID used to call an inference request

Pound key (#) indicates that a variable is referenced. The matched character string must be enclosed in single quotation marks.

```
#Built-in variable == 'Character string'  
#{Built-in variable} matches 'Regular expression'
```

– Example 1:

If the account name in the inference request is **User A**, the specified version is matched.

```
#DOMAIN_NAME == 'zhangsan'
```

– Example 2:

If the account name in the inference request starts with **op**, the specified version is matched.

```
#DOMAIN_NAME matches 'op.*'
```

Table 7-30 Common regular expressions

Character	Description
.	Match any single character except \n. To match any character including \n, use (.\n).
*	Match the subexpression that it follows for zero or multiple times. For example, zo* can match z and zoo .
+	Match the subexpression that it follows for once or multiple times. For example, zo+ can match zo and zoo , but cannot match z .
?	Match the subexpression that it follows for zero or one time. For example, do(es)? can match does or do in does .
^	Match the start of the input string.
\$	Match the end of the input string.
{n}	<i>n</i> is a non-negative integer, which matches exactly <i>n</i> number of occurrences of an expression. For example, o{2} cannot match o in Bob , but can match two os in food .
x y	Match x or y. For example, z food can match z or food , and (z f)ood can match zood or food .
[xyz]	Character set, where any single character in it can be matched. For example, [abc] can match a in plain .

- If multiple versions of a real-time service are deployed for gated launch, customized settings can be used to access different versions through the header.

Start with **#HEADER_**, indicating that the header is referenced as a condition.

```
#HEADER_{key} == '{value}'  
#HEADER_{key} matches '{value}'
```

– Example 1:

If the header of an inference HTTP request contains a version and the value is **0.0.1**, the condition is met. Otherwise, the condition is not met.

```
#HEADER_version == '0.0.1'
```

– Example 2:

If the header of an inference HTTP request contains **testheader** and the value starts with **mock**, the rule is matched.

```
#HEADER_testheader matches 'mock.*'
```

– Example 3:

If the header of an inference HTTP request contains **uid** and the hash code value meets the conditions described in the following algorithm, the rule is matched.

```
#HEADER_uid.hashCode() % 100 < 10
```

- If a real-time service version supports different runtime configurations, you can use **Setting Name** and **Setting Value** to specify customized runtime parameters so that different users can use different running configurations.

Example:

When user A accesses the model, the user uses configuration A. When user B accesses the model, the user uses configuration B. When matching a running configuration, ModelArts adds a header to the request and also the customized running parameters specified by **Setting Name** and **Setting Value**.

7.1.7.2 Viewing Events of a Real-Time Service

During the whole lifecycle of a service, every key event is automatically recorded. You can view the events on the details page of the service at any time.

This helps you better understand the service deployment and running process and accurately locate faults when a task exception occurs. You can check the following events:

Table 7-31 Events

Event Type	Event (<i>xxx</i> should be replaced with the actual value.)	Solution
Normal	The service starts to deploy.	N/A
Abnormal	Insufficient resources. Wait until idle resources are sufficient.	Wait until the resources are released and try again.
Abnormal	Insufficient <i>xxx</i> . The scheduling failed. Supplementary information: <i>xxx</i>	Learn about resource insufficiency details based on the supplementary information. For details, see FAQs .
Normal	The image starts to create.	N/A

Event Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	Failed to create model image xxx. For details, see logs : nxxx.	Locate and rectify the fault based on the build logs.
Abnormal	Failed to create the image.	Contact technical support.
Normal	The image is created.	N/A
Abnormal	Service xxx failed. Error: xxx	Locate and rectify the fault based on the error information.
Abnormal	Failed to update the service. Perform a rollback.	Contact technical support.
Normal	The service is being updated.	N/A
Normal	The service is being started.	N/A
Normal	The service is being stopped.	N/A
Normal	The service has been stopped.	N/A
Normal	Auto stop has been disabled.	N/A
Normal	Auto stop has been enabled. The service will stop after xs.	N/A
Normal	The service stops when the auto stop time expires.	N/A
Abnormal	The service is stopped because the quota exceeds the upper limit.	Contact technical support.
Abnormal	Failed to automatically stop the service. Error: xxx	Locate and rectify the fault based on the error information.
Normal	Service instances deleted from resource pool xxx.	N/A
Normal	Service instances stopped in resource pool xxx.	N/A

Event Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	The batch service failed. Try again later. Error: xxx	Locate and rectify the fault based on the error information.
Normal	The service has been executed.	N/A
Abnormal	Failed to stop the service. Error: xxx	Locate and rectify the fault based on the error information.
Normal	The subscription license xxx is to expire.	N/A
Normal	Service xxx started.	N/A
Abnormal	Failed to start service xxx.	For details about how to locate and rectify the fault, see FAQs .
Abnormal	Service deployment timed out. Error: xxx	Locate and rectify the fault based on the error information.
Normal	Failed to update the service. The update has been rolled back.	N/A
Abnormal	Failed to update the service. The rollback failed.	Contact technical support.
Normal	[model 0.0.1] OBS bucket, OBS parallel file system, or SFS Turbo mounted successfully. [%s] %s volume successfully.	N/A

During service deployment and running, key events can both be manually and automatically refreshed.

Viewing Events

1. In the navigation pane of the ModelArts console, choose **Service Deployment > Real-Time Services**. In the service list, click the name or ID of the target service to go to its details page.

2. View the events in the **Events** tab.

7.1.7.3 Managing the Lifecycle of a Real-Time Service

Starting a Service

A service not in the **Deploying** state can be started. To start a service, use either of the following methods:

- Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service. Click **Start** in the **Operation** column to start the target service.
- Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service. Click the name of the target service to access its details page. Click **Start** in the upper right corner of the page to start the service.

Stopping a Service

Stop a service in either of the following ways:

- Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**. Choose **More > Stop** in the **Operation** column to stop the target service.
- Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**. Click the name of the target service to access its details page. Click **Stop** in the upper right corner of the page to stop the service.

Deleting a Service

If a service is no longer in use, delete it to release resources.

Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**.

- In the real-time service list, choose **More > Delete** in the **Operation** column of the target service to delete it.
- Select services in the real-time service list and click **Delete** above the list to delete services in batches.
- Click the name of the target service. On the displayed service details page, click **Delete** in the upper right corner to delete the service.

NOTE

- A deleted service cannot be recovered.
- A service cannot be deleted without agency authorization.
- If **Advanced Log Management** is enabled for a real-time service, you are advised to delete the LTS logs and streams when you delete the service. This prevents additional fees incurred by the logs and streams.

Restarting a Service

You can restart a real-time service only when the service is in the **Running** or **Alarm** state. Batch services and edge services cannot be restarted. You can restart a real-time service in either of the following ways:

- Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**. Choose **More > Restart** in the **Operation** column to restart the target service.
- Log in to the ModelArts console and choose **Service Deployment > Real-Time Services**. Click the name of the target service to access its details page. Click **Restart** in the upper right corner of the page to restart the service.

7.1.7.4 Modifying a Real-Time Service

For a deployed service, you can modify its basic information to match service changes and change the model version to upgrade it.

You can modify the basic information about a service in either of the following ways:

Method 1: Modify the Service Information on the Service Management Page

Method 2: Modify the Service Information on the Service Details Page

Prerequisites

The service has been deployed. The service in the **Deploying** state cannot be upgraded by modifying the service information.

Constraints

- Improper upgrade operations will interrupt services during the upgrade.
- ModelArts supports hitless rolling upgrade of real-time services in some scenarios. Prepare for and fully verify the upgrade.

Table 7-32 Scenarios for hitless rolling upgrade

Meta Model Source	Using a Public Resource Pool	Using a Dedicated Resource Pool
Training job	Not supported	Not supported
Container image	Not supported	Supported. The custom image for creating a model must meet Specifications for Custom Images .
OBS	Not supported	Not supported

Method 1: Modify the Service Information on the Service Management Page

1. Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service.
2. In the service list, click **Modify** in the **Operation** column of the target service, modify basic service information, and submit the modification task as prompted.

When some parameters are modified, the system automatically restarts the service for the modification to take effect. When you submit a service modification task, if a restart is required, a dialog box will be displayed.

For details about the real-time service parameters, see [Deploying a Model as a Real-Time Service](#). To modify a real-time service, you also need to set **Max. Invalid Instances** to set the maximum number of nodes that can be concurrently upgraded, during which time these nodes are invalid.

When modifying a real-time service's parameters, you can add a custom environment variable to migrate the service to a new resource pool.

Method 2: Modify the Service Information on the Service Details Page

1. Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service.
2. Click the name of the target service to access its details page.
3. Click **Modify** in the upper right corner of the page, modify the service details, and submit the modification task as prompted.

When some parameters are modified, the system automatically restarts the service for the modification to take effect. When you submit a service modification task, if a restart is required, a dialog box will be displayed.

For details about the real-time service parameters, see [Deploying a Model as a Real-Time Service](#). To modify a real-time service, you also need to set **Max. Invalid Instances** to set the maximum number of nodes that can be concurrently upgraded, during which time these nodes are invalid.

When modifying a real-time service's parameters, you can add a custom environment variable to migrate the service to a new resource pool.

7.1.7.5 Viewing Performance Metrics of a Real-Time Service on Cloud Eye

ModelArts Metrics

The cloud service platform provides Cloud Eye to help you better understand the statuses of ModelArts real-time services and model loads. You can use Cloud Eye to automatically monitor your ModelArts real-time services and model loads in real time and manage alarms and notifications so that you can obtain the performance metrics of ModelArts and models.

Table 7-33 ModelArts metrics

ID	Name	Description	Value Range	Monitored Object	Monitoring Interval
cpu_usage	CPU Usage	CPU usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
mem_usage	Memory Usage	Memory usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
gpu_util	GPU Usage	GPU usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
gpu_mem_usage	GPU Memory Usage	GPU memory usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
npu_util	NPU Usage	NPU usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
npu_mem_usage	NPU Memory Usage	NPU memory usage of ModelArts Unit: %	≥ 0%	ModelArts model loads	1 minute
successfully_called_times	Number of Successful Calls	Times that ModelArts services have been successfully called Unit: counts/minute	≥ counts/minute	ModelArts model loads ModelArts real-time services	1 minute
failed_called_times	Number of Failed Calls	Times that ModelArts services failed to be called Unit: counts/minute	≥ counts/minute	ModelArts model loads ModelArts real-time services	1 minute
total_called_times	Total Calls	Times that ModelArts services are called Unit: counts/minute	≥ counts/minute	ModelArts model loads ModelArts real-time services	1 minute

ID	Name	Description	Value Range	Monitored Object	Monitoring Interval
tp_99	TP99	Collects the response durations of each call over the last minute, arranges them in ascending order, and then excludes the top 1% of values. The highest remaining value represents the TP99.	≥ ms	ModelArts real-time services	1 minute
tp_999	TP99.9	Collects the response durations of each call over the last minute, arranges them in ascending order, and then excludes the top 0.1% of values. The highest remaining value represents the TP99.9.	≥ ms	ModelArts real-time services	1 minute

ID	Name	Description	Value Range	Monitored Object	Monitoring Interval
If a monitored object has multiple dimensions, all dimensions are mandatory when you use APIs to query the metrics. • The following provides an example of using the multi-dimensional dim to query a single monitoring metric: dim.0=service_id,530cd6b0-86d7-4818-837f-935f6a27414d&dim.1="model_id,3773b058-5b4f-4366-9035-9bbd9964714a" • The following provides an example of using the multi-dimensional dim to query monitoring metrics in batches: "dimensions": [{ "name": "service_id", "value": "530cd6b0-86d7-4818-837f-935f6a27414d" } { "name": "model_id", "value": "3773b058-5b4f-4366-9035-9bbd9964714a" }]					

Table 7-34 Dimension description

Key	Value
service_id	Real-time service ID
model_id	Model ID

Setting Alarm Rules

Setting alarm rules allows you to customize the monitored objects and notification policies so that you can know the status of ModelArts real-time services and models in a timely manner.

An alarm rule includes the alarm rule name, monitored object, metric, threshold, monitoring interval, and whether to send a notification. This section describes how to set alarm rules for ModelArts services and models.

NOTE

Only real-time services in the **Running** status can be interconnected with CES.

Prerequisites:

- A ModelArts real-time service has been created.

- ModelArts monitoring has been enabled on Cloud Eye. To do so, log in to the Cloud Eye console. On the Cloud Eye page, click **Custom Monitoring**. Then, enable ModelArts monitoring as prompted.

Set an alarm rule in any of the following ways:

- Set an alarm rule for all ModelArts services.
- Set an alarm rule for a ModelArts service.
- Set an alarm rule for a model version.
- Set an alarm rule for a metric of a service or model version.

Method 1: Setting an Alarm Rule for All ModelArts Services

Step 1 Log in to the management console.

Step 2 In the **Service List**, click **Cloud Eye** under **Management & Governance**.

Step 3 In the navigation pane on the left, choose **Alarm Management > Alarm Rules** and click **Create Alarm Rule**.

Step 4 On the **Create Alarm Rule** page, set **Resource Type** to **ModelArts**, **Dimension** to **Service**, and **Method** to **Configure manually**, and set alarm policies. Then, confirm settings and click **Create**.

----End

Method 2: Setting an Alarm Rule for a Single Service

Step 1 Log in to the management console.

Step 2 In the **Service List**, click **Cloud Eye** under **Management & Governance**.

Step 3 In the navigation pane, choose **Cloud Service Monitoring > ModelArts**.

Step 4 Locate a real-time service for which you want to create an alarm rule and click **Create Alarm Rule** in the **Operation** column.

Step 5 On the **Create Alarm Rule** page, create an alarm rule for ModelArts real-time services and models as prompted.

----End

Method 3: Setting an Alarm Rule for a Model Version

Step 1 Log in to the management console.

Step 2 In the **Service List**, click **Cloud Eye** under **Management & Governance**.

Step 3 In the navigation pane, choose **Cloud Service Monitoring > ModelArts**.

Step 4 Click the down arrow next to the target real-time service name. Then, click **Create Alarm Rule** in the **Operation** column of the target version.

Step 5 On the **Create Alarm Rule** page, create an alarm rule for model loads as prompted.

----End

Method 4: Setting an Alarm Rule for a Metric of a Service or Model Version

- Step 1** Log in to the management console.
- Step 2** In the **Service List**, click **Cloud Eye** under **Management & Governance**.
- Step 3** In the navigation pane, choose **Cloud Service Monitoring > ModelArts**.
- Step 4** Click the down arrow next to the target real-time service name. Then, click the target version and view alarm rule details.
- Step 5** On the alarm rule details page, click the plus sign (+) in the upper right corner of a metric and set an alarm rule for the metric.

----End

Viewing Monitoring Metrics

Cloud Eye on the cloud service platform monitors the statuses of ModelArts real-time services and model loads. You can obtain the monitoring metrics of each ModelArts real-time service and model on the management console. It takes a period of time to transmit and display monitored data. The statuses displayed on the Cloud Eye console are obtained 5 to 10 minutes before. You can view the monitoring data of a newly created real-time service 5 to 10 minutes later.

Prerequisites:

- The ModelArts real-time service is running properly.
- Alarm rules have been configured on the Cloud Eye page. For details, see [Setting Alarm Rules](#).
- The real-time service has been properly running for at least 10 minutes.
- The monitored data and graphics are available for a new real-time service after the service runs for at least 10 minutes.
- Cloud Eye does not display the metrics of a faulty or deleted real-time service. The monitoring metrics can be viewed after the real-time service starts or recovers.

The monitoring data is unavailable without alarm rules configured in Cloud Eye. For details, see [Setting Alarm Rules](#).

- Step 1** Log in to the management console.
- Step 2** In the **Service List**, click **Cloud Eye** under **Management & Governance**.
- Step 3** In the navigation pane, choose **Cloud Service Monitoring > ModelArts**.
- Step 4** View monitoring graphs.
- Viewing monitoring graphs of a real-time service: Click **View Metric** in the **Operation** column.
 - Viewing monitoring graphs of the model loads: Click  next to the target real-time service, and click **View Metric** in the **Operation** column of the target model.
- Step 5** In the monitoring area, you can select a period to view the monitoring data.

You can view the monitoring data in the last 1 hour, 3 hours, or 12 hours. To view the monitoring curve of a longer time range, click  to enlarge the graph.

----End

7.1.7.6 Configuring Auto Restart upon a Real-Time Service Fault

Description

When an Snt9b hardware fault is detected, the system automatically resets the Snt9B chip and restarts the real-time inference service. This helps to improve the recovery speed.

Constraints

Only synchronous real-time services using Snt9b resources are supported.

This function will reset the entire node. Before enabling it, ensure that the deployed real-time service is using specifications with 8 x N cards and properly evaluate the impacts on services running on the node.

Enabling Auto Restart

When deploying a real-time service, select **Configure Now** next to **Advanced Configuration** and enable auto restart.

7.1.8 Managing Batch Inference Jobs

7.1.8.1 Viewing Details About a Batch Service

After a model is deployed as a batch service, you can access the **Batch Services** page to view its details.

1. Log in to the ModelArts console and choose **Service Deployment > Batch Services**.
2. Click the name of the target service to access its details page.

View service information. For details, see [Table 7-35](#).

Table 7-35 Batch service parameters

Parameter	Description
Name	Name of the batch service.
Service ID	ID of the batch service.
Status	Status of the batch service.
Job ID	Job ID of the batch service.
Instance Flavor	Node flavor of the batch service.

Parameter	Description
Instances	Number of nodes of the batch service.
Start Time	Start time of the batch service job.
Environment Variable	Environment variables added during batch service creation.
End Time	End time of the batch service job.
Description	Service description, which you can click the edit button to modify.
Input Path	OBS path to the input data in the batch service.
Output Path	OBS path to the output data in the batch service.
Model Name & Version	Name and version of the model used by the batch service.

3. Switch between tabs on the details page of a batch service to view more details. For details, see [Table 7-36](#).

Table 7-36 Batch service tabs

Parameter	Description
Events	This page displays key operations during service use, such as the service deployment progress, detailed causes of deployment exceptions, and time points when a service is started, stopped, or modified. Events are saved for one month and will be automatically cleared then. For details about how to view events of a service, see Viewing Events of a Real-Time Service.

Parameter	Description
Logs	This page displays the log information about each model in the service. You can view logs generated in the latest 5 minutes, latest 30 minutes, latest 1 hour, and user-defined time segment. You can select the start time and end time when defining a time segment. Log search rules: • Do not enter a string that contains any of the following delimiters: <code>,"";=()[]{}@&<>/:\n\t\r</code>. • You can use exact search by keyword. A keyword refers to the word between two adjacent delimiters. • You can use fuzzy search by keyword. For example, you can enter error, er?or, rro*, or er*r. • You can enter a phrase for exact search. For example, Start to refresh. • Before enabling this feature, you can combine keywords with && or . For example, query logs&&erro* or query logs erro*. After enabling this feature, you can combine keywords with AND or OR. For example, query logs AND erro* or query logs OR erro*.

7.1.8.2 Viewing Events of a Batch Service

During the whole lifecycle of a service, every key event is automatically recorded. You can view the events on the details page of the service at any time.

This helps you better understand the service deployment and running process and accurately locate faults when a task exception occurs. You can check the following events:

Table 7-37 Events

Event Type	Event (xxx should be replaced with the actual value.)	Solution
Normal	The service starts to deploy.	N/A
Abnormal	Insufficient resources. Wait until idle resources are sufficient.	Wait until the resources are released and try again.

Event Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	Insufficient xxx. The scheduling failed. Supplementary information: xxx	Learn about resource insufficiency details based on the supplementary information. For details, see FAQs .
Normal	The image starts to create.	N/A
Abnormal	Failed to create model image xxx. For details, see logs :/nxxx.	Locate and rectify the fault based on the build logs.
Abnormal	Failed to create the image.	Contact technical support.
Normal	The image is created.	N/A
Abnormal	Service xxx failed. Error: xxx	Locate and rectify the fault based on the error information.
Abnormal	Failed to update the service. Perform a rollback.	Contact technical support.
Normal	The service is being updated.	N/A
Normal	The service is being started.	N/A
Normal	The service is being stopped.	N/A
Normal	The service has been stopped.	N/A
Normal	Auto stop has been disabled.	N/A
Normal	Auto stop has been enabled. The service will stop after xs.	N/A
Normal	The service stops when the auto stop time expires.	N/A
Abnormal	The service is stopped because the quota exceeds the upper limit.	Contact technical support.

Event Type	Event (xxx should be replaced with the actual value.)	Solution
Abnormal	Failed to automatically stop the service. Error: xxx	Locate and rectify the fault based on the error information.
Normal	Service instances deleted from resource pool xxx.	N/A
Normal	Service instances stopped in resource pool xxx.	N/A
Abnormal	The batch service failed. Try again later. Error: xxx	Locate and rectify the fault based on the error information.
Normal	The service has been executed.	N/A
Abnormal	Failed to stop the service. Error: xxx	Locate and rectify the fault based on the error information.
Normal	The subscription license xxx is to expire.	N/A
Normal	Service xxx started.	N/A
Abnormal	Failed to start service xxx.	For details about how to locate and rectify the fault, see FAQs .
Abnormal	Service deployment timed out. Error: xxx	Locate and rectify the fault based on the error information.
Normal	Failed to update the service. The update has been rolled back.	N/A
Abnormal	Failed to update the service. The rollback failed.	Contact technical support.
Normal	[model 0.0.1] OBS bucket, OBS parallel file system, or SFS Turbo mounted successfully. [%s] %s volume successfully.	N/A

During service deployment and running, key events can both be manually and automatically refreshed.

Viewing Events

1. In the navigation pane of the ModelArts console, choose **Service Deployment > Batch Services**. In the service list, click the name or ID of the target service to go to its details page.
2. View the events in the **Events** tab.

7.1.8.3 Managing the Lifecycle of a Batch Service

Starting a Service

A service not in the **Deploying** state can be started. To start a service, use either of the following methods:

- Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service. Click **Start** in the **Operation** column to start the target service.
- Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service. Click the name of the target service to access its details page. Click **Start** in the upper right corner of the page to start the service.

Stopping a Service

Stop a service in either of the following ways:

- Log in to the ModelArts console and choose **Service Deployment > Batch Services**. Click **Stop** in the **Operation** column to stop the target service.
- Log in to the ModelArts console and choose **Service Deployment > Batch Services**. Click the name of the target service to access its details page. Click **Stop** in the upper right corner of the page to stop the service.

Deleting a Service

If a service is no longer in use, delete it to release resources.

Log in to the ModelArts console and choose **Service Deployment > Batch Services**.

- In the batch service list, click **Delete** in the **Operation** column of the target service to delete it.
- Select services in the batch service list and click **Delete** above the list to delete services in batches.
- Click the name of the target service. On the displayed service details page, click **Delete** in the upper right corner to delete the service.

 NOTE

- A deleted service cannot be recovered.
- A service cannot be deleted without agency authorization.

Restarting a Service

Batch services cannot be restarted.

7.1.8.4 Modifying a Batch Service

For a deployed service, you can modify its basic information to match service changes and change the model version to upgrade it.

You can modify the basic information about a service in either of the following ways:

Method 1: Modify the Service Information on the Service Management Page

Method 2: Modify the Service Information on the Service Details Page

Prerequisites

The service has been deployed. The service in the **Deploying** state cannot be upgraded by modifying the service information.

Constraints

- Improper upgrade operations will interrupt services during the upgrade.
- ModelArts supports hitless rolling upgrade of real-time services in some scenarios. Prepare for and fully verify the upgrade.

Table 7-38 Scenarios for hitless rolling upgrade

Meta Model Source	Using a Public Resource Pool	Using a Dedicated Resource Pool
Training job	Not supported	Not supported
Container image	Not supported	Supported. The custom image for creating a model must meet .
OBS	Not supported	Not supported

Method 1: Modify the Service Information on the Service Management Page

1. Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service.
2. In the service list, click **Modify** in the **Operation** column of the target service, modify basic service information, and submit the modification task as prompted.

When some parameters are modified, the system automatically restarts the service for the modification to take effect. When you submit a service

modification task, if a restart is required, a dialog box will be displayed. For details about the batch service parameters, see [Deploying a Model as a Batch Inference Service](#).

Method 2: Modify the Service Information on the Service Details Page

1. Log in to the ModelArts console and choose **Service Deployment** from the navigation pane. Go to the service management page of the target service.
2. Click the name of the target service to access its details page.
3. Click **Modify** in the upper right corner of the page, modify the service details, and submit the modification task as prompted.

When some parameters are modified, the system automatically restarts the service for the modification to take effect. When you submit a service modification task, if a restart is required, a dialog box will be displayed. For details about the batch service parameters, see [Deploying a Model as a Batch Inference Service](#).