ModelArts

Best Practices

Issue	01
Date	2025-08-12

HUAWEI

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 Best Practices

This document provides ModelArts samples concerning a variety of scenarios and AI engines to help you quickly understand the process and operations of using ModelArts for AI development.

Configuring ModelArts Standard Permissions

Sample	Function	Scenario	Description
ModelArts Standard Permission Management	IAM permission configuration and management	Permission assignment for IAM users	Assign specific ModelArts operation permissions to the IAM users under a Huawei Cloud account. This prevents exceptions from occurring due to permissions when the IAM users access ModelArts.

ModelArts Standard Model Training

Table 1-1 Custom algorithm samples

Sample	Image	Function	Scenario	Description
Building a Handwritten Digit Recognition Model with ModelArts Standard	PyTorch	Algorithm customization	Handwritten digit recognition	Use your customized algorithm to train a handwritten digit recognition model and deploy the model for prediction.

ModelArts Standard Inference Deployment

Table 1-2 Inference deployment samples

Sample	Image	Function	Scenario	Description
Migrating a Third-Party Inference Framework to a Custom Inference Engine	-	Third-party frameworks Inference deployment	-	ModelArts allows the deployment of third-party inference frameworks. This section describes how to migrate TF Serving and Triton to a custom inference engine.

2 DeepSeek Inference Applications on MaaS

2.1 Quickly Building Your Own AI Assistant with ModelArts Studio (MaaS) DeepSeek API and Cherry Studio

Context

Large Language Models (LLMs) are now essential for advancing natural language processing applications. The DeepSeek series offers LLMs designed for efficiency and strong performance.

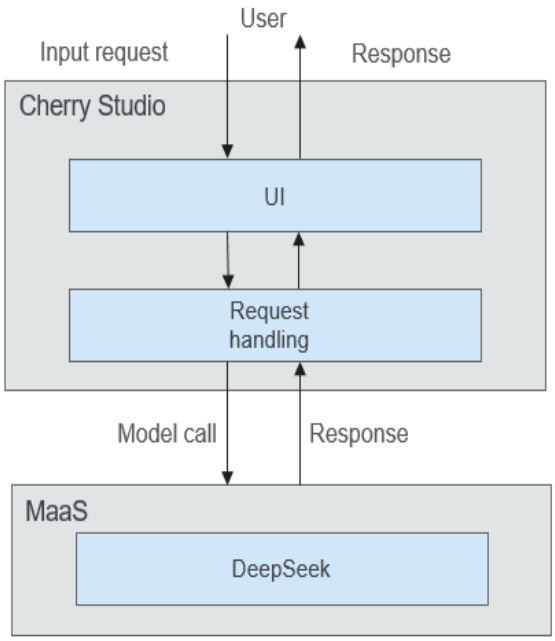
To simplify their use, several tools and platforms have been developed. Among them, Cherry Studio stands out as a versatile open-source desktop application compatible with Windows, macOS, and Linux. It combines popular LLMs like OpenAI, DeepSeek, and Gemini, enabling both cloud-based and local operations. Its features include dialog knowledge bases, AI painting, translation, and seamless model switching.

ModelArts Studio (MaaS) integrates DeepSeek models into its platform and enhances their accuracy and efficiency using AI Cloud Service.

This guide explains how to access Cherry Studio with MaaS DeepSeek APIs to create your own AI assistant swiftly.

Solution Architecture

Figure 2-1 Solution architecture



- Users submit requests through Cherry Studio.
- Cherry Studio sends the requests to MaaS DeepSeek.
- MaaS DeepSeek processes the requests and returns the results to Cherry Studio.
- Cherry Studio optimizes the results and returns them to users.

Prerequisites

- You have registered a Huawei Cloud account and completed real-name authentication. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.
- You have deployed a service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

Step 1: Downloading and Installing Cherry Studio

You can download and install Cherry Studio from the **official website** or **open-source address**.

Step 2: Obtaining MaaS DeepSeek Interconnection Information

Obtain key details about the target MaaS DeepSeek model service for connecting with Cherry Studio.

1. Create an API key for authentication when calling the MaaS DeepSeek model service.

   You can create a up to 30 keys. Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to **ModelArts Studio (MaaS) console** and select **CN Southwest-Guiyang1** from the top navigation bar.
   b. In the navigation pane, choose **API Keys**.
   c. On the **API Keys** page, click **Create API Key**, enter the tag and description, and click **OK**.

      The tag and description cannot be modified after the key is created.

      Table 2-1 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

   d. In the **Your Key** dialog box, copy the key and store it securely.
   e. After the key is saved, click **Close**.

      After you click **Close**, the key cannot be viewed again.

2. Obtain the API URL and model name of the MaaS DeepSeek model service.

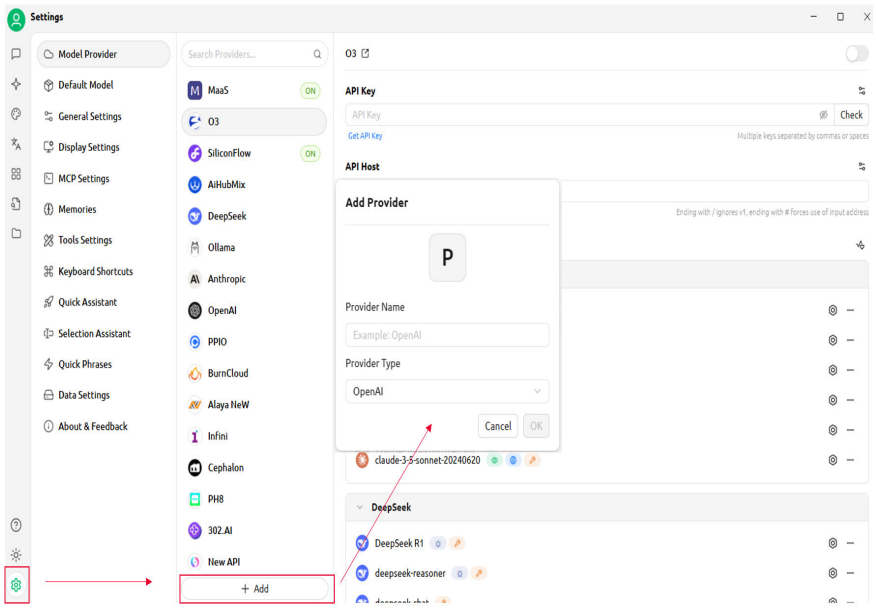
   a. In the navigation pane of the **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.
   b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.
   c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.
   d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cherry Studio.

Step 3: Configuring MaaS DeepSeek in Cherry Studio

1. Add a MaaS provider in Cherry Studio.

   a. In the lower left corner of the Cherry Studio client, click the settings icon and click **Add** in **Model Provider**.

Figure 2-2 Adding a provider



b. Set the provider name and provider type and click **OK**.

Table 2-2 Parameters for adding a provider

Parameter	Description
Provider Name	Enter **MaaS**, which can be changed as needed.
Provider Type	Set this parameter to **OpenAI**.

2. Add the MaaS DeepSeek API key and API URL in Cherry Studio.
   a. In the lower left corner of the Cherry Studio client, click the settings icon.
   b. Click **MaaS** and configure the API key and API URL.

Parameter	Description
API Key	API key created in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**.
API Host	Basic API URL of the MaaS service obtained in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**. Remove **/v1/chat/completions** from the URL and enter it.

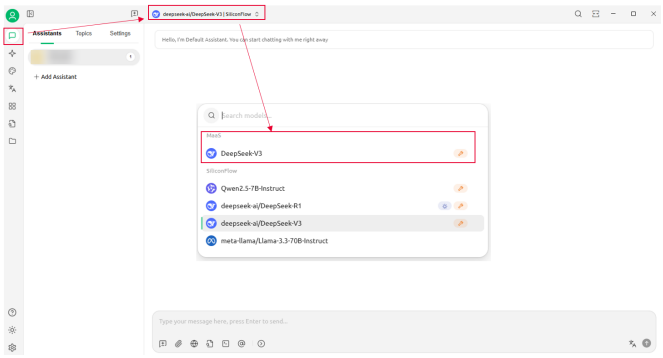
3. Add a model to Cherry Studio.
   a. In the **Models** area, click **Add**.
   b. Set the model ID, model name, and group name, and click **Add Model**.

Parameter	Description
Model ID	Model name obtained in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**
Model Name	Custom model name
Group Name	Custom group name

Step 4: Using MaaS DeepSeek in Cherry Studio

1. In the navigation pane of Cherry Studio, click  and select the configured model.

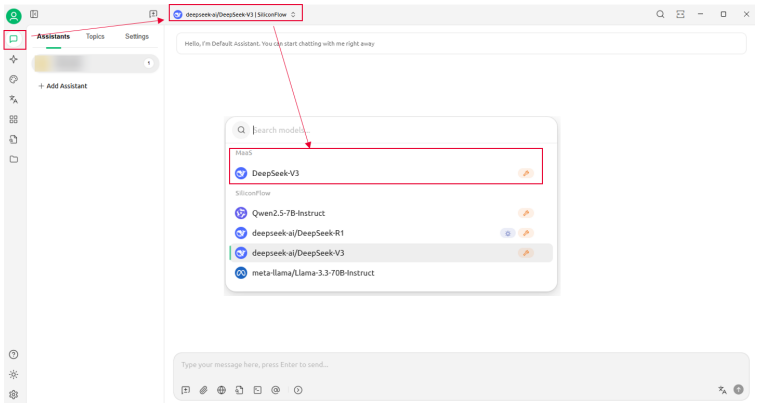
   Figure 2-3 Selecting a model

2. Enter text in the text box to start a conversation.

   Choose the model name in the top menu to switch models.

   Figure 2-4 Model Q&A example

2.2 Building a Code Editor Using a ModelArts Studio (MaaS) DeepSeek API and Cursor

This section describes how to use Cursor to call a DeepSeek model deployed on ModelArts Studio to build a code editor.

Context

Cursor is a modern code editor for developers that uses AI technology. It combines the strong features of traditional editors like VS Code with AI-powered smart coding tools. These include smart code completion, natural language programming, and code library understanding, which boost development speed. Cursor also supports popular AI models like OpenAI's GPT-4 and DeepSeek, and offers flexible customization. This makes it ideal for users from beginners to professionals.

ModelArts Studio (MaaS) deploys DeepSeek models on its platform, allowing developers use them via API calls.

Prerequisites

- You have registered a Huawei Cloud account and completed real-name authentication. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.
- You have deployed a service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

Step 1: Downloading and Installing Cursor

Download and install Cursor from the **official website**.

Step 2: Preparing for MaaS Model API Access

1. Create an API key.

   Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to **ModelArts Studio (MaaS) console** and select **CN Southwest-Guiyang1** from the top navigation bar.

   b. In the navigation pane, choose **API Keys**.

   c. On the **API Keys** page, click **Create API Key**, enter the tag and description, and click **OK**.

      The tag and description cannot be modified after the key is created.

      Table 2-3 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

   d. In the **Your Key** dialog box, copy the key and store it securely.

e. After the key is saved, click **Close**.

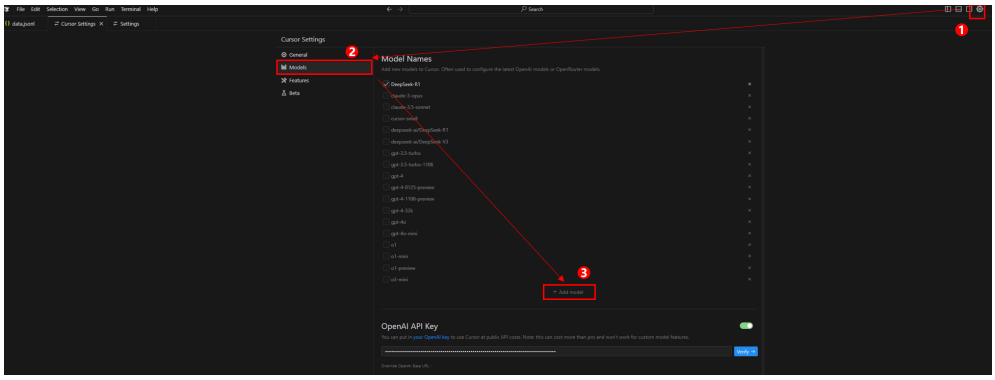
After you click **Close**, the key cannot be viewed again.

2. Use a model from **My Services**.
   a. In the navigation pane of the **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.
   b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.
   c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.
   d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cursor.

Step 3: Configuring the MaaS API in Cursor

1. Click the settings icon in the upper right corner of the Cursor platform.
2. In the **Cursor Settings** area, choose **Models** and click **Add model**.

Figure 2-5 Adding a model

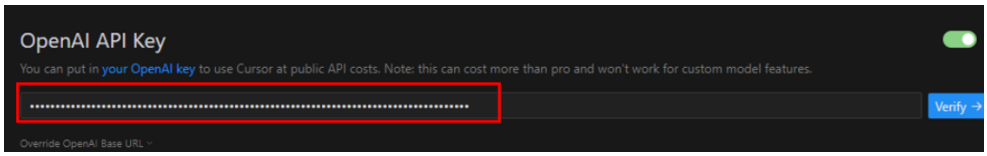


3. Enter the model name obtained in **Step 2.2** in the text box and click **Add model** on the right.
4. Select only the added MaaS model. Choosing others might cause the verification to fail.

Figure 2-6 Selecting a MaaS model

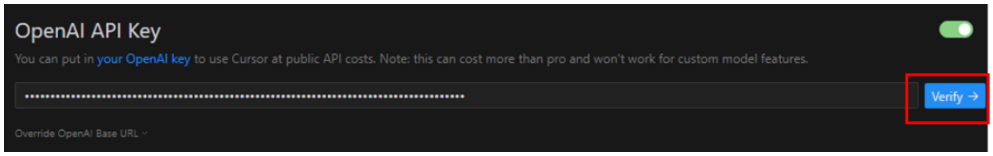
5. In the **OpenAI Key** area, enter the API key created in **Step 2.1**.

Figure 2-7 Entering the API key



6. Click **Override Openai Base URL**. Change the basic API URL to the one from **Step 2.2**, removing **/chat/completions** at the end. Then click **Save**.
7. Click **Verify** to verify the API connectivity. If no errors show, the setup is complete and you can start using the API.

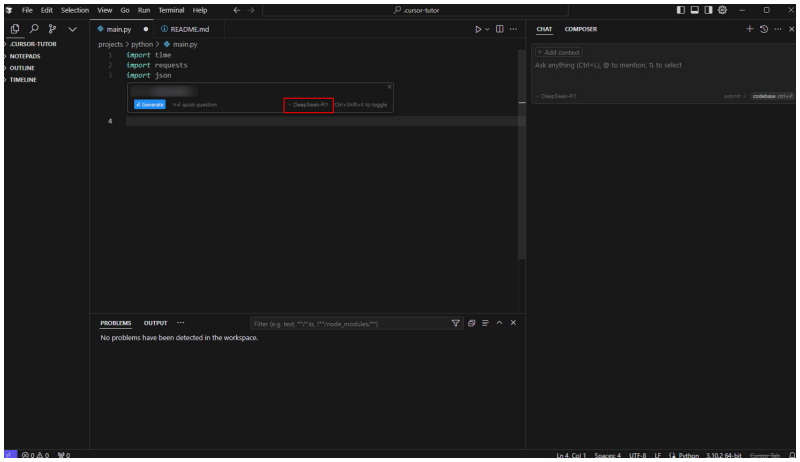
Figure 2-8 Verifying connectivity



Step 4: Using the MaaS API in Cursor to Generate Code

On the code editing page, choose the configured model in the red box for tasks like dialog, code generation, and code parsing.

Figure 2-9 Using the MaaS API



2.3 Quickly Building an AI Coding Assistant Using a ModelArts Studio (MaaS) DeepSeek API and Cline

This section describes how to use Cline to call a DeepSeek model deployed on ModelArts Studio to build an AI coding assistant.

Context

Cline is a VS Code plug-in that uses large language models (LLMs) to handle complex software development tasks. It offers a convenient and efficient coding experience. Advantages of Cline:

- Deep integration with ModelArts Studio (MaaS): Cline connects to DeepSeek model services on MaaS.
- File management and code correction: You can easily create and edit files while monitoring real-time linter and editor errors. Cline detects issues like missing imports or syntax errors, suggests fixes, and improves your coding efficiency.
- Terminal interaction and instant response: Cline includes a terminal for running commands and viewing outputs in real time. It helps you quickly find and fix server issues, keeping development smooth.
- One-stop web solution: For web tasks, Cline starts websites in a headless browser, simulates user actions, and captures screenshots and logs. This helps identify and correct runtime and visual errors, ensuring high-quality web applications.

Prerequisites

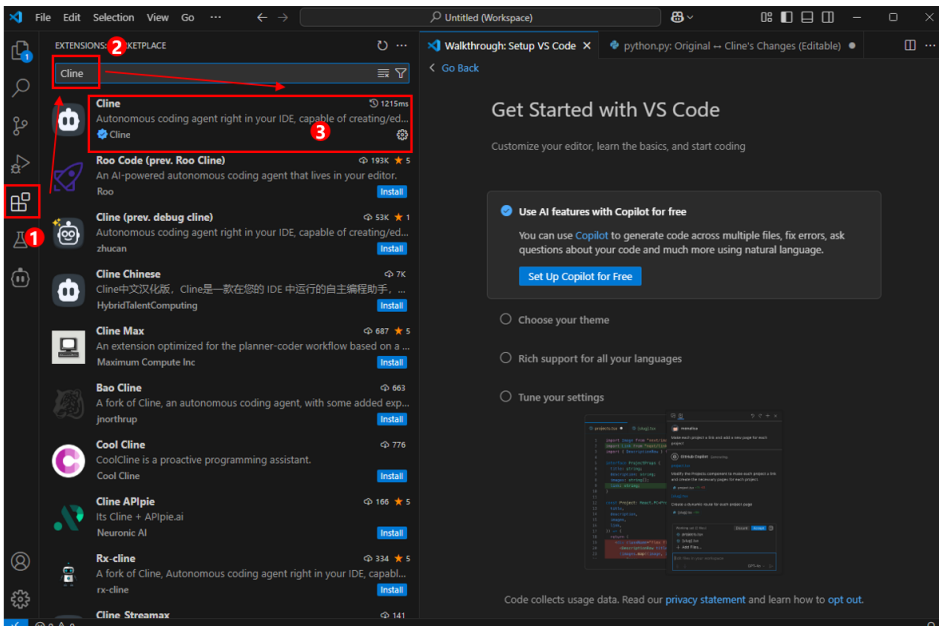
- You have registered a Huawei Cloud account and completed real-name authentication. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services** and **Real-Name Authentication**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.
- You have deployed a service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

Step 1: Installing Cline in VS Code

1. Open VS Code. Click  on the navigation pane, enter **Cline** in the search box, and click **Install**.

   If you see a small robot icon  on the left, Cline is installed.

   Figure 2-10 Installing Cline



Step 2: Preparing for MaaS Model API Access

1. Create an API key.

   Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to **ModelArts Studio (MaaS) console** and select **CN Southwest-Guiyang1** from the top navigation bar.

   b. In the navigation pane, choose **API Keys**.

   c. On the **API Keys** page, click **Create API Key**, enter the tag and description, and click **OK**.

      The tag and description cannot be modified after the key is created.

      Table 2-4 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

d. In the **Your Key** dialog box, copy the key and store it securely.
e. After the key is saved, click **Close**.
After you click **Close**, the key cannot be viewed again.

2. Use a model from **My Services**.

> **CAUTION**
>
> The Cline plug-in uses a long system prompt to enhance model code generation. This requires a model with a large context length. Use the DeepSeek-R1-671B-32K or DeepSeek-V3-671B-32K model on the **ModelArts Studio (MaaS) console**. Access fails if the context length is less than 32K.

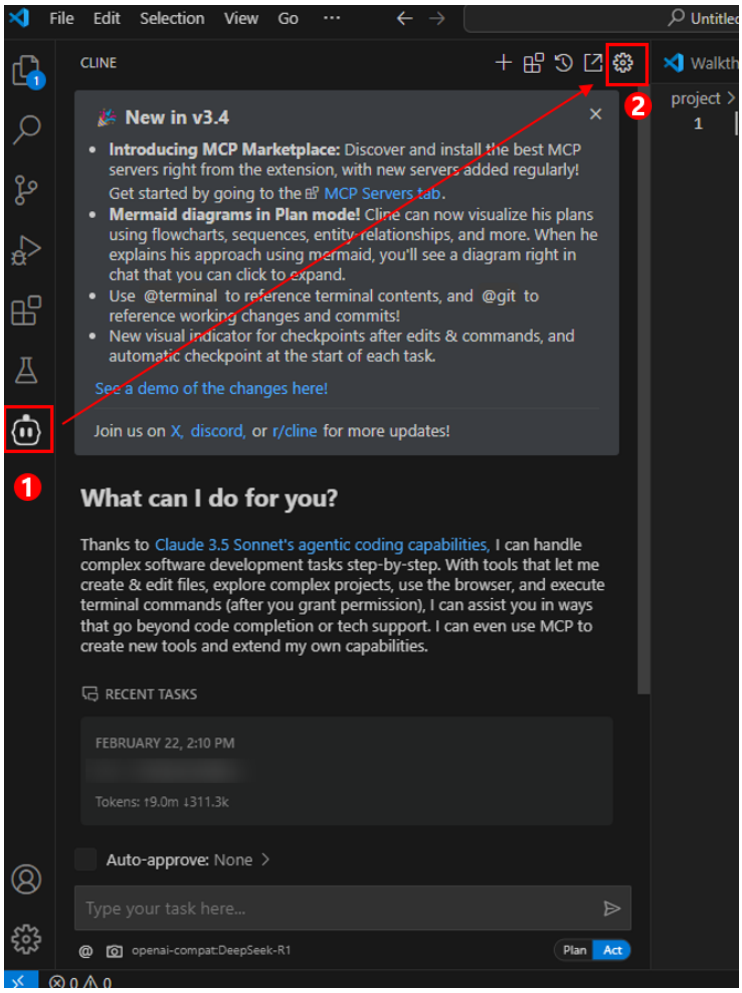
a. In the navigation pane of the **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.
b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.
c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.
d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cline.

Step 3: Configuring the MaaS API in Cline

1. Configure the MaaS model service.

a. Open VS Code, click  in the navigation pane to open the Cline plug-in, and click  in the upper right corner.

Figure 2-11 Opening the cline plug-in



b. On the **Settings** page, configure related information and click **Done**.

Table 2-5 Cline configuration

Parameter	Description
API Provider	Select **OpenAI Compatible**.
Base URL	Remove **/chat/completions** from the API URL obtained in **Step 2.2**.
API Key	API key created in **Step 2.1**.
Model ID	Model name obtained in **Step 2.2**.

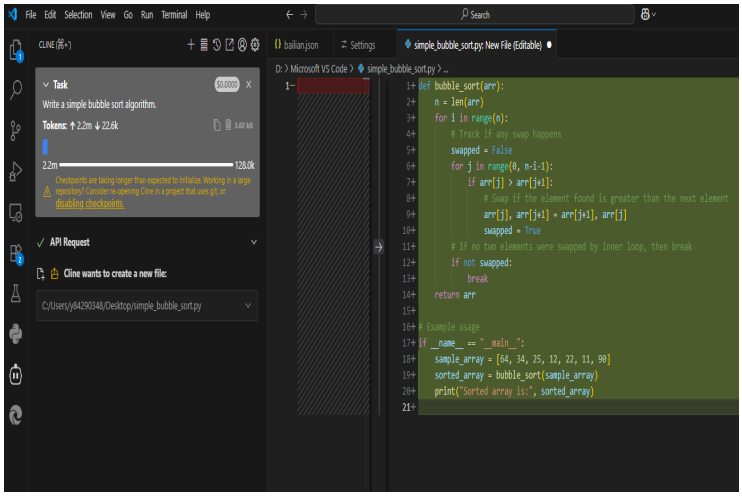
2. Use the Cline plug-in in VS Code to call the MaaS API for automatic code generation.

a. In the navigation pane of VS Code, click  .

b. Select the configured MaaS service shown in the red box in the lower left corner to generate code.

   Cline can generate, correct, and optimize code.

Figure 2-12 Code generation example



3 Permissions Management

3.1 Basic Concepts

ModelArts allows you to configure fine-grained permissions for refined management of resources and permissions. This is commonly used by large enterprises, but it is complex for individual users. It is recommended that individual users configure permissions for using ModelArts by referring to **Assigning Permissions to Individual Users for Using ModelArts**.

NOTE

If you meet any of the following conditions, read this document.

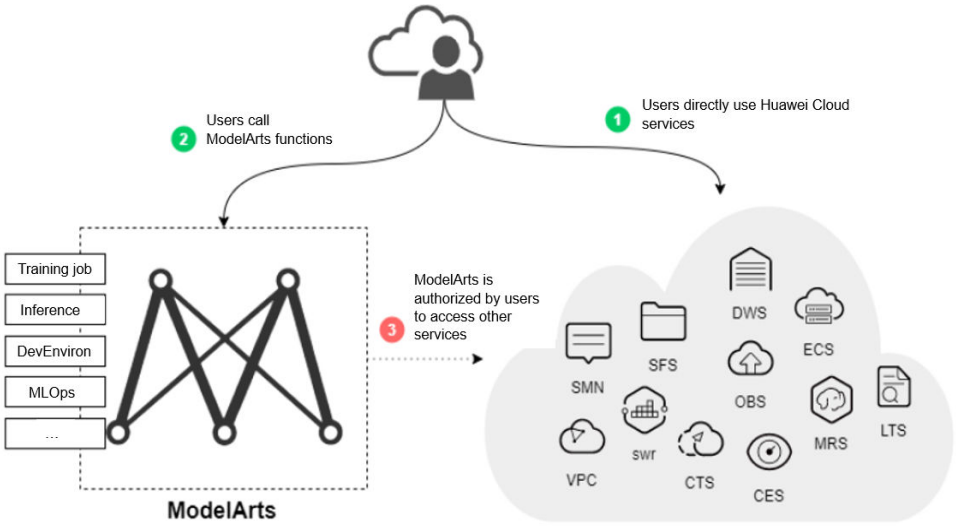
- You are an enterprise user with multiple departments. You need to specify users in different departments to access their dedicated resources. And there should be multiple roles, including administrator, algorithm developer, and application O&M engineer.
- You are an enterprise user and aim to restrict different roles to use specific functions. Logically, there should be multiple isolated environments, such as the development environment, pre-production environment, and production environment. You also want the operation permissions to vary depending on the environment. Or you need to control permissions of specific IAM user or user group.
- You are an individual user, and you have created multiple IAM users. You need to assign different ModelArts permissions to different IAM users.
- You need to understand the concepts and operations of ModelArts permissions management.

ModelArts uses Identity and Access Management (IAM) for most permissions management functions. Before reading below, learn about **Basic Concepts**. This helps you better understand this document.

To implement fine-grained permissions management, ModelArts provides permission control, agency authorization, and workspace. The following describes the details.

ModelArts Permissions and Agencies

Figure 3-1 Permissions management



Like other services, ModelArts functions are controlled by IAM permissions. For example, if you as an IAM user need to create a training job on ModelArts, you must have the **modelarts:trainJob:create** permission. For details about how to assign permissions to a user (you need to add the user to a user group and then assign permissions to the user group), see **Permissions Management**.

ModelArts must access other services for AI computing. For example, ModelArts must access OBS to read your data for training. For security purposes, ModelArts must be authorized to access other cloud services. This is agency authorization.

The following summarizes permissions management:

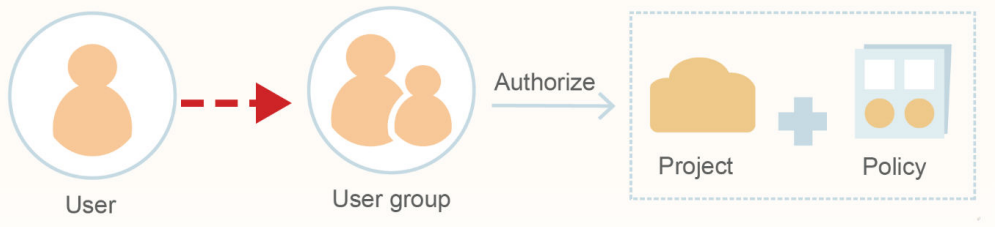
- Your access to any cloud service is controlled through IAM. You must have the permissions of the cloud service. (The required service permissions vary depending on the functions you use.)
- To use ModelArts functions, you need to grant permissions through IAM.
- ModelArts must be authorized by you to access other cloud services for AI computing.

ModelArts Permissions Management

By default, new IAM users do not have any permissions assigned. You need to add the user to a user group and grant the user group with policies, so that the users in the group can inherit the permissions. After authorization, users can perform operations on ModelArts based on permissions.

CAUTION

- ModelArts is a project-level service deployed and accessed in specific physical regions. When you authorize an agency, you can set the scope for the permissions you select to all resources, enterprises projects, or region-specific projects. If you specify region-specific projects, the selected permissions will be applied to resources in these projects.
- ModelArts supports enterprise projects. You can specify an enterprise project when selecting the authorization scope. For details, see **Creating a User Group and Assigning Permissions**.



When assigning permissions to a user group, IAM does not directly assign specific permissions to the user group. Instead, IAM needs to add the permissions to a policy and then assign the policy to the user group. To facilitate user permissions management, each cloud service provides some preset policies for you to directly use. If the preset policies cannot meet your requirements of fine-grained permissions management, you can customize policies.

Table 3-1 lists all the preset system-defined policies supported by ModelArts.

Table 3-1 System-defined policies supported by ModelArts

Policy	Description	Type
ModelArts FullAccess	Administrator permissions for ModelArts. Users granted these permissions can operate and use ModelArts.	System-defined policy
ModelArts CommonOperations	Common user permissions for ModelArts. Users granted these permissions can operate and use ModelArts, but cannot manage dedicated resource pools.	System-defined policy
ModelArts Dependency Access	Permissions on dependent services for ModelArts	System-defined policy

Generally, ModelArts FullAccess is assigned only to administrators. If fine-grained management is not required, assigning ModelArts CommonOperations to all users will meet the development requirements of most small teams. If you want to customize policies for fine-grained permissions management, see **IAM**.

NOTE

When you assign ModelArts permissions to a user, the system does not automatically assign the permissions of other services to the user. This ensures security and prevents unexpected unauthorized operations. In this case, however, you must separately assign permissions of different services to users so that they can perform some ModelArts operations.

For example, if an IAM user needs to use OBS data for training and the ModelArts training permission has been configured for the IAM user, the IAM user still needs to be assigned with the OBS read, write, and list permissions. The OBS list permission allows you to select the training data path on ModelArts. The read permission is used to preview data and read data for training. The write permission is used to save training results and logs.

- For individual users or small organizations, it is a good practice to configure the **Tenant Administrator** policy that applies to global services for IAM users. In this way, IAM users can obtain all user permissions except IAM. However, this may cause security issues. (For an individual user, its default IAM user belongs to the **admin** user group and has the **Tenant Administrator** permission.)
- If you want to restrict user operations, configure the minimum permissions of OBS for ModelArts users. For details, see **OBS Permissions Management**. For details about fine-grained permissions management of other cloud services, see the corresponding cloud service documents.

ModelArts Agency Authorization

ModelArts must be authorized by users to access other cloud services for AI computing. In the IAM permission system, such authorization is performed through agencies.

For details about the basic concepts and operations of agencies, see **Cloud Service Delegation**.

To simplify agency authorization, ModelArts supports automatic agency authorization configuration. You only need to configure an agency for yourself or specified users on the **Permission Management** page of the ModelArts console.

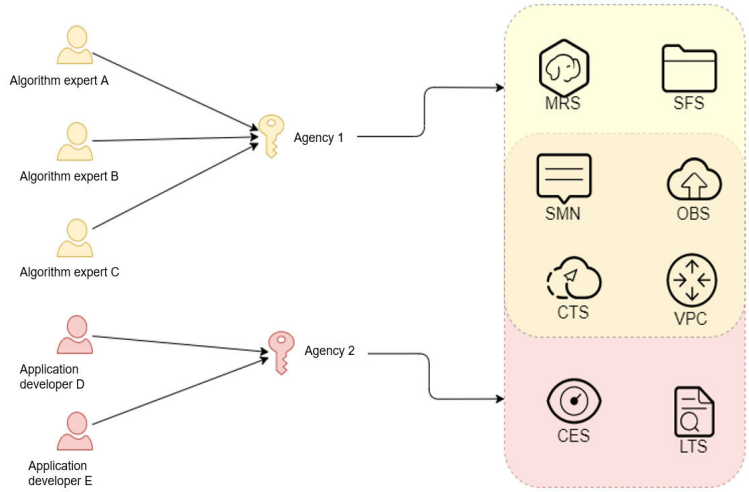
NOTE

- Only users with the IAM agency management permission can perform this operation. Generally, members in the IAM admin user group have this permission.
- ModelArts agency authorization is region-specific, which means that you must perform agency authorization in each region you use.

On the **Permission Management** page of the ModelArts console, after you click **Add Authorization**, you can configure an agency for a specific user or all users. Generally, an agency named **modelarts_agency_**<*Username*>*_Random ID* is created by default. In the **Permissions** area, you can select the preset permission configuration or select the required policies. If both options cannot meet your requirements, you can create an agency on the IAM management page (you need to delegate ModelArts to access your resources), and then use an existing agency instead of adding an agency on the **Add Authorization** page.

ModelArts associates multiple users with one agency. This means that if two users need to configure the same agency, you do not need to create an agency for each user. Instead, you only need to configure the same agency for the two users.

Figure 3-2 Mapping between users and agencies



NOTE

Each user can use ModelArts only after being associated with an agency. However, even if the permissions assigned to the agency are insufficient, no error is reported when the API is called. An error occurs only when the system uses unauthorized functions. For example, you enable message notification when creating a training job. Message notification requires SMN authorization. However, an error occurs only when messages need to be sent for the training job. The system ignores some errors, and other errors may cause job failures. When you implement permission minimization, ensure that you will still have sufficient permissions for the required operations on ModelArts.

Strict Authorization

In strict authorization mode, explicit authorization by the account administrator is required for IAM users to access ModelArts. The administrator can add the required ModelArts permissions to common users through authorization policies.

In non-strict authorization mode, IAM users can use ModelArts without explicit authorization. The administrator needs to configure the deny policy for IAM users to prevent them from using some ModelArts functions.

The administrator can change the authorization mode on the **Permission Management** page.

NOTICE

The strict authorization mode is recommended. In this mode, IAM users must be authorized to use ModelArts functions. In this way, the permission scope of IAM users can be accurately controlled, minimizing permissions granted to IAM users.

Managing Resource Access Using Workspaces

Workspace enables enterprise customers to split their resources into multiple spaces that are logically isolated and to manage access to different spaces. As an enterprise user, you can submit the request for enabling the workspace function to your technical support manager.

After workspace is enabled, a default workspace is created. All resources you have created are in this workspace. A workspace is like a ModelArts twin. You can switch between workspaces in the upper left corner of the ModelArts console. Jobs in different workspaces do not affect each other.

When creating a workspace, you must bind it to an enterprise project. Multiple workspaces can be bound to the same enterprise project, but one workspace cannot be bound to multiple enterprise projects. You can use workspaces for refined restrictions on resource access and permissions of different users. The restrictions are as follows:

- Users must be authorized to access specific workspaces (this must be configured on the pages for creating and managing workspaces). This means that access to AI assets such as datasets and algorithms can be managed using workspaces.
- In the preceding permission authorization operations, if you set the scope to enterprise projects, the authorization takes effect only for workspaces bound to the selected projects.

NOTE

- Restrictions on workspaces and permission authorization take effect at the same time. That is, a user must have both the permission to access the workspace and the permission to create training jobs (the permission applies to this workspace) so that the user can submit training jobs in this workspace.
- If you have enabled an enterprise project but have not enabled a workspace, all operations are performed in the default enterprise project. Ensure that the permissions on the required operations apply to the default enterprise project.
- The preceding restrictions do not apply to users who have not enabled any enterprise project.

Summary

Key features of ModelArts permissions management:

- If you are an individual user, you do not need to consider fine-grained permissions management. Your account has all permissions to use ModelArts by default.
- All functions of ModelArts are controlled by IAM. You can use IAM authorization to implement fine-grained permissions management for specific users.
- All users (including individual users) can use specific functions only after agency authorization on ModelArts (**Settings** > **Add Authorization**). Otherwise, unexpected errors may occur.
- If you have enabled the enterprise project function, you can also enable ModelArts workspace and use both basic authorization and workspace for refined permissions management.

3.2 Permission Management Mechanisms

3.2.1 IAM

This section describes the IAM permission configurations for all ModelArts functions.

IAM Permissions

If you need to assign different permissions to employees in your enterprise to access your purchased ModelArts resources, Identity and Access Management (IAM) is a good choice for fine-grained permissions management. IAM provides identity authentication, fine-grained permissions management, and access control. IAM helps you secure access to your cloud resources. If your Huawei account can meet your requirements and you do not need an IAM account to manage user permissions, skip this chapter.

IAM is a free service. You only pay for the resources in your account.

With IAM, you can assign permissions to control users' access to specific resources. For example, if the software developers in your enterprise need to own permissions to use ModelArts, yet you do not want them to own high-risk operation permissions such as deleting ModelArts, you can grant permissions using IAM to limit their permission on ModelArts.

For details about IAM, see **What is IAM?**.

Role/Policy-based Authorization

ModelArts supports role/policy-based authorization. By default, new IAM users do not have any permissions. You need to add a user to one or more groups, and assign permissions policies or roles to these groups. Users inherit permissions of the groups to which they are added. This process is called authorization. The users then inherit permissions from the groups and can perform specified operations on cloud services.

ModelArts is a project-level service deployed for specific regions. When you set **Scope** to **Region-specific projects** and select the specified projects (for example, **ap-southeast-2**) in the specified regions (for example, **AP-Bangkok**), the users only have permissions for ModelArts resources in the selected projects. If you set **Scope** to **All resources**, the users have permissions for ModelArts resources in all region-specific projects. When accessing ModelArts, the users need to switch to a region where they have been authorized to use cloud services.

Table 3-2 lists all system-defined policies supported by ModelArts. If preset ModelArts permissions cannot meet your requirements, create a custom policy by referring to **Policy Fields in JSON Format**.

Table 3-2 System-defined policies supported by ModelArts

Policy	Description	Type
ModelArts FullAccess	All permissions for ModelArts administrators	System-defined policy

Policy	Description	Type
ModelArts CommonOperations	All operation permissions for ModelArts common users, which do not include managing dedicated resource pools.	System-defined policy
ModelArts Dependency Access	Permissions on dependent services for ModelArts	System-defined policy

ModelArts depends on other cloud services. To check or view the cloud services, configure the corresponding permissions on the ModelArts console, as shown in the following table.

Table 3-3 Roles or policies that are required for performing operations on the ModelArts console

Console Function	Dependency	Role/Policy Required
Data management (dataset/data labeling/data processing)	Object Storage Service (OBS)	OBS Administrator
	Data Lake Insight (DLI)	DLI FullAccess
	MapReduce Service (MRS)	MRS Administrator
	GaussDB(DWS)	DWS Administrator
	Cloud Trace Service (CTS)	CTS Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Development environment notebook/ Image management/ Elastic node server	OBS	OBS Administrator
	Cloud Secret Management Service (CSMS)	CSMS ReadOnlyAccess
	CTS	CTS Administrator
	Elastic Cloud Server (ECS)	ECS FullAccess
	Software Repository for Container (SWR)	SWR Admin
	Scalable File Service (SFS)	SFS Turbo FullAccess

Console Function	Dependency	Role/Policy Required
	Application Operations Management (AOM)	AOM FullAccess
	Key Management Service (KMS)	KMS CMKFullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Algorithm management/ Training management/ Workflow/ ExeML	OBS	OBS Administrator
	Simple Message Notification (SMN)	SMN Administrator
	CTS	CTS Administrator
	Enterprise Project Management Service (EPS)	EPS FullAccess
	SFS Turbo	SFS ReadOnlyAccess
	SWR	SWR Admin
	AOM	AOM FullAccess
	KMS	KMS CMKFullAccess
	Virtual Private Cloud (VPC)	VPC FullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Model management/ Real-time service/Batch service/Edge service/Edge deployment dedicated resource pool	OBS	OBS Administrator
	Cloud Eye	CES ReadOnlyAccess
	SMN	SMN Administrator
	EPS	EPS FullAccess
	CTS	CTS Administrator
	Log Tank Service (LTS)	LTS FullAccess
	Virtual Private Cloud (VPC)	VPC FullAccess
	SWR	SWR Admin
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access

Console Function	Dependency	Role/Policy Required
AI Gallery	OBS	OBS Administrator
	CTS	CTS Administrator
	SWR	SWR Admin
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Elastic cluster (including standard and lite resource pools)	CTS	CTS Administrator
	Cloud Container Engine (CCE)	CCE Administrator
	Bare Metal Server (BMS)	BMS FullAccess
	Image Management Service (IMS)	IMS FullAccess
	Data Encryption Workshop (DEW)	DEW KeypairReadOnlyAccess
	VPC	VPC FullAccess
	ECS	ECS FullAccess
	SFS	SFS Turbo FullAccess
	OBS	OBS Administrator
	AOM	AOM FullAccess
	Tag Management Service (TMS)	TMS FullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
	Billing Center	BSS Administrator
	Elastic Volume Service (EVS)	EVS FullAccess

If system-defined policies cannot meet your requirements, you can create a custom policy. For details about the actions supported by custom policies, see **ModelArts Resource Permissions**.

To create a custom policy, choose either visual editor or JSON.

- Visual editor: Select cloud services, actions, resources, and request conditions without the need to know policy syntax.

- JSON: Create a JSON policy or edit an existing one.

For details, see **Creating a Custom Policy**. This section provides examples of common custom ModelArts policies.

- Example 1: Grant permission to manage images.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- Example 2: Grant permission to deny creating, updating, and deleting a dedicated resource pool.

  A policy with only "Deny" permissions must be used together with other policies. If the permissions granted to an IAM user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "swr:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "smn:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
            ],
            "Effect": "Deny"
        }
    ]
}
```

- Example 3: Create a custom policy containing multiple actions.

  A custom policy can contain actions of multiple services that are of the global or project-level type. The following is an example policy containing actions of multiple services:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```



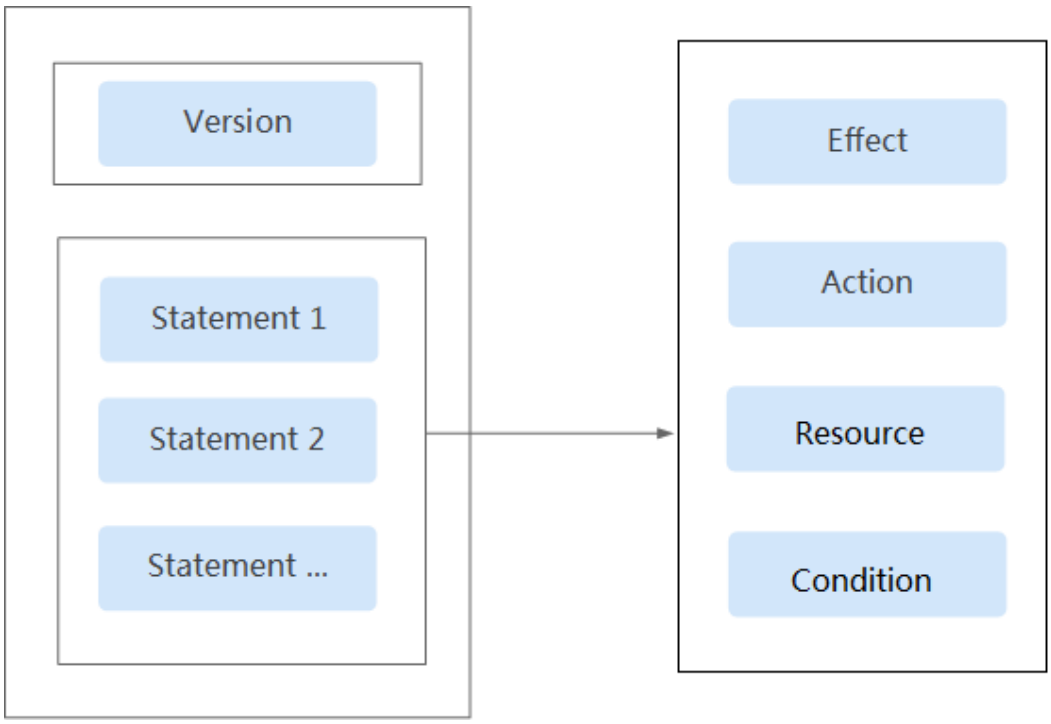
```
            "modelarts:service:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lts:logs:list"
        ]
    }
  ]
}
```

Policy Fields in JSON Format

Policy Structure

A policy consists of a version and one or more statements (indicating different actions).

Figure 3-3 Policy structure



Policy Parameters

The following describes policy parameters. You can create custom policies by specifying the parameters. For details, see **Custom Policy Use Cases**.

Table 3-4 Policy parameters

Parameter	Description	Value
Version	Policy version	**1.1**: indicates policy-based access control.

Parameter		Description	Value
Statement: authorization statement of a policy	Effect	Whether to allow or deny the operations defined in the action	• **Allow**: indicates the operation is allowed. • **Deny**: indicates the operation is not allowed. **NOTE** If the policy used to grant user permissions contains both **Allow** and **Deny** for the same action, **Deny** takes precedence.
	Action	Operation to be performed on the service	Format: "*Service name*:*Resource type*:*Action*". Wildcard characters (*) are supported, indicating all options. Example: **modelarts:notebook:list** indicates the permission to view a notebook instance list. **modelarts** indicates the service name, **notebook** indicates the resource type, and **list** indicates the operation. View all actions of a service in its *API Reference*.
	Condition	Condition for a policy to take effect, including **condition keys and operators**	Format: "*Condition operator*:{*Condition key*:[*Value 1*,*Value 2*]}" If you set multiple conditions, the policy takes effect only when all the conditions are met. Example: **StringEndWithIfExists": {"g:UserName":["specialCharacter"]}**: The statement is valid for users whose names end with **specialCharacter**.
	Resource	Resources on which a policy takes effect	Format: *Service name*:<*Region*>:<*Account ID*>:*Resource type*:*Resource path*. Asterisks (*) are supported for resource type, indicating all resources. **NOTE** ModelArts authorization does not allow you to specify a resource path.

ModelArts Resource Types

Administrators can specify the scope based on ModelArts resource types. The following table lists the resource types supported by ModelArts:

Table 3-5 Resource types supported by ModelArts role/policy-based authorization

Resource Type	Description
notebook	Notebook instances in DevEnviron
exemlProject	ExeML projects
exemlProjectInf	ExeML-powered real-time inference service
exemlProjectTrain	ExeML-powered training jobs
exemlProjectVersion	ExeML project version
workflow	Workflow
pool	Dedicated resource pool
network	Networking of a dedicated resource pool
trainJob	Training job
trainJobLog	Runtime logs of a training job
trainJobInnerModel	Preset model
model	Models
service	Real-time service
nodeservice	Edge service
workspace	Workspace
dataset	Dataset
dataAnnotation	Dataset labels
aiAlgorithm	Algorithm for training jobs
image	Image
devserver	Elastic BMS

ModelArts Resource Permissions

For details, see "Permissions Policies and Supported Actions" in *ModelArts API Reference*.

- **Data Management Permissions**
- **DevEnviron Permissions**
- **Training Job Permissions**
- **Model Management Permissions**
- **Service Management Permissions**

3.2.2 Dependencies and Agencies

Function Dependency

Function Dependency Policies

When using ModelArts, you may be required to use other cloud services. For example, before submitting a training job, you must select OBS paths for storing the dataset and logs, respectively. Therefore, when configuring fine-grained authorization policies for a user, the administrator must configure dependent permissions so that the user can use required functions.

NOTE

- If you use ModelArts as the root user (default IAM user with the same name as the account), the root user has all permissions by default.
- Ensure that the current user has the dependent policy permissions for agency authorization. For example, if you want to grant the SWR Admin permission to a ModelArts agency, ensure that you have the SWR Admin permission.

Table 3-6 Basic configuration

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Global configuration	IAM	iam:users:listUsers	Obtain a user list. This action is required by the administrator only.
Basic function	IAM	iam:tokens:assume	(Mandatory) Use an agency to obtain temporary authentication credentials.
Basic function	BSS	bss:balance:view	Show the balance of the current account on the page after resources are created on the ModelArts console.

Table 3-7 Managing workspaces

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Workspace	IAM	iam:users:listUsers	Authorize an IAM user to use a workspace.
	ModelArts	modelarts:*:delete*	Clear resources in a workspace when deleting it.

Table 3-8 Managing notebook instances

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Lifecycle management of development environment instances	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get	Start, stop, create, delete, and update an instance.
	AOM	aom:metric:get aom:metric:list aom:alarm:list	
	VPC	vpc:securityGroups:get vpc:vpcs:list vpc:securityGroups:get vpc:vpcs:list	

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Dynamically mounting storage	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	Dynamically mount storage.
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	
Image management	ModelArts	modelarts:image:register modelarts:image:listGroup	Register and view an image on the **Image Management** page.
Saving an image	SWR	SWR Admin	The **SWR Admin** policy contains the maximum scope of SWR permissions, which can be used to: • Save a running development environment instance as an image. • Create a notebook instance using a custom image.
Using the SSH function	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	Configure a login key for a notebook instance.
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list kps:domainKeypairs:createkmskey	
	KMS	kms:cmk:list	

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Mounting an SFS Turbo file system	SFS Turbo	SFS Turbo FullAccess	Read and write an SFS directory as an IAM user. Mount an SFS file system that is not created by you to a notebook instance using a dedicated resource pool.
Viewing all Instances	ModelArts	modelarts:notebook:listAllNotebooks	View development environment instances of all users on the ModelArts management console. This action is required by the development environment instance administrator.
	IAM	iam:users:listUsers	
Local VS Code plug-in or PyCharm Toolkit	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas (This policy is required if the **workspace** function is enabled.)	Access a notebook instance from local VS Code and submit training jobs.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	OBS	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 obs:object:ModifyObjectMetaData	
	IAM	iam:projects:listProjects	Obtain an IAM project list through local PyCharm for access configurations.

Table 3-9 Elastic node server

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Elastic node server lifecycle management	ModelArts	modelarts:devserver:create modelarts:devserver:listByUser modelarts:devserver:list modelarts:devserver:get modelarts:devserver:delete modelarts:devserver:start modelarts:devserver:stop modelarts:devserver:sync	Create, start, and stop an instance, obtain the instance list, obtain all instances of a tenant, obtain instance details, and synchronize instance status.
	ECS	ecs:serverKeypairs:createecs:*:get	
	IAM	iam:users:getUser iam:users:listUsers iam:projects:listProjects	
	VPC	vpc.*.list	
	EPS	eps.*.list	
	EVS	evs.*.list	
	IMS	ims.*.list ims.*.get	

Table 3-10 Managing training jobs

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Training management	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	Create a training job and view training logs.
		modelarts:workspace:getQuota	Obtain a workspace quota. This policy is required if **workspace** function is enabled.
		modelarts:tag:list	Use Tag Management Service (TMS) in a training job.
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	Use the configured agency authorization.
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	Use SFS Turbo in a training job.
	SWR	SWR Admin	Use a custom image to create a training job.
	SMN	smn:topic:publish smn:topic:list	Notify training job status changes through SMN.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	OBS	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 obs:object:ModifyObjectMetaData	Run a training job using a dataset in an OBS bucket.

Table 3-11 Using workflows

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Using a dataset	ModelArts	modelarts:dataset:getDataset modelarts:dataset:createDataset modelarts:dataset:createDatasetVersion modelarts:dataset:createImportTask modelarts:dataset:updateDataset modelarts:processTask:createProcessTask modelarts:processTask:getProcessTask modelarts:dataset:listDatasets	Use ModelArts datasets in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Model management	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	Manage ModelArts models in a workflow.
Deploying a service	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	Manage ModelArts real-time services in a workflow.
Training jobs	ModelArts	modelarts:trainJob:get modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJobVersion:list modelarts:trainJobVersion:create modelarts:trainJob:delete modelarts:trainJobVersion:delete modelarts:trainJobVersion:stop	Manage ModelArts training jobs in a workflow.
Workspace	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	Use ModelArts workspaces in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing data	OBS	obs:bucket:ListAllMybuckets (Obtaining a bucket list) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:GetBucketLocation (Obtaining the bucket location) obs:object:GetObject (Obtaining object content and metadata) obs:object:GetObjectVersion (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:object:DeleteObject (Deleting an object or batch deleting objects) obs:object:DeleteObjectVersion (Deleting an object or batch deleting objects) obs:object:ListMultipartUpload-Parts (Listing uploaded parts) obs:object:AbortMultipartUpload (Aborting multipart uploads) obs:object:GetObjectAcl (Obtaining an object ACL) obs:object:GetObjectVersionAcl (Obtaining an object ACL) obs:bucket:PutBucketAcl (Configuring a bucket ACL) obs:object:PutObjectAcl (Configuring an object ACL)	Use OBS data in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Executing a workflow	IAM	iam:users:listUsers (Obtaining users) iam:agencies:getAgency (Obtaining details about a specified agency) iam:tokens:assume (Obtaining an agency token)	Call other ModelArts services when a workflow is running.
Integrating DLI	DLI	dli:jobs:get (Obtaining job details) dli:jobs:list_all (Viewing a job list) dli:jobs:create (Creating a job)	Integrate DLI into a workflow.
Integrating MRS	MRS	mrs:job:get (Obtaining job details) mrs:job:submit (Creating and executing a job) mrs:job:list (Viewing a job list) mrs:job:stop (Stopping a job) mrs:job:batchDelete (Batch deleting jobs) mrs:file:list (Viewing a file list)	Integrate MRS into a workflow.

Table 3-12 Model management

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Model management	SWR	SWR Admin	Use a custom engine when you import a model from a custom image or OBS. SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	OBS	obs:bucket:ListAllMybuckets (Obtaining a bucket list) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:GetBucketLocation (Obtaining the bucket location) obs:object:GetObject (Obtaining object content and metadata) obs:object:GetObjectVersion (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:object:DeleteObject (Deleting an object or batch deleting objects) obs:object:DeleteObjectVersion (Deleting an object or batch deleting objects) obs:object:ListMultipartUpload-Parts (Listing uploaded parts) obs:object:AbortMultipartUpload (Aborting multipart uploads) obs:object:GetObjectAcl (Obtaining an object ACL) obs:object:GetObjectVersionAcl (Obtaining an object ACL) obs:bucket:PutBucketAcl (Configuring a bucket ACL) obs:object:PutObjectAcl (Configuring an object ACL)	Import a model from OBS. Specify an OBS path for model conversion.

Table 3-13 Managing service deployment

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Real-time services	LTS	lts:logs:list (Obtaining the log list)	Show LTS logs.
	OBS	obs:bucket:GetBucketPolicy (Obtaining a bucket policy) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListAllMyBuckets (Obtaining a bucket list) obs:bucket:PutBucketPolicy (Configuring a bucket policy) obs:bucket:DeleteBucketPolicy (Deleting a bucket policy)	Mount external volumes to a container when services are running.
Batch services	OBS	obs:object:GetObject (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:bucket:CreateBucket (Creating a bucket) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:ListAllMyBuckets (Obtaining a bucket list)	Create batch services and perform batch inference.
Edge services	CES	ces:metricData:list: (Obtaining metric data)	View monitoring metrics.
	IEF	ief:deployment:delete (Deleting a deployment)	Manage edge services.
AOM metric alarm events	AOM	aom:alarm:list	View AOM monitoring information.

Table 3-14 Managing datasets

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing datasets and labels	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUpload-Parts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	Manage datasets in OBS. Label OBS data. Create a data management job.
Managing table datasets	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describeTable	Manage DLI data in a dataset.
Managing table datasets	GaussDB(DWS)	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	Manage DWS data in a dataset.
Managing table datasets	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	Manage MRS data in a dataset.
Auto labeling	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	Enable auto labeling.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Team labeling	IAM	iam:projects:listProjects (Obtaining tenant projects) iam:users:listUsers (Obtaining users) iam:agencies:createAgency (Creating an agency) iam:quotas:listQuotasForProject (Obtaining the quotas of a project)	Manage labeling teams.

Table 3-15 Managing resources

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing resource pools	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	Create, renew, and unsubscribe from a resource pool.
	CCE	cce:cluster:list cce:cluster:get	Obtain the CCE cluster list, cluster details, and cluster certificates.
	KMS	kms:cmk:list kms:cmk:getMaterial	Obtain the key pairs created by the user.
	AOM	aom:metric:get	Obtain the monitoring data of a resource pool.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion	Obtain AI diagnostic logs.
	ECS	ecs:availabilityZones:list ecs:cloudServerFlavors:get ecs:cloudServerQuotas:get ecs:quotas:get ecs:serverKeypairs:list	Obtain the AZs, specifications, and quotas, and configure keys.
	EVS	evs:types:get evs:quotas:get	Query EVS disk types and quotas.
	BMS	bms:serverFlavors:get	Query BMS specifications. Dependent permissions must be configured in the IAM project view.
	DEW	kps:domainKeypairs:list	Configure a key pair. Dependent permissions must be configured in the IAM project view.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Network management	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete vpc:securityGroups:get	Create and delete ModelArts networks, and interconnect VPCs.
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	Interconnect your network with SFS Turbo.

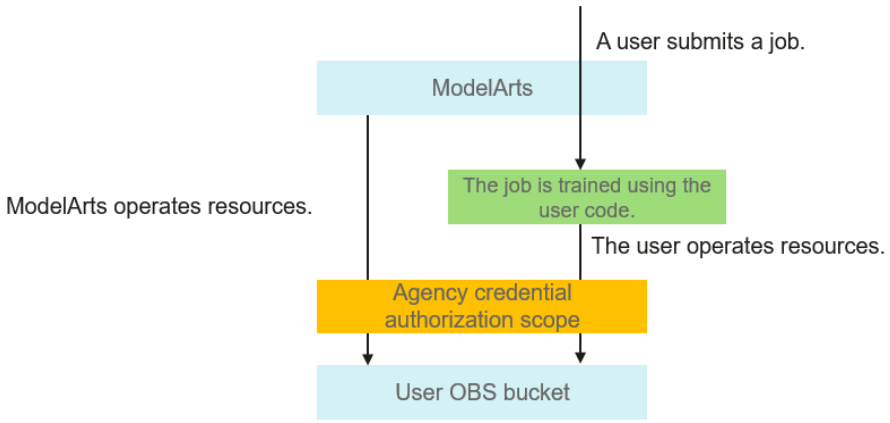
Application Scenario	Dependent Service	Dependent Policy	Supported Function
Edge resource pool	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	Add, delete, modify, and search for edge pools.

Agency authorization

To simplify operations when you use ModelArts, certain operations are automatically performed on the ModelArts backend, for example, downloading the datasets in an OBS bucket to a workspace before a training job is started and dumping training job logs to the OBS bucket.

ModelArts does not save your token authentication credentials. Before performing operations on your resources (such as OBS buckets) in a backend asynchronous job, you are required to explicitly authorize ModelArts through an IAM agency. ModelArts will use the agency to obtain a temporary authentication credential for performing operations on your resources. For details, see **Adding Authorization**.

Figure 3-4 Agency authorization



As shown in **Figure 3-4**, after authorization is configured on ModelArts, ModelArts uses the temporary credential to access and operate your resources, relieving you

from some complex and time-consuming operations. The agency credential will also be synchronized to your jobs (including notebook instances and training jobs). You can use the agency credential to access your resources in the jobs.

You can use either of the following methods to authorize ModelArts using an agency:

One-click authorization

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 mode, 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 the second method.

Custom authorization

The administrator creates different agency authorization policies for different users in IAM, and configures the created agency for ModelArts users. When creating an agency for an IAM user, the administrator specifies the minimum permissions for the agency based on the user's permissions to control the resources that the user can access when they use ModelArts. For details, see **Assigning Basic Permissions for Using ModelArts**.

Risks in Unauthorized Operations

The agency authorization of a user is independent. Theoretically, the agency authorization scope of a user can be beyond the authorization scope of the authorization policy configured for the user group. Any improper configuration will result in unauthorized operations.

To prevent unauthorized operations, only a tenant administrator is allowed to configure agencies for users to ensure the security of agency authorization.

Minimal Agency Authorization

When configuring agency authorization, an administrator must strictly control the authorization scope.

ModelArts asynchronously and automatically performs operations such as job preparation and clearing. The required agency authorization is within the basic authorization scope. If you use only some functions of ModelArts, the administrator can filter out the basic permissions that are not used according to the agency authorization configuration. Conversely, if you need to obtain resource permissions beyond the basic authorization scope in a job, the administrator can add new permissions to the agency authorization configuration. In a word, the agency authorization scope must be minimized and customized based on service requirements.

Basic Agency Authorization Scope

To customize the permissions for an agency, select permissions based on your service requirements.

Table 3-16 Basic agencies and authorizations in the development environment

Application Scenario	Dependent Service	Agency Authorization	Description
Performing operations on OBS data in a notebook instance	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	You can use either of the following methods to perform operations on OBS data in a notebook instance: • Use ModelArts SDK to perform operations on OBS data. • Use the notebook file upload function to perform operations on OBS data. • On the ModelArts console, add an OBS bucket to the **/data** directory of a notebook instance, and perform operations on OBS data in file mode.
Reporting notebook instance events	AOM	aom:alarm:put	During the lifecycle of a notebook instance, some events are reported to the AOM account. For details, see **Viewing Notebook Events**.
Interconnecting VPC with a notebook instance	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnets:get	Add a NIC in the notebook instance for interconnecting with specified services in the VPC.
Connecting to a notebook instance through VS Code with one click	ModelArts	modelarts:notebook:get	Manage notebook instance details. Click **VS Code** to obtain the instance details and easily modify the instance information by writing the SSH configuration to the local VS Code.

Application Scenario	Dependent Service	Agency Authorization	Description
Stopping a notebook instance	ModelArts	modelarts:notebook:stop	Stops a running notebook instance.
Updating the auto stop time of a notebook instance	ModelArts	modelarts:notebook:updateStopPolicy	Update the auto stop time of a notebook instance.
MindInsight/ TensorBoard used in OBS parallel file systems	ModelArts	modelarts:notebook:umountStorage modelarts:notebook:getMountedStorage modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage	If MindInsight or TensorBoard is enabled in a notebook instance, and you need to access the OBS parallel file system, configure the permissions on the left.

Table 3-17 Basic agency authorization for training jobs

Application Scenario	Dependent Service	Agency Authorization	Description
Accessing OBS files for training jobs	OBS	obs:bucket:HeadBucket obs:bucket:GetBucketLocation obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:GetObject obs:object:GetObjectVersion obs:object:GetObjectAcl obs:object:GetObjectVersionAcl	You need to obtain OBS operation permissions when configuring a training job, including the code directory, input, output, and the OBS bucket path for storing logs.

Application Scenario	Dependent Service	Agency Authorization	Description
Starting a training job using a custom container image.	SWR	SWR Admin	When a training job is started using a custom container image, you need to obtain a temporary login command of the SWR container image to download the container image. SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.
Notification of training job status changes	SMN	smn:template:list smn:template:create smn:topic:list smn:topic:publish	To configure training job status change notifications, you must have the SMN operation permissions to send template-based notifications.
Mounting SFS Turbo to a training job	SFS Turbo	SFS Turbo ReadOnlyAccess	To mount SFS Turbo to a training job, you must have the SFS Turbo read permission to obtain its details by ID.
Reporting audit logs	CTS	CTS Administrator	Configure the CTS permission to report events. CTS does not support fine-grained permissions for event reporting. Therefore, you need to configure the administrator permission.

Table 3-18 Basic agency authorization for inference deployment

Application Scenario	Dependent Service	Agency Authorization	Description
Real-time services	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	Configure LTS for reporting logs of real-time services.
Batch services	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	This parameter is mandatory when a batch service is used.
Edge services	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	This parameter is mandatory when an edge service is used. The edge service is deployed through IEF.
Importing a model from OBS	OBS	obs:object:DeleteObject obs:object:GetObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	(Mandatory) If a parallel file system is used, you need to configure obs:bucket:HeadBucket.
Importing a model from the container image	SWR	SWR Admin	(Mandatory) SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.

Application Scenario	Dependent Service	Agency Authorization	Description
Using ModelArts Edge	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	(Optional) This function must be enabled if ModelArts Edge is used.
AOM metric alarm events	AOM	aom:log:get aom:alarm:get aom:metric:put aom:alarm:put aom:event:put aom:event:list aom:event:get	Enable this function to view alarms and events on AOM.
Reporting monitoring metrics to CES	CES	ces:metricMeta:create	Enable this function to report monitoring metrics to CES.
Message subscription and push	SMN	smn:topic:list smn:topic:publish smn:application:publish	(Optional) Enable this function for message subscription and push.

Table 3-19 Basic agency authorization for managing data

Application Scenario	Dependent Service	Agency Authorization	Description
Data labeling and processing	ModelArts	modelarts:trainJob:create modelarts:trainJob:update modelarts:trainJob:delete modelarts:trainJob:get modelarts:trainJob:list modelarts:trainJob:logExport modelarts:aiAlgorithm:get modelarts:model:get modelarts:service:list modelarts:model:create modelarts:workspace:list modelarts:workspace:get modelarts:trainJobInnerModel:list	(Mandatory) Create and query training jobs, as well as querying algorithms.
Accessing OBS data	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUpload-Parts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	(Mandatory) Store, query, and delete data in OBS.

Application Scenario	Dependent Service	Agency Authorization	Description
Accessing DLI data	DLI	dli:queue:createQueue dli:queue:dropQueue dli:queue:scaleQueue dli:queue:submitJob dli:database:displayDatabase dli:database:displayAllTables dli:table:describeTable dli:table:showPrivileges dli:table:dropTable	(Optional) Enable this function if you need to view the DLI data.
Accessing MRS data	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:file:list	(Optional) Enable this function if you need to view the MRS data.
Accessing GaussDB(DWS) data	GaussDB(DWS)	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	(Optional) Enable this function if you need to view the GaussDB(DWS) data.

Table 3-20 Basic agency authorization for managing dedicated resource pools

Application Scenario	Dependent Service	Agency Authorization	Description
Interconnecting a dedicated resource pool with SFS Turbo resources	SFS Turbo	sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic	Enable this function as needed.

Application Scenario	Dependent Service	Agency Authorization	Description
Interconnecting ModelArts network with VPC and adding related routes	VPC	vpc:vpcs:get vpc:subnets:get vpc:peerings:accept vpc:routes:create vpc:routes:delete vpc:routes:get vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:routes:list	Enable this function as needed.
Using ModelArts Lite Cluster resource pools	CCE APM	cce:cluster:get cce:node:get cce:node:list cce:job:get cce:node:create cce:node:delete cce:node:remove cce:addonInstance:get cce:addonInstance:list cce:addonInstance:create cce:addonInstance:update cce:addonInstance:delete apm:icmgr:create	This function must be enabled if ModelArts Lite Cluster resource pools are used. ModelArts uses an agency to manage CCE clusters, synchronize cluster information, and manage nodes.

Application Scenario	Dependent Service	Agency Authorization	Description
	ECS BMS EVS DEW	ecs:cloudServers:create ecs:cloudServers:delete ecs:cloudServers:get ecs:cloudServers:start ecs:cloudServers:stop ecs:cloudServers:reboot ecs:cloudServers:redeploy ecs:cloudServers:listServerInterfaces ecs:cloudServers:changeVpc ecs:cloudServerFlavors:get ecs:quotas:get ecs:cloudServers:batchSetServerTags ecs:cloudServers:list bms:servers:create bms:serverFlavors:get evs:types:get evs:volumes:list evs:quotas:get evs:volumes:get kps:domainKeypairs:get	This function must be enabled if ModelArts Lite Cluster resource pools are used. ModelArts uses an agency to manage the lifecycle of BMSs and ECSs.
	IMS	ims:images:get ims:images:share	This function must be enabled if ModelArts Lite Cluster resource pools are used. Share the node system image with your account before creating a ModelArts Lite Cluster dedicated resource pool node.

3.2.3 Workspace

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.

Workspace supports the following types of access control:

- **PUBLIC**: publicly accessible to tenants (including both tenant accounts and all their user accounts)
- **PRIVATE**: accessible only to the creator and tenant accounts
- **INTERNAL**: accessible to the creator, tenant accounts, and specified IAM user accounts. When **Authorization Type** is set to **INTERNAL**, specify one or more accessible IAM user accounts.

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 workspaces 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.

As an enterprise user, you can submit the request for enabling the workspace function to your technical support.

3.3 Configuration Practices in Typical Scenarios

3.3.1 Assigning Permissions to Individual Users for Using ModelArts

Certain ModelArts functions require access to Object Storage Service (OBS), Software Repository for Container (SWR), and Intelligent EdgeFabric (IEF). Before using ModelArts, your account must be authorized to access these services. Otherwise, these functions will be unavailable.

Constraints

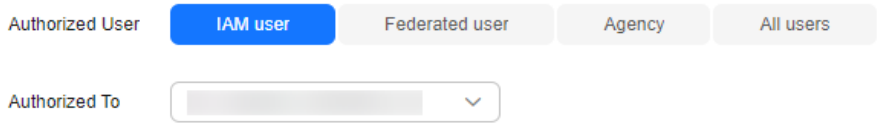
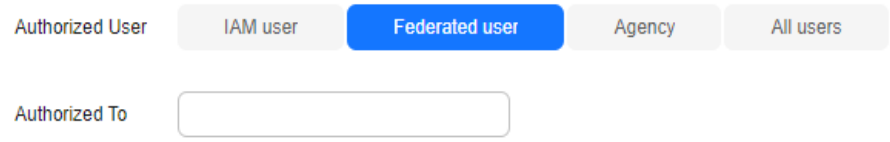
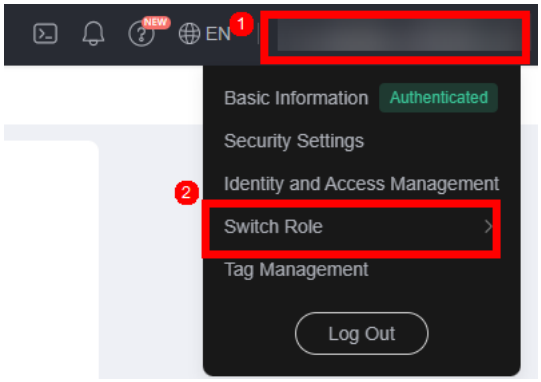
- Only a tenant account can perform agency authorization to authorize the current account or all IAM users under the current account.
- Multiple IAM users or accounts can use the same agency.
- A maximum of 50 agencies can be created under an account.
- If you use ModelArts for the first time, add an agency. Generally, common user permissions are sufficient for your requirements. You can configure permissions for refined permissions management.
- 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, configure the parameters.

Table 3-21 Parameters

Parameter	Description
Authorized User	Options: **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 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 current account, including those created in the future. For individual users, choose **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 3-5** Selecting an IAM user  • **Federated user**: Enter the username or user ID of the target federated user. **Figure 3-6** Selecting 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 3-7** Switch Role  **NOTE** ModelArts does not support the creation of agencies with identity policy permissions.
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**.

Parameter	Description
Add agency > Agency Name	The system automatically creates a changeable agency name.
Add agency > Permissions > Common User	**Common User** 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. Click **View permissions** to view common user permissions.
Add agency > Permissions > Custom	If you need refined permissions management, select **Custom** to flexibly assign permissions to the created agency. You can select permissions from the permission list as required.

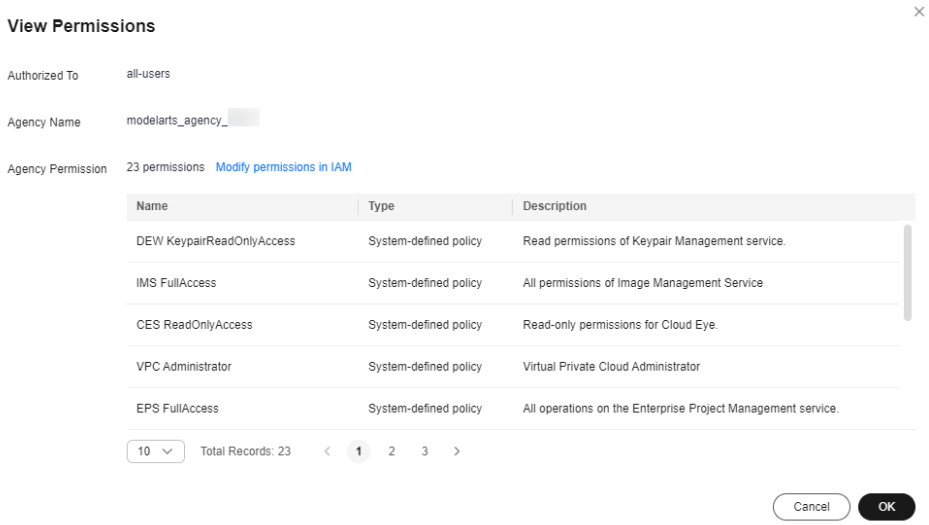
3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

Viewing Authorized Permissions

You can view the configured authorizations on the **Permission Management** page. Click **View Permissions** in the **Authorization Content** column to view the permission details.

Figure 3-8 View Permissions

Figure 3-9 Common user permissions



3.3.2 Assigning Basic Permissions for Using ModelArts

3.3.2.1 Scenario

Certain ModelArts functions require the permission to access other services. This section describes how to assign specific permissions to IAM users when they use ModelArts.

Permissions

The permissions of IAM users are controlled by their tenant user. Logging in as a tenant user, you can assign permissions to the target user group through IAM. Then, the permissions are assigned to all members in the user group. The following authorization list uses the system-defined policies of ModelArts and other services as an example.

Table 3-22 Service authorization

Target Service	Description	IAM Permission	Mandatory
ModelArts	Assign permissions to IAM users for using ModelArts. The 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 IAM users.	ModelArts CommonOperations	Yes
	The users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when selecting this option.	ModelArts FullAccess	No Select either **ModelArts FullAccess** or **ModelArts CommonOperations**.
Object Storage Service (OBS)	Assign permissions to IAM users for using OBS. ModelArts data management, development environments, training jobs, and model deployment require OBS for forwarding data.	OBS OperateAccess	Yes

Target Service	Description	IAM Permission	Mandatory
Software Repository for Container (SWR)	Assign permissions to IAM users for using SWR. ModelArts custom images require the SWR FullAccess permission.	SWR OperateAccess	Yes
Key Management Service (KMS)	To use remote SSH of ModelArts notebook, IAM users require KMS authorization.	KMS CMKFullAccess	No
Intelligent EdgeFabric (IEF)	Assign permissions to IAM users for using IEF. Tenant administrator permissions are required so that ModelArts edge services depending on IEF can be used.	Tenant Administrator	No
Cloud Eye	Assign permissions to IAM users for using Cloud Eye. Using Cloud Eye, you can view the running statuses of ModelArts real-time services and AI application loads, and set monitoring alarms.	CES FullAccess	No
Simple Message Notification (SMN)	Assign permissions to IAM users for using SMN. SMN is used with Cloud Eye.	SMN FullAccess	No
Virtual Private Cloud (VPC)	During the creation of a dedicated resource pool for ModelArts, IAM users require VPC permissions so that they can customize networks.	VPC FullAccess	No

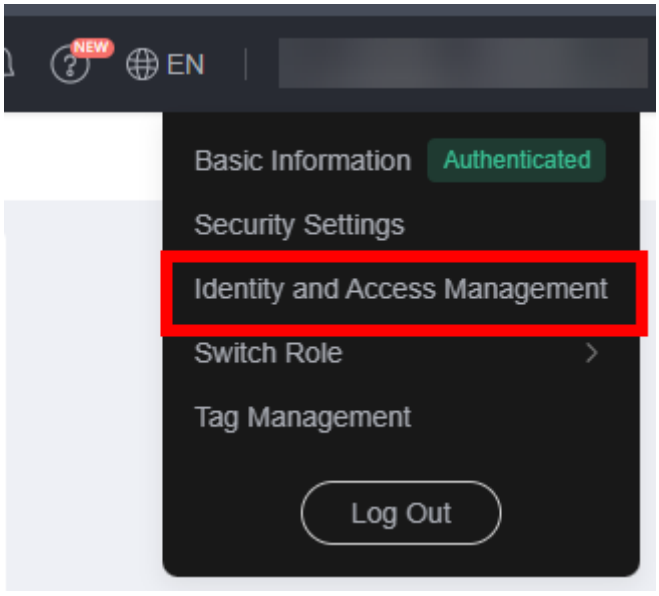
Target Service	Description	IAM Permission	Mandatory
SFS	Assign permissions to IAM users for using SFS. SFS file systems can be mounted to ModelArts dedicated resource pools to serve as storage for development environments or training.	SFS Turbo FullAccess SFS FullAccess	No

3.3.2.2 Step 1 Creating a User Group and Adding Users to the User Group

Multiple IAM users can be created under a tenant user, and the permissions of the IAM users are managed by group. This section describes how to create a user group and IAM users and add the IAM users to the user group.

1. Log in to the management console as a tenant user, hover 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.

   Figure 3-10 Identity and Access Management

2. Create a user group. In the navigation pane on the left, choose **User Groups**. Click **Create User Group** in the upper right corner. Then, set **Name** to **UserGroup-2** and click **OK**.

   After the user group is created, the system automatically switches to the user group list. Then, you can add existing IAM users to the user group through user group management. If there is no existing IAM user, create users and add them to the user group.

3. Create IAM users and add them to the user group. In the navigation pane on the left, choose **Users**. On the displayed page, click **Create User** in the upper right corner. On the **Create User** page, add multiple users.

   Set parameters as prompted and click **Next**.

4. On the **Add User to Group** page, select **UserGroup-2** and click **Create**.

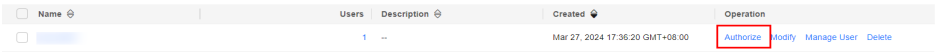
   The system will automatically add the two users to the target group one by one.

3.3.2.3 Step 2 Assigning Permissions for Using Cloud Services

An IAM user can use cloud services such as ModelArts and OBS only after they are assigned with permissions from the tenant user. This section describes how to assign the permissions to use cloud services to all IAM users in a user group.

1. On the user group list page of IAM, click **Authorize** of the target user group. The **Authorize User Group** page is displayed.

   Figure 3-11 Authorize

2. Before assigning permissions, learn about minimum permissions requirements of each ModelArts module, as shown in **Table 3-22**.
3. 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 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 IAM users.
   - The users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when selecting this option.

4. Assign permissions for using OBS. Search for **OBS** and select **OBS Administrator**. ModelArts training jobs use OBS to forward data. Therefore, the permissions for using OBS are necessary.
5. Assign permissions for using SWR. Search for **SWR** and select **SWR FullAccess**. ModelArts custom images require the SWR FullAccess permission.
6. (Optional) Assign the key management permission. Remote SSH of ModelArts notebook requires the key management permission. Search for **DEW** and select **DEW KeypairFullAccess**.

   DEW key management permission is configured in the following regions: **CN North-Beijing1**, **CN North-Beijing4**, **CN East-Shanghai1**, **CN East-Shanghai2**, **CN South-Guangzhou**, **CN Southwest-Guiyang1**, **CN-Hong Kong**, and **AP-Singapore**. In other regions, the KMS key management permission is configured. In this example, the **CN-Hong Kong** region is used. Therefore, the DEW key management permission is to be configured.

7. (Optional) Assign permissions for using IEF. ModelArts requires the Tenant Administrator permission so that edge services depending on IEF can be used.

Tenant Administrator has the permission to manage all cloud services, not only ModelArts. Exercise caution when assigning the Tenant Administrator permission.

8. (Optional) Assign permissions for using Cloud Eye and SMN. On the details page of a ModelArts real-time service deployed for inference, the number of calls is available. Click **View Details** to obtain more information. If you want to view the overall running status of ModelArts real-time services and AI application loads on Cloud Eye, assign Cloud Eye permissions to IAM users.

   To view monitoring data only, select **CES ReadOnlyAccess**.

   To set alarm monitoring on Cloud Eye, you also need to add **CES FullAccess** and SMN permissions.

9. (Optional) Assign permissions for using VPC. To enable custom network configuration when creating a dedicated resource pool, assign permissions for using VPC.
10. (Optional) Assign permissions for using SFS and SFS Turbo. To mount an SFS system to a dedicated resource pool as the storage for the development environment or training, assign the permission to use the SFS system.
11. Click **View Selected** in the upper left corner and confirm the selected permissions.
12. Click **Next** and set the minimum authorization scope. Select **Region-specific projects**, select the region to be authorized, and click **OK**.
13. 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.

3.3.2.4 Step 3 Configuring Agent-based ModelArts Access Authorization

After assigning IAM permissions, configure ModelArts access authorization for IAM users on the ModelArts page so that ModelArts can access dependent services such as OBS, SWR, and IEF.

In agent-based ModelArts access authorization, only tenant users are allowed to configure for their IAM users. In this example, use the administrator account to configure access authorization for all the users.

1. Use the tenant account to log in to the ModelArts management console. Select your region in the upper left corner.
2. In the navigation pane on the left, click **Permissions** to go to the permission management page.
3. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **All users** and click **Add agency** to configure the agency-based authorization for all IAM users under the account.
   - **Common User**: You can use basic ModelArts functions, for example, accessing data and creating and managing training jobs, but not to manage resources. Select this option generally.
   - **Custom**: You can flexibly assign permissions to the created agency. Select this option for refined permissions management. You can select permissions from the permission list as required.
4. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

3.3.2.5 Step 4 Verifying User Permissions

It takes 15 to 30 minutes for the permissions configured in **4** to take effect. Therefore, wait for 30 minutes after the configuration and then verify the configuration.

1. Log in to the ModelArts management console as an IAM in **UserGroup-2**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. Check ModelArts permissions.
   a. Select the target region in the upper left corner, which must be the same as that in the authorization configuration.
   b. In the navigation pane on the left of the ModelArts management console, choose **DevEnviron** > **Notebook**. The ModelArts permissions and agency authorization are configured correctly if no message shows insufficient permissions.

      If a message is displayed, indicating that you need to authorize access, the ModelArts agency authorization has not been configured. In this case, follow the instructions in **Step 3 Configuring Agent-based ModelArts Access Authorization** to configure the authorization.
   c. In the navigation pane on the left of the ModelArts management console, choose **DevEnviron** > **Notebook** and click **Create**. If this operation is successful, you have obtained ModelArts operation permissions.

      Alternatively, you can try other functions, such as **Training Management** > **Training Jobs**. If the operation is successful, you can use ModelArts properly.
3. Verify OBS permissions.
   a. In the service list in the upper left corner, select OBS. The OBS management console is displayed.
   b. Click **Create Bucket** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
4. Verify SWR permissions.
   a. In the service list in the upper left corner, select SWR. The SWR management console is displayed.
   b. If an SWR page can be properly displayed, you have obtained SWR operation permissions.
5. Verify other optional permissions.
6. Experience ModelArts.

3.3.3 Separately Assigning Permissions to Administrators and Developers

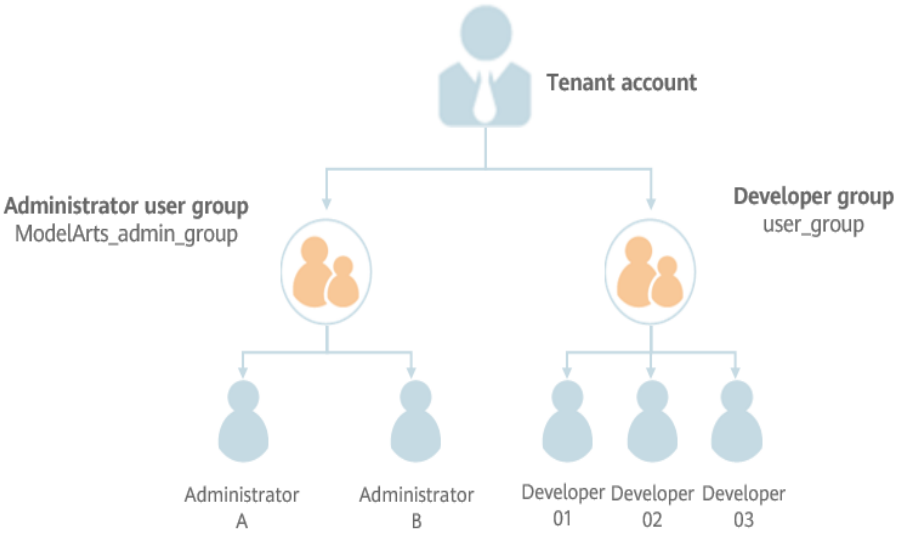
In small- and medium-sized teams, administrators need to globally control ModelArts resources, and developers only need to focus on their own instances. Generally, the **te_admin** permission of a developer account must be configured by the tenant account. This section uses notebook as an example to describe how to assign different permissions to administrators and developers through custom policies.

Scenarios

To develop a project using notebook, administrators need full control permissions for using ModelArts dedicated resource pools, and access and operation permissions on all notebook instances.

To use development environments, developers only need operation permissions for using their own instances and dependent services. They do not need to perform operations on ModelArts dedicated resource pools or view notebook instances of other users.

Figure 3-12 Account relationships



Configuring Permissions for an Administrator

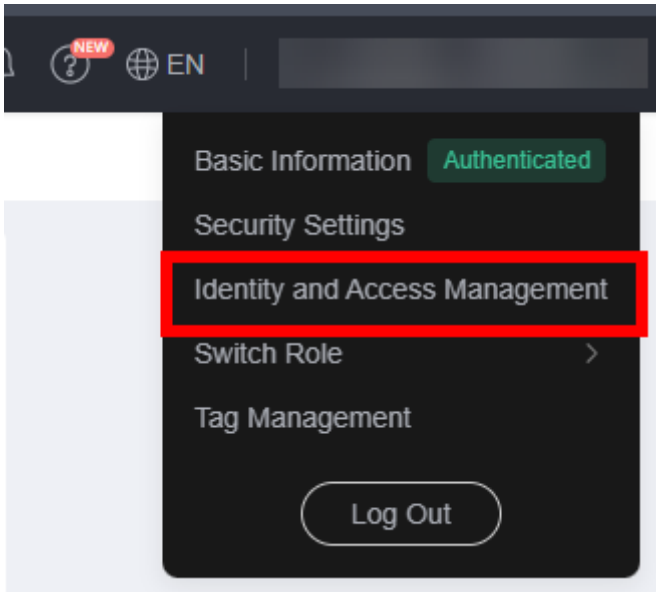
Assign full control permissions to administrators for using ModelArts dedicated resource pools and all notebook instances. The procedure is as follows:

Step 1 Use a tenant account to create an administrator user group **ModelArts_admin_group** and add administrator accounts to **ModelArts_admin_group**. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 2 Create a custom policy.

1. Log in to the management console using an administrator account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.

Figure 3-13 Identity and Access Management



2. Create custom policy 1 and assign IAM and OBS permissions to the user. In the navigation pane of the IAM console, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy1_IAM_OBS** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

   The custom policy **Policy1_IAM_OBS** is as follows, which grants IAM and OBS operation permissions to the user. You can directly copy and paste the content.

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "iam:users:listUsers",
            "iam:projects:listProjects",
            "obs:object:PutObject",
            "obs:object:GetObject",
            "obs:object:GetObjectVersion",
            "obs:bucket:HeadBucket",
            "obs:object:DeleteObject",
            "obs:bucket:CreateBucket",
            "obs:bucket:ListBucket"
            ]
      }
   ]
}
```

3. Repeat **Step 2.2** to create custom policy 2 and grant the user the permissions to perform operations on dependent services ECS, SWR, MRS, and SMN as well as ModelArts. Set **Policy Name** to **Policy2_AllowOperation** and **Policy View** to **JSON**, configure the policy content, and click **OK**.

   The custom policy **Policy2_AllowOperation** is as follows, which grants the user the permissions to perform operations on dependent services ECS, SWR, MRS, and SMN as well as ModelArts. You can directly copy and paste the content.

```
{
   "Version": "1.1",
   "Statement": [
```



```
    {
        "Effect": "Allow",
        "Action": [
            "ecs:serverKeypairs:list",
            "ecs:serverKeypairs:get",
            "ecs:serverKeypairs:delete",
            "ecs:serverKeypairs:create",
            "swr:repository:getNamespace",
            "swr:repository:listNamespaces",
            "swr:repository:deleteTag",
            "swr:repository:getRepository",
            "swr:repository:listTags",
            "swr:instance:createTempCredential",
            "mrs:cluster:get",
            "modelarts:*:*"
        ]
    }
  ]
}
```

Step 3 Grant the policy created in **Step 2** to the administrator group **ModelArts_admin_group**.

1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **ModelArts_admin_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS** and **Policy2_AllowOperation**. Click **Next**.
2. Specify the scope as **All resources** and click **OK**.

Step 4 Configure agent-based ModelArts access authorization for an administrator to allow ModelArts to access dependent services such as OBS.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select an administrator account for **Authorized To**, select **Add agency**, and select **Common User** for **Permissions**. Permissions control is not required for administrators, so use default setting **Common User**.
3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

Step 5 Test administrator permissions.

1. Log in to the ModelArts management console as the administrator. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. In the navigation pane of the ModelArts management console, choose **Dedicated Resource Pools** and click **Create**. If the console does not display a message indicating insufficient permissions, the permissions have been assigned to the administrator.

----End

Configuring Permissions for a Developer

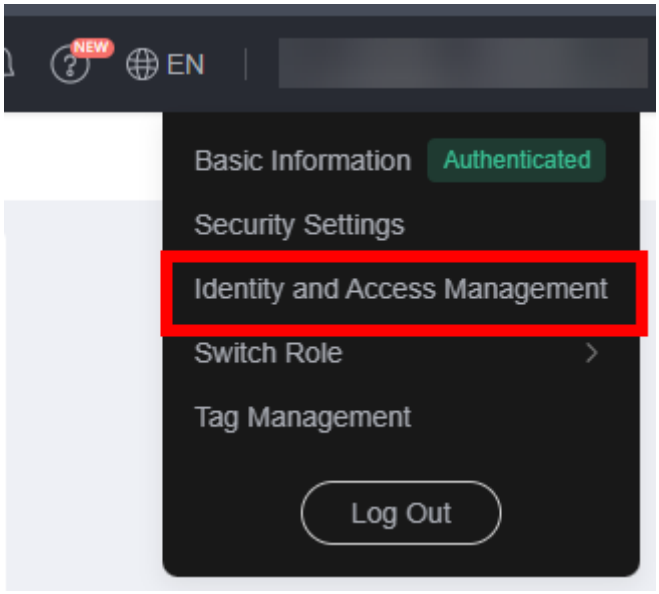
Use IAM for fine-grained control of developer permissions. The procedure is as follows:

Step 1 Use a tenant account to create a developer user group **user_group** and add developer accounts to **user_group**. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 2 Create a custom policy.

1. Log in to the management console using a tenant account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.

   Figure 3-14 Identity and Access Management

2. Create custom policy 3 to prevent users from performing operations on ModelArts dedicated resource pools and viewing notebook instances of other users.

   In the navigation pane of the IAM console, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy3_DenyOperation** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

   The custom policy **Policy3_DenyOperation** is as follows. You can copy and paste the content.

```
{
   "Version": "1.1",
   "Statement": [
         {
          "Effect": "deny",
          "Action": [
              "modelarts:pool:create",
              "modelarts:pool:update",
              "modelarts:pool:delete",
      "modelarts:notebook:listAllNotebooks"
            ]

         }
   ]
}
```

Step 3 Grant the custom policy to the developer user group **user_group**.

1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **user_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS**, **Policy2_AllowOperation**, and **Policy3_DenyOperation**. Click **Next**.
2. Specify the scope as **All resources** and click **OK**.

Step 4 Configure agent-based ModelArts access authorization for a developer to allow ModelArts to access dependent services such as OBS.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select a developer account for **Authorized To**, add an agency **ma_agency_develop_user**, set **Permissions** to **Custom**, and select **OBS Administrator**. Developers only need OBS authorization to allow developers to access OBS when using notebook.
3. Click **Create**.
4. On the **Permission Management** page, click **Add Authorization** again. On the **Add Authorization** page that is displayed, configure an agency for other developer users.

   On the **Add Authorization** page, set **Authorized User** to **IAM user**, select a developer account for **Authorized To**, and select the existing agency **ma_agency_develop_user** created before.

Step 5 Test developer permissions.

1. Log in to the ModelArts management console as an IAM user in **user_group**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. In the navigation pane of the ModelArts management console, choose **Dedicated Resource Pools** and click **Create**. If the console does not display a message indicating insufficient permissions, the permissions have been assigned to the developer.

----End

3.3.4 Assigning the Required Permissions

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

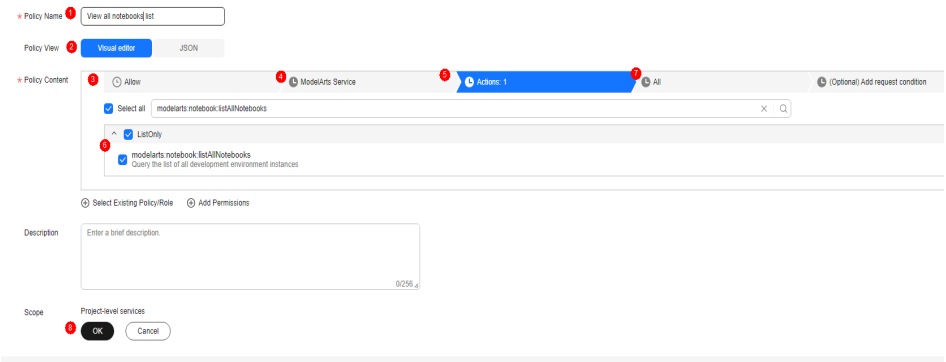
current IAM project. After the permission is granted, you can access OBS and SWR of IAM users in a notebook instance.

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 3-15**.

   - **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 3-15 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 Configuration for a Notebook Instance**. ModelArts.6789: Failed to find SSH key pair KeyPair-xxx on the ECS key pair page. Update the key pair and try again later.

3.3.5 Logging In to a Training Container Using Cloud Shell

Application Scenario

You can use Cloud Shell provided by the ModelArts console to log in to a running training container.

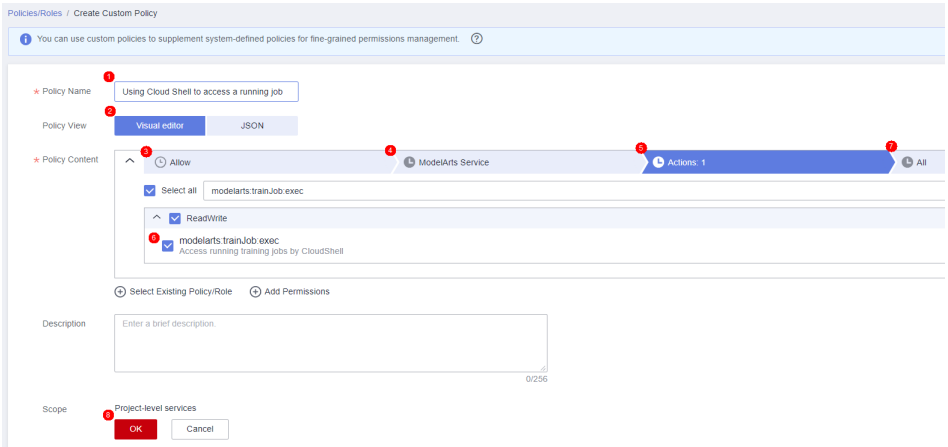
Constraints

Only dedicated resource pools support Cloud Shell. The training job must be in the **Running** state.

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.

Figure 3-16 Creating a custom policy



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.

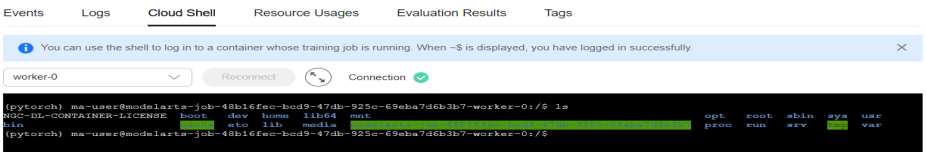
   If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

Using Cloud Shell

1. Configure parameters based on **Preparation: Assigning the Cloud Shell Permission to an IAM User**.
2. On the ModelArts console, choose **Model Training** > **Training Jobs** from the navigation pane.
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.

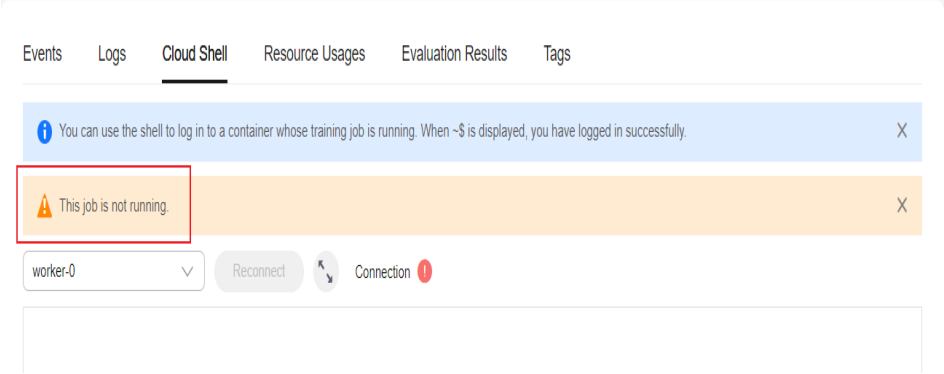
   Verify that the login is successful, as shown in the following figure.

Figure 3-17 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.

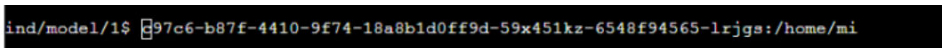
Figure 3-18 Error message



NOTE

A path display exception may occur when you log in to the Cloud Shell page. In this case, press **Enter** to rectify the fault.

Figure 3-19 Abnormal path



3.3.6 Prohibiting a User from Using a Public Resource Pool

This section describes how to control the ModelArts permissions of a user so that the user is not allowed to use a public resource pool to create training jobs, create notebook instances, or deploy inference services.

Context

Through permission control, ModelArts dedicated resource pool users can be prohibited from using a public resource pool to create training jobs, create notebook instances, or deploy inference services.

To control the permissions, configure the following permission policy items:

- **modelarts:notebook:create**: allows you to create a notebook instance.
- **modelarts:trainJob:create**: allows you to create a training job.
- **modelarts:service:create**: allows you to create an inference service.

Procedure

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. In the navigation pane, choose **Permissions** > **Policies/Roles**. On the **Policies/Roles** page, click **Create Custom Policy** in the upper right corner, configure parameters, and click **OK**.
   - **Policy Name**: Configure the policy name.
   - **Policy View**: Select **Visual editor** or **JSON**.
   - **Policy Content**: Select **Deny**. In **Select service**, search for **ModelArts** and select it. In **ReadWrite** under **Actions**, search for **modelarts:trainJob:create**, **modelarts:notebook:create**, and **modelarts:service:create** and select them. **All**: Retain the default setting. In **Add request condition**, click **Add Request Condition**. In the displayed dialog box, set **Condition Key** to **modelarts:poolType**, **Operator** to **StringEquals**, and **Value** to **public**.

     The policy content in JSON view is as follows:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "modelarts:trainJob:create",
                "modelarts:notebook:create",
```



```
            "modelarts:service:create"
        ],
        "Condition": {
            "StringEquals": {
                "modelarts:poolType": [
                    "public"
                ]
            }
        }
    }
  ]
}
```

3. In the navigation pane, choose **User Groups**. On the **User Groups** page, locate the row containing the target user group and click **Authorize** in the **Operation** column. On the **Authorize User Group** page, select the custom policy 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 one, add users to it through user group management, and configure authorization for the user group. If the target user is not in a user group, add the user to a user group through user group management.

4. Add the policy to the user's agency authorization. This prevents the user from breaking the permission scope through a token on the tenant plane.

   In the navigation pane, choose **Agencies**. Locate the agency used by the user group on ModelArts and click **Modify** in the **Operation** column. In the **Permissions** tab, click **Authorize**, select the created custom policy, and click **Next**. Select the scope for authorization and click **OK**.

Verification

Log in to the ModelArts console as an IAM user, choose **Model Training** > **Training Jobs**, and click **Create Training Job**. On the page for creating a training job, only a dedicated resource pool can be selected for **Resource Pool**.

Log in to the ModelArts console as an IAM user, choose **Development Workspace** > **Notebook**, and click **Create**. On the page for creating a notebook instance, only a dedicated resource pool can be selected for **Resource Pool**.

Log in to the ModelArts console as an IAM user, choose **Model Deployment** > **Real-Time Services**, and click **Deploy**. On the page for service deployment, only a dedicated resource pool can be selected for **Resource Pool**.

3.3.7 Authorizing ModelArts to Use SFS Turbo

This section describes how to configure ModelArts agency permissions to enable you to associate and disassociate your dedicated resource pool network with an SFS Turbo file system.

- To add an agency and authorize it to perform operations on SFS Turbo, see **Adding an Agency and Granting Permissions for SFS Turbo**.
- To add permissions to an existing agency for SFS Turbo, see **Adding Permissions to an Existing Agency for SFS Turbo**.

Context

After creating a network on the console, you can click **More** in the **Operation** column to access the options of **Associate SFS Turbo** and **Disassociate SFS Turbo** on the **Network** page. When you associate a network with an SFS Turbo file system, the file system joins the network and becomes available for training and development purposes.

Before you can associate or disassociate an SFS Turbo file system, you must grant ModelArts the permissions to perform operations on SFS Turbo.

To control the permissions, configure the following permission policy items:

- sfsturbo:shares:addShareNic: allows you to create NICs.
- sfsturbo:shares:deleteShareNic: allows you to delete NICs.
- sfsturbo:shares:showShareNic: allows you to obtain NIC details.
- sfsturbo:shares:listShareNics: allows you to obtain NICs.

Constraints

This function is available in certain regions.

Adding an Agency and Granting Permissions for SFS Turbo

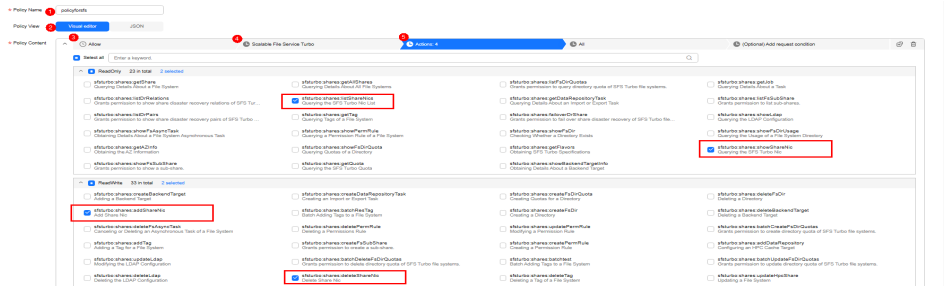
1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**. The **Permission Management** page is displayed.
2. Click **Add Authorization**. On the displayed page, configure parameters.
   - **Authorized User**: Select **IAM user**, **Federated user**, **Agency**, or **All users** as required.
   - **Authorized To**: Choose an authorized object.
   - **Agency**: Select **Add agency**.
   - **Permissions**: In common mode, select **sfsturbo:shares:addShareNic**, **sfsturbo:shares:deleteShareNic**, **sfsturbo:shares:showShareNic**, and **sfsturbo:shares:listShareNics** under SFS Turbo.
3. Click **Create**. After you authorize the agency, any user with the permission can associate or disassociate SFS Turbo.

Adding Permissions to an Existing Agency for SFS Turbo

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. In the navigation pane, choose **Permissions** > **Policies/Roles**. On the **Policies/Roles** page, click **Create Custom Policy** in the upper right corner, configure parameters, and click **OK**.
   - Policy Name: Enter a custom policy name, for example, **Authorizing ModelArts to use SFS Turbo**.
   - **Policy View**: Select **Visual editor** or **JSON**.
   - **Policy Content**: Select **Allow**. In **Select service**, search for **SFSTurbo** and select it. In **ReadOnly** under **Actions**, search for

sfsturbo:shares:showShareNic and **sfsturbo:shares:listShareNics** and select them. In **ReadWrite** under **Actions**, search for **sfsturbo:shares:addShareNic** and **sfsturbo:shares:deleteShareNic** and select them. Select **All** for **All**.

Figure 3-20 Creating a custom policy (visual editor)



The policy content in JSON view is as follows:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sfsturbo:shares:addShareNic",
                "sfsturbo:shares:listShareNics",
                "sfsturbo:shares:deleteShareNic",
                "sfsturbo:shares:showShareNic"
            ]
        }
    ]
}
```

3. The **modelarts_agency** agency is used as an example. On the IAM console, click **Agencies**. Locate **modelarts_agency** and click **Authorize** in the **Operation** column. Choose the custom policy created in **2** and click **Next**. Then, specify the authorization scope and click **OK**.

   After you authorize the agency, any user with the permission can associate or disassociate SFS Turbo.

Verification

Log in to the ModelArts management console, choose **Dedicated Resource Pools** > **Network**. Then, click **More** in the **Operation** column of the target network, and choose **Associate SFS Turbo**. The operation is successful.

Log in to the ModelArts management console, choose **Dedicated Resource Pools** > **Network**. Then, click **More** in the **Operation** column of the target network, and choose **Disassociate SFS Turbo**. The operation is successful.

3.3.8 Assigning SFS Turbo Folder-Level Access Permissions to an IAM User

Scenario

Grant access permission of specific SFS Turbo folders to IAM users.

> **NOTE**
>
> Granting the IAM user the SFS Turbo folder-level access permission is a whitelist function. Submit a service ticket to apply for the permission as needed.

Constraints

- Ensure that you have enabled strict authorization. Log in to the ModelArts console. In the navigation pane on the left, choose **Settings**. On the **Permission Management** page, click **Enable strict authorization**.
- If the ModelArts permission for the IAM user were not configured prior to enabling strict authorization mode, the IAM user might lose access to ModelArts once that mode is activated. Configure ModelArts permissions based on the service requirements. For details, see **Dependencies and Agencies**.

Procedure

Step 1 Log in to the management console using the main account, 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.

Step 2 On the IAM console, choose **Permissions** > **Policies/Roles** from the navigation pane on the left, click **Create Custom Policy** in the upper right corner, and configure the policy as follows:

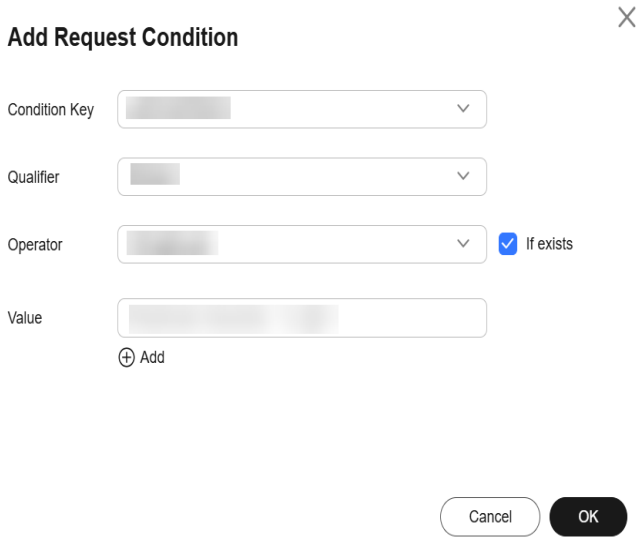
- **Policy Name**: Enter a policy name, for example, **ma_sfs_turbo**.
- **Policy View**: Select **JSON**.
- **Policy Content**: Enter the following information:

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "<modelarts_action>"
         ],
         "Condition": {
            "StringEqualsIfExists": {
               "modelarts:sfsId": [
                  "<your_ssf_id>"
               ],
               "modelarts:sfsPath": [
                  "<sfs_path>"
               ],
               "modelarts:sfsOption": [
                  "<sfs_option>"
               ]
            }
         }
      }
   ]
}
```


NOTE

- Before the preceding policies are created, all IAM users can mount to SFS Turbo by default. After you create the preceding SFS permission control policies, the IAM users who are not granted with the permission cannot mount to SFS Turbo when creating a training job on the ModelArts console, except the IAM users with the **Tenant Administrator** permission.
- Currently, you can configure only permit policies, that is, the **Effect** in the policy can only be set to **Allow**. Do not configure deny policies.
- The **Condition** parameter must use the **StringEqualsIfExists** field, and **If exists** must be enabled for the corresponding visualized view.

Figure 3-21 Enabling **If exists**



Replace *<modelarts_action>*, *<your_ssf_id>*, *<sfs_path>*, and *<sfs_option>* with actual parameters as you need. The following table describes the parameters.

Table 3-23 Parameter description

Parameter	Description

Action	Scenario in which the SFS Turbo folder access permission is granted.
	• **modelarts:notebook:create**: The permission is granted during development environment instance creation.
	• **modelarts:trainJob:create**: The permission is granted during training job creation.
	Multiple actions are supported, the following shows an example:
	`"Action": [`
	`  "modelarts:trainJob:create",`
	`  "modelarts:notebook:create"`
	`],`

Parameter	Description
modelarts:sfsId	SFS Turbo ID, which can be obtained on the SFS Turbo details page. You can enter multiple IDs, the following shows an example: "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"],
modelarts:sfsPath	Path of the SFS Turbo folder whose permissions need to be configured. You can enter multiple paths, the following shows an example: "modelarts:sfsPath": ["/path1", "/path2/path2-1"], If there are multiple SFS IDs, the SFS paths will apply to all SFS IDs. As shown in the following example, permission to access **/path1** and **/path2/path2-1** of both **0e51c7d5-d90e-475a-b5d0-ecf896da3b0d** and **2a70da1e-ea87-4ee4-ae1e-55df846e7f41** are configured. "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"], "modelarts:sfsPath": ["/path1", "/path2/path2-1"],

Parameter	Description

modelarts:sfs Option	Type of the access permission. The following parameters are supported:
	• **readonly**: Read-only permission
	• **readwrite**: Read and write permission. When you create a development environment instance, **modelarts:notebook:create** must be set to **readwrite**.
	To add multiple SFS options to a custom policy, add a JSON structure to Statement, the following shows an example:
	{
	"Version": "1.1",
	"Statement": [
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:trainJob:create"
	],
	"Condition": {
	"StringEqualsIfExists": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path1"
	],
	"modelarts:sfsOption": [
	"readonly"
	]
	}
	}
	},
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:trainJob:create"
	],
	"Condition": {
	"StringEqualsIfExists": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path2"
	],
	"modelarts:sfsOption": [
	"readwrite"
	]
	}
	}
	}
	]
	}

Step 3 Create a user group and add the user to the user group. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 4 Grant a policy to the user group. On the user group list page of IAM, click **Authorize** of the target user group. The **Authorize User Group** page is displayed. Select the **ma_sfs_turbo** policy created in **Step 2**. Click **Next** and then **OK**.

Step 5 Add the **IAM ReadOnlyAccess** permission to an existing ModelArts agency.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Permission Management**. On the displayed page, locate the target agency, choose **View Permissions** in the **Operation** column, and click **Modify permission in IAM**.
2. On the IAM console, choose **Agencies** from the navigation pane on the left, and choose **Permissions** > **Authorize**. Search for **IAM ReadOnlyAccess**, enable it, and click **Next** and **OK**.

Step 6 Verify that the permission is granted.

Log in to ModelArts as the IAM user, only the configured SFS Turbo folders are displayed during training job creation and notebook creation.

----End

4 Notebook

4.1 Migrating the Conda Environment on a Notebook Instance to an SFS Disk

This section describes how to migrate the Conda environment on a notebook instance to an SFS disk. In this way, the Conda environment will not be lost after the notebook instance is restarted.

The procedure is as follows:

1. **Creating a Virtual Environment and Saving It to the SFS Directory**
2. **Cloning the Existing Virtual Environments to the SFS Disk**
3. **Restarting the Image to Activate the Virtual Environment in the SFS Disk**
4. **Saving and Sharing the Virtual Environment**

Prerequisites

You have created a notebook instance by setting **Resource Type** to **Dedicated resource pool** and **Storage** to **SFS** and opened the terminal.

Creating a Virtual Environment and Saving It to the SFS Directory

Create a conda virtual environment.

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

View the existing conda virtual environments. The name of the newly created virtual environment may be empty in the output.

```
# shell
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
                      /home/ma-user/work/envs/user_conda/sfs-new-env
```


Append the new virtual environment to conda envs.

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

View the existing conda virtual environments. The new virtual environment is properly displayed, and you can switch to it by name.

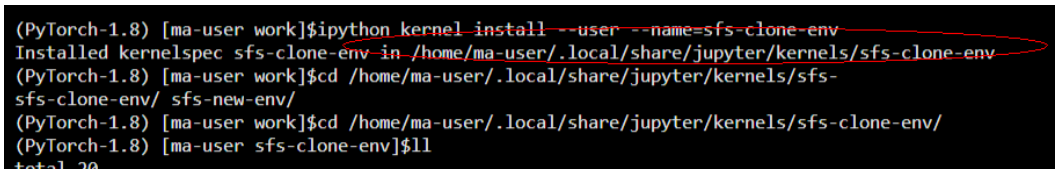
```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
```

(Optional) Register the new virtual environment with the JupyterLab kernel, so that you can directly use it in JupyterLab.

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-new-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-new-env/logo-*
```

Note: **.local/share/jupyter/kernels/sfs-new-env** is used as an example only. Replace it with the actual installation path.

Figure 4-1 Installation path output



Refresh the JupyterLab page. The new kernel is displayed.

NOTE

After the notebook instance is restarted, the kernel needs to be registered again.

Cloning the Existing Virtual Environments to the SFS Disk

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-clone-env --clone PyTorch-1.8 -
y
Source:      /home/ma-user/anaconda3/envs/PyTorch-1.8
Destination: /home/ma-user/work/envs/user_conda/sfs-clone-env
Packages: 20
Files: 39687
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/ma-user/work/envs/user_conda/sfs-clone-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```


View the cloned virtual environments. If the name of the newly created virtual environment is empty, handle the issue according to **Append the new virtual environment to conda envs**.

```
# shell
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
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env           *  /home/ma-user/work/envs/user_conda/sfs-new-env
```

(Optional) Register the new virtual environment with the JupyterLab kernel, so that you can directly use it in JupyterLab.

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-clone-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/logo-*
```

Note: **.local/share/jupyter/kernels/sfs-clone-env** is used as an example only. Replace it with the actual installation path.

Refresh the JupyterLab page. The new kernel is displayed.

Restarting the Image to Activate the Virtual Environment in the SFS Disk

Method 1: Use the complete conda env path.

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

Method 2: Append the virtual environment to conda envs and activate it using its name.

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

Method 3: Use Python or pip in the virtual environment.

```
# shell
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/pip list
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/python -V
```

Saving and Sharing the Virtual Environment

Package the virtual environment to be migrated.

```
# shell
pip install conda-pack
conda pack -n sfs-clone-env -o sfs-clone-env.tar.gz --ignore-editable-packages
Collecting packages...
Packing environment at '/home/ma-user/work/envs/user_conda/sfs-clone-env' to 'sfs-clone-env.tar.gz'
[########################################] | 100% Completed |  3min 33.9s
```

Decompress the package to the SFS directory.

```
# shell
```



```
mkdir /home/ma-user/work/envs/user_conda/sfs-tar-env
tar -zxvf sfs-clone-env.tar.gz -C /home/ma-user/work/envs/user_conda/sfs-tar-env
```

View the existing conda virtual environments.

```
# shell
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
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
sfs-tar-env              /home/ma-user/work/envs/user_conda/sfs-tar-env
test-env                 /home/ma-user/work/envs/user_conda/test-env
```


5 Model Training

5.1 Building a Handwritten Digit Recognition Model with ModelArts Standard

This section describes how to modify a local custom algorithm to train and deploy models on ModelArts.

Scenarios

This case describes how to use PyTorch 1.8 to recognize handwritten digit images. An official MNIST dataset is used in this case.

Through this case, you can learn how to train jobs, deploy an inference model, and perform prediction on ModelArts.

Process

Before performing the following operations, complete necessary operations by referring to **Preparations**.

1. **Step 1 Prepare Training Data**: Download the MNIST dataset.
2. **Step 2: Preparing Training Files and Inference Files**: Write training and inference code.
3. **Step 3: Creating an OBS Bucket and Upload Files to OBS**: Create an OBS bucket and folder, and upload the dataset, training script, inference script, and inference configuration file to OBS.
4. **Step 4 Create a Training Job**: Train a model.
5. **Step 5 Deploying the Model for Inference**: Import the trained model to ModelArts, create a model, and deploy the model as a real-time service.
6. **Step 6 Performing Prediction**: Upload a handwritten digit image and send an inference request to obtain the inference result.
7. **Step 7 Releasing Resources**: Stop the service and delete the data in OBS to stop billing.

Preparations

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

  To use ModelArts, access to Object Storage Service (OBS), SoftWare Repository for Container (SWR), and Intelligent EdgeFabric (IEF) is required. If this is the first time you use ModelArts, configure an agency to authorize access to these services.

  a. Log in to the **ModelArts console** using your Huawei Cloud account. In the navigation pane on the left, choose **Settings**. On the **Global Configuration** page, click **Add Authorization**.

  b. Configure the parameters as follows on the displayed page:

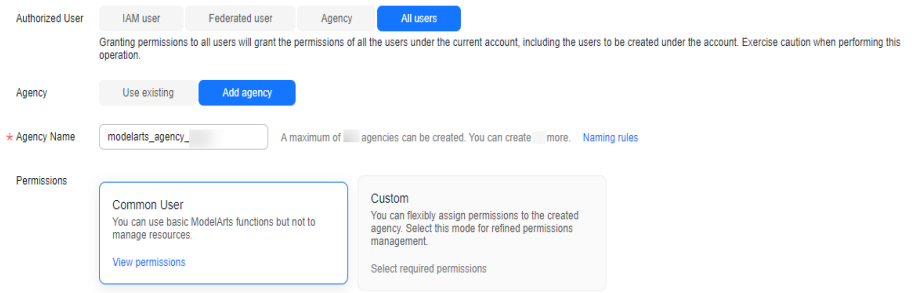
  Authorized User: **All users**.

  Agency: **Add agency**.

  Permissions: **Common User**.

  Select "I have read and agree to the ModelArts Service Statement" and click **Create**.

  Figure 5-1 Configuring an agency



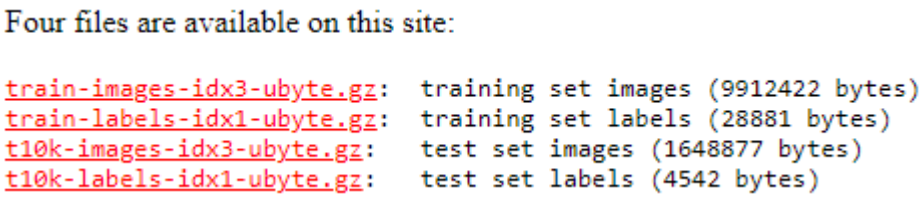
  c. After the configuration, view the agency configurations of your account on the **Global Configuration** page.

  Figure 5-2 Viewing agency configurations

Step 1 Prepare Training Data

Download the MNIST dataset from a web browser. Ensure the four files in **Figure 5-3** are all downloaded.

Figure 5-3 MNIST dataset



- **train-images-idx3-ubyte.gz**: compressed package of the training set, which contains 60,000 samples.
- **train-labels-idx1-ubyte.gz**: compressed package of the training set labels, which contains the labels of the 60,000 samples
- **t10k-images-idx3-ubyte.gz**: compressed package of the validation set, which contains 10,000 samples.
- **t10k-labels-idx1-ubyte.gz**: compressed package of the validation set labels, which contains the labels of the 10,000 samples

Step 2: Preparing Training Files and Inference Files

In this case, ModelArts provides the training script, inference script, and inference configuration file.

NOTE

When pasting code from a .py file, create a .py file. Otherwise, the error message "SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence" may be displayed.

Create the training script **train.py** on the local host. The content is as follows:

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# Define a network model.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
```



```
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


# Train the model. Set the model to the training mode, load the training data, calculate the loss function,
and perform gradient descent.
def train(args, 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.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 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()))
            if args.dry_run:
                break


# Validate the model. Set the model to the validation mode, load the validation data, and calculate the loss
function and accuracy.
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()
            pred = output.argmax(dim=1, keepdim=True)
            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)))


# The following is PyTorch MNIST.
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)


def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
```



```
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
}


def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
    """Read an SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
       Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# The above is pytorch mnist.
# --- end


# Raw MNIST dataset processing
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
```



```
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
            test.pt
    """
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # Define the preset running parameters of the training job.
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')

    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
```



```
                    help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True,
                    help='For Saving the current Model')
    args = parser.parse_args()

    use_cuda = not args.no_cuda and torch.cuda.is_available()

    torch.manual_seed(args.seed)

    # Set whether to use GPU or CPU to run the algorithm.
    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # Define the data preprocessing method.
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # Convert the raw MNIST dataset to a PyTorch MNIST dataset.
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # Create a training dataset and a validation dataset.
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)

    # Create iterators for the training dataset and the validation dataset.
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # Initialize the neural network model and copy the model to the compute device.
    model = Net().to(device)
    # Define the training optimizer and learning rate for gradient descent calculation.
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # Train the neural network and perform validation in each epoch.
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # Save the model and make it adapted to the ModelArts inference model package specifications.
    if args.save_model:

        # Create the model directory in the path specified in train_url.
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # Save the model to the model directory based on the ModelArts inference model package
specifications.
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # Copy the inference code and configuration file to the model directory.
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
```



```
    shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

if __name__ == '__main__':
    main()
```

Create the inference script **customize_service.py** on the local host. The content is as follows:

```
import os
import log
import json

import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# Define model preprocessing.
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# Model inference service
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 customized function to load the model.
        self.model = Mnist(model_path)

         # Load labels.
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # Receive the request data and convert it to the input format acceptable to the model.
    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

    # Post-process the inference result to obtain the expected output format. The result is the returned value.
    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

    # Perform forward inference on the input data to obtain the inference result.
    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

# Define a network.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


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)

    # Turn the model to inference mode.
    model.eval()

    return model
```

Infer the configuration file **config.json** on the local host. The content is as follows:

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```


Step 3: Creating an OBS Bucket and Upload Files to OBS

Upload the data, code file, inference code file, and inference configuration file obtained in the previous step to an OBS bucket. When running a training job on ModelArts, read data and code files from the OBS bucket.

1. Log in to OBS management console and create an OBS bucket and folder.

```
{OBS bucket}                    # OBS bucket name, which is customizable, for example, test-modelarts-xx
    -{OBS folder}          # OBS folder name, which is customizable, for example, pytorch
        - mnist-data       # OBS folder, which is used to store the training dataset. The folder name is customizable, for example, mnist-data.
        - mnist-code       # OBS folder, which is used to store training script train.py. The folder name is customizable, for example, mnist-code.
            - infer        # OBS folder, which is used to store inference script customize_service.py and configuration file config.json
        - mnist-output     # OBS folder, which is used to store trained models. The folder name is customizable, for example, mnist-output.
```

CAUTION

- The region where the created OBS bucket resides must be the same as that where ModelArts is used. Otherwise, the OBS bucket will be unavailable for training. For details, see **Check whether the OBS bucket and ModelArts are in the same region.**
- When creating an OBS bucket, do not set the archive storage class. Otherwise, training models will fail.

2. Upload the MNIST dataset package obtained in **Step 1 Prepare Training Data** to the **mnist-data** folder on OBS.

CAUTION

- When uploading data to OBS, do not encrypt the data. Otherwise, the training will fail.
- Files do not need to be decompressed. Directly upload compressed packages to OBS.

3. Upload the training script **train.py** to the **mnist-code** folder.
4. Upload the inference script **customize_service.py** and inference configuration file **config.json** to the **infer** folder in **mnist-code**.

Step 4 Create a Training Job

1. Log in to the ModelArts management console and select the same region as the OBS bucket.
2. In the navigation pane on the left, choose **Permission Management** and check whether access authorization has been configured for the current account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
3. In the navigation pane, choose **Model Training** > **Training Jobs**. On the **Training Jobs** page, click **Create Training Job**.

4. Set parameters.
   - **Algorithm Type**: Select **Custom algorithm**.
   - **Boot Mode**: Select **Preset image** and then select **PyTorch** and **pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64** from the drop-down lists.
   - **Code Directory**: Select the created OBS code directory, for example, **/test-modelarts-xx/pytorch/mnist-code/** (replace **test-modelarts-xx** with your OBS bucket name).
   - **Boot File**: Select the training script **train.py** uploaded to the code directory.
   - **Input**: Add one input and set its name to **data_url**. Set the data path to your OBS directory, for example, **/test-modelarts-xx/pytorch/mnist-data/** (replace **test-modelarts-xx** with your OBS bucket name).
   - **Output**: Add one output and set its name to **train_url**. Set the data path to your OBS directory, for example, **/test-modelarts-xx/pytorch/mnist-output/** (replace **test-modelarts-xx** with your OBS bucket name). Do not pre-download to a local directory.
   - **Resource Type**: Select the specifications of a single-card GPU. If there are free GPU specifications, you can select them for training.
   - Retain default settings for other parameters.

     NOTE

     The sample code runs on a single node with a single card. If you select a flavor with multiple GPUs, the training will fail.

5. Click **Submit**, confirm parameter settings for the training job, and click **Yes**.

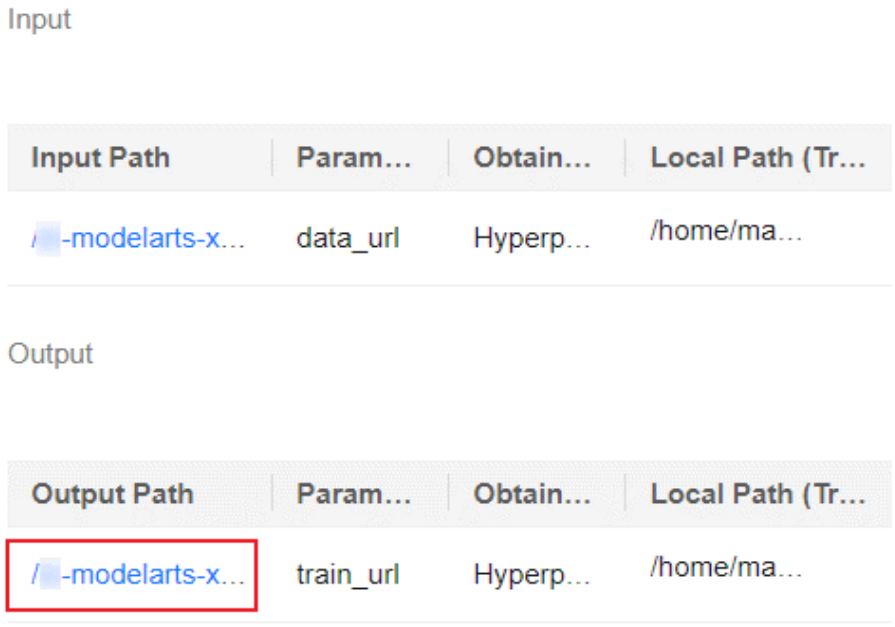
   The system automatically switches back to the **Training Jobs** page. When the training job status changes to **Completed**, the model training is completed.

   NOTE

   In this case, the training job will take about 10 minutes.

6. Click the training job name. On the job details page that is displayed, check whether there are error messages in logs. If so, the training failed. Identify the cause and locate the fault based on the logs.
7. In the lower left corner of the training details page, click the training output path to go to OBS, as shown in **Figure 5-4**. Then, check whether the **model** folder is available and whether there are any trained models in the folder. If there is no **model** folder or trained model, the training input may be incomplete. In this case, completely upload the training data and train the model again.

Figure 5-4 Output path



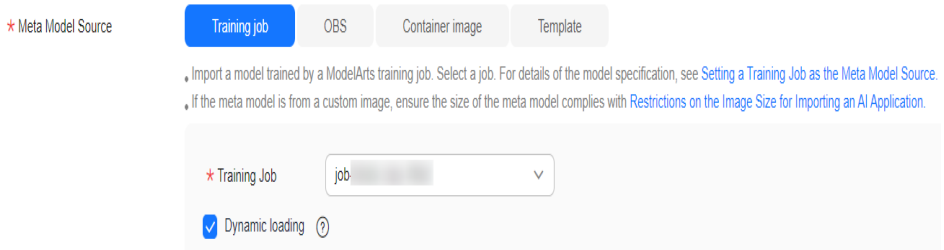
Step 5 Deploying the Model for Inference

After the model training is completed, create a model and deploy the model as a real-time service.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Model Management (AI Applications)**. In the displayed **My Model** tab, click **Create Model**.
2. On the **Create Model** page, configure the parameters and click **Create now**.

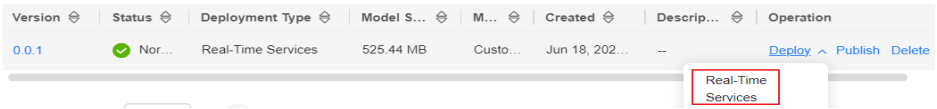
   Choose **Training Job** for **Meta Model Source**. Select the training job completed in **Step 4 Create a Training Job** from the drop-down list and enable **Dynamic loading**. The values of **AI Engine** will be automatically configured.

   Figure 5-5 Meta Model Source

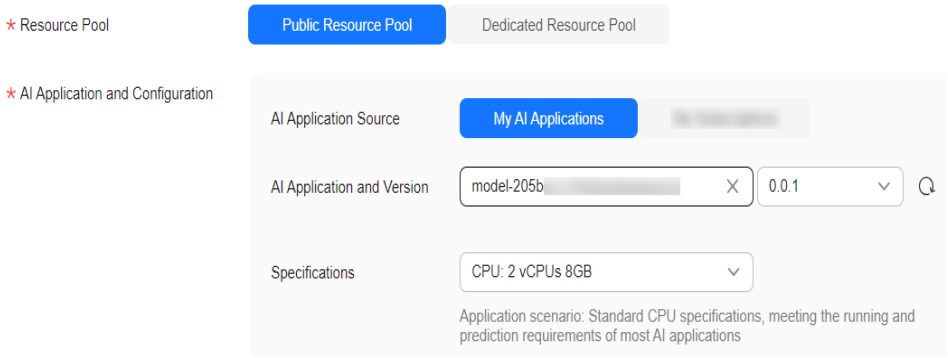
3. Wait until the model status changes to **Normal**. Then, the model is created. Locate the target model and click **Deploy** in the **Operation** column. On the displayed **Versions** page, locate the version and choose **Deploy** > **Real-Time Services** in the **Operation** column.

Figure 5-6 Deploying a real-time service



4. On the **Deploy** page, configure parameters and create a real-time service as prompted. In this example, use CPU specifications. If there are free CPU specifications, you can select them for deployment. (Each user can deploy only one real-time service for free. If you have deployed one, delete it first before deploying a new one for free.)

Figure 5-7 Deploying a model



After you submit the service deployment request, the system automatically switches to the **Real-Time Services** page. When the service status changes to **Running**, the service has been deployed.

Step 6 Performing Prediction

1. On the **Real-Time Services** page, click the name of the real-time service. The real-time service details page is displayed.
2. Click the **Prediction** tab, set **Request Type** to **multipart/form-data**, **Request Parameter** to **image**, click **Upload** to upload a sample image, and click **Predict**.

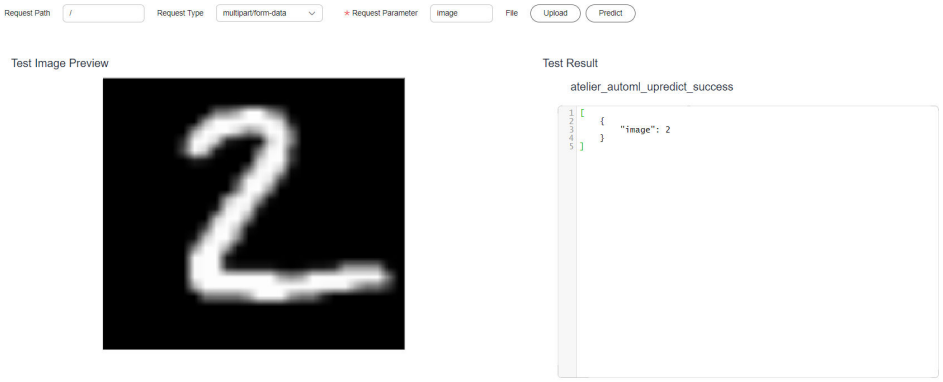
   After the prediction is complete, the prediction result is displayed in the **Test Result** pane. According to the prediction result, the digit on the image is **2**.

   NOTE

   The MNIST used in this case is a simple dataset used for demonstration, and its algorithms are also simple neural network algorithms used for teaching. The models generated using such data and algorithms are applicable only to teaching but not to complex prediction scenarios. The prediction is accurate only if the image used for prediction is similar to the image in the training dataset (white characters on black background).

Figure 5-8 Example

Figure 5-9 Prediction results



Step 7 Releasing Resources

If you do not need to use this model and real-time service anymore, release the resources to stop billing.

- On the **Real-Time Services** page, locate the target service and click **Stop** or **Delete** in the **Operation** column.
- In the **My Model** tab, locate the target model and click **Delete** in the **Operation** column.
- On the **Training Jobs** page, delete the completed training job.
- Go to OBS and delete the OBS bucket, folders, and files used in this example.

FAQs

- Why Is a Training Job Always Queuing?

  If the training job is always queuing, the selected resources are limited in the resource pool, and the job needs to be queued. In this case, wait for resources. For details, see **Why Is a Training Job Always Queuing**.

- Why Can't I Find My Created OBS Bucket After I Select an OBS Path in ModelArts?

  Ensure that the created bucket is in the same region as ModelArts. For details, see **Incorrect OBS Path on ModelArts**.

5.2 Running a GPU-based Training Job on ModelArts Standard

5.2.1 Scenario

The data volume and compute used for training vary among AI models. Select a proper storage and training solution to improve training efficiency and resource cost-effectiveness. ModelArts Standard supports multiple training scenarios to meet different requirements, including single-node single-card, single-node multi-card, and multi-node multi-card.

There are public resource pools and dedicated resource pools. If you use a dedicated resource pool, the resources are not shared with other users, ensuring

high efficiency. For enterprises with multiple users, use a dedicated resource pool for AI model training.

This section provides an E2E guidance to help you understand how to select a proper training solution and perform model training on ModelArts Standard.

For different data volumes and algorithms, the solutions are as follows:

- Single-node single-card: If the data volume is small (1 GB training data) and the compute is low (one Vnt1 card), use OBS parallel file system to store data and code.
- Single-node multi-card: If the data volume is medium (50 GB training data) and the compute is medium (eight Vnt1 cards), use SFS to store data and code.
- Multi-node multi-card: If the data volume is large (1 TB training data) and the compute is high (four nodes with eight Vnt1 cards), use SFS to store data and a common OBS bucket to store code, and use distributed training.

Table 5-1 Services required in different scenarios and purchase recommendations

Scenario	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
Single-node single-card	Pay-per-use (parallel file system)	×	Free	Free	Monthly	Free	×	Pay-per-use
Single-node multi-card	×	Monthly (HPC 500 GB)	Free	Free	Monthly	Free	Monthly (Ubuntu 18.04, at least 2 vCPUs and 8 GB memory, 100 GB local storage space, dynamic BGP with EIP, and 10 Mbit/s bandwidth)	×

Scenario	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
Multi-node multi-card	Pay-per-use (common OBS bucket)	Monthly (HPC 500 GB)	Free	Free	Monthly	Free	Monthly (Ubuntu 18.04, at least 2 vCPUs and 8 GB memory, 100 GB local storage space, dynamic BGP with EIP, and 10 Mbit/s bandwidth)	×

Table 5-2 Training performance of different open-source datasets

Algorithm and Data	Resource Flavor	Number of Epochs	Estimated Running Duration (hh:mm:ss)
Algorithm: PyTorch official example for ImageNet Data: ImageNet classification data subset	One node with one Vnt1 card	10	0:05:03
Algorithm: YOLOX Data: COCO 2017 dataset	One node with one Vnt1 card	10	03:33:13
	One node with eight Vnt1 cards	10	01:11:48
	Four nodes with eight Vnt1 cards	10	0:36:17
Algorithm: Swin-Transformer Data: ImageNet21K	One node with one Vnt1 card	10	197:25:03
	One node with eight Vnt1 cards	10	26:10:25

Algorithm and Data	Resource Flavor	Number of Epochs	Estimated Running Duration (hh:mm:ss)
	Four nodes with eight Vnt1 cards	10	07:08:44

Table 5-3 Average response time for different training operations

Operation	Description	Estimated Duration
Downloading an image	Time when an image (25 GB) is downloaded for the first time	8 minutes
Scheduling resources	Duration from the time when a training job starts to be created to the time when the training job becomes running (resources are sufficient and the image is cached)	20 seconds
Accessing the training list page	Time to access the training job list page with 50 records on it	6 seconds
Loading logs	Time to load 1 MB logs on the training details page	2.5 seconds
Accessing the training details page	Time to access the training details page where there are no logs	2.5 seconds
Accessing the JupyterLab page	Time to access the JupyterLab page and load the page content	0.5 seconds
Accessing the notebook list page	Time to access the notebook list page with 50 instances on it	4.5 seconds

NOTE

The preceding data is for reference only. The image download time depends on the node specifications, disk type (high I/O or common I/O), and whether SSD is used.

5.2.2 Preparations

Before using dedicated resource pools for training on ModelArts Standard, perform the following operations.

Purchasing Service Resources

Table 5-4 Purchasing service resources

Service	Description	Details
SFS Turbo	SFS charges you based on the storage capacity and usage duration you select. You can also buy a yearly or monthly package that suits your resource needs and plans. In case of arrears, you have 15 days to renew the service. Otherwise, your file system resources will be removed. You can use SFS to store data and code.	**How Do I Purchase SFS?**
SWR	SWR has two editions: Enterprise Edition and Shared Edition. SWR Shared Edition is free, metered by storage space and traffic. SWR Enterprise Edition supports pay-per-use billing mode. You can use SWR to upload custom images.	**Uploading an Image**
OBS	OBS offers two billing modes: pay-per-use and yearly/monthly. Choose the one that suits your needs. OBS has two storage modes. For single-node and single-card training, use the file system. For multi-node and multi-card training, use the common OBS bucket.	• **Creating a Bucket** • **Creating a Parallel File System**
Virtual Private Cloud (VPC)	A VPC enables you to provision logically isolated, configurable, and manageable virtual networks. VPC interconnection allows you to use resources across VPCs, improving resource utilization.	**Creating a VPC**
Elastic Cloud Server (ECS)	ECSs are more cost-effective than physical servers. Within minutes, you can obtain ECS resources from the cloud service platform. ECS resources are flexible and on-demand. You can use ECS to mount SFS Turbo storage. **NOTE** The ECS and SFS must be in the same VPC for mounting SFS.	**Purchasing an ECS in Custom Config Mode**
Data Encryption Workshop (DEW)	When you use a notebook instance for code debugging, if you want to enable remote SSH development, you need DEW to select a key pair. In this way, you can log in to the ECS using the key pair, improving security. Key pairs can be created free of charge.	**How Do I Create a Key Pair?**

Assigning Permissions

Step 1 Configure IAM permissions.

1. Use a Huawei Cloud tenant account to create a developer user group **user_group** and add developer accounts to **user_group**. For details, see **Step 1 Create a User Group and Add Users to the User Group**.
2. Create a custom policy.
   a. Log in to the management console using a Huawei Cloud tenant account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.
   b. In the navigation pane on the left, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy1** or **Policy2** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

      NOTE

      To define permissions required to access both global and project-level services, enclose the permissions in two separate custom policies for refined authorization. **Learn more**.

      - The content of **Policy1** is as follows:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
            ],
            "Effect": "Deny"
        },
        {
            "Action": [
                "sfsturbo:*:*",
                "vpc:*:*",
                "dss:*:get",
                "dss:*:list"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "ecs:*:*",
                "evs:*:get",
                "evs:*:list",
                "evs:volumes:create",
                "evs:volumes:delete",
                "evs:volumes:attach",
                "evs:volumes:detach",
                "evs:volumes:manage",
                "evs:volumes:update",
                "evs:volumes:use",
                "evs:volumes:uploadImage",
                "evs:snapshots:create",
```



```
            "vpc:*:get",
            "vpc:*:list",
            "vpc:networks:create",
            "vpc:networks:update",
            "vpc:subnets:update",
            "vpc:subnets:create",
            "vpc:ports:*",
            "vpc:routers:get",
            "vpc:routers:update",
            "vpc:securityGroups:*",
            "vpc:securityGroupRules:*",
            "vpc:floatingIps:*",
            "vpc:publicIps:*",
            "ims:images:create",
            "ims:images:delete",
            "ims:images:get",
            "ims:images:list",
            "ims:images:update",
            "ims:images:upload"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
            "vpc:*:*",
            "ecs:*:get*",
            "ecs:*:list*"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
            "kms:cmk:*",
            "kms:dek:*",
            "kms:grant:*",
            "kms:cmkTag:*",
            "kms:partition:*"
         ],
         "Effect": "Allow"
      }
   ]
}
```

- The content of **Policy2** is as follows:

```
{
   "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"
         ],
         "Effect": "Allow"
      }
   ]
}
```

3. Grant the custom policy to the developer user group **user_group**.

a. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **user_group**, click **Authorize** in the **Operation** column, and select **Policy1**, **Policy2**, and **SWR Admin**. Click **Next**.

NOTE

SoftWare Repository for Container (SWR) permissions include SWR FullAccess, SWR OperateAccess, and SWR ReadOnlyAccess. However, these permissions are available only for SWR Enterprise Edition, which has been suspended OBT. You need to select **SWR Admin**.

b. Set the minimum authorization scope, select **All resources** for **Scope** and click **OK**.

For details about permission management, see **Basic Concepts**.

Step 2 Configure ModelArts agency permissions.

Configure ModelArts agency permissions to allow ModelArts to access dependent services such as OBS.

1. Log in to the **ModelArts console** using your Huawei Cloud account. In the navigation pane on the left, choose **Settings**. On the **Global Configuration** page, click **Add Authorization**.
2. Configure the parameters as follows on the displayed page:

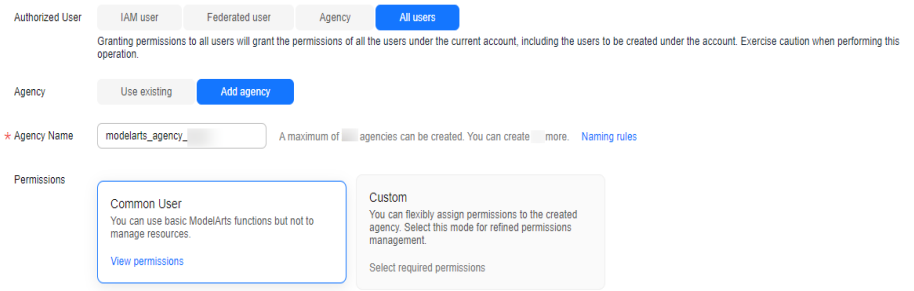
   Authorized User: **All users**.

   Agency: **Add agency**.

   Permissions: **Common User**.

   Select "I have read and agree to the ModelArts Service Statement" and click **Create**.

Figure 5-10 Configuring an agency



3. After the configuration, view the agency configurations of your account on the **Global Configuration** page.

Figure 5-11 Viewing agency configurations

Step 3 Configure SWR organization permissions.

Grant permissions to IAM users in an organization so that they can read, edit, and manage all images in the organization.

Only accounts and IAM users who have the **Manage** permission can add permissions for other users.

1. Log in to the SWR console.
2. In the navigation pane on the left, choose **Organization Management**. On the displayed page, click the target organization in the list.
3. In the **Users** tab, click **Add Permission**. In the displayed dialog box, enter an IAM username, select permissions for the user and click **OK**.

For details about SWR authorization management, see **Granting Permissions of an Organization**.

NOTE

You need to grant the SWR organization permission to the IAM user if they do not have the SWR Admin permission.

Step 4 Verify user permissions.

You may need to wait for 30 minutes before the permission settings are applied. Then, verify the configuration.

1. Log in to the ModelArts management console as an IAM in **UserGroup-2**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. Verify ModelArts permissions.
   a. In the upper left corner of the service list, select ModelArts. The ModelArts management console is displayed.
   b. On the ModelArts management console, check whether you can create notebook instances and training jobs, and register images.
3. Verify SFS permissions.
   a. In the upper left corner of the service list, select SFS. The SFS management console is displayed.
   b. Click **Create File System** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
4. Verify ECS permissions.
   a. In the upper left corner of the service list, select ECS. The ECS management console is displayed.
   b. Click **Buy ECS** in the upper right corner. If this operation is successful, you have obtained ECS operation permissions.
5. Verify VPC permissions.
   a. In the upper left corner of the service list, select VPC. The VPC management console is displayed.
   b. Click **Create VPC** in the upper right corner. If this operation is successful, you have obtained VPC operation permissions.
6. Verify DEW permissions.
   a. In the upper left corner of the service list, select DEW. The DEW management console is displayed.

b. Choose **Key Pair Service** > **Private Key Pairs** and click **Create Key Pair**. If this operation is successful, you have obtained DEW operation permissions.

7. Verify OBS permissions.
   a. In the upper left corner of the service list, select OBS. The OBS management console is displayed.
   b. Click **Create Bucket** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
8. Verify SWR permissions.
   a. In the upper left corner of the service list, select SWR. The SWR management console is displayed.
   b. If an SWR page can be properly displayed, you have obtained SWR operation permissions.
   c. Click **upload an image** in the upper right corner. If authorized organizations are displayed, you have obtained the SWR organization permission.

----End

Creating a Dedicated Resource Pool

ModelArts provides dedicated compute resources, which can be used for notebook instances, training jobs, and model deployment. The resources provided in a dedicated resource pool are exclusive, featuring higher resource efficiency than a public resource pool. To use a dedicated resource pool, create one. For details, see **Creating a Standard Dedicated Resource Pool**.

- **Network**: Select the network that has been connected to the VPC. To create a network and interconnect with VPC, see **Configuring the Standard Dedicated Resource Pool to Access the Internet**.
- **Specifications Type** and **Nodes**: Configure the values based on your needs.

Mounting an SFS Turbo File System to an ECS

After you mount an SFS Turbo file system to an ECS, you can upload the training data to SFS Turbo through the ECS. To do so, perform the following operations:

1. Check the cloud service environment.
   - The ECS and the shared SFS disk belong to the same VPC or interconnected VPCs.
   - The base image of the ECS server is Ubuntu 18.04.
   - The ECS and SFS Turbo are in the same subnet.
2. Set the Huawei Cloud image source in the ECS.
   ```
   sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
   sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
   ```
3. Install the NFS client and mount the target disk.
   ```
   sudo apt-get update
   sudo apt-get install nfs-common
   ```
4. Obtain the command for mounting the SFS Turbo file system.

a. Log in to the SFS console.

b. In the navigation pane on the left, choose **SFS Turbo** > **File Systems**. Then, click the file system name to view its details.

c. In the **Basic Info** tab, obtain and copy the Linux mounting command.

5. Mount the NFS storage to the ECS server.

   Ensure that the corresponding directory exists and run the following commands:

```
mkdir -p /mnt/sfs_turbo
mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
```

Granting the Read Permission to ModelArts Users on the ECS

When a custom image is used for training on ModelArts, the default user is **ma-user** and the default user group is **ma-group**. If a file in the ECS is called during training, grant the read permission to **ma-user**. Otherwise, error "Permission denied" will be displayed.

1. Create **ma-user** and **ma-group** in the ECS.

```
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_group} ] && [ ${default_group} != "ma-group" ]; then \
    groupdel -f ${default_group}; \
    groupadd -g 100 ma-group; \
fi && \
if [ -z ${default_group} ]; then \
    groupadd -g 100 ma-group; \
fi && \
if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
    userdel -r ${default_user}; \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
if [ -z ${default_user} ]; then \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
# set bash as default
rm /bin/sh && ln -s /bin/bash /bin/sh
```

2. Check the created user information.

```
id ma-user
```

   If the following information is displayed, the creation is successful:

```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

Installing and Configuring the OBS CLI

obsutil is a command line tool for accessing and managing OBS. You can use this tool to perform common configuration and management operations on OBS, such as creating buckets, as well as uploading, downloading, and deleting files/folders.

For details about how to install and configure obsutil, see **obsutil Quick Start**.

NOTICE

Replace the AK/SK and endpoint in the commands with the actual values.

(Optional) Configuring Workspaces

ModelArts allows you to set fine-grained permissions for IAM users and resource isolation between different workspaces. ModelArts workspaces support project resource isolation and settlement of different projects.

If you have enabled the enterprise project function, you can bind an enterprise project ID when creating a workspace, add a user group to the enterprise project, and set fine-grained permissions for users in the group.

If you have not enabled the enterprise project function, create an independent workspace on ModelArts. The enterprise project functions are unavailable.

NOTE

The workspace is a whitelist function. To use this function, submit a service ticket.

6 Model Inference

6.1 Enabling a ModelArts Standard Inference Service to Access the Internet

This section describes how to enable an inference service to access the Internet.

Applications

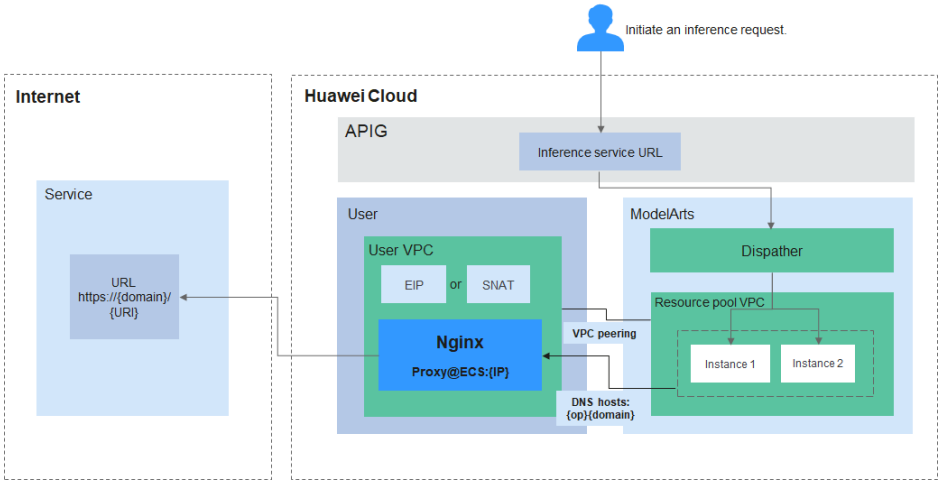
An inference service accesses the Internet in the following scenarios:

- After an image is input, the inference service calls OCR on the Internet and then processes data using NLP.
- The inference service downloads files from the Internet and analyzes the files.
- The inference service sends back the analysis result to the terminal on the Internet.

Solution Design

Use the algorithm on the instance where the inference service is deployed to access the Internet.

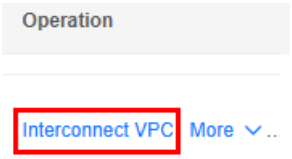
Figure 6-1 Networking for an inference service to access the Internet



Step 1: Interconnecting a ModelArts Dedicated Resource Pool to a VPC

1. Create a VPC and subnet first. For details, see **Creating a VPC and Subnet**.
2. Create a ModelArts dedicated resource pool network.
   a. Log in to the ModelArts console. In the navigation pane on the left, choose **Network** under **Resource Management**.
   b. Click **Create Network**.
   c. In the displayed **Create Network** dialog box, configure the parameters.
   d. Confirm the settings and click **OK**.
3. Interconnect the dedicated resource pool to the VPC.
   a. On the ModelArts console, choose **Network** under **Resource Management** from the navigation pane.
   b. Locate the created network and click **Interconnect VPC** in the **Operation** column.

      Figure 6-2 Interconnecting the VPC

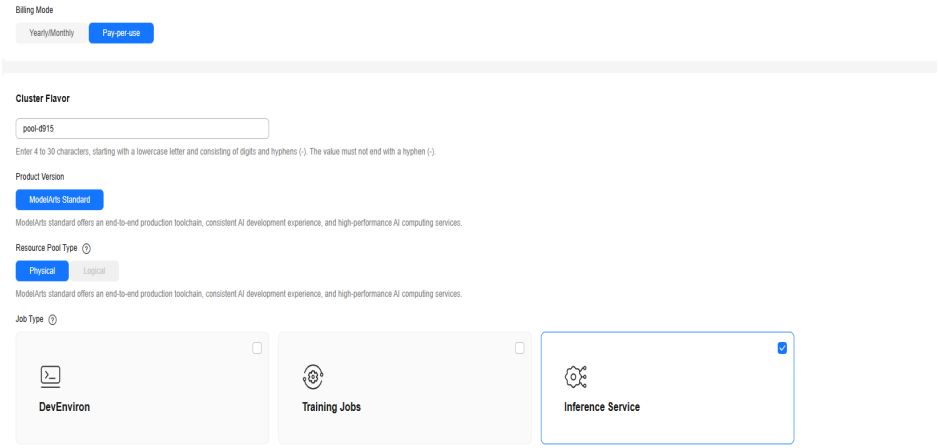
   c. In the displayed dialog box, enable **Interconnect VPC**, and choose the created VPC and subnet from the drop-down lists.

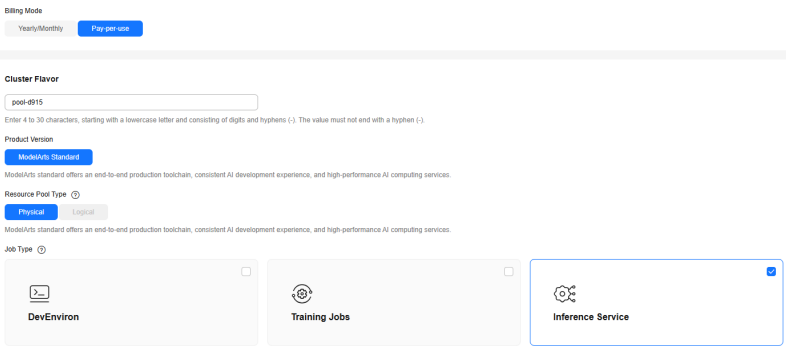
      NOTE

      The peer network to be interconnected cannot overlap with the current CIDR block.
4. Create a ModelArts dedicated resource pool.
   a. On the ModelArts console, choose **Standard Cluster** under **Resource Management** from the navigation pane.
   b. Click **Buy Standard Cluster** and configure the parameters on the displayed page.

      Job Type includes **Inference Service**. For **Network**, choose the network that has been interconnected to the VPC.

      Figure 6-3 Job types



c. Click Buy Now. Confirm the information and click **Submit**.

Step 2: Using Docker to Install and Configure a Forward Proxy

1. Purchase an ECS. For details, see **Purchasing an ECS**. You can select the latest Ubuntu image and the created VPC.
2. Assign an EIP. For details, see **Assigning an EIP**.
3. Bind the EIP to the ECS. For details, see **Binding an EIP to an Instance**.
4. Log in to the ECS and run the following command to install Docker. If it has been installed, skip this step.
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
5. Install the Squid container.
   ```
   docker pull ubuntu/squid
   ```
6. Create a host directory.
   ```
   mkdir –p /etc/squid/
   ```
7. Open and configure the **whitelist.conf** file.
   ```
   vim whitelist.conf
   ```
   The configuration content is the addresses that can be accessed by security control. Wildcard characters can be configured. For example:
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   NOTE

   If the address cannot be accessed, configure the access domain name in the browser.
8. Open and configure the **squid.conf** file.
   ```
   vim squid.conf
   ```
   The configuration details are as follows:
   ```
   # An ACL named 'whitelist'
   acl whitelist dstdomain '/etc/squid/whitelist.conf'

   # Allow whitelisted URLs through
   http_access allow whitelist

   # Block the rest
   http_access deny all

   # Default port
   http_port 3128
   ```
9. Set the permissions on the host directory and configuration files as follows:
   ```
   chmod 640 -R /etc/squid
   ```
10. Start the Squid instance.
    ```
    docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
    ```
11. Go to Docker and refresh Squid.


```
docker exec –it squid bash
root@{container_id}:/# squid -k reconfigure
```

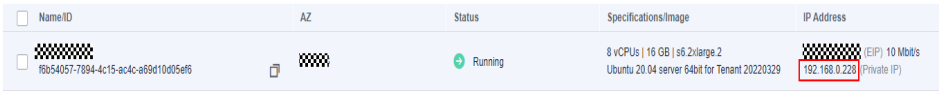
Step 3: Setting the DNS Proxy and Calling a Public IP Address

1. When you customize a model image, set the proxy to the private IP address and port of the proxy server.

   ```
   proxies = {
     "http": "http://{proxy_server_private_ip}:3128",
     "https": "http://{proxy_server_private_ip}:3128"
   }
   ```

   The IP address of the proxy service is the private IP address of the ECS created in **Step 2: Using Docker to Install and Configure a Forward Proxy**. For details about how to obtain the IP address, see **Viewing ECS Details (List View)**.

   Figure 6-4 Private IP address

2. When you call the public IP address, use the service URL to send the service request. For example:

   ```
   https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
   ```

6.2 E2E O&M Solution of ModelArts Inference Services

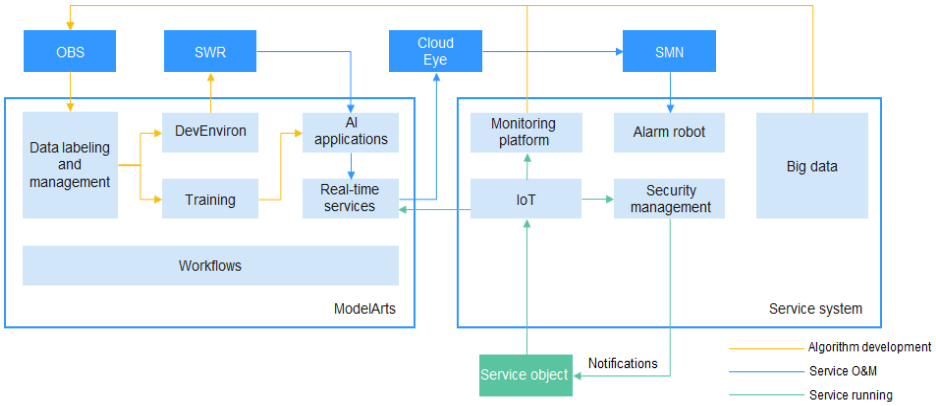
The end-to-end O&M of ModelArts inference services involves the entire AI process including algorithm development, service O&M, and service running.

Overview

End-to-End O&M Process

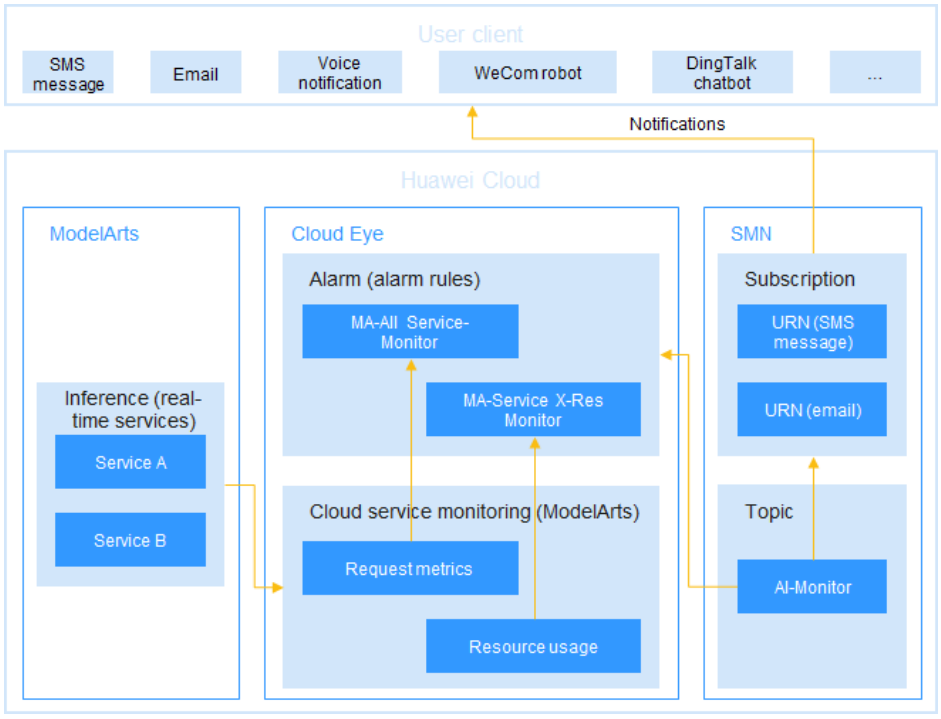
- During algorithm development, store service data in Object Storage Service (OBS), and then label and manage the data using ModelArts data management. After the data is trained, obtain an AI model and create model images using a development environment.
- During service O&M, use an image to create a model and deploy the model as a real-time service. You can obtain the monitoring data of the ModelArts real-time service on the Cloud Eye management console. Configure alarm rules so that you can be notified of alarms in real time.
- During service running, access real-time service requests into the service system and then configure service logic and monitoring.

Figure 6-5 End-to-end O&M process for inference services



During the entire O&M process, service request failures and high resource usage are monitored. When the resource usage threshold is reached, the system will send an alarm notification to you.

Figure 6-6 Alarming process



Advantages

End-to-end service O&M enables you to easily check service running at both peak and off-peak hours and detect the health status of real-time services in real time.

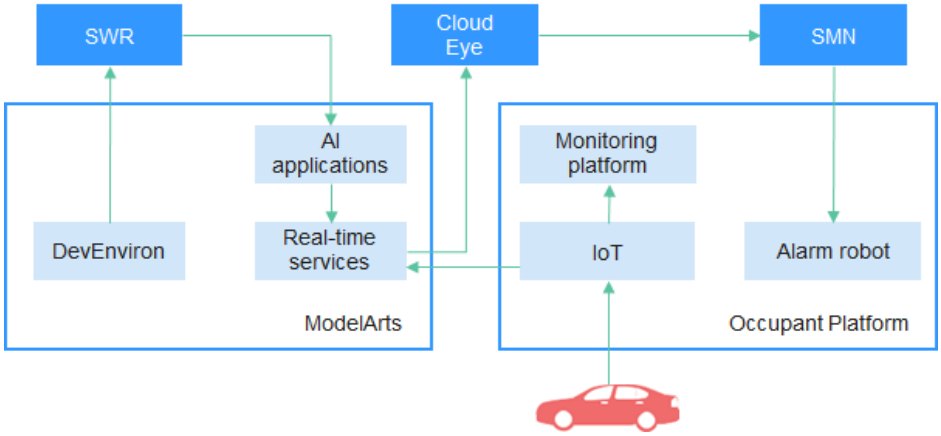
Constraints

End-to-end service O&M applies only to real-time services because Cloud Eye does not monitor batch or edge inference services.

Procedure

This section uses an occupant safety algorithm in travel as an example to describe how to use ModelArts for process-based service deployment and update, as well as automatic service O&M and monitoring.

Figure 6-7 Occupant safety algorithm implementation



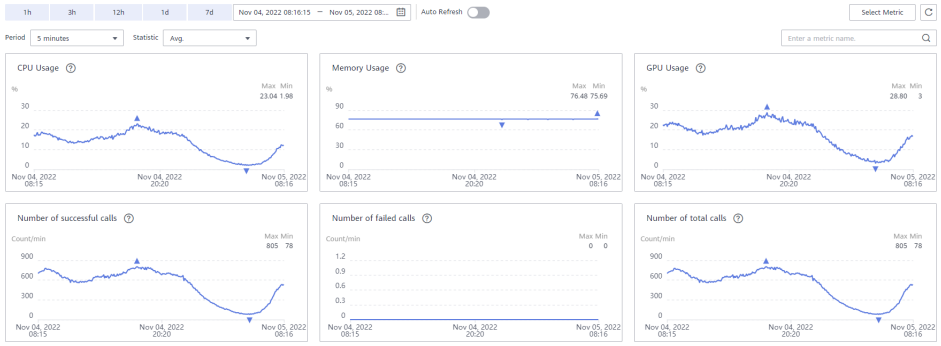
Step 1 Use a custom image to build a model that can be used on the ModelArts Standard inference platform based on the locally developed model. For details, see **Creating a Custom Image on ECS**.

Step 2 On the ModelArts management console, deploy the created model as a real-time service.

Step 3 Log in to the Cloud Eye management console, configure ModelArts alarm rules and enable notifications with a topic subscribed to. For details, see **Setting Alarm Rules**.

After the configuration, choose **Cloud Service Monitoring** > **ModelArts** in the navigation pane on the left to view the requests and resource usage of the real-time service.

Figure 6-8 Viewing service monitoring metrics



When an alarm is triggered based on the monitored data, the object who has subscribed to the target topic will receive a message notification.

----End

6.3 Creating a Model Using a Custom Engine

When using a custom engine to create a model, you can select your image stored in SWR as the engine and specify a file directory in OBS as the model package. In this way, bring-your-own images can be used to meet your dedicated requirements.

Specifications for Using a Custom Engine to Create a Model

To use a custom engine to create a model, ensure the SWR image, OBS model package, and file size comply with the following requirements:

- SWR image specifications
  - A common user named **ma-user** in group **ma-group** must be built in the SWR image. Additionally, the UID and GID of the user must be 1000 and 100, respectively. The following is the dockerfile command for the built-in user:
    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```
  - Specify a command for starting the image. In the dockerfile, specify **cmd**. The following shows an example:
    ```
    CMD sh /home/mind/run.sh
    ```
    Customize the startup entry file **run.sh**. The following is an example.
    ```
    #!/bin/bash

    # User-defined script content
    ...

    # run.sh calls app.py to start the server. For details about app.py, see "HTTPS Example".
    python app.py
    ```
    NOTE

    You can also customize the boot command for starting an image. Enter the customized command during model creation.
  - The service can use the HTTPS/HTTP protocol and listening container port. The port and protocol can be set based on the site requirements. The default request protocol and port number provided by ModelArts are HTTPS and 8080, respectively. For details, see the **HTTPS example**.
  - (Optional) The health check URL must be **/health**.
- OBS model package specifications

  The name of the model package must be **model**. For details about model package specifications, see **Model Package Structure**.
- 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 host.
3. Run the following command in the same directory as the model package folder to start the service:
   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   NOTE

   This command is used for simulation only because the directory mounted to **-v** is assigned the root permission. In the cloud environment, after the model file is downloaded from OBS to **/home/mind/model**, the file owner will be changed to **ma-user**.

4. Start another terminal on the local computer and run the following command to obtain the expected inference result:
   ```
   curl https://127.0.0.1:8080/${Request path to the inference service}
   ```

Deployment Example

The following section describes how to use a custom engine to create a model.

1. Create a model and view model details.

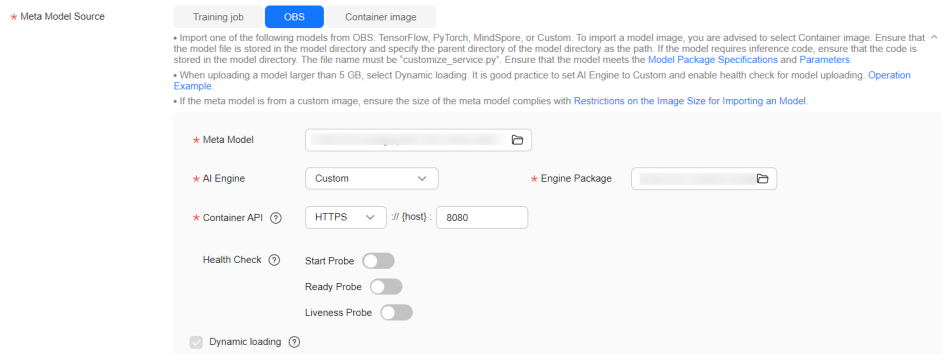
Log in to the ModelArts console. In the navigation pane on the left, choose **Model Management**. Then, click **Create Model** and configure the parameters as follows:

- **Meta Model Source**: Choose **OBS**.
- **Meta Model**: Select a model package selected from OBS
- **AI Engine**: Choose **Custom**.
- **Engine Package**: Select an SWR image.
- **Container API**: Set the port and protocol based on the actual image usage.

  Retain the default settings for other parameters.

Click **Create now**. Wait until the model status changes to **Normal**.

Figure 6-9 Creating a model



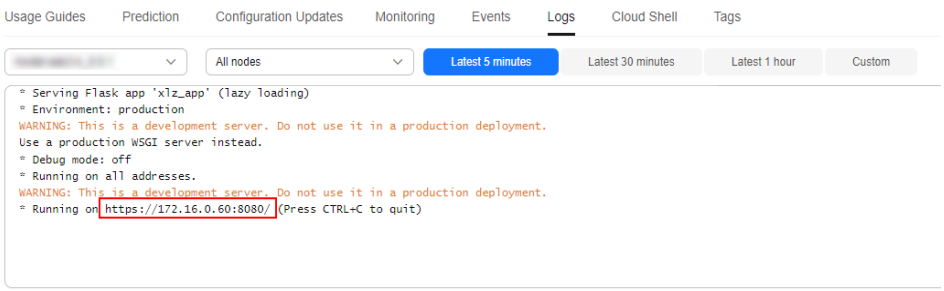
Click the model name to access its details page.

2. Deploy the AI application as a service and view service details.

   On the model details page, choose **Deploy** > **Real-Time Services** in the upper right corner. On the **Deploy** page, select a proper instance flavor (for example, **CPU: 2 vCPUs 8 GB**), retain default settings for other parameters, and click **Next**. When the service status changes to **Running**, the service has been deployed.

   Click the service name. On the page that is displayed, view the service details. Click the **Logs** tab to view the service logs.

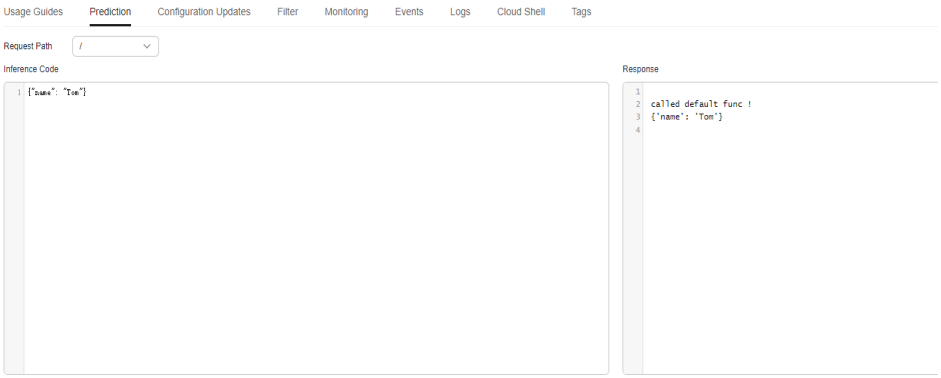
   Figure 6-10 Logs

3. Use the service for prediction.

   On the service details page, click the **Prediction** tab to use the service for prediction.

Figure 6-11 Prediction



6.4 Using a Large Model to Create a Model on ModelArts Standard and Deploy It as a Real-Time Service

Context

Currently, a large model can have hundreds of billions or even trillions of parameters, and its size becomes larger and larger. A large model with hundreds of billions of parameters exceeds 200 GB, and poses new requirements for version management and production deployment of the platform. For example, importing models requires dynamic adjustment of the tenant storage quota. Slow model loading and startup require a flexible timeout configuration in the deployment. The service recovery time needs to be shortened in the event that the model needs to be reloaded upon a restart caused by a load exception.

To address the preceding requirements, the ModelArts inference platform provides a solution to model management and service deployment in large model application scenarios.

Constraints

- You need to apply for the size quota of a model and add the whitelist cached using the local storage of the node.
- You need to use the custom engine **Custom** to configure dynamic loading.
- A dedicated resource pool is required to deploy the service.
- The disk space of the dedicated resource pool must be greater than 1 TB.

Procedure

1. **Applying for Increasing the Size Quota of a Model and Using the Local Storage of the Node to Cache the Whitelist**
2. **Uploading Model Data and Verifying the Consistency of Uploaded Objects**
3. **Creating a Dedicated Resource Pool**
4. **Creating a Model**

5. **Deploying a Real-Time Service**

Applying for Increasing the Size Quota of a Model and Using the Local Storage of the Node to Cache the Whitelist

During service deployment, the dynamically loaded model package is stored in the temporary disk space by default. When the service is stopped, the loaded files are deleted, and they need to be reloaded when the service is restarted. To avoid repeated loading, the platform allows the model package to be loaded from the local storage space of the node in the resource pool and keeps the loaded files valid even when the service is stopped or restarted (using the hash value to ensure data consistency).

To use a large model, you need to use a custom engine and enable dynamic loading when importing the model. In this regard, you need to perform the following operations:

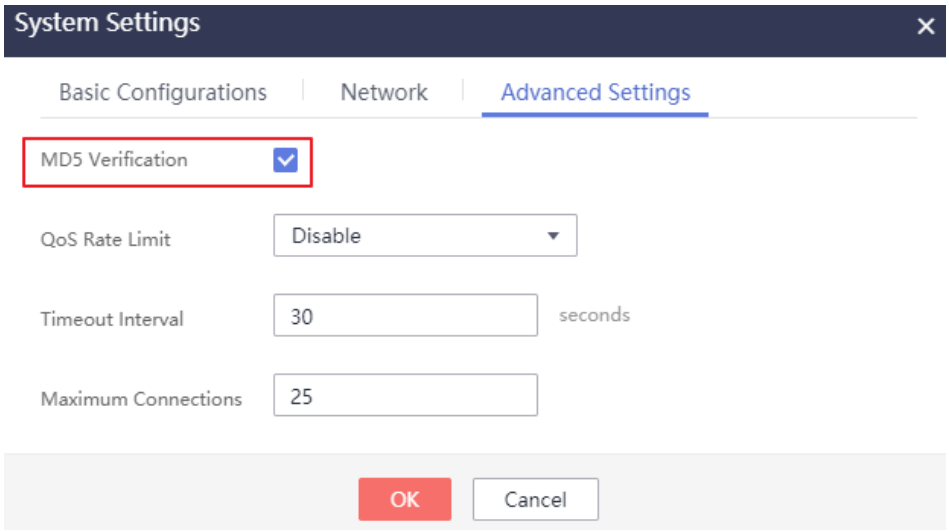
- If the model size exceeds the default quota, submit a service ticket to increase the size quota of a single model. The default size quota of a model is 20 GB.
- Submit a service ticket to add the whitelist cached using the local storage of the node.

Uploading Model Data and Verifying the Consistency of Uploaded Objects

To ensure data integrity during dynamic loading, you need to verify the consistency of uploaded objects when uploading model data to OBS. obsutil, OBS Browser+, and OBS SDKs support verification of data consistency during upload. You can select a method that meets your requirements. For details, see **Verifying Data Consistency During Upload**.

For example, if you upload data via OBS Browser+, enable MD5 verification, as shown in **Figure 6-12**. When dynamic loading is enabled and the local persistent storage of the node is used, OBS Browser+ checks data consistency during data upload.

Figure 6-12 Configuring MD5 verification for OBS Browser+



Creating a Dedicated Resource Pool

To use the local persistent storage, you need to create a dedicated resource pool whose disk space is greater than 1 TB. You can view the disk information on the **Specifications** tab of the **Basic Information** page of the dedicated resource pool. If a service fails to be deployed and the system displays a message indicating that the disk space is insufficient, see **What Do I Do If Resources Are Insufficient When a Real-Time Service Is Deployed, Started, Upgraded, or Modified**.

Figure 6-13 Viewing the disk information of the dedicated resource pool

Creating a Model

If you use a large model to create a model and import the model from OBS, complete the following configurations:

1. Use a custom engine and enable dynamic loading.

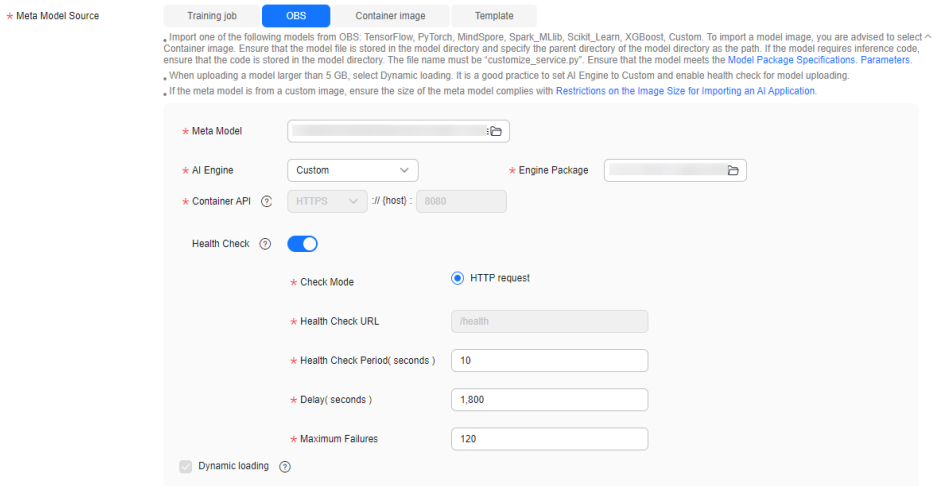
   To use a large model, you need to use a custom engine and enable dynamic loading when importing the model. You can create a custom engine to meet special requirements for image dependency packages and inference frameworks in large model scenarios. For details about how to create a custom engine, see **Creating a Model Using a Custom Engine**.

   When you use a custom engine, dynamic loading is enabled by default. The model package is separated from the image, and the model is dynamically loaded to the service load during service deployment.

2. Configure health check.

   Health check is mandatory for the models imported using a large model to identify unavailable services that are displayed as started.

   Figure 6-14 Using a custom engine, enabling dynamic loading, and configuring health check



Deploying a Real-Time Service

When deploying the service, complete the following configurations:

1. Customize the deployment timeout interval.

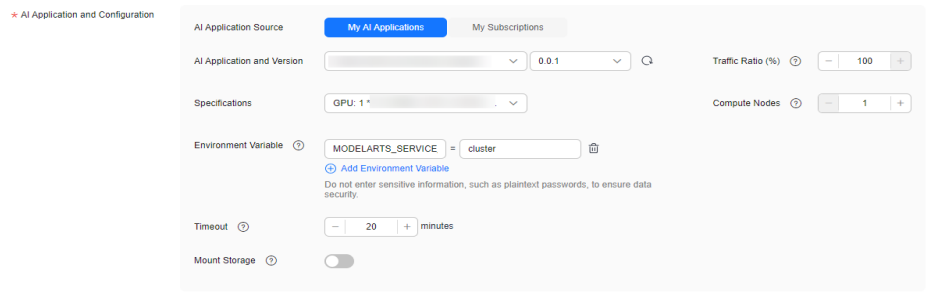
   Generally, the time for loading and starting a large model is longer than that for a common model. Set **Timeout** to a proper value. Otherwise, the timeout may elapse prior to the completion of the model startup, and the deployment may fail.

2. Add an environment variable.

   During service deployment, add the following environment variable to set the service traffic load balancing policy to cluster affinity, preventing unready service instances from affecting the prediction success rate:

   ```
   MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
   ```

   Figure 6-15 Customizing the deployment timeout interval and adding an environment variable

You are advised to deploy multiple instances to improve service reliability.

6.5 Migrating a Third-Party Inference Framework to a Custom Inference Engine

Context

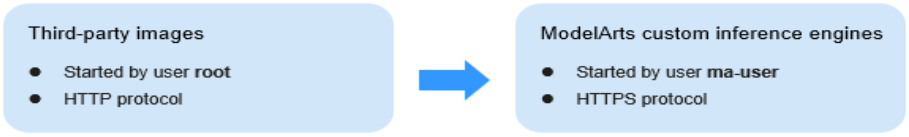
ModelArts allows the deployment of third-party inference frameworks. This section describes how to migrate TF Serving and Triton to a custom inference engine.

- TensorFlow Serving (TF Serving) is a flexible, high-performance model deployment system for machine learning. It provides model version management and service rollback capabilities. By configuring parameters such as the model path, model port, and model name, native TF Serving images can quickly start providing services which can be accessed through gRPC and HTTP RESTful APIs.
- Triton is a high-performance inference service framework. It supports multiple service protocols, including HTTP and gRPC. Additionally, Triton is compatible with various inference engine backends such as TensorFlow, TensorRT, PyTorch, and ONNX Runtime. Notably, it enables multi-model concurrency and dynamic batching, effectively optimizing GPU utilization and enhancing inference service performance.

The migration of a third-party framework to a ModelArts inference framework requires reconstruction of the native third-party framework image. After that, ModelArts model version management and dynamic model loading can be used. This section shows how to complete such a reconstruction. After an image of the custom engine is created, you can use it to create a model and deploy and manage services using the model.

The following figure shows the reconstruction items.

Figure 6-16 Reconstruction items



The reconstruction process may differ for images from various frameworks. For details, see the migration procedure specific to the target framework.

- **Migrating TF Serving**
- **Migrating Triton**

Migrating TF Serving

Step 1 Add user **ma-user**.

The image is built based on the native **tensorflow/serving:2.8.0** image. The user group **100** exists in the image by default. Run the following command in the Dockerfile to add user **ma-user**:

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

Step 2 Set up an Nginx proxy to support HTTPS.

After the protocol is converted to HTTPS, the exposed port changes from 8501 of TF Serving to 8080.

1. Run the following commands in the Dockerfile to install and configure Nginx:

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

2. Create the Nginx directory.

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. Write the nginx.conf file.


```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. Write the modelarts-model-server.conf configuration file.

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
```



```
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8501;
    }
}
```

5. Create a startup script.

   NOTE

   Before executing the TF Serving startup script, you must create an SSL certificate.

   The sample code of the startup script **run.sh** is as follows:

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText
sh /usr/bin/tf_serving_entrypoint.sh
```

Step 3 Modify the default model path to support ModelArts model dynamic loading.

Run the following commands in the Dockerfile to change the default model path:

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----End

Dockerfile example:

```
FROM tensorflow/serving:2.8.0
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENV MODEL_BASE_PATH /home/mind
```



```
ENV MODEL_NAME model
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

Migrating Triton

This section uses the **nvcr.io/nvidia/tritonserver:23.03-py3** image provided by NVIDIA for adaptation and the open-source foundation model LLaMA 7B for inference.

Step 1 Add user **ma-user**.

The **triton-server** user, whose ID is 1000, exists in the Triton image by default. Change the **triton-server** user ID and add the **ma-user** user by running this command in the Dockerfile.

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

Step 2 Set up an Nginx proxy to support HTTPS.

1. Run the following commands in the Dockerfile to install and configure Nginx:

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
```

2. Create the Nginx directory as follows:

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. Write the nginx.conf file.

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
```



```
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. Write the modelarts-model-server.conf configuration file.

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

5. Create a startup script **run.sh**.

   NOTE

   Before executing the Triton startup script, you must create an SSL certificate.

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
```



```
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText

bash /home/mind/model/triton_serving.sh
```

Step 3 Set up **tensorrtllm_backend**.

1. Obtain the source code of **tensorrtllm_backend**; install dependencies (TensorRT, CMake, and PyTorch); compile and install.

```
# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver
```

2. Create the startup script **triton_serving.sh** of Triton serving. The following is an example for the LLaMA model:

```
MODEL_NAME=llama_7b
MODEL_DIR=/home/mind/model/${MODEL_NAME}
OUTPUT_DIR=/tmp/llama/7B/trt_engines/fp16/1-gpu/
MAX_BATCH_SIZE=1
export LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH}

# build tensorrt_llm engine
cd /opt/tritonserver/tensorrtllm_backend/tensorrt_llm/examples/llama
python build.py --model_dir ${MODEL_DIR} \
                --dtype float16 \
                --remove_input_padding \
                --use_gpt_attention_plugin float16 \
                --enable_context_fmha \
                --use_weight_only \
                --use_gemm_plugin float16 \
                --output_dir ${OUTPUT_DIR} \
                --paged_kv_cache \
                --max_batch_size ${MAX_BATCH_SIZE}

# set config parameters
cd /opt/tritonserver/tensorrtllm_backend
mkdir triton_model_repo
```



```
cp all_models/inflight_batcher_llm/* triton_model_repo/ -r

python3 tools/fill_template.py -i triton_model_repo/preprocessing/config.pbtxt tokenizer_dir:${MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:${MAX_BATCH_SIZE},preprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/postprocessing/config.pbtxt tokenizer_dir:${MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:${MAX_BATCH_SIZE},postprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/ensemble/config.pbtxt triton_max_batch_size:${MAX_BATCH_SIZE}
python3 tools/fill_template.py -i triton_model_repo/tensorrt_llm/config.pbtxt triton_max_batch_size:${MAX_BATCH_SIZE},decoupled_mode:False,max_beam_width:1,engine_dir:${OUTPUT_DIR},max_tokens_in_paged_kv_cache:2560,max_attention_window_size:2560,kv_cache_free_gpu_mem_fraction:0.5,exclude_input_in_output:True,enable_kv_cache_reuse:False,batching_strategy:V1,max_queue_delay_microseconds:600

# launch tritonserver
python3 scripts/launch_triton_server.py --world_size 1 --model_repo=triton_model_repo/
while true; do sleep 10000; done
```

Description of some parameters:

- **MODEL_NAME**: name of the OBS folder where the model weight file in Hugging Face format is stored.
- **OUTPUT_DIR**: path to the model file converted by TensorRT-LLM in the container.

The complete Dockerfile is as follows:

```
FROM nvcr.io/nvidia/tritonserver:23.03-py3

# add ma-user and install nginx
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user && \
    apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
```



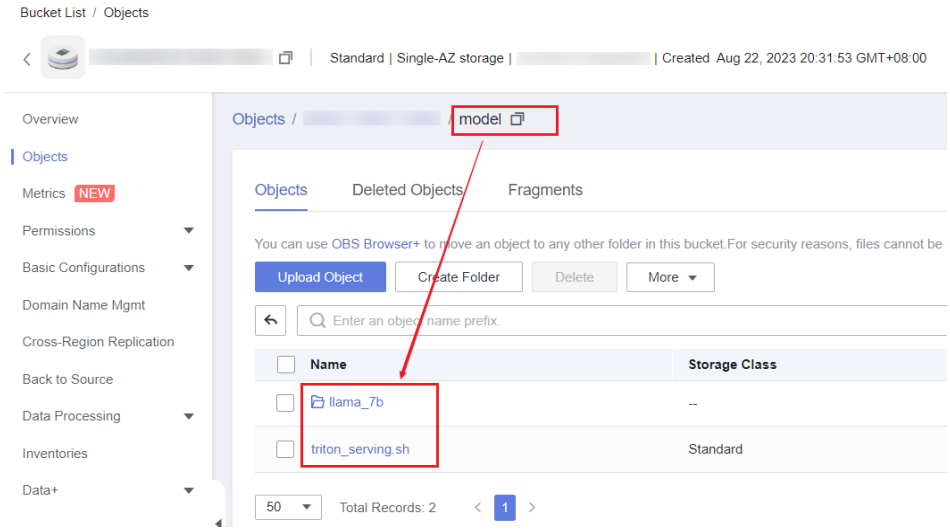
```
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

After the image is created, register the image with Huawei Cloud SWR for deploying inference services on ModelArts.

Step 4 Use the adapted image to deploy a real-time inference service on ModelArts.

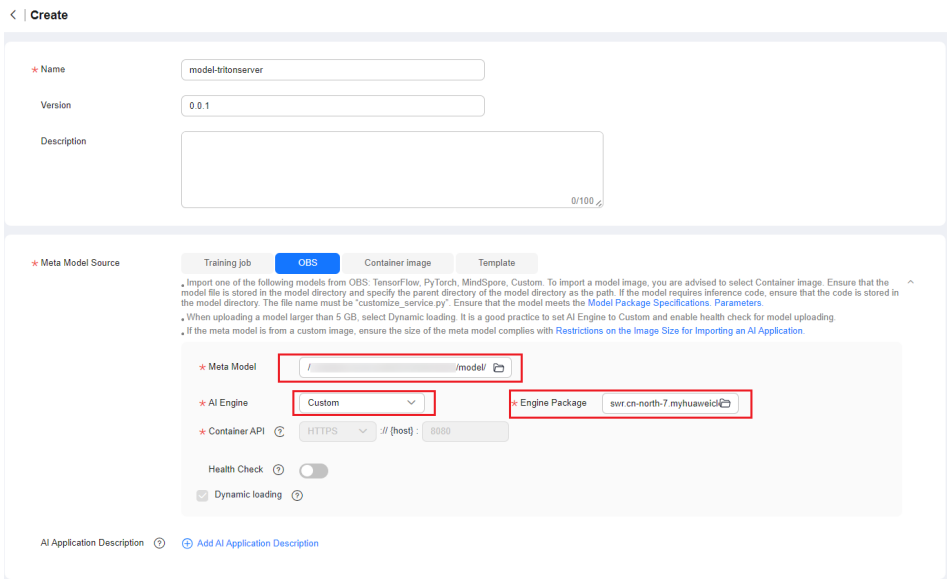
1. Create a **model** directory in OBS and upload the **triton_serving.sh** file and **llama_7b** folder to the **model** directory.

Figure 6-17 Uploading files to the **model** directory



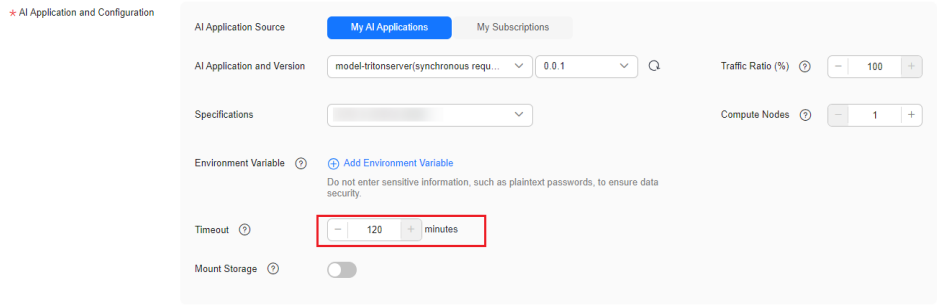
2. Create a model. Set **Meta Model Source** to **OBS** and select the meta model from the **model** directory. Set **AI Engine** to **Custom**. Set **Engine Package** to the image created in **Step 3**.

Figure 6-18 Creating a model



3. Deploy the created model as a real-time service. Generally, the time for loading and starting a large model is longer than that for a common model. Set **Timeout** to a proper value. Otherwise, the timeout may elapse prior to the completion of the model startup, and the deployment may fail.

Figure 6-19 Deploying a real-time service



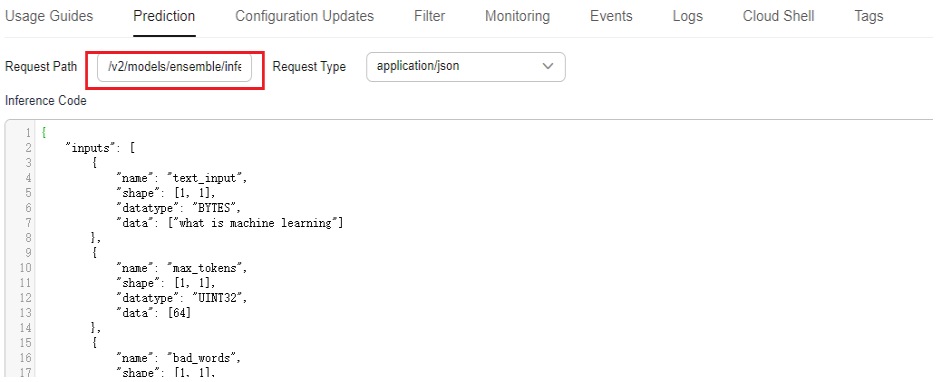
4. Call the real-time service for foundation model inference. Set the request path to **/v2/models/ensemble/infer**. The following is an example call:

```
{
  "inputs": [
    {
      "name": "text_input",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": ["what is machine learning"]
    },
    {
      "name": "max_tokens",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [64]
    },
    {
      "name": "bad_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "stop_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "pad_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    },
    {
      "name": "end_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    }
  ],
  "outputs": [
    {
      "name": "text_output"
    }
  ]
}
```


NOTE

- In "inputs", the element with the "name" "text_input" represents the input, and its "data" field specifies a specific input statement. In this example, the input statement is "what is machine learning".
- The element with the "name" "max_tokens" indicates the maximum number of output tokens. In this case, the value is **64**.

Figure 6-20 Calling a real-time service



----End

6.6 Enabling High-Speed Access to an Inference Service Through VPC Peering

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.

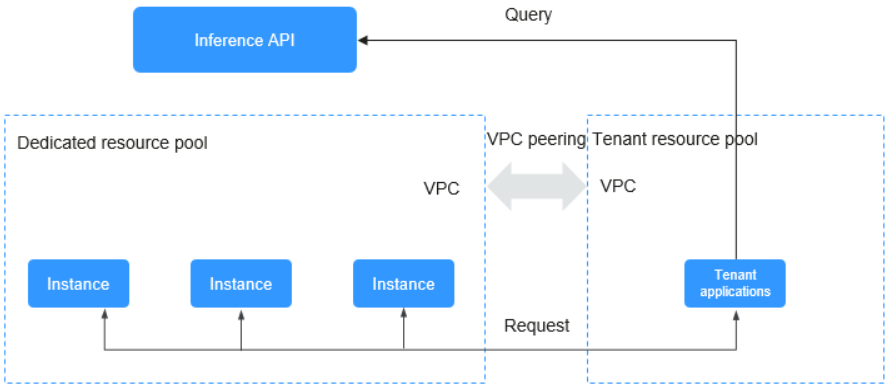
In high-speed access through VPC peering, your service requests are directly sent to instances through VPC peering but not through the inference platform. This accelerates service access.

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 6-21 High-speed access through VPC peering



Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.

Preparations

Deploy a real-time service in a dedicated resource pool and ensure the service is running.

NOTICE

- Only the services deployed in a dedicated resource pool support high-speed access through VPC peering.
- High-speed access through VPC peering is available only for real-time services.
- Due to traffic control, there is a limit on how often you can get the IP address and port number of a real-time service. The number of calls of each tenant account cannot exceed 2000 per minute, and that of each IAM user account cannot exceed 20 per minute.
- High-speed access through VPC peering is available only for the services deployed using the AI applications imported from custom images.

Procedure

To enable high-speed access to a real-time service through VPC peering, perform the following operations:

1. **Interconnect the dedicated resource pool to the VPC.**
2. **Create an ECS in the VPC.**

3. **Obtain the IP address and port number of the real-time service.**
4. **Access the service through the IP address and port number.**

Step 1 Interconnect the dedicated resource pool to the VPC.

Log in to the ModelArts console, choose **AI Dedicated Resource Pools** > **Elastic Clusters**, locate the dedicated resource pool where the service is deployed, and click its name/ID to go to the resource pool details page. Obtain the network configuration. Switch back to the dedicated resource pool list, click the **Network** tab, locate the network associated with the dedicated resource pool, and interconnect it with the VPC. After the VPC is accessed, the VPC will be displayed on the network list and resource pool details pages. Click the VPC to go to the details page.

Figure 6-22 Obtaining the network configuration

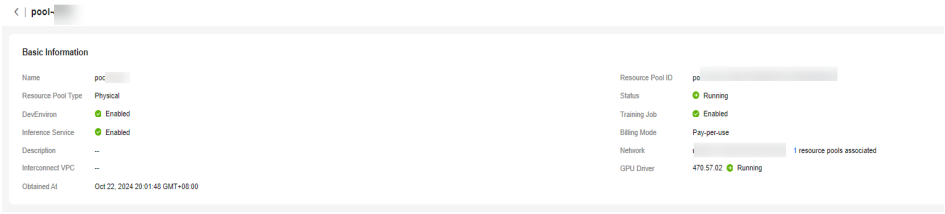


Figure 6-23 Interconnecting the VPC

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 6-24 Selecting a VPC when purchasing an ECS

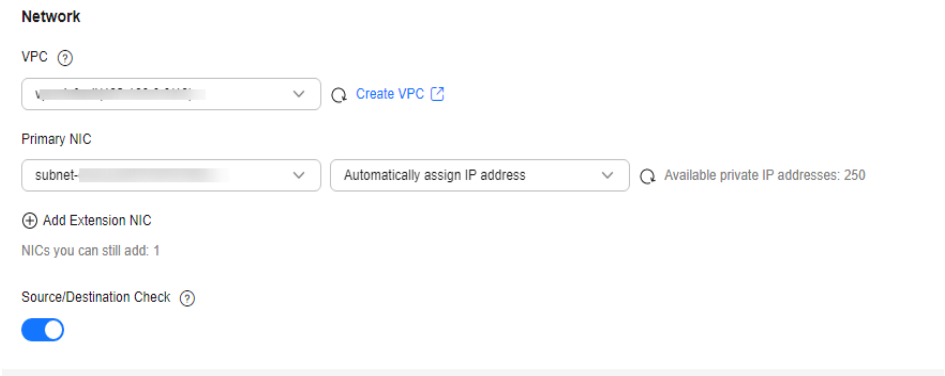
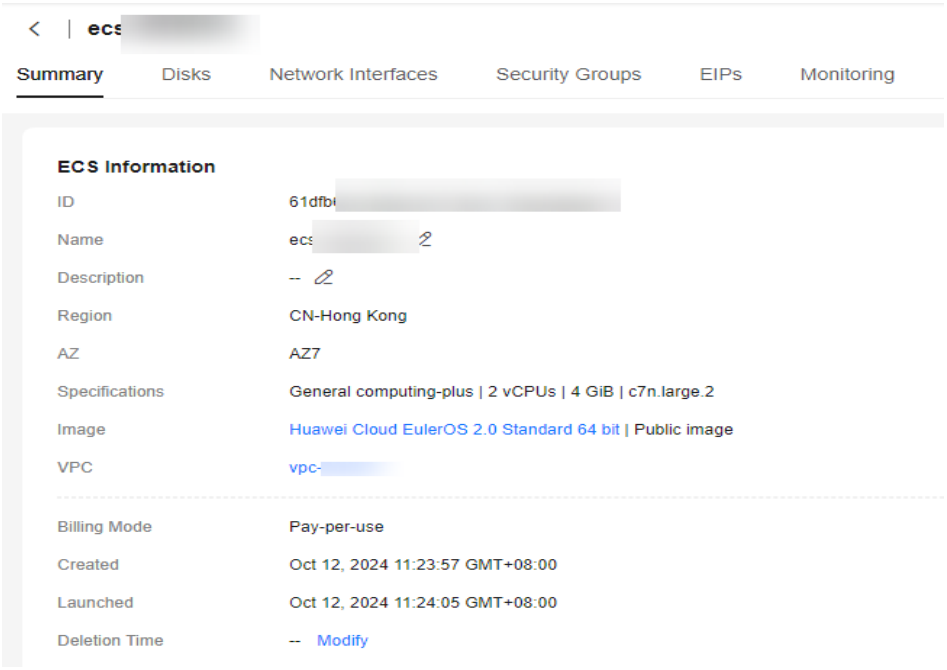


Figure 6-25 Viewing VPC information



Step 3 Obtain the IP address and port number of the real-time service.

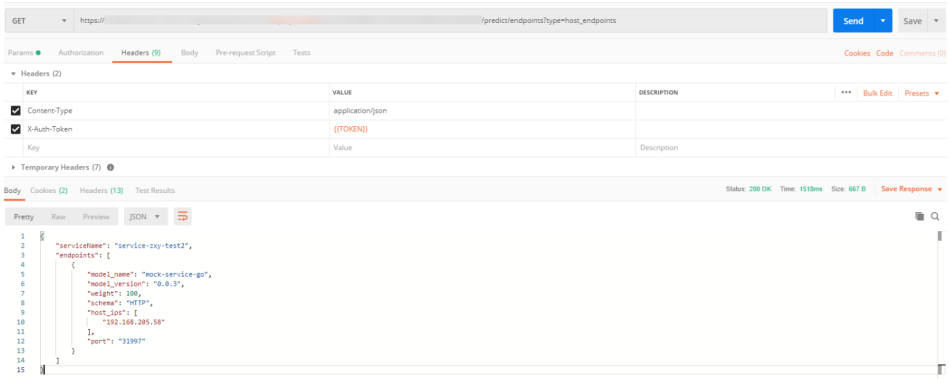
GUI software, for example, Postman can be used to obtain the IP address and port number. Alternatively, log in to the ECS, create a Python environment, and execute code to obtain the service IP address and port number.

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 6-26 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 **Before You Start**.

```
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 6-27 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

6.7 Full-Process Development of WebSocket Real-Time Services

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 an OSI model. The WebSocket communication protocol was established by IETF in 2011 as standard RFC 6455 and supplemented by RFC 7936. The WebSocket API in the Web IDL is standardized by W3C.

WebSocket simplifies data exchange between the client and the 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.

Prerequisites

- You are experienced in developing Java and familiar with JAR packaging.
- You have basic knowledge and calling methods of WebSocket.
- You are familiar with the method of creating an image using Docker.

Constraints

- WebSocket supports only the deployment of real-time services.
- It supports only real-time services deployed using models imported from custom images.

Preparations

Before using WebSocket in ModelArts for inference, bring your own custom image. The custom image must be able to provide complete WebSocket services in a standalone environment, for example, completing WebSocket handshakes and exchanging data between the client to the server. The model inference is implemented in the custom image, including downloading the model, loading the model, performing preprocessing, completing inference, and assembling the response body.

Procedure

To develop a WebSocket real-time service, perform the following operations:

- **Uploading the Image to SWR**
- **Creating a Model Using an Image**
- **Deploying the Model as a Real-Time Service**
- **Calling the WebSocket Real-Time Service**

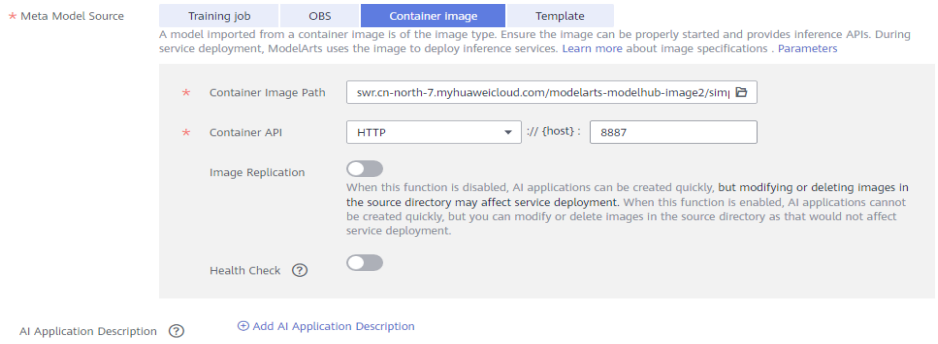
Uploading the Image to SWR

Upload the local image to SWR.

Creating a Model Using an Image

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Model Management (AI Applications)**. On the displayed page, click **Create Model**.
2. Configure the AI application.
   - **Meta Model Source**: Select **Container image**.
   - **Container Image Path**: Select the path specified in **Uploading the Image to SWR**.
   - **Container API**: Configure this parameter based on site requirements.
   - **Health Check**: Retain default settings. If health check has been configured in the image, configure the health check parameters based on those configured in the image.

Figure 6-28 Model configuration parameters

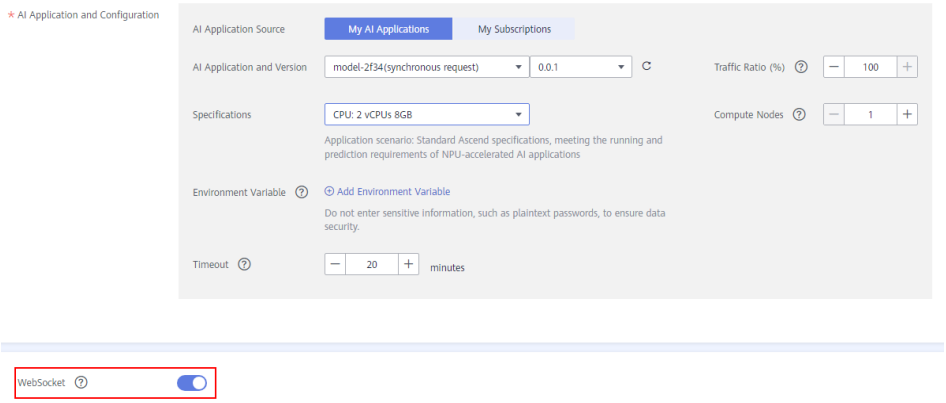


3. Click **Create now**. Wait until the model status changes to **Normal**.

Deploying the Model as a Real-Time Service

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Model Deployment** > **Real-Time Services**. On the displayed page, click **Deploy**.
2. Configure the service.
   - **Model and Version**: Select the model and version created in **Creating a Model Using an Image**.
   - **WebSocket**: Enable this function.

   Figure 6-29 WebSocket



3. Click **Next**, confirm the configuration, and click **Submit**. In the real-time service list you will be redirected to, check the service status. When it changes to **Running**, the real-time service has been deployed.

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:

- **Token-based Authentication**

- **AK/SK**
- **App Authentication**

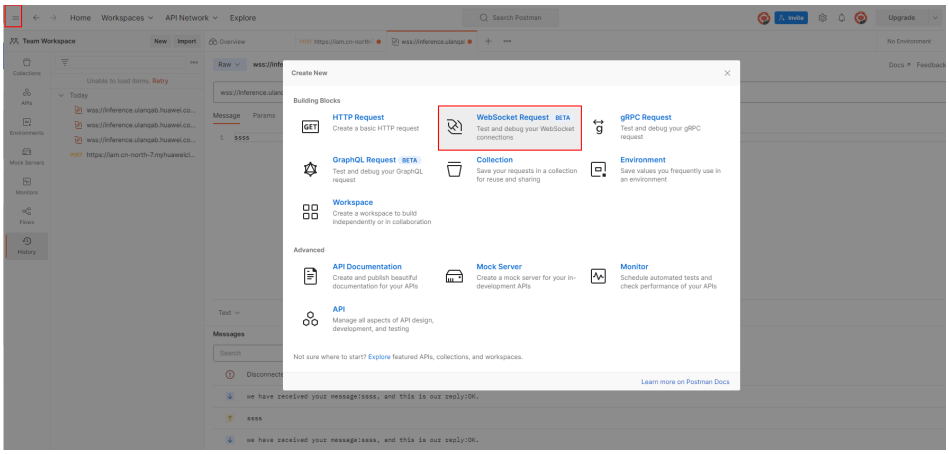
The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call WebSocket.

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).

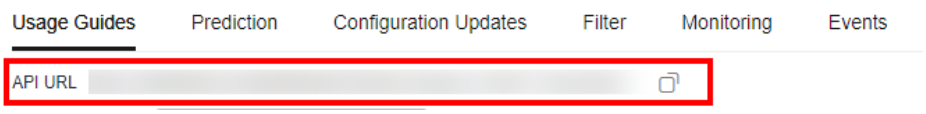
   Figure 6-30 WebSocket Request

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 6-31 Obtaining the API address

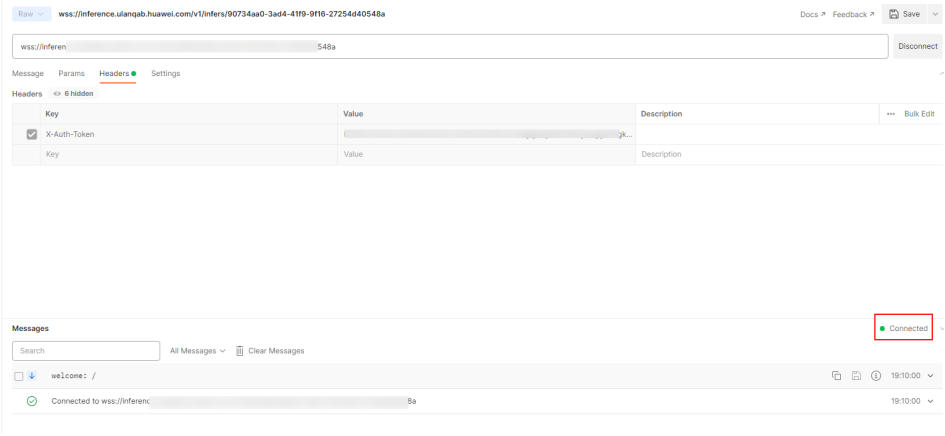
   

NOTE

- 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 6-32 Connection established



NOTICE

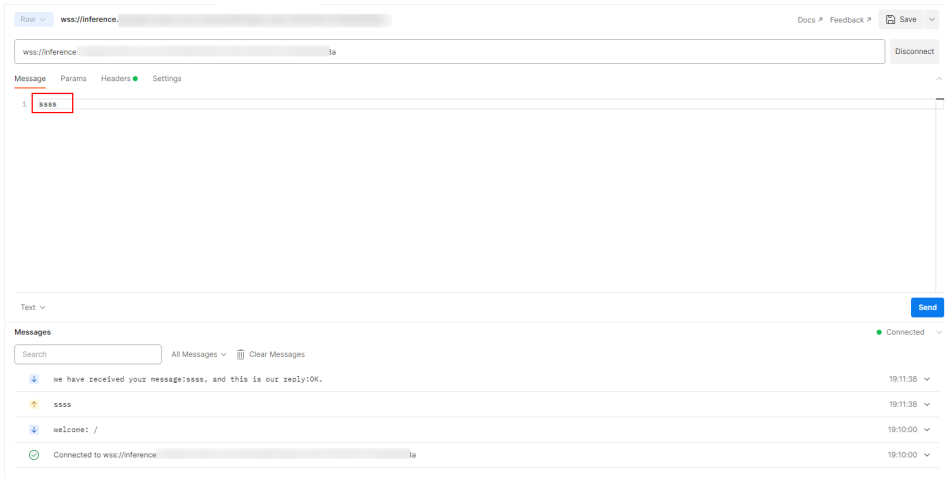
Preferentially check the WebSocket service provided by the custom image. The type of implementing WebSocket varies depending on the tool you used. Possible issues are as follows: A WebSocket connection can be established but cannot be maintained, or the connection is interrupted after one request and 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 6-33 Sending data



----End

6.8 Creating a Custom Image and Using It to Create a Model

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

The procedure is as follows:

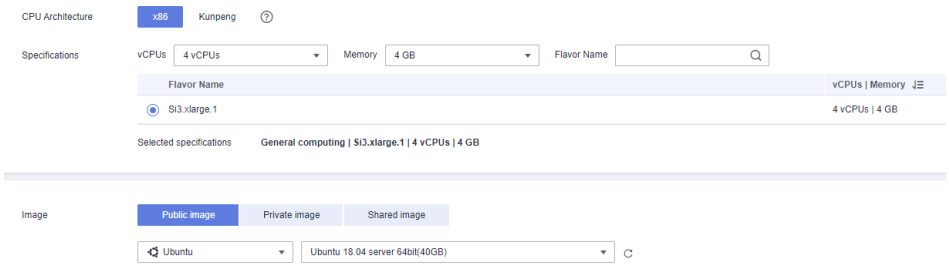
1. **Building an Image Locally**: Create a custom image package locally. For details, see **Specifications for Custom Images**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Creating a Model Using a Custom Image**: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

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 6-34 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'
     ```



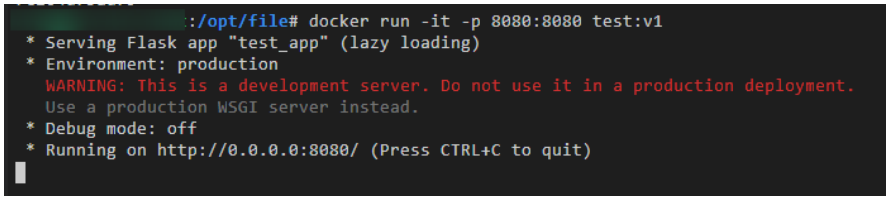
```
@app.route('/', methods=['POST'])
def default_func():
    print("---------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# 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
   ```

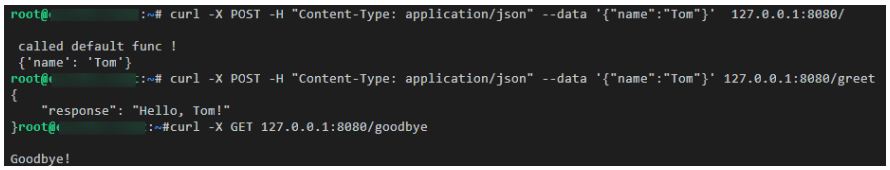
   Figure 6-35 Starting a custom image

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 6-36 Testing API functions

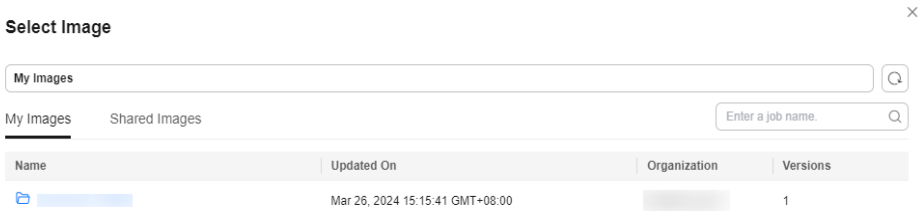
3. Upload the custom image to SWR.
4. After the custom image is uploaded, view the uploaded image on the **My Images** > **Private Images** page of the SWR console.

Creating a Model Using a Custom Image

Import a meta model. For details, see **Importing a Meta Model from a Container Image**. Key parameters are as follows:

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

Figure 6-37 Created private image



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

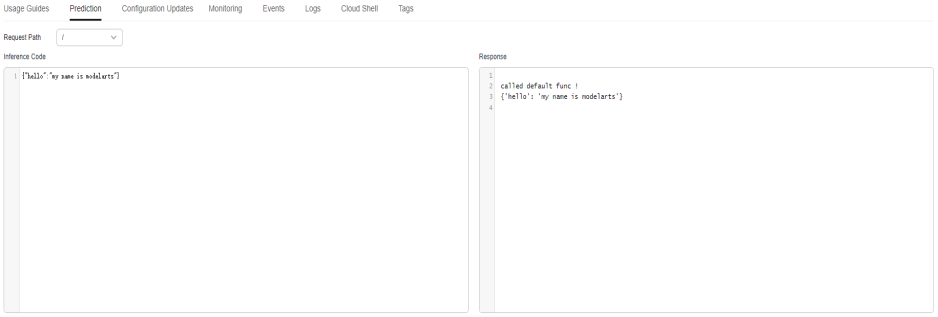
  The configuration file is as follows:

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

Deploying the Model as a Real-Time Service

1. Deploy the model as a real-time service. For details, see **Deploying 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 6-38 Accessing the real-time service



7 Best Practices of Security Configuration

Scenario

Security is a shared responsibility between Huawei Cloud and yourself. Huawei Cloud ensures the security of cloud services for a secure cloud. As a tenant, you should properly use the security capabilities provided by cloud services to protect data and securely use the cloud.

This section provides actionable guidance for enhancing the overall security of ModelArts. You can continuously evaluate the security status of your ModelArts resources and enhance their overall security by combining different security capabilities provided by ModelArts. By doing this, data stored in ModelArts can be protected from leakage and tampering both at rest and in transit.

Consider the following aspects for your security configurations:

- **Using an IP Address Whitelist for Access to Notebook**
- **Using a Dedicated Resource Pool in the Production Environment**
- **Running a Custom Image as a Non-root User**
- **Not Using Hard-coded Credentials During Development**
- **Using Independent Agencies for Different IAM Users**

Using an IP Address Whitelist for Access to Notebook

ModelArts Standard notebook instances can be directly connected in SSH mode and authenticated using key pairs. If you have higher security requirements, configure an IP address whitelist to limit access to the instance exclusively to approved endpoints. For details, see the **Whitelist** parameter in **Creating a Notebook Instance**.

Using a Dedicated Resource Pool in the Production Environment

When you use the training, inference, and development environments, you shall use the dedicated resource pool in the production environment, which provides exclusive compute resources and enhanced secure resource isolation capabilities. For details about how to use a dedicated resource pool, see **Creating a Standard Dedicated Resource Pool**.

When using ModelArts for full-process AI development, you can use two different resource pools.

Public resource pools: provide large-scale public compute clusters, which are allocated based on job parameter settings. Resources are isolated by job. You will be billed based on resource specifications, usage duration, and the number of instances used in a public resource pool, regardless of tasks (training, deployment, or development). Public resource pools are provided by ModelArts by default and do not need to be created or configured. You can directly select a public resource pool during AI development.

Dedicated resource pools: provide dedicated compute resources, which can be used for notebook instances, training jobs, and model deployment. The resources provided in a dedicated resource pool are exclusive, featuring higher resource efficiency than a public resource pool.

To use a dedicated resource pool, you need to purchase one and select it during AI development.

Running a Custom Image as a Non-root User

You can create a Dockerfile for a custom image and then push it to SWR. To enhance permission control, you shall explicitly define the default running user as a non-root user when customizing an image. This helps reduce security risks during container runtime.

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

For details about how to use custom images in ModelArts Standard, see **Application Scenarios of Custom Images**.

Not Using Hard-coded Credentials During Development

If you want to develop an algorithm and publish it to the production environment in ModelArts Standard Notebook, you shall check the password, AK/SK, database connection, OBS connection, and SWR connection information used in the code. Do not use fixed authentication credentials to facilitate subsequent algorithm update and maintenance. You shall encrypt the preceding sensitive information and save it in the program configuration file.

Using Independent Agencies for Different IAM Users

To use ModelArts resources, ensure that you have obtained the agency authorization from the user. For better permission control over IAM users, you shall grant agency permissions to each IAM user individually on the ModelArts global configuration page. The same agency credential shall not be shared by multiple IAM users. For details about agency authorization, see **Creating an IAM User and Granting ModelArts Permissions**.

