ModelArts

User Guide (ModelArts Standard)

Issue 01

Date 2024-10-24

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 ModelArts Standard Usage..... 1
2 ModelArts Standard Preparations..... 6
2.1 Configuring Access Authorization for ModelArts Standard..... 6
2.1.1 Configuring Agency Authorization for ModelArts with One Click..... 6
2.1.2 Creating an IAM User and Granting ModelArts Permissions..... 12
2.2 Creating and Managing a Workspace..... 20
2.3 Creating an OBS Bucket for ModelArts to Store Data..... 25
3 ModelArts Standard Resource Management..... 28
3.1 About ModelArts Standard Resource Pools..... 28
3.2 Creating a Standard Dedicated Resource Pool..... 30
3.3 Managing Standard Dedicated Resource Pools..... 37
3.3.1 Viewing Details About a Standard Dedicated Resource Pool..... 37
3.3.2 Resizing a Standard Dedicated Resource Pool..... 41
3.3.3 Upgrading the Standard Dedicated Resource Pool Driver..... 44
3.3.4 Rectifying a Faulty Node in a Standard Dedicated Resource Pool..... 46
3.3.5 Modifying the Job Types Supported by a Standard Dedicated Resource Pool..... 52
3.3.6 Migrating Standard Dedicated Resource Pools and Networks to Other Workspaces..... 53
3.3.7 Configuring the Standard Dedicated Resource Pool to Access the Internet..... 54
3.3.8 Using TMS Tags to Manage Resources by Group..... 56
3.3.9 Releasing Standard Dedicated Resource Pools and Deleting the Network..... 58
4 Using ExeML for Zero-Code AI Development..... 60
4.1 Introduction to ExeML..... 60
4.2 Using ExeML for Image Classification..... 61
4.2.1 Preparing Image Classification Data..... 61
4.2.2 Creating an Image Classification Project..... 63
4.2.3 Labeling Image Classification Data..... 65
4.2.4 Training an Image Classification Model..... 68
4.2.5 Deploying an Image Classification Service..... 69
4.3 Using ExeML for Object Detection..... 71
4.3.1 Preparing Object Detection Data..... 71
4.3.2 Creating an Object Detection Project..... 74
4.3.3 Labeling Object Detection Data..... 76

4.3.4 Training an Object Detection Model....80
4.3.5 Deploying an Object Detection Service....81
4.4 Using ExeML for Predictive Analytics....83
4.4.1 Preparing Predictive Analysis Data....83
4.4.2 Creating a Predictive Analytics Project....86
4.4.3 Training a Predictive Analysis Model....88
4.4.4 Deploying a Predictive Analytics Service....90
4.5 Using ExeML for Sound Classification....91
4.5.1 Preparing Sound Classification Data....92
4.5.2 Creating a Sound Classification Project....93
4.5.3 Labeling Sound Classification Data....95
4.5.4 Training a Sound Classification Model....97
4.5.5 Deploying a Sound Classification Service....98
4.6 Using ExeML for Text Classification....100
4.6.1 Preparing Text Classification Data....100
4.6.2 Creating a Text Classification Project....101
4.6.3 Labeling Text Classification Data....103
4.6.4 Training a Text Classification Model....106
4.6.5 Deploying a Text Classification Service....107
4.7 Tips....108
4.7.1 How Do I Quickly Create an OBS Bucket and a Folder When Creating a Project?....109
4.7.2 Where Are Models Generated by ExeML Stored? What Other Operations Are Supported?....110

5 Using Workflows for Low-Code AI Development....111

5.1 What Is Workflow?....111
5.2 Managing a Workflow....114
5.2.1 Searching for a Workflow....115
5.2.2 Viewing the Running Records of a Workflow....116
5.2.3 Managing a Workflow....118
5.2.4 Retrying, Stopping, or Running a Workflow Phase....119
5.3 Workflow Development Command Reference....120
5.3.1 Core Concepts of Workflow Development....120
5.3.2 Configuring Workflow Parameters....128
5.3.3 Configuring the Input and Output Paths of a Workflow....130
5.3.4 Creating Workflow Phases....133
5.3.4.1 Creating a Dataset Phase....133
5.3.4.2 Creating a Dataset Labeling Phase....139
5.3.4.3 Creating a Dataset Import Phase....145
5.3.4.4 Creating a Dataset Release Phase....154
5.3.4.5 Creating a Training Job Phase....160
5.3.4.6 Creating a Model Registration Phase....177
5.3.4.7 Creating a Service Deployment Phase....185
5.3.5 Creating a Multi-Branch Workflow....193

5.3.5.1 Multi-Branch Workflow 193
5.3.5.2 Creating a Condition Phase to Control Branch Execution 193
5.3.5.3 Configuring Phase Parameters to Control Branch Execution 200
5.3.5.4 Configuring Multi-Branch Phase Data 205
5.3.6 Creating a Workflow 206
5.3.7 Publishing a Workflow 208
5.3.7.1 Publishing a Workflow to ModelArts 208
5.3.7.2 Publishing a Workflow to AI Gallery 211
5.3.8 Advanced Workflow Capabilities 212
5.3.8.1 Using Big Data Capabilities (DLI/MRS) in a Workflow 212
5.3.8.2 Specifying Certain Phases to Run in a Workflow 214

6 Development Environments 215

6.1 Application Scenarios 215
6.2 Creating a Notebook Instance 216
6.3 Using a Notebook Instance for AI Development Through JupyterLab 229
6.3.1 Using JupyterLab to Develop and Debug Code Online 229
6.3.2 Common Functions of JupyterLab 231
6.3.3 Using Git to Clone the Code Repository in JupyterLab 242
6.3.4 Uploading Files to JupyterLab 248
6.3.4.1 Uploading Files from a Local Path to JupyterLab 248
6.3.4.2 Cloning GitHub Open-Source Repository Files to JupyterLab 255
6.3.4.3 Uploading OBS Files to JupyterLab 256
6.3.4.4 Uploading Remote Files to JupyterLab 259
6.3.5 Downloading a File from JupyterLab to a Local PC 260
6.3.6 Using MindInsight Visualization Jobs in JupyterLab 262
6.3.7 Using TensorBoard Visualization Jobs in JupyterLab 264
6.4 Using Notebook Instances Remotely Through PyCharm 267
6.4.1 Connecting to a Notebook Instance Through PyCharm Toolkit 268
6.4.2 Manually Connecting to a Notebook Instance Through PyCharm 276
6.4.3 Uploading Data to a Notebook Instance Through PyCharm 282
6.5 Using Notebook Instances Remotely Through VS Code 283
6.5.1 Connecting to a Notebook Instance Through VS Code 283
6.5.2 Installing VS Code 284
6.5.3 Connecting to a Notebook Instance Through VS Code Toolkit 285
6.5.4 Manually Connecting to a Notebook Instance Through VS Code 294
6.5.5 Uploading and Downloading Files in VS Code 298
6.6 Using a Notebook Instance Remotely with SSH 300
6.7 Managing Notebook Instances 306
6.7.1 Searching for a Notebook Instance 306
6.7.2 Updating a Notebook Instance 307
6.7.3 Starting, Stopping, or Deleting a Notebook Instance 309
6.7.4 Saving a Notebook Instance 310

6.7.5 Dynamically Expanding EVS Disk Capacity....312
6.7.6 Dynamically Mounting an OBS Parallel File System....314
6.7.7 Viewing Notebook Events....315
6.7.8 Notebook Cache Directory Alarm Reporting....321
6.8 ModelArts CLI Command Reference....326
6.8.1 ModelArts CLI Commands....326
6.8.2 (Optional) Installing ma-cli Locally....328
6.8.3 Autocompletion for ma-cli Commands....329
6.8.4 ma-cli Authentication....330
6.8.5 ma-cli image Commands for Building Images....332
6.8.6 ma-cli ma-job Commands for Training Jobs....344
6.8.7 ma-cli dli-job Commands for Submitting DLI Spark Jobs....356
6.8.8 Using ma-cli to Copy OBS Data....369
6.9 Using Moxing Commands in a Notebook Instance....370
6.9.1 Introduction to MoXing Framework....370
6.9.2 Getting Started....371
6.9.3 Introducing MoXing Framework....373
6.9.4 Mapping Between mox.file and Local APIs and Switchover....374
6.9.5 Sample Code for Common Operations....375
6.9.6 Sample Code for Advanced Applications....379
7 Data Management....382
7.1 Introduction to Data Preparation....382
7.2 Getting Started....383
7.3 Creating a Dataset....389
7.3.1 Dataset Overview....389
7.3.2 Creating a Dataset....392
7.3.3 Modifying a Dataset....399
7.4 Importing Data....400
7.4.1 Introduction to Data Importing....400
7.4.2 Importing Data from OBS....402
7.4.2.1 Introduction to Importing Data from OBS....402
7.4.2.2 Importing Data from an OBS Path....404
7.4.2.3 Specifications for Importing Data from an OBS Directory....407
7.4.2.4 Importing a Manifest File....412
7.4.2.5 Specifications for Importing a Manifest File....414
7.4.3 Importing Data from DLI....431
7.4.4 Importing Data from MRS....432
7.4.5 Importing Data from DWS....432
7.4.6 Importing Data from Local Files....433
7.5 Data Analysis and Preview....434
7.5.1 Auto Grouping....434
7.5.2 Data Filtering....437

7.5.3 Data Feature Analysis....437
7.6 Labeling Data....444
7.7 Publishing Data....444
7.7.1 Introduction to Data Publishing....444
7.7.2 Publishing a Data Version....445
7.7.3 Managing Data Versions....447
7.8 Exporting Data....448
7.8.1 Introduction to Exporting Data....448
7.8.2 Exporting Data to a New Dataset....449
7.8.3 Exporting Data to OBS....450
8 Model Training....451
8.1 Model Training Process....451
8.2 Preparing Model Training Code....454
8.2.1 Boot File of a Preset Image....454
8.2.2 Developing Code for Training Using a Preset Image....459
8.2.3 Developing Code for Training Using a Custom Image....461
8.3 Preparing a Model Training Image....465
8.4 Creating a Debug Training Job....466
8.4.1 Using PyCharm Toolkit to Create and Debug a Training Job....466
8.5 Creating an Algorithm....471
8.6 Creating a Production Training Job....482
8.7 Incremental Model Training....496
8.8 Distributed Model Training....498
8.8.1 Overview....498
8.8.2 Creating a Single-Node Multi-Card Distributed Training Job (DataParallel)....499
8.8.3 Creating a Multiple-Node Multi-Card Distributed Training Job (DistributedDataParallel)....501
8.8.4 Example: Creating a DDP Distributed Training Job (PyTorch + GPU)....511
8.8.5 Example: Creating a DDP Distributed Training Job (PyTorch + NPU)....514
8.9 Automatic Model Tuning (AutoSearch)....516
8.9.1 Overview....516
8.9.2 Creating a Training Job for Automatic Model Tuning....518
8.10 High Model Training Reliability....521
8.10.1 Training Job Fault Tolerance Check....522
8.10.2 Training Log Failure Analysis....526
8.10.3 Detecting Training Job Suspension....527
8.10.4 Training Job Rescheduling....528
8.10.5 Resumable Training....528
8.10.6 Enabling Unconditional Auto Restart....530
8.11 Managing Model Training Jobs....531
8.11.1 Viewing Training Job Details....531
8.11.2 Viewing the Resource Usage of a Training Job....534
8.11.3 Viewing the Model Evaluation Result....536

8.11.4 Viewing Training Job Events....540
8.11.5 Viewing Training Job Logs....542
8.11.6 Priority of a Training Job....551
8.11.7 Using Cloud Shell to Debug a Production Training Job....553
8.11.8 Rebuilding, Stopping, or Deleting a Training Job....557
8.11.9 Managing Environment Variables of a Training Container....558
8.11.10 Viewing Training Job Tags....563
9 Inference Deployment....564
9.1 Overview....564
9.2 Creating an AI Application....565
9.2.1 Creation Methods....565
9.2.2 Importing a Meta Model from a Training Job....567
9.2.3 Importing a Meta Model from OBS....570
9.2.4 Importing a Meta Model from a Container Image....575
9.3 Specifications for Creating an AI Application....579
9.3.1 Model Package Structure....580
9.3.2 Specifications for Editing a Model Configuration File....581
9.3.3 Specifications for Writing a Model Inference Code File....597
9.3.4 Specifications for Using a Custom Engine to Create an AI Model....602
9.3.5 Examples of Custom Scripts....605
9.4 Deploying an AI Application as Real-Time Inference Jobs....615
9.4.1 Deploying and Using Real-Time Inference....615
9.4.2 Deploying a Model as a Real-Time Service....617
9.4.3 Authentication Methods for Accessing Real-time Services....625
9.4.3.1 Accessing a Real-Time Service Through Token-based Authentication....625
9.4.3.2 Accessing a Real-Time Service Through AK/SK-based Authentication....633
9.4.3.3 Accessing a Real-Time Service Through App Authentication....639
9.4.4 Accessing a Real-Time Service Through Different Channels....650
9.4.4.1 Accessing a Real-Time Service Through a Public Network....650
9.4.4.2 Accessing a Real-Time Service Through a VPC High-Speed Channel....651
9.4.5 Accessing a Real-Time Service Using Different Protocols....655
9.4.5.1 Accessing a Real-Time Service Using WebSocket....655
9.4.5.2 Accessing a Real-Time Service Using Server-Sent Events....658
9.5 Deploying an AI Application as a Batch Inference Service....659
9.6 Managing AI Applications....666
9.6.1 Viewing Details About an AI Application....666
9.6.2 Viewing Events of an AI Application....670
9.6.3 Managing AI Application Versions....674
9.7 Managing a Synchronous Real-Time Service....675
9.7.1 Viewing Details About a Real-Time Service....675
9.7.2 Viewing Events of a Real-Time Service....682
9.7.3 Managing the Lifecycle of a Real-Time Service....685

9.7.4 Modifying a Real-Time Service ... 686
9.7.5 Viewing Performance Metrics of a Real-Time Service on Cloud Eye ... 688
9.7.6 Integrating a Real-Time Service API into the Production Environment ... 693
9.8 Managing Batch Inference Jobs ... 693
9.8.1 Viewing Details About a Batch Service ... 693
9.8.2 Viewing Events of a Batch Service ... 695
9.8.3 Managing the Lifecycle of a Batch Service ... 698
9.8.4 Modifying a Batch Service ... 700

10 Image Management ... 702
10.1 Application Scenarios of Custom Images ... 702
10.2 Preset Images Supported by ModelArts ... 704
10.2.1 ModelArts Preset Image Updates ... 704
10.2.2 ModelArts Unified Images ... 705
10.2.3 Preset Dedicated Images in Notebook Instances ... 707
10.2.4 Preset Dedicated Images for Training ... 713
10.2.5 Preset Dedicated Images for Inference ... 716
10.3 Creating a Custom Image for a Notebook Instance ... 721
10.3.1 Creating a Custom Image ... 721
10.3.2 Creating a Custom Image on ECS and Using It ... 723
10.3.3 Creating a Custom Image Using Dockerfile ... 729
10.3.4 Creating a Custom Image Using the Image Saving Function ... 732
10.4 Creating a Custom Image for Model Training ... 734
10.4.1 Creating a Custom Training Image ... 734
10.4.2 Creating a Custom Training Image Using a Preset Image ... 735
10.4.3 Migrating Existing Images to ModelArts ... 738
10.4.4 Creating a Custom Training Image (PyTorch + CPU/GPU) ... 741
10.4.5 Creating a Custom Training Image (MPI + CPU/GPU) ... 746
10.4.6 Creating a Custom Training Image (Tensorflow + GPU) ... 754
10.4.7 Creating a Custom Training Image (MindSpore + Ascend) ... 761
10.5 Creating a Custom Image for Inference ... 778
10.5.1 Creating a Custom Image for an AI Application ... 778
10.5.2 Creating a Custom Image in a Notebook Instance Using the Image Saving Function ... 781
10.5.3 Creating a Custom Image in a Notebook Instance Using Dockerfile ... 790
10.5.4 Creating a Custom Image on ECS ... 800

11 Resource Monitoring ... 805
11.1 Overview ... 805
11.2 Viewing Monitoring Metrics on the ModelArts Console ... 805
11.3 Viewing All ModelArts Monitoring Metrics on the AOM Console ... 806
11.4 Using Grafana to View AOM Monitoring Metrics ... 830
11.4.1 Installing and Configuring Grafana ... 830
11.4.1.1 Installing and Configuring Grafana on Windows ... 831
11.4.1.2 Installing and Configuring Grafana on Linux ... 832

11.4.1.3 Installing and Configuring Grafana on a Notebook Instance....834
11.4.2 Configuring a Grafana Data Source....837
11.4.3 Configuring a Dashboard to View Metric Data....841
12 Viewing Audit Logs....847
12.1 ModelArts Key Operations Traced by CTS....847
12.2 Viewing ModelArts Audit Logs....853

1 ModelArts Standard Usage

This chapter aims to help you learn how to use ModelArts Standard and get started with the ModelArts service quickly.

For developers who are experienced in coding, debugging, and working with AI engines, ModelArts provides online coding environments as well as an E2E AI development process that covers data preparation, model training, model management, and service deployment.

This document describes how to perform AI development on the ModelArts management console. If you use the APIs or SDKs for development, view ***ModelArts SDK Reference*** or ***ModelArts API Reference***.

To view the examples of AI development lifecycle, see ***Getting Started*** and ***Best Practices***.

Application Scenarios of ModelArts Standard

ModelArts Standard is a one-stop development platform for AI developers. It provides a user-friendly management console with end-to-end AI development toolchains, covering ExeML, data preparation, development environment, model training, model management, and service deployment.

- ModelArts Standard ExeML helps you build AI models without coding. ExeML automates model design, parameter tuning and training, and model compression and deployment based on labeled data. With ExeML, you only need to upload data and perform simple operations as prompted on the ExeML GUI to train and deploy models. For details, see **Introduction to ExeML**.
- ModelArts Standard's workflow is a low-code AI development pipeline tool, covering data labeling, data processing, model development, training, model evaluation, and service deployment. Workflows are executed in visualized mode. For details, see **What Is Workflow?**.
- ModelArts Standard's development environment, notebook, provides a cloud-based JupyterLab environment and local IDE plug-ins, helping you write training and inference code and use cloud resources to debug the code. For details, see **Notebook Application Scenarios**.
- ModelArts Standard's model training provides GUI-based training, debugging, and production environments. You can use your own data and algorithms to

train models using the compute resources provided by ModelArts Standard. For details, see **Model Training**.

- ModelArts Standard's inference deployment provides a GUI-based production environment for inference deployment. After an AI model is developed, you can manage it and quickly deploy it as an inference service. You can perform online inference and prediction or integrate AI inference capabilities into your IT platform by calling APIs. For details, see **Overview**.

Process for Using ModelArts Standard

The AI development lifecycle on ModelArts Standard allows you to experience end-to-end AI development, from preparing data to deploying a model as a service. It takes developers' habits into consideration and provides a variety of engines and scenarios for you to choose. You can use the ModelArts Standard functions as needed in each phase during AI development. The following describes the entire process from data preparation to service development using ModelArts.

Figure 1-1 ModelArts Standard usage process

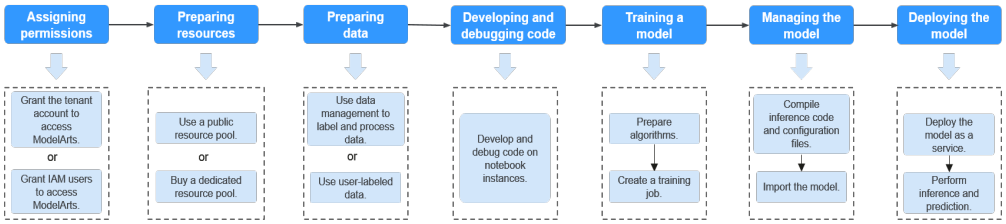


Table 1-1 Process description

Task	Subtask	Description	Reference
Assigning permissions	Configuring agency authorization for ModelArts	ModelArts depends on other cloud services, and you need to configure agency authorization to allow ModelArts to access these services.	**Configuring Agency Authorization for ModelArts with One Click**
	Creating an IAM user and granting ModelArts permissions	For enterprise or university users, you need to create independent IAM users and sub-users and configure fine-grained permissions for them to achieve refined resource and permission management.	**Creating an IAM User and Granting ModelArts Permissions**

Task	Subtask	Description	Reference
(Optional) Creating an OBS bucket	Creating an OBS bucket for ModelArts to store data	ModelArts does not support data storage itself. The input data, output data, and cached data generated during AI development using ModelArts Standard can be stored in OBS buckets. Therefore, you are advised to create an OBS bucket before using ModelArts. You can also create an OBS bucket later when needed.	**Creating an OBS Bucket for ModelArts to Store Data**
(Optional) Preparing resources	Creating a dedicated ModelArts Standard resource pool	ModelArts Standard supports both public and dedicated resource pools. Public resource pool: When creating a training or inference task, you can choose the public resource pool directly without having to create one by yourself. If you use the public resource pool, skip this step. Dedicated resource pool: You need to purchase and create a dedicated resource pool first, but the resources are exclusively used by yourself. This step is mandatory if you use the dedicated resource pool.	**Creating a Standard Dedicated Resource Pool**
(Optional) Preparing data	Creating a dataset	ModelArts Standard supports data management. You can create datasets in ModelArts Standard for managing, preprocessing, and labeling data. If you have prepared data for training, you can directly upload the data to OBS without using the data management function.	**Creating a Dataset** **Labeling Data** **Publishing a Dataset**

Task	Subtask	Description	Reference
Developing and debugging code in the development environment	Creating a notebook instance	Create a notebook instance as the development environment for debugging training and inference code. You are advised to debug the training code in the development environment before creating a production training job.	**Creating a Notebook Instance**
Training a model	Preparing algorithms	Before creating a training job, you need to prepare an algorithm. You can subscribe to an algorithm in AI Gallery or use your own algorithm.	**Preparing Algorithms**
	Creating a training job	Create a training job, select the available dataset version, and use the compiled training script. After training is complete, a model is generated and stored in OBS.	**Creating a Training Job**
Managing AI applications	Compiling inference code and configuration files	Following the model package specifications provided by ModelArts, compile inference code and configuration files for your model, and save them to the training output path.	**Model Package Specifications**
	Creating an AI application	Import a trained model to ModelArts to create an AI application, facilitating AI application deployment and publishing.	**Creating an AI Application**
Deploying AI applications	Deploying a model as a service	Deploy a model as a real-time, batch, or edge service.	• **Deploying a Model as a Real-Time Service** • **Deploying a Model as a Batch Service**

Task	Subtask	Description	Reference
	Accessing the service	After the service is deployed, access the real-time or edge service, or view the prediction result of the batch service.	• **Accessing a Real-Time Service** • **Viewing the Prediction Result of a Batch Service**

2 ModelArts Standard Preparations

2.1 Configuring Access Authorization for ModelArts Standard

2.1.1 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.

Constraints

- Huawei Cloud account
  - Only a Huawei Cloud account can use an agency to authorize the current account or all IAM users under the current account.

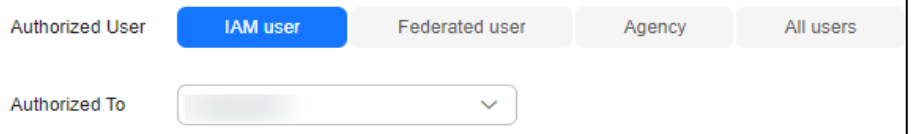
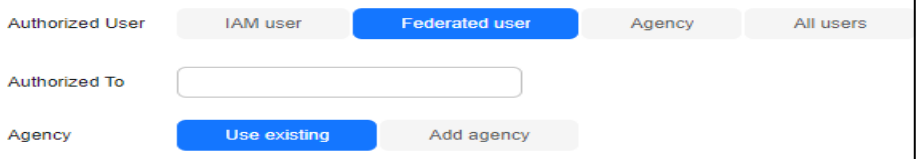
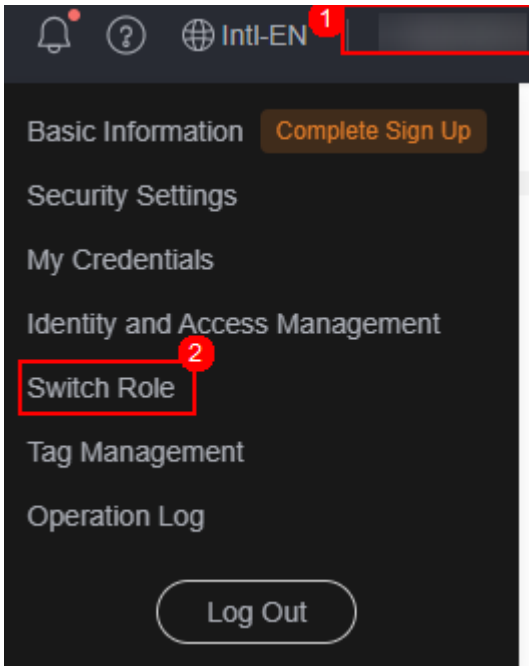
  - Multiple IAM users or accounts can use the same agency.
  - A maximum of 50 agencies can be created under an account.
  - If you use ModelArts for the first time, add an agency. Generally, common user permissions are sufficient for your requirements. You can also customize permissions for refined permissions management.
- IAM user
  - If you have obtained the authorization, you can view the authorization information on the **Permission Management** page.
  - If you have not been authorized, ModelArts will display a message indicating that you have not been authorized when you access the **Add Authorization** page. In this case, contact your administrator to add authorization.

Adding Authorization

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**. The **Permission Management** page is displayed.
2. Click **Add Authorization**. On the **Add Authorization** page that is displayed, set parameters.

Table 2-1 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. For details about IAM users, see **IAM User**. • **Federated user**: A federated user is also called a virtual enterprise user. For details about federated users, see **Configuring Federated Identity Authentication**. • **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 2-1** Selecting an IAM user  • **Federated user**: Enter the username or user ID of the target federated user. **Figure 2-2** 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 2-3** 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 > **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.

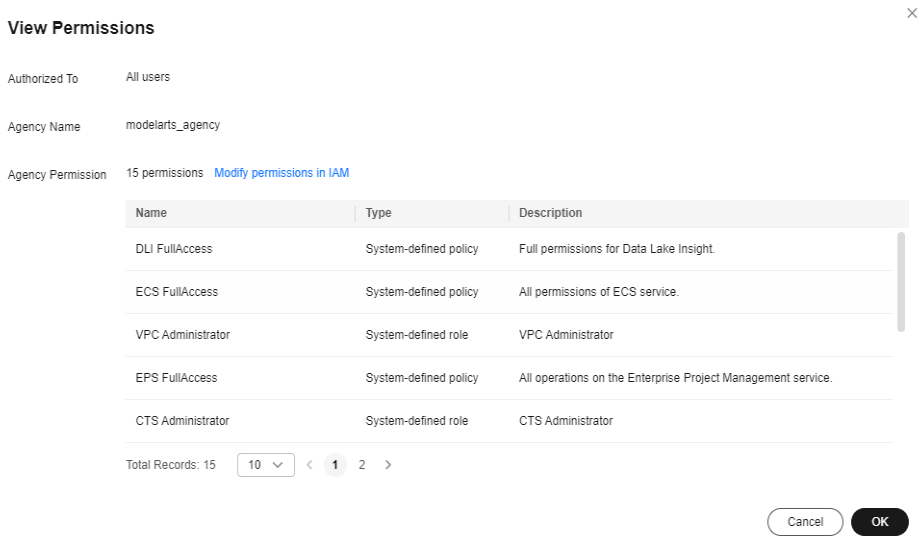
3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

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 2-4 View Permissions

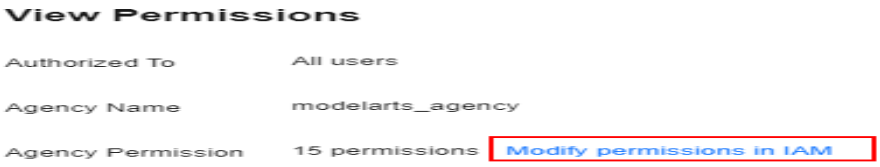
Figure 2-5 Common user permissions



Modifying the Authorization Scope

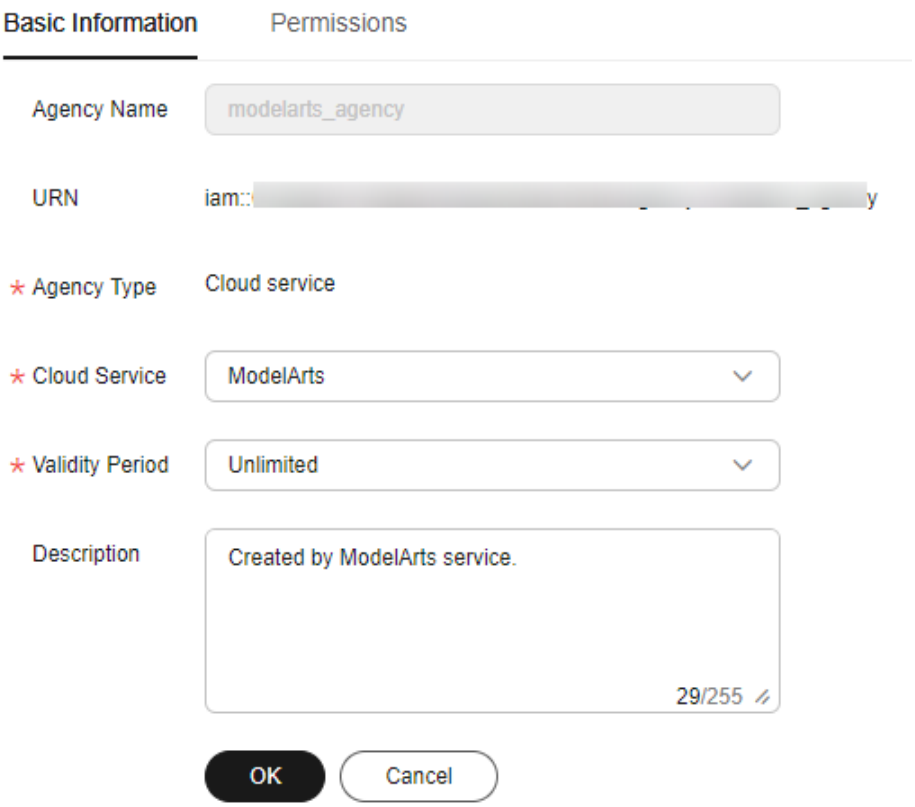
1. To modify the authorization scope, click **Modify permissions in IAM** in the **View Permissions** dialog box.

   Figure 2-6 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 2-7 Agency information

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.

- **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?

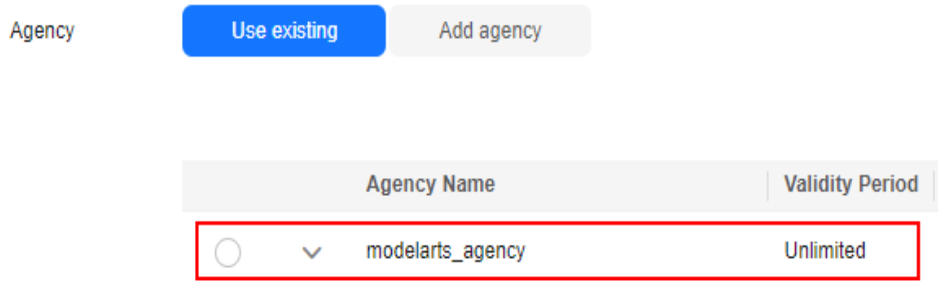
   On the **Add Authorization** page, set **Agency** to **Add agency** and select **Common User**, which provides the permissions to use all basic ModelArts functions. For example, you can access data, and create and manage training jobs. Select this option generally.

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?**

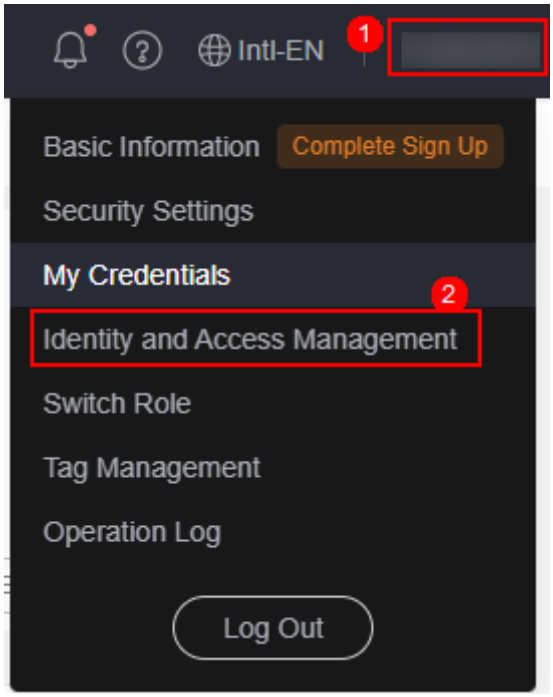
3. How do I delete an existing agency from the agency list?

   Figure 2-8 Use existing

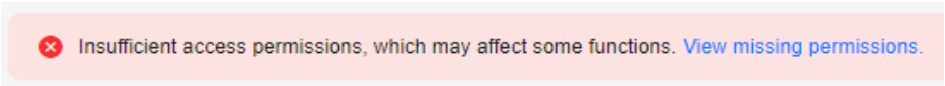
   Go to the IAM console, choose **Agencies** in the navigation pane, and delete the target agency.

Figure 2-9 Identity and Access Management



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

Figure 2-10 Insufficient 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.

2.1.2 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 2-2**.

Table 2-2 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

Target Service	Authorization Description	IAM Permission	Mandatory
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
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.

Log in to the IAM console. Choose **User Groups** and click **Create User Group**. Enter a user group name and click **OK**.

Step 2 Configure permissions for the user group.

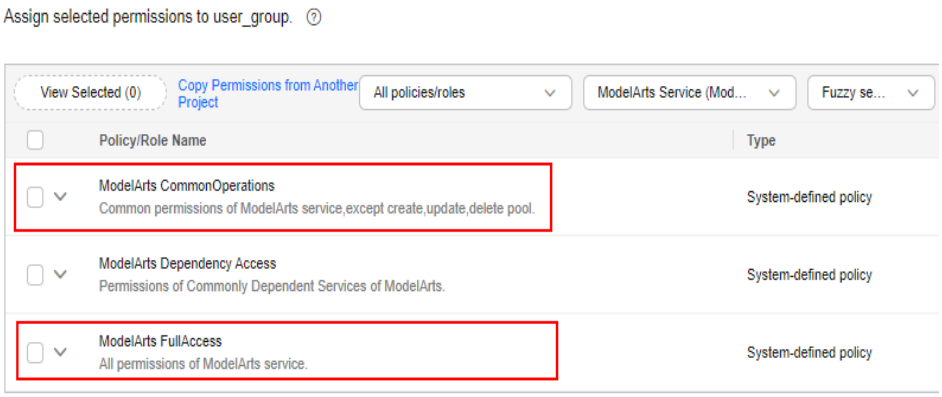
In the user group list, locate the created user group, click **Authorize** in the **Operation** column, and perform the following operations.

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 2-11 Assigning permissions for using ModelArts

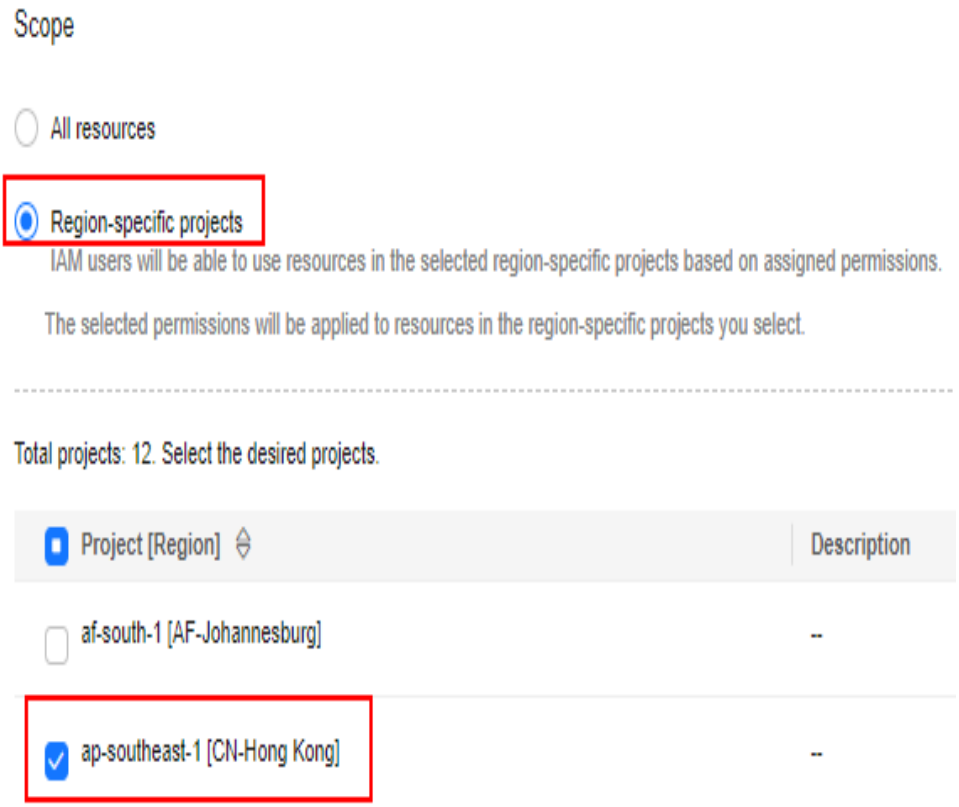
2. Configure permissions for using other dependent services. For example, to use OBS, search for **OBS** and select **OBS OperateAccess**. ModelArts training jobs use OBS to forward data. Therefore, the permissions for using OBS are necessary.

   For permissions of more cloud services, such as SWR, see **Table 2-2**.

3. Click **Next** and set the minimum authorization scope. Select **Region-specific projects**, select the region to be authorized, and click **OK**.

   In this example, you are allowed to use services in the **CN-Hong Kong** region only.

Figure 2-12 Selecting an authorization scope



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 as the IAM user** and verify permissions.

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 2-2**.

----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 more about other functions and dependent services, see **ModelArts Standard Role/Policy-based Authorization**.

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.

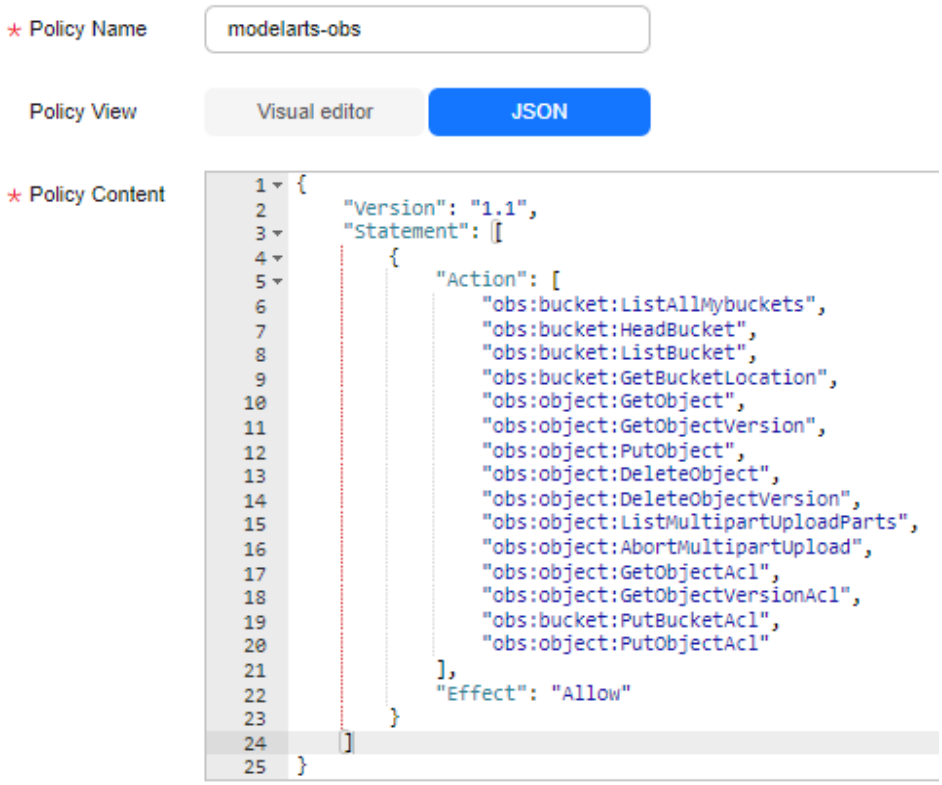
ModelArts is accessible through region-specific projects, but OBS is globally accessible, so you need to create separate policies for the two services and then apply these policies to the users.

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:

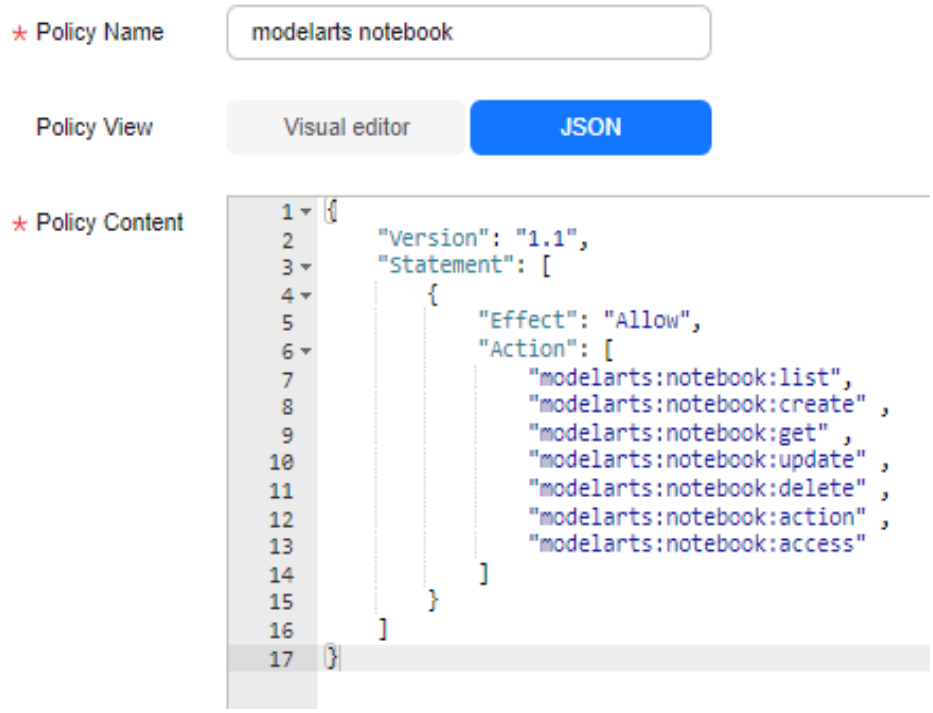
   - **Policy Name**: Enter a policy name.
   - **Policy View**: Select **JSON**.
   - **Policy Content**: Configure the policy by referring to **Example Custom Policy with OBS Permissions for ModelArts**.

Figure 2-13 Minimum OBS permissions



2. Create a custom policy for the permission to use the ModelArts development environment, as shown in **Figure 2-14**. 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**. For the actions that can be added to ModelArts custom policies, see **ModelArts API Reference > Permissions Policies and Supported Actions**.

Figure 2-14 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** > **Development Workspace** > **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: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"
      ]
    }
  ]
}
```

Related References

Here are some other permission configuration examples for your reference.

- **Separately Assigning Permissions to Administrators and Developers**
- **Viewing All Notebook Instances of an IAM Project**
- **Prohibiting a User from Using a Public Resource Pool**
- **Granting SFS Turbo Folder Access Permissions to a Sub-User**

2.2 Creating and Managing a Workspace

NOTE

The workspace is a whitelist function. If you have trial requirements, submit a service ticket to apply for the permission.

Background

ModelArts allows you to create multiple workspaces to develop algorithms and manage and deploy models for different service objectives. In this way, the

development outputs of different applications are allocated to different workspaces for simplified management.

Workspaces can be used to implement logical resource isolation, resource quota management, fine-grained authentication, and resource clearing. Workspaces are presented to enterprise projects as integrated ModelArts resources.

Workspaces support the following access control modes:

- **PUBLIC**: publicly accessible to tenants (including both the tenant account and all its sub-accounts)
- **PRIVATE**: accessible only to the creator and the tenant account
- **INTERNAL**: accessible to the creator, the tenant account, and specified IAM users. When the authorization type is set to **INTERNAL**, specify one or more accessible IAM users.

A default workspace is allocated to each IAM project of each account. The access control of the default workspace is **PUBLIC**.

Workspace access control allows the access of only certain users. This function can be used in the following scenarios:

- **Education**: A teacher allocates an **INTERNAL** workspace to each student and allows the workspace to be accessed only by specified students. In this way, students can separately perform experiments on ModelArts.
- **Enterprises**: An administrator creates a workspace for production tasks and allows only O&M personnel to use the workspace, and creates a workspace for routine debugging and allows only developers to use the workspace. In this way, different enterprise roles can use resources only in a specified workspace.

Prerequisites

You have whitelisted the workspace function and configured basic permissions for using ModelArts. For details, see **Assigning Basic Permissions for Using ModelArts**.

Creating a Workspace

1. Log in to the ModelArts console.
2. In the navigation pane on the left, choose **Workspace**.
3. Click **Create workspace**.

Table 2-3 Creating a workspace

Parameter	Description
Workspace name	Enter the name of the workspace. This parameter is mandatory. The value can contain 4 to 64 visible characters, including letters, digits, hyphens (-), and underscores (_).

Parameter	Description
Description	Enter the description of the workspace, with a maximum of 256 characters.
Enterprise Project	Select an enterprise project. This parameter is mandatory. If no enterprise project is available, click **Create Enterprise Project** to go to the **Enterprise Project Management** page, create an enterprise project, and bind it to the workspace. Enterprise projects provide a cloud resource management mode, in which cloud resources and members are centrally managed by project.
Authorization Type	Select the access control mode for the workspace. • **PUBLIC**: publicly accessible to tenants (including both the tenant account and all its sub-accounts) • **PRIVATE**: accessible only to the creator and the tenant account • **INTERNAL**: accessible to the creator, the tenant account, and specified sub-users When **Authorization Type** is set to **INTERNAL**, you need to set **Authorized User** and **Authorized To** to specify the sub-users who can access the workspace. When **Authorized User** is set to **IAM user**, select one or more IAM users in **Authorized To**. If **Authorized User** is set to **Federated user**, enter one or more usernames or IDs of federated users in **Authorized To**. When **Authorized User** is set to **Agency**, select one or more agencies in **Authorized To**.

4. After setting the workspace parameters, click **Create Now**.

Managing Workspace Quotas

After a workspace is created, you can view and modify the quota information.

1. In the navigation pane of the ModelArts console, choose **Workspace**.
2. On the **Workspace** page, locate the target workspace, and click **Quota Management** in the **Operation** column to go to the workspace details page.
3. In the **Quota Info** area, view the quota value, used quota, and last modification time.
4. Click **Modify Quota** on the right of **Quota Info** to change the quota value. For details about the configurations, see **Table 2-4**.

Table 2-4 Quota information

Quota	Value Description	Unit
ExeML training duration (predictive analytics)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
ExeML training duration (image classification, object detection, and sound classification)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
GPU-based training job duration (calculated based on a single Pnt1 per node)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
CPU-based training job duration (calculated based on a single vCPU per node)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
Visualization job duration	The value ranges from **1** to **60000** and is unlimited by default.	Minute
CPU-based development environment duration (calculated based on the number of vCPUs)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
GPU-based development environment duration (calculated based on the number of Pnt1 GPUs)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
CPU-based inference service duration (calculated based on the number of nodes)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
GPU-based inference service duration (calculated based on the number of nodes)	The value ranges from **1** to **60000** and is unlimited by default.	Minute
vCPUs for CPU-based training jobs	The value ranges from **1** to **10000** and is unlimited by default.	vCPU

Quota	Value Description	Unit
Cards for GPU-based training jobs	The value ranges from **1** to **1000** and is unlimited by default.	Card
RAM size for training jobs	The value ranges from **1** to **100000** and is unlimited by default.	GB
GPU-based auto labeling duration	The value ranges from **1** to **60000** and is unlimited by default.	Minute

5. After changing the quota values of the workspace, click **Submit**. If data in the **Quota Value** column is refreshed, the quotas have been modified successfully.

Modifying a Workspace

After a workspace is created, you can modify its information.

1. In the navigation pane of the ModelArts console, choose **Workspace**.
2. On the **Workspace** page, locate the target workspace and click **Modify** in the **Operation** column.

   You can modify the name, description, enterprise project, and authorization type of the workspace. For details, see **Table 2-4**.
3. After modifying the parameters, click **Modify Now**.

Deleting a Workspace

You can delete a workspace that is no longer needed. After a workspace is deleted, all resources in the workspace are deleted by default. The default workspace cannot be deleted.

NOTE

A deleted workspace cannot be recovered.

1. In the navigation pane of the ModelArts console, choose **Workspace**.
2. On the **Workspace** page, locate the target workspace and click **Delete** in the **Operation** column. In the **Delete Workspace** dialog box, confirm the workspace information and resources to be deleted, enter **DELETE** in the text box, and click **OK**. The workspace status changes to **Deleting**. After the resources are cleared, the workspace is deleted from the list.

2.3 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 2-15 Interaction between ModelArts and OBS

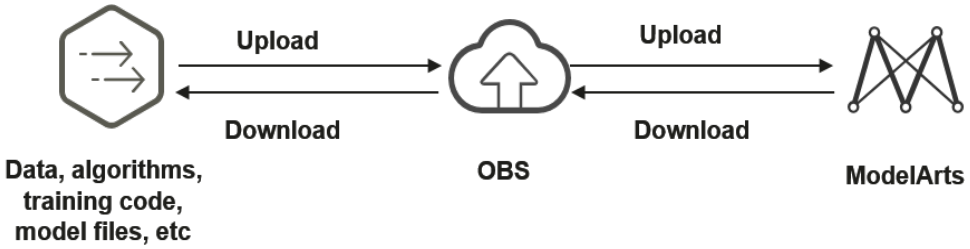


Table 2-5 Relationship between ModelArts and OBS

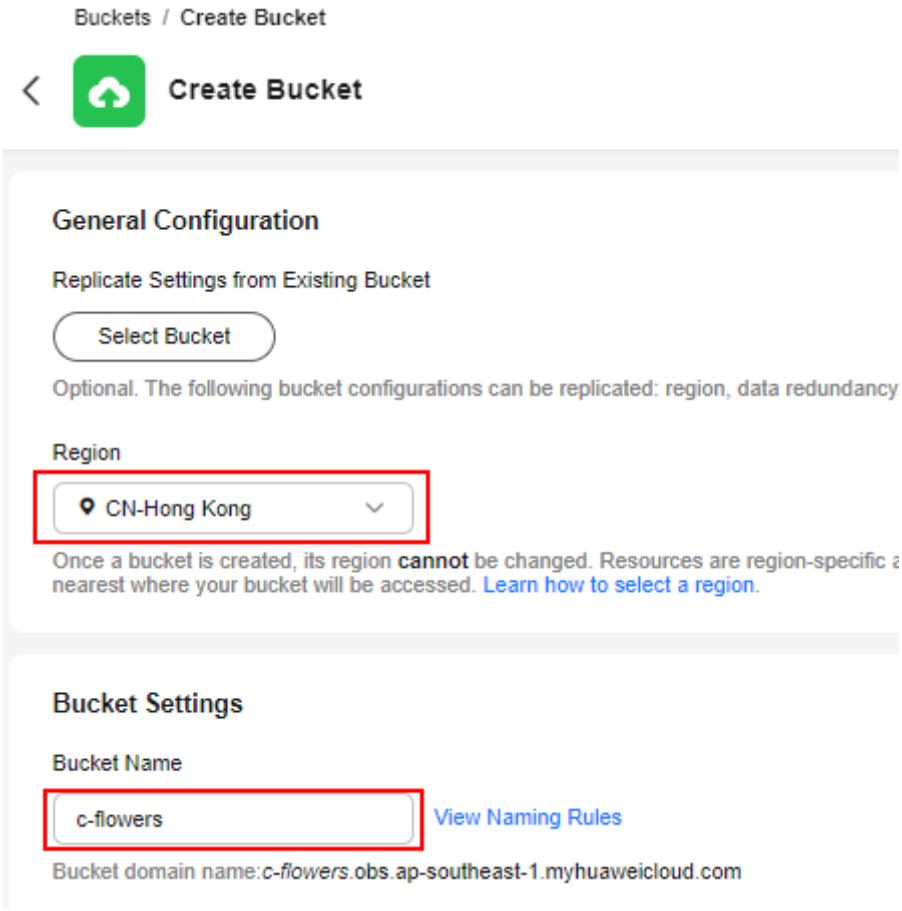
Function	Subtask	Relationship
Standard workflow of ModelArts Standard ExeML	Data labeling	The data labeled on ModelArts is stored in OBS.
	Auto training	After a training job is complete, the generated model is stored in OBS.
	Model deployment	ModelArts deploys models stored in OBS as real-time services.
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.

Function	Subtask	Relationship
	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 an AI application.

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.

Figure 2-16 Creating a bucket



NOTE

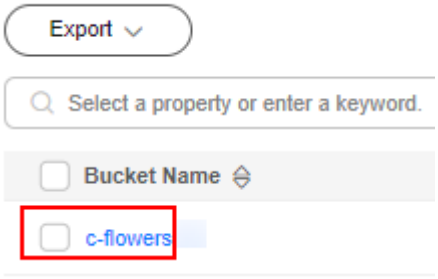
The created OBS bucket and ModelArts are in the same region. For example, if ModelArts is located in the CN-Hong Kong region, select CN-Hong Kong when creating an OBS bucket.

For details about how to obtain the region where the OBS bucket and ModelArts are located, see **Check whether the OBS bucket and ModelArts are in the same region**.

Do not enable **Default Encryption**. ModelArts cannot read the data from encrypted OBS buckets.

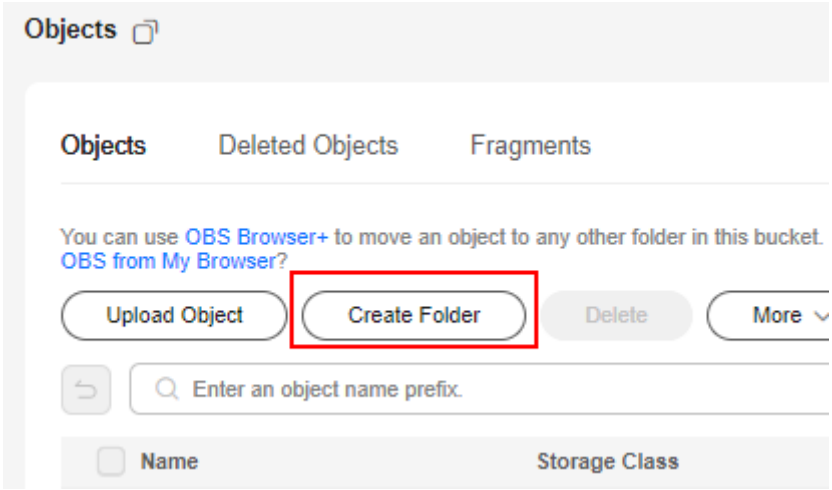
2. In the bucket list, click the bucket name to access its details page.

Figure 2-17 Bucket list



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 2-18 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?

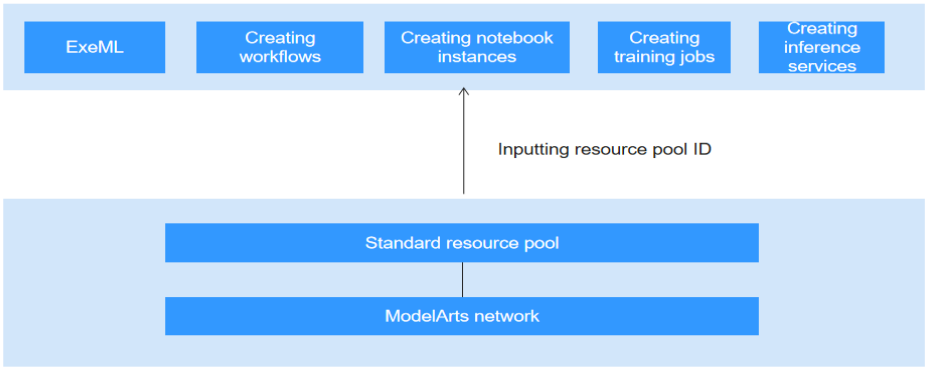
To resolve the preceding problems or other OBS path exceptions, see **Incorrect OBS Path on ModelArts**.

3 ModelArts Standard Resource Management

3.1 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.

Figure 3-1 Using a standard resource pool for AI development



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**.
- Rectify the faulty nodes as needed by referring to **Rectifying a Faulty Node in a Standard Dedicated Resource Pool**.
- 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**.
- Administrators can isolate permissions on resources within workspaces for IAM users. You can move resources to the corresponding workspace by referring to **Migrating Standard Dedicated Resource Pools and Networks to Other Workspaces**.
- Manage resource pools by tag by referring to **Using TMS Tags to Manage Resources by Group**.
- If a dedicated resource pool is no longer needed, delete it by referring to **Releasing Standard Dedicated Resource Pools and Deleting the Network**.

3.2 Creating a Standard Dedicated Resource Pool

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

Prerequisites

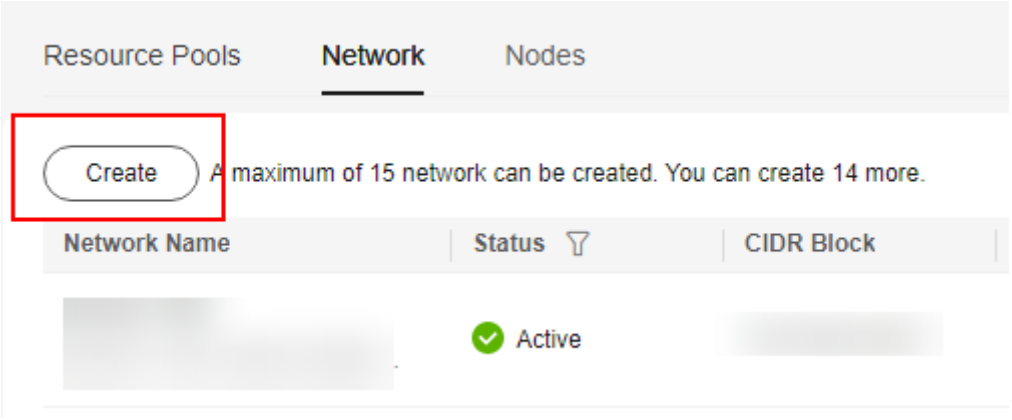
- A VPC is available.
- A subnet is available.

Creating 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 Cluster**.
2. Click the **Network** tab and click **Create**.

   Figure 3-2 Network list

3. In the **Create Network** dialog box, set parameters.
   - **Network Name**: customizable name
   - **CIDR Block**: You can select **Preset** or **Custom**.

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)

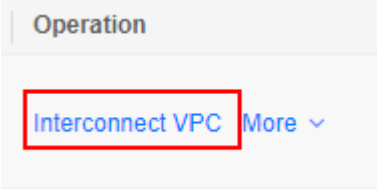
4. Confirm the settings and click **OK**.

(Optional) Interconnecting a VPC with a ModelArts Network

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 3-3 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 3-4 Parameters for interconnecting a VPC with a network

– 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. 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.

Buying a Dedicated AI Cluster

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**.
2. On the **Elastic Clusters** page, click **Buy Dedicated AI Cluster**, and configure the parameters by referring to the following table.

Table 3-1 AI dedicated cluster 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**.
Resource Pool Type	-	You can select **Physical** or **Logical**. If there is no logical specification, **Logical** is not displayed.
Job Type	-	Select job types supported by the resource pool based on service requirements. • **Physical**: **DevEnviron**, **Training Job**, and **Inference Service** are supported. • **Logical**: Only **Training Job** is supported.

Parameter	Sub-Parameter	Description
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 **Creating a Network**.
Specification Management	Specifications	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 nodes in an AZ.
	Nodes	Select the number of nodes in a dedicated resource pool. More nodes mean higher computing performance. If **AZ** is set to **Specifies AZ**, you do not need to configure **Nodes**. **NOTE** It is a good practice to create no more than 30 nodes at a time. Otherwise, the creation may fail due to traffic limiting. You can buy nodes by rack for certain specifications. The total number of nodes is the number of racks multiplied by the number of nodes per rack. 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. **Figure 3-5** Purchasing a rack of nodes 

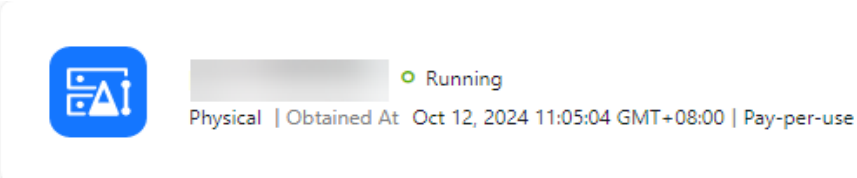
Parameter	Sub-Parameter	Description
	Advanced Configuration	This allows you to set the container engine space. You must enter an integer for the container engine space. It cannot be less than 50 GB, which is the default and minimum value. The maximum value depends on the specifications. To see the valid 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.
Required Duration	-	Select the time length for which you want to use the resource pool. This parameter is mandatory only when the **Yearly/Monthly** billing mode is selected.
Auto-renewal	-	Specifies whether to enable auto-renewal. This parameter is mandatory only when the **Yearly/Monthly** billing mode is selected. • Monthly subscriptions renew each month. • Yearly subscriptions renew each year.
Advanced Options	-	Select **Configure Now** to set the tag information, CIDR block, and controller node distribution.

Parameter	Sub-Parameter	Description
Tags	-	ModelArts can work with Tag Management Service (TMS). When creating resource-consuming tasks in ModelArts, for example, training jobs, configure tags for these tasks so that ModelArts can use tags to manage resources by group. For details about how to use tags, see **Using TMS Tags to Manage Resources by Group**. **NOTE** You can select a predefined TMS tag from the tag drop-down list or customize a tag. Predefined tags are available to all service resources that support tags. Customized tags are available only to the service resources of the user who has created the tags.
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.
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 **Next** and confirm the settings. Then, click **Submit** to create the dedicated resource pool.

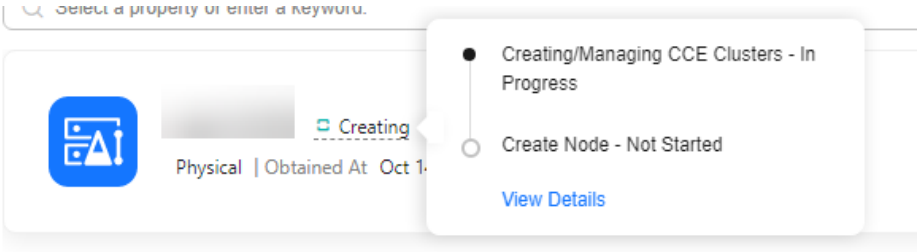
- After a resource pool is created, its status changes to **Running**. Only when the number of available nodes is greater than 0, tasks can be delivered to this resource pool.

Figure 3-6 Viewing a resource pool



- Hover over **Creating** to view the details about the creation process. Click **View Details** to go the operation record page.

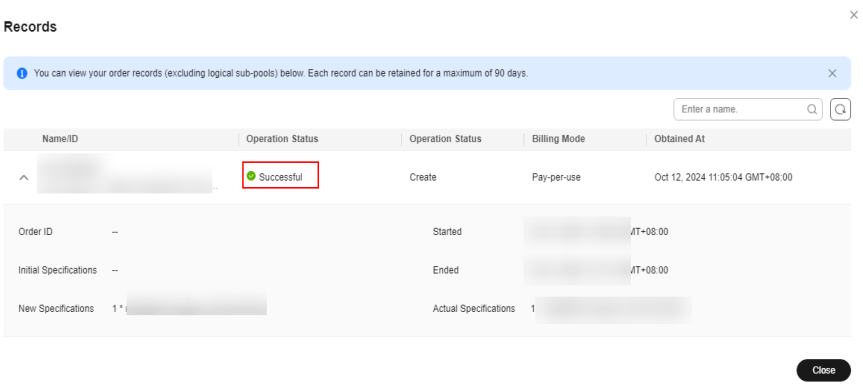
Figure 3-7 Creating



- You can view the task records of the resource pool by clicking **Records** in the upper right corner of the resource pool list.

Figure 3-8 Operation records

Figure 3-9 Viewing the resource pool status



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.

3.3 Managing Standard Dedicated Resource Pools

3.3.1 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 **AI Dedicated Resource Pools** > **Elastic Clusters**.
- Click next to the resource pool type or status in the table header. In the top right corner of the list, select **Name** or **Resource ID** to filter resource pools. To obtain the resource ID, go to the **Billing Center** > **Orders** > **My Orders** page and click **Details** in the **Operation** column of the target order.
- 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. Click **More** in the top right corner to perform operations such as resize or delete the resource pool. The available operations vary depending on the resource pool.
  - In the **Network** area of **Basic Information**, you can click the number of resource pools associated to view associated resource pools.
  - In the extended information area, you can view the monitoring information, jobs, nodes, specifications, tags, and events. For details, see the following section.

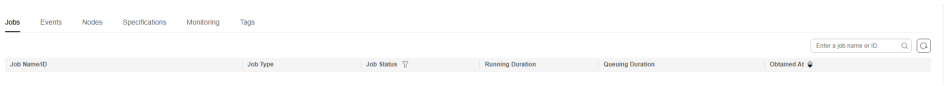
Viewing Jobs in a Resource Pool

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.

Figure 3-10 Jobs



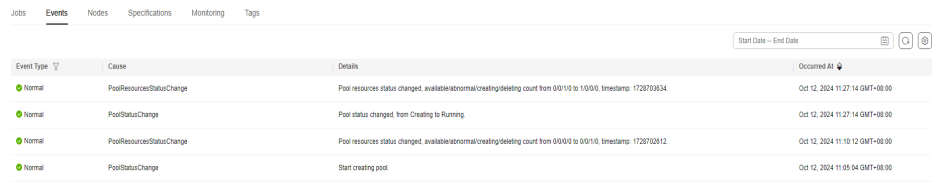
Viewing Resource Pool Events

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.

Figure 3-11 Events



Viewing Resource Pool Nodes

On the resource pool details page, click **Nodes**. You can view all nodes in the resource pool and the resource usage of each node.

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. Currently, you can replace and reset nodes. For details, see **Rectifying a Faulty Node in a Standard Dedicated Resource Pool**.

Viewing Resource Pool Specifications

On the resource pool details page, click **Specifications**. You can view the specifications used by the resource pool and the number of each specification. If

the container engine size is not configured when you create the resource pool, the default value is displayed.

Figure 3-12 Viewing resource pool specifications

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.

Figure 3-13 Viewing monitoring metrics

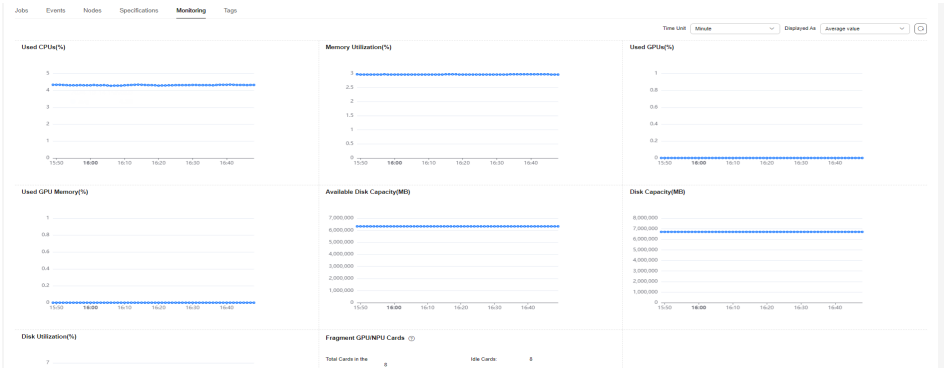


Table 3-2 Monitoring metrics

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	GPU memory used by a measured object	MB	≥0
Used NPUs	Percentage of the used NPU memory to the total NPU memory	%	0%–100%

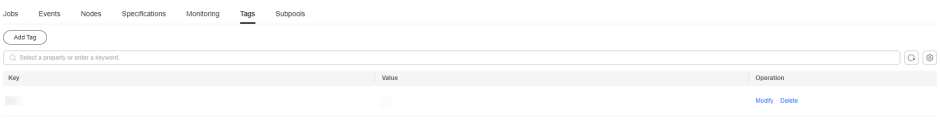
Parameter	Description	Unit	Value Range
Used NPU Memory	NPU memory used by a measured object	≥0	≥0
Available Disk Capacity (GB)	Available disk capacity of a measured object	≥0	≥0
Disk Capacity (GB)	Total disk capacity of a monitored object	≥0	≥0
Disk Usage	Disk usage of the monitored object	%	0%–100%
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.	/	/

Viewing Tags

You can add tags to a resource pool for quick search.

On the resource pool details page, click **Tags**. You can view, add, modify, and delete tags of a resource pool. For details about how to use tags, see **How Does ModelArts Use Tags to Manage Resources by Group?**

Figure 3-14 Tags



NOTE

You can add up to 20 tags.

3.3.2 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 pool in ModelArts.

- You can add nodes for existing flavors in the resource pool.
- You can delete nodes 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 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:

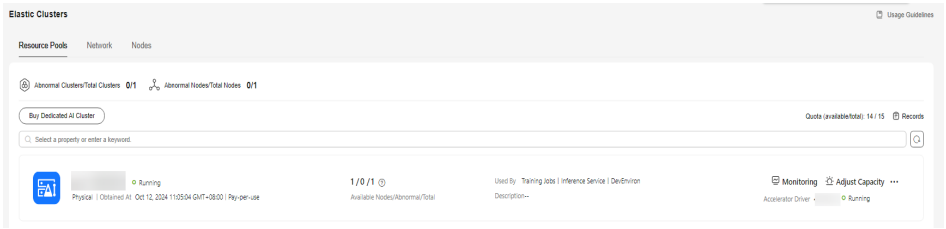
- Adjusting the number of nodes of existing specifications
- Resizing the container engine space

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

   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 or unsubscribe from it.

   Figure 3-15 Viewing resource pools

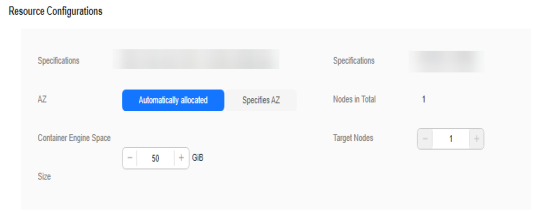
2. Add or delete nodes.

   To resize a resource pool, locate the resource pool on the **Elastic Clusters** page, and click **Adjust Capacity**. For Yearly/Month resource pools, only **Expand Capacity** is displayed on the page. To scale in such a resource pool, click the resource pool name to access the details page, and unsubscribe some nodes.

In the **Resource Configurations** area, set **AZ** to **Automatically allocated** or **Specifies AZ**. Click **Submit** and then **OK** to save the changes.

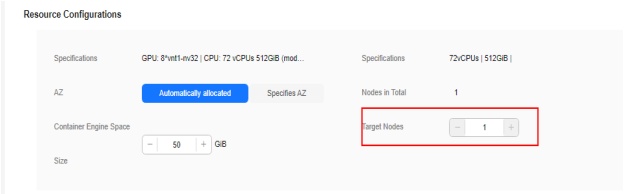
- If you select **Automatically allocated**, you can increase or decrease the number of target nodes to implement scaling. This allows you to adjust the number of nodes based on service requirements. Scale-out is the process of increasing the number of target nodes, while scale-in is the process of decreasing the number of target nodes. After the scaling, nodes are automatically allocated to AZs.
- If you select **Specifies AZ**, you can allocate nodes to different AZs.

Figure 3-16 Configuring resources on a single node



When you purchase a resource pool, the nodes for certain specifications can be purchased by rack. When you resize the resource pool, the nodes are also added or deleted by rack. You can choose to purchase nodes by rack when creating a resource pool, which cannot be modified when resizing a resource pool. You can configure the rack quantity to change the number of **Target Nodes**.

Figure 3-17 Configuring resources by rack



NOTE

When you add nodes, you can specify the billing mode that is not the mode for charging resource pools. 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.

3. Resizing the container engine

   If you need larger container engine size, perform any of the following operations:

   - For new resource pools, you can specify the container engine space when creating a resource pool. For details, see advanced configurations of **Specification Management** in **Creating a Standard Dedicated Resource Pool**.
   - For existing resource pools, you can specify the container engine space to add nodes.

- Method 1: Click the resource pool name to view its details. In the **Specifications** tab, locate the specification, and click **Change the container engine space size** in the **Operation** column. The space will be modified automatically.
- Method 2: Locate the resource pool and click **Adjust Capacity**. If it is a yearly/monthly resource pool, only **Expand Capacity** is displayed.

NOTICE

Resizing the container engine space is only applicable to new nodes. Furthermore, dockerBaseSize may vary across nodes of this flavor within the resource pool. Consequently, this can lead to discrepancies in the status of tasks distributed among different nodes.

Figure 3-18 Resizing the container engine in the **Specifications** tab

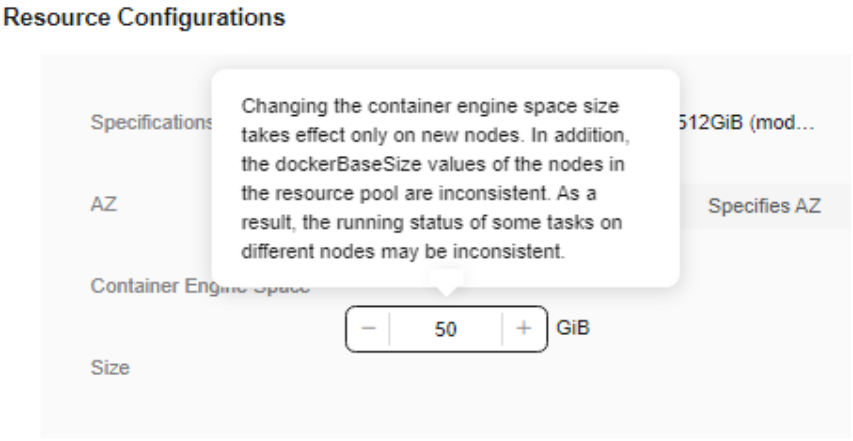
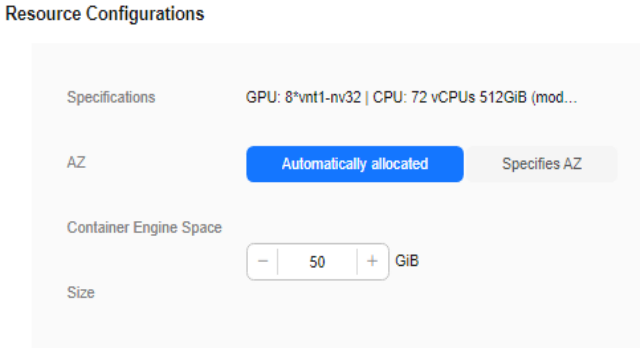


Figure 3-19 Resizing the container engine on the **Resize** page



3.3.3 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 Huawei engineers.

Upgrading the Driver

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**.
2. In the resource pool list, locate the target resource pool, and choose ··· > **Upgrade Driver** in the **Operation** column.
3. The **Upgrade Driver** dialog box displays the driver type, number of nodes, current version, target version, and upgrade mode of the dedicated resource pool. Modify the following parameters:
   - **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**: Select **Secure upgrade** or **Forcible upgrade**.
   - **Rolling Mode**: Once enabled, you can upgrade the driver in rolling mode. Currently, **By node percentage** and **By node quantity** are supported.
     - **By node percentage**: The number of nodes to be upgraded is the percentage multiplied by the total number of nodes in the resource pool.
     - **By node quantity**: The number of nodes to be upgraded is the value of this parameter.

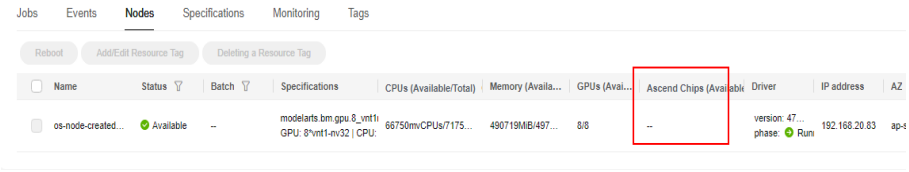
     For different upgrade modes, the policies for upgrading nodes are different.

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

NOTE

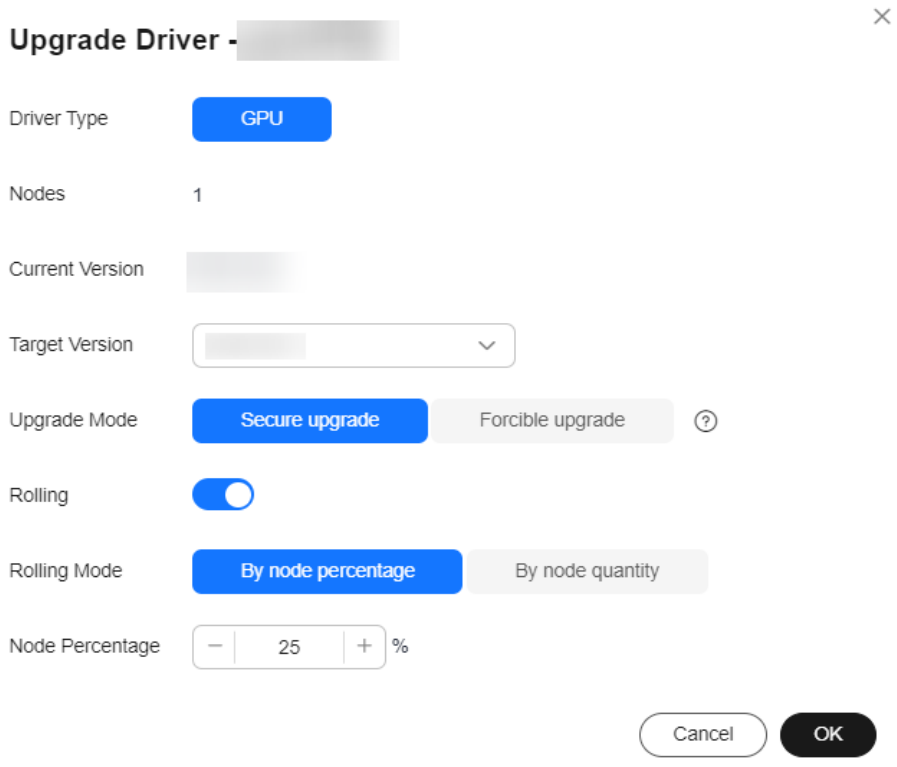
- To check whether a node has any service, go to the resource pool details page. In the **Nodes** tab, check whether all GPUs and Ascend chips are available. If yes, the node has no services.

  Figure 3-20 Checking whether a node has services



- During the rolling upgrade, the nodes with abnormal drivers do not affect the upgrade and will also be upgraded.

Figure 3-21 Upgrading a driver



4. Click **OK** to start the driver upgrade.

3.3.4 Rectifying a Faulty Node in a Standard Dedicated Resource Pool

This section describes how to rectify the faulty nodes. Currently, you can replace, reset, and restart a node, as well as configuring an HA redundant 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.

  In the **Nodes** tab, locate the node to be replaced, 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.

How do I set HA nodes?

- 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 3-22 Enabling HA redundancy

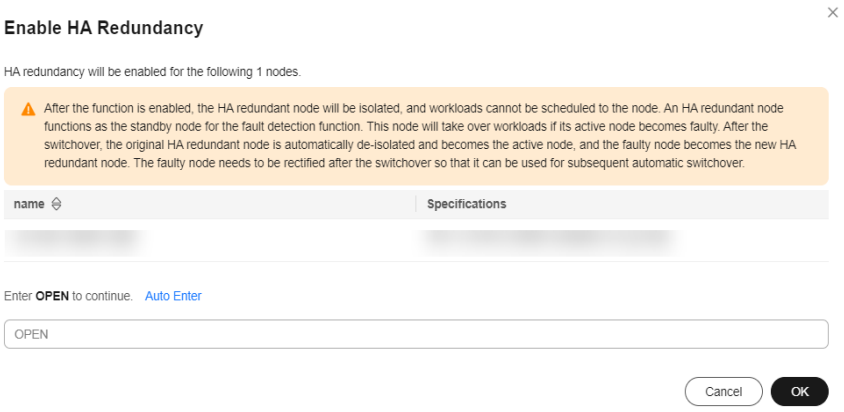
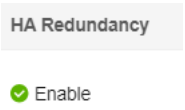


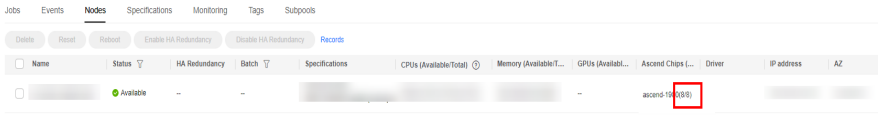
Figure 3-23 HA redundant node



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.

Figure 3-24 Checking whether a node has 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 3-25 Disabling HA redundancy

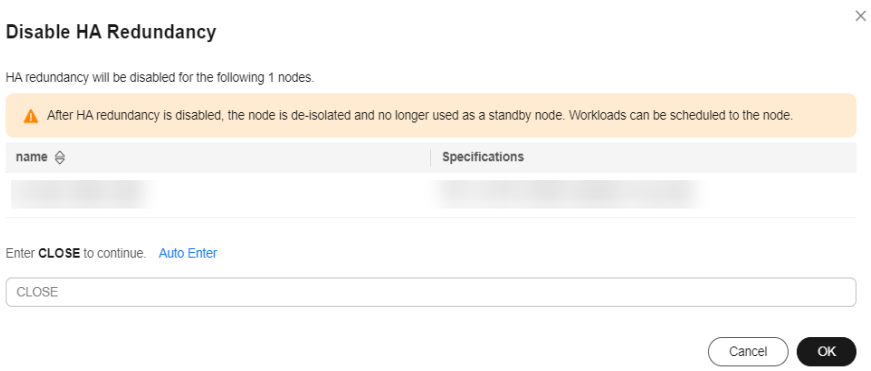
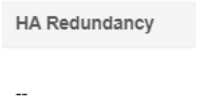


Figure 3-26 Non-HA redundancy



- 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.

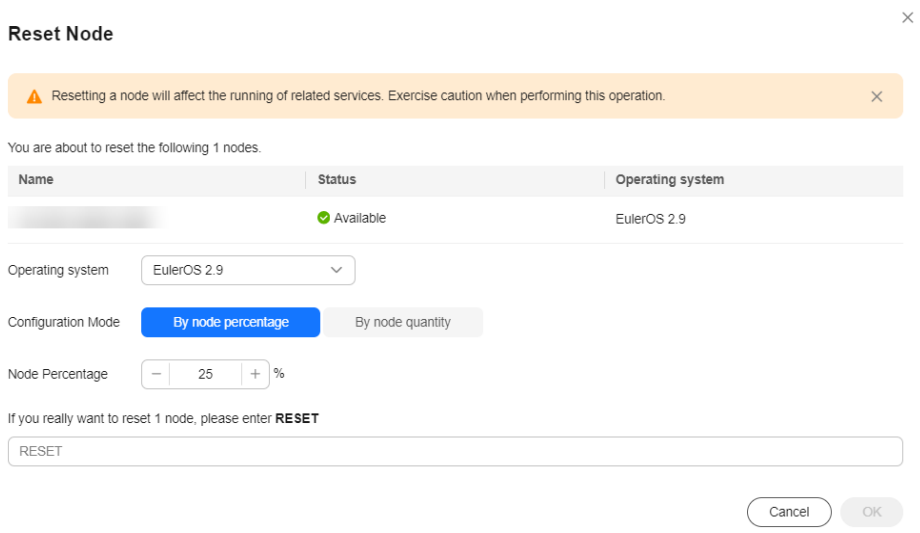
Configure the parameters described in the table below.

Table 3-3 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 3-27 Resetting a node



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 3-28 Viewing resource pool nodes

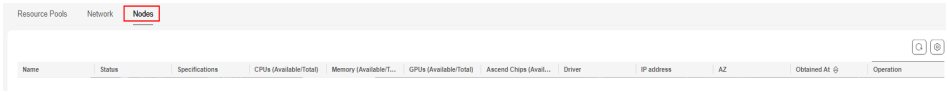
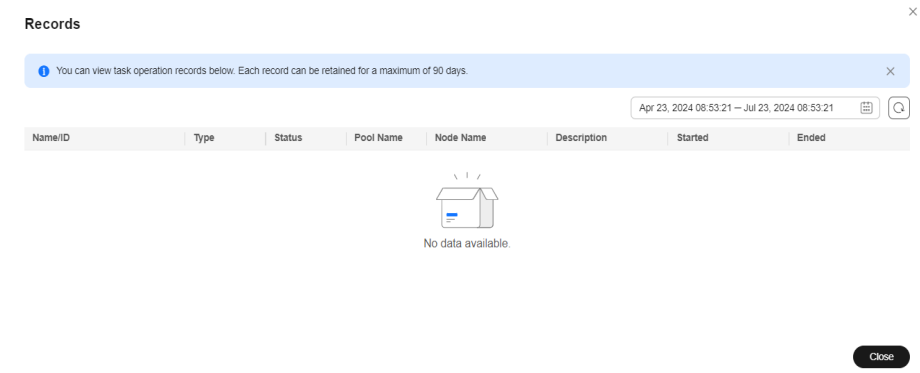


Figure 3-29 Operation records



- Restarting the node:

  In the **Nodes** tab, locate the target node, and click **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.

Figure 3-30 Restarting a node

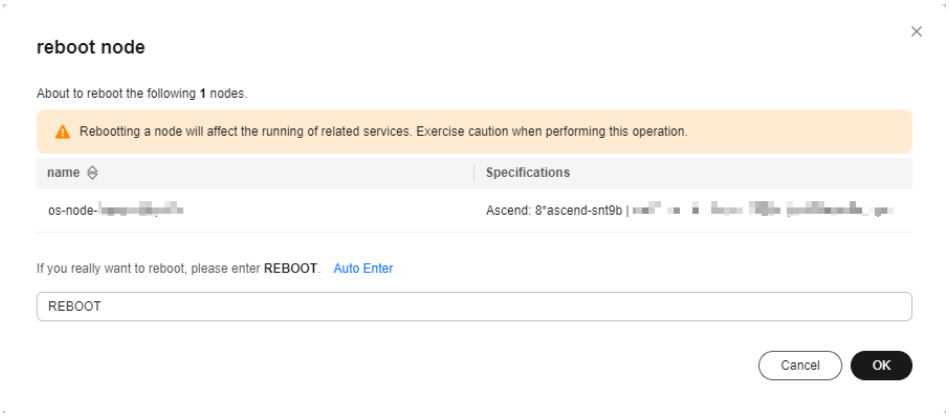
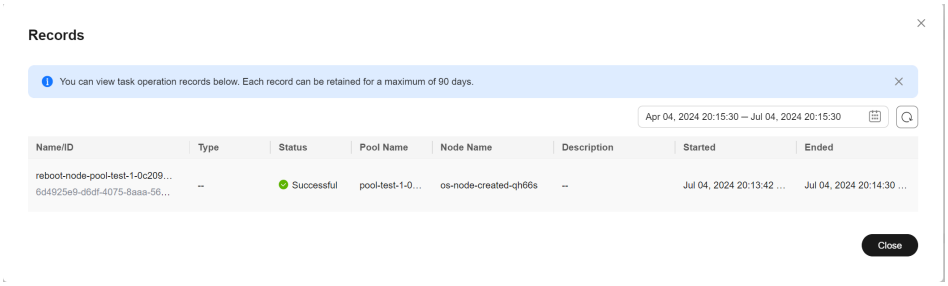


Figure 3-31 Operation records



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, unsubscribing from, or releasing a node
  - For a pay-per-use resource pool, 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.

3.3.5 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

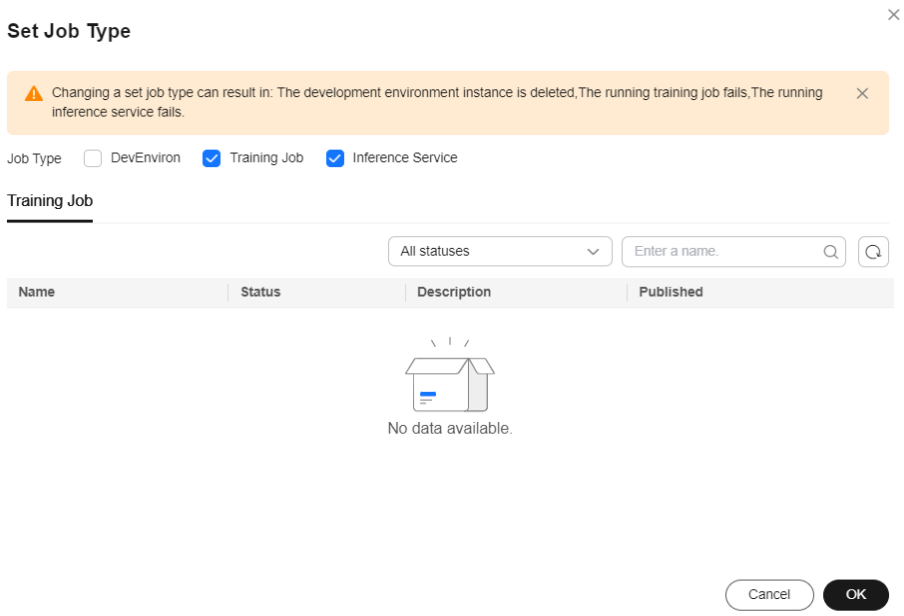
The target dedicated resource pool must be running.

Procedure

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

2. In the **Operation** column of a resource pool, choose **More** > **Set Job Type**.
3. In the **Set Job Type** dialog box, select job types.

Figure 3-32 Setting the job type



4. Click **OK**.

3.3.6 Migrating Standard Dedicated Resource Pools and Networks to Other Workspaces

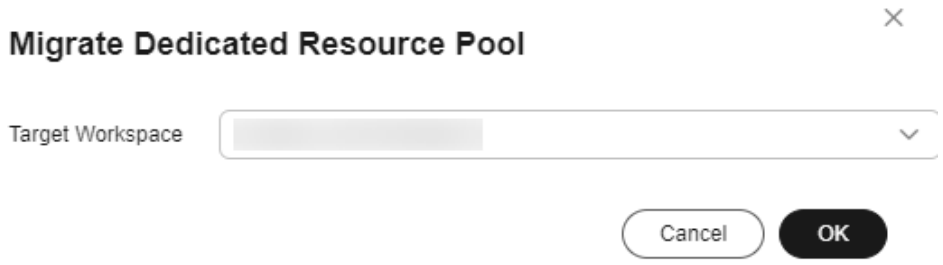
Background

The workspace of a dedicated resource pool is associated with an enterprise project, which involves bill collection. ModelArts provides workspaces to isolate resource operation permissions of different IAM users. Workspace migration includes resource pool migration and network migration. For details, see the following sections.

Migrating the Workspace for a Resource Pool

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**.
2. In the resource pool list, choose **More** > **Migrate Workspace** in the **Operation** column of the target resource pool.
3. In the displayed **Migrate Dedicated Resource Pool** dialog box, select the target workspace and click **OK**.

Figure 3-33 Migrating the Workspace



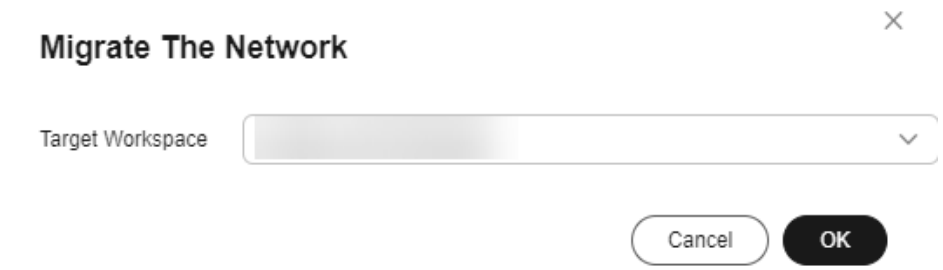
NOTE

IAM users can only migrate resource pools in their own workspaces.

Migrating the Workspace for a Network

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Dedicated Resource Pools** > **Elastic Clusters**. Then, click the **Network** tab.
2. In the network list, choose **More** > **Migrate Workspace** in the **Operation** column of the target network.
3. In the displayed dialog box, select the target workspace and click **OK**.

Figure 3-34 Migrating the workspace



NOTE

IAM users can only migrate networks in their own workspaces.

3.3.7 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

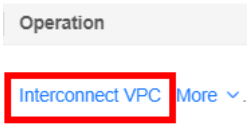
- You have obtained the ECS where the SNAT function is to be deployed.
- The ECS where the SNAT function is to be deployed runs the Linux OS.
- The ECS where the SNAT function is to be deployed has only one network interface card (NIC) configured.

Step 1: Interconnect with the VPC

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

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

   Figure 3-35 Interconnect VPC

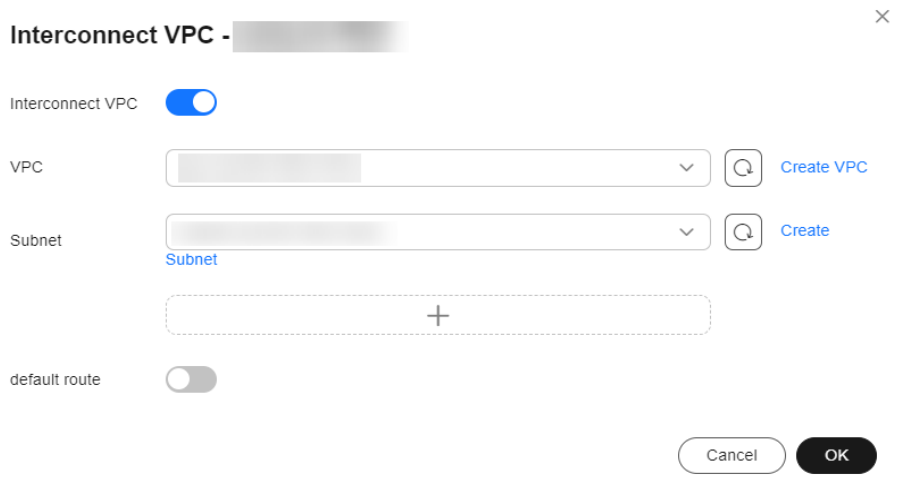
   

2. In the displayed dialog box, click the button on the right of **Interconnect VPC**, and select an available VPC and subnet from the drop-down lists.

   NOTE

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

   Figure 3-36 Parameters for interconnecting a VPC with a network

   - 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. 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**.

3.3.8 Using TMS Tags to Manage Resources by Group

ModelArts can work with Tag Management Service (TMS). When creating resource-consuming tasks in ModelArts, for example, training jobs, configure tags for these tasks so that ModelArts can use tags to manage resources by group.

ModelArts allows you to configure tags when you create training jobs, notebook instances, or real-time inference services.

Operation Process

1. **Step 1 Create Predefined Tags on TMS**
2. **Step 2 Add a Tag to a ModelArts Task**
3. **Step 3 Obtain ModelArts Resource Usage by Resource Type in TMS**

Step 1 Create Predefined Tags on TMS

Log in to the TMS console and create tags on the **Predefined Tags** page. The created tags are global and can be used in all Huawei Cloud regions.

Step 2 Add a Tag to a ModelArts Task

When creating a notebook instance, training job, or real-time inference services in ModelArts, configure a tag for the task.

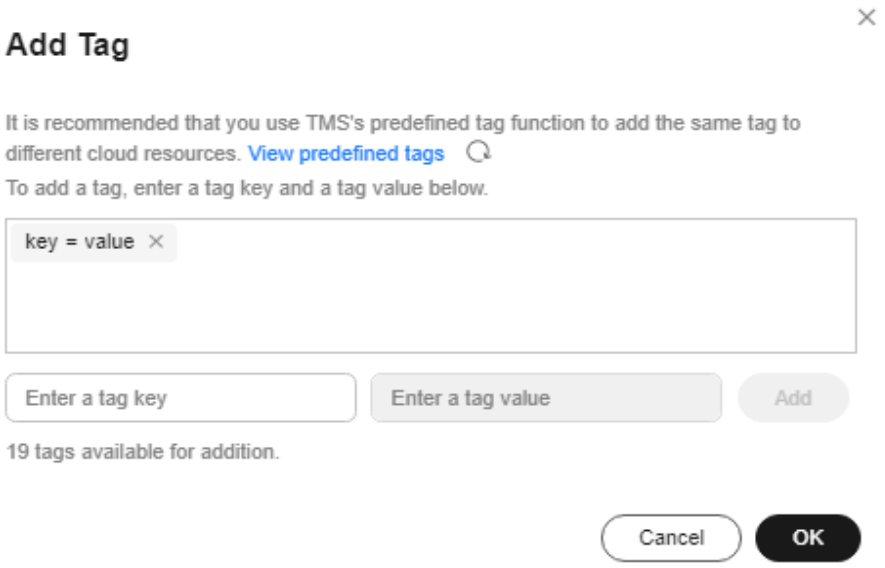
- Add a tag to a ModelArts notebook instance.

  Add a tag when you create a notebook instance. Alternatively, after creating a notebook instance, add a tag on the **Tags** tab on the instance details page.
- Add a tag to a ModelArts training job.

  Add a tag when you create a training job. Alternatively, after creating a training job, add a tag on the **Tags** tab on the job details page.
- Add a tag to a ModelArts real-time service.

  Add a tag when you create a real-time service. Alternatively, after creating a real-time service, add a tag on the **Tags** tab on the service details page.

Figure 3-37 Adding a Label



NOTE

When adding a tag to a ModelArts task, you can create new tags by specifying the keys and values of the new tags. The created tags are available for only the current project.

Step 3 Obtain ModelArts Resource Usage by Resource Type in TMS

Log in to the TMS console. On the **Resources Tag** page, view resource tasks in specified regions based on resource types and tags.

- **Region**: For details about Huawei Cloud regions, see **What Are Regions and AZs?**
- **Resource Type**: **Table 3-4** lists the resource types that can be viewed on ModelArts.
- **Resource Tag**: If no tag is specified, all resources are displayed, regardless of whether the resources are configured with tags. One or multiple tags can be selected to obtain resource usage.

Table 3-4 Resource types that can be viewed on ModelArts

Resource Type	Description
ModelArts-Notebook	Notebook instances in ModelArts DevEnviron
ModelArts-TrainingJob	ModelArts training jobs
ModelArts-RealtimeService	ModelArts real-time inference services
ModelArts-ResourcePool	ModelArts dedicated resource pools

> **NOTE**
>
> If your organization has configured tag policies for ModelArts, add tags to resources based on the policies. If a tag does not comply with the tag policies, resource creation may fail. Contact your organization administrator to learn more about tag policies.

3.3.9 Releasing Standard Dedicated Resource Pools and Deleting the Network

Deleting a Resource Pool

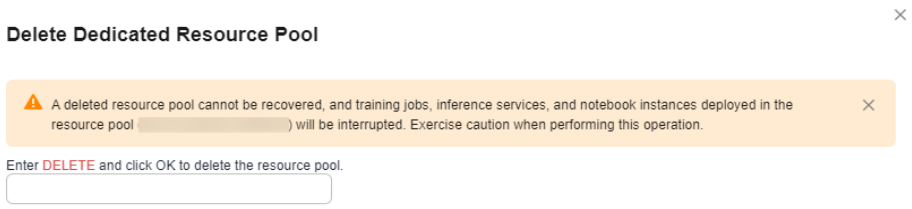
If a dedicated resource pool is no longer needed for AI service development, you can delete the resource pool to release resources.

> **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 **AI Dedicated Resource Pools** > **Elastic Clusters**.
2. Locate the the target resource pool 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.

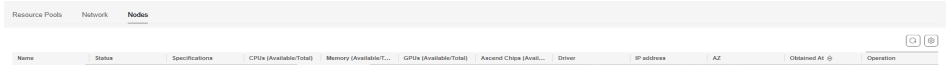
Figure 3-38 Deleting a resource pool



Releasing a Free Node

Nodes that are not managed by the resource pool are considered as free nodes. To view the information about free nodes, log in to the ModelArts management console, choose **Dedicated Resource Pools** > **Elastic Cluster**, and click the **Nodes** tab.

Figure 3-39 Nodes



Release the free nodes resources according to the following content:

- For a yearly/monthly node whose resources are not expired, click **Unsubscribe** in the **Operation** column.
- For a yearly/monthly node whose resources are expired (in the grace period), click **Release** in the **Operation** column.

NOTE

Unsubscription and release operations cannot be undone. Exercise caution when performing this operation.

Deleting a Network

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

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

4 Using ExeML for Zero-Code AI Development

4.1 Introduction to ExeML

ExeML Functions

ModelArts ExeML is a customized code-free model development tool that helps you start codeless AI application development with high flexibility. ExeML automates model design, parameter tuning and training, and model compression and deployment based on the labeled data. With ExeML, you only need to upload data and perform simple operations as prompted on the ExeML GUI to train and deploy models.

You can use ExeML to quickly build models for sound classification, text classification, image classification, predictive analytics, and object detection. ExeML is widely used in industrial, retail, and security sectors.

- Image classification: identifies a class of objects in images.
- Object detection: identifies the position and class of each object in an image.
- Predictive analytics: classifies or predicts structured data.
- Sound classification: classifies and identifies different sounds.
- Text classification: identifies the category of a piece of text.

ExeML Process

With ModelArts ExeML, you can develop AI models without coding. You only need to upload data, create a project, label the data, train a model, and deploy the trained model. For details, see **Figure 4-1**. In the new-version ExeML, this process can be finished by a workflow. You can develop a DAG through a workflow. DAG execution is to use a task execution template to perform data labeling, dataset publishing, model training, model registration, and service deployment in sequence. For more information about workflows, see What Is Workflow?.

Figure 4-1 ExeML process

ExeML Projects

- **Image Classification**

  An image classification project aims to classify images. You only need to add images and label them. Then, an image classification model can be quickly generated for automatically classifying offerings, vehicle types, and defective goods. For example, in the quality check scenario, you can upload a product image, label the image as qualified or unqualified, and train and deploy a model to inspect product quality.

- **Object Detection**

  An object detection project aims to identify the class and location of objects in images. You only need to add images and label objects in the images with proper bounding boxes. The labeled images will be used as a training set for building a model to identify multiple objects or provide the number of objects in a single image. Object detection can also be used to inspect employees' dress code and perform unattended inspection of article placement.

- **Predictive Analytics**

  A predictive analytics project is an automated model training application for structured data, which can classify or predict structured data. Predictive analytics can be used for user profile analysis and targeted marketing, as well as predictive maintenance of manufacturing equipment based on real-time data to identify equipment faults.

- **Sound Classification**

  A sound classification project identifies whether a certain sound is contained in an audio file. Sound classification can be used to monitor abnormal sounds in production or security scenarios.

- **Text Classification**

  A text classification project identifies the class of a piece of text. It can be used in emotion analysis or news classification.

4.2 Using ExeML for Image Classification

4.2.1 Preparing Image Classification Data

Before using ModelArts ExeML to build a model, upload data to an OBS bucket. The OBS bucket and ModelArts must be in the same region.

Requirements on Datasets

- Check that all images are undamaged and in a compatible format. The supported formats are JPG, JPEG, BMP, and PNG.

- Do not store data of different projects in the same dataset.
- Collect at least two classes of images with a similar number of images in each class. Make sure each class has a minimum of 20 images.
- To ensure the prediction accuracy of models, the training samples must be similar to the real-world use cases.
- To ensure the generalization capability of models, datasets should cover all possible scenarios.

Uploading Data to OBS

In this section, the OBS console is used to upload data.

Upload files to OBS according to the following specifications:

- The name of files cannot contain plus signs (+), spaces, or tabs.
- If you do not need to upload training data in advance, create an empty folder to store files generated in the future, for example, **/bucketName/data-cat**.
- If you need to upload images to be labeled in advance, create an empty folder and save the images in the folder. An example of the image directory structure is **/bucketName/data-cat/cat.jpg**.
- If you want to upload labeled images to the OBS bucket, upload them according to the following specifications:
  - The dataset for image classification requires storing labeled objects and their label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object is **10.jpg**, the name of the label file must be **10.txt**.

    Example of data files:

    ```
    ├─<dataset-import-path>
    │       10.jpg
    │       10.txt
    │       11.jpg
    │       11.txt
    │       12.jpg
    │       12.txt
    ```

  - Only images in JPG, JPEG, PNG, and BMP formats are supported. When uploading images on the OBS console, ensure that the size of an image does not exceed 5 MB and the total size of images to be uploaded in one attempt does not exceed 8 MB. If the data volume is large, use OBS Browser+ to upload images.
  - A label name can contain a maximum of 32 characters, including letters, digits, hyphens (-), and underscores (_).
  - The specifications of image classification label files (.txt) are as follows:

    Each row contains only one label.

    ```
    flower
    book
    ...
    ```

Procedure for uploading data to OBS:

Perform the following operations to upload data to OBS for model training and building.

1. Log in to OBS Console and **create a bucket** in the same region as ModelArts. If an available bucket exists, ensure that the OBS bucket and ModelArts are in the same region.
2. **Upload the local data** to the OBS bucket. If you have a large amount of data, use OBS Browser+ to upload data or folders. The uploaded data must meet the dataset requirements of the ExeML project.

NOTE

Upload data from unencrypted buckets. Otherwise, training will fail because data cannot be decrypted.

Creating a Dataset

After the data preparation is completed, create a dataset of the type supported by the project. For details, see **Creating a Dataset**.

4.2.2 Creating an Image Classification Project

ModelArts ExeML supports sound classification, text classification, image classification, predictive analytics, and object detection projects. You can create any of them based on your needs. Perform the following operations to create an ExeML project.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. Click **Create Project** in the box of your desired project. The page for creating an ExeML project is displayed.
3. On the project creation page, set parameters by referring to **Table 4-1**.

Table 4-1 Parameters

Parameter	Description
Name	Name of an ExeML project • Enter a maximum of 64 characters. Only digits, letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory. • Start with a letter. • The name must be unique.
Description	Brief description of a project
Dataset	You can select a dataset or click **Create Dataset** to create one. • Existing dataset: Select a dataset from the drop-down list box. Only datasets of the same type are displayed. • Creating a dataset: Click **Create Dataset** to create a dataset. For details, see **Creating a Dataset**.

Parameter	Description
Output Path	Select an OBS path for storing ExeML data. **NOTE** The output path stores all data generated in the ExeML project.
Training Flavor	Select a training flavor for this ExeML project. You will be billed based on different flavors. **NOTE** • You can choose the package that you have bought. In the configuration fee area, you will see your remaining package quota and how much you will pay for any extra usage.

4. Click **Create Project**. Then, the ExeML workflow is displayed.
5. Wait until the workflow of the image classification project executes the following phases in sequence:
   a. **Label Data**: Check data labeling.
   b. **Publish Dataset Version**: Publish a version for the labeled dataset.
   c. **Check Data**: Check whether any exception occurs in your dataset.
   d. **Classify Images**: Train the dataset of the published version to generate a model.
   e. **Register Model**: Register the trained model with model management.
   f. **Deploy Service**: Deploy the generated model as a real-time service.

Quickly Searching for a Project

On the ExeML overview page, you can use the search box to quickly search for and filter workflows based on the ExeML type (or project name).

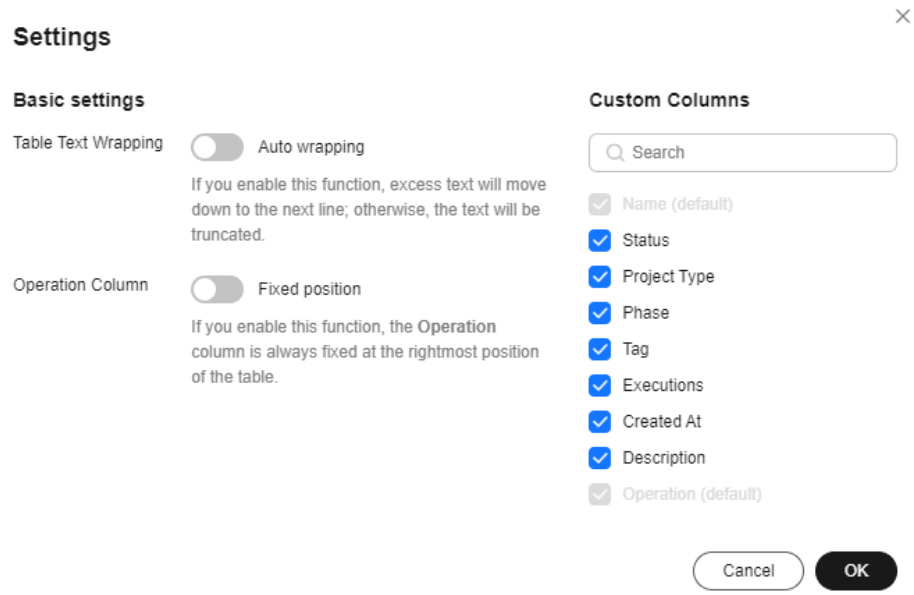
1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. In the search box above the ExeML project list, filter the desired workflows based on the required property, such as name, status, project type, current phase, and tag.
3. To adjust the basic settings of ExeML and select the columns you want to see, click  on the right of the search box.

   Table Text Wrapping: This function is disabled by default. If you enable this function, excess text will move down to the next line; otherwise, the text will be truncated.

   Operation Column: This function is disabled by default. If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.

   Custom Columns: By default, all items are selected. You can select columns you want to see.

Figure 4-2 Customizing table columns



4. Click **OK**. Then, the columns will be displayed based on the settings.
5. To arrange ExeML projects by a specific property, click in the table header.

4.2.3 Labeling Image Classification Data

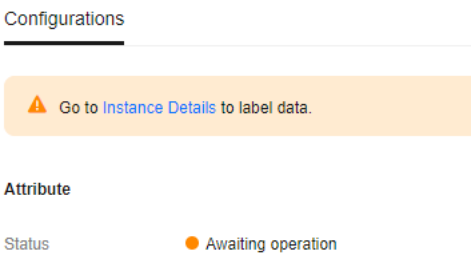
Model training requires a large number of labeled images. Therefore, before model training, add labels to the images that are not labeled. ModelArts allows you to add labels in batches by one click. You can also modify or delete labels that have been added to images.

NOTE

The number of labeled images in the dataset must be no fewer than 100. Otherwise, checking the dataset will fail, affecting your model training.

After the project is created, you will be directed to the ExeML page and the project starts to run. Click the data labeling phase. After the status changes to **Awaiting operation**, confirm the data labeling status in the dataset. You can also modify labels, add data, or delete data in the dataset.

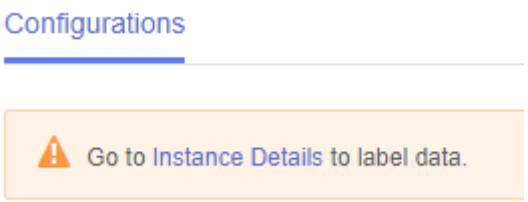
Figure 4-3 Data labeling status



Labeling Images

1. On the labeling phase of the new-version ExeML, click **Instance Details**. The data labeling page is displayed.

   Figure 4-4 Clicking Instance Details

2. Select the images to be labeled in sequence, or tick **Select Images on Current Page** to select all images on the page, and then add labels to the images in the right pane.
3. After selecting an image, input a label in the **Label** text box, or select an existing label from the drop-down list. Click **OK**. The selected image is labeled. For example, you can select multiple images containing tulips and add label **tulips** to them. Then select other unlabeled images and label them as **sunflowers** and **roses**. After the labeling is complete, the images are saved on the **Labeled** tab page.
   a. You can add multiple labels to an image.
   b. A label consists of letters, digits, hyphens (-), and underscores (_).
4. After all the images are labeled, view them on the **Labeled** tab page or view **All Labels** in the right pane to check the name and quantity of the labels.

Synchronizing or Adding Images

On the labeling phase, click **Instance Details** to go to the data labeling page. Then, add images from your local PC or synchronize image data from OBS.

Figure 4-5 Adding local images

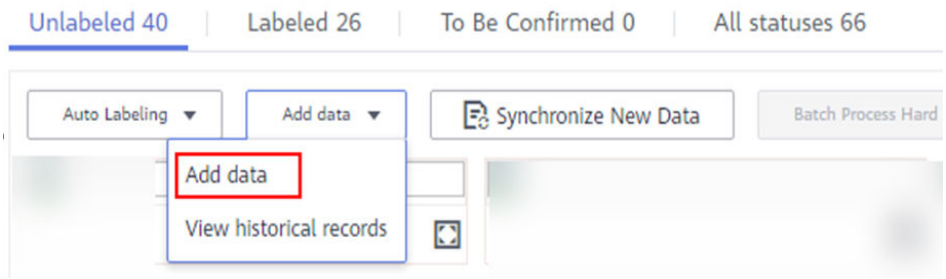
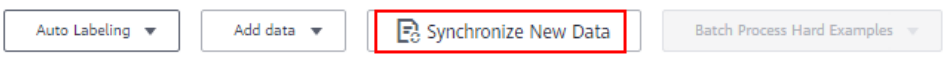


Figure 4-6 Synchronizing OBS images



- **Add data**: You can click **Add data** to quickly add images on a local PC to ModelArts. These images will be automatically synchronized to the OBS path specified during project creation.

- **Synchronize New Data**: You can upload images to the OBS directory specified during project creation and click **Synchronize New Data** to quickly add the images in the OBS directory to ModelArts.
- **Delete Image**: You can delete images one by one, or tick **Select Current Page** to delete all images on the page.

  NOTE

  The deleted images cannot be recovered. Exercise caution when performing this operation.

Modifying Labeled Data

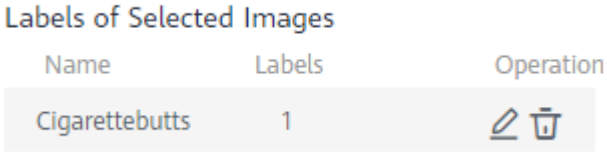
After labeling data, you can modify the labeled data on the **Labeled** tab page.

- **Modifying based on images**

  On the data labeling page, click the **Labeled** tab, and select one or more images to be modified from the image list. Modify the image information in the label information area on the right.

  - Adding a label: In the **Label** text box, select an existing label or enter a new label name, and then click .
  - Modifying a label: In the **Labels of Selected Images** area, click the editing icon in the **Operation** column, enter the correct label name in the text box, and click the check mark icon to complete the modification.

    Figure 4-7 Modifying a label

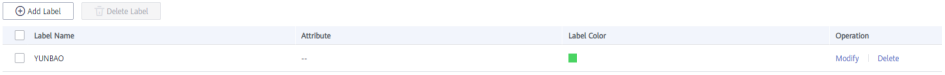
    

  - Deleting a label: In the **Labels of Selected Images** area, click in the **Operation** column to delete the label.

- **Modifying based on labels**

  On the labeling overview page, click **Label Management**. Information about all labels is displayed.

  Figure 4-8 Information about all labels

  - Modifying a label: In the **Operation** column of the target label, click **Modify**, enter the new label, and click **OK**.
  - Deleting a label: In the **Operation** column of the target label, click **Delete**, and click **OK**.

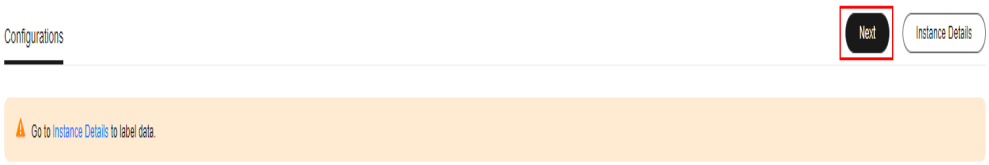
NOTE

Deleted tags cannot be restored.

Resuming Workflow Execution

After confirming data labeling, go back to the ExeML page. Click **Next**. Then, the workflow continues to run in sequence until all phases are executed.

Figure 4-9 Resuming the workflow execution



4.2.4 Training an Image Classification Model

After labeling the images, perform model training to obtain the required image classification model. Ensure that the labeled images meet the requirements specified in **Prerequisites**. Otherwise, checking the dataset will fail.

Prerequisites

1. The number of labeled images in your dataset is greater than or equal to 100.
2. At least two classes of samples are required for training, and each class with at least 5 samples.

Procedure

1. Ensure all your dataset has been labeled. For details, see **Labeling Image Classification Data**.
2. In the data labeling phase of the new-version ExeML, click **Next** and wait until the workflow enters the training phase.
3. Wait until the training is complete. No manual operation is required. If you close or exit the page, the system continues training until it is complete.
4. On the image classification phase, wait until the training status changes from **Running** to **Completed**.
5. After the training, click  on the image classification phase to view metric information. For details about the evaluation result parameters, see **Table 4-2**.

Table 4-2 Evaluation result parameters

Parameter	Description	Description
Recall	Recall	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish positive samples.
Precision	Precision	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish negative samples.
Accuracy	Accuracy	Fraction of correctly predicted samples over all samples. It shows the general ability of a model to recognize samples.
F1 Score	F1 score	Harmonic average of the precision and recall of a model. It is used to evaluate the quality of a model. A high F1 score indicates a good model.

NOTE

An ExeML project supports multiple rounds of training, and each round generates an AI application version. For example, the first training version is **0.0.1**, and the next version is **0.0.2**. The trained models can be managed by training version. After the trained model meets your requirements, deploy the model as a service.

4.2.5 Deploying an Image Classification Service

Deploying a Service

You can deploy a model as a real-time service that provides a real-time test UI and monitoring capabilities. After model training is complete, you can deploy a version with the ideal accuracy and in the **Successful** status as a service. The procedure is as follows:

1. On the phase execution page, after the service deployment status changes to **Awaiting input**, double-click **Deploy Service**. On the configuration details page, configure resource parameters.
2. On the service deployment page, select the resource specifications used for service deployment.
   - **AI Application Source**: defaults to the generated AI application.
   - **AI Application and Version**: The current AI application version is automatically selected, which is changeable.
   - **Resource Pool**: defaults to public resource pools.
   - **Traffic Ratio**: defaults to **100** and supports a value range of 0 to 100.
   - **Specifications**: Select available specifications based on the list displayed on the console. The specifications in gray cannot be used in the current

environment. If there are no specifications after you select a public resource pool, no public resource pool is available in the current environment. In this case, use a dedicated resource pool or contact the administrator to create a public resource pool.

- **Compute Nodes**: an integer ranging from 1 to 5. The default value is **1**.
- **Auto Stop**: enables a service to automatically stop at a specified time. If this function is not enabled, the real-time service continuously runs and fees are incurred accordingly. Auto stop is enabled by default and its default value is **1 hour later**.

  The auto stop options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, enter any integer from 1 to 24 in the text box on the right.

  NOTE

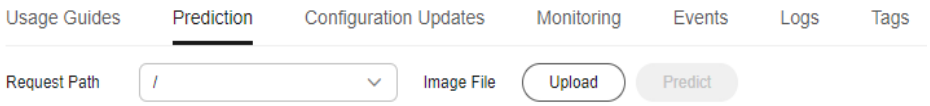
  You can choose the package that you have bought when you select specifications. On the configuration fee tag, you can view your remaining package quota and how much you will pay for any extra usage.

3. After configuring resources, click **Next**. Wait until the status changes to **Executed**. The AI application has been deployed as a real-time service.

Testing a Service

After the service is deployed, click **Instance Details** to go to the real-time service details page. Click the **Prediction** tab to test the service. You can also use code to test a service. For details, see **Accessing Real-Time Services**.

Figure 4-10 Testing the service



The following describes the procedure for performing a service test after the image classification model is deployed as a service on the ExeML page.

1. After the model is deployed, click **Instance Details** in the service deployment phase to go to the service page. On the **Prediction** tab page, click **Upload** and select a local image for test.
2. Click **Prediction** to conduct the test. After the prediction is complete, label **sunflowers** and its detection score are displayed in the prediction result area on the right. If the model accuracy does not meet your expectation, add images on the **Label Data** tab page, label the images, and train and deploy the model again. **Table 4-3** describes the parameters in the prediction result. If you are satisfied with the model prediction result, call the API to access the real-time service as prompted. For details, see **Accessing Real-Time Services**.

   Only JPG, JPEG, BMP, and PNG images are supported.

Figure 4-11 Prediction result

Table 4-3 Parameters in the prediction result

Parameter	Description
predicted_label	Image prediction label
scores	Prediction confidence of top 5 labels

NOTE

- A running real-time service continuously consumes resources. If you do not need to use the real-time service, stop the service to stop billing. To do so, click **Stop** in the **More** drop-down list in the **Operation** column. If you want to use the service again, click **Start**.
- If you enable the auto stop function, the service automatically stops after the specified time and no fee is generated.

4.3 Using ExeML for Object Detection

4.3.1 Preparing Object Detection Data

Before using ModelArts ExeML to build a model, upload data to an OBS bucket. The OBS bucket and ModelArts must be in the same region.

Requirements on Datasets

- Ensure that no damaged image exists. The supported image formats include JPG, JPEG, BMP, and PNG.
- Do not store data of different projects in the same dataset.
- To ensure the prediction accuracy of models, the training samples must be similar to the actual application scenarios.
- To ensure the generalization capability of models, datasets should cover all possible scenarios.

- In an object detection dataset, if the coordinates of the bounding box exceed the boundaries of an image, the image cannot be identified as a labeled image.

Uploading Data to OBS

In this section, the OBS console is used to upload data.

Upload files to OBS according to the following specifications:

- The name of files in a dataset cannot contain Chinese characters, plus signs (+), spaces, or tabs.
- If you do not need to upload training data in advance, create an empty folder to store files generated in the future. for example, **/bucketName/data-cat**.
- If you need to upload images to be labeled in advance, create an empty folder and save the images in the folder. An example of the image directory structure is **/bucketName/data-cat/cat.jpg**.
- If you want to upload labeled images to the OBS bucket, upload them according to the following specifications:
  - The dataset for object detection requires storing labeled objects and their label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object is **IMG_20180919_114745.jpg**, the name of the label file must be **IMG_20180919_114745.xml**.

    The label files for object detection must be in PASCAL VOC format. For details about the format, see **Table 4-4**.

    Example of data files:

    ```
    ├─<dataset-import-path>
    │       IMG_20180919_114732.jpg
    │       IMG_20180919_114732.xml
    │       IMG_20180919_114745.jpg
    │       IMG_20180919_114745.xml
    │       IMG_20180919_114945.jpg
    │       IMG_20180919_114945.xml
    ```

  - Images in JPG, JPEG, PNG, and BMP formats are supported. When uploading images on the OBS console, ensure that image is no larger than 5 MB and the total size of images to be uploaded in one attempt does not exceed 8 MB. If the data volume is large, use OBS Browser+ to upload images.
  - A label name can contain a maximum of 32 characters, including Chinese characters, letters, digits, hyphens (-), and underscores (_).

Table 4-4 PASCAL VOC format description

Field	Mandatory	Description
folder	Yes	Directory where the data source is located
filename	Yes	Name of the file to be labeled

Field	Mandatory	Description
size	Yes	Image pixel • **width**: image width. This parameter is mandatory. • **height**: image height. This parameter is mandatory. • **depth**: number of image channels. This parameter is mandatory.
segmented	Yes	Segmented or not
object	Yes	Object detection information. Multiple **object{}** functions are generated for multiple objects. • **name**: class of the labeled object. This parameter is mandatory. • **pose**: shooting angle of the labeled object. This parameter is mandatory. • **truncated**: whether the labeled object is truncated (**0** indicates that the object is not truncated). This parameter is mandatory. • **occluded**: whether the labeled object is occluded (**0** indicates that the object is not occluded). This parameter is mandatory. • **difficult**: whether the labeled object is difficult to identify (**0** indicates that the object is easy to identify). This parameter is mandatory. • **confidence**: confidence score of the labeled object. The value ranges from **0** to **1**. This parameter is optional. • **bndbox**: bounding box type. This parameter is mandatory. For details about the possible values, see **Table 4-5**.

Table 4-5 Description of bounding box types

type	Shape	Labeling Information
bndbox	Rectangle	Coordinates of the upper left and lower right points <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>

Example of the label file in KITTI format:

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>bag</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <bndbox>
        <xmin>108</xmin>
        <ymin>101</ymin>
        <xmax>251</xmax>
        <ymax>238</ymax>
     </bndbox>
  </object>
</annotation>
```

Procedure for uploading data to OBS:

Perform the following operations to import data to the dataset for model training and building.

1. Log in to OBS Console and **create a bucket** in the same region as ModelArts. If an available bucket exists, ensure that the OBS bucket and ModelArts are in the same region.
2. **Upload the local data** to the OBS bucket. If you have a large amount of data, use OBS Browser+ to upload data or folders. The uploaded data must meet the dataset requirements of the ExeML project.

NOTE

- Upload data from unencrypted buckets. Otherwise, training will fail because data cannot be decrypted.
- At least two classes of samples are required for training, and each class must have at least 50 samples.

Creating a Dataset

After the data preparation is completed, create a dataset of the type supported by the project. For details, see **Creating a Dataset**.

4.3.2 Creating an Object Detection Project

ModelArts ExeML supports sound classification, text classification, image classification, predictive analytics, and object detection projects. You can create any of them based on your needs. Perform the following operations to create an ExeML project.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.

2. Click **Create Project** in the box of your desired project. The page for creating an ExeML project is displayed.
3. On the project creation page, set parameters by referring to **Table 4-6**.

Table 4-6 Parameters

Parameter	Description
Name	Name of a project • Enter a maximum of 64 characters. Only digits, letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory. • Start with a letter. • The name must be unique.
Description	Brief description of a project
Dataset Source	You can select a dataset or click **Create Dataset** to create one. • Existing dataset: Select a dataset from the drop-down list box. Only datasets of the same type are displayed. • Creating a dataset: Click **Create Dataset** to create a dataset. For details, see **Creating a Dataset**.
Output Path	An OBS path for storing ExeML data **NOTE** The output path stores all data generated in the ExeML project.
Training Flavor	Select a training flavor for this ExeML project. You will be billed based on different flavors. **NOTE** • You can choose the package that you have bought. In the configuration fee area, you will see your remaining package quota and how much you will pay for any extra usage.

4. Click **Create Project**. Then, the ExeML workflow is displayed.
5. Wait until the workflow of the object detection project executes the following phases in sequence:
   a. **Label Data**: Check data labeling.
   b. **Publish Dataset Version**: Publish a version for the labeled dataset.
   c. **Check Data**: Check whether any exception occurs in your dataset.
   d. **Detect Objects**: Train the dataset of the published version to generate a model.
   e. **Register Model**: Register the trained model with model management.
   f. **Deploy Service**: Deploy the generated model as a real-time service.

Quickly Searching for a Project

On the ExeML overview page, you can use the search box to quickly search for and filter workflows based on the ExeML type (or project name).

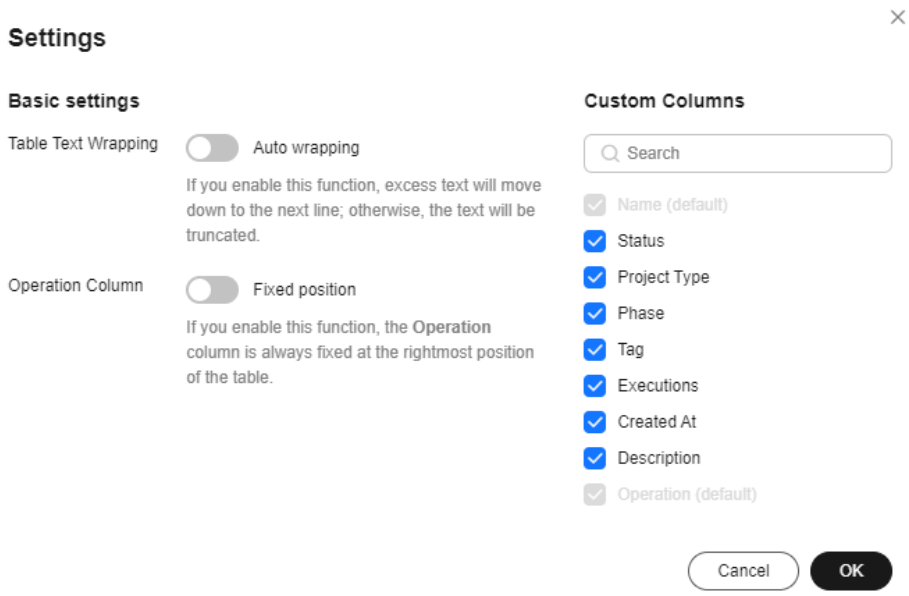
1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. In the search box above the ExeML project list, filter the desired workflows based on the required property, such as name, status, project type, current phase, and tag.
3. To adjust the basic settings of ExeML and select the columns you want to see, click  on the right of the search box.

   Table Text Wrapping: This function is disabled by default. If you enable this function, excess text will move down to the next line; otherwise, the text will be truncated.

   Operation Column: This function is disabled by default. If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.

   Custom Columns: By default, all items are selected. You can select columns you want to see.

   Figure 4-12 Customizing table columns

4. Click **OK**. Then, the columns will be displayed based on the settings.
5. To arrange ExeML projects by a specific property, click  in the table header.

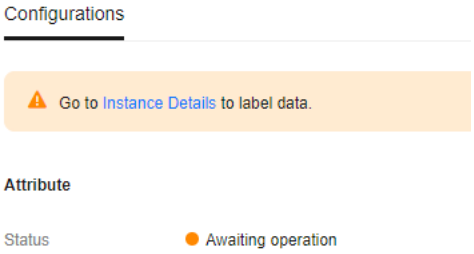
4.3.3 Labeling Object Detection Data

Before data labeling, consider how to design labels. The labels must correspond to the distinct characteristics of the detected images and are easy to identify (the detected object in an image is highly distinguished from the background). Each label specifies the expected recognition result of the detected images. After the label design is complete, prepare images based on the designed labels. It is recommended that the number of all images to be detected be greater than 100. If the labels of some images are similar, prepare more images. At least two classes of samples are required for training, and each class with at least 50 samples.

- During labeling, the variance of a class should be as small as possible. That is, the labeled objects of the same class should be as similar as possible. The labeled objects of different classes should be as different as possible.
- The contrast between the labeled objects and the image background should be as stark as possible.
- In object detection labeling, a target object must be entirely contained within a labeling box. If there are multiple objects in an image, do not relabel or miss any objects.

After a project is created, you will be redirected to the new-version ExeML and the project starts to run. When the data labeling phase changes to **Awaiting operation**, manually confirm data labeling in the dataset. You can also add or delete data in the dataset and modify labels.

Figure 4-13 Data labeling status



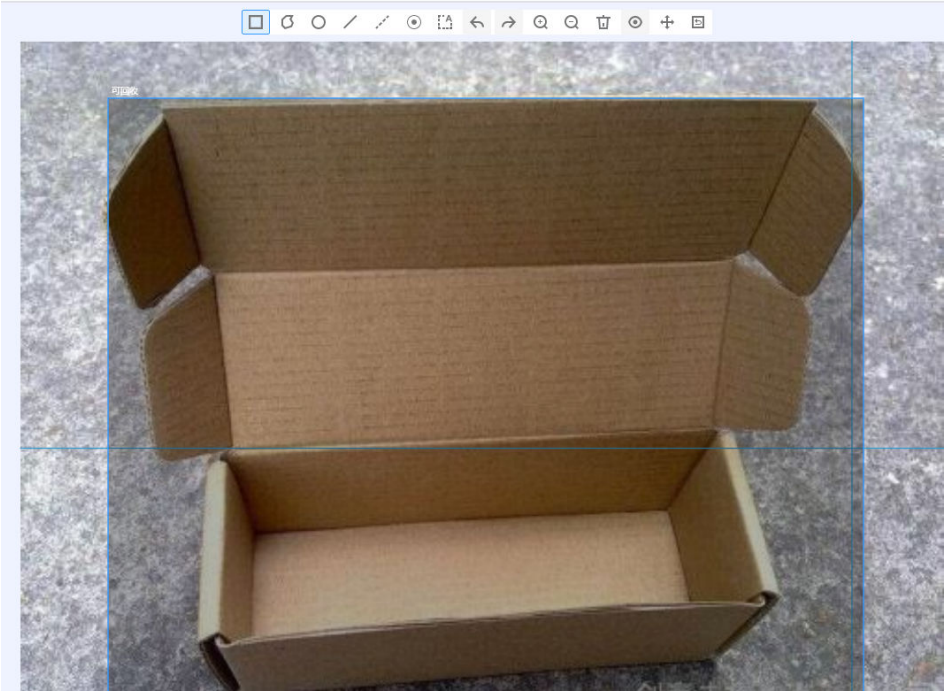
Labeling Images

1. On the labeling phase of the new-version ExeML, click **Instance Details**. The data labeling page is displayed. Click an image to go to the labeling page.
2. Left-click and drag the mouse to select the area where the target object is located. In the dialog box that is displayed, select the label color, enter the label name, for example, **yunbao**, and press **Enter**. After the labeling is complete, the status of the images changes to **Labeled**.

   More descriptions of data labeling are as follows:

   - You can click the arrow keys in the upper and lower parts of the image, or press the left and right arrow keys on the keyboard to select another image. Then, repeat the preceding operations to label the image. If an image contains more than one object, you can label all the objects.
   - You can add multiple labels with different colors for an object detection ExeML project for easy identification. After selecting an object, select a new color and enter a new label name in the dialog box that is displayed to add a new label.
   - In an ExeML project, object detection supports only rectangular labeling boxes. In the **Data Management** function, more types of labeling boxes are supported for object detection datasets.
   - In the **Label Data** window, you can scroll the mouse to zoom in or zoom out on the image to quickly locate the object.

Figure 4-14 Image labeling for object detection



NOTE

For an object detection dataset, you can add multiple labeling boxes and labels to an image during labeling. The labeling boxes cannot extend beyond the image boundary.

3. After all images in the image directory are labeled, return to the ExeML workflow page and click **Next**. The workflow automatically publishes a data labeling version and performs training.

Synchronizing or Adding Images

In the labeling phase, click **Instance Details** to go to the data labeling page. Then, add images from your local PC or synchronize images from OBS.

Figure 4-15 Adding local images

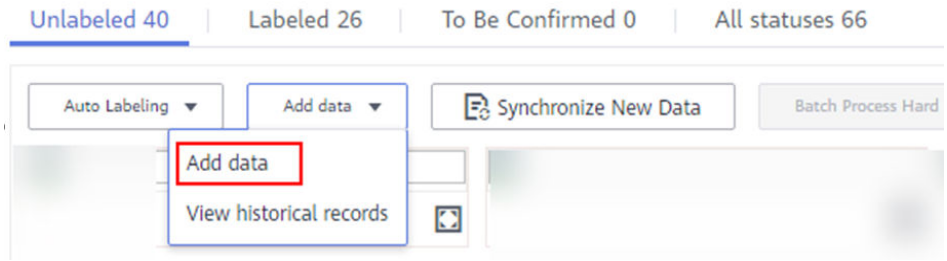
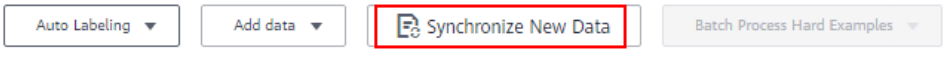


Figure 4-16 Synchronizing images from OBS



- **Add data**: You can quickly add images on a local PC to ModelArts. These images will be automatically synchronized to the OBS path specified during project creation. Click **Add data** to import data.
- **Synchronize New Data**: You can upload images to the OBS directory specified during project creation and click **Synchronize New Data** to quickly add the new images in the OBS directory to ModelArts.
- **Delete**: You can delete images one by one, or select **Select Images on Current Page** to delete all images on the page.

  NOTE

  Deleted images cannot be recovered.

Modifying Labeled Data

After labeling data, you can modify the labeled data on the **Labeled** tab page.

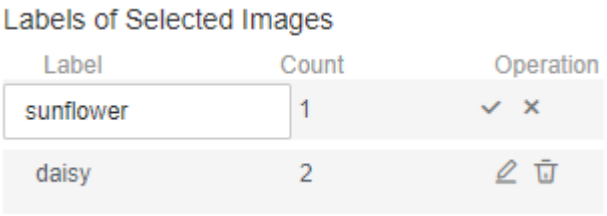
- **Modifying based on images**

  On the dataset details page, click the **Labeled** tab, and then select the image to be modified. Modify the image information in the label information area on the right.

  - Modifying a label: In the **Labeling** area, click the editing icon, enter the correct label name in the text box, and click the check mark to complete the modification. The label color cannot be modified.
  - Deleting a label: In the **Labeling** area, click the deletion button to delete a label for the image.

    After the label is deleted, click the project name in the upper left corner of the page to exit the labeling page. The image will be returned to the **Unlabeled** tab page.

    Figure 4-17 Editing an object detection label

- **Modifying based on labels**

  On the labeling job overview page, click **Label Management** on the right. You will see the label management page, which shows information about all labels.

  Figure 4-18 Label management page

  - Modifying a label: Click **Modify** in the **Operation** column. In the displayed dialog box, enter a new label and click **OK**. After the

modification, the images that have been added with the label use the new label name.

- Deleting a label: Click the delete button in the **Operation** column. In the displayed dialog box, confirm the operation and click **OK**.

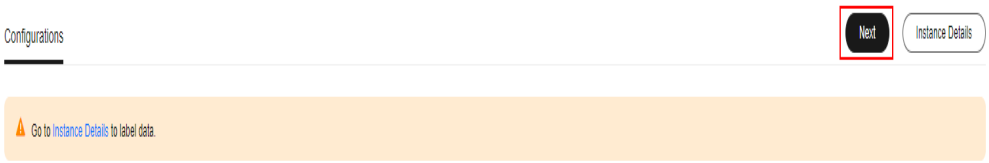
  NOTE

  Deleted tags cannot be restored.

Resuming Workflow Execution

After confirming data labeling, return back to the new-version ExeML. Click **Next**. Then, the workflow continues to run in sequence until all phases are executed.

Figure 4-19 Resuming the workflow execution



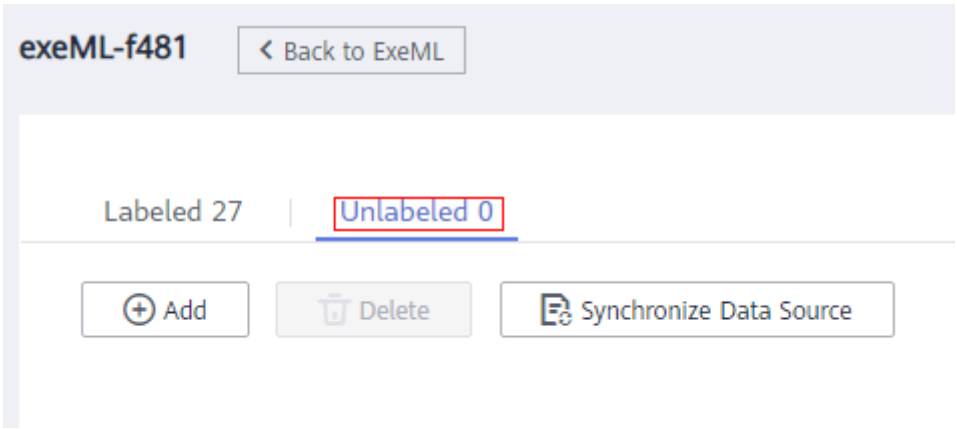
4.3.4 Training an Object Detection Model

After labeling the images, perform auto training to obtain an appropriate model version.

Procedure

1. On the ExeML page of the new version, click the name of the target project. Then, click **Instance Details** on the labeling phase to label data.

   Figure 4-20 Finding unlabeled data

2. Return to the labeling phase of the new-version ExeML, click **Next** and wait until the workflow enters the training phase.
3. Wait until the training is complete. No manual operation is required. If you close or exit the page, the system continues training until it is complete.
4. On the object detection phase, wait until the training status changes from **Running** to **Completed**.

5. After the training, click  on the object detection phase to view metric information. For details about the evaluation result parameters, see **Table 4-7**.

Table 4-7 Evaluation result parameters

Parameter	Description
Recall	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish positive samples.
Precision	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish negative samples.
Accuracy	Fraction of correctly predicted samples over all samples. It shows the general ability of a model to recognize samples.
F1 Score	Harmonic average of the precision and recall of a model. It is used to evaluate the quality of a model. A high F1 score indicates a good model.

NOTE

An ExeML project supports multiple rounds of training, and each round generates an AI application version. For example, the first training version is **0.0.1**, and the next version is **0.0.2**. The trained models can be managed by training version. After the trained model meets your requirements, deploy the model as a service.

4.3.5 Deploying an Object Detection Service

Deploying a Service

You can deploy a model as a real-time service that provides a real-time test UI and monitoring capabilities. After the model is trained, you can deploy a **Successful** version with ideal accuracy as a service. The procedure is as follows:

1. On the phase execution page, after the service deployment status changes to **Awaiting input**, double-click **Deploy Service**. On the configuration details page, configure resource parameters.
2. On the service deployment page, select the resource specifications used for service deployment.
   - **AI Application Source**: defaults to the generated AI application.
   - **AI Application and Version**: The current AI application version is automatically selected, which is changeable.
   - **Resource Pool**: defaults to public resource pools.
   - **Traffic Ratio**: defaults to **100** and supports a value range of 0 to 100.
   - **Specifications**: Select available specifications based on the list displayed on the console. The specifications in gray cannot be used in the current

environment. If there are no specifications after you select a public resource pool, no public resource pool is available in the current environment. In this case, use a dedicated resource pool or contact the administrator to create a public resource pool.

- **Compute Nodes**: an integer ranging from 1 to 5. The default value is **1**.
- **Auto Stop**: enables a service to automatically stop at a specified time. If this function is not enabled, the real-time service continuously runs and fees are incurred accordingly. Auto stop is enabled by default and its default value is **1 hour later**.

  The auto stop options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, enter any integer from 1 to 24 in the text box on the right.

  NOTE

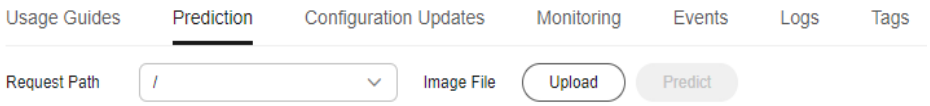
  You can choose the package that you have bought when you select specifications. On the configuration fee tag, you can view your remaining package quota and how much you will pay for any extra usage.

3. After configuring resources, click **Next**. Wait until the status changes to **Executed**. The AI application has been deployed as a real-time service.

Testing a Service

After the service is deployed, click **Instance Details** to go to the real-time service details page. Click the **Prediction** tab to test the service. You can also use code to test a service. For details, see **Accessing Real-Time Services**.

Figure 4-21 Testing the service



The following describes the procedure for performing a service test after the object detection model is deployed as a service on the ExeML page.

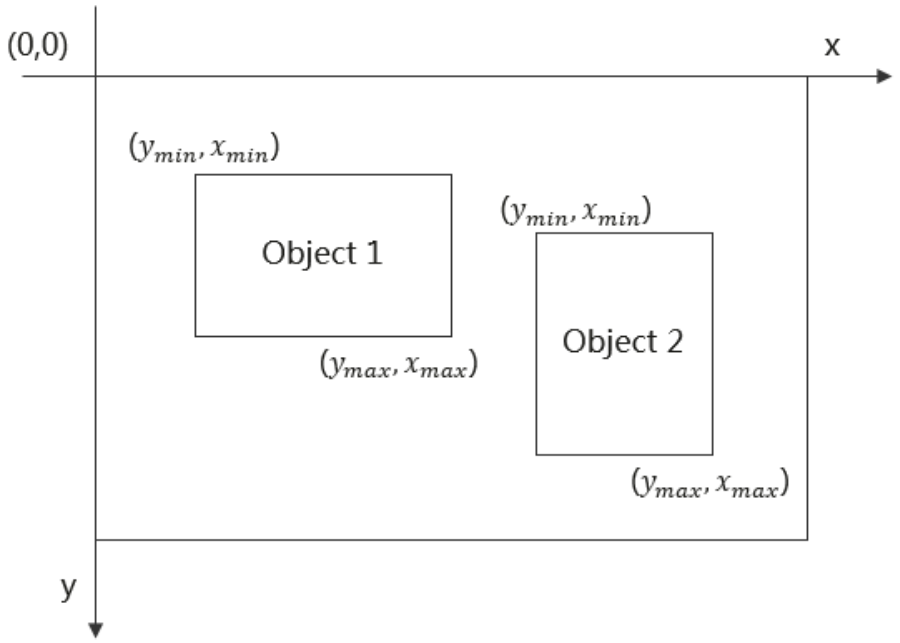
1. After the model is deployed, click **Instance Details** in the service deployment phase to go to the service page. On the **Prediction** tab page, click **Upload** and select a local image for test.
2. Click **Predict** to perform the test. After the prediction is complete, the result is displayed in the **Test Result** pane on the right. If the model accuracy does not meet your expectation, add images on the **Label Data** tab page, label the images, and train and deploy the model again. **Table 4-8** describes the parameters in the prediction result. If you are satisfied with the model prediction result, call the API to access the real-time service as prompted. For details, see **Accessing Real-Time Services**.

   Currently, only JPG, JPEG, BMP, and PNG images are supported.

Table 4-8 Parameters in the prediction result

Parameter	Description
detection_classes	Label of each detection box
detection_boxes	Coordinates of four points (y_min, x_min, y_max, and x_max) of each detection box, as shown in **Figure 4-22**
detection_scores	Confidence of each detection box

Figure 4-22 Illustration for coordinates of four points of a detection box



NOTE

- A running real-time service keeps consuming resources. If you do not need to use the real-time service, click **Stop** in the **Version Manager** pane to stop the service so that charges will no longer be incurred. If you want to use the service again, click **Start**.
- If you enable the auto stop function, the service automatically stops after the specified time and no fee is generated.

4.4 Using ExeML for Predictive Analytics

4.4.1 Preparing Predictive Analysis Data

Before using ModelArts to build a predictive analytics model, upload data to OBS. The OBS bucket and ModelArts must be in the same region. For example, if the OBS bucket is in the CN North-Beijing4 region, ensure that the ModelArts

management console is also in the CN North-Beijing4 region. Otherwise, data cannot be obtained.

Requirements on Datasets

The data set used in the predictive analytics project must be a table dataset in .csv format. For details about the table dataset, see **Creating a Dataset**.

NOTE

To convert the data from .xlsx to .csv, perform the following operations:

Save the original table data in .xlsx. Choose **File** > **Save As**, select a local address, set **Save as type:** to **CSV (Comma delimited)**, and click **Save**. Then, click **OK** in the displayed dialog box.

Requirements on the training data:

- The number of columns in the training data must be the same, and there has to be at least 100 data records (a feature with different values is considered as different data records).
- The training columns cannot contain timestamp data (such as yy-mm-dd or yyyy-mm-dd).
- If a column has only one value, the column is considered invalid. Ensure that there are at least two values in the label column and no data is missing.

  NOTE

  The label column is the training target specified in a training task. It is the output (prediction item) for the model trained using the dataset.
- In addition to the label column, the dataset must contain at least two valid feature columns. Ensure that there are at least two values in each feature column and that the percentage of missing data must be lower than 10%.
- Due to the limitation of the feature filtering algorithm, place the predictive data column at the last. Otherwise, the training may fail.

Example of a table dataset:

The following table takes the bank deposit predictive dataset as an example. Data sources include age, occupation, marital status, cultural level, and whether there is a personal mortgage or personal loan.

Table 4-9 Fields and meanings of data sources

Field	Meaning	Type	Description
attr_1	Age	Int	Age of the customer
attr_2	Occupation	String	Occupation of the customer
attr_3	Marital status	String	Marital status of the customer
attr_4	Education status	String	Education status of the customer
attr_5	Real estate	String	Housing situation of the customer

Field	Meaning	Type	Description
attr_6	Loan	String	Loan of the customer
attr_7	Deposit	String	Deposit of the customer

Table 4-10 Sample data of the dataset

attr_1	attr_2	attr_3	attr_4	attr_5	attr_6	attr_7
31	blue-collar	married	secondary	yes	no	no
41	management	married	tertiary	yes	yes	no
38	technician	single	secondary	yes	no	no
39	technician	single	secondary	yes	no	yes
39	blue-collar	married	secondary	yes	no	no
39	services	single	unknown	yes	no	no

Uploading Data to OBS

In this section, the OBS console is used to upload data.

Upload files to OBS according to the following specifications:

The OBS path of the predictive analytics projects must comply with the following rules:

- The OBS path of the input data must redirect to the data files. The data files must be stored in a folder in an OBS bucket rather than the root directory of the OBS bucket, for example, **/obs-xxx/data/input.csv**.
- There must be at least 100 lines of valid data in .csv. There cannot be more than 200 columns of data and the total data size must be smaller than 100 MB.

Procedure for uploading data to OBS:

Perform the following operations to import data to the dataset for model training and building.

1. Log in to the OBS console and **create a bucket** in the same region as ModelArts. If an available bucket exists, ensure that the OBS bucket and ModelArts are in the same region.

2. **Upload the local data** to the OBS bucket. If you have a large amount of data, use OBS Browser+ to upload data or folders. The uploaded data must meet the dataset requirements of the ExeML project.

NOTE

Upload data from unencrypted buckets. Otherwise, training will fail because data cannot be decrypted.

Creating a Dataset

After the data is prepared, create a proper dataset. For details, see **Creating a Dataset**.

FAQs

How do I process Schema information when creating a table dataset using data selected from OBS?

Schema information includes the names and types of table columns, which must be the same as those of the imported data.

- If the original table contains a table header, enable **Contain Table Header**. The first row of the file will be used as column names. You do not need to modify the Schema information.
- If the original table does not contain a table header, disable **Contain Table Header**. After data is selected from OBS, the column names will be used as the first row of the table by default. Change the column names to **attr_1**, **attr_2**, ..., **attr_n**. **attr_n** is the prediction column placed at last.

4.4.2 Creating a Predictive Analytics Project

ModelArts ExeML supports sound classification, text classification, image classification, predictive analytics, and object detection projects. You can create any of them based on your needs. Perform the following operations to create an ExeML project.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. Click **Create Project** in the box of your desired project.
3. On the displayed page, set the parameters by referring to **Table 4-11**. The default billing mode is **Pay-per-use**.

Table 4-11 Parameters

Parameter	Description
Name	Name of a project • Enter a maximum of 64 characters. Only digits, letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory. • Start with a letter. • The name must be unique.
Description	Brief description of a project
Datasets	You can select a dataset or click **Create Dataset** to create one. • Existing dataset: Select a dataset from the drop-down list box. Only datasets of the same type are displayed. • Creating a dataset: Click **Create Dataset** to create a dataset. For details, see **Creating a Dataset**.
Label Column	Select the column you want to predict. The label column is the output of an inference model. During model training, all information is used to train an inference model. The model uses the data of other columns as the input and outputs the inference result in the label column. You can publish the model as a real-time inference service.
Output Path	Select an OBS path for storing ExeML data. **NOTE** The output path stores all data generated in the ExeML project.
Training Flavor	Select a training flavor for this ExeML project. You will be billed based on different flavors. **NOTE** • You can choose the package that you have bought. In the configuration fee area, you will see your remaining package quota and how much you will pay for any extra usage.

4. Click **Create Project**. Then, the ExeML workflow is displayed.
5. Wait until the workflow of the predictive analytics project executes the following phases in sequence:
   a. **Publish Dataset Version**: Publish a version for the labeled dataset.
   b. **Check Data**: Check whether any exception occurs in your dataset.
   c. **Predict**: Train the dataset of the published version to generate a model.
   d. **Register Model**: Register the trained model with model management.
   e. **Deploy Service**: Deploy the generated model as a real-time service.

Quickly Searching for a Project

On the ExeML overview page, you can use the search box to quickly search for and filter workflows based on the ExeML type (or project name).

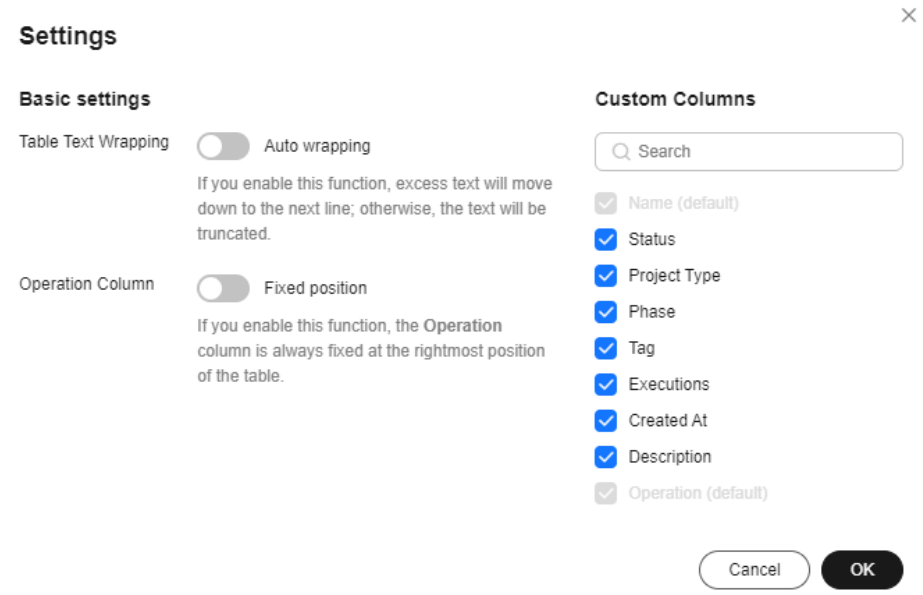
1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. In the search box above the ExeML project list, filter the desired workflows based on the required property, such as name, status, project type, current phase, and tag.
3. To adjust the basic settings of ExeML and select the columns you want to see, click  on the right of the search box.

   Table Text Wrapping: This function is disabled by default. If you enable this function, excess text will move down to the next line; otherwise, the text will be truncated.

   Operation Column: This function is disabled by default. If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.

   Custom Columns: By default, all items are selected. You can select columns you want to see.

   Figure 4-23 Customizing table columns

4. Click **OK**. Then, the columns will be displayed based on the settings.
5. To arrange ExeML projects by a specific property, click  in the table header.

4.4.3 Training a Predictive Analysis Model

After the ExeML task is created, a model is trained for predictive analytics. You can publish the model as a real-time inference service.

Procedure

1. On the ExeML page of the new version, click the name of the target project to view the execution status of the current workflow.
2. On the predictive analytics phase, wait until the phase status changes from **Running** to **Executed**.
3. Click  to view the training details, such as the label column, data type, accuracy, and evaluation result.

   The example is a discrete value of binary classification. For details about the evaluation result parameters, see **Table 4-12**.

   For details about the evaluation results generated for different data types of label columns, see **Evaluation Results**.

NOTE

An ExeML project supports multiple rounds of training, and each round generates an AI application version. For example, the first training version is **0.0.1**, and the next version is **0.0.2**. The trained models can be managed by training version. After the trained model meets your requirements, deploy the model as a service.

Evaluation Results

The parameters in evaluation results vary depending on the training data type.

- Discrete values

  The evaluation parameters include recall, precision, accuracy, and F1 score, which are described in the following table.

  Table 4-12 Parameters in discrete value evaluation results

Parameter	Description
Recall	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish positive samples.
Precision	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish negative samples.
Accuracy	Fraction of correctly predicted samples over all samples. It shows the general ability of a model to recognize samples.
F1 Score	Harmonic average of the precision and recall of a model. It is used to evaluate the quality of a model. A high F1 score indicates a good model.

- Continuous values

  The evaluation parameters include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). The three error values represent a difference between a real value and a predicted value.

During multiple rounds of modeling, a group of error values is generated for each round of modeling. Use these error values to determine the quality of a model. A smaller error value indicates a better model.

4.4.4 Deploying a Predictive Analytics Service

Deploying a Service

You can deploy a model as a real-time service that provides a real-time test UI and monitoring capabilities. After the model is trained, you can deploy a **Successful** version with ideal accuracy as a service. The procedure is as follows:

1. On the phase execution page, after the service deployment status changes to **Awaiting input**, double-click **Deploy Service**. On the configuration details page, configure resource parameters.
2. On the service deployment page, select the resource specifications used for service deployment.
   - **AI Application Source**: defaults to the generated AI application.
   - **AI Application and Version**: The current AI application version is automatically selected, which is changeable.
   - **Resource Pool**: defaults to public resource pools.
   - **Traffic Ratio**: defaults to **100** and supports a value range of 0 to 100.
   - **Specifications**: Select available specifications based on the list displayed on the console. The specifications in gray cannot be used in the current environment. If there are no specifications after you select a public resource pool, no public resource pool is available in the current environment. In this case, use a dedicated resource pool or contact the administrator to create a public resource pool.
   - **Compute Nodes**: an integer ranging from 1 to 5. The default value is **1**.
   - **Auto Stop**: enables a service to automatically stop at a specified time. If this function is disabled, a real-time service will continue to run and charges will continue to be incurred. The auto stop function is enabled by default. The default value is **1 hour later**.

     The options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, enter any integer from 1 to 24 in the text box on the right.

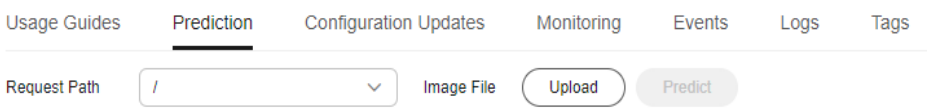
     NOTE

     You can choose the package that you have bought when you select specifications. On the configuration fee tag, you can view your remaining package quota and how much you will pay for any extra usage.
3. After configuring resources, click **Next** and confirm the operation. Wait until the status changes to **Executed**, which means the AI application has been deployed as a real-time service.

Testing the Service

After the service is deployed, click **Instance Details** to go to the real-time service details page. Click the **Prediction** tab to test the service. You can also use code to test a service. For details, see **Accessing Real-Time Services**.

Figure 4-24 Testing the service



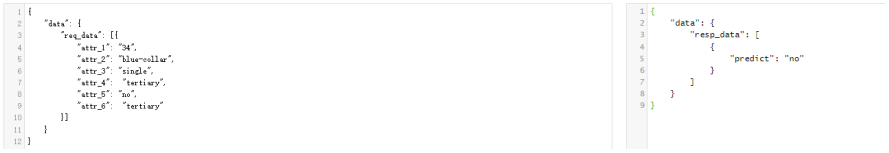
The following describes the procedure for performing a service test after the predictive analytics model is deployed as a service on the ExeML page.

1. After the model is deployed, you can test the model using code. In ExeML, click **Instance Details** on the **Deploy Service** page to go to the real-time service page. On the **Prediction** tab page, enter the debugging code in the **Inference Code** area.
2. Click **Predict** to perform the test. After the prediction is complete, the result is displayed in the **Test Result** pane on the right. If the model accuracy does not meet your expectation, train and deploy the model again on the **Label Data** tab page. If you are satisfied with the model prediction result, call the API to access the real-time service as prompted. For details, see **Accessing Real-Time Services**.
   - In the input code, the label column of a predictive analytics database must be named **class**. Otherwise, the prediction will fail.

```
{
   "data": {
      "req_data": [{
         "attr_1": "34",
         "attr_2": "blue-collar",
         "attr_3": "single",
         "attr_4":  "tertiary",
         "attr_5": "no",
         "attr_6":  "tertiary"
      }]
   }
}
```

   - In the preceding code snippet, **predict** is the inference result of the label column.

Figure 4-25 Prediction result



NOTE

- A running real-time service continuously consumes resources. If you do not need to use the real-time service, stop the service to stop billing. To do so, click **Stop** in the **More** drop-down list in the **Operation** column. If you want to use the service again, click **Start**.
- If you enable auto stop, the service automatically stops at the specified time and no fees will be generated then.

4.5 Using ExeML for Sound Classification

4.5.1 Preparing Sound Classification Data

Before using ModelArts ExeML to build a model, upload data to an OBS bucket. The OBS bucket and ModelArts must be in the same region.

Requirements for Sound Classification Data

- Only 16-bit WAV files are supported. All sub-formats of WAV are supported.
- The audio must be longer than 1 second and the file must be no larger than 4 MB.
- Add more sound files to improve model precision. Prepare at least 20 sound files for each class. Ensure that the total duration of each class is no shorter than 5 minutes.
- Ensure that the sound files are authentic and cover all scenarios in real life.
- The quality of the training set has a great impact on the precision of the model. Set the sampling rate to the precision of the training set.
- The labeling quality has a great impact on the model precision. Do not mislabel objects.
- Only Chinese and English are supported for audio labeling.

Uploading Data to OBS

In this section, the OBS console is used to upload data.

Upload files to OBS according to the following specifications:

- If you do not need to upload training data in advance, create an empty folder to store files generated in the future, for example, **/bucketName/data-cat**.
- If you need to upload training data in advance, create an empty folder, and save the sound files to be labeled in the folder, for example, **/bucketName/data-cat/cat.wav**.

Procedure for uploading data to OBS:

Perform the following operations to import data to the dataset for model training and building.

1. Log in to the OBS console and **create a bucket** in the same region as ModelArts. If an available bucket exists, ensure that the OBS bucket and ModelArts are in the same region.
2. **Upload the local data** to the OBS bucket. If you have a large amount of data, use OBS Browser+ to upload data or folders. The uploaded data must meet the dataset requirements of the ExeML project.

NOTE

- Upload data from unencrypted buckets. Otherwise, training will fail because data cannot be decrypted.
- Training sound files must be classified into at least two classes, and each class must contain at least 20 sound files.

Creating a Dataset

After the data preparation is completed, create a dataset of the type supported by the project. For details, see **Creating a Dataset**.

4.5.2 Creating a Sound Classification Project

ModelArts ExeML supports sound classification, text classification, image classification, predictive analytics, and object detection projects. You can create any of them based on your needs. Perform the following operations to create an ExeML project.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. Click **Create Project** in the box of your desired project. The page for creating an ExeML project is displayed.
3. On the displayed page, configure parameters by referring to **Table 4-13**. The default billing mode is **Pay-per-use**.

Table 4-13 Parameters

Parameter	Description
Name	Name of a project • Enter a maximum of 64 characters. Only digits, letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory. • Start with a letter. • The name must be unique.
Description	Brief description of a project
Dataset Source	You can select a dataset or click **Create Dataset** to create one. • Existing dataset: Select a dataset from the drop-down list box. Only datasets of the same type are displayed. • Creating a dataset: Click **Create Dataset** to create a dataset. For details, see **Creating a Dataset**.
Output Path	An OBS path for storing ExeML data **NOTE** The output path stores all data generated in the ExeML project.
Training Flavor	Select a training flavor for this ExeML project. You will be billed based on different flavors. **NOTE** • You can choose the package that you have bought. In the configuration fee area, you will see your remaining package quota and how much you will pay for any extra usage.

4. Click **Create Project**. Then, the ExeML workflow is displayed.
5. Wait until the workflow of the sound classification project executes the following phases in sequence:
   a. **Label Data**: Check data labeling.
   b. **Publish Dataset Version**: Publish a version for the labeled dataset.
   c. **Check Data**: Check whether any exception occurs in your dataset.
   d. **Classify Sounds**: Train the dataset of the published version to generate a model.
   e. **Register Model**: Register the trained model with model management.
   f. **Deploy Service**: Deploy the generated model as a real-time service.

Quickly Searching for a Project

On the ExeML overview page, you can use the search box to quickly search for and filter workflows based on the ExeML type (or project name).

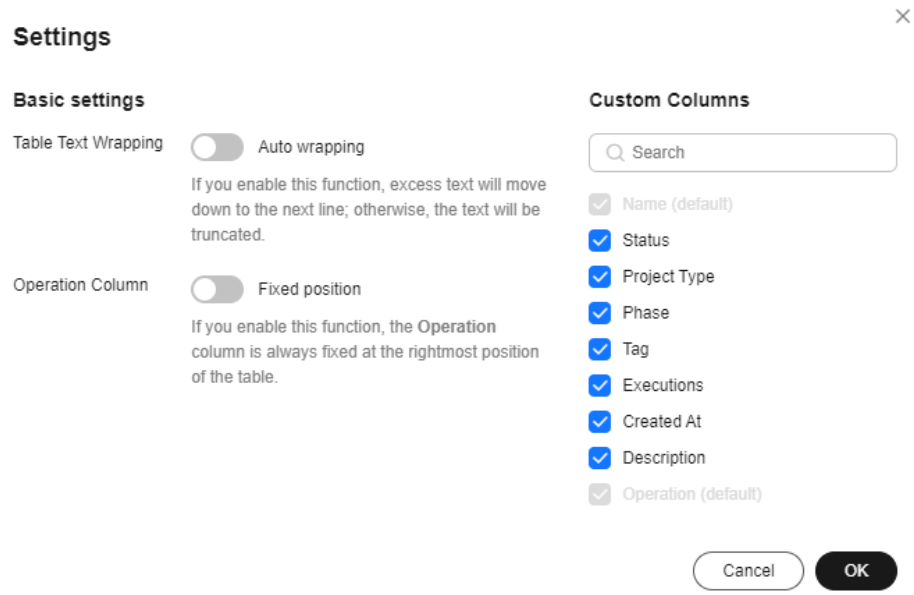
1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. In the search box above the ExeML project list, filter the desired workflows based on the required property, such as name, status, project type, current phase, and tag.
3. To adjust the basic settings of ExeML and select the columns you want to see, click  on the right of the search box.

   Table Text Wrapping: This function is disabled by default. If you enable this function, excess text will move down to the next line; otherwise, the text will be truncated.

   Operation Column: This function is disabled by default. If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.

   Custom Columns: By default, all items are selected. You can select columns you want to see.

Figure 4-26 Customizing table columns

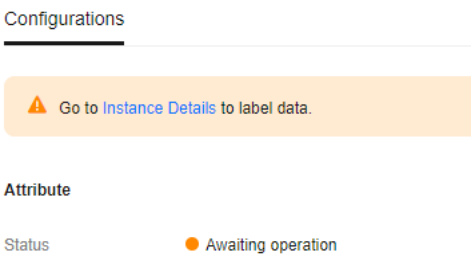


4. Click **OK**. Then, the columns will be displayed based on the settings.
5. To arrange ExeML projects by a specific property, click in the table header.

4.5.3 Labeling Sound Classification Data

After a project is created, you will be redirected to the new-version ExeML and the project starts to run. When the data labeling phase changes to **Awaiting operation**, manually confirm data labeling in the dataset. You can also add or delete data in the dataset and modify labels.

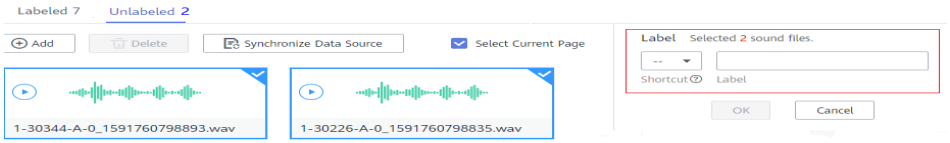
Figure 4-27 Data labeling status



Labeling Sound Files

1. On the labeling phase of the new-version ExeML, click **Instance Details**. The data labeling page is displayed. Click an image to go to the labeling page.
2. On the labeling page, click the **Unlabeled** tab. All unlabeled sound files are displayed. Select the sound files to be labeled in sequence, or tick **Select Current Page** to select all sound files on the page, and then add labels to the sound files in the right pane.

Figure 4-28 Labeling a sound file



3. Add a label. Play a sound file, and select the sound file. In the **Label** area, enter a label name or select an existing label from the drop-down list on the right, and select a shortcut from the drop-down list on the left. Click **OK**. The selected sound file is labeled.
4. After all the sound files are labeled, view them on the **Labeled** tab page or view the list of **All Labels** in the right pane to learn the name and quantity of the labels.

Synchronizing or Adding Sound Files

On the data labeling phase, click **Instance Details**. The labeling page is displayed. When creating a sound classification project, you can select local data or synchronize data in OBS as the training data.

- **Add Audio**: You can quickly add sound files on a local PC to ModelArts and synchronize the files to the OBS path specified during project creation. Click **Add data** to import data.

  NOTE

  Only 16-bit WAV files are supported. The size of a sound file cannot exceed 4 MB. The total size of all sound files uploaded in one attempt cannot exceed 8 MB.

- **Synchronize Data Source**: To quickly obtain the latest sound files in the OBS bucket, click **Synchronize Data Source** to add sound files in OBS to ModelArts.
- **Delete Audio**: You can delete sound files one by one, or tick **Select Current Page** to delete all sound files on the page.

  NOTE

  The deleted sound files cannot be recovered. Exercise caution when performing this operation.

Modifying Labeled Data

After labeling data, you can modify the labeled data on the **Labeled** tab page.

- **Modifying based on audio**

  On the dataset details page, click the **Labeled** tab. Select one or more audio files to be modified from the audio list. Modify the label in the label details area on the right.

  - Modifying a label: In the **File Labels** area, click the editing icon in the **Operation** column, enter the correct label name in the text box, and click the check mark icon.
  - Deleting a label: In the **File Labels** area, click the deletion icon in the **Operation** column. In the displayed dialog box, click **OK**.

- **Modifying based on labels**

On the data labeling page, click **Label Management** on the right. You will see information about all labels.

- Modifying a label: Click the edit button in the **Operation** column. In the displayed dialog box, enter the new label name, select the new shortcut, and click **OK**. After the modification, the new label applies to the audio files that contain the original label.
- Deleting a label: Click the delete button in the **Operation** column. In the displayed dialog box, confirm the operation and click **OK**.

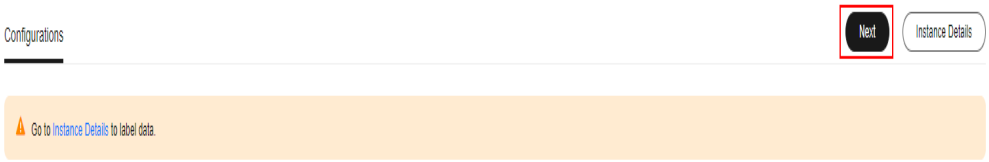
  NOTE

  Deleted tags cannot be restored.

Resuming Workflow Execution

After confirming data labeling, return back to the new-version ExeML. Click **Next**. Then, the workflow continues to run in sequence until all phases are executed.

Figure 4-29 Resuming the workflow execution



4.5.4 Training a Sound Classification Model

After labeling the sound files, train a model. You can perform model training to obtain the required sound classification model. Training sound files must be classified into at least two classes, and each class must contain at least five sound files.

Procedure

Before starting the training, label data and then perform auto training.

1. On the ExeML page of the new version, click the name of the target project. Then, click **Instance Details** on the labeling phase to label data.
2. Return to the labeling phase of the new-version ExeML, click **Next** and wait until the workflow enters the training phase.
3. Wait until the training is complete. No manual operation is required. If you close or exit the page, the system continues training until it is complete.
4. On the sound classification phase, wait until the training status changes from **Running** to **Executed**.
5. After the training, click  on the sound classification phase to view metric information.

Table 4-14 Evaluation result parameters

Parameter	Description
Recall	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish positive samples.
Precision	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish negative samples.
Accuracy	Fraction of correctly predicted samples over all samples. It shows the general ability of a model to recognize samples.
F1 Score	Harmonic average of the precision and recall of a model. It is used to evaluate the quality of a model. A high F1 score indicates a good model.

NOTE

An ExeML project supports multiple rounds of training, and each round generates an AI application version. For example, the first training version is **0.0.1**, and the next version is **0.0.2**. The trained models can be managed by training version. After the trained model meets your requirements, deploy the model as a service.

4.5.5 Deploying a Sound Classification Service

Deploying a Service

You can deploy a model as a real-time service that provides a real-time test UI and monitoring capabilities. After the model is trained, you can deploy a **Successful** version with ideal accuracy as a service. The procedure is as follows:

1. On the **Dashboard** page, after the service deployment status changes to **Awaiting input**, double-click **Deploy Service**. On the configuration details page, configure resource parameters.
2. On the service deployment page, select the resource specifications used for service deployment.
   - **AI Application Source**: defaults to the generated AI application.
   - **AI Application and Version**: The current AI application version is automatically selected, which is changeable.
   - **Resource Pool**: defaults to public resource pools.
   - **Traffic Ratio**: defaults to **100** and supports a value range of 0 to 100.
   - **Specifications**: Select available specifications based on the list displayed on the console. The specifications in gray cannot be used in the current environment. If there are no specifications after you select a public resource pool, no public resource pool is available in the current environment. In this case, use a dedicated resource pool or contact the administrator to create a public resource pool.
   - **Compute Nodes**: an integer ranging from 1 to 5. The default value is **1**.

- **Auto Stop**: enables a service to automatically stop at a specified time. If this function is not enabled, the real-time service continuously runs and fees are incurred accordingly. Auto stop is enabled by default and its default value is **1 hour later**.

  The auto stop options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, enter any integer from 1 to 24 in the text box on the right.

  NOTE

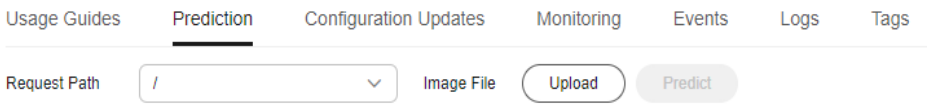
  You can choose the package that you have bought when you select specifications. On the configuration fee tag, you can view your remaining package quota and how much you will pay for any extra usage.

3. After configuring resources, click **Next** and confirm the operation. Wait until the status changes to **Executed**, which means the AI application has been deployed as a real-time service.

Testing a Service

After the service is deployed, click **Instance Details** to go to the real-time service details page. Click the **Prediction** tab to test the service. You can also use code to test a service. For details, see **Accessing Real-Time Services**.

Figure 4-30 Testing the service



The following describes the procedure for performing a service test after the sound classification model is deployed as a service on the ExeML page.

1. After the model is deployed, you can add a sound file for test. On the **ExeML** page, go to the service deployment phase, click **Instance Details** to go to the **Deploy Service** tab page, select the service version in the **Running** status, click **Upload** in the service test area, and upload a local sound file to perform the test.
2. Click **Predict** to perform the test. After the prediction is complete, the result is displayed in the **Test Result** pane on the right. If the model accuracy does not meet your expectation, add sound files on the **Label Data** tab page, label the files, and train and deploy the model again. **Table 4-15** describes the parameters in the prediction result. If you are satisfied with the model prediction result, call the API to access the real-time service as prompted. For details, see **Accessing Real-Time Services**.

Table 4-15 Parameters in the prediction result

Parameter	Description
predicted_label	Prediction type of the audio segment
score	Confidence score for the predicated class

NOTE

A running real-time service keeps consuming resources. If you do not need to use the real-time service, click **Stop** in the **Version Manager** pane to stop the service so that charges will no longer be incurred. If you want to use the service again, click **Start**.

If you enable the auto stop function, the service automatically stops after the specified time and no fee is generated.

4.6 Using ExeML for Text Classification

4.6.1 Preparing Text Classification Data

Before using ModelArts ExeML to build a model, upload data to an OBS bucket. The OBS bucket and ModelArts must be in the same region.

Requirements on Datasets

- Files must be in TXT or CSV format, and cannot exceed 8 MB.
- Use line feed characters to separate rows in files, and each row of data represents a labeled object.
- Currently, only Chinese is supported.

Uploading Data to OBS

In this section, the OBS console is used to upload data.

Requirements for files uploaded to OBS:

- If you do not need to upload training data in advance, create an empty folder to store files generated in the future.
- If you need to upload files to be labeled in advance, create an empty folder, and save the files in the folder. An example of the file directory structure is **/bucketName/data/text.csv**.
- A label name can contain a maximum of 32 characters, including Chinese characters, letters, digits, hyphens (-), and underscores (_).
- If you want to upload labeled text files to an OBS bucket, upload them according to the following specifications:
  - The objects and files to be labels must be in the same directory. The objects must be in one-to-one relationship with the files. For example, if the object file name is **COMMENTS_114745.txt**, the label file name must be **COMMENTS_114745_result.txt**.

    The following shows an example of data file.

    ```
    ├─<dataset-import-path>
    │       COMMENTS_114732.txt
    │       COMMENTS_114732_result.txt
    │       COMMENTS_114745.txt
    │       COMMENTS_114745_result.txt
    │       COMMENTS_114945.txt
    │       COMMENTS_114945_result.txt
    ```

  - The labeling objects and files are text files and correspond to each other by line.

Procedures for uploading data from OBS:

Perform the following operations to import data to the dataset for model training and building.

1. Log in to OBS Console and **create a bucket** in the same region as ModelArts. If an available bucket exists, ensure that the OBS bucket and ModelArts are in the same region.
2. **Upload the local data** to the OBS bucket. If you have a large amount of data, use OBS Browser+ to upload data or folders. The uploaded data must meet the dataset requirements of the ExeML project.

NOTE

- Upload data from unencrypted buckets. Otherwise, training will fail because data cannot be decrypted.
- Training text files must be classified into at least two classes, and each class must contain at least 20 rows.

Creating a Dataset

After the data preparation is completed, create a dataset of the type supported by the project. For details, see **Creating a Dataset**.

4.6.2 Creating a Text Classification Project

ModelArts ExeML supports sound classification, text classification, image classification, predictive analytics, and object detection projects. You can create any of them based on your needs. Perform the following operations to create an ExeML project.

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. Click **Create Project** in the box of your desired project. The page for creating an ExeML project is displayed.
3. On the displayed page, configure parameters by referring to **Table 4-16**. The default billing mode is **Pay-per-use**.

Table 4-16 Parameters

Parameter	Description
Name	Name of a project • Enter a maximum of 64 characters. Only digits, letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory. • Start with a letter. • The name must be unique.
Description	Brief description of a project

Parameter	Description
Datasets	You can select a dataset or click **Create Dataset** to create one. • Existing dataset: Select a dataset from the drop-down list box. Only datasets of the same type are displayed. • Creating a dataset: Click **Create Dataset** to create a dataset. For details, see Creating a Dataset.
Output Path	Select an OBS path for storing ExeML data. **NOTE** The output path stores all data generated in the ExeML project.
Training Flavor	Select a training flavor for this ExeML project. You will be billed based on different flavors. **NOTE** • You can choose the package that you have bought. In the configuration fee area, you will see your remaining package quota and how much you will pay for any extra usage.

4. Click **Create Project**. Then, the ExeML workflow is displayed.
5. Wait until the workflow of the text classification project executes the following phases in sequence:
   a. **Label Data**: Check data labeling.
   b. **Publish Dataset Version**: Publish a version for the labeled dataset.
   c. **Check Data**: Check whether any exception occurs in your dataset.
   d. **Classify Text**: Train the dataset of the published version to generate a model.
   e. **Register Model**: Register the trained model with model management.
   f. **Deploy Service**: Deploy the generated model as a real-time service.

Quickly Searching for a Project

On the ExeML overview page, you can use the search box to quickly search for and filter workflows based on the ExeML type (or project name).

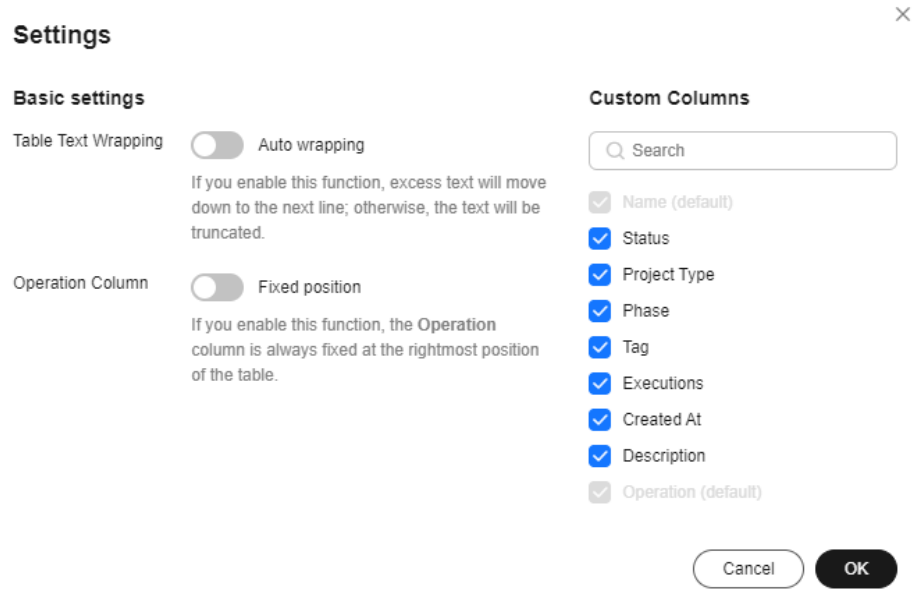
1. Log in to the ModelArts console. In the navigation pane, choose **ExeML**.
2. In the search box above the ExeML project list, filter the desired workflows based on the required property, such as name, status, project type, current phase, and tag.
3. To adjust the basic settings of ExeML and select the columns you want to see, click  on the right of the search box.

   Table Text Wrapping: This function is disabled by default. If you enable this function, excess text will move down to the next line; otherwise, the text will be truncated.

   Operation Column: This function is disabled by default. If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.

Custom Columns: By default, all items are selected. You can select columns you want to see.

Figure 4-31 Customizing table columns

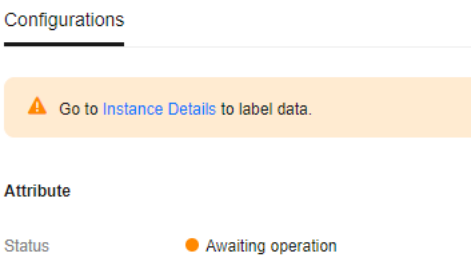


4. Click **OK**. Then, the columns will be displayed based on the settings.
5. To arrange ExeML projects by a specific property, click  in the table header.

4.6.3 Labeling Text Classification Data

After a project is created, you will be redirected to the new-version ExeML and the project starts to run. When the data labeling phase changes to **Awaiting operation**, manually confirm data labeling in the dataset. You can also add or delete data in the dataset and modify labels.

Figure 4-32 Data labeling status



Double-click **Label Data** and click **Instance Details**. The data labeling page is displayed.

Data Labeling for Text Classification

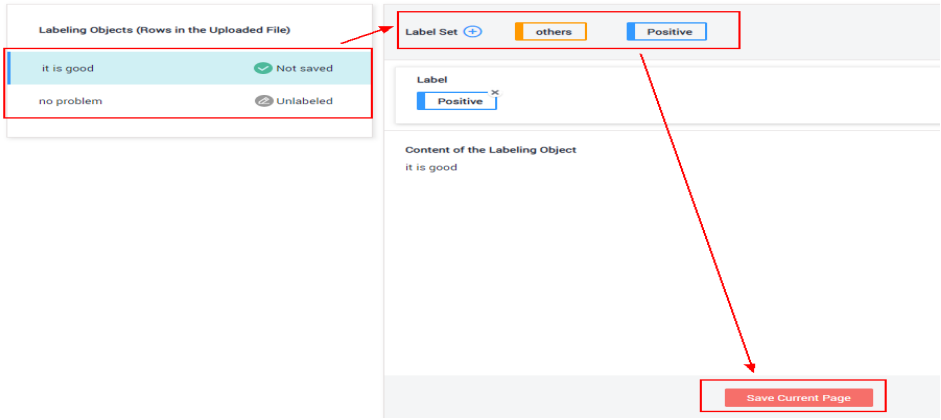
1. Select a text to be labeled in **Labeling Objects** and click different labels in the **Label Set** area to label the text.

You can add only one label for a text object.

2. After confirming the file label, click **Save Current Page** in the lower right corner to save the labeling.

   If a large number of objects are included in **Labeling Objects**, the page turning icon is displayed in the lower part of the area. After labeling objects on this page, click **Save Current Page** before you turn to the next page. If you turn pages before saving the labellings, the labeling information on the previous page will be lost. You need to re-label for text data.

Figure 4-33 Data labeling - text classification



Adding or Deleting Data

In an ExeML project, the data source is the OBS directory corresponding to the input path of the dataset. If the data in the directory cannot meet your requirements, add or delete data on the ExeML page of ModelArts.

- **Adding a file**

  On the **Unlabeled** tab, click **Add data** in the top left corner. In the displayed dialog box, select a local file and upload it.

  The format of the file to be uploaded must meet **requirement on datasets** of the text classification type.

- **Deleting a text object**

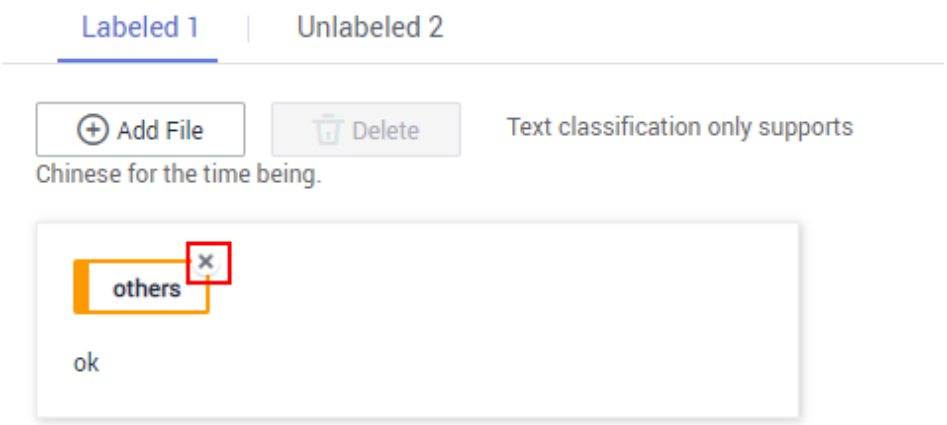
  On the **Labeled** or **Unlabeled** tab page, select a text object to be deleted and click **Delete** in the upper left corner. In the dialog box that is displayed, confirm the deletion information and click **OK**.

  On the **Labeled** tab page, you can tick **Select Current Page** and click **Delete** to delete all text objects and their labeling information on the current page.

Modifying Labeled Data

For labeled text data, only labels of the text object can be deleted. To delete a label, go to the **Labeled** tab, locate the label name area, and click the cross icon next to the label. After the label is deleted, the text object is displayed on the **Unlabeled** tab page.

Figure 4-34 Deleting a labeled text



Modifying a Label

After an ExeML project for text classification is created, you can modify labels based on service changes, including label adding, modification, and deletion.

- Adding a label

  On the **Unlabeled** tab, click the plus sign (+) on the right of **Label Set**. In the **Add Label** dialog box that appears, set **Label Name** and **Label Color**, and click **OK**.

- Modifying a label

  On the **Labeled** tab, locate the **All Labels** area, and click the edit button in the **Operation** column of the label you want to change. In the **Modify** Label dialog box, set **Label Name** and **Label Color** and click **OK**.

- Deleting a label

  In the lower part of **All labels** on the **Labeled** tab page, select a label to be deleted and click the deletion icon in the **Operation** column. In the displayed **Delete** dialog box, select **Delete label** or **Delete the label and objects with only the label**, and click **OK**.

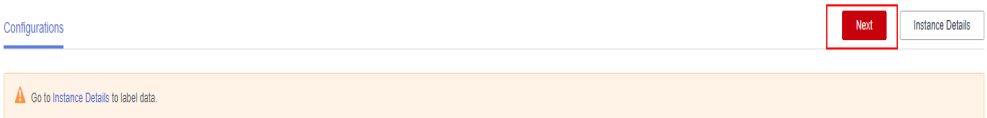
  NOTE

  The deleted labels cannot be recovered. Exercise caution when performing this operation.

Resuming Workflow Execution

After confirming data labeling, return back to the new-version ExeML. Click **Next**. Then, the workflow continues to run in sequence until all phases are executed.

Figure 4-35 Resuming the workflow execution



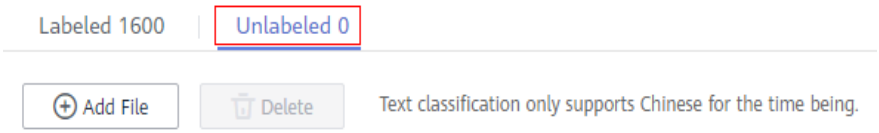
4.6.4 Training a Text Classification Model

After labeling the data, train a model. You can perform model training to obtain the required text classification model. The text used for training has at least two classifications (that is, more than two labels), and the number of texts in each classification is more than 20. Before clicking **Next**, ensure that the labeled text meets the requirements.

Procedure

1. On the ExeML page of the new version, click the name of the target project. Then, click **Instance Details** on the labeling phase to label data.

   Figure 4-36 Finding unlabeled files

2. Return to the labeling phase of the new-version ExeML, click **Next** and wait until the workflow enters the training phase.
3. Wait until the training is complete. No manual operation is required. If you close or exit the page, the system continues training until it is complete.
4. On the text classification phase, wait until the training status changes from **Running** to **Executed**.
5. After the training, click  on the text classification phase to view metric information. For details about the evaluation result parameters, see **Table 4-17**.

   Table 4-17 Evaluation result parameters

Parameter	Description
Recall	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish positive samples.
Precision	Fraction of correctly predicted samples over all samples predicted as a class. It shows the ability of a model to distinguish negative samples.
Accuracy	Fraction of correctly predicted samples over all samples. It shows the general ability of a model to recognize samples.
F1 Score	Harmonic average of the precision and recall of a model. It is used to evaluate the quality of a model. A high F1 score indicates a good model.

NOTE

An ExeML project supports multiple rounds of training, and each round generates a version. For example, the first training version is **0.0.1**, and the next version is **0.0.2**. The trained models can be managed by training version. After the trained model meets your requirements, deploy the model as a service.

4.6.5 Deploying a Text Classification Service

Deploying a Service

You can deploy a model as a real-time service that provides a real-time test UI and monitoring capabilities. After the model is trained, you can deploy a **Successful** version with ideal accuracy as a service. The procedure is as follows:

1. On the **Dashboard** page, after the service deployment status changes to **Awaiting input**, double-click **Deploy Service**. On the configuration details page, configure resource parameters.
2. On the service deployment page, select the resource specifications used for service deployment.
   - **AI Application Source**: defaults to the generated AI application.
   - **AI Application and Version**: The current AI application version is automatically selected, which is changeable.
   - **Resource Pool**: defaults to public resource pools.
   - **Traffic Ratio**: defaults to **100** and supports a value range of 0 to 100.
   - **Specifications**: Select available specifications based on the list displayed on the console. The specifications in gray cannot be used in the current environment. If there are no specifications after you select a public resource pool, no public resource pool is available in the current environment. In this case, use a dedicated resource pool or contact the administrator to create a public resource pool.
   - **Compute Nodes**: an integer ranging from 1 to 5. The default value is **1**.
   - **Auto Stop**: enables a service to automatically stop at a specified time. If this function is not enabled, the real-time service continuously runs and fees are incurred accordingly. Auto stop is enabled by default and its default value is **1 hour later**.

     The auto stop options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, enter any integer from 1 to 24 in the text box on the right.

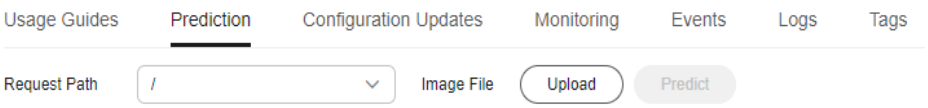
     NOTE

     You can choose the package that you have bought when you select specifications. On the configuration fee tag, you can view your remaining package quota and how much you will pay for any extra usage.
3. After configuring resources, click **Next** and confirm the operation. Wait until the status changes to **Executed**, which means the AI application has been deployed as a real-time service.

Testing a Service

After the service is deployed, click **Instance Details** to go to the real-time service details page. Click the **Prediction** tab to test the service. You can also use code to test a service. For details, see **Accessing Real-Time Services**.

Figure 4-37 Testing the service



The following describes the procedure for performing a service test after the text classification model is deployed as a service on the ExeML page.

1. After the model is deployed, you can add a text file for test. On the **ExeML** page, click the target project, go to the **Deploy Service** tab page, select the service version in the **Running** status, and enter text to be tested in the text box in the **Service Test** area.
2. Click **Predict** to perform the test. After the prediction is complete, the result is displayed in the **Test Result** pane on the right. If the model accuracy does not meet your expectation, add text data files on the **Label Data** tab page, label the files, and train and deploy the model again. **Table 4-18** describes the parameters in the prediction result. If you are satisfied with the model prediction result, call the API to access the real-time service as prompted. For details, see **Accessing Real-Time Services**.

Table 4-18 Parameters in the prediction result

Parameter	Description
predicted_label	Prediction type of the text
score	Confidence score for the predicated class

NOTE

A running real-time service keeps consuming resources. If you do not need to use the real-time service, click **Stop** in the **Version Manager** pane to stop the service so that charges will no longer be incurred. If you want to use the service again, click **Start**.

If you enable the auto stop function, the service automatically stops after the specified time and no fee is generated.

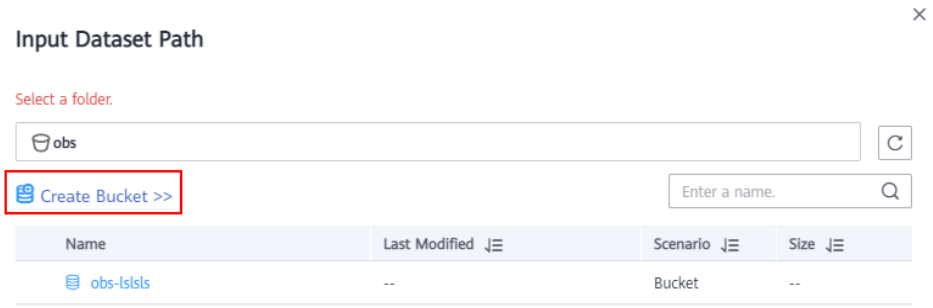
4.7 Tips

4.7.1 How Do I Quickly Create an OBS Bucket and a Folder When Creating a Project?

When creating a project, select a training data path. This section describes how to quickly create an OBS bucket and folder when you select the training data path.

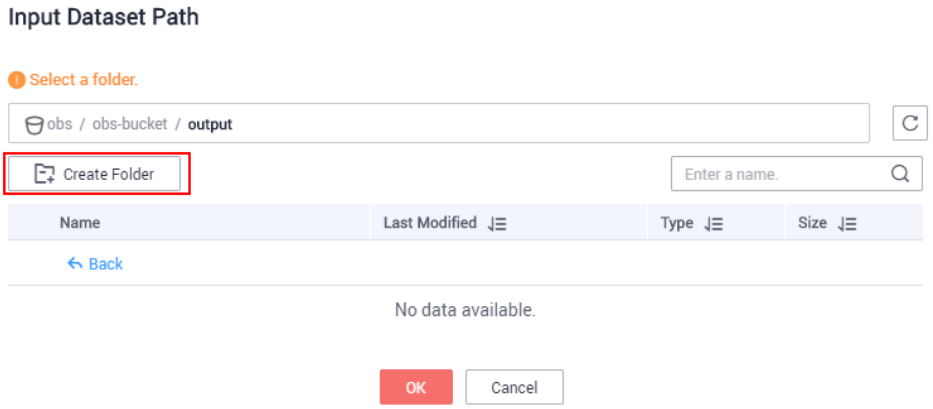
1. On the page for creating an ExeML project, click on the right of **Input Dataset Path**. The **Input Dataset Path** dialog box is displayed.
2. Click **Create Bucket**. The **Create Bucket** page is displayed. For details, see **Creating a Bucket** in *Object Storage Service Console Operation Guide*.

Figure 4-38 Creating an OBS bucket



3. Select the bucket, and click **Create Folder**. In the dialog box that is displayed, enter the folder name and click **OK**.
   - The name cannot contain the following special characters: \/:*?"<>|
   - The name cannot start or end with a period (.) or slash (/).
   - The absolute path of a folder cannot exceed 1,023 characters.
   - Any single slash (/) separates and creates multiple levels of folders at once.

Figure 4-39 Creating a folder



4.7.2 Where Are Models Generated by ExeML Stored? What Other Operations Are Supported?

Unified Model Management

For an ExeML project, after the model training is complete, the generated model is automatically displayed on the **AI Application Management > AI Applications** page. See the following figure. The model name is automatically generated by the system. Its prefix is the same as the name of the ExeML project for easy identification.

CAUTION

Models generated by ExeML cannot be downloaded.

Figure 4-40 Models generated by ExeML

AI Application Name	Latest Version	Status	Deployment Type
[redacted]	0.0.1	Normal	Real-Time Services/Batch Ser...
[redacted]	0.0.2	Normal	Real-Time Services/Batch Ser...
[redacted]	0.0.2	Normal	Real-Time Services/Batch Ser...

What Other Operations Are Supported for Models Generated by ExeML?

- **Deploying models as real-time, edge, and batch services**

  On the **ExeML** page, models can only be deployed as real-time services. You can deploy models as batch services or edge services on the **AI Application Management > AI Applications** page.

- **Creating a version**

  When creating a new version, you can select a meta model only from a ModelArts training job, OBS, model template, or custom image. You cannot create a version from the original ExeML project.

- **Deleting a model or its version**

5 Using Workflows for Low-Code AI Development

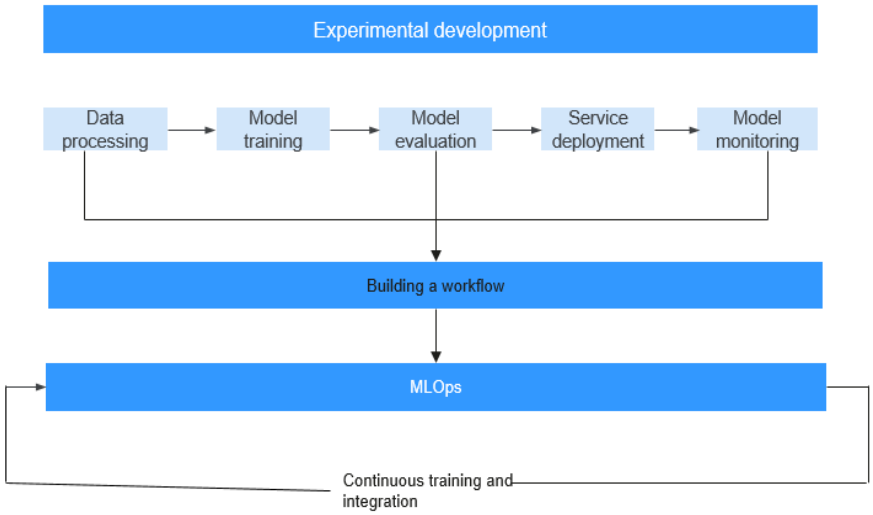
5.1 What Is Workflow?

MLOps Overview

Understanding MLOps is essential before learning about workflows.

Machine Learning Operations (MLOps) are a set of practices with machine learning (ML) and DevOps combined. The ML development process consists of project design, data engineering, model building, and model deployment. AI development is not a unidirectional pipeline job. During development, multiple iterations of experiments are performed based on the data and model results. To achieve better model results, algorithm engineers perform diverse data processing and model optimization based on the data features and labels of existing datasets. Traditional AI development ends with a one-off delivery of the final model output by iterative experimentation. As time passes after an application is released however, model drift occurs, leading to worsening effects when applying new data and features to the existing model. Iterative experimentation of MLOps forms a fixed pipeline which contains data engineering, model algorithms, and training configurations. You can use the pipeline to continuously perform iterative training on data that is being continuously generated. This ensures that the AI application of the model, built using the pipeline, is always in an optimum state.

Figure 5-1 MLOps



An entire MLOps link, which covers everything from algorithm development to service delivery and O&M, requires an implementation tool. Originally, the development and delivery processes were conducted separately. The models developed by algorithm engineers were delivered to downstream system engineers. In this process, algorithm engineers are highly involved, which is different from MLOps. There are general delivery cooperation rules in each enterprise. When it comes to project management, working process management needs to be added to AI projects as the system does not simply build and manage pipelines, but acts as a job management system.

The tool for the MLOps link must support the following features:

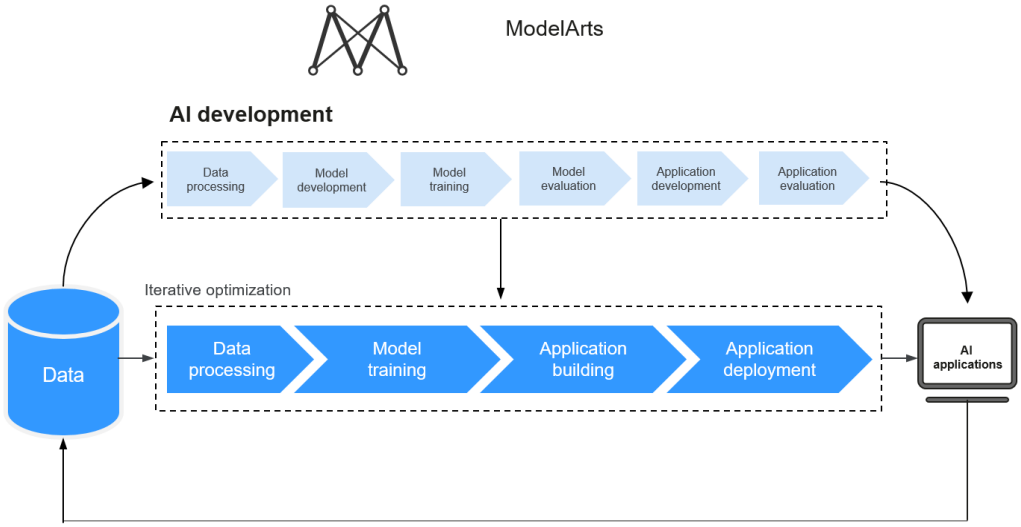
- Process analysis: Accumulated industry sample pipelines help you quickly design AI projects and processes.
- Process definition and redefinition: You can use pipelines to quickly define AI projects and design workflows for model training and release for inference.
- Resource allocation: You can use account management to allocate resource quotas and permissions to participants (including developers and O&M personnel) in the pipeline and view resource usage.
- Task arrangement: Sub-tasks can be arranged based on sub-pipelines. Additionally, notifications can be enabled for efficient management and collaboration.
- Process quality and efficiency evaluation: Pipeline execution views are provided, and checkpoints for different phases such as data evaluation, model evaluation, and performance evaluation are added so that AI project managers can easily view the quality and efficiency of the pipeline execution.
- Process optimization: In each iteration of the pipeline, you can customize core metrics and obtain affected data and causes. In this way, you can quickly determine the next iteration based on these metrics.

Workflow Overview

A workflow is a pipeline tool developed based on service scenarios for deploying models or applications. In ML, a pipeline may involve data labeling, data

processing, model development and training, model evaluation, application development, and application evaluation.

Figure 5-2 Workflow



Different from traditional ML-based model building, workflows can be used to develop production pipelines. Based on MLOps, workflows enable runtime recording, monitoring, and continuous running. The development and continuous iteration of a workflow are separated in products based on roles and concepts.

A pipeline consists of multiple phases. The functions required by the pipeline and the function parameters are called through workflow SDKs. When developing a pipeline, you can use SDKs to describe phases and the relationships between phases. Developing a pipeline is the development state of the workflow. After a pipeline is determined, you can consolidate and provide it for others to use. You do not need to pay attention to what algorithms are used in the pipeline or how the pipeline is implemented. Instead, you only need to check whether the models or applications produced by the pipeline meet the release requirements. If not, you need to check whether the data and parameters need to be adjusted for iteration. Using such a consolidated pipeline is the running state of the workflow.

The development and running states of a workflow are as follows:

- Development state: Workflow Python SDKs are used to develop and debug a pipeline.
- Running state: You can configure and run a produced pipeline in visualized mode.

Leveraging DevOps principles and practices, workflows orchestrate ModelArts capabilities to help you efficiently train, develop, and deploy AI models.

Different functions are implemented in the development and running states of a workflow.

Workflow Development State

Based on service requirements, you can use Python SDKs provided by ModelArts workflows to offer each ModelArts capability as a step in a pipeline. This is a familiar and flexible development mode for AI developers. Python SDKs support:

- Development and building: You can use Python code to create and orchestrate workflows with flexibility.
- Commissioning: The debug and run modes are supported. The run mode supports partial execution and fully execution of a workflow.
- Publishing: The debugged workflows can be fixed and published to the running state for configuration and execution.
- Experiment record: for persistence and the management of experiments.
- Sharing: Workflows can be published to AI Gallery as assets and shared with other users.

Workflow Running State

Workflows are executed in visualized mode. You only need to pay attention to some simple parameter settings to start a workflow. Running workflows are released from the development state or subscribed to from AI Gallery.

Running workflows are released from the development state or subscribed to from AI Gallery.

A running workflow supports:

- Unified configuration management: The parameters and resources required for a workflow are centrally managed.
- Easy-to-use operations: You can start, stop, retry, copy, and delete workflows.
- Running record: records historical running parameters and statuses of the workflow.

Workflow Components

A workflow is the description of a directed acyclic graph (DAG). You can develop a DAG through a workflow. A DAG consists of phases and the relationships between phases. To define a DAG, specify the execution content and sequence on phases. A green rectangle indicates a phase, and the link between phases shows the phase relationship. A DAG is actually an ordered job execution template.

Sample Workflows

ModelArts provides abundant scenario-oriented sample workflows. You can subscribe to them in **AI Gallery**.

5.2 Managing a Workflow

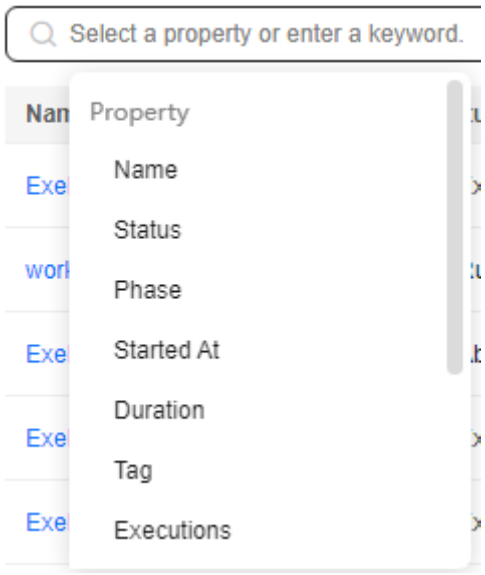
5.2.1 Searching for a Workflow

Procedure

On the workflow list page, you can use the search box to quickly search for workflows based on workflow properties.

1. Log in to the ModelArts console. In the navigation pane, choose **Development Workspace** > **Workflow**.
2. In the search box above the workflow list, filter workflows based on the required property, such as the name, status, current phase, start time, running duration, or tag.

   Figure 5-3 Property

3. Click on the right of the search box to set the content you want to display on the workflow list page and modify other display settings.
   - **Table Text Wrapping**: This feature is disabled by default. If you enable this feature, excess text will move down to the next line; otherwise, the text will be truncated.
   - **Operation Column**: This feature is enabled by default. If you enable this feature, the **Operation** column is always fixed at the rightmost position of the table.
   - **Custom Columns**: By default, all items are selected. You can select columns you want to see.
4. Click **OK**.
5. To arrange workflows by a specific property, click in the table header.

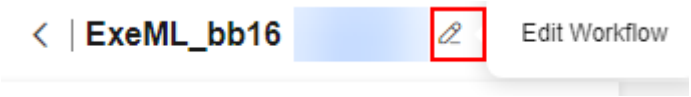
Editing a Workflow Name and Tag

You can rename a workflow and add a tag to make it easier to find.

1. On the ModelArts console, choose **Development Workspace** > **Workflow** from the navigation pane. The workflow list page is displayed.

2. On the workflow list page, click the name of the target workflow.
3. Click in the upper left corner.

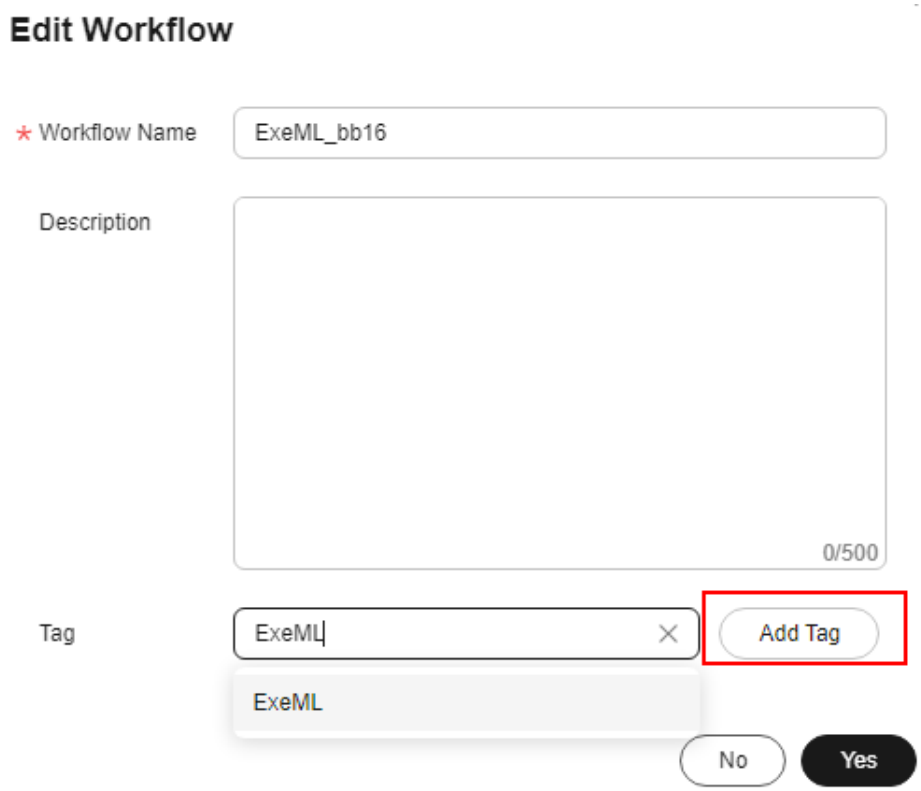
Figure 5-4 Editing a workflow



4. In the displayed dialog box, modify the workflow name and tag.

   Enter a tag and click **Add Tag**. The new tag is displayed below. You can add multiple tags at a time. After the tags are added, click **Yes**.

Figure 5-5 Adding a tag



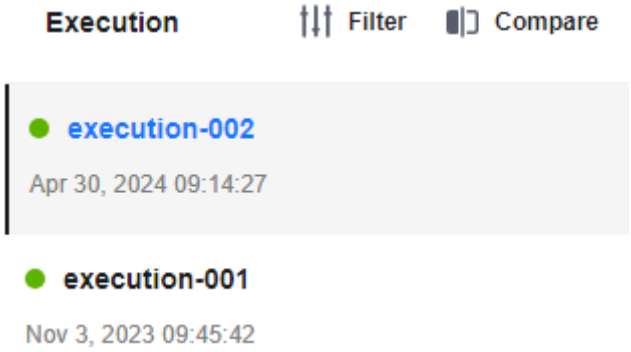
5. Workflows with tags can be filtered by tag in the search box.

5.2.2 Viewing the Running Records of a Workflow

All runtime statuses of a workflow are recorded.

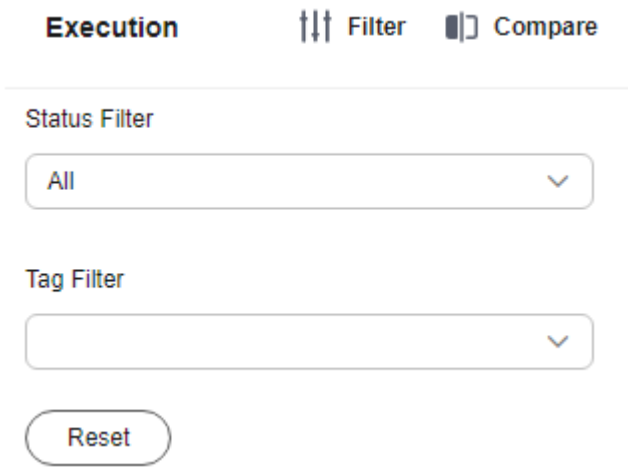
1. On the workflow list page, click the name of the target workflow.
2. On the workflow details page, view all runtime records of the workflow in the left pane.

Figure 5-6 Viewing execution records



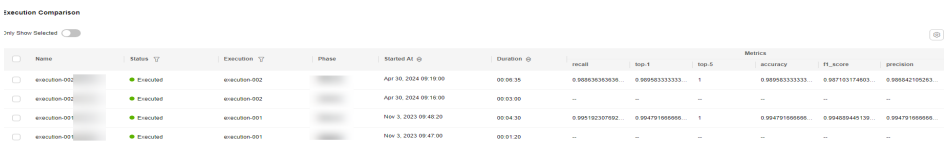
3. Delete or edit the runtime records, or rerun the workflow.
   - To delete a runtime record that is no longer needed, click **Delete**. In the displayed dialog box, click **Yes**.
   - To distinguish a runtime record from others, click **Edit Tag** to add a tag to it.
   - To rerun the workflow, click **Rerun** on a runtime record.
4. Filter and compare all runtime records of the workflow.
   - **Filter**: You can filter all runtime records by status or tag.

     Figure 5-7 Filtering

   - **Compare**: You can compare all runtime records by status, execution record, start time, duration, and metrics.

     Figure 5-8 Comparison

After you click **Start** to run a workflow, the execution record list is refreshed. In addition, the data is updated on both the DAG and dashboard. An execution record is added after each startup.

On the workflow details page, you can click any phase to view its information, including attributes, input, output, and parameters.

5.2.3 Managing a Workflow

Starting a Workflow

Log in to the ModelArts console. In the navigation pane, choose **Development Workspace** > **Workflow**.

You can run a workflow in any of the following ways:

- On the workflow list page, click **Start** in the **Operation** column. In the displayed dialog box, click **OK**.
- On the runtime configuration page, click **Start** in the upper right corner. In the displayed dialog box, click **OK**.
- On the workflow configuration page, click **Start** in the upper right corner. In the displayed dialog box, click **OK**.

  NOTE

  After a workflow is started, you will be charged on a pay-per-use basis. After the workflow is complete, you can stop it to avoid unnecessary fees.

Stopping a Workflow

Log in to the ModelArts console. In the navigation pane, choose **Development Workspace** > **Workflow**.

You can stop a running workflow in either of the following ways:

- Workflow list page

  When a workflow is running, the **Stop** button is available in the **Operation** column. Click **Stop**. In the displayed dialog box, click **OK**.
- Click the name of a running workflow and click **Stop** in the upper right corner of the displayed page. In the displayed dialog box, click **OK**.

  NOTE

  The **Stop** button is available only for a workflow that is running.

  After a workflow is stopped, the associated training jobs and real-time services are also stopped.

Copying a Workflow

A workflow can have only one running instance. If you want to concurrently run a workflow, copy the workflow. To do so, click **More** in the **Operation** column and select **Copy**. In the displayed dialog box, a new name is automatically generated in the format of "Original workflow name_copy".

You can rename the new workflow. Ensure that the name complies with naming specifications.

NOTE

A workflow name is 1 to 64 characters long, starting with a letter and containing only letters, digits, underscores (_), and hyphens (-).

Deleting a Workflow

Log in to the ModelArts console. In the navigation pane, choose **Development Workspace** > **Workflow**.

You can delete a workflow in either of the following ways:

- Workflow list page
  a. Click **More** in the **Operation** column and select **Delete**.
  b. In the displayed dialog box, enter **delete** and click **OK**.
- Runtime configuration page

  Click **Delete** in the upper right corner of the page. In the displayed dialog box, enter **DELETE** and click **OK**.

  NOTE

  - Deleted workflows cannot be recovered.
  - When a workflow is deleted, the associated real-time service or training job is not deleted. You need to manually delete the real-time service or training job on the **Model Training** > **Training Jobs** page or **Model Deployment** > **Real-Time Services** page.

5.2.4 Retrying, Stopping, or Running a Workflow Phase

Retrying, Stopping, or Proceeding a Workflow Phase

- Retrying a phase

  If executing a single phase failed, you can click **Retry** to re-execute the current phase without restarting the workflow. Before the retry, you can modify configurations on the **Permission Management** page. The modification takes effect after the affected phase is retried.
- Stopping a phase

  Click a phase to view its details. On this page, you can stop the running phase.
- Proceeding a phase

  If parameters need to be configured during the runtime of a single phase, the phase is awaiting operation. After the parameters are configured, you can click **Proceed** to proceed to the execution of the current phase.

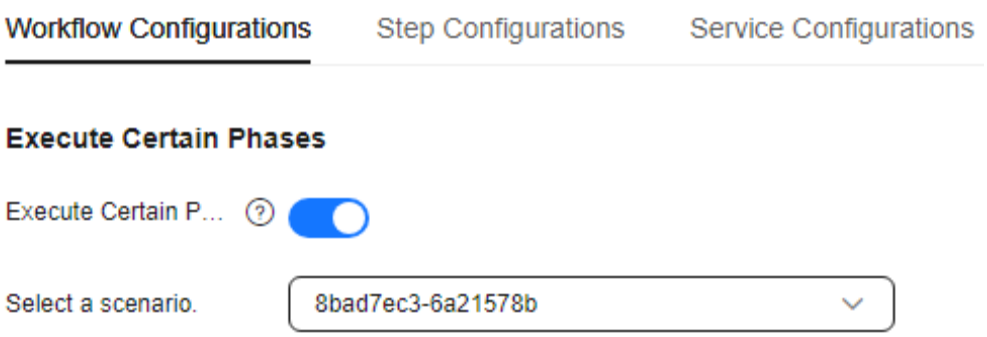
Running Specific Workflow Phases

To reduce the time consumed by repeated execution in large-scale and complex workflows, you can choose specific phases to execute in sequence.

To run specific workflow phases, specify the target phases when developing a workflow. The procedure is as follows:

1. Predefine the phases you want to execute when you use the SDK to create a workflow. For details, see **Specifying Certain Phases to Run in a Workflow**.
2. When configuring a workflow, enable **Execute Certain Phases**, select phases to be executed, and configure parameters for these phases.

   Figure 5-9 Partial execution



3. After saving the configuration, click **Start** to execute certain phases.

5.3 Workflow Development Command Reference

5.3.1 Core Concepts of Workflow Development

Workflow

A workflow is a DAG that consists of phases and the relationships between phases.

A directed line segment shows the dependency between phases. The dependency decides the order of phase execution. In this example, the workflow runs from left to right after it starts. The DAG can handle the multi-branch structure as well. You can design the DAG flexibly according to the real situation. In the multi-branch situation, phases in parallel branches can run at the same time. For details, see **Configuring Multi-Branch Phase Data**.

Table 5-1 Workflow

Parameter	Description	Mandatory	Data Type
name	Workflow name. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter.	Yes	str
desc	Workflow description	Yes	str
steps	Phases contained in a workflow	Yes	list[Step]

Parameter	Description	Mandatory	Data Type
storages	Unified storage objects	No	Storage or list[Storage]
policy	Workflow configuration policy, which is used for partial execution	No	Policy

Step

A step is the smallest unit of a workflow. In a DAG, a step is also a phase. Different types of steps have different service abilities. The main parts of a step are as follows.

Table 5-2 Step

Parameter	Description	Mandatory	Data Type
name	Phase name. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter.	Yes	str
title	Title of a phase, which is displayed in the DAG. If this parameter is not configured, the name is displayed by default.	No	str
step_type	Type of a phase, which determines the function of the phase	Yes	enum
inputs	Inputs of a phase	No	AbstractInput or list[AbstractInput]
outputs	Outputs of a phase	No	AbstractOutput or list[AbstractOutput]
properties	Node properties	No	dict
policy	Phase execution policy, which includes the phase scheduling interval, the phase execution timeout interval, and the option to skip phase execution	No	StepPolicy
depend_steps	List of dependency phases. This parameter determines the DAG structure and phase execution sequence.	No	Step or list[Step]

Table 5-3 StepPolicy

Parameter	Description	Mandatory	Data Type
poll_interval_seconds	Phase scheduling interval. The default value is 1 second.	Yes	str
max_execution_minutes	Phase execution timeout interval. The default value is 10080 minutes, that is, 7 days.	Yes	str
skip_conditions	Conditions that determine whether a phase is skipped	No	Condition or condition list

Step is a superclass of a phase. It has a conceptual role and is not used directly by you. Different types of phase are created based on functions, including **CreateDatasetStep**, **LabelingStep**, **DatasetImportStep**, **ReleaseDatasetStep**, **JobStep**, **ModelStep**, **ServiceStep** and **ConditionStep**. For details, see **Creating Workflow Phases**.

Data

Data objects are used for phase input and are classified into the following types:

- Actual data objects, which are specified when you create a workflow
  - Dataset: defines existing datasets. This object is used for data labeling and model training.
  - LabelTask: defines existing labeling jobs. This object is usually used for data labeling and dataset version release.
  - OBSPath: defines an OBS path. This object is used for model training, dataset import, and model import.
  - ServiceData: defines an existing service. This object is used only for service update.
  - SWRImage: defines an existing SWR path. This object is used for model registration.
  - GalleryModel: defines a model subscribed from AI Gallery. This object is used for model registration.
- Placeholder data objects, which are specified when a workflow is running
  - DatasetPlaceholder: defines datasets to be specified when a workflow is running. This object is used for data labeling and model training.
  - LabelTaskPlaceholder: defines labeling jobs to be specified when a workflow is running. This object is used for data labeling and dataset version release.
  - OBSPlaceholder: defines an OBS path to be specified when a workflow is running. This object is used for model training, dataset import, and model import.
  - ServiceUpdatePlaceholder: defines existing services to be specified when a workflow is running. This object is used only for service update.

  - SWRImagePlaceholder: defines an SWR path to be specified when a workflow is running. This object is used for model registration.
  - ServiceInputPlaceholder: defines model information required for service deployment when a workflow is running. This object is used only for service deployment and update.
  - DataSelector: supports multiple data types. Currently, this object can be used only on the job phase (only OBS or datasets are supported).
- Data selection object:

  DataConsumptionSelector: selects a valid output from the outputs of multiple dependency phases as the data input. This object is usually used for conditional branching. (When creating a workflow, the output of which dependency phase will be used as the data input source is not specified. The data input source should be automatically selected based on the actual execution status of the dependency phases.)

Table 5-4 Dataset

Parameter	Description	Mandatory	Data Type
dataset_name	Dataset name	Yes	str
version_name	Dataset version	No	str

Example:

```
example = Dataset(dataset_name = "**", version_name = "**")
# Obtain the dataset name and version name from ModelArts datasets.
```

NOTE

When a dataset is used as the input of a phase, configure **version_name** based on service requirements. For example, **version_name** is not required for LabelingStep and ReleaseDatasetStep, but mandatory for JobStep.

Table 5-5 LabelTask

Parameter	Description	Mandatory	Data Type
dataset_name	Dataset name	Yes	str
task_name	Labeling job name	Yes	str

Example:

```
example = LabelTask(dataset_name = "**", task_name = "**")
# Obtain the dataset name and labeling job name from ModelArts datasets of the new version.
```


Table 5-6 OBSPath

Parameter	Description	Mandatory	Data Type
obs_path	OBS path	Yes	str, Storage

Example:

```
example = OBSPath(obs_path = "**")
# Obtain the OBS path from Object Storage Service.
```

Table 5-7 ServiceData

Parameter	Description	Mandatory	Data Type
service_id	Service ID	Yes	str

Example:

```
example = ServiceData(service_id = "**")
# Obtain the service ID in ModelArts Real-Time Services. This object describes a specified real-time service and is used for service update.
```

Table 5-8 SWRImage

Parameter	Description	Mandatory	Data Type
swr_path	SWR path to a container image	Yes	str

Example:

```
example = SWRImage(swr_path = "**")
# Container image path, which is used as the input for model registration
```

Table 5-9 GalleryModel

Parameter	Description	Mandatory	Data Type
subscription_id	Subscription ID of a subscribed model	Yes	str
version_num	Version number of a subscribed model	Yes	str

Example:


```
example = GalleryModel(subscription_id="**", version_num="**")
# Subscribed model object, which is used as the input of the model registration phase
```

Table 5-10 DatasetPlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
data_type	Data Type	No	DataTypeEnum
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool
default	Default value of a data object	No	Dataset

Example:

```
example = DatasetPlaceholder(name = "**", data_type = DataTypeEnum.IMAGE_CLASSIFICATION)
# Dataset object placeholder. Configure data_type to specify supported data types.
```

Table 5-11 OBSPlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
object_type	OBS object type. Only "file" and "directory" are supported.	Yes	str
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool
default	Default value of a data object	No	OBSPath

Example:

```
example = OBSPlaceholder(name = "**", object_type = "directory" )
# OBS object placeholder. You can set object_type to file or directory.
```

Table 5-12 LabelTaskPlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str

Parameter	Description	Mandatory	Data Type
task_type	Type of a labeling job	No	LabelTaskTypeEnum
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool

Example:

```
example = LabelTaskPlaceholder(name = "**")
# LabelTask object placeholder
```

Table 5-13 ServiceUpdatePlaceholder

Parameter	Description	Mandatory	Data Type
name	Parameter	Yes	str
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool

Example:

```
example = ServiceUpdatePlaceholder(name = "**")
# ServiceData object placeholder, which is used as the input for service update
```

Table 5-14 SWRImagePlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool

Example:

```
example = SWRImagePlaceholder(name = "**" )
# SWRImage object placeholder, which is used as the input for model registration
```


Table 5-15 ServiceInputPlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
model_name	Model name	Yes	str or Placeholder
model_version	Model version	No	str
envs	Environment variables	No	dict
delay	Whether service deployment information is configured when the phase is running. The default value is **True**.	No	bool

Example:

```
example = ServiceInputPlaceholder(name = "**" , model_name = "model_name")
# This object is used as the input for service deployment or service update.
```

Table 5-16 DataSelector

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
data_type_list	Supported data types. Currently, only **obs** and **dataset** are supported.	Yes	list
delay	Whether the data object is configured when the phase is running. The default value is **False**.	No	bool

Example:

```
example = DataSelector(name = "**" ,data_type_list=["obs", "dataset"])
# This object is used as the input of the job phase.
```

Table 5-17 DataConsumptionSelector

Parameter	Description	Mandatory	Data Type
data_list	Output data objects of a dependency phase	Yes	list

Example:

```
example = DataConsumptionSelector(data_list=[step1.outputs["step1_output_name"].as_input(), step2.outputs["step2_output_name"].as_input()])
# Use the valid output from either step 1 or step 2 as the input. If step 1 is skipped and has no output, use the valid output from step 2 as the input. (Make sure that data_list has only one valid output.)
```

5.3.2 Configuring Workflow Parameters

Description

A workflow parameter is a placeholder object that can be configured when the workflow runs. The following data types are supported: int, str, bool, float, Enum, dict, and list. You can display fields (such as algorithm hyperparameters) in a phase as placeholders in a transparent way. You can modify and use the default values that are set for them.

Parameter Overview (Placeholder)

Parameter	Description	Mandatory	Data Type
name	Parameter name, which must be globally unique.	Yes	str
placeholder_type	Parameter type. The mapping between placeholder types and actual data types: PlaceholderType.INT -> int PlaceholderType.STR -> str PlaceholderType.BOOL -> bool PlaceholderType.FLOAT -> float PlaceholderType.ENUM -> Enum PlaceholderType.JSON -> dict PlaceholderType.LIST -> list • When the type is PlaceholderType.ENUM, the **enum_list** field cannot be empty. • When the type is PlaceholderType.LIST, the **placeholder_format** field cannot be empty and can only be set to **str**, **int**, **float**, or **bool**, indicating the data types in the list.	Yes	PlaceholderType
default	Default parameter value. The data type must be the same as that of **placeholder_type**.	No	Any
placeholder_format	Supported data formats. Currently, **obs**, **flavor**, **train_flavor**, **swr**, and **pacific** are supported.	No	str

Parameter	Description	Mandatory	Data Type
delay	Whether parameters are set when the workflow is running. The default value is **False**, indicating that parameters are set before the workflow runs. If the value is **True**, parameters are set in an action of the phase where they are needed.	No	bool
description	Parameter description.	No	str
enum_list	List of enumerated values of a parameter. This parameter is mandatory only for parameters of PlaceholderType.ENUM type.	No	list
constraint	Constraints on parameters. This parameter only supports the constraints of training specifications and is not visible to you.	No	dict
required	Whether the parameter is mandatory. ● The default value is **True**. ● This parameter cannot be set to **False** for **Delay**. This parameter is optional at the frontend during execution.	No	bool

Examples

- Integer parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_int", placeholder_type=wf.PlaceholderType.INT, default=1, description="This is an integer parameter.")
```
- String parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_str", placeholder_type=wf.PlaceholderType.STR, default="default_value", description="This is a string parameter.")
```
- Bool parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_bool", placeholder_type=wf.PlaceholderType.BOOL, default=True, description="This is a bool parameter.")
```
- Float parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_float", placeholder_type=wf.PlaceholderType.FLOAT, default=0.1, description="This is a float parameter.")
```
- Enumeration parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_enum", placeholder_type=wf.PlaceholderType.ENUM, default="a", enum_list=["a", "b"], description="This is an enumeration parameter.")
```
- Dictionary parameter


```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_dict", placeholder_type=wf.PlaceholderType.JSON, default={"key": "value"}, description="This is a dictionary parameter.")
```

- List parameter

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_list", placeholder_type=wf.PlaceholderType.LIST, default=[1, 2], placeholder_format="int", description="This is a list parameter and its value is an integer.")
```

5.3.3 Configuring the Input and Output Paths of a Workflow

Description

Unified storage is used for workflow directory management. It centrally manages all storage paths of a workflow with these functions:

- Input directory management: When developing a workflow, you can centrally manage all data storage paths. You can store data and configure the root directory based on your own requirements. This function orchestrates directories but does not create them.
- Output directory management: When developing a workflow, you can centrally manage all output paths. You do not need to create output directories. Instead, you only need to configure the root path before the workflow runs and view the output data in the specified directories based on your directory orchestration rules. In addition, multiple executions of the same workflow are output to different directories, isolating data for different executions.

Common Usage

- **InputStorage** (Path concatenation)

  This object is used to centrally manage input directories. The following is an example:

  ```
  import modelarts.workflow as wf
  storage = wf.data.InputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.
  input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # Add a slash (/) after a directory, for example, storage.join("/input/data/").
  ```

  When a workflow is running, if the root path of the storage object is **/root/**, the obtained path will be **/root/directory_path**.

- **OutputStorage** (Directory creation)

  This object is used to centrally manage output directories and ensure that multiple executions of the same workflow are output to different directories. The following is an example:

  ```
  import modelarts.workflow as wf
  storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.
  output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # Only a directory can be created but not files.
  ```

  When a workflow is running, if the root path of the storage object is set to **/root/**, the system will automatically create a relative directory and the obtained path will be **/root/***Execution ID*/**directory_path**.

Advanced Usage

Storage

This object contains capabilities of InputStorage and OutputStorage and can be flexibly used based on your needs.

Parameter	Description	Mandatory	Data Type
name	Name.	Yes	str
title	If this parameter is left blank, the value of **name** is used by default.	No	str
description	Description.	No	str
create_dir	Whether to create a directory. The default value is **False**.	No	bool
with_execution_id	Whether to combine **execution_id** when a directory is created. The default value is **False**. This parameter can be set to **True** only when **create_dir** is set to **True**.	No	bool

The following is an example:

- Implementing **InputStorage** capabilities

  ```
  import modelarts.workflow as wf
  # Create a Storage object (with_execution_id=False, create_dir=False).
  storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=False, create_dir=False)
  input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # Add a slash (/) after a directory, for example, storage.join("/input/data/").

  When a workflow is running, if the root path of the storage object is /root/, the obtained path will be /root/directory_path.
  ```

- Implementing **OutputStorage** capabilities

  ```
  import modelarts.workflow as wf
  # Create a Storage object (with_execution_id=True, create_dir=True).
  storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=True, create_dir=True)
  output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # Only a directory can be created.

  When a workflow is running, if the root path of the storage object is set to /root/, the system will automatically create a relative directory and the obtained path will be /root/Execution ID/directory_path.
  ```

- Implementing different capabilities of a Storage object through the **join** method

  ```
  import modelarts.workflow as wf
  # Create a Storage object. Assume that the root directory of the Storage object is /root/.
  storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=False, create_dir=False)
  input_data1 = wf.data.OBSPath(obs_path = storage) # The obtained path is /root/.
  input_data2 = wf.data.OBSPath(obs_path = storage.join("directory_path")) # The obtained path is /root/directory_path. Ensure that the path exists.
  output_path1 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path", with_execution_id=False, create_dir=True)) # The system automatically creates a directory /root/directory_path.
  output_path2 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path", with_execution_id=True, create_dir=True)) # The system automatically creates a directory /root/Execution ID/directory_path.
  ```


Chain call is supported for **Storage**.

The following is an example:

```
import modelarts.workflow as wf
# Create a base class Storage object. Assume that the root directory of the Storage object is /root/.
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=False, create_dir=Fals)
input_storage = storage.join("directory_path_1") # The obtained path is /root/directory_path_1.
input_storage_next = input_storage.join("directory_path_2") # The obtained path is /root/directory_path_1/
directory_path_2.
```

Examples

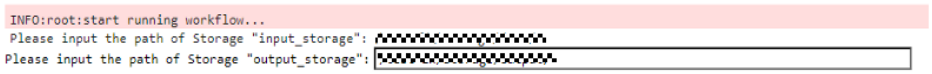
Unified storage is mainly used in the job phase. The following code uses a workflow that contains only the training phase as an example.

```
from modelarts import workflow as wf

# Create an InputStorage object. Assume that the root directory of the Storage object is /root/input-data/.
input_storage = wf.data.InputStorage(name="input_storage", title="title_info",
description="description_info") # Only name is mandatory.

# Create an OutputStorage object. Assume that the root directory of the Storage object is /root/output/.
output_storage = wf.data.OutputStorage(name="output_storage_name", title="title_info",
description="description_info") # Only name is mandatory.

# Use JobStep to define a training phase, and set OBS paths for storing inputs and outputs.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters,
including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(subscription_id="subscription_ID",
item_version_id="item_version_ID"), # Algorithm used for training. In this example, an algorithm subscribed
to from AI Gallery is used.
    inputs=[
        wf.steps.JobInput(name="data_url_1", data=wf.data.OBSPath(obs_path = input_storage.join("/
dataset1/new.manifest"))),  # The obtained path is /root/input-data/dataset1/new.manifest.
        wf.steps.JobInput(name="data_url_1", data=wf.data.OBSPath(obs_path = input_storage.join("/
dataset1/new.manifest"))),  # The obtained path is /root/input-data/dataset1/new.manifest.
    ],
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/model/"))), # The training output
path is /root/output/Execution ID/model/.
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
                flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")
        ),
        log_export_path=wf.steps.job_step.LogExportPath(obs_url=output_storage.join("/logs/"))  # The log
output path is /root/output/Execution ID/logs/.
    )# Training flavors
)

# Define a workflow that contains only the job phase.
workflow = wf.Workflow(
    name="test-workflow",
    desc="this is a test workflow",
    steps=[job_step],
    storages=[input_storage, output_storage] # Add Storage objects used in this workflow.
)
```

Configuring Root Paths in the Development State

Use the **run** method of the workflow object, and input root paths in the text box that is displayed when the workflow starts to run.

Figure 5-10 Inputting root paths



You must enter a valid path. If the path does not exist, an error will occur. The path format must be */Bucket name/Folder path/*.

Configuring Root Paths in the Running State

Use the **release** method of the workflow object to release the workflow to the running state. On the ModelArts console, go to the **Workflow** page, find the target workflow, and configure root paths.

Figure 5-11 Configuring root paths

5.3.4 Creating Workflow Phases

5.3.4.1 Creating a Dataset Phase

Description

This phase integrates capabilities of the ModelArts dataset module, allowing you to create datasets of the new version. This phase is used to centrally manage existing data by creating datasets. It is usually followed by a dataset import phase or a labeling phase.

Parameter Overview

You can use CreateDatasetStep to create a dataset creation phase. The following is an example of defining a CreateDatasetStep.

Table 5-18 CreateDatasetStep

Parameter	Description	Mandatory	Data Type
name	Name of a dataset creation phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str

Parameter	Description	Mandatory	Data Type
inputs	Inputs of the dataset creation phase.	Yes	CreateDatasetInput or a list of CreateDatasetInput
outputs	Outputs of the dataset creation phase.	Yes	CreateDatasetOutput or a list of CreateDatasetOutput
properties	Configurations for dataset creation.	Yes	DatasetProperties
title	Title for frontend display.	No	str
description	Description of the dataset creation phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-19 CreateDatasetInput

Parameter	Description	Mandatory	Data Type
name	Input name of the dataset creation phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of the dataset creation phase.	Yes	OBS object. Currently, only OBSPath, OBSConsumption, OBSPlaceholder, and DataConsumptionSelector are supported.

Table 5-20 CreateDatasetOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the dataset creation phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str
config	Output configurations of the dataset creation phase.	Yes	Currently, only OBSOutputConfig is supported.

Table 5-21 DatasetProperties

Parameter	Description	Mandatory	Data Type
dataset_name	Dataset name. The value contains 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.	Yes	str, Placeholder
dataset_format	Dataset format. The default value is 0, indicating the file type.	No	**0**: file **1**: table
data_type	Data type. The default value is **FREE_FORMAT**.	No	DataTypeEnum
description	Description	No	str
import_data	Whether to import data. The default value is **False**. Currently, only table data is supported.	No	bool
work_path_type	Type of the dataset output path. Currently, only OBS is supported. The default value is **0**.	No	int
import_config	Configurations for label import. The default value is **None**. When creating a dataset based on labeled data, you can specify this parameter to import labeling information.	No	ImportConfig

Table 5-22 Importconfig

Parameter	Description	Mandatory	Data Type
import_annotations	Whether to automatically import the labeling information in the input directory, supporting detection, image classification, and text classification. The options are as follows: • **true**: The labeling information in the input directory is imported. (Default) • **false**: The labeling information in the input directory is not imported.	No	str, Placeholder
import_type	Import mode. The options are as follows: • **dir**: imported from an OBS path • **manifest**: imported from a manifest file	No	**0**: file type ImportTypeEnum
annotation_format_config	Configurations of the imported labeling format.	No	DAnnotationFormaTypeEtConumfiglist

Table 5-23 AnnotationFormatConfig

Parameter	Description	Mandatory	Data Type
format_name	Name of a labeling format	No	AnnotationForma-tEnum
scene	Labeling scenario, which is optional	No	LabelTaskTypeEnum

Enumeration	Value
ImportTypeEnum	DIR MANIFEST

Enumeration	Value
DataTypeEnum	IMAGE TEXT AUDIO TABULAR VIDEO FREE_FORMAT
AnnotationFormatEnum	MA_IMAGE_CLASSIFICATION_V1 MA_IMAGENET_V1 MA_PASCAL_VOC_V1 YOLO MA_IMAGE_SEGMENTATION_V1 MA_TEXT_CLASSIFICATION_COMBINE_V1 MA_TEXT_CLASSIFICATION_V1 MA_AUDIO_CLASSIFICATION_DIR_V1

Examples

There are two scenarios:

- Creating a dataset using unlabeled data
- Creating a dataset using labeled data with labels imported

Creating a dataset using unlabeled data

Data preparation: Store unlabeled data in an OBS folder.

```
from modelarts import workflow as wf
# Use CreateDatasetStep to create a dataset of the new version using OBS data.

# Define parameters of the dataset output path.
dataset_output_path = wf.Placeholder(name="dataset_output_path",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# Define the dataset name.
dataset_name = wf.Placeholder(name="dataset_name", placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset", # Name of a dataset creation phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Creation", # Title, which defaults to the value of name
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep inputs, configured when the workflow is running; the data field can also be represented by the wf.data.OBSPath(obs_path="fake_obs_path") object.
    outputs=wf.steps.CreateDatasetOutput(name="output_name",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep outputs
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # If the dataset name does not exist, a dataset will be created using this name. If the dataset name exists, the corresponding dataset will be used.
        data_type=wf.data.DataTypeEnum.IMAGE, # Data type of the dataset, for example, image
```



```
    )
)
# Ensure that the dataset name is not used by others under the account. Otherwise, the dataset created by others will be used in the subsequent phases.

workflow = wf.Workflow(
    name="create-dataset-demo",
    desc="this is a demo workflow",
    steps=[create_dataset]
)
```

Creating a dataset using labeled data with labels imported

Data preparation: Store labeled data in an OBS folder.

For details about specifications for importing labeled data from an OBS directory, see **Specifications for Importing Data from an OBS Directory**.

```
from modelarts import workflow as wf
# Use CreateDatasetStep to create a dataset of the new version using OBS data.

# Define parameters of the dataset output path.
dataset_output_path = wf.Placeholder(name="dataset_placeholder_name",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# Define the dataset name.
dataset_name = wf.Placeholder(name="dataset_placeholder_name",
placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset", # Name of a dataset creation phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Creation", # Title, which defaults to the value of name
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep inputs, configured when the workflow is running; the data field can also be represented by the wf.data.OBSPath(obs_path="fake_obs_path") object.
    outputs=wf.steps.CreateDatasetOutput(name="output_name",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep outputs
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # If the dataset name does not exist, a dataset will be created using this name. If the dataset name exists, the corresponding dataset will be used.
        data_type=wf.data.DataTypeEnum.IMAGE, # Data type of the dataset, for example, image
        import_config=wf.steps.ImportConfig(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # Labeling format of labeled data
                scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # Labeling scene
            ]
        )
    )
)
# Ensure that the dataset name is not used by others under the account. Otherwise, the dataset created by others will be used in the subsequent phases.

workflow = wf.Workflow(
    name="create-dataset-demo",
    desc="this is a demo workflow",
    steps=[create_dataset]
)
```


5.3.4.2 Creating a Dataset Labeling Phase

Description

This phase integrates capabilities of the ModelArts dataset module, allowing you to label datasets. The labeling phase is used to create labeling jobs or label existing jobs.

Parameter Overview

You can use LabelingStep to create a labeling phase. The following is an example of defining a LabelingStep.

Table 5-24 LabelingStep

Parameter	Description	Mandatory	Data Type
name	Name of a labeling phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
inputs	Inputs of the labeling phase.	Yes	LabelingInput or LabelingInput list
outputs	Outputs of the labeling phase.	Yes	LabelingOutput or LabelingOutput list
properties	Configurations for dataset labeling.	Yes	LabelTaskProperties
title	Title for frontend display.	No	str
description	Description of the labeling phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-25 LabelingInput

Parameter	Description	Mandatory	Data Type
name	Input name of the labeling phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of the labeling phase.	Yes	Dataset or labeling job object. Currently, only Dataset, DatasetConsumption, DatasetPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, and DataConsumptionSelector are supported.

Table 5-26 LabelingOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the labeling phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str

Table 5-27 LabelTaskProperties

Parameter	Description	Mandatory	Data Type
task_type	Type of a labeling job. Jobs of the specified type are returned.	Yes	LabelTaskTypeEnum
task_name	Labeling job name. The value contains 1 to 100 characters, including only letters, digits, hyphens (-), and underscores (_). This parameter is mandatory when the input is a dataset object.	No	str, Placeholder
labels	Labels to be created.	No	Label
properties	Attributes of a labeling job. You can update this field to record custom information.	No	dict

Parameter	Description	Mandatory	Data Type
auto_sync_dataset	Whether to automatically synchronize the result of a labeling job to the dataset. The options are as follows: • **true**: The labeling result of the labeling job is automatically synchronized to the dataset. (Default) • **false**: The labeling result of the labeling job is not automatically synchronized to the dataset.	No	bool
content_labeling	Whether to enable content labeling for speech paragraph labeling. This function is enabled by default.	No	bool
description	Labeling job description. The description contains 0 to 256 characters and does not support the following special characters: ^!<>=&"'	No	str

Table 5-28 Label

Parameter	Description	Mandatory	Data Type
name	Tag name	No	str

Parameter	Description	Mandatory	Data Type
property	Basic attribute key-value pair of a label, such as color and shortcut keys	No	str, dic, Placeholder
type	Tag type	No	LabelTypeEnum

Enumeration	Value
LabelTaskTypeEnum	IMAGE_CLASSIFICATION OBJECT_DETECTION IMAGE_SEGMENTATION TEXT_CLASSIFICATION NAMED_ENTITY_RECOGNITION TEXT_TRIPLE AUDIO_CLASSIFICATION SPEECH_CONTENT SPEECH_SEGMENTATION DATASET_TABULAR VIDEO_ANNOTATION FREE_FORMAT

Sample Code of a Dataset Labeling Phase

There are three scenarios:

- Scenario 1: Creating a labeling job for a specified dataset and labeling the dataset

  Scenarios:

  – You have created only one unlabeled dataset and need to label it when the workflow is running.

  – After a dataset is imported, the dataset needs to be labeled.

  Data preparation: Create a dataset on the ModelArts console.

```
from modelarts import workflow as wf
# Use LabelingStep to create a labeling job for the input dataset and label it.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Define the name parameters of the labeling job.
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # Name of the labeling phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
```

```
unique in a workflow.
    title="Dataset Labeling", # Title, which defaults to the value of name
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,   # Labeling job type, for example, image classification
        task_name=task_name   # If the labeling job name does not exist, a job will be created using this name. If the labeling job name exists, the corresponding job will be used.
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=dataset), # LabelingStep inputs. The dataset object is configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name") for the data field.
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep outputs
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

- Scenario 2: Labeling a specified job

  Scenarios:

  – You have created a labeling job and need to label it when the workflow is running.

  – After a dataset is imported, the dataset needs to be labeled.

  Data preparation: Create a labeling job using a specified dataset on the ModelArts console.

```
from modelarts import workflow as wf
# Input a labeling job and label it.

# Define a dataset labeling job.
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

labeling = wf.steps.LabelingStep(
    name="labeling", # Name of the labeling phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Labeling", # Title, which defaults to the value of name
    inputs=wf.steps.LabelingInput(name="input_name", data=label_task), # LabelingStep inputs. The labeling job object is configured when the workflow is running. You can also use wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name") for the data field.
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep outputs
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

- Scenario 3: Creating a labeling job based on the output of the dataset creation phase

  Scenario: The outputs of the dataset creation phase are used as the inputs of the labeling phase.

```
from modelarts import workflow as wf

# Define parameters of the dataset output path.
dataset_output_path = wf.Placeholder(name="dataset_output_path", placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# Define the dataset name.
dataset_name = wf.Placeholder(name="dataset_name", placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
```



```
    name="create_dataset", # Name of a dataset creation phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Creation", # Title, which defaults to the value of name
    inputs=wf.steps.CreateDatasetInput(name="input_name", data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep inputs, configured when the workflow is running; the data field can also be represented by the wf.data.OBSPath(obs_path="fake_obs_path") object.
    outputs=wf.steps.CreateDatasetOutput(name="create_dataset_output", config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep outputs
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # If the dataset name does not exist, a dataset will be created using this name. If the dataset name exists, the corresponding dataset will be used.
        data_type=wf.data.DataTypeEnum.IMAGE, # Data type of the dataset, for example, image
    )
)

# Define the name parameters of the labeling job.
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # Name of the labeling phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Labeling", # Title, which defaults to the value of name
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,    # Labeling job type, for example, image classification
        task_name=task_name   # If the labeling job name does not exist, a job will be created using this name. If the labeling job name exists, the corresponding job will be used.
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=create_dataset.outputs["create_dataset_output"].as_input()), # LabelingStep inputs. The data source is the outputs of the dataset creation phase.
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep outputs
    depend_steps=create_dataset # Preceding dataset creation phase
)
# create_dataset is an instance of wf.steps.CreateDatasetStep. create_dataset_output is the name field value of wf.steps.CreateDatasetOutput.

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[create_dataset, labeling]
)
```

5.3.4.3 Creating a Dataset Import Phase

Description

This phase integrates capabilities of the ModelArts dataset module, allowing you to import data to datasets. The dataset import phase is used to import data from a specified path to a dataset or a labeling job. The application scenarios are as follows:

- This phase is used for continuous data update. You can import raw data or labeled data to a labeling job and label the data in the labeling phase.
- Some labeled raw data can be directly imported to a dataset or labeling job, and the dataset with version information can be obtained in the dataset release phase.

Parameter Overview

You can use DatasetImportStep to create a dataset import phase. The following is an example of defining a DatasetImportStep.

Table 5-29 DatasetImportStep

Parameter	Description	Mandatory	Data Type
name	Name of a dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
inputs	Inputs of the dataset import phase.	Yes	DatasetImportInput or DatasetImportInput list
outputs	Outputs of the dataset import phase.	Yes	DatasetImportOutput or DatasetImportOutput list
properties	Configurations for dataset import.	Yes	ImportDataInfo
title	Title for frontend display.	No	str
description	Description of the dataset import phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-30 DatasetImportInput

Parameter	Description	Mandatory	Data Type
name	Input name of the dataset import phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of the dataset import phase.	Yes	Dataset, OBS, or labeling job object. Currently, only Dataset, DatasetConsumption, DatasetPlaceholder, OBSPath, OBSConsumption, OBSPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, and DataConsumptionSelector are supported.

Table 5-31 DatasetImportOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the dataset import phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str

Table 5-32 ImportDataInfo

Parameter	Description	Mandatory	Data Type
annotation_format_config	Configurations of the imported labeling format.	No	AnnotationFormatConfig
excluded_labels	Samples with specified labels are not imported.	No	Label list

Parameter	Description	Mandatory	Data Type
import_annotated	Whether to import the labeled samples in the original dataset to the **To Be Confirmed** tab. The default value is **false**, indicating that the labeled samples in the original dataset are not imported to the **To Be Confirmed** tab. The options are as follows: • **true**: The labeled samples in the original dataset are imported to the **To Be Confirmed** tab. • **false**: The labeled samples in the original dataset are not imported to the **To Be Confirmed** tab.	No	bool
import_annotations	Whether to import labels. The options are as follows: • **true**: The labels are imported. (Default) • **false**: The labels are not imported.	No	bool

Parameter	Description	Mandatory	Data Type
import_samples	Whether to import samples. The options are as follows: • **true**: The samples are imported. (Default) • **false**: The samples are not imported.	No	bool
import_type	Import mode. The options are as follows: • **dir**: imported from an OBS path • **manifest**: imported from a manifest file	No	ImportTypeEnum
included_labels	Samples with specified labels are imported.	No	Label list
label_format	Label format. This parameter is used only for text datasets.	No	LabelFormat

Table 5-33 AnnotationFormatConfig

Parameter	Description	Mandatory	Data Type
format_name	Name of a labeling format	No	AnnotationFormatEnum
parameters	Advanced parameters of the labeling format	No	AnnotationFormatParameters
scene	Labeling scenario, which is optional	No	LabelTaskTypeEnum

Table 5-34 AnnotationFormatParameters

Parameter	Description	Mandatory	Data Type
difficult_only	Whether to import only hard examples. The options are as follows: • **true**: Only hard examples are imported. • **false**: All the samples are imported. (Default)	No	bool
included_labels	Samples with specified labels are imported.	No	Label list
label_separator	Separator between labels. By default, the comma (,) is used as the separator. The separator needs to be escaped. The separator can contain only one character, which must be a letter, a digit, or any of the following special characters: !@#$ %^&*_=|?/':.;,	No	str
sample_label_separator	Separator between the text and label. By default, the **Tab** key is used as the separator. The separator needs to be escaped. The separator can contain only one character, which must be a letter, a digit, or any of the following special characters: !@#$ %^&*_=|?/':.;,	No	str

Examples

There are three scenarios:

- Scenario 1: Updating a dataset by importing data from a specified path
  - You import labeled data (with label information) in a specified path to a dataset. Then, you can create a dataset release phase to release a version.

    Data preparation: Create a dataset on the ModelArts console and upload labeled data to OBS.

    ```
    from modelarts import workflow as wf
    # Use DatasetImportStep to import data in a specified path to a dataset and output the dataset.

    # Define a dataset.
    ```



```
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Define OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type must be file or directory.

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # Name of the dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Import", # Title, which defaults to the value of name
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # The target dataset is configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="dataset_name") for the data field.
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # Storage path to the imported dataset, configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="obs_path") for the data field.
    ],# DatasetImportStep inputs
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep outputs
    properties=wf.steps.ImportDataInfo(
        annotation_format_config=[
            wf.steps.AnnotationFormatConfig(
                format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # Labeling format of labeled data, for example, image classification
                scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # Labeling scene
            )
        ]
    )
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- You import unlabeled data in a specified path to a dataset. Then, you can add a labeling phase to label the imported data.

  Data preparation: Create a dataset on the ModelArts console and upload unlabeled data to OBS.

```
from modelarts import workflow as wf
# Use DatasetImportStep to import data in a specified path to a dataset and output the dataset.

# Define a dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Define OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type must be file or directory.

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # Name of the dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Import", # Title, which defaults to the value of name
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # The target dataset is configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="dataset_name") for the data field.
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # Storage path to the imported dataset, configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="obs_path") for the data field.
    ],# DatasetImportStep inputs
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep outputs
)

workflow = wf.Workflow(
```



```
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- Scenario 2: Updating a labeling job by importing data from a specified path
  - You import labeled data in a specified path to a labeling job. Then, you can create a dataset release phase to release a version.

    Data preparation: Create a labeling job using a specified dataset and upload the labeled data to OBS.

```
from modelarts import workflow as wf
# Use DatasetImportStep to import data in a specified path to a labeling job and output the labeling job.

# Define a labeling job.
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# Define the OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type must be file or directory.

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # Name of the dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Import", # Title, which defaults to the value of name
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # Labeling job object, configured when the workflow is running. You can also use wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name") for the data field.
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # Storage path to the imported dataset, configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="obs_path") for the data field.
    ],# DatasetImportStep inputs
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep outputs
    properties=wf.steps.ImportDataInfo(
        annotation_format_config=[
            wf.steps.AnnotationFormatConfig(
                format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # Labeling format of labeled data, for example, image classification
                scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # Labeling scene
            )
        ]
    )
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

  - You import unlabeled data in a specified path to a labeling job. Then, you can add a labeling phase to label the imported data.

    Data preparation: Create a labeling job using a specified dataset and upload the unlabeled data to OBS.

```
from modelarts import workflow as wf
# Use DatasetImportStep to import data in a specified path to a labeling job and output the labeling job.

# Define a labeling job.
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# Define the OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) #
```

```
object_type must be file or directory.

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # Name of the dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Import", # Title, which defaults to the value of name
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # Labeling job object, configured when the workflow is running. You can also use wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name") for the data field.
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # Storage path to the imported dataset, configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="obs_path") for the data field.
    ],# DatasetImportStep inputs
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep outputs
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- Scenario 3: Creating a dataset import phase using the outputs of the dataset creation phase.

```
from modelarts import workflow as wf
# Use DatasetImportStep to import data in a specified path to a dataset and output the dataset.

# Define the OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type must be file or directory.

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # Name of the dataset import phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Import", # Title, which defaults to the value of name
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=create_dataset.outputs["create_dataset_output"].as_input()), # The outputs of the dataset creation phase are used as the inputs of the dataset import phase.
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # Storage path to the imported dataset, configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="obs_path") for the data field.
    ],# DatasetImportStep inputs
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep outputs
    depend_steps=create_dataset # Preceding dataset creation phase
)
# create_dataset is an instance of wf.steps.CreateDatasetStep. create_dataset_output is the name field value of wf.steps.CreateDatasetOutput.

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

5.3.4.4 Creating a Dataset Release Phase

Description

This phase integrates capabilities of the ModelArts dataset module, enabling automatic dataset version release. The dataset release phase is used to release versions of existing datasets or labeling jobs. Each version is a data snapshot and

can be used for subsequent data source tracing. The application scenarios are as follows:

- After data labeling is completed, a dataset version can be automatically released and used as inputs in subsequent phases.
- When data update is required for model training, you can use the dataset import phase to import data and then use the dataset release phase to release a version for subsequent phases.

Parameter Overview

You can use ReleaseDatasetStep to create a dataset release phase. The following is an example of defining a ReleaseDatasetStep.

Table 5-35 ReleaseDatasetStep

Parameter	Description	Mandatory	Data Type
name	Name of a dataset release phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
inputs	Inputs of the dataset release phase.	Yes	ReleaseDatasetInput or ReleaseDatasetInput list
outputs	Outputs of the dataset release phase.	Yes	ReleaseDatasetOutput or ReleaseDatasetOutput list
title	Title for frontend display.	No	str
description	Description of the dataset release phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-36 ReleaseDatasetInput

Parameter	Description	Mandatory	Data Type
name	Input name of the dataset release phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of the dataset release phase.	Yes	Dataset or labeling job object. Currently, only Dataset, DatasetConsumption, DatasetPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, and DataConsumptionSelector are supported.

Table 5-37 ReleaseDatasetOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the dataset release phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str
dataset_version_config	Configurations for dataset version release.	Yes	DatasetVersionConfig

Table 4 **DatasetVersionConfig**

Parameter	Description	Mandatory	Data Type
version_name	Dataset version name. By default, the dataset version is named in ascending order of V001 and V002.	No	str or Placeholder
version_format	Version format, which defaults to **Default**. You can also set it to **CarbonData**.	No	str
train_evaluate_sample_ratio	Ratio between the training set and validation set, which defaults to **1.00**. The value ranges from 0 to 1.00. For example, **0.8** indicates the ratio for the training set is 80%, and that for the validation set is 20%.	No	str or Placeholder
clear_hard_property	Whether to clear hard examples. The default value is **True**.	No	bool or Placeholder
remove_sample_usage	Whether to clear existing usage information of a dataset. The default value is **True**.	No	bool or Placeholder

Parameter	Description	Mandatory	Data Type
label_task_type	Type of a labeling job. If the input is a dataset, this field is mandatory and is used to specify the labeling scenario of the dataset version. If the input is a labeling job, this field does not need to be configured.	No	LabelTaskTypeEnum The following types are supported: • IMAGE_CLASSIFICATION • OBJECT_DETECTION = 1 • IMAGE_SEGMENTATION • TEXT_CLASSIFICATION • NAMED_ENTITY_RECOGNITION • TEXT_TRIPLE • AUDIO_CLASSIFICATION • SPEECH_CONTENT and SPEECH_SEGMENTATION • TABLE • VIDEO_ANNOTATION
description	Version description.	No	str

NOTE

If there is no special requirement, use the default values.

Examples

Scenario 1: Releasing a dataset version

Scenario: When data in a dataset is updated, this phase can be used to release a dataset version for subsequent phases to use.

```
from modelarts import workflow as wf
# Use ReleaseDatasetStep to release a version of the input dataset and output the dataset with version information.

# Define a dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Define the split ratio between the training set and validation set
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
```



```
    name="release_dataset", # Name of the dataset release phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Version Release", # Title, which defaults to the value of name
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=dataset), # ReleaseDatasetStep inputs. The dataset object is configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="dataset_name") for the data field.
    outputs=wf.steps.ReleaseDatasetOutput(
        name="output_name",
        dataset_version_config=wf.data.DatasetVersionConfig(
            label_task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # Labeling job type for dataset version release
            train_evaluate_sample_ratio=train_ration # Split ratio between the training set and validation set
            )
    ) # ReleaseDatasetStep outputs
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

Scenario 2: Releasing a labeling job version

When data or labeling information of a labeling job is updated, this phase can be used to release a dataset version for subsequent phases to use.

```
from modelarts import workflow as wf
# Use ReleaseDatasetStep to release a version of the input labeling job and output the dataset with version information.

# Define a labeling job.
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# Define the split ratio between the training set and validation set
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # Name of the dataset release phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Version Release", # Title, which defaults to the value of name
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=label_task), # ReleaseDatasetStep inputs The labeling job object is configured when the workflow is running. You can also use wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name") for the data field.
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # Split ratio between the training set and validation set
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

Scenario 3: Creating a dataset release phase based on the labeling phase

Scenario: The outputs of the labeling phase are used as the inputs of the dataset release phase.

```
from modelarts import workflow as wf
# Use ReleaseDatasetStep to release a version of the input labeling job and output the dataset with version information.
```



```
# Define the split ratio between the training set and validation set
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # Name of the dataset release phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Version Release", # Title, which defaults to the value of name
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=labeling_step.outputs["output_name"].as_input()), # ReleaseDatasetStep inputs
The labeling job object is configured when the workflow is running. You can also use wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name") for the data field.
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # Split ratio between the training set and validation set
    depend_steps = [labeling_step] # Preceding labeling phase
)
# labeling_step is an instance object of wf.steps.LabelingStep and output_name is the value of the name field of wf.steps.LabelingOutput.

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

5.3.4.5 Creating a Training Job Phase

Description

This phase defines the algorithm, input, and output of a job for data processing, model training, or model evaluation. The application scenarios are as follows:

- Data preprocessing such as image enhancement and noise reduction
- Model training for object detection and image classification

Parameter Overview

You can use JobStep to create a job phase. The following is an example of defining a JobStep.

Table 5-38 JobStep

Parameter	Description	Mandatory	Data Type
name	Name of a job phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
algorithm	Algorithm object.	Yes	BaseAlgorithm, Algorithm, AIGalleryAlgorithm
spec	Job specifications.	Yes	JobSpec
inputs	Inputs of a job phase.	Yes	JobInput or JobInput list
outputs	Outputs of a job phase.	Yes	JobOutput or JobOutput list
title	Title for frontend display.	No	str
description	Description of a job phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-39 JobInput

Parameter	Description	Mandatory	Data Type
name	Input name of the job phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of a job phase.	Yes	Dataset or OBS object. Currently, only Dataset, DatasetPlaceholder, DatasetConsumption, OBSPath, OBSConsumption, OBSPlaceholder, and DataConsumptionSelector are supported.

Table 5-40 JobOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the job phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str

Parameter	Description	Mandatory	Data Type
obs_config	OBS output configuration.	No	OBSOutputConfig
model_config	Model output configuration.	No	ModelConfig
metrics_config	Metrics configuration.	No	MetricsConfig

Table 5-41 OBSOutputConfig

Parameter	Description	Mandatory	Data Type
obs_path	Existing OBS directory	Yes	str, Placeholder, Storage
metric_file	Name of the file that stores metric information	No	str, Placeholder

Table 5-42 BaseAlgorithm

Parameter	Description	Mandatory	Data Type
id	Algorithm ID	No	str
subscription_id	Subscription ID of the subscribed algorithm	No	str
item_version_id	Version ID of the subscribed algorithm	No	str
code_dir	Code directory	No	str, Placeholder, Storage
boot_file	Boot file	No	str, Placeholder, Storage
command	Boot command	No	str, Placeholder
parameters	Algorithm hyperparameters	No	AlgorithmParameters list
engine	Information about the image used by the job	No	JobEngine
environments	Environment variables	No	dict

Table 5-43 Algorithm

Parameter	Description	Mandatory	Data Type
algorithm_id	Algorithm ID	Yes	str
parameters	Algorithm hyperparameters	No	List of algorithm parameters

Table 5-44 AIGalleryAlgorithm

Parameter	Description	Mandatory	Data Type
subscription_id	Subscription ID of the subscribed algorithm	Yes	str
item_version_id	Version ID of the subscribed algorithm	Yes	str
parameters	Algorithm hyperparameters	No	List of algorithm parameters

Table 5-45 AlgorithmParameters

Parameter	Description	Mandatory	Data Type
name	Name of an algorithm hyperparameter	Yes	str
value	Value of an algorithm hyperparameter	Yes	int, bool, float, str, Placeholder, Storage

Table 5-46 JobEngine

Parameter	Description	Mandatory	Data Type
engine_id	Image ID	No	str, Placeholder
engine_name	Image name	No	str, Placeholder
engine_version	Image version	No	str, Placeholder
image_url	Image URL	No	str, Placeholder

Table 5-47 JobSpec

Parameter	Description	Mandatory	Data Type
resource	Resource information	Yes	JobResource
log_export_path	Log output path	No	LogExportPath
schedule_policy	Job scheduling policy	No	SchedulePolicy
volumes	Information about the file system mounted to the job	No	list[Volume]

Table 5-48 JobResource

Parameter	Description	Mandatory	Data Type
flavor	Resource flavor.	Yes	Placeholder
node_count	Number of nodes. The default value is **1**. If there are multiple nodes, distributed training is supported.	No	int, Placeholder

Table 5-49 SchedulePolicy

Parameter	Description	Mandatory	Data Type
priority	Job scheduling priority. The value can only be **1**, **2**, or **3**, indicating low, medium, and high priorities, respectively.	Yes	int, Placeholder

Table 5-50 Volume

Parameter	Description	Mandatory	Data Type
nfs	NFS file system object. In a volume object, only one of **nfs**, **pacific**, and **pfs** can be configured.	No	NFS
pacific	Pacific file system object. In a volume object, only one of **nfs**, **pacific**, and **pfs** can be configured.	No	Placeholder
pfs	OBS parallel file system object. In a volume object, only one of **nfs**, **pacific**, and **pfs** can be configured.	No	PFS, Placeholder

Table 5-51 NFS

Parameter	Description	Mandatory	Data Type
nfs_server_path	Service address of the NFS file system.	Yes	str, Placeholder
local_path	Path mounted to the container.	Yes	str, Placeholder
read_only	Indicates if the mount mode is set to read-only.	No	bool, Placeholder

Table 5-52 PFS

Parameter	Description	Mandatory	Data Type
pfs_path	Path of the parallel file system	Yes	str, Placeholder
local_path	Path mounted to the container	Yes	str, Placeholder

Obtaining Resource Flavors

Before creating a job phase, perform the following operations to obtain supported training flavors and engines:

- Import packages.
```
from modelarts.session import Session
from modelarts.estimatorV2 import TrainingJob
from modelarts.workflow.client.job_client import JobClient
```
- Initialize a session.
```
# If you develop a workflow in a local IDEA, initialize a session as follows:
# 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/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# Decrypt the information if it is encrypted.
session = Session(
    access_key=__AK, # AK information of your account
    secret_key=__SK, # SK information of your account
    region_name="***", # Region to which your account belongs
    project_id="***" # Project ID of your account
)

# If you develop a workflow in a notebook environment, initialize a session:
session = Session()
```
- Obtain public resource pools.
```
# Obtain the specification list of public resource pools.
spec_list = TrainingJob(session).get_train_instance_types(session) # A list is returned. You can download it.
print(spec_list)
```
- Obtain dedicated resource pools.
```
# Obtain the list of running dedicated resource pools.
pool_list = JobClient(session).get_pool_list() # A list of dedicated resource pools is returned.
pool_id_list = JobClient(session).get_pool_id_list() # An ID list of dedicated resource pools is returned.
The following lists the flavor IDs of dedicated resource pools. Select one as required.
    modelarts.pool.visual.xlarge (1 card)
    modelarts.pool.visual.2xlarge (2 cards)
    modelarts.pool.visual.4xlarge (4 cards)
    modelarts.pool.visual.8xlarge (8 cards)
```
- Obtain engine types.
```
# Obtain engine types.
engine_dict = TrainingJob(session).get_engine_list(session) # A dictionary is returned. You can download it.
print(engine_dict)
```

Examples

There are seven scenarios:

- Using an algorithm subscribed to in AI Gallery
- Using an algorithm in Algorithm Management
- Using a custom algorithm (code directory+boot file+official image)
- Using a custom algorithm (code directory+boot command+official image)
- Creating a job phase based on the dataset release phase
- Job phase with visualization
- Using the DataSelector object as the input, which supports OBS or datasets

Using an Algorithm Subscribed from AI Gallery

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # Algorithm subscription ID. You can also enter the version number.
        item_version_id="item_version_id", # Algorithm version ID. You can also enter the version number instead.
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool, float, or string.
        ]
    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name") for the data field.
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    )# Training flavors
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

Using an Algorithm in Algorithm Management

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
```



```
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.Algorithm(
        algorithm_id="algorithm_id", # Algorithm ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool,
float, or string.
        ]
    ), # Algorithm used for training. An algorithm from Algorithm Management is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the
hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when the
workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name",
version_name="fake_version_name") for the data field.
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
    )# Training flavors
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

Using a Custom Algorithm (Code Directory + Boot File + Official Image)

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters,
including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # Code directory
        boot_file="fake_boot_file", # Boot file path, which must be in the code directory
        engine=wf.steps.JobEngine(engine_name="fake_engine_name",
engine_version="fake_engine_version"), # Name and version of the official image

        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool,
float, or string.
        ]
    ), # The custom algorithm is implemented using the code directory, boot file, and official image.
```



```
    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name") for the data field.
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    )# Training flavors
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

Using a Custom Algorithm (Code Directory + Boot Command + Custom Image)

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # Code directory
        command="fake_command", # Boot command
        engine=wf.steps.JobEngine(image_url="fake_image_url"), # Custom image URL, in the format of Organization name/Image name:Version name. Do not contain the domain name; If image_url is required to be configurable in the running state, use the following: image_url=wf.Placeholder(name="image_url", placeholder_type=wf.PlaceholderType.STR, placeholder_format="swr", description="Custom image")
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool, float, or string.
        ]
    ), The custom algorithm is implemented using the code directory, boot command, and custom image.

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name") for the data field.
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    )# Training flavors
)
```



```
workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

NOTE

The preceding four methods use a dataset as the input. If you want to use an OBS path as the input, set **data** of **JobInput** to **data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")** or **data=wf.data.OBSPath(obs_path="fake_obs_path")**.

In addition, you can specify a dataset or OBS path when creating a workflow to reduce configuration operations and facilitate debugging in the development state. You are advised to use placeholders to create a workflow you want to publish to the running state or AI Gallery. In this case, you can configure parameters before workflow execution.

Creating a Job Phase Based on the Dataset Release Phase

Scenario: The output of the dataset release phase is used as the input of the job phase.

```
from modelarts import workflow as wf

# Define the dataset object.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Define the split ratio between the training set and validation set
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # Name of the dataset release phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Dataset Version Release", # Title, which defaults to the value of name
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=dataset), # ReleaseDatasetStep inputs. The dataset object is configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="dataset_name") for the data field.
    outputs=wf.steps.ReleaseDatasetOutput(
        name="output_name",
        dataset_version_config=wf.data.DatasetVersionConfig(
            label_task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # Labeling job type for dataset version release
            train_evaluate_sample_ratio=train_ration # Split ratio between the training set and validation set
            )
    ) # ReleaseDatasetStep outputs
)

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id", # Version ID of the subscribed algorithm
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
```



```
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool,
float, or string.
        ]
    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the
value of an algorithm hyperparameter does not need to be changed, you do not need to configure the
hyperparameter in parameters. Hyperparameter values will be automatically filled.


    inputs=wf.steps.JobInput(name="data_url",
data=release_version_step.outputs["output_name"].as_input()), # The output of the dataset release phase is
used as the input of JobStep.
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
    ), # Training flavors
    depend_steps=release_version_step # Preceding dataset release phase
)
# release_version_step is an instance object of wf.steps.ReleaseDatasetStep and output_name is the
value of the name field of wf.steps.ReleaseDatasetOutput.

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[release_version_step, job_step],
    storages=[storage]
)
```

Job Phase With Visualization

Phase visualization enables you to view the metrics generated by your workflows in real time. You can also display the external disks of each phase separately. To use phase visualization, you need to add and configure an output for showing metrics through the MetricsConfig object, based on the original job phase.

Table 5-53 MetricsConfig

Parameter	Description	Mandatory	Data Type
metric_files	Metric files. Supported element types: str, Placeholder, and Storage.	Yes	list
realtime_visualization	Whether to display the output metrics in real time. The default value is **False**.	No	bool
visualization	Whether to display visualization phases separately. The default value is **True**.	No	bool

The output metrics file must contain standard JSON data with a maximum size of 1 MB. The data formats must match the supported ones.

- Key-value pair data

```
[
  {
    "key": "loss",
    "title": "loss",
    "type": "float",
    "data": {
      "value": 1.2
    }
  },
  {
    "key": "accuracy",
    "title": "accuracy",
    "type": "float",
    "data": {
      "value": 1.6
    }
  }
]
```

- Line chart data

```
[
  {
    "key": "metric",
    "title": "metric",
    "type": "line chart",
    "data": {
      "x_axis": [
        {
          "title": "step/epoch",
          "value": [
            1,
            2,
            3
          ]
        }
      ],
      "y_axis": [
        {
          "title": "value",
          "value": [
            0.5,
            0.4,
            0.3
          ]
        }
      ]
    }
  }
]
```

- Histogram data

```
[
  {
    "key": "metric",
    "title": "metric",
    "type": "histogram",
    "data": {
      "x_axis": [
        {
          "title": "step/epoch",
          "value": [
            1,
            2,
```



```
                3
            ]
        }
    ],
    "y_axis": [
        {
            "title": "value",
            "value": [
                0.5,
                0.4,
                0.3
            ]
        }
    ]
  }
 }
]
```

- Confusion matrix

```
[
  {
    "key": "confusion_matrix",
    "title": "confusion_matrix",
    "type": "table",
    "data": {
      "cell_value": [
        [
          1,
          2
        ],
        [
          2,
          3
        ]
      ],
      "col_labels": {
        "title": "labels",
        "value": [
          "daisy",
          "dandelion"
        ]
      },
      "row_labels": {
        "title": "predictions",
        "value": [
          "daisy",
          "dandelion"
        ]
      }
    }
  }
]
```

- One-dimensional table

```
[
  {
    "key": "Application Evaluation Results",
    "title": "Application Evaluation Results",
    "type": "one-dimensional-table",
    "data": {
      "cell_value": [
        [
          10,
          2,
          0.5
        ]
      ],
      "labels": [
        "samples",
```

```
            "maxResTine",
            "p99"
        ]
    }
  }
]
```

Example:

```
from modelarts import workflow as wf

# Create a Storage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=True, create_dir=True) # Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id", # Algorithm version ID. You can also enter the version
number instead.
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer,
bool, float, or string.
        ]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to
configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.


    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when
the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name",
version_name="fake_version_name") for the data field.
    outputs=[
    wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),# JobStep outputs
        wf.steps.JobOutput(name="metrics_output",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # Metrics are output to the configured path by the job script.
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
    )# Training flavors
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```


NOTE

Workflow does not automatically retrieve the metrics produced by training. You need to extract the metrics from the algorithm code, create the **metrics.json** file in the required data format, and upload the file to the OBS path specified in MetricsConfig. Workflow only reads, renders, and displays the data.

Using the DataSelector Object as the Input, Which Supports OBS or Datasets

You can use this method when you can choose the input type. The DataSelector object allows you to select either a dataset object or an OBS object as the training input. Here is a code sample:

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Define the DataSelector object.
data_selector = wf.data.DataSelector(name="input_data", data_type_list=["dataset", "obs"])

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # Algorithm subscription ID. You can also enter the version number.
        item_version_id="item_version_id", # Algorithm version ID. You can also enter the version number instead.
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool, float, or string.
        ]
    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=data_selector), # JobStep inputs are configured when the workflow is running. You can choose OBS or datasets as the input.
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    )# Training flavors
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```


NOTE

When using DataSelector as the input, ensure that the algorithm input supports both datasets and OBS.

5.3.4.6 Creating a Model Registration Phase

Description

This phase integrates capabilities of ModelArts AI application management. This enables trained models to be registered in AI Application Management for service deployment and update. The application scenarios are as follows:

- Registering models trained from ModelArts training jobs
- Registering models from custom images

Parameter Overview

You can use ModelStep to create a model registration phase. The following is an example of defining a ModelStep.

Table 5-54 ModelStep

Parameter	Description	Mandatory	Data Type
name	Name of a model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
inputs	Inputs of the model registration phase.	No	ModelInput or ModelInput list
outputs	Outputs of the model registration phase.	Yes	ModelOutput or ModelOutput list
title	Title for frontend display.	No	str
description	Description of the model registration phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-55 ModelInput

Parameter	Description	Mandatory	Data Type
name	Input name of the model registration phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str
data	Input data object of the model registration phase.	Yes	OBS, SWR, or subscribed model object. Currently, only OBSPath, SWRImage, OBSConsumption, OBSPlaceholder, SWRImagePlaceholder, DataConsumptionSelector, and GalleryModel are supported.

Table 5-56 ModelOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the model registration phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str
model_config	Configurations for model registration.	Yes	ModelConfig

Table 5-57 ModelConfig

Parameter	Description	Mandatory	Data Type
model_type	Model type. Supported types: **TensorFlow**, **MXNet**, **Caffe**, **Spark_MLlib**, **Scikit_Learn**, **XGBoost**, **Image**, **PyTorch**, **Template**, and **Custom**. The default value is **TensorFlow**.	Yes	str
model_name	Model name. Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.	No	str, Placeholder
model_version	Model version in the format of *Digit.Digit.Digit*. The value range of the digits is [1, 99]. If this parameter is left blank, the version number automatically increases. **CAUTION** No part of the version number can start with 0. For example, **01.01.01** is not allowed.	No	str, Placeholder
runtime	Model runtime environment. The options of **runtime** are the same as those of **model_type**.	No	str, Placeholder
description	Model description that consists of 1 to 100 characters. The following special characters cannot be contained: &!'"<>=	No	str
execution_code	OBS path for storing the execution code. By default, this parameter is left blank. The name of the execution code file is fixed to **customize_service.py**. The inference code file must be stored in the **model** directory. This parameter is left blank. The system can automatically identify the inference code in the **model** directory.	No	str
dependencies	Package required for the inference code and model. By default, this parameter is left blank. It is read from the configuration file.	No	str
model_metrics	Model precision, which is read from the configuration file.	No	str

Parameter	Description	Mandatory	Data Type
apis	All **apis** input and output parameters of a model (optional), which are parsed from the configuration file.	No	str
initial_config	Model configuration information.	No	dict
template	Template configuration items. This parameter is mandatory when **model_type** is set to **Template**.	No	Template
dynamic_load_mode	Dynamic loading mode. Currently, only **Single** is supported.	No	str, Placeholder
prebuild	Whether the model is prebuilt. The default value is **False**.	No	bool, Placeholder
install_type	Model installation type. The value can be **real_time**, **edge**, **batch**. If this parameter is left blank, all types are supported by default.	No	list[str]

Table 5-58 Template

Parameter	Description	Mandatory	Data Type
template_id	ID of the used template. The template has a built-in input and output mode.	Yes	str, Placeholder
infer_format	Input and output mode. When this parameter is used, the input and output mode built in the template does not take effect.	No	str, Placeholder
template_inputs	Template input configuration, specifying the source path for configuring a model	Yes	list of TemplateInputs object

Table 5-59 TemplateInputs

Parameter	Description	Mandatory	Data Type
input_id	Input item ID, which is obtained from the template details.	Yes	str, Placeholder

Parameter	Description	Mandatory	Data Type
input	Template input path, which can be an OBS file path or OBS directory path. When you use a template with multiple input items to create a model, if the target paths **input_properties** specified in the template are the same, the OBS directory or OBS file name entered here must be unique to prevent files from being overwritten.	Yes	str, Placeholder, Storage

Examples

There are six scenarios:

- Registering models output by JobStep
- Registering a model using OBS data
- Registering a model using a template
- Registering a model using a custom image
- Registering a model using a custom image and OBS
- Registering a model using a subscribed model and OBS

Registering a Model from a Training Job (Model Source: JobStep Output)

```
import modelarts.workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # Only name is mandatory.

# Define an input dataset.
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# Use JobStep to define a training phase. Use a dataset as the input, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # Algorithm subscription ID. You can also enter the version number.
        item_version_id="item_version_id", # Algorithm version ID. You can also enter the version number instead.
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # Algorithm hyperparameters are represented using placeholders, which can be integer, bool, float, or string.
        ]
    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
```



```
    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep inputs are configured when the workflow is running. You can also use wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name") for the data field.
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep outputs
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    )# Training flavors
)

# Define a model registration phase using ModelStep. The output of JobStep is used as the input of ModelStep.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # The output of JobStep is used as the input of ModelStep.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep outputs
    depend_steps=job_step # Preceding job phase
)
# job_step is an instance object of wf.steps.JobStep and train_url is the value of the name field of wf.steps.JobOutput.

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[job_step, model_registration],
    storages=[storage]
)
```

Registering a Model from a Training Job (Model Source: A Trained Model Stored in OBS)

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from OBS.

# Define the OBS data.
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type must be file or directory.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=obs),  # ModelStep inputs are configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep outputs
)

workflow = wf.Workflow(
    name="model-step-demo",
```



```
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

Registering a Model Using a Template

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. Register a model using a preset template.

# Define a preset template object. Fields in the template object can be represented by placeholders.
template = wf.steps.Template(
    template_id="fake_template_id",
    infer_format="fake_infer_format",
    template_inputs=[
        wf.steps.TemplateInputs(
            input_id="fake_input_id",
            input="fake_input_file"
        )
    ]
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    outputs=wf.steps.ModelOutput(
          name='model_output',
          model_config=wf.steps.ModelConfig(
              model_name=model_name,
              model_type="Template",
              template=template
          )
    ) # ModelStep outputs
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

Registering a Model from a Custom Image

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from the URL of a custom image.

# Define the image data.
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name="input",data=swr), # ModelStep inputs are configured when the workflow is running. You can also use wf.data.SWRImage(swr_path="fake_path") for the data field.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep outputs
)

workflow = wf.Workflow(
```



```
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

Registering a Model Using a Custom Image and OBS

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from the URL of a custom image.

# Define the image data.
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# Define OBS model data.
model_obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # 
object_type must be file or directory.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 
64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and 
must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=[
        wf.steps.ModelInput(name="input",data=swr), # ModelStep inputs are configured when the workflow 
is running. You can also use wf.data.SWRImage(swr_path="fake_path") for the data field.
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep inputs are configured when the 
workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    ],
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    ) # ModelStep outputs
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo orkflow",
    steps=[model_registration]
)
```

Registering a Model Using a Subscribed Model and OBS

This mode is similar to the custom image + OBS mode, except that you obtain a custom image from a subscribed model.

Example:

```
import modelarts.workflow as wf

# Define the subscribed model object.
base_model = wf.data.GalleryModel(subscription_id="fake_subscription_id", version_num="fake_version") # 
Model subscribed to from AI Gallery, generally published by a developer

# Define OBS model data.
model_obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # 
object_type must be file or directory.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
```



```
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=[
        wf.steps.ModelInput(name="input",data=base_model) # Use a subscribed model as the ModelStep input.
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep inputs are configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    ],
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    ) # ModelStep outputs
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

In the preceding example, the system automatically obtains the custom image from the subscribed model and registers and generates a model based on the entered OBS model path. **model_obs** can be replaced with the dynamic output of JobStep.

NOTE

The value of **model_type** can be **TensorFlow**, **MXNet**, **Caffe**, **Spark_MLlib**, **Scikit_Learn**, **XGBoost**, **Image**, **PyTorch**, **Template**, or **Custom**.

If **model_type** is not set for **wf.steps.ModelConfig**, **TensorFlow** is used by default.

- If the model type of your workflow does not need to be changed, refer to the preceding examples.
- If the model type of your workflow needs to be changed in multiple executions, write the parameter using placeholders.

  model_type = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.ENUM, default="TensorFlow", enum_list=["TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch", "Template", "Custom"], description="Model type")

5.3.4.7 Creating a Service Deployment Phase

Description

This phase integrates capabilities of ModelArts service management to enable service deployment and update in a workflow. The application scenarios are as follows:

- Deploying a model as a web service
- Updating an existing service (gray update supported)

Parameter Overview

You can use ServiceStep to create a service deployment phase. The following is an example of defining a ServiceStep.

Table 5-60 ServiceStep

Parameter	Description	Mandatory	Data Type
name	Name of a service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
inputs	Inputs of the service deployment phase.	No	ServiceInput or ServiceInput list
outputs	Outputs of the service deployment phase.	Yes	ServiceOutput or ServiceOutput list
title	Title for frontend display.	No	str
description	Description of the service deployment phase.	No	str
policy	Phase execution policy.	No	StepPolicy
depend_steps	Dependent phases.	No	Step or step list

Table 5-61 ServiceInput

Parameter	Description	Mandatory	Data Type
name	Input name of the service deployment phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The input name of a step must be unique.	Yes	str

Parameter	Description	Mandatory	Data Type
data	Input data object of the service deployment phase.	Yes	Model list or service object. Currently, only ServiceInputPlaceholder, ServiceData, and ServiceUpdatePlaceholder are supported.

Table 5-62 ServiceOutput

Parameter	Description	Mandatory	Data Type
name	Output name of the service deployment phase. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter. The output name of a step must be unique.	Yes	str
service_config	Configurations for service deployment.	Yes	ServiceConfig

Table 4 ServiceConfig

Parameter	Description	Mandatory	Data Type
infer_type	Inference mode. The value can be **real-time**, **batch**, or **edge**. The default value is **real-time**. • **real-time**: real-time service. The model is deployed as a web service. • **batch**: batch service. A batch service can perform inference on batch data and automatically stops after data processing is completed. • **edge**: edge service. A model is deployed as a web service on an edge node through IEF. You must create an edge node on IEF beforehand.	Yes	str
service_name	Service name. Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed. **NOTE** If you do not specify this parameter, the default service name is generated automatically.	No	str, Placeholder
description	Service description, which contains a maximum of 100 characters. By default, this parameter is left blank.	No	str
vpc_id	ID of the VPC to which a real-time service instance is deployed. By default, this parameter is left blank. In this case, ModelArts allocates a dedicated VPC to each user, and users are isolated from each other. To access other service components in the VPC of the service instance, set this parameter to the ID of the corresponding VPC. Once a VPC is configured, it cannot be modified. If both **vpc_id** and **cluster_id** are configured, only the dedicated resource pool takes effect.	No	str

Parameter	Description	Mandatory	Data Type
subnet_network_id	ID of a subnet. By default, this parameter is left blank. This parameter is mandatory when **vpc_id** is configured. Enter the network ID displayed in the subnet details on the VPC management console. A subnet provides dedicated network resources that are isolated from other networks.	No	str
security_group_id	Security group. By default, this parameter is left blank. This parameter is mandatory when **vpc_id** is configured. A security group is a virtual firewall that provides secure network access control policies for service instances. A security group must contain at least one inbound rule to permit the requests whose protocol is TCP, source address is **0.0.0.0/0**, and port number is **8080**.	No	str
cluster_id	ID of a dedicated resource pool. By default, this parameter is left blank, indicating that no dedicated resource pool is used. When using a dedicated resource pool to deploy services, ensure that the cluster is running properly. After this parameter is configured, the network configuration of the cluster is used, and the **vpc_id** parameter does not take effect. If both this parameter and **cluster_id** in **real-time config** are configured, **cluster_id** in **real-time config** is preferentially used.	No	str
additional_properties	Additional configurations.	No	dict
apps	Whether to enable application authentication for service deployment. Multiple application names can be entered.	No	str, Placeholder, list
envs	Environment variables.	No	dict

Example:


```
example = ServiceConfig()
# This object is used in the output of the service deployment phase.
```

If there is no special requirement, use the default values.

Examples

There are three scenarios:

- Deploying a real-time service
- Modifying a real-time service
- Getting the inference address from the service deployment phase

Deploying a Real-Time Service

```
import modelarts.workflow as wf
# Use ServiceStep to define a service deployment phase and specify a model for service deployment.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

service_step = wf.steps.ServiceStep(
    name="service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Deploying a Service",  # Title
    inputs=wf.steps.ServiceInput(name="si_service_ph",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder1",
                                                    # Restrictions on the model name: Only the model name specified here can be used in the running state; use the same model name as **model_name** of the model registration phase.
                                                    model_name=model_name)),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)

workflow = wf.Workflow(
    name="service-step-demo",
    desc="this is a demo workflow",
    steps=[service_step]
)
```

Modifying a Real-Time Service

Scenario: When you use a new model version to update an existing service, ensure that the name of the new model version is the same as that of the deployed service.

```
import modelarts.workflow as wf
# Use ServiceStep to define a service deployment phase and specify a model for service update.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# Define a service object.
service = wf.data.ServiceUpdatePlaceholder(name="placeholder_name")

service_step  = wf.steps.ServiceStep(
    name="service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Service Update", # Title
    inputs=[wf.steps.ServiceInput(name="si2",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
                                                    # Restrictions on the model name: Only the model
```



```
name specified here can be used in the running state.
                                                    model_name=model_name)),
        wf.steps.ServiceInput(name="si_service_data", data=service) # ServiceStep inputs are configured
when the workflow is running. You can also use wf.data.ServiceData(service_id="fake_service") for the
data field.
    ], # ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)

workflow = wf.Workflow(
    name="service-step-demo",
    desc="this is a demo workflow",
    steps=[service_step]
)
```

Getting the Inference Address from the Service Deployment Phase

The service deployment phase supports the output of the inference address. You can use the **get_output_variable("access_address")** method to obtain the output and use it in subsequent phases.

- For services deployed in the public resource pool, you can use **access_address** to obtain the inference address registered on the public network from the output.
- For services deployed in a dedicated resource pool, you can get the internal inference address from the output using **cluster_inner_access_address**, in addition to the public inference address. The internal address can only be accessed by other inference services.

```
import modelarts.workflow as wf

# Define model name parameters.
sub_model_name = wf.Placeholder(name="si_placeholder1",
placeholder_type=wf.PlaceholderType.STR)

sub_service_step = wf.steps.ServiceStep(
    name="sub_service_step", # Name of the service deployment phase. The name contains a
maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must
start with a letter and must be unique in a workflow.
    title="Subservice",  # Title
    inputs=wf.steps.ServiceInput(
        name="si_service_ph",
        data=wf.data.ServiceInputPlaceholder(name="si_placeholder1", model_name=sub_model_name)
    ),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)


main_model_name = wf.Placeholder(name="si_placeholder2",
placeholder_type=wf.PlaceholderType.STR)

# Obtain the inference address output by the subservice and transfer the address to the main service
through envs.
main_service_config = wf.steps.ServiceConfig(
    infer_type="real-time",
    envs={"infer_address":
sub_service_step.outputs["service_output"].get_output_variable("access_address")} # Obtain the
inference address output by the subservice and transfer the address to the main service through envs.
)

main_service_step = wf.steps.ServiceStep(
    name="main_service_step", # Name of the service deployment phase. The name contains a
maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must
start with a letter and must be unique in a workflow.
    title="Main service",  # Title
    inputs=wf.steps.ServiceInput(
        name="si_service_ph",
```

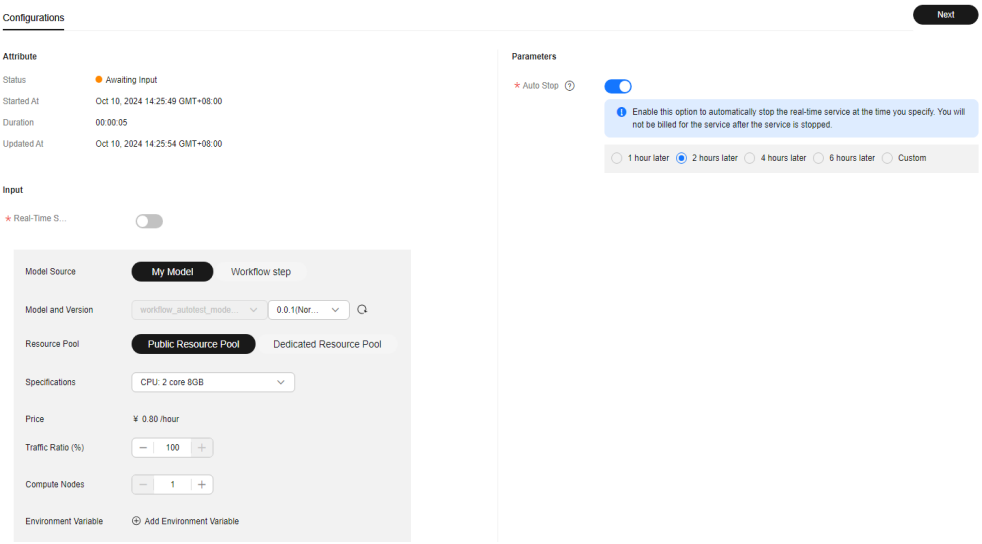
```
        data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
model_name=main_model_name)
    ),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output", service_config=main_service_config), #
ServiceStep outputs
    depend_steps=sub_service_step
)

workflow = wf.Workflow(
    name="service-step-demo",
    desc="this is a demo workflow",
    steps=[sub_service_step, main_service_step]
)
```

Configuring Information for Deploying a Synchronous Service

After the service deployment phase is started in the development state (usually a notebook instance), configure the information based on the following format in the logs.

```
Please enter the ServiceInputPlaceholder "service_model" in the following format:
"[{'model_name': '******', 'model_version': '*.*.*', 'specification':'[redacted]', 'weight': 50}, {'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 30,
'envs':{'k1': 'v1', 'k2': 'v2'}},{'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 20, 'envs':{'******': '******', '******': '******'}},]"
Note that:(1) The "[" at the beginning and "]" at the end are required.
          (2) The sum of the weights must be equal to 100.
          (3) All model must have the same model name. Two model versions cannot be the same.
```

1. On the ModelArts console, choose **Development Workspace** > **Workflow** from the navigation pane.
2. Configure the information after the service deployment phase is started. After the configuration, click **Next**.



Configuring Information for Deploying an Asynchronous Service

1. On the ModelArts console, choose **Workflow** from the navigation pane.
2. Configure the information after the service deployment phase is started. Select an asynchronous inference AI application and a version, and configure service startup parameters. After the configuration, click **Next**.

NOTE

After you select the required AI application and version, the system automatically matches the service startup parameters.

5.3.5 Creating a Multi-Branch Workflow

5.3.5.1 Multi-Branch Workflow

You can implement multi-branch in two ways. However, the condition phase is limited to two branches. **Configuring Phase Parameters to Control Branch Execution** allows you to replace the ConditionStep capability without adding new phases, offering more flexibility.

Creating a Condition Phase to Control Branch Execution is used for conditional branching in the execution of phases based on condition value comparison or metrics output by the preceding phase.

Configuring Phase Parameters to Control Branch Execution is used for complex scenarios that involve multiple branches. When each execution starts, the workflow decides which branches to run and which ones to skip based on the relevant configuration information. This way, only some branches are executed. This function has a similar use case as ConditionStep, but it is more powerful.

5.3.5.2 Creating a Condition Phase to Control Branch Execution

Description

This phase is used for conditional branching in the execution of phases based on condition value comparison or metrics output by the preceding phase. The application scenarios are as follows:

You need to determine the subsequent process based on different input values. If you need to determine whether to retrain or register a model based on the model precision output by the training phase, you can use this phase to control the process.

Parameter Overview

You can use ConditionStep to create a condition phase. The following is an example of defining a ConditionStep.

Table 5-63 ConditionStep

Parameter	Description	Mandatory	Data Type
name	Name of a condition phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str

Parameter	Description	Mandatory	Data Type
conditions	List of conditions. The AND operation is used for multiple conditions.	Yes	Condition or condition list
if_then_steps	Steps to be executed if the calculation result of the condition expression is **True**.	No	str or str list
else_then_steps	Steps to be executed if the calculation result of the condition expression is **False**.	No	str or str list
title	Title for frontend-phase display.	No	str
description	Description of a condition phase.	No	str
depend_steps	Dependent phases.	No	Step or step list

Table 5-64 Condition

Parameter	Description	Mandatory	Data Type
condition_type	Condition type. The "==", ">", ">=", "in", "<", "<=", "!=", and "or" operators are supported.	Yes	ConditionTypeEnum
left	Left value of a condition expression.	Yes	int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo
right	Right value of a condition expression	Yes	int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo

Table 5-65 MetricInfo

Parameter	Description	Mandatory	Data Type
input_data	Metric input. Currently, only the output of JobStep is supported.	Yes	JobStep output
json_key	Key value corresponding to the metric information to be obtained	Yes	str

Description of the structure:

- **Condition object, which consists of the condition type, left value, and right value**
  - The condition type is obtained from ConditionTypeEnum. The "==", ">", ">=", "in", "<", "<=", "!=", and "or" operators are supported. The following table describes the mapping.

Enumeration	Operator
ConditionTypeEnum.EQ	==
ConditionTypeEnum.GT	>
ConditionTypeEnum.GTE	>=
ConditionTypeEnum.IN	in
ConditionTypeEnum.LT	<
ConditionTypeEnum.LTE	<=
ConditionTypeEnum.NOT	!=
ConditionTypeEnum.OR	or

  - The left and right values support the following types: integer, float, string, bool, placeholder, sequence, condition, and MetricInfo.
  - A condition phase supports a list of condition objects. The && operation is performed between multiple conditions.
- **if_then_steps and else_then_steps**
  - if_then_steps indicates a list of phases that are ready for execution if conditions evaluate to **true**. In this case, steps in else_then_steps are skipped.
  - else_then_steps indicates a list of phases that are ready for execution if conditions evaluate to **false**. In this case, steps in if_then_steps are skipped.

Examples

Refer to simple or advanced examples as needed.

Simple Examples

- Implemented using parameter configurations

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL,
default=True)

# Condition object
condition = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value,
right=True) # Condition object, including the type, left value, and right value.

# Condition phase
condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    conditions=condition, # Condition objects. The relationship between the conditions is &&.
    if_then_steps="job_step_1", # If conditions evaluate to true, job_step_1 is ready for execution, and
job_step_2 is skipped.
    else_then_steps="job_step_2" # If conditions evaluate to false, job_step_2 is ready for execution,
and job_step_1 is skipped.
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
model_step_1 = wf.steps.ModelStep(
    name="model_step_1",
    depend_steps=job_step_1
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
model_step_2 = wf.steps.ModelStep(
    name="model_step_2",
    depend_steps=job_step_2
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[condition_step, job_step_1, job_step_2, model_step_1, model_step_2]
)
```

  NOTE

  Scenario description: **job_step_1** and **job_step_2** indicate two training phases that depend on **condition_step**. condition_step parameters determine the subsequent phase execution.

  Execution analysis:

  – If the default value of **left_value** is **True**, the calculation result of the condition logical expression is **True**. Then, **job_step_1** is executed,

job_step_2 is skipped, and all phases contained in the branches that use **job_step_2** as the unique root node are skipped. That is, **model_step_2** is skipped. Therefore, **condition_step**, **job_step_1**, and **model_step_1** are executed.

- If **left_value** is set to **False**, the calculation result of the condition logical expression is **False**. Then, **job_step_2** is executed, **job_step_1** is skipped, and all phases contained in the branches that use **job_step_1** as the unique root node are skipped. That is, **model_step_1** is skipped, and **condition_step**, **job_step_2**, and **model_step_2** are executed.

- Implemented by obtaining the metric information output by JobStep

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info")  # The name field is mandatory, and the title and
description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version
number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to
configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("dir
ectory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # Metric output path. Metric information is automatically output by the job script
based on the specified data format. (In the example, the metric information needs to be output to the
metrics.json file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")
        )
     )  # Training flavors
)

# Define a condition object.
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    conditions=condition_lt, # Condition objects. The relationship between the conditions is &&.
```



```
    if_then_steps="training_job_retrain", # If conditions evaluate to true, training_job_retrain is ready for execution, and model_registration is skipped.
    else_then_steps="model_registration", # If conditions evaluate to false, model_registration is ready for execution, and training_job_retrain is skipped.
    depend_steps=job_step
)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_retrain = wf.steps.JobStep(
    name="training_job_retrain",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification retraining",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_retrain"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path_retrain/metrics.json", create_dir=False))) # Metric output path. Metric information is automatically output by the job script based on the specified data format. (In the example, the metric information needs to be output to the metrics.json file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor_retrain",
placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")
        )
     ),  # Training flavors
    depend_steps=condition_step
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # job_step output is used as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep outputs
    depend_steps=condition_step,
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[job_step, condition_step, job_step_retrain, model_step],
    storages=storage
)
```

In this example, ConditionStep obtains the accuracy output by **job_step** and compares it with the preset value to determine whether to retrain or register the model. When the accuracy output by **job_step** is less than the threshold 0.5, the calculation result of **condition_lt** is **True**. In this case,

job_step_retrain runs and **model_step** skips. Otherwise, **job_step_retrain** skips and **model_step** runs.

NOTE

For details about the format requirements of the metric file generated by **job_step**, see **Creating a Training Job Phase**. In the condition phase, only the metric data whose type is float can be used as the input.

The following is an example of the **metrics.json** file:

```
[
  {
    "key": "loss",
    "title": "loss",
    "type": "float",
    "data": {
      "value": 1.2
    }
  },
  {
    "key": "accuracy",
    "title": "accuracy",
    "type": "float",
    "data": {
      "value": 0.8
    }
  }
]
```

Advanced Examples

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL, default=True)
condition1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value, right=True)

internal_condition_1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=10, right=9)
internal_condition_2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.LT, left=10, right=9)

# The result of condition2 is internal_condition_1 || internal_condition_2.
condition2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.OR, left=internal_condition_1,
right=internal_condition_2)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and
must be unique in a workflow.
    conditions=[condition1, condition2], # Condition objects. The relationship between the conditions is &&.
    if_then_steps=["job_step_1"], # If conditions evaluate to true, job_step_1 is ready for execution, and
job_step_2 is skipped.
    else_then_steps=["job_step_2"] # If conditions evaluate to false, job_step_2 is ready for execution, and
job_step_1 is skipped.
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

workflow = wf.Workflow(
    name="condition-demo",
```



```
    desc="this is a demo workflow",
    steps=[condition_step, job_step_1, job_step_2],
)
```

ConditionStep supports nested condition phases. You can flexibly design tit based on different scenarios.

NOTE

The condition phase can only support two branches, which is very limiting. You can use the new branch function to replace the ConditionStep capability without creating new phases. For details, see **Configuring Phase Parameters to Control Branch Execution**.

5.3.5.3 Configuring Phase Parameters to Control Branch Execution

Function

You can use parameters or metrics from training output to decide whether to run a phase. This way, you can control the process.

Application Scenarios

This function is used for complex scenarios that involve multiple branches. When each execution starts, the workflow decides which branches to run and which ones to skip based on the relevant configuration information. This way, only some branches are executed. This function has a similar use case as ConditionStep, but it is more powerful. This function applies to the dataset creation phase, labeling phase, dataset import phase, dataset release phase, job phase, model registration phase, and service deployment phase.

Controlling the Execution of a Single Phase

- Implemented using parameter configurations

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
                                description="description_info")  # Only name is mandatory.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version
number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to
```



```
configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep input is configured when the workflow is running. You can also use
data=wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep output
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
    ), # Training flavors
    policy=wf.steps.StepPolicy(
        skip_conditions=[condition_equal] # Determines whether to skip job_step based on the
calculation result of skip_conditions.
    )
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=storage
)
```

In this example, **job_step** has a skip policy that is controlled by a bool parameter. If the placeholder parameter named **is_skip** is set to **True**, then **job_step** is skipped when **condition_equal** evaluates to **True**. Otherwise, **job_step** is run. For more details about the condition object, see **Creating a Condition Phase to Control Branch Execution**.

- Implemented by obtaining the metric information output by JobStep

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info")  # The name field is mandatory, and the title and
description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version
number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to
configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("dir
ectory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # Metric output path. Metric information is automatically output by the job script
based on the specified data format. (In the example, the metric information needs to be output to the
```



```
**metrics.json** file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")
        )
    )  # Training flavors
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# Define a condition object.
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a
maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must
start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  #
job_step output is used as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  #
ModelStep outputs
    depend_steps=job_step # Preceding job phase
    policy=wf.steps.StepPolicy(skip_conditions=condition_lt) # Determines whether to skip model_step
based on the calculation result of skip_conditions.
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step, model_step],
    storages=storage
)
```

In this example, **model_step** has a skip policy. The model registration depends on whether the accuracy output by **job_step** meets the preset value. When the accuracy output by **job_step** is less than the threshold 0.5, the calculation result of **condition_lt** is **True**. In this case, **model_step** skips. Otherwise, **model_step** runs.

NOTE

For details about the format requirements of the metric file generated by **job_step**, see **Creating a Training Job Phase**. In the condition phase, only the metric data whose type is float can be used as the input.

The following is an example of the **metrics.json** file:

```
[
    {
        "key": "loss",
        "title": "loss",
        "type": "float",
        "data": {
            "value": 1.2
        }
    },
    {
        "key": "accuracy",
        "title": "accuracy",
        "type": "float",
```



```
        "data": {
            "value": 0.8
        }
    }
]
```

Controlling Partial Execution of Multiple Branches

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True, create_dir=True, description="description_info")  # The name field is mandatory, and the title and description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

condition_equal_a = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=wf.Placeholder(name="job_step_a_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_a = wf.steps.JobStep(
    name="training_job_a",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_a")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
     ),  # Training flavors
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_a)
)

condition_equal_b = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=wf.Placeholder(name="job_step_b_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_b = wf.steps.JobStep(
    name="training_job_b",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url",
```

```
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_b")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
    ),  # Training flavors
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_b)
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of
64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and
must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_a.outputs["train_url"].as_input(),
job_step_b.outputs["train_url"].as_input()])),  # Select the output of job_step_a or job_step_b as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep
outputs
    depend_steps=[job_step_a, job_step_b],  # Preceding job phase
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step_a, job_step_b, model_step],
    storages=storage
)
```

In this example, both **job_step_a** and **job_step_b** have a skip policy that is controlled by parameters. When the parameter values are different, the execution of **model_step** can be divided into the following cases (**model_step** has no skip policy configured, so it follows the default rule).

job_step_a_is_skip	job_step_b_is_skip	Whether to Execute model_step
True	True	No
	False	Yes
False	True	Yes
	False	Yes

CAUTION

Default rule: A phase is automatically skipped if all the phases it depends on are skipped. Otherwise, the phase is run. This logic can apply to any phase.

Based on the previous example, if you want to override the default rule and make **model_step** run when **job_step_a** and **job_step_b** are skipped, you only need to configure a skip policy in **model_step**. The skip policy takes precedence over the default rule.

5.3.5.4 Configuring Multi-Branch Phase Data

Function

This function is only for the scenario where multiple branches are run. When you create a workflow phase, the data input source of the phase is uncertain. The data input source could be the output of any of the phases it depends on. Only after all dependency phases are run, the valid output is automatically selected as the input based on the actual execution situation.

Examples

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_true", placeholder_type=wf.PlaceholderType.BOOL), right=True)
condition_step = wf.steps.ConditionStep(
    name="condition_step",
    conditions=[condition_equal],
    if_then_steps=["training_job_1"],
    else_then_steps=["training_job_2"],
)

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
                                description="description_info")  # Only name is mandatory.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_1 = wf.steps.JobStep(
    name="training_job_1",  # Name of a training phase. The name contains a maximum of 64 characters,
including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number
instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the
value of an algorithm hyperparameter does not need to be changed, you do not need to configure the
hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep input is configured when the workflow is running. You can also use
data=wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep output
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
     ),  # Training flavors
    depend_steps=[condition_step]
)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_2 = wf.steps.JobStep(
    name="training_job_2",  # Name of a training phase. The name contains a maximum of 64 characters,
including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
```



```
unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number
instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the
value of an algorithm hyperparameter does not need to be changed, you do not need to configure the
hyperparameter in parameters. Hyperparameter values will be automatically filled.

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep input is configured when the workflow is running. You can also use
data=wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep output
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")

        )
     ),  # Training flavors
    depend_steps=[condition_step]
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of
64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and
must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_1.outputs["train_url"].as_input(),
job_step_2.outputs["train_url"].as_input()])),  # Select the output of job_step_1 or job_step_2 as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep
outputs
    depend_steps=[job_step_1, job_step_2] # Preceding job phase
)# job_step is an instance object of wf.steps.JobStep and train_url is the value of the name field of
wf.steps.JobOutput.


workflow = wf.Workflow(name="data-select-demo",
                       desc="this is a test workflow",
                       steps=[condition_step, job_step_1, job_step_2, model_step],
                       storages=storage
                       )
```

NOTE

The workflow in this example has two parallel branches, but only one branch runs at a time, depending on the configuration of **condition_step**. The input source of **model_step** is either **job_step_1** or **job_step_2**'s output. If **job_step_1** runs and **job_step_2** is skipped, **model_step** uses **job_step_1**'s output as input, and vice versa.

5.3.6 Creating a Workflow

To create a workflow, define each phase by referring to **Creating Workflow Phases**. Follow these steps:

1. Sort out scenarios, understand preset steps' functions, and determine the DAG structure.

2. Debug single-phase functions like training or inference on ModelArts.
3. Choose the code template that matches the phase function and fill in the details.
4. Arrange phases according to the DAG structure to create a workflow.

Importing the Workflow Data Package

When creating a workflow, required objects are imported through workflow packages. The details are as follows:

```
from modelarts import workflow as wf
```

Import the data package:

```
wf.data.DatasetTypeEnum
wf.data.Dataset
wf.data.DatasetVersionConfig
wf.data.DatasetPlaceholder
wf.data.ServiceInputPlaceholder
wf.data.ServiceData
wf.data.ServiceUpdatePlaceholder
wf.data.DataTypeEnum
wf.data.ModelData
wf.data.GalleryModel
wf.data.OBSPath
wf.data.OBSOutputConfig
wf.data.OBSPlaceholder
wf.data.SWRImage
wf.data.SWRImagePlaceholder
wf.data.Storage
wf.data.InputStorage
wf.data.OutputStorage
wf.data.LabelTask
wf.data.LabelTaskPlaceholder
wf.data.LabelTaskConfig
wf.data.LabelTaskTypeEnum
wf.data.MetricsConfig
wf.data.TripartiteServiceConfig
wf.data.DataConsumptionSelector
```

Import the policy package:

```
wf.policy.Policy
wf.policy.Scene
```

Import the steps package:

```
wf.steps.MetricInfo
wf.steps.Condition
wf.steps.ConditionTypeEnum
wf.steps.ConditionStep
wf.steps.LabelingStep
wf.steps.LabelingInput
wf.steps.LabelingOutput
wf.steps.LabelTaskProperties
wf.steps.ImportDataInfo
wf.steps.DataOriginTypeEnum
wf.steps.DatasetImportStep
wf.steps.DatasetImportInput
wf.steps.DatasetImportOutput
wf.steps.AnnotationFormatConfig
wf.steps.AnnotationFormatParameters
wf.steps.AnnotationFormatEnum
wf.steps.Label
wf.steps.ImportTypeEnum
wf.steps.LabelFormat
```



```
wf.steps.LabelTypeEnum
wf.steps.ReleaseDatasetStep
wf.steps.ReleaseDatasetInput
wf.steps.ReleaseDatasetOutput
wf.steps.CreateDatasetStep
wf.steps.CreateDatasetInput
wf.steps.CreateDatasetOutput
wf.steps.DatasetProperties
wf.steps.SchemaField
wf.steps.ImportConfig
wf.steps.JobStep
wf.steps.JobMetadata
wf.steps.JobSpec
wf.steps.JobResource
wf.steps.JobTypeEnum
wf.steps.JobEngine
wf.steps.JobInput
wf.steps.JobOutput
wf.steps.LogExportPath
wf.steps.MrsJobStep
wf.steps.MrsJobInput
wf.steps.MrsJobOutput
wf.steps.MrsJobAlgorithm
wf.steps.ModelStep
wf.steps.ModelInput
wf.steps.ModelOutput
wf.steps.ModelConfig
wf.steps.Template
wf.steps.TemplateInputs
wf.steps.ServiceStep
wf.steps.ServiceInput
wf.steps.ServiceOutput
wf.steps.ServiceConfig
wf.steps.StepPolicy
```

Import the workflow package:

```
wf.workflow
wf.Subgraph
wf.Placeholder
wf.PlaceholderType
wf.AlgorithmParameters
wf.BaseAlgorithm
wf.Algorithm
wf.AIGalleryAlgorithm
wf.resource
wf.SystemEnv
wf.add_whitelist_users
wf.delete_whitelist_users
```

5.3.7 Publishing a Workflow

5.3.7.1 Publishing a Workflow to ModelArts

You can publish a workflow to ModelArts in two ways: **Publishing to the Running State** and **Publishing and Executing the Workflow**. Publishing to the running state requires configuring input and output parameters on the workflow page. Publishing and executing the workflow allows you to modify the code and run it directly through the SDK, eliminating the need to configure and run it on the console.

Publishing to the Running State

After creating a workflow, you can use the **release()** method to publish the workflow to the running state for configuration and execution (on the workflow page of the management console).

Run the following command:

```
workflow.release()
```

After the preceding command is executed, if the log indicates that the workflow is published, you can go to the ModelArts workflow page to view the workflow. On the workflow details page, click **Configure** to configure parameters.

The **release_and_run()** method is based on the **release()** method and allows you to publish and run workflows in the development state, without the need to configure and execute workflows on the console.

CAUTION

Note the following when using this method:

- For all configuration objects related to placeholders in the workflow, you need to either set default values or use fixed data objects directly.
- The method executes differently depending on the workflow object's name. It creates and runs a new workflow if the name does not exist. It updates and runs the existing workflow if the name already exists, using the new workflow structure for the new execution.
  ```
  workflow.release_and_run()
  ```

Publishing and Executing the Workflow

With this method, you can publish and run workflows on the SDK without using the console. You need to modify the workflow code as follows:

```
from modelarts import workflow as wf

# Define a unified output storage path.
output_storage = wf.data.OutputStorage(name="output_storage", description="Unified configuration of output storage", default="**")

# Dataset object
dataset = wf.data.DatasetPlaceholder(name="input_data", default=wf.data.Dataset(dataset_name="**", version_name="**"))

# Create a training job.
job_step = wf.steps.JobStep(
    name="training_job",
    title="Image Classification Training",
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="**", # Subscription ID of the image classification algorithm. Obtain the subscription ID on the algorithm management page. This parameter is optional.
        item_version_id="10.0.0", # Version number of the subscribed algorithm. This parameter is optional.
        parameters=[
            wf.AlgorithmParameters(name="task_type", value="image_classification_v2"),
            wf.AlgorithmParameters(name="model_name", value="resnet_v1_50"),
            wf.AlgorithmParameters(name="do_train", value="True"),
            wf.AlgorithmParameters(name="do_eval_along_train", value="True"),
            wf.AlgorithmParameters(name="variable_update", value="horovod"),
            wf.AlgorithmParameters(name="learning_rate_strategy",
```



```
value=wf.Placeholder(name="learning_rate_strategy", placeholder_type=wf.PlaceholderType.STR,
default="0.002", description="Learning rate for training. 10:0.001,20:0.0001 indicates that the learning rate
of the first 10 epochs is 0.001 and that of the next 10 epochs is 0.0001. If the epoch is not specified, the
learning rate will be adjusted based on the validation precision. The training will be stopped if the precision
is not significantly improved anymore.")),
        wf.AlgorithmParameters(name="batch_size", value=wf.Placeholder(name="batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="Number of images trained in each step
(on a single card)")),
        wf.AlgorithmParameters(name="eval_batch_size", value=wf.Placeholder(name="eval_batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="Number of images validated in each
step (on a single card)")),
        wf.AlgorithmParameters(name="evaluate_every_n_epochs",
value=wf.Placeholder(name="evaluate_every_n_epochs", placeholder_type=wf.PlaceholderType.FLOAT,
default=1.0, description="Validation is performed every n epochs.")),
        wf.AlgorithmParameters(name="save_model_secs", value=wf.Placeholder(name="save_model_secs",
placeholder_type=wf.PlaceholderType.INT, default=60, description="Model saving frequency (unit: s)")),
        wf.AlgorithmParameters(name="save_summary_steps",
value=wf.Placeholder(name="save_summary_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="Summary saving frequency (unit: step)")),
        wf.AlgorithmParameters(name="log_every_n_steps",
value=wf.Placeholder(name="log_every_n_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="Log printing frequency (unit: step)")),
        wf.AlgorithmParameters(name="do_data_cleaning",
value=wf.Placeholder(name="do_data_cleaning", placeholder_type=wf.PlaceholderType.STR, default="True",
description="Whether to clean data. If the data format is abnormal, the training fails. You are advised to
enable this function to ensure training stability. If the data volume is too large, data cleaning may take a
long time. You can clean data offline. (Formats including BMP, JPEG, PNG, and RGB three-channel are
supported.) You are advised to use JPEG.")),
        wf.AlgorithmParameters(name="use_fp16", value=wf.Placeholder(name="use_fp16",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use mixed precision.
Mixed precision accelerates training but causes precision loss. Enable this parameter unless precision is
strictly required.")),
        wf.AlgorithmParameters(name="xla_compile", value=wf.Placeholder(name="xla_compile",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use XLA for accelerated
training. This function is enabled by default.")),
        wf.AlgorithmParameters(name="data_format", value=wf.Placeholder(name="data_format",
placeholder_type=wf.PlaceholderType.ENUM, default="NCHW", enum_list=["NCHW", "NHWC"],
description="Input data format. NHWC indicates channel last, and NCHW indicates channel first. This
parameter defaults to NCHW (faster).")),
        wf.AlgorithmParameters(name="best_model", value=wf.Placeholder(name="best_model",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to save and use the model
with the highest precision instead of the latest model during training. The default value is True, indicating
that the optimal model is saved. Within a certain error range, the latest high precision model is saved as
the optimal model.")),
        wf.AlgorithmParameters(name="jpeg_preprocess", value=wf.Placeholder(name="jpeg_preprocess",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use the JPEG
preprocessing acceleration operator (only JPEG data is supported) to accelerate data reading and improve
performance. This function is enabled by default. If the data format is not JPEG, enable data cleaning to use
the function."))
      ]
   ),
   inputs=[wf.steps.JobInput(name="data_url", data=dataset)],
   outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/")))],
   spec=wf.steps.JobSpec(
      resource=wf.steps.JobResource(
         flavor=wf.Placeholder(
            name="training_flavor",
            placeholder_type=wf.PlaceholderType.JSON,
            description="Training flavor",
            default={"flavor_id": "**"}
         )
      )
   )
)

# Create a workflow.
workflow = wf.Workflow(
   name="image-classification-ResNeSt",
```

```
    desc="this is a image classification workflow",
    steps=[job_step],
    storages=[output_storage]
)
```

1. Fill in the actual values for all ** in the code above. The configuration mainly involves these three items:
   - Unified storage: default value of **output_storage**. Enter an existing OBS path in the format of */OBS bucket name/Folder path/*.
   - Dataset object: Enter the dataset name and version number.
   - Training flavor: Configure GPU resources since the algorithm in this example can run only on GPUs. You can use free flavor **modelarts.p3.large.public.free**.
2. After the configuration, run this code:
   ```
   workflow.release_and_run()
   ```
3. After the execution, go to the ModelArts console. In the navigation pane, choose **Workflow** to view the workflow status.

5.3.7.2 Publishing a Workflow to AI Gallery

You can publish workflows to AI Gallery and share them with other users. To do so, run this code:

```
workflow.release_to_gallery()
```

Once the release is done, you can visit AI Gallery to see the asset details. The asset permission is set to private by default, but you can change it if you want.

1. Go to AI Gallery.
2. Choose **My Gallery** > **My Assets** > **Workflow**.
3. In the **My Publishes** tab, view the workflow published to AI Gallery.
4. Click the workflow name to view the workflow details.

The **release_to_gallery()** method contains the following input parameters.

Parameter	Description	Mandatory	Type
content_id	Workflow asset ID	No	str
version	Version number of the workflow asset. The format is *x.x.x*.	No	str
desc	Description of the workflow asset version	No	str
title	Workflow asset name. If this parameter is not specified, the workflow name is used by default.	No	str

Parameter	Description	Mandatory	Type
visibility	Visibility of the workflow asset. The value can be **public**, **group** (whitelist), and **private** (visible only to you). The default value is **private**.	No	str
group_users	Whitelist. This parameter is mandatory only when visibility is set to **group**. You can only enter **domain_id**.	No	list[str]

You can use the method in two ways based on the input parameters:

- workflow.release_to_gallery(title="*Asset name*") publishes a new workflow asset with version 1.0.0. If the workflow includes an algorithm that is not from AI Gallery, it also publishes the algorithm to AI Gallery with version 1.0.0.
- workflow.release_to_gallery(content_id="**", title="*Asset name*") publishes a new version based on the specified workflow asset. The version number increases automatically. If the workflow includes an AI Gallery algorithm, it updates the algorithm asset with a higher version number.

Workflow asset whitelist setting:

You can specify the **visibility** and **group_users** fields of the **release_to_gallery** method when you publish an asset for the first time. To change the whitelist for accessing a certain asset, use this command:

```
from modelarts import workflow as wf

# Add specified whitelist users.
wf.add_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])

# Delete specified whitelist users.
wf.delete_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])
```

NOTE

When you modify the whitelist for a workflow asset, the system automatically obtains the information of the algorithm asset that the workflow version depends on and sets the whitelist for the algorithm asset as well.

5.3.8 Advanced Workflow Capabilities

5.3.8.1 Using Big Data Capabilities (DLI/MRS) in a Workflow

Function

This phase calls MRS for big data cluster computing. It is used for batch data processing and model training.

Application Scenarios

You can use MRS Spark for big data computing in this phase.

Examples

On the Huawei Cloud MRS console, check available MRS clusters of your account. If no MRS cluster is available, create one with Spark selected.

You can use MrsStep to create a job phase. The following is an example of defining a MrsStep:

- **Specifying a boot script and cluster**

```
from modelarts import workflow as wf
# Define a MrsJobStep using MrsStep.

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file="obs://spark-sql/wordcount.py",    # OBS path to the boot script
    parameters=[wf.AlgorithmParameters(name="run_args", value="--master,yarn-cluster")]
)
inputs = wf.steps.MrsJobInput(name="mrs_input", data=wf.data.OBSPath(obs_path="/spark-sql/
mrs_input/"))  # OBS path to the input data
outputs = wf.steps.MrsJobOutput(name="mrs_output",
obs_config=wf.data.OBSOutputConfig(obs_path="/spark-sql/mrs_output")) # OBS path to the output
data
step = wf.steps.MrsJobStep(
        name="mrs_test",          # Step name, which can be customized
        mrs_algorithm=algorithm,
        inputs=inputs,
        outputs=outputs,
        cluster_id="cluster_id_xxx"   # MRS cluster ID
)
```

- **Configuring a cluster and boot script**

```
from modelarts import workflow as wf
# Define a phase using MrsJobStep.
run_arg_description = "Program execution parameter, which is used as the program running
environment parameter. The default value is (--master,yarn-cluster)".
app_arg_description = "Program execution parameter, which is used as the input parameter of the
boot script, for example, (--param_a=3,--param_b=4). This parameter is optional and left blank by
default."
mrs_outputs_description = "Data output path, which can be obtained from train_url in the parameter
list."
cluster_id_description = "cluster id of MapReduce Service"

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file=wf.Placeholder(name="boot_file",
                             description="Program boot script",
                             placeholder_type=wf.PlaceholderType.STR,
                             placeholder_format="obs"),
    parameters=[wf.AlgorithmParameters(name="run_args",
                                       value=wf.Placeholder(name="run_args",
                                                            description=run_arg_description,
                                                            default="--master,yarn-cluster",
                                                            placeholder_type=wf.PlaceholderType.STR),
                                       ),
                wf.AlgorithmParameters(name="app_args",
                                       value=wf.Placeholder(name="app_args",
                                                            description=app_arg_description,
                                                            default="",
```



```
                                        placeholder_type=wf.PlaceholderType.STR)
                                )
                        ]
        )

inputs = wf.steps.MrsJobInput(name="data_url",
                              data=wf.data.OBSPlaceholder(name="data_url",object_type="directory"))

outputs = wf.steps.MrsJobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=wf.Placeholder(name="train_url",
placeholder_type=wf.PlaceholderType.STR,
placeholder_format="obs",description=mrs_outputs_description)))

mrs_job_step = wf.steps.MrsJobStep(
    name="mrs_job_test",
    mrs_algorithm=algorithm,
    inputs=inputs,
    outputs=outputs,
    cluster_id=wf.Placeholder(name="cluster_id", placeholder_type=wf.PlaceholderType.STR,
description=cluster_id_description, placeholder_format="cluster")
)
```

- **Using an MRS phase on the console**

  After a workflow is published, configure phase parameters such as the data input, data output, boot script, and cluster ID on the workflow configuration page.

5.3.8.2 Specifying Certain Phases to Run in a Workflow

Workflows support predefined scenarios to enable partial execution. You can split the DAG into different branches based on the scenarios during workflow development. Then, you can run each branch independently as a separate workflow. The sample code is as follows:

```
workflow =wf.Workflow(
    name="image_cls",
    desc="this is a demo workflow",
    steps=[label_step, release_data_step, training_step, model_step, service_step],
    policy=wf.policy.Policy(
        scenes=[
            wf.policy.Scene(
                scene_name="Model training",
                scene_steps=[label_step, release_data_step, training_step]
            ),
            wf.policy.Scene(
                scene_name="Service deployment",
                scene_steps=[model_step, service_step]
            ),
        ]
    )
)
```

This example shows a workflow with five phases. The policy defines two preset scenarios: model training and service deployment. When the workflow is published to the running state, partial execution is disabled by default and all phases run. You can specify certain scenarios to run on the global configuration page.

NOTE

You can define the same phase in different running scenarios using partial execution. However, you must ensure that the data dependency between phases is correct. Partial execution can only be configured and used in the running state and cannot be debugged in the development state.

6 Development Environments

6.1 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 and use dynamic mounting to simulate OBS objects as a local file system. 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**.

6.2 Creating a Notebook Instance

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

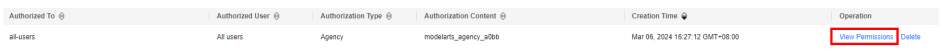
Context

- Notebook is billed as follows:
  - A running notebook instance will be billed based on used resources. The fees vary depending on your selected resources. For details, see **Pricing Details**. When a notebook instance is not used, stop it.
  - If you select EVS for storage when creating a notebook instance, the EVS disk will be continuously billed if the instance is not deleted. Stop and delete the notebook instance if it is not required.
- When a notebook instance is created, auto stop is enabled by default. The notebook instance will automatically stop at the specified time.
- Only running notebook instances can be accessed or stopped.
- A maximum of 10 notebook instances can be created under one account.

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**.

   Figure 6-1 Viewing agency configurations

2. Log in to the ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create** in the upper right corner. On the **Create Notebook** page, configure parameters.

a. Configure basic information of the notebook instance, including its name, description, and auto stop status. For details, see **Table 6-1**.

Figure 6-2 Basic information of a notebook instance

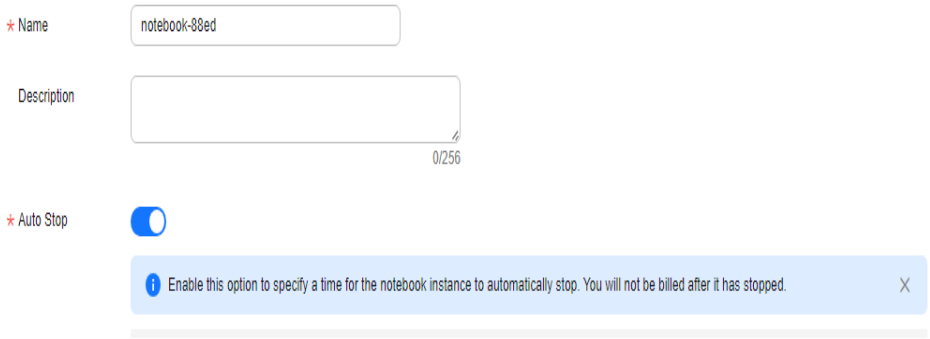


Table 6-1 Basic parameters

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
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 and its resource billing will stop then. 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 provided for you to renew the auto stop time.

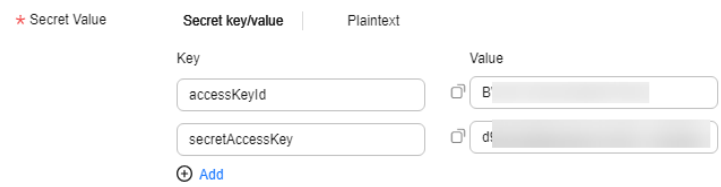
b. Configure notebook parameters, such as the image and instance flavor. For details, see **Table 6-2**.

Table 6-2 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. Public resource pools are billed based on the running duration of your notebook instances. Select a created dedicated resource pool based on site requirements. If no dedicated resources are available, purchase one. **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.
Type	Processor type, which can be **CPU** or **GPU**. The chips vary depending on the selected image. GPUs deliver better performance that CPUs but at a higher cost. Select a chip type as needed.

Parameter	Description
Flavor	The flavor of your notebook instance. Select a flavor based on your needs. • CPU **2vCPUs 8GB**: General-purpose Intel CPU flavor, ideal for rapid data exploration and experiments **8vCPUs 32GB**: General computing-plus Intel CPU flavor, ideal for compute-intensive applications • GPU **GPU: 1*Vnt1(32GB)|CPU: 8vCPUs 64GB**: Single GPU with 32 GB of memory, ideal for algorithm training and debugging in deep learning scenarios **GPU: 1*Tnt004(16GB)|CPU: 8vCPUs* 32GB**: Single GPU with 16 GB of memory, ideal for inference computing such as computer vision, video processing, and NLP tasks **GPU: 1*Pnt1(16GB)|CPU: 8vCPUs 64GB**: Single GPU with 16 GB of memory, ideal for algorithm training and debugging in deep learning scenarios

Parameter	Description
Storage	The value can be **EVS**, **SFS**, **OBS**, or **PFS**. Configure this parameter based on your needs. **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**. **NOTE** For details about how to set permissions to access SFS Turbo folders, see **Permissions Management**. – **Scalable File Service**: Select a created SFS Turbo file system. To create an SFS Turbo file system, log in to **Huawei Cloud**. – **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, **create a secret**. Enter **accessKeyId** and **secretAccessKey** under **Key**, and enter the AKs/SKs obtained from **My Credentials** > **Access Keys** under **Value**.

Parameter	Description
	Figure 6-3 Configuring the secret values  **EVS** and **SFS** are all mounted to the **/home/ma-user/work** directory. You can add a data storage path during the runtime of a notebook instance by referring to **Dynamically Mounting an OBS Parallel File System**. The data is retained in **/home/ma-user/work**, 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.
Extended Storage	**NOTE** This parameter is a whitelist function. If you have trial requirements, submit a service ticket to apply for permissions. If you need multiple data storage paths, click **Add Extended Storage** to add more storage mount directories. You can add an OBS, PFS, or SFS directory. Constraints: • For each type, a maximum of five directories can be mounted. • The directories must be unique and cannot be mounted to a blacklisted directory. Nested mounting is allowed. Blacklisted directories are those with the following prefixes: **/data/**, **/cache/**, **/dev/**, **/etc/**, **/bin/**, **/lib/**, **/sbin/**, **/modelarts/**, **/train-worker1-log/**, **/var/**, **/resource_info/**, **/usr/**, **/sys/**, **/run/**, **/tmp/**, **/infer/**, and **/opt/** After this parameter is configured, the notebook instance details page is displayed. Click Storage **Storage** > **Extended Storage** to view or edit the extended storage information. If the number of storage devices does not reach the maximum, you can click **Add Extended Storage** on the right.

Parameter	Description
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. To do so, choose **Key Pair Service** > **Account Key Pairs** and click **Create Key Pair**. 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.
Whitelist	Set a whitelist after remote SSH is enabled. This parameter is optional. Add the IP addresses for remotely accessing the notebook instance to the whitelist, for example, the IP address of your local PC or the public IP address of the source device. A maximum of five IP addresses can be added and separated by commas (,). If the parameter is left blank, all IP addresses will be allowed for remote SSH access. If your source device and 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. After a notebook instance is created, you can change the whitelist IP addresses on the instance details page.

c. (Optional) Add tags to the notebook instance. Enter a tag key and value and click **Add**.

Table 6-3 Adding a tag

Parameter	Description
Tags	ModelArts can work with Tag Management Service (TMS). When creating resource-consuming tasks in ModelArts, for example, training jobs, configure tags for these tasks so that ModelArts can use tags to manage resources by group. For details about how to use tags, see **Using TMS Tags to Manage Resources by Group**. After adding a tag, you can view, modify, or delete the tag on the notebook instance details page.

NOTE

You can select a predefined TMS tag from the tag drop-down list or customize a tag. Predefined tags are available to all service resources that support tags. Customized tags are available only to the service resources of the user who has created the tags.

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 when 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.

   On the **Storage** tab page, click **Mount Storage** to mount an OBS parallel file system to the instance for reading data. For details, see **Dynamically Mounting an OBS Parallel File System**.

   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 6-4**. The following mounting points are not saved when images are saved.

Table 6-4 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 6-5 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 4,096 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 Huawei technical support. • Only OBS PFS in the same region can be mounted. PFS buckets mounted as persistent storage for AI development and exploration. 1. 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**. 2. Storage for code. After debugging on a notebook instance, specify the OBS path as the code path for starting training, facilitating	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
	temporary modification. 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 Huawei 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 dedicated 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 target 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.

1. How do I use EVS in a development environment?

   When creating a notebook instance, select a small-capacity EVS disk. You can scale out the disk as needed. For details, see **Dynamically Expanding EVS Disk Capacity**.

2. How do I use an OBS parallel file system in a development environment?

   When training data in a notebook instance, you can use the datasets mounted to a notebook container, and use an OBS parallel file system. For details, see **Dynamically Mounting an OBS Parallel File System**.

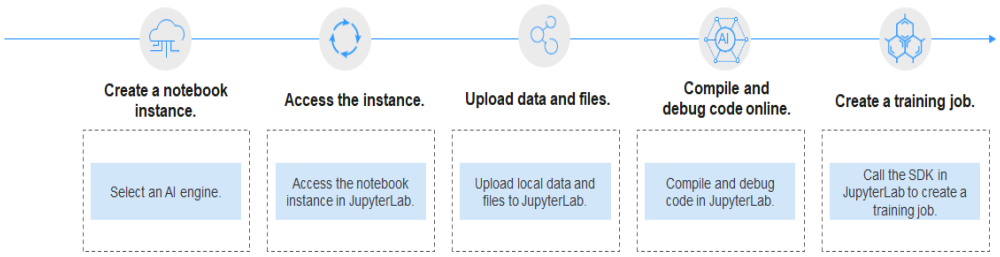
6.3 Using a Notebook Instance for AI Development Through JupyterLab

6.3.1 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 6-4** shows the operation process.

Figure 6-4 Using JupyterLab to develop and debug code online



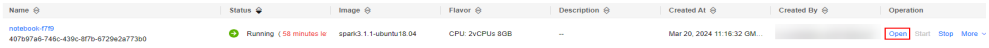
Procedure

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**.

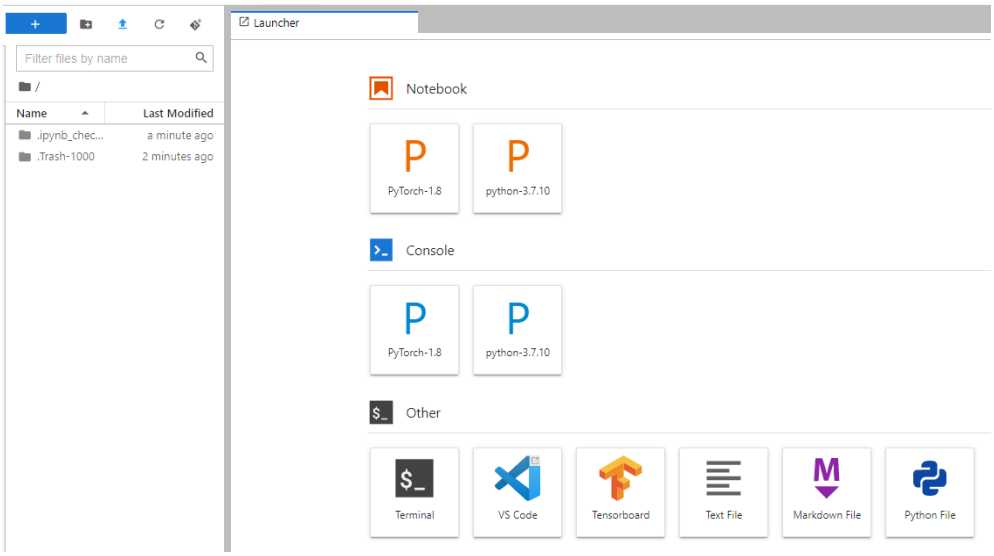
2. After the notebook instance is created, it is in the **Running** state. Locate it in the list and click **Open** in the **Operation** column to access JupyterLab.

   Figure 6-5 Accessing a notebook instance

3. The **Launcher** page is automatically displayed. Perform required operations. For details, see **JupyterLab Documentation**.

   Figure 6-6 JupyterLab homepage

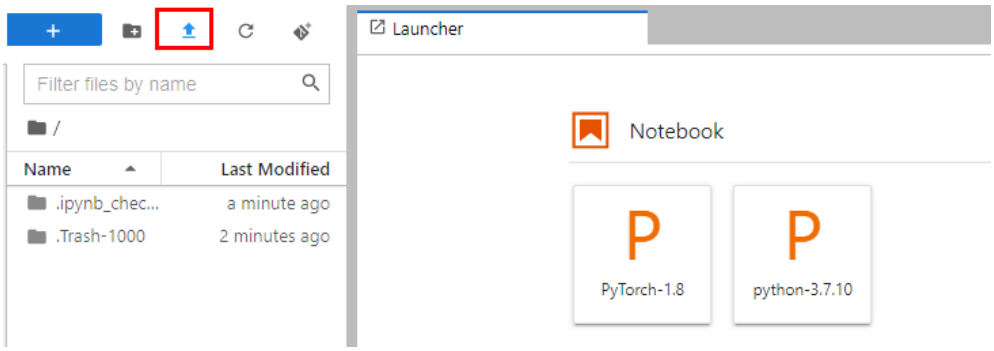
   

   NOTE

   The notebook and console kernels and versions displayed on the **Launcher** page vary depending on the AI engine based on which a notebook instance is created. **Figure 6-6** shows an example only. Obtain the notebook and console kernels and versions on the management console.

4. Upload training data and code files to JupyterLab. For details, see **Uploading Files from a Local Path to JupyterLab**.

Figure 6-7 Button for uploading a file

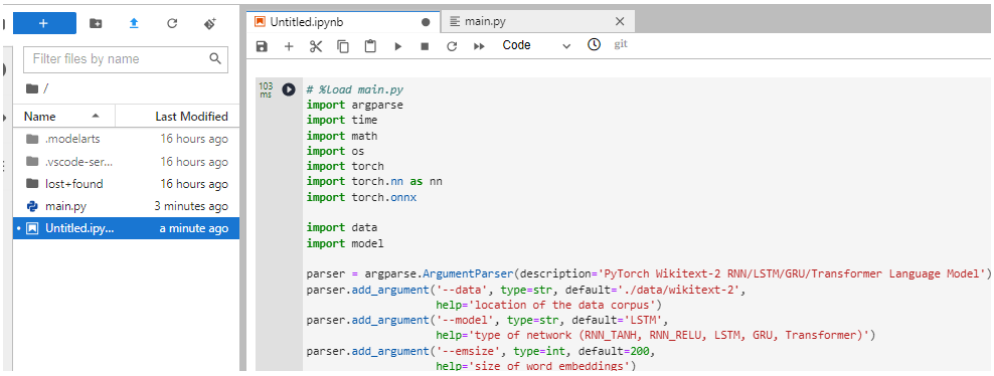


5. In the navigation pane on the left, double-click the uploaded code file, compile the file in JupyterLab, and debug it. For details about how to use JupyterLab, see **Common Functions of JupyterLab**.

   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 6-8 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**.

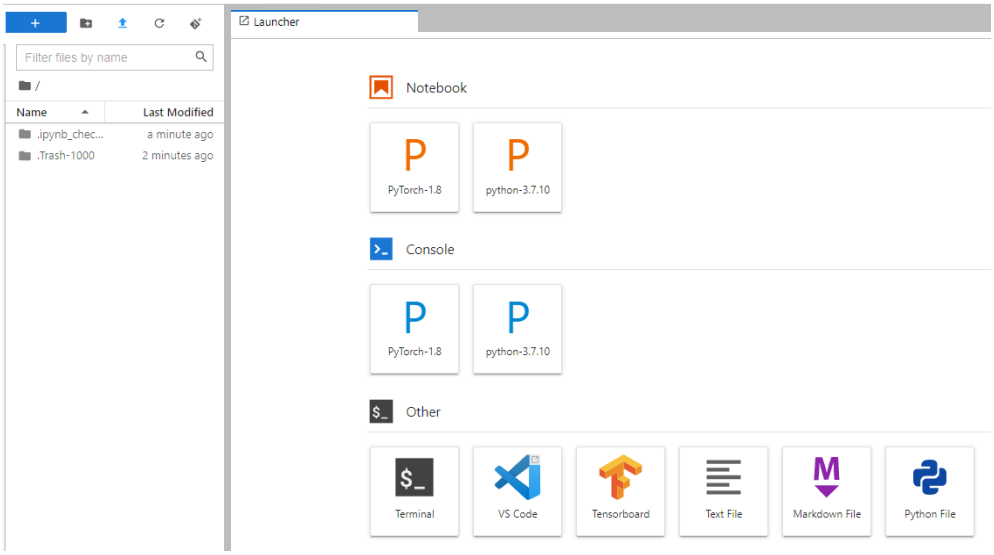
6.3.2 Common Functions of JupyterLab

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 **Development Workspace** > **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**.

Figure 6-9 JupyterLab homepage



NOTE

The notebook and console kernels and versions displayed on the **Launcher** page vary depending on the AI engine based on which a notebook instance is created. **Figure 6-9** shows an example only. Obtain the notebook and console kernels and versions on the management console.

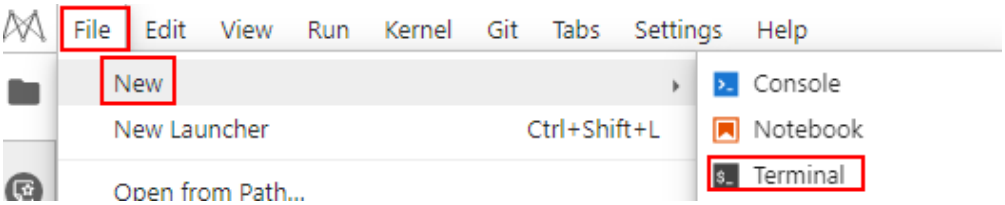
- **Notebook**: Select a kernel for running notebook, for example, TensorFlow or Python.
- **Console**: Call the terminal for command control.
- **Other**: Edit other files.

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 6-10 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:

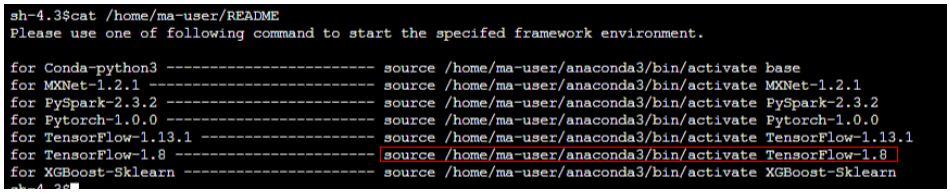
cat /home/ma-user/README

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 6-11 Activating the environment



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

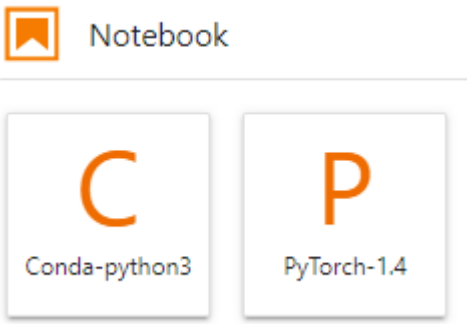
pip install Shapely

Creating an IPYNB File in JupyterLab

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

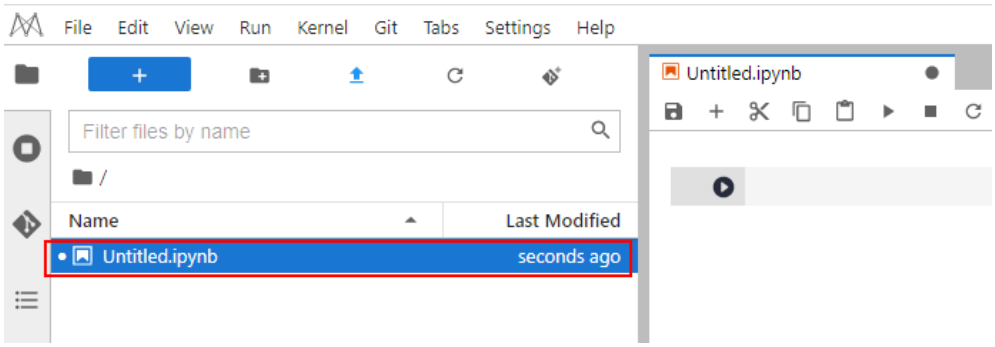
The AI engines supported by each notebook instance vary depending on the runtime environment. The following figure is only an example. Select an AI engine based on site requirements.

Figure 6-12 Selecting an AI engine and creating IPYNB file



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

Figure 6-13 Created IPYNB file



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.

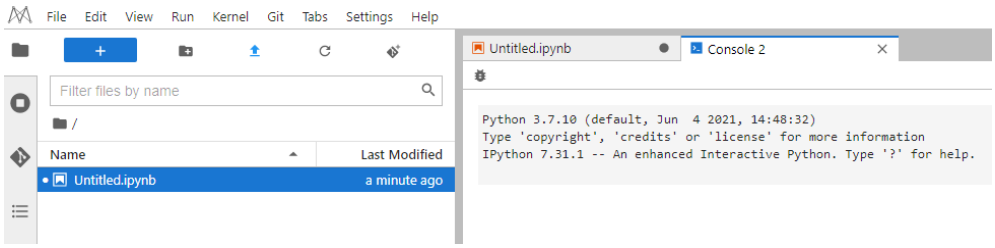
On the JupyterLab homepage, click a proper AI engine in the **Console** area to create a notebook file.

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.

Figure 6-14 Selecting an AI engine and creating a console

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

Figure 6-15 Creating a notebook file (console)

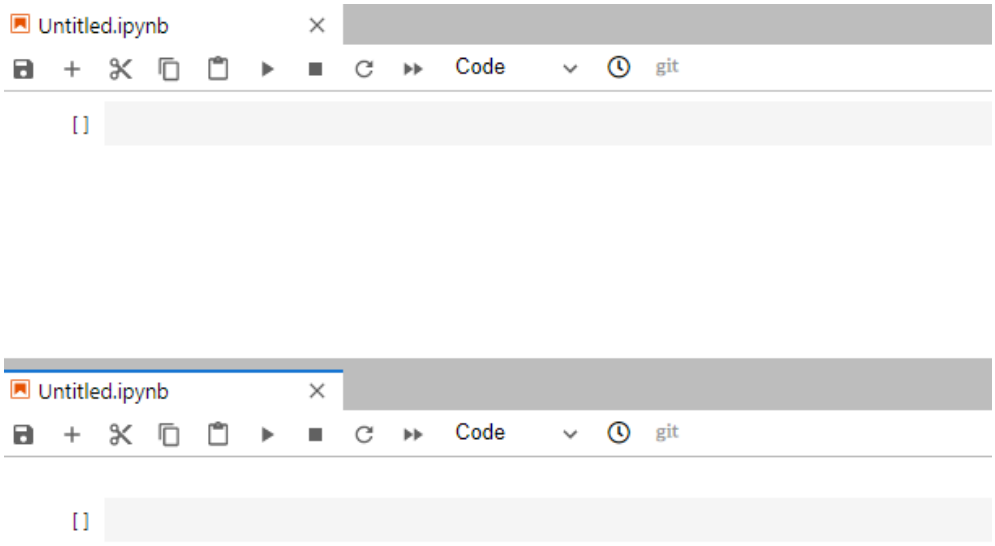


Editing a File in JupyterLab

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.

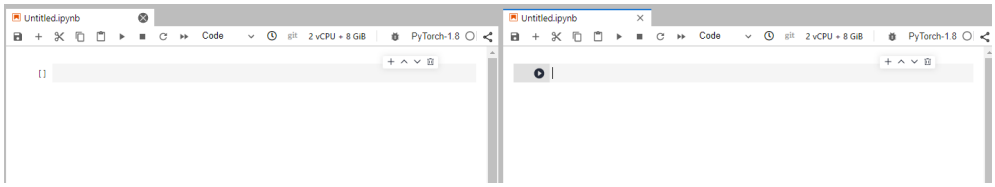
Figure 6-16 Customized display of multiple files



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 6-17 Multiple views of a file



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

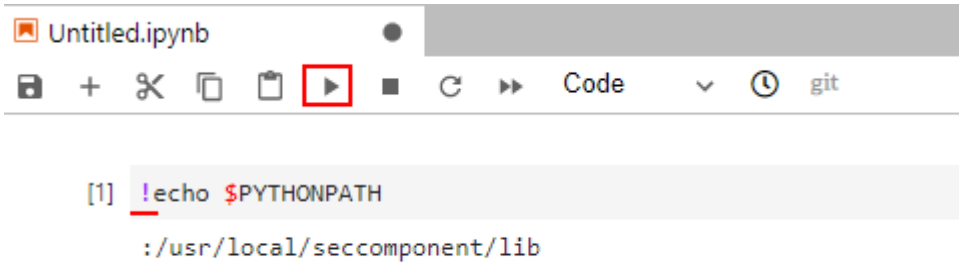
For example, install an external library Shapely.

```
!pip install Shapely
```

For example, obtain PythonPath.

```
!echo $PYTHONPATH
```


Figure 6-18 Running code

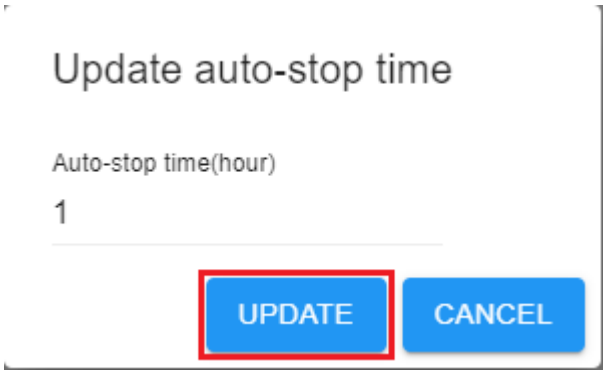


Renewing or Automatically Stopping a Notebook Instance

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 6-19 Remaining duration

Figure 6-20 Renewing an instance



Common JupyterLab Buttons and Plug-ins

Figure 6-21 Common JupyterLab buttons and plug-ins

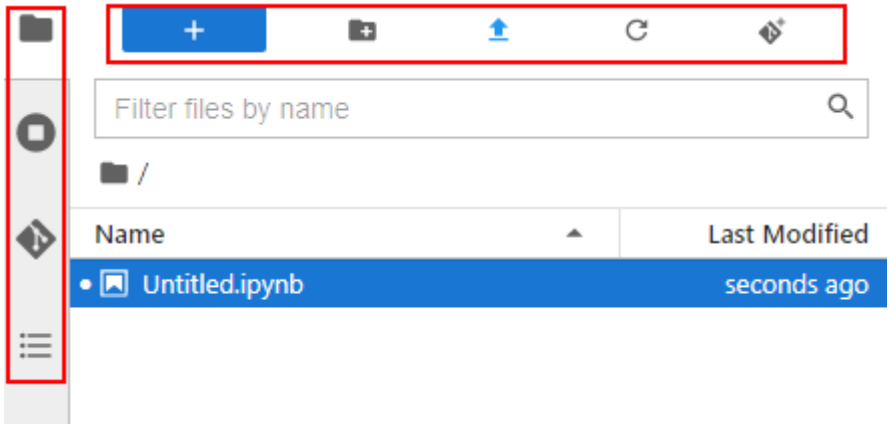


Table 6-6 JupyterLab buttons

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.
	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 6-7 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.

Plug-in	Description
	Show the document organization.

Figure 6-22 Buttons in the navigation bar

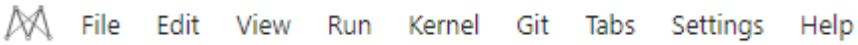


Table 6-8 Buttons in the navigation bar

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.
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 6-23 Buttons in the menu bar of an IPYNB file

Table 6-9 Buttons in the menu bar of an IPYNB file

Button	Description
	Save a file.

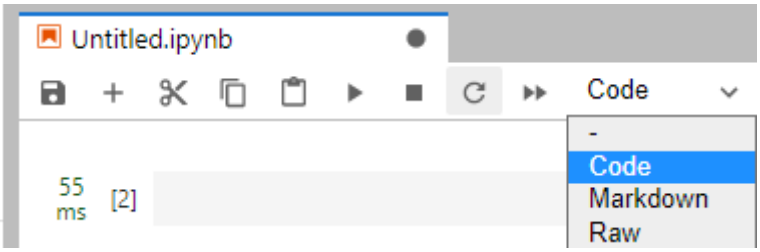
Button	Description
+	Add a new cell.
	Cut the selected cell.
	Copy the selected cell.
	Paste the selected cell.
	Execute the selected cell.
	Terminate a kernel.
	Restart a kernel.
	Restart a kernel and run all code of the current notebook again.
Code	There are four options in the drop-down list: **Code**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

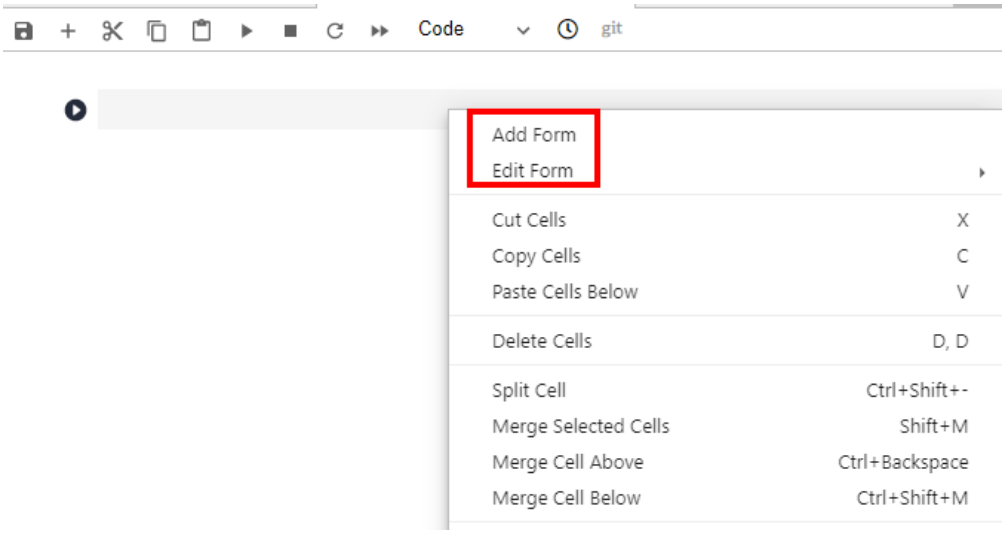
- The **Add Form** and **Edit Form** buttons are available only to the shortcut menu of code cells, as shown below.

Figure 6-24 Viewing a code cell



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

Figure 6-25 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 6-26 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 6-10 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 6-27**. 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 6-28**. • 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 6-27 Form field types

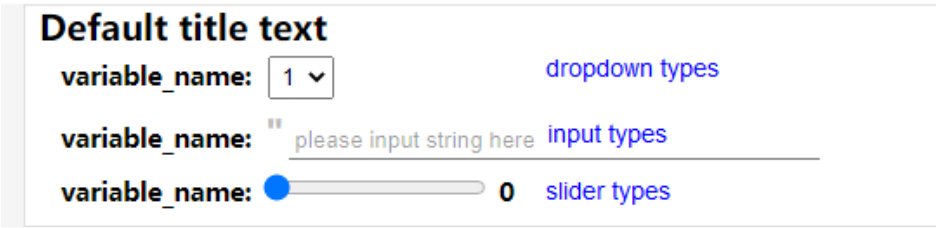


Figure 6-28 Creating a dropdown form

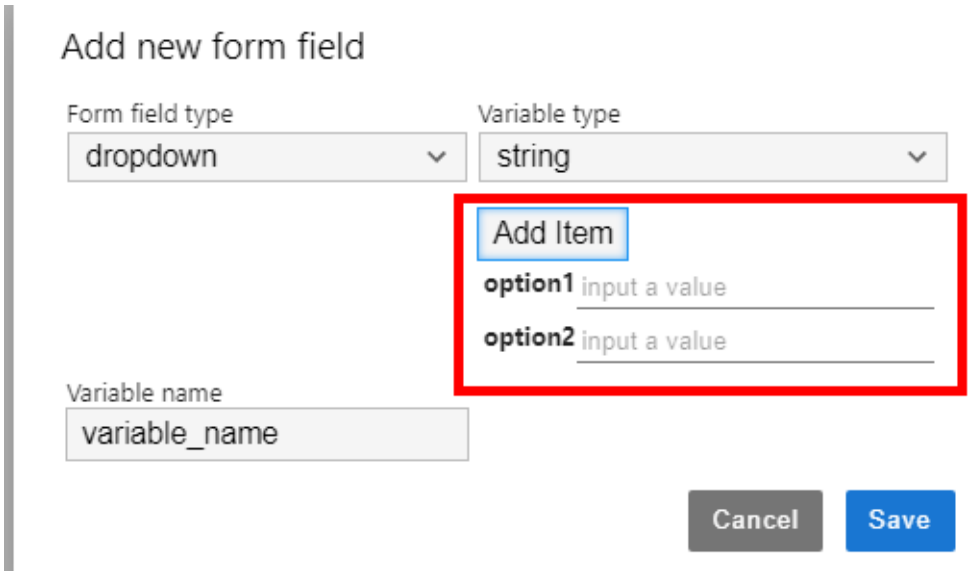
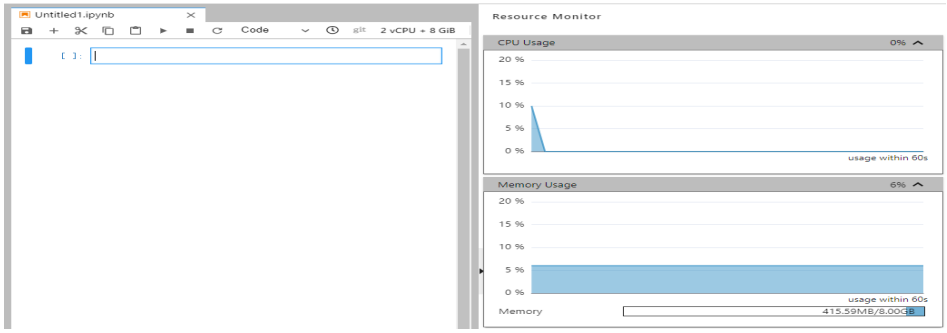


Figure 6-29 Deleting a form

Monitoring Resources

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

Figure 6-30 Resource usage



6.3.3 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

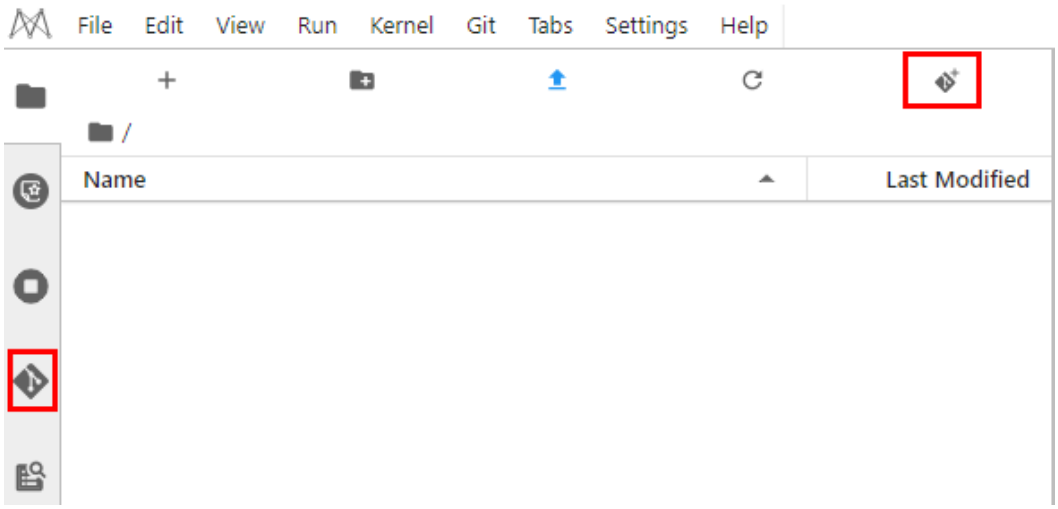
The notebook instance is running.

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 6-31 shows the Git plug-in of JupyterLab.

Figure 6-31 Git plug-in

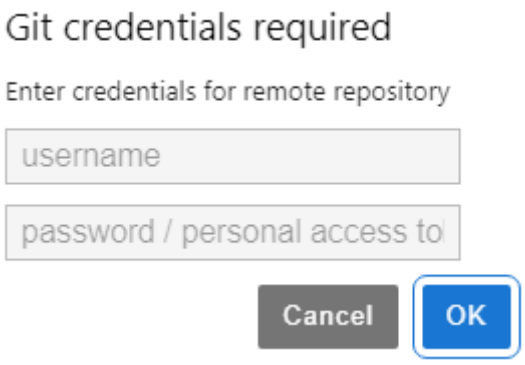


Cloning a GitHub Open-Source Code Repository

Access a GitHub open-source code repository at https://github.com/jupyterlab/extension-examplesitHub. 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.



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 6-32 Cloning a GitHub private code repository (only authorization using a personal access token is supported)

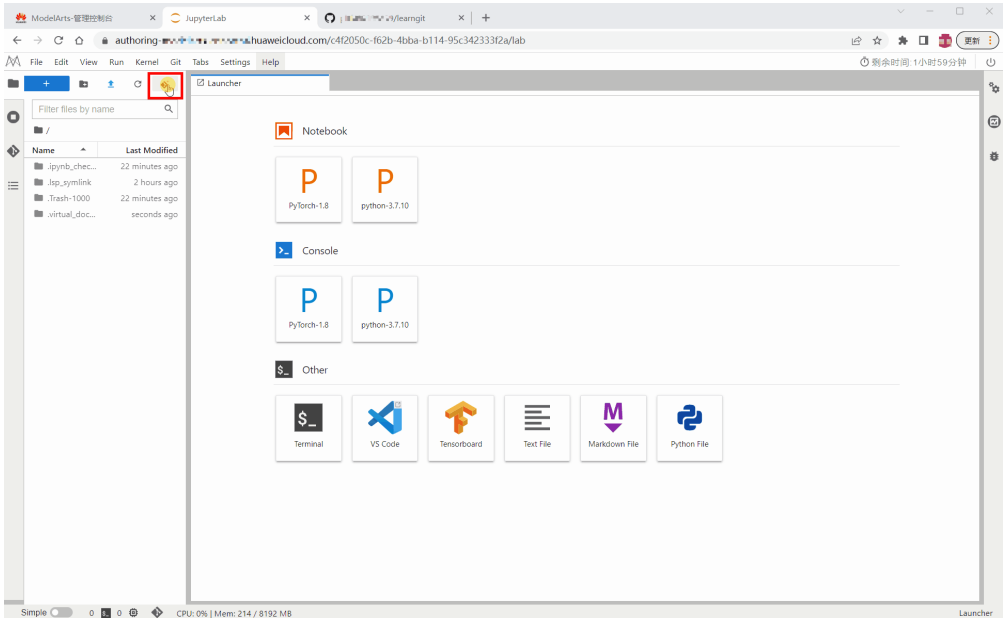
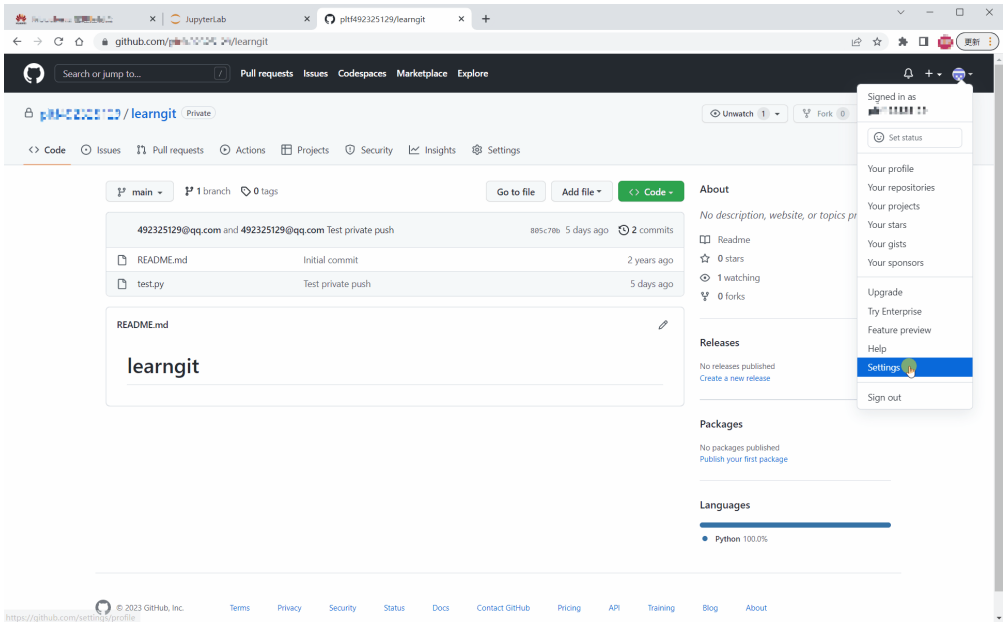


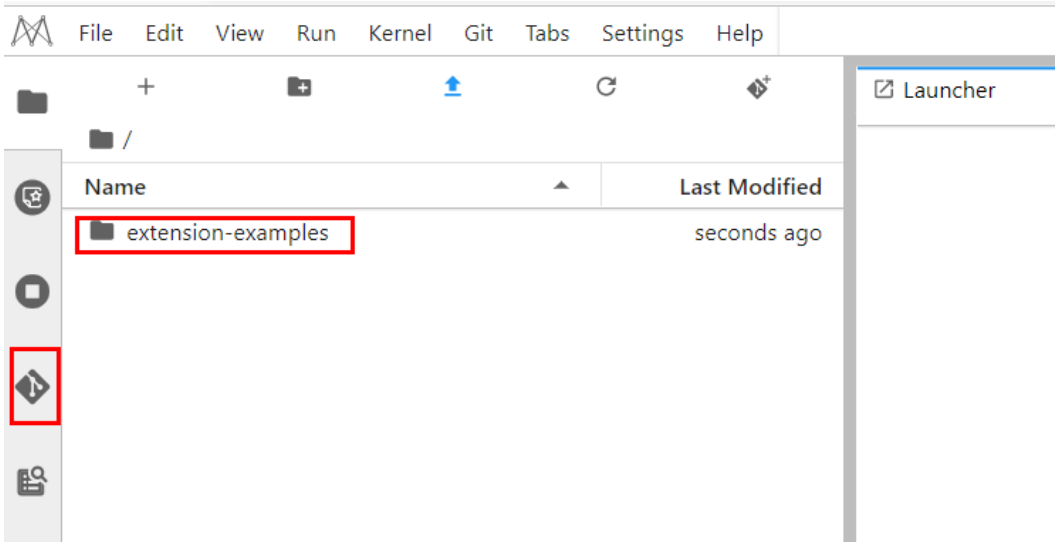
Figure 6-33 Obtaining a personal access token



Viewing a Code Repository

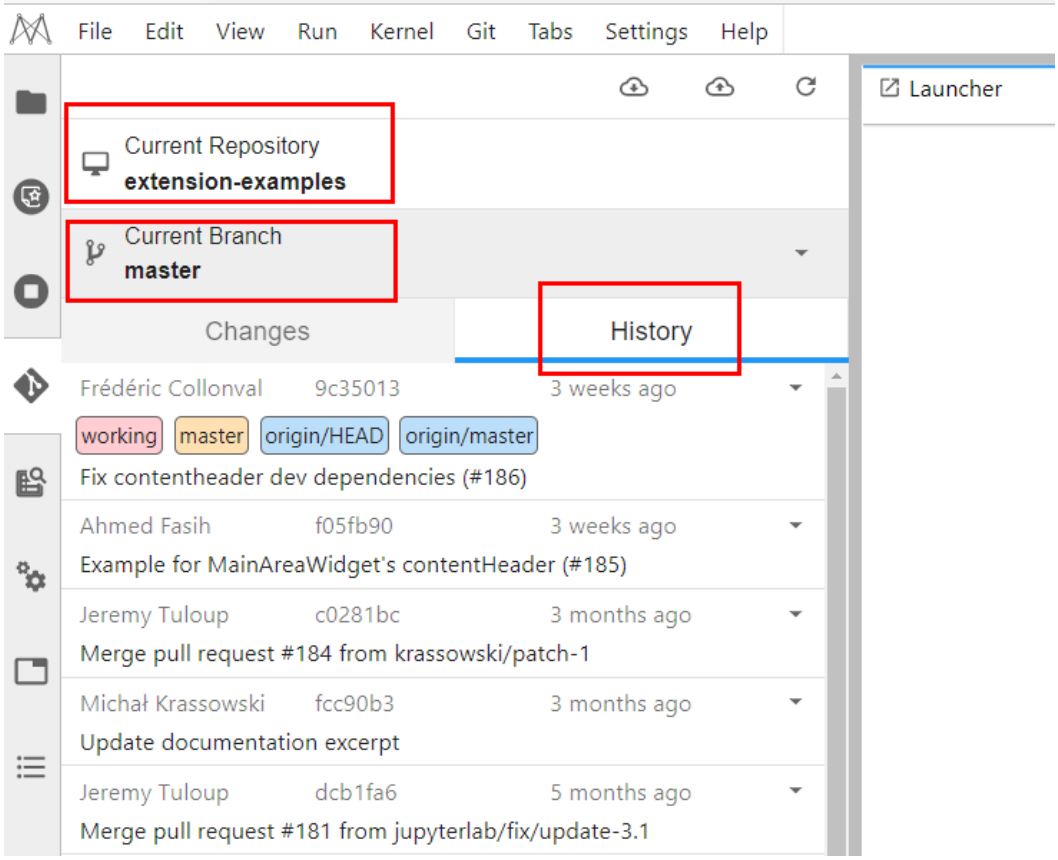
In the list under **Name**, double-click the folder you want to use and click the Git plug-in icon on the left to access the code repository corresponding to the folder.

Figure 6-34 Opening the folder and starting the Git plug-in



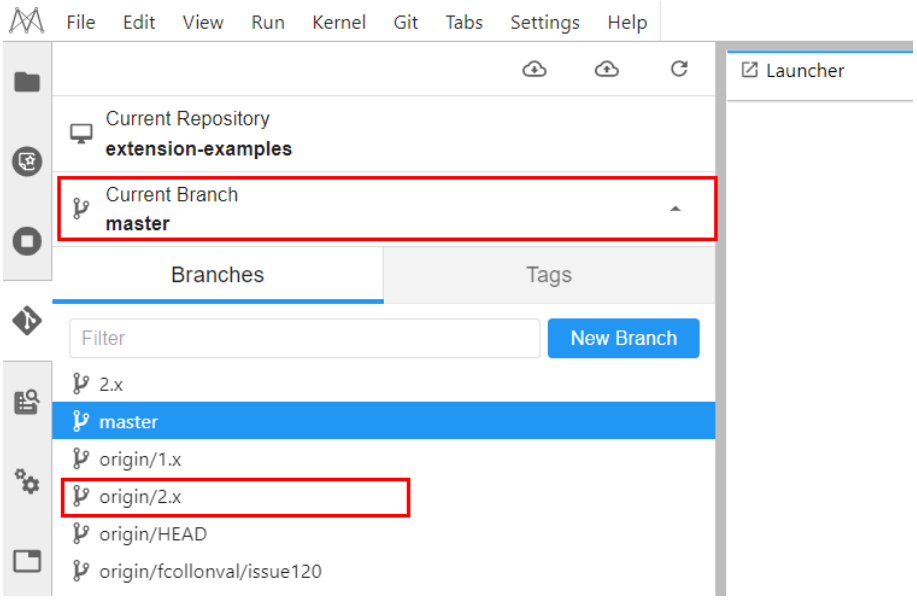
You can view the information current code repository, such as the repository name, branch, and historical submission records.

Figure 6-35 Viewing a code repository



NOTE

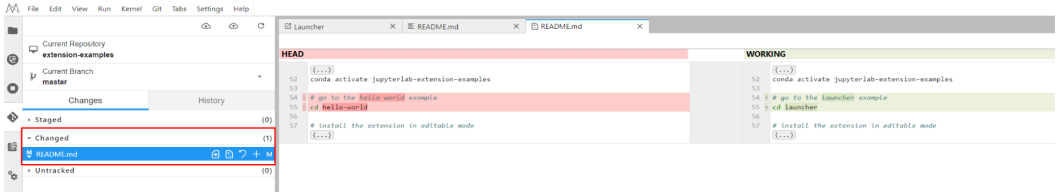
By default, the Git plug-in clones the master branch. To switch another branch, click **Current Branch** to expand all branches and click the target branch name.



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.

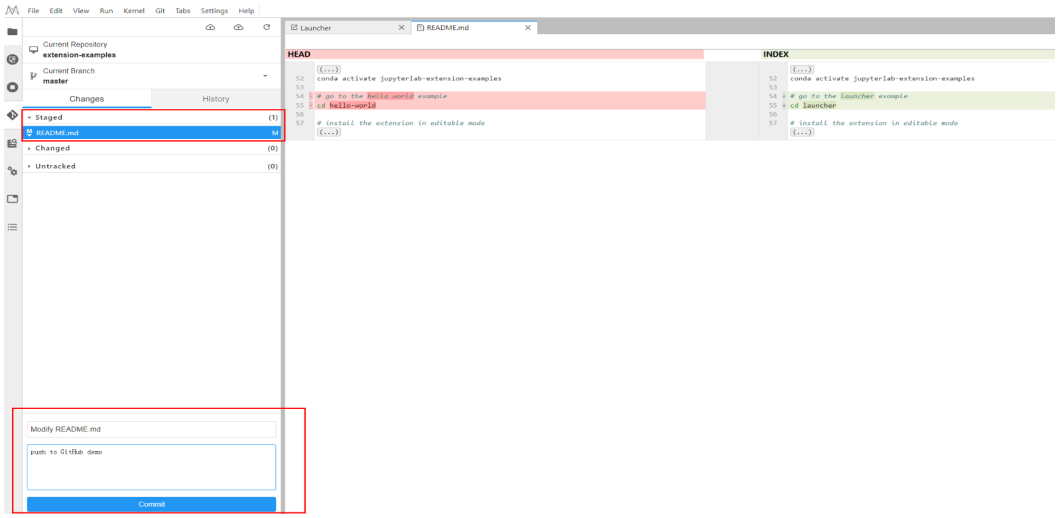
Figure 6-36 Viewing modifications



Committing Modifications

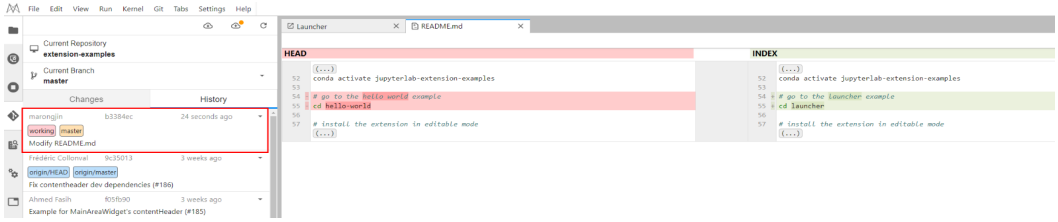
After confirming that the modifications are correct, click **Stage this change** on the right of the file name, which is equivalent to running the **git add** command. The file enters the **Staged** state. Enter the message to be committed in the lower left corner and click **Commit** that is equivalent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

Figure 6-37 Committing modifications



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

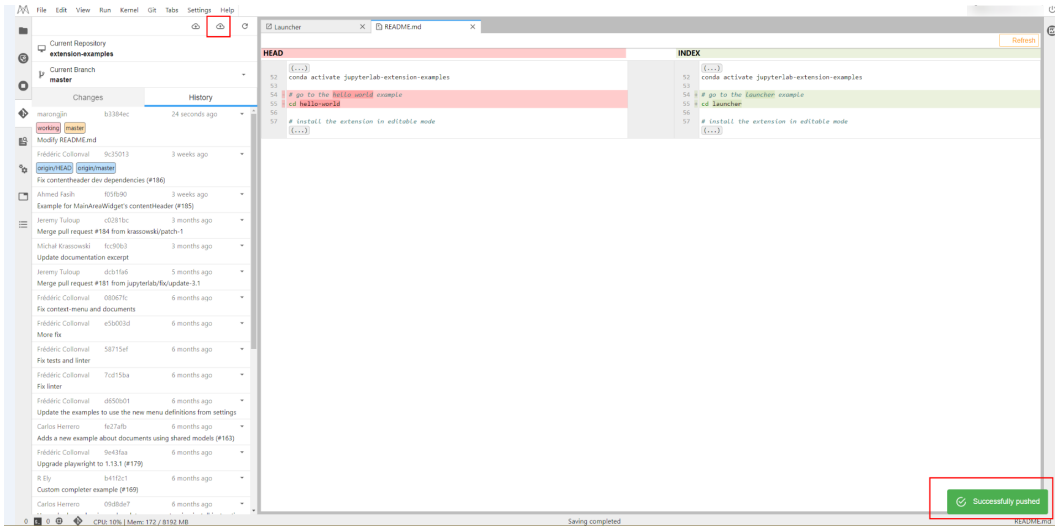
Figure 6-38 Checking whether the committing is successful



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 **What Do I Do If the Git Plug-in Password Is Invalid?**

Figure 6-39 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.

6.3.4 Uploading Files to JupyterLab

6.3.4.1 Uploading Files from a Local Path to JupyterLab

JupyterLab provides multiple methods for uploading files.

Methods for Uploading a File

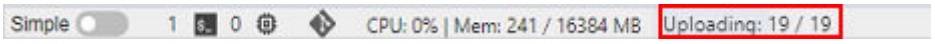
- For a file not exceeding 100 MB, directly upload it to the target notebook instance. Detailed information, such as the file size, upload progress, and upload speed are displayed.
- For a file that exceeds 100 MB but does not exceed 50 GB, upload the file to OBS (an object bucket or a parallel file 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.

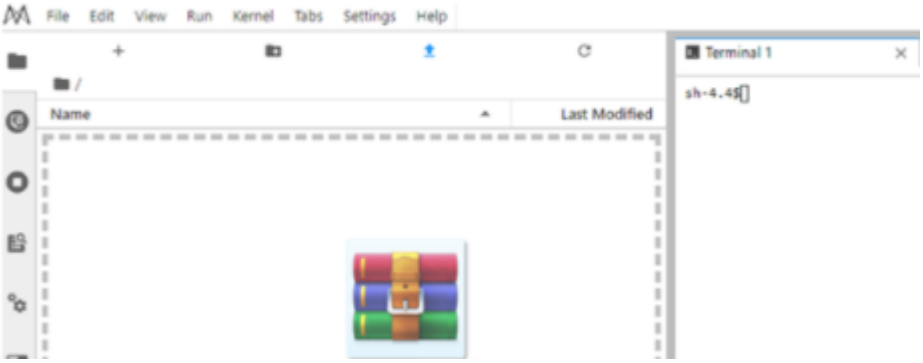
- When multiple files are uploaded in a batch, the total number of files to be uploaded and the number of files that have been uploaded are displayed at the bottom of the JupyterLab window.



Uploading File

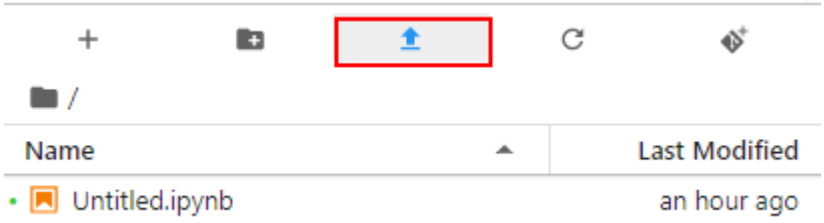
Method 1: Use JupyterLab to open a running notebook environment.

Figure 6-40 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.

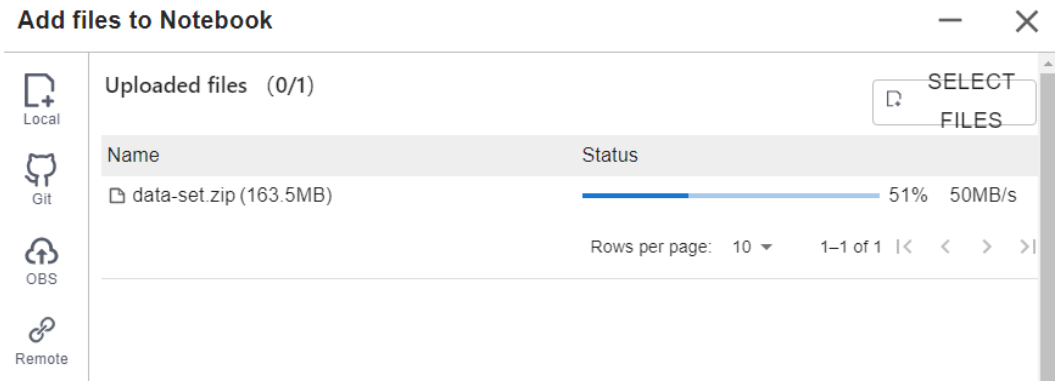
Figure 6-41 Clicking the upload button



Uploading a Local File Smaller Than 100 MB to JupyterLab

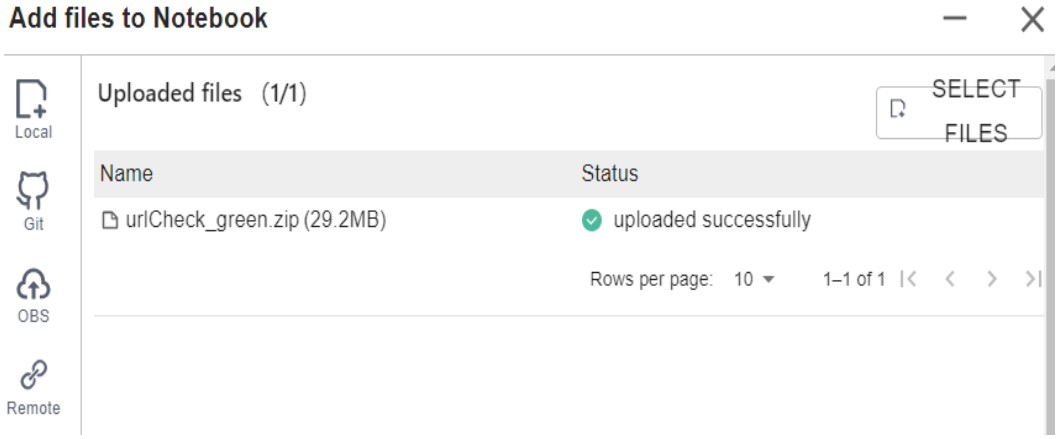
For a file not exceeding 100 MB, directly upload it to the target notebook instance. Detailed information, such as the file size, upload progress, and upload speed are displayed.

Figure 6-42 Uploading a file less than 100 MB



A message is displayed after the file is uploaded.

Figure 6-43 Uploaded

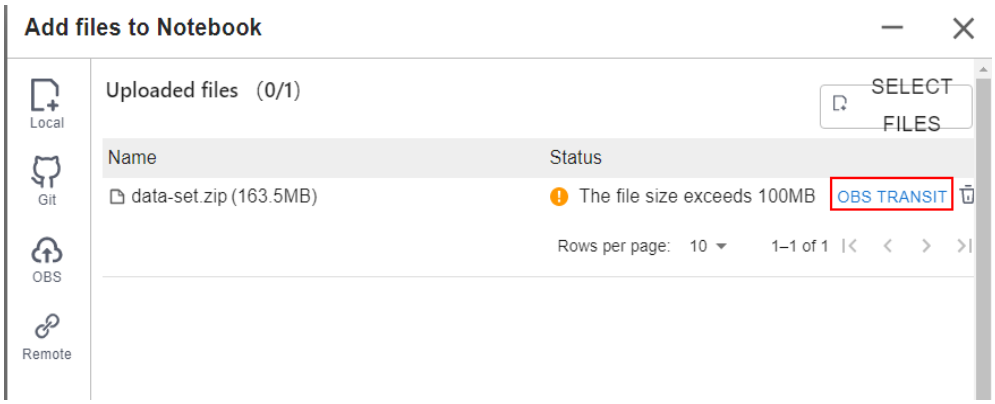


Uploading a Local File that Is 100 MB to 50 GB to JupyterLab

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.

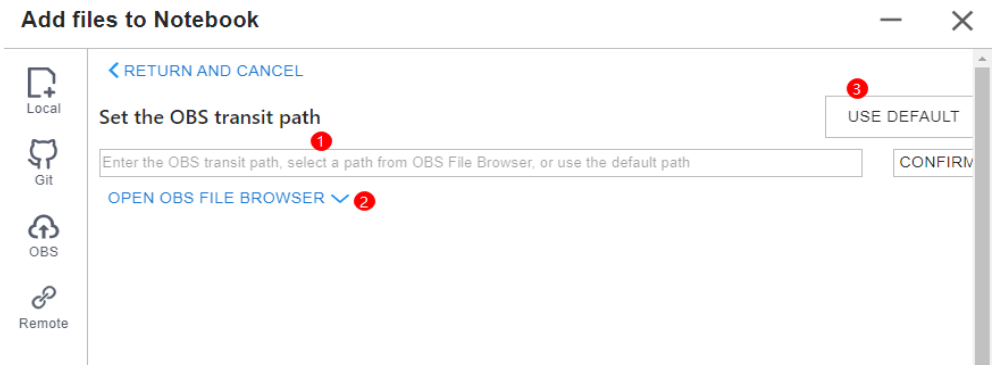
For example, in the scenario shown in the following figure, upload the file through OBS.

Figure 6-44 Uploading a large file through OBS



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

Figure 6-45 Uploading a file through OBS

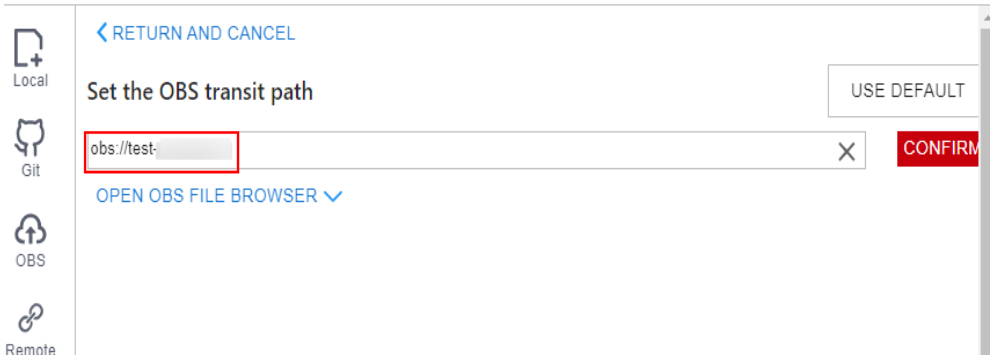


NOTE

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 6-49**.

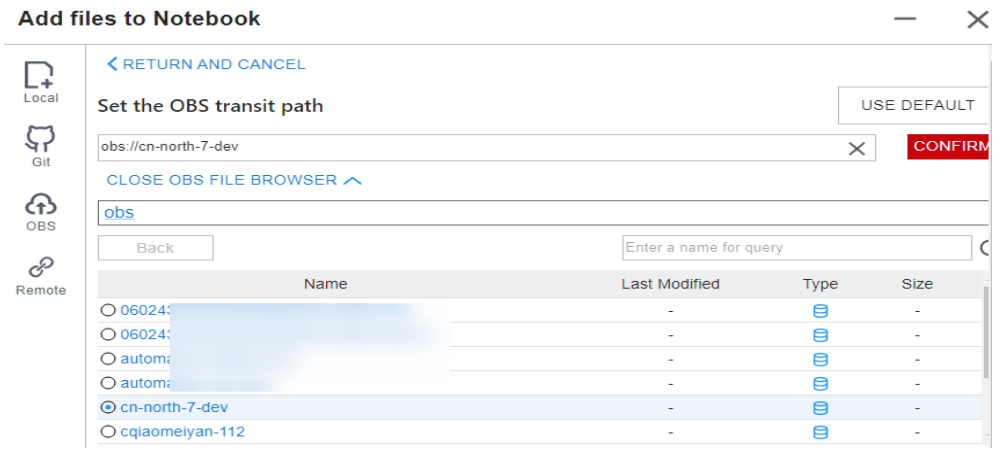
- Method 1: Enter a valid OBS path in the text box and click **OK**.

  Figure 6-46 Configuring an OBS path

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

Figure 6-47 OBS File Browser



- Method 3: Use the default path.

Figure 6-48 Using the default path to upload a file

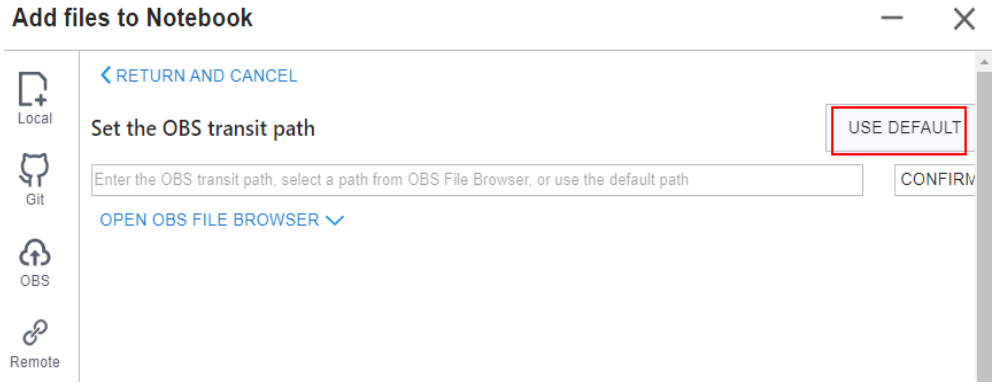
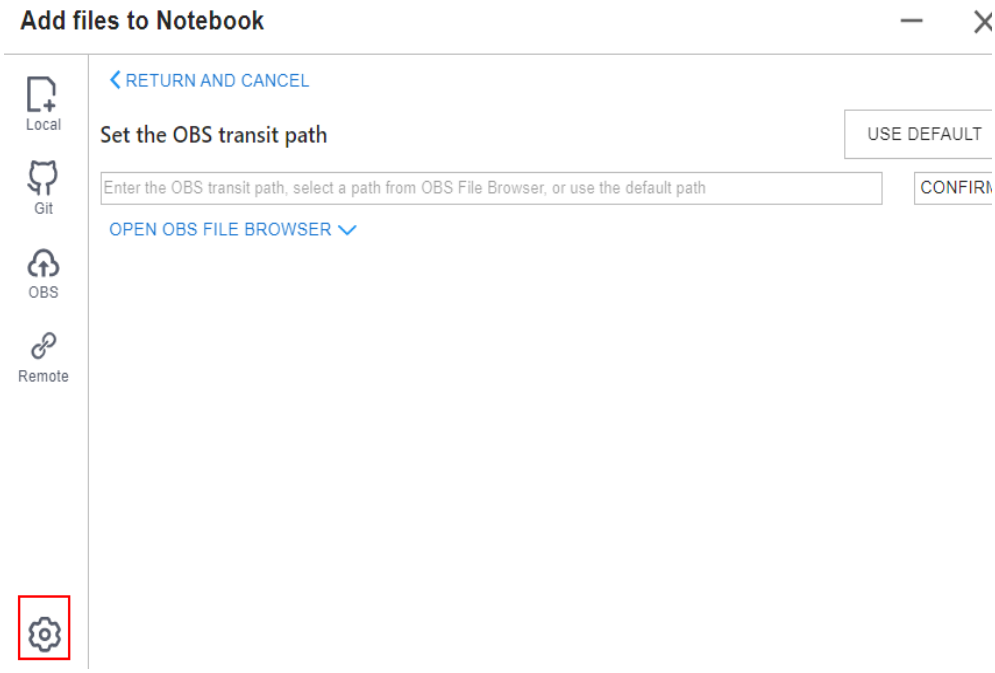
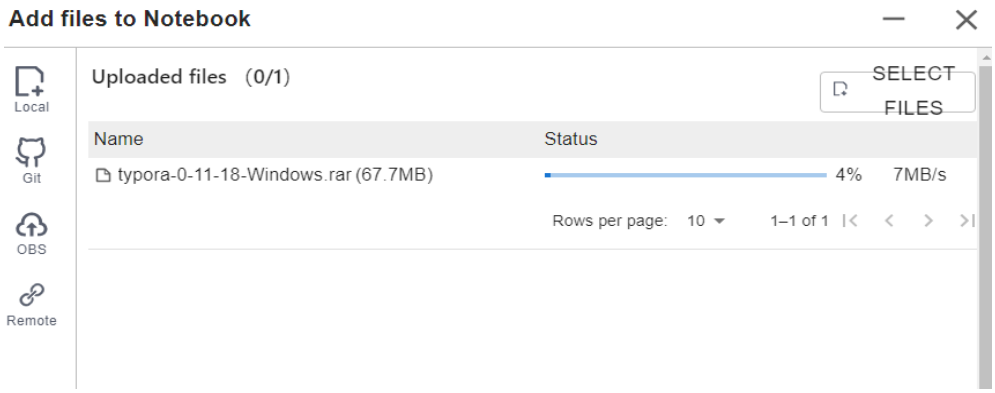


Figure 6-49 Setting an OBS path to upload a local file



After the OBS path is set, upload a file.

Figure 6-50 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.
```

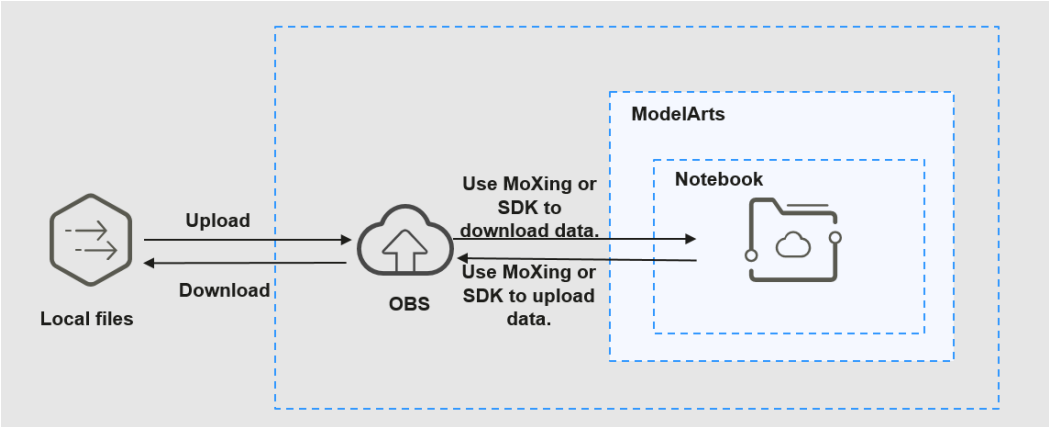
For more details, search for the decompression command in mainstream search engines.

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 6-51 Uploading and downloading large files in a notebook instance



The procedure is as follows:

1. Upload the file from a local path to OBS. For details, see **Uploading an Object**.

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: Call the ModelArts MoXing API for reading an OBS file.

     ```
     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 6-52** 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 6-53**.

Figure 6-52 Terminal

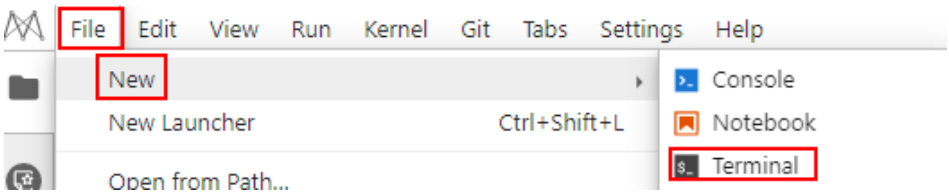
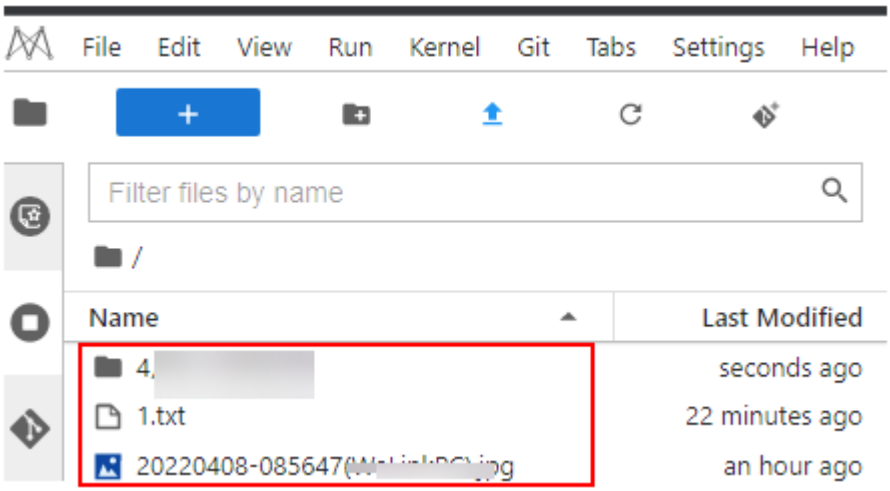


Figure 6-53 File downloaded to a notebook instance



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.

For details, see **Incorrect OBS Path on ModelArts**.

6.3.4.2 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 Git on the left to go to the page for cloning files from a GitHub open-source repository.

Figure 6-54 File upload icon

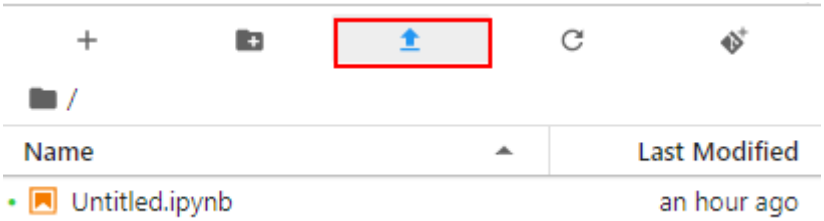
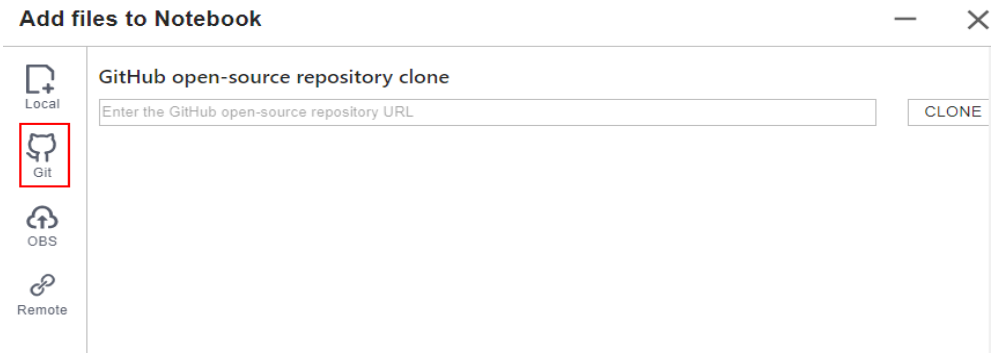


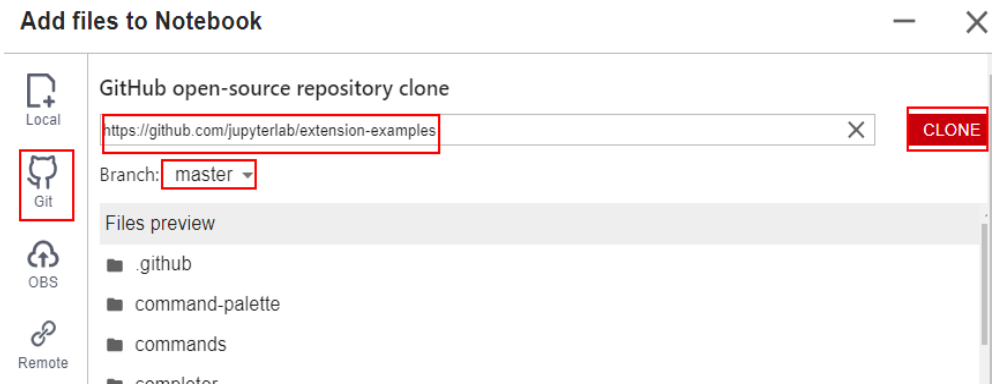
Figure 6-55 Page for cloning files from a GitHub open-source repository



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

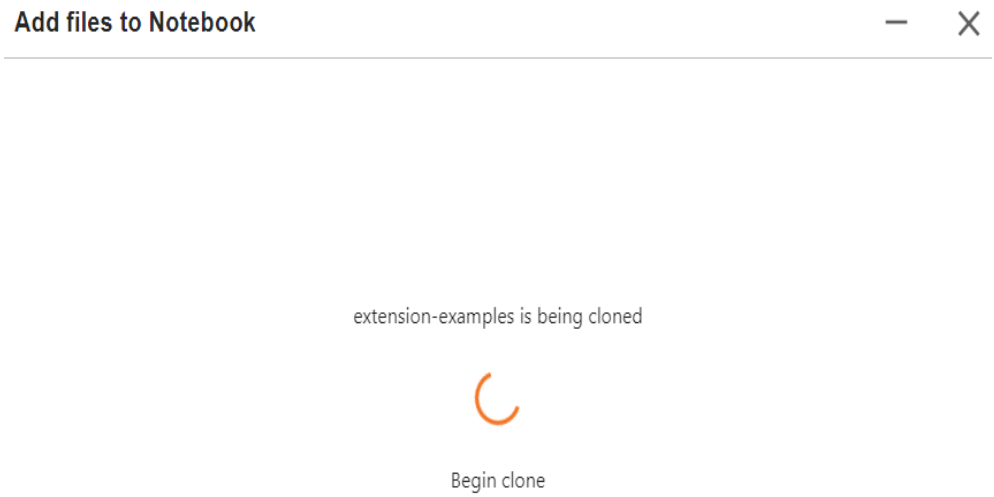
   GitHub open-source repository address: https://github.com/jupyterlab/extension-examples

Figure 6-56 Entering a valid address of a GitHub open-source repository



4. View the clone process.

Figure 6-57 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.

6.3.4.3 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.

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 OBS file upload page.

Figure 6-58 File upload icon

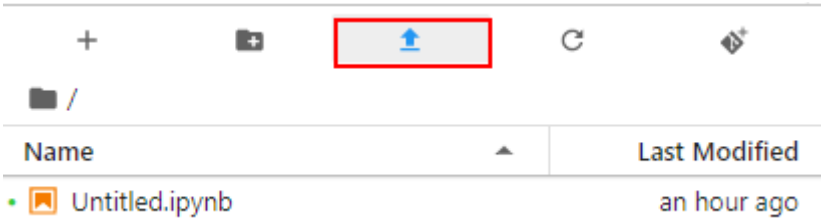
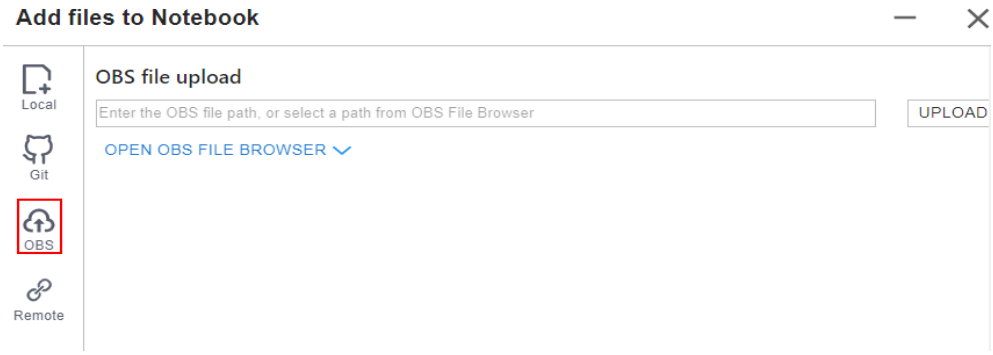
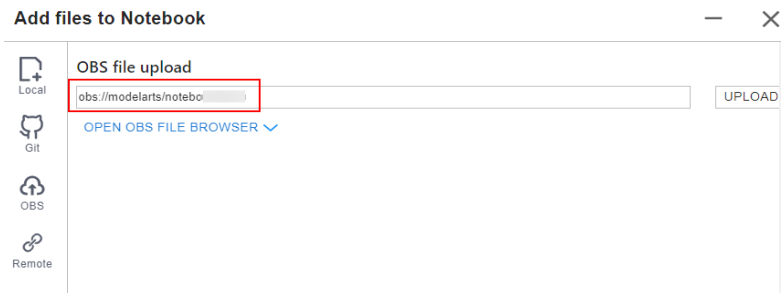


Figure 6-59 OBS file upload



3. 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 6-60 Entering a valid OBS file path

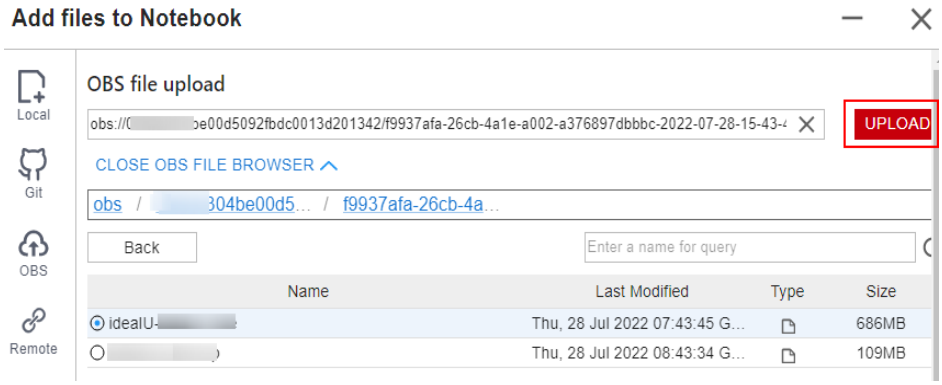
     

     NOTE

     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 6-61 Uploading an OBS File



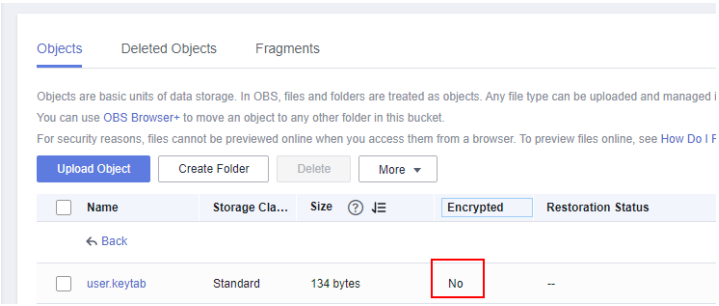
Error Handling

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

- **Scenario 1**

  Possible causes:

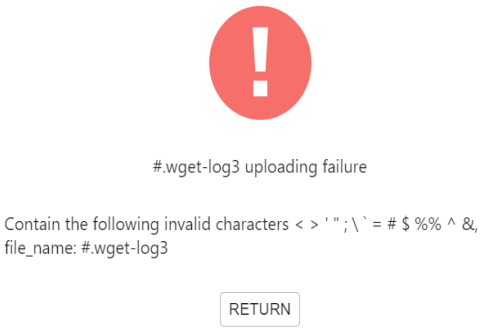
  - The OBS path is set to a folder instead of a file path.
  - The file in OBS is encrypted. In this case, go to the OBS console and ensure that the file is encrypted.

  - 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. For details, see **How Do I Check Whether ModelArts and an OBS Bucket Are in the Same 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. For details, see **Check Whether You Have Permission to Access 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 6-62 File uploading failure

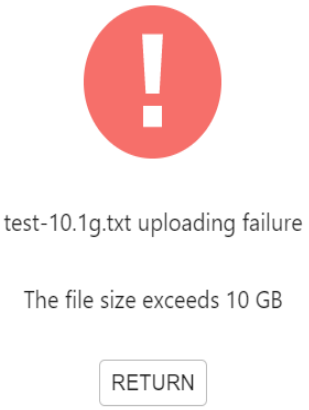


Possible causes:

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

- **Scenario 3**

Figure 6-63 File uploading failure



Possible causes:

The uploaded file exceeded 50 GB.

6.3.4.4 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 6-64 File upload icon

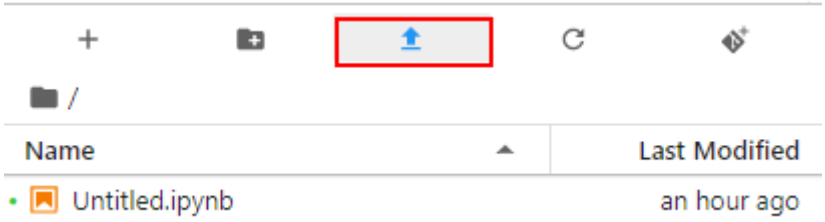
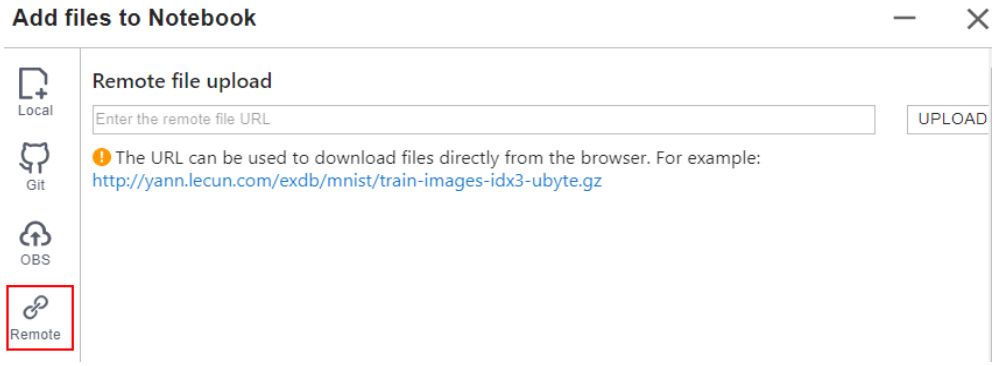
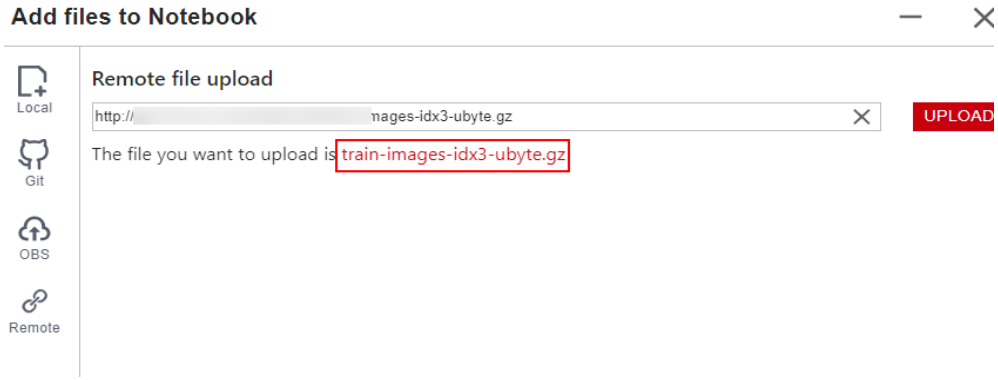


Figure 6-65 Remote file upload page



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

Figure 6-66 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.

6.3.5 Downloading a File from JupyterLab to a Local PC

Files created in JupyterLab can be downloaded to a local path.

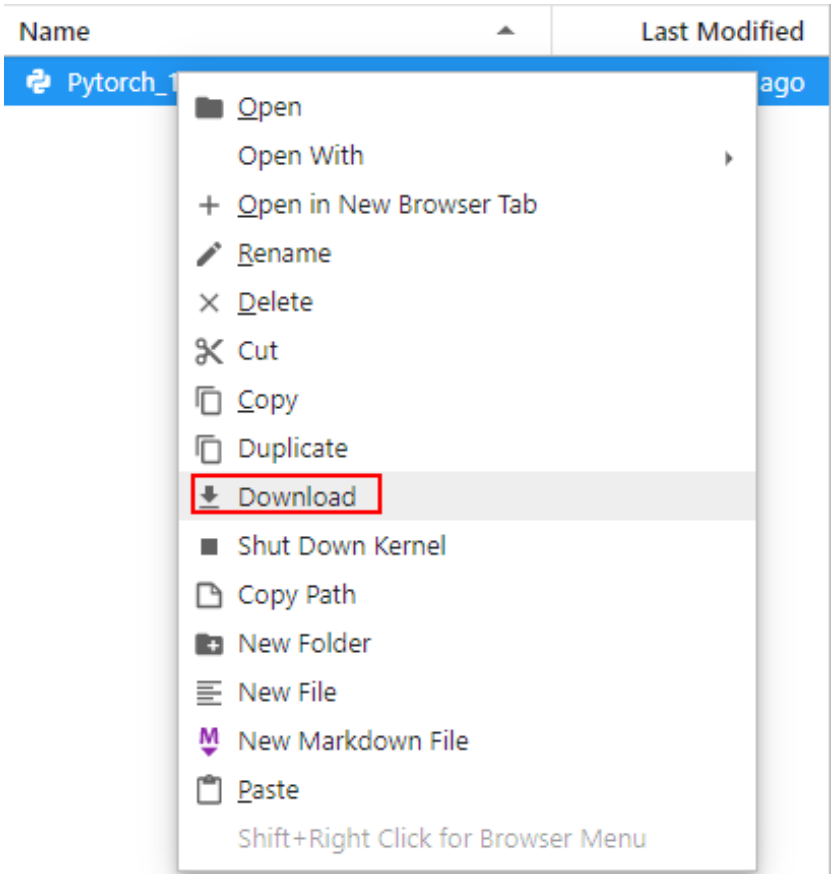
- If a file is less than or equal to 100 MB, directly download it from JupyterLab. For details, see **Downloading a File Less Than or Equal to 100 MB**.

- 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 6-67 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. In the notebook instance, create an IPYNB file larger than 100 MB and use MoXing to upload it to OBS. Example code is as follows:

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

   /home/ma-user/work/obs_file.txt is the path to the file stored in the notebook instance. **obs://bucket_name/obs_file.txt** is the path of the file uploaded to OBS, where **bucket_name** is the name of the bucket created in OBS, and **obs_file.txt** is the uploaded file.
2. 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 HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_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")
```

6.3.6 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.
- A running visualization job is not billed separately. When the target notebook instance is stopped, the billing stops.

- If the summary file is stored in OBS, OBS storage will be billed separately. After a job is complete, stop the notebook instance and clear OBS data to stop billing.

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 **Development Workspace** > **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 and ensure that the OBS parallel file system and ModelArts are in the same region. 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.

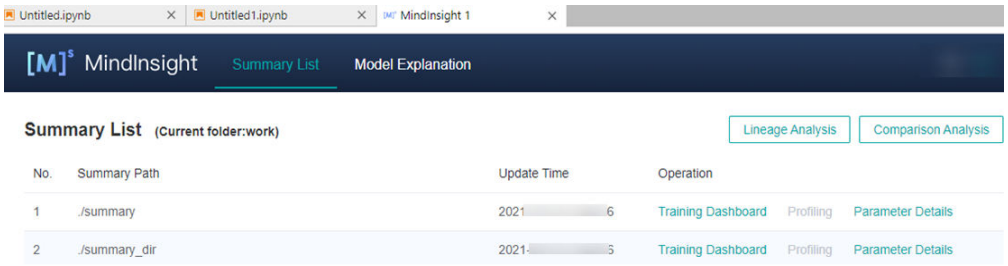
Click 

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 6-68 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 6-69 Stopping an instance

6.3.7 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**.

Precautions

- A running visualization job is not billed separately. When the target notebook instance is stopped, the billing stops.
- If the summary file is stored in OBS, you will be charged for the storage. After a job is complete, stop the notebook instance and clear OBS data to stop billing.

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 **Development Workspace** > **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.

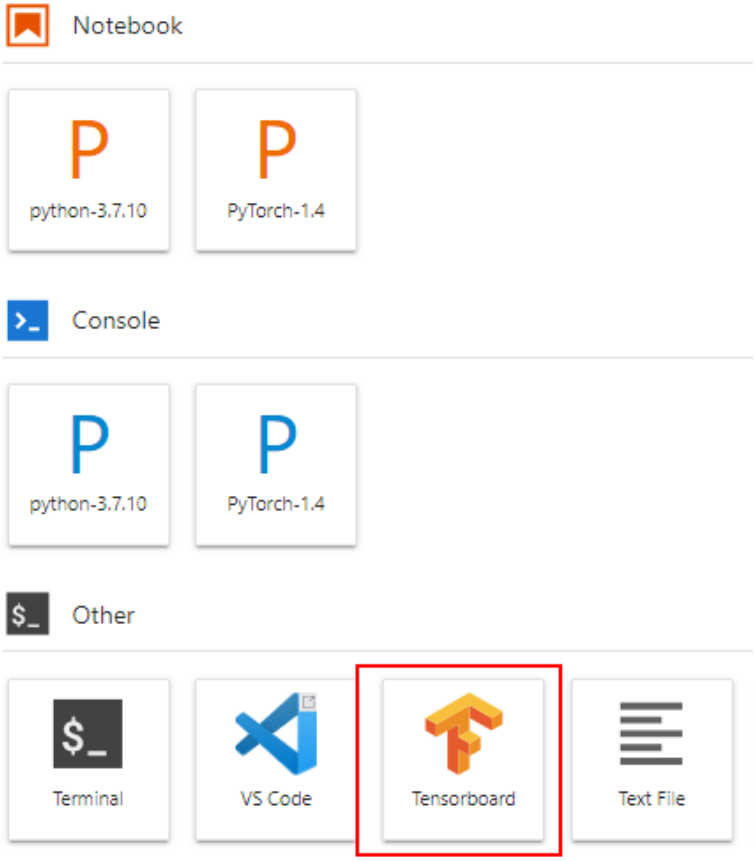
You can upload the summary data to the **/home/ma-user/work/** directory in the development environment or store it in the OBS parallel file system.

- 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 and ensure that the OBS parallel file system and ModelArts are in the same region. 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 6-70 Starting TensorBoard in JupyterLab

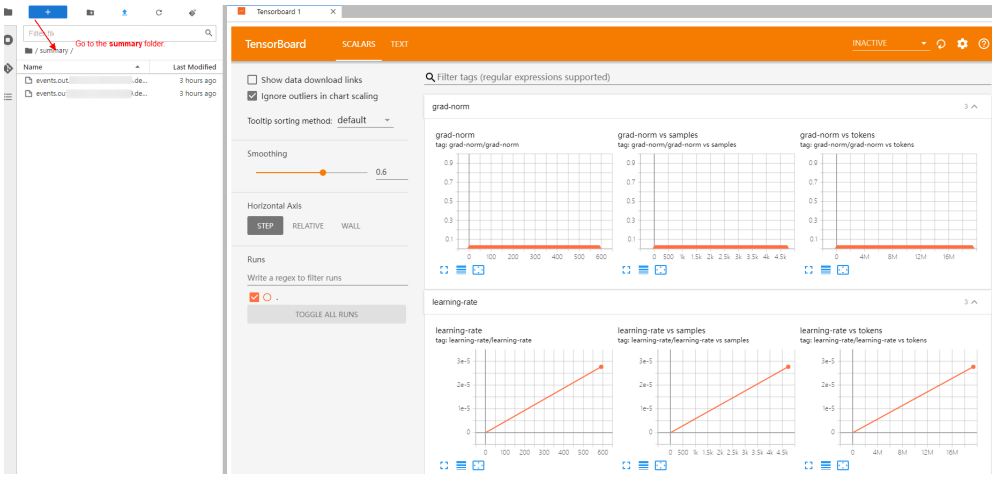


1. Open JupyterLab, in the navigation pane on the left, create the **summary** folder, and upload data to **/home/ma-user/work/summary**. The folder name must be **summary**.

2. Go to the **summary** folder and click 

to go to the TensorBoard page. See **Figure 6-71**.

Figure 6-71 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.

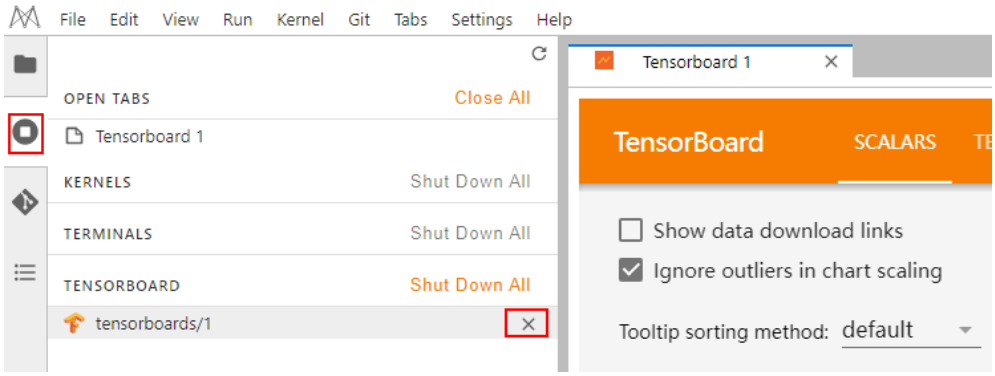
For more functions, see **Get started with TensorBoard**.

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 6-72 Clicking SHUT DOWN to stop an instance



6.4 Using Notebook Instances Remotely Through PyCharm

6.4.1 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 6-11 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

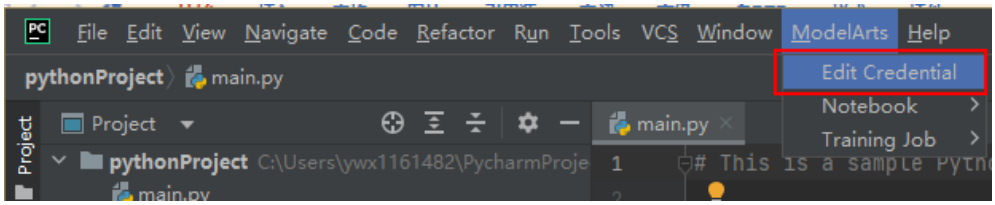
Create a notebook instance with remote SSH enabled and whitelist configured. Ensure that the instance is running. For details, see **Creating a Notebook Instance**.

Step 3 Logging In to the Plug-in

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 6-73 Edit Credential

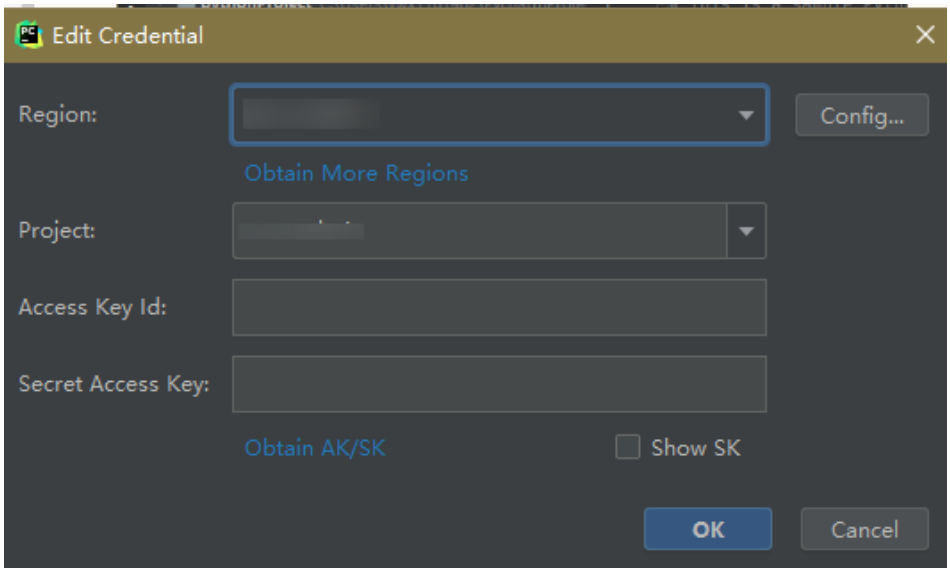
   

   NOTE

   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).

2. In the displayed dialog box, select the region where ModelArts is located, enter the AK and SK, and click **OK**. For details about how to obtain the AK/SK, see **How Do I Obtain an Access Key?**
   - **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.

Figure 6-74 Entering the region and access keys



3. View the verification result.

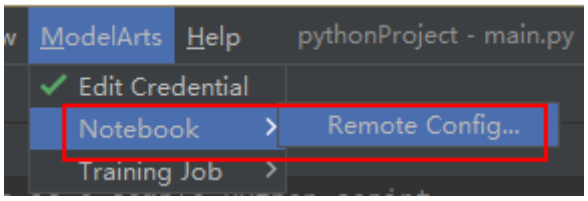
   In the **Event Log** area, if information similar to the following is displayed, the access key has been successfully added:

   ```
   16:01Validate Credential Success: The HUAWEI CLOUDcredential is valid.
   ```

Step 4 Automatically Configuring PyCharm Toolkit

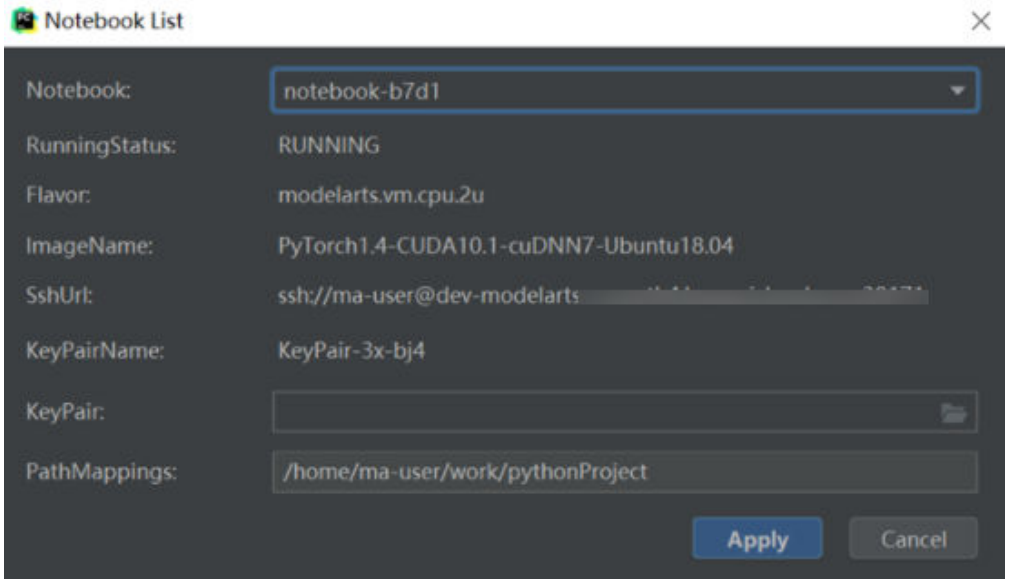
1. In the local PyCharm development environment, choose **ModelArts** > **Notebook** > **Remote Config...**, and configure PyCharm Toolkit.

   Figure 6-75 Remotely connecting to PyCharm Toolkit



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

Figure 6-76 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.

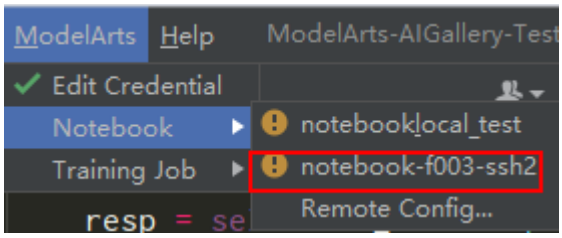
3. Click **Apply**. After the configuration is complete, restart the IDE for the configuration to take effect.

   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

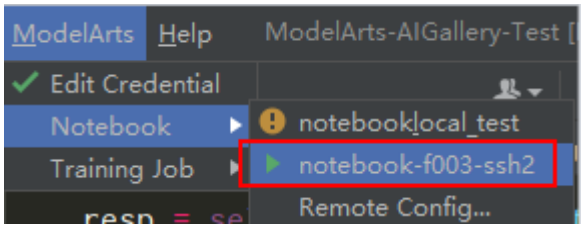
Click the notebook instance name and connect it to the local IDE as prompted. The connection is kept for 4 hours by default.

Figure 6-77 Starting the connection



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

Figure 6-78 Interrupting the connection



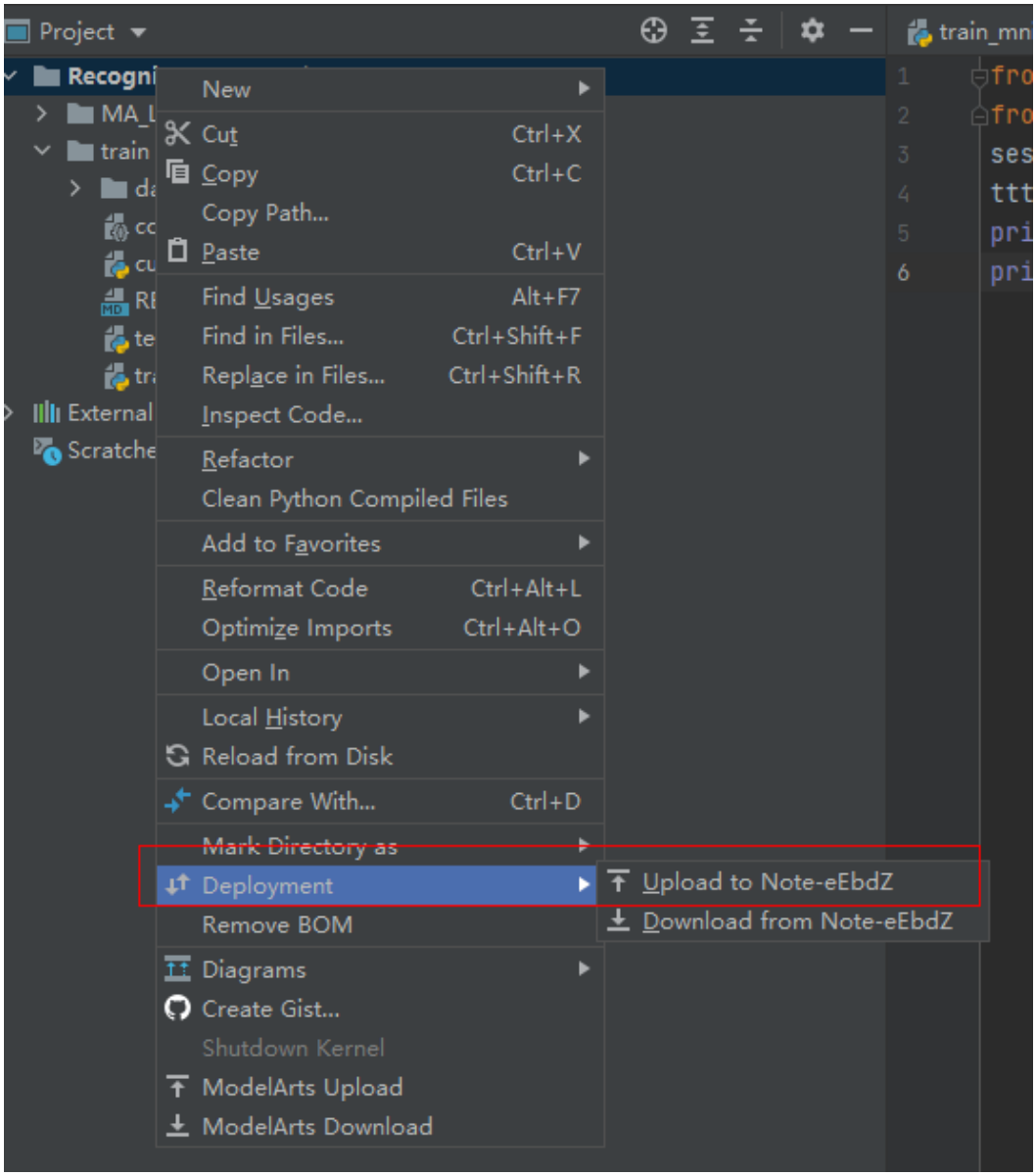
Step 6 Uploading Local Files to a Notebook Instance

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 6-79 Synchronizing local data to a notebook instance

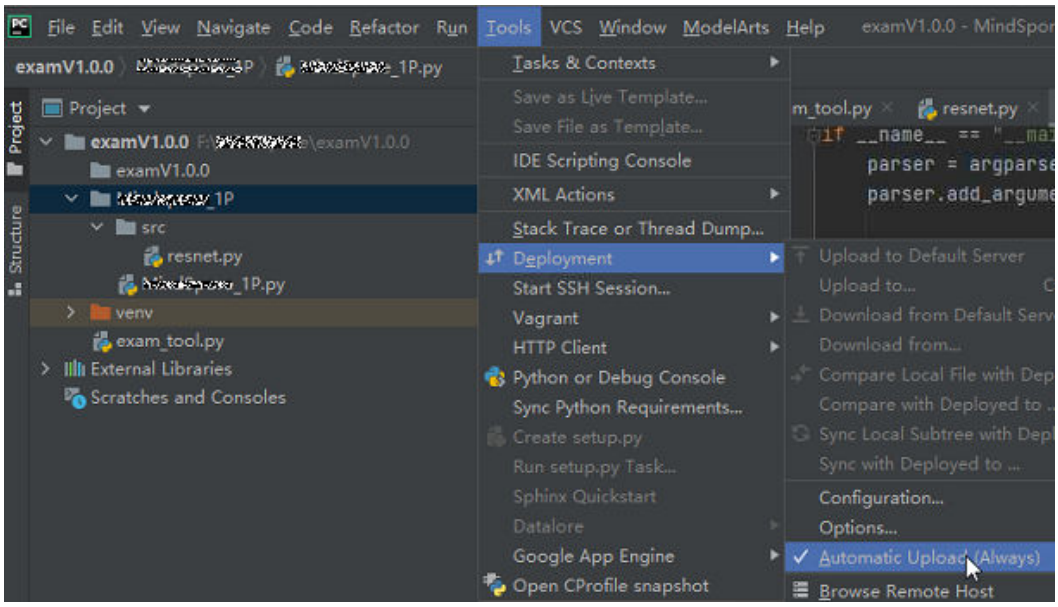


Follow-up synchronization

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.

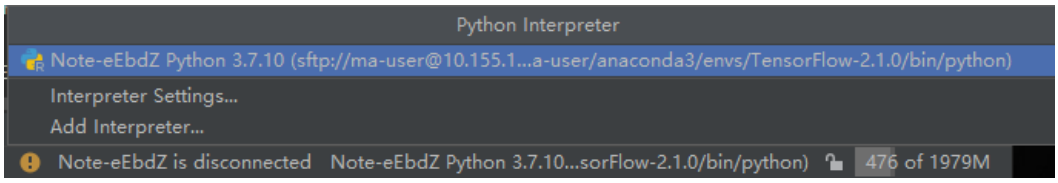
Figure 6-80 Enabling Automatic Upload



Step 7 Remotely Debugging the Code

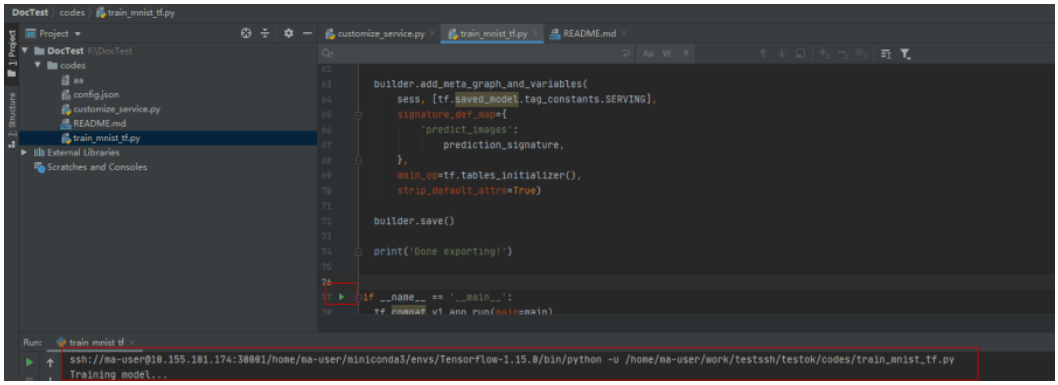
Click **Interpreter** in the lower right corner of the local IDE and select a notebook Python interpreter.

Figure 6-81 Selecting a Python interpreter



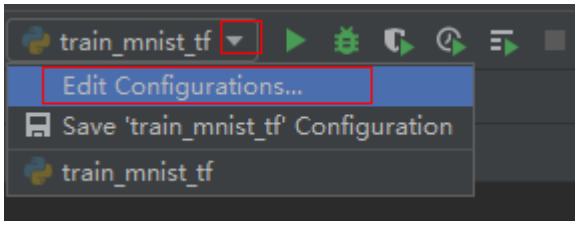
Run the code in the notebook instance. The logs are displayed locally.

Figure 6-82 Runtime logs



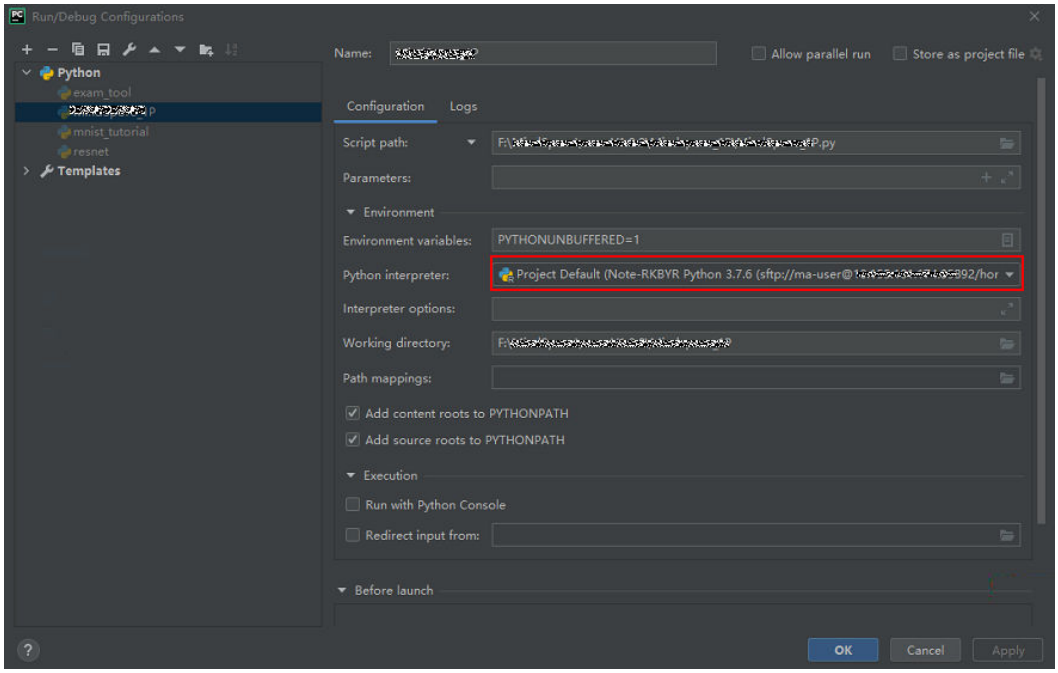
Click **Run/Debug Configurations** in the upper right corner of the local IDE to set runtime parameters.

Figure 6-83 Setting runtime parameters (1)



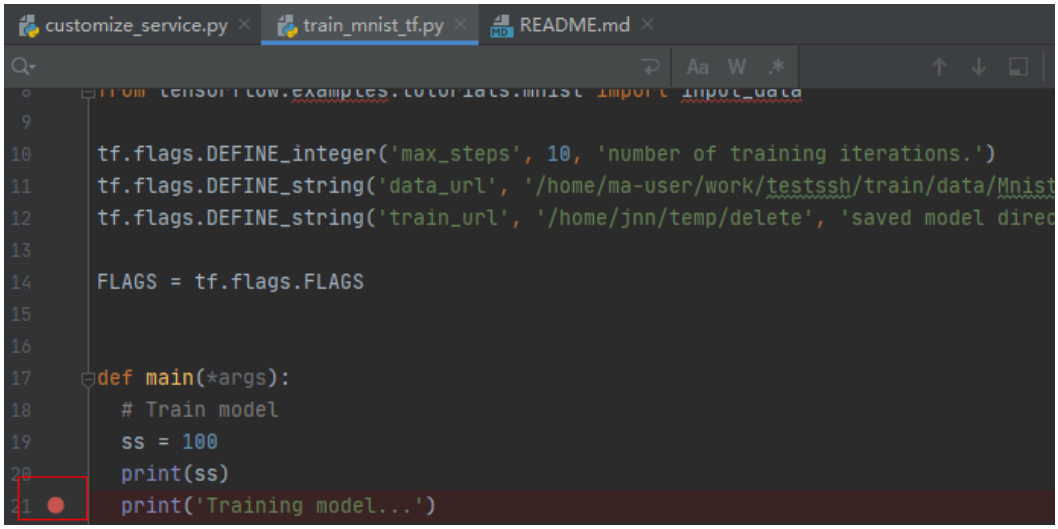
Select the Python interpreter that remotely connects to the target notebook instance.

Figure 6-84 Setting runtime parameters (2)



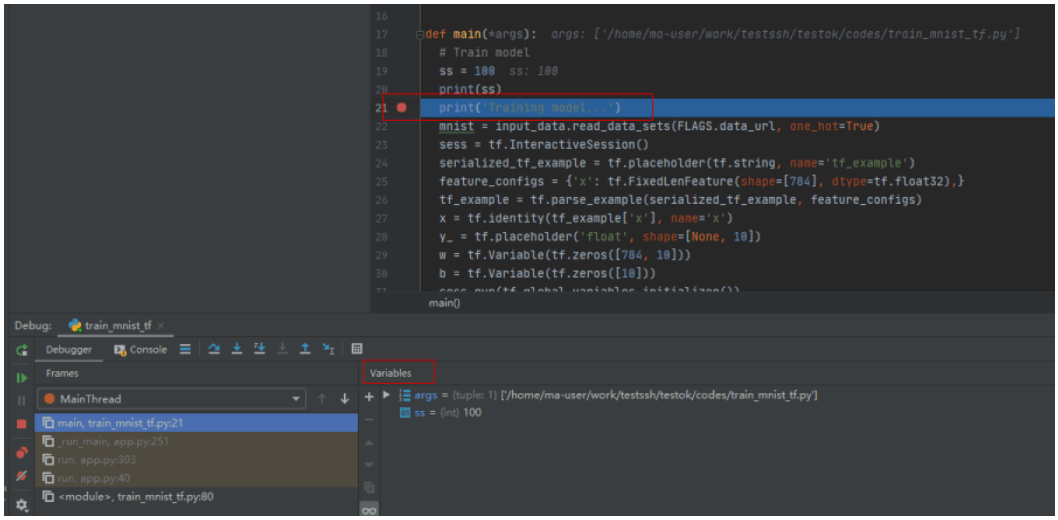
To debug code, set breakpoints and run the program in debug mode.

Figure 6-85 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 6-86 Viewing variable values in debug mode



6.4.2 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 6-87 Instance details page

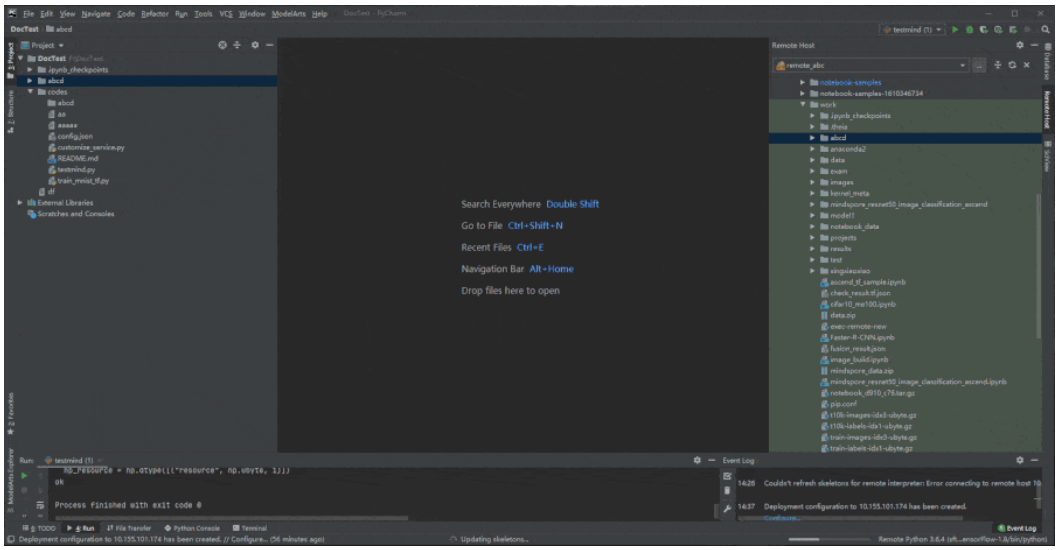
- The key pair is available.

  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 6-88 Configuring SSH



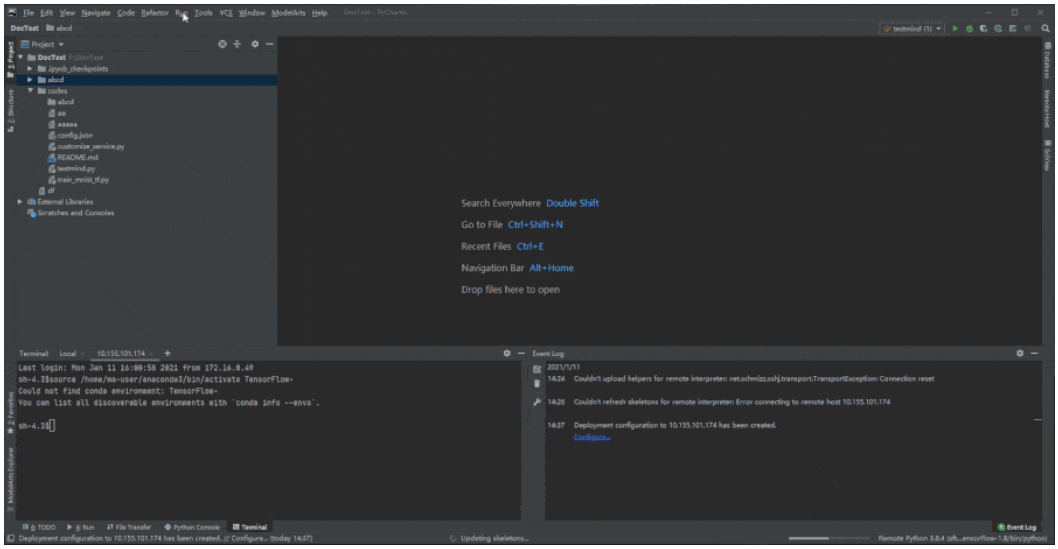
Step 2 Obtain the Path to the Virtual Environment Built in the Development Environment

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 6-89 Obtaining the path to the virtual environment built in the development environment



Step 3 Configure a Python Interpreter

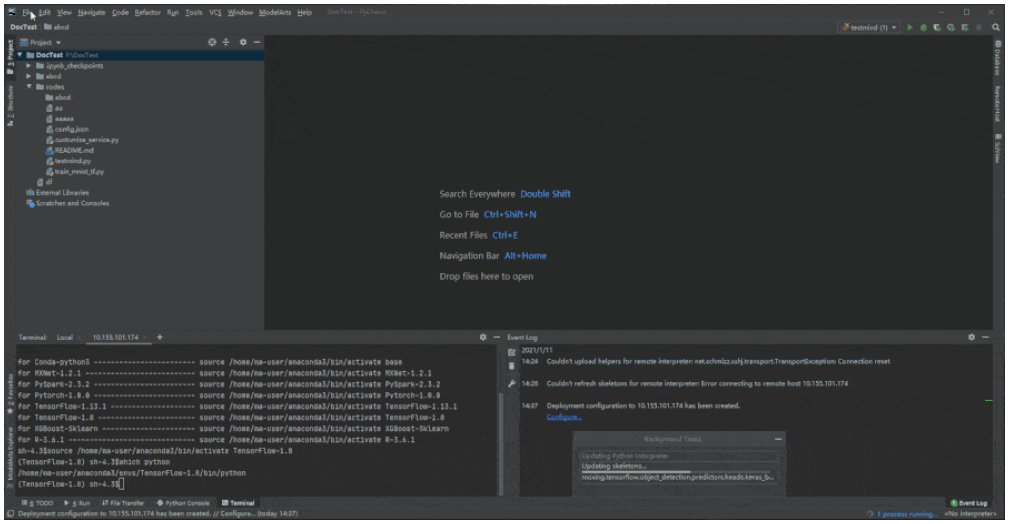
1. Choose **File** > **Settings** > **Project:** *Python project* > **Python Interpreter**. Then, click  and **Add** to add an interpreter.
2. Select **Existing server configuration**, choose the SSH configuration from the drop-down list, and click **Next**.
3. Configure the Python interpreter.
   - **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.

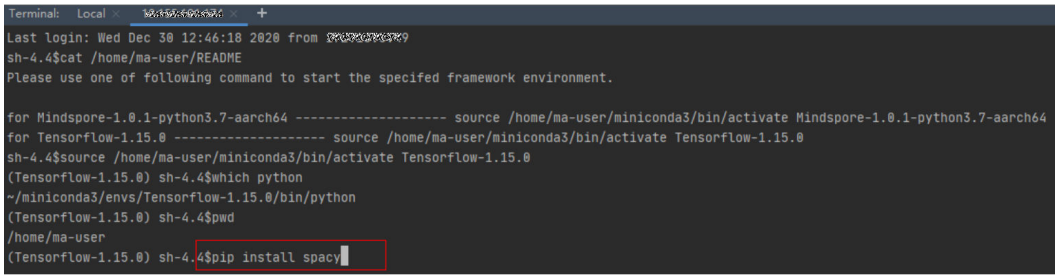
Figure 6-90 Configuring a Python interpreter



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.

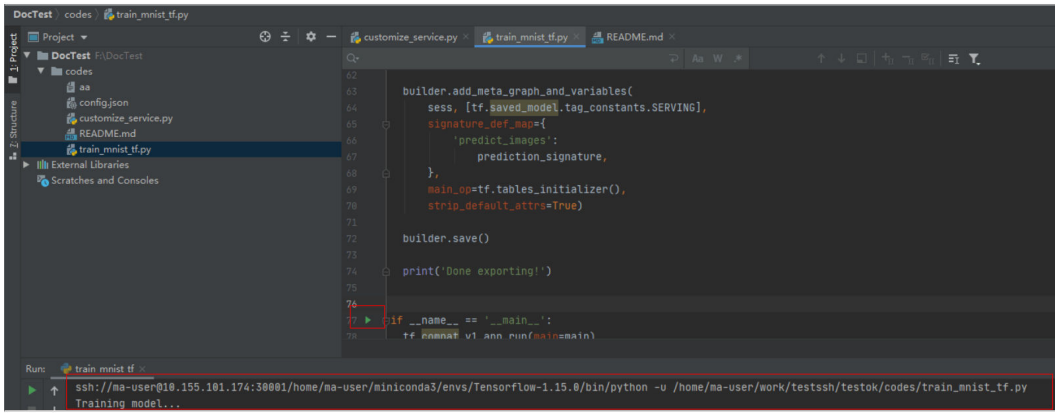


Step 5 Debug Code in the Development Environment

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.

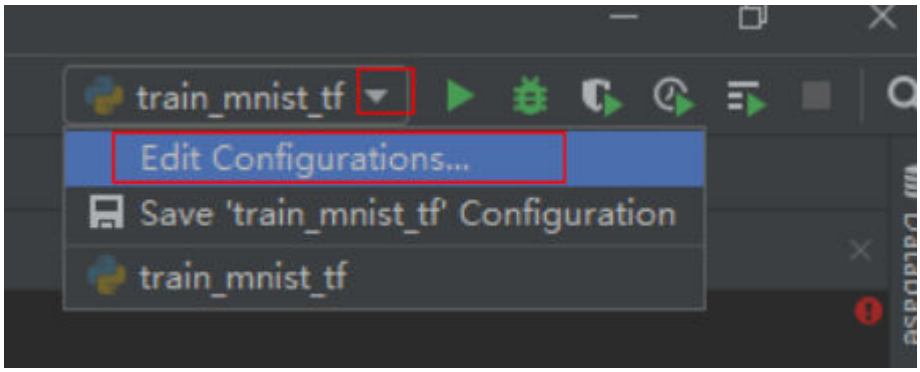
Run the code in the local IDE. The logs can be displayed locally.

Figure 6-91 Debugging code



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

Figure 6-92 Setting runtime parameters



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

Figure 6-93 Code breakpoint

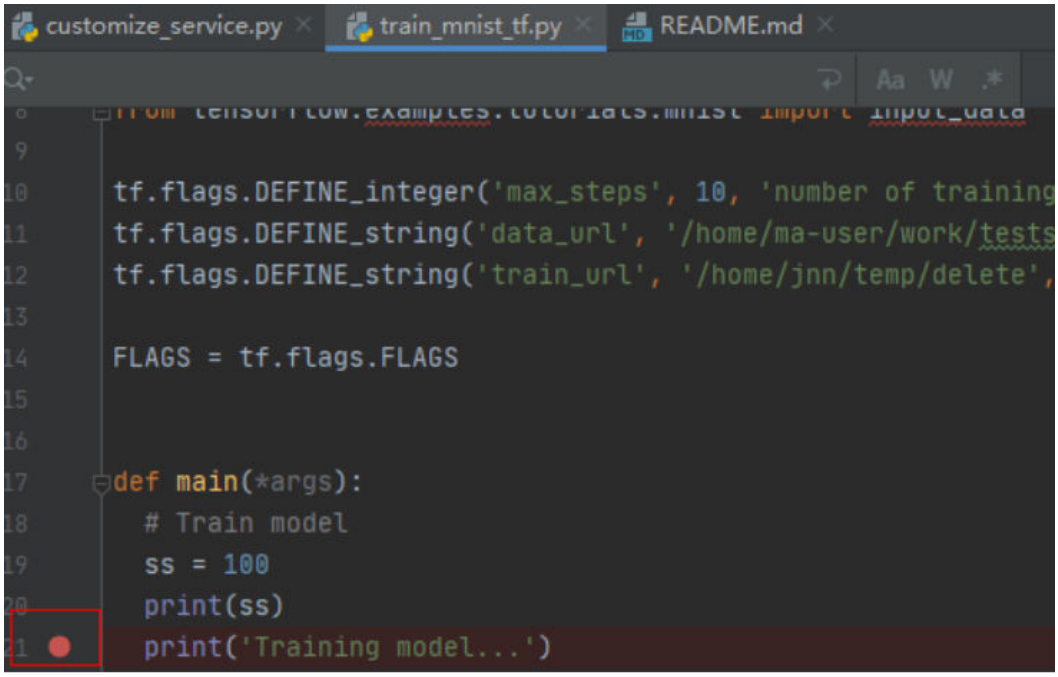
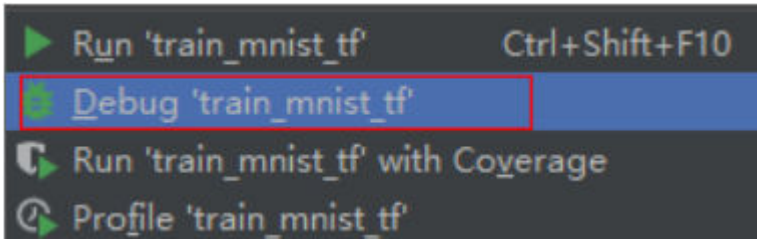
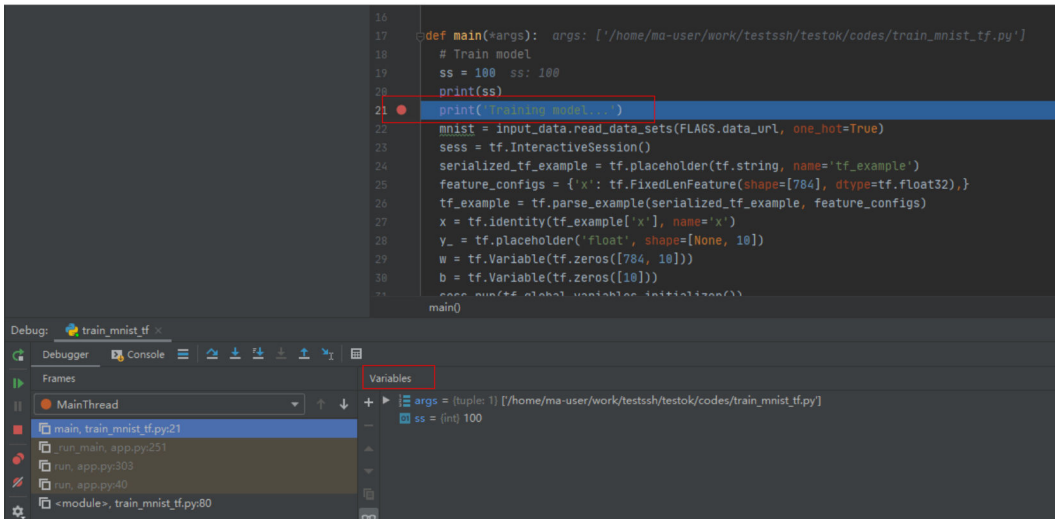


Figure 6-94 Debugging in debug mode



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

Figure 6-95 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.

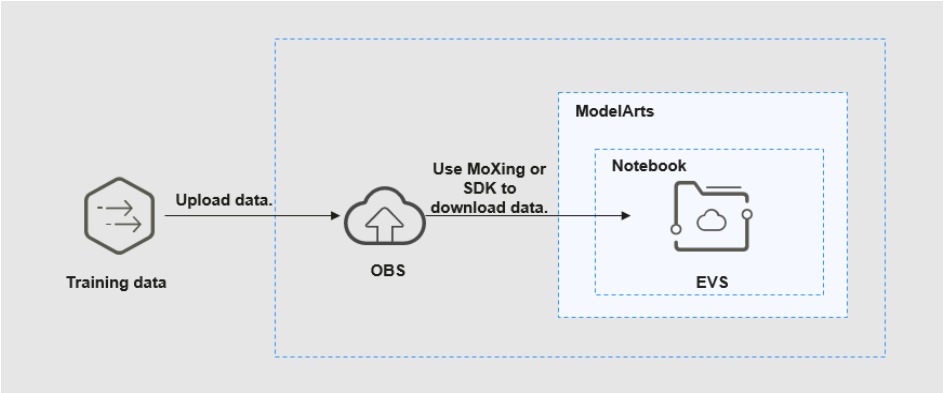
When configuring a Python interpreter in the cloud development environment, select **Automatically upload** so that any local file modification can be automatically uploaded to the cloud. If you do not select **Automatically upload**, manually upload the directory or code after you modify the local code.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

6.4.3 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.

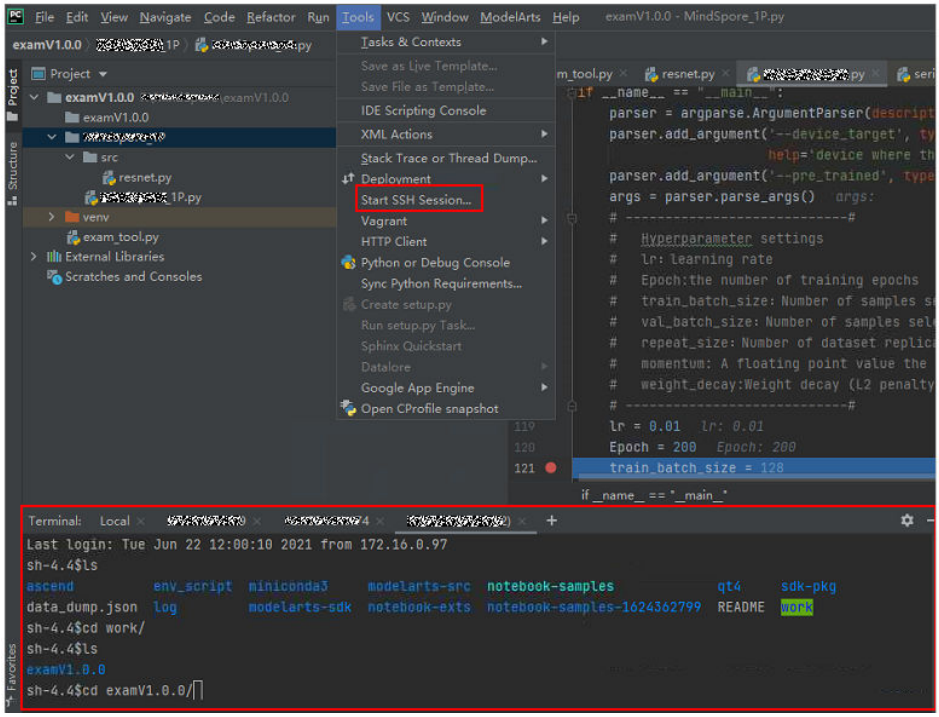
If the data is larger than 500 MB, upload it to OBS, and then download it to the notebook instance.

Figure 6-96 Uploading data to a notebook Instance through OBS



1. Upload data to OBS. For details, see **Uploading an Object**.
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 6-97 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')
```

6.5 Using Notebook Instances Remotely Through VS Code

6.5.1 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**

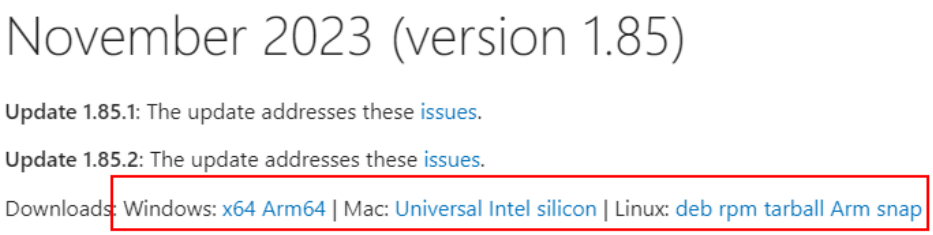
In this mode, use the VS Code Remote-SSH plug-in to configure connection information and connect to an instance.

Installing VS Code

Install VS Code first.

- **Download URL**:
  Download address: **https://code.visualstudio.com/updates/v1_85**

  Figure 6-98 VS Code download URL

- **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 Windows, double-click the installation package to complete the installation.
  In Linux, run the command **sudo dpkg -i code_1.85.2-1705561292_amd64.deb** to install VS Code.

  NOTE

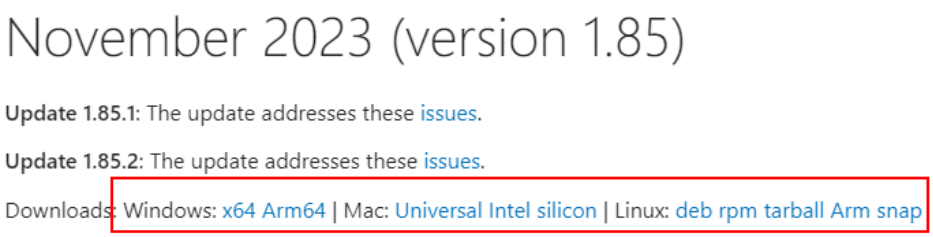
  Linux system users must install VS Code as a non-root user.

6.5.2 Installing VS Code

Download URL:

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

Figure 6-99 VS Code download URL



VS Code version requirements:

Use VS Code 1.85.2 for remote connection.

VS Code installation guide:

In Windows, double-click the installation package to complete the installation.

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.

6.5.3 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.

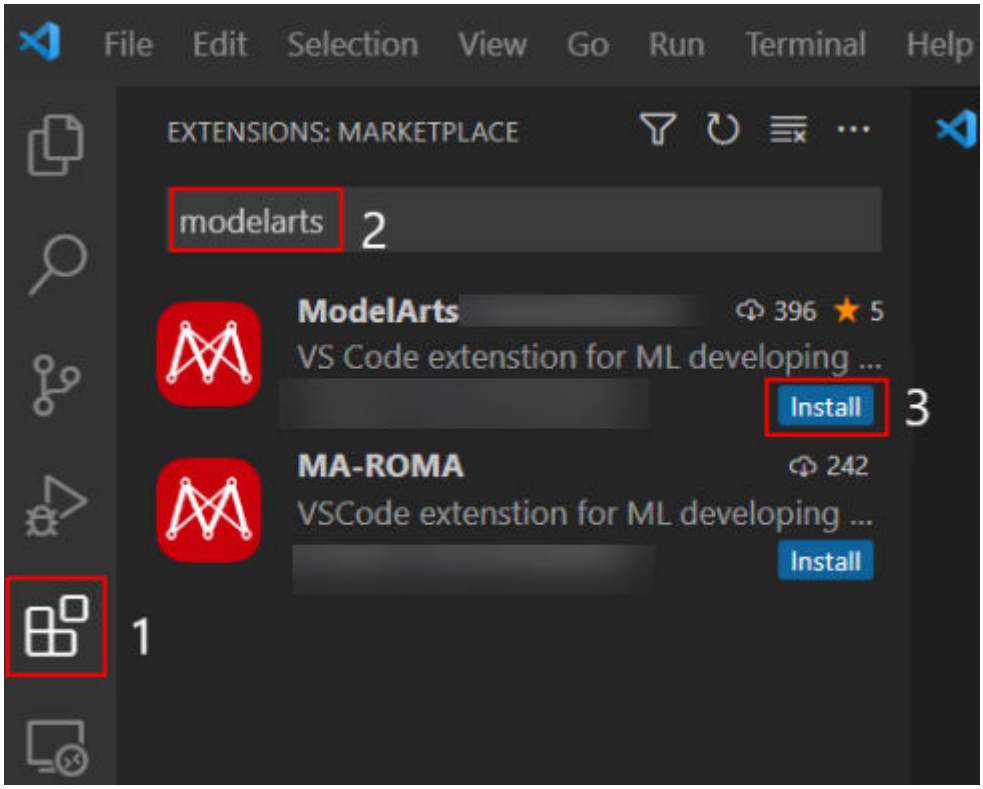
Prerequisites

You have downloaded and installed VS Code. For details, see **Installing VS Code**.

Step 1 Install the VS Code Plug-in

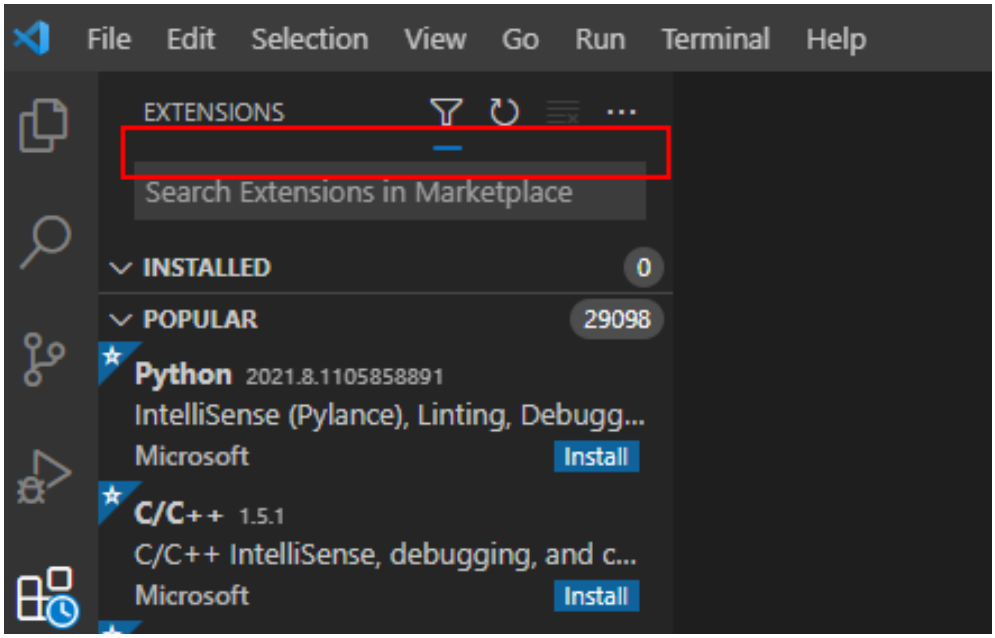
1. Search for **ModelArts-HuaweiCloud** in the **EXTENSIONS** text box and click **Install**.

   Figure 6-100 Installing the VS Code plug-in



2. Wait for about 1 to 2 minutes.

Figure 6-101 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 6-102 Installation completion message

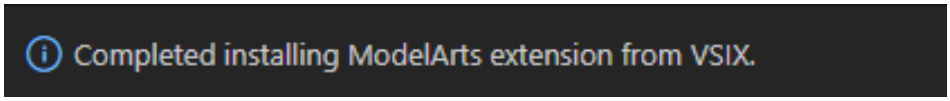
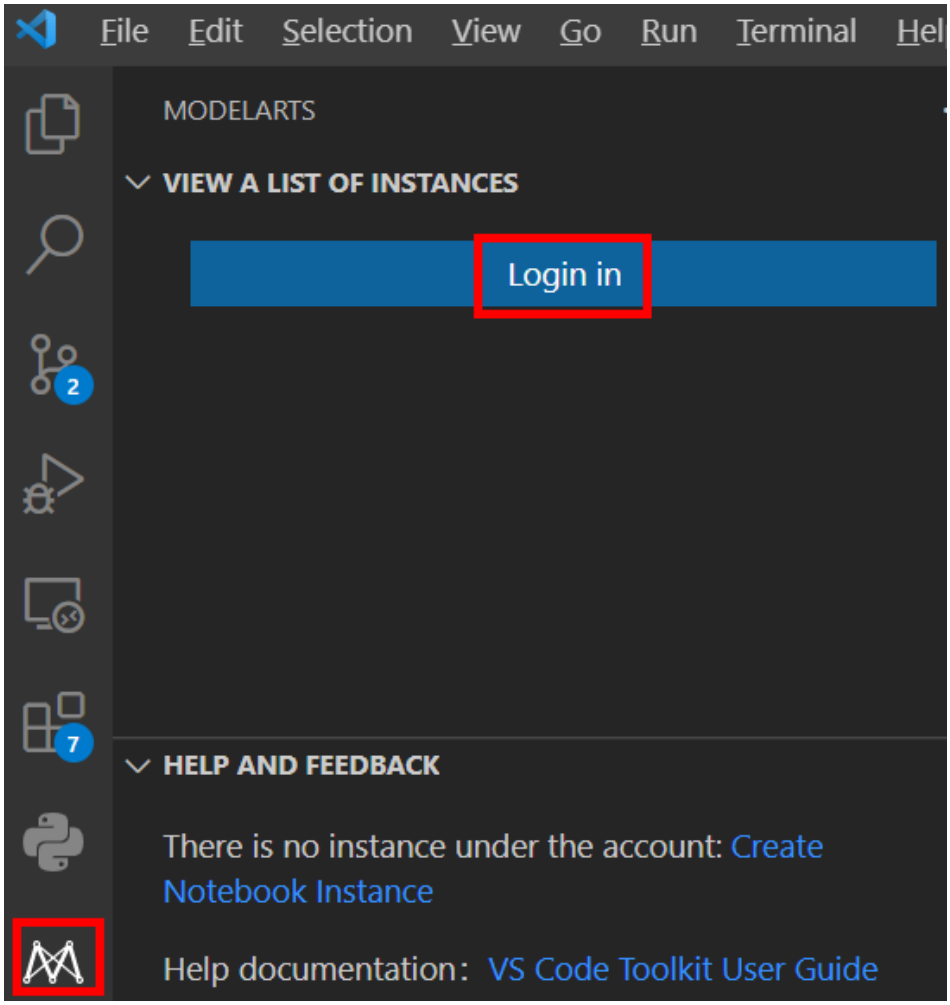
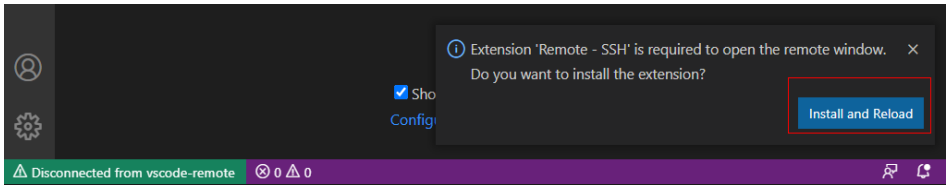


Figure 6-103 Installation completed



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**.

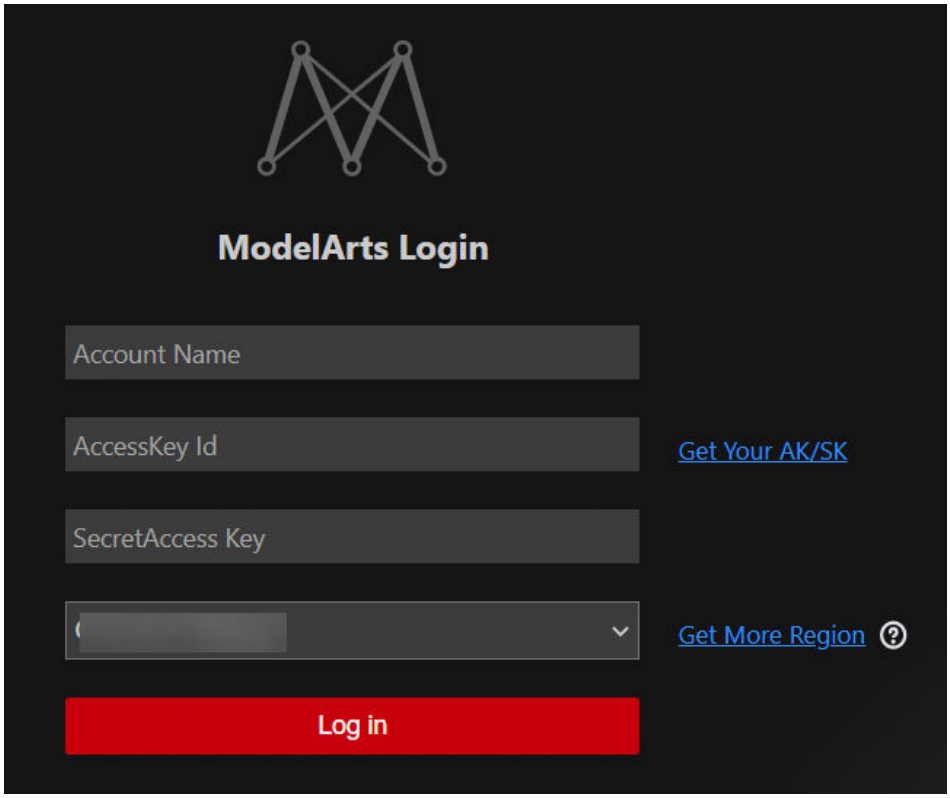
Figure 6-104 Reconnecting remote SSH



Step 2 Log In to the VS Code Plug-in

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

Figure 6-105 Logging in to the plug-in



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

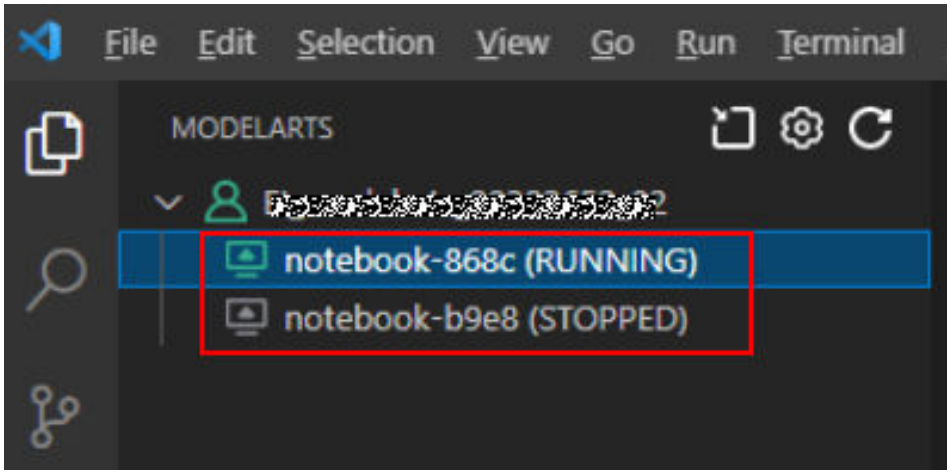
- **Name**: Custom username, which is displayed only on the VS Code page and is not associated with any Huawei Cloud account.
- AK and SK: Access key pair. To create a key pair, choose **My Credentials** > **API Credentials** > **Access Keys**, and click **Create Access Key**. For details, see **How Do I Obtain an 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 6-106 Login succeeded



Step 3 Create a Notebook Instance

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.

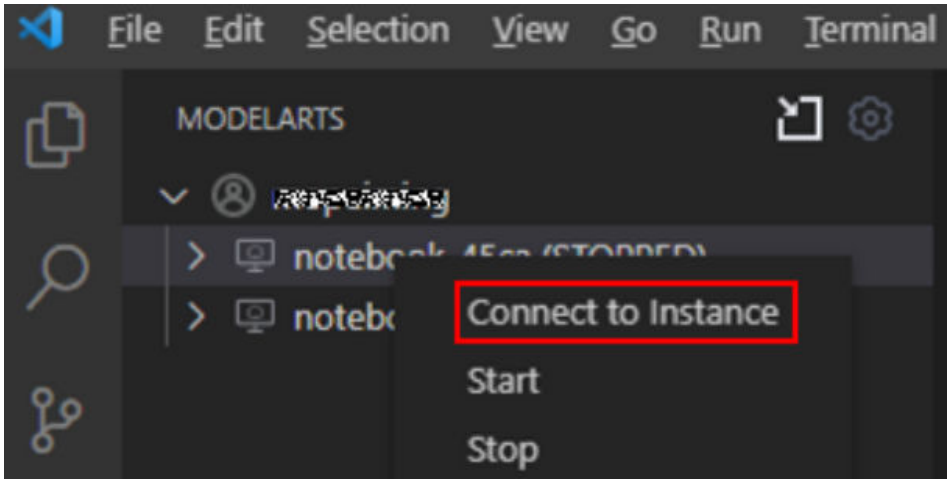
Create a notebook instance with remote SSH enabled. For details, see **Creating a Notebook Instance**.

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.

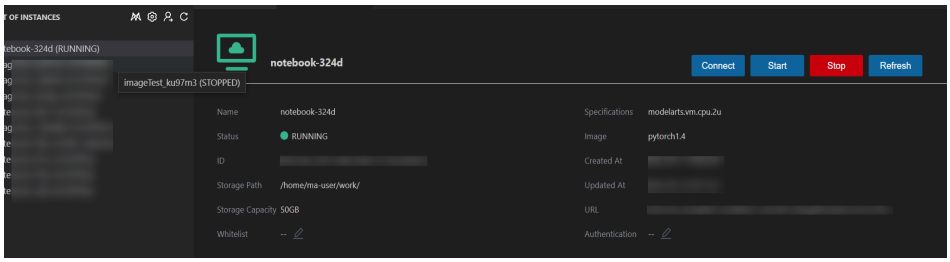
   The notebook instance can either be running or stopped. If it is stopped, the VS Code plug-in starts the instance and then connects to it.

Figure 6-107 Connecting to a notebook instance



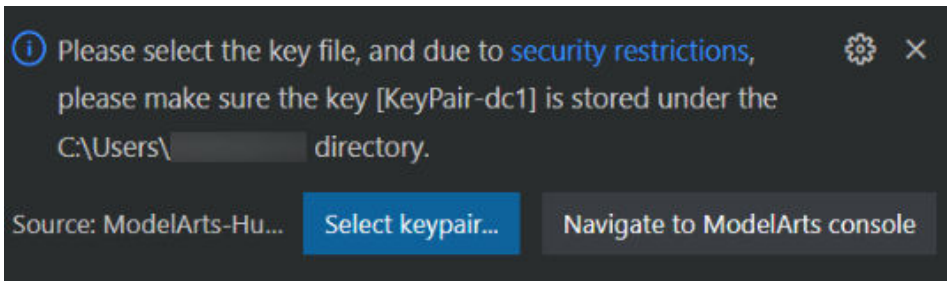
Alternatively, click the instance name. On the instance details page, click **Connect**. Then, the system automatically starts and connects to the notebook instance.

Figure 6-108 Viewing details about a notebook instance



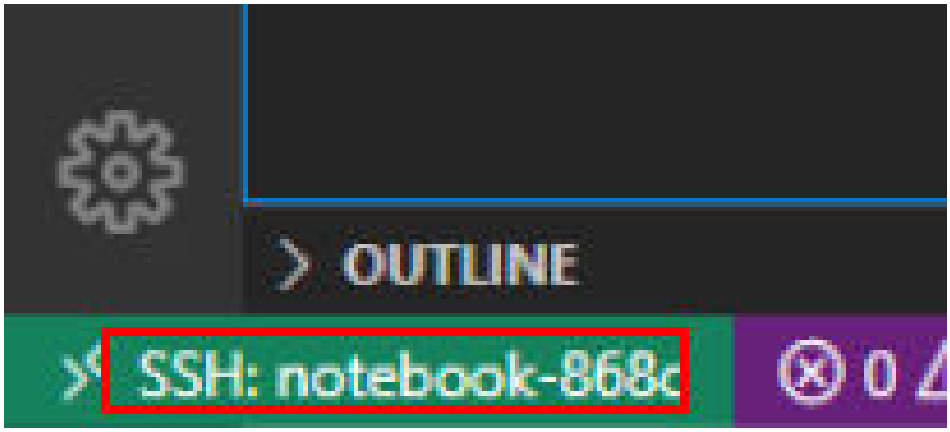
2. When you connect to a notebook instance for the first time, the system prompts you in the lower right corner to configure the key file. In this case, select the local .pem key file and click **OK**.

Figure 6-109 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 6-110 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 6-111 Open Folder

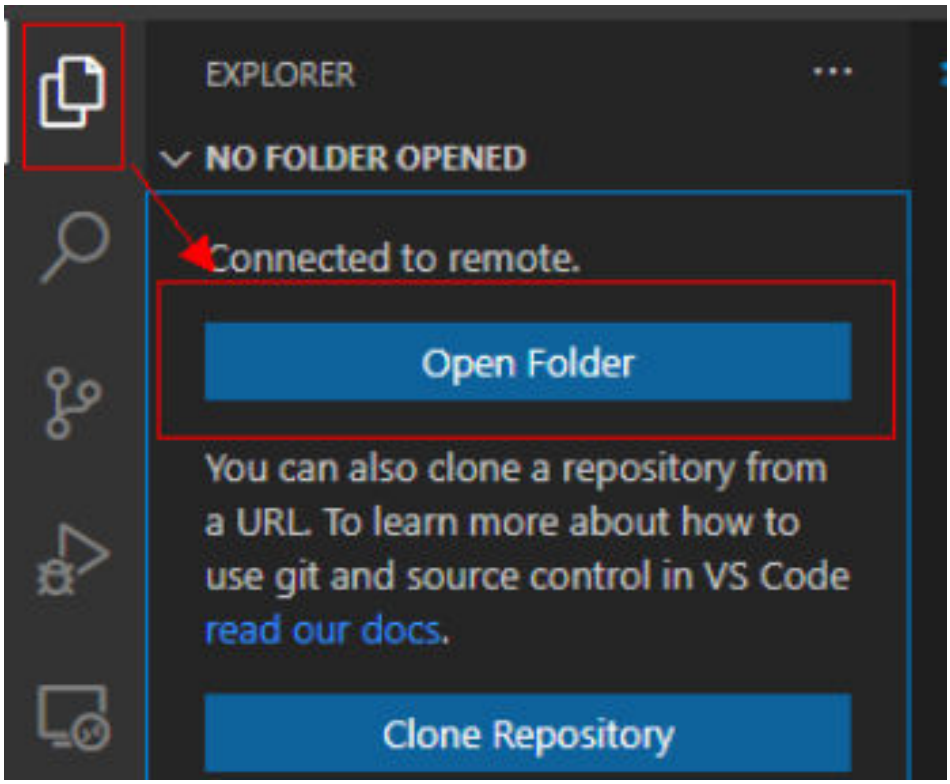
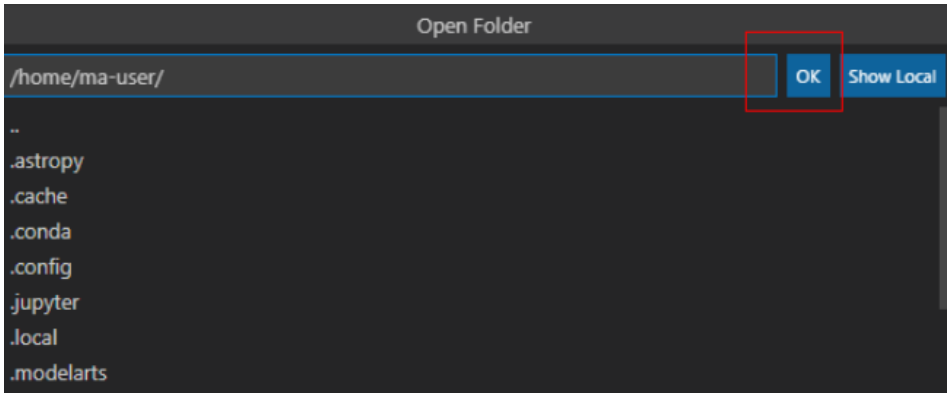


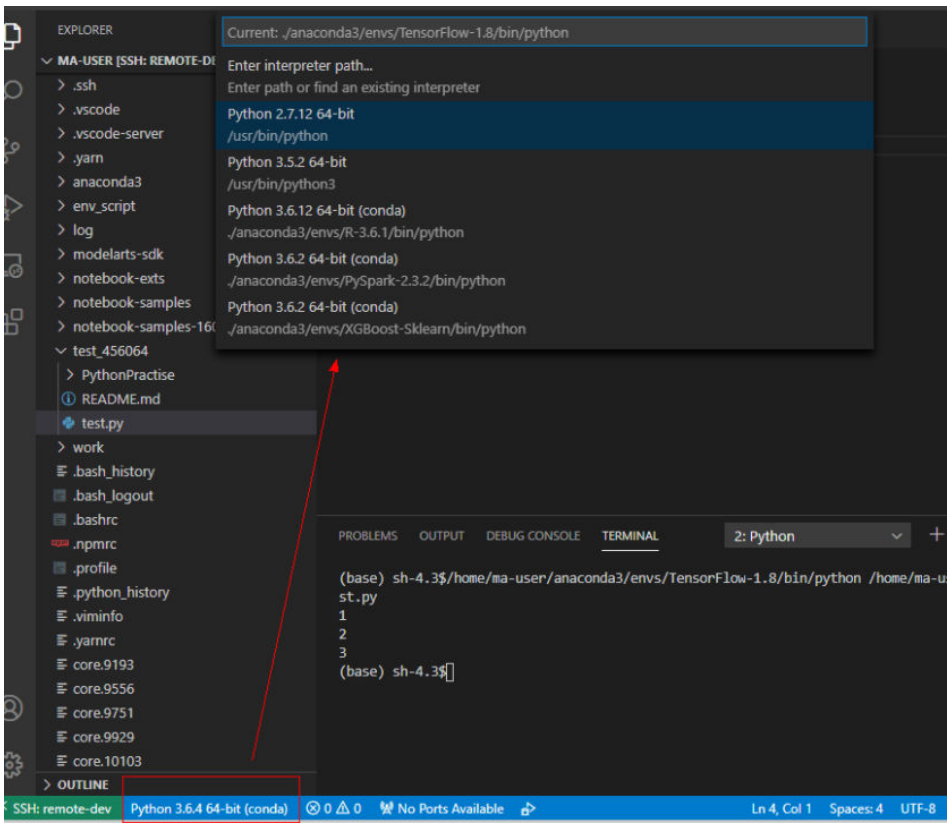
Figure 6-112 Selecting a file path



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 6-113 Selecting a Python version



2. Click the execution button to run the code. The code output is shown on the **TERMINAL** tab page.
   - 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:

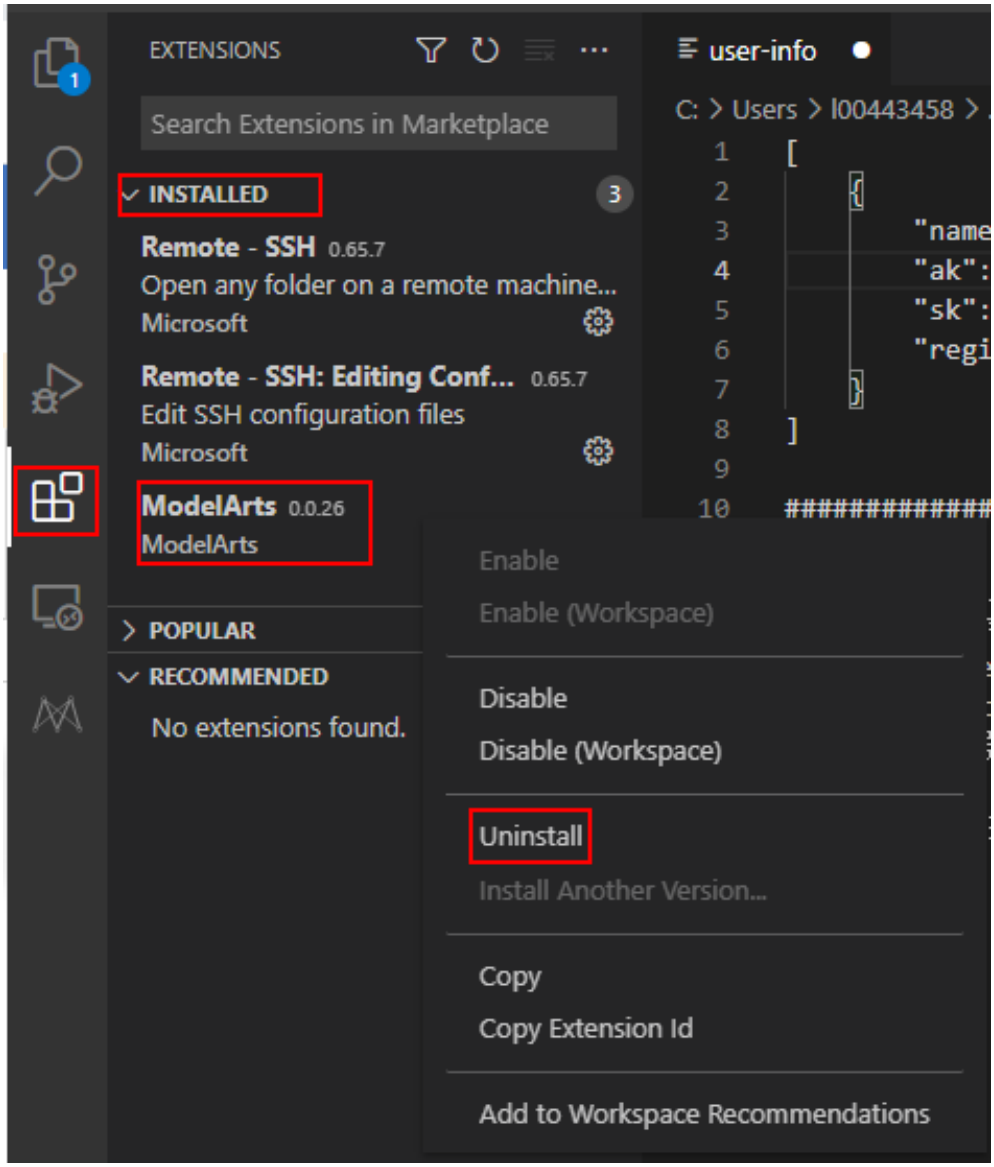
```
nohup your_train_job.sh > output.log 2>&1  &  tail -f  output.log
```

- 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 6-114**.

Figure 6-114 Uninstalling the VS Code plug-in



6.5.4 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 6-115 Instance details page

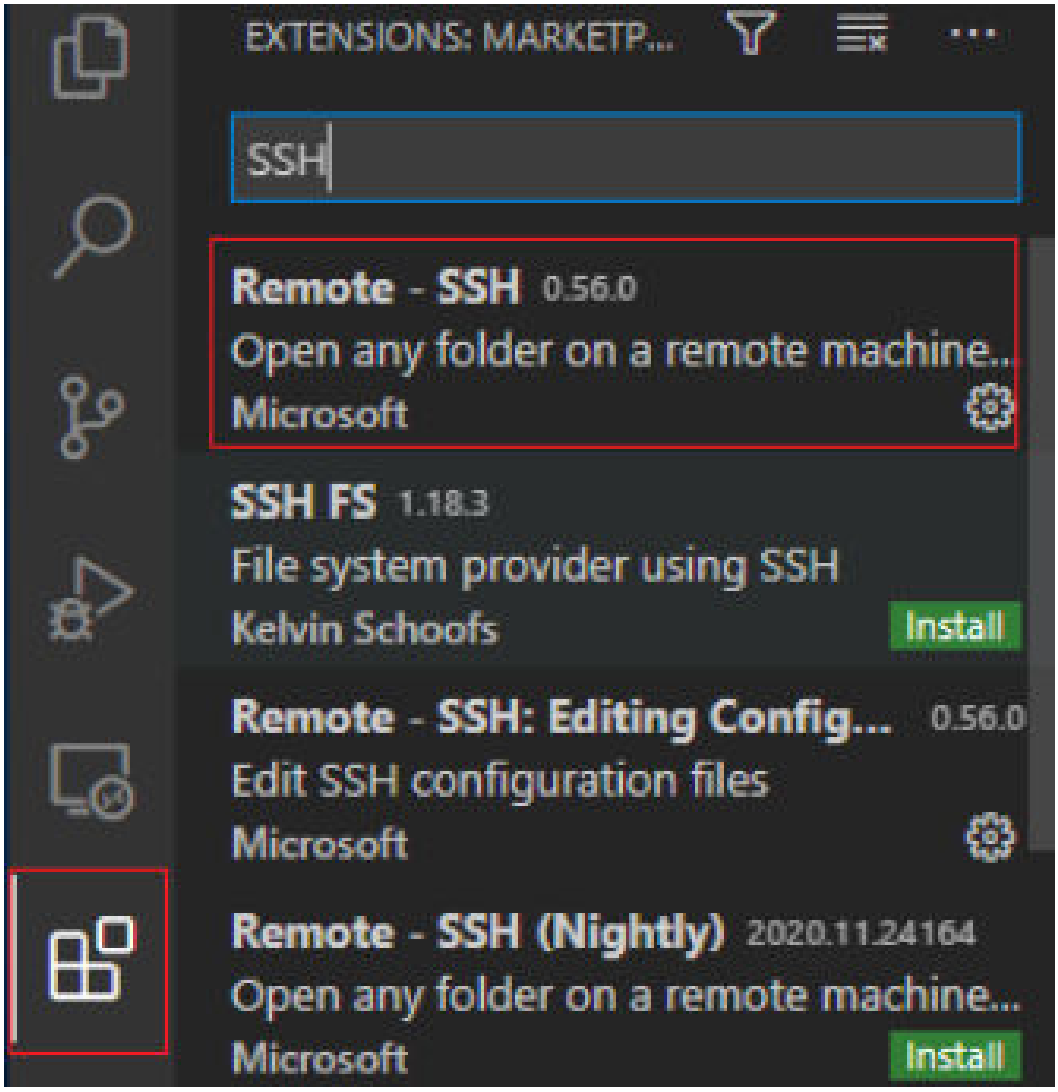
- The key pair is available.

  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.

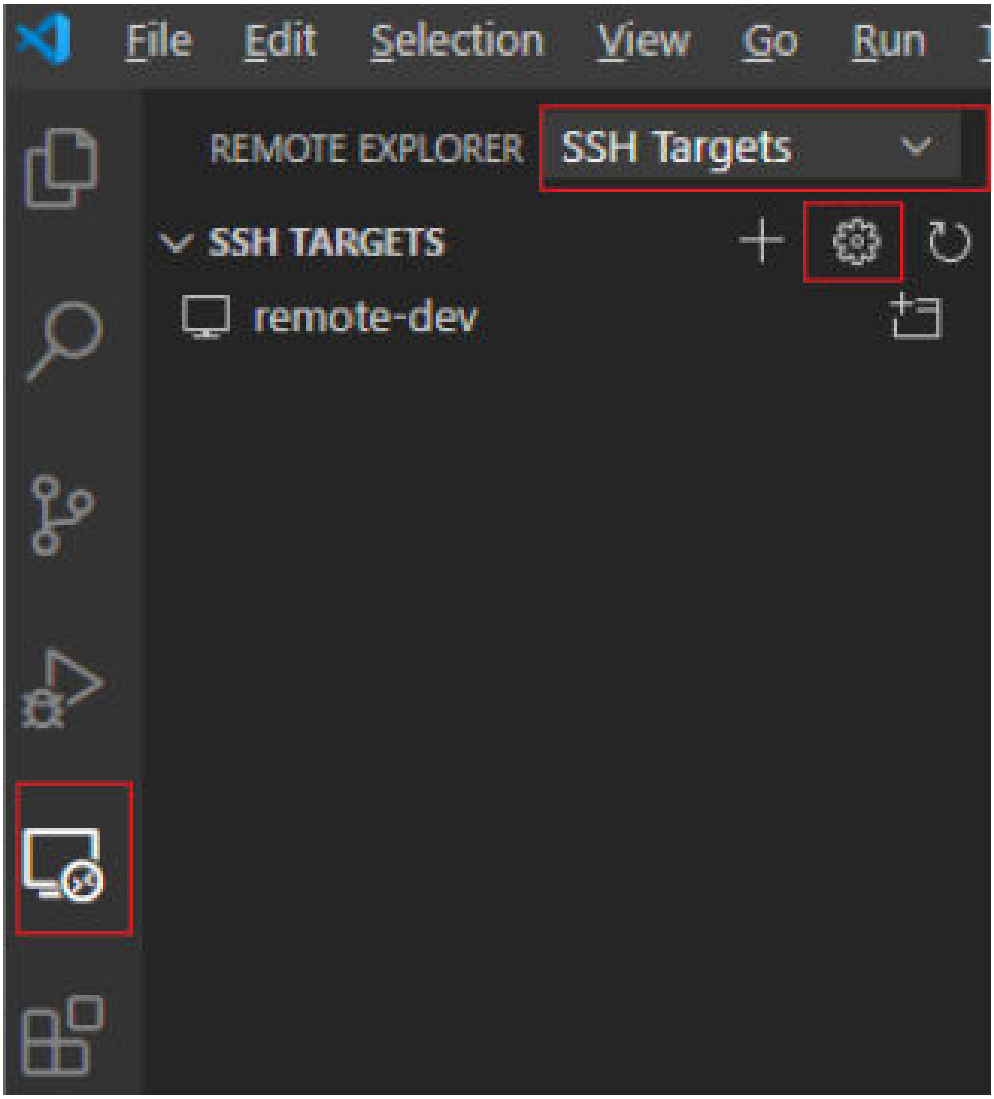
Figure 6-116 Adding the Remote-SSH plug-in



Step 2 Configure SSH

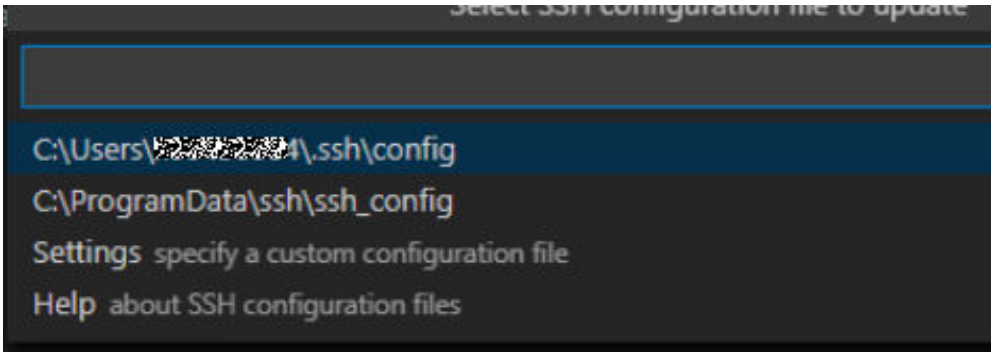
1. In the local VS Code development environment, click  on the left, select **SSH Targets** from the drop-down list box, and click  . The SSH configuration file path is displayed.

Figure 6-117 Configuring SSH Targets



2. Click the SSH configuration path and configure SSH.

Figure 6-118 SSH configuration file path




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

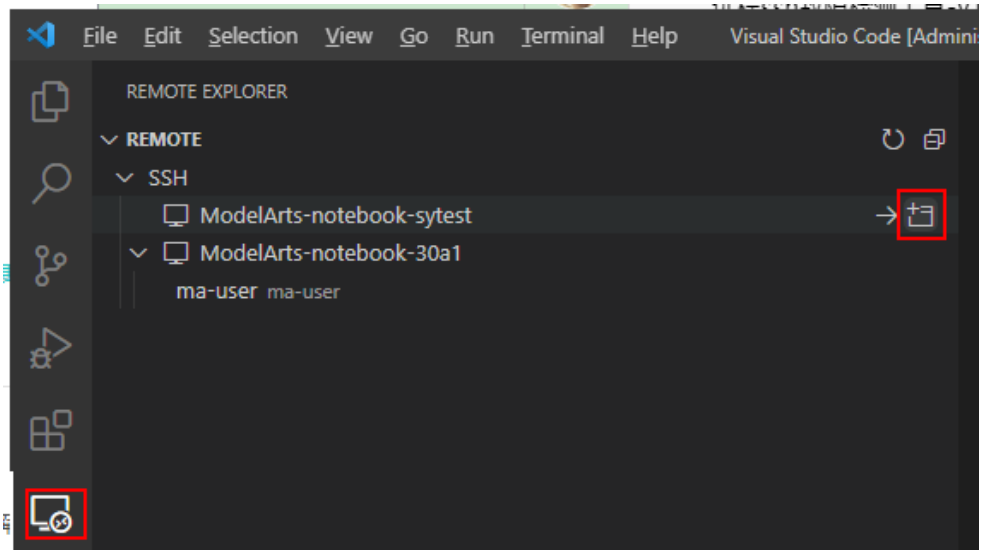


```
UserKnownHostsFile=/dev/null
StrictHostKeyChecking no
```

- **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 6-119 Opening the development environment



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

Figure 6-120 Selecting a notebook running environment

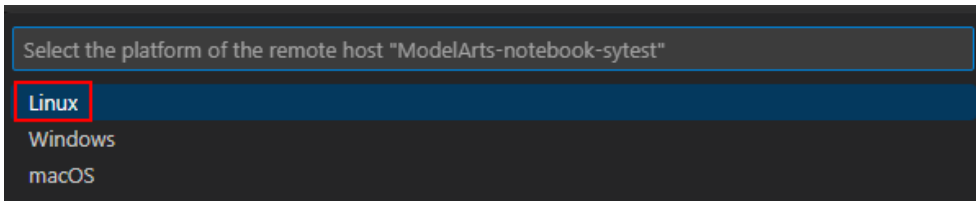
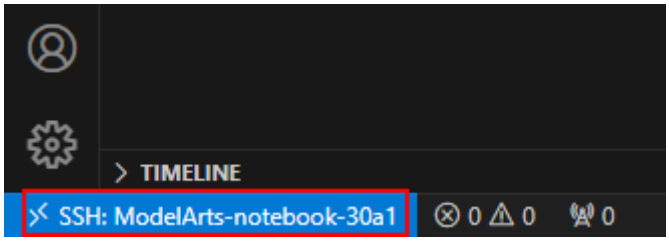


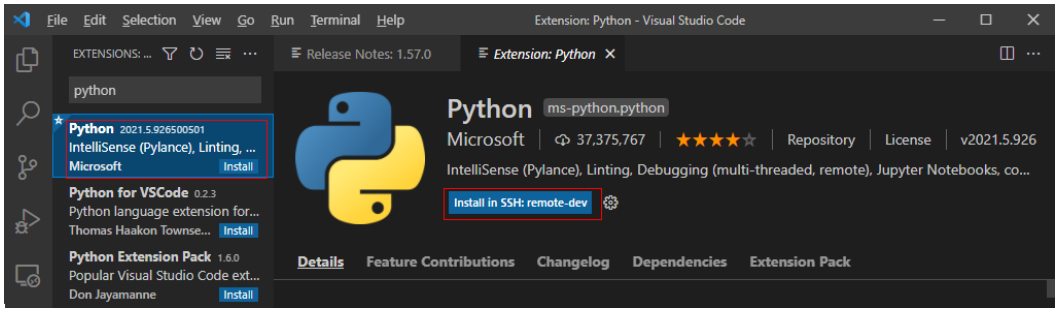
Figure 6-121 Remote connection succeeded



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 6-122 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
   ```

6.5.5 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. For details, see **Uploading an Object**.

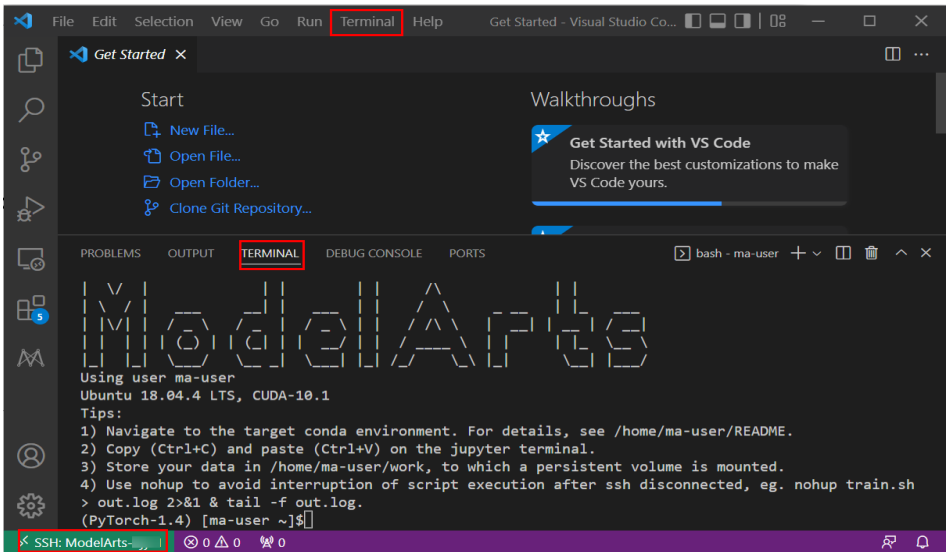
   Alternatively, use ModelArts SDK in the terminal of local VS Code to upload data to OBS. To do so, click **Terminal** on the top menu bar. Enter **python** and press **Enter** in terminal to access the Python environment.

   ```
   python
   ```

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

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 6-123 Opening the terminal in the remote VS Code environment

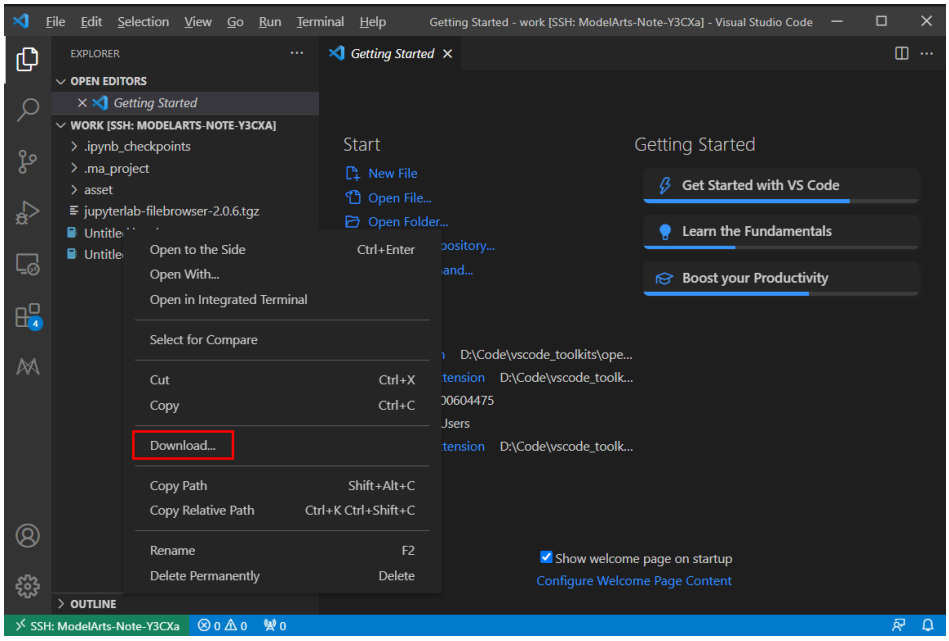

   ```
   # Manually access the development environment using the source command.
   cat /home/ma-user/README
   # Select the target environment.
   source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
   # Enter python and press Enter to access the Python environment.
   python
   ```

   Then, perform OBS transfer operations by referring to **Uploading a File to OBS**.

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 6-124 Downloading files from a notebook instance to a local directory in VS Code



6.6 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 6-125 Instance details page

- 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.

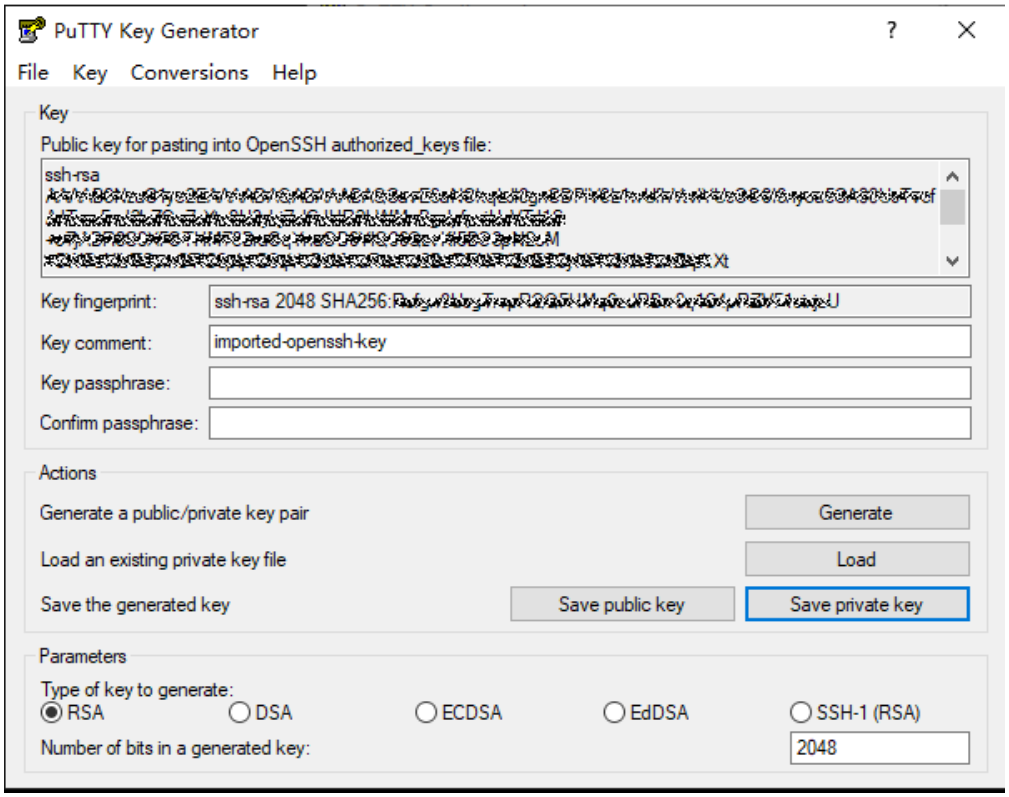
Step 1 Install the SSH Tool

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**.

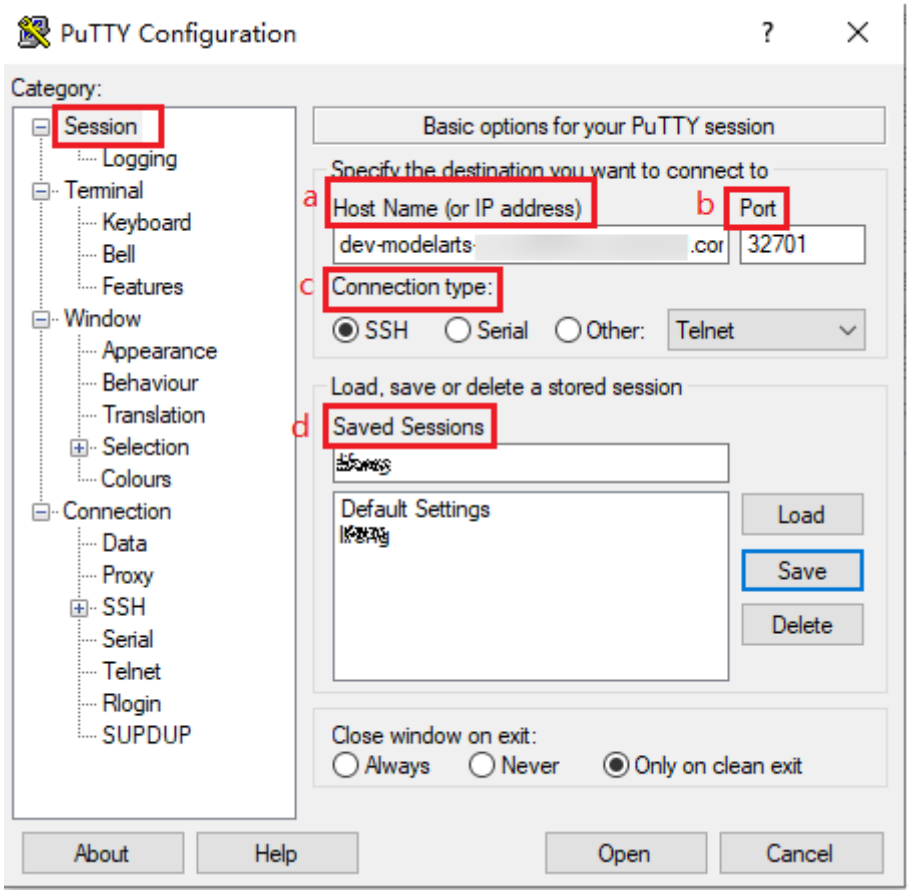
Figure 6-126 Converting the .pem key pair file to a .ppk key pair file



Step 3 Use SSH to Connect to a Notebook Instance

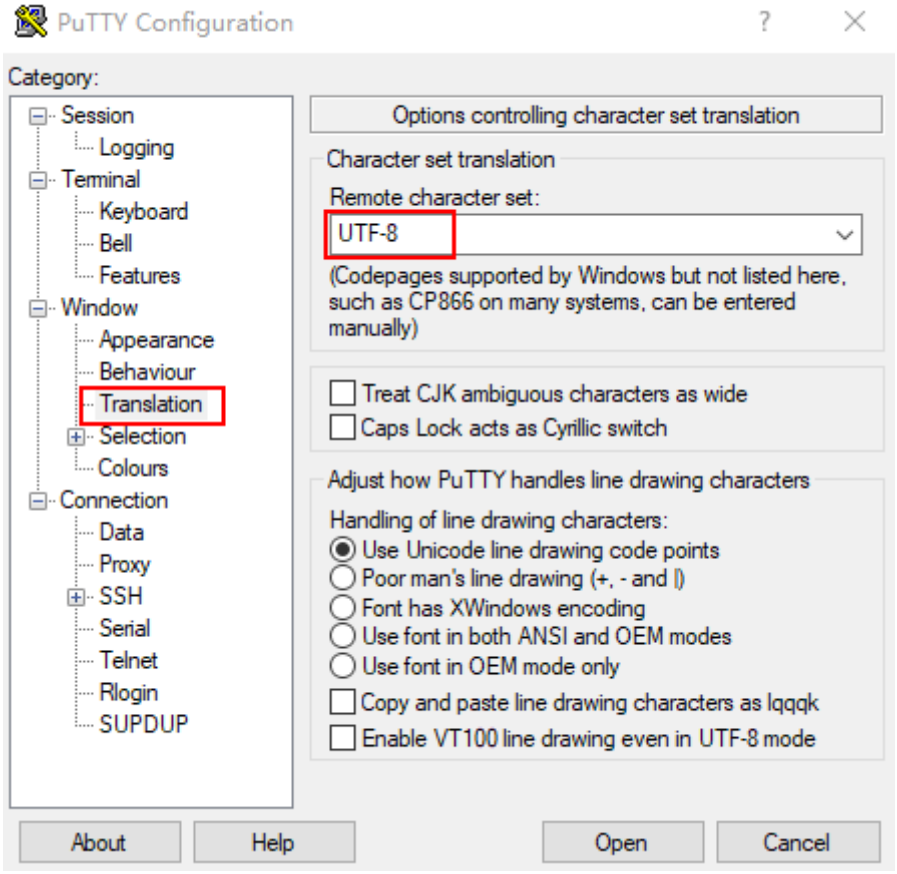
1. Run PuTTY.
2. Click **Session** and set the following parameters:
   a. **Host Name (or IP address)**: address for accessing the in-cloud notebook instance. Obtain the address on the page providing detailed information of the target notebook instance .
   b. **Port**: port number for accessing the 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 6-127 Configuring **Session**



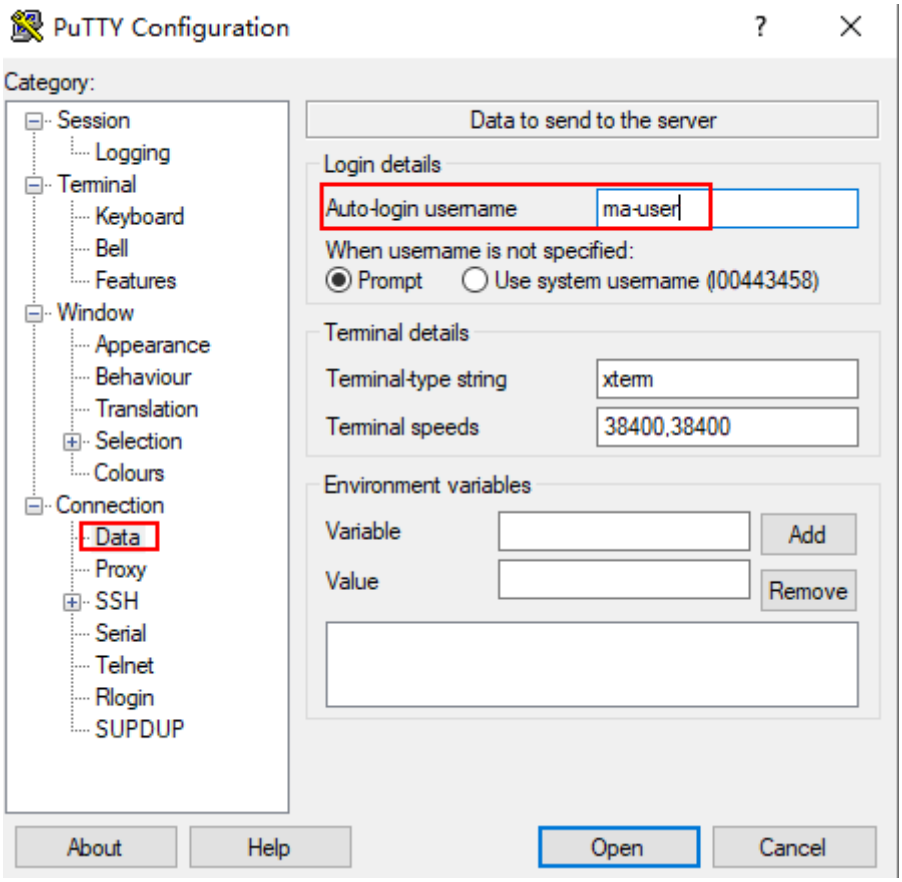
3. Choose **Window** > **Translation** and select **UTF-8** from the drop-down list box in the **Remote character set** area.

Figure 6-128 Setting the character format

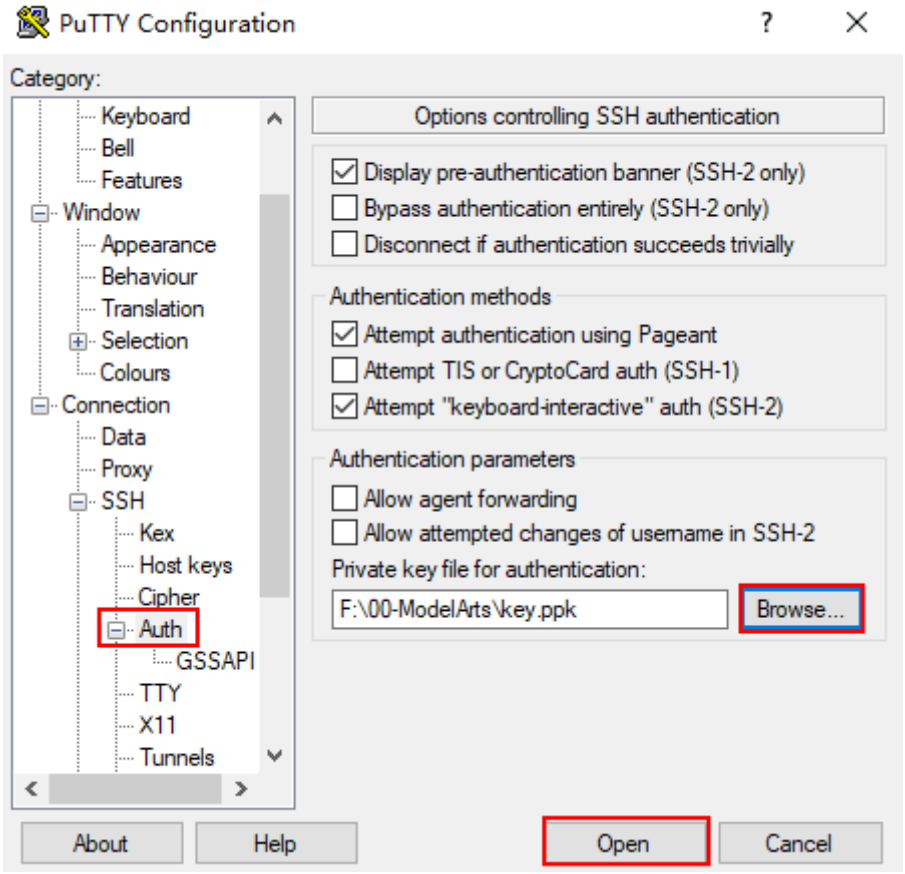


4. Choose **Connection** > **Data** and enter **ma-user** for **Auto-login username**.

Figure 6-129 Entering a username

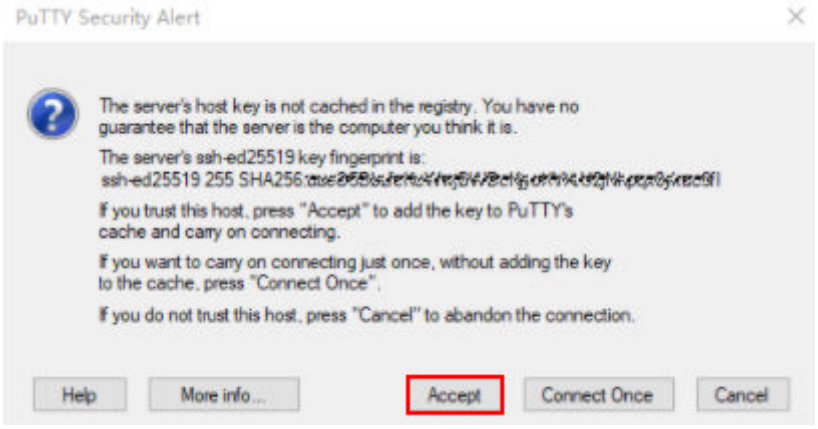


5. Choose **Connection** > **SSH** > **Auth**, click **Browse**, and select the .ppk file generated in **step 2**.



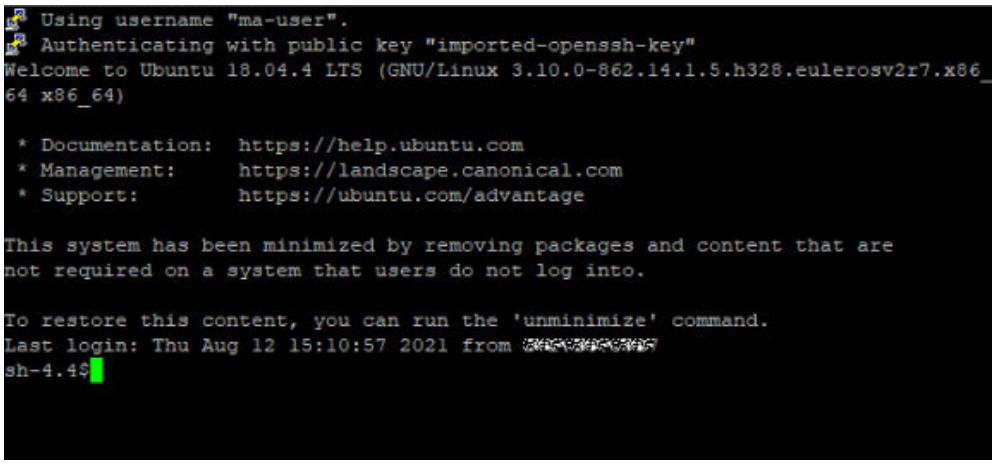
6. Click **Open**. If you are logging in to the instance for the first time, PuTTY displays a security warning dialog box, asking if you want to accept the instance security certificate. Click **Accept** to save the certificate to your local registry.

   Figure 6-130 Asking if you want to accept the instance security certificate



7. Connect to the notebook instance.

Figure 6-131 Connecting to a notebook instance



6.7 Managing Notebook Instances

6.7.1 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 **Development Workspace** > **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.

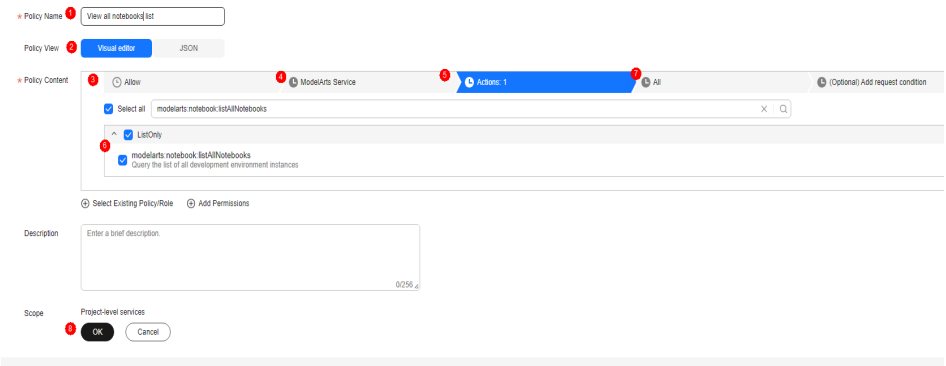
1. Log in to the ModelArts management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions** > **Policies/Roles** 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, as shown in **Figure 6-132**.

   - **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.

Figure 6-132 Creating a custom policy



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 Configurations**. ModelArts.6789: Failed to find SSH key pair KeyPair-xxx on the ECS key pair page. Update the key pair and try again later.

6.7.2 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 **Development Workspace** > **Notebook**.

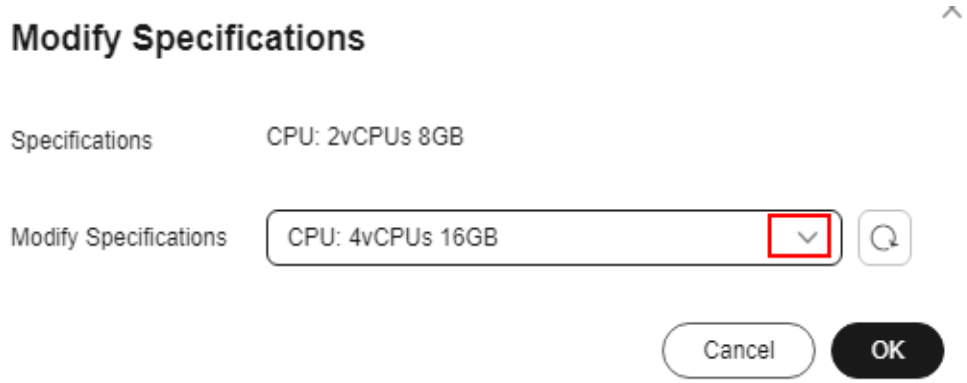
2. In the notebook list, click **More** in the **Operation** column of the target notebook instance and select **Change Image**.
3. In the **Change Image** dialog box, select a new image and click **OK**. After the modification, you can view the new image on the notebook list page.

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 **Development Workspace** > **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.

Figure 6-133 Selecting specifications



NOTE

The specifications can be changed only when there are other options in the cluster. If no other specifications are available, the change cannot be performed.

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 **Development Workspace** > **Notebook**.
2. Click the target notebook instance. Enable remote SSH and change the key pair and whitelist.

NOTE

For manually enabled remote SSH, see **Figure 6-134**. After the SSH configuration is updated, the remote SSH function cannot be disabled.

For remote SSH enabled by default in the selected image, see **Figure 6-135**.

Figure 6-134 Updating SSH configuration

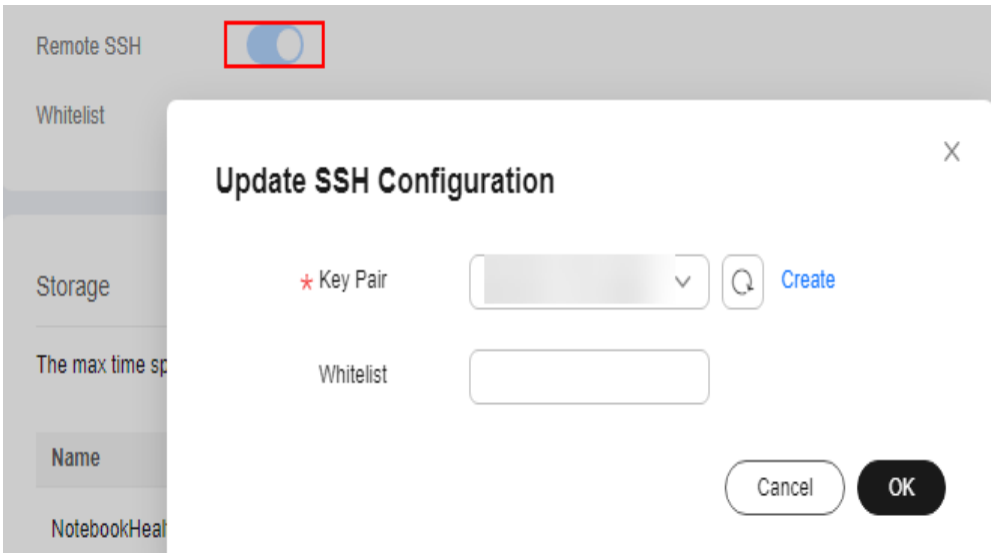
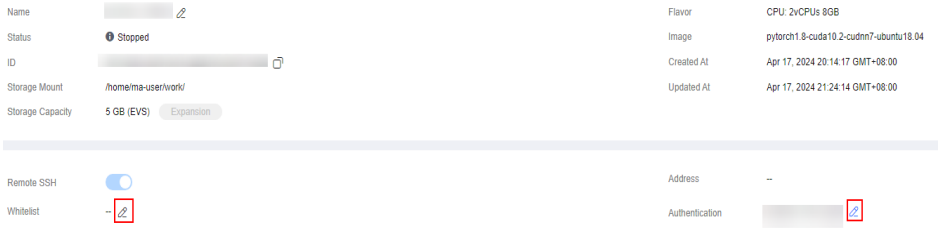


Figure 6-135 Changing the whitelist and key pair



- 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 Huawei Cloud 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.

6.7.3 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 **Development Workspace** > **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**.
> - The notebook instance will no longer be billed. However, if the instance is attached with an EVS disk, the storage space will still be billed.

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 **Development Workspace** > **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.

6.7.4 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

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

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

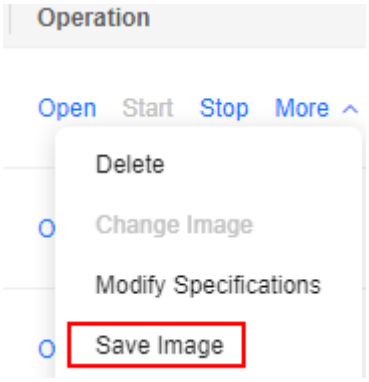
Prerequisites

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

Saving an Image

1. In the notebook instance list, select the target notebook instance and choose **Save Image** from the **More** drop-down list in the **Operation** column. The **Save Image** dialog box is displayed.

   Figure 6-136 Saving an image

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 6-137 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.

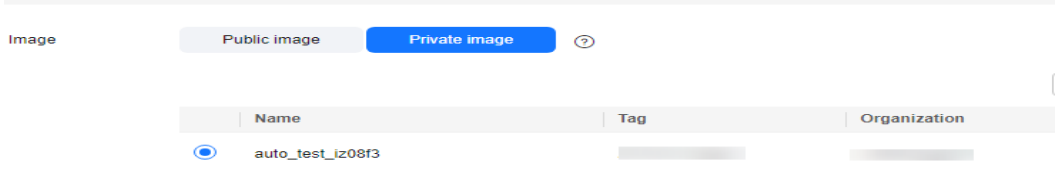
4. After the image is saved, the instance status changes to **Running**. View the image on the **Image Management** page.
5. Click the name of the image to view its details.

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 6-138 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.

6.7.5 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 6-139 Selecting EVS when creating a notebook instance

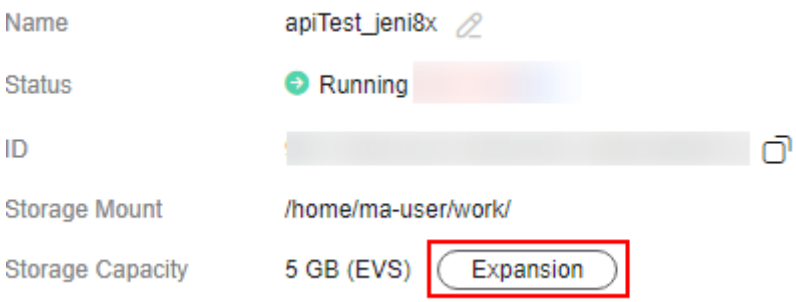
  

- Up to 100 GB can be expanded at a time. Additionally, the total capacity after expansion cannot exceed 4,096 GB.
- If the original capacity of an EVS disk is 4,096 GB, the disk capacity cannot be expanded.
- After the instance is stopped, the expanded capacity still takes effect. The billing is based on the expanded EVS disk capacity.
- An EVS disk is billed as long as it is used. To stop billing an EVS disk, delete data from the EVS disk and release the disk.

Procedure

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Click the name of a running notebook instance. On the instance details page, click **Expansion**.

   Figure 6-140 Instance details page

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.

Figure 6-141 Capacity expansion

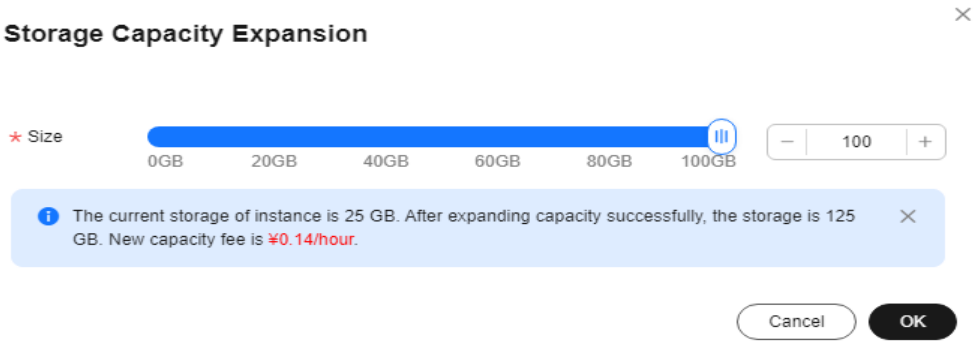
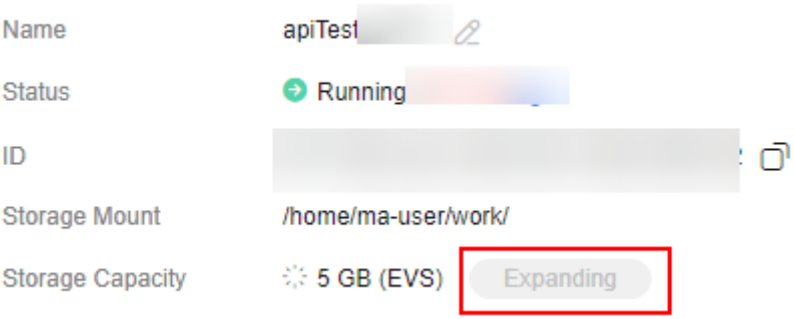


Figure 6-142 Expanding



6.7.6 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). For details, see **About Parallel File System**.

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.

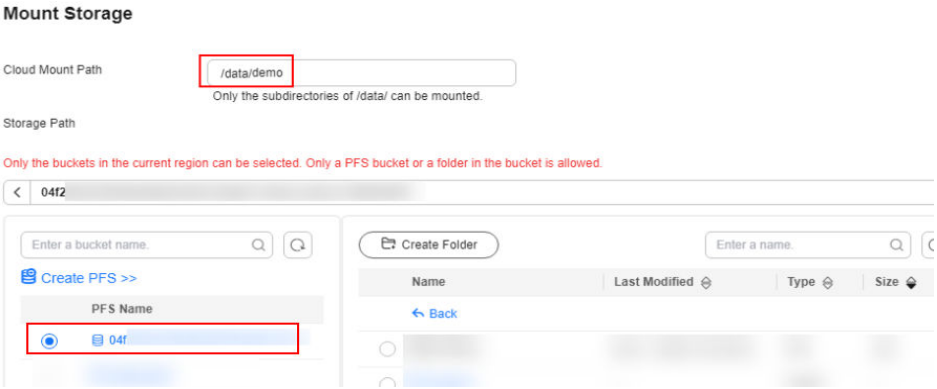
ModelArts notebook supports only the mounting of an OBS parallel file system to **/data/** of a notebook container.

Procedure

Method 1: Through the ModelArts management console

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

   Figure 6-143 Dynamically mounting an OBS parallel file system



3. View the mounting result on the notebook instance details page.

   Figure 6-144 Successful mounting

6.7.7 Viewing Notebook Events

Instance statuses and key operations such as creating, starting, and stopping an instance, and changing the instance flavor are recorded in the backend. You can view the events on the notebook instance details page to monitor the instance statuses. You can refresh events on the right of the **Event** tab. You can also set the interval for automatically refreshing events to 30 seconds, 1 minute, or 5 minutes.

Figure 6-145 Viewing notebook instance events and configuring automatic refresh

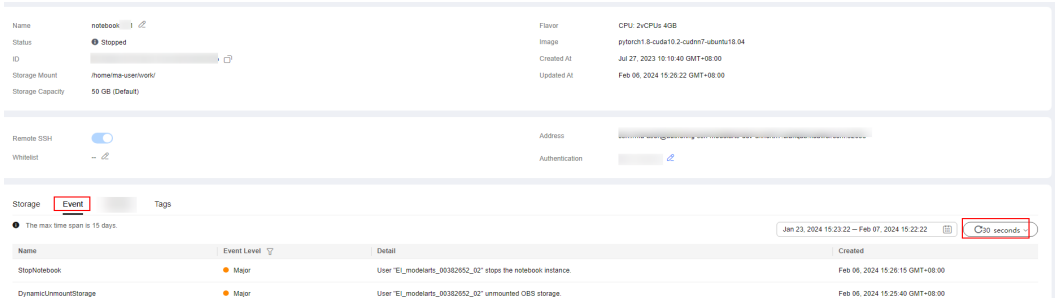


Table 6-12 Events during instance creation

Event	Description	Severity
Scheduled	The instance has been scheduled.	Warning
PullingImage	The image is being pulled.	Warning
PulledImage	The image has been pulled.	Warning
NotebookHealthy	The instance is running and healthy.	Major
CreateNotebookFailed	Creating an instance failed.	Critical
PullImageFailed	Pulling the image failed.	Critical
FailedCreate	Failed to create notebook container. Please contact SRE to check node {node_name}	Critical
CreateContainerError	Failed to create container. Please contact SRE to check node {node_name}	Critical
FailedAttachVolume	Failed to attach volume. Please contact SRE to check node {node_name}	Major
MountVolumeFailed	Mount volume failed; Check whether the DEW secret is correct if the instance cannot change to running in five minutes	Critical

Event	Description	Severity
	Mount volume failed; Check if vpc of sfs-turbo is interconnected if the instance cannot change to running in five minutes	Critical
	Mount volume failed; Please contact SRE to check node {node_name} if the instance cannot change to running in five minutes	Critical

Table 6-13 Events during instance startup

Event Name	Description	Severity
EmptyDirExceeded	Usage of empty-dir volume exceeds its limit. A new container will be scheduled and created automatically soon.	Critical
NodeResourcePressure	Insufficient node resources. A new container will be scheduled and created automatically soon.	Critical
EphemeralStorageExceeded	Local ephemeral storage exceeds its limit. A new container will be scheduled and created automatically soon.	Critical
FailedToStartContainer	Failed to start container. Please contact SRE to check node {node_name}	Critical
Scheduled	The instance has been scheduled.	Warning
PullingImage	The image is being pulled.	Warning
PulledImage	The image has been pulled.	Warning
NotebookHealthy	The instance is running and healthy.	Major
RunHookScript	Running a custom script	Warning
StartNotebookFailed	Starting the instance failed.	Critical
PullImageFailed	Pulling the image failed.	Critical

Event Name	Description	Severity
CreateKernelFailed	Creating a Jupyter kernel failed because the conda command is unavailable. (The conda environments are not being detected and added as Jupyter kernels. Ensure that {*conda_env*} is available and the command {*conda_cmd*} env list can be run properly.)	Major
	Creating a Jupyter kernel failed due to permission issues. (Kernels are not showing up in Jupyter Notebook due to permission issues. Ensure that the uid {*ma_uid*} has write permissions on {*conda_path*}.)	Major
ConfigurationError	Configuring the ModelArts SDK and CLI paths in the conda environment failed due to unavailable conda command. (The ModelArts SDK and CLI are unavailable in the conda environments due to conda environment issues. Ensure that {*conda_env*} is available and the command {*conda_cmd*} env list can be run properly.)	Major
	Configuring the ModelArts SDK and CLI paths in the conda environment failed due to permission issues. (The ModelArts SDK and CLI are unavailable in the conda environments due to conda environment issues. Ensure that the uid {*ma_uid*} has write permissions on {*conda_path*}.)	Major
FailedToPullImageReason	Failed to pull image. Please make sure the image exists in SWR repo, otherwise contact SRE to check node {node_name}	Major
	Failed to pull image. Please contact SRE to check node {node_name} **NOTE** {*node_name*} indicates the node name. It is a variable and is generally in the format of an IP address, for example, **192.168.1.1**.	

Table 6-14 Events during instance stopping

Event	Description	Severity
StopNotebook	The instance has been stopped.	Major
StopNotebookResourceIdle	The notebook instance will automatically stop or has automatically stopped because resources are idle.	Major

Table 6-15 Events during instance update

Event	Description	Severity
UpdateName	Updating the instance name	Warning
UpdateDescription	Updating the instance description	Warning
UpdateFlavor	Updating the instance flavor	Major
UpdateImage	Updating the instance image	Major
UpdateStorageSize	The instance storage size is being updated. (User %s is updating storage size from %s GB to %s GB.)	Major
	The instance storage size has been updated. (User %s updated the storage size.)	Major
UpdateKeyPair	Configured the instance key pair. (User %s updated the instance key pair to {*%s*}.)	Major
	Updating the instance key pair (User %s updated the instance key pair from %s to %s.)	Major
UpdateWhitelist	Updating the instance access whitelist	Major
UpdateHook	Updating a custom script	Major
UpdateStorageSizeFailed	Updating the storage size failed because the resources are sold out. (EVS disks are sold out.)	Critical

Event	Description	Severity
	Updating the storage size failed due to an internal error. (Updating the EVS disk size failed. The O&M personnel are handling the fault.)	Critical

Table 6-16 Events during image saving

Event	Description	Severity
SaveImage	The image has been saved.	Major
SavedImageFailed	Saving the image failed due to processes in D status. (There are processes in 'D' status. Check process status using 'ps -aux' and kill all the processes in 'D' status.)	Critical
	Saving the image failed because the image is too large. (The container size (%dG) is greater than the threshold (%dG).)	Critical
	Saving the image failed due to the limit on the number of layers. (There are too many layers in your image.)	Critical
	Saving the image failed due to task timeout. (The O&M personnel are handling the fault.)	Critical
	Saving the image failed due to SWR service issues.	Critical

Table 6-17 Events during instance running

Event Name	Description	Severity
NotebookUnhealthy	The instance is unhealthy.	Critical
OutOfMemory	The instance is out of memory.	Critical
JupyterProcessKilled	The Jupyter process has been stopped.	Critical
CacheVolumeExceed-Quota	The **/cache** file size has exceeded the upper limit.	Critical
NotebookHealthy	The instance has been restored to the healthy state.	Major
EVSSoldOut	EVS disks are sold out.	Critical

Table 6-18 Events for dynamic OBS mounting

Event	Description	Severity
DynamicMountStorage	The OBS storage is mounted.	Major
DynamicUnmountStor-age	The OBS storage is unmounted.	Major

Table 6-19 Events triggered on the user side

Event	Description	Severity
RefreshCredentialsFailed	Authentication failed.	Critical

6.7.8 Notebook Cache Directory Alarm Reporting

When creating a notebook instance, you can select CPU, GPU, or Ascend resources based on the service data volume. If you select GPU or Ascend resources, ModelArts mounts hard disks to the cache directory. You can use this directory to store temporary files.

Capacity alarms are not generated for the cache directory of the notebook instance by default. Exceeding the capacity limit will restart the notebook instance. After the restart, multiple configurations are reset, discarding your data and losing the environment. This will affect your experience. You are advised to enable the monitoring and alarms for the cache directory usage and report the data to AOM.

Configuration Process

1. Enter the basic alarm information.
2. **Set an alarm rule.**
   a. Configure monitoring metrics.
   b. Set alarm triggering conditions.
3. **Configure alarm notifications.**
   a. Create a topic, configure the topic policy, and subscribe to the topic.
   b. Create an alarm action rule.
   c. Select the created action rule.

Configuring Alarm Settings

1. Log in to the AOM console.
2. Choose **Alarm Center** > **Alarm Rules** and click **Create Alarm Rule**.
3. Enter the basic alarm information.

4. Set an alarm rule.

   Rule Type: Select **Threshold alarm**.

   Monitored Object: Select **Select resource objects**. Click **Select Resource Object**. A new dialog box is displayed.

   - **Add By**: Select **Dimension**.
   - **Metric Name**: Click **Custom Metrics** and select the cache metrics to be monitored. Example: **ma_container_notebook_cache_dir_size_bytes** (total size of the cache directory) and **ma_container_notebook_cache_dir_util** (usage of the cache directory)
   - **Dimension**: Select a metric dimension, for example, *service_id:xxx*, and click **Confirm**.

   After setting the monitored object, set **Statistic** and **Statistical Period**.

   Alarm Condition: Set this parameter based on your needs.

Figure 6-146 Select Monitored Object

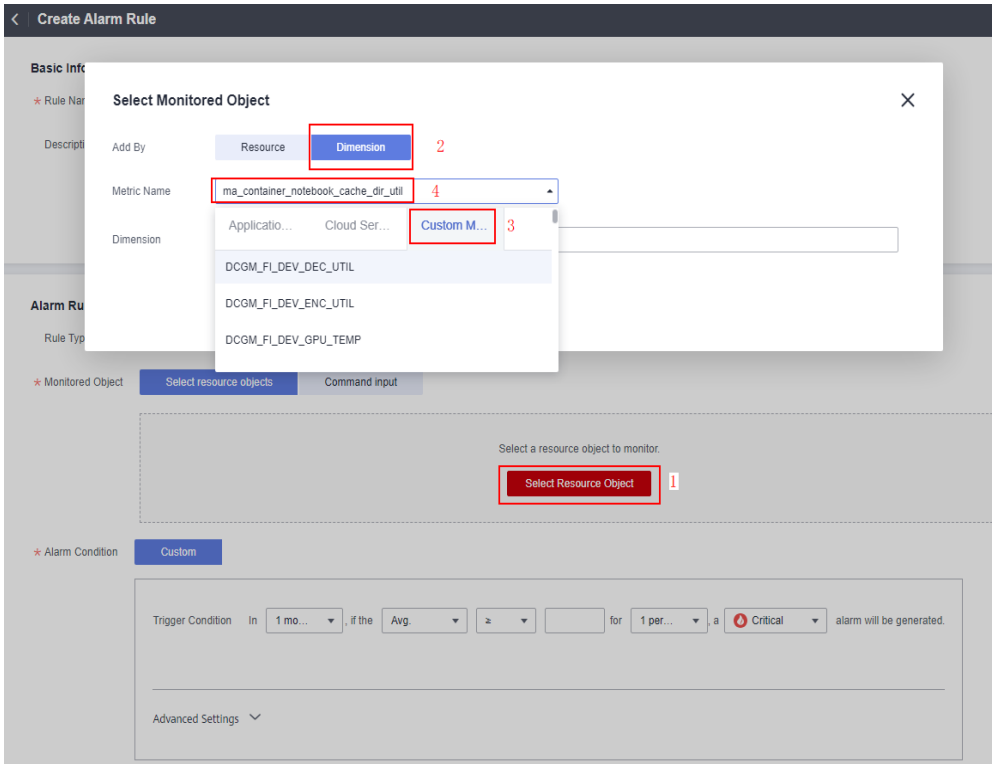


Figure 6-147 Configuring statistics method

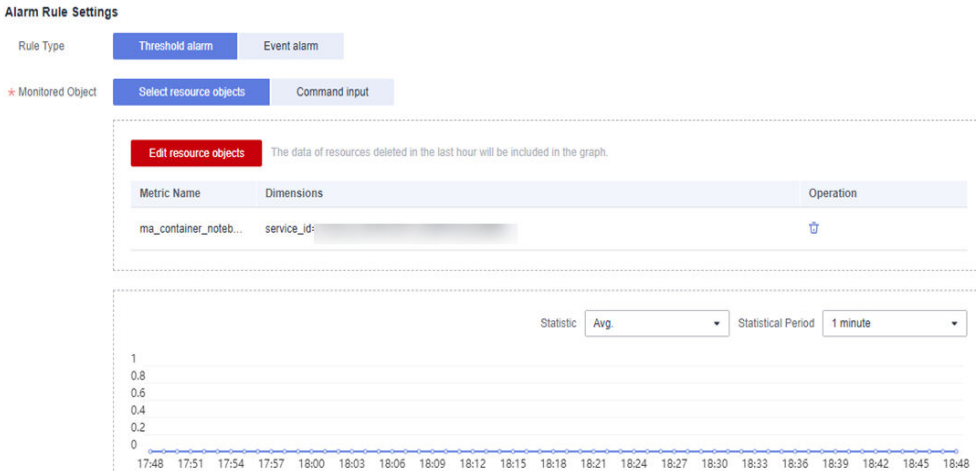
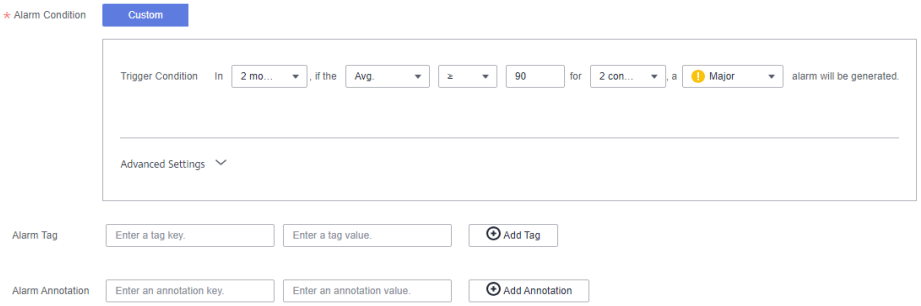


Figure 6-148 Configuring alarm conditions



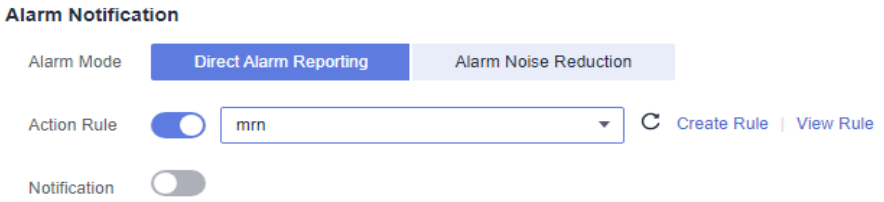
5. Configure alarm notifications and click **Create Now**.

   Alarm Mode: Select **Direct Alarm Reporting**.

   Action Rule: Enable it and select the created action rule. If the existing alarm action rules cannot meet your requirements, click **Create Rule** to create an action rule. For details, see **Creating an Alarm Action Rule**.

   Notification: Enable it.

   Figure 6-149 Configuring alarm notifications

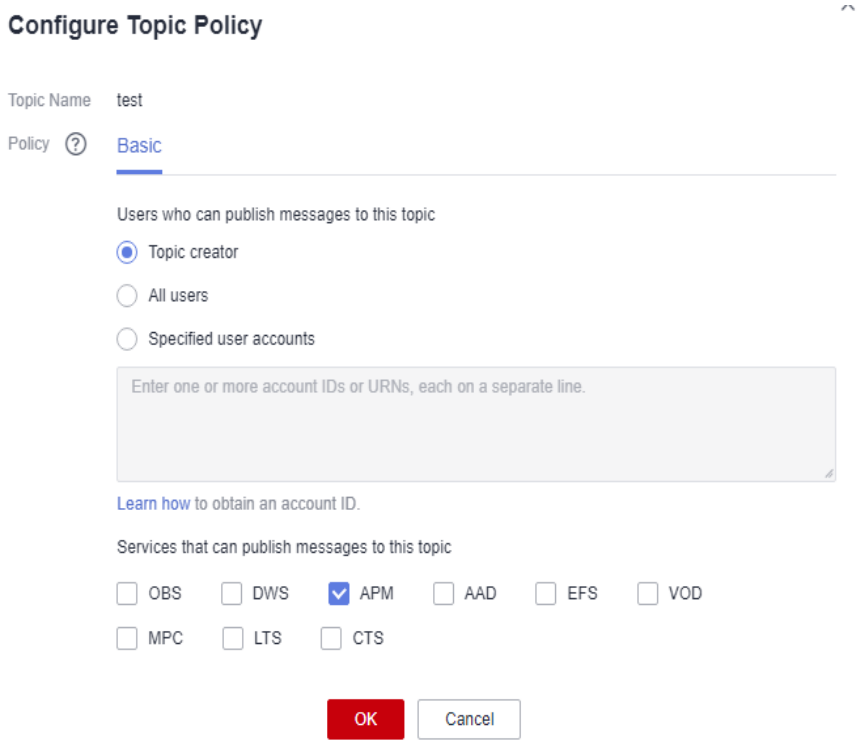
   

   Create a topic in SMN to configure alarm notification rules.

   - **Creating a Topic**
     i. Go to the SMN console. In the navigation pane, choose **Topic Management** > **Topics**.
     ii. Click **Create Topic**. Enter a topic name, select an enterprise project, and click **OK**.
     iii. Locate the target topic and choose **More** > **Configure Topic Policy** in the **Operation** column.

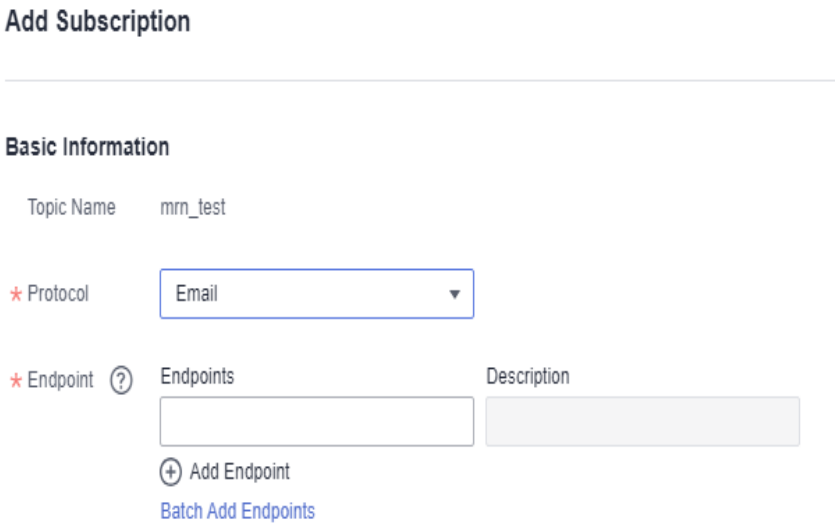
     Select **APM** to allow AOM alarms to trigger SMN.

     Figure 6-150 Configure Topic Policy

iv. Click **Add Subscription** in the **Operation** column of the topic. After the subscription is successful, a notification is received once the alarm conditions are met.

Select a protocol, such as email or SMS, and enter the endpoints, such as email addresses or mobile numbers. Click **OK**.



A record is displayed in the subscription list, but the record is in the **Unconfirmed** state.

After receiving the email, confirm the subscription.

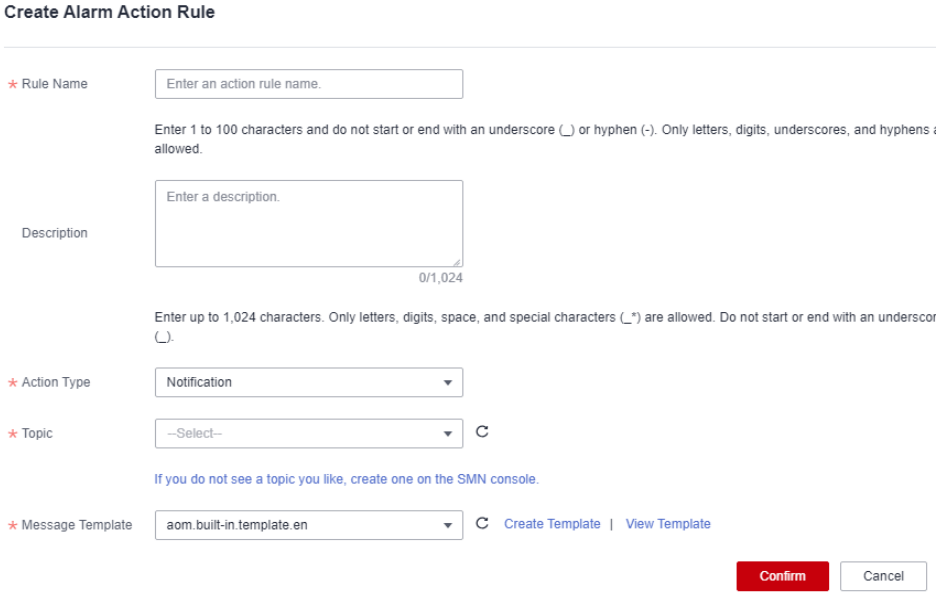
Then, the subscription is in the confirmed state.

- **Creating an Alarm Action Rule**

  An action rule specifies how AOM notifies you when an alarm is triggered. After an alarm action rule is enabled, the system sends notifications based on the associated SMN topic and message template.

  Enter the action rule name, select the action rule type, select the topic created in **the previous step**, select a message template, and click **Confirm**.

Figure 6-151 Create Alarm Action Rule



In the **Alarm Notification** area of the **Create Alarm Rule** page, set **Action Rule** to the newly created alarm action rule and click **Create Now**.

After the configuration is complete, you will receive an email notification once the alarm conditions are met.

6.8 ModelArts CLI Command Reference

6.8.1 ModelArts CLI Commands

Description

ModelArts CLI, also called ma-cli, is a cross-platform command line tool used to connect to ModelArts and run management commands on ModelArts resources. You can use the interactive command prompt or script to run commands on a terminal. ma-cli allows you to interact with cloud services through ModelArts notebook and on-premises VMs. You can run ma-cli commands for command autocomplete and authentication, as well as creating images, submitting ModelArts training jobs and DLI Spark jobs, and copying OBS data.

Application Scenarios

- ma-cli has been integrated into ModelArts notebook and can be directly used.

  Log in to the ModelArts console, choose **Development Workspace** > **Notebook**, create a notebook instance, start a terminal, and run ma-cli commands.
- In local Windows or Linux, install ma-cli and then use it on a local terminal. For details, see **(Optional) Installing ma-cli Locally**.

NOTE

- ma-cli cannot be used in Git Bash.
- Terminals such as Linux Bash, Zsh, Fish, WSL, and PowerShell are recommended. To ensure the security of your sensitive information, it is important to prevent any potential leakage when using terminals.

Command Preview

```
$ ma-cli -h
Usage: ma-cli [OPTIONS] COMMAND [ARGS]...

Options:
  -V, -v, --version       1.2.1
  -C, --config-file TEXT  Configure a file path for authorization.
  -D, --debug             Debugging mode, in which the full stack trace will be displayed when an error occurs.
  -P, --profile TEXT      CLI connection profile to be used. The default profile is DEFAULT.
  -h, -H, --help          Show the help information and exit.

Commands:
  configure        Configure authentication and endpoints for the CLI.
  image            Obtain registered images, register or unregister images, debug images, and create images in
Notebook.
  obs-copy         Copy files or directories between OBS and a local path.
  ma-job           Submit ModelArts jobs and obtain jod details.
  dli-job          Submit DLI spark jobs and obtain jod details.
  auto-completion  Auto complete ma-cli command in terminal, support "bash(default)/zsh/fish".
```

Among the preceding parameters, parameters **-C**, **-D**, **-P**, and **-h** are globally optional.

- **-C** indicates that you can manually specify the authentication configuration file when running this command. By default, the **~/.modelarts/ma-cli-profile.yaml** configuration file is used.
- **-P** indicates a group of authentication information in the authentication file. The default value is **DEFAULT**.
- **-D** indicates whether to enable the debugging mode (disabled by default). After the debugging mode is enabled, the error stack information of the command will be printed. If this mode is disabled, only the error information will be printed.
- **-h** indicates that the help information about the command will be displayed.

Commands

Table 6-20 ma-cli commands

Command	Description
configure	ma-cli authentication using a username and password or an SK/SK
image	ModelArts image creation, registration, and registered image query
obs-copy	Copying files or folders between a local path and OBS
ma-job	Managing ModelArts training jobs, including job submission and resource query

Command	Description
dli-job	DLI Spark job submission and resource management
auto-completion	Command autocomplete

6.8.2 (Optional) Installing ma-cli Locally

Application Scenarios

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

Step 1: Install ModelArts SDKs

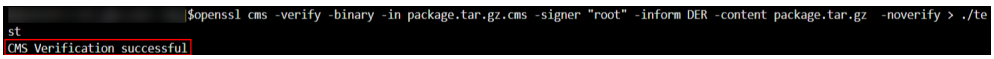
Install ModelArts SDKs by referring to **Installing the ModelArts SDK Locally**.

Step 2: Download ma-cli

1. **Download the ma-cli software package**.
2. Verify the software package signature.
   a. **Download the signature verification file of the software package**.
   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.



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 *.*.*
   ```
2. Run **pip --version** to check whether the general package management tool pip is available.
   ```
   C:\Users\xxx>pip --version
   pip **.*.* from c:\users\xxx\appdata\local\programs\python\python**\lib\site-packages\pip (python *.*)
   ```
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.

6.8.3 Autocompletion for ma-cli Commands

CLI autocomplete enables you to get a list of supported **ma-cli** commands by typing a command prefix and pressing **Tab** on your terminal. Autocomplete for **ma-cli** commands needs to be enabled in Terminal. After running the **ma-cli auto-completion** command, you can copy and run the commands as prompted on the current terminal to automatically complete the **ma-cli** commands. Bash, Fish, and Zsh shells are supported. The default shell is Bash.

Take the Bash command as an example. Run the **eval "$ (_MA_CLI_COMPLETE=bash_source ma-cli)"** command in Terminal to enable autocomplete.

```
eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

Run the **ma-cli auto-completion Zsh** or **ma-cli auto-completion Fish** command to view the autocomplete command in Zsh or Fish.

Available Commands

```
$ ma-cli auto-completion -h
Usage: ma-cli auto-completion [OPTIONS] [[Bash|Zsh|Fish]]

  Auto complete ma-cli command in terminal.

  Example:

  # print bash auto complete command to terminal
  ma-cli auto-completion Bash

Options:
  -H, -h, --help  Show this message and exit.
```

```
# By default, the autocomplete command for Bash is displayed.

$ ma-cli auto-completion

Tips: please paste following shell command to your terminal to activate auto complation.

[ OK ] eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"

# After the preceding command is executed, autocomplete has been enabled on the terminal.

$ eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

```
# The autocomplete command for Fish is displayed.
$ ma-cli auto-completion Fish
Tips: please paste following shell command to your terminal to activate auto complation.

[ OK ] eval (env _MA_CLI_COMPLETE=fish_source ma-cli)
```


6.8.4 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 6-21 Authentication CLI parameters

Parameter	Type	Mandatory	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

Parameter	Type	Mandatory	Description
-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.

6.8.5 ma-cli image Commands for Building Images

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

Commands for Creating an Image

```
$ ma-cli image -h
Usage: ma-cli image [OPTIONS] COMMAND [ARGS]...
  Obtain registered images, register or unregister images, debug images, and create images in Notebook.

Options:
  -H, -h, --help  Show this message and exit.

Commands:
  add-template, at  List build-in dockerfile templates.
  build             Build docker image in Notebook.
  debug             Debug SWR image as a Notebook in ECS.
  df                Query disk usage.
  get-image, gi     Query registered image in ModelArts.
  get-template, gt  List build-in dockerfile templates.
  prune             Prune image build cache.
  register          Register image to ModelArts.
  unregister        Unregister image from ModelArts.
```

Table 6-22 Commands for creating an image

Command	Description
get-template	Obtain an image creation template.

Command	Description
add-template	Load an image creation template.
get-image	Obtain registered ModelArts images.
register	Register SWR images with ModelArts image management.
unregister	Deregister a registered image from ModelArts image management.
build	Build an image using a Dockerfile (only supported in ModelArts Notebook).
df	Obtain image creation cache, which can only be used in ModelArts notebook.
prune	Clear image creation cache, which can only be used in ModelArts notebook.
debug	Debug an SWR image on an ECS to check whether the image can be used in ModelArts notebook. (Only the ECSs with Docker installed can be used.)

Using ma-cli image get-template to Query an Image Building Template

ma-cli provides some common image building templates, which contain the guidance for developing Dockerfiles on ModelArts notebook.

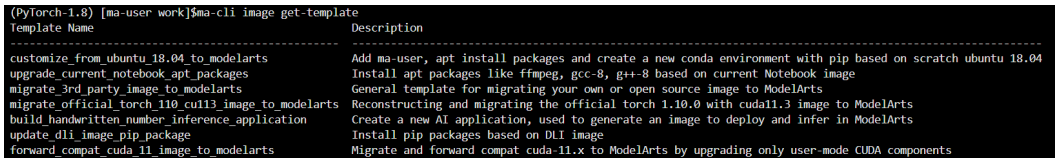
```
$ ma-cli image get-template -h
Usage: ma-cli image get-template [OPTIONS]

  List build-in dockerfile templates.

  Example:

  # List build-in dockerfile templates
  ma-cli image get-template [--filer <filter_info>]  [--page-num <yourPageNum>] [--page-size
<yourPageSize>]

Options:
  --filter TEXT                  filter by keyword.
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -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.
(PyTorch-1.4) [ma-user work]$
```


Table 6-23 Parameters

Parameter	Data Type	Mandatory	Description
--filter	String	No	Template name keyword for filtering templates.
-pn / --page-num	Int	No	Image page index. The default value is page 1.
-ps / --page-size	Int	No	Number of images displayed on each page. The default value is **20**.

Example: Obtain an image building template.

```
ma-cli image get-template
```



Using ma-cli image add-template to Load an Image Building Template

The **add-template** command is used to load image templates to a specified folder. By default, the path where the current command is located is used,

for example, **${current_dir}/.ma/${template_name}/**. You can also use **--dest** to specify the path. If a template folder with the same name already exists in the target path, use the **--force | -f** parameter to forcibly overwrite the existing template folder.

```
$ ma-cli image add-template -h
Usage: ma-cli image add-template [OPTIONS] TEMPLATE_NAME

  Add buildin dockerfile templates into disk.

  Example:

  # List build-in dockerfile templates
  ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts --force

Options:
  --dst TEXT          target save path.
  -f, --force         Override templates that has been installed.
  -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 6-24 Parameters

Parameter	Data Type	Mandatory	Description
--dst	String	No	Target path for loading a template. The current path is used by default.
-f / --force	Bool	No	Whether to forcibly overwrite an existing template with the same name. By default, the template is not overwritten.

Example: Load the **customize_from_ubuntu_18.04_to_modelarts** image building template.

```
ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
[ OK ] Successfully add configuration template [ customize_from_ubuntu_18.04_to_modelarts ] under folder [ /home/ma-user/work/.ma/customize_from_ubuntu_18.04_to_modelarts ]
```

Using ma-cli image get-image to Query Registered ModelArts Images

A path to a base image is provided in a Dockerfile typically. Public images and SWR public or private images can be obtained from open-source image repositories such as Docker Hub. ma-cli allows you to obtain ModelArts preset images, registered images, and their SWR addresses.

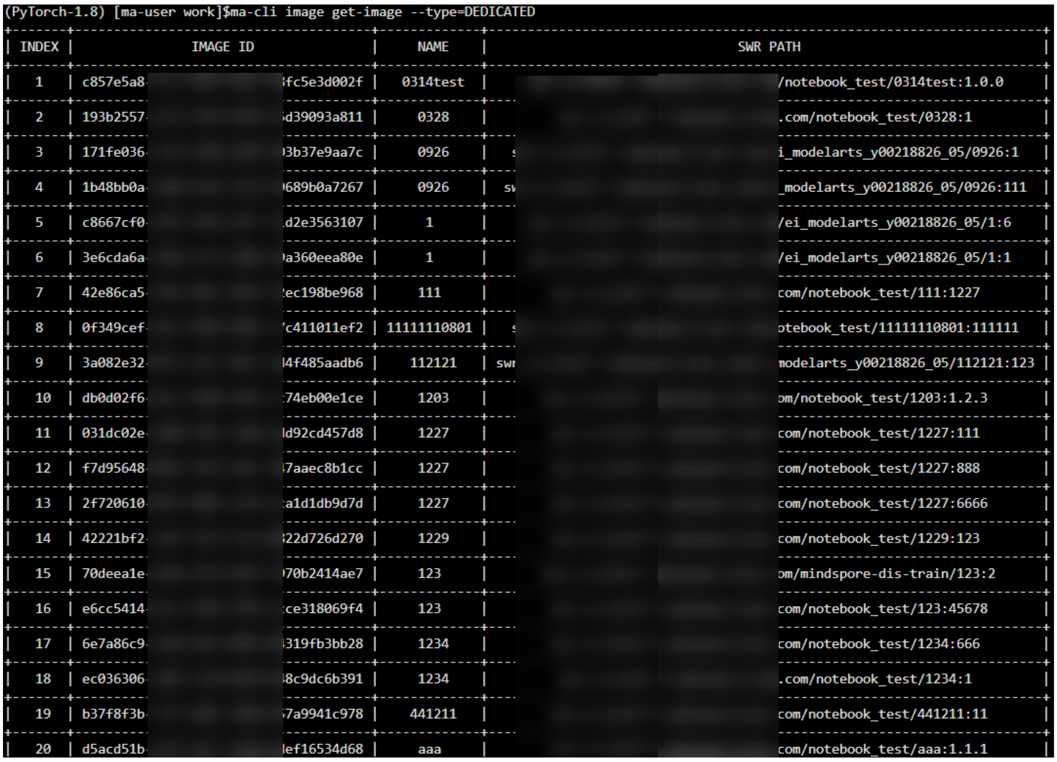
```
$ma-cli image get-image -h
Usage: ma-cli image get-image [OPTIONS]

  Get registered image list.

  Example:

  # Query images by image type and only image id, show name and swr_path
  ma-cli image get-image --type=DEDICATED

  # Query images by image id
  ma-cli image get-image --image-id ${image_id}

  # Query images by image type and show more information
  ma-cli image get-image --type=DEDICATED -v

  # Query images by image name
  ma-cli image get-image --filter=torch

Options:
  -t, --type [BUILD_IN|DEDICATED|ALL]
                                  Image type(default ALL)
  -f, --filter TEXT               Image name to filter
  -v, --verbose                   Show detailed information on image.
  -i, --image-id TEXT             Get image details by image id
  -n, --image-name TEXT           Get image details by image name
  -wi, --workspace-id TEXT        The workspace where you want to query image(default "0")
  -pn, --page-num INTEGER RANGE   Specify which page to query  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query  [x>=1]
  -C, --config-file PATH          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 6-25 Parameters

Parameter	Data Type	Mandatory	Description
-t / --type	String	No	Type of the images to be obtained. The options are **BUILD_IN**, **DEDICATED**, and **ALL**. ● **BUILD_IN**: preset images ● **DEDICATED**: custom images registered with ModelArts ● **ALL**: all images
-f / --filter	String	No	Image name keyword for filtering images.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.
-i / --image-id	String	No	ID of the image whose details are to be obtained.
-n / --image-name	String	No	Name of the image whose details are to be obtained.
-wi / --workspace-id	String	No	Workspace in which the image information is to be obtained.
-pn / --page-num	Int	No	Image page index. The default value is page 1.
-ps / --page-size	Int	No	Number of images displayed on each page. The default value is **20**.

Example: Obtain custom images registered with ModelArts.

```
ma-cli image get-image --type=DEDICATED
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image get-image --type=DEDICATED
```

INDEX	IMAGE ID	NAME	SWR PATH
1	c857e5a8[redacted]fc5e3d002f	0314test	[redacted]/notebook_test/0314test:1.0.0
2	193b2557[redacted]d39093a811	0328	[redacted].com/notebook_test/0328:1
3	171fe036[redacted]3b37e9aa7c	0926	[redacted]i_modelarts_y00218826_05/0926:1
4	1b48bb0a[redacted]689b0a7267	0926	[redacted]_modelarts_y00218826_05/0926:111
5	c8667cf0[redacted]d2e3563107	1	[redacted]/ei_modelarts_y00218826_05/1:6
6	3e6cda6a[redacted]a360eea80e	1	[redacted]/ei_modelarts_y00218826_05/1:1
7	42e86ca5[redacted]ec198be968	111	[redacted]com/notebook_test/111:1227
8	0f349cef[redacted]c411011ef2	11111110801	[redacted]otebook_test/11111110801:111111
9	3a082e32[redacted]4f485aadb6	112121	[redacted]modelarts_y00218826_05/112121:123
10	db0d02f6[redacted]74eb00e1ce	1203	[redacted]om/notebook_test/1203:1.2.3
11	031dc02e[redacted]d92cd457d8	1227	[redacted]com/notebook_test/1227:111
12	f7d95648[redacted]7aaec8b1cc	1227	[redacted]com/notebook_test/1227:888
13	2f720610[redacted]a1d1db9d7d	1227	[redacted]com/notebook_test/1227:6666
14	42221bf2[redacted]22d726d270	1229	[redacted]com/notebook_test/1229:123
15	70deea1e[redacted]70b2414ae7	123	[redacted]om/mindspore-dis-train/123:2
16	e6cc5414[redacted]ce318069f4	123	[redacted]com/notebook_test/123:45678
17	6e7a86c9[redacted]319fb3bb28	1234	[redacted]com/notebook_test/1234:666
18	ec036306[redacted]8c9dc6b391	1234	[redacted].com/notebook_test/1234:1
19	b37f8f3b[redacted]7a9941c978	441211	[redacted]com/notebook_test/441211:11
20	d5acd51b[redacted]ef16534d68	aaa	[redacted]com/notebook_test/aaa:1.1.1

Using ma-cli image build to Build an Image in ModelArts Notebook

Run the **ma-cli image build** command to build an image based on a specified Dockerfile. This command is available only on ModelArts notebook instances.

```
$ ma-cli image build -h
Usage: ma-cli image build [OPTIONS] FILE_PATH

  Build docker image in Notebook.

  Example:

  # Build a image and push to SWR
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1

  # Build a image and push to SWR, dockerfile context path is current dir
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1 -context .

  # Build a local image and save to local path and OBS
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile --target ./build.tar --
obs_path obs://bucket/object --swr-path my_organization/my_image:0.0.1

Options:
  -t, --target TEXT        Name and optionally a tag in the 'name:tag' format.
  -swr, --swr-path TEXT    SWR path without swr endpoint, eg:organization/image:tag.  [required]
  --context DIRECTORY      build context path.
  -arg, --build-arg TEXT   build arg for Dockerfile.
  -obs, --obs-path TEXT    OBS path to save local built image.
  -f, --force              Force to overwrite the existing swr image with the same name and tag.
  -C, --config-file PATH   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 6-26 Parameters

Parameter	Data Type	Mandatory	Description
FILE_PATH	String	Yes	Path where the Dockerfile is stored.
-t / --target	String	No	Local path for storing the generated TAR package. The current directory is used by default.
-swr / --swr-path	String	Yes	SWR image name, which is in the format of "organization/image_name:tag". This parameter can be omitted when a TAR package is saved for building an image.
--context	String	No	Context used for copying data when a Dockerfile is used.
-arg / --build-arg	String	No	Parameter for building an image. If there are multiple parameters, run **--build-arg VERSION=18.04 --build-arg ARCH=X86_64**.
-obs / --obs-path	String	No	OBS path for automatically uploading the generated TAR package.
-f / --force	Bool	No	Whether to forcibly overwrite an existing SWR image with the same name. By default, the SWR image is not overwritten.

Example: Build an image in ModelArts notebook.

```
ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr notebook_test/my_image:0.0.1
```

In this command, **.ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile** is the path where the Dockerfile is stored, and **notebook_test/my_image:0.0.1** is the SWR path of the new image.

Using ma-cli image df to Obtain Image Building Caches in ModelArts Notebook

Run the **ma-cli image df** command to obtain image creation caches. This command is available only on ModelArts notebook instances.

```
$ ma-cli image df -h
Usage: ma-cli image df [OPTIONS]

  Query disk usage used by image-building in Notebook.

  Example:

  # Query image disk usage
  ma-cli image df

Options:
  -v, --verbose      Show detailed information on disk usage.
  -D, --debug        Debug Mode. Shows full stack trace when error occurs.

  -h, -H, --help     Show this message and exit.
```

Table 6-27 Parameters

Parameter	Data Type	Mandatory	Description
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

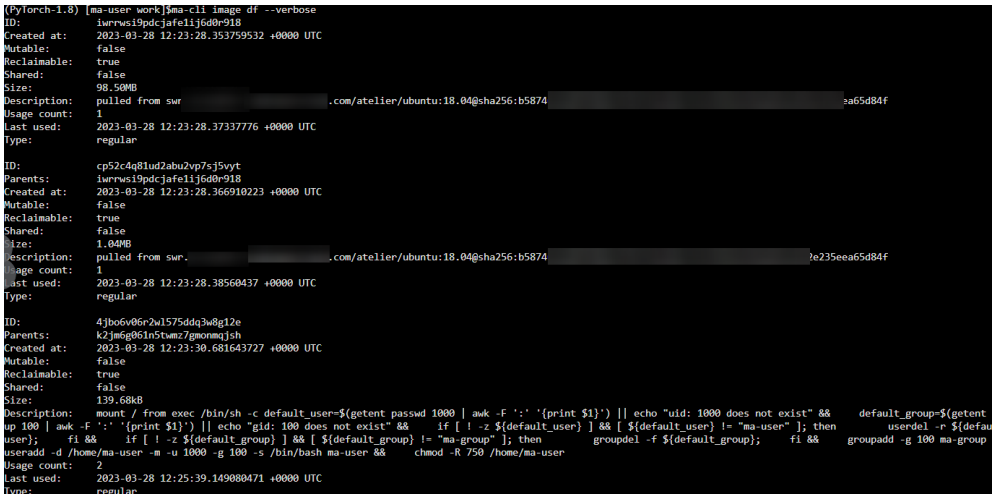
Example: View all image caches in ModelArts notebook.

```
ma-cli image df
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image df
ID                                                              RECLAIMABLE     SIZE            LAST ACCESSED
iwrrwsi9pdcjafe1ij6d0r918                                       true            98.50MB
cp52c4q81ud2abu2vp7sj5vyt                                       true            1.04MB
4jbo6v06r2w1575ddq3w8g12e                                       true            139.68kB
ojdjw5mok71s1nh2cauant051                                       true            86.86kB
k2jm6g061n5twmz7gmonmqjsh                                       true            16.55kB
efu5kwgig1ve44fe7smbrncnh*                                      true            8.19kB
uzikwqk5taxnslvajm14jrbje*                                      true            4.10kB
2g8p0qcb014g3qva7ucawkv87*                                      true            4.10kB
Reclaimable:    99.80MB
Total:          99.80MB
```

Example: Obtain details about the disk space occupied by image caches.

```
ma-cli image df --verbose
```

Using ma-cli image prune to Clear Image Building Caches in ModelArts Notebook

Run the **ma-cli image prune** command to clear image creation caches. This command is available only on ModelArts notebook instances.

```
$ ma-cli image prune -h
Usage: ma-cli image prune [OPTIONS]

  Prune image build cache by image-building in Notebook.

  Example:

  # Prune image build cache
  ma-cli image prune

Options:
  -ks, --keep-storage INTEGER  Amount of disk space to keep for cache below this limit (in MB) (default: 0).
  -kd, --keep-duration TEXT    Keep cache newer than this limit, support second(s), minute(m) and hour(h)
(default:   0s).
  -v, --verbose                Show more verbose output.
  -D, --debug                  Debug Mode. Shows full stack trace when error occurs.
  -h, -H, --help               Show this message and exit.
```

Table 6-28 Parameters

Parameter	Data Type	Mandatory	Description
-ks / --keep-storage	Int	No	Size of the cache to be retained, in MB. The default value is **0**, indicating that all caches will be cleared.
-kd / --keep-duration	String	No	Duration in which the caches are to be retained. The unit can be **s** (second), **m** (minute), or **h** (hour). The default value is **0s**, indicating that all caches will be cleared.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Retain 1 MB of image caches not to be cleared.

```
ma-cli image prune -ks 1
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image prune -ks 1
ID                                                              RECLAIMABLE     SIZE    LAST ACCESSED
uzikwqk5taxnslvajm14jrbje*                                      true            4.10kB
4jbo6v06r2wl575ddq3w8g12e                                       true            139.68kB
k2jm6g061n5twmz7gmonmqjsh                                       true            16.55kB
ojdjw5mok71s1nh2cauant051                                       true            86.86kB
cp52c4q81ud2abu2vp7sj5vyt                                       true            1.04MB
iwrrwsi9pdcjafe1ij6d0r918                                       true            98.50MB
Total:  99.79MB
```

Using ma-cli image register to Register an SWR Image with ModelArts Image Management

After an image is debugged, run the **ma-cli image register** command to register it with ModelArts image management so that it can be used in ModelArts.

```
$ma-cli image register -h
Usage: ma-cli image register [OPTIONS]

  Register image to ModelArts.

  Example:

  # Register image into ModelArts service
  ma-cli image register --swr-path=xx

  # Share SWR image to DLI service
  ma-cli image register -swr xx -td

  # Register image into ModelArts service and specify architecture to be 'AARCH64'
  ma-cli image register --swr-path=xx --arch AARCH64

Options:
  -swr, --swr-path TEXT           SWR path without swr endpoint, eg:organization/image:tag.  [required]
  -a, --arch [X86_64|AARCH64]     Image architecture (default: X86_64).
  -s, --service [NOTEBOOK|MODELBOX]
                                  Services supported by this image(default NOTEBOOK).
  -rs, --resource-category [CPU|GPU|ASCEND]
                                  The resource category supported by this image (default: CPU and GPU).
  -wi, --workspace-id TEXT        The workspace to register this image (default: "0").
  -v, --visibility [PUBLIC|PRIVATE]
                                  PUBLIC: every user can use this image. PRIVATE: only image owner can use this
image (Default: PRIVATE).
  -td, --to-dli                   Register swr image to DLI, which will share SWR image to DLI service.
  -d, --description TEXT          Image description (default: "").
  -C, --config-file PATH          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.
```

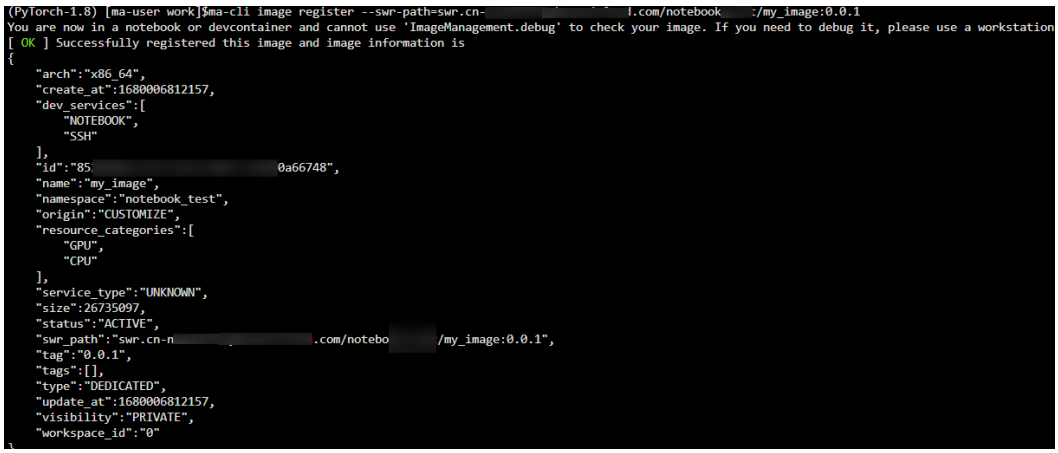
Table 6-29 Parameters

Parameter	Data Type	Mandatory	Description
-swr / --swr-path	String	Yes	SWR path to the image to be registered.
-a / --arch	Sring	No	Architecture of the image to be registered. The value can be **X86_64** or **AARCH64**. The default value is **X86_64**.

Parameter	Data Type	Mandatory	Description
-s / --service	String	No	Service type of the image to be registered. The value can be **NOTEBOOK** or **MODELBOX**. The default value is **NOTEBOOK**. You can also specify multiple values, for example, **-s NOTEBOOK -s MODELBOX**.
-rs / --resource-category	String	No	Resource type that can be used by the image to be registered. The default values are **CPU** and **GPU**.
-wi / --workspace-id	String	No	Workspace to which the image is to be registered. The default workspace ID is **0**.
-v / --visibility	Bool	No	Visibility of the image to be registered. The value can be **PRIVATE** (visible only to the image owner) or **PUBLIC** (visible to all users). The default value is **PRIVATE**.
-td / --to-dli	Bool	No	Whether to register the image with DLI.
-d/ --description	String	No	Description of the image. By default, this parameter is left blank.

Example: Register an SWR image with ModelArts.

```
ma-cli image register --swr-path=xx
```



Using ma-cli image unregister to Deregister an Image

Run the **ma-cli image unregister** command to deregister an image from ModelArts.

```
$ ma-cli image unregister -h
Usage: ma-cli image unregister [OPTIONS]

  Unregister image from ModelArts.
```

```
Example:

# Unregister image
ma-cli image unregister --image-id=xx

# Unregister image and delete it from swr
ma-cli image unregister --image-id=xx -d

Options:
  -i, --image-id TEXT      Unregister image details by image id.  [required]
  -d, --delete-swr-image   Delete the image from swr.
  -C, --config-file PATH   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 6-30 Parameters

Parameter	Data Type	Mandatory	Description
-i / -image-id	String	Yes	ID of the image to be deregistered.
-d / --delete-swr-image	Bool	No	Whether to delete a deregistered SWR image. It is disabled by default.

Debugging an SWR Image on an ECS

ma-cli allows you to debug an SWR image on an ECS to determine whether the image can be used in a ModelArts development workspace.

```
ma-cli image debug -h
Usage: ma-cli image debug [OPTIONS]

  Debug SWR image as a Notebook in ECS.

  Example:

  # Debug cpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region=

  # Debug gpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region= --gpu

Options:
  -swr, --swr-path TEXT           SWR path without SWR endpoint, eg:organization/image:tag.  [required]
  -r, --region TEXT               Region name.  [required]
  -s, --service [NOTEBOOK|MODELBOX]
                                  Services supported by this image(default NOTEBOOK).
  -a, --arch [X86_64|AARCH64]     Image architecture(default X86_64).
  -g, --gpu                       Use all gpus to debug.
  -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 6-31 Parameters

Parameter	Data Type	Mandatory	Description
-swr / --swr-path	String	Yes	SWR path to the image to be debugged.
-r / --region	String	Yes	Region where the image to be debugged is located.
-s / --service	String	No	Service type of the image to be debugged. The value can be **NOTEBOOK** or **MODELBOX**. The default value is **NOTEBOOK**.
-a / --arch	Sring	No	Architecture of the image to be debugged. The value can be **X86_64** or **AARCH64**. The default value is **X86_64**.
-g / --gpu	Bool	No	Whether to use GPUs for debugging. It is disabled by default.

6.8.6 ma-cli ma-job Commands for Training Jobs

Run the **ma-cli ma-job** command to submit training jobs, obtain training job logs, events, used AI engines, and resource specifications, and stop training jobs.

```
$ ma-cli ma-job -h
Usage: ma-cli ma-job [OPTIONS] COMMAND [ARGS]...

  ModelArts job submission and query job details.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
  delete      Delete training job by job id.
  get-engine  Get job engines.
  get-event   Get job running event.
  get-flavor  Get job flavors.
  get-job     Get job details.
  get-log     Get job log details.
  get-pool    Get job engines.
  stop        Stop training job by job id.
  submit      Submit training job.
```

Table 6-32 Commands supported by training jobs

Command	Description
get-job	Obtain ModelArts training jobs and their details.
get-log	Obtain runtime logs of a ModelArts training job.
get-engine	Obtain ModelArts AI engines for training.
get-event	Obtain ModelArts training job events.

Command	Description
get-flavor	Obtain ModelArts resource specifications for training.
get-pool	Obtain ModelArts resource pools dedicated for training.
stop	Stop a ModelArts training job.
submit	Submit a ModelArts training job.
delete	Delete a training job with a specified job ID.

Using ma-cli ma-job get-job to Obtain a ModelArts Training Job

Run the **ma-cli ma-job get-job** command to obtain training jobs or details about a specific job.

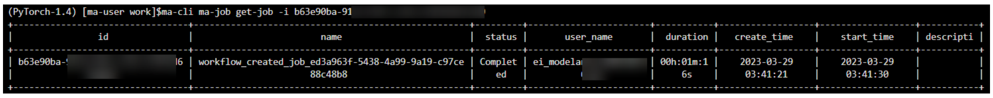
```
$ ma-cli ma-job get-job -h
Usage: ma-cli ma-job get-job [OPTIONS]

  Get job details.

  Example:

  # Get train job details by job name
  ma-cli ma-job get-job -n ${job_name}

  # Get train job details by job id
  ma-cli ma-job get-job -i ${job_id}

  # Get train job list
  ma-cli ma-job get-job --page-size 5 --page-num 1

Options:

  -i, --job-id TEXT               Get training job details by job id.
  -n, --job-name TEXT             Get training job details by job name.
  -pn, --page-num INTEGER         Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [1<=x<=50]
  -v, --verbose                   Show detailed information about training job details.
  -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 6-33 Parameters

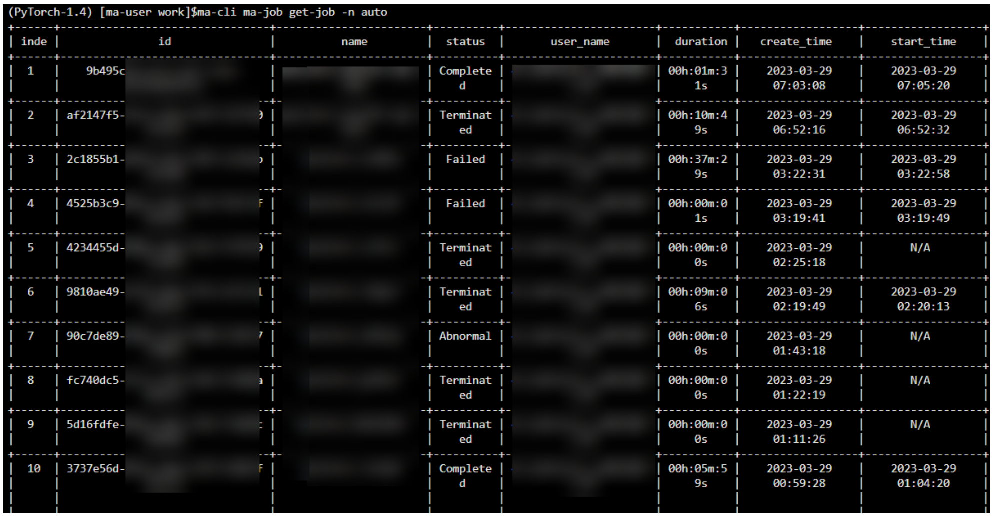
Parameter	Data Type	Mandatory	Description
-i / --job-id	String	No	ID of the job whose details are to be obtained.
-n / --job-name	String	No	Name of the job to be queried or name keyword used to filter training jobs.
-pn / --page-num	Int	No	Page number. The default value is **1**.

Parameter	Data Type	Mandatory	Description
-ps / --page-size	Int	No	Number of training jobs displayed on each page. The default value is **10**.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

- Example: Obtain a training job with a specified job ID.
  ma-cli ma-job get-job -i *b63e90xxx*



- Example: Filter training jobs by job name keyword **auto**.
  ma-cli ma-job get-job -n *auto*



Using ma-cli ma-job submit to Submit a ModelArts Training Job

Run the **ma-cli ma-job submit** command to submit a ModelArts training job.

When running this command, use the **YAML_FILE** parameter to specify the path to the configuration file of the target job. If this parameter is not specified, the configuration file is empty. The configuration file is in YAML format, and its parameters are values of **OPTIONS** in the command. If you specify both the **YAML_FILE** and the **OPTIONS** parameters, the **OPTIONS** value will overwrite the same items in the configuration file.

```
$ma-cli ma-job submit -h
Usage: ma-cli ma-job submit [OPTIONS] [YAML_FILE]...

 Submit training job.

 Example:

 ma-cli ma-job submit --code-dir obs://your_bucket/code/
                      --boot-file main.py
                      --framework-type PyTorch
```

```
                --working-dir /home/ma-user/modelarts/user-job-dir/code
                --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
                --data-url obs://your_bucket/dataset/
                --log-url obs://your_bucket/logs/
                --train-instance-type modelarts.vm.cpu.8u
                --train-instance-count 1

Options:
  --name TEXT                  Job name.
  --description TEXT           Job description.
  --image-url TEXT             Full swr custom image path.
  --uid TEXT                   Uid for custom image (default: 1000).
  --working-dir TEXT           ModelArts training job working directory.
  --local-code-dir TEXT        ModelArts training job local code directory.
  --user-command TEXT          Execution command for custom image.
  --pool-id TEXT               Dedicated pool id.
  --train-instance-type TEXT   Train worker specification.
  --train-instance-count INTEGER  Number of workers.
  --data-url TEXT              OBS path for training data.
  --log-url TEXT               OBS path for training log.
  --code-dir TEXT              OBS path for source code.
  --output TEXT                Training output parameter with OBS path.
  --input TEXT                 Training input parameter with OBS path.
  --env-variables TEXT         Env variables for training job.
  --parameters TEXT            Training job parameters (only keyword parameters are supported).
  --boot-file TEXT             Training job boot file path behinds `code_dir`.
  --framework-type TEXT        Training job framework type.
  --framework-version TEXT     Training job framework version.
  --workspace-id TEXT          The workspace where you submit training job(default "0")
  --policy [regular|economic|turbo|auto]
                               Training job policy, default is regular.
  --volumes TEXT               Information about the volumes attached to the training job.
  -q, --quiet                  Exit without waiting after submit successfully.
  -C, --config-file PATH       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 6-34 Parameters

Parameter	Data Type	Mandatory	Description
YAML_FILE	String	No	Configuration file of a training job. If this parameter is not specified, the configuration file is empty.
--code-dir	String	Yes	OBS path to the training source code.
--data-url	String	Yes	OBS path to the training data.
--log-url	String	Yes	OBS path to training logs.
--train-instance-count	String	Yes	Number of compute nodes in a training job. The default value is **1**, indicating a standalone node.

Parameter	Data Type	Mandatory	Description
--boot-file	String	No	Boot file specified when you use a preset command to submit a training job. This parameter can be omitted when you use a custom image or a custom command to submit a training job.
--name	String	No	Name of a training job.
--description	String	No	Description of a training job.
--image-url	String	No	SWR URL of a custom image, which is in the format of "organization/image_name:tag".
--uid	String	No	UID of the custom image. The default value is **1000**.
--working-dir	String	No	Work directory where an algorithm is executed.
--local-code-dir	String	No	Local directory of the training container to which the algorithm code directory is downloaded.
--user-command	String	No	Command for executing a custom image. The directory must be under **/home**. When **code-dir** is prefixed with **file://**, this parameter does not take effect.
--pool-id	String	No	Resource pool ID selected for a training job. You can log in to the ModelArts console, choose **Dedicated Resource Pools** in the navigation pane on the left, and view the resource pool ID in the dedicated resource pool list.
--train-instance-type	String	No	Resource flavor selected for a training job.
--output	String	No	Training output. After this parameter is specified, the training job will upload the output directory of the training container corresponding to the specified output parameter in the training script to a specified OBS path. To specify multiple parameters, use **--output output1=obs://bucket/output1 --output output2=obs://bucket/output2**.

Parameter	Data Type	Mandatory	Description
--input	String	No	Training input. After this parameter is specified, the training job will download the data from OBS to the training container and transfer the data storage path to the training script through the specified parameter. To specify multiple parameters, use **--input data_path1=obs://bucket/data1 --input data_path2=obs://bucket/data2**.
--env-variables	String	No	Environment variables input during training. To specify multiple parameters, use **--env-variables ENV1=env1 --env-variables ENV2=env2**.
--parameters	String	No	Training input parameters. To specify multiple parameters, use **--parameters "--epoch 0 --pretrained"**.
--framework-type	String	No	Framework type selected for a training job.
--framework-version	String	No	Framework version selected for a training job.
-q / --quiet	Bool	No	Whether to exit directly without printing the job status synchronously after a training job is submitted.
--workspace-id	String	No	Workspace where a training job is deployed. The default value is **0**.
--policy	String	No	Training resource flavor mode. The options are **regular**, **economic**, **turbo**, and **auto**.
--volumes	String	No	EFS disks to be mounted. To specify multiple parameters, use **--volumes**. "local_path=/xx/yy/zz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/" -volumes "local_path=/xxx/yyy/zzz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/"

Example: Submitting a Training Job Based on a Preset ModelArts Image

Submit a training job by specifying the **OPTIONS** parameter.

```
ma-cli ma-job submit --code-dir obs://your-bucket/mnist/code/ \
                --boot-file main.py \
```



```
                --framework-type PyTorch \
                --working-dir /home/ma-user/modelarts/user-job-dir/code \
                --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```

Example of **train.yaml** using a preset image:

```
# Example .ma/train.yaml (preset image)
# pool_id: pool_xxxx
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
working-dir: /home/ma-user/modelarts/user-job-dir/code
framework-type: PyTorch
framework-version: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
boot-file: main.py
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
output:
   - name: output_dir
     obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
   - name: data_url
     obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
   - epoch: 10
   - learning_rate: 0.01
   - pretrained:

##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
  - efs:
     local_path: /xx/yy/zz
     read_only: false
     nfs_server_path: xxx.xxx.xxx.xxx:/
```

Example: Using a Custom Image to Create a Training Job

Submit a training job by specifying the **OPTIONS** parameter.

```
ma-cli ma-job submit --image-url atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e \
                --code-dir obs://your-bucket/mnist/code/ \
                --user-command "export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH &&
cd /home/ma-user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/
python main.py" \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```


Example of **train.yaml** using a custom image:

```
# Example .ma/train.yaml (custom image)
image-url: atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e
user-command: export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH && cd /home/ma-
user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python main.py
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
output:
  - name: output_dir
    obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
  - name: data_url
    obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
  - epoch: 10
  - learning_rate: 0.01
  - pretrained:

##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
 - efs:
    local_path: /xx/yy/zz
    read_only: false
    nfs_server_path: xxx.xxx.xxx.xxx:/
```

Using ma-cli ma-job get-log to Obtain ModelArts Training Job Logs

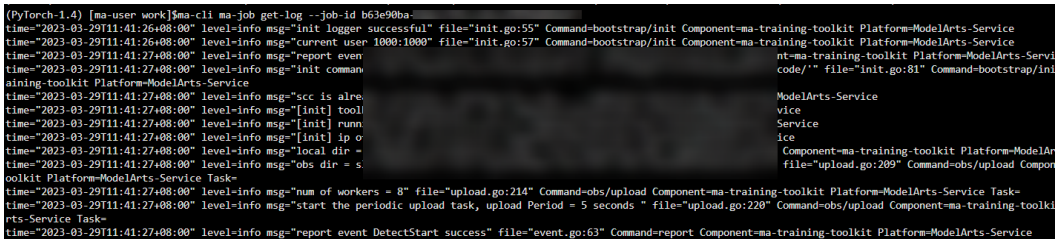
Run the **ma-cli ma-job get-log** command to obtain ModelArts training job logs.

```
$ ma-cli ma-job get-log -h
Usage: ma-cli ma-job get-log [OPTIONS]

  Get job log details.

  Example:

  # Get job log by job id
  ma-cli ma-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job details by job id.  [required]
  -t, --task-id TEXT      Get training job details by task id (default "worker-0").
  -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.
```


Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the job whose logs are to be obtained.
-t / --task-id	String	No	ID of the task whose logs are to be obtained. The default value is **work-0**.

Example: Obtain logs of a specified training job.

```
ma-cli ma-job get-log --job-id b63e90baxxx
```



Using ma-cli ma-job get-event to Obtain ModelArts Training Job Events

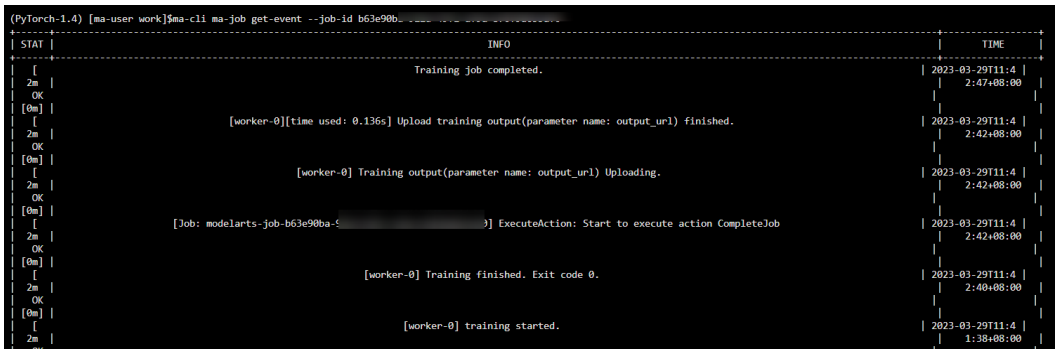
Run the **ma-cli ma-job get-event** command to obtain ModelArts training job events.

```
$ ma-cli ma-job get-event -h
Usage: ma-cli ma-job get-event [OPTIONS]

  Get job running event.

  Example:

  # Get training job running event
  ma-cli ma-job get-event --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -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.
```

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the job whose events are to be obtained.

Example: Obtain events of a specified training job.

```
ma-cli ma-job get-event --job-id b63e90baxxx
```

Using ma-cli ma-job get-engine to Obtain the AI Engines Used by ModelArts Training Jobs

Run the **ma-cli ma-job get-engine** command to obtain the AI engines used by ModelArts training jobs.

```
$ ma-cli ma-job get-engine -h
Usage: ma-cli ma-job get-engine [OPTIONS]

  Get job engine info.

  Example:

  # Get training job engines
  ma-cli ma-job get-engine

Options:
  -v, --verbose           Show detailed information about training engines.
  -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 6-35 Parameters

Parameter	Data Type	Mandatory	Description
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain the AI engines used by training jobs.

```
ma-cli ma-job get-engine
```



```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-engine
+-------+-----------------------------------------------------------+-----------------------+----------+
| index |                         engine id                         |      engine name      | run user |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   1   |                    caffe-1.0.0-python2.7                  |         Caffe         |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   2   |                    horovod-cp36-tf-1.16.2                 |        Horovod        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   3   | horovod_0.20.0-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   4   | horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   5   |                 kungfu-0.2.2-tf-1.13.1-python3.6          |         KungFu        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   6   |        mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64        |          MPI          |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   7   |       mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   8   |       mindspore_1.8.0-cann_5.1.2-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   9   |       mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   10  |                    mxnet-1.2.1-python3.6                  |         MXNet         |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   11  |    optverse_0.2.0-pygrassland_1.1.0-py_3.7-ubuntu_18.04-x86_64    |           OR          |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   12  |                      pytorch-cp36-1.0.0                   |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   13  |                      pytorch-cp36-1.3.0                   |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   14  |                      pytorch-cp36-1.4.0                   |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   15  |       pytorch_1.8.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   16  |        pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   17  |       pytorch_1.8.1-cann_5.1.2-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   18  |       pytorch_1.8.1-cann_6.0.0-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   19  |        pytorch_1.8.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   20  |        pytorch_1.8.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   21  |        pytorch_1.9.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   22  |                       ray-cp36-0.7.4                      |          Ray          |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
```

Using ma-cli ma-job get-flavor to Obtain the Resource Flavors Used by ModelArts Training Jobs

Run the **ma-cli ma-job get-flavor** command to obtain the resource flavors used by ModelArts training jobs.

```
$ ma-cli ma-job get-flavor -h
Usage: ma-cli ma-job get-flavor [OPTIONS]

  Get job flavor info.

  Example:

  # Get training job flavors
  ma-cli ma-job get-flavor

Options:
  -t, --flavor-type [CPU|GPU|Ascend]
                                  Type of training job flavor.
  -v, --verbose                   Show detailed information about training flavors.
  -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 6-36 Parameters

Parameter	Data Type	Mandatory	Description
-t / --flavor-type	String	No	Resource flavor type. If this parameter is not specified, all resource flavors are returned by default.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain the resource flavors and types of training jobs.

```
ma-cli ma-job get-flavor
```

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-flavor
+-------+-------------------------------+-----------------------------------+-------------+
| index |          flavor id            |           flavor name             | flavor type |
+-------+-------------------------------+-----------------------------------+-------------+
|   1   |    modelarts.kat1.8xlarge     | Computing NPU(8*Ascend) instance  |    Ascend   |
+-------+-------------------------------+-----------------------------------+-------------+
|   2   |    modelarts.kat1.xlarge      |  Computing NPU(Ascend) instance   |    Ascend   |
+-------+-------------------------------+-----------------------------------+-------------+
|   3   |     modelarts.vm.cpu.2u       |    Computing CPU(2U) instance     |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   4   |     modelarts.vm.cpu.8u       |    Computing CPU(8U) instance     |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   5   |  modelarts.vm.cpu.8u16g.119   |    Computing CPU(8U) instance     |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   6   |   modelarts.vm.v100.large     |   Computing GPU(V100) instance    |     GPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   7   | modelarts.vm.v100.large.free  |   Computing GPU(V100) instance    |     GPU     |
+-------+-------------------------------+-----------------------------------+-------------+
```

Using ma-cli ma-job stop to Stop a ModelArts Training Job

Run the **ma-cli ma-job stop** command to stop a training job with a specified job ID.

```
$ ma-cli ma-job stop -h
Usage: ma-cli ma-job stop [OPTIONS]

  Stop training job by job id.

  Example:

  Stop training job by job id
  ma-cli ma-job stop --job-id ${job_id}
```



```
Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -y, --yes               Confirm stop operation.
  -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 6-37 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of a ModelArts training job
-y / --yes	Bool	No	Whether to forcibly stop a training job

Example: Stop a running training job.

```
ma-cli ma-job stop --job-id efd3e2f8xxx
```

6.8.7 ma-cli dli-job Commands for Submitting DLI Spark Jobs

```
$ma-cli dli-job -h
Usage: ma-cli dli-job [OPTIONS] COMMAND [ARGS]...

  DLI spark job submission and query job details.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
  get-job       Get DLI spark job details.
  get-log       Get DLI spark log details.
  get-queue     Get DLI spark queues info.
  get-resource  Get DLI resources info.
  stop          Stop DLI spark job by job id.
  submit        Submit dli spark batch job.
  upload        Upload local file or OBS object to DLI resources.
```

Table 6-38 Commands for submitting DLI Spark jobs

Command	Description
get-job	Obtain DLI Spark jobs and their details.
get-log	Obtain runtime logs of a DLI Spark job.
get-queue	Obtain DLI queues.
get-resource	Obtain DLI group resources.
stop	Stop a DLI Spark job.
submit	Submit a DLI Spark job.

Command	Description
upload	Upload local files or OBS files to a DLI group.

Using ma-cli dli-job get-job to Obtain a DLI Spark Job

Run the **ma-cli dli-job get-job** command to obtain the DLI Spark jobs or details about a job.

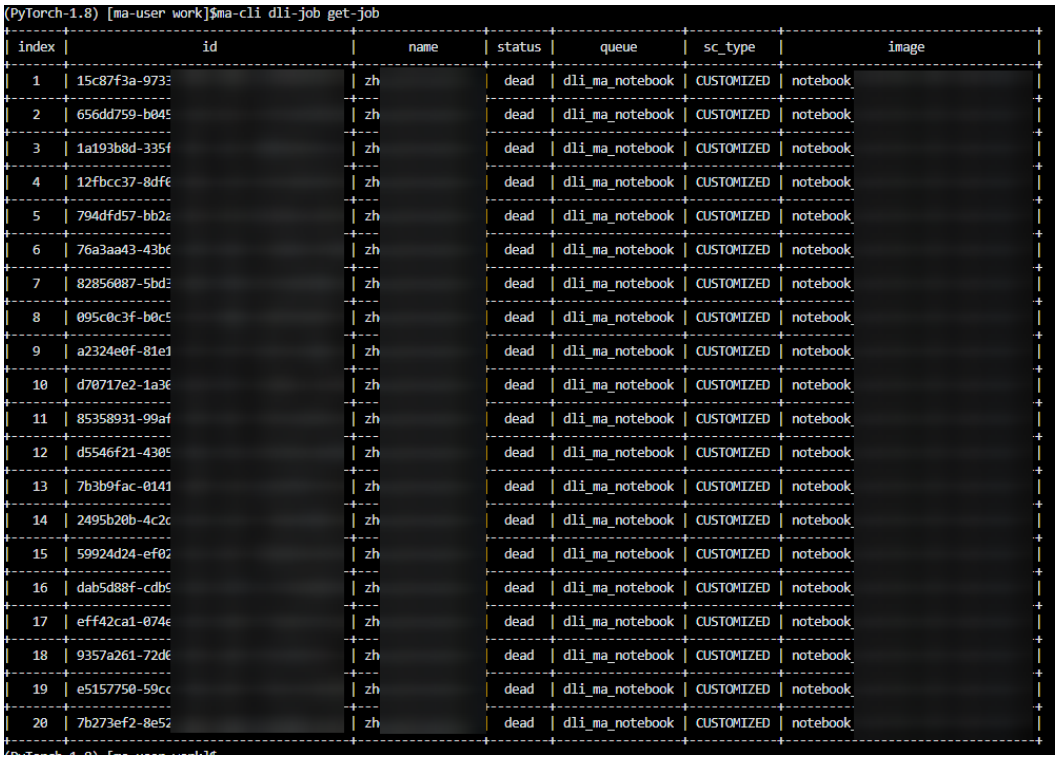
```
ma-cli dli-job get-job -h
Usage: ma-cli dli-job get-job [OPTIONS]

  Get DLI Spark details.

  Example:

  # Get DLI Spark job details by job name
  ma-cli dli-job get-job -n ${job_name}

  # Get DLI Spark job details by job id
  ma-cli dli-job get-job -i ${job_id}

  # Get DLI Spark job list
  ma-cli dli-job get-job --page-size 5 --page-num 1

Options:
  -i, --job-id TEXT              Get DLI Spark job details by job id.
  -n, --job-name TEXT            Get DLI Spark job details by job name.
  -pn, --page-num INTEGER RANGE  Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -v, --verbose                  Show detailed information about DLI Spark job details.
  -C, --config-file PATH         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 6-39 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	No	ID of the DLI Spark job whose details are to be obtained.
-n / --job-name	String	No	Name of the DLI Spark job to be queried or name keyword used to filter DLI Spark jobs.
-pn / --page-num	Int	No	Page number. The default value is **1**.
-ps / --page-size	Int	No	Number of jobs displayed on each page. The default value is **20**.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain all DLI Spark jobs.


```
ma-cli dli-job get-job
```



index	id	name	status	queue	sc_type	image
1	15c87f3a-973	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
2	656dd759-b04	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
3	1a193b8d-335f	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
4	12fbcc37-8df	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
5	794dfd57-bb2	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
6	76a3aa43-43b	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
7	82856087-5bd	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
8	095c0c3f-b0c	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
9	a2324e0f-81e1	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
10	d70717e2-1a3	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
11	85358931-99af	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
12	d5546f21-430	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
13	7b3b9fac-0141	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
14	2495b20b-4c2c	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
15	59924d24-ef02	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
16	dab5d88f-cdb	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
17	eff42ca1-074	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
18	9357a261-72d	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
19	e5157750-59cc	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook
20	7b273ef2-8e52	zh	dead	dli_ma_notebook	CUSTOMIZED	notebook

Using ma-cli dli-job submit to Submit a DLI Spark Job

Run the **ma-cli dli-job submit** command to submit a DLI Spark job.

When running this command, use the **YAML_FILE** parameter to specify the path to the configuration file of the target job. If this parameter is not specified, the configuration file is empty. The configuration file is in YAML format, and its parameters are values of **OPTIONS** in the command. If you specify both the **YAML_FILE** and the **OPTIONS** parameters, the **OPTIONS** value will overwrite the same items in the configuration file.

Command parameters preview

```
ma-cli dli-job submit -h
Usage: ma-cli dli-job submit [OPTIONS] [YAML_FILE]...

  Submit DLI Spark job.

  Example:

  ma-cli dli-job submit  --name test-spark-from-sdk
                         --file test/sub_dli_task.py
                         --obs-bucket dli-bucket
                         --queue dli_test
                         --spark-version 2.4.5
                         --driver-cores 1
                         --driver-memory 1G
                         --executor-cores 1
                         --executor-memory 1G
                         --num-executors 1

Options:
  --file TEXT                 Python file or app jar.
  -cn, --class-name TEXT      Your application's main class (for Java / Scala apps).
```



```
--name TEXT                  Job name.
--image TEXT                 Full swr custom image path.
--queue TEXT                 Execute queue name.
-obs, --obs-bucket TEXT      DLI obs bucket to save logs.
-sv, --spark-version TEXT    Spark version.
-st, --sc-type [A|B|C]       Compute resource type.
--feature [basic|custom|ai]  Type of the Spark image used by a job (default: basic).
-ec, --executor-cores INTEGER  Executor cores.
-em, --executor-memory TEXT  Executor memory (eg. 2G/2048MB).
-ne, --num-executors INTEGER  Executor number.
-dc, --driver-cores INTEGER  Driver cores.
-dm, --driver-memory TEXT    Driver memory (eg. 2G/2048MB).
--conf TEXT                  Arbitrary Spark configuration property (eg. <PROP=VALUE>).
--resources TEXT             Resources package path.
--files TEXT                 Files to be placed in the working directory of each executor.
--jars TEXT                  Jars to include on the driver and executor class paths.
-pf, --py-files TEXT         Python files to place on the PYTHONPATH for Python apps.
--groups TEXT                User group resources.
--args TEXT                  Spark batch job parameter args.
-q, --quiet                  Exit without waiting after submit successfully.
-C, --config-file PATH       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.
```

YAML file preview

```
# dli-demo.yaml
name: test-spark-from-sdk
file: test/sub_dli_task.py
obs-bucket: ${your_bucket}
queue: dli_notebook
spark-version: 2.4.5
driver-cores: 1
driver-memory: 1G
executor-cores: 1
executor-memory: 1G
num-executors: 1

## [Optional]
jars:
  - ./test.jar
  - obs://your-bucket/jars/test.jar
  - your_group/test.jar

## [Optional]
files:
  - ./test.csv
  - obs://your-bucket/files/test.csv
  - your_group/test.csv

## [Optional]
python-files:
  - ./test.py
  - obs://your-bucket/files/test.py
  - your_group/test.py

## [Optional]
resources:
  - name: your_group/test.py
    type: pyFile
  - name: your_group/test.csv
    type: file
  - name: your_group/test.jar
    type: jar
  - name: ./test.py
    type: pyFile
  - name: obs://your-bucket/files/test.py
    type: pyFile
```



```
## [Optional]
groups:
  - group1
  - group2
```

Example of submitting a DLI Spark job by specifying **OPTIONS**:

```
$ ma-cli dli-job submit --name test-spark-from-sdk \
                        --file test/sub_dli_task.py \
                        --obs-bucket ${your_bucket} \
                        --queue dli_test \
                        --spark-version 2.4.5 \
                        --driver-cores 1 \
                        --driver-memory 1G \
                        --executor-cores 1 \
                        --executor-memory 1G \
                        --num-executors 1
```

Table 6-40 Parameters

Parameter	Data Type	Mandatory	Description
YAML_FILE	String	No	Local path to the configuration file of a DLI Spark job. If this parameter is not specified, the configuration file is empty.
--file	String	Yes	Program entry file. It can be a local file path, an OBS path, or the name of a JAR or PyFile package that has been uploaded to the DLI resource management system.
-cn / --class_name	String	Yes	Java/Spark main class of the batch processing job.
--name	String	No	Specified job name. The value consists of a maximum of 128 characters.
--image	String	No	Path to a custom image in the format of "Organization name/Image name:Image version". This parameter is valid only when **feature** is set to **custom**. You can use this parameter with the **feature** parameter to specify a custom Spark image for running a job.
-obs / --obs-bucket	String	No	OBS bucket for storing a Spark job. Configure this parameter when you need to save jobs. It can also be used as a transit station for submitting local files to **resources**.
-sv/ --spark-version	String	No	Spark version used by a job.

Parameter	Data Type	Mandatory	Description
-st / `--sc-type	String	No	If the current Spark version is 2.3.2, leave this parameter blank. If the current Spark version is 2.3.3, configure this parameter when **feature** is set to **basic** or **ai**. If this parameter is not specified, the default Spark version 2.3.2 will be used.
--feature	String	No	Job feature, indicating the type of the Spark image used by a job. The default value is **basic**. • **basic**: A base Spark image provided by DLI is used. • **custom**: A custom Spark image is used. • **ai**: An AI image provided by DLI is used.
--queue	String	No	Queue name. Set this parameter to the name of a created DLI queue. The queue must be of the common type. For details about how to obtain a queue name, see **Table 6-42**.
-ec / --executor-cores	String	No	Number of CPU cores of each Executor in a Spark application. This configuration will replace the default setting in **sc_type**.
-em / --executor-memory	String	No	Executor memory of a Spark application, for example, 2 GB or 2048 MB. This configuration will replace the default setting in **sc_type**. The unit must be provided. Otherwise, the startup fails.
-ne / --num-executors	String	No	Number of Executors in a Spark application. This configuration will replace the default setting in **sc_type**.
-dc / --driver-cores	String	No	Number of CPU cores of the Spark application driver. This configuration will replace the default setting in **sc_type**.
-dm / --driver-memory	String	No	Driver memory of a Spark application, for example, 2 GB or 2048 MB. This configuration will replace the default setting in **sc_type**. The unit must be provided. Otherwise, the startup fails.
--conf	Array of String	No	**batch** configuration. For details, see **Spark Configuration**. To specify multiple parameters, use **--conf conf1 --conf conf2**.

Parameter	Data Type	Mandatory	Description
--resources	Array of String	No	Name of a resource package, which can be a local file, OBS path, or a file that has been uploaded to the DLI resource management system. To specify multiple parameters, use **--resources resource1 --resources resource2**.
--files	Array of String	No	Name of the file package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--files file1 --files file2**.
--jars	Array of String	No	Name of the JAR package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--jars jar1 --jars jar2**.
-pf /--python-files	Array of String	No	Name of the PyFile package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--python-files py1 --python-files py2**.
--groups	Array of String	No	Resource group name. To specify multiple parameters, use **--groups group1 --groups group2**.
--args	Array of String	No	Parameters passed to the main class, that is, program parameters. To specify multiple parameters, use **--args arg1 --args arg2**.
-q / --quiet	Bool	No	Whether to exit directly without printing the job status synchronously after a DLI Spark job is submitted.

Example

- Submit a DLI Spark job using **YAML_FILE**.
  ```
  $ma-cli dli-job submit dli_job.yaml
  ```



```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit ./dli-job.yaml
[ OK ] Current DLI job id is: 01b698b8-9fd6-4a8e-bc3c-6821c6405b14
[ OK ] starting
[ OK ] running
[ OK ] success
[ OK ] Successfully submit DLI spark job [ 01b698b8-9fd6-4a8e-bc3c-6821c6405b14 ].
```

- Submit a DLI Spark job by specifying the **OPTIONS** parameter.

```
$ma-cli dli-job submit --name test-spark-from-sdk \
>                      --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                      --queue dli_ma_notebook \
>                      --spark-version 2.4.5 \
>                      --driver-cores 1 \
>                      --driver-memory 1G \
>                      --executor-cores 1 \
>                      --executor-memory 1G \
>                      --num-executors 1
```

```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit --name test-spark-from-sdk \
>                      --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                      --queue dli_ma_notebook \
>                      --spark-version 2.4.5 \
>                      --driver-cores 1 \
>                      --driver-memory 1G \
>                      --executor-cores 1 \
>                      --executor-memory 1G \
>                      --num-executors 1
[ OK ] Current DLI job id is: ae856c20-e9ae-49ca-8409-7a02652297b8
[ OK ] starting
```

Using ma-cli dli-job get-log to Obtain Run Logs of a DLI Spark Job

Run the **ma-cli dli-job get-log** command to obtain the run logs of a DLI Spark job.

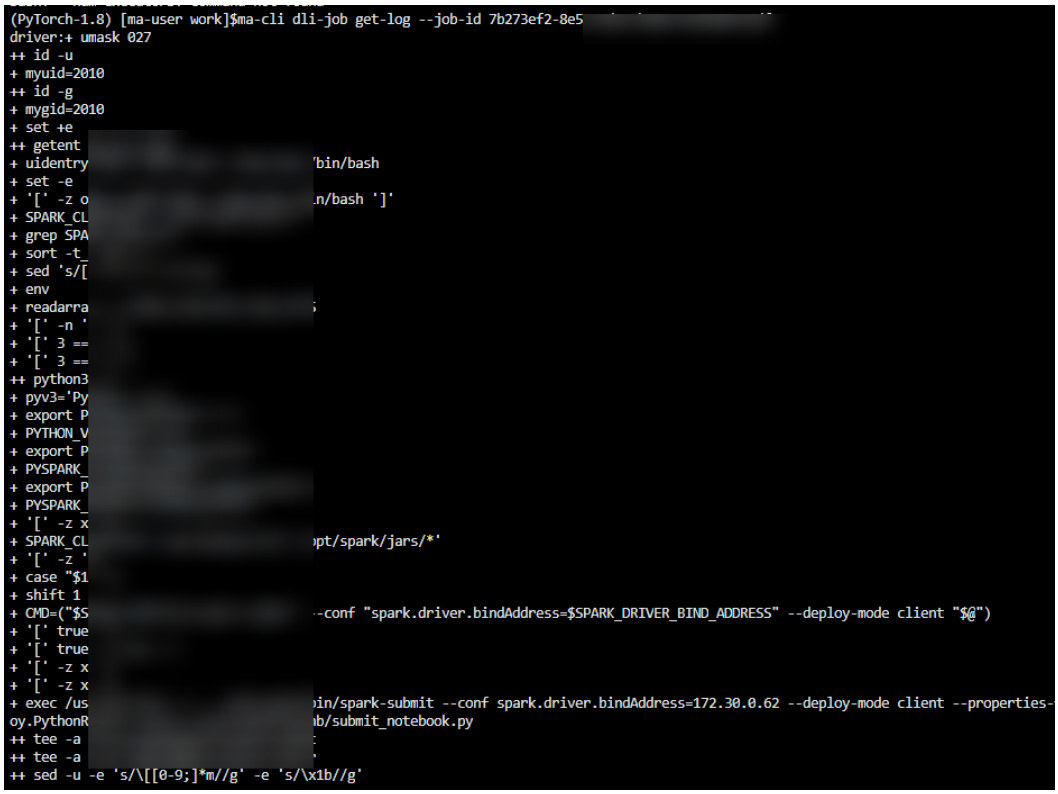
```
$ ma-cli dli-job get-log -h
Usage: ma-cli dli-job get-log [OPTIONS]

  Get DLI spark job log details.

  Example:

  # Get job log by job id
  ma-cli dli-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get DLI spark job details by job id.  [required]
  -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 6-41 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the DLI Spark job whose logs are to be obtained.

Example: Obtain the run logs of a specified DLI Spark job.

```
ma-cli dli-job get-log --job-id ${your_job_id}
```

Using ma-cli dli-job get-queue to Obtain a DLI Queue

Run the **ma-cli dli-job get-queue** command to obtain a DLI queue.

```
ma-cli dli-job get-queue -h
Usage: ma-cli dli-job get-queue [OPTIONS]

  Get DLI queues info.

  Example:

  # Get DLI queue details by queue name
  ma-cli dli-job get-queue --queue-name $queue_name}

Options:
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -n, --queue-name TEXT           Get DLI queue details by queue name.
  -t, --queue-type [sql|general|all]
                                  DLI queue type (default "all").
  -tags, --tags TEXT              Get DLI queues by tags.
  -C, --config-file PATH          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 6-42 Parameters

Parameter	Data Type	Mandatory	Description
-n / --queue-name	String	No	Name of the DLI queue to be obtained.

Parameter	Data Type	Mandatory	Description
-t / --queue-type	String	No	Type of the DLI queue to be obtained. The value can be **sql**, **general**, or **all**. The default value is **all**.
-tags / --tags	String	No	Tags of the DLI queue to be obtained.
-pn / --page-num	Int	No	DLI queue page number. The default value is **1**.
-ps / --page-size	Int	No	Number of DLI queues displayed on each page. The default value is **20**.

Example: Query information about the queue named **dli_ma_notebook**.

```
ma-cli dli-job get-queue --queue-name dli_ma_notebook
```

```
(PyTorch-1.8) [ma-user work]$ ma-cli dli-job get-queue --queue-name dli_ma_notebook
{'chargingMode': 1,
 'create_time': 1668585417422,
 'cuCount': 16,
 'cu_spec': 16,
 'description': '',
 'enterprise_project_id': '0',
 'is_success': True,
 'message': '',
 'owner': '                    ',
 'queueName': 'dli_ma_notebook',
 'queueType': 'general',
 'queue_id': 8____,
 'resource_id': '242e7af8-c9d1                    e',
 'resource_mode': 1,
 'resource_type': 'container',
 'support_spark_versions': ['2.3.2', '2.4.5', '3.1.1']}
```

Using ma-cli dli-job get-resource to Obtain DLI Group Resources

Run the **ma-cli dli-job get-resource** command to obtain details about DLI resources, such as the resource name and resource type.

```
$ ma-cli dli-job get-resource -h
Usage: ma-cli dli-job get-resource [OPTIONS]

  Get DLI resource info.

  Example:

  # Get DLI resource details by resource name
  ma-cli dli-job get-resource --resource-name ${resource_name}

Options:
  -n, --resource-name TEXT        Get DLI resource details by resource name.
  -k, --kind [jar|pyFile|file|modelFile]
                                  DLI resources type.
  -g, --group TEXT                Get DLI resources by group.
```



```
-tags, --tags TEXT            Get DLI resources by tags.
-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 6-43 Parameters

Parameter	Data Type	Mandatory	Description
-n / --resource-name	String	No	Resource name of the DLI group resources to be queried.
-k / --kind	String	No	Resource type of the DLI group resources to be queried. The type can be JAR, PyFile, file, or modelFile.
-g / --group	String	No	Group name of the DLI group resources to be queried.
-tags / --tags	String	No	Tags of the DLI group resources to be queried.

Example: Obtain information about all DLI group resources.

```
ma-cli dli-job get-resource
```


Using ma-cli dli-job upload to Upload Files to a DLI Group

Run the **ma-cli dli-job upload** command to upload local files or OBS files to a DLI group.

```
$ ma-cli dli-job upload -h
Usage: ma-cli dli-job upload [OPTIONS] PATHS...

  Upload DLI resource.

  Tips: --obs-path is need when upload local file.

  Example:

  # Upload an OBS path to DLI resource
  ma-cli dli-job upload obs://your-bucket/test.py -g test-group --kind pyFile

  # Upload a local path to DLI resource
  ma-cli dli-job upload ./test.py -g test-group -obs ${your-bucket} --kind pyFile

  # Upload local path and OBS path to DLI resource
  ma-cli dli-job upload ./test.py obs://your-bucket/test.py -g test-group -obs ${your-bucket}
```



```
Options:
  -k, --kind [jar|pyFile|file]  DLI resources type.
  -g, --group TEXT              DLI resources group.
  -tags, --tags TEXT            DLI resources tags, follow --tags `key1`=`value1`.
  -obs, --obs-bucket TEXT       OBS bucket for upload local file.
  -async, --is-async            whether to upload resource packages in asynchronous mode. The default value is
False.
  -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 6-44 Parameters

Parameter	Data Type	Mandatory	Description
PATHS	String	Yes	Paths to the local files or OBS files to be uploaded to a DLI group. Multiple paths can be specified at a time.
-k / --kind	String	No	Type of the file to be uploaded, which can be JAR, PyFile, or file.
-g / --group	String	No	Name of the DLI group to which files are to be uploaded.
-tags / --tags	String	No	Tag of the files to be uploaded.
-obs / --obs-bucket	String	No	If the files to be uploaded contain a local path, specify an OBS bucket for transit.
-async / --is-async	Bool	No	Whether to asynchronously upload files. This method is recommended.

Example

- Upload local files to a DLI group.
  ma-cli dli-job upload ./test.py -obs ${your-bucket} --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload ./test.py -obs obs://[redacted] --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

- Upload OBS files to a DLI group.
  ma-cli dli-job upload obs://your-bucket/test.py --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload obs://[redacted]/test.py --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

Using ma-cli dli-job stop to Stop a DLI Spark Job

Run the **ma-cli dli-job stop** command to stop a DLI Spark job.

```
$ ma-cli dli-job stop -h
Usage: ma-cli dli-job stop [OPTIONS]

  Stop DLI spark job by job id.

  Example:
```



```
 Stop training job by job id
 ma-cli dli-job stop --job-id ${job_id}

Options:
 -i, --job-id TEXT       Get DLI spark job event by job id.  [required]
 -y, --yes               Confirm stop operation.
 -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 6-45 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	DLI Spark job ID
-y / --yes	Bool	No	Whether to forcibly stop a specified DLI Spark job

Example

```
ma-cli dli-job stop -i ${your_job_id}
```

```
(PyTorch-1.8) [ma-user work]$ma-cli dli-job stop -i 4b[redacted]861
[ WARN ] Spark job 4b[redacted]1 will be stopped, do you want to continue (y for confirm)? [y/N]: y
[ OK ] Successfully stop spark batch job [ 4b[redacted]1 ].
```

6.8.8 Using ma-cli to Copy OBS Data

Run the **ma-cli obs-copy [SRC] [DST]** command to copy a local file to an OBS folder or an OBS file or folder to a local path.

```
$ma-cli obs-copy -h
Usage: ma-cli obs-copy [OPTIONS     ] SRC DST

 Copy file or directory between OBS and local path. Example:

 # Upload local file to OBS path
 ma-cli obs-copy ./test.zip obs://your-bucket/copy-data/

 # Upload local directory to OBS path
 ma-cli obs-copy ./test/ obs://your-bucket/copy-data/

 # Download OBS file to local path
 ma-cli obs-copy obs://your-bucket/copy-data/test.zip ./test.zip

 # Download OBS directory to local path
 ma-cli obs-copy obs://your-bucket/copy-data/ ./test/

Options:
 -d, --drop-last-dir     Whether to drop last directory when copy folder. if True, the last directory of the
source folder will not copy to the destination folder.  [default: False]
 -C, --config-file PATH  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 6-46 Parameters

Parameter	Type	Mandatory	Description
-d / --drop-last-dir	Bool	No	If you specify this parameter, the last-level directory of the source folder will not be copied to the destination folder. This parameter is valid only for copying folders.

Examples

Upload a file to OBS.

```
$ ma-cli obs-copy ./test.csv obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/test.csv ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Upload a folder to **obs://${your_bucket}/test-copy/data/**.

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/data/ ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Upload a folder to **obs://${your_bucket}/test-copy/** with **--drop-last-dir** specified.

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/ --drop-last-dir
[ OK ] local src path: [ /home/ma-user/work/data ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Download a folder from OBS to a local disk.

```
$ ma-cli obs-copy obs://${your_bucket}/test-copy/ ~/work/test-data/
[ OK ] obs src path: [ obs://${your_bucket}/test-copy/ ]
[ OK ] local dst path: [ /home/ma-user/work/test-data/ ]
```

6.9 Using Moxing Commands in a Notebook Instance

6.9.1 Introduction to MoXing Framework

MoXing Framework provides basic common components for MoXing. For example, it facilitates access to Huawei Cloud 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 OBS Python SDKs 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.

OBS provides many functions and tools, such as SDK, API, OBS Console, and OBS Browser. ModelArts mox.file provides a set of APIs for accessing OBS. The APIs simulate the operations of a local file system, allowing users to operate OBS files conveniently. 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/')
```

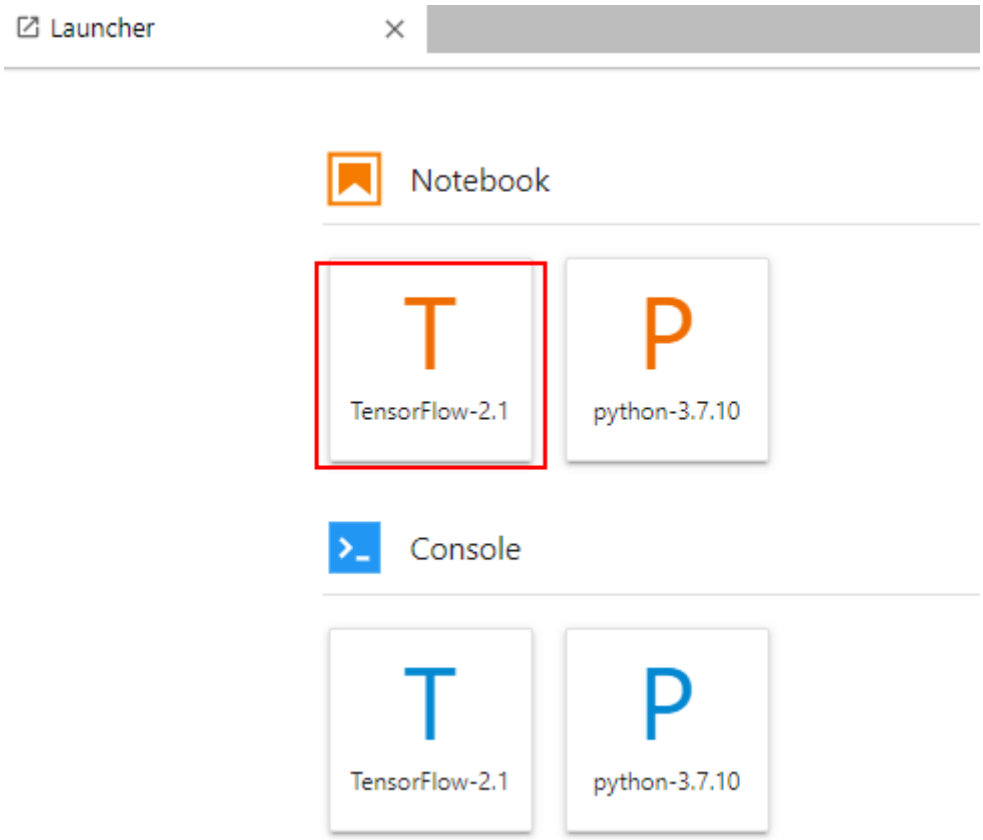
6.9.2 Getting Started

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

Logging In to ModelArts and Creating a Notebook Instance

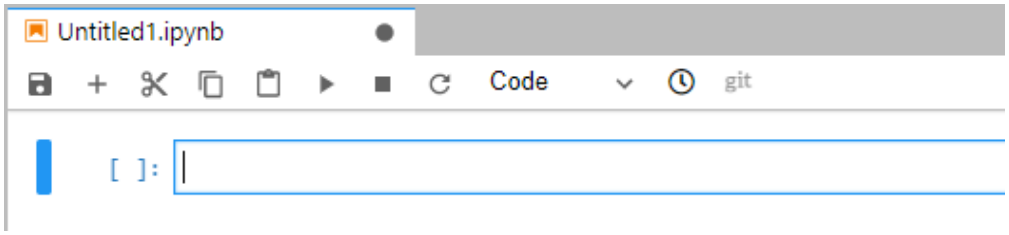
1. Log in to the ModelArts management console. In the left navigation pane, choose **DevEnviron > Notebook** to access the **Notebook** page.
2. Click **Create**. On the **Create Notebook** page that is displayed, create a notebook instance by referring to **Creating a Notebook Instance**.
3. After a 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** tab page of JupyterLab, for example, click **TensorFlow** to create a file for encoding.

Figure 6-152 Selecting an AI engine



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

Figure 6-153 Encoding page



Calling mox.file.

Enter the following code to implement the following simple functions:

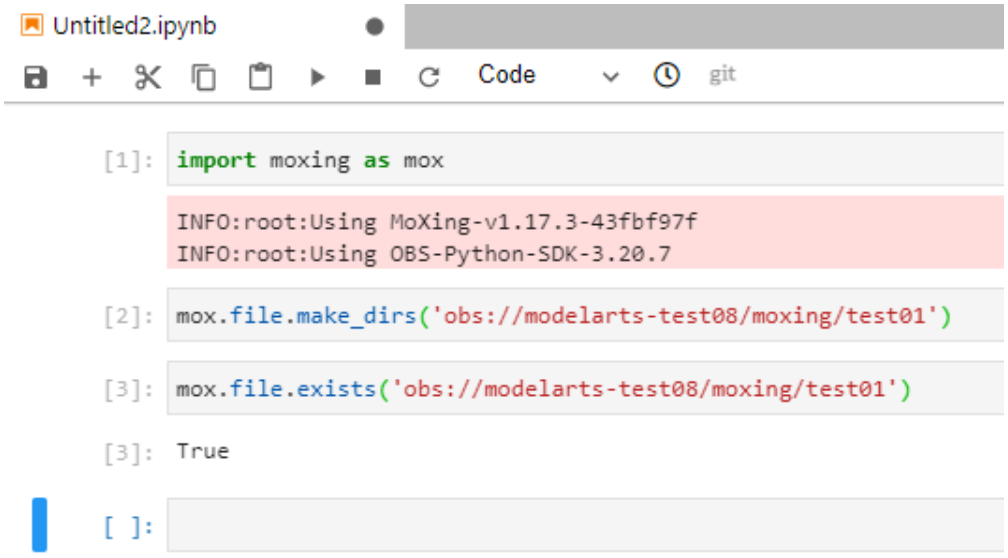
1. Introduce MoXing Framework.
2. Create the **test01** folder in the existing **modelarts-test08/moxing** directory.
3. Check whether the **test01** folder exists. If the folder exists, the preceding operation is successful.

```
import moxing as mox

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

Figure 6-154 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 6-154 Example



6.9.3 Introducing MoXing Framework

Before using MoXing Framework, you need to introduce the MoXing Framework module at the beginning of the code.

Introducing MoXing Framework

Run the following code to import the MoXing 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. Use the following Python code to configure the environment variable. You need to configure the environment variables before importing MoXing.

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


6.9.4 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 6-47 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

6.9.5 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')
    ```

6.9.6 Sample Code for Advanced Applications

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 an OBS file is read, an HTTP connection is called to read the network stream. You need to close the file after the file is read. 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 an API That Supports OBS Paths

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,)))
```


7 Data Management

7.1 Introduction to Data Preparation

NOTE

Data management is being upgraded and is invisible to users who have not used data management.

The driving forces behind AI are computing power, algorithms, and data. Data quality affects model precision. Generally, a large amount of high-quality data is more likely to train a high-precision AI model. Models trained using normal data achieves 85% to 90% accuracy, while commercial applications have higher requirements. If you want to improve the model accuracy to 96% or even 99%, a large amount of high-quality data is required. In this case, the data must be more refined, scenario-based, and professional. The preparation of a large amount of high-quality data has become a challenging issue in AI development.

ModelArts is a one-stop AI development platform that supports AI lifecycle development, including data processing, algorithm development, model training, and model deployment. In addition, ModelArts provides AI Gallery that can be used to share data, algorithms, and models. ModelArts data management provides end-to-end data preparation, processing, and labeling.

ModelArts data management provides the following functions for you to obtain high-quality AI data:

- Data acquisition
  - Allows you to import data from OBS, MRS, DLI, and GaussDB(DWS).
  - Provides 18+ data augmentation operators to increase data volume for training.
- Improved data quality
  - Allows you to preview various formats of data including images, text, audios, and videos, helping you identify data quality.
  - Allows you to filter data by multiple search criteria, such as sample attributes and labeling information.
  - Provides 12+ labeling tools for refined, scenario-based, and professional data labeling.

  - Performs feature analysis based on samples and labeling results, helping you understand data quality.
- More efficient data preparation
  - Allows you to manage data by version for more efficient data management.
  - Provides capabilities such as interactive labeling and auto labeling for more efficient data labeling.
  - Enables team labeling and team labeling management for labeling a large amount of data.

7.2 Getting Started

This section uses preparing data for training an object detection model as an example to describe how to analyze and label sample data. During actual service development, you can select one or more data management functions to prepare data based on service requirements. The operation process is as follows:

- **Making Preparations**
- **Creating a Dataset**
- **Analyzing Data**
- **Labeling Data**
- **Publishing Data**
- **Exporting Data**

NOTE

Data management is being upgraded and is invisible to users who have not used data management.

Preparations

Before using data management of ModelArts, complete the following preparations:

When using data management, ModelArts needs to access dependent services such as OBS. Therefore, grant permissions on the **Global Configuration** page. For details, see **Configuring Agency Authorization (Recommended)**.

Creating a Dataset

In this example, an OBS path is used as the input path to create a dataset. Perform the following operations to create an object detection dataset and import the data to the dataset:

Step 1 Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.

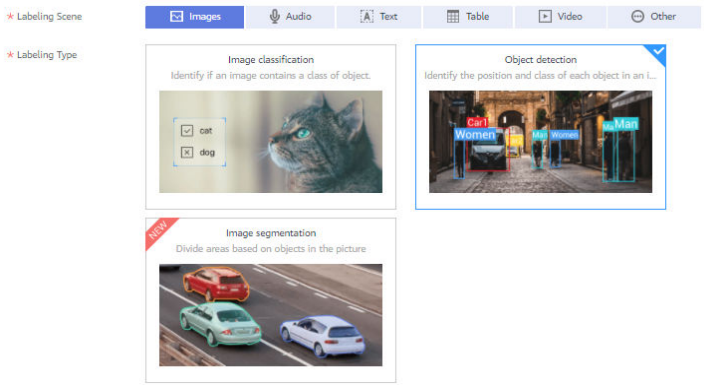
Step 2 Click **Create**. On the **Create Dataset** page, create a dataset based on the data type and data labeling requirements.

1. Set the basic information, the name and description of the dataset.

Figure 7-1 Basic information of a dataset

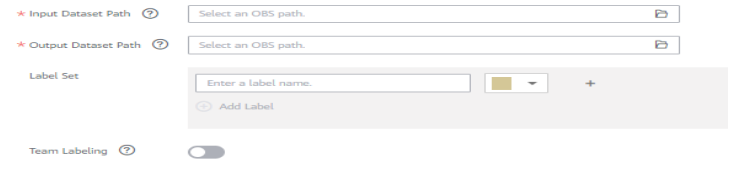
2. Set labeling scene and type. In this example, choose **Images** and **Object detection**.

Figure 7-2 Dataset labeling scene and type



3. Select an OBS path as **Input Dataset Path**, and select another OBS path as **Output Dataset Path**.

Figure 7-3 Input and output dataset path



4. After setting the parameters, click **Create** in the lower right corner to create a dataset.

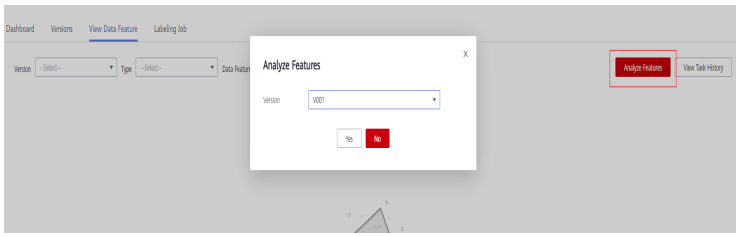
----End

Analyzing Data

After a dataset is created, you can perform data analysis based on image features, such as blurs and brightness, to better understand the data quality and determine whether the dataset meets your algorithm and model requirements.

1. Create a feature analysis task.
   a. Before performing feature analysis, publish a dataset version. On the dataset **Dashboard** page, click **Publish** in the upper right corner to publish a new version of the dataset.
   b. After the version is published, go to the **Dashboard** page. Click **View Data Feature** and **Feature Analysis**. In the displayed dialog box, select the newly published dataset version and click **OK** to start feature analysis.

Figure 7-4 Starting feature analysis



c. View the task progress.

You can click **View Task History** to view the task progress. When the task status changes to **Successful**, the task is complete.

Figure 7-5 Feature analysis progress

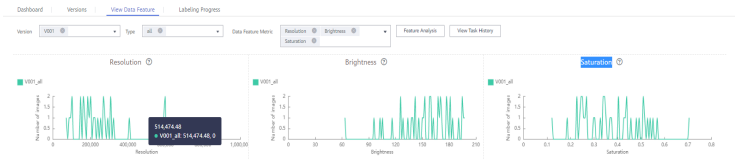
Dataset Version	Task ID	Created	Duration(hh:...	Status
V002	ZJrO7W8735...	Mar 16, 2022 ...	00:01:23	Successful

2. View feature analysis results.

After feature analysis is complete, you can select **Version**, **Type**, and **Data Feature Metric** on the **View Data Feature** tab page. Then, the selected versions and metrics are displayed on the page. The displayed chart helps you understand data distribution for a better understanding of your data.

- **Version**: Select one or more versions for comparison.
- **Type**: Select types to be analyzed. The values **all**, **train**, **eval**, and **inference** are available for you to select. They indicate all, training, evaluation, and inference, respectively.
- **Data Feature Metric**: Select the metrics to be displayed. For details about the metrics, see **Data feature metrics**.

Figure 7-6 Viewing feature analysis results



In feature analysis results, for example, image brightness distribution is uneven, which means images of a certain brightness are lacking. This greatly affects model training. In this case, increase images of that brightness to make data more even for subsequent model building.

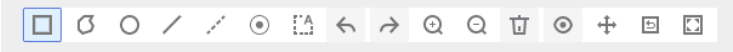
Labeling Data

- Manual labeling

a. On the **Unlabeled** tab page, click an image. The system automatically directs you to the page for labeling the image.

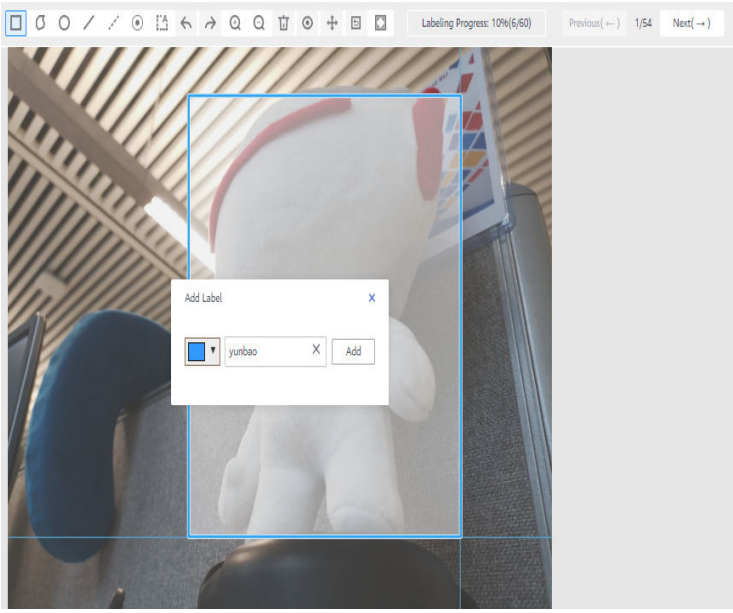
b. On the toolbar of the labeling page, select a proper labeling tool. In this example, a rectangle is used for labeling.

Figure 7-7 Labeling tools



c. Drag the mouse to select an object, enter a new label name in the displayed text box. If labels already exist, select one from the drop-down list box. Click **Add**.

Figure 7-8 Adding an object detection label



d. Click **Back to Data Labeling Preview** in the upper left part of the page to view the labeling information. In the dialog box that is displayed, click **Yes** to save the labeling settings. The selected image is automatically moved to the **Labeled** tab page. On the **Unlabeled** and **All** tab pages, the labeling information is updated along with the labeling process, including the added label names and the number of images for each label.

- Auto labeling

  Auto labeling allows you to automatically label remaining data after a small amount of data is manually labeled.

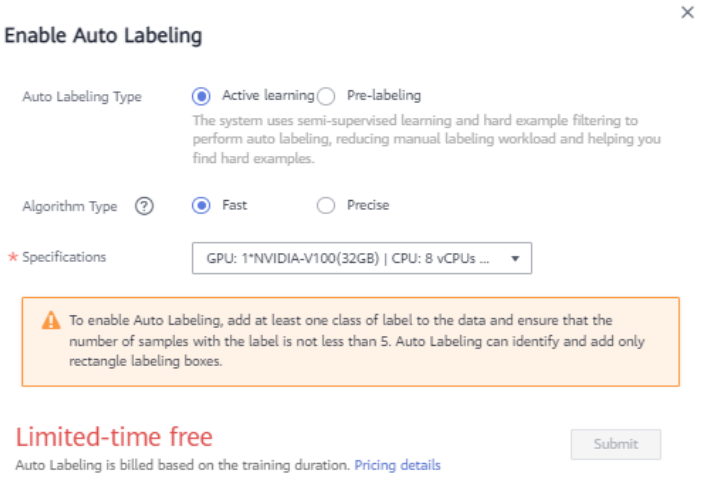
  a. On the dataset details page, click **Auto Label** in the upper right corner.

  b. In the **Enable Auto Labeling** dialog box, set the following parameters and click **Submit**.

     - **Auto Labeling Type**: **Active learning**
     - **Algorithm Type**: **Fast**

     Retain the default values of other parameters.

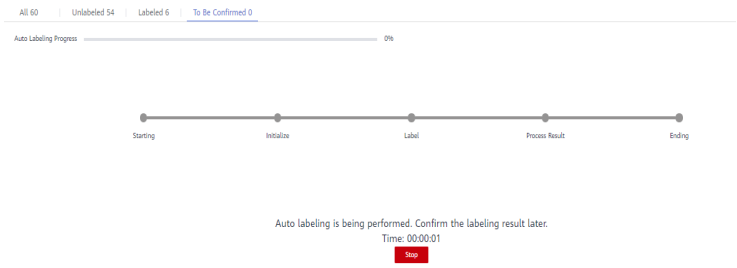
Figure 7-9 Starting auto labeling



c. View auto labeling progress.

After auto labeling is started, you can view the task progress on the **To Be Confirmed** tab page. After a task is complete, you can view the automatically labeled data on the **To Be Confirmed** tab page.

Figure 7-10 Viewing auto labeling progress

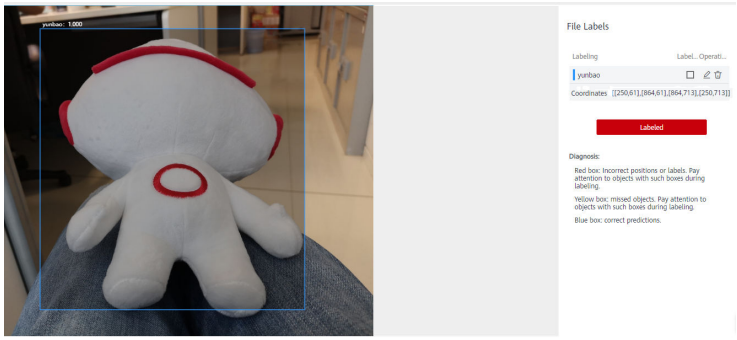


d. Confirm auto labeling results.

After auto labeling is complete, click the image on the **To be confirmed** tab page. On the labeling details page, you can view or modify the auto labeling result.

For correct labeling, click **Labeled** on the right. For wrong labeling, correct wrong labels. For auto labeling of object detection datasets, confirm images one by one. Ensure that all images are confirmed and go to the next step.

Figure 7-11 Confirming auto labeling results



Publishing Data

ModelArts training management allows you to create training jobs using ModelArts datasets or files in an OBS directory. If a dataset is used as the data source of a training job, specify a dataset and version. Therefore, you must have published a dataset version. For details, see **Publishing a Data Version**.

NOTE

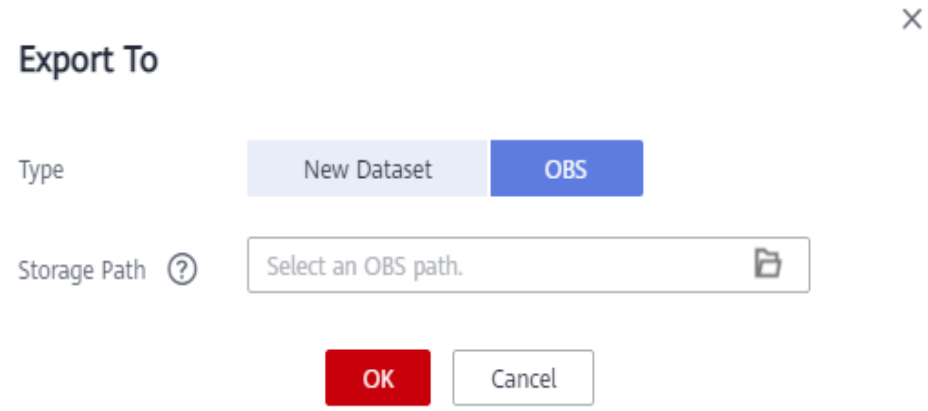
Data that is from the same source and labeled in different batches are differentiated by version. This facilitates subsequent model building and development. You can select specified versions.

Exporting Data

ModelArts training management allows you to create training jobs using ModelArts datasets or files in an OBS directory. If you create a training job using an OBS directory, export the prepared data to OBS.

1. Export data to OBS.
   a. On the dataset details page, select or filter the data to be exported, and click **Export** in the upper right corner.
   b. Set **Type** to **OBS**, enter related information, and click **OK**.
      Storage Path: path where the data to be exported is stored. You are advised not to save data to the input or output path of the current dataset.

Figure 7-12 Exporting to OBS



c. After the data is exported, view it in the specified path.

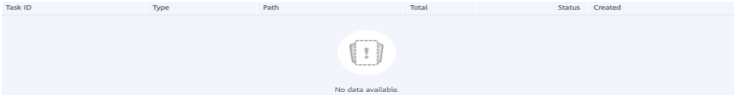
2. View task history.

   After exporting data, you can view the export task details in **Export History**.

   a. On the dataset details page, click **Export History** in the upper right corner.

   b. In the **View Task History** dialog box, view the export task history of the current dataset. You can view the task ID, creation time, export type, export path, total number of exported samples, and export status.

   Figure 7-13 Export history



7.3 Creating a Dataset

Before using ModelArts to prepare data, create a dataset. Then, you can perform operations on the dataset, such as importing data, analyzing data, and labeling data.

7.3.1 Dataset Overview

NOTE

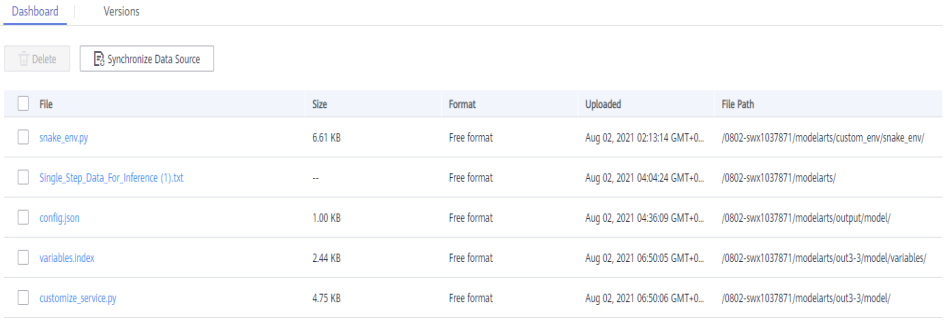
Data management is being upgraded and is invisible to users who have not used data management.

Dataset Types

ModelArts supports the following types of datasets:

- Images: in .jpg, .png, .jpeg, or .bmp format for image classification, image segmentation, and object detection
- Audio: in .wav format for sound classification, speech labeling, and speech paragraph labeling
- Text: in .txt or .csv format for text classification, named entity recognition, and text triplet labeling
- Video: in .mp4 format for video labeling
- Free format: allows data in any format. Labeling is not available for free format data. The free format applies if labeling is not required or needs to be customized. Select this format if your data is in multiple formats or your data is not in any of the preceding formats.

Figure 7-14 Example of a dataset in free format



- Tables

  Table: applies to structured data processing such as tables. The file format can be CSV. Tables cannot be labeled but you can preview up to 100 data records in a table.

Dataset Functions

Different types of datasets support different functions, such as auto labeling and team labeling. For details, see **Table 7-1**.

Table 7-1 Functions supported by different types of datasets

Dataset Type	Labeling Type	Creating a Dataset	Importing Data	Exporting Data	Publishing a Dataset	Modifying a Dataset	Managing Dataset Versions	Auto Grouping	Data Features
Image	Image classification	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
	Object detection	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
	Image segmentation	Supported	Supported	Supported	Supported	Supported	Supported	Supported	N/A
Audio	Sound classification	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A

Dataset Type	Labeling Type	Creating a Dataset	Importing Data	Exporting Data	Publishing a Dataset	Modifying a Dataset	Managing Dataset Versions	Auto Grouping	Data Features
	Speech labeling	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
	Speech paragraph labeling	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
Text	Text classification	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
	Named entity recognition	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
	Text triplet	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
Video	Video labeling	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A
Free format	Free format	Supported	N/A	_	Supported	Supported	Supported	N/A	N/A
Table	Table	Supported	Supported	N/A	Supported	Supported	Supported	N/A	N/A

Specifications Restrictions

- The maximum numbers of samples and labels in a single text, video, or audio database other than a table dataset are 1,000,000 and 10,000, respectively.
- The maximum size of a sample in a single text, video, or audio database other than an image dataset is 5 GB.
- The maximum size of an image for object detection, image segmentation, or image classification is 25 MB.

- The maximum size of a manifest file is 5 GB.
- The maximum size of a text file in a line is 100 KB.
- The maximum size of a labeling result file is 100 MB.

7.3.2 Creating a Dataset

Before using ModelArts to manage data, create a dataset. Then, you can perform operations on the dataset, such as labeling data, importing data, and publishing the dataset. This section describes how to create a dataset of the non-table type (image, audio, text, video, and free format) and table type.

NOTE

Data management is being upgraded and is invisible to users who have not used data management.

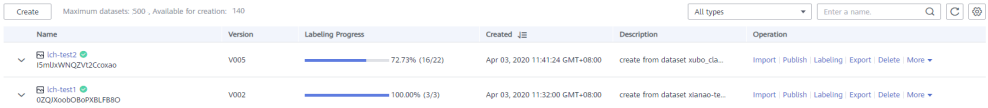
Prerequisites

- You have been authorized to access OBS. To do so, click the **Settings** page in the navigation pane of the ModelArts management console and add access authorization using an agency.
- OBS buckets and folders for storing data are available. In addition, the OBS buckets and ModelArts are in the same region. OBS parallel file systems are not supported. Select object storage.
- OBS buckets are not encrypted. ModelArts does not support encrypted OBS buckets. When creating an OBS bucket, do not enable bucket encryption.

Image, Audio, Text, Video, and Free Format

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.

   Figure 7-15 Dataset management page

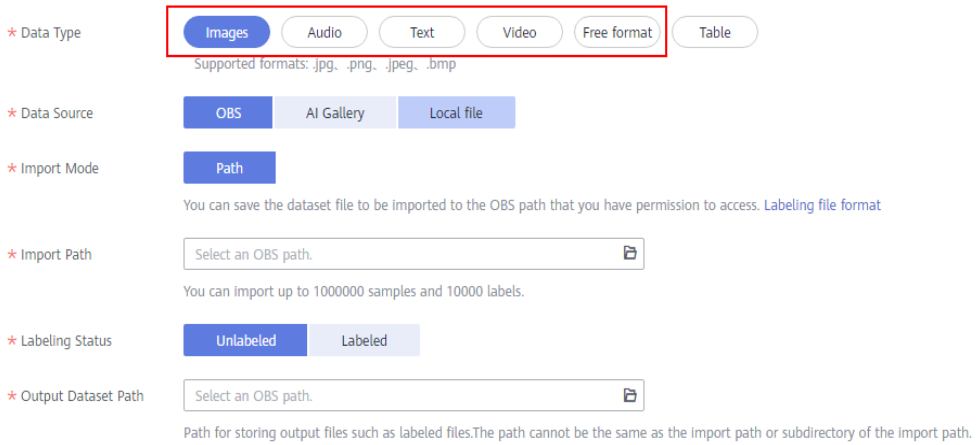
   

   NOTE

   The number of datasets that can be created under an account in a region is limited. For details, see the number displayed on the **Dataset** page.

2. Click **Create**. On the **Create Dataset** page, create a dataset based on the data type and data labeling requirements.

Figure 7-16 Parameter settings

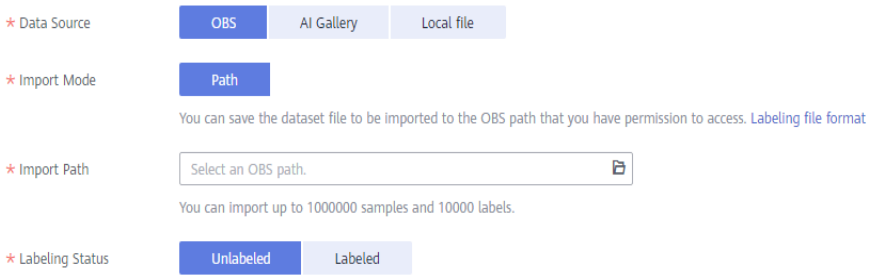


- **Name**: name of the dataset, which is customizable
- **Description**: details about the dataset
- **Data Type**: Select a data type based on your needs.
- **Data Source**

  i. Importing data from OBS

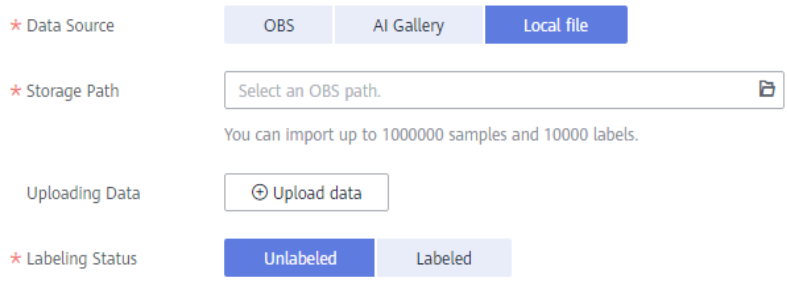
  If data is available in OBS, select **OBS** for **Data Source**, and configure other mandatory parameters. The labeling formats of the input data vary depending on the dataset type. For details about the labeling formats supported by ModelArts, see **Importing Data**.

  Figure 7-17 Importing data from OBS

  ii. Importing data from a local path

  If data is not stored in OBS and the required data cannot be downloaded from AI Gallery, ModelArts enables you to upload the data from a local path. Before uploading data, configure **Storage Path** and **Labeling Status**. Click **Upload data** to select the local file for uploading. Select a labeling format when the labeling status is **Labeled**. The labeling formats of the input data vary depending on the dataset type. For details about the labeling formats supported by ModelArts, see **Importing Data**.



- For more details about parameters, see **Table 7-2**.

Table 7-2 Dataset parameters

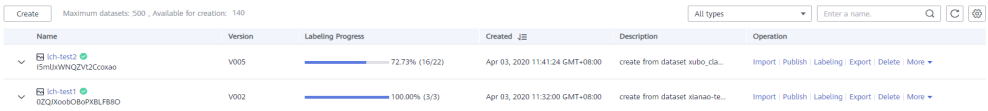
Parameter	Description
Import Path	OBS path from which your data is to be imported. This path is used as the data storage path of the dataset. **NOTE** OBS parallel file systems are not supported. Select an OBS bucket. When you create a dataset, data in the OBS path will be imported to the dataset. If you modify data in OBS, the data in the dataset will be inconsistent with that in OBS. As a result, certain data may be unavailable. To modify data in a dataset, follow the operations provided in **Import Mode** or **Importing Data from an OBS Path**. If the numbers of samples and labels of the dataset exceed quotas, importing the samples and labels will fail.
Labeling Status	Labeling status of the selected data, which can be **Unlabeled** or **Labeled**. If you select **Labeled**, specify a labeling format and ensure the data file complies with format specifications. Otherwise, the import may fail. Only image (object detection, image classification, and image segmentation), audio (sound classification), and text (text classification) labeling tasks support the import of labeled data.
Output Dataset Path	OBS path where your labeled data is stored. **NOTE** • Ensure that your OBS path name contains letters, digits, and underscores (_) and does not contain special characters, such as ~'@#$%^&*{}[]:;+=<>/ and spaces. • The dataset output path cannot be the same as the data input path or subdirectory of the data input path. • It is a good practice to select an empty directory as the dataset output path. • OBS parallel file systems are not supported. Select an OBS bucket.

3. After setting the parameters, click **Submit**.

Table

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.

   Figure 7-18 Dataset management page

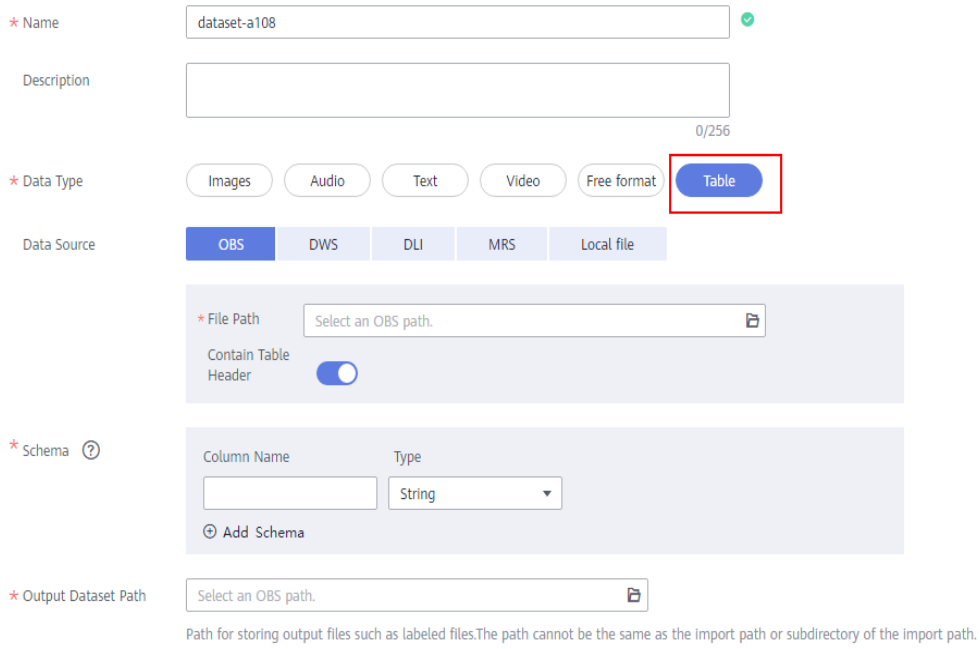
   

   NOTE

   The number of datasets that can be created under an account in a region is limited. For details, see the number displayed on the **Dataset** page.

2. Click **Create**. On the **Create Dataset** page, create a table dataset based on the data type and data labeling requirements.

   Figure 7-19 Parameters of a table dataset

   - **Name**: name of the dataset, which is customizable
   - **Description**: details about the dataset
   - **Data Type**: Select a data type based on your needs.
   - For more details about parameters, see **Table 7-3**.

Table 7-3 Dataset parameters

Parameter	Description
Data Source (OBS)	• **File Path**: Browse all OBS buckets of the account and select the directory where the data file to be imported is located. • **Contain Table Header**: This setting is enabled by default, indicating that the imported file contains table headers. – If the original table contains table headers and this setting is enabled, first rows (table header) of the imported file are used as column names. You do not need to modify the schema information. – If the original table does not contain table headers, you need to disable this setting and change column names in **Schema** to **attr_1**, **attr_2**, ..., and **attr_n**. **attr_n** is the last column, indicating the prediction column. For details about OBS functions, see ***Object Storage Service Console Operation Guide***.
Data Source (DWS)	• **Cluster Name**: All DWS clusters of the current account are automatically displayed. Select the required DWS cluster from the drop-down list. • **Database Name**: Enter the name of the database where the data is located based on the selected DWS cluster. • **Table Name**: Enter the name of the table where the data is located based on the selected database. • **User Name**: Enter the username of the DWS cluster administrator. • **Password**: Enter the password of the DWS cluster administrator. For details about DWS functions, see ***Data Warehouse Service User Guide***. **NOTE** To import data from DWS, use DLI functions. If you do not have the permission to access DLI, create a DLI agency as prompted.

Parameter	Description
Data Source (DLI)	• **Queue Name**: All DLI queues of the current account are automatically displayed. Select the required queue from the drop-down list. • **Database Name**: All databases are displayed based on the selected queue. Select the required database from the drop-down list. • **Table Name**: All tables in the selected database are displayed. Select the required table from the drop-down list. For details about DLI functions, see ***Data Lake Insight User Guide***.
Data Source (MRS)	• **Cluster Name**: All MRS clusters of the current account are automatically displayed. However, streaming clusters do not support data import. Select the required cluster from the drop-down list. • **File Path**: Enter the HDFS file path based on the selected cluster. • **Contain Table Header**: If this setting is enabled, the imported file contains table headers. For details about MRS functions, see ***MapReduce Service User Guide***.
Local file	**Storage Path**: Select an OBS path.
Schema	Names and types of table columns, which must be the same as those of the imported data. Set the column name based on the imported data and select the column type. For details about the supported types, see **Table 7-4**. Click **Add Schema** to add a new record. When creating a dataset, you must specify a schema. Once created, the schema cannot be modified. When data is imported from OBS, the schema of the CSV file in the file path is automatically obtained. If the schemas of multiple CSV files are inconsistent, an error will be reported. **NOTE** After you select data from OBS, column names in **Schema** are automatically displayed, which is the first-row data of the table by default. To ensure the correct prediction code, you need to change column names in **Schema** to **attr_1**, **attr_2**, ..., and **attr_n**. **attr_n** is the last column, indicating the prediction column.

Parameter	Description
Output Dataset Path	OBS path for storing table data. The data imported from the data source is stored in this path. The path cannot be the same as the file path in the OBS data source or subdirectories of the file path. After a table dataset is created, the following four directories are automatically generated in the storage path: • **annotation**: version publishing directory. Each time a version is published, a subdirectory with the same name as the version is generated in this directory. • **data**: data storage directory. Imported data is stored in this directory. • **logs**: directory for storing logs. • **temp**: temporary working directory.

Table 7-4 Schema data types

Type	Description	Storage Space	Range
String	String type	N/A	N/A
Short	Signed integer	2 bytes	-32768 to 32767
Int	Signed integer	4 bytes	-2147483648 to 2147483647
Long	Signed integer	8 bytes	-9223372036854775808 to 9223372036854775807
Double	Double-precision floating point	8 bytes	N/A
Float	Single-precision floating point	4 bytes	N/A
Byte	Signed integer	1 byte	-128 to 127
Date	Date type in the format of "yyyy-MM-dd", for example, 2014-05-29	N/A	N/A
Timestamp	Timestamp that represents date and time in the format of "yyyy-MM-dd HH:mm:ss"	N/A	N/A

Type	Description	Storage Space	Range
Boolean	Boolean type	1 byte	TRUE/FALSE

NOTE

When using a CSV file, pay attention to the following:

- When the data type is set to **String**, the data in the double quotation marks is regarded as one record by default. Ensure the double quotation marks in the same row are closed. Otherwise, the data will be too large to display.
- If the number of columns in a row of the CSV file is different from that defined in the schema, the row will be ignored.

3. After setting the parameters, click **Submit**.

7.3.3 Modifying a Dataset

The basic information of a created dataset can be modified to keep pace with service changes.

Prerequisites

A created dataset is available.

Modifying the Basic Information of a Dataset

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, choose **More** > **Modify** in the **Operation** column of the target dataset.
3. Modify the basic information by referring to **Table 7-5** and click **OK**.

Figure 7-20 Modify Dataset

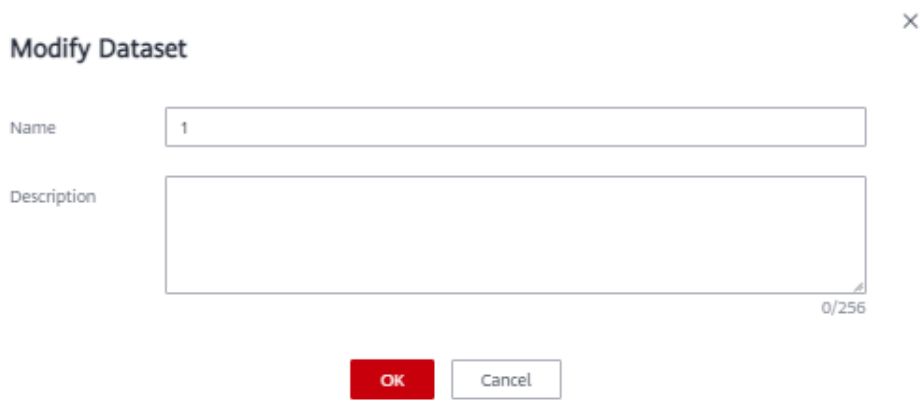


Table 7-5 Parameters

Parameter	Description
Name	Name of a dataset, which must be 1 to 64 characters long and start with a letter. Only letters, digits, underscores (_), and hyphens (-) are allowed. The name must start with a letter.
Description	Brief description of the dataset.

7.4 Importing Data

7.4.1 Introduction to Data Importing

After a dataset is created, you can import more data. ModelArts allows you to import data from different data sources.

- **Importing Data from OBS**
- **Importing Data from DLI**
- **Importing Data from MRS**
- **Importing Data from DWS**
- **Importing Data from Local Files**

ModelArts AI Gallery provides a large number of built-in datasets, including file and table datasets. You can download and use the built-in datasets from AI Gallery. You can also import your data to ModelArts.

File Data Sources

You can import data by downloading built-in datasets from AI Gallery, or from OBS or a local file. After the import, the data from the import path is automatically synchronized to the data source path of the dataset.

- **OBS**: Import data from an OBS path or a manifest file.
- **Local file**: Import local data that has been uploaded to an OBS path.

Table Data Sources

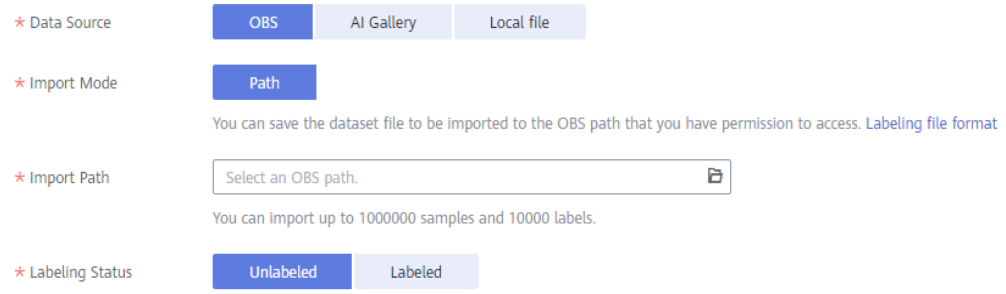
You can import data by downloading built-in datasets from AI Gallery, or from OBS, DWS, DLI, MRS, and local files.

Import Mode

There are five modes for importing data to a dataset.

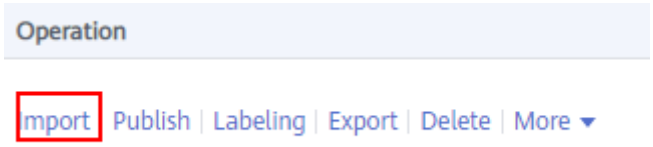
- When you create a dataset, select an import path. The data is automatically synchronized from the import path.

Figure 7-21 Importing data when creating a dataset



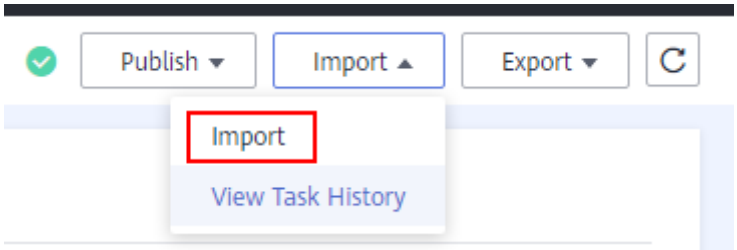
- After a dataset is created, click **Import** in the **Operation** column on the dataset list page.

Figure 7-22 Importing data on the dataset list page



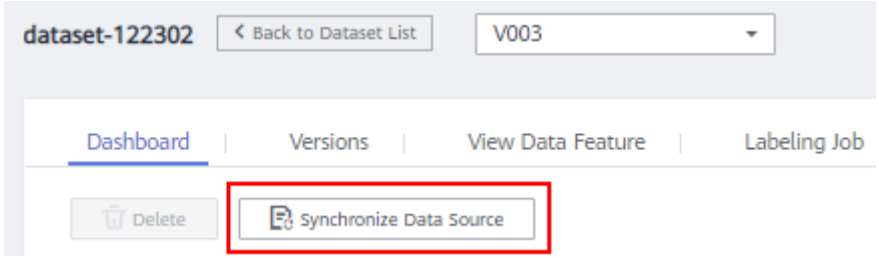
- On the dataset list page, click a dataset. On the dataset details page, choose **Import** > **Import**.

Figure 7-23 Importing data on the dataset details page



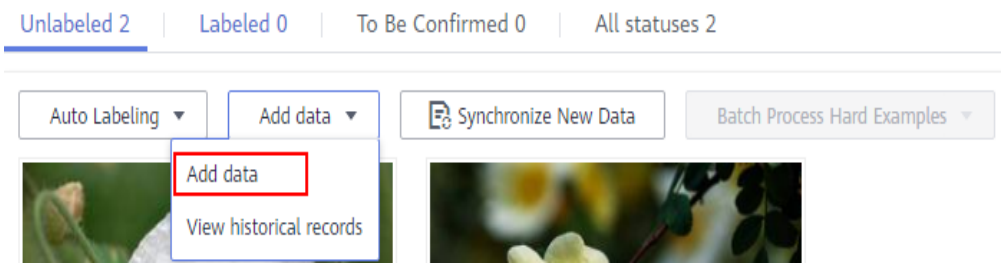
- On the dataset list page, click a dataset. On the dataset details page, click **Synchronize Data Source** to synchronize data from OBS.

Figure 7-24 Synchronizing data sources on the dataset details page



- Add data on the labeling job details page.

Figure 7-25 Adding data on the labeling job details page



7.4.2 Importing Data from OBS

7.4.2.1 Introduction to Importing Data from OBS

Import Modes

You can import data from OBS through an OBS path or a manifest file.

- OBS path: indicates that the dataset to be imported has been stored in an OBS path. In this case, select an OBS path that you can access. In addition, the directory structure in the OBS path must comply with the specifications. For details, see **Specifications for Importing Data from an OBS Directory**. This import mode is available only for the following types of datasets: **Image classification**, **Object detection**, **Text classification**, **Table**, and **Sound classification**. For other types of datasets, data can be imported only through a manifest file.
- Manifest file: indicates that the dataset file is in the manifest format and the manifest file has been uploaded to OBS. The manifest file defines the mapping between labeling objects and content. For details about the specifications of manifest files, see **Specifications for Importing a Manifest File**.

NOTE

Before importing an object detection dataset, ensure that the labeling range of the labeling file does not exceed the size of the original image. Otherwise, the import may fail.

Table 7-6 Import modes supported by datasets

Dataset Type	Labeling Type	From an OBS Path	From a Manifest File
Images	Image classification	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Image Classification**	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Image Classification**

Dataset Type	Labeling Type	From an OBS Path	From a Manifest File
	Object detection	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Object Detection**	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Object Detection**
	Image segmentation	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Image Segmentation**	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Image Segmentation**
Audio	Sound classification	Supported You can import unlabeled or labeled data. Follow the format specifications described in **Sound Classification**.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Sound Classification**
	Speech labeling	Supported You can import unlabeled data.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Speech Labeling**
	Speech paragraph labeling	Supported You can import unlabeled data.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Speech Paragraph Labeling**
Text	Text classification	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Text Classification**	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Text Classification**

Dataset Type	Labeling Type	From an OBS Path	From a Manifest File
	Named entity recognition	Supported You can import unlabeled data.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Named Entity Recognition**
	Text triplet	Supported You can import unlabeled data.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Text Triplet**
Video	Video labeling	Supported You can import unlabeled data.	Supported You can import unlabeled or labeled data. Format specifications of labeled data: **Video Labeling**
Other	Free format	Supported You can import unlabeled data.	N/A
Table	Table	Supported You can also import data from DWS, DLI, or MRS. Follow the format specifications described in **Tables**.	N/A

7.4.2.2 Importing Data from an OBS Path

Prerequisites

- A dataset is available.
- The data to be imported is stored in OBS. The manifest file is stored in OBS.
- The OBS bucket and ModelArts are in the same region and you can operate the bucket.

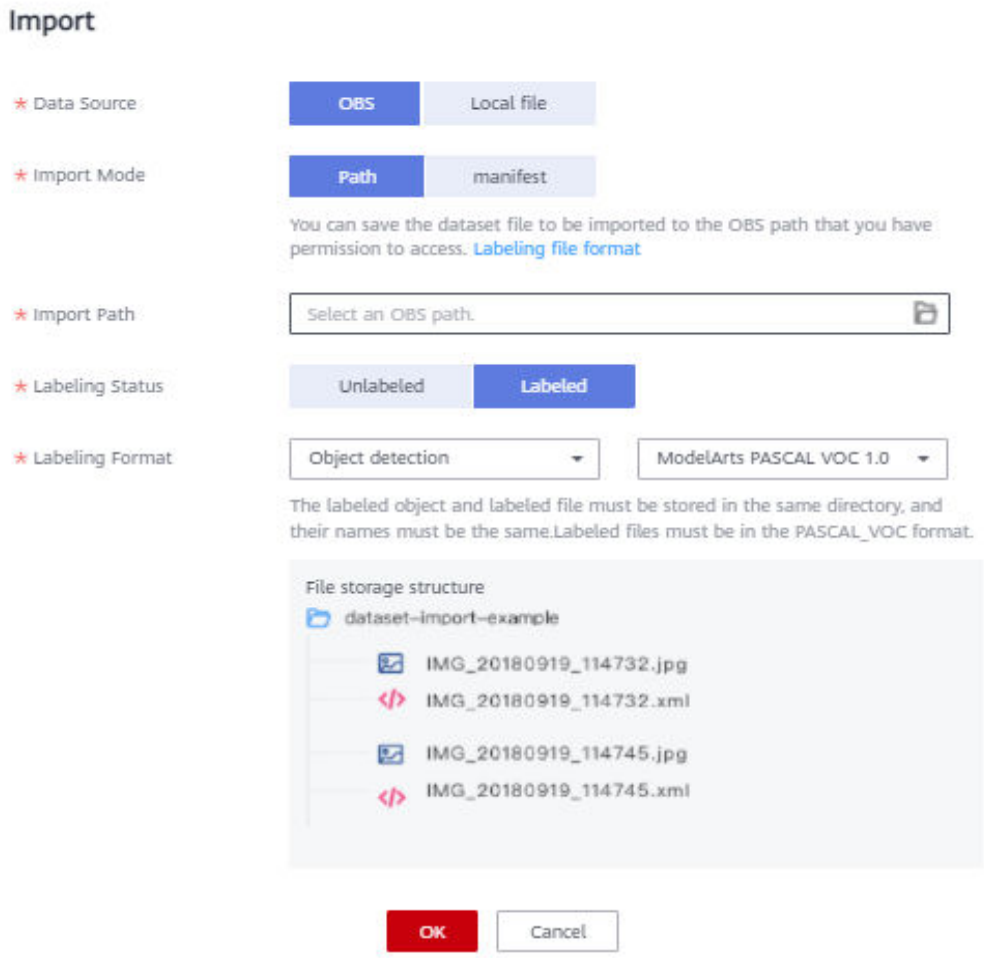
Importing File Data from an OBS Path

The parameters on the GUI for data import vary according to the dataset type. The following uses a dataset of the image classification type as an example.

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. Locate the row that contains the desired dataset and click **Import** in the **Operation** column. Alternatively, click the dataset name to go to the **Dashboard** tab page of the dataset, and click **Import** in the upper right corner.
3. In the **Import** dialog box, configure parameters as follows and click **OK**.
   - **Data Source**: **OBS**
   - **Import Mode**: **Path**
   - **Import Path**: OBS path for storing data
   - **Labeling Status**: **Labeled**
   - **Advanced Feature Settings**: disabled by default

     Import by Tag enables the system to automatically obtain the labels of the current dataset. Click **Add Label** to add a label. This parameter is optional. If **Import by Tag** is disabled, you can add or delete labels for imported data when labeling data.

Figure 7-26 Importing data from an OBS path



After the data is imported, it will be automatically synchronized to the dataset. On the **Datasets** page, click the dataset name to view its details and create a labeling job to label the data.

Labeling Status of File Data

The labeling status can be **Unlabeled** or **Labeled**.

- **Unlabeled**: Only the labeling object (such as unlabeled images or texts) is imported.
- **Labeled**: Both the labeling object and content are imported. Labeling content importing is not supported for datasets in free format.

  To ensure that the labeling content can be correctly read, you must store data in strict accordance with the specifications.

  If **Import Mode** is set to **Path**, store the data to be imported according to the labeling file specifications. For details, see **Specifications for Importing Data from an OBS Directory**.

  If **Import Mode** is set to **manifest**, the manifest file specifications must be met.

  NOTE

  - If the labeling status is set to **Labeled**, ensure that the folder or manifest file complies with the format specifications. Otherwise, the import may fail.
  - After the import of labeled data, check whether the imported data is in the labeled state.

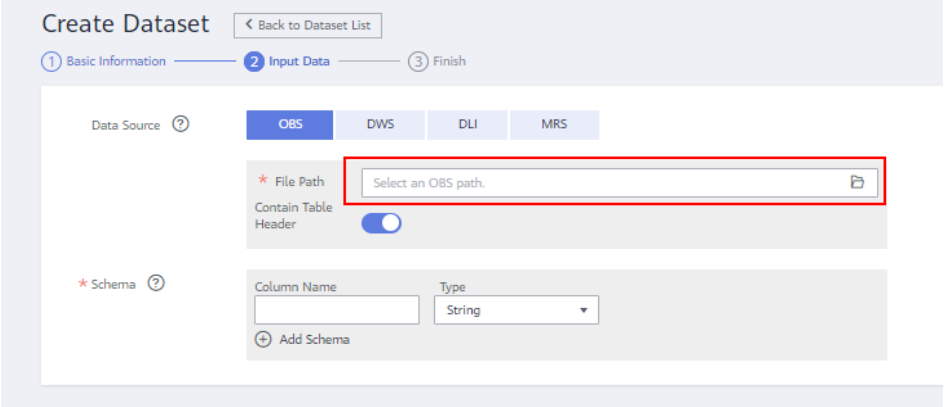
Importing a Table Dataset from OBS

ModelArts allows you to import table data (CSV files) from OBS.

Import description:

- The prerequisite for successful import is that the schema of the data source must be the same as that specified during dataset creation. The schema indicates column names and types of a table. Once specified during dataset creation, the values cannot be changed.
- When a CSV file is imported from OBS, the data type is not validated, but the number of columns must be the same as that in the schema of the dataset. If the data format is invalid, the data is set to null. For details, see **Table 7-4**.
- You must select the directory where the CSV file is stored. The number of columns in the CSV file must be the same as that in the dataset schema. The schema of the CSV file can be automatically obtained.

```
├─dataset-import-example
│      table_import_1.csv
│      table_import_2.csv
│      table_import_3.csv
│      table_import_4.csv
```

7.4.2.3 Specifications for Importing Data from an OBS Directory

When importing data from OBS, the data storage directory and file name must comply with the ModelArts specifications.

Only the following labeling types of data can be imported by **Labeling Format**: image classification, object detection, image segmentation, text classification, and sound classification.

A table dataset can import data from sources such as OBS, DWS, DLI, and MRS.

NOTE

- To import data from an OBS directory, you must have the read permission on the OBS directory.
- The OBS buckets and ModelArts must be in the same region.

Image Classification

Data for image classification can be stored in two formats:

Format 1: ModelArts imageNet 1.0

- Images with the same label must be stored in the same directory, with the label name as the directory name. If there are multiple levels of directories, the last level is used as the label name.

  In the following example, **Cat** and **Dog** are label names.

```
dataset-import-example
├─Cat
│      10.jpg
│      11.jpg
│      12.jpg
│
└─Dog
       1.jpg
       2.jpg
       3.jpg
```

Format 2: ModelArts image classification 1.0

- The image and labeled file must be stored in the same directory, with the content in the labeled file used as label names.

  In the following example, **import-dir-1** and **import-dir-2** are the imported subdirectories:


```
dataset-import-example
├─import-dir-1
│      10.jpg
│      10.txt
│      11.jpg
│      11.txt
│      12.jpg
│      12.txt
└─import-dir-2
       1.jpg
       1.txt
       2.jpg
       2.txt
```

The following shows a label file for a single label, for example, the **1.txt** file:

```
Cat
```

The following shows a label file for multiple labels, for example, the **2.txt** file:

```
Cat
Dog
```

- Only images in JPG, JPEG, PNG, and BMP formats are supported. The size of a single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

Object Detection

Data for object detection can be stored in two formats:

Format 1: ModelArts PASCAL VOC 1.0

- The simple mode of object detection requires you to store labeled objects and your label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object file is **IMG_20180919_114745.jpg**, the name of the label file must be **IMG_20180919_114745.xml**.

  The label files must be in PASCAL VOC format. For details about the format, see **Table 7-14**.

  Example:

  ```
  ├─dataset-import-example
  │      IMG_20180919_114732.jpg
  │      IMG_20180919_114732.xml
  │      IMG_20180919_114745.jpg
  │      IMG_20180919_114745.xml
  │      IMG_20180919_114945.jpg
  │      IMG_20180919_114945.xml
  ```

  A label file example is as follows:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <annotation>
     <folder>NA</folder>
     <filename>bike_1_1593531469339.png</filename>
     <source>
        <database>Unknown</database>
     </source>
     <size>
        <width>554</width>
        <height>606</height>
        <depth>3</depth>
     </size>
     <segmented>0</segmented>
     <object>
        <name>Dog</name>
        <pose>Unspecified</pose>
  ```



```
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>279</xmin>
      <ymin>52</ymin>
      <xmax>474</xmax>
      <ymax>278</ymax>
    </bndbox>
  </object>
  <object>
    <name>Cat</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>279</xmin>
      <ymin>198</ymin>
      <xmax>456</xmax>
      <ymax>421</ymax>
    </bndbox>
  </object>
</annotation>
```

- Only images in JPG, JPEG, PNG, and BMP formats are supported. A single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

Format 2: YOLO

- A YOLO dataset must comply with the following structure:

```
└─ yolo_dataset/
   │
   ├── obj.names  # Label set file
   ├── obj.data   # Files and relative paths for recording dataset information
   ├── train.txt  # Relative path of images in the training set
   ├── valid.txt  # Relative path of images in the validation set
   │
   ├── obj_train_data/  # Directory where the images in the training set and the corresponding label files are stored
   │   ├── image1.txt  # BBox label list for image 1
   │   ├── image1.jpg
   │   ├── image2.txt
   │   ├── image2.jpg
   │   ├── ...
   │
   ├── obj_valid_data/  # Directory where the images in the validation set and the corresponding label files are stored
   │   ├── image101.txt
   │   ├── image101.jpg
   │   ├── image102.txt
   │   ├── image102.jpg
   │   ├── ...
```

  A YOLO dataset supports only training sets and validation sets. If other sets are imported, they will be invalid in the YOLO dataset.

- **obj.data** contains the following content and at least one of the **train** and **valid** subsets must be contained. The file paths are relative paths.

```
classes = 5 # Optional
names = <path/to/obj.names># For example, obj.names
train = <path/to/train.txt># For example, train.txt
valid = <path/to/valid.txt># Optional, for example, valid.txt
backup = backup/ # Optional
```

- The **obj.names** file records the label list. Each row label is used as the file index.

```
label1 # index of label 1: 0
label2 # index of label 2: 1
```



```
label3
...
```

- The file paths in **train.txt** and **valid.txt** are relative paths, and the file list must be in one-to-one relationship with the files in the directories. The file structures of the two files are as follows:

```
<path/to/image1.jpg># For example, obj_train_data/image.jpg
<path/to/image2.jpg># For example, obj_train_data/image.jpg
...
```

- The .txt files in the **obj_train_data/** and **obj_valid_data/** directories contain the BBox label information of the corresponding images. Each line indicates a BBox label.

```
# image1.txt:
# <label_index> <x_center> <y_center> <width> <height>
0 0.250000 0.400000 0.300000 0.400000
3 0.600000 0.400000 0.400000 0.266667
```

  x_center, **y_center**, **width**, and **height** indicate the normalized parameters for the target bounding box: the x-coordinate and y-coordinate of the center point, width, and height.

- Only images in JPG, JPEG, PNG, and BMP formats are supported. A single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

Image Segmentation

ModelArts image segmentation 1.0:

- Labeled objects and their label files (in one-to-one relationship with the labeled objects) must be in the same directory. For example, if the name of the labeled object file is **IMG_20180919_114746.jpg**, the name of the label file must be **IMG_20180919_114746.xml**.

  Fields **mask_source** and **mask_color** are added to the label file in PASCAL VOC format. For details about the format, see **Table 7-10**.

  Example:

```
├─dataset-import-example
│      IMG_20180919_114732.jpg
│      IMG_20180919_114732.xml
│      IMG_20180919_114745.jpg
│      IMG_20180919_114745.xml
│      IMG_20180919_114945.jpg
│      IMG_20180919_114945.xml
```

  A label file example is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
   <folder>NA</folder>
   <filename>image_0006.jpg</filename>
   <source>
      <database>Unknown</database>
   </source>
   <size>
      <width>230</width>
      <height>300</height>
      <depth>3</depth>
   </size>
   <segmented>1</segmented>
   <mask_source>obs://xianao/out/dataset-8153-Jmf5ylLjRmSacj9KevS/annotation/V001/
segmentationClassRaw/image_0006.png</mask_source>
   <object>
      <name>bike</name>
      <pose>Unspecified</pose>
```



```
    <truncated>0</truncated>
    <difficult>0</difficult>
    <mask_color>193,243,53</mask_color>
    <occluded>0</occluded>
    <polygon>
      <x1>71</x1>
      <y1>48</y1>
      <x2>75</x2>
      <y2>73</y2>
      <x3>49</x3>
      <y3>69</y3>
      <x4>68</x4>
      <y4>92</y4>
      <x5>90</x5>
      <y5>101</y5>
      <x6>45</x6>
      <y6>110</y6>
      <x7>71</x7>
      <y7>48</y7>
    </polygon>
  </object>
</annotation>
```

Text Classification

txt and csv files can be imported for text classification, with the text encoding format of UTF-8 or GBK.

Labeled objects and labels for text classification can be stored in two formats:

- ModelArts text classification combine 1.0: The labeled objects and labels for text classification are in the same text file. You can specify a separator to separate the labeled objects and labels, as well as multiple labels.

  For example, the following shows an example text file. The **Tab** key is used to separate the labeled objects from the labels.

  ```
  It touches good and responds quickly. I don't know how it performs in the future.    positive
  Three months ago, I bought a very good phone and replaced my old one with it. It can operate longer between charges.  positive
  Why does my phone heat up if I charge it for a while? The volume button stuck after being pressed down.  negative
  It's a gift for Father's Day. The delivery is fast and I received it in 24 hours. I like the earphones because the bass sounds feel good and they would not fall off.  positive
  ```

- ModelArts text classification 1.0: The labeled objects and labels for text classification are text files, and correspond to each other based on the rows. For example, the first row in a label file indicates the label of the first row in the file of the labeled object.

  For example, the content of the labeled object **COMMENTS_20180919_114745.txt** is as follows:

  ```
  It touches good and responds quickly. I don't know how it performs in the future.
  Three months ago, I bought a very good phone and replaced my old one with it. It can operate longer between charges.
  Why does my phone heat up if I charge it for a while? The volume button stuck after being pressed down.
  It's a gift for Father's Day. The delivery is fast and I received it in 24 hours. I like the earphones because the bass sounds feel good and they would not fall off.
  ```

  The content of the label file **COMMENTS_20180919_114745_result.txt** is as follows:

  ```
  positive
  negative
  negative
  positive
  ```


This data format requires you to store labeled objects and your label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object file is **COMMENTS_20180919_114745.txt**, the name of the label file must be **COMMENTS _20180919_114745_result.txt**.

Example of data files:

```
├─dataset-import-example
│      COMMENTS_20180919_114732.txt
│      COMMENTS _20180919_114732_result.txt
│      COMMENTS _20180919_114745.txt
│      COMMENTS _20180919_114745_result.txt
│      COMMENTS _20180919_114945.txt
│      COMMENTS _20180919_114945_result.txt
```

Sound Classification

ModelArts audio classification dir 1.0: Sound files with the same label must be stored in the same directory, and the label name is the directory name.

Example:

```
dataset-import-example
├─Cat
│      10.wav
│      11.wav
│      12.wav
│
└─Dog
       1.wav
       2.wav
       3.wav
```

Tables

CSV files can be imported from OBS. Select the directory where the files are stored. The number of columns in the CSV file must be the same as that in the dataset schema. The schema of the CSV file can be automatically obtained.

```
├─dataset-import-example
│      table_import_1.csv
│      table_import_2.csv
│      table_import_3.csv
│      table_import_4.csv
```

7.4.2.4 Importing a Manifest File

Prerequisites

- You have created a dataset.
- You have stored the data to be imported in OBS. You have stored the manifest file in OBS.
- The OBS bucket and ModelArts are in the same region and you can operate the bucket.

Importing File Data from a Manifest File

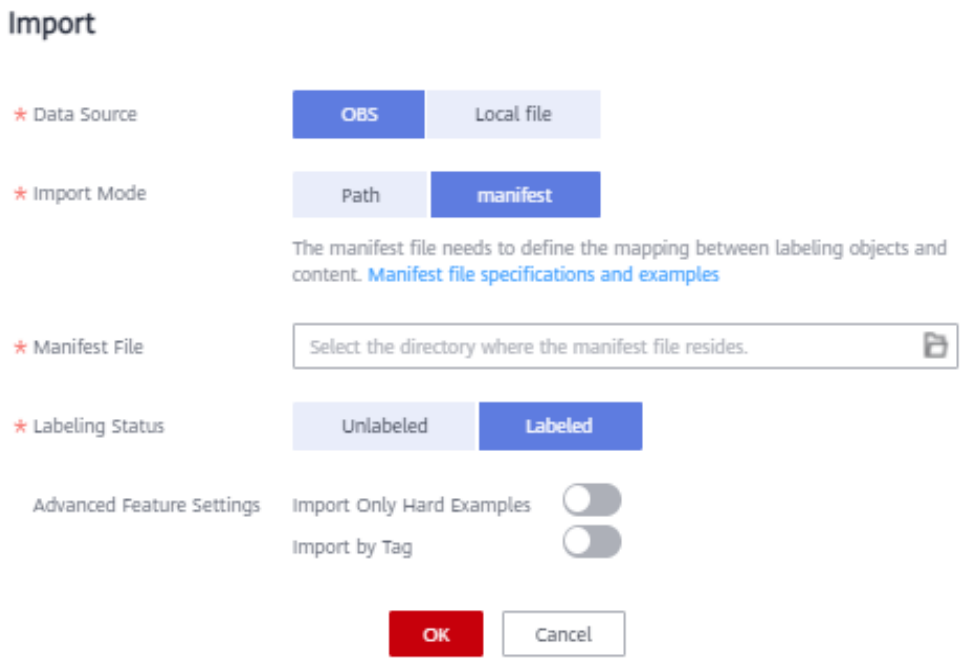
The parameters for data import vary according to the dataset type. The following uses an image dataset as an example.

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. Locate the row that contains the desired dataset and click **Import** in the **Operation** column. Alternatively, you can click the dataset name to go to the **Dashboard** tab page of the dataset, and click **Import** in the upper right corner.
3. In the **Import** dialog box, set the parameters as follows and click **OK**.
   - **Data Source**: **OBS**
   - **Import Mode**: **manifest**
   - **Manifest File**: OBS path for storing the manifest file
   - **Labeling Status**: **Labeled**
   - **Advanced Feature Settings**: disabled by default

     Import by Tag The system automatically obtains the labels of the dataset. You can click **Add Label** to add a label. This parameter is optional. If **Import by Tag** is disabled, you can add or delete labels for imported data when labeling data.

     Import Only Hard Examples: If this parameter is selected, only the **hard** attribute data of the manifest file is imported.

Figure 7-27 Importing a manifest file



After the data is imported, it will be automatically synchronized to the dataset. On the **Datasets** page, click the dataset name to view its details and create a labeling job to label the data.

Labeling Status of File Data

The labeling status can be **Unlabeled** or **Labeled**.

- **Unlabeled**: Only the labeling object (such as unlabeled images or texts) is imported.
- **Labeled**: Both the labeling object and content are imported. Labeling content importing is not supported for datasets in free format.

  To ensure that the labeling content can be correctly read, you must store data in strict accordance with the specifications.

  If **Import Mode** is set to **Path**, store the data to be imported according to the labeling file specifications.

  If **Import Mode** is set to **manifest**, the manifest file specifications must be met. For details, see **Specifications for Importing a Manifest File**.

  NOTE

  If the labeling status is set to **Labeled**, ensure that the folder or manifest file complies with the format specifications. Otherwise, the import may fail.

7.4.2.5 Specifications for Importing a Manifest File

The manifest file defines the mapping between labeled objects and content. The manifest file import mode means that the manifest file is used for dataset import. The manifest file can be imported from OBS. When importing a manifest file from OBS, ensure that you have the permissions to access the directory where the manifest file is stored.

NOTE

There are many requirements on the manifest file compilation. Import new data from OBS. Generally, manifest file import is used for data migration of ModelArts in different regions or using different accounts. If you have labeled data in a region using ModelArts, you can obtain the manifest file of the published dataset from the output path. Then you can import the dataset using the manifest file to ModelArts of other regions or accounts. The imported data carries the labeling information and does not need to be labeled again, improving development efficiency.

The manifest file that contains information about the original file and labeling can be used in labeling, training, and inference scenarios. The manifest file that contains only information about the original file can be used in inference scenarios or used to generate an unlabeled dataset. The manifest file must meet the following requirements:

- The manifest file uses the UTF-8 encoding format.
- The manifest file uses the JSON Lines format (**jsonlines.org**). A line contains one JSON object.

  ```
  {"source": "/path/to/image1.jpg", "annotation": ... }
  {"source": "/path/to/image2.jpg", "annotation": ... }
  {"source": "/path/to/image3.jpg", "annotation": ... }
  ```

  In the preceding example, the manifest file contains multiple lines of JSON object.
- The manifest file can be generated by you, third-party tools, or ModelArts Data Labeling. The file name can be any valid file name. To facilitate the internal use of the ModelArts system, the file name generated by the ModelArts data labeling function consists of the following strings: **DatasetName-VersionName.manifest**. For example, **animal-v20190123113030412 3.manifest**.

Image Classification

```
{
   "source":"s3://path/to/image1.jpg",
   "usage":"TRAIN",
   "hard":"true",
   "hard-coefficient":0.8,
   "id":"0162005993f8065ef47eefb59d1e4970",
   "annotation": [
      {
         "type": "modelarts/image_classification",
         "name": "cat",
         "property": {
            "color":"white",
            "kind":"Persian cat"
         },
         "hard":"true",
         "hard-coefficient":0.8,
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      },
      {
         "type": "modelarts/image_classification",
         "name":"animal",
         "annotated-by":"modelarts/active-learning",
         "confidence": 0.8,
         "creation-time":"2019-01-23 11:30:30"
      }],
   "inference-loc":"/path/to/inference-output"
}
```

Table 7-7 Parameters

Parameter	Mandatory	Description
source	Yes	URI of an object to be labeled. For details about data source types and examples, see **Table 7-8**.
usage	No	By default, the parameter value is left blank. Possible values are as follows: • **TRAIN**: The object is used for training. • **EVAL**: The object is used for evaluation. • **TEST**: The object is used for testing. • **INFERENCE**: The object is used for inference. If the parameter value is left blank, you decide how to use the object.
id	No	Sample ID exported from the system. You do not need to set this parameter when importing the sample.
annotation	No	If the parameter value is left blank, the object is not labeled. The value of **annotation** consists of an object list. For details about the parameters, see **Table 7-9**.
inference-loc	No	This parameter is available when the file is generated by the inference service, indicating the location of the inference result file.

Table 7-8 Data source types

Type	Example
OBS	"source":"s3://path-to-jpg"
Content	"source":"content://I love machine learning"

Table 7-9 annotation objects

Parameter	Mandatory	Description
type	Yes	Label type. Possible values are as follows: • **image_classification**: image classification • **text_classification**: text classification • **text_entity**: named entity recognition • **object_detection**: object detection • **audio_classification**: sound classification • **audio_content**: speech labeling • **audio_segmentation**: speech paragraph labeling
name	Yes/No	This parameter is mandatory for the classification type but optional for other types. This example uses the image classification type.
id	Yes/No	Label ID. This parameter is mandatory for triplets but optional for other types. The entity label ID of a triplet is in **E+number** format, for example, **E1** and **E2**. The relationship label ID of a triplet is in **R+number** format, for example, **R1** and **R2**.
property	No	Labeling property. In this example, the cat has two properties: color and kind.
hard	No	Indicates whether the example is a hard example. **True** indicates that the labeling example is a hard example, and **False** indicates that the labeling example is not a hard example.
annotated-by	No	The default value is **human**, indicating manual labeling. • human
creation-time	No	Time when the labeling job was created. It is the time when labeling information was written, not the time when the manifest file was generated.

Parameter	Mandatory	Description
confidence	No	Confidence score of machine labeling. The value ranges from 0 to 1.

Image Segmentation

```
{
   "annotation": [{
      "annotation-format": "PASCAL VOC",
      "type": "modelarts/image_segmentation",
      "annotation-loc": "s3://path/to/annotation/image1.xml",
      "creation-time": "2020-12-16 21:36:27",
      "annotated-by": "human"
   }],
   "usage": "train",
   "source": "s3://path/to/image1.jpg",
   "id": "16d196c19bf61994d7deccafa435398c",
   "sample-type": 0
}
```

- The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.
- **annotation-loc** indicates the path for saving the label file. This parameter is mandatory for image segmentation and object detection but optional for other labeling types.
- **annotation-format** indicates the format of the label file. This parameter is optional. The default value is **PASCAL VOC**. Only **PASCAL VOC** is supported.
- **sample-type** indicates a sample format. Value **0** indicates image, **1** text, **2** audio, **4** table, and **6** video.

Table 7-10 PASCAL VOC format parameters

Parameter	Mandatory	Description
folder	Yes	Directory where the data source is located
filename	Yes	Name of the file to be labeled
size	Yes	Image pixel • **width**: image width. This parameter is mandatory. • **height**: image height. This parameter is mandatory. • **depth**: number of image channels. This parameter is mandatory.
segmented	Yes	Segmented or not
mask_source	No	Segmentation mask path

Parameter	Mandatory	Description
object	Yes	Object detection information. Multiple **object{}** functions are generated for multiple objects. • **name**: type of the labeled content. This parameter is mandatory. • **pose**: shooting angle of the labeled content. This parameter is mandatory. • **truncated**: whether the labeled content is truncated (**0** indicates that the content is not truncated). This parameter is mandatory. • **occluded**: whether the labeled content is occluded (**0** indicates that the content is not occluded). This parameter is mandatory. • **difficult**: whether the labeled object is difficult to identify (**0** indicates that the object is easy to identify). This parameter is mandatory. • **confidence**: confidence score of the labeled object. The value ranges from 0 to 1. This parameter is optional. • **bndbox**: bounding box type. This parameter is mandatory. For details about the possible values, see **Table 7-11**. • **mask_color**: label color, which is represented by the RGB value. This parameter is mandatory.

Table 7-11 Bounding box types

Parameter	Shape	Labeling information
polygon	Polygon	Coordinates of points <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6> <x7>100<x7> <y7>100<y7>

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
   <folder>NA</folder>
   <filename>image_0006.jpg</filename>
   <source>
      <database>Unknown</database>
   </source>
   <size>
      <width>230</width>
      <height>300</height>
      <depth>3</depth>
   </size>
   <segmented>1</segmented>
   <mask_source>obs://xianao/out/dataset-8153-Jmf5ylLjRmSacj9KevS/annotation/V001/
segmentationClassRaw/image_0006.png</mask_source>
   <object>
      <name>bike</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <mask_color>193,243,53</mask_color>
      <occluded>0</occluded>
      <polygon>
         <x1>71</x1>
         <y1>48</y1>
         <x2>75</x2>
         <y2>73</y2>
         <x3>49</x3>
         <y3>69</y3>
         <x4>68</x4>
         <y4>92</y4>
         <x5>90</x5>
         <y5>101</y5>
         <x6>45</x6>
         <y6>110</y6>
```



```
        <x7>71</x7>
        <y7>48</y7>
      </polygon>
   </object>
</annotation>
```

Text Classification

```
{
   "source": "content://I like this product ",
   "id":"XGDVGS",
   "annotation": [
      {
         "type": "modelarts/text_classification",
         "name": " positive",
         "annotated-by": "human",
         "creation-time": "2019-01-23 11:30:30"
      } ]
}
```

The **content** parameter indicates the text to be labeled (in UTF-8 encoding format, which can be Chinese). The other parameters are the same as those described in **Image Classification**. For details, see **Table 7-7**.

Named Entity Recognition

```
{
   "source":"content://Michael Jordan is the most famous basketball player in the world.",
   "usage":"TRAIN",
   "annotation":[
      {
         "type":"modelarts/text_entity",
         "name":"Person",
         "property":{
            "@modelarts:start_index":0,
            "@modelarts:end_index":14
         },
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      },
      {
         "type":"modelarts/text_entity",
         "name":"Category",
         "property":{
            "@modelarts:start_index":34,
            "@modelarts:end_index":44
         },
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }
   ]
}
```

The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.

Table 7-12 describes the property parameters. For example, if you want to extract **Michael** from **"source":"content://Michael Jordan"**, the value of **start_index** is **0** and that of **end_index** is **7**.

Table 7-12 property parameters

Parameter	Data type	Description
@modelarts:start_index	Integer	Start position of the text. The value starts from 0, including the characters specified by **start_index**.
@modelarts:end_index	Integer	End position of the text, excluding the characters specified by **end_index**.

Text Triplet

```
{
  "source":"content://"Three Body" is a series of long science fiction novels created by Liu Cix.",
  "usage":"TRAIN",
  "annotation":[
    {
      "type":"modelarts/text_entity",
      "name":"Person",
      "id":"E1",
      "property":{
        "@modelarts:start_index":67,
        "@modelarts:end_index":74
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_entity",
      "name":"Book",
      "id":"E2",
      "property":{
        "@modelarts:start_index":0,
        "@modelarts:end_index":12
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_triplet",
      "name":"Author",
      "id":"R1",
      "property":{
        "@modelarts:from":"E1",
        "@modelarts:to":"E2"
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_triplet",
      "name":"Works",
      "id":"R2",
      "property":{
        "@modelarts:from":"E2",
        "@modelarts:to":"E1"
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    }
  ]
}
```


The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.

Table 5 property parameters describes the **property** parameters. **@modelarts:start_index** and **@modelarts:end_index** are the same as those of named entity recognition. For example, when **source** is set to **content://"Three Body" is a series of long science fiction novels created by Liu Cix.**, **Liu Cix** is an entity person, **Three Body** is an entity book, the person is the author of the book, and the book is works of the person.

Table 7-13 property parameters

Parameter	Data type	Description
@modelarts:start_index	Integer	Start position of the triplet entities. The value starts from 0, including the characters specified by **start_index**.
@modelarts:end_index	Integer	End position of the triplet entities, excluding the characters specified by **end_index**.
@modelarts:from	String	Start entity ID of the triplet relationship
@modelarts:to	String	Entity ID pointed to in the triplet relationship

Object Detection

```
{
   "source":"s3://path/to/image1.jpg",
   "usage":"TRAIN",
   "hard":"true",
   "hard-coefficient":0.8,
   "annotation": [
      {
         "type":"modelarts/object_detection",
         "annotation-loc": "s3://path/to/annotation1.xml",
         "annotation-format":"PASCAL VOC",
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }]
}
```

- The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.
- **annotation-loc** indicates the path for saving the label file. This parameter is mandatory for object detection and image segmentation but optional for other labeling types.
- **annotation-format** indicates the format of the label file. This parameter is optional. The default value is **PASCAL VOC**. Only **PASCAL VOC** is supported.

Table 7-14 PASCAL VOC format parameters

Parameter	Mandatory	Description
folder	Yes	Directory where the data source is located
filename	Yes	Name of the file to be labeled
size	Yes	Image pixel • **width**: image width. This parameter is mandatory. • **height**: image height. This parameter is mandatory. • **depth**: number of image channels. This parameter is mandatory.
segmented	Yes	Segmented or not
object	Yes	Object detection information. Multiple **object{}** functions are generated for multiple objects. • **name**: type of the labeled content. This parameter is mandatory. • **pose**: shooting angle of the labeled content. This parameter is mandatory. • **truncated**: whether the labeled content is truncated (**0** indicates that the content is not truncated). This parameter is mandatory. • **occluded**: whether the labeled content is occluded (**0** indicates that the content is not occluded). This parameter is mandatory. • **difficult**: whether the labeled object is difficult to identify (**0** indicates that the object is easy to identify). This parameter is mandatory. • **confidence**: confidence score of the labeled object. The value ranges from 0 to 1. This parameter is optional. • **bndbox**: bounding box type. This parameter is mandatory. For details about the possible values, see **Table 7-15**.

Table 7-15 Bounding box types

Parameter	Shape	Labeling information
point	Point	Coordinates of a point <x>100<x> <y>100<y>

Parameter	Shape	Labeling information
line	Line	Coordinates of points <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>200<y2>
bndbox	Rectangle	Coordinates of the upper left and lower right points <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>
polygon	Polygon	Coordinates of points <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6>
circle	Circle	Center coordinates and radius <cx>100<cx> <cy>100<cy> <r>50<r>

Example:

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>point</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
```



```
    <occluded>0</occluded>
    <difficult>0</difficult>
    <point>
      <x1>456</x1>
      <y1>596</y1>
    </point>
  </object>
  <object>
    <name>line</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <line>
      <x1>133</x1>
      <y1>651</y1>
      <x2>229</x2>
      <y2>561</y2>
    </line>
  </object>
  <object>
    <name>bag</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <bndbox>
      <xmin>108</xmin>
      <ymin>101</ymin>
      <xmax>251</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
  <object>
    <name>boots</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <hard-coefficient>0.8</hard-coefficient>
    <polygon>
      <x1>373</x1>
      <y1>264</y1>
      <x2>500</x2>
      <y2>198</y2>
      <x3>437</x3>
      <y3>76</y3>
      <x4>310</x4>
      <y4>142</y4>
    </polygon>
  </object>
  <object>
    <name>circle</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <circle>
      <cx>405</cx>
      <cy>170</cy>
      <r>100<r>
    </circle>
  </object>
</annotation>
```

Sound Classification

```
{
"source":
```



```
"s3://path/to/pets.wav",
   "annotation": [
      {
         "type": "modelarts/audio_classification",
         "name":"cat",
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }
   ]
}
```

The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.

Speech Labeling

```
{
   "source":"s3://path/to/audio1.wav",
   "annotation":[
      {
         "type":"modelarts/audio_content",
         "property":{
            "@modelarts:content":"Today is a good day."
         },
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }
   ]
}
```

- The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.
- The **@modelarts:content** parameter in **property** indicates speech content. The data type is **String**.

Speech Paragraph Labeling

```
{
   "source":"s3://path/to/audio1.wav",
   "usage":"TRAIN",
   "annotation":[
      {

"type":"modelarts/audio_segmentation",
          "property":{
             "@modelarts:start_time":"00:01:10.123",
             "@modelarts:end_time":"00:01:15.456",

             "@modelarts:source":"Tom",

             "@modelarts:content":"How are you?"
          },
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      },
      {
         "type":"modelarts/audio_segmentation",
          "property":{
             "@modelarts:start_time":"00:01:22.754",
             "@modelarts:end_time":"00:01:24.145",
             "@modelarts:source":"Jerry",
             "@modelarts:content":"I'm fine, thank you."
          },
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }
   ]
}
```


- The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.
- **Table 7-16** describes the **property** parameters.

Table 7-16 property parameters

Parameter	Data type	Description
@modelarts:start_time	String	Start time of the sound. The format is **hh:mm:ss.SSS**. **hh** indicates the hour, **mm** indicates the minute, **ss** indicates the second, and **SSS** indicates the millisecond.
@modelarts:end_time	String	End time of the sound. The format is **hh:mm:ss.SSS**. **hh** indicates the hour, **mm** indicates the minute, **ss** indicates the second, and **SSS** indicates the millisecond.
@modelarts:source	String	Sound source
@modelarts:content	String	Sound content

Video Labeling

```
{
  "annotation": [{
    "annotation-format": "PASCAL VOC",
    "type": "modelarts/object_detection",
    "annotation-loc": "s3://path/to/annotation1_t1.473722.xml",
    "creation-time": "2020-10-09 14:08:24",
    "annotated-by": "human"
  }],
  "usage": "train",
  "property": {
    "@modelarts:parent_duration": 8,
    "@modelarts:parent_source": "s3://path/to/annotation1.mp4",
    "@modelarts:time_in_video": 1.473722
  },
  "source": "s3://input/path/to/annotation1_t1.473722.jpg",
  "id": "43d88677c1e9a971eeb692a80534b5d5",
  "sample-type": 0
}
```

- The parameters such as **source**, **usage**, and **annotation** are the same as those described in **Image Classification**. For details, see **Table 7-7**.
- **annotation-loc** indicates the path for saving the label file. This parameter is mandatory for object detection but optional for other labeling types.
- **annotation-format** indicates the format of the label file. This parameter is optional. The default value is **PASCAL VOC**. Only **PASCAL VOC** is supported.
- **sample-type** indicates a sample format. Value **0** indicates image, **1** text, **2** audio, **4** table, and **6** video.

Table 7-17 property parameters

Parameter	Data type	Description
@modelarts:parent_duration	Double	Duration of the labeled video, in seconds
@modelarts:time_in_video	Double	Timestamp of the labeled video frame, in seconds
@modelarts:parent_source	String	OBS path of the labeled video

Table 7-18 PASCAL VOC format parameters

Parameter	Mandatory	Description
folder	Yes	Directory where the data source is located
filename	Yes	Name of the file to be labeled
size	Yes	Image pixel • **width**: image width. This parameter is mandatory. • **height**: image height. This parameter is mandatory. • **depth**: number of image channels. This parameter is mandatory.
segmented	Yes	Segmented or not

Parameter	Mandatory	Description
object	Yes	Object detection information. Multiple **object{}** functions are generated for multiple objects. • **name**: type of the labeled content. This parameter is mandatory. • **pose**: shooting angle of the labeled content. This parameter is mandatory. • **truncated**: whether the labeled content is truncated (**0** indicates that the content is not truncated). This parameter is mandatory. • **occluded**: whether the labeled content is occluded (**0** indicates that the content is not occluded). This parameter is mandatory. • **difficult**: whether the labeled object is difficult to identify (**0** indicates that the object is easy to identify). This parameter is mandatory. • **confidence**: confidence score of the labeled object. The value ranges from 0 to 1. This parameter is optional. • **bndbox**: bounding box type. This parameter is mandatory. For details about the possible values, see **Table 7-19**.

Table 7-19 Bounding box types

Parameter	Shape	Labeling information
point	Point	Coordinates of a point <x>100<x> <y>100<y>
line	Line	Coordinates of points <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>200<y2>
bndbox	Rectangle	Coordinates of the upper left and lower right points <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>

Parameter	Shape	Labeling information
polygon	Polygon	Coordinates of points <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6>
circle	Circle	Center coordinates and radius <cx>100<cx> <cy>100<cy> <r>50<r>

Example:

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932_t1.473722.jpg.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>point</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <point>
        <x1>456</x1>
        <y1>596</y1>
     </point>
  </object>
  <object>
     <name>line</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <line>
        <x1>133</x1>
        <y1>651</y1>
        <x2>229</x2>
        <y2>561</y2>
     </line>
  </object>
```



```
  <object>
    <name>bag</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <bndbox>
      <xmin>108</xmin>
      <ymin>101</ymin>
      <xmax>251</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
  <object>
    <name>boots</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <hard-coefficient>0.8</hard-coefficient>
    <polygon>
      <x1>373</x1>
      <y1>264</y1>
      <x2>500</x2>
      <y2>198</y2>
      <x3>437</x3>
      <y3>76</y3>
      <x4>310</x4>
      <y4>142</y4>
    </polygon>
  </object>
  <object>
    <name>circle</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <circle>
      <cx>405</cx>
      <cy>170</cy>
      <r>100<r>
    </circle>
  </object>
</annotation>
```

7.4.3 Importing Data from DLI

Data importing from DLI is supported for table datasets.

To import data from DLI, select the DLI queue, database, and table name. The schema (column name and type) of the selected table must be the same as that of the dataset. The schema of the selected table can be automatically obtained.

- **Queue Name**: All DLI queues of the current account are automatically displayed. Select the required queue from the drop-down list.
- **Database Name**: All databases are displayed based on the selected queue. Select the required database from the drop-down list.
- **Table Name**: All tables in the selected database are displayed. Select the required table from the drop-down list.

NOTE

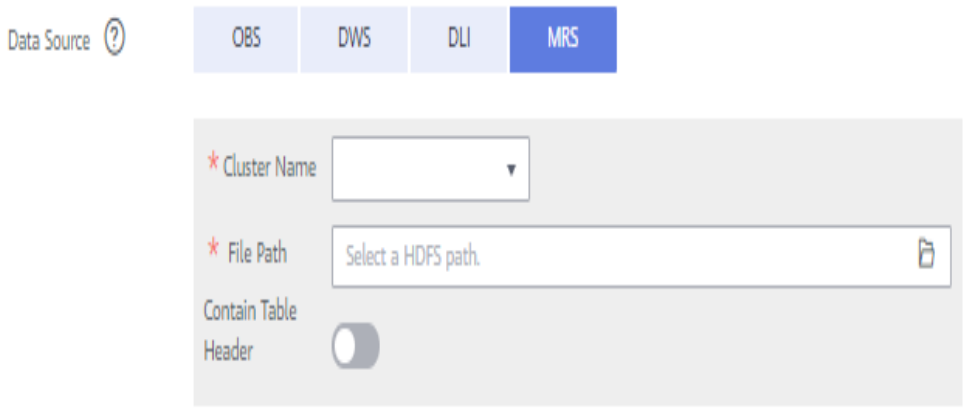
The default queue of DLI is used only for experience. Different accounts may preempt resources. Therefore, resources need to be queued. You may not be able to obtain required resources each time to perform related operations.

DLI supports schema mapping. That is, the schema field name of the imported table can be different from that of the dataset, but the type must be the same.

7.4.4 Importing Data from MRS

To import data in CSV format stored on HDFS from MRS, select an existing MRS cluster and select the file name or directory from the HDFS file list. The number of columns in the imported file must be the same as that of the dataset schema.

Figure 7-28 Importing data from MRS

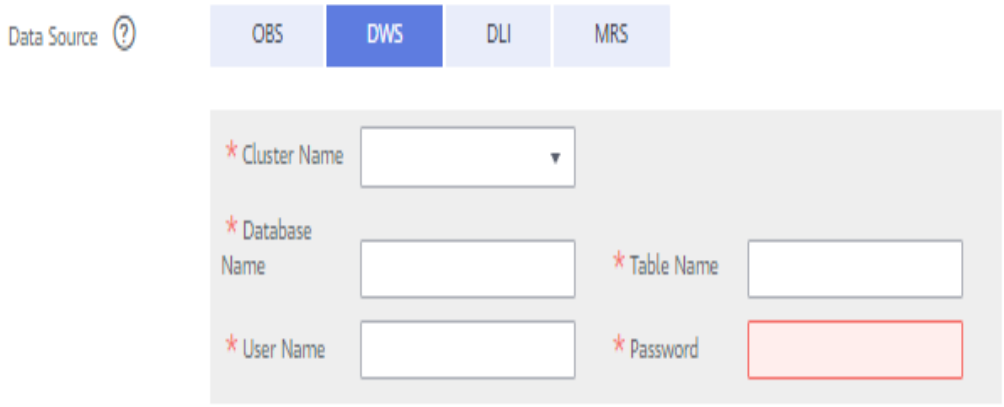


- **Cluster Name**: All MRS clusters of the current account are automatically displayed. However, streaming clusters do not support data import. Select the required cluster from the drop-down list.
- **File Path**: Enter the HDFS file path based on the selected cluster.
- **Contain Table Header**: If this setting is enabled, the imported file contains table headers.

7.4.5 Importing Data from DWS

To import data from DWS, select a DWS cluster and enter the database name, table name, username, and password. The schema (column name and type) of the imported table must be the same as that of the dataset.

Figure 7-29 Importing data from DWS



- **Cluster Name**: All DWS clusters of the current account are automatically displayed. Select the required DWS cluster from the drop-down list.
- **Database Name**: Enter the name of the database where the data is located based on the selected DWS cluster.
- **Table Name**: Enter the name of the table where the data is located based on the selected database.
- **User Name**: Enter the username of the DWS cluster administrator.
- **Password**: Enter the password of the DWS cluster administrator.

NOTE

To import data from DWS, use DLI functions. If you do not have the permission to access DLI, create a DLI agency as prompted.

7.4.6 Importing Data from Local Files

Prerequisites

- You have created a dataset.
- You have created an OBS bucket. The OBS bucket and ModelArts are in the same region and you can operate the bucket.

Import Operation

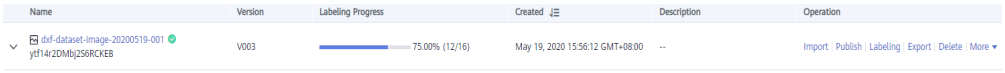
Both file and table data can be uploaded from local files. The data uploaded from local files should be stored in an OBS directory. You must have created an OBS bucket.

In a single batch upload, a maximum of 100 files can be uploaded at a time, and the total size of the files cannot exceed 5 GB.

The parameters on the GUI for data import vary according to the dataset type. The following uses a dataset of the image classification type as an example.

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. Locate the row that contains the desired dataset and click **Import** in the **Operation** column.

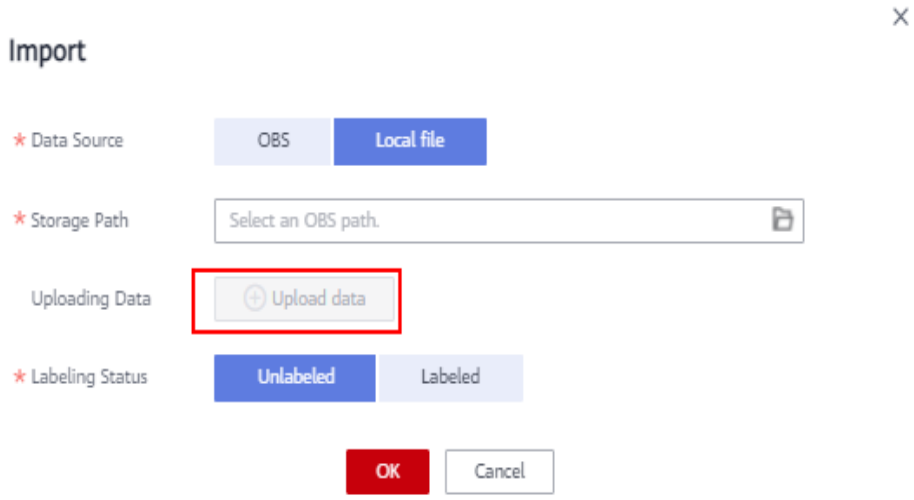
Figure 7-30 Importing data



Alternatively, you can click the dataset name to go to the **Dashboard** tab page of the dataset, and click **Import** in the upper right corner.

3. In the **Import** dialog box, set the parameters as follows and click **OK**.
   - **Data Source**: **Local file**
   - **Storage Path**: Select an OBS path.
   - **Uploading Data**: Click **Upload data**, upload local data, and click **OK**.

Figure 7-31 Importing data from local files



7.5 Data Analysis and Preview

Generally, the quality of raw data cannot meet training requirements, for example, invalid or duplicate data exists. To help you improve data quality, ModelArts provides the following capabilities:

- **Auto Grouping**: pre-classifies data through clustering to allow you to label data based on clustering results, which ensures that different labels have the same or the almost same number of samples.
- **Data Filtering**: enables you to filter data based on sample attributes and auto grouping results.
- **Data Feature Analysis**: analyzes data features or labeling results, such as the brightness and bounding box distribution, helping you analyze data balance and improve the model training effect.

7.5.1 Auto Grouping

To improve the precision of auto labeling algorithms, you can evenly label multiple classes. ModelArts provides built-in grouping algorithms. You can enable auto grouping to improve data labeling efficiency.

Auto grouping can be understood as data labeling preprocessing. Clustering algorithms are used to cluster unlabeled images, and images are labeled or cleaned by group based on the clustering result.

For example, a user searches for *XX* through a search engine, downloads and uploads related images to the dataset, and then uses the auto grouping function to classify *XX* images, such as papers, posters, images confirmed as *XX*, and others. The user can quickly remove unwanted images from a group or select all images of a type and add labels to the images.

NOTE

Only datasets of image classification, object detection, and image segmentation types support the auto grouping function.

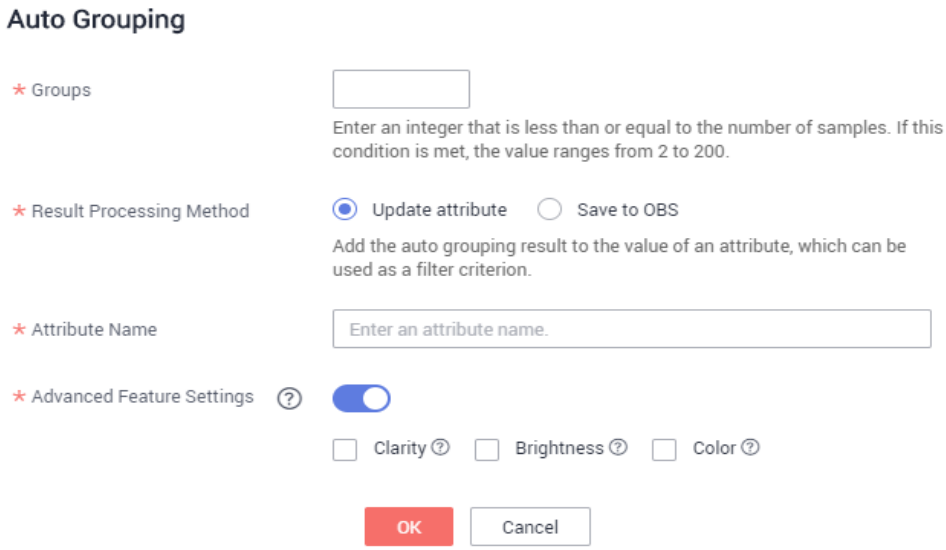
Starting Auto Grouping Tasks

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Label Data**.
2. In the labeling job list, select a labeling job of the object detection or image classification type and click the labeling job name to go to the labeling job details page.
3. On the **All statuses** tab page of the dataset details page, choose **Auto Grouping** > **Start Task**.

   NOTE

   You can start auto group tasks or view task history only on the **All** tab page.
4. In the displayed **Auto Grouping** dialog box, set parameters and click **OK**.
   - **Groups**: Enter an integer from 2 to 200. The parameter value indicates the number of groups into which images are divided.
   - **Result Processing Method**: Select **Update attribute** or **Save to OBS**.
   - **Attribute Name**: If you select **Update attribute**, you need to enter an attribute name.
   - **Result Storage Path**: If you select **Save to OBS**, specify an OBS path.
   - **Advanced Feature Settings**: After this function is enabled, you can select **Clarity**, **Brightness**, and **Color** dimensions for the auto grouping function so that the grouping is based on the image brightness, color, and clarity. You can select multiple options.

Figure 7-32 Auto grouping

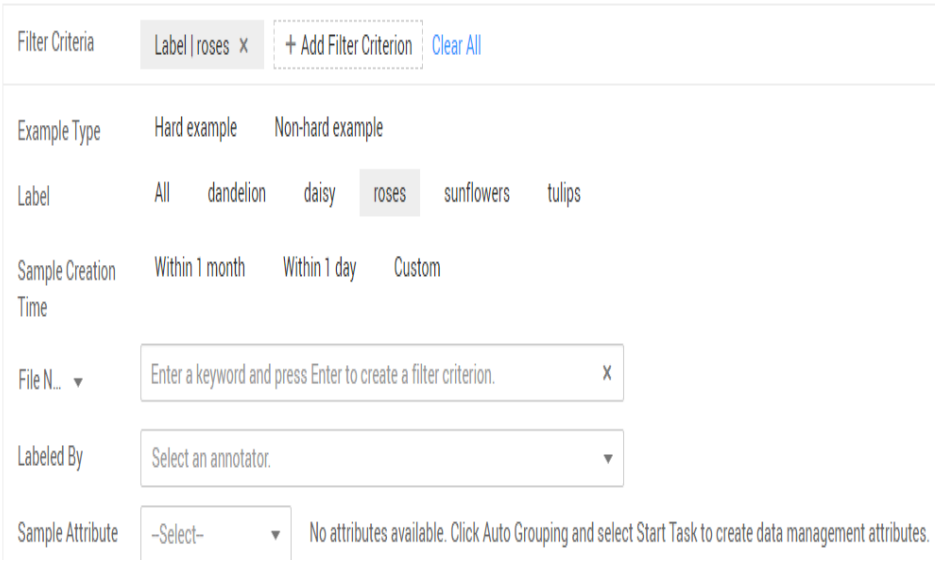


5. After the task is submitted, the task progress is displayed in the upper right corner of the page. After the task is complete, you can view the history of the auto grouping tasks to learn task status.

Viewing the Auto Grouping Result

On the **All** tab page of the dataset details page, expand **Filter Criteria**, set **Sample Attribute** to the attribute name of the auto grouping task, and set the sample attribute value to filter the grouping result.

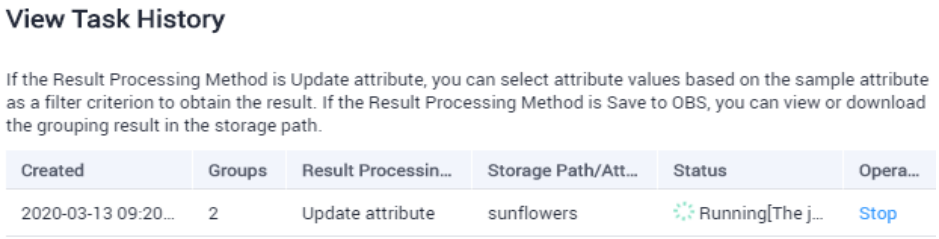
Figure 7-33 Viewing the auto grouping result



Viewing Auto Grouping Task History

On the **All** tab page of the dataset details page, choose **Auto Grouping** > **View Task History**. In the **View Task History** dialog box, basic information about the auto grouping tasks of the current dataset is displayed.

Figure 7-34 Auto grouping task history



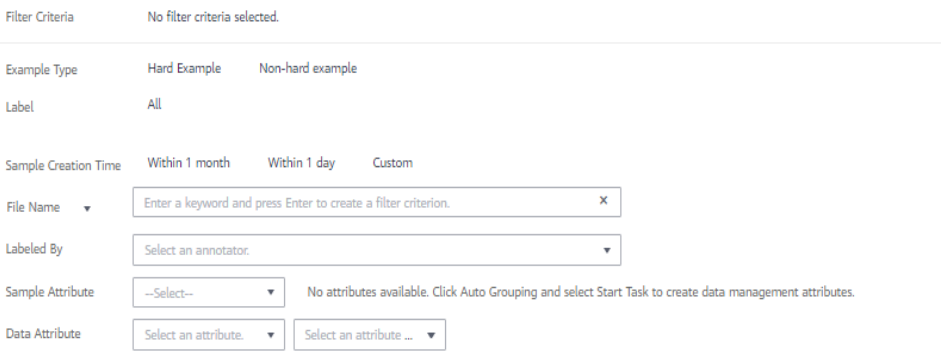
7.5.2 Data Filtering

On the **Dashboard** tab page of the dataset, the summary of the dataset is displayed by default. In the upper right corner of the page, click **Label**. The dataset details page is displayed, showing all data in the dataset by default. On the **All**, **Unlabeled**, or **Labeled** tab page, you can add filter criteria in the filter criteria area to quickly filter the data you want to view.

The following filter criteria are supported. You can set one or more filter criteria.

- **Example Type**: Select **Hard example** or **Non-hard example**.
- **Label**: Select **All** or one or more labels you specified.
- **Sample Creation Time**: Select **Within 1 month**, **Within 1 day**, or **Custom** to customize a time range.
- **File Name** or **Path**: Filter files by file name or file storage path.
- **Labeled By**: Select the name of the user who labeled the image.
- **Sample Attribute**: Select the attribute generated by auto grouping. This filter criterion can be used only after **auto grouping** is enabled.
- **Data Attribute**: This criterion is not supported.

Figure 7-35 Filter criteria



7.5.3 Data Feature Analysis

Images or target bounding boxes are analyzed based on image features, such as blurs and brightness to draw visualized curves to help process datasets.

You can also select multiple versions of a dataset to view their curves for comparison and analysis.

Background

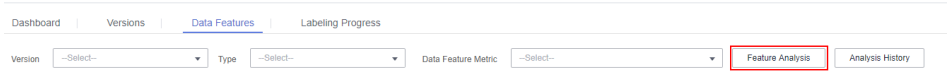
- Data feature analysis is only available for image datasets of the image classification and object detection types.
- Data feature analysis is only available for the published datasets. The published dataset versions in **Default** format support data feature analysis.
- A data scope for feature analysis varies depending on the dataset type.
  - In a dataset of the object detection type, if the number of labeled samples is 0, the **View Data Feature** tab page is unavailable and data features are not displayed after a version is published. After the images are labeled and the version is published, the data features of the labeled images are displayed.
  - In a dataset of the image classification type, if the number of labeled samples is 0, the **View Data Feature** tab page is unavailable and data features are not displayed after a version is published. After the images are labeled and the version is published, the data features of all images are displayed.
- The analysis result is valid only when the number of images in a dataset reaches a certain level. Generally, more than 1,000 images are required.
- Image classification supports the following data feature metrics: **Resolution**, **Aspect Ratio**, **Brightness**, **Saturation**, **Blur Score**, and **Colorfulness** Object detection supports all data feature metrics. **Supported Data Feature Metrics** provides all data feature metrics supported by ModelArts.

Data Feature Analysis

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. Select a dataset and click **Data Features** in the **Operation** column. The **Data Features** tab page of the dataset page is displayed.

   You can also click a dataset name to go to the dataset page and click the **Data Features** tab.
3. By default, feature analysis is not started for published datasets. You need to manually start feature analysis tasks for each dataset version. On the **Data Features** tab page, click **Feature Analysis**.

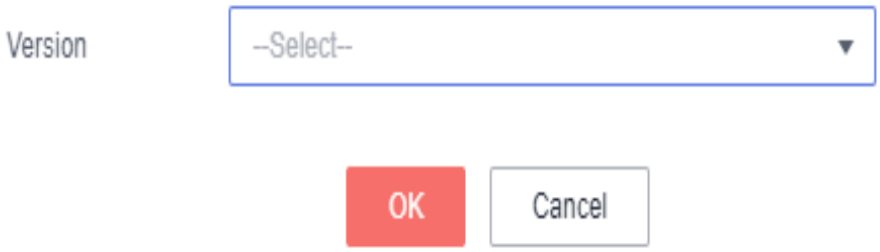
   Figure 7-36 Feature Analysis

4. In the dialog box that is displayed, configure the dataset version for feature analysis and click **OK** to start analysis.

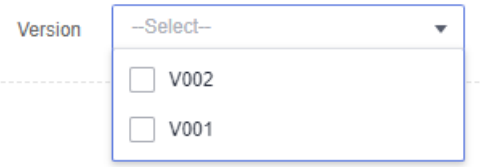
   Version: Select a published version of the dataset.

Figure 7-37 Starting a data feature analysis task



5. After a data feature analysis task is started, it takes a certain period of time to complete, depending on the data volume. If the selected version is displayed in the **Version** drop-down list and can be selected, the analysis is complete.

   Figure 7-38 Selecting a version for which feature analysis has been performed

6. View the data feature analysis result.

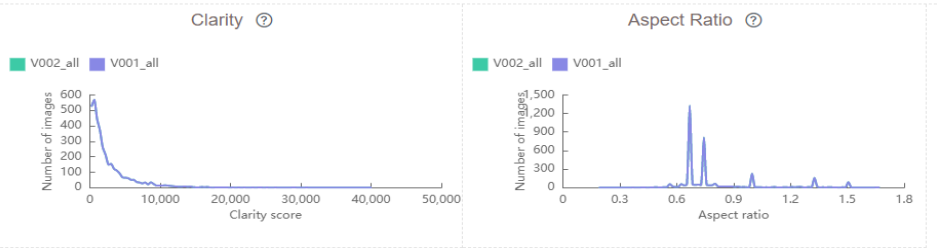
   Version: Select the version to be compared from the drop-down list You can also select only one version.

   Type: Select the type to be analyzed. The value can be **all**, **train**, **eval**, or **inference**.

   Data Feature Metric: Select metrics to be displayed from the drop-down list. For details, see **Supported Data Feature Metrics**.

   Then, the selected version and metrics are displayed on the page, as shown in **Figure 7-39**. The displayed chart helps you understand data distribution for better data processing.

   Figure 7-39 Data feature analysis

7. View historical records of the analysis task.

   After data feature analysis is complete, you can click **Task History** on the right of the **Data Features** tab page to view historical analysis tasks and their statuses in the dialog box that is displayed.

Figure 7-40 Viewing the task history

Supported Data Feature Metrics

Table 7-20 Data feature metrics

Metric	Description	Explanation
Resolution	Image resolution. An area value is used as a statistical value.	Metric analysis results are used to check whether there is an offset point. If an offset point exists, you can resize or delete the offset point.
Aspect Ratio	An aspect ratio is a proportional relationship between an image's width and height.	The chart of the metric is in normal distribution, which is generally used to compare the difference between the training set and the dataset used in the real scenario.

Metric	Description	Explanation
Brightness	Brightness is the perception elicited by the luminance of a visual target. A larger value indicates better image brightness.	The chart of the metric is in normal distribution. You can determine whether the brightness of the entire dataset is high or low based on the distribution center. You can adjust the brightness based on your application scenario. For example, if the application scenario is night, the brightness should be lower.
Saturation	Color saturation of an image. A larger value indicates that the entire image color is easier to distinguish.	The chart of the metric is in normal distribution, which is generally used to compare the difference between the training set and the dataset used in the real scenario.
Blur Score Clarity	Image clarity, which is calculated using the Laplace operator. A larger value indicates clearer edges and higher clarity.	You can determine whether the clarity meets the requirements based on the application scenario. For example, if data is collected from HD cameras, the clarity must be higher. You can sharpen or blur the dataset and add noises to adjust the clarity.
Colorfulness	Horizontal coordinate: Colorfulness of an image. A larger value indicates richer colors. Vertical coordinate: Number of images	Colorfulness on the visual sense, which is generally used to compare the difference between the training set and the dataset used in the real scenario.
Bounding Box Number	Horizontal coordinate: Number of bounding boxes in an image Vertical coordinate: Number of images	It is difficult for a model to detect a large number of bounding boxes in an image. Therefore, more images containing many bounding boxes are required for training.

Metric	Description	Explanation
Std of Bounding Boxes Area Per Image Standard Deviation of Bounding Boxes Per Image	Horizontal coordinate: Standard deviation of bounding boxes in an image. If an image has only one bounding box, the standard deviation is 0. A larger standard deviation indicates higher bounding box size variation in an image. Vertical coordinate: Number of images	It is difficult for a model to detect a large number of bounding boxes with different sizes in an image. You can add data for training based on scenarios or delete data if such scenarios do not exist.
Aspect Ratio of Bounding Boxes	Horizontal coordinate: Aspect ratio of the target bounding boxes Vertical coordinate: Number of bounding boxes in all images	The chart of the metric is generally in Poisson distribution, which is closely related to application scenarios. This metric is mainly used to compare the differences between the training set and the validation set. For example, if the training set is a rectangle, the result will be significantly affected if the validation set is close to a square.
Area Ratio of Bounding Boxes	Horizontal coordinate: Area ratio of the target bounding boxes, that is, the ratio of the bounding box area to the entire image area. A larger value indicates a higher ratio of the object in the image. Vertical coordinate: Number of bounding boxes in all images	The metric is used to determine the distribution of anchors used in the model. If the target bounding box is large, set the anchor to a large value.

Metric	Description	Explanation
Marginalization Value of Bounding Boxes	Horizontal coordinate: Marginalization degree, that is, the ratio of the distance between the center point of the target bounding box and the center point of the image to the total distance of the image. A larger value indicates that the object is closer to the edge. (The total distance of an image is the distance from the intersection point of a ray (that starts from the center point of the image and passes through the center point of the bounding box) and the image border to the center point of the image.) Vertical coordinate: Number of bounding boxes in all images	Generally, the chart of the metric is in normal distribution. The metric is used to determine whether an object is at the edge of an image. If a part of an object is at the edge of an image, you can add a dataset or do not label the object.
Overlap Score of Bounding Boxes Overlap Score of Bounding Boxes	Horizontal coordinate: Overlap degree, that is, the part of a single bounding box overlapped by other bounding boxes. The value ranges from 0 to 1. A larger value indicates that more parts are overlapped by other bounding boxes. Vertical coordinate: Number of bounding boxes in all images	The metric is used to determine the overlapping degree of objects to be detected. Overlapped objects are difficult to detect. You can add a dataset or do not label some objects based on your needs.
Brightness of Bounding Boxes Brightness of Bounding Boxes	Horizontal coordinate: Brightness of the image in the target bounding box. A larger value indicates brighter image. Vertical coordinate: Number of bounding boxes in all images	Generally, the chart of the metric is in normal distribution. The metric is used to determine the brightness of an object to be detected. In some special scenarios, the brightness of an object is low and may not meet the requirements.

Metric	Description	Explanation
Blur Score of Bounding Boxes Clarity of Bounding Boxes	Horizontal coordinate: Clarity of the image in the target bounding box. A larger value indicates higher image clarity. Vertical coordinate: Number of bounding boxes in all images	The metric is used to determine whether the object to be detected is blurred. For example, a moving object may become blurred during collection and its data needs to be collected again.

7.6 Labeling Data

Model training requires a large amount of labeled data. Therefore, before training a model, label data. You can create a manual labeling job labeled by one person or by a group of persons (team labeling), or enable auto labeling to quickly label images. You can also modify existing labels, or delete them and re-label.

- Manual labeling: allows you to manually label data.
- Auto labeling: allows you to automatically label remaining data after a small amount of data is manually labeled.
- Team labeling: allows you to perform collaborative labeling for a large amount of data.

For details about data labeling, see **Introduction to Data Labeling**.

7.7 Publishing Data

7.7.1 Introduction to Data Publishing

ModelArts distinguishes data of the same source according to versions processed or labeled at different time, which facilitates the selection of dataset versions for subsequent model building and development.

About Dataset Versions

- For a newly created dataset (before publishing), there is no dataset version information. The dataset must be published before being used for model development or training.
- The default naming rules of dataset versions are V001 and V002 in ascending order. You can customize the version number during publishing.
- You can set any version to the current version. Then the details of the version are displayed on the dataset details page.
- You can obtain the dataset in the manifest file format corresponding to each dataset version based on the value of **Storage Path**. The dataset can be used when you import data or filter hard examples.
- The version of a table dataset cannot be changed.

7.7.2 Publishing a Data Version

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. Locate the row containing the target dataset and click **Publish** in the **Operation** column. Alternatively, click the dataset name to go to the **Dashboard** tab page of the dataset, and click **Publish** in the upper right corner.
3. In the displayed dialog box, set the parameters and click **OK**.

Figure 7-41 Publishing a dataset version

Table 7-21 Parameters for publishing a dataset

Parameter	Description
Version	The naming rules of V001 and V002 in ascending order are used by default. A version name can be customized. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Format	Only table datasets support version format setting. Available values are **CSV** and **CarbonData**. **NOTE** If the exported CSV file contains any command starting with =, +, -, or @, ModelArts automatically adds the Tab setting and escapes the double quotation marks (") for security purposes.

Parameter	Description
Splitting	Only image classification, object detection, text classification, and sound classification datasets support data splitting. By default, this function is disabled. After this function is enabled, set the training and validation ratios. Enter a value ranging from 0 to 1 for **Training Set Ratio**. After the training set ratio is set, the validation set ratio is determined. The sum of the training set ratio and the validation set ratio is 1. **NOTE** To ensure the model accuracy, you are advised to set the training set ratio to 0.8 or 0.9. The training set ratio is the ratio of sample data used for model training. The validation set ratio is the ratio of the sample data used for model validation. The training and validation ratios affect the performance of training templates.
Description	Description of the current dataset version.
Hard Example	Only image classification and object detection datasets support hard example attributes. By default, this function is disabled. After this function is enabled, information such as the hard example attributes of the dataset are written to the corresponding manifest file.

Directory Structure of Dataset Versions

Datasets are managed based on OBS directories. After a new version is published, the directory is generated based on the new version in the output dataset path.

Take an image classification dataset as an example. After the dataset is published, the directory structure of related files generated in OBS is as follows:

```
|-- user-specified-output-path
   |-- DatasetName-datasetId
      |-- annotation
         |-- VersionMame1
            |-- VersionMame1.manifest
         |-- VersionMame2
            ...
         |-- ...
```

The following uses object detection as an example. If a manifest file is imported to the dataset, the following provides the directory structure of related files after the dataset is published:

```
|-- user-specified-output-path
   |-- DatasetName-datasetId
      |-- annotation
         |-- VersionMame1
            |-- VersionMame1.manifest
            |-- annotation
               |-- file1.xml
```



```
        |-- VersionMame2
            ...
        |-- ...
```

Take video labeling as an example. After the dataset is published, the labeling result file (XML) is stored in the dataset output directory.

```
|-- user-specified-output-path
    |-- DatasetName-datasetId
        |-- annotation
            |-- VersionMame1
                |-- VersionMame1.manifest
                |-- annotations
                  |-- images
                      |-- videoName1
                          |-- videoName1.timestamp.xml
                       |-- videoName2
                          |-- videoName2.timestamp.xml
        |-- VersionMame2
            ...
        |-- ...
```

The key frames for video labeling are stored in the dataset input directory.

```
|-- user-specified-input-path
    |-- images
      |-- videoName1
          |-- videoName1.timestamp.jpg
       |-- videoName2
          |-- videoName2.timestamp.jpg
```

7.7.3 Managing Data Versions

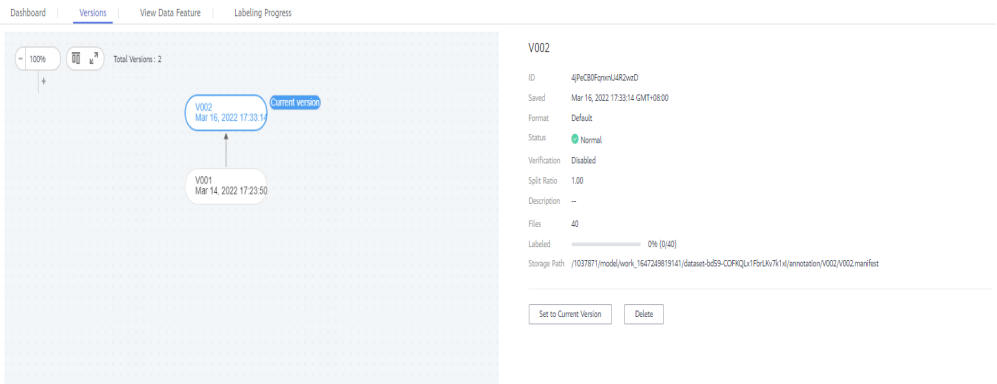
During data preparation, you can publish data into multiple versions for dataset management. You can view version updates, set the current version, and delete versions.

Viewing Dataset Version Updates

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, choose **More > Manage Version** in the **Operation** column. The **Manage Version** tab page is displayed.

   You can view basic information about the dataset, and view the versions and publish time on the left.

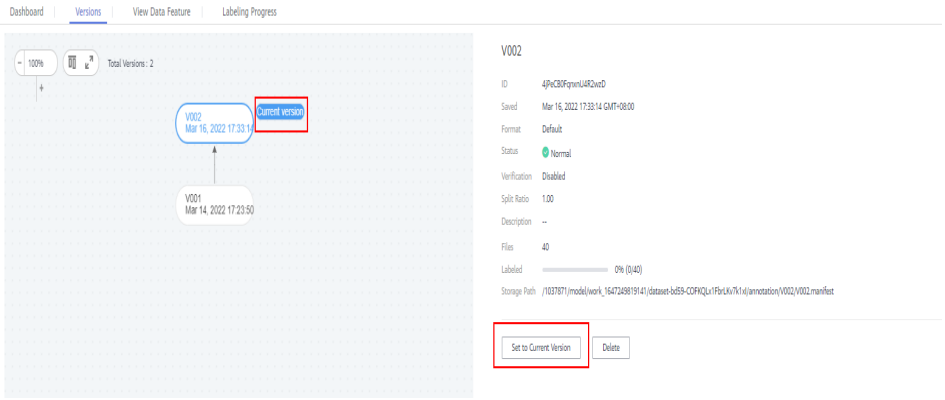
   Figure 7-42 Viewing dataset versions



Setting to Current Version

1. Log in to the ModelArts management console.. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, choose **More** > **Manage Version** in the **Operation** column. The **Manage Version** tab page is displayed.
3. On the **Manage Version** tab page, select the desired dataset version, and click **Set to Current Version** in the basic information area on the right. After the setting is complete, **Current version** is displayed to the right of the version name.

Figure 7-43 Setting to current version



NOTE

Only the version in **Normal** status can be set to the current version.

Deleting a Dataset Version

1. Log in to the ModelArts management console.. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, choose **More** > **Manage Version** in the **Operation** column. The **Manage Version** tab page is displayed.
3. Locate the row that contains the target version, and click **Delete** in the **Operation** column. In the dialog box that is displayed, click **OK**.

NOTE

Deleting a dataset version does not remove the original data. Data and its labeling information are still stored in the OBS directory. However, this affects version management. Exercise caution when performing this operation.

7.8 Exporting Data

7.8.1 Introduction to Exporting Data

You can select data or filter data based on the filter criteria in a dataset and export to a new dataset or the specified OBS path. The historical export records can be viewed in task history.

Only datasets of image classification, object detection, and image segmentation types can be exported.

- For image classification datasets, only the label files in TXT format can be exported.
- For object detection datasets, only XML label files in Pascal VOC format can be exported.
- For image segmentation datasets, only XML label files in Pascal VOC format and mask images can be exported.

7.8.2 Exporting Data to a New Dataset

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, select an image dataset and click the dataset name to go to the **Dashboard** tab page of the dataset.
3. Click **Export** in the upper right corner. In the displayed **Export To** dialog box, enter the related information and click **OK**.

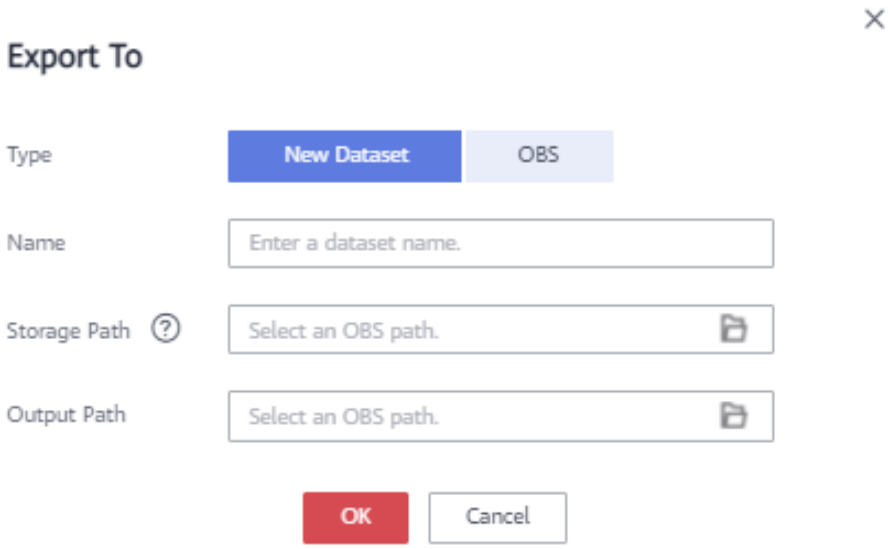
   Type: **New Dataset**.

   Name: name of the new dataset

   Storage Path: input path of the new dataset, that is, the OBS path where the data to be exported is stored

   Output Path: output path of the new dataset, that is, the output path after labeling is complete The output path cannot be the same as the storage path, and the output path cannot be a subdirectory of the storage path.

   Figure 7-44 Exporting to a new dataset



4. After the data is exported, view it in the specified path. After the data is exported, you can view the new dataset in the dataset list.
5. On the **Dashboard** tab page, click **Export History** in the upper right corner. In the displayed dialog box, view the task history of the dataset.

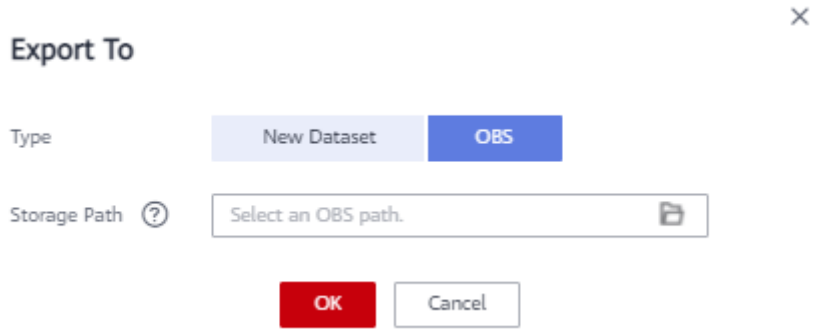
7.8.3 Exporting Data to OBS

1. Log in to the **ModelArts management console.**. In the navigation pane, choose **Data Management** > **Datasets**.
2. In the dataset list, select an image dataset and click the dataset name to go to the **Dashboard** tab page of the dataset.
3. Click **Export** in the upper right corner. In the displayed **Export To** dialog box, enter the related information and click **OK**.

   Type: **OBS**.

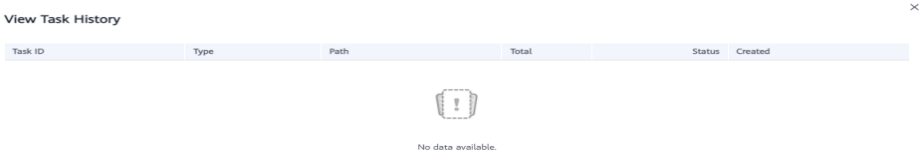
   Storage Path: path where the data to be exported is stored. You are advised not to save data to the input or output path of the current dataset.

   Figure 7-45 Exporting data to OBS



4. After the data is exported, view it in the specified path.
5. On the **Dashboard** tab page, click **Export History** in the upper right corner. In the displayed dialog box, view the task history of the dataset.

   Figure 7-46 Viewing the task history



8 Model Training

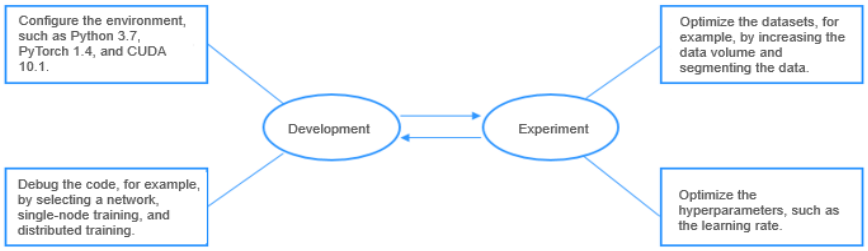
8.1 Model Training Process

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 8-1 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 8-2 ModelArts Standard model training process

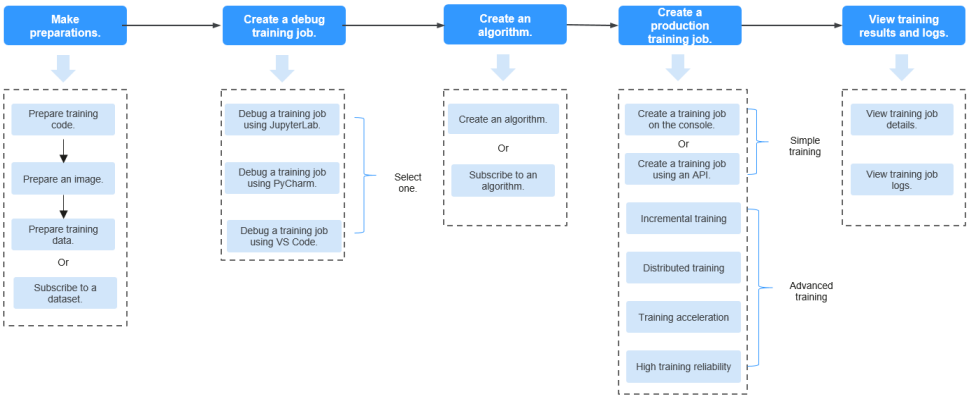


Table 8-1 ModelArts Standard model training process

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 unpreprocessed 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 or subscribe to an algorithm from AI Gallery.
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 8-2**. • 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 8-2 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 and those subscribed from AI Gallery. This allows you to quickly create training jobs and build models using these algorithms.
Using a subscribed algorithm to create a training job	You can subscribe to algorithms in AI Gallery to quickly create training jobs and build models.

8.2 Preparing Model Training Code

8.2.1 Boot File of a Preset Image

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.

- All Ascend accelerators run on Arm servers, which means the upper-layer Docker images are Arm images.
- The NVIDIA CUDA (unified computing architecture) driver is installed in images for GPU scenarios. The Huawei Cloud CANN (heterogeneous computing architecture) driver is installed in images powered by the Ascend-Powered-Engine, adapting to the underlying hardware version.

When a training job is submitted, ModelArts Standard automatically runs the boot file. The boot process repeats as many times as there are training cards.

In a single-node job, the boot file is executed once for each card in a task. For example, if there is one card, the boot file is executed once. If there are eight cards, the boot file is executed eight times. So, do not listen on ports in the boot file.

The following environment variables are set 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 code example **mindspore-verification.py** of Ascend-Powered-Engine, see **the training script mindspore-verification.py**.

The command for starting Ascend-Powered-Engine in standalone mode is the same as that in distributed mode.

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.

For details about the code example of the PyTorch-GPU framework, see **Example: Creating a DDP Distributed Training Job (PyTorch + GPU)** (Method 1).

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>**, and **--job_name <MA_TASK_NAME>**.

The following parameters must be parsed in the boot file.

- *VC_TASK_INDEX*: task serial number, for example, **0/1/2**.
- *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**.

For details, see **the example code file mnist.py (single-node) of the TensorFlow-GPU framework**.

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
```

8.2.2 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 **Boot File of a Preset Image**.

- Input path

  The training data must be uploaded to an OBS bucket or stored in the **dataset**. 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.
   - For details about how to install the Python dependency package, see **How Do I Create a Training Job When a Dependency Package Is Referenced by the Model to Be Trained?**
   - For details about how to install a C++ dependency library, see **How Do I Install a Library That C++ Depends on?**
   - For details about how to load parameters to a pre-trained model, see **How Do I Load Some Well Trained Parameters During Job Training?**

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'))
```

8.2.3 Developing Code for Training Using a Custom Image

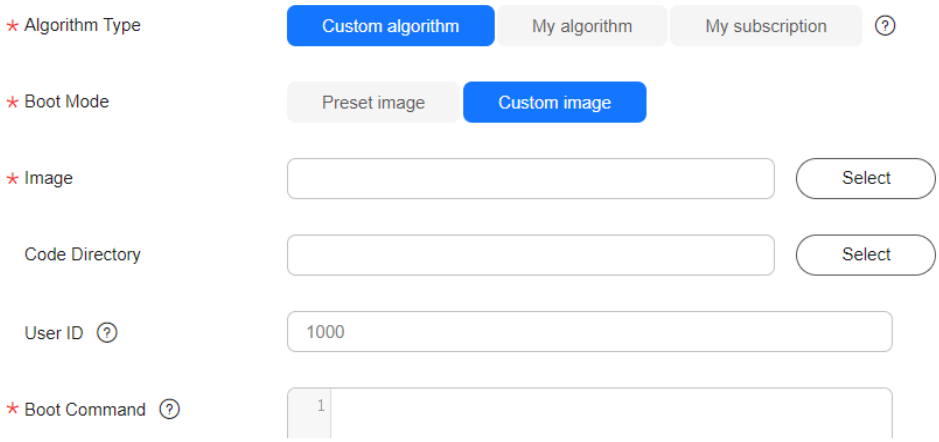
If the preset images offered by ModelArts Standard do not meet your needs, create custom images for model training.

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.

Figure 8-3 Selecting a custom image



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 the NVIDIA Collective Communication Library (NCCL). 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 8-3 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": "jobxxxxxxxx-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 8-4**.

Table 8-4 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.

8.3 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.

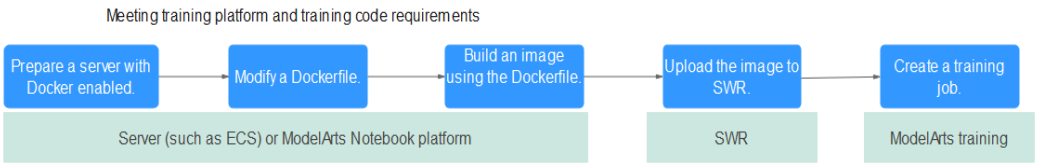
Table 8-5 ModelArts training base images

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_1804-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 8-4** shows the process of creating an image.

Figure 8-4 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:

- **Creating a Custom Image for Training (PyTorch + CPU/GPU)**
- **Creating a Custom Image for Training (MPI + CPU/GPU)**
- **Creating a Custom Image for Training (TensorFlow + GPU)**

8.4 Creating a Debug Training Job

8.4.1 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, 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.

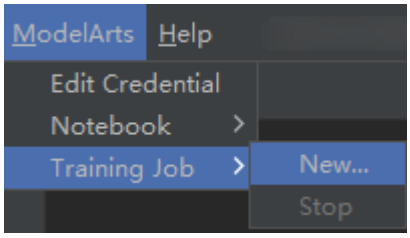
Prerequisites

- **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 8-5 Configuring job parameters

2. In the displayed dialog box, configure the training job parameters. For details, see **Table 8-6**.

Table 8-6 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. The supported AI engines are the same as **the frequently-used frameworks supported by training jobs** on the ModelArts management console.
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.

Parameter	Description
Data OBS Path	OBS path for storing training data, for example, **/test-modelarts2/mnist/dataset-mnist/**, in which **test-modelarts2** indicates a bucket name.
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**. Dedicated resource pool specifications are displayed only for users who have purchased dedicated resource pools.
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 nodes and the total number of nodes are displayed. The value of **Compute Nodes** cannot exceed the number of available nodes.

Figure 8-6 Configuring training job parameter (public resource pool)

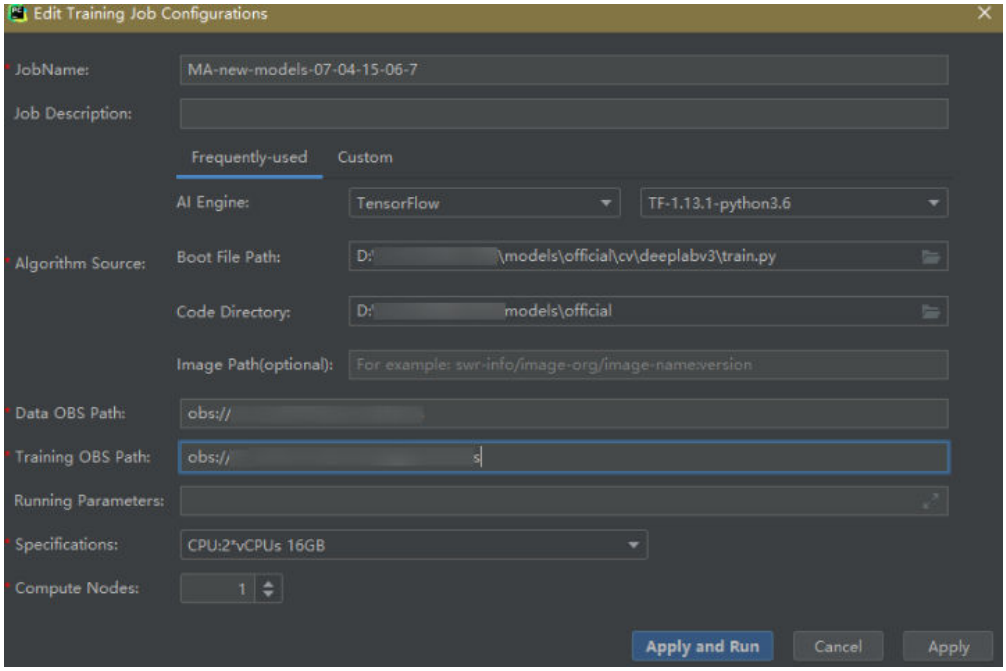


Figure 8-7 Configuring training job parameters (dedicated resource pool)

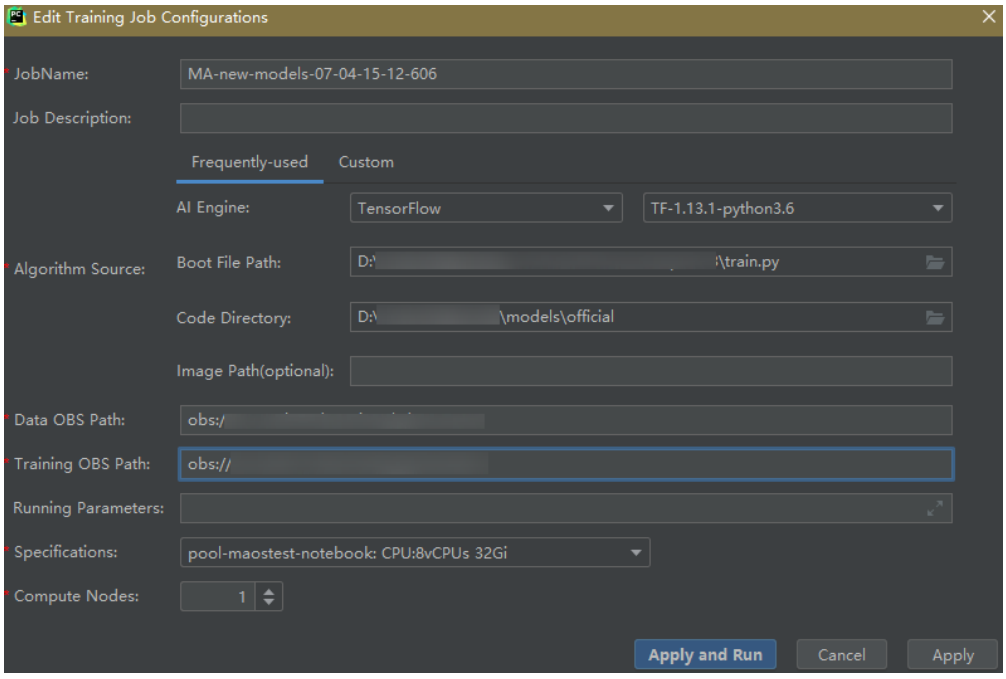
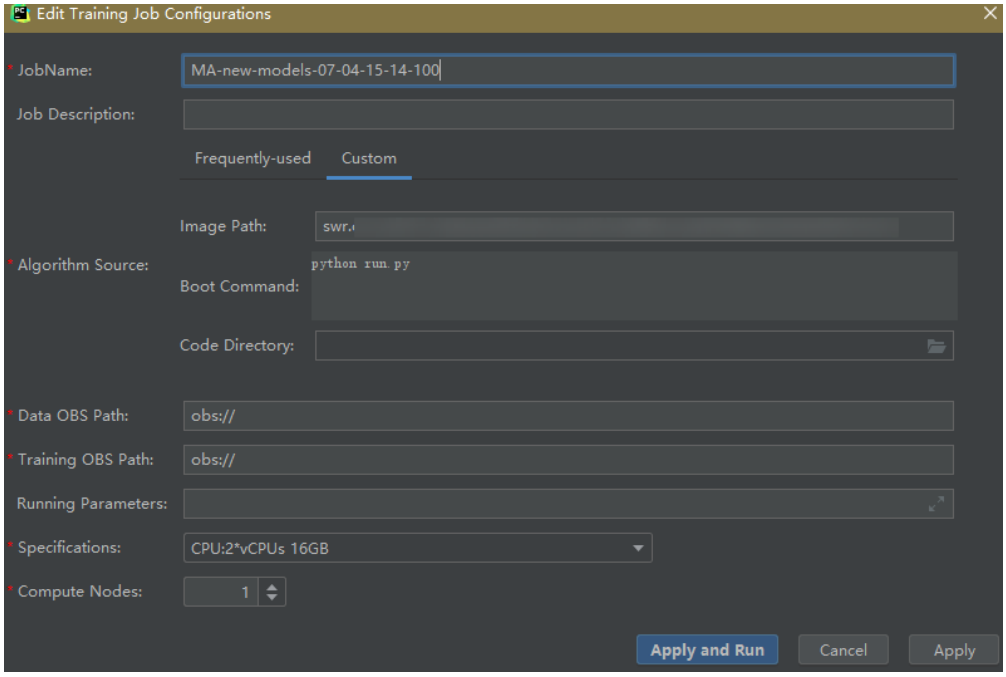


Figure 8-8 Configuring training job parameters (custom image)

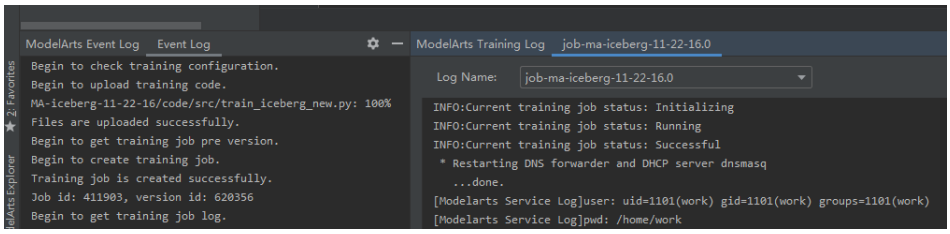


3. Click **Apply and Run**. Then, local code is automatically uploaded to the cloud and training is started. The training job status is displayed in the **Training Log** area in real time. If information similar to **Current training job status: Successful** is displayed in the training logs, the training 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**.

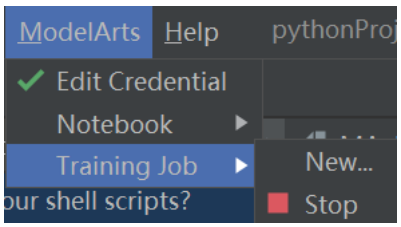
Figure 8-9 Training log example



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 8-10 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.

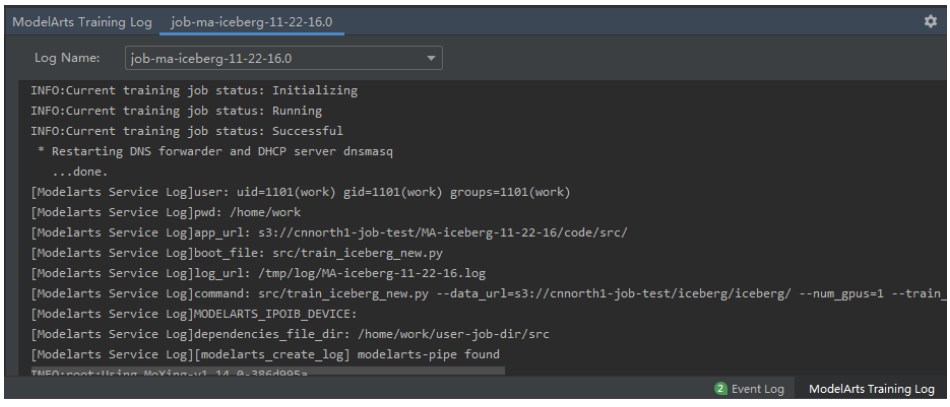
  For example, when the **train-job-01** job is submitted, a folder named **train-job-01** is created in the **test-modelarts2** bucket. In this folder, three sub-folders (**output**, **log**, and **code**) are created to store the outputted model, logs, and training code, respectively. Sub-folders will 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 8-11 Viewing training logs



8.5 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 subscribed algorithm

  You can directly subscribe to algorithms in ModelArts AI Gallery and use them to build models without writing code.

- Using a preset image

  To use a custom algorithm, use a framework built in ModelArts. ModelArts supports most mainstream AI engines. For details, see **Built-in Training Engines**. 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.

Procedure

Your locally developed algorithms or algorithms developed using other tools can be uploaded to ModelArts for unified management. Note the following when creating an algorithm:

1. **Prerequisites**

2. Accessing the Algorithm Creation Page
3. Set the algorithm boot mode. The options are as follows:
   - Using a Preset Image
   - Using a Preset Image with Customization
   - Using a Customer Image
4. Configuring Pipelines
5. Configuring Hyperparameters
6. Supported Policies
7. Adding Training Constraints
8. Previewing the Runtime Environment
9. Viewing Algorithm Details

Prerequisites

- Data is available. You can create a dataset in ModelArts or upload an existing dataset used for training to OBS.
- 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**.
- At least one empty folder has been created in OBS for storing the training output.
- The OBS directory you use and ModelArts are in the same region.

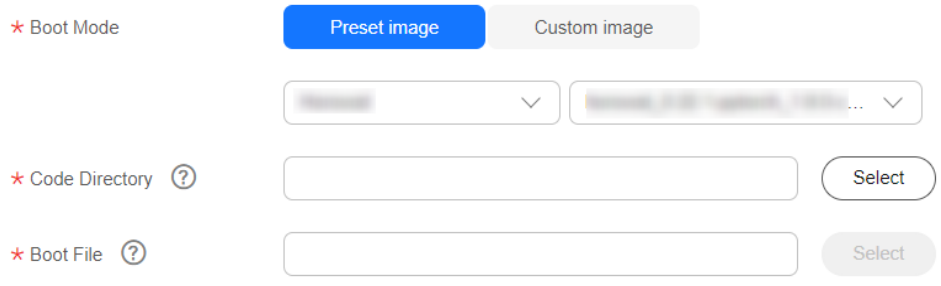
Accessing the Algorithm Creation Page

1. Log in to the ModelArts console. In the navigation pane, choose **Asset Management** > **Algorithm Management**.
2. In the **My algorithm** tab, click **Create Algorithm**. Enter the basic algorithm information, including **Name** and **Description**.

Using a Preset Image

Select a preset image to create an algorithm.

Figure 8-12 Using a preset image to create an algorithm



Set **Image**, **Code Directory**, and **Boot File** based on the algorithm code. Ensure that the framework of the AI image you select is the same as the one you use for editing algorithm code. For example, if TensorFlow is used for editing algorithm code, select a TensorFlow image when you create an algorithm.

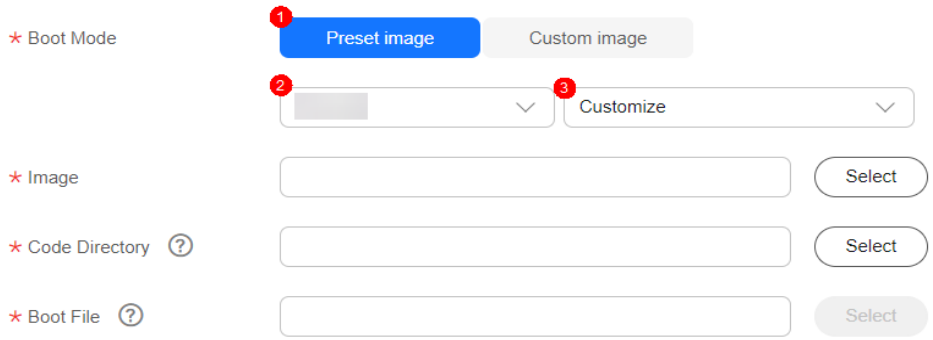
Table 8-7 Parameters

Parameter	Description
Boot Mode	Select **Preset image**. Select a preset image and its version used by the algorithm.
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

Select a preset image with customization to create an algorithm.

Figure 8-13 Creating an algorithm using a preset image with customization



Set **Image**, **Code Directory**, and **Boot File** based on the algorithm code. Ensure that the framework of the AI image you select is the same as the one you use for

editing algorithm code. For example, if TensorFlow is used for editing algorithm code, select a TensorFlow image when you create an algorithm.

Table 8-8 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**.
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 Customer Image

Select a custom image to create an algorithm.

Figure 8-14 Creating an algorithm using a custom image

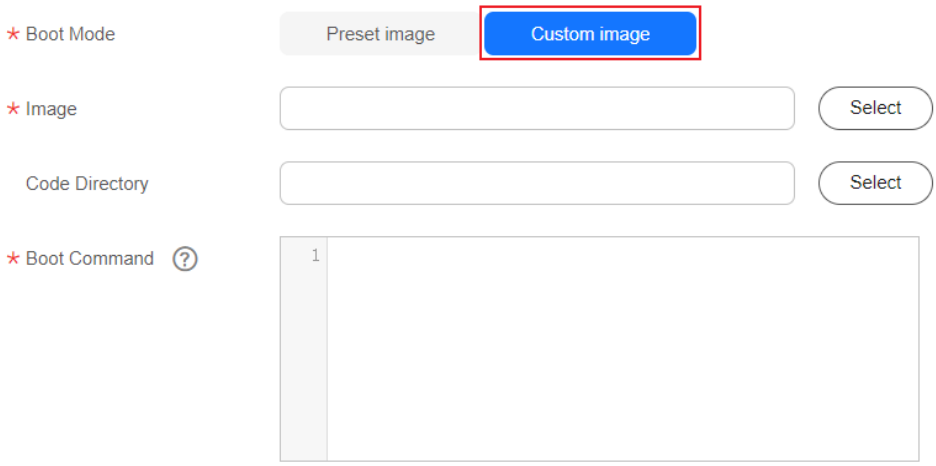


Table 8-9 Parameters

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**.

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. 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.
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: `python ${MA_JOB_DIR}/demo-code/train.py` • If the training boot script is a .sh file, **main.sh** for example, the boot command is as follows: `bash ${MA_JOB_DIR}/demo-code/main.sh` You can use semicolons (;) and ampersands (&&) to combine multiple commands. **demo-code** in the command is the last-level OBS directory where the code is stored. Replace it with the actual one.

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

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

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

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

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

  ```
  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
```

Configuring 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**.

When you use a preset image to create an algorithm, configure the input and output parameters defined in the algorithm code.

- Input configurations

Table 8-10 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**.
Constraints	Enable this parameter to specify the input source. You can select a storage path or ModelArts dataset. This parameter is optional. If you select a ModelArts dataset, set the following parameters: • **Labeling Type**: For details, see **Creating a Manual Labeling Job**. • **Data Format**, which can be **Default**, **CarbonData**, or both. **Default** indicates the manifest format. • **Data Segmentation** is available only for image classification, object detection, text classification, and sound classification datasets. The options are **Segmented dataset**, **Dataset not segmented**, and **Unlimited**. For details, see **Publishing a Data Version**.
Add	Add multiple input data sources based on your algorithm.

- Output configurations

Table 8-11 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.

Parameter	Description
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.

Configuring Hyperparameters

When you create an algorithm, ModelArts Standard 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.

1. Import hyperparameters.

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

2. Edit hyperparameters.

   NOTE

   To ensure data security, do not enter sensitive information, such as plaintext passwords.

   For details, see **Table 8-12**.

Table 8-12 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.
Restrain	Click **Restrain**. 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.

Parameter	Description
Description	Enter the description of the hyperparameter. Only letters, digits, spaces, hyphens (-), underscores (_), commas (,), and periods (.) are allowed.

Supported Policies

Auto search on ModelArts automatically finds the optimal hyperparameters without any code modification. For details about parameter settings, see **Creating a Training Job for Automatic Model Tuning**.

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.

Adding Training Constraints

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

- **Resource Type**: Select the required resource types.
- **Multicard Training**: Choose whether to support multi-card training.
- **Distributed Training**: Choose whether to support distributed training.

Previewing the Runtime Environment

When creating an algorithm, click the arrow on 

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

Viewing Algorithm Details

To view the algorithm information, go to the ModelArts Standard console, choose **Asset Management** > **Algorithm Management**, and click the target algorithm to open its details page.

Publishing an Algorithm

Publish your created algorithm to AI Gallery, making it available for other users to use.

To publish an algorithm, go to the ModelArts Standard console, choose **Asset Management** > **Algorithm Management**, and click the algorithm to open its details page. Click **Publish** to proceed, then configure parameters on the **Publish in AI Gallery** page and click **Publish** again.

- If the algorithm is published for the first time, set **Publish Mode** to **Create Asset**, enter an asset title, and select the available region.

- To update a published algorithm, set **Publish Mode** to **Add asset version**, select an asset title from the **Item Title** drop-down list, and enter the offering version.

NOTE

To publish an asset in AI Gallery for the first time, select **I have read and agree to Huawei Cloud AI Gallery Publishers' Agreement and Huawei Cloud AI Gallery Service Agreement**.

Go to AI Gallery to view the asset details.

Deleting an Algorithm

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.

8.6 Creating a Production 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 a ModelArts API. For details, see **Using PyTorch to Create a Training Job**.

Prerequisites

- Data for training uploaded to an OBS directory.
- At least one empty folder in OBS for storing training output.

NOTE

An OBS bucket not encrypted.

- Account not in arrears (paid resources required for training jobs).
- OBS directory and ModelArts in the same region.
- Access authorization configured. For details, see **Configuring Agency Authorization for ModelArts with One Click**.
- 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 Add tags if you want to manage training jobs by group. For details, see **(Optional) Adding Tags**.

Step 7 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 on the left, choose **Model Training** > **Training Jobs**.
3. Click **Create Training Job**.

Configuring Basic Information

On the **Create Training Job** page, configure parameters.

Table 8-13 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.
Runtime Type	Job type, which can be **Production** or **Debug**. Choose **Production** to create a production job. To create a debug job, see **Creating a Debug Training Job**.

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 8-14**.

Table 8-14 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**.

Parameter	Description
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.
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	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	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.

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 8-15**.

Table 8-15 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: `python ${MA_JOB_DIR}/demo-code/train.py` • If the training boot script is a .sh file, **main.sh** for example, the boot command is as follows: `bash ${MA_JOB_DIR}/demo-code/main.sh` You can use semicolons (;) and ampersands (&&) to combine multiple commands. **demo-code** in the command is the last-level OBS directory where the code is stored. Replace it with the actual one. **NOTE** To ensure data security, do not enter sensitive information, such as plaintext passwords.
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. 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.

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 8-16**.

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 8-16 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 8-10**.
	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. 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: `import argparse` `parser = argparse.ArgumentParser()` `parser.add_argument('--data_path')` `args, unknown = parser.parse_known_args()` `data_path = args.data_path` • If you select **Environment variables**, use this code to obtain the data: `import os` `data_path = os.getenv("data_path", "")`
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 8-11**.

Parameter	Sub-Parameter	Description
	Data path	Click **Data path** and select the storage path to the training output data from an OBS bucket. During training, the system automatically synchronizes files from the local code directory of the training container to the data path. **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: `import argparse` `parser = argparse.ArgumentParser()` `parser.add_argument('--train_url')` `args, unknown = parser.parse_known_args()` `train_url = args.train_url` • If you select **Environment variables**, use this code to obtain the data: `import os` `train_url = os.getenv("train_url", "")`
	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 **Configuring Hyperparameters**. **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**. **NOTE** To ensure data security, do not enter sensitive information, such as plaintext passwords.
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. 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 8-17**.

Table 8-17 Configuring a public resource pool for a training job

Parameter	Description
Resource Pool	Select **Public resource pool**.
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.  **NOTICE** The resource flavor **GPU:*n**tnt004** (*n* indicates a specific number) does not support multi-process training.
Compute Nodes	Select the number of compute nodes as required. The default value is **1**. • If only one compute node 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 compute nodes are 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 set. • This function is disabled by default. ModelArts automatically stores the logs for 30 days. You can download all logs on the job details page to a local path. • After this feature is enabled, set **Job Log Path**. The system permanently stores training logs to the specified OBS path.

Parameter	Description
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.
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. For details, see **Adding a Subscription**. • SMN charges you for the number of notification messages. For details, see **Billing**. • Only training jobs using GPUs or NPUs support **JobHanged** events.
Auto Stop	When using paid resources, you can determine whether to enable 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.

Configuring a Dedicated Resource Pool

To configure a dedicated resource pool, refer to **Table 8-18**.

Table 8-18 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.  **NOTICE** The resource flavor **GPU:*n**tnt004** (*n* indicates a specific number) does not support multi-process training.
Compute Nodes	Select the number of compute nodes as required. The default value is **1**. • If only one compute node 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 compute nodes are 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. For details about the parameters, see **Configuring SFS Turbo Mount Options**. Alternatively, retain the default settings below: `mountOptions:` `- vers=3` `- timeo=600` `- nolock` `- hard` **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. • The mounting path cannot be a **/** directory or a default mounting path, such as **/cache** and **/home/ma-user/modelarts**. • For details about how to set permissions for SFS Turbo folders, see **Permissions Management**.
Persistent Log Saving	If you select CPU or GPU flavors, **Persistent Log Saving** is available for you to set. • This feature is disabled by default. ModelArts automatically stores the logs for 30 days. You can download all logs on the job details page to a local path. • After this feature is enabled, set **Job Log Path**. The system permanently stores training logs to the specified OBS path.

Parameter	Description
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.
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. For details, see **Adding a Subscription**. • SMN charges you for the number of notification messages. For details, see **Billing**. • Only training jobs using GPUs or NPUs support **JobHanged** events.
Auto Stop	When using paid resources, you can determine whether to enable 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.

(Optional) Adding Tags

If you want to manage training jobs by group using tags, select **Configure Now** for **Advanced Configuration** to set tags for training jobs. For details about how to use tags, see **Using TMS Tags to Manage Resources by Group**.

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.

NOTE

You are billed for the resources you choose when your training job runs.

8.7 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

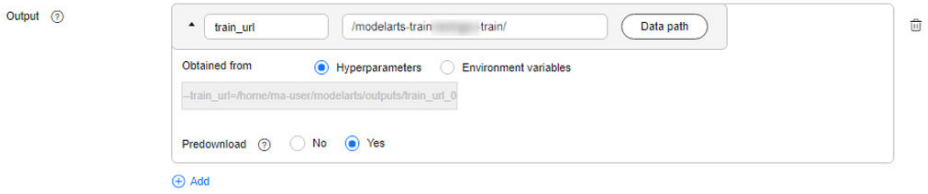
The checkpoint mechanism enables incremental training.

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 8-15 Configuring training output



reload ckpt for PyTorch

1. Use either of the following methods to save a PyTorch model.
   - Save model parameters only.
     ```
     state_dict = model.state_dict()
     torch.save(state_dict, path)
     ```
   - Save the entire model (not recommended).
     ```
     torch.save(model, path)
     ```
2. Save the data generated during model training at regular intervals based on steps and time.

   The data includes the network weight, optimizer weight, and epoch, which will be used to resume the interrupted training.

   ```
   checkpoint = {
           "net": model.state_dict(),
           "optimizer": optimizer.state_dict(),
           "epoch": epoch
   }
   if not os.path.isdir('model_save_dir'):
       os.makedirs('model_save_dir')
   torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
   ```
3. Check the complete code example below.

   ```
   import os
   import argparse
   parser = argparse.ArgumentParser()
   parser.add_argument("--train_url", type=str)
   args, unparsed = parser.parse_known_args()
   args = parser.parse_known_args()
   # train_url is set to /home/ma-user/modelarts/outputs/train_url_0.
   train_url = args.train_url

   # Check whether there is a model file in the output path. If there is no file, the model will be trained from the beginning by default. If there is a model file, the CKPT file with the maximum epoch value will be loaded as the pre-trained model.
   if os.listdir(train_url):
       print('> load last ckpt and continue training!!')
       last_ckpt = sorted([file for file in os.listdir(train_url) if file.endswith(".pth")])[-1]
       local_ckpt_file = os.path.join(train_url, last_ckpt)
       print('last_ckpt:', last_ckpt)
       # Load the checkpoint.
       checkpoint = torch.load(local_ckpt_file)
       # Load the parameters that can be learned by the model.
       model.load_state_dict(checkpoint['net'])
   ```



```
    # Load optimizer parameters.
    optimizer.load_state_dict(checkpoint['optimizer'])
    # Obtain the saved epoch. The model will continue to be trained based on the epoch value.
    start_epoch = checkpoint['epoch']
start = datetime.now()
total_step = len(train_loader)
for epoch in range(start_epoch + 1, args.epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        ...

    # Save the network weight, optimizer weight, and epoch during model training.
    checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
        }
    if not os.path.isdir(train_url):
        os.makedirs(train_url)
        torch.save(checkpoint, os.path.join(train_url, 'ckpt_best_{}.pth'.format(epoch)))
```

8.8 Distributed Model Training

8.8.1 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.
- **Example: Creating a DDP Distributed Training Job (PyTorch + GPU)**: describes the procedure and code example of distributed debugging adaptation.
- **Example: Creating a DDP Distributed Training Job (PyTorch + NPU)**: provides a complete code sample of distributed parallel training for the classification task of ResNet18 on the CIFAR-10 dataset.
- **Debugging a Training Job**: describes how to use the SDK to debug a single-node or multi-node training job on the ModelArts development environment.

8.8.2 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

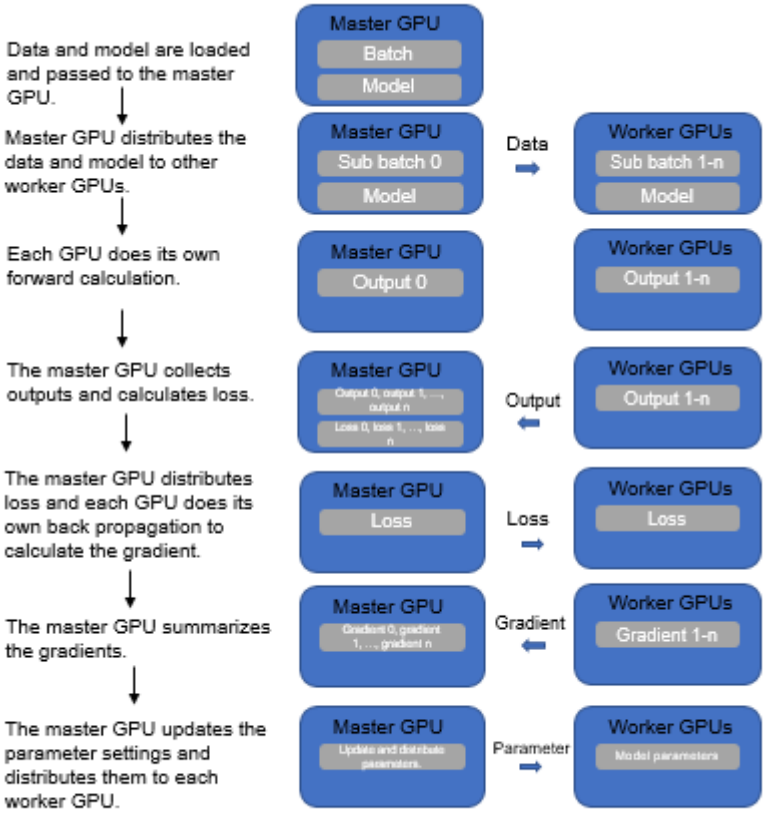
The process of single-node multi-card parallel training is as follows:

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 8-16 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 ###
```


8.8.3 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.

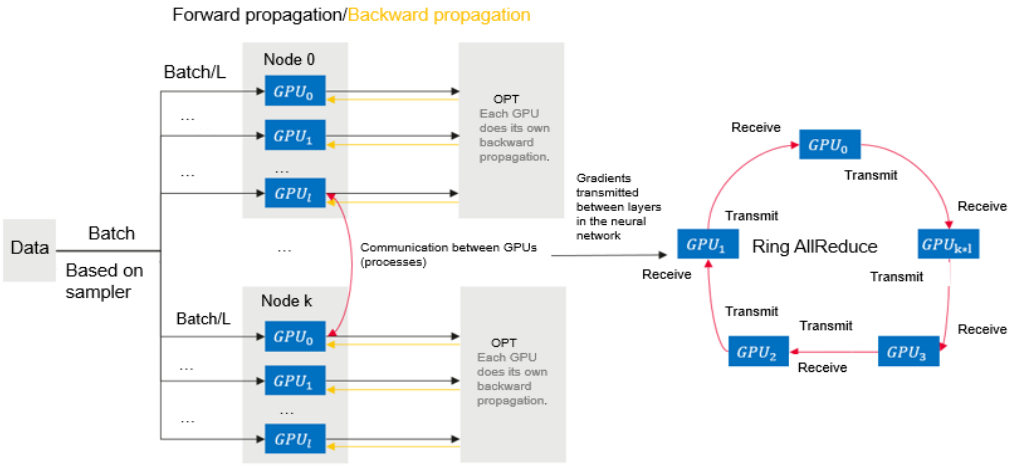
Training Process

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 8-17 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 8-19 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.

8.8.4 Example: Creating a DDP Distributed Training Job (PyTorch + GPU)

This topic describes three methods of using a training job to start PyTorch DDP training and provides their sample code.

- Use PyTorch preset images and run the **mp.spawn** command.
- Use custom images.
  - Run the **torch.distributed.launch** command.
  - Run the **torch.distributed.run** command.

Creating a Training Job

- Method 1: Use the preset PyTorch framework and run the **mp.spawn** command to start a training job.

  For details about parameters for creating a training job, see **Table 8-20**.

  Table 8-20 Creating a training job (preset image)

Parameter	Description
Algorithm Type	Select **Custom algorithm**.
Boot Mode	Select **Preset image** then **PyTorch**. Select a version as needed.
Code Directory	Select the training code path from your OBS bucket, for example, **obs://test-modelarts/code/**.
Boot File	Select the Python boot script of the training job in the code directory, for example, **obs://test-modelarts/code/main.py**.
Hyperparameter	To use a single-node multi-card flavor, set the hyperparameters **world_size** and **rank**. If you choose a flavor with multiple compute nodes, these hyperparameters are automatically set by ModelArts.

- Method 2: Use a custom image and run the **torch.distributed.launch** command to start a training job.

  For details about parameters for creating a training job, see **Table 8-21**.

  Table 8-21 Creating a training job (custom image + **torch.distributed.launch**)

Parameter	Description
Algorithm Type	Select **Custom algorithm**.
Boot Mode	Select **Custom image**.

Parameter	Description
Image	Select a PyTorch image for training.
Code Directory	Select the training code path from your OBS bucket, for example, **obs://test-modelarts/code/**.
Boot Command	Enter the Python boot command of the image, for example: `bash ${MA_JOB_DIR}/code/torchlaunch.sh`

- Method 3: Use a custom image and run the **torch.distributed.run** command to start a training job.

  For details about parameters for creating a training job, see **Table 8-22**.

  Table 8-22 Creating a training job (custom image + **torch.distributed.run**)

Parameter	Description
Algorithm Type	Select **Custom algorithm**.
Boot Mode	Select **Custom image**.
Image	Select a PyTorch image for training.
Code Directory	Select the training code path from your OBS bucket, for example, **obs://test-modelarts/code/**.
Boot Command	Enter the Python boot command of the image, for example: `bash ${MA_JOB_DIR}/code/torchrun.sh`

Code Example

Upload the following files to an OBS bucket:

```
code                          # Root directory of the code
 └─torch_ddp.py                 # Code file for PyTorch DDP training
 └─main.py                    # Boot file for starting training using the PyTorch preset image and the mp.spawn command
 └─torchlaunch.sh             # Boot file for starting training using the custom image and the torch.distributed.launch command
 └─torchrun.sh                # Boot file for starting training using the custom image and the torch.distributed.run command
```

torch_ddp.py

```
import os
import torch
import torch.distributed as dist
import torch.nn as nn
import torch.optim as optim
from torch.nn.parallel import DistributedDataParallel as DDP

# Start training by running mp.spawn.
def init_from_arg(local_rank, base_rank, world_size, init_method):
    rank = base_rank + local_rank
    dist.init_process_group("nccl", rank=rank, init_method=init_method, world_size=world_size)
    ddp_train(local_rank)
```



```
# Start training by running torch.distributed.launch or torch.distributed.run.
def init_from_env():
    dist.init_process_group(backend='nccl', init_method='env://')
    local_rank=int(os.environ["LOCAL_RANK"])
    ddp_train(local_rank)

def cleanup():
    dist.destroy_process_group()

class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = nn.Linear(10, 10)
        self.relu = nn.ReLU()
        self.net2 = nn.Linear(10, 5)
    def forward(self, x):
        return self.net2(self.relu(self.net1(x)))

def ddp_train(device_id):
    # create model and move it to GPU with id rank
    model = ToyModel().to(device_id)
    ddp_model = DDP(model, device_ids=[device_id])
    loss_fn = nn.MSELoss()
    optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)
    optimizer.zero_grad()
    outputs = ddp_model(torch.randn(20, 10))
    labels = torch.randn(20, 5).to(device_id)
    loss_fn(outputs, labels).backward()
    optimizer.step()
    cleanup()

if __name__ == "__main__":
    init_from_env()
```

main.py

```
import argparse
import torch
import torch.multiprocessing as mp

parser = argparse.ArgumentParser(description='ddp demo args')
parser.add_argument('--world_size', type=int, required=True)
parser.add_argument('--rank', type=int, required=True)
parser.add_argument('--init_method', type=str, required=True)
args, unknown = parser.parse_known_args()

if __name__ == "__main__":
    n_gpus = torch.cuda.device_count()
    world_size = n_gpus * args.world_size
    base_rank = n_gpus * args.rank
    # Call the start function in the DDP sample code.
    from torch_ddp import init_from_arg
    mp.spawn(init_from_arg,
        args=(base_rank, world_size, args.init_method),
        nprocs=n_gpus,
        join=True)
```

torchlaunch.sh

```
#!/bin/bash
# Default system environment variables. Do not modify them.
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

# Custom environment variables to specify the Python script and parameters.
```



```
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

CMD="python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr $MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

torchrun.sh

NOTICE

In PyTorch 2.1, you must set **rdzv_backend** to **static: --rdzv_backend=static**.

```
#!/bin/bash
# Default system environment variables. Do not modify them.
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

# Custom environment variables to specify the Python script and parameters.
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

if [[ $NODE_RANK == 0 ]]; then
    EXT_ARGS="--rdzv_conf=is_host=1"
else
    EXT_ARGS=""
fi

CMD="python -m torch.distributed.run \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    $EXT_ARGS \
    --nproc_per_node=$NGPUS_PER_NODE \
    --rdzv_id=$JOB_ID \
    --rdzv_backend=c10d \
    --rdzv_endpoint=$MASTER_ADDR:$MASTER_PORT \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
    "
echo $CMD
$CMD
```

8.8.5 Example: Creating a DDP Distributed Training Job (PyTorch + NPU)

This section describes how to use a custom image and boot command to start PyTorch DDP training powered by Ascend accelerator cards.

Prerequisites

An Ascend accelerator card resource pool is available.

Creating a Training Job

The following table describes the parameters you need to configure during training job creation.

Table 8-23 Parameters for creating a training job

Parameter	Description
Algorithm Type	Select **Custom algorithm**.
Boot Mode	Select **Custom image**.
Image	Select a custom image for training.
Code Directory	Select the code directory required for this training job, for example, **obs://test-modelarts/ascend/code/** in this case.
Boot Command	Python boot command of the image, for example, **bash ${MA_JOB_DIR}/code/run_torch_ddp_npu.sh** in this case. For details about the complete code of the boot script, see **Code Example**.

(Optional) Enabling Ranktable Dynamic Routing

To use ranktable dynamic routing for network acceleration, contact technical support to enable cabinet scheduling permission. Additionally, the training job must meet the following requirements:

- The training job must use Python 3.7 or 3.9.
- The training job must have at least 3 task nodes.
- The training job must use the same rank number in the code. The algorithm accelerates routing by changing the rank number.

Follow these steps and start the training job to accelerate the network.

- Change **NODE_RANK="$VC_TASK_INDEX"** in the training boot script to **NODE_RANK="$RANK_AFTER_ACC"**.
- Change **MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"** in the training boot script to **MASTER_ADDR="${MA_VJ_NAME}-${MA_TASK_NAME}-${MA_MASTER_INDEX}.${MA_VJ_NAME}"**.
- When creating the training job, set the environment variable **ROUTE_PLAN** to **true**. For details, see **Managing Environment Variables of a Training Container**.

Code Example

The following shows an example boot script of a training job.

NOTE

To store the generated plog data, you need to specify the path in the startup script as **/home/ma-user/modelarts/log/modelarts-job-{*id*}/worker-{*index*}/**. The system will automatically upload the ***.log** file in the **/home/ma-user/modelarts/log/** directory to the OBS log directory of your training job. The system will only upload the log files (larger than 0 MB) in the local directory to the corresponding parent directory. Unlike MindSpore, PyTorch NPU plog logs are organized by worker instead of rank ID. PyTorch NPU does not rely on the rank table file.

```
#!/bin/bash

# MA preset envs
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
# also indicates NPU per node
NGPUS_PER_NODE="$MA_NUM_GPUS"

# self-define, it can be changed to >=10000 port
MASTER_PORT="38888"

# replace ${MA_JOB_DIR}/code/torch_ddp.py to the actutal training script
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

export HCCL_WHITELIST_DISABLE=1

# 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}

echo "plog path: ${ASCEND_PROCESS_LOG_PATH}"

# set hccl timeout time in seconds
export HCCL_CONNECT_TIMEOUT=1800

# replace ${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python to the actual python
CMD="${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr=$MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

8.9 Automatic Model Tuning (AutoSearch)

8.9.1 Overview

ModelArts automatically searches for optimal hyperparameters for your models, saving time and effort.

During training, hyperparameters like **learning_rate** and **weight_decay** need to be adjusted. ModelArts hyperparameter search optimizes these settings automatically, outperforming manual tuning in speed and precision.

ModelArts supports the following hyperparameter search algorithms:

- Bayesian Optimization (SMAC)
- Tree-structured Parzen Estimator (TPE)
- Simulated Annealing

Bayesian Optimization (SMAC)

Bayesian optimization assumes a functional relationship between hyperparameters and the objective function. It estimates the mean and variance of objective function values at other search points using Gaussian process regression, based on the evaluation values of the searched hyperparameters. The mean and variance are then used to construct the acquisition function, which identifies the next search point as its maximum value. Bayesian optimization reduces the number of iterations and search time by leveraging previous evaluation results, but it can struggle to find the global optimal solution.

Table 8-24 Bayesian optimization parameters

Parameter	Description	Recommended Value
num_samples	Number of hyperparameter groups to search	This integer ranges from 10 to 20. Larger values increase search time but improve results.
kind	Acquisition function type	This string defaults to **ucb**. Other options are **ei** and **poi**, but it is best to stick with the default.
kappa	Adjustment parameter for the **ucb** acquisition function, representing the upper confidence boundary	This float value should remain unchanged.
xi	Adjustment parameter for **poi** and **ei** acquisition functions.	This float value should remain unchanged.

Tree-structured Parzen Estimator (TPE)

The TPE algorithm uses a Gaussian mixture model to learn model hyperparameters. On each trial, TPE fits two Gaussian mixture models: one to the best objective values and another to the remaining values. It selects the hyperparameter value that maximizes the ratio of these two models.

Table 8-25 TPE parameters

Parameter	Description	Recommended Value
num_samples	Number of hyperparameter groups to search	This integer ranges from 10 to 20. Larger values increase search time but improve results.
n_initial_points	Number of random evaluations of the objective function before using tree-structured parzen estimators	This integer should remain unchanged.
gamma	Quantile used by the TPE algorithm to split l(x) and g(x)	This float value ranges from 0 to 1 and should remain unchanged.

Simulated Annealing

The simulated annealing algorithm is a simple and effective search method that uses the smoothness of the response surface. It starts with a previous trial point and samples each hyperparameter from a distribution similar to the prior, but with a higher concentration around the chosen point. Over time, the algorithm focuses on sampling points closer to the best ones. Occasionally, it may select a runner-up trial as the best to avoid local optima with a certain probability.

Table 8-26 Simulated annealing parameters

Parameter	Description	Recommended Value
num_samples	Number of hyperparameter groups to search	This integer ranges from 10 to 20. Larger values increase search time but improve results.
avg_best_idx	Mean of the geometric distribution used to select trials for exploration, based on their scores	This float value should remain unchanged.
shrink_coef	Rate at which the sampling neighborhood size decreases as more points are explored	This float value should remain unchanged.

8.9.2 Creating a Training Job for Automatic Model Tuning

Context

To use ModelArts hyperparameter search, the AI engine must be either **pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64** or **tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**, and the hyperparameter to be optimized must be a float value.

To perform a hyperparameter search without any code modification, follow these steps:

1. **Preparations**
2. **Creating an Algorithm**
3. **Creating a Training Job**
4. **Viewing Details About a Hyperparameter Search Job**

Preparations

- Create a dataset in ModelArts or upload a training dataset to an OBS directory.
- Upload your training script to an OBS directory. For details about how to develop a training script, see **Developing Code for Training Using a Preset Image**.
- Print search indicator parameters in the training code.
- Create at least one empty folder in OBS for storing training outputs.
- Make sure your account is not in arrears, as training jobs consume resources.
- Make sure your OBS directory and ModelArts are in the same region.

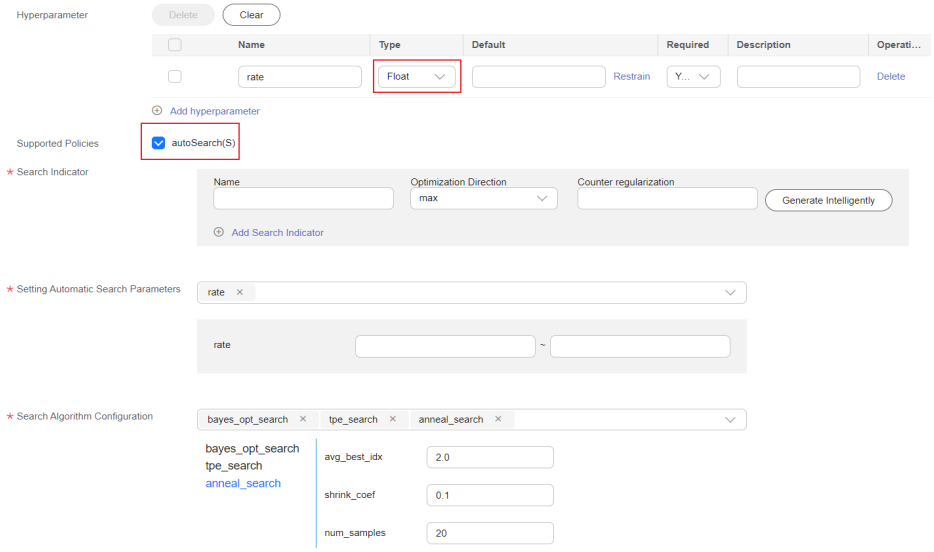
Creating an Algorithm

Log in to the ModelArts console and create an algorithm by referring to **Creating an Algorithm**. The image must use the **pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64** or **tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64** engine.

To define hyperparameters for optimization, specify the name, type, default value, and constraints in **Hyperparameter**. For details, see **Configuring Hyperparameters**.

To enable auto search for the algorithm, select **autoSearch(S)**, print search parameters in the code, and configure the following parameters. ModelArts uses a regular expression to obtain search indicator parameters during an auto search, and then performs hyperparameter optimization based on the specified optimization direction.

Figure 8-18 Enabling auto search



- Search Indicator

  The search indicator represents the value of the objective function, such as loss or accuracy. By optimizing and converging this value in the desired direction, you can find the optimal hyperparameters to enhance model accuracy and convergence speed.

  Table 8-27 Search indicator parameters

Parameter	Description
Name	Enter a search indicator name. This value must be identical to the search indicator parameter in the code.
Optimization Direction	Select **max** or **min**.
Counter regularization	Enter a regular expression or click **Generate Intelligently** to generate a regular expression automatically.

- Setting Automatic Search Parameters

  Select hyperparameters from the **Hyperparameters** configuration. Only float-type hyperparameters are supported. After selecting **autoSearch(S)**, set the value range.

- Search Algorithm Configuration

  ModelArts has three built-in algorithms for hyperparameter search. You can select one or more algorithms as needed. The algorithms and their parameter description are as follows:

  - bayes_opt_search: **Bayesian Optimization (SMAC)**
  - tpe_search: **Tree-structured Parzen Estimator (TPE)**
  - anneal_search: **Simulated Annealing**

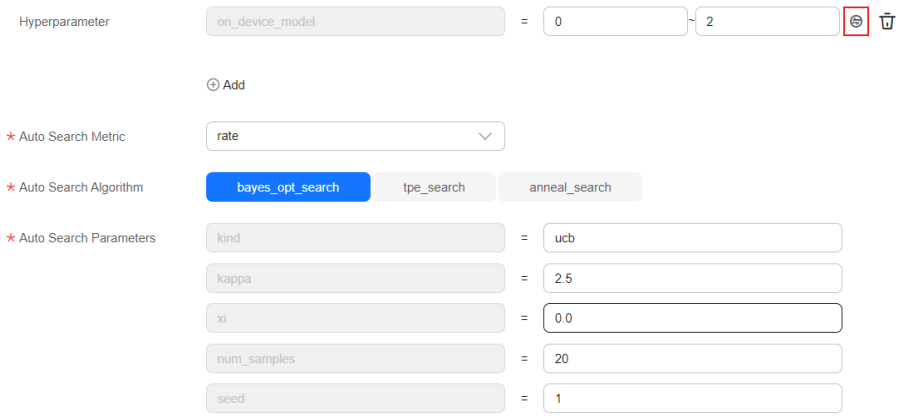
After creating the algorithm, use it to create a training job.

Creating a Training Job

Log in to the ModelArts console and create a training job by referring to **Creating a Production Training Job**.

If you select an algorithm that supports hyperparameter search, click the button for range setting to enable hyperparameter search.

Figure 8-19 Enabling hyperparameter search



After enabling the hyperparameter search, you can configure the search indicator, search algorithm, and search algorithm parameters. These parameters have the same values as the hyperparameters of the algorithm you created.

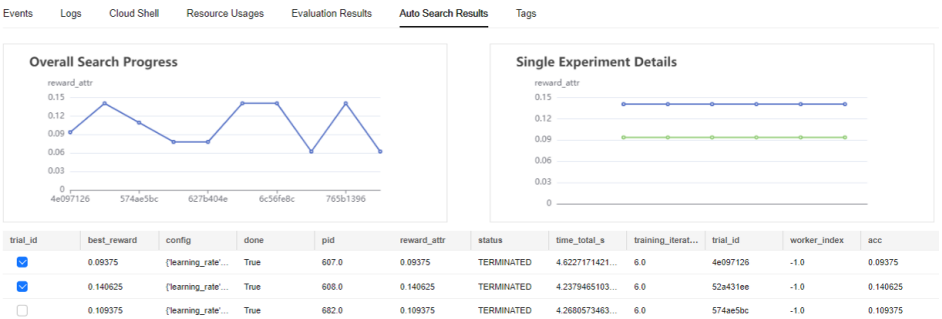
The hyperparameter search job will take some time to run after it is created.

Viewing Details About a Hyperparameter Search Job

After a training job is completed, you can review the results of the automated hyperparameter search to evaluate the job's performance.

If the training job is a hyperparameter search job, go to the training job details page and click the **Auto Search Results** tab to view the hyperparameter search results.

Figure 8-20 Hyperparameter search results



8.10 High Model Training Reliability

8.10.1 Training Job Fault Tolerance Check

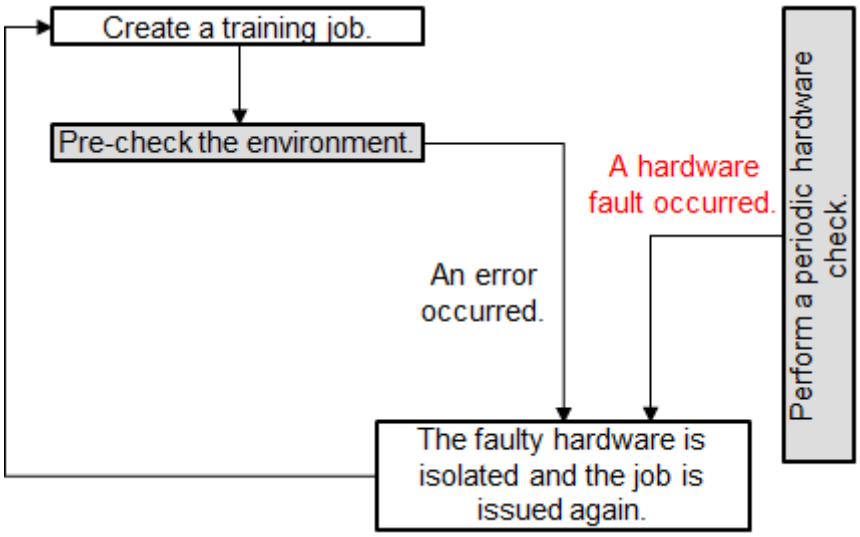
During model training, a training failure may occur due to a hardware fault. For hardware faults, ModelArts provides fault tolerance check to isolate faulty nodes to improve user experience in training.

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.

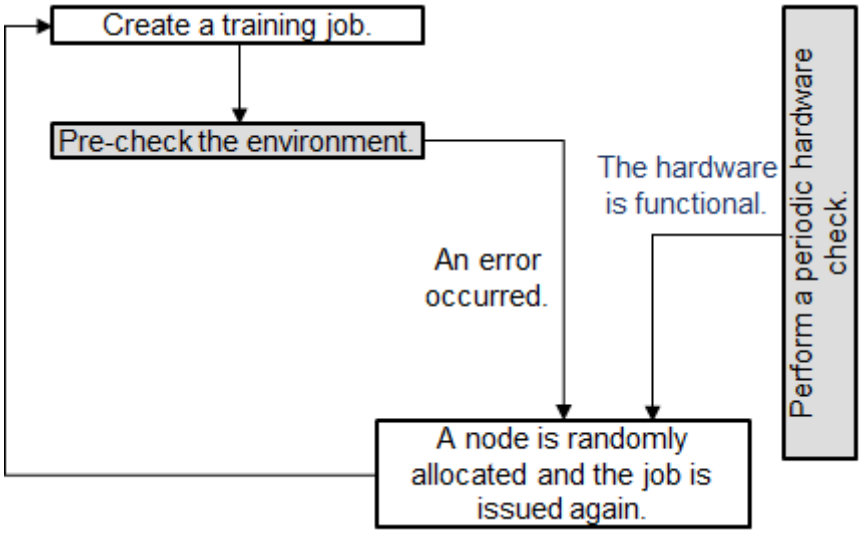
- Scenario 1: The environment pre-check fails, and the hardware is faulty. Then, ModelArts automatically isolates all faulty nodes and issues the training job again.

  Figure 8-21 Pre-check failure and hardware fault



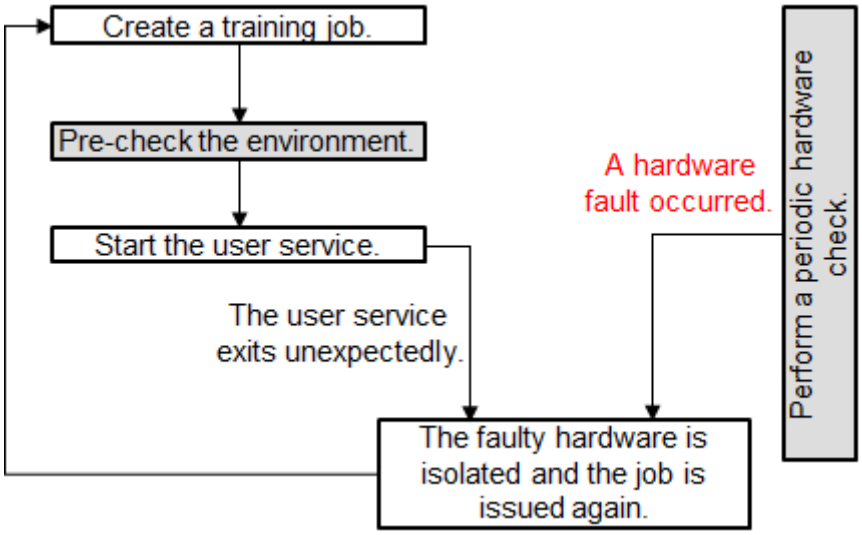
- Scenario 2: The environment pre-check fails but the hardware is functional. Then, ModelArts randomly allocates nodes and issues the training job again.

Figure 8-22 Pre-check failure but functional hardware



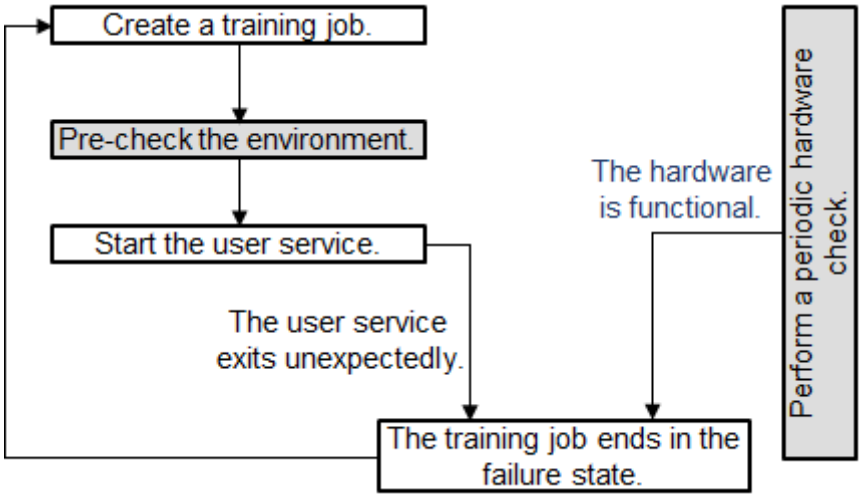
- Scenario 3: The environment pre-check is successful and the user service starts. A hardware fault occurs and the user service exits unexpectedly. Then, ModelArts automatically isolates all faulty nodes and issues the training job again.

Figure 8-23 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 8-24 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**.

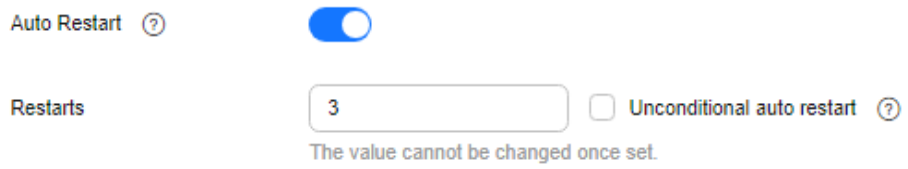
Enabling Fault Tolerance Check

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 8-25 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 8-26 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

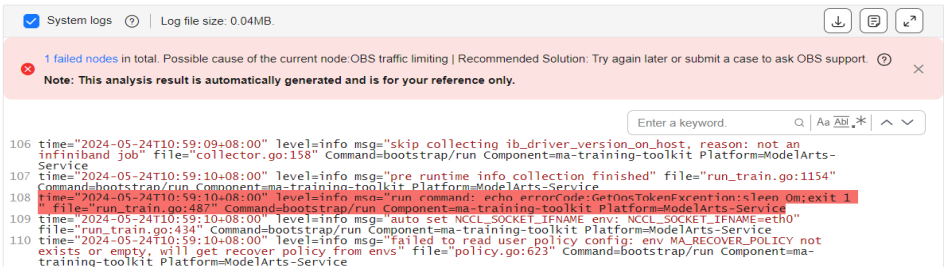
With fault tolerance enabled, if a training job is restarted due to a hardware fault, you can obtain the pre-trained model in the code to restore the training to the state before the restart. To do so, add reload ckpt to the code. For details, see **Resumable Training**.

8.10.2 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).

Figure 8-27 Identifying training faults



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.

8.10.3 Detecting Training Job Suspension

Overview

A training job may be suspended due to unknown reasons. If the suspension cannot be detected promptly, resources cannot be released, leading to a waste. To minimize resource cost and improve user experience, ModelArts provides suspension detection for training jobs. With this function, suspension can be automatically detected and displayed on the log details page. You can also enable notification so that you can be promptly notified of job suspension.

Detection Rules

Determine whether a job is suspended based on the monitored job process status and resource usage. A process is started to periodically monitor the changes of the two metrics.

- Job process status: If the process I/O of a training job changes, the next detection period starts. If the process I/O of the job remains unchanged in multiple detection periods, the resource usage detection starts.
- Resource usage: If the process I/O remains unchanged, the system collects the GPU or NPU usage within a certain period of time and determines whether the resource usage changes based on the variance and median of the GPU or NPU usage within the period. If the GPU usage is not changed, the job is suspended.

Constraints

Suspension can be detected only for training jobs that run on GPUs or NPUs.

Procedure

Suspension detection is automatically performed during job running. No additional configuration is required. After detecting that a job is suspended, the system displays a message on the training job details page, indicating that the job may be suspended. If you want to be notified of suspension (by SMS or email), enable event notification on the job creation page.

Cases

Common cases and solutions to training job suspension are as follows:

- **Data Replication Suspension**
- **Suspension Before Training**
- **Suspension During Training**
- **Suspension in the Last Training Epoch**

8.10.4 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.

Figure 8-28 Viewing fault recovery details

8.10.5 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.

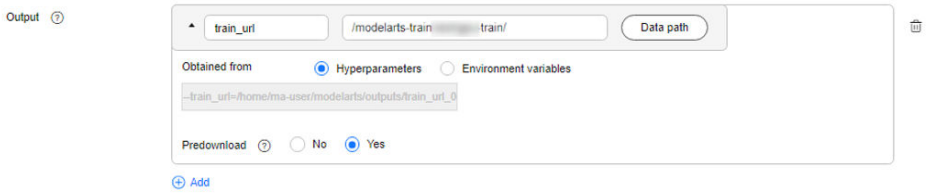
To resume a training job, load a checkpoint and use the checkpoint information to initialize the training status. To do so, add reload ckpt to the code.

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.

Figure 8-29 Configuring training output



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.

reload ckpt for PyTorch

1. Use either of the following methods to save a PyTorch model.
   - Save model parameters only.
     ```
     state_dict = model.state_dict()
     torch.save(state_dict, path)
     ```
   - Save the entire model (not recommended).
     ```
     torch.save(model, path)
     ```
2. Save the data generated during model training at regular intervals based on steps and time.

   The data includes the network weight, optimizer weight, and epoch, which will be used to resume the interrupted training.

   ```
   checkpoint = {
           "net": model.state_dict(),
           "optimizer": optimizer.state_dict(),
           "epoch": epoch
   }
   if not os.path.isdir('model_save_dir'):
       os.makedirs('model_save_dir')
   torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
   ```
3. Check the complete code example below.
   ```
   import os
   import argparse
   parser = argparse.ArgumentParser()
   ```



```
parser.add_argument("--train_url", type=str)
args, unparsed = parser.parse_known_args()
args = parser.parse_known_args()
# train_url is set to /home/ma-user/modelarts/outputs/train_url_0.
train_url = args.train_url

# Check whether there is a model file in the output path. If there is no file, the model will be trained from the beginning by default. If there is a model file, the CKPT file with the maximum epoch value will be loaded as the pre-trained model.
if os.listdir(train_url):
    print('> load last ckpt and continue training!!')
    last_ckpt = sorted([file for file in os.listdir(train_url) if file.endswith(".pth")])[-1]
    local_ckpt_file = os.path.join(train_url, last_ckpt)
    print('last_ckpt:', last_ckpt)
    # Load the checkpoint.
    checkpoint = torch.load(local_ckpt_file)
    # Load the parameters that can be learned by the model.
    model.load_state_dict(checkpoint['net'])
    # Load optimizer parameters.
    optimizer.load_state_dict(checkpoint['optimizer'])
    # Obtain the saved epoch. The model will continue to be trained based on the epoch value.
    start_epoch = checkpoint['epoch']
start = datetime.now()
total_step = len(train_loader)
for epoch in range(start_epoch + 1, args.epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        ...

    # Save the network weight, optimizer weight, and epoch during model training.
    checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
        }
    if not os.path.isdir(train_url):
        os.makedirs(train_url)
        torch.save(checkpoint, os.path.join(train_url, 'ckpt_best_{}.pth'.format(epoch)))
```

8.10.6 Enabling Unconditional Auto Restart

Context

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**.

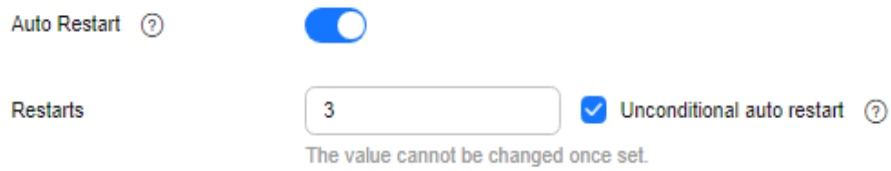
Procedure

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 8-30 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, set **fault-tolerance/job-retry-num** to a value ranging from **1** to **128**. To enable 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"
      }
    }
  }
  ```

8.11 Managing Model Training Jobs

8.11.1 Viewing Training Job Details

1. Log in to the ModelArts console.
2. In the navigation pane, choose **Model Training** > **Training Jobs**.
3. In the training job list, click the target job name to switch to the training job details page.
4. On the left side of the training job details page, view basic job settings and algorithm parameters.
   - **Basic job settings**

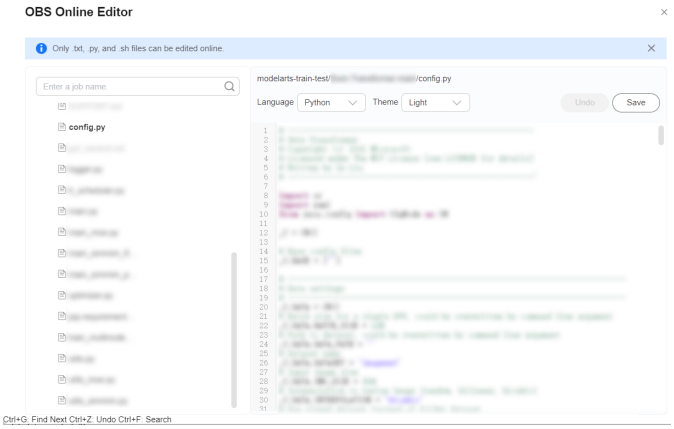
Table 8-28 Basic job settings

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	Running duration of the 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 8-29 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.
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.

Parameter	Description
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 compute nodes set 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.

Parameter	Description
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.
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.

8.11.2 Viewing the Resource Usage of a Training Job

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.

Figure 8-31 Resource Usages

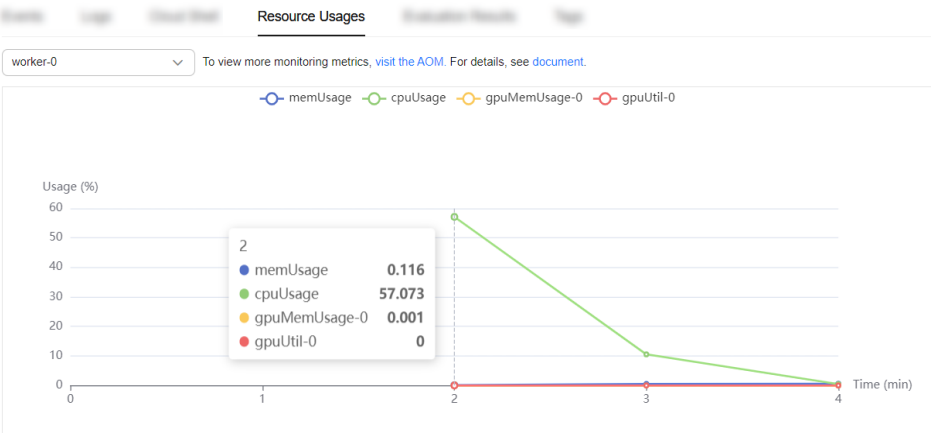


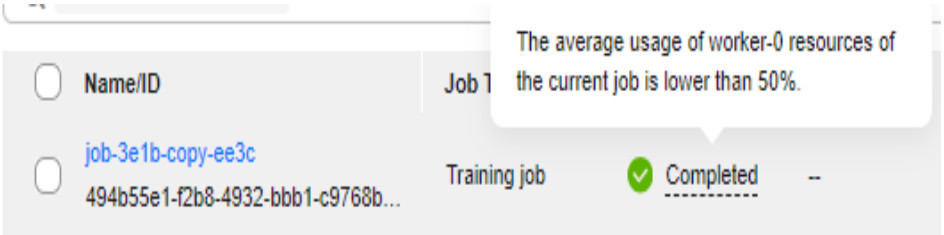
Table 8-30 Parameters

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.

Figure 8-32 Job resource usage in the 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.

8.11.3 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)
        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'][:]
    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__":
    evalution()
```

8.11.4 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: 1 hour)
- The training job has been stopped. (Maximum running duration: 3 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.

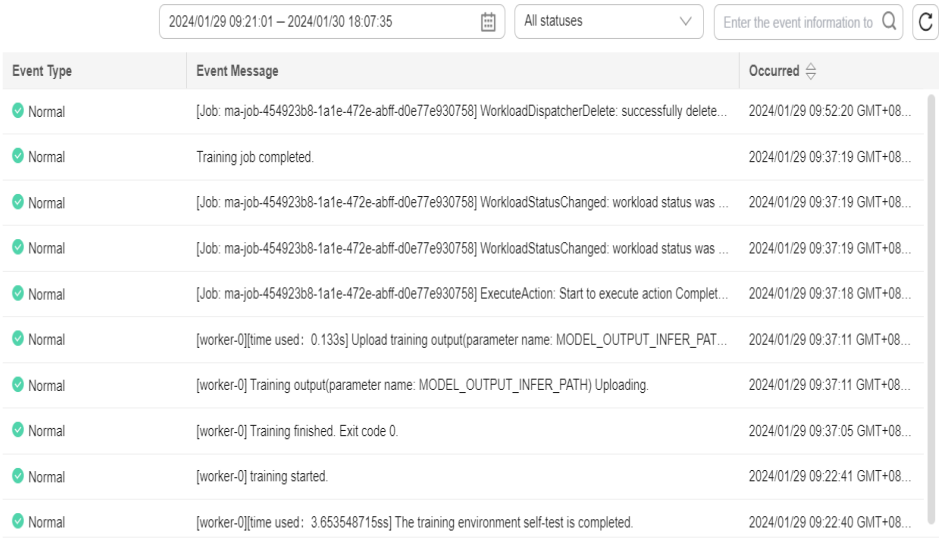
During the training process, key events can be manually or automatically refreshed.

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.

Figure 8-33 Events



8.11.5 Viewing Training Job Logs

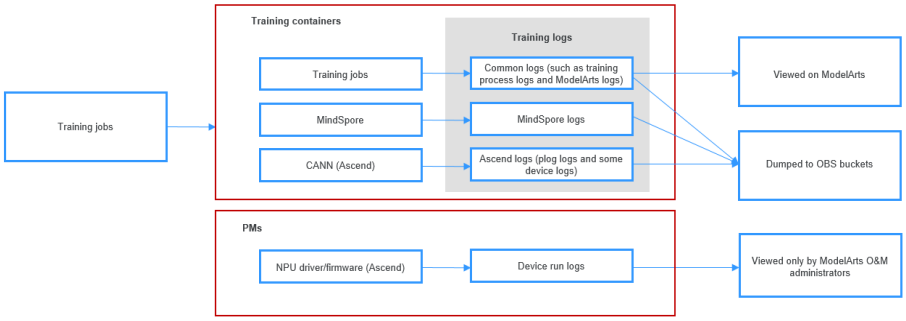
Overview

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 8-34 ModelArts training logs



NOTE

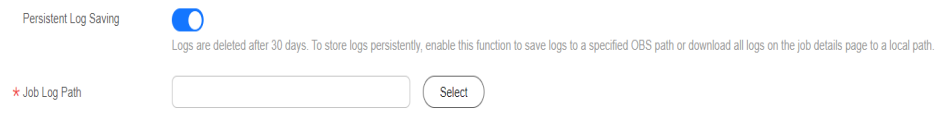
Separate MindSpore logs are generated only in the MindSpore+Ascend training scenario. Logs of other AI engines are contained in common logs.

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. For Ascend training, you need to configure the OBS path for storing training logs by default. You need to manually enable **Persistent Log Saving** for training jobs using other resources.

  Figure 8-35 Enabling persistent log saving



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 8-31 Common log types

Type	Description
Training process log	Standard output of your training code.
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 8-32 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. `# 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}`	**~/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. `# 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}`	**~/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

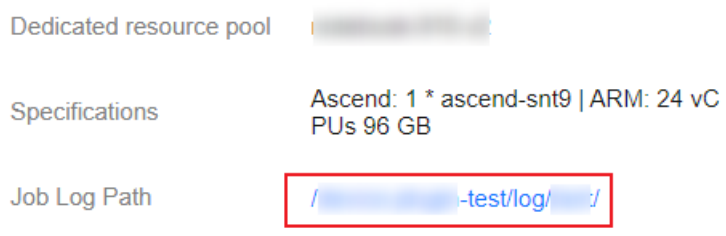
Type	Description	Name
proc log	**proc log** is a redirection file of single-node training logs, helping you quickly obtain logs of a compute node. Training jobs using custom images do not involve **proc log**. **proc log** for training using a preset image is generated in the training container and automatically 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 compute nodes multiplied by the number of NPUs minus 1. If a single compute node 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. `# 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}`	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 compute node 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 compute nodes. 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.

Figure 8-36 Job Log Path



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.

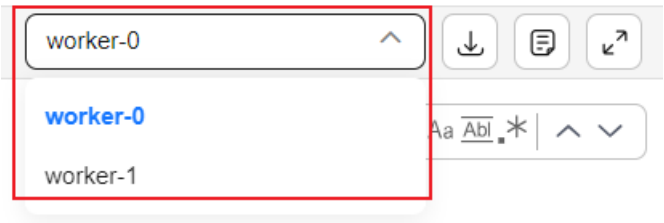
Viewing Training Job Logs

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

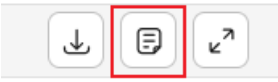
  You can preview training logs on the system log pane. If multiple compute nodes are used, you can choose the target node from the drop-down list on the right.

  Figure 8-37 Viewing logs of different compute nodes

  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.

  Figure 8-38 Viewing all logs

  NOTE

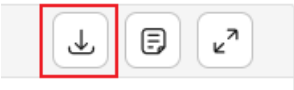
  - If the total size of all logs exceeds 500 MB, the log page may be frozen. In this case, download the logs to view them locally.
  - A log preview link can be accessed by anyone within one hour after it is generated. You can share the link with others.
  - **Ensure that no privacy information is contained 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.

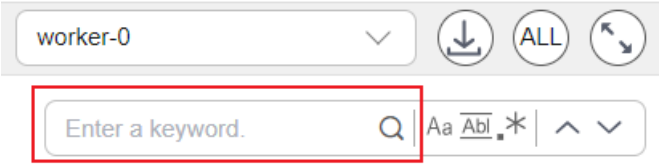
  Figure 8-39 Downloading logs

- 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, as shown in **Figure 8-40**.

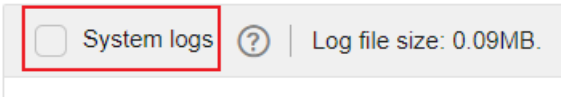
  Figure 8-40 Searching for logs by keyword

  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

  Figure 8-41 System logs

  If **System logs** is selected, system logs and user logs are displayed. If **System logs** is deselected, only user logs are displayed.

8.11.6 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 tasks, 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**.

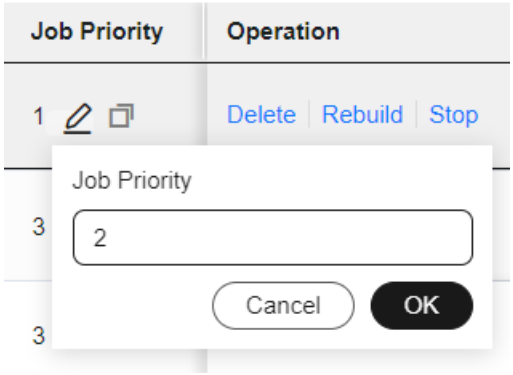
Configuring the Priority

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**.

Figure 8-42 Changing the job priority



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 Huawei Cloud 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.

8.11.7 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 Huawei Cloud 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.

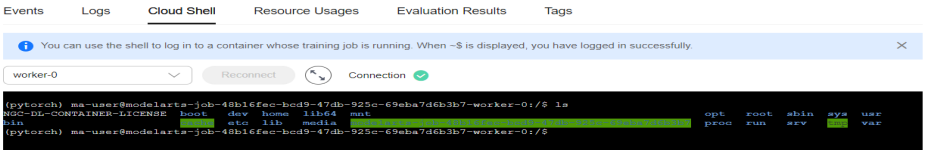
   If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

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.

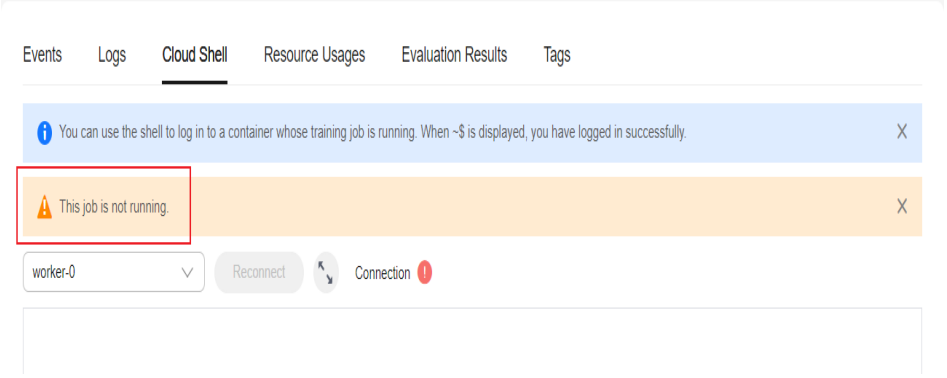
   Verify that the login is successful, as shown in the following figure.

Figure 8-43 Cloud Shell page



If the job is not running or the permission is insufficient, Cloud Shell cannot be used. In this case, locate the fault as prompted.

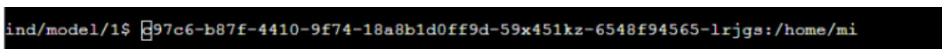
Figure 8-44 Error message



NOTE

If you encounter a path display issue when logging into Cloud Shell, press **Enter** to resolve the problem.

Figure 8-45 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

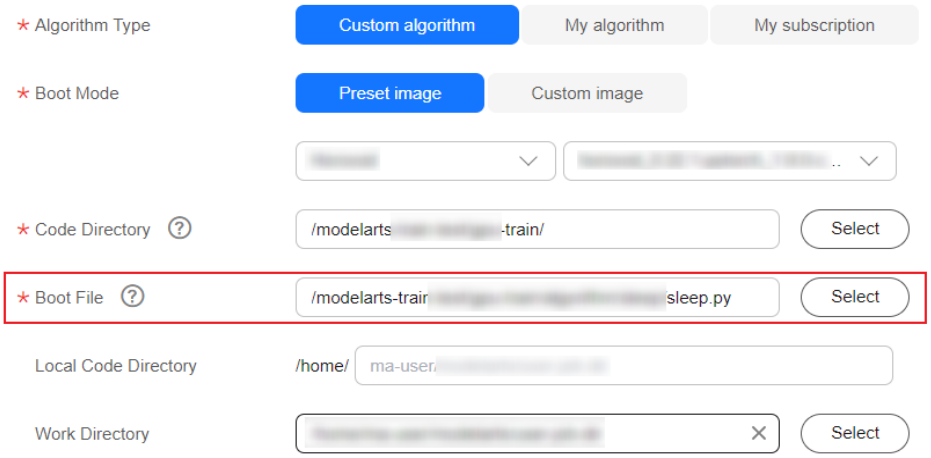
- For training jobs using a preset image

  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')
```

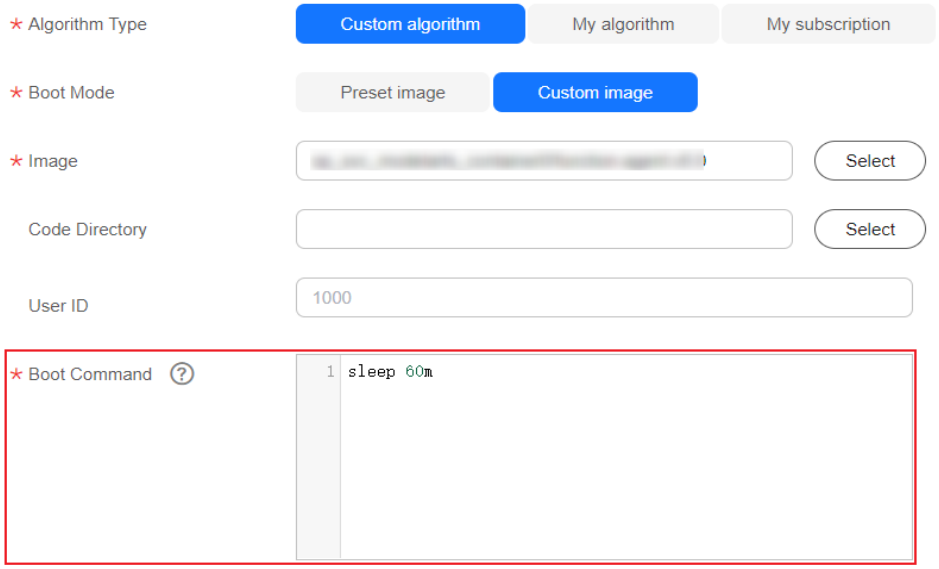
  Figure 8-46 Using a preset image



- 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.

  Figure 8-47 Using a custom image



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 **Model Training** > **Training Jobs**.

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.myhuaweicloud.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

8.11.8 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.

After a training job is stopped, its billing stops on ModelArts.

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 to avoid unnecessary charges.

- 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.

Figure 8-48 Searching for a training job

8.11.9 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

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 8-33, **Table 8-34**, **Table 8-35**, **Table 8-36**, **Table 8-37**, **Table 8-38**, and **Table 8-39** list environment variables preset in a training container.

The environment variable values are examples only.

Table 8-33 Path environment variables

Variable	Description	Example
PATH	Executable file paths	**PATH=/usr/local/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin**
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 8-34 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 compute nodes, 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	Compute nodes 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
${MA_VJ_NAME}-${MA_TASK_NAME}-N.${MA_VJ_NAME}	Communication domain name of a node. For example, the communication domain name of node 0 is **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}**. N indicates the number of compute nodes.	For example, if there are four compute nodes, the environment variables are as follows: **${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-1.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-2.${MA_VJ_NAME}** **${MA_VJ_NAME}-${MA_TASK_NAME}-3.${MA_VJ_NAME}**

Table 8-35 NCCL environment variables

Variable	Description	Example
NCCL_VERSION	NCCL version	**NCCL_VERSION=2.7.8**
NCCL_DEBUG	NCCL log level	**NCCL_DEBUG=INFO**
NCCL_IB_HCA	InfiniBand NIC to use for communication	**NCCL_IB_HCA=^mlx5_bond_0**
NCCL_SOCKET_IFNAME	IP interface to use for communication	**NCCL_SOCKET_IFNAME=bond0,eth0**

Table 8-36 OBS environment variables

Variable	Description	Example
S3_ENDPOINT	OBS endpoint	N/A
S3_VERIFY_SSL	Whether to use SSL to access OBS	**S3_VERIFY_SSL=0**
S3_USE_HTTPS	Whether to use HTTPS to access OBS	**S3_USE_HTTPS=1**

Table 8-37 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.myhuaweicloud.com**
MA_PIP_URL	Address of the pip source	**MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/**
MA_APIGW_ENDPOINT	ModelArts API Gateway address	**MA_APIGW_ENDPOINT=https://modelarts.region.cn-east-3.myhuaweicloud.com**

Table 8-38 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-8a83-000000000000-worker-1**

Table 8-39 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**

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 8-49 Viewing logs

```
NV_LIBCUBLAS_DEV_VERSION=11.3.1.68-1
NV_CUDA_COMPAT_PACKAGE=cuda-compat-11-2
NV_CUDNN_PACKAGE_DEV=libcudnn8-dev=8.1.1.33-1+cuda11.2
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64
NV_LIBNCCL_DEV_PACKAGE=libnccl-dev=2.8.4-1+cuda11.2
MA_ENGINE_VERSION=
MA_NUM_HOSTS=1
VC_WORKER_HOSTS=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0.modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
VK_TASK_INDEX=0
_=/usr/bin/env
MA_SCRIPT_INTERPRETER=
NV_LIBNPP_DEV_PACKAGE=libnpp-dev-11-2=11.2.1.68-1
MA_MAX_BACKOFF=0
HOSTNAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0
MA_IAM_USER_ID=79098163dd814fd986eca7ef0325d086
MA_CURRENT_IP=10.0.0.62
NV_LIBNPP_VERSION=11.2.1.68-1
NV_NVPROF_DEV_PACKAGE=cuda-nvprof-11-2=11.2.67-1
MA_MOUNT_PATH=/home/ma-user/modelarts
NVIDIA_VISIBLE_DEVICES=all
MA_ENGINE_TYPE=
NV_NVPROF_VERSION=11.2.67-1
NV_LIBCUSPARSE_VERSION=11.3.1.68-1
MODELARTS_SCC_SERVICE_PORT=60687
MA_HOME=/home/ma-user
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.247.0.1
NV_LIBCUBLAS_DEV_PACKAGE=libcublas-dev-11-2=11.3.1.68-1
MA_VJ_NAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/
NCCL_VERSION=2.8.4-1
KUBERNETES_PORT=tcp://10.247.0.1:443
PWD=/
NVARCH=x86_64
HOME=/home/ma-user
```

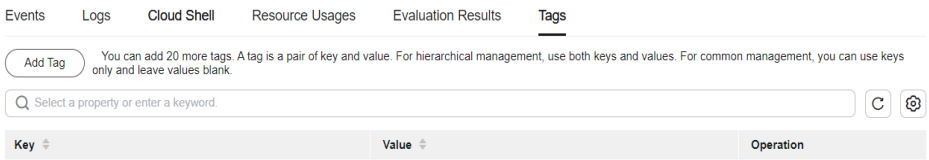
8.11.10 Viewing Training Job Tags

You can add tags to a training job for quick search.

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 **Tags**.

   Tags can be added, modified, and deleted. For details about how to use tags, see **How Does ModelArts Use Tags to Manage Resources by Group?**

Figure 8-50 Viewing training tags



NOTE

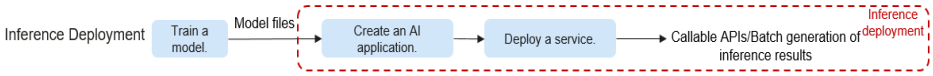
You can add up to 20 tags to a training job.

9 Inference Deployment

9.1 Overview

Developed AI models can be used to create AI applications, which can then be quickly deployed as inference services. These services can be integrated into your IT platform by calling APIs or generate batch results.

Figure 9-1 Introduction to inference

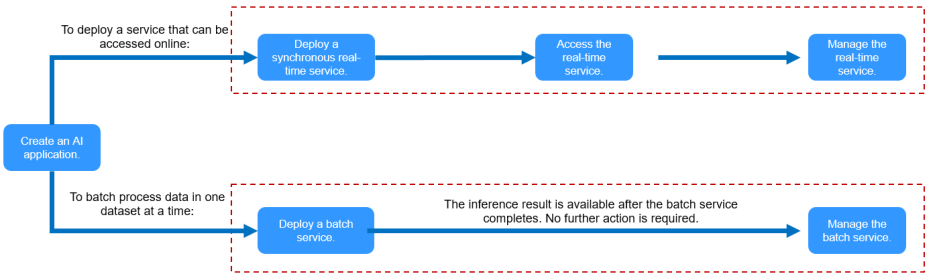


1. Train a model: Models can be trained in ModelArts or your local development environment. A locally developed model must be uploaded to Huawei Cloud OBS.
2. Create an AI application: Import the model file and inference file to the ModelArts model repository and manage them by version. Use these files to build an executable AI application.
3. Deploy a service: Deploy the AI application as a service type based on your needs.
   - **Deploying an AI Application as Real-Time Inference Jobs**

     Deploy an AI application as a web service with real-time UI and monitoring. This service provides you a callable API.
   - **Deploying an AI Application 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 9-2 Different inference scenarios



9.2 Creating an AI Application

9.2.1 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 AI applications using meta models from training jobs, OBS, or container images, and centrally manage all iterated and debugged AI applications.

Constraints

- After deploying a model in an ExeML project, it is automatically added to the AI application list. ExeML-generated AI applications can only be deployed, not downloaded.
- All users can create AI applications and manage AI application versions at no cost.

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 an AI application 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 to create an AI application 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 an AI application for service deployment.

Supported AI Engines for ModelArts Inference

If you import a model from OBS to ModelArts and use it to create an AI application, the following AI engines and versions are supported.

NOTE

- Runtime environments marked with **recommended** are from unified images, which will be used as mainstream base images for inference. Unified images provide comprehensive installation packages. For details, see **Preset Dedicated Images for Inference**.
- Images of the old version will be discontinued. Use unified images instead.
- The base images to be removed are no longer maintained.
- 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 9-1 Supported AI engines, their runtime environments, and default start commands

Engine	Runtime Environment	Note
TensorFlow	python3.6 python2.7 (unavailable soon) tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7 (unavailable soon) tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 (recommended)	• TensorFlow 1.8.0 is used in **python2.7** and **python3.6**. • The model can run on both CPUs and GPUs when using **python3.6**, **python2.7**, or **tf2.1-python3.7**. If the runtime environment has a suffix of **cpu** or **gpu**, the model can only run on CPUs or GPUs respectively. • The default runtime environment is **python2.7**. • Default start command: **sh /home/mind/run.sh**
Spark_MLlib	python2.7 (unavailable soon) python3.6 (unavailable soon)	• Spark_MLlib 2.3.2 is used in **python2.7** and **python3.6**. • The default runtime environment is **python2.7**. • **python2.7** and **python3.6** can only be used to run models on CPUs. • Default start command: **bash /home/work/predict/bin/run.sh**

Engine	Runtime Environment	Note
Scikit_Learn	python2.7 (unavailable soon) python3.6 (unavailable soon)	• Scikit_Learn 0.18.1 is used in **python2.7** and **python3.6**. • The default runtime environment is **python2.7**. • **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 (unavailable soon) python3.6 (unavailable soon)	• XGBoost 0.80 is used in **python2.7** and **python3.6**. • The default runtime environment is **python2.7**. • **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	python2.7 (unavailable soon) python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7 (unavailable soon) pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 (recommended)	• PyTorch 1.0 is used in **python2.7**, **python3.6**, and **python3.7**. • The model can run on both CPUs and GPUs when using **python2.7**, **python3.6**, **python3.7**, **pytorch1.4-python3.7**, or **pytorch1.5-python3.7**. • The default runtime environment is **python2.7**. • Default start command: **sh /home/mind/run.sh**
MindSpore	aarch64 (recommended)	**aarch64** can only be used to run models on Snt3 chips. • Default start command: **sh /home/mind/run.sh**

9.2.2 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 an AI Application**.

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. In the navigation pane, choose **AI Applications**.
2. Click **Create Applications**.
3. Configure parameters.
   a. Enter basic information. For details, see **Table 9-2**.

Table 9-2 Basic information

Parameter	Description
Name	Name of the AI application. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Version	Version of the AI application. The default value is **0.0.1** for the first import. **NOTE** After an AI application is created, you can **create new versions** using different meta models for optimization.
Description	Brief description of the AI application.

   b. Set **Meta Model Source** to **Training job**. For details, see **Table 9-3**.

Figure 9-3 Importing a meta model from a training job

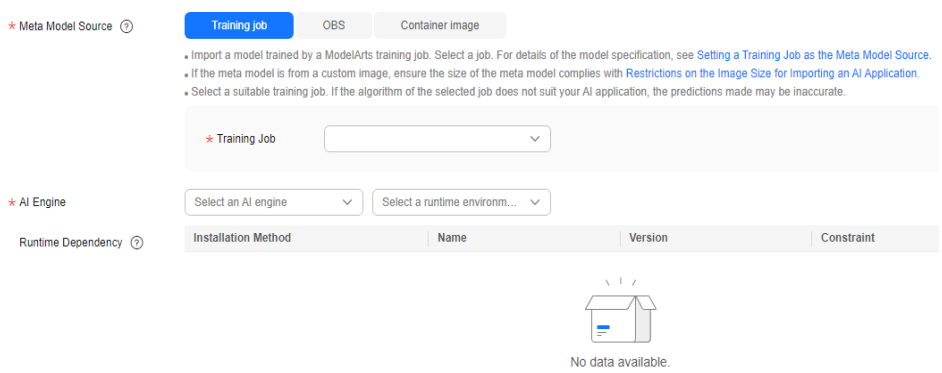


Table 9-3 Meta model source parameters

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 model 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 AI application. You can directly copy the inference code URL for use.
Runtime Dependency	Dependencies that the selected model has on the environment. For example, if you need to install **tensorflow** using **pip**, make sure the version is 1.8.0 or newer.
AI Application Description	AI application descriptions to help other developers better understand and use your application. Click **Add AI Application Description** and set the document name and URL. You can add up to three AI application descriptions.
Deployment Type	Choose the service types for application deployment. The service types you select will be the only options available for deployment. For example, selecting **Real-Time Services** means the AI application can only be deployed as real-time services.

c. Confirm the configurations and click **Create now**.

In the AI application list, you can view the created AI application and its version. When the status changes to **Normal**, the AI application 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 AI application list, click **Deploy** in the **Operation** column of the target AI application. Locate the target version, click **Deploy** and choose a service type selected during AI application creation.

9.2.3 Importing a Meta Model from OBS

Import a model trained using a mainstream AI engine to ModelArts to create an AI application 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 an AI Application**.

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. In the navigation pane, choose **AI Applications**.
2. Click **Create Applications**.
3. Configure parameters.

   a. Enter basic information. For details, see **Table 9-4**.

Table 9-4 Basic information

Parameter	Description
Name	Name of the AI application. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.

Parameter	Description
Version	Version of the AI application. The default value is **0.0.1** for the first import. **NOTE** After an AI application is created, you can **create new versions** using different meta models for optimization.
Description	Brief description of the AI application.

b. Set **Meta Model Source** to **OBS**. For details, see **Table 9-5**.

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 AI application cannot be created.

Figure 9-4 Importing a meta model from OBS

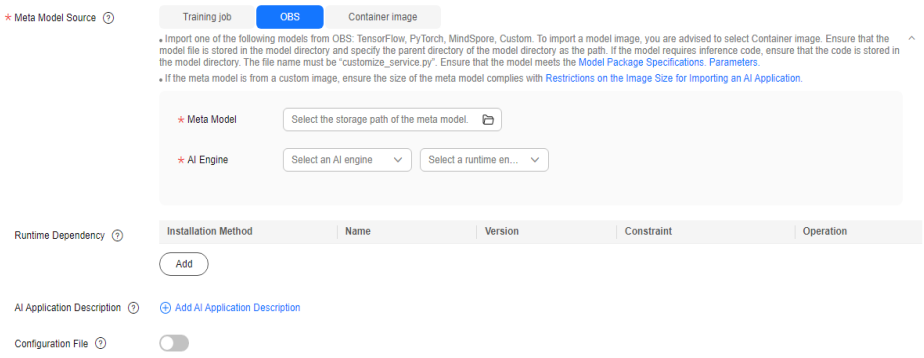


Table 9-5 Meta model source parameters

Parameter	Description
Meta Model	OBS path for storing the meta model. The OBS path cannot contain spaces. Otherwise, the creation of the AI application will fail.
AI Engine	AI engine, which is automatically set according to the model storage path you select, used by the meta model. If you set **AI Engine** to **Custom**, you will need to specify the protocol and port number in **Container API** to start the model. By default, the request protocol is **HTTPS** and the port number is **8080**. You can adjust the protocol and port number as needed.

Parameter	Description
Container API	Set the protocol and port number of the inference API defined by the model. **NOTE** This parameter is displayed only when **AI Engine** is set to **Custom**. By default, the request protocol is **HTTPS** and the port number is **8080**. You can adjust the protocol and port number as needed.

Parameter	Description
Health Check	Specifies health check on a model. This parameter is displayed when **AI Engine** is set to **Custom**. Once you select a non-custom engine and runtime environment, this parameter is displayed if this engine supports health check. Once you select a custom engine, you must select a container image for the engine package. The health check can be set up only if the container image includes a health check API. Otherwise, the AI application 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**: Retain the default setting **HTTP request**. ● **Health Check URL**: Retain the default setting **/health**. ● **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 consecutive health checks during operation, it will enter the alarm state.

Parameter	Description
	NOTE To use a custom engine to create an AI application, ensure that the custom engine complies with the specifications for custom engines. For details, see **Creating an AI Application Using a Custom Engine**. If health check is enabled for an AI application, the associated services will stop three minutes after receiving the stop instruction.
Dynamic loading	You can enable it for quick model 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. For example, if you need to install **tensorflow** using **pip**, make sure the version is 1.8.0 or newer.
AI Application Description	AI application descriptions to help other developers better understand and use your application. Click **Add AI Application 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 application deployment. The service types you select will be the only options available for deployment. For example, selecting **Real-Time Services** means the AI application 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. For details, see **Table 9-1**. 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 an AI application is being published.

Parameter	Description
API Configuration	You can enable it to edit RESTful APIs to define the AI application input and output formats. 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. Check the information and click **Create now**.

In the AI application list, you can view the created AI application and its version. When the status changes to **Normal**, the AI application 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 AI application list, click **Deploy** in the **Operation** column of the target AI application. Locate the target version, click **Deploy** and choose a service type selected during AI application creation.

9.2.4 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**.
- The configuration file for your trained model must be uploaded to OBS. The file must comply with **Specifications for Editing a Model Configuration File**.
- The meta model size must comply with **Restrictions on the Size of an Image for Importing an AI Application**.

Prerequisites

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

Procedure

1. Log in to the ModelArts console. In the navigation pane, choose **AI Applications**.
2. Click **Create Applications**.
3. Configure parameters.
   a. Enter basic information. For details, see **Table 9-6**.

Table 9-6 Basic information

Parameter	Description
Name	Name of the AI application. The value can contain 1 to 64 visible characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.
Version	Version of the AI application. The default value is **0.0.1** for the first import. **NOTE** After an AI application is created, you can **create new versions** using different meta models for optimization.
Description	Brief description of the AI application.

b. Select the meta model source and configure related parameters. Set **Meta Model Source** to **Container image**. For details, see **Table 9-7**.

Figure 9-5 Importing a meta model from a container image

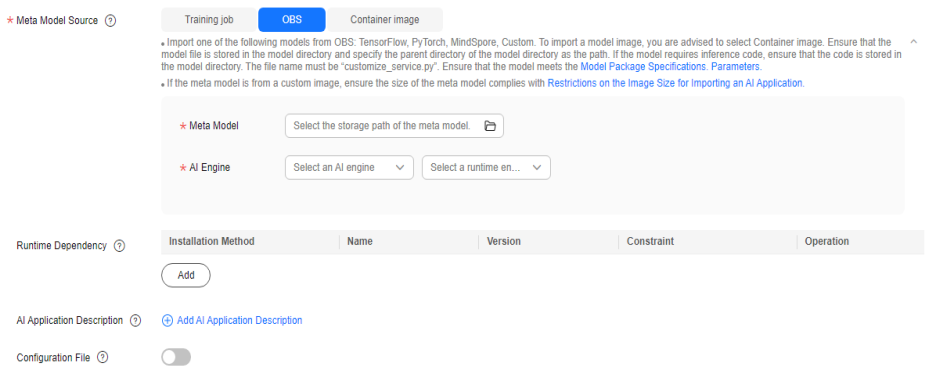


Table 9-7 Meta model source parameters

Parameter	Description
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.

Parameter	Description
Container API	Set the protocol and port number of the inference API defined by the model. **NOTE** The default request protocol and port number provided by ModelArts are HTTP and 8080, respectively. Set them based on the actual custom image.
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, AI applications 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, AI applications 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, AI applications will fail to be created.

Parameter	Description
Health Check	Specifies health check on an AI application. This parameter is configurable only when a health check API is configured in the custom image. Otherwise, creating the AI application 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 an AI application, the associated services will stop three minutes after receiving the stop instruction.
AI Application Description	AI application descriptions to help other developers better understand and use your application. Click **Add AI Application Description** and set the document name and URL. You can add up to three descriptions.
Deployment Type	Choose the service types for application deployment. The service types you select will be the only options available for deployment. For example, selecting **Real-Time Services** means the AI application 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 an AI application is being published.
API Configuration	You can enable it to edit RESTful APIs to define the AI application input and output formats. 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. Check the information and click **Create now**.

In the AI application list, you can view the created AI application and its version. When the status changes to **Normal**, the AI application 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 AI application list, click **Deploy** in the **Operation** column of the target AI application. Locate the target version, click **Deploy** and choose a service type selected during AI application creation.

9.3 Specifications for Creating an AI Application

9.3.1 Model Package Structure

When creating an AI application, 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 provides samples and sample code for multiple engines. You can edit your configuration files and inference code by referring to **ModelArts Samples**. ModelArts also provides custom script examples of common AI engines. For details, see **Examples of Custom Scripts**.

If you encounter any problem when importing a meta model, **contact Huawei Cloud technical support**.

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 referenced in model inference code
│   │   ├── saved_model.pb (Mandatory) Protocol buffer file, which contains the graph description of the model
│   │   ├── variables Name of a fixed sub-directory, which contains the weight and deviation rate of
```



```
the model. It is mandatory for the main file of a *.pb model.
|   |   |   ├── variables.index Mandatory
|   |   |   ├── variables.data-00000-of-00001 Mandatory
|   |   ├──config.json (Mandatory) Model configuration file. The file name is fixed to config.json. Only one model configuration file is allowed.
|   |   ├──customize_service.py (Mandatory) Model inference code. The file name is fixed to customize_service.py. 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 the PyTorch-based model package

  When publishing the model, you only need to specify the **resnet** directory.

```
OBS bucket or directory name
├── resnet
|   ├── 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 referenced in model inference code
|   |   ├── resnet50.pth (Mandatory) PyTorch model file, which contains variable and weight information and is saved as state_dict
|   |   ├──config.json (Mandatory) Model configuration file. The file name is fixed to config.json. Only one model configuration file is allowed.
|   |   ├──customize_service.py (Mandatory) Model inference code. The file name is fixed to customize_service.py. 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.

9.3.2 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 9-8 describes the parameters.

Table 9-8 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**.

Parameter	Mandatory	Type	Description
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**, an AI application 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. Python 2.7 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 *Supported AI Engines for ModelArts Inference*.
metrics	No	Object	Model precision information, including the F1 score, recall, precision, and accuracy. For details about the **metrics** object structure, see **Table 9-9**. The result is displayed in the model precision area on the AI application details page.

Parameter	Mandatory	Type	Description
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 9-10**. 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 9-13** 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. **NOTE** The **dependencies** parameter accepts multiple dependency arrays in a list format. It applies to scenarios where the default installation packages have dependency relationships. The top packages are installed first. The wheel package can be used for dependency installation, and it must be stored in the same directory as the model file. For details, see **How Do I Edit the Installation Package Dependency Parameters in a Model Configuration File When Importing a Model?**
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 9-15**.

Table 9-9 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 9-10 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 9-11**.
response	No	Object	Response body. For details, see **Table 9-12**.

Table 9-11 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 9-12 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 9-13 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 9-14**.

Table 9-14 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 9-15 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": "applicaton/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": {
                    "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  -7.1861157
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": "applicaton/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": "applicaton/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": "applicaton/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"
                }
            ]
        }
    ]
}
```

9.3.3 Specifications for Writing a Model Inference Code File

This section describes the general method of editing model inference code in ModelArts. For details about the custom script examples (including inference code examples) of mainstream AI engines, see **Examples of Custom Scripts**. 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 9-16**. The ModelArts environment has already configured the necessary Python packages for import statements, so you do not need to install them separately.

Table 9-16 Parent class and import statement of each model type

Model Type	Parent Class	Import Statement
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
MindSpore	SingleNodeService	from model_service.model_service import SingleNodeService

2. The following methods can be overridden.

Table 9-17 Methods to be overridden

Method	Description
__init__(self, model_name, model_path)	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.
__init__(self, model_path)	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**).
_preprocess(self, data)	Preprocess method, which is called before an inference request and converts API request data into the model's expected input format.
_inference(self, data)	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.
_postprocess(self, data)	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 an AI application 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. For more TensorFlow inference code examples, see **Tensorflow** and **Tensorflow2.1**.

- Inference code

```
from PIL import Image
import numpy as np
from model_service.tfserving_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 base inference 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.tfserving_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.tfserving_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)
```

9.3.4 Specifications for Using a Custom Engine to Create an AI Model

When using a custom engine to create an AI application, 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 an AI application 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. The service available to port 8080 in the container is automatically registered with APIG. You can access the service through the APIG URL.

Specifications for Using a Custom Engine to Create an AI Application

To use a custom engine to create an AI application, 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
    ```


  - The service must be HTTPS enabled, and it is available on port 8080. For details, see the **HTTPS example**.
  - (Optional) On port 8080, 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 an AI application.

1. Create an AI application and view its details.

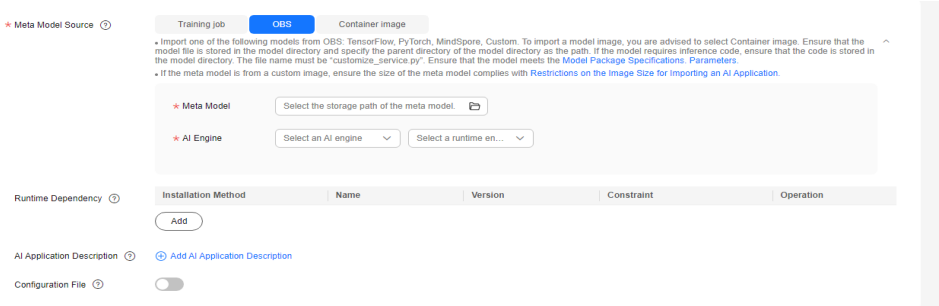
   Log in to the ModelArts console, choose **AI Applications** from the navigation pane and click **Create Applications**. Configure the following parameters:

   - **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 application status changes to **Normal**.

   Figure 9-6 Creating an AI application

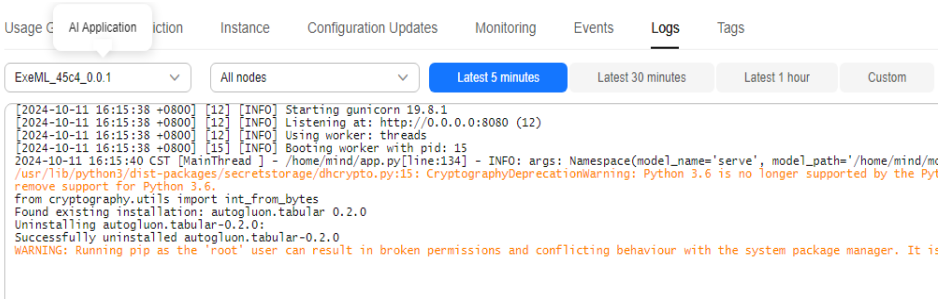
   Click the AI application name. On the displayed page, view details about the AI application.

2. Deploy the AI application as a service and view service details.

   On the AI application details page, choose **Deploy** > **Real-Time Services** in the upper right corner. On the **Deploy** page, select a proper compute node 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.

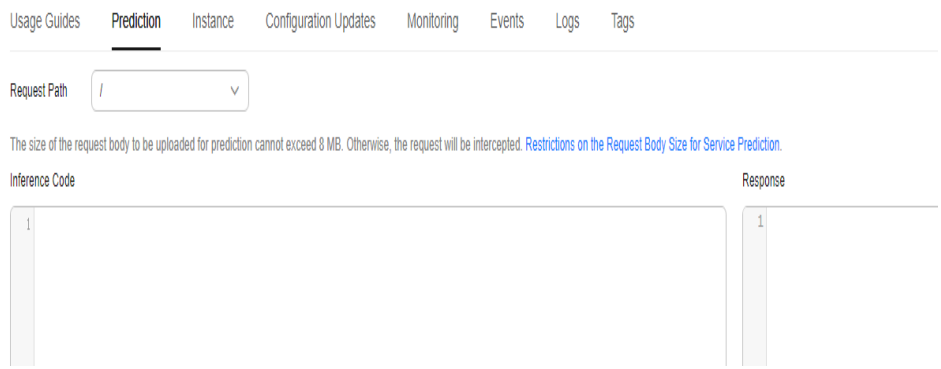
Figure 9-7 Service logs



3. Use the service for prediction.

   On the service details page, click the **Prediction** tab to use the service for prediction.

Figure 9-8 Service prediction



9.3.5 Examples of Custom Scripts

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.test.images,
        y_: mnist.test.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 9-16**. 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.tfserving_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 9-16**.

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_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 9-16**.

```
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
```

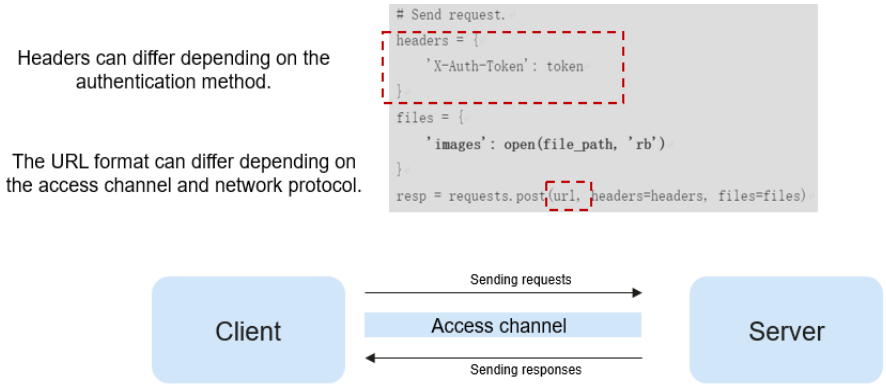
9.4 Deploying an AI Application as Real-Time Inference Jobs

9.4.1 Deploying and Using Real-Time Inference

After creating an AI application, 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 9-9 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.
- **AK/SK-based authentication**: AK/SK is used to sign requests and the signature is then added to the request for authentication. AK/SK-based authentication supports API requests with a body not larger than 12 MB. For API requests with a larger body, token-based authentication is recommended.
- **App authentication**: Add a parameter to the request header to complete the authentication. The authentication is simple and permanently valid.

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 WebSocket**: WebSocket simplifies data exchange between the client and server and allows the server to proactively push data to the client. In the WebSocket API, if the initial handshake between the client and the server is successful, a persistent connection will be established between them and data can be transferred bidirectionally.
- **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.

9.4.2 Deploying a Model as a Real-Time Service

After creating an AI application, you can deploy it as a real-time service and call the service for prediction.

Constraints

A user can create up to 20 real-time services.

Prerequisites

- An AI application in the **Normal** state is available in ModelArts.
- The account is not in arrears to ensure available resources for service running.

Procedure

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Model 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 9-18**.

Table 9-18 Basic parameters

Parameter	Description
Name	Name of a real-time service.
Auto Stop	Time for your service to automatically stop running. This helps you avoid unnecessary billing. If you disable this feature, your real-time service will continue running and you will be billed accordingly. By default, this feature is enabled and set to stop the service 1 hour after it starts. The options are **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later**, and **Custom**. If you select **Custom**, you can enter any integer from 1 to 24.
Description	Brief description for a real-time service.

   b. Enter key information including the resource pool and AI application configurations. For details, see **Table 9-19**.

Table 9-19 Parameters

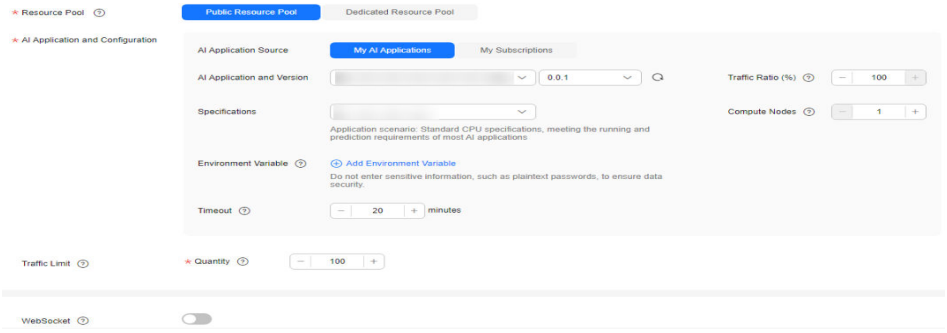
Parameter	Sub-Parameter	Description
Resource Pool	Public Resource Pool	CPU/GPU resource pools are available for you to select. The pricing for resource pools varies depending on their flavors. For details, see **Product Pricing Details**. Public resource pools only support the pay-per-use billing mode.
	Dedicated Resource Pool	Select a dedicated resource pool flavor. The physical pools with logical subpools created are not supported temporarily.
AI Application and Configuration	AI Application Source	Choose **My AI Applications** or **My Subscriptions** as needed.
	AI Application and Version	Select the AI application and version that are in the **Normal** status.
	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 an AI application, 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%**.
	Specifications	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, use a dedicated resource pool or contact the administrator to create a public 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.
	Compute Nodes	Number of instances for the current AI application 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.

Parameter	Sub-Parameter	Description
	Environment Variable	Set environment variables and inject them to the pod. To ensure data security, do not enter sensitive information, such as plaintext passwords, 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.
	Add AI Application Version and Configuration	If the selected AI application has multiple versions, you can add multiple versions and configure a traffic ratio. You can use gray release to smoothly upgrade the AI application 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. The storage volume type can be OBS parallel file system or SFS Turbo. **SFS Turbo** • **File System Name**: Select the target SFS Turbo file system. A cross-region SFS Turbo file system cannot be selected. • **Mount Path**: Enter the mount path of the container, for example, **/sfs-turbo-mount/**. 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 **Interconnecting a VPC with a ModelArts Network** . - 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, **/obs-mount/** and **/obs-mount/tmp/**. • 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.
Traffic Limit	N/A	Maximum number of times a service can be accessed within a second. You can configure this parameter as needed.

Parameter	Sub-Parameter	Description
WebSocket	N/A	Whether to deploy a real-time service as a WebSocket service. For details about WebSocket real-time services, see **Full-Process Development of WebSocket Real-Time Services**. **NOTE** • This feature is supported only if the AI application is WebSocket-compliant and comes from a container image. • After this feature is enabled, **Traffic Limit** and **Data Collection** cannot be set. • This parameter cannot be changed after the service is deployed.
Application Authentication	Application	This feature is disabled by default. To enable this feature, see **Accessing a Real-Time Service Through App Authentication** for details and configure parameters as required.

Figure 9-10 Setting AI application information



c. (Optional) Configure advanced settings.

Table 9-20 Advanced settings

Parameter	Description
Tags	ModelArts can work with Tag Management Service (TMS). When creating resource-consuming tasks in ModelArts, for example, training jobs, configure tags for these tasks so that ModelArts can use tags to manage resources by group. For details about how to use tags, see **How Does ModelArts Use Tags to Manage Resources by Group?** **NOTE** You can select a predefined TMS tag from the tag drop-down list or customize a tag. Predefined tags are available to all service resources that support tags. Custom tags are available only to the service resources of the user who has created the tags.

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 an AI application is deployed as a real-time service, 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 AI application – either by using a JSON text or a file.

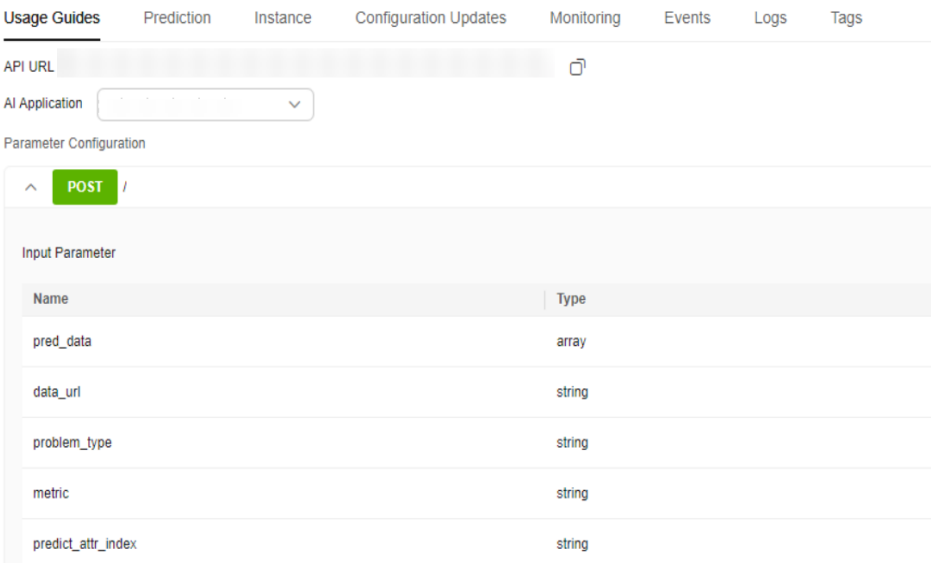
- **JSON Text Prediction**: If your AI application uses JSON text as input, you can simply paste the code into the **Prediction** tab to test the service.
- **File Prediction**: If your AI application uses files as input, you can add images, audios, or videos into the **Prediction** tab to test the service.

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**, **Accessing a Real-Time Service Through AK/SK-based Authentication**, or **Accessing a Real-Time Service Through App 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.

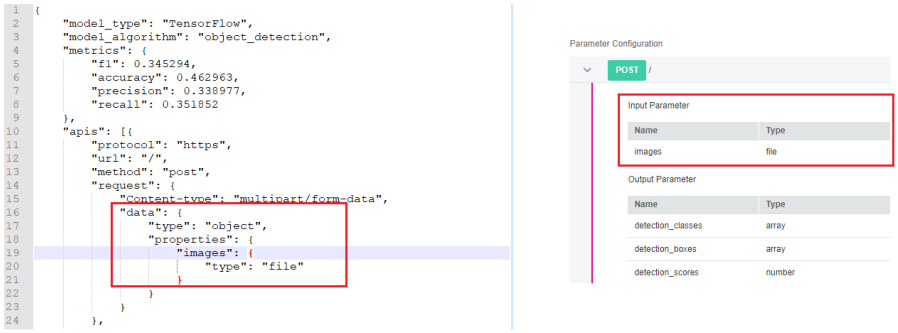
Figure 9-11 Usage Guides



The input parameters in the **Usage Guides** tab vary depending on the AI application source that you select.

- If your meta model comes from ExeML or a built-in algorithm, the input and output parameters are defined by ModelArts. For details, see the **Usage Guides** tab. In the **Prediction** tab, enter the corresponding JSON text or file for service testing.
- 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.

Figure 9-12 Mapping between the configuration file and Usage Guides

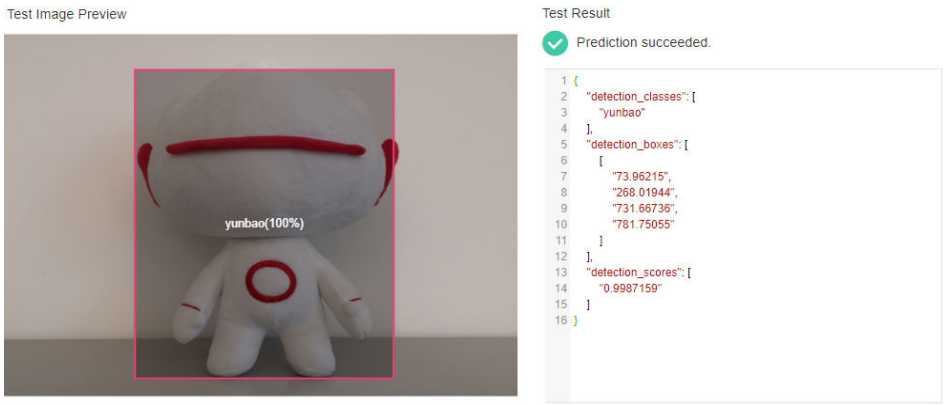


The prediction methods for different input requests are as follows:

- **JSON Text Prediction**
  a. Log in to the ModelArts console and choose **Model 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 **Model 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. In **Figure 9-13**, the label, position coordinates, and confidence score are displayed.

Figure 9-13 Image prediction



Using Cloud Shell to Debug a Real-Time Service Instance Container

You can use Cloud Shell provided by the ModelArts console to log in to the instance container of a running real-time service.

Constraints:

- Cloud Shell can only access a container when the associated real-time service is deployed within a dedicated resource pool
- Cloud Shell can only access a container when the associated real-time service is running.

Step 1 Log in to the ModelArts console. In the navigation pane, choose **Model Deployment** > **Real-Time Services**.

Step 2 On the real-time service list page, click the name or ID of the target service.

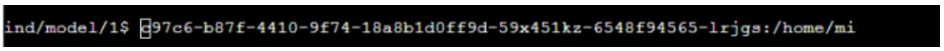
Step 3 Click the **Cloud Shell** tab and select the target AI application version and compute node. When the connection status changes to , you have logged in to the instance container.

If the server disconnects due to an error or remains idle for 10 minutes, you can select **Reconnect** to regain access to the container instance.

NOTE

If you encounter a path display issue when logging in to Cloud Shell, press **Enter** to resolve the problem.

Figure 9-14 Path display issue



----End

9.4.3 Authentication Methods for Accessing Real-time Services

9.4.3.1 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**.
- **Method 4: Use Java 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.

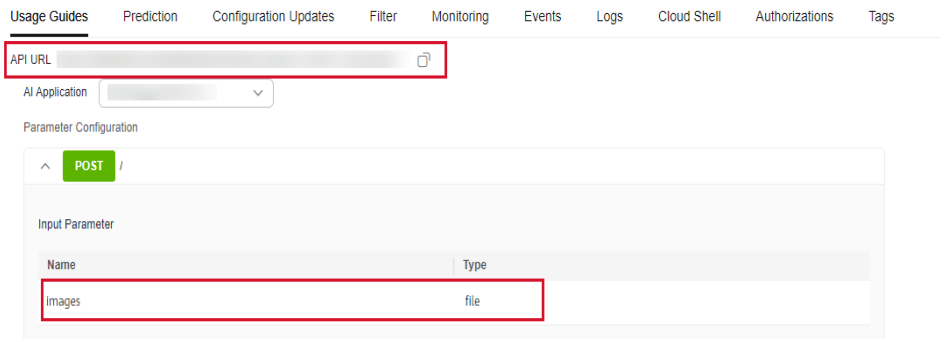
Prerequisites

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**.

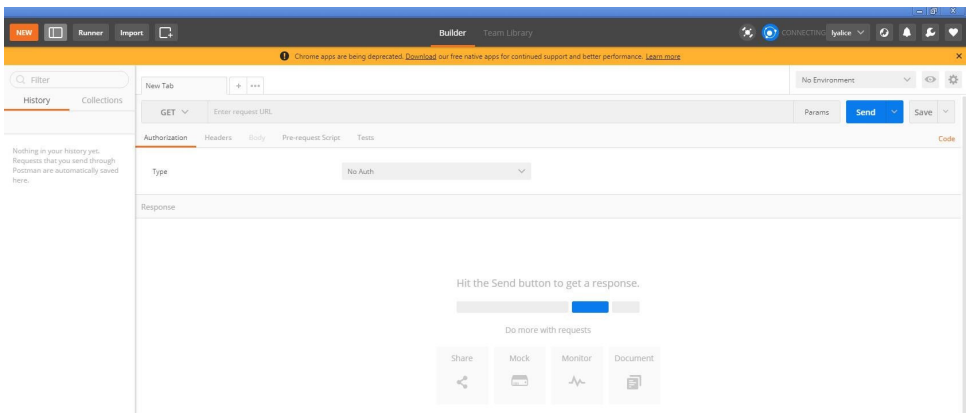
  Figure 9-15 Obtaining the API URL and file prediction input parameters of a real-time service



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. **Figure 9-16** shows the Postman interface.

   Figure 9-16 Postman interface

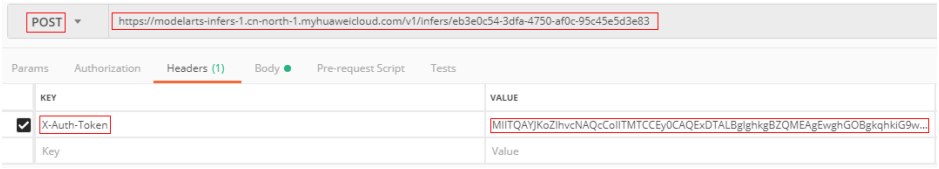


3. Set parameters on Postman. The following uses image classification as an example.
   - Select a POST task and copy the API URL to the POST text box. On the **Headers** tab page, set **Key** to **X-Auth-Token** and **Value** to the user token.

     NOTE

     You can also use the AK and SK to encrypt API calling requests. For details, see **Overview of Session Authentication**.

     Figure 9-17 Parameter settings

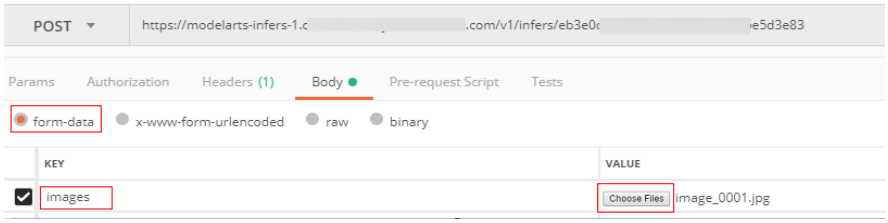
     

   - On the **Body** tab page, file input and text input are available.

     - **File input**

       Select **form-data**. Set **KEY** to the input parameter of the AI application, 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). See **Figure 9-18**.

       Figure 9-18 Setting parameters on the **Body** tab page

     - **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 the inference result is returned after API calling, 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, such as **sepal_length** and **sepal_width** in this example are determined by the AI application.

4. After setting the parameters, click **send** to send the request. The result will be displayed in **Response**.
   - Inference result using file input: **Figure 9-19** shows an example. The field values in the return result vary with the AI application.
   - Inference result using text input: **Figure 9-20** shows an example. 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 AI application, for example, **sepal_length** and **predictresult** in this example.

Figure 9-19 File inference result

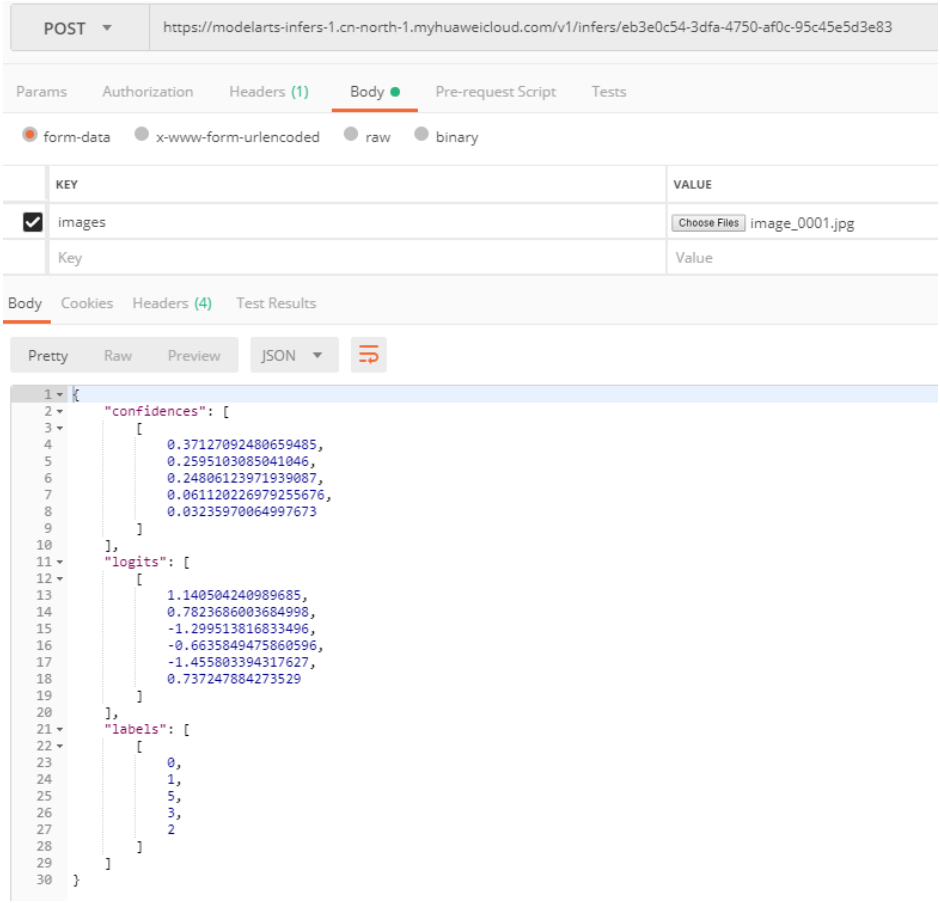
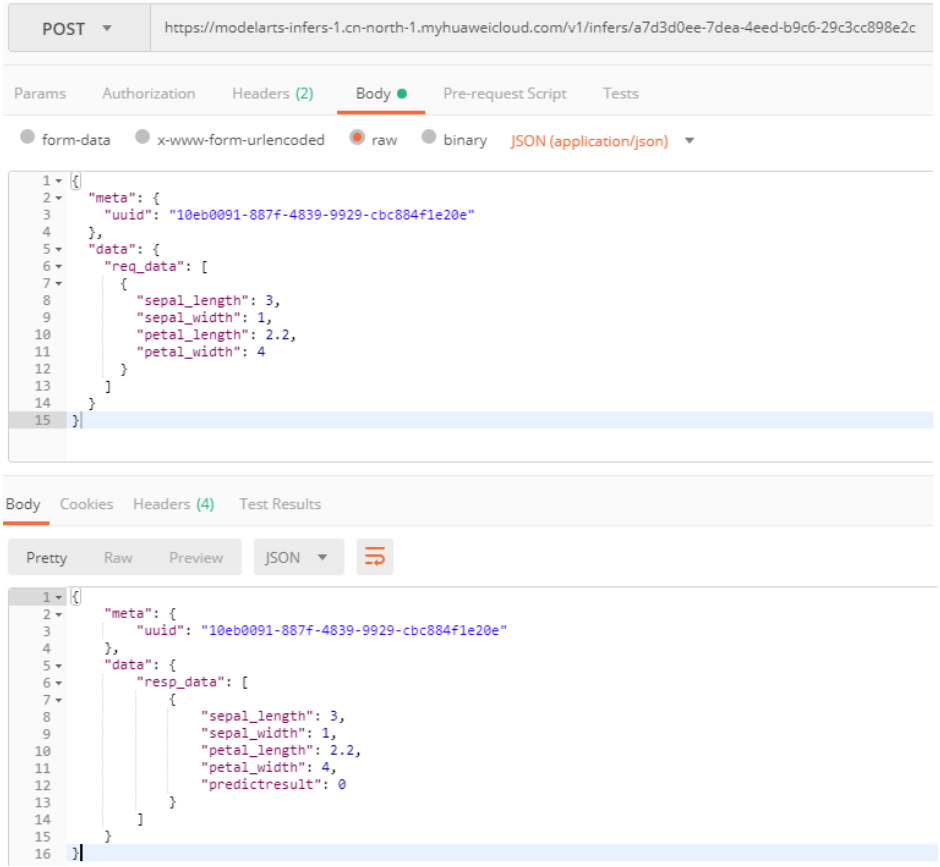


Figure 9-20 Text inference result



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. For details, see **Integrating the Python SDK for API request signing**.
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. The input parameter **images** obtained in **Prerequisites** is an example.
   - **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

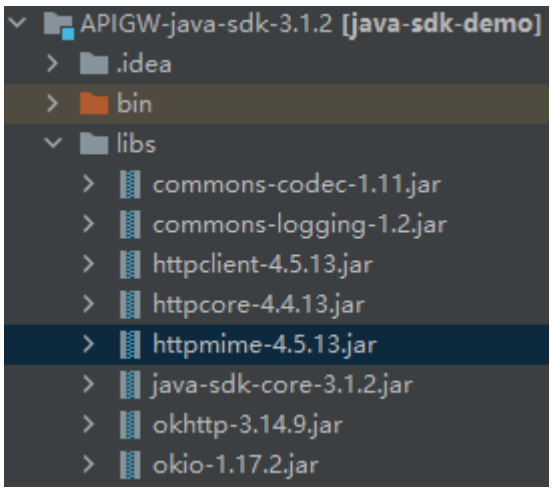
parameter of the string type. The input parameter **images** obtained in **Prerequisites** is an example. The value of **base64_data** in **body** is of the string type.

Method 4: Use Java to Send an Inference Request

1. Download the Java SDK and configure it in the development tool. For details, see **Integrating the Java SDK for API request signing**.
2. (Optional) If the input of the inference request is in the file format, the Java project depends on the httpmime module.
   a. Add **httpmime-x.x.x.jar** to the **libs** folder. **Figure 9-21** 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 9-21 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 MyTokenFile {

    public static void main(String[] args) {
        // Config url, token and filePath
        String url = "URL of the real-time service";
        String token = "User token";
        String filePath = "Local path to the inference file";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader("X-Auth-Token", token);

            // Add a body if you have specified the PUT or POST method. 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. The file **images** obtained in **Prerequisites** is used as an example.

- **Text input (JSON)**

  The following is an example of the 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 MyTokenTest {

    public static void main(String[] args) {
        // Config url, token and body
        String url = "URL of the real-time service";
        String token = "User token";
```

```
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            httpPost.setHeader("X-Auth-Token", token);

            // 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.

9.4.3.2 Accessing a Real-Time Service Through AK/SK-based Authentication

If a real-time service is in the **Running** state, it has been deployed. This service provides a standard, callable RESTful API. You can call the API using AK/SK-based authentication.

When AK/SK-based authentication is used, you can use the APIG SDK or ModelArts SDK to access APIs. For details, see **Overview of Session Authentication**. This section describes how to use the APIG SDK to access a real-time service. The process is as follows:

1. **Obtaining an AK/SK Pair**
2. **Obtaining Information About a Real-Time Service**
3. Send an inference request.
   - **Method 1: Use Python to Send an Inference Request**
   - **Method 2: Use Java to Send an Inference Request**

NOTE

1. AK/SK-based authentication supports API requests with a body not larger than 12 MB. For API requests with a larger body, use token-based authentication.
2. The local time on the client must be synchronized with the clock server to avoid a large offset in the value of the **X-Sdk-Date** request header. API Gateway checks the time format and compares the time with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the 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.

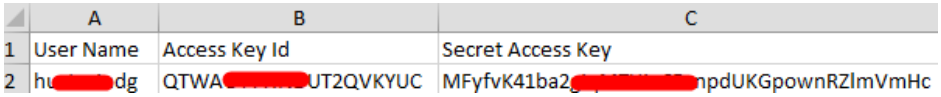
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Obtaining an AK/SK Pair

If an AK/SK pair is already available, skip this step. Find the downloaded AK/SK file, which is usually named **credentials.csv**.

The file contains the username, AK, and SK.

Figure 9-22 credential.csv



	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

To generate an AK/SK pair, follow these steps:

1. Sign up and log in to the console.
2. Click the username and choose **My Credentials** from the drop-down list.
3. On the **My Credentials** page, choose **Access Keys** in the navigation pane.
4. Click **Create Access Key**.
5. Complete the identity authentication, download the access key, and keep it secure.

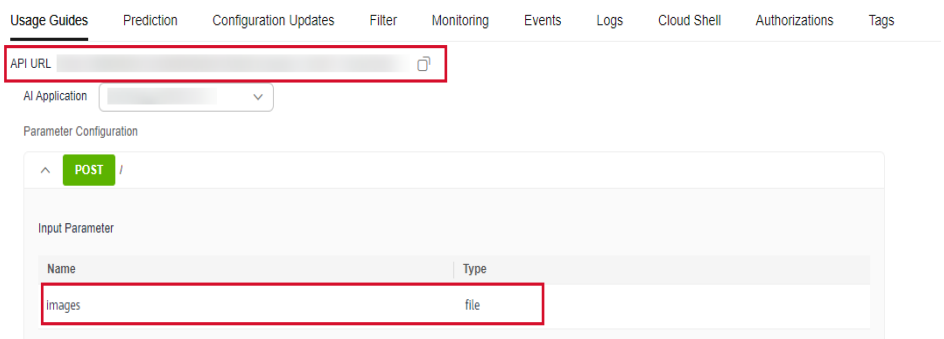
Obtaining Information About a Real-Time Service

To call an API, you will need the URL and input parameters of the real-time service. Follow these steps to obtain this information:

1. Log in to the ModelArts console. In the navigation pane, choose **Model Deployment** > **Real-Time Services**.
2. Click the name of the target service to access its details page.
3. Obtain the URL and input parameters of the service.

   The API URL is the service URL. If **apis** defines a path in the model configuration file, append the user-defined path to the URL, for example, *{URL of the real-time service}*/**predictions/poetry**.

   Figure 9-23 Obtaining the API URL and file prediction input parameters of a real-time service



Method 1: Use Python to Send an Inference Request

1. Download the Python SDK and configure it in the development tool. For details, see **Integrating the Python SDK for API request signing**.
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.
    url = "URL of the real-time service"
    # 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/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    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 = ak
    sig.Secret = sk
    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)
```

file_path is the local path to the inference file. 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**).

Request body format of **files**: files = {"Request parameter": ("Load path", File content, "File type")}. For details about parameters of **files**, see **Table 9-21**.

Table 9-21 Parameters of **files**

Parameter	Mandatory	Description
Request parameter	Yes	Parameter name of the real-time service.
File path	No	Path for storing the file.

Parameter	Mandatory	Description
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.
    url = "URL of the real-time service"
    # 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/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    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 AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    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

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. If a file is input, convert the file into text using Base64.

   - **File input**

     The following is an example request body (JSON) for reading the local inference file and performing Base64 encoding.

```
package com.apig.sdk.demo;
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.commons.codec.binary.Base64;
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;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
public class MyAkSkTest2 {
    public static void main(String[] args) {
        String url = "URL of the real-time service";
       // Hard-coded 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/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        String filePath = "Local path to the inference file";
        try {
            // Create request
            Request request = new Request();
            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);
            // 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);
            // build your json body
            String body = "{\"image\":\"" + getBase64FromFile(filePath) + "\"}";
            // 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 = HttpClients.createDefault().execute(signedRequest);
            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
```



```
            e.printStackTrace();
        }
    }
    /**
     * Convert the file into a byte array and Base64 encode it
     * @return
     */
    private static String getBase64FromFile(String filePath) {
        // Convert the file into a byte array
        InputStream in = null;
        byte[] data = null;
        try {
            in = new FileInputStream(filePath);
            data = new byte[in.available()];
            in.read(data);
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Base64 encode
        return new String(Base64.encodeBase64(data));
    }
}
```

CAUTION

If using Base64 encoding, you need to add a decoding step to your model inference code to handle the request body.

- **Text input (JSON)**

```
// 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) {
        String url = "URL of the real-time service";
       // Hard-coded 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/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");

        try {
            // Create request
            Request request = new Request();

            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);

            // 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.
        String body = "{}";
        request.setBody(body);

        // Sign the request.
        HttpRequestBase signedRequest = Client.sign(request);

        // Send request.
        CloseableHttpResponse response = HttpClients.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.

9.4.3.3 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

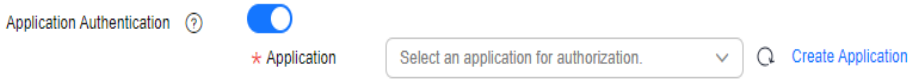
- An AI application in the **Normal** state is available in ModelArts.
- The account is not in arrears to ensure available resources for service running.
- The local path to the inference file has been obtained. The path can be an absolute path (for example, **D:/test.png** for Windows and **/opt/data/test.png** 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 **Model 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 9-24 Enabling application authentication

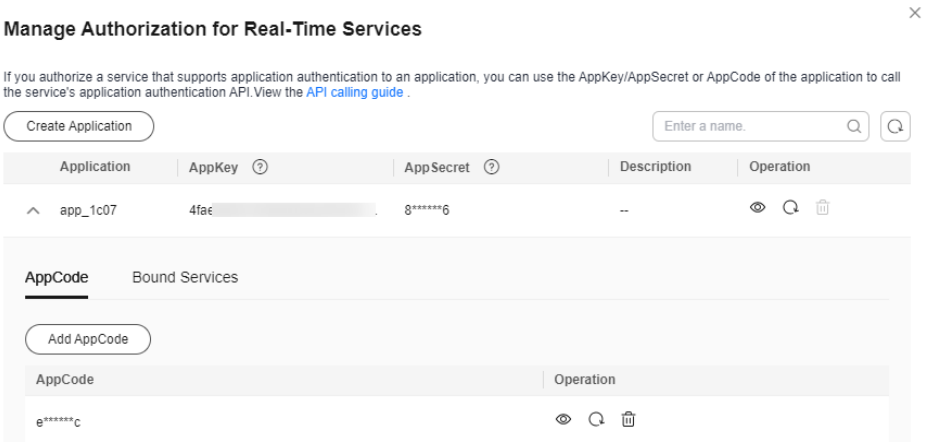
     

3. Select an application for authorization from the drop-down list. If no application is available, follow these steps to create one:
   - Click **Create Application**, enter the application name and description, and click **OK**. By default, the application name is prefixed with **app_**. You can change this name if needed.
   - On the **Model 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 **Model 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 or AppCode.

Figure 9-25 Managing authorization for real-time services



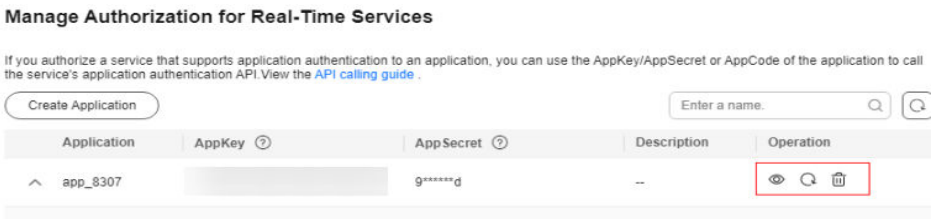
- **Creating an application**

  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 9-26 Query Plaintext, Reset, or Delete

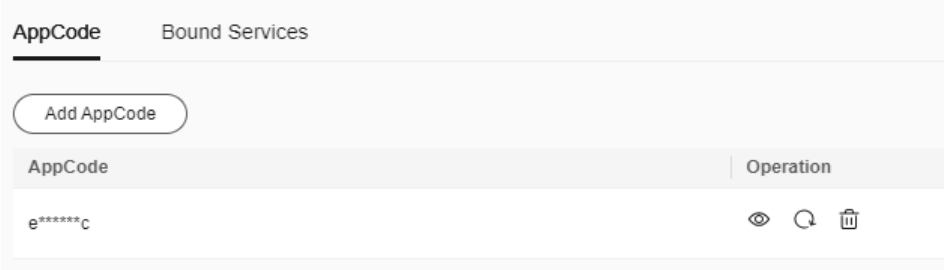
- **Unbinding a service**

  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 or AppCode**

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. Click next to the application name to show the drop-down list. Click **Add AppCode** to automatically generate an AppCode. Then, click in the **Operation** column to view the complete AppCode.

Figure 9-27 Adding the AppCode



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.

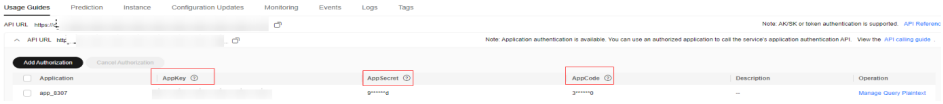
You can obtain the API URL (**url** of the real-time service), AppKey/AppSecret (**app_key** and **app_secret**), and AppCode (**app_code**) from the **Usage Guides** tab

on the service details page (see **Figure 9-28**). Use the API URL for application authentication in the second line of the figure.

Modify the API URL in the following scenarios:

- If **apis** defines a path in the model configuration file, append the user-defined path to the URL, for example, *{URL of the real-time service}*/**predictions/poetry**.
- If an SD WebUI inference service is deployed, add a slash (/) to the end of the calling address, for example, **https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42/**.

Figure 9-28 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 9-22**.

Table 9-22 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 = HttpClients.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)
```

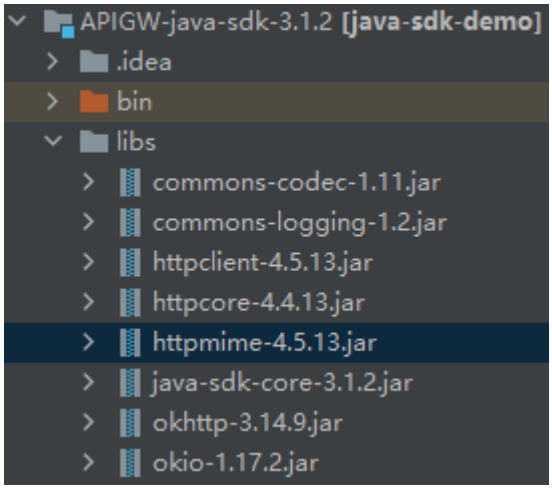
The **body** name is determined by the input parameter of the real-time service. The parameter name must be the same as that of the input parameter of the string type. **image** is used as an example. The value of **base64_data** in **body** is of the string type.

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 9-29** 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 9-29 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.

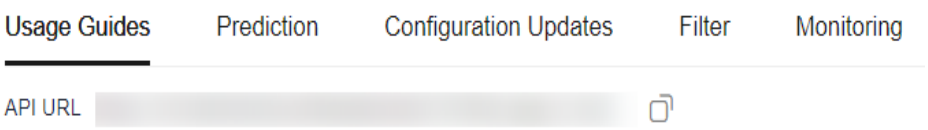
9.4.4 Accessing a Real-Time Service Through Different Channels

9.4.4.1 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 9-30 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 AK/SK-based Authentication**
- **Accessing a Real-Time Service Through App Authentication**

9.4.4.2 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.

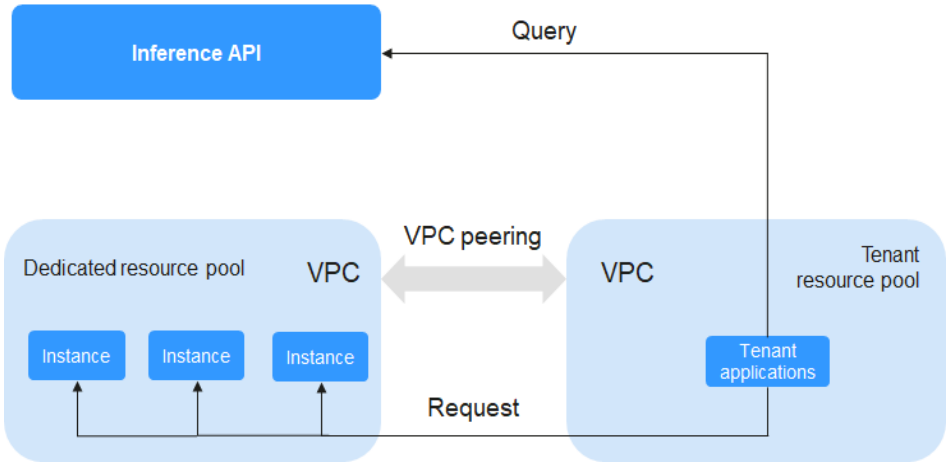
In high-speed access through VPC peering, your service requests are directly sent to instances through VPC peering but not through the inference platform. This accelerates service access.

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 9-31 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

- To use a new-version dedicated resource pool to deploy services, see **About ModelArts Standard Resource Pools**.
- 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 9-32 Obtaining the network configuration

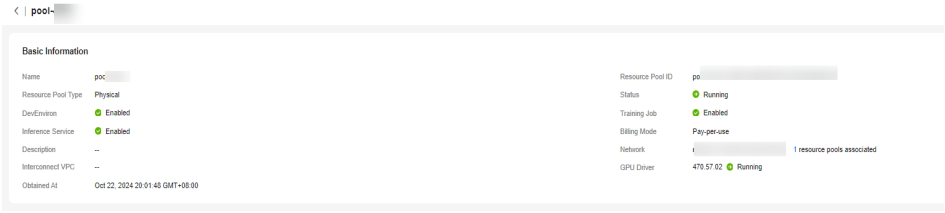


Figure 9-33 Interconnecting the VPC

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 9-34 Selecting a VPC when purchasing an ECS

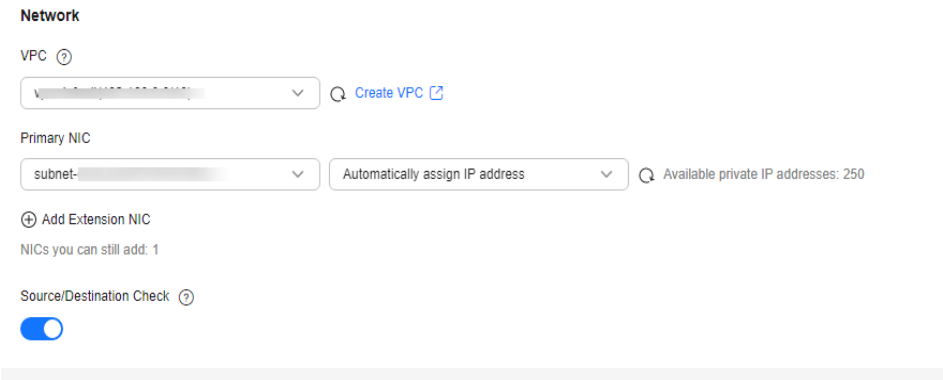
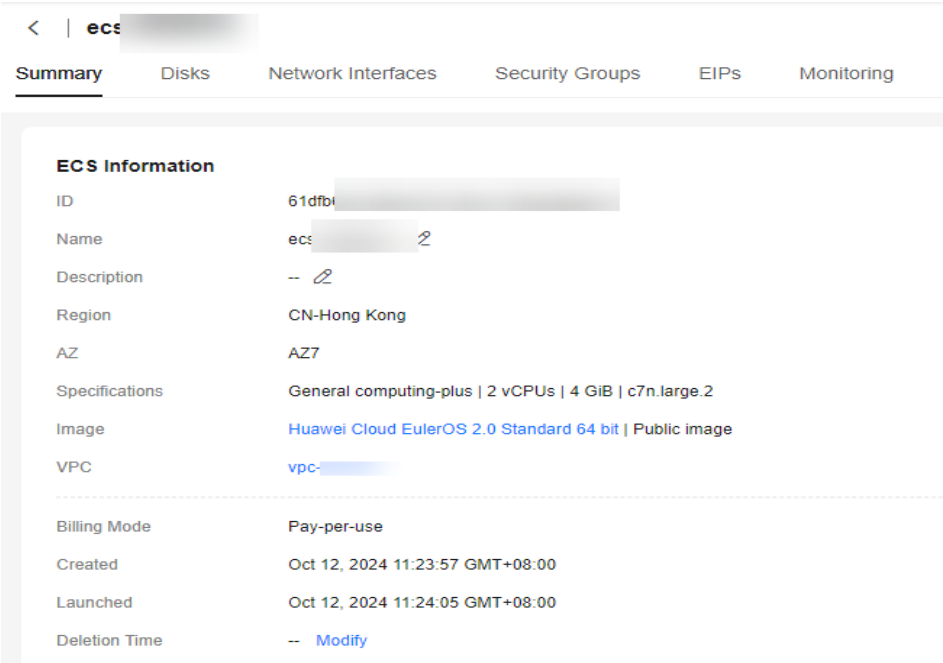


Figure 9-35 Viewing VPC information



Step 3 Obtain the IP address and port number of the real-time service.

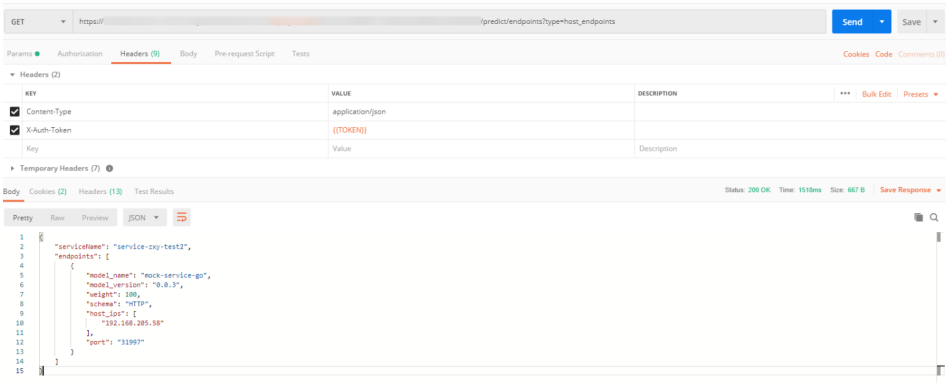
GUI software, for example, Postman can be used to obtain the IP address and port number. Alternatively, log in to the ECS, create a Python environment, and execute code to obtain the service IP address and port number.

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 9-36 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. To obtain it, see **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 9-37 Accessing a real-time service

- Create a Python environment and execute Python code to access the real-time service.


```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    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

9.4.5 Accessing a Real-Time Service Using Different Protocols

9.4.5.1 Accessing a Real-Time Service Using WebSocket

Context

WebSocket is a network transmission protocol that supports full-duplex communication over a single TCP connection. It is located at the application layer in the OSI model. The WebSocket communication protocol was established by IETF as standard RFC 6455 in 2011 and supplemented by RFC 7936. The WebSocket API in the Web IDL is standardized by W3C.

WebSocket simplifies data exchange between the client and server and allows the server to proactively push data to the client. In the WebSocket API, if the initial handshake between the client and server is successful, a persistent connection can be established between them and bidirectional data transmission can be performed.

Prerequisites

- A real-time service has been deployed with **WebSocket** enabled.
- The image for importing the AI application is WebSocket-compliant.

Constraints

- WebSocket supports only the deployment of real-time services.
- WebSocket supports only real-time services deployed using AI applications 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 a WebSocket Real-Time Service

WebSocket itself does not require additional authentication. ModelArts WebSocket is WebSocket Secure-compliant, regardless of whether WebSocket or WebSocket Secure is enabled in the custom image. WebSocket Secure supports only one-way authentication, from the client to the server.

You can use one of the following authentication methods provided by ModelArts:

- **Accessing a Real-Time Service Through Token-based Authentication**
- **Accessing a Real-Time Service Through AK/SK-based Authentication**
- **Accessing a Real-Time Service Through App Authentication**

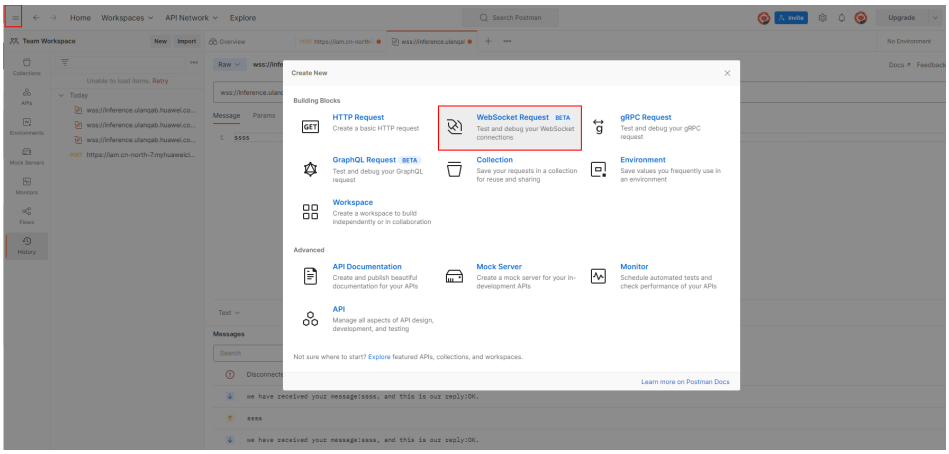
The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call WebSocket.

1. **Establish a WebSocket connection.**
2. **Exchange data between the WebSocket client and the server.**

Step 1 Establish a WebSocket connection.

1. Open Postman of a version later than 8.5, for example, 10.12.0. Click  in the upper left corner and choose **File** > **New**. In the displayed dialog box, select **WebSocket Request** (beta version currently).

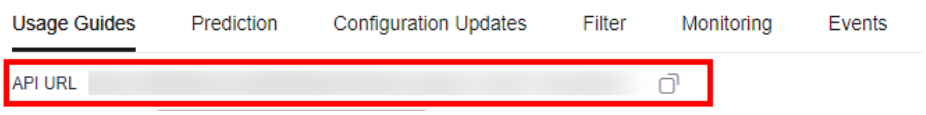
   Figure 9-38 WebSocket Request

2. Configure parameters for the WebSocket connection.

   Select **Raw** in the upper left corner. Do not select **Socket.IO** (a type of WebSocket implementation, which requires that both the client and the server run on **Socket.IO**). In the address box, enter the **API Address** obtained on the **Usage Guides** tab on the service details page. If there is a finer-grained URL in the custom image, add the URL to the end of the address. If **queryString** is available, add this parameter in the **params** column. Add authentication information into the header. The header varies depending on the authentication mode, which is the same as that in the HTTPS-compliant inference service. Click **Connect** in the upper right corner to establish a WebSocket connection.

   Figure 9-39 Obtaining the API address

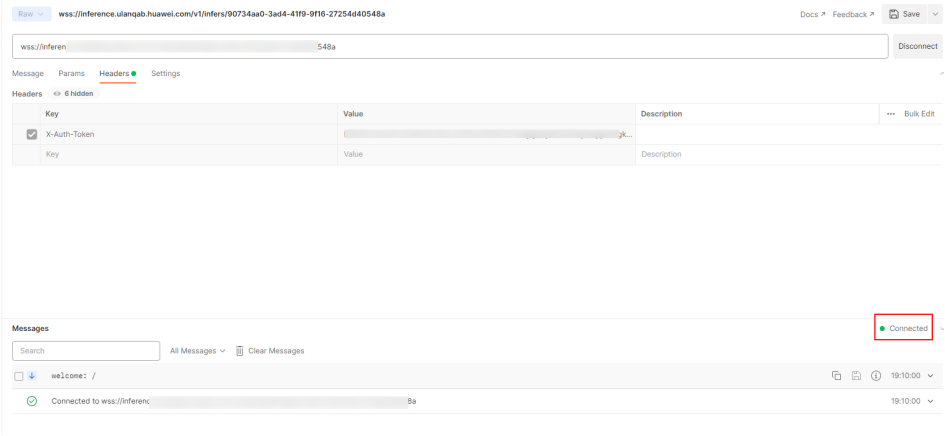
   

NOTE

- If the information is correct, **CONNECTED** will be displayed in the lower right corner.
- If establishing the connection failed and the status code is 401, check the authentication.
- If a keyword such as **WRONG_VERSION_NUMBER** is displayed, check whether the port configured in the custom image is the same as that configured in WebSocket or WebSocket Secure.

The following shows an established WebSocket connection.

Figure 9-40 Connection established



NOTICE

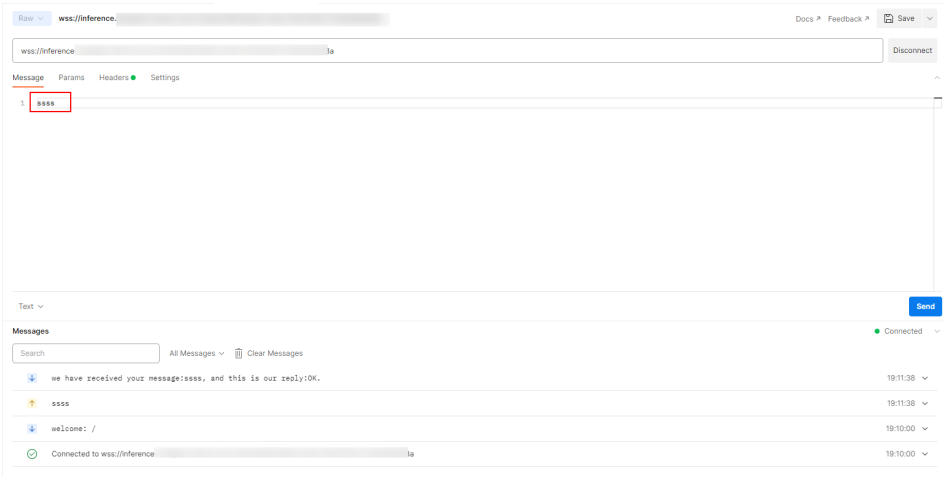
Preferentially check the WebSocket service provided by the custom image. The type of implementing WebSocket varies depending on the tool you used. Possible issues are as follows: A WebSocket connection can be established but cannot be maintained, or the connection is interrupted after one request and needs to be reconnected. ModelArts only ensures that it will not affect the WebSocket status in a custom image (the API address and authentication mode may be changed on ModelArts).

Step 2 Exchange data between the WebSocket client and the server.

After the connection is established, WebSocket uses TCP for full-duplex communication. The WebSocket client sends data to the server. The implementation types vary depending on the client, and the lib package may also be different for the same language. Different implementation types are not considered here.

The format of the data sent by the client is not limited by the protocol. Postman supports text, JSON, XML, HTML, and Binary data. Take text as an example. Enter the text data in the text box and click **Send** on the right to send the request to the server. If the text is oversized, Postman may be suspended.

Figure 9-41 Sending data



----End

9.4.5.2 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 AI application is SSE-compliant.

Constraints

- SSE supports only the deployment of real-time services.
- It supports only real-time services deployed using AI applications 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.

You can use one of the following authentication methods provided by ModelArts:

- **Accessing a Real-Time Service Through Token-based Authentication**
- **Accessing a Real-Time Service Through AK/SK-based Authentication**
- **Accessing a Real-Time Service Through App Authentication**

The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call an SSE service.

Figure 9-42 Calling an SSE service

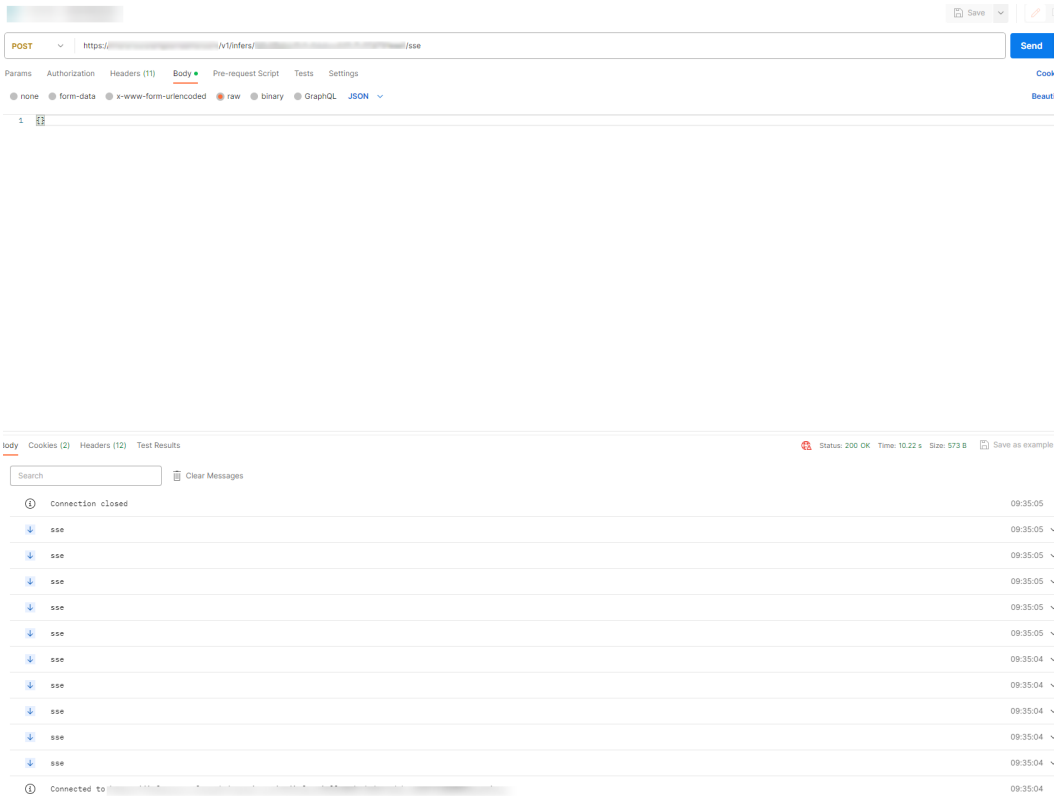


Figure 9-43 Response header **Content-Type**

NOTE

In normal cases, the value of **Content-Type** in the response header is **text/event-stream;charset=UTF-8**.

9.5 Deploying an AI Application as a Batch Inference Service

After an AI application is prepared, you can deploy it as a batch service. The **Model Deployment** > **Batch Services** page lists all batch services.

Prerequisites

- An AI application in the **Normal** state is available in ModelArts.
- 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 AI application, different parameters are entered. If the AI application input is a JSON file, a configuration file is required to generate a mapping file. If the AI application input is a file, no mapping file is required.
- Batch services can only be deployed in a public resource pool, but not a dedicated resource pool.

Procedure

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Model 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 the resource pool and AI application configurations.

Table 9-23 Parameters

Parameter	Description
AI Application Source	Choose **My AI Applications** or **My Subscriptions** as needed.
AI Application and Version	Select an AI application and version that are running properly.
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.

Parameter	Description
Request Path	URL used for calling the AI application API in a batch service, and also the request path of the AI application service. Its value is obtained from the **url** field of **apis** in the AI application configuration file.
Mapping Relationship	If the AI application input is in JSON format, the system automatically generates the mapping based on the configuration file corresponding to the AI application. If the AI application 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 AI application. 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.
Compute Nodes	Number of instances for the current AI application 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, such as plaintext passwords, 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.

Parameter	Description
Runtime Log Output	This feature is disabled by default. The runtime logs of batch services are stored only in the ModelArts log system. You can query the runtime logs in the **Logs** tab of the service details page. If this feature is enabled, the runtime logs of batch 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. For details about LTS log management, see **Log Tank Service**. **NOTE** • This cannot be disabled once it is enabled. • You will be billed for log query and storage features provided by LTS. For details, see **LTS Pricing Details**. • 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.

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://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/data.csv?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%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://infers-data.obs.xxx.com:443/xgboosterdata/2.jpg?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9jmvDGgT9VBC7dC1EyfSJrUcqfB_N0ykCsfrA1Tt_IQYZFDu_HyqVk-GunUcTVdDfWlCV3TrYcpmznZjliAnYUO89kAwCYGeRZsCsC0ePu4PHMsBvYV9gWmN9AUZIDn1sfRL4voBpwQnp6tnAgHW49y5a6hP2hCAoQ-95SpUriJ434QlymoeKfTHVMKOeZxZea-JxOvevOCGI5CcGehEJaz48sgH81UiHzl21zocNB_hpPfus2jY6KPglEJxMv6Kwmro-ZBXWuSJUDOnSYXI-3ciYjg9-h10b8W3sW1mOTFCWNGoWsd74it7l_5-7UUhoIeyPByO_REwkur2FOJsuMpGlRaPyglZxXm_jfdLFXobYtzZhbul4yWXga6oxTOkfcwykTOYH0NPoPRt5MYGYweOXXxFs3d5w2rd0y7p0QYhyTzIkk5CIz7FlWNapFISL7zdhsl8RfchTqESq94KgkeqatSF_iIvnYMW2r8P8x2k_eb6NJ7U_q5ztMbO9oWEcfr0D2f7n7Bl_nb2HIB_H9tjzKvqwngaimYhBbMRPfibvttW86GiwVP8vrC27FOn39Be9z2hSfJ_8pHej0yMlyNqZ481FQ5vWT_vFV3JHM-7I1ZB0_hIdaHfItm-J69cTfHSEOzt7DGaMIES1o7U3w%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 **Model 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 AI application prediction result.

If the prediction is successful, the directory contains the prediction result file and AI application 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.
- AI application prediction result:
  - 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

9.6 Managing AI Applications

9.6.1 Viewing Details About an AI Application

Viewing the AI Application List

You can view all created AI applications on the AI application list page. The AI application list page displays the following information.

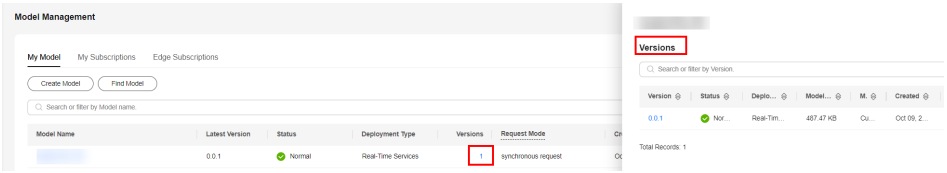
Table 9-24 AI application list

Parameter	Description
AI Application Name	Name of an AI application.
Latest Version	Latest version of an AI application.
Status	Status of an AI application.
Deployment Type	Types of the services that an AI application can be deployed as.
Versions	Number of AI application versions.

Parameter	Description
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	Time when an AI application is created.
Description	Description of an AI application.
Operation	• **Deploy**: Deploy an AI application as real-time services, batch services, or edge services. • **Create Version**: Create an AI application version. The settings of the last version are used by default, except for the version. You can change the parameter settings. • **Delete**: Delete the AI application. **NOTE** If an AI application version has been deployed as a service, you must delete the associated service before deleting the AI application version. A deleted AI application cannot be recovered.

Click the number in **Versions** to view the version list.

Figure 9-44 Version list



The version list displays the following information.

Table 9-25 Version list

Parameter	Description
Version	Current version of an AI application.
Status	Status of an AI application.
Deployment Type	Types of the services that an AI application can be deployed as.
Model Size	Size of an AI application.
Model Source	Model source of an AI application.

Parameter	Description
Created	Time when an AI application is created.
Description	Description of an AI application.
Operation	• **Deploy**: Deploy an AI application as real-time services, batch services, or edge services. • **Delete**: Delete a version of an AI application.

Viewing Details About an AI Application

After an AI application is created, you can view its information on the details page.

1. Log in to the ModelArts console, and choose **AI Applications** from the navigation pane. The **My AI Applications** tab is displayed by default.
2. Click the name of the target AI application. The application details page is displayed.

   On the application details page, you can view the basic information and model precision of the AI application, and switch tab pages to view more information.

   Table 9-26 Basic information about an AI application

Parameter	Description
Name	Name of an AI application.
Status	Status of an AI application
Version	Current version of an AI application.
ID	ID of an AI application.
Description	Click the edit button to add the description of an AI application.
Deployment Type	Types of the services that an AI application 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.
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.

Parameter	Description
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 AI application 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 olde-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	Size of an AI application.
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**.

Parameter	Description
AI Application Description	Description document added during the creation of an AI application.
Instruction Set Architecture	System architecture.
Inference Accelerator	Type of inference accelerator cards.

Table 9-27 Tabs

Parameter	Description
Model Precision	Model recall, precision, accuracy, and F1 score of an AI application.
Parameter Configuration	API configuration, input parameters, and output parameters of an AI application.
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.
Events	The progress of key operations during AI application creation. Events are stored for three months and will be automatically cleared then. For details about how to view events of an AI application, see **Viewing Events of an AI Application**.
Constraint	Displays the constraints of service deployment, such as the request mode, boot command, and model encryption, based on the settings during AI application creation. For AI applications 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 an AI application was deployed. Click a service name to go to the service details page.

9.6.2 Viewing Events of an AI Application

During the creation of an AI application, every key event is automatically recorded. You can view the events on the details page of the AI application at any time.

This helps you better understand the process of creating an AI application and locate faults more accurately when a task exception occurs. 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**
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**

Type	Event (*xxx* should be replaced with the actual value.)	Solution
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
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.

Type	Event (*xxx* should be replaced with the actual value.)	Solution
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.
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.

Type	Event (*xxx* should be replaced with the actual value.)	Solution
Abnormal	Failed to import model file *xxx* because it is larger than 5 GB.	The size of the model file *xxx* 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 *xxx* does not exist in path *xxx*.	Correct the subpath and try again, or contact technical support.
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 AI application 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 **AI Applications**. In the AI application list, click the name of the target AI application to go to its details page.
2. View the events on the **Events** tab page.

9.6.3 Managing AI Application Versions

For AI application lineage and tuning, ModelArts provides AI application versioning.

Prerequisites

An AI application has been created in ModelArts.

Creating a Version

On the **AI Applications** page, click **Create Version** in the **Operation** column of the target AI application. On the **Create Version** page, configure parameters. For details, see **Creating an AI Application**. Click **Create now**.

Deleting a Version

On the **AI Applications** page, click the number in **Versions**. Click **Delete** in the **Operation** column of the target version.

NOTE

If an AI application version has been deployed as a service, you must delete the associated service before deleting the AI application version. A deleted version cannot be recovered.

Deleting an AI Application

On the **AI Applications** page, click **Delete** in the **Operation** column of the target AI application.

NOTE

If an AI application version has been deployed as a service, you must delete the associated service before deleting the AI application version. A deleted AI application cannot be recovered.

9.7 Managing a Synchronous Real-Time Service

9.7.1 Viewing Details About a Real-Time Service

After an AI application 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 **Model Deployment** > **Real-Time Services**.
2. Click the name of the target service to access its details page.

   View service information. For details, see **Table 9-28**.

Table 9-28 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.

Parameter	Description
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**.
Traffic Limit	Maximum number of times a service can be accessed within a second.
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. For details about LTS log management, see **Log Tank Service**. **NOTE** • This cannot be disabled once it is enabled. • You will be billed for log query and storage features provided by LTS. For details, see **LTS Pricing Details**. • 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 9-29**.

Table 9-29 Details of a real-time service

Parameter	Description
Usage Guides	This page displays the API URL, AI application information, input parameters, and output parameters. You can click to copy the API URL to call the service. If application authentication is supported, you can view the API URL and authorization management details, including the application name, AppKey, and AppSecret, in the **Usage Guides**. You can also add or cancel authorization for an application.
Prediction	You can perform real-time prediction on this page. For details, see **Testing Real-Time Service Prediction**.

Parameter	Description
Instance	Instance information of the synchronous real-time service. The number of instances is the same as the number of compute nodes 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**: AI application name, version, status, instance flavor, traffic ratio, number of compute nodes, 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 AI application information.
Monitoring	This page displays resource usage and AI application calls. • **Resource Usage**: includes the used and available CPU, memory, GPU, and NPU resources. • **AI Application Calls**: indicates the number of AI application calls. The statistics collection starts after the AI application 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**.

Parameter	Description
Logs	This page displays the log information about each AI application 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. If this feature is enabled, the logs stored in LTS will be displayed. You can click **View Complete Logs on LTS** to view all logs. 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***.
Tags	Tags that have been added to the service. Tags can be added, modified, and deleted. For details about how to use tags, see **How Does ModelArts Use Tags to Manage Resources by Group?**
Cloud Shell	You can use Cloud Shell provided by the ModelArts console to log in to the instance container of a running real-time service. For details, see **Using Cloud Shell to Debug a Real-Time Service Instance Container**.

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 9-30 Parameters for **Custom Settings**

Parameter	Mandatory	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 9-31 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:

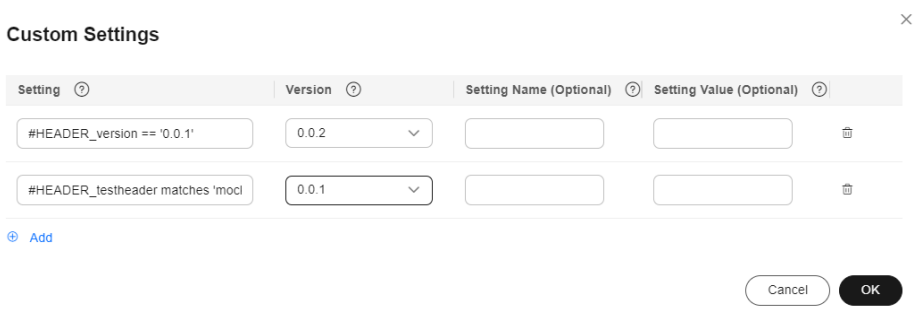
  If the account name in the inference request starts with **op**, the specified version is matched.

  ```
  #DOMAIN_NAME matches 'op.*'
  ```

Table 9-32 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}	*n* is a non-negative integer, which matches exactly *n* number of occurrences of an expression. For example, **o{2}** cannot match **o** in **Bob**, but can match two **o**s 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**.

Figure 9-45 Traffic distribution by user



- 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.

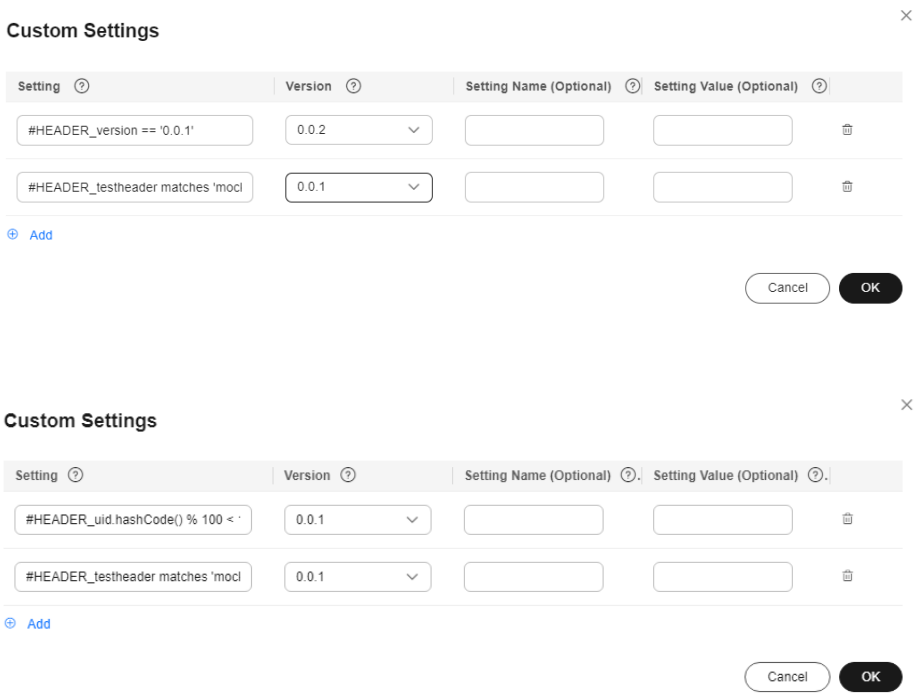
    ```
    #HEADER_testheader matches 'mock.*'
    ```

  - 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
    ```

    Figure 9-46 Using the header to access different versions

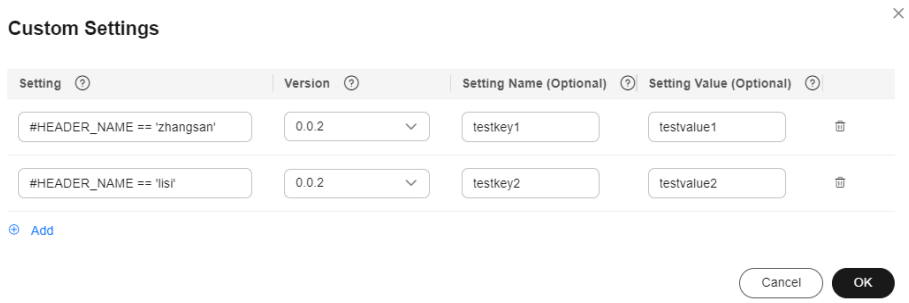
- If a real-time service version supports different runtime configurations, you can use **Setting Name** and **Setting Value** to specify customized runtime parameters so that different users can use different running configurations.

  Example:

  When user A accesses the AI application, the user uses configuration A. When user B accesses the AI application, 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**.

Figure 9-47 Customized running parameters added for a customized configuration rule



9.7.2 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 9-33 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.
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

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 *x*s.	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 **Model 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.

9.7.3 Managing the Lifecycle of a Real-Time Service

Starting a Service

A service not in the **Deploying** state can be started. A service is billed from the time when it is running. To start a service, use either of the following methods:

- Log in to the ModelArts console and choose **Model 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 **Model 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.

NOTE

Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be started.

Stopping a Service

A stopped service will no longer be billed. Stop a service in either of the following ways:

- Log in to the ModelArts console and choose **Model Deployment** from the navigation pane. Go to the service management page of the target service. Click **Stop** in the **Operation** column to stop a service. (For a real-time service, choose **More** > **Stop** in the **Operation** column.)
- Log in to the ModelArts console and choose **Model 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 **Stop** in the upper right corner of the page to stop the service.

  NOTE

  Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be stopped.

Deleting a Service

If a service is no longer in use, delete it to release resources.

Log in to the ModelArts console and choose **Model Deployment** from the navigation pane. Go to the service management page of the target service.

- 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.

- 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.
- 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 **Model Deployment** > **Real-Time Services** from the navigation pane. Choose **More** > **Restart** in the **Operation** column to restart the target service.
- Log in to the ModelArts console and choose **Model Deployment** > **Real-Time Services** from the navigation pane. 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.

NOTE

Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be restarted.

9.7.4 Modifying a Real-Time Service

For a deployed service, you can modify its basic information to match service changes and change the AI application 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 9-34 Scenarios for hitless rolling upgrade

Meta Model Source for Creating an AI Application	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 an AI application 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 **Model 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.

Method 2: Modify the Service Information on the Service Details Page

1. Log in to the ModelArts console and choose **Model 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.

9.7.5 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 9-35 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

ID	Name	Description	Value Range	Monitored Object	Monitoring Interval
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
disk_read_rate	Disk Read Rate	Disk read rate of ModelArts Unit: bit/minute	≥ bit/minute	ModelArts model loads	1 minute
disk_write_rate	Disk Write Rate	Disk write rate of ModelArts Unit: bit/minute	≥ bit/minute	ModelArts model loads	1 minute
send_bytes_rate	Uplink rate	Outbound network traffic rate of ModelArts. Unit: bit/minute	≥ bit/minute	ModelArts model loads	1 minute
recv_bytes_rate	Downlink rate	Inbound network traffic rate of ModelArts.	≥ bit/minute	ModelArts model loads	1 minute
req_count_2xx	2xx Responses	Number of times that the API returns a 2xx response	≥ counts/minute	ModelArts real-time services	1 minute
req_count_4xx	4xx Errors	Number of times that the API returns a 4xx error	≥ counts/minute	ModelArts real-time services	1 minute
req_count_5xx	5xx Errors	Number of times that the API returns a 5xx error	≥ counts/minute	ModelArts real-time services	1 minute
avg_latency	Average Latency	Average latency of the API	≥ 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 9-36 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

9.7.6 Integrating a Real-Time Service API into the Production Environment

For a real-time service API that has been commissioned, you can integrate it into the production environment.

Prerequisites

The real-time service is running. Otherwise, applications in the production environment will be unavailable.

Integration Mode

ModelArts real-time services provide standard RESTful APIs, which can be accessed using HTTPS. ModelArts provides SDKs for calling real-time service APIs. For details about how to call the SDKs, see "Scenario 1: Perform an inference test using the predictor" in ***SDK Reference***.

In addition, you can use common development tools and languages to call the APIs. You can search for and obtain the guides for calling standard RESTful APIs on the Internet.

9.8 Managing Batch Inference Jobs

9.8.1 Viewing Details About a Batch Service

After an AI application 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 **Model Deployment** > **Batch Services**.
2. Click the name of the target service to access its details page.

   View service information. For details, see **Table 9-37**.

Table 9-37 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.

Parameter	Description
Instance Flavor	Node flavor of the batch service.
Compute Nodes	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.
AI Application Name & Version	Name and version of the AI application used by the batch service.
Advanced Log Management	This feature is disabled by default. The runtime logs of batch services are stored only in the ModelArts log system. If this feature is enabled, the runtime logs of batch 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. For details about LTS log management, see **Log Tank Service**. **NOTE** • This cannot be disabled once it is enabled. • You will be billed for log query and storage features provided by LTS. For details, see **LTS Pricing Details**. • 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.

3. Switch between tabs on the details page of a batch service to view more details. For details, see **Table 9-38**.

Table 9-38 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**.
Logs	This page displays the log information about each AI application 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. If this feature is enabled, the logs stored in LTS will be displayed. You can click **View Complete Logs on LTS** to view all logs. 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***.

9.8.2 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 9-39 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.
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 *x*s.	N/A

Event Type	Event (*xxx* should be replaced with the actual value.)	Solution
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
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

Event Type	Event (*xxx* should be replaced with the actual value.)	Solution
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 **Model 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.

9.8.3 Managing the Lifecycle of a Batch Service

Starting a Service

A service not in the **Deploying** state can be started. A service is billed from the time when it is running. To start a service, use either of the following methods:

- Log in to the ModelArts console and choose **Model 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 **Model 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.

NOTE

Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be started.

Stopping a Service

A stopped service will no longer be billed. Stop a service in either of the following ways:

- Log in to the ModelArts console and choose **Model Deployment** from the navigation pane. Go to the service management page of the target service. Click **Stop** in the **Operation** column to stop a service. (For a real-time service, choose **More** > **Stop** in the **Operation** column.)

- Log in to the ModelArts console and choose **Model 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 **Stop** in the upper right corner of the page to stop the service.

  NOTE

  Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be stopped.

Deleting a Service

If a service is no longer in use, delete it to release resources.

Log in to the ModelArts console and choose **Model Deployment** from the navigation pane. Go to the service management page of the target service.

- 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.
- 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.
- 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 **Model Deployment** > **Real-Time Services** from the navigation pane. Choose **More** > **Restart** in the **Operation** column to restart the target service.
- Log in to the ModelArts console and choose **Model Deployment** > **Real-Time Services** from the navigation pane. 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.

NOTE

Services deployed on ModelArts edge nodes or ModelArts edge resource pools cannot be restarted.

9.8.4 Modifying a Batch Service

For a deployed service, you can modify its basic information to match service changes and change the AI application 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 9-40 Scenarios for hitless rolling upgrade

Meta Model Source for Creating an AI Application	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 an AI application 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 **Model 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 an AI Application 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 **Model 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 an AI Application as a Batch Inference Service**.

10 Image Management

10.1 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 AI application.

There are two types of preset images:

- Unified image: Use a unified image when you create a notebook instance, training job, or AI application. All images released later are unified images. For details, see **ModelArts 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. For details, see **Preset Dedicated Images in Notebook Instances**, **Preset Dedicated Images for Training**, and **Preset Dedicated Images for Inference**.

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 AI applications are obtained from the SWR service management list.

Figure 10-1 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 AI applications.

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

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. You can obtain the SWR address of the image in the ModelArts preset image list. For details, see **Preset Images Supported by ModelArts** and .

- **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.
- **Creating a Custom Image for Inference**

  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 AI applications. The AI applications created in this way can be centrally managed and deployed as services.

10.2 Preset Images Supported by ModelArts

10.2.1 ModelArts Preset Image Updates

This section describes the updates of ModelArts preset images, including changes of dependencies, so that you can learn about the changes and use images more easily.

Run the following command on the terminal to query the dependencies contained in the image:

```
pip list
```

Unified Image Updates

Table 10-1 Unified image updates

Image	Updated	Description
mindspore_2.3.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	Based on the commercial version of Ascend 415, MindSpore is updated to 2.3.0-rc4, and CANN is updated to 8.0.rc1.
pytorch_2.1.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	Based on the commercial version of Ascend 415, CANN is updated to 8.0.rc1.
pytorch_1.11.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	Based on the commercial version of Ascend 415, CANN is updated to 8.0.rc1.

10.2.2 ModelArts Unified Images

Unified Image List

ModelArts provides unified images of Arm+Ascend specifications, including MindSpore and PyTorch. You can use the images to develop environment, train models, and deploy services. For details, see **Unified Image List**.

Table 10-2 MindSpore

Preset Image	Supported Processor	Applicable Scope
mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment

Table 10-3 PyTorch

Preset Image	Supported Processor	Applicable Scope
pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment

Preset Image	Supported Processor	Applicable Scope
pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook, training, and inference deployment

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

Table 10-4 Information about the image

AI Engine	URL
mindspore 2.2.0 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	swr.<region>.myhuaweicloud.com/atelier/mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83

pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

Table 10-5 Information about the image

AI Engine	URL
pytorch 2.1.0 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	**/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83**

pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

Table 10-6 Information about the image

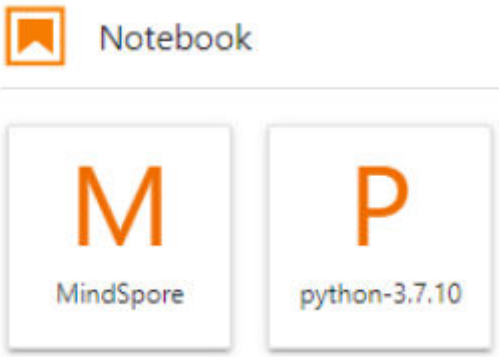
AI Engine	URL
pytorch 1.11 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	**/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83** Example: CN-Hong Kong swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83

10.2.3 Preset Dedicated Images in Notebook Instances

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

The images preset in ModelArts DevEnviron include:

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



  Select a Conda environment based on whether MindSpore is used for debugging.
- Notebook: a web application that enables you to code on the GUI and combine the code, mathematical equations, and visualized content into a document.
- JupyterLab plug-ins: enable flavor changing, case sharing to AI Gallery for communication, and instance stopping to improve user experience.

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

Table 10-7 Preset x86 images

Engine	Image
PyTorch	pytorch1.8-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.10-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.4-cuda10.1-cudnn7-ubuntu18.04
Tensorflow	tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04
	tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04
MindSpore	mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04
	mindspore1.7.0-py3.7-ubuntu18.04
	mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04
	mindspore1.2.0-openmpi2.1.1-ubuntu18.04
No AI engine (base images dedicated for image customization)	conda3-cuda10.2-cudnn7-ubuntu18.04
	conda3-ubuntu18.04

x86-powered PyTorch Base Images

PyTorch contains three types of images:

- **Image 1: pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**
- **Image 2: pytorch1.10-cuda10.2-cudnn7-ubuntu18.04**
- **Image 3: pytorch1.4-cuda10.1-cudnn7-ubuntu18.04**

Image 1: pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

Table 10-8 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
PyTorch 1.8	Yes (CUDA 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

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

Table 10-9 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Pytorch 1.10	Yes (CUDA 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c

Image 3: pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

Table 10-10 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Pytorch 1.4	Yes (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e

x86-powered TensorFlow Base Images

TensorFlow contains two types of images:

- **Image 1: tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04**

- Image 2: tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

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

Table 10-11 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Tensorflow 2.1	Yes (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e

Image 2: tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

Table 10-12 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Tensorflow 1.13-gpu	Yes (CUDA 10.0)	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

x86-powered MindSpore Base Images

MindSpore contains four types of images:

- Image 1: mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04
- Image 2: mindspore1.7.0-py3.7-ubuntu18.04
- Image 3: mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04
- Image 4: mindspore1.2.0-openmpi2.1.1-ubuntu18.04

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

Table 10-13 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Mindspore -gpu 1.7.0	Yes (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e

Image 2: mindspore1.7.0-py3.7-ubuntu18.04

Table 10-14 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Mindspore 1.7.0	No	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e

Image 3: mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

Table 10-15 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Mindspore -gpu 1.2.0	Yes (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e

Image 4: mindspore1.2.0-openmpi2.1.1-ubuntu18.04

Table 10-16 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
Mindspore 1.2.0	No	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e

x86-powered Custom Dedicated Base Images

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

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

Image 1: conda3-cuda10.2-cudnn7-ubuntu18.04

Table 10-17 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
None	Yes (CUDA 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c

Image 2: conda3-ubuntu18.04

Table 10-18 Information about the image

AI Engine	Whether to Use GPUs (CUDA Version)	URL
None	No	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c Example: CN North-Beijing4 swr.cn-north-4.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c CN East-Shanghai1 swr.cn-east-3.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c CN South-Guangzhou swr.cn-south-1.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c

10.2.4 Preset Dedicated Images for Training

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

Available Training Base Images

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

Table 10-19 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

Engine	Version
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

NOTE

Supported AI engines vary depending on regions.

PyTorch Base Images

This section describes preset PyTorch images.

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

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

- Image address: swr.{*Region*}.myhuaweicloud.com/aip/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-roma-20220309171256-40adcc1
- Image creation time: 20220309171256 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.2.89
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages**

TensorFlow Base Images

This section describes preset TensorFlow images.

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

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

- Image address: swr.{*Region*}.myhuaweicloud.com/aip/tensorflow_2_1:train-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Image creation time: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python**, Python 3.7.10

- Third-party package installation path: **/home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages**

Horovod Base Images

This section describes preset Horovod images.

- **Engine Version 1: horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Engine Version 2: horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**

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

- Image address: swr.{*Region*}.myhuaweicloud.com/aip/horovod_tensorflow:train-horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Image creation time: 20210912152543 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages**

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

- Image address: swr.{*Region*}.myhuaweicloud.com/aip/horovod_pytorch:train-horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Image creation time: 20210912152543 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages**

MPI Base Images

This section describes preset mindspore_1.3.0 images.

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

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

- Image address: swr.{*Region*}.myhuaweicloud.com/aip/mindspore_1_3_0:train-mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-roma-20211104202338-f258e59
- Image creation time: 20211104202338 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages**

10.2.5 Preset Dedicated Images for Inference

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

x86 (CPU/GPU)-powered Base Images

Table 10-20 TensorFlow

AI Engine Version	Runtime Environment	Image	URI
2.1.0	CPU GPU(cuda10.1)	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
1.15.5	CPU GPU(cuda11.4)	tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
2.6.0	CPU GPU(cuda11.2)	tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

Table 10-21 PyTorch

AI Engine Version	Runtime Environment	Image	URI
1.8.0	CPU GPU(cuda 10.2)	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
1.8.2	CPU GPU(cuda 11.1)	pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

Table 10-22 MindSpore

AI Engine Version	Runtime Environment	Image name	URI
1.7.0	CPU	mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
1.7.0	GPU(cuda 10.1)	mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

AI Engine Version	Runtime Environment	Image name	URI
1.7.0	GPU(cuda 11.1)	mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64	swr.{*Region ID*}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

TensorFlow (CPU/GPU)-powered Base Images

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

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

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

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

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

- Image address: swr.{*Region ID*}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
- Image creation time: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 20.04.4 LTS
- CUDA: 11.4.3
- cuDNN: 8.2.4.15
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python**, Python 3.8.13

- Third-party package installation path: **/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages**

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

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

PyTorch (CPU/GPU)-powered Base Images

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

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

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

- Image address: swr.{*Region ID*}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
- Image creation time: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 10.2.89
- cuDNN: 7.6.5.32
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages**

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

- Image address: swr.{*Region ID*}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- Image creation time: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39

- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages**

MindSpore (CPU/GPU)-powered Base Images

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

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

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

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

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

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

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

- Image address: swr.{*Region ID*}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Image creation time: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Image system version: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1

- cuDNN: 8.0.5.39
- Path and version of the Python interpreter: **/home/ma-user/anaconda3/envs/MindSpore/bin/python**, Python 3.7.10
- Third-party package installation path: **/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages**

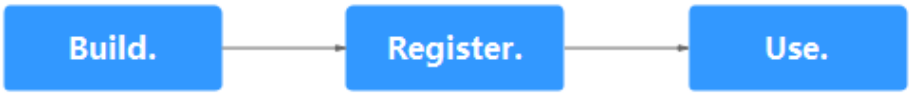
10.3 Creating a Custom Image for a Notebook Instance

10.3.1 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 10-2 Creating a custom image (for scenarios 1 and 2)



Scenario 1: Build and register an image based on the preset image or third-party image in a notebook instance, configure the Docker environment on the server, and compile the Dockerfile. For details, see **Creating a Custom Image on ECS and Using It**.

Scenario 2: Use **ma-cli** to create and register an image based on the preset or third-party image provided by a notebook instance for AI development. In this case, the notebook instance is the platform for creating the image. For details, see **Creating a Custom Image Using Dockerfile**.

Scenario 3: 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.

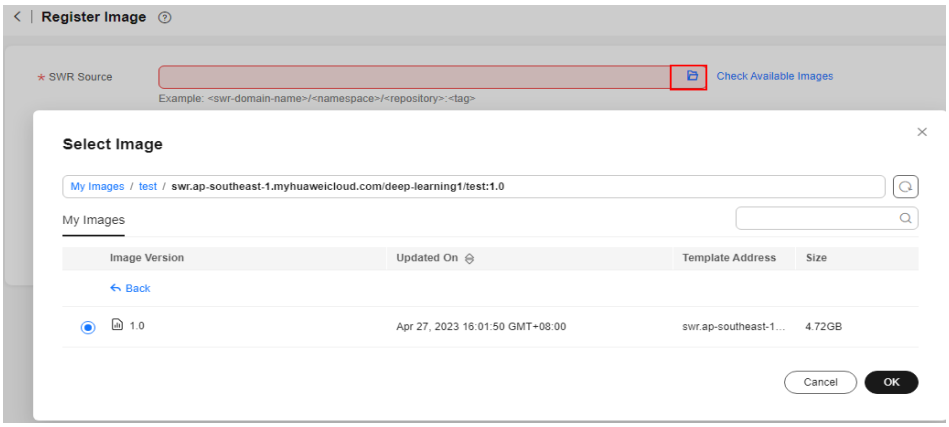
NOTE

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**.

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

     Figure 10-3 Selecting an image source

   - **Architecture** and **Type**: Configure them based on the actual framework of the custom image.
3. View the registered image on the **Image Management** page.

10.3.2 Creating a Custom Image on ECS and Using It

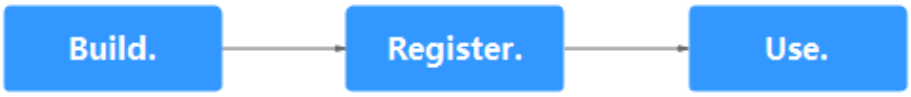
Application Scenarios and Process

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

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

The following figure shows the whole process.

Figure 10-4 Creating and debugging an image



Specifications for Custom Images

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

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

  x86: Ubuntu 18.04 or Ubuntu 20.04

  Arm: Euler 2.8.3 or Euler 2.10.7

  NOTE

  There may be a compatibility issue for Ubuntu 20.04.6. Use an earlier version.
- If an image error occurs due to unmet requirements, check the image specifications and rectify the fault by referring to **Troubleshooting for Custom Images in Notebook Instances**. If the fault persists, contact Huawei technical support.

Procedure

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

Preparing a Docker Server and Configuring the Environment

Prepare a server with Docker enabled. If no such a server is available, create an ECS, buy an EIP, and install required software on it.

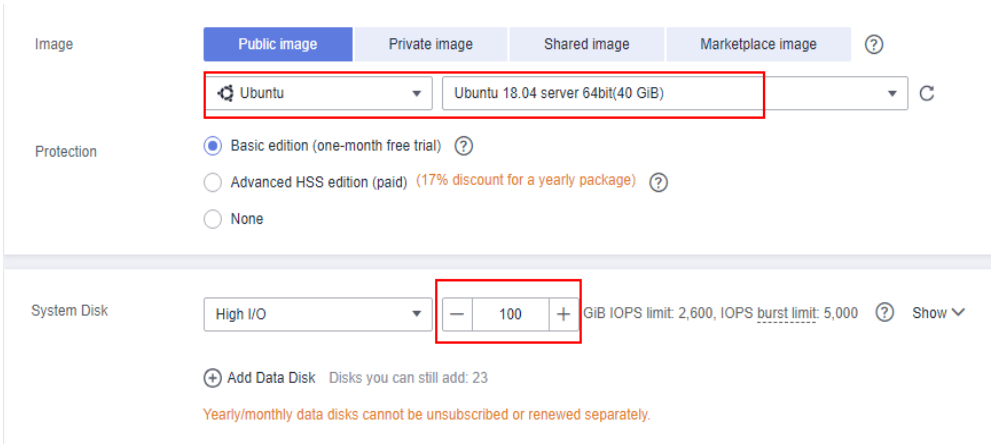
ModelArts provides Ubuntu scripts for you to install Docker easier.

NOTE

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

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

   Figure 10-5 Selecting an image and a disk

2. Purchase an EIP and bind it to the ECS. For details, see **Configure Network**.
3. Configure the VM environment.
   a. Run the following command on the Docker ECS to download the installation script:

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

      NOTE

      Only Ubuntu scripts are supported.

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

      ```
      bash install_on_ubuntu1804.sh
      ```

      Figure 10-6 Configured


      ```
      source /etc/profile
      ```

      The installation script is executed to:

      i. Install Docker.

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

Creating a Custom Image

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

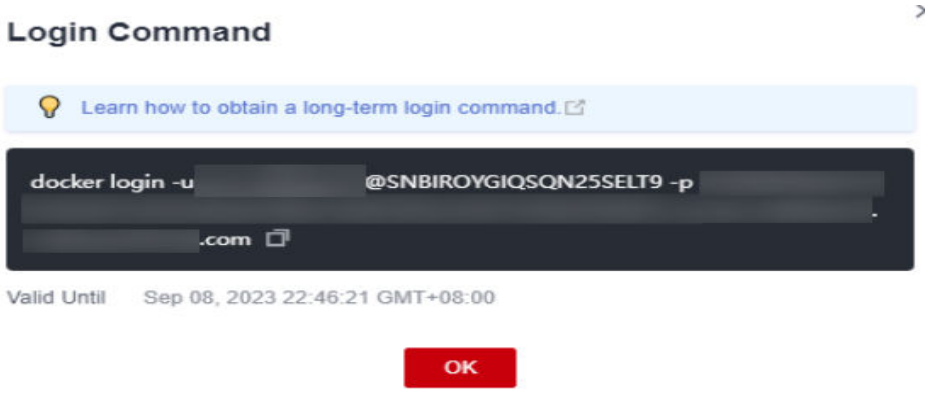
1. Querying Base Images (Skip This Step for Third-Party Images)

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

2. Access SWR.

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

   Figure 10-7 Obtaining the login command

   NOTE

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

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

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

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

4. Compile a Dockerfile.

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

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

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

5. Build an image.

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

   - **-t** specifies the new image path, including region information, organization name, image name, and version. Set this parameter based on the real-life scenario. Use a complete SWR address for debugging and registration.
   - **-f** specifies the Dockerfile name. Set this parameter based on the real-life scenario.
   - The period (.) at the end specifies that the context is the current directory. Set this parameter based on the real-life scenario.

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

   Figure 10-8 Image created

Registering a New Image

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

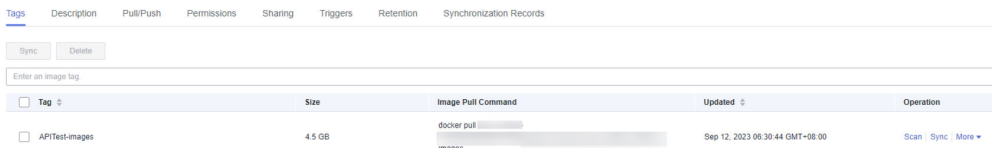
1. Upload the image to SWR.

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

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

   The image is then available on SWR.

   Figure 10-9 Uploading the image to SWR

2. Register an image.

   Registering an image on the ModelArts console

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

   a. Click **Register**. Set **SWR Source** to the image pushed to SWR in **step 1**.

      Paste the complete SWR address or click to select a private image from SWR for registration.

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

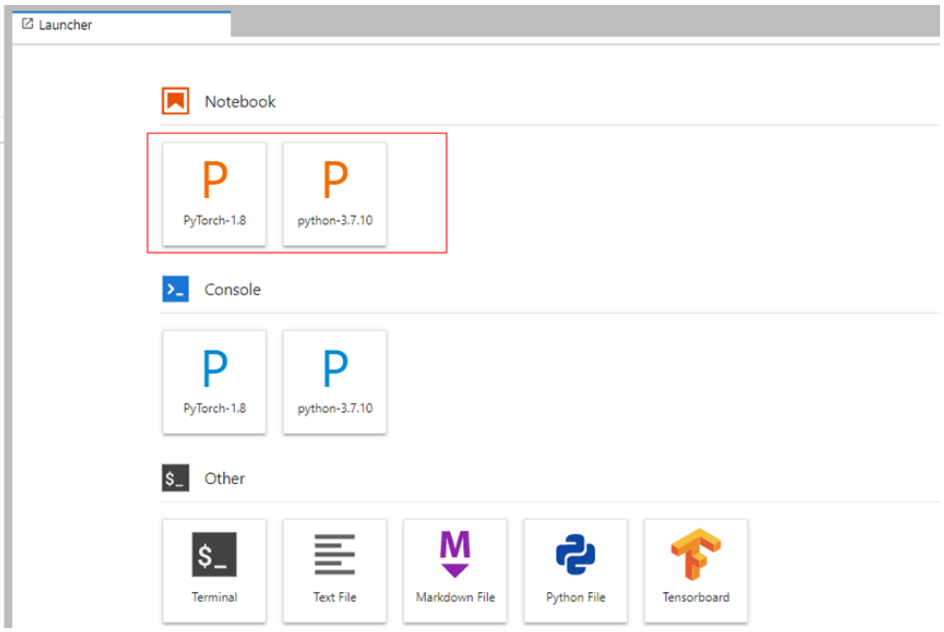
NOTE

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

Using a New Image to Create a Development Environment

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

Figure 10-10 Accessing a development environment



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

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

Figure 10-11 Checking the conda environment

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

Dockerfile on a ModelArts Base Image

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


```
FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/notebook2.0-pytorch-1.4-kernel-cp37:3.3.3-release-v1-20220114

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

Dockerfile on a Non-ModelArts Base Image

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

NOTE

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

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

```
# Replace it with the actual image version.
FROM ubuntu:18.04
# Set the user ma-user whose UID is 1000 and the user group ma-group whose GID is 100
USER root
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
```



```
    fi && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
    fi && \
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user && \
# Grant the read, write, and execute permissions on the target directory to the user ma-user.
chmod -R 750 /home/ma-user

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

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

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

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

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

#Install FFmpeg and GCC
USER root
RUN apt-get -y install ffmpeg && \
    apt -y install gcc-8 g++-8
```

10.3.3 Creating a Custom Image Using Dockerfile

Scenario

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

Procedure

1. Create a notebook instance.
2. Create a custom image in the notebook instance.
3. Register the image on ModelArts.
4. Create a notebook instance and verify the new image.

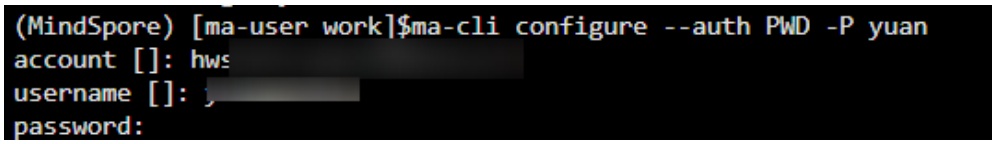
Creating a Notebook Instance

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**. On the displayed page, click **Create Notebook**. Set **Image** to **Public image** and select a PyTorch image. Retain the default values for other parameters. For details, see **Creating a Notebook Instance**.
2. After the notebook instance is created and in the **Running** state, locate it in the notebook list, and click **Open** in the **Operation** column. On the displayed **JupyterLab** page, click **Terminal**.

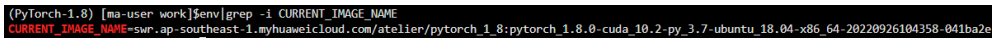
Creating a Custom Image in a Notebook Instance

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

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



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



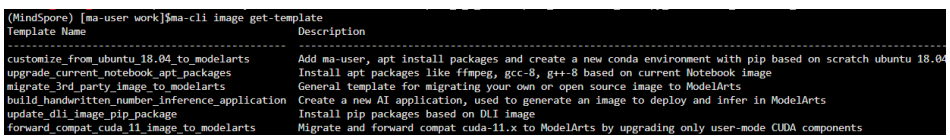
Step 3 Create an image.

1. Obtain the SWR address of the base image.

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

2. Load an image creation template.

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



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

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



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

3. Modify a Dockerfile.

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

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

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

   # Set proxy to download internet resources
   ENV HTTP_PROXY=http://proxy.modelarts.com:80 \
       http_proxy=http://proxy.modelarts.com:80 \
       HTTPS_PROXY=http://proxy.modelarts.com:80 \
       https_proxy=http://proxy.modelarts.com:80

   USER root

   # Config apt source which can accelerate the apt package download speed. Also install ffmpeg and gcc-8 in root mode
   RUN cp -f /etc/apt/sources.list /etc/apt/sources.list.bak && \
       sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
       sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
       apt update && \
       apt -y install ffmpeg && \
       apt install -y --no-install-recommends gcc-8 g++-8 && apt-get autoremove -y && \
       update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8

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

4. Build an image.

   Run the **ma-cli image build** command to build an image with the Dockerfile. For details about the command, see **ma-cli Image Building Commands**.

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

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

----End

Registering an Image

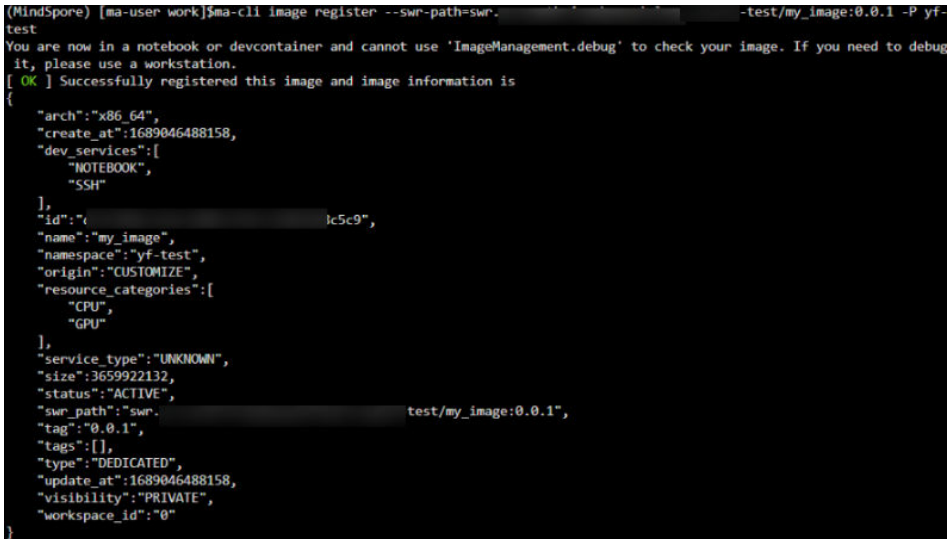
After an image is created, register it with ModelArts image management so that the image can be used in ModelArts.

Use either of the following methods:

- **Method 1**: Run the **ma-cli image register** command to register an image. Then, the information of the registered image is returned, including image ID and name, as shown in the following figure. For details about the command, see **ma-cli Image Building Commands**.

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


Figure 10-12 Registering an image



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

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

  a. Click **Register**. Paste the complete SWR address or click to select a private image from SWR for registration.

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

Creating and Using a Notebook Instance

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

10.3.4 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

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

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

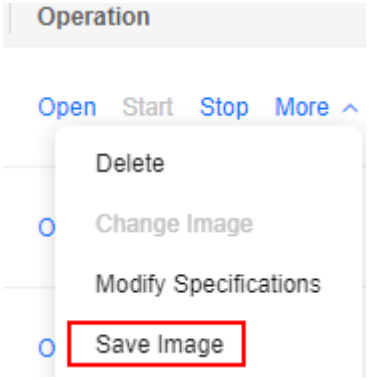
Prerequisites

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

Saving an Image

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 click **More** > **Save Image** in the **Operation** column. The **Save Image** dialog box is displayed.

   Figure 10-13 Saving an image

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

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

   Users in an organization can share all images in the organization.
4. The image is saved as a snapshot. The storage duration depends on the image size. During this period of time, do not perform any operations on the instance.

   Figure 10-14 Saving an image

   NOTICE

   The time required for saving an image as a snapshot will be counted in the instance running duration. If the instance running duration expires before the snapshot is saved, saving the image will fail.
5. After the image is saved, the instance status changes to **Running**. View the image on the **Image Management** page.

6. Click the image name to view its details.

Creating a Notebook Instance Using a Custom Image

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

You can use either of the following methods:

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

Figure 10-15 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**.

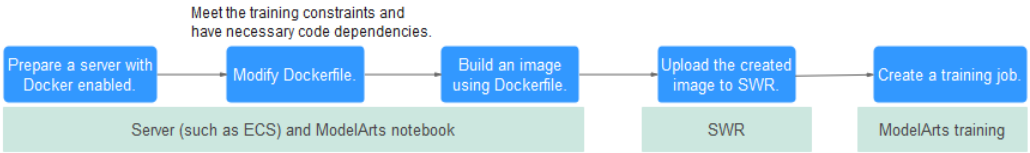
10.4 Creating a Custom Image for Model Training

10.4.1 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 10-16 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:

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

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

Creating a Custom Training Image (Tensorflow + GPU)

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).

10.4.2 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.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

4. Create an image based on a training base image provided by ModelArts. Save the edited Dockerfile in the **context** folder. For details about how to obtain a training base image, see **Preset Dedicated Images for Training**.

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

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

   # Configure the preset environment variables of the container image.
   # Add the Python interpreter path to the PATH environment variable.
   # Set PYTHONUNBUFFERED to 1 to prevent log loss.
   ```



```
ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
    PYTHONUNBUFFERED=1

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

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

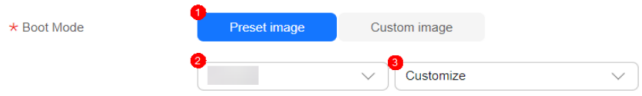
      Figure 10-17 Login command executed on ECS

   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.

      SWR URL of the custom image: swr.<*Region*>.myhuaweicloud.com/deep-learning/tf-1.13.2:latest
7. Create a training job on ModelArts.
   a. Log in to the ModelArts console.
   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 10-23**. For details about the parameters, see **Creating a Production Training Job**.

      Table 10-23 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. 

Parameter	Description
Image	Select **the image uploaded to SWR** for container image.
Code Directory	Mandatory. Select the OBS directory where the training code file is stored. • Upload code to the OBS bucket beforehand. The total size of files in the directory cannot exceed 5 GB, the number of files cannot exceed 1,000, and the folder depth cannot exceed 32. • The training code file is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container when the training job is started. **demo-code** is the last-level OBS directory for storing the code. For example, if **Code Directory** is set to **/test/code**, the training code file is downloaded to the **${MA_JOB_DIR}/code** directory of the training container.
Boot File	Mandatory. Select the Python boot script of the training job in the code directory. ModelArts supports only the boot file written in Python. Therefore, the boot file must end with .py.

10.4.3 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}
```

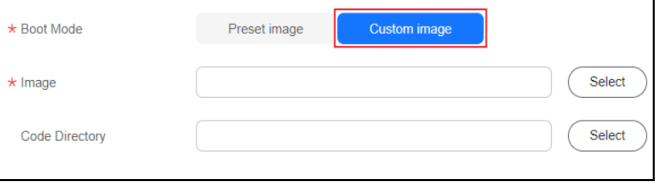
3. Upload the new image to SWR. For details, see **6**.
4. Create a training job on ModelArts.

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 10-23**. For details about the parameters, see **Creating a Production Training Job**.

Table 10-24 Creating a training job using a custom image

Parameter	Description
Algorithm Type	Mandatory. Select **Custom algorithm**.
Boot Mode	Mandatory. Select **Custom image**. 

Parameter	Description
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 **${MA_JOB_DIR}/demo-code** directory of the training container when the training job is started. **demo-code** is the last-level OBS directory for storing the code. For example, if **Code Directory** is set to **/test/code**, the training code file is downloaded to the **${MA_JOB_DIR}/code** directory of the training container.
User ID	User ID for running the container. The default value **1000** is recommended. If the UID needs to be specified, its value must be within the specified range. The UID ranges of different resource pools are as follows: • Public resource pool: **1000** to **65535** • Dedicated resource pool: **0** to **65535**
Boot Command	Mandatory. Command for booting an image. When a training job is running, the boot command is automatically executed after the code directory is downloaded. • If the training boot script is a .py file, **train.py** for example, the boot command is as follows: `python ${MA_JOB_DIR}/demo-code/train.py` • If the training boot script is a .sh file, **main.sh** for example, the boot command is as follows: `bash ${MA_JOB_DIR}/demo-code/main.sh` You can use semicolons (;) and ampersands (&&) to combine multiple commands. **demo-code** in the command is the last-level OBS directory where the code is stored. Replace it with the actual one.

Parameter	Description
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.

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

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

NOTE

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

Scenarios

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

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

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

Procedure

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

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

Prerequisites

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

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 10-25** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

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

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

Table 10-25 Folder to create

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

Step 2 Preparing the Training Script and Uploading It to OBS

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

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

```
import torch
import torch.nn as nn

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

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

Step 3 Preparing a Host

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

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

Step 4 Creating a Custom Image

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

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

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

1. Install Docker.

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

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

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

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

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

   The following information is displayed:

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

   NOTE

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

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

   ```
   mkdir -p context
   ```

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

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

   NOTE

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

5. Download the **torch*.whl** files. Download the following .whl files from https://download.pytorch.org/whl/torch_stable.html:
   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   NOTE

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

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

6. Download the Miniconda3-py37_4.12.0-Linux-x86_64.sh installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.
7. Store the pip source file, torch*.whl file, and Miniconda3 installation file in the **context** folder, which is as follows:


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

8. Write the container image Dockerfile.

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

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

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

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

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

# Copy the installation files to the /tmp directory in the base container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

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

# Install torch*.whl using the default Miniconda3 Python environment in /home/ma-user/
miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

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

# Install vim and cURL in Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

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

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

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



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

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

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

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

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

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

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

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

Step 5 Uploading an Image to SWR

1. Log in to the SWR console and select 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 following command to tag the uploaded image:

```
#Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

   b. Run the following command to upload the image:

```
#Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

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

Step 6 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console and check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization 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**. The training job list is displayed by default.
3. On the **Create Training Job** page, set required parameters and click **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Uploading an Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/pytorch/demo-code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py**. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Resource Pool**: **Public resource pools**
   - **Resource Type**: Select **CPU** or **GPU**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: Set this parameter to the OBS path for storing training logs, for example, **obs://test-modelarts/pytorch/log/**.
4. Check the parameters of the training job and click **Submit**.
5. Wait until the training job is completed.

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

   Figure 10-18 Run logs of training jobs with GPU specifications

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

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

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

NOTE

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

Scenarios

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

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

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

Procedure

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

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

Prerequisites

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

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 10-26** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

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

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

Table 10-26 Folder to create

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

Name	Description
obs://test-modelarts/mpi/log/	Stores training log files.

Step 2 Preparing Script Files and Uploading Them to OBS

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

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

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

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

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

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

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

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

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

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

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

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

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

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

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

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



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

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

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

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

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

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

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

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

    RET_CODE=$?

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

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

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

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

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

exit $RET_CODE
```


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

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

```
import os
import socket

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

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

Step 3 Preparing an Image Server

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

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

Step 4 Creating a Custom Image

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

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

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

1. Install Docker.

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

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

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

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

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

   The following information is displayed:

   ```
   Engine:
    Version:          18.09.0
   ```

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

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


```
mkdir -p context
```

4. Download the Miniconda3 installation file.

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

5. Download the openmpi 3.0.0 installation file.

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

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

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

7. Write the Dockerfile of the container image.

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

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

# Basic container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

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

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

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

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

# Install vim, cURL, net-tools, and the SSH tool in Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# Install the Open MPI 3.0.0 file written using Horovod v0.22.1.
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version
```



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

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

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

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

# Configure sshd to support SSH password-free login.
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

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

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

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

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

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

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

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

Step 5 Uploading an Image to SWR

1. Log in to the SWR console and select 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. 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 following command to tag the uploaded image:
      ```
      #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
      sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
      ```
   b. Run the following command to upload the image:
      ```
      #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
      sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
      ```
6. After the image is uploaded, choose **My Images** on the left navigation pane of the SWR console to view the uploaded custom images.

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

Step 6 Creating a Training Job on ModelArts

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

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

procedure and selected resources. After the training job is executed, the log similar to the following is output.

Figure 10-19 Run logs of worker-0 with one compute node and GPU specifications

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

Set **Compute Nodes** to **2** and run the training job. **Figure 10-20** and **Figure 10-21** show the log information.

Figure 10-20 Run logs of worker-0 with two compute nodes and GPU specifications

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

Figure 10-21 Run logs of worker-1 with two compute nodes and GPU specifications

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

10.4.6 Creating a Custom Training Image (Tensorflow + GPU)

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

NOTE

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

Scenario

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

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

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

Procedure

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

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

Prerequisites

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

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 10-27** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

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

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

Table 10-27 Required OBS folders

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

Step 2 Creating a Dataset and Uploading It to OBS

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

Step 3 Preparing the Training Script and Uploading It to OBS

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

mnist.py is as follows:

```
import argparse
import tensorflow as tf

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

mnist = tf.keras.datasets.mnist

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

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

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

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

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

Step 4 Preparing a Server

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

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Set **CPU Architecture** to **x86** and **Image** to **Public image**. Ubuntu 18.04 images are recommended.

Step 5 Creating a Custom Image

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

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

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

1. Install Docker.

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

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

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

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

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

   The following information is displayed:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTE

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

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

   ```
   mkdir -p context
   ```

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

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

   NOTE

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

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

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

6. Download the Miniconda3 installation file.

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

7. Write the container image Dockerfile.

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

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

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04 AS builder

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

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

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

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

# Install the TensorFlow .whl file using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

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

# Create the container image.
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04

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

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



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

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

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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

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

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

   Go to **Linux Drivers**. In the **Download** tab, set **Version** to **5.4-3.5.8.0-LTS**, **OS Distribution Version** to **Ubuntu 18.04**, **Architecture** to **x86_64**, and download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

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

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

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

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

    The following log shows that the image has been created.

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

Step 6 Uploading the Image to SWR

1. Log in to the SWR console and select a region. It must share the same region with ModelArts. Otherwise, the image cannot be selected.
2. Click **Create Organization** in the upper right corner and enter an organization name. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.
3. Click **Generate Login Command** in the upper right corner to obtain the login command. In this example, the temporary login command is copied.
4. Log in to the local environment as user **root** and enter the copied temporary login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.

```
# Replace the region, domain, as well as organization name deep-learning with the actual values.
sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
```


b. Run the following command to upload the image:

```
# Replace the region, domain, as well as organization name deep-learning with the actual values.
sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
```

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

Step 7 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization 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**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Creating a Custom Image**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/tensorflow/code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/code** directory of the training container. **code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **python ${MA_JOB_DIR}/code/mnist.py**. **code** (customizable) is the last-level directory of the OBS path.
   - **Training Input**: Click **Add Training Input**. Enter **data_path** for the name, select the OBS path to **mnist.npz**, for example, **obs://test-modelarts/tensorflow/data/mnist.npz**, and set **Obtained from** to **Hyperparameters**.
   - **Resource Pool**: Select **Public resource pools**.
   - **Resource Type**: Select **GPU**.
   - **Compute Nodes**: Enter **1**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mindspore-gpu/log/**
4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

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

Figure 10-22 Run logs of training jobs with GPU specifications

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

10.4.7 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 10-28**.

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

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

Table 10-28 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. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:
```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

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

6. 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 https://repo.huaweicloud.com/repository/conf/Ubuntu-Ports-bionic.list
```

> **NOTE**
>
> To obtain the APT source file, go to Huawei Mirrors at **https://mirrors.huaweicloud.com/home** and search for **Ubuntu-Ports**.

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 Huawei  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 Huawei 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 Huawei  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 Management** > **Training Jobs**.
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 10-23 Runtime logs of a training job powered by Ascend resources in a dedicated resource pool

```
Ascend Envs
------
JOB_ID:  modelarts-job-2436291a-8543-4ab8-84ad-2dda8f1e4f5c
RANK_TABLE_FILE:  /home/ma-user/modelarts/rank_table/jobstart_hccl.json
RANK_SIZE:  1
ASCEND_DEVICE_ID:  0
DEVICE_ID:  0
RANK_ID:  0
------
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [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_V_1_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_fmk_process()
    self.wait_fmk_process_end(base_period)
    log.info('End destroy training processes')

def send_sigterm_to_fmk_process(self):
    # send SIGTERM to fmk processes (and process group)
    for r_index in range(len(self.fmk_processes) - 1, -1, -1):
        fmk = self.fmk[r_index]
        fmk_process = self.fmk_processes[r_index]
        if fmk_process.poll() is not None:
```



```
                log.info('proc-rank-%s-device-%s (pid: %d) has exited before receiving the term signal',
                        fmk.rank_id, fmk.device_id, fmk_process.pid)
                del self.fmk_processes[r_index]
                del self.fmk[r_index]

            try:
                os.killpg(fmk_process.pid, signal.SIGTERM)
            except ProcessLookupError:
                pass

    def wait_fmk_process_end(self, base_period):
        test_cnt = 0
        period = base_period
        while len(self.fmk_processes) > 0 and test_cnt < self.max_test_proc_cnt:
            for r_index in range(len(self.fmk_processes) - 1, -1, -1):
                fmk = self.fmk[r_index]
                fmk_process = self.fmk_processes[r_index]
                if fmk_process.poll() is not None:
                    log.info('proc-rank-%s-device-%s (pid: %d) has exited',
                            fmk.rank_id, fmk.device_id, fmk_process.pid)
                    del self.fmk_processes[r_index]
                    del self.fmk[r_index]
            if not self.fmk_processes:
                break

            time.sleep(period)
            period *= 2
            test_cnt += 1

        if len(self.fmk_processes) > 0:
            for r_index in range(len(self.fmk_processes) - 1, -1, -1):
                fmk = self.fmk[r_index]
                fmk_process = self.fmk_processes[r_index]
                if fmk_process.poll() is None:
                    log.warn('proc-rank-%s-device-%s (pid: %d) has not exited within the max waiting time,
'
                            'send kill signal',
                            fmk.rank_id, fmk.device_id, fmk_process.pid)
                    os.killpg(fmk_process.pid, signal.SIGKILL)
```

- fmk.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
```


10.5 Creating a Custom Image for Inference

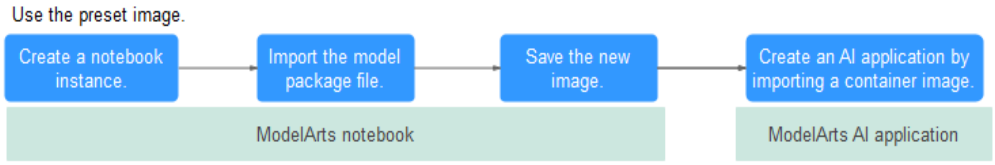
10.5.1 Creating a Custom Image for an AI Application

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

Procedure

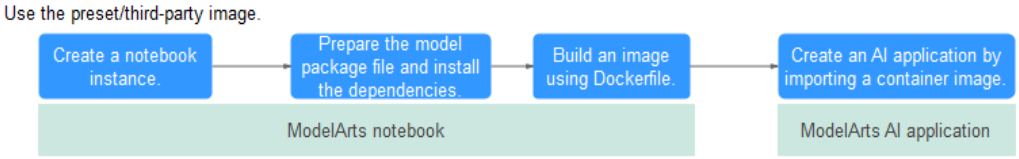
Scenario 1: The environment software of the preset image meets the requirements. You only need to import a model package to create an AI application by saving the image. For details, see **Creating a Custom Image in a Notebook Instance Using the Image Saving Function**.

Figure 10-24 Creating a custom image for an AI application (scenario 1)



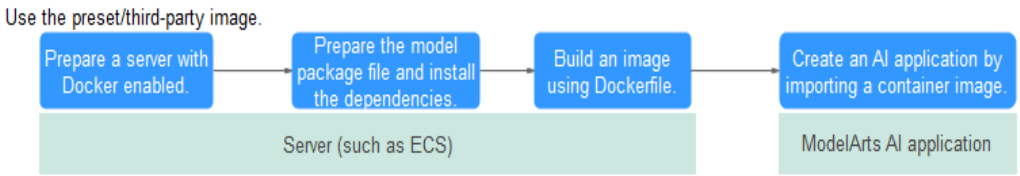
Scenario 2: The preset image does not meet the software environment requirements. You need to import a model package and create the image using Dockerfile. For details, see **Creating a Custom Image in a Notebook Instance Using Dockerfile**.

Figure 10-25 Creating a custom image for an AI application (scenario 2)



Scenario 3: The preset does not meet the software environment requirements. You need to import a model package. The new image is larger than 35 GB and needs to be created on a server such as ECS. For details, see **Creating a Custom Image on ECS**.

Figure 10-26 Creating a custom image for an AI application (scenario 3)



Constraints

- No malicious code is allowed.
- The image for creating an AI application cannot be larger than 50 GB.
- For AI applications 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 an AI application in asynchronous mode.

Specifications for Custom Images

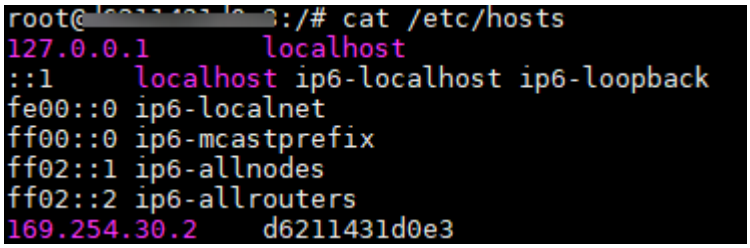
- **External APIs**

  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 10-27 Example of obtaining **listen_ip**

  – 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 10-29 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
  ```


10.5.2 Creating a Custom Image in a Notebook Instance Using the Image Saving Function

Scenario

This section describes how to import a local model package to ModelArts notebook for debugging and saving, and then deploy the saved image for inference.

Procedure:

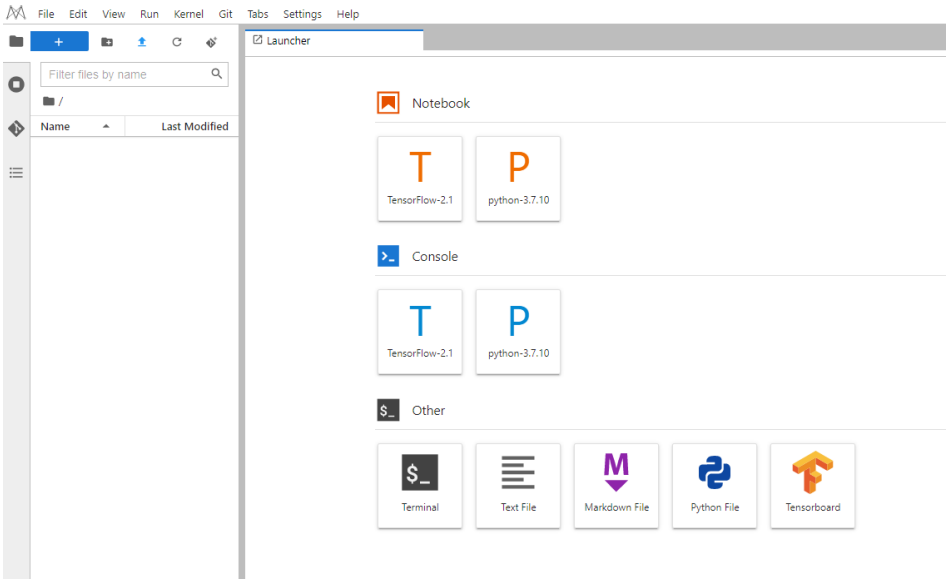
1. **Step1 Copying a Model Package in a Notebook Instance**
2. **Step2 Debugging a Model in a Notebook Instance**
3. **Step3 Saving an Image in a Notebook Instance**
4. **Step4 Using the Saved Image for Inference Deployment**

Step1 Copying a Model Package in a Notebook Instance

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Click **Create Notebook** in the upper right corner. Configure the parameters on the displayed page.
   a. Configure basic information of the notebook instance, including its name, description, and auto stop status.
   b. Select an image and configure resource specifications for the instance.
      - **Image**: Select the **pytorch1.8-cuda10.2-cudnn7-ubuntu18.04** image. For details about the image, see **Engine Version 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**.
      - **Resource Type**: Select a public resource pool or a dedicated resource pool. A public resource pool is used as an example.
      - **Type**: **GPU** is recommended.
      - **Flavor**: GP Tnt004 is recommended.
3. Click **Next**. Confirm the information and click **Submit**.

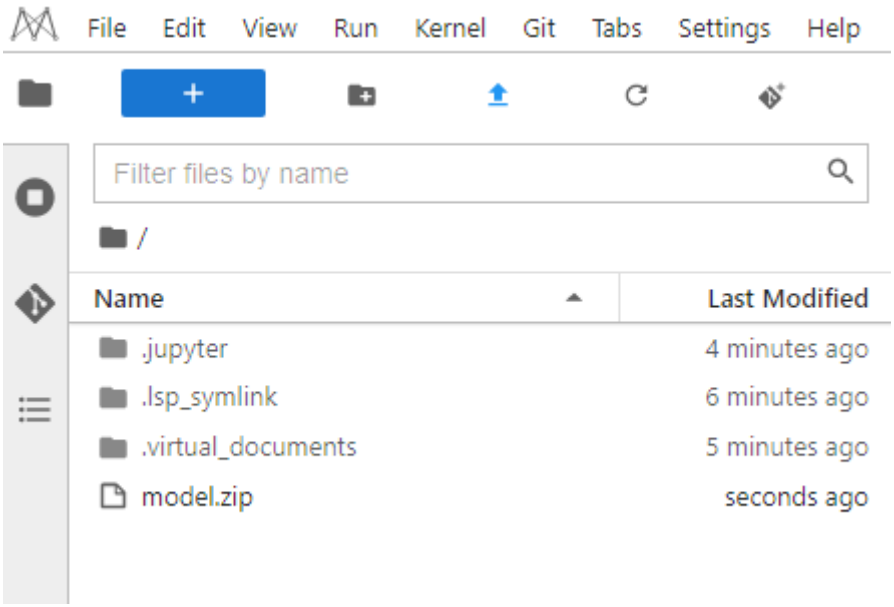
   Switch to the notebook instance list. The notebook instance is being created. It will take several minutes.
4. Wait until the notebook status changes to **Running**. Then, locate the notebook in the list and click **Open** in the **Operation** column. The JupyterLab Launcher page is displayed.

Figure 10-28 JupyterLab Launcher



5. Click to upload the model package file to the notebook instance. The default working directory of the instance is **/home/ma-user/work/**. Prepare the model package file. For details, see **Sample Model Package File**.

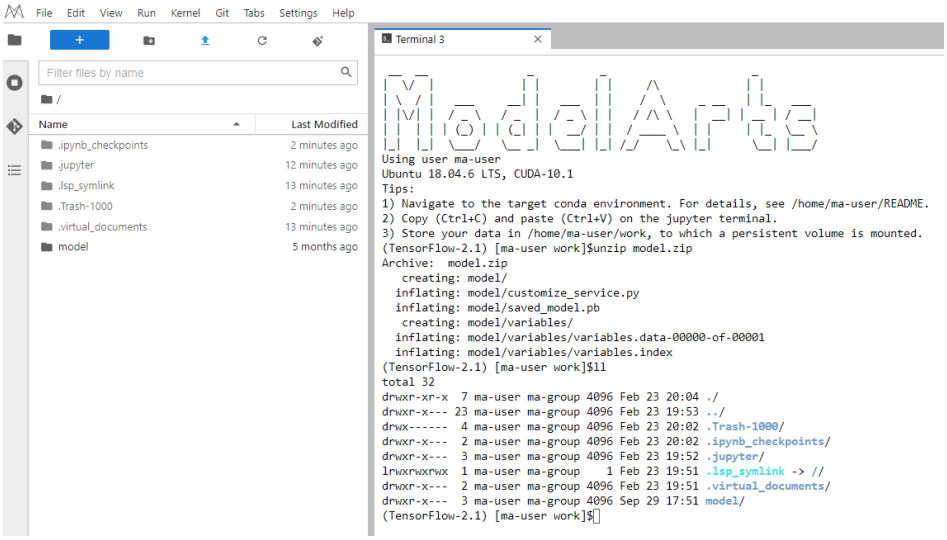
Figure 10-29 Uploading a model package



6. Start the Terminal. Decompress **model.zip** and delete it.

```
# Decompress the ZIP file.
unzip model.zip
```


Figure 10-30 Decompressing **model.zip** on the Terminal



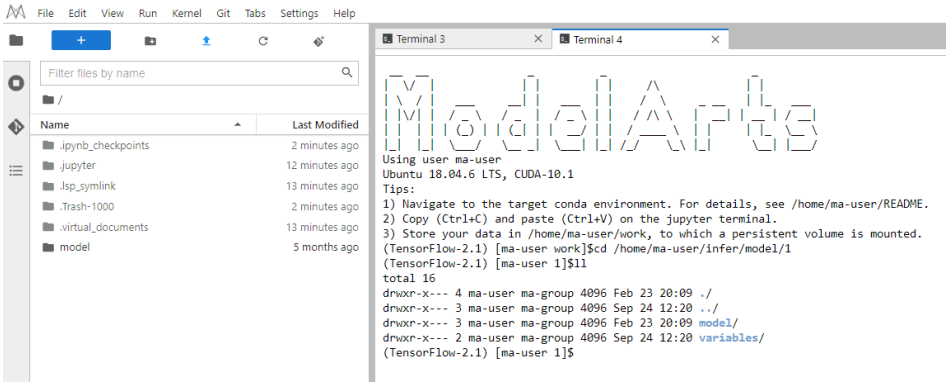
7. In the **Terminal** tab, run the copy command.

```
cp -rf model/*  /home/ma-user/infer/model/1
```

Check whether the image file is copied.

```
cd /home/ma-user/infer/model/1
ll
```

Figure 10-31 Image file copied

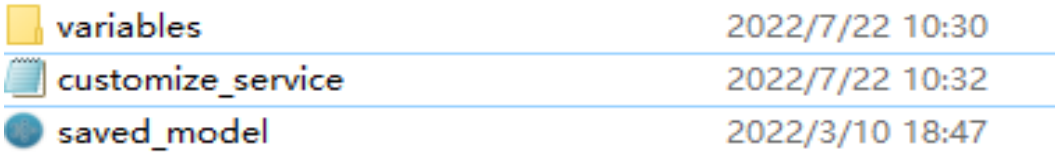


Sample Model Package File

A model file in the model package file **model.zip** must be prepared by yourself. The following uses a handwritten digit recognition model as an example.

The inference script file **customize_service.py** must be available in the model directory for model pre-processing and post-processing.

Figure 10-32 Model directory of the inference model



For details about the inference script **customize_service.py**, see **Specifications for Writing a Model Inference Code File customize_service.py**.

The content of the **customize_service.py** file used in this case is as follows:

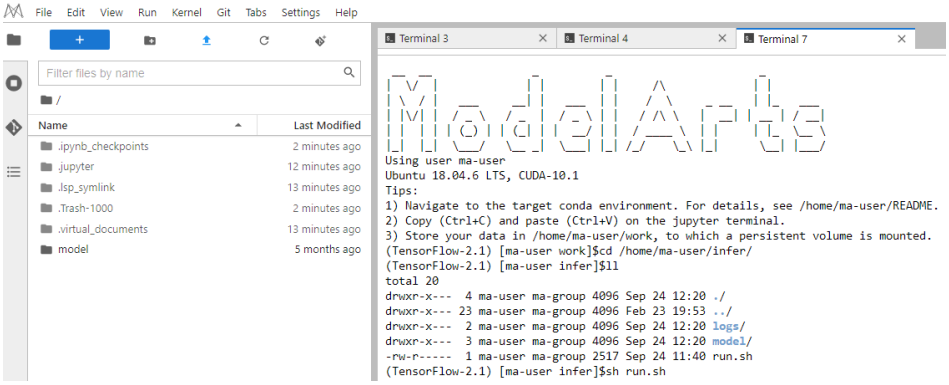
```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService


class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 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())
        }
```


Step2 Debugging a Model in a Notebook Instance

1. In a new **Terminal** tab, go to the **/home/ma-user/infer/** directory, run the **run.sh** script, and predict the model. In a base image, **run.sh** is used as the boot script by default. The start command is as follows:

```
sh  run.sh
```

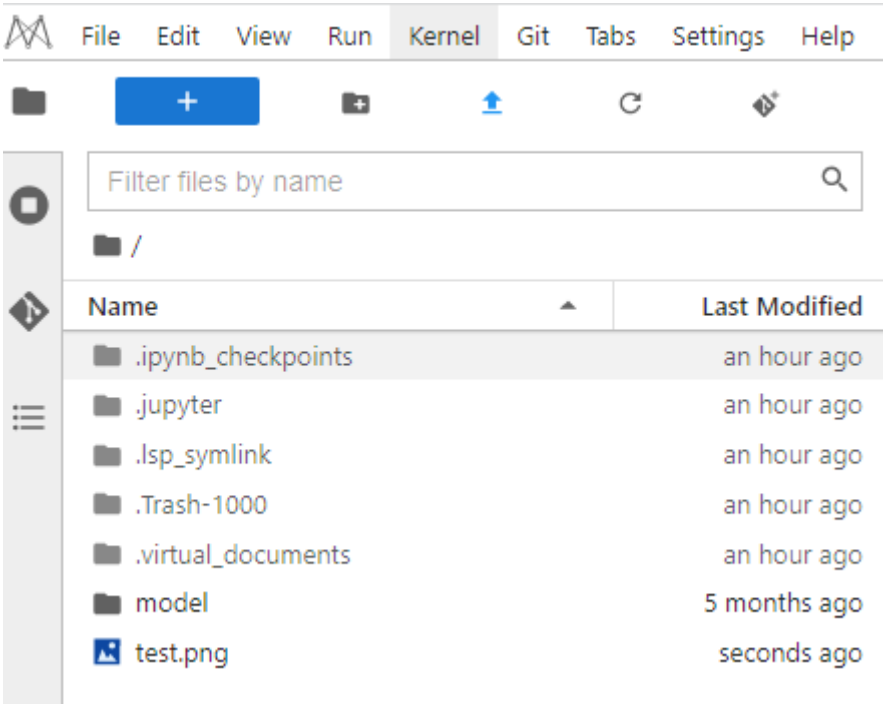
Figure 10-33 Running the boot script



2. Upload an image with a handwritten digit to the notebook instance for prediction.

Figure 10-34 Handwritten digit

Figure 10-35 Uploading an image for prediction



3. Open a new Terminal and run the following command for prediction:

```
curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/
```

Figure 10-36 Prediction

If the model file or inference script file is modified during debugging, restart the **run.sh** script. To do so, run the following command to stop Nginx and then run the **run.sh** script:

```
# Obtain the Nginx process.
ps -ef |grep nginx
# Stop all Nginx-related processes.
```



```
kill -9 {Process ID}
# Execute run.sh.
sh run.sh
```

You can also run the **pkill nginx** command to stop all Nginx processes.

```
# Stop all Nginx processes.
pkill nginx
# Execute run.sh.
sh run.sh
```

Figure 10-37 Restarting **run.sh**

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user   6474     1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user   6476  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   6477  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   7925  1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

Step3 Saving an Image in a Notebook Instance

NOTE

A running notebook instance must be available.

1. Locate the target notebook instance in the list and choose **More** > **Save Image** in the **Operation** column.
2. In the displayed **Save Image** dialog box, configure the parameters. Then, click **OK**.

   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, which will take about 5 minutes. During this period of time, do not perform any operations on the instance. (You can still perform operations on the accessed JupyterLab page and local IDE.)

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 image name to view its details.

Step4 Using the Saved Image for Inference Deployment

Import the custom image debugged in **Step2 Debugging a Model in a Notebook Instance** to AI applications and deploy it as a real-time service.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Applications**. Click **Create Applications** on the displayed page.
2. Configure the parameters for the AI application.
   - **Meta Model Source**: Select **Container image**.
   - **Container Image Path**: Click to select an image file. For details about the path, see the SWR address in **5**.
   - **Container API**: Select **HTTPS**.
   - **host**: Set to **8443**.
   - **Deployment Type**: Select **Real-Time Services**.
3. Enter the boot command.
   ```
   sh /home/ma-user/infer/run.sh
   ```
4. Enable APIs, edit the API, and click **Save**. Specify a file as the input. The following shows a code example.

Figure 10-38 API definition

The API definition is as follows:

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
```

```
            "result": {
               "type": "integer"
            }
         }
      }
   }
}]
```

NOTICE

After enabling this function, you can edit RESTful APIs to define the AI application input and output formats.

- If you edit APIs when creating an AI application, the system will automatically identify the prediction type after the created AI application is deployed.
- If you do not edit APIs when creating an AI application, you will be required to select a request type for prediction after the created AI application is deployed. The request type can be **application/json** or **multipart/form-data**. Select a proper type based on the meta model.
  - If you select **application/json**, enter code for text prediction.
  - If you select **multipart/form-data**, configure the request parameter. The request parameter value is the **KEY** value in the **Body** tab **when GUI-based software (Postman as an example) is used for prediction**. It is also the parameter name in the **prediction request sent by running the cURL command**.

5. After the APIs are configured, click **Create now**. Wait until the AI application runs properly.
6. Locate the created AI application in the list and click **Deploy** in the **Operation** column. In the displayed version list, locate the target version and click **Real-Time Services** in the **Operation** column.
7. On the **Deploy** page, configure the key parameters as follows:

   Name: Enter a custom real-time service name or use the default name.

   Resource Pool: Select a public resource pool.

   AI Application Source and **AI Application and Version**: The AI application and version will be automatically selected.

   Specifications: Choose **CPU: 2 vCPUs 8GB**.

   Retain default settings for other parameters.
8. After configuring the parameters, click **Next**, confirm parameter settings, and click **Submit**.
9. In the navigation pane on the left, choose **Service Deployment** > **Real-Time Services**. When the service status changes to **Running**, the service is deployed. Click **Predict** in the **Operation** column. The **Prediction** page on the service details page is displayed. Upload an image for prediction.

10.5.3 Creating a Custom Image in a Notebook Instance Using Dockerfile

Scenario

For AI engines that are not supported by ModelArts, you can import the models you compile to ModelArts from custom images for creating AI applications.

This section describes how to use a base image in ModelArts notebook to create an inference image, use the image to create an AI application, and deploy the application as a real-time service.

Procedure:

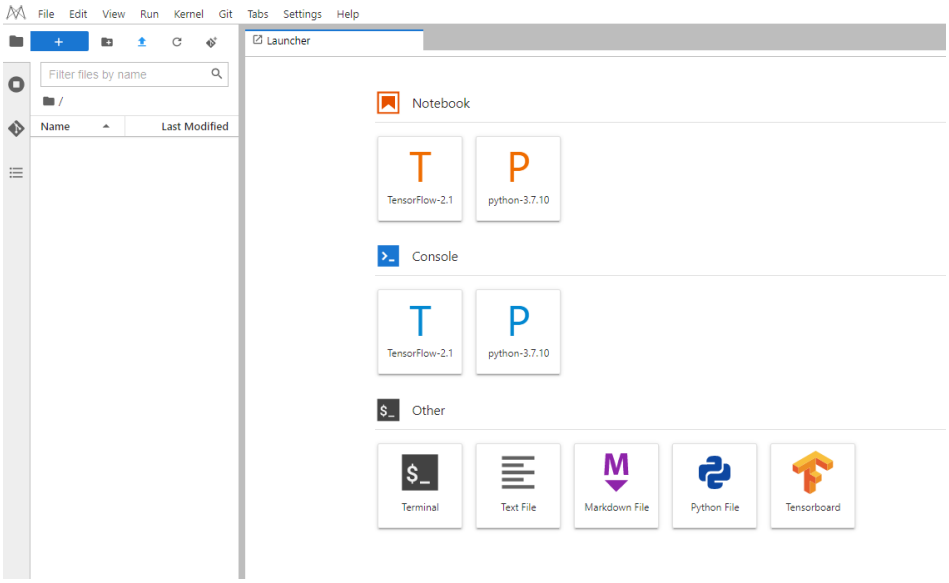
1. **Step 1 Creating an Image in a Notebook Instance**. For details, see **Specifications for Custom Images**.
2. **Step 2 Registering the Image in Image Management**.
3. **Step 3 Changing and Debugging the Image in a Notebook Instance**.
4. **Step 4 Using a Debugged Image for Inference Deployment**.

Step 1 Creating an Image in a Notebook Instance

This section uses a TensorFlow base image provided in ModelArts as an example to create an image in ModelArts notebook.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Permission Management**. Check whether the access authorization has been configured. If not, configure access authorization by referring to **Configuring Agency Authorization for ModelArts with One Click**.
2. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create Notebook** in the upper right corner. Configure the parameters on the displayed page.
   a. Configure basic information of the notebook instance, including its name, description, and auto stop status.
   b. Select an image and configure resource specifications for the instance.
      - **Image**: Select a public image that can run on CPUs, for example, **pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**.
      - **Resource Type**: Select a public resource pool or a dedicated resource pool. A public resource pool is used as an example.
      - **Type**: **GPU** is recommended.
      - **Flavor**: GP Tnt004 is recommended.
4. Click **Next**. Confirm the information and 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**.
5. Access the running notebook instance.

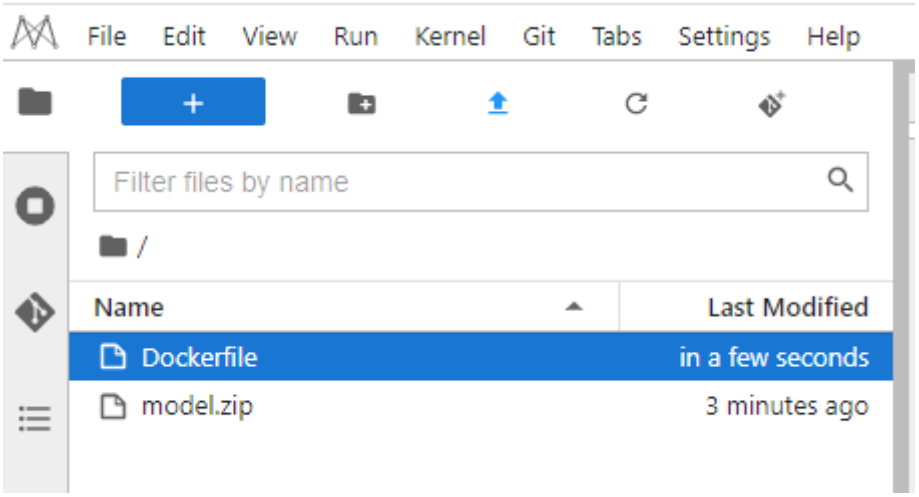
Figure 10-39 Accessing a notebook instance



6. Click to upload the Dockerfile and model package file to the notebook instance. The default working directory of the instance is **/home/ma-user/work/**.

   For details about the Dockerfile, see **Dockerfile Template**. Prepare the model package file. For details, see **Sample Model Package File**.

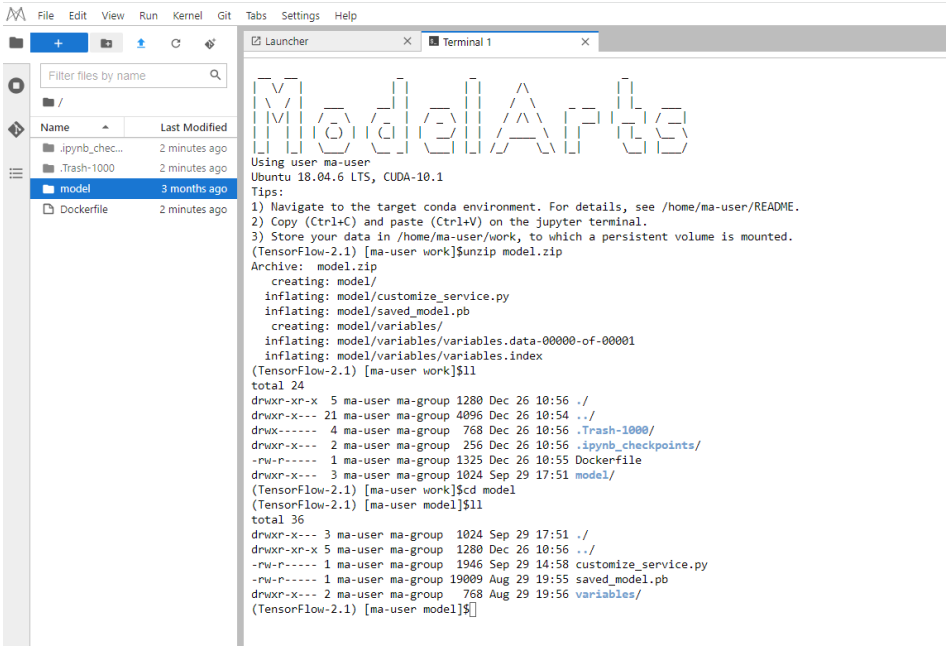
Figure 10-40 Uploading Dockerfile and model package file



7. Start the Terminal. Decompress **model.zip** and delete it.

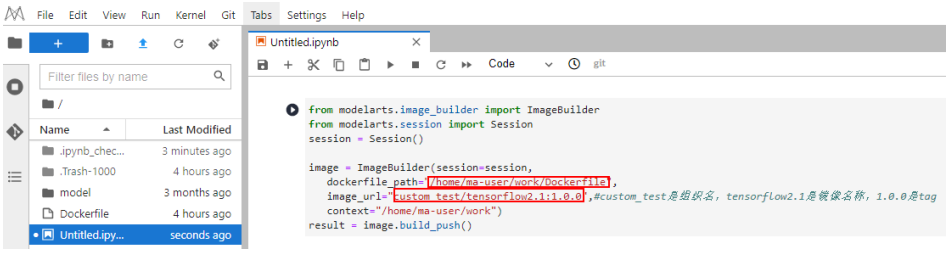
```
# Decompress the ZIP file.
unzip model.zip
```


Figure 10-41 Decompressing **model.zip** on the Terminal



8. Open a new IPYNB file, start the image creation script, and specify the paths to the Dockerfile and image.

Figure 10-42 Starting the image creation script

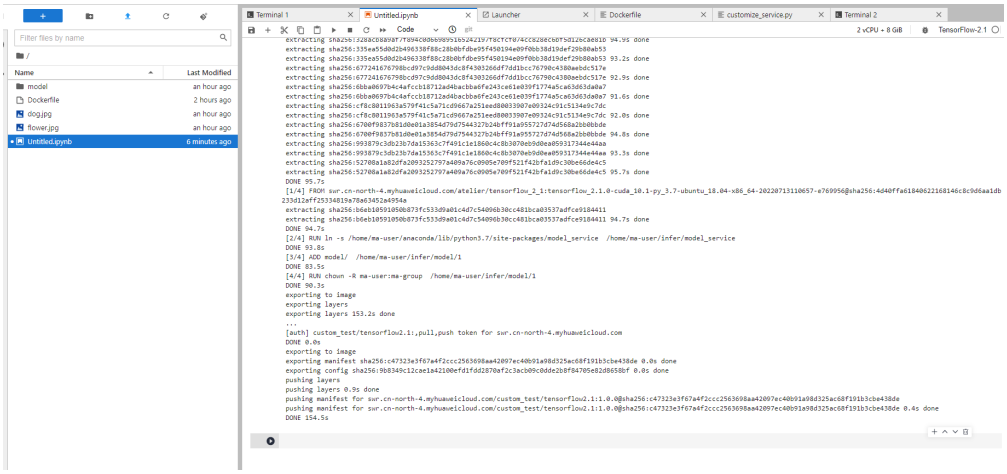


The image creation script is as follows:

```
from modelarts.image_builder import ImageBuilder
from modelarts.session import Session
session = Session()

image = ImageBuilder(session=session,
  dockerfile_path="/home/ma-user/work/Dockerfile",
  image_url="custom_test/pytorch1.8:1.0.0",# custom_test is the organization name, pytorch1.8 is the image name, and 1.0.0 is the tag.
  context="/home/ma-user/work")
result = image.build_push()
```

Wait until the image is created. The image will be automatically pushed to SWR.

Figure 10-43 Image being created



Dockerfile Template

The following provides a sample Dockerfile, which can be saved as another Dockerfile. For details about the available base images, see **Preset Dedicated Images for Inference**.

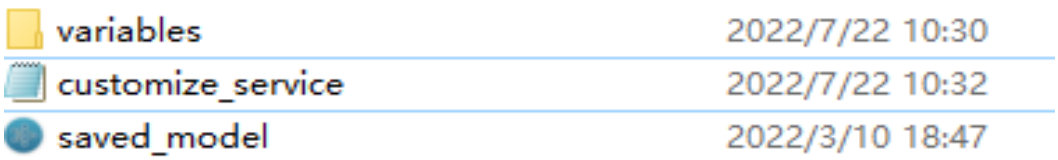
```
FROM swr.cn-north-4.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817

# Create a soft link from /home/ma-user/anaconda/lib/python3.7/site-packages/model_service to /home/ma-user/infer/model_service, which is the built-in inference framework code directory.
# if the installed python version of this base image is python3.8, you should create a soft link from '/home/ma-user/anaconda/lib/python3.8/site-packages/model_service' to '/home/ma-user/infer/model_service'.
USER root
RUN ln -s /home/ma-user/anaconda/lib/python3.7/site-packages/model_service  /home/ma-user/infer/model_service
USER ma-user

# here we supply a demo, you can change it to your own model files
ADD model/  /home/ma-user/infer/model/1
USER root
RUN chown -R ma-user:ma-group  /home/ma-user/infer/model/1
USER ma-user

# default MODELARTS_SSL_CLIENT_VERIFY switch is "true". In order to debug, we set it to be "false"
ENV MODELARTS_SSL_CLIENT_VERIFY="false"

# change your port and protocol here, default is 8443 and https
# ENV MODELARTS_SERVICE_PORT=8080
# ENV MODELARTS_SSL_ENABLED="false"

# add pip install here
# RUN pip install numpy==1.16.4
# RUN pip install -r requirements.txt

# default cmd, you can chage it here
# CMD sh /home/ma-user/infer/run.sh
```

Sample Model Package File

A model file in the model package file **model.zip** must be prepared by yourself. The following uses a handwritten digit recognition model as an example.

The inference script file **customize_service.py** must be available in the model directory for model pre-processing and post-processing.

Figure 10-44 Model directory of the inference model



For details about the inference script **customize_service.py**, see **Specifications for Writing a Model Inference Code File customize_service.py**.

The content of the **customize_service.py** file used in this case is as follows:

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService


class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 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
```



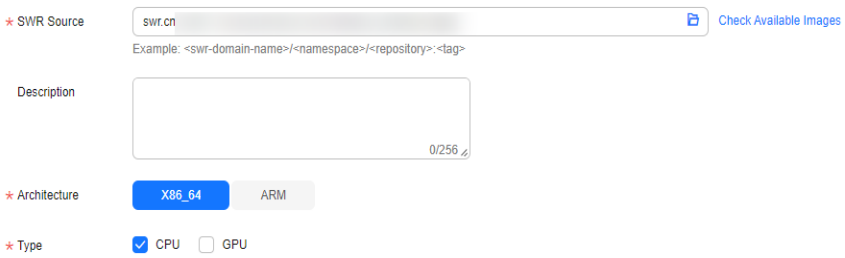
```
def _inference(self, data):

    return self.predict(data)

def _postprocess(self, data):

    return {
        "result": int(data["output"].numpy()[0].argmax())
    }
```

Step 2 Registering the Image in Image Management

Register the custom image created in **Step 1 Creating an Image in a Notebook Instance** in ModelArts image management for future use.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Image Management**. Then, click **Register** on the displayed page.
2. Specify the image source, select the architecture and type, and click **Register**.
   - **SWR Source**: The image source is **swr.cn-north-4-myhuaweicloud.com/custom_test/tensorflow2.1:1.0.0**, where **custom_test/tensorflow2.1:1.0.0** is the image path set in the image creation script in **8**.
   - **Architecture**: Select **X86_64**.
   - **Type**: Select **CPU**.

   Figure 10-45 Registering an image

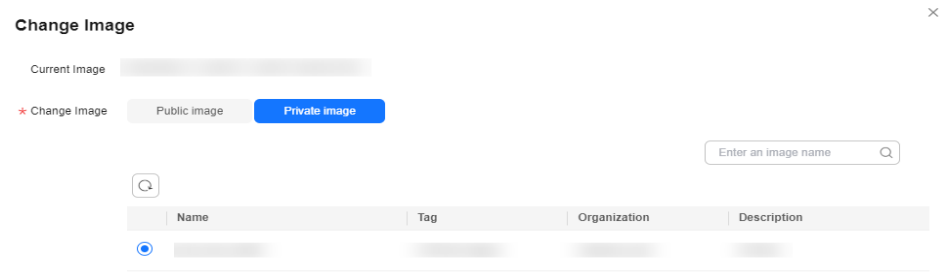
3. View the registered image on the **Image Management** page.

Step 3 Changing and Debugging the Image in a Notebook Instance

Debug the created custom image for inference. Then, import the debugged image to ModelArts AI applications and deploy it as a real-time service.

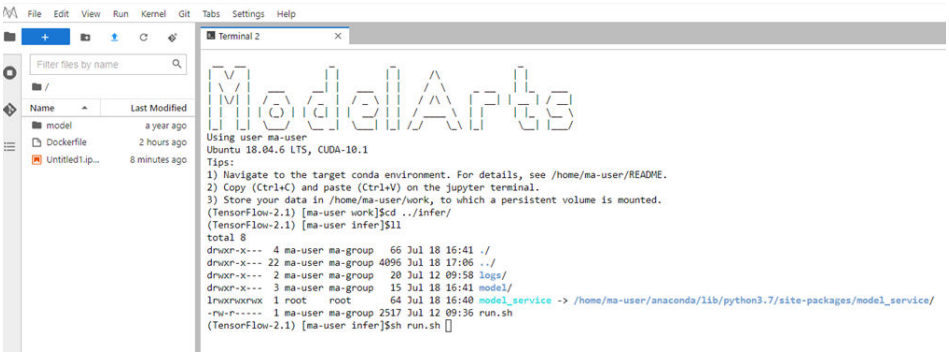
1. Log in to the ModelArts console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**. Stop the notebook instance created in **Step 1 Creating an Image in a Notebook Instance**.
2. Locate the target notebook instance in the list and choose **More** > **Change Image** in the **Operation** column. In the displayed dialog box, set **Change Image** to **Private image**, and select the image registered in **Step 2 Registering the Image in Image Management**, as shown in **Figure 10-46**.

Figure 10-46 Changing an Image



3. Start the notebook instance and access it. Go to the Terminal page, run the boot script **run.sh** in the working directory, and run the model for prediction. In a base image, **run.sh** is used as the boot script by default.

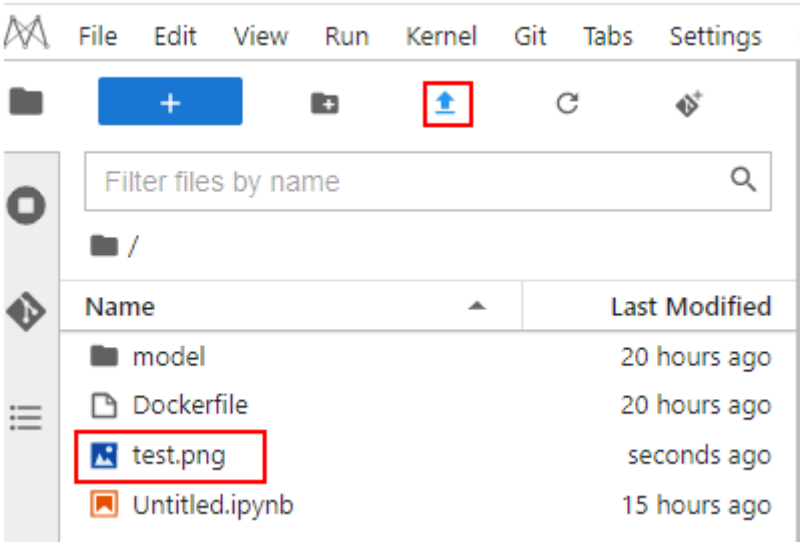
Figure 10-47 Running the boot script



4. Upload an image with a handwritten digit to the notebook instance for prediction.

Figure 10-48 Handwritten digit

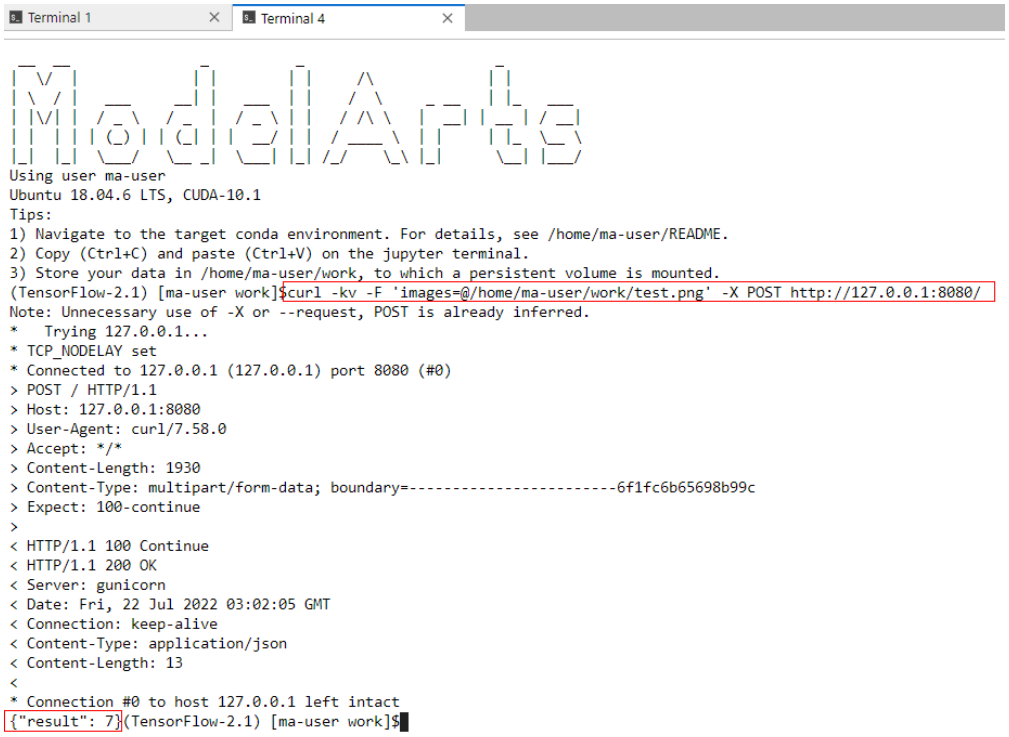
Figure 10-49 Uploading an image for prediction



5. Open a new Terminal and run the following command for prediction:

```
curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/
```

Figure 10-50 Prediction



If the model file or inference script file is modified during debugging, restart the **run.sh** script. To do so, run the following command to stop Nginx and then run the **run.sh** script:

```
# Obtain the Nginx process.
ps -ef |grep nginx
# Stop all Nginx-related processes.
kill -9 {Process ID}
# Execute run.sh.
sh run.sh
```


You can also run the **pkill nginx** command to stop all Nginx processes.

```
# Stop all Nginx processes.
pkill nginx
# Execute run.sh.
sh run.sh
```

Figure 10-51 Restarting **run.sh**

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user   6474     1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user   6476  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   6477  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   7925  1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

Step 4 Using a Debugged Image for Inference Deployment

Import the custom image debugged in **Step 3 Changing and Debugging the Image in a Notebook Instance** to AI applications and deploy it as a real-time service.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **AI Applications**. Click **Create Applications** on the displayed page.
2. Configure the parameters for the AI application.
   - **Meta Model Source**: Select **Container image**.
   - **Container Image Path**: Click and select the created custom image.
   - **Container API**: Select **HTTPS**.
   - **host**: Set to **8443**.
   - **Deployment Type**: Select **Real-Time Services**.
3. Enable APIs, edit the API, and click **Save**. Specify a file as the input. The following shows a code example.

Figure 10-52 API definition

The API definition is as follows:

```
[{
 "url": "/",
 "method": "post",
 "request": {
  "Content-type": "multipart/form-data",
  "data": {
   "type": "object",
   "properties": {
    "images": {
     "type": "file"
    }
   }
  }
 },
 "response": {
  "Content-type": "applicaton/json",
  "data": {
   "type": "object",
   "properties": {
    "result": {
     "type": "integer"
    }
   }
  }
 }
}]
```

NOTICE

After enabling this function, you can edit RESTful APIs to define the AI application input and output formats.

- If you edit APIs when creating an AI application, the system will automatically identify the prediction type after the created AI application is deployed.
- If you do not edit APIs when creating an AI application, you will be required to select a request type for prediction after the created AI application is deployed. The request type can be **application/json** or **multipart/form-data**. Select a proper type based on the meta model.
  - If you select **application/json**, enter code for text prediction.
  - If you select **multipart/form-data**, configure the request parameter. The request parameter value is the **KEY** value in the **Body** tab **when GUI-based software (Postman as an example) is used for prediction**. It is also the parameter name in the **prediction request sent by running the cURL command**.

4. After the APIs are configured, click **Create now**. Wait until the AI application runs properly.
5. Locate the created AI application in the list and click **Deploy** in the **Operation** column. In the displayed version list, locate the target version and click **Real-Time Services** in the **Operation** column.
6. On the **Deploy** page, configure the key parameters as follows:

   Name: Enter a custom real-time service name or use the default name.

   Resource Pool: Select a public resource pool.

   AI Application Source and **AI Application and Version**: The AI application and version will be automatically selected.

Specifications: Select specifications with limited time offer.

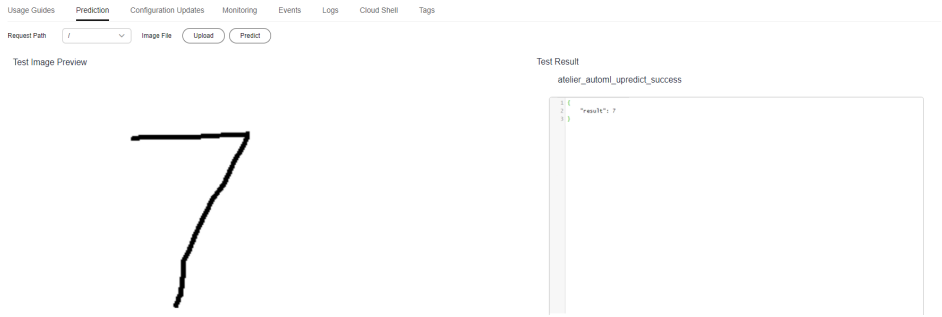
Retain default settings for other parameters.

NOTE

If free specifications have been sold out, use charged CPU specifications instead. Resource fees are displayed on the GUI.

7. After configuring the parameters, click **Next**, confirm parameter settings, and click **Submit**.
8. In the navigation pane on the left, choose **Service Deployment** > **Real-Time Services**. When the service status changes to **Running**, the service is deployed. Click **Predict** in the **Operation** column. The **Prediction** page on the service details page is displayed. Upload an image for prediction.

Figure 10-53 Prediction



10.5.4 Creating a Custom Image on ECS

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

The procedure is as follows:

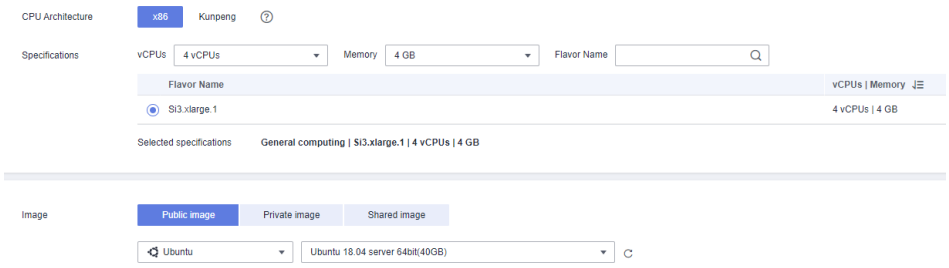
1. **Building an Image Locally**: Create a custom image package locally. For details, see **Specifications for Custom Images**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Creating an AI Application Using a Custom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

Building an Image Locally

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

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. When creating the ECS, select an Ubuntu 18.04 public image.

Figure 10-54 Creating an ECS using an x86 public image



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

   The file structure is as follows:
   ```
   self-define-images/
     --Dockerfile
     --test_app.py
   ```

   - **Dockerfile**
     ```
     From ubuntu:18.04
     # Configure the Huawei Cloud source and install Python, Python3-PIP, and Flask.
     RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
       sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
     && \
       sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
     && \
       apt-get update && \
       apt-get install -y python3 python3-pip && \
       pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
     repository/pypi/simple  Flask

     # Copy the application code to the image.
     COPY test_app.py /opt/test_app.py

     # Specify the boot command of the image.
     CMD python3  /opt/test_app.py
     ```
   - **test_app.py**
     ```
     from flask import Flask, request
     import json
     app = Flask(__name__)

     @app.route('/greet', methods=['POST'])
     def say_hello_func():
         print("----------- in hello func ----------")
         data = json.loads(request.get_data(as_text=True))
         print(data)
         username = data['name']
         rsp_msg = 'Hello, {}!'.format(username)
         return json.dumps({"response":rsp_msg}, indent=4)

     @app.route('/goodbye', methods=['GET'])
     def say_goodbye_func():
         print("----------- in goodbye func ----------")
         return '\nGoodbye!\n'
     ```



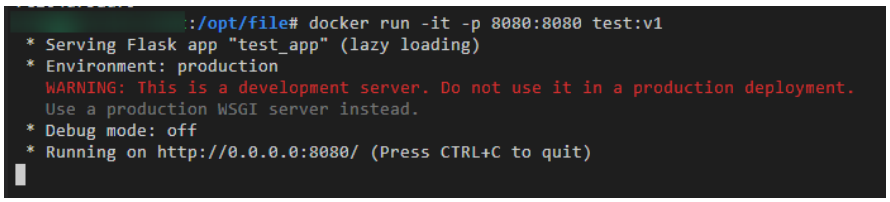
```
@app.route('/', methods=['POST'])
def default_func():
    print("---------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. Switch to the **self-define-images** folder and run the following command to create custom image **test:v1**:
   ```
   docker build -t test:v1 .
   ```
5. Run **docker images** to view the custom image you have created.

Verifying the Image Locally and Uploading It to SWR

1. Run the following command in the local environment to start the custom image:
   ```
   docker run -it -p 8080:8080 test:v1
   ```

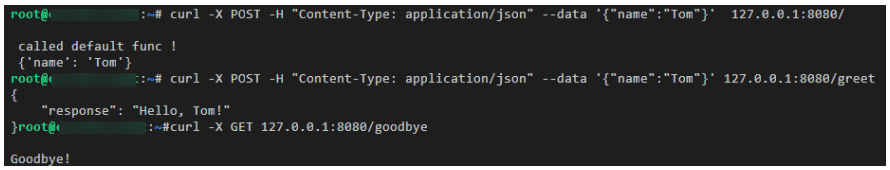
   Figure 10-55 Starting a custom image

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 10-56 Testing API functions

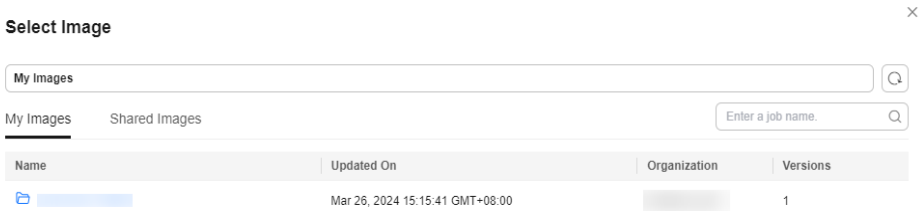
3. Upload the custom image to SWR.
4. After the custom image is uploaded, view the uploaded image on the **My Images** > **Private Images** page of the SWR console.

Creating an AI Application 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.

Figure 10-57 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 AI application 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**.

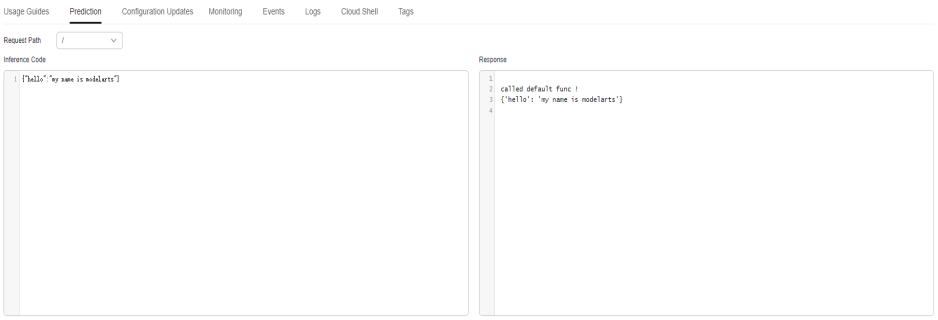
  The configuration file is as follows:

```
[{
      "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 AI Application 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.

Figure 10-58 Accessing the real-time service



11 Resource Monitoring

11.1 Overview

ModelArts Standard allows you to view metrics in the following ways:

- **Viewing Monitoring Metrics on the ModelArts Console**: You can view the monitoring metrics on the ModelArts **Overview** page or the monitoring tab of each module.
- **Viewing All ModelArts Monitoring Metrics on the AOM Console**: 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.
- **Using Grafana to View AOM Monitoring Metrics**: Grafana provides different views and templates for monitoring, which allow you to see and analyze the real-time resource usage on dashboards clearly. This meets your diverse requirements.

  After data source configuration in Grafana, you can view all ModelArts Standard monitoring metrics stored in AOM using Grafana.

  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**

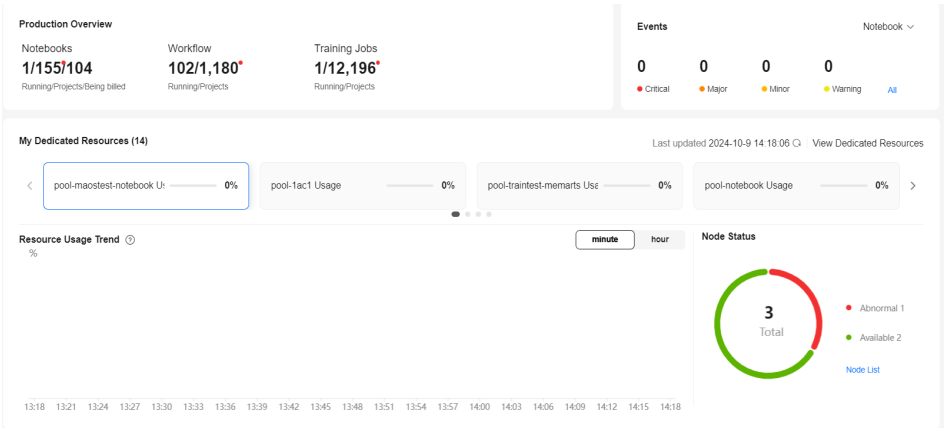
11.2 Viewing Monitoring Metrics on the ModelArts Console

Viewing ModelArts Monitoring Metrics on the Overview Page

On the ModelArts console's **Overview** page, you can see a summary of your production, including resource usage and training job details. Click on the links to

dive deeper into production overviews, resource pools, or individual training jobs for more information.

Figure 11-1 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

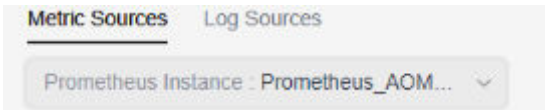
- Training jobs: When a training job is running, you can view the usage of resources such as CPUs, GPUs, and NPUs. For details, see **Viewing the Resource Usage of a Training Job**.
- Real-time services: After deploying a model as a real-time service, you can see how much CPU, memory, and GPU resources it is using, as well as how many AI application calls it has received. For details, see **Viewing Details About a Real-Time Service**.

11.3 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.

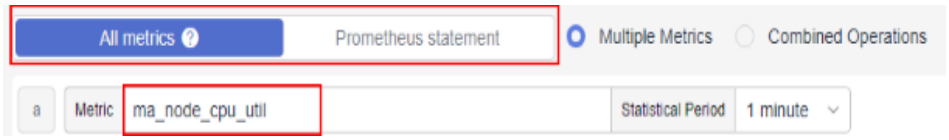
1. Log in to the console and search for **AOM** to go to the AOM console.
2. In the navigation pane on the left, choose **Metric Browsing**.
3. Select the **Prometheus_AOM_Default** instance from the drop-down list.

Figure 11-2 Specifying the metric source



4. Select one or more metrics from **All metrics** or **Prometheus statement**.

Figure 11-3 Adding a metric



For details about how to view metrics, see **Application Operations Management > User Guide (2.0) > Metric Browsing** in the Huawei Cloud Help Center.

The following table lists the metrics and labels supported by ModelArts.

Table 11-1 Container-level metrics

Category	Name	Metric	Description	Unit	Value Range
CPU	CPU Usage	ma_container_cpu_util	CPU usage of a measured object	%	0%–100%
	Used CPU Cores	ma_container_cpu_used_core	Number of CPU cores used by a measured object	Cores	≥ 0
	Total CPU Cores	ma_container_cpu_limit_core	Total number of CPU cores that have been requested for a measured object	Cores	≥ 1
Memory	Total Physical Memory	ma_container_memory_capacity_megabytes	Total physical memory that has been requested for a measured object	MB	≥ 0
	Physical Memory Usage	ma_container_memory_util	Percentage of the used physical memory to the total physical memory	%	0%–100%

Category	Name	Metric	Description	Unit	Value Range
	Used Physical Memory	ma_container_memory_used_megabytes	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
Storage	Disk Read Rate	ma_container_disk_read_kilobytes	Volume of data read from a disk per second	KB/s	≥ 0
	Disk Write Rate	ma_container_disk_write_kilobytes	Volume of data written into a disk per second	KB/s	≥ 0
GPU memory	Total GPU Memory	ma_container_gpu_mem_total_megabytes	Total GPU memory of a training job	MB	> 0
	GPU Memory Usage	ma_container_gpu_mem_util	Percentage of the used GPU memory to the total GPU memory	%	0%–100%
	Used GPU Memory	ma_container_gpu_mem_used_megabytes	GPU memory used by a measured object	MB	≥ 0
GPU	GPU Usage	ma_container_gpu_util	GPU usage of a measured object	%	0%–100%

Category	Name	Metric	Description	Unit	Value Range
	GPU Memory Bandwidth Usage	ma_container_gpu_mem_copy_util	GPU memory bandwidth usage of a measured object For example, the maximum memory bandwidth of GPVnt1 is 900 GB/s. If the current memory bandwidth is 450 GB/s, the memory bandwidth usage is 50%.	%	0%–100%
	GPU Encoder Usage	ma_container_gpu_enc_util	GPU encoder usage of a measured object	%	%
	GPU Decoder Usage	ma_container_gpu_dec_util	GPU decoder usage of a measured object	%	%
	GPU Temperature	DCGM_FI_DEV_GPU_TEMP	GPU temperature	°C	Natural number
	GPU Power	DCGM_FI_DEV_POWER_USAGE	GPU power	Watt (W)	> 0
	GPU Memory Temperature	DCGM_FI_DEV_MEMORY_TEMP	GPU memory temperature	°C	Natural number
Network I/O	Downlink Rate (BPS)	ma_container_network_receive_bytes	Inbound traffic rate of a measured object	Bytes/s	≥ 0
	Downlink Rate (PPS)	ma_container_network_receive_packets	Number of data packets received by a NIC per second	Packets/s	≥ 0

Category	Name	Metric	Description	Unit	Value Range
	Downlink Error Rate	ma_container_network_receive_error_packets	Number of error packets received by a NIC per second	Packets/s	≥ 0
	Uplink Rate (BPS)	ma_container_network_transmit_bytes	Outbound traffic rate of a measured object	Bytes/s	≥ 0
	Uplink Error Rate	ma_container_network_transmit_error_packets	Number of error packets sent by a NIC per second	Packets/s	≥ 0
	Uplink Rate (PPS)	ma_container_network_transmit_packets	Number of data packets sent by a NIC per second	Packets/s	≥ 0
Notebook service metrics	Notebook Cache Directory Size	ma_container_notebook_cache_dir_size_bytes	A high-speed local disk is attached to the **/cache** directory for GPU notebook instances. This metric indicates the total size of the directory.	Bytes	≥ 0
	Notebook Cache Directory Utilization	ma_container_notebook_cache_dir_util	A high-speed local disk is attached to the **/cache** directory for GPU notebook instances. This metric indicates the utilization of the directory.	%	0%–100%

Table 11-2 Node metrics (collected only in dedicated resource pools)

Category	Name	Metric	Description	Unit	Value Range
CPU	Total CPU Cores	ma_node_cpu_limit_core	Total number of CPU cores that have been requested for a measured object	Cores	≥ 1
	Used CPU Cores	ma_node_cpu_used_core	Number of CPU cores used by a measured object	Cores	≥ 0
	CPU Usage	ma_node_cpu_util	CPU usage of a measured object	%	0%–100%
	CPU I/O Wait Time	ma_node_cpu_iowait_counter	Disk I/O wait time accumulated since system startup	jiffies	≥ 0
Memory	Physical Memory Usage	ma_node_memory_util	Percentage of the used physical memory to the total physical memory	%	0%–100%
	Total Physical Memory	ma_node_memory_total_megabytes	Total physical memory that has been applied for a measured object	MB	≥ 0

Category	Name	Metric	Description	Unit	Value Range
Network I/O	Downlink Rate (BPS)	ma_node_network_receive_rate_bytes_seconds	Inbound traffic rate of a measured object	Bytes/s	≥ 0
	Uplink Rate (BPS)	ma_node_network_transmit_rate_bytes_seconds	Outbound traffic rate of a measured object	Bytes/s	≥ 0
Storage	Disk Read Rate	ma_node_disk_read_rate_kilobytes_seconds	Volume of data read from a disk per second (Only data disks used by containers are collected.)	KB/s	≥ 0
	Disk Write Rate	ma_node_disk_write_rate_kilobytes_seconds	Volume of data written into a disk per second (Only data disks used by containers are collected.)	KB/s	≥ 0
	Total Cache	ma_node_cache_space_capacity_megabytes	Total cache of the Kubernetes space	MB	≥ 0
	Used Cache	ma_node_cache_space_used_capacity_megabytes	Used cache of the Kubernetes space	MB	≥ 0

Category	Name	Metric	Description	Unit	Value Range
	Total Container Space	ma_node_container_space_capacity_megabytes	Total container space	MB	≥ 0
	Used Container Space	ma_node_container_space_used_capacity_megabytes	Used container space	MB	≥ 0
	Disk Information	ma_node_disk_info	Basic disk information	N/A	≥ 0
	Total Reads	ma_node_disk_reads_completed_total	Total number of successful reads	N/A	≥ 0
	Merged Reads	ma_node_disk_reads_merged_total	Number of merged reads	N/A	≥ 0
	Bytes Read	ma_node_disk_read_bytes_total	Total number of bytes that are successfully read	Bytes	≥ 0
	Read Time Spent	ma_node_disk_read_time_seconds_total	Time spent on all reads	Seconds	≥ 0
	Total Writes	ma_node_disk_writes_completed_total	Total number of successful writes	N/A	≥ 0
	Merged Writes	ma_node_disk_writes_merged_total	Number of merged writes	N/A	≥ 0

Category	Name	Metric	Description	Unit	Value Range
	Written Bytes	ma_node_disk_written_bytes_total	Total number of bytes that are successfully written	Bytes	≥ 0
	Write Time Spent	ma_node_disk_write_time_seconds_total	Time spent on all write operations	Seconds	≥ 0
	Ongoing I/Os	ma_node_disk_io_now	Number of ongoing I/Os	N/A	≥ 0
	I/O Execution Duration	ma_node_disk_io_time_seconds_total	Time spent on executing I/Os	Seconds	≥ 0
	I/O Execution Weighted Time	ma_node_disk_io_time_weighted_seconds_tota	Weighted time spent on executing I/Os	Seconds	≥ 0
GPU	GPU Usage	ma_node_gpu_util	GPU usage of a measured object	%	0%–100%
	Total GPU Memory	ma_node_gpu_mem_total_megabytes	Total GPU memory of a measured object	MB	> 0
	GPU Memory Usage	ma_node_gpu_mem_util	Percentage of the used GPU memory to the total GPU memory	%	0%–100%
	Used GPU Memory	ma_node_gpu_mem_used_megabytes	GPU memory used by a measured object	MB	≥ 0

Category	Name	Metric	Description	Unit	Value Range
	Tasks on a Shared GPU	node_gpu_share_job_count	Number of tasks running on a shared GPU	Number	≥ 0
	GPU Temperature	DCGM_FI_DEV_GPU_TEMP	GPU temperature	°C	Natural number
	GPU Power	DCGM_FI_DEV_POWER_USAGE	GPU power	Watt (W)	> 0
	GPU Memory Temperature	DCGM_FI_DEV_MEMORY_TEMP	GPU memory temperature	°C	Natural number
InfiniBand or RoCE network	Total Amount of Data Received by a NIC	ma_node_infiniband_port_received_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.	Double words (32 bits)	≥ 0
	Total Amount of Data Sent by a NIC	ma_node_infiniband_port_transmitted_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.	Double words (32 bits)	≥ 0

Category	Name	Metric	Description	Unit	Value Range
NFS mounting status	NFS Getattr Congestion Time	ma_node_mountstats_getattr_backlog_wait	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

Category	Name	Metric	Description	Unit	Value Range
	NFS Getattr Round Trip Time	ma_node_mountstats_getattr_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 reply34. 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

Category	Name	Metric	Description	Unit	Value Range
	NFS Access Congestion Time	ma_node_mountstats_access_backlog_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

Category	Name	Metric	Description	Unit	Value Range
	NFS Access Round Trip Time	ma_node_mountstats_access_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 reply34. 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

Category	Name	Metric	Description	Unit	Value Range
	NFS Lookup Congestion Time	ma_node_mountstats_lookup_backlog_wait	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

Category	Name	Metric	Description	Unit	Value Range
	NFS Lookup Round Trip Time	ma_node_mountstats_lookup_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 reply34. 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

Category	Name	Metric	Description	Unit	Value Range
	NFS Read Congestion Time	ma_node_mountstats_read_backlog_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

Category	Name	Metric	Description	Unit	Value Range
	NFS Read Round Trip Time	ma_node_mountstats_read_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 reply34. 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

Category	Name	Metric	Description	Unit	Value Range
	NFS Write Congestion Time	ma_node_mountstats_write_backlog_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

Category	Name	Metric	Description	Unit	Value Range
	NFS Write Round Trip Time	ma_node_mountstats_write_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 reply34. 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

Table 11-3 Diagnosis (InfiniBand, collected only in dedicated resource pools)

Category	Name	Metric	Description	Unit	Value Range
InfiniBand 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

Table 11-4 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
	user_id	User ID of the account to which the user who submits the job belongs

Classification	Label	Description
	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

Classification	Label	Description
	gpu_type	Type of a node GPU
	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 InfiniBand 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

11.4 Using Grafana to View AOM Monitoring Metrics

11.4.1 Installing and Configuring Grafana

11.4.1.1 Installing and Configuring Grafana on Windows

Description

This section describes how to install and configure Grafana on a Windows operating system.

Procedure

1. Download the Grafana installation package.

   Go to the **download link**, click **Download the installer**, and wait until the download is successful.

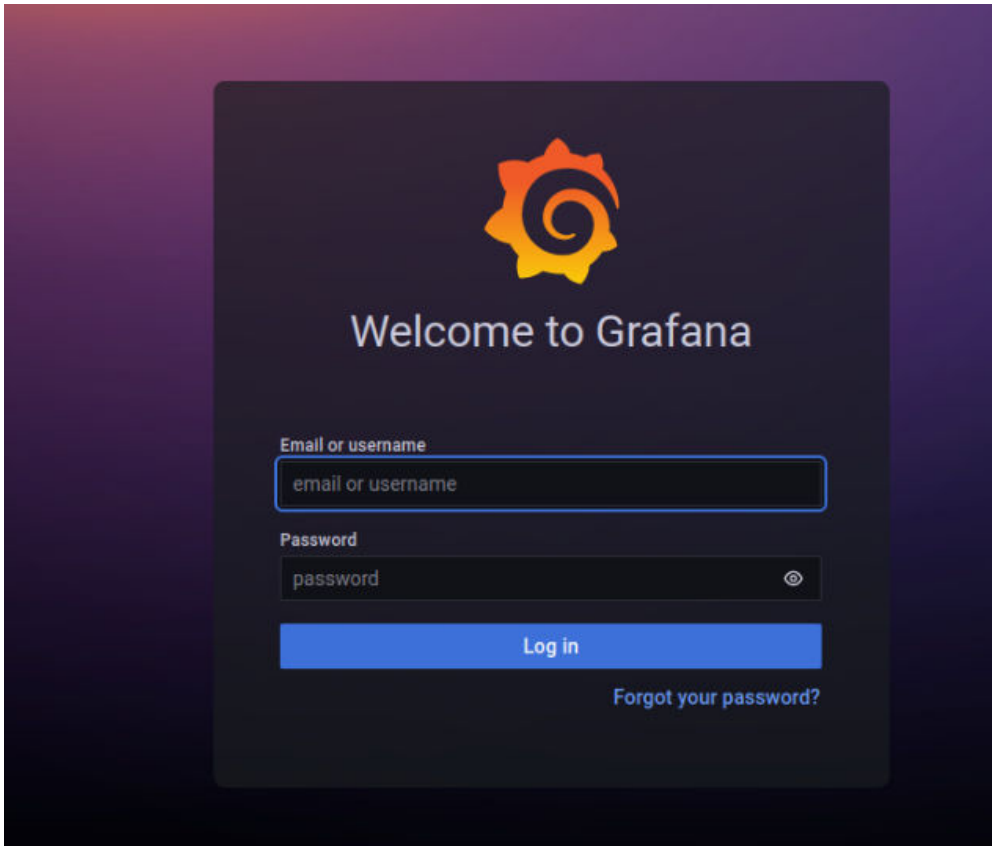
2. Install Grafana.

   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.



11.4.1.2 Installing and Configuring Grafana on Linux

Prerequisites

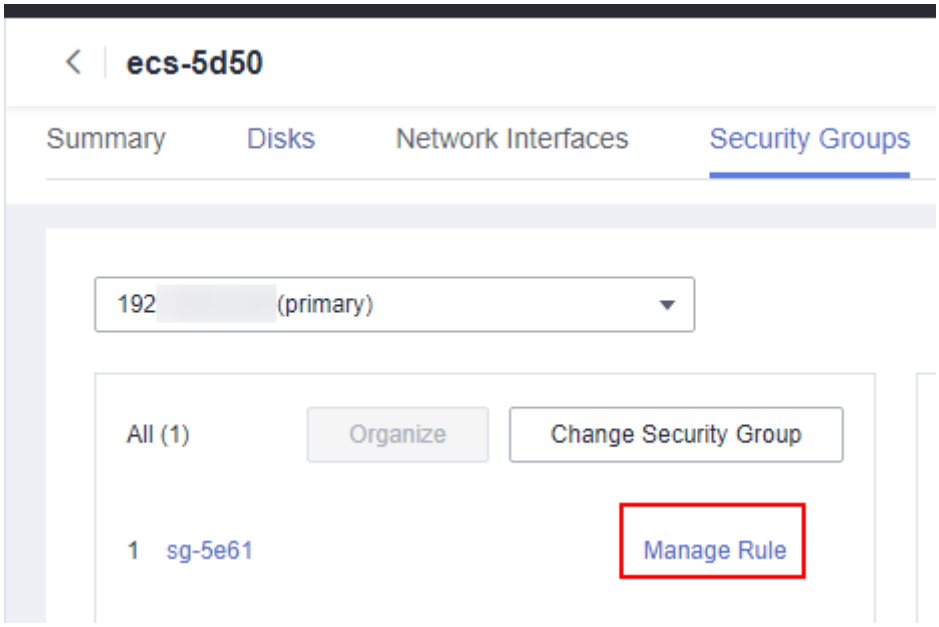
- An Ubuntu server that is accessible to the Internet is available. If no, the following conditions must be met:
- You have obtained an ECS. (You are advised to select 8 vCPUs or higher, Ubuntu image of 22.04 version, and 100 GB local storage.) For details, see **Purchasing an ECS**.
- You have purchased an EIP and bound it to the ECS. For details, see **Assigning an EIP and Binding It to an ECS**.

Procedure

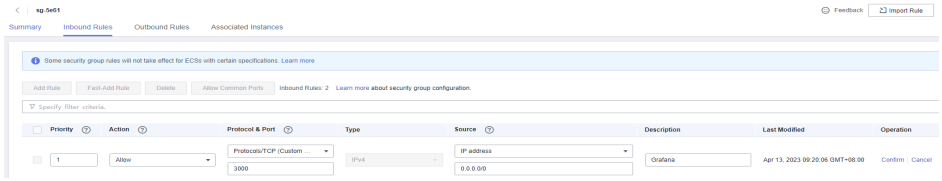
1. Log in to the ECS. Select a login method. For details, see .
2. Install libfontconfig1:
   ```
   sudo apt-get install -y adduser libfontconfig1
   ```
   The operation is successful if the following information is displayed.
   ```
   root@ecs-9ec3:~# sudo apt-get install -y adduser libfontconfig1
   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:

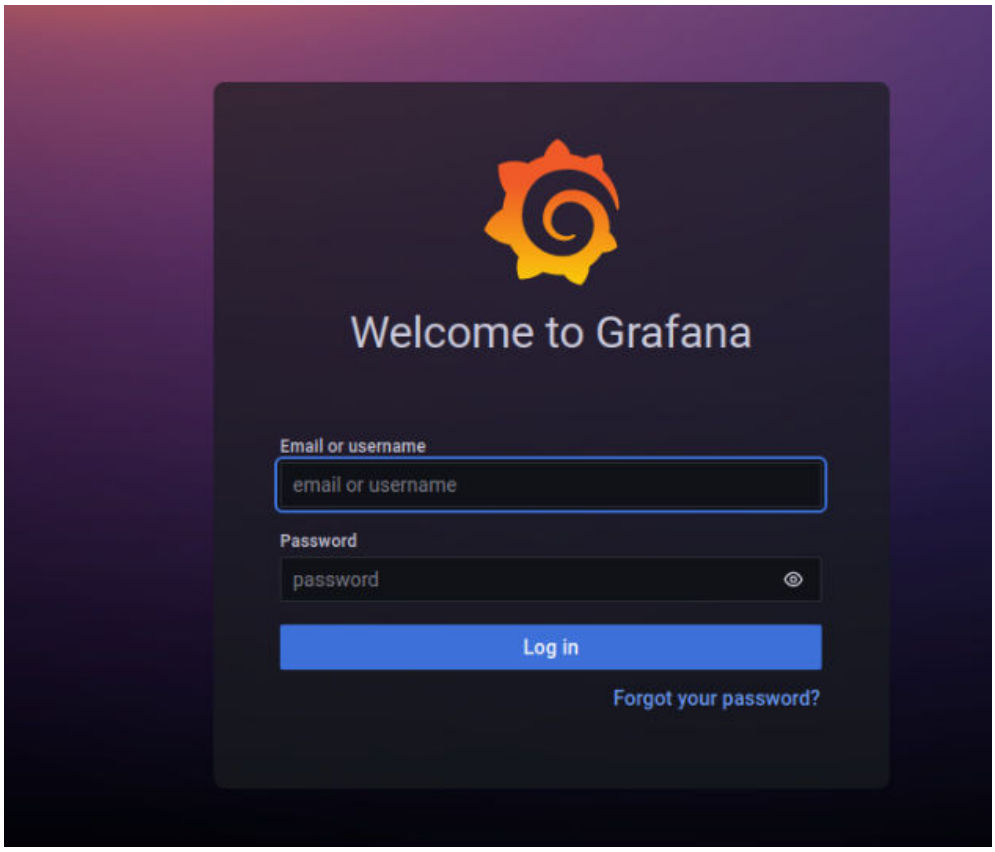
a. Click the ECS name to go to the ECS details page. Then, click the **Security Groups** tab, and click **Manage Rule**.



b. Click **Inbound Rules** and allow inbound traffic from TCP port 3000. By default, all outbound traffic is allowed.



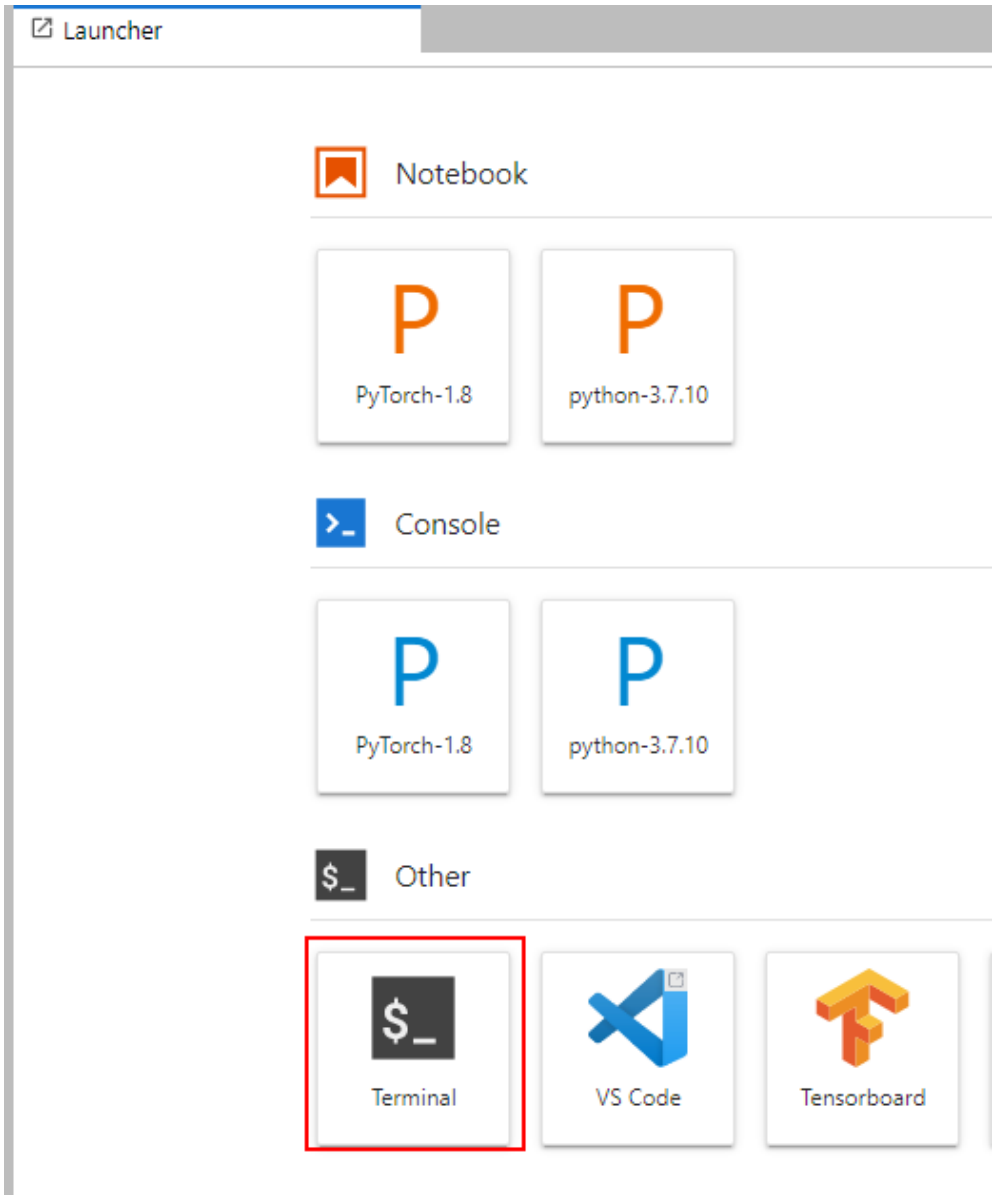
7. Access **http://{*EIP*}:3000** in a browser. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.



11.4.1.3 Installing and Configuring Grafana on a Notebook Instance

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
```

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

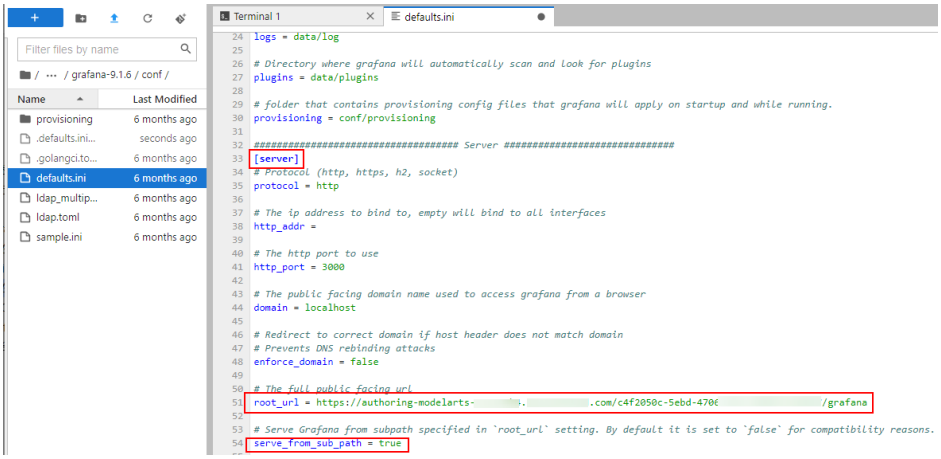
If **jupyter_notebook_config.py** (path: **/home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py**) contains the **c.ServerProxy.servers** field, add the corresponding key-value pair.

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.

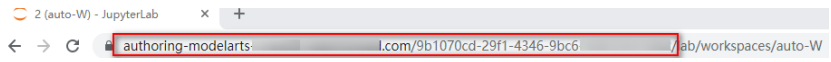
b. Change the **root_url** and **serve_from_sub_path** fields in **[server]**.

Figure 11-4 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 **Development Workspace** > **Notebook**. In the notebook instance list, locate the target instance, and choose **More** > **Save Image** in the **Operation** column.

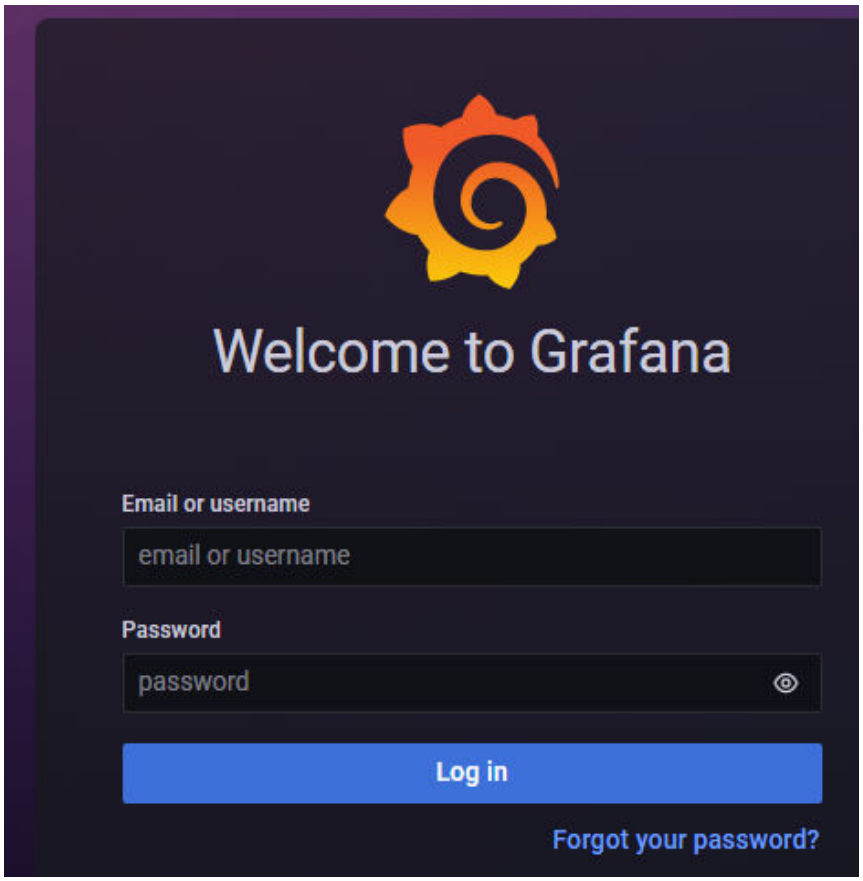
b. In the **Save Image** dialog box, configure parameters. Click **OK** to save the image.

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.



11.4.2 Configuring a Grafana Data Source

Before viewing ModelArts monitoring data on Grafana, configure the data source.

Prerequisites

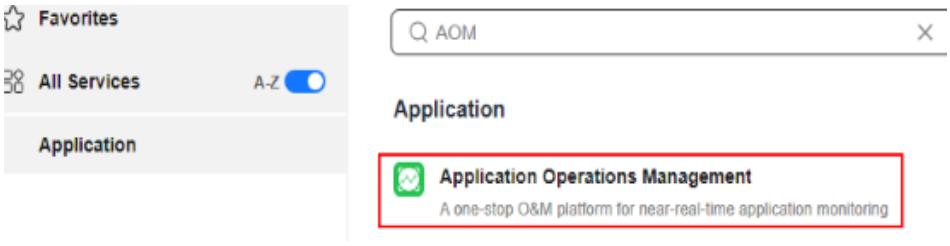
Grafana has been installed.

Procedure

1. Obtain the Grafana data source configuration code.

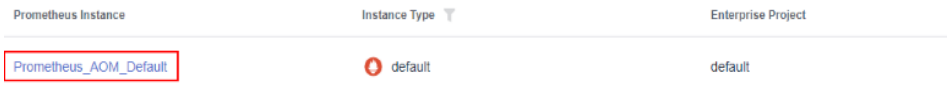
a. Log in to the AOM console.

Figure 11-5 AOM console



b. In the navigation pane on the left, choose **Prometheus Monitoring** > **Instances**. Then, click the **Prometheus_AOM_Default** instance.

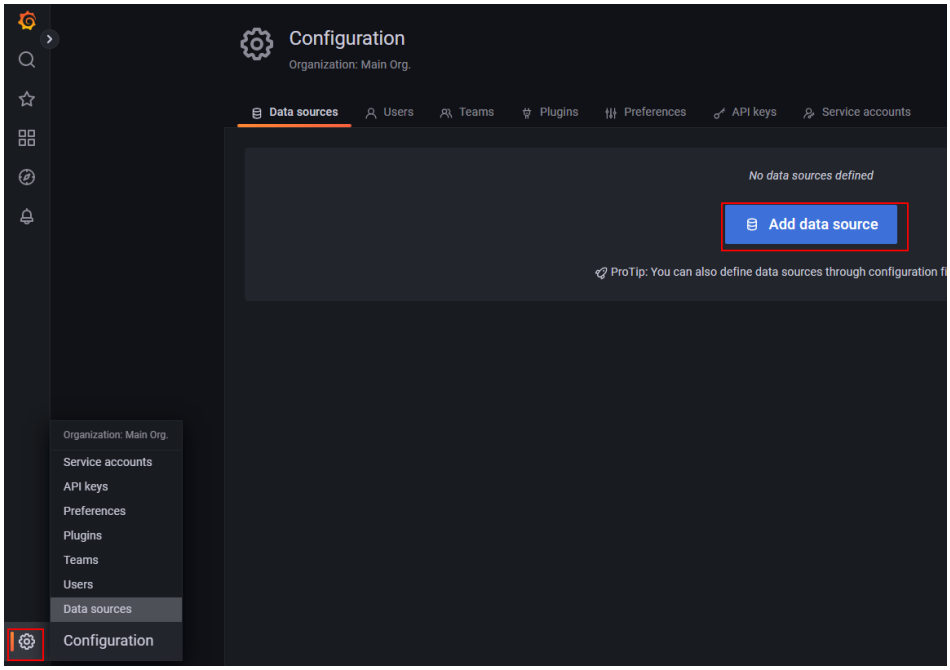
Figure 11-6 Prometheus_AOM_Default



c. In the **Settings** tab, obtain the Grafana data source configuration code of the instance in the **Grafana Data Source Info** area.

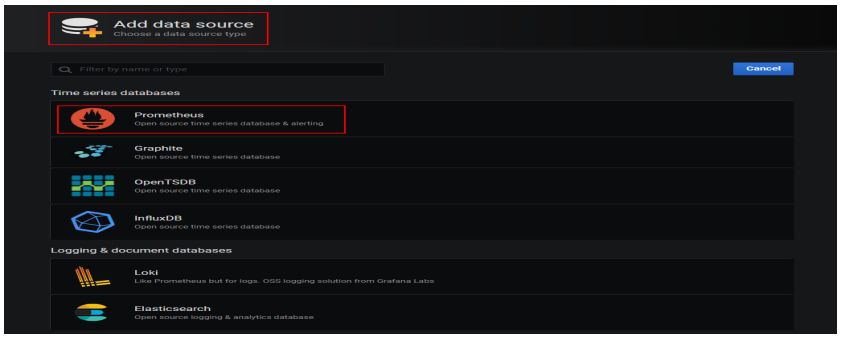
2. Add a data source to Grafana.
   a. Log in to Grafana. The default username and password for the first login are **admin**. After the login is successful, change the password as prompted.
   b. In the navigation pane, choose **Configuration** > **Data Sources**. Then, click **Add data source**.

Figure 11-7 Configuring Grafana



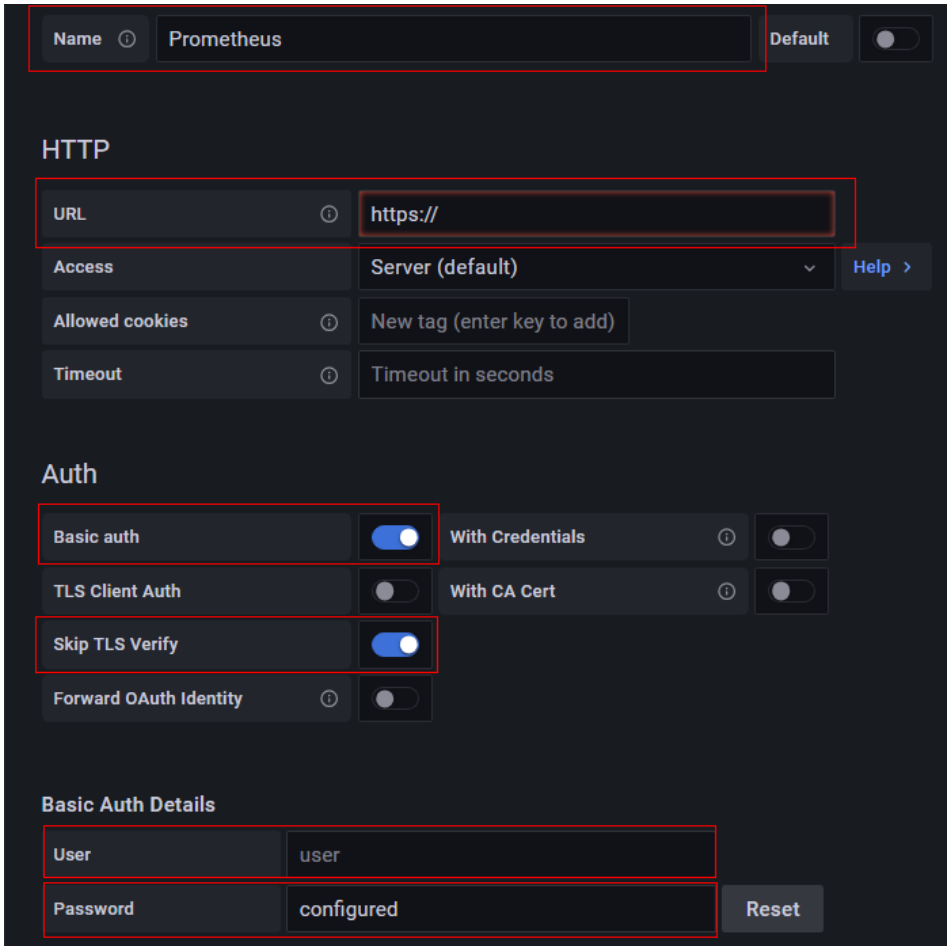
c. Click **Prometheus** to access the configuration page.

Figure 11-8 Prometheus configuration page



d. Configure parameters as shown in the following figure.

Figure 11-9 Configuring a Grafana data source



NOTE

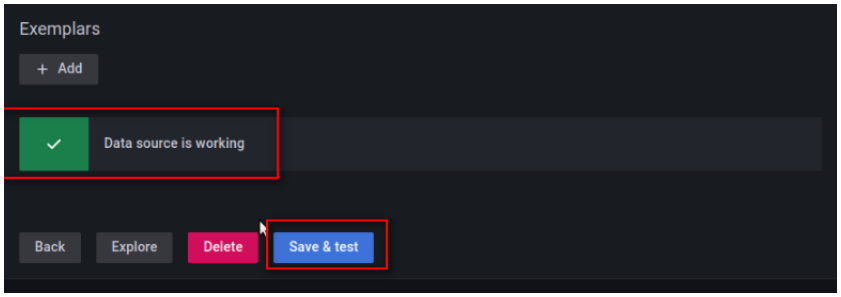
The actual Grafana version varies depending on the installation method. **Figure 11-9** is only an example.

Table 11-5 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.

e. After the configuration, click **Save & test**. If the message **Data source is working** is displayed, the data source is configured.

Figure 11-10 Data source added



11.4.3 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

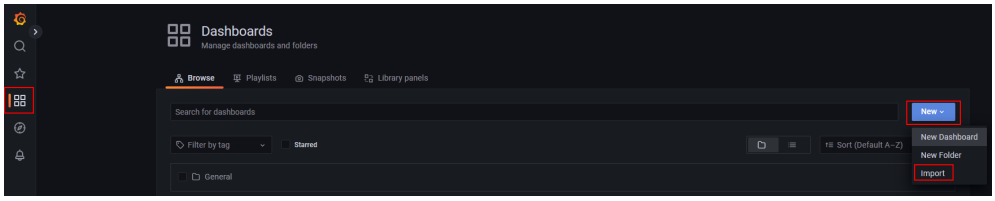
ModelArts provides templates for cluster view, node view, user view, task view, and task details view. These templates can be downloaded from Grafana official documents. You can import and use them on Dashboards.

Table 11-6 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.

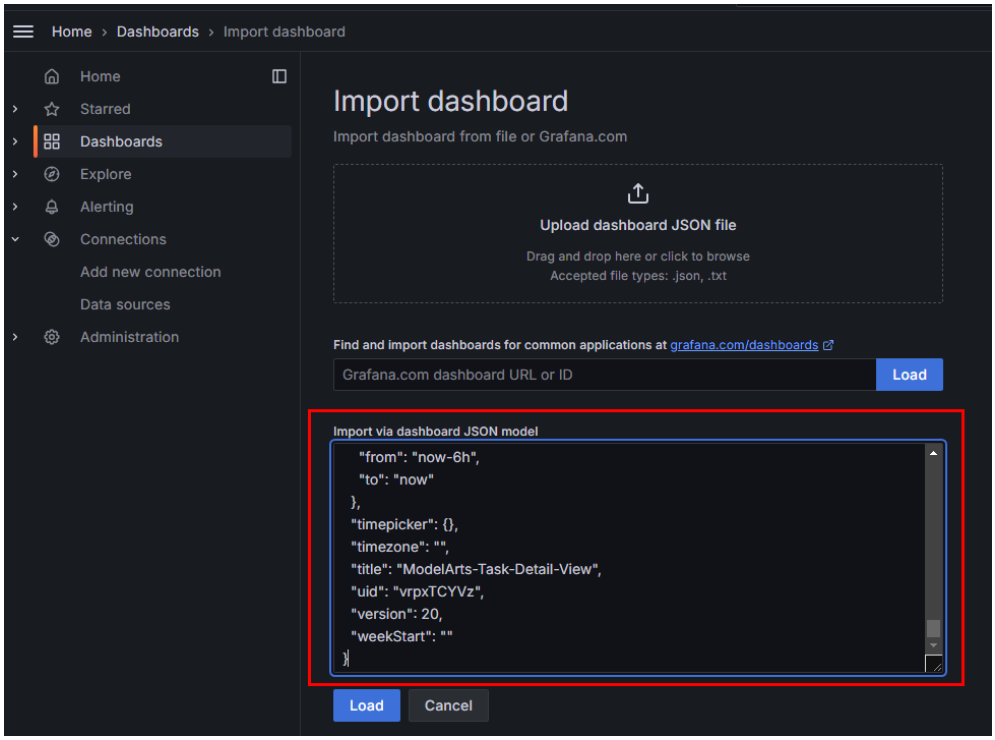
   Copy the template download URL provided in **Preparations** to a web browser and copy the content of the JSON file. Paste the content to the dashboard template and click **Load**.

Figure 11-11 Copying the content of the JSON file

https:// -modelarts-sdk.obs. .com/metrics/grafana/dashboards/ModelArts-Cluster-View.json

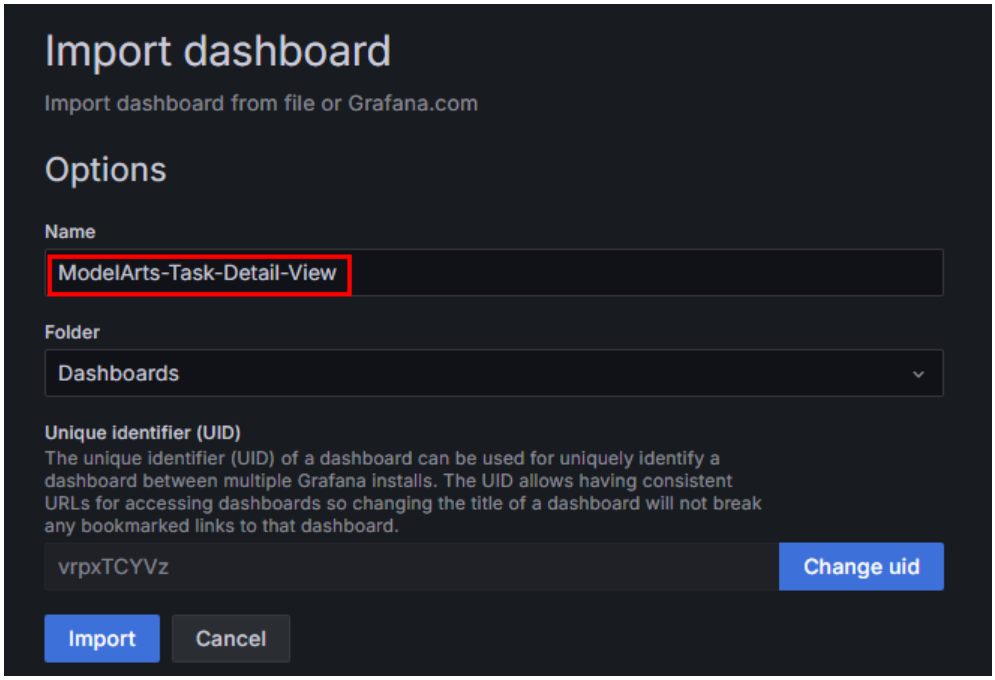

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": {
          "type": "grafana",
          "uid": "-- Grafana --"
        },
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "target": {
          "limit": 100,
          "matchAny": false,
          "tags": [],
          "type": "dashboard"
        },
        "type": "dashboard"
      }
    ]
  },
  "description": "",
  "editable": true,
  "fiscalYearStartMonth": 0,
  "graphTooltip": 0,
  "id": 60,
  "links": [],
  "liveNow": false,
  "panels": [
    {
      "collapsed": false,
      "gridPos": {
        "h": 1,
        "w": 24,
        "x": 0,
        "y": 0
      },
      "id": 26,
      "panels": [],
      "title": "Overview",
      "type": "row"
    },
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${data_source}"
```

Figure 11-12 Pasting the content of the JSON file to the dashboard template



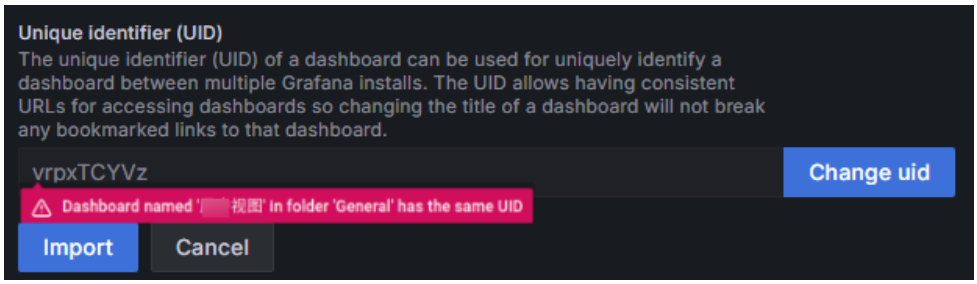
3. Change the view name and click **Import**.

Figure 11-13 Changing the view name

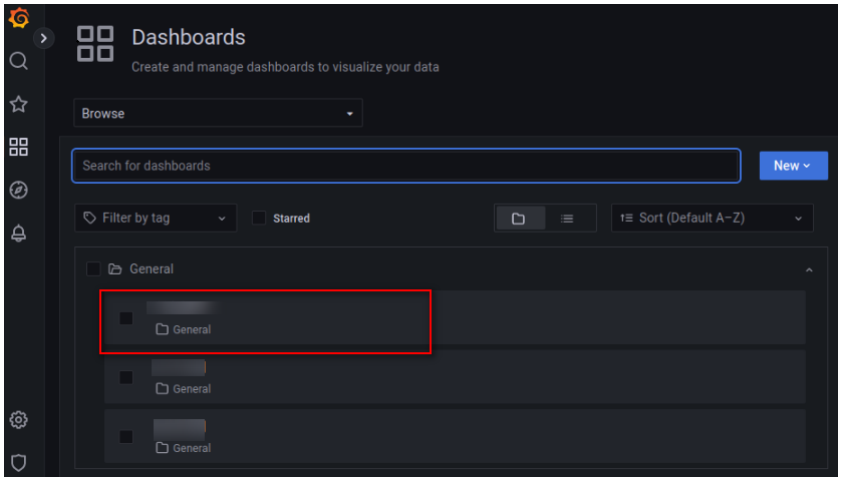


Note: If a message is displayed, indicating that the UID is duplicate, click **Change uid**, change the UID in the JSON file, and click **Import**.

Figure 11-14 Changing the UID



4. After the import, view the imported views in **Dashboards**. Then, click a view to open the monitoring page.

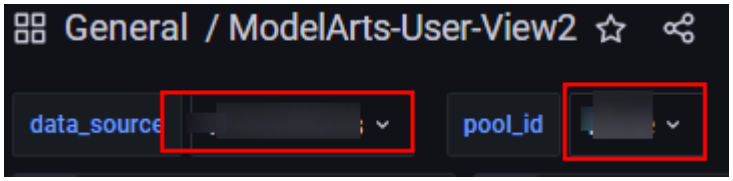


5. Use the template.

   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 11-15 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

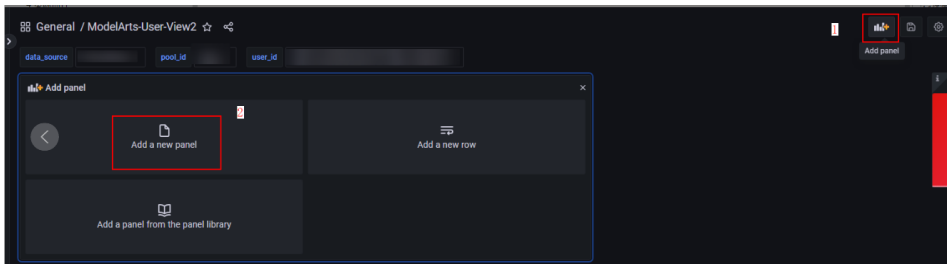
  Figure 11-16 Changing the time range for obtaining data

  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

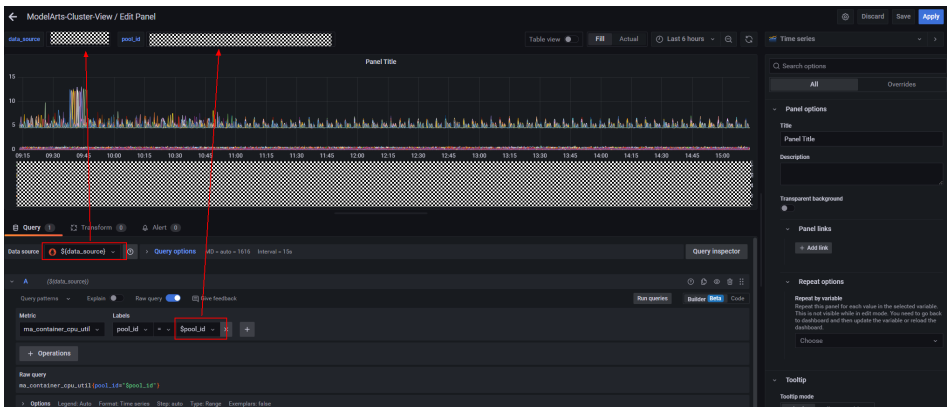
  Figure 11-17 Adding a panel

  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 11-18 Using the current dashboard settings

Creating a Dashboard to View Metrics

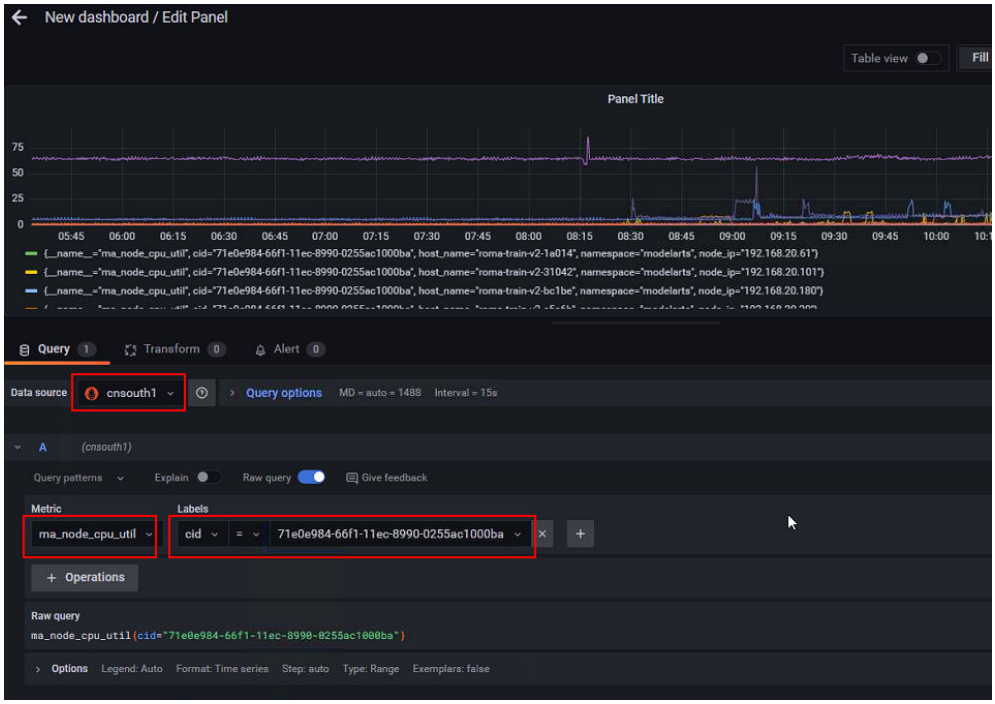
1. Open **Dashboards**, click **New**, and choose **New Dashboard**.
2. Click **Add a new panel**.
3. On the **New dashboard / Edit Panel** page, set the following parameters:

   Data source: **Configured Grafana data source**

   Metric: Metric name. You can obtain the metric to be queried by referring to **Table 11-1**, **Table 11-2**, and **Table 11-3**.

   Labels: Used for filtering the metric. For details, see and **Table 11-4**.

Figure 11-19 Creating a dashboard to view metrics



12 Viewing Audit Logs

12.1 ModelArts Key Operations Traced by CTS

With CTS, you can obtain operations associated with ModelArts for later query, audit, and backtrack operations.

Prerequisites

CTS has been enabled.

Key Data Management Operations Traced by CTS

Table 12-1 Key data management operations traced by CTS

Operation	Resource Type	Trace
Creating a dataset	Dataset	createDataset
Deleting a dataset	Dataset	deleteDataset
Updating a dataset	Dataset	updateDataset
Publishing a dataset version	Dataset	publishDatasetVersion
Deleting a dataset version	Dataset	deleteDatasetVersion
Synchronizing the data source	Dataset	syncDataSource
Exporting a dataset	Dataset	exportDataFromDataset
Creating an auto labeling task	Dataset	createAutoLabelingTask
Creating an auto grouping task	Dataset	createAutoGroupingTask

Operation	Resource Type	Trace
Creating an auto deployment task	Dataset	createAutoDeployTask
Importing samples to a dataset	Dataset	importSamplesToDataset
Creating a dataset label	Dataset	createLabel
Updating a dataset label	Dataset	updateLabel
Deleting a dataset label	Dataset	deleteLabel
Deleting a dataset label and its samples	Dataset	deleteLabelWithSamples
Adding samples	Dataset	uploadSamples
Deleting samples	Dataset	deleteSamples
Stopping an auto labeling task	Dataset	stopTask
Creating a team labeling task	Dataset	createWorkforceTask
Deleting a team labeling task	Dataset	deleteWorkforceTask
Starting the acceptance of a team labeling task	Dataset	startWorkforceSampling-Task
Approving, rejecting, or canceling the acceptance of a team labeling task	Dataset	updateWorkforceSam-plingTask
Submitting sample review comments for an acceptance task	Dataset	acceptSamples
Adding a label to a sample	Dataset	updateSamples
Sending an email to team labeling members	Dataset	sendEmails
Starting a team labeling task as the team manager	Dataset	startWorkforceTask
Updating a team labeling task	Dataset	updateWorkforceTask
Adding a label to a team-labeled sample	Dataset	updateWorkforceTask-Samples

Operation	Resource Type	Trace
Reviewing team labeling results	Dataset	reviewSamples
Creating a labeling team member	Workforce	createWorker
Updating labeling team members	Workforce	updateWorker
Deleting a labeling team member	Workforce	deleteWorker
Deleting labeling team members in batches	Workforce	batchDeleteWorker
Creating a labeling team	Workforce	createWorkforce
Updating a labeling team	Workforce	updateWorkforce
Deleting a labeling team	Workforce	deleteWorkforce
Automatically creating an IAM agency	IAM	createAgency
Logging in to the labeling console as a team labeling member	labelConsoleWorker	workerLoginLabelConsole
Logging out of the labeling console as a team labeling member	labelConsoleWorker	workerLogOutLabelConsole
Changing the password for logging in to the labeling console as a team labeling member	labelConsoleWorker	workerChangePassword
Handling the issue that the password for logging in to the labeling console as a team labeling member is lost	labelConsoleWorker	workerForgetPassword
Resetting the password for logging in to the labeling console through the URL as a team labeling member	labelConsoleWorker	workerResetPassword

Key Development Environment Operations Traced by CTS

Table 12-2 Key development environment operations traced by CTS

Operation	Resource Type	Trace
Creating a notebook instance	Notebook	createNotebook
Deleting a notebook instance	Notebook	deleteNotebook
Opening a notebook instance	Notebook	openNotebook
Starting a notebook instance	Notebook	startNotebook
Stopping a notebook instance	Notebook	stopNotebook
Updating a notebook instance	Notebook	updateNotebook
Deleting a NotebookApp	NotebookApp	deleteNotebookApp
Switching CodeLab specifications	NotebookApp	updateNotebookApp

Key Training Job Operations Traced by CTS

Table 12-3 Key training job operations traced by CTS

Operation	Resource Type	Trace
Creating a training job	ModelArtsTrainJob	createModelArtsTrainJob
Creating a training job version	ModelArtsTrainJob	createModelArtsTrainVersion
Stopping a training job	ModelArtsTrainJob	stopModelArtsTrainVersion
Modifying the description of a training job	ModelArtsTrainJob	updateModelArtsTrainDesc
Deleting a training job version	ModelArtsTrainJob	deleteModelArtsTrainVersion
Deleting a training job	ModelArtsTrainJob	deleteModelArtsTrainJob
Configuring a training job	ModelArtsTrainConfig	createModelArtsTrainConfig

Operation	Resource Type	Trace
Modifying training job configurations	ModelArtsTrainConfig	updateModelArtsTrain-Config
Deleting training job configurations	ModelArtsTrainConfig	deleteModelArtsTrain-Config
Creating a visualization job	ModelArtsTensorboard-Job	createModelArtsTensor-boardJob
Deleting a visualization job	ModelArtsTensorboard-Job	deleteModelArtsTensor-boardJob
Modifying the description of a visualization job	ModelArtsTensorboard-Job	updateModelArtsTensor-boardDesc
Stopping a visualization job	ModelArtsTensorboard-Job	stopModelArtsTensor-boardJob
Restarting a visualization job	ModelArtsTensorboard-Job	restartModelArtsgTensor-boardJob

Key AI Application Management Operations Traced by CTS

Table 12-4 Key AI application management operations traced by CTS

Operation	Resource Type	Trace
Creating an AI application	Model	addModel
Updating an AI application	Model	updateModel
Deleting an AI application	Model	deleteModel
Creating a model conversion task	Convert	addConvert
Updating a model conversion task	Convert	updateConvert
Deleting a model conversion task	Convert	deleteConvert

Key Service Management Operations Traced by CTS

Table 12-5 Key service management operations traced by CTS

Operation	Resource Type	Trace
Deploying a service	Service	addService
Deleting a service	Service	deleteService
Updating a service	Service	updateService
Starting or stopping a service	Service	startOrStopService
Adding an access key	Service	addAkSk
Deleting an access key	Service	deleteAkSk
Creating a dedicated resource pool	Cluster	createCluster
Deleting a dedicated resource pool	Cluster	deleteCluster
Adding a node to a dedicated resource pool	Cluster	addClusterNode
Deleting a node from a dedicated resource pool	Cluster	deleteClusterNode
Obtaining the result of creating a dedicated resource pool	Cluster	createClusterResult

Key AI Gallery Operations Traced by CTS

Table 12-6 Key AI Gallery operations traced by CTS

Operation	Resource Type	Trace
Publishing an asset	ModelArts_Market	create_content
Modifying asset information	ModelArts_Market	modify_content
Publishing an asset version	ModelArts_Market	add_version
Subscribing to an asset	ModelArts_Market	subscription_content
Removing an asset from favorites	ModelArts_Market	cancel_star_content
Liking an asset	ModelArts_Market	like_content

Operation	Resource Type	Trace
Unliking an asset	ModelArts_Market	cancel_like_content
Publishing an activity	ModelArts_Market	publish_activity
Signing up an activity	ModelArts_Market	regist_activity
Modifying user information	ModelArts_Market	update_user

Key Resource Management Operations Traced by CTS

Table 12-7 Key resource management operations traced by CTS

Operation	Resource Type	Trace
Creating a resource pool	PoolV2	CreatePoolV2
Deleting a resource pool	PoolV2	DeletePoolV2
Updating a resource pool	PoolV2	UpdatePoolV2
Creating a network	NetworksV1	CreateNetworksV1
Deleting a network	NetworksV1	DeleteNetworksV1
Updating a network	NetworksV1	UpdateNetworksV1

12.2 Viewing ModelArts Audit Logs

After CTS is enabled, CTS starts recording operations on ModelArts. The CTS console stores the operation records generated in the last seven days. This section describes how to view operation records of the last seven days on the CTS console.

Procedure

1. Log in to the CTS console.
2. Click  in the upper left corner and select a region.
3. In the navigation pane on the left, choose **Trace List**.
4. Specify filters as needed. You can query traces using a combination of the following filters:
   - **Trace Source**, **Resource Type**, and **Search By**:

     Select a filter from the drop-down list.

     When you select **Trace Name**, you need to enter a specific trace name.

     When you select **Resource ID**, you need to enter a specific resource ID.

     When you select **Resource Name**, you need to enter a specific resource name.

- **Operator**: Select a specific operator (a user other than tenant).
- **Trace Status**: Select **All trace statuses**, **Normal**, **Warning**, or **Incident**.
- **Operation Time**: You can query traces generated during any time range in the last seven days.

5. Click ⌄ on the left of a trace to expand its details.
6. Locate the target trace and click **View Trace** in the **Operation** column. In the displayed **View Trace** dialog box, view the trace structure details.

   For details about the key fields in the CTS trace structure, see **Cloud Trace Service User Guide**.

